Programação do ADOBE® ACTIONSCRIPT® 3.0

Adobe

Programação do ActionScript™ 3.0

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA

Conteúdo

Capítulo 1: Sobre este manual

Este manual fornece informações básicas para o desenvolvimento de aplicativos no Adobe® ActionScript® 3.0. Para entender melhor as idéias e as técnicas descritas, você deve estar familiarizado com os conceitos gerais de programação, como tipo de dados, variáveis, loops e funções. Você também deve compreender os conceitos básicos de programação orientada a objetos, como classes e herança. Conhecimento anterior do ActionScript 1.0 ou ActionScript 2.0 é útil, mas não é necessário.

Uso deste manual

Os capítulos deste manual estão organizados nos seguintes grupos lógicos para ajudá-lo a localizar melhor as áreas relacionadas da documentação do ActionScript:

Capítulos	**Descrição**
Capítulos de 2 a 5 - visão geral da programação do ActionScript	Discute os principais conceitos do ActionScript 3.0, incluindo sintaxe de linguagem, instruções e operadores e programação orientada a objetos do ActionScript.
Capítulos de 6 a 11 - principais tipos de dados e classes do ActionScript 3.0	Descreve os tipos de dados de nível superior no ActionScript 3.0.
Capítulos de 12 a 32 - APIs do Flash Player e do Adobe AIR	Descreve recursos importantes que são implementados nos pacotes e nas classes específicas do Adobe Flash Player 10 e do Adobe AIR, incluindo manipulação de eventos, trabalho com objetos de exibição e com a lista de exibição, rede e comunicação, entrada e saída de arquivos, interface externa, modelo de segurança de aplicativos e muito mais.

Este manual também contém vários arquivos de amostra que demonstram os conceitos de programação de aplicativos quanto a classes importantes ou utilizadas normalmente. Os arquivos de amostra são compactados para que se tornem mais fáceis de carregar e usar com o Adobe® Flash® CS4 Professional e podem incluir arquivos delimitadores. Entretanto, o principal código de amostra é ActionScript 3.0 puro que pode ser usado em qualquer ambiente de desenvolvimento.

O ActionScript 3.0 pode ser escrito e compilado de várias formas, inclusive:

- Com o uso do ambiente de desenvolvimento do Adobe Flex Builder 3
- Com o uso de qualquer editor de texto e um compilador de linha de comando, como aquele fornecido com o Flex Builder 3
- Com o uso da ferramenta de autoria do Adobe® Flash® CS4 Professional

Para obter mais informações sobre ambientes de desenvolvimento do ActionScript , consulte “Introdução ao ActionScript 3.0” na página 4

Para compreender as amostras de código neste manual, você não precisa ter experiência anterior nos ambientes de desenvolvimento integrados para ActionScript, como o Flex Builder ou as ferramentas de autoria do Flash. Entretanto, talvez você queira fazer referência na documentação àquelas ferramentas para saber como usá-las para escrever e compilar o código do ActionScript 3.0 Para obter mais informações, consulte “Acesso à documentação do ActionScript” na página 2.

Acesso à documentação do ActionScript

Como este manual concentra-se na descrição do ActionScript 3.0, que é uma linguagem rica e poderosa de programação orientada a objetos, ele não abrange totalmente o processo nem o fluxo de trabalho de desenvolvimento de aplicativos dentro de uma determinada ferramenta ou arquitetura de servidor. Portanto, além da *Programação do ActionScript 3.0*, talvez você queira consultar outras fontes de documentação enquanto projeta, desenvolve, testa e implanta aplicativos do ActionScript 3.0.

Documentação do ActionScript 3.0

Este manual fornece os conceitos da linguagem de programação do ActionScript 3.0 além de detalhes e amostras da implementação ilustrando importante recursos de linguagem. Entretanto, este manual não é uma referência completa quanto à linguagem. Para isso, consulte Referência dos componentes e da linguagem do ActionScript 3.0, que descreve cada classe, método, propriedade e evento na linguagem. A Referência de componentes e linguagem do ActionScript 3.0 fornece informações de referência detalhadas sobre a linguagem principal, os componentes da ferramenta de autoria do Flash (nos pacotes fl) e as APIs do Flash Player e do Adobe AIR (nos pacotes flash).

Documentação do Flash

Se você usar a ferramenta de autoria do Flash, talvez queira consultar estes manuais:

Manual	**Descrição**
Uso do Flash	Descreve como desenvolver aplicativos dinâmicos da Web na ferramenta de autoria do Flash.
Programação do ActionScript 3.0	Descreve a utilização específica da linguagem do ActionScript 3.0 e das APIs principais do Flash Player e do Adobe AIR.
Referência de componentes e linguagem do ActionScript 3.0	Fornece exemplos de sintaxe, utilização e código para os componentes da ferramenta de autoria do Flash e para as APIs do ActionScript 3.0.
Uso dos componentes do ActionScript 3.0	Explica os detalhes do uso de componentes para desenvolvimento de aplicativos desenvolvidos no Flash.
Desenvolvimento de aplicativos do Adobe AIR com o Flash CS4 Professional	Descreve como desenvolver e implantar aplicativos do Adobe AIR utilizando o ActionScript 3.0 e a API do Adobe AIR no Flash.
Aprendizagem do ActionScript 2.0 no Adobe Flash	Fornece uma visão geral da sintaxe do ActionScript 2.0 e explica como usar o ActionScript 2.0 quando está trabalhando com diferentes tipos de objetos.
Referência de linguagem do ActionScript 2.0	Fornece exemplos de sintaxe, utilização e código para os componentes da ferramenta de autoria do Flash e para as APIs do ActionScript 2.0.
Uso dos componentes do ActionScript 2.0	Explica os detalhes de como usar os componentes do ActionScript 2.0 para desenvolvimento de aplicativos criados no Flash.
Referência dos componentes e da linguagem do ActionScript 2.0	Descreve cada componentes disponível na Arquitetura de componentes da Adobe versão 2, junto com sua API.
Ampliação do Flash	Descreve os objetos, os métodos e as propriedades disponíveis na API JavaScript.
Introdução ao Flash Lite 2.x	Explica como usar o Adobe® Flash® Lite™ 2.x para desenvolver aplicativos e fornece exemplos de sintaxe, utilização e código para os recursos do ActionScript disponíveis com o Flash Lite 2.x
Desenvolvimento de aplicativos do Flash Lite 2.x	Explica como desenvolver aplicativos do Flash Lite 2.x.
Introdução ao ActionScript do Flash Lite 2.x	Mostra como desenvolver aplicativos com o Flash Lite 2.x e descreve todos os recursos do ActionScript disponíveis para os desenvolvedores do Flash Lite 2.x.

Manual	Descrição
Referência de linguagem do ActionScript do Flash Lite 2.x	Fornece exemplos de sintaxe, utilização e código para a API do ActionScript 2.0 disponível no Flash Lite 2.x.
Introdução ao Flash Lite 1.x	Apresenta uma introdução ao Flash Lite 1.x e descreve como testar o conteúdo utilizando o emulador do Adobe® Device Central CS4.
Desenvolvimento de aplicativos do Flash Lite 1.x	Descreve como desenvolver aplicativos para dispositivos móveis utilizando o Flash Lite 1.x.
Aprendizagem do ActionScript no Flash Lite 1.x	Explica como usar o ActionScript em aplicativos do Flash Lite 1.x e descreve todos os recursos do ActionScript disponíveis com o Flash Lite 1.x.
Referência de linguagem do ActionScript do Flash Lite 1.x	Fornece a sintaxe e a utilização dos elementos do ActionScript disponíveis com o Flash Lite 1.x.

Recursos de aprendizagem do ActionScript

Além do conteúdo desses manuais, a Adobe fornece artigos, conceitos de design e exemplos no Adobe Developer Center e no Adobe Design Center atualizados regularmente.

Adobe Developer Center

O Adobe Developer Center é o seu recurso para obter informações atualizadas sobre o ActionScript, artigos sobre desenvolvimento de aplicativos reais e informações sobre problemas importantes. Consulte o Developer Center em www.adobe.com/devnet/.

Adobe Design Center

Saiba das novidades em design digital e gráficos de movimento. Navegue nos trabalhos dos principais artistas, descubra novas tendências de design e aprimore suas habilidades com tutoriais, fluxos de trabalho importantes e técnicas avançadas. Confira duas vezes por mês os tutoriais e artigos recentes, além de peças de galeria inspiradoras. Consulte o Design Center em www.adobe.com/designcenter/.

Capítulo 2: Introdução ao ActionScript 3.0

Este capítulo fornece uma visão geral do Adobe® ActionScript® 3.0, a mais recente e revolucionária versão do ActionScript.

Sobre o ActionScript

O ActionScript é a linguagem de programação dos ambientes de tempo de execução Adobe® Flash® Player e Adobe® AIR™. Ele permite interatividade, manipulação de dados e muito mais no conteúdo e nos aplicativos do Flash, Flex e AIR.

O ActionScript é executado com a AVM (ActionScript Virtual Machine), que faz parte do Flash Player e do AIR. O código do ActionScript em geral é compilado no *formato de código de bytes* (um tipo de linguagem de programação escrita e entendida por computadores) por um compilador, como o criado no Adobe® Flash® CS4 Professional ou no Adobe® Flex™ Builder™ ou como o disponível no Adobe® Flex™ SDK. O código de bytes é incorporado aos arquivos SWF, que são executados pelo Flash Player e pelo AIR.

O ActionScript 3.0 oferece um modelo de programação robusto que parecerá familiar aos desenvolvedores com um conhecimento básico de programação orientada a objetos. Alguns dos recursos principais do ActionScript 3.0 que foram aprimorados em relação à versão anterior incluem:

- Uma nova ActionScript Virtual Machine, chamada AVM2, que usa um novo conjunto de instruções de código de bytes e fornece aprimoramentos de desempenho significativos
- Uma base de código de compilador moderna que executa otimizações mais avançadas do que as versões anteriores do compilador
- Uma API (Interface de programação de aplicativo) expandida e aprimorada, com controle de baixo nível de objetos e um autêntico modelo orientado a objetos
- Uma API XML baseada na especificação de linguagem ECMAScript para XML (E4X) (ECMA-357 edição 2) E4X é a extensão de linguagem para ECMAScript que adiciona XML como um tipo de dados nativo da linguagem.
- Um modelo de evento baseado na Especificação de eventos DOM (Document Object Model) nível 3

Vantagens do ActionScript 3.0

O ActionScript 3.0 vai além dos recursos de script de suas versões anteriores. Ele foi criado para facilitar a criação de aplicativos altamente complexos com grandes conjuntos de dados e bases de código reutilizáveis orientadas a objetos. Embora o ActionScript 3.0 não seja necessário para o conteúdo executado no Adobe Flash Player, ele permite melhorias de desempenho que só estão disponíveis com a AVM2, a nova máquina virtual. O código do ActionScript 3.0 pode ser executado até 10 vezes mais rápido do que o código do ActionScript existente.

A versão antiga da AVM1 (ActionScript Virtual Machine) executa os códigos ActionScript 1.0 e ActionScript 2.0. As versões 9 e 10 do Flash Player oferecem suporte a AVM1 para compatibilidade com conteúdo existente e herdado de versões anteriores. Para obter mais informações, consulte “Compatibilidade com versões anteriores” na página 7.

Novidades do ActionScript 3.0

Embora contenha muitas classes e recursos que parecerão familiares aos programadores do ActionScript, em termos de arquitetura e conceito, o ActionScript 3.0 é diferente de suas versões anteriores. Os aprimoramentos do ActionScript 3.0 incluem novos recursos da linguagem central e uma API do Flash Player melhorada que fornece mais controle sobre objetos de baixo nível.

Nota: *Os aplicativos do Adobe® AIR™ também podem usar as APIs do Flash Player.*

Recursos da linguagem central

A linguagem central define os blocos de construção básicos da linguagem de programação, como instruções, expressões, condições, loops e tipos. O ActionScript 3.0 contém vários recursos novos que aceleram o processo de desenvolvimento.

Exceções de tempo de execução

O ActionScript 3.0 relata mais condições de erros que suas versões anteriores. As exceções de tempo de execução são usadas para condições de erro comuns, melhorar a experiência de depuração e permitir o desenvolvimento de aplicativos que manipulam erros de forma robusta. Os erros de tempo de execução fornecem rastreamentos de pilha anotados com informações sobre o arquivo de origem e o número de linha, ajudando a detectar os erros rapidamente.

Tipos de tempo de execução

No ActionScript 2.0, as anotações de tipo eram basicamente um recurso de desenvolvedor; em tempo de execução, todos os valores são tipificados dinamicamente. No ActionScript 3.0, as informações de tipo são preservadas em tempo de execução e usadas para diversos fins. O Flash Player e o Adobe AIR fazem a verificação de tipos em tempo de execução, melhorando a segurança de tipos do sistema. As informações sobre tipo também são usadas para retratar variáveis em representações, melhorando o desempenho e reduzindo o uso de memória.

Classes seladas

O ActionScript 3.0 apresenta o conceito de classes seladas. Uma classe selada possui apenas o conjunto fixo de propriedades e métodos que foram definidos em tempo de compilação e não é possível adicionar outros. Ela permite uma verificação em tempo de compilação mais rígida, resultando em programas mais robustos. Ela também melhora o uso de memória por não exigir uma tabela de hash interna para cada ocorrência de objeto. As classes dinâmicas também são possíveis usando a palavra-chave `dynamic`. Todas as classes no ActionScript 3.0 são seladas por padrão, mas podem ser declaradas para se tornar dinâmicas com a palavra-chave `dynamic`.

Fechamentos de método

O ActionScript 3.0 permite um fechamento de método que lembra automaticamente de sua ocorrência de objeto original. Esse recurso é útil para a manipulação de eventos. No ActionScript 2.0, os fechamentos de método não lembravam de qual ocorrência de objeto tinham sido extraídos, gerando um comportamento inesperado quando o fechamento de método era invocado. A classe mx.utils.Delegate era uma solução conhecida, mas não é mais necessária.

ECMAScript para XML (E4X)

O ActionScript 3.0 implementa o ECMAScript para XML (E4X), recentemente padronizado como ECMA-357. O E4X oferece um conjunto fluente de construções de linguagem para manipular XML. Diferentemente das APIs tradicionais de análise de XML, o XML com E4X funciona como um tipo de dados nativo da linguagem. O E4X simplifica o desenvolvimento de aplicativos que manipulam XML, reduzindo drasticamente a quantidade de código necessária. Para obter mais informações sobre a implementação do E4X do ActionScript 3.0, consulte "Trabalho com XML" na página 230.

Para exibir a especificação E4X do ECMA, vá para www.ecma-international.org.

Expressões regulares
O ActionScript 3.0 inclui suporte nativo para expressões regulares, o que permite pesquisar e manipular seqüências de caracteres rapidamente. Ele implementa o suporte a expressões regulares conforme definidas na especificação de linguagem ECMAScript (ECMA-262) edição 3.

Espaços para nomes
Os espaços para nomes são semelhantes aos especificadores de acesso tradicionais usados para controlar a visibilidade de declarações (`public`, `private`, `protected`). Eles funcionam como especificadores de acesso personalizados, que podem ter os nomes que você escolher. Os espaços para nomes são equipados com um URI (Identificador Universal de Recursos) para evitar colisões e também são usados para representar nomes para espaços XML no trabalho com E4X.

Novos tipos primitivos
O ActionScript 2.0 tem um único tipo numérico, Number, um número de ponto flutuante de precisão dupla. O ActionScript 3.0 contém os tipos int e uint. O tipo int é um inteiro assinado de 32 bits que permite ao código ActionScript aproveitar os rápidos recursos matemáticos de inteiros da CPU. O tipo int é útil para contadores de loop e variáveis em que os inteiros são usados. O tipo uint é um tipo inteiro de 32 bits não assinado, útil para valores de cores RGB, contagens de bytes etc.

Recursos da API do Flash Player

As APIs do Flash Player no ActionScript 3.0 contêm várias classes que permitem controlar objetos em um nível baixo. A arquitetura da linguagem foi projetada para ser mais intuitiva do que a das outras versões. As novas classes são tantas que não cabe tratar em detalhes aqui, mas as seções a seguir destacam algumas alterações significativas.

Nota: *Os aplicativos do Adobe® AIR™ também podem usar as APIs do Flash Player.*

Modelo de eventos DOM3
O modelo de eventos DOM (Document Object Model) nível 3 fornece um meio padrão de gerar e manipular mensagens de eventos para que os objetos nos aplicativos possam interagir e se comunicar, mantendo seu estado e respondendo a alterações. Padronizado segundo as Especificações de eventos DOM nível 3 do World Wide Web Consortium, este modelo fornece um mecanismo mais claro e eficiente do que os sistemas de eventos disponíveis nas versões anteriores do ActionScript.

Os eventos e eventos de erros estão localizados no pacote flash.events. A estrutura dos componentes Flash usa o mesmo modelo de eventos que a API do Flash Player, por isso o sistema de eventos é unificado na plataforma Flash.

API de lista de exibição
A API para acessar a lista de exibição do Flash Player e do Adobe AIR (a árvore que contém os elementos visuais no aplicativo) consiste em classes para trabalhar com primitivas visuais.

A nova classe Sprite é um bloco de construção leve, semelhante à classe MovieClip, porém mais apropriada como uma classe base para componentes da UI. A nova classe Shape representa formas de vetor brutas. Essas classes podem ser instanciadas naturalmente com o operador `new` e atribuídas a um pai dinamicamente a qualquer momento.

O gerenciamento de profundidade agora é automático e incorporado ao Flash Player e ao Adobe AIR, tornando desnecessária a atribuição de números de profundidade. Novos métodos foram fornecidos para especificar e gerenciar a ordem z de objetos.

Manipulação de dados e conteúdo dinâmicos

O ActionScript 3.contém mecanismos para carregar e manipular ativos e dados no seu aplicativo que são intuitivos e consistentes na API. A nova classe Loader fornece um único mecanismo para carregar arquivos SWF e ativos de imagem e fornece uma forma de acessar informações detalhadas sobre o conteúdo carregado. A classe URLLoader fornece um mecanismo separado para carregar texto e dados binários em aplicativos orientados a dados. A classe Socket fornece um meio de ler e gravar dados binários nos soquetes do servidor em qualquer formato.

Acesso a dados de baixo nível

Varias APIs fornecem acesso de baixo nível a dados que nunca antes estiveram disponíveis no ActionScript. Para dados obtidos por download, a classe URLStream, que é implementada por URLLoader, fornece acesso a dados como dados binários brutos enquanto estão sendo baixados. A classe ByteArray permite otimizar a leitura, a gravação e o trabalho com dados binários. A nova API Sound fornece controle detalhado de som por meio das classes SoundChannel e SoundMixer. Novas APIs que lidam com segurança fornecem informações sobre os privilégios de segurança de um arquivo SWF ou conteúdo carregado, permitindo manipular melhor os erros de segurança.

Trabalho com texto

O ActionScript 3.0 contém um pacote flash.text para todas as APIs relacionadas a texto. A classe TextLineMetrics fornece uma métrica detalhada para uma linha de texto em um campo de texto; ela substitui o método `TextFormat.getTextExtent()` no ActionScript 2.0. A classe TextField contém vários novos métodos de baixo nível interessantes que podem fornecer informações específicas sobre uma linha de texto ou um único caractere em um campo de texto. Esses métodos incluem `getCharBoundaries()`, que retorna um retângulo representando a caixa delimitadora de um caractere, `getCharIndexAtPoint()`, que retorna o índice do caractere em um ponto especificado, e `getFirstCharInParagraph()`, que retorna o índice do primeiro caractere em um parágrafo. Os métodos de nível de linha incluem `getLineLength()`, que retorna o número de caracteres em uma linha de texto especificada, e `getLineText()`, que retorna o texto da linha especificada. Uma nova classe Font fornece um meio de gerenciar fontes incorporadas em arquivos SWF.

Compatibilidade com versões anteriores

Como sempre, o Flash Player fornece compatibilidade total com conteúdo publicado em versões anteriores. Qualquer conteúdo executado em versões anteriores do Flash Player pode ser executado no Flash Player 9 e em versões posteriores. A introdução do ActionScript 3.0 no Flash Player 9, contudo, apresenta alguns desafios de interoperabilidade entre o conteúdo antigo e o novo executado no Flash Player 9 ou posterior. Esses problemas de compatibilidade incluem os seguinte:

- Um único arquivo SWF não pode combinar o código do ActionScript 1.0 ou 2.0 com o código do ActionScript 3.0.
- O código do ActionScript 3.0 pode ser carregado em um arquivo SWF escrito no ActionScript 1.0 ou 2.0, mas não pode acessar as variáveis e funções do arquivo SWF.
- Os arquivos SWF escritos no ActionScript 1.0 ou 2.0 não podem carregar arquivos SWF escritos no ActionScript 3.0. Isso significa que os arquivos SWF criados no Flash 8 ou no Flex Builder 1.5 ou versões anteriores não podem carregar arquivos SWF do ActionScript 3.0.

  A única exceção a essa regra é que um arquivo SWF do ActionScript 2.0 pode ser substituído por um arquivo SWF do ActionScript 3.0, contanto que não tenha carregado nada antes em qualquer um de seus níveis. Um arquivo SWF do ActionScript 2.0 pode fazer isso por meio de uma chamada a `loadMovieNum()`, transmitindo um valor de 0 ao parâmetro `level`.

- Em geral, os arquivos SWF escritos no ActionScript 1.0 ou 2.0 devem ser migrados para trabalhar com os arquivos SWF escritos no ActionScript 3.0. Por exemplo, digamos que você criou um player de mídia usando o ActionScript 2.0. Ele carrega vários conteúdos que também foram criados usando o ActionScript 2.0. Se você criar um novo conteúdo no ActionScript 3.0, não poderá carregá-lo no player de mídia. Será necessário migrar o player de vídeo para o ActionScript 3.0.

  Se, no entanto, você criar um player de mídia no ActionScript 3.0, ele poderá executar carregamentos simples do seu conteúdo do ActionScript 2.0.

As tabelas a seguir resumem as limitações das versões anteriores do Flash Player em relação ao carregamento de novo conteúdo e execução de código, bem como as limitações de scripts entre arquivos SWF escritos em versões diferentes do ActionScript.

Funcionalidade com suporte	**Flash Player 7**	**Flash Player 8**	**Flash Player 9 e 10**
Pode carregar SWFs publicados para	7 e anterior	8 e anterior	9 (ou 10) e anterior
Contém este AVM	AVM1	AVM1	AVM1 e AVM2
Executa SWFs gravados no ActionScript	1.0 e 2.0	1.0 e 2.0	1.0, 2.0 e 3.0

Na tabela a seguir, "Funcionalidade com suporte" refere-se ao conteúdo executado no Flash Player 9 ou posterior. O conteúdo executado no Flash Player 8 ou anterior só pode ser carregar, exibir, executar e cruzar scripts no ActionScript 1.0 e 2.0.

Funcionalidade com suporte	**Conteúdo criado no ActionScript 1.0 e 2.0**	**Conteúdo criado no ActionScript 3.0**
Pode carregar conteúdo e executar código no conteúdo criado no	ActionScript 1.0 e 2.0 apenas	ActionScript 1.0, 2.0 e 3.0
Pode cruzar conteúdo de script criado no	ActionScript 1.0 e 2.0 apenas (ActionScript 3.0 até LocalConnection)	ActionScript 1.0 e 2.0 até LocalConnection. ActionScript 3.0

Capítulo 3: Introdução do ActionScript

Este capítulo foi criado para oferecer uma introdução à programação do ActionScript e o conhecimento necessário para que você entenda os conceitos e exemplos do resto deste manual. Começaremos com uma discussão sobre os conceitos básicos de programação, descritos no contexto de sua aplicação ao ActionScript. Também abordaremos os fundamentos de como organizar e criar um aplicativo do ActionScript.

Fundamentos de programação

Como o ActionScript é uma linguagem de programação, para conhecê-lo, primeiro é necessário compreender alguns conceitos gerais de programação de computador.

O que os programas de computador fazem

Em primeiro lugar, é bom saber o que é um programa de computador e o que ele faz. Um programa de computador consiste em dois aspectos principais:

- Ele é uma série de instruções ou etapas que o computador deve executar.
- Cada etapa envolve a manipulação de algumas informações ou dados.

Em termos gerais, um programa de computador é apenas uma série de comandos passo a passo que você fornece ao computador e ele executa. Cada comando é conhecido como *instrução*. Como você verá neste manual, no ActionScript, cada instrução é escrita com um ponto-e-vírgula no final.

Em essência, tudo o que uma determinada instrução faz em um programa é manipular alguns dados que estão armazenados na memória do computador. Em um caso simples, você pode instruir o computador a adicionar dois números e armazenar o resultado na memória. Em um caso mais complexo, imagine que existe um desenho de um retângulo em um lugar na tela e você quer escrever um programa a fim de movê-lo para outro lugar. O computador está mantendo controle de determinadas informações sobre o retângulo (as coordenadas x, y nas quais ele está localizado, sua largura, altura, cor etc.). Cada uma dessas informações está armazenada em um local na memória do computador. Um programa que move o retângulo para um local diferente deve ter etapas como "alterar a coordenada x para 200; alterar a coordenada y para 150" (ou seja, deve especificar novos valores a serem usados para as coordenadas x e y). É claro que o computador faz alguma coisa com esses dados para realmente transformar os números na imagem que aparece na tela; mas, para o nível de detalhe que nos interessa, basta saber que o processo de "mover um retângulo na tela", na verdade, implica em alterar alguns dados na memória do computador.

Variáveis e constantes

Como a programação envolve principalmente a alteração de algumas informações na memória do computador, é preciso encontrar um meio de representar uma informação isolada no programa. Uma *variável* é um nome que representa um valor na memória do computador. Durante a escrita de instruções para manipular valores, o nome da variável é escrito no lugar do valor; sempre que se deparar com o nome da variável no seu programa, o computador consultará a memória e usará o valor que encontrar nela. Por exemplo, se você tiver duas variáveis chamadas `value1` e `value2`, cada uma contendo um número, para adicionar esses dois números, você pode escrever a instrução:

```
value1 + value2
```

Quando executar as etapas, o computador verificará os valores de cada variável e os adicionará juntos.

No ActionScript 3.0, uma variável consiste em três partes diferentes:

- O nome da variável
- O tipo de dados que pode ser armazenado nela
- O valor real armazenado na memória do computador

Acabamos de ver como o computador usa o nome como alocador de espaço para o valor. O tipo de dados também é importante. Durante a criação de uma variável no ActionScript, você especifica o tipo de dados que ela conterá; a partir daí, as instruções do programa só podem armazenar esse tipo de dados na variável, e você pode manipular o valor usando as características específicas associadas a esse tipo de dados. No ActionScript, a criação de uma variável (conhecida como *declarar* a variável) requer o uso da instrução `var`:

```
var value1:Number;
```

Nesse caso, instruímos o computador a criar uma variável chamada `value1`, que contém apenas dados Number ("Number" é um tipo de dados específico definido no ActionScript). Você também pode armazenar um valor na variável imediatamente:

```
var value2:Number = 17;
```

No Adobe Flash CS4 Professional, existe outro meio de declarar uma variável. Durante a colocação de um símbolo de clipe de filme, símbolo de botão ou campo de texto no Palco, você pode lhe dar um nome de ocorrência no Inspetor de propriedades. Por baixo do pano, o Flash cria uma variável com o mesmo nome que o nome de ocorrência, que você pode usar no código do ActionScript para fazer referência a esse item do Palco. Assim, por exemplo, se você der a um símbolo de clipe de filme no Palco o nome de ocorrência `rocketShip`, sempre que usar a variável `rocketShip` no código do ActionScript, na verdade, estará manipulando esse clipe de filme.

Uma constante é muito semelhante a uma variável no sentido de que é um nome que representa um valor na memória do computador, com um tipo de dados específico. A diferença é que um valor só pode ser atribuído a uma constante uma única vez no processamento do aplicativo do ActionScript. Assim que é atribuído, o valor da constante é o mesmo em todo o aplicativo. A sintaxe para declarar uma constante é a mesma que para declarar uma variável, exceto que você usa a palavra-chave `const` em vez da palavra-chave `var`:

```
const SALES_TAX_RATE:Number = 0.07;
```

Uma constante é útil para definir um valor que é usado em vários locais em um projeto e que não é alterado sob circunstâncias normais. O uso de uma constante em vez de um valor literal torna o código mais legível. Por exemplo, é mais fácil entender a finalidade de uma linha de código que multiplica um preço pela `SALES_TAX_RATE`, comparado a uma linha de código que multiplica o preço por `0,07`. Além disso, se o valor definido por uma constante tiver de ser alterado e você usar uma constante que represente esse valor em todo o seu projeto, bastará alterar o valor em um único lugar (a declaração da constante), em vez de alterá-lo em vários locais como aconteceria se você usasse valores literais codificados.

Tipos de dados

No ActionScript, há vários tipos de dados que você pode usar como os tipos de dados da variável que você criar. Alguns deles podem ser entendidos como "simples" ou "fundamentais":

- Seqüência de caracteres: um valor textual, como um nome ou o texto do capítulo de um livro
- Numérico: o ActionScript 3.0 inclui três tipos de dados específicos para dados numéricos:
  - Número: qualquer valor numérico, incluindo valores com ou sem uma fração
  - int: um inteiro (um número inteiro sem uma fração)
  - uint: um inteiro "sem sinal", que significa um número inteiro que não pode ser negativo

- Booleano: um valor do tipo verdadeiro ou falso, tal como se uma opção está ativa ou se dois valores são iguais

Os tipos de dados simples representam uma única informação: por exemplo, um único número ou uma única seqüência de texto. Entretanto, a maioria dos tipos de dados definidos no ActionScript poderia ser descrita como tipos de dados complexos, porque representam um conjunto de valores agrupados. Por exemplo, uma variável com o tipo de dados Date representa um valor único: um momento no tempo. No entanto, esse valor de data na verdade é representado com diversos valores: dia, mês, ano, horas, minutos, segundos etc., que são todos números individuais. Portanto, embora pensemos em uma data como um valor único (e a tratemos dessa forma, criando uma variável Date), internamente, o computador a entende como um grupo de diversos valores que, juntos, definem uma única data.

A maioria dos tipos de dados embutidos, bem como os definidos pelos programadores, são tipos de dados complexos. Alguns tipos de dados complexos que talvez você conheça são:

- MovieClip: um símbolo de clipe de filme
- TextField: um campo de texto de entrada ou dinâmico
- SimpleButton: um símbolo de botão
- Date: informações sobre um momento único no tempo (uma data e hora)

Duas palavras que, em geral, são usadas como sinônimos de tipos de dados são classe e objeto. Uma *classe* é simplesmente a definição de um tipo de dados; é como um modelo para todos os objetos dos tipos de dados, é como dizer "todas as variáveis do tipo de dados Example têm estas características: A, B e C". Um *objeto*, porém, é apenas uma ocorrência real de uma classe; uma variável cujo tipo de dados é MovieClip poderia ser descrita como um objeto MovieClip. Estas são formas diferentes de dizer a mesma coisa:

- O tipo de dados da variável `myVariable` é Number.
- A variável `myVariable` é uma ocorrência de Number.
- A variável `myVariable` é um objeto Number.
- A variável `myVariable` é uma ocorrência da classe Number.

Trabalho com o objetos

O ActionScript é conhecido como uma linguagem de programação orientada a objetos. Programação orientada a objetos é simplesmente uma abordagem da programação, nada mais do que uma forma de organizar o código em um programa, usando objetos.

Anteriormente, definimos um programa de computador como uma série de etapas ou instruções que o computador executa. De forma conceitual, então, podemos imaginar um programa de computador como uma longa lista de instruções. Entretanto, na programação orientada a objetos, as instruções do programa são divididas em objetos diferentes, ou seja, o código é agrupado em blocos de funcionalidade, de forma que tipos relacionados de funcionalidade ou partes relacionadas de informação são agrupados em um único contêiner.

Na verdade, se você já trabalhou com símbolos no Flash, então está acostumado a trabalhar com objetos. Imagine que você definiu um símbolo de clipe de filme (digamos que seja o desenho de um retângulo) e colocou uma cópia dele no Palco. Esse símbolo de clipe de filme também é (literalmente) um objeto no ActionScript; é uma ocorrência da classe MovieClip.

Há diversas características do clipe de filme que você pode modificar. Por exemplo, quando ele é selecionado, há valores que você pode alterar no Inspetor de propriedades, como a coordenada x, sua largura ou vários ajustes de cor, como alterar seu alfa (transparência) ou aplicar um filtro de sombra. Outras ferramentas do Flash permitem fazer mais alterações, como usar a ferramenta Transformação livre para girar o retângulo. Tudo o que é possível fazer para modificar um símbolo de clipe de filme no ambiente de autoria do Flash também pode ser feito no ActionScript, alterando as partes dos dados que são colocadas em um único pacote chamado objeto MovieClip.

Na programação orientada a objetos do ActionScript, há três tipos de características que qualquer classe pode incluir:

- Propriedades
- Métodos
- Eventos

Juntos, esses elementos ajudam a gerenciar as partes dos dados usadas pelo programa e a decidir quais ações são executadas em uma determinada ordem.

Propriedades

Uma propriedade representa uma das partes dos dados que são compactados em um objeto. Um objeto de música pode ter propriedades chamadas `artist` e `title`; a classe MovieClip tem propriedades como `rotation`, `x`, `width` e `alpha`. As propriedades são tratadas como variáveis individuais; na verdade, elas podem ser entendidas apenas como as variáveis "filho" contidas em um objeto.

A seguir, apresentamos alguns exemplos de código do ActionScript que usa propriedades. Esta linha de código move o MovieClip chamado `square` para 100 pixels na coordenada x:

```
square.x = 100;
```

Este código usa a propriedade de rotação fazendo com que o MovieClip `square` gire para corresponder à rotação do MovieClip `triangle`:

```
square.rotation = triangle.rotation;
```

Este código altera a escala horizontal do MovieClip `square` para que ele fique uma vez e meia maior do que antes:

```
square.scaleX = 1.5;
```

Observe a estrutura comum: você usa uma variável (`square`, `triangle`) como o nome do objeto, seguido de um ponto (`.`) e, depois, o nome da propriedade (`x`, `rotation`, `scaleX`). O ponto, conhecido como *operador ponto*, é usado para indicar que você está acessando um dos elementos filho de um objeto. A estrutura inteira reunida, "nome da variável-ponto-nome da propriedade", é usada como uma única variável, como um nome para um único valor na memória do computador.

Métodos

Um *method* é uma ação que pode ser executada por um objeto. Por exemplo, se você criou um símbolo de clipe de filme no Flash com vários quadros-chave e animação em sua linha de tempo, esse clipe de filme pode ser reproduzido, parar ou ser instruído a mover o indicador de reprodução para um quadro específico.

Este código instrui o MovieClip chamado `shortFilm` a iniciar a reprodução:

```
shortFilm.play();
```

Esta linha faz o MovieClip chamado `shortFilm` parar a reprodução (o indicador de reprodução pára; é como pausar um vídeo):

```
shortFilm.stop();
```

Este código faz um MovieClip chamado `shortFilm` mover o indicador de reprodução para o Quadro 1 e parar a reprodução (é como retroceder um vídeo):

```
shortFilm.gotoAndStop(1);
```

Como você pode ver, os métodos, assim como as propriedades, são acessados escrevendo o nome do objeto (uma variável), um ponto e o nome do método seguido de parênteses. Os parênteses são uma forma de indicar que você está *chamando* o método ou, em outras palavras, instruindo o objeto a executar aquela ação. Às vezes, os valores (ou as variáveis) são colocados entre parênteses, como uma forma de passar adiante informações adicionais necessárias para executar a ação. Esses valores são conhecidos como *parâmetros* de método. Por exemplo, o método `gotoAndStop()` precisa saber para qual quadro deve ir, por isso requer um único parâmetro entre parênteses. Outros métodos, como `play()` e `stop()`, são auto-explicativos, por isso não requerem informações extras. No entanto, eles ainda são escritos com parênteses.

Diferentemente das propriedades (e variáveis), os métodos não são usados como alocadores de espaço de valor. Entretanto, alguns métodos podem executar cálculos e retornar um resultado que pode ser usado como uma variável. Por exemplo, o método `toString()` da classe Number converte o valor em sua representação de texto:

```
var numericData:Number = 9;
var textData:String = numericData.toString();
```

Por exemplo, você usaria o método `toString()` se quisesse exibir o valor de uma variável Number em um campo de texto na tela. A propriedade `text` da classe TextField (que representa o conteúdo real do texto exibido na tela) é definida como uma String, por isso pode conter apenas valores de texto. Esta linha de código converte o valor numérico na variável `numericData` em texto e faz com que apareça na tela, no objeto TextField chamado `calculatorDisplay`:

```
calculatorDisplay.text = numericData.toString();
```

Eventos

Descrevemos um programa de computador como uma série instruções que o computador executa em etapas. Alguns programas de computador simples consistem em nada mais do que algumas etapas que ele executa e depois é encerrado. Entretanto, os programas do ActionScript são criados para continuar a execução e esperar a entrada do usuário ou a ocorrência de outras coisas. Os eventos são mecanismos que determinam quais instruções o computador executa em um determinado momento.

Basicamente, os *eventos* são fenômenos que acontecem, sobre os quais o ActionScript é informado e aos quais pode responder. Muitos eventos estão relacionados à interação do usuário, como quando um usuário clica em um botão ou pressiona uma tecla no teclado, mas também há outros tipos. Por exemplo, se você usar o ActionScript para carregar uma imagem externa, há um evento que poderá avisá-lo quando o carregamento da imagem for concluído. Em essência, quando um programa ActionScript é executado, o Adobe Flash Player e o Adobe AIR aguardam determinados eventos e, então, executam o código ActionScript específico definido para esses eventos.

Tratamento de eventos básico

A técnica para especificar determinadas ações que devem ser executadas em resposta a eventos específicos é conhecida como *tratamento de eventos*. Durante a escrita do código do ActionScript para executar o tratamento de eventos, há três elementos importantes a serem identificados:

- A origem do evento: para qual objeto o evento deverá ocorrer? Por exemplo, qual botão será clicado ou qual objeto Loader está carregando a imagem? A origem do evento também é conhecida como *destino do evento*, porque é o objeto para o qual o evento é destinado pelo Flash Player ou AIR (ou seja, onde o evento realmente acontece).
- O evento: o que irá acontecer, a qual fenômeno você deseja responder? É importante identificar isso, porque muitos objetos acionam diversos eventos.

- A resposta: qual ou quais etapas você deseja executar quando o evento acontecer?

Sempre que você escrever código do ActionScript para manipular eventos, ele deve incluir estes três elementos, e o código deve seguir a estrutura básica (os elementos em negrito são alocadores de espaço que você preenche conforme o caso específico):

```
function eventResponse(eventObject:EventType):void
{
    // Actions performed in response to the event go here.
}

eventSource.addEventListener(EventType.EVENT_NAME, eventResponse);
```

Esse código faz duas coisas. Primeiro, ele define uma função, que é a forma de especificar as ações que você deseja executar em resposta ao evento. Em seguida, o método `addEventListener()` do objeto de origem é chamado basicamente inscrevendo a função do evento especificado para que, quando o evento acontecer, as ações da função sejam executadas Vejamos cada uma dessas partes com mais detalhes.

Uma *função* fornece um meio de agrupar as ações, com um único nome, que é como um nome de atalho, para executar as ações. Ela é idêntica a um método, exceto que não está necessariamente associada a uma classe específica (na verdade, um método poderia ser definido como uma função associada a uma classe específica). Durante a criação de uma função para tratamento de eventos, você deve escolher o nome da função (chamada de `eventResponse` neste caso). Você também deve especificar um parâmetro (chamado `eventObject` neste exemplo). A especificação de um parâmetro de função é como declarar uma variável, por isso você também deve indicar o tipo de dados do parâmetro. (Neste exemplo, o tipo de dados do parâmetro é `EventType`.)

Cada tipo de evento que você deseja escutar é associado a uma classe do ActionScript. O tipo de dados que você define para o parâmetro de função é sempre a classe associada ao evento específico ao qual deseja responder. Por exemplo, um evento `click` (acionado quando o usuário clica em um item com o mouse) é associado à classe MouseEvent. Para escrever uma função de ouvinte para um evento `click`, você define essa função com um parâmetro com o tipo de dados MouseEvent. Finalmente, entre chaves ({ ... }), você escreve as instruções que deseja que o computador execute quando o evento ocorrer.

Depois de escrever a função de tratamento de eventos, é necessário informar ao objeto de origem de evento (o objeto para o qual o evento acontece, por exemplo, o botão) que você deseja que a função seja chamada quando o evento ocorrer. É possível fazer isso chamando o método `addEventListener()` desse objeto (todos os objetos que possuem eventos também têm um método `addEventListener()`). O método `addEventListener()` usa dois parâmetros:

- O primeiro é o nome do evento específico ao qual você deseja responder. Novamente, cada evento é afiliado a uma classe específica, e essa classe terá um valor especial predefinido para cada evento (como se fosse o próprio nome exclusivo do evento, que você deve usar para o primeiro parâmetro).
- O segundo é o nome da função de resposta do evento. Observe que um nome de função é escrito sem parênteses quando transmitido como um parâmetro.

Exame do processo de tratamento de eventos

Veja a seguir uma descrição passo a passo do processo que acontece durante a criação de um ouvinte de eventos. Neste caso, é um exemplo de criação de uma função de ouvinte que é chamada quando um objeto `myButton` é clicado.

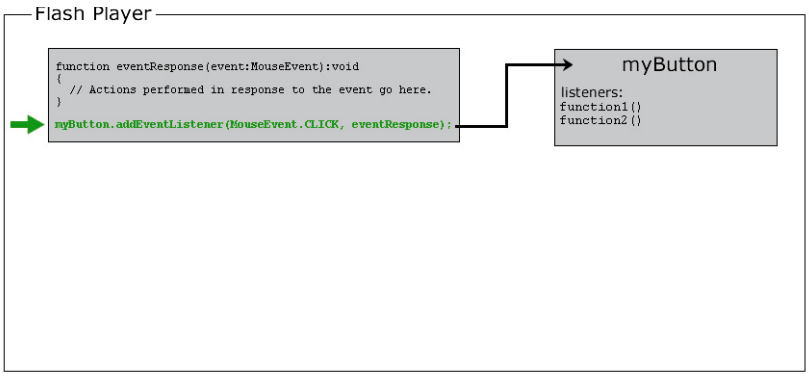
O código real escrito pelo programador é o seguinte:

```
function eventResponse(event:MouseEvent):void
{
    // Actions performed in response to the event go here.
}

myButton.addEventListener(MouseEvent.CLICK, eventResponse);
```

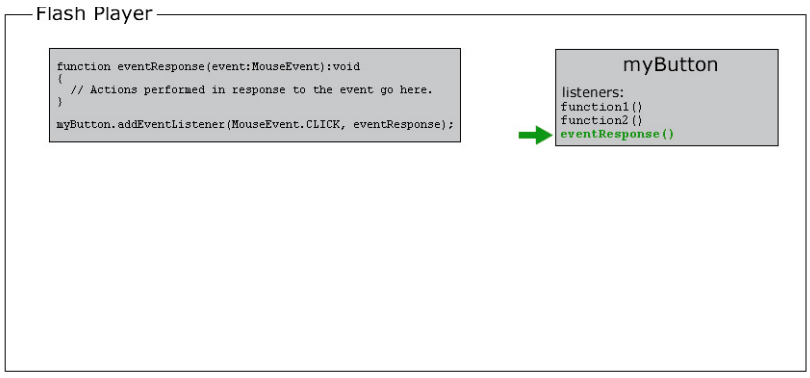
Apresentaremos a seguir o funcionamento desse código, quando executado no Flash Player. O comportamento é o mesmo no Adobe AIR:

1 Quando o arquivo SWF é carregado, o Flash Player registra o fato de que há uma função chamada `eventResponse()`.

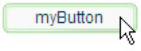
2 Em seguida, o Flash Player executa o código (especificamente, as linhas de código que não estão contidas em uma função). Neste caso, é apenas uma linha de código: chamar o método `addEventListener()` no objeto de origem do evento (chamado `myButton`) e transmitir a função `eventResponse` como um parâmetro.



a Internamente, `myButton` tem uma lista de funções que estão ouvindo cada um de seus eventos, por isso quando seu método `addEventListener()` é chamado, `myButton`armazena a função `eventResponse()` na lista de ouvintes de eventos.

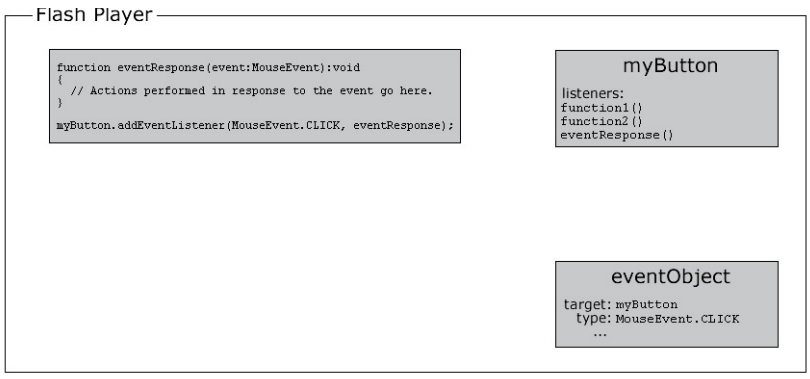


3 Em algum momento, o usuário clica no objeto `myButton`, acionando o evento `click` (identificado como `MouseEvent.CLICK` no código).

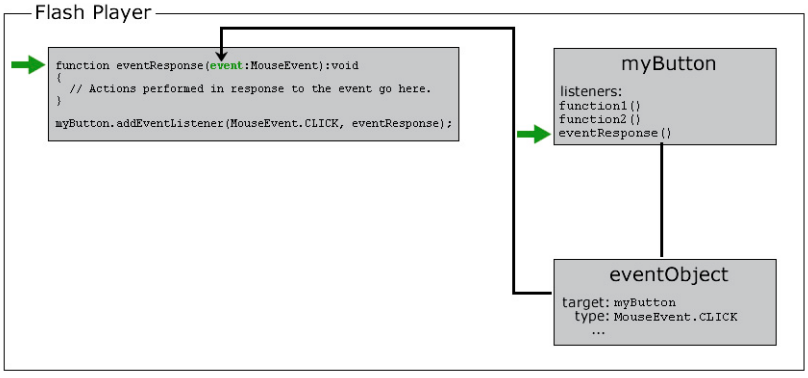


Então, acontece o seguinte:

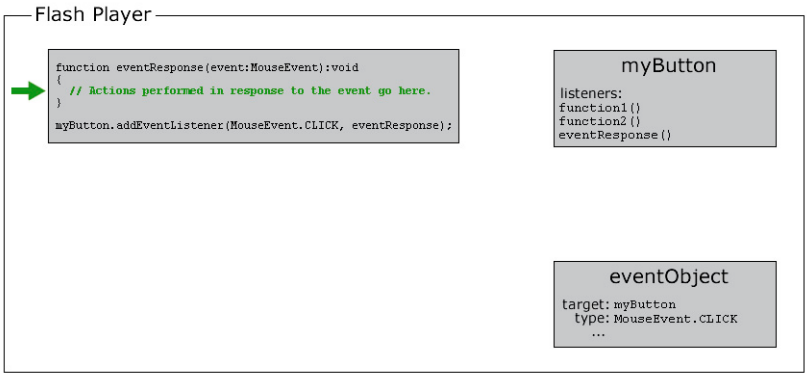
a O Flash Player cria um objeto, uma ocorrência da classe associada ao evento em questão (MouseEvent, no exemplo citado). Para muitos eventos, essa será uma ocorrência da classe Event; para eventos do mouse, será uma ocorrência de MouseEvent e, para outros eventos, será uma ocorrência da classe associada a esse evento. Esse objeto criado é conhecido como o *objeto de evento* e contém informações específicas sobre o evento que ocorreu: qual é o tipo de evento, onde ele ocorreu e outras informações específicas de evento se aplicável.



b Em seguida, o Flash Player verifica a lista de ouvintes de eventos armazenada por `myButton`. Ele verifica todas essas funções, chamando uma de cada vez e transmitindo o objeto de evento para a função como um parâmetro. Como a função `eventResponse()` é um dos ouvintes de `myButton`, como parte desse processo, o Flash Player chama a função `eventResponse()`.



c Quando a função `eventResponse()` é chamada, o código nela é executado e as ações especificadas são realizadas.



Exemplos de tratamento de eventos

A seguir, mostramos alguns exemplos mais concretos de eventos para lhe dar uma idéia de alguns elementos de eventos comuns e as possíveis variações disponíveis durante a escrita do código de tratamento de eventos:

- Clicar em um botão para iniciar a reprodução do clipe de filme atual. No exemplo a seguir, `playButton` é o nome da ocorrência do botão, e `this` é um nome especial que significa "o objeto atual":

```
this.stop();

function playMovie(event:MouseEvent):void
{
    this.play();
}

playButton.addEventListener(MouseEvent.CLICK, playMovie);
```

- Detectar tipos em um campo de texto: Neste exemplo, `entryText` é um campo de texto de entrada, e `outputText` é um campo de texto dinâmico:

```
function updateOutput(event:TextEvent):void
{
    var pressedKey:String = event.text;
    outputText.text = "You typed: " + pressedKey;
}

entryText.addEventListener(TextEvent.TEXT_INPUT, updateOutput);
```

- Clicar em um botão para navegar em uma URL. Nesse caso, `linkButton` é o nome de ocorrência do botão:

```
function gotoAdobeSite(event:MouseEvent):void
{
    var adobeURL:URLRequest = new URLRequest("http://www.adobe.com/");
    navigateToURL(adobeURL);
}

linkButton.addEventListener(MouseEvent.CLICK, gotoAdobeSite);
```

Criar ocorrências de objetos

Obviamente, para que você possa usar um objeto no ActionScript, primeiro, ele deve existir. Uma parte da criação de um objeto é declarar uma variável; porém, a declaração de uma variável só cria um lugar vazio na memória do computador. Você deve atribuir um valor real à variável, isto é, criar um objeto e armazená-lo na variável, antes de tentar usá-lo ou manipulá-lo. O processo de criar um objeto é conhecido como *instanciar* o objeto, em outras palavras, criar uma ocorrência de uma classe específica.

Existe uma maneira simples de criar uma ocorrência de objeto que não envolve o ActionScript. No Flash, quando você coloca um símbolo de clipe de filme, símbolo de botão ou campo de texto no Palco e lhe atribui um nome de ocorrência no Inspetor de propriedades, o Flash automaticamente declara uma variável com esse nome de ocorrência, cria uma ocorrência de objeto e armazena o objeto na variável. Da mesma forma, quando você usa a linguagem MXML para criar um componente no Adobe Flex Builder (codificando uma tag MXML ou colocando o componente no editor no modo Design) e lhe atribui uma ID (no markup MXML ou na exibição Propriedades do Flex), essa ID se torna o nome de uma variável do ActionScript, e uma ocorrência do componente é criada e armazenada nessa variável.

Entretanto, nem sempre você criará um objeto visualmente. Também há várias formas de criar ocorrências de objeto usando somente o ActionScript. Primeiro, com vários tipos de dados do ActionScript, você pode criar uma ocorrência usando uma *expressão literal*, um valor escrito diretamente no código do ActionScript. Eis alguns exemplos:

- Valor numérico literal (insere o número diretamente):

```
var someNumber:Number = 17.239;
var someNegativeInteger:int = -53;
var someUint:uint = 22;
```

- Valor da seqüência de caracteres literal (envolve o texto com aspas duplas):

```
var firstName:String = "George";
var soliloquy:String = "To be or not to be, that is the question...";
```

- Valor booleano literal (usa os valores literais `true` ou `false`):

```
var niceWeather:Boolean = true;
var playingOutside:Boolean = false;
```

- Valor Array literal (envolve uma lista de valores separados por vírgula entre colchetes):

```
var seasons:Array = ["spring", "summer", "autumn", "winter"];
```

- Valor XML literal (insere o XML diretamente):

```
var employee:XML = <employee>
        <firstName>Harold</firstName>
        <lastName>Webster</lastName>
    </employee>;
```

O ActionScript também define expressões literais para os tipos de dados Array, RegExp, Object e Function. Para obter detalhes sobre essas classes, consulte "Trabalho com matrizes" na página 157, "Uso de expressões regulares" na página 209 e "Tipo de dados Object" na página 60.

Para qualquer outro tipo de dados, para criar uma ocorrência de objeto, use o operador `new` como o nome da classe, desta forma:

```
var raceCar:MovieClip = new MovieClip();
var birthday:Date = new Date(2006, 7, 9);
```

O ato de criar de um objeto usando o operador `new`, muitas vezes, é descrito como "chamar o construtor da classe" Um *construtor* é um método especial que é chamado como parte do processo de criar uma ocorrência de uma classe. Observe que, ao criar uma ocorrência dessa forma, você coloca parênteses depois do nome da classe e, às vezes, especifica valores de parâmetro (duas ações que também são executadas ao chamar um método).

Observe que, mesmo para esses tipos de dados que permitem criar ocorrências usando uma expressão literal, você ainda pode usar o operador `new` para criar uma ocorrência de objeto. Por exemplo, as duas linhas de código a seguir fazem exatamente a mesma coisa:

```
var someNumber:Number = 6.33;
var someNumber:Number = new Number(6.33);
```

É importante se familiarizar com a forma como `new` *ClassName*`()` cria objetos. Se for necessário criar uma ocorrência de qualquer tipo de dados ActionScript que não tenha uma representação visual (e não possa ser criada colocando um item no Palco do Flash ou no modo Design do editor XML do Flex Builder), isso só será possível criando o objeto diretamente no ActionScript pelo uso do operador `new`.

No Flash especificamente, o operador `new` também pode ser usado para criar uma ocorrência de um símbolo de clipe de filme que é definido na Biblioteca mas não é colocado no Palco. Para obter mais informações sobre isso, consulte "Criação de objetos MovieClip com o ActionScript" na página 413.

Elementos de programa comuns

Além de declarar variáveis, criar ocorrências de objetos e manipular objetos usando suas propriedades e métodos, há outros blocos de construção que você precisa usar para criar um programa do ActionScript.

Operadores

Os *operadores* são símbolos (ou, ocasionalmente, palavras) essenciais que são usados para executar cálculos. Eles são muito usados em operações matemáticas e também para comparar valores. Como regra geral, um operador usa um ou mais valores e "obtém" um único resultado. Por exemplo:

- O operador de adição (`+`) adiciona dois valores, tendo como resultado um único número:

  ```
  var sum:Number = 23 + 32;
  ```

- O operador de multiplicação (`*`) multiplica dois valores, tendo como resultado um único número:

  ```
  var energy:Number = mass * speedOfLight * speedOfLight;
  ```

- O operador de igualdade (`==`) compara se dois valores são iguais, tendo como resultado um único valor verdadeiro ou falso (booleano):

  ```
  if (dayOfWeek == "Wednesday")
  {
      takeOutTrash();
  }
  ```

  Conforme mostrado aqui, o operador de igualdade e os outros operadores de "comparação", em geral, são usados com a instrução `if` para determinar se algumas instruções devem ser executadas.

Para obter detalhes e exemplos do uso de operadores, consulte "Operadores" na página 70.

Comentários

Durante a escrita do ActionScript, muitas vezes, você desejará fazer anotações, talvez para explicar como algumas linhas de código funcionam e por que escolheu uma determinada opção. Os *comentários de código* são uma ferramenta que você pode usar para escrever um texto que o computador deve ignorar no seu código. O ActionScript inclui dois tipos de comentários:

- Comentário de uma linha: um comentário de uma linha é criado colocando duas barras em qualquer lugar de uma linha. Tudo que vier depois das barras até o fim da linha será ignorado pelo computador:

```
// This is a comment; it's ignored by the computer.
var age:Number = 10; // Set the age to 10 by default.
```

- Comentário de várias linhas: um comentário de várias linhas inclui um marcador de comentário inicial (`/*`), o conteúdo do comentário e um marcador de comentário final (`*/`). Tudo entre os marcadores inicial e final é ignorado pelo computador, independentemente do número de linhas que o comentário contém:

```
/*
This might be a really long description, perhaps describing what
a particular function is used for or explaining a section of code.

In any case, these lines are all ignored by the computer.
*/
```

Outro uso comum dos comentários é "desativar" temporariamente uma ou mais linhas de código, por exemplo, se você estiver testando uma forma diferente de fazer algo ou tentando descobrir por que um código do ActionScript não está funcionando conforme o esperado.

Controle do fluxo

Muitas vezes em um programa, você desejará repetir determinadas ações, executar apenas algumas e outras não, executar ações conforme condições específicas etc. O *controle de fluxo* é o controle sobre as ações que são executadas. Há vários tipos de elementos de controle de fluxo disponíveis no ActionScript.

- Funções: as funções são como atalhos: fornecem um meio de agrupar uma série de ações sob um único nome e podem ser usadas para realizar cálculos. As funções são especialmente importantes para tratar eventos, mas também são usadas como uma ferramenta geral para agrupar uma série de instruções. Para obter mais informações sobre funções, consulte “Funções” na página 80.
- Loops: as estruturas de loop permitem designar um conjunto de instruções que o computador executará por um determinado número de vezes ou até que alguma condição seja alterada. Com freqüência, os loops são usados para manipular vários itens relacionados, empregando uma variável cujo valor é alterado sempre que o computador completa o loop. Para obter mais informações sobre loops, consulte “Repetição” na página 77.
- Instruções condicionais: as instruções condicionais fornecem um meio de designar algumas instruções que são executadas somente sob determinadas circunstâncias ou de fornecer conjuntos alternativos de instruções para condições diferentes. O tipo mais comum de instrução condicional é a instrução `if`. A instrução `if` verifica um valor ou uma expressão entre parênteses. Se o valor for `true`, as linhas de código entre chaves são executadas; caso contrário, elas são ignoradas. Por exemplo:

```
if (age < 20)
{
    // show special teenager-targeted content
}
```

  A parceira da instrução `if`, a instrução `else`, permite designar instruções alternativas a serem executadas caso a condição não seja `true`:

```
if (username == "admin")
{
    // do some administrator-only things, like showing extra options
}
else
{
    // do some non-administrator things
}
```

Para obter mais informações sobre instruções condicionais, consulte “Condicionais” na página 76.

Exemplo: Peça de portfólio de animação

Este exemplo foi criado para oferecer-lhe uma oportunidade de ver pela primeira vez como é possível juntar partes do ActionScript para obter um aplicativo completo, se não robusto, do ActionScript. A peça de portfólio de animação é um exemplo de como você pode usar uma animação linear existente (por exemplo, uma peça criada para um cliente) e adicionar alguns elementos interativos menores apropriados para incorporar essa animação em um portfólio on-line. O comportamento interativo que adicionaremos à animação incluirá dois botões nos quais o espectador poderá clicar: um para iniciar a animação e outro para navegar em uma URL separada (como o menu do portfólio ou a home page do autor).

O processo de criar essa peça pode ser dividido nestas seções principais:

1. Preparar o arquivo FLA para adicionar elementos interativos e do ActionScript.
2. Criar e adicionar os botões.
3. Escrever o código do ActionScript.
4. Testar o aplicativo.

Preparação para adicionar interatividade

Antes que possamos adicionar elementos interativos à nossa animação, é bom configurar o arquivo FLA criando alguns locais para adicionar o novo conteúdo. Isso inclui a criação de espaço real no Palco onde os botões podem ser colocados e também a criação de "espaço" no arquivo FLA para manter separados os itens diferentes.

Para configurar o FLA para adicionar elementos interativos:

1. Se você ainda não tiver uma animação linear à qual irá adicionar a interatividade, crie um novo arquivo FLA com uma animação simples como uma interpolação de movimento ou uma interpolação de forma. Caso contrário, abra o arquivo FLA contendo a animação que você está exibindo no projeto e salve-o com um novo nome para criar um novo arquivo funcional.
2. Decida o local na tela em que você deseja que os dois botões apareçam (um para iniciar a animação e outro para vincular ao portfólio do autor ou à home page). Se necessário, limpe o Palco ou adicione espaço para esse novo conteúdo. Se a animação já não tiver uma, você pode querer criar uma tela de abertura no primeiro quadro (provavelmente, você deslocará a animação para que ela comece no Quadro 2 ou depois).
3. Adicione uma nova camada, acima das outras camadas na Linha de tempo, e renomei-a como **buttons**. Essa será a camada na qual você adicionará os botões.
4. Adicione uma nova camada, acima das camadas de botões, e renomei-a como **actions**. Nela, você adicionará o código do ActionScript para seu aplicativo.

Criação e adição de botões

Em seguida, precisamos realmente criar e posicionar os botões que serão a parte central do nosso aplicativo interativo.

Para criar e adicionar botões ao FLA:

1. Usando as ferramentas de desenho, crie a aparência visual do seu primeiro botão (o botão "reproduzir") na camada de botões. Por exemplo, você pode desenhar uma figura oval na horizontal com texto em cima.
2. Usando a ferramenta Seleção, selecione todas as partes gráficas do botão sozinho.
3. No menu principal, escolha Modificar > Converter em símbolo.
4. Na caixa de diálogo, escolha Botão como o tipo de símbolo, dê um nome ao símbolo e clique em OK.
5. Com o botão selecionado, no Inspetor de propriedades, dê ao botão o nome de ocorrência **playButton**.
6. Repita as etapas de 1 a 5 para criar o botão que levará o espectador à home page do autor. Chame este botão de **homeButton**.

Gravação do código

O código do ActionScript para este aplicativo pode ser dividido em três conjuntos de funcionalidade, mas todos serão inseridos no mesmo lugar. As três coisas que o código precisa fazer são:

- Parar o indicador de reprodução assim que o arquivo SWF carregar (quando o indicador de reprodução entrar no Quadro 1).
- Monitorar um evento para iniciar a reprodução do arquivo SWF quando o usuário clicar no botão de reprodução.
- Monitorar um evento para enviar o navegador à URL apropriada quando o usuário clicar no botão da home page do autor.

Para criar um código que pare o indicador de reprodução quando ele entrar no Quadro 1:

1. Selecione o quadro-chave no Quadro 1 da camada de ações.
2. Para abrir o painel Ações, no menu principal, escolha Janela > Ações.
3. No painel Script, digite o seguinte código:

```
stop();
```

Para escrever código para iniciar a animação quando o botão de reprodução for clicado:

1. No fim do código digitado nas etapas anteriores, adicione duas linhas vazias.
2. Digite o seguinte código na parte inferior do script:

```
function startMovie(event:MouseEvent):void
{
    this.play();
}
```

   Este código define uma função chamada `startMovie()`. Quando `startMovie()` é chamado, elefaz com que a linha de tempo principal comece a reproduzir.

3. Na linha seguinte ao código adicionado na etapa anterior, digite esta linha de código:

```
playButton.addEventListener(MouseEvent.CLICK, startMovie);
```

   Esta linha de código registra a função `startMovie()` como um ouvinte para o evento `click` de `playButton`. Em outras palavras, com ela, sempre que o botão chamado `playButton` é clicado, a função `startMovie()` é chamada.

Para escrever o código por meio do qual o navegador acessa uma URL quando o botão da home page for clicado:

1 No fim do código digitado nas etapas anteriores, adicione duas linhas vazias.

2 Digite este código na parte inferior do script:

```
function gotoAuthorPage(event:MouseEvent):void
{
    var targetURL:URLRequest = new URLRequest("http://example.com/");
    navigateToURL(targetURL);
}
```

Este código define uma função chamada `gotoAuthorPage()`. Essa função primeiro cria uma ocorrência de URLRequest representando a URL http://example.com/ e depois repassa essa URL para a função `navigateToURL()`, fazendo o navegador do usuário abrir nessa URL.

3 Na linha seguinte ao código adicionado na etapa anterior, digite esta linha de código:

```
homeButton.addEventListener(MouseEvent.CLICK, gotoAuthorPage);
```

Esta linha de código registra a função `gotoAuthorPage()` como um ouvinte para o evento `click` de `homeButton`. Em outras palavras, com ela, sempre que o botão chamado `homeButton` é clicado, a função `gotoAuthorPage()` é chamada.

Teste do aplicativo

Neste momento, o aplicativo deve estar funcionando completamente. Vamos testá-lo para ver se isso é verdade.

Para testar o aplicativo:

1 Do menu principal, escolha Controlar > Testar filme. O Flash cria o arquivo SWF e o abre em uma janela do Flash Player.

2 Tente os dois botões para verificar se eles agem conforme o esperado.

3 Se os botões não funcionarem, veja algumas coisas que você pode verificar:

- Os dois botões têm nomes de ocorrência distintos?
- As chamadas de método `addEventListener()` usam os mesmos nomes que os nomes de ocorrência dos botões?
- Os nomes de evento corretos foram usados nas chamadas de método `addEventListener()`?
- Foi especificado o parâmetro correto para cada uma das funções? (As duas devem ter um único parâmetro com o tipo de dados MouseEvent.)

Tudo isso, além de outros erros possíveis, gera uma mensagem de erro quando você escolhe o comando Testar filme ou clica no botão. Verifique se há erros de compilação no painel Erros do compilador (aqueles que acontecem quando você escolhe Testar filme pela primeira vez) e verifique se há erros de tempo de execução no painel Saída (erros que acontecem quando o SWF é reproduzido, tal como quando você clica em um botão).

Criação de aplicativos com o ActionScript

O processo de escrever no ActionScript para criar um aplicativo vai além de apenas conhecer a sintaxe e os nomes das classes que serão usadas. Embora a maioria das informações neste manual seja destinada a esses dois tópicos (sintaxe e uso de classes do ActionScript), outras informações também são úteis, tal como quais programas podem ser usados para escrever código do ActionScript, como o ele pode ser organizado e incluído em um aplicativo e quais etapas devem ser executadas após o desenvolvimento do aplicativo do ActionScript.

Opções para organizar seu código

Você pode usar o código do ActionScript 3.0 para acionar tudo, desde simples animações gráficas até sistemas complexos de processamento de transações de cliente-servidor. Dependendo do tipo de aplicativo que está sendo criado, você pode preferir usar uma ou mais dessas formas de incluir o ActionScript no seu projeto.

Armazenamento de código em quadros em uma linha de tempo do Flash

No ambiente de autoria do Flash, você pode adicionar código do ActionScript a qualquer quadro na linha de tempo. Esse código será executado enquanto o filme estiver sendo reproduzido, quando o indicador de reprodução entrar nesse quadro.

A colocação de código do ActionScript em quadros fornece um meio simples de adicionar comportamentos a aplicativos criados na ferramenta de autoria do Flash. Você pode adicionar código a qualquer quadro na linha de tempo principal ou na linha de tempo de qualquer símbolo do MovieClip. Entretanto, essa flexibilidade tem um preço. Durante a criação de aplicativos grandes, é fácil perder o controle de quais quadros contêm quais scripts. Com o tempo, pode ficar mais difícil manter o aplicativo.

Muitos desenvolvedores simplificam a organização do código do ActionScript na ferramenta de autoria do Flash, colocando o código somente no primeiro quadro de uma linha de tempo ou em uma camada específica no documento Flash. Isso facilita a localização e manutenção do código nos arquivos FLA do Flash. Entretanto, para usar o mesmo código em outro projeto Flash, é necessário copiar e colar o código no novo arquivo.

Para poder usar o código do ActionScript em outros projetos Flash no futuro, você deve armazená-lo em arquivos externos do ActionScript (arquivos de texto com a extensão .as).

Armazenamento de código em arquivos do ActionScript

Se o seu projeto envolve uma quantidade significativa de código do ActionScript, a melhor maneira de organizar seu código é em arquivos de origem do ActionScript separados (arquivos de texto com a extensão .as). Um arquivo do ActionScript pode ser estruturado de duas formas, dependendo da maneira como você pretende usá-lo no aplicativo.

- Código do ActionScript não estruturado: linhas de código do ActionScript, incluindo instruções ou definições de funções, escritas como se fossem inseridas diretamente em um script de linha de tempo, arquivo MXML etc.

  O ActionScript escrito dessa forma pode ser acessado usando a instrução `include` no ActionScript ou a tag `<mx:Script>` no MXML do Adobe Flex. A instrução `include` do ActionScript faz com que o conteúdo de um arquivo externo do ActionScript seja inserido em um local específico e dentro de um determinado escopo em um script, como se fosse inserido diretamente. Na linguagem MXML do Flex, a tag `<mx:Script>` permite especificar um atributo de origem que identifica um arquivo externo do ActionScript a ser carregado nesse momento no aplicativo. Por exemplo, a seguinte tag carregará um arquivo externo do ActionScript chamado Box.as:

  ```
  <mx:Script source="Box.as" />
  ```

- Definição da classe do ActionScript: uma definição de uma classe do ActionScript, incluindo suas definições de método e propriedade.

  Durante a definição de uma classe, é possível acessar o código do ActionScript na classe, criando uma ocorrência da classe e usando suas propriedades, seus métodos e eventos, assim como faria com qualquer classe embutida do ActionScript. Isso requer duas partes:

  - Use a instrução `import` para especificar o nome completo da classe, para que o compilador do ActionScript saiba onde encontrá-la. Por exemplo, se você deseja usar a classe MovieClip no ActionScript, primeiro é necessário importar essa classe usando seu nome completo, incluindo o pacote e a classe:

```
import flash.display.MovieClip;
```

Se preferir, você pode importar o pacote que contém a classe MovieClip, que é equivalente a escrever instruções `import` separadas para cada classe no pacote:

```
import flash.display.*;
```

As únicas exceções à regra de que uma classe deve ser importada se for referida no código são as classes de nível superior, que não são definidas em um pacote.

Nota: *No Flash, para scripts anexados aos quadros na Linha de tempo, as classes embutidas (nos pacotes flash.*) são importadas automaticamente. Entretanto, sempre que você escrever suas próprias classes, se estiver trabalhando com componentes de autoria Flash (os pacotes fl.*) ou se estiver trabalhando no Flex, precisará importar explicitamente qualquer classe para escrever um código que crie ocorrências dessa classe.*

- Escreva um código que se refira especificamente ao nome da classe (normalmente declarando uma variável com essa classe como seu tipo de dados e criando uma ocorrência da classe a ser armazenada na variável). Fazendo referência a outro nome de classe no código do ActionScript, você instrui o compilador a carregar a definição dessa classe. Por exemplo, dada uma classe externa chamada Box, esta instrução faz com que uma nova ocorrência da classe Box seja criada:

  ```
  var smallBox:Box = new Box(10,20);
  ```

  Quando encontra a referência à classe Box pela primeira vez, o compilador pesquisa o código-fonte carregado para localizar a definição dessa classe.

Escolha da ferramenta correta

Dependendo da necessidade do seu projeto e dos recursos disponíveis, é possível usar uma das diversas ferramentas (ou várias ferramentas combinadas) para escrever e editar o código do ActionScript.

Ferramenta de autoria do Flash

Além de seus recursos gráficos e de criação de animações, o Adobe Flash CS4 Professional inclui ferramentas para trabalhar com códigos do ActionScript, anexados a elementos em um arquivo FLA ou em arquivos externos, exclusivos do ActionScript. A ferramenta de autoria do Flash é ideal para projetos que envolvem grande quantidade de animação ou vídeo, ou quando você deseja criar a maior parte dos ativos gráficos sozinho, especialmente projetos com uma interação mínima do usuário ou funcionalidade que exija o ActionScript. Outro motivo para escolher essa ferramenta ao desenvolver projetos do ActionScript é caso prefira criar ativos visuais e escrever código no mesmo aplicativo. Você também pode usar a autoria do Flash se quiser usar componentes da interface do usuário pré-criados, mas com um SWF menor ou uma atribuição de capa visual mais fácil são prioridades essenciais para seu projeto.

O Adobe Flash CS4 Professional inclui duas ferramentas para escrever códigos do ActionScript:

- Painel Ações: disponível para trabalhar em um arquivo FLA, este painel permite escrever código do ActionScript anexado a quadros em uma linha de tempo.
- Janela Script: a janela Script é um editor de texto dedicado para trabalhar com arquivos de código do ActionScript (.as).

Flex Builder

O Adobe Flex Builder é a principal ferramenta para criar projetos com a estrutura do Flex. Além do layout visual e das ferramentas de edição de MXML, o Flex Builder também inclui um editor do ActionScript, por isso ele pode ser usado para criar projetos do Flex ou somente ActionScript. Os aplicativos do Flex possuem várias vantagens, incluindo um rico conjunto de controles pré-criados da interface do usuário, controles flexíveis de layout dinâmico e mecanismos embutidos para trabalhar com fontes de dados externas e vincular dados externos a elementos da interface do usuário. Entretanto, por causa do código adicional necessário para fornecer esses recursos, os aplicativos do Flex podem ter um arquivo SWF maior e não podem ser completamente encapados com tanta facilidade quanto seus equivalentes do Flash.

Use o Flex Builder se estiver criando aplicativos da Internet avançados, orientados a dados e cheios de recursos com o Flex e quiser editar código do ActionScript, código MXML e criar o layout do aplicativo visualmente, tudo com uma única ferramenta.

Editor do ActionScript de terceiros

Como os arquivos do ActionScript (.as) são armazenados como arquivos de texto simples, qualquer programa capaz de editar arquivos de texto sem formatação pode ser usado para escrever arquivos do ActionScript. Além dos produtos do ActionScript da Adobe, foram criados diversos programas de edição de texto de terceiros com recursos específicos do ActionScript. Você pode escrever um arquivo MXML ou classes do ActionScript usando qualquer programa de editor de texto. É possível, então, criar um aplicativo SWF (um aplicativo do Flex ou somente ActionScript) a partir desses arquivos usando o SDK do Flex, que inclui as classes de estrutura do Flex, além do compilador do Flex. Opcionalmente, muitos desenvolvedores usam um editor do ActionScript de terceiros para escrever classes do ActionScript, junto com a ferramenta de autoria do Flash para criar conteúdo gráfico.

Você pode optar por usar um editor do ActionScript de terceiros se:

- Preferir escrever código do ActionScript em um programa separado junto com a criação de elementos visuais no Flash.
- Usar um aplicativo para programação que não seja do ActionScript (como criação de páginas HTML ou de aplicativos em outra linguagem de programação) e quiser usar o mesmo aplicativo para o código do ActionScript também.
- Quiser criar projetos do Flex ou somente ActionScript usando o SDK do Flex sem o Flash ou o Flex Builder.

Alguns dos notáveis editores de código que fornecem suporte específico ao ActionScript incluem:

- Adobe Dreamweaver® CS4
- ASDT
- FDT
- FlashDevelop
- PrimalScript
- SE|PY

O processo de desenvolvimento do ActionScript

Quer você tenha um projeto do ActionScript grande ou pequeno, o uso de um processo para criar e desenvolver seu aplicativo o ajudará a trabalhar com mais eficiência. As etapas a seguir descrevem um processo de desenvolvimento básico para criar um aplicativo que usa o ActionScript 3.0:

1 Crie seu aplicativo.

Você deve descrever seu aplicativo de alguma forma antes de começar a criá-lo.

2 Componha o código do ActionScript 3.0.

   Você pode criar código do ActionScript usando o Flash, Flex Builder, Dreamweaver ou um editor de texto.

3 Crie um arquivo de aplicativo do Flash ou Flex para executar o código.

   Na ferramenta de autoria do Flash, isso envolve a criação de um novo arquivo FLA, a definição de configurações de publicação, adição de componentes da interface do usuário ao aplicativo e a referência ao código do ActionScript. No ambiente de desenvolvimento do Flex, a criação de um novo aplicativo envolve a definição do aplicativo e a adição dos componentes da interface do usuário usando MXML e a referência ao código do ActionScript.

4 Publique e teste o aplicativo do ActionScript.

   Isso significa executar o aplicativo de dentro do ambiente de desenvolvimento do Flex ou de autoria do Flash e verificar se ele faz tudo conforme o esperado.

Observe que não é preciso seguir essas etapas na ordem nem concluir uma etapa completamente antes de começar outra. Por exemplo, você pode criar uma tela do aplicativo (etapa 1) e imagens gráficas, botões etc. (etapa 3), antes de escrever código do ActionScript (etapa 2) e testar (etapa 4). Ou você pode criar parte disso e depois adicionar um botão ou um elemento da interface por vez, escrevendo o ActionScript para cada um e testando-o durante a criação. Embora seja útil lembrar desses quatro estágios do processo de desenvolvimento, no mundo real, costuma ser mais eficiente executar os estágios avançando e retrocedendo, conforme apropriado.

Criação de suas próprias classes

O processo de criar as classes que serão usadas nos projetos pode parecer assustador. Entretanto, a parte mais difícil da criação de uma classe é a tarefa de criá-la, identificando os métodos, as propriedades e os eventos que ela incluirá.

Estratégias para criar uma classe

O tópico de criação orientada a objetos é complexo; existem cargos totalmente dedicados ao estudo acadêmico e à prática profissional dessa disciplina. No entanto, apresentamos algumas sugestões de abordagens que podem ajudá-lo a começar.

1 Pense na função que as ocorrências dessa classe exercerão no aplicativo. Em geral, os objetos cumprem uma destas três funções:

   - Objeto de valor: esses objetos servem basicamente como contêineres de dados, isto é, eles costumam ter diversas propriedades e menos métodos (ou, às vezes, nenhum método). Em geral, eles são representações de código de itens definidos claramente, como uma classe Song (representando uma única música do mundo real) ou classe Playlist (representando um grupo conceitual de músicas) em um aplicativo de player de música.
   - Objeto de exibição: são os objetos que realmente aparecem na tela. Exemplos incluem elementos da interface do usuário como uma lista suspensa ou exibição de status, elementos gráficos como criaturas em um videogame etc.
   - Estrutura do aplicativo: esses objetos exercem uma ampla gama de funções de suporte na lógica ou no processamento executados pelos aplicativos. Exemplos incluem um objeto que executa determinados cálculos em uma simulação biológica, um que é responsável por sincronizar valores entre um controle de mostrador e uma exibição de volume em um aplicativo de player de música, um que gerencia as regras em um videogame ou um que carrega uma imagem salva em um aplicativo de desenho.

2 Escolha a funcionalidade específica de que a classe precisará. Os diferentes tipos de funcionalidade, em geral, se tornam métodos da classe.

3 Se a classe for servir como um objeto de valor, decida quais dados as ocorrências incluirão. Esses itens são bons candidatos para propriedades.

4 Como a classe está sendo criada especificamente para seu projeto, o mais importante é fornecer a funcionalidade de que o aplicativo precisa. Talvez ajude responder a estas questões:

- Que informações o aplicativo irá armazenar, controlar e manipular? Decidir isso ajuda a identificar os objetos de valor e as propriedades que você deseja.
- Quais conjuntos de ações deverão ser executados, por exemplo, quando o aplicativo for carregado pela primeira vez, um botão for clicado, um filme parar de ser reproduzido etc.? Esses são bons candidatos para métodos (ou propriedades, se as "ações" envolverem apenas a alteração de valores individuais).
- Quais informações a classe deve saber para executar cada ação específica? Essas informações se tornam os parâmetros do método.
- Conforme o aplicativo executar seu trabalho, o que mudará na sua classe que outras partes do aplicativo deverá saber? Esses itens são bons candidatos para eventos.

5 Se houver um objeto existente semelhante àquele de que você precisa, mas que não tenha alguma funcionalidade adicional que você deseja adicionar, considere a criação de uma subclasse (uma classe que aproveite a funcionalidade de uma classe existente, em vez de definir todas as suas próprias funcionalidades). Por exemplo, se você quiser criar uma classe que seja um objeto visual na tela, pode usar o comportamento de um dos objetos de exibição existentes (por exemplo, Sprite ou MovieClip) como base para sua classe. Nesse caso, MovieClip (ou Sprite) seria a *classe base*, e sua classe estenderia essa classe. Para obter mais informações sobre a criação de uma subclasse, consulte "Herança" na página 110.

Escrita do código para uma classe

Depois de fazer um plano de criação para sua classe ou, pelo menos, ter uma idéia das informações sobre as quais ela deverá ter controle e quais ações deverão ser executadas, a sintaxe real da escrita de uma classe é bem direta.

Veja as etapas mínimas para criar sua própria classe do ActionScript:

1 Abra um novo documento de texto, em um programa específico do ActionScript como o Flex Builder ou o Flash, em uma ferramenta de programação geral como o Dreamweaver ou em qualquer programa que permita trabalhar com documento de texto sem formatação.

2 Insira uma ocorrência de `class` para definir o nome da classe. Para isso, digite as palavras `public class`, o nome da classe e as chaves que envolverão o conteúdo da classe (as definições de método e propriedade). Por exemplo:

```
public class MyClass
{
}
```

A palavra `public` indica que a classe pode ser acessada de qualquer outro código. Para obter alternativas, consulte "Atributos de espaço para nomes de controle de acesso" na página 96.

3 Digite uma instrução `package` para indicar o nome do pacote no qual sua classe se encontrará. A sintaxe é a palavra `package`, seguida do nome completo do pacote e das chaves (que envolverão o bloco da instrução `class`). Por exemplo, o código na etapa anterior seria alterado da seguinte forma:

```
package mypackage
{
    public class MyClass
    {
    }
}
```

4 Defina cada propriedade na classe usando a instrução `var` dentro do corpo da classe; a sintaxe é a mesma usada para declarar qualquer variável (com a adição do modificador `public`). Por exemplo, a adição destas linhas entre as chaves de definição da classe criará propriedades chamadas `textVariable`, `numericVariable` e `dateVariable`:

```
public var textVariable:String = "some default value";
public var numericVariable:Number = 17;
public var dateVariable:Date;
```

5 Defina cada método na classe usando a mesma sintaxe usada para definir uma função. Por exemplo:

- Para criar um método `myMethod()`, digite:

```
public function myMethod(param1:String, param2:Number):void
{
    // do something with parameters
}
```

- Para criar um construtor (o método especial que é chamado como parte do processo de criar uma ocorrência de uma classe), crie um método cujo nome corresponda exatamente ao nome da classe:

```
public function MyClass()
{
    // do stuff to set initial values for properties
    // and otherwise set up the object
    textVariable = "Hello there!";
    dateVariable = new Date(2001, 5, 11);
}
```

  Se você não incluir um método construtor na classe, o compilador criará automaticamente um construtor vazio (sem nenhum parâmetro e nenhuma instrução) na sua classe.

Há mais alguns elementos de classe que você pode definir, que são mais complexos.

- *Assessores* são um cruzamento especial entre um método e uma propriedade. Durante a escrita do código para definir a classe, você escreve o assessor como um método para poder executar várias ações (em vez de apenas ler ou atribuir um valor, que é tudo o que você pode fazer ao definir uma propriedade). Entretanto, na criação de uma ocorrência da classe, você trata o assessor como uma propriedade, usando apenas o nome para ler ou atribuir o valor. Para obter mais informações, consulte "Métodos de acessor get e set" na página 103.
- Os eventos no ActionScript não são definidos usando uma sintaxe específica. Em vez disso, você define eventos na classe usando a funcionalidade da classe EventDispatcher para manter controle de ouvintes de evento e notificá-los dos eventos. Para obter mais informações sobre a criação de eventos nas suas próprias classes, consulte"Manipulação de eventos" na página 251.

Exemplo: Criação de um aplicativo básico

Você pode criar arquivos de código do ActionScript externo com uma extensão .as usando o Flash, Flex Builder, Dreamweaver ou qualquer editor de texto.

O ActionScript 3.0 pode ser usado em diversos ambientes de desenvolvimento de aplicativo, incluindo as ferramentas de autoria do Flash e o Flex Builder.

Esta seção apresenta as etapas de criação e de aprimoramento de um aplicativo ActionScript 3.0 simples usando a ferramenta de autoria do Flash ou o Flex Builder. O aplicativo que você criar apresentará um padrão simples para usar arquivos externos de classe do ActionScript 3.0 nos aplicativos Flash e Flex. Esse padrão se aplicará a todos os outros aplicativos de exemplo deste manual.

Criação do seu aplicativo do ActionScript

Você deve ter uma idéia do aplicativo que deseja ter antes de começar a criá-lo.

A representação do seu design pode ser tão simples quanto o nome do aplicativo e uma breve instrução de sua finalidade ou tão complicado quanto um conjunto de documentos de requisitos contendo vários diagramas UML (Unified Modeling Language). Este manual não discute a disciplina do design de software em detalhes, mas é importante ter em mente que ele é uma etapa essencial no desenvolvimento de aplicativos do ActionScript.

Nosso primeiro exemplo de um aplicativo do ActionScript será um aplicativo "Hello World" padrão, pois seu design é muito simples:

- O aplicativo será chamado de HelloWorld.
- Ele exibirá um único campo de texto contendo as palavras "Hello World!".
- Para ser reutilizado facilmente, ele usará uma única classe orientada a objetos, chamada Greeter, que pode ser usada de dentro de um documento Flash ou um aplicativo do Flex.
- Depois de criar uma versão básica do aplicativo, você adicionará uma nova funcionalidade que exija do usuário a inserção de um nome de usuário e faça o aplicativo verificar esse nome em relação ao uma lista de usuários conhecidos.

Com essa definição concisa estabelecida, você pode começar a criar o aplicativo em si.

Criação do projeto HelloWorld e da classe Greeter

A instrução do design para o aplicativo Hello World dizia que seu código deve ser fácil de reutilizar. Com esse objetivo em mente, o aplicativo usa uma única classe orientada a objetos, chamada Greeter, que é usada de dentro de um aplicativo criado no Flex Builder ou na ferramenta de autoria do Flash.

Para criar a classe Greeter na ferramenta de autoria do Flash:

1 Na ferramenta autoria do Flash, selecione Arquivo > Novo.

2 Na caixa de diálogo Novo documento, selecione o arquivo do ActionScript e clique em OK.

Uma nova janela de edição do ActionScript será exibida.

3 Selecione Arquivo > Salvar. Selecione uma pasta para conter o aplicativo, chame o arquivo do ActionScript de **Greeter.as** e clique em OK.

Continue com "Adição de código à classe Greeter" na página 31.

Adição de código à classe Greeter

A classe Greeter define um objeto, `Greeter`, que poderá ser usado no aplicativo HelloWorld.

Para adicionar código à classe Greeter:

1 Digite o seguinte código no novo arquivo:

```
package
{
    public class Greeter
    {
        public function sayHello():String
        {
            var greeting:String;
            greeting = "Hello World!";
            return greeting;
        }
    }
}
```

A classe Greeter inclui um único método `sayHello()` , que retorna uma seqüência de caracteres que diz “Hello World!”.

2 Clique em Arquivo > Salvar para salvar esse arquivo do ActionScript.

A classe Greeter agora está pronta para ser usada em um aplicativo.

Criação de um aplicativo que usa o código do ActionScript

A classe Greeter que você criou define um conjunto independente de funções de software, mas não representa um aplicativo completo. Para usar a classe, é necessário criar um documento do Flash ou aplicativo do Flex.

O aplicativo HelloWorld cria uma nova ocorrência da classe Greeter. Veja como anexar a classe Greeter ao seu aplicativo.

Para criar um aplicativo do ActionScript usando a ferramenta de autoria do Flash:

1 Selecione Arquivo > Novo.

2 Na caixa de diálogo Novo documento, selecione Documento Flash e clique em OK.

Uma nova janela do Flash será exibida.

3 Selecione Arquivo > Salvar. Selecione uma pasta que contenha o arquivo de classe Greeter.as, chame o documento Flash de **HelloWorld.fla** e clique em OK.

4 Na paleta Ferramentas do Flash, selecione a ferramenta Texto e arraste até o Palco para definir um novo campo de texto, com aproximadamente 300 pixels de largura e 100 de altura.

5 No painel Propriedades, com o campo de texto ainda selecionado no Palco, defina o tipo de texto como "Texto dinâmico" e digite **mainText** como o nome de ocorrência do campo de texto.

6 Clique no primeiro quadro da linha de tempo principal.

7 No painel Ações, digite o seguinte script:

```
var myGreeter:Greeter = new Greeter();
mainText.text = myGreeter.sayHello();
```

8 Salve o arquivo.

Continue com “Publicação e teste do aplicativo do ActionScript” na página 33.

Publicação e teste do aplicativo do ActionScript

O desenvolvimento de software é um processo interativo. Você escreve um código, tenta compilá-lo e o edita até obter uma compilação limpa. Depois, você executa o aplicativo compilado, testa-o para ver se ele cumpre o design pretendido e, caso negativo, edita o código novamente até que o faça. Os ambiente de desenvolvimento do Flash e do Flex Builder oferecem vários meios de publicar, testar e depurar aplicativos.

Veja as etapas básicas para testar o aplicativo HelloWorld em cada ambiente.

Para publicar e testar um aplicativo do ActionScript usando a ferramenta de autoria do Flash:

1. Publique seu aplicativo e observe se há erros de compilação. Na ferramenta de autoria do Flash, selecione Controlar > Testar filme para compilar o código do ActionScript e executar o aplicativo HelloWorld.
2. Se forem exibidos erros ou avisos na janela Saída durante o teste do aplicativo, corrija as causas dos erros nos arquivos HelloWorld.fla ou HelloWorld.as e tente testar o aplicativo novamente.
3. Se não houver nenhum erro de compilação, você verá uma janela do Flash Player mostrando o aplicativo Hello World.

Você acabou de criar um aplicativo orientado a objetos simples, mas completo, que usa o ActionScript 3.0. Continue com "Aprimoramento do aplicativo HelloWorld" na página 33.

Aprimoramento do aplicativo HelloWorld

Para tornar o aplicativo um pouco mais interessante, agora você o fará solicitar e validar um nome de usuário em relação a uma lista de nomes predefinida.

Primeiro, você atualizará a classe Greeter para adicionar nova funcionalidade. Depois, você atualizará o aplicativo para usar a nova funcionalidade.

Para atualizar o arquivo Greeter.as:

1. Abra o arquivo Greeter.as.
2. Altere o conteúdo do arquivo com o seguinte (as linhas novas e alteradas são mostradas em negrito):

```
package
{
    public class Greeter
    {
        /**
         * Defines the names that should receive a proper greeting.
         */
        public static var validNames:Array = ["Sammy", "Frank", "Dean"];

        /**
         * Builds a greeting string using the given name.
         */
        public function sayHello(userName:String = ""):String
        {
            var greeting:String;
            if (userName == "")
            {
                greeting = "Hello. Please type your user name, and then press
                            the Enter key.";
            }
            else if (validName(userName))
            {
```

```
                greeting = "Hello, " + userName + ".";
            }
            else
            {
                greeting = "Sorry " + userName + ", you are not on the list.";
            }
            return greeting;
        }

        /**
         * Checks whether a name is in the validNames list.
         */
        public static function validName(inputName:String = ""):Boolean
        {
            if (validNames.indexOf(inputName) > -1)
            {
                return true;
            }
            else
            {
                return false;
            }
        }
    }
}
```

A classe Greeter agora tem vários novos recursos:

- A matriz `validNames` lista os nomes de usuário válidos. A matriz é inicializada com uma lista de três nomes quando a classe Greeter é carregada.
- O método `sayHello()` agora aceita um nome de usuário e altera a saudação com base em algumas condições. Se `userName` for uma seqüência de caracteres vazia (`""`), a propriedade `greeting` será definida para solicitar um nome ao usuário. Se o nome do usuário for válido, a saudação se tornará `"Hello, `*userName*`"`. Finalmente, se as duas condições não forem atendidas, a variável `greeting` será definida como `"Sorry `*userName*`, you are not on the list"`.
- O método `validName()` retornará `true` se `inputName` for encontrado na matriz `validNames` e `false` se não for encontrado. A instrução `validNames.indexOf(inputName)` verifica cada seqüência de caracteres na matriz `validNames` em relação à seqüência de caracteres `inputName`. O método `Array.indexOf()` retornará a posição de índice da primeira ocorrência de um objeto em uma matriz ou o valor -1 se o objeto não for encontrado nela.

Em seguida, você editará o arquivo Flash ou Flex que faz referência a essa classe do ActionScript.

Para modificar o aplicativo usando a ferramenta de autoria do Flash:

1. Abra o arquivo HelloWorld.fla.
2. Modifique o script no Quadro 1 para que uma seqüência de caracteres (`""`) seja transmitida ao método `sayHello()` da classe Greeter:

```
var myGreeter:Greeter = new Greeter();
mainText.text = myGreeter.sayHello("");
```

3. Selecione a ferramenta Texto na paleta Ferramentas e crie dois novos campos de texto no Palco, lado a lado e diretamente sob o campo de texto `mainText` existente.
4. No primeiro campo de texto novo, digite o texto **User Name:** para servir de rótulo.

5 No outro campo de texto novo, e no Inspetor de propriedades, selecione InputText como o tipo de campo de texto. Selecione Linha única como o Tipo de linha. Digite **textIn** como o nome de ocorrência.

6 Clique no primeiro quadro da linha de tempo principal.

7 No painel Ações, adicione as seguintes linhas no final do script existente:

```
mainText.border = true;
textIn.border = true;

textIn.addEventListener(KeyboardEvent.KEY_DOWN, keyPressed);

function keyPressed(event:KeyboardEvent):void
{
    if (event.keyCode == Keyboard.ENTER)
    {
        mainText.text = myGreeter.sayHello(textIn.text);
    }
}
```

O novo código adiciona a seguinte funcionalidade:

- As primeiras duas linhas simplesmente definem bordas para os dois campos de texto.
- Um campo de texto de entrada, como o campo `textIn`, possui um conjunto de eventos que ele pode despachar. O método `addEventListener()` permite definir uma função que é executada quando um tipo de evento ocorre. Neste caso, o evento é o pressionamento de uma tecla no teclado.
- A função personalizada `keyPressed()` verifica se a tecla que foi pressionada é a tecla Enter. Caso afirmativo, ela chama o método `sayHello()` do objeto `myGreeter` , transmitindo o texto do campo de texto `textIn` como um parâmetro. Esse método retorna uma seqüência de caracteres de saudação com base no valor transmitido. A seqüência de caracteres retornada é atribuída à propriedade `text` do campo de texto `mainText`.

O script completo para o Quadro 1 é o seguinte:

```
var myGreeter:Greeter = new Greeter();
mainText.text = myGreeter.sayHello("");

mainText.border = true;
textIn.border = true;

textIn.addEventListener(KeyboardEvent.KEY_DOWN, keyPressed);

function keyPressed(event:KeyboardEvent):void
{
    if (event.keyCode == Keyboard.ENTER)
    {
        mainText.text = myGreeter.sayHello(textIn.text);
    }
}
```

8 Salve o arquivo.

9 Selecione Controlar > Testar filme para executar o aplicativo.

Durante a execução do aplicativo, você será solicitado a inserir um nome de usuário. Se for válido (Sammy, Frank ou Dean), o aplicativo exibirá a mensagem de confirmação "hello".

Execução de exemplos subseqüentes

Agora que desenvolveu e executou o aplicativo "Hello World" do ActionScript 3.0, você tem o conhecimento básico necessário para executar os outros exemplos de código apresentados neste manual.

Teste de listagens de código de exemplo dos capítulos

Durante a leitura deste manual, talvez você queira testar as listagens de código de exemplo usadas para ilustrar os diversos tópicos. Esse teste pode envolver a exibição do valor de variáveis em determinados pontos no programa ou a exibição ou interação com o conteúdo na tela. Para testar o conteúdo ou a interação visual, os elementos necessários serão descritos antes ou dentro da listagem de código (bastará apenas criar um documento com os elementos conforme descrito para testar o código). Caso você queira exibir o valor de uma variável em um determinado ponto no programa, há algumas formas de fazer isso. Uma delas é usar um depurador, como aqueles criados no Flex Builder e no Flash. Para um teste simples, porém, pode ser mais fácil apenas imprimir os valores de variável em um local que permita exibi-los.

As etapas a seguir o ajudarão a criar um documento Flash que você pode usar para testar uma listagem de código e exibir valores de variável:

Para criar um documento Flash para testar os exemplos dos capítulos:

1. Crie um novo documento Flash e salve-o no seu disco rígido.
2. Para exibir os valores de teste em um campo de texto no Palco, ative a ferramenta Texto e crie um novo campo Texto dinâmico no Palco. Um campo de texto largo e alto, com o Tipo de linha definido como Multilinha e a borda ativada será muito útil. No Inspetor de propriedades, atribua um nome de ocorrência ao campo de texto (por exemplo, "outputText"). Para escrever os valores no campo de texto, você adicionará um código que chama o método `appendText()` ao código de exemplo (descrito abaixo).
3. Se preferir, você pode adicionar uma chamada de função `trace()` à listagem de código (conforme descrito abaixo) para exibir os resultados do exemplo.
4. Para testar um determinado exemplo, copie a listagem de código para o painel Ações; se necessário, adicione uma chamada de função ou um valor `trace()` ao campo de texto usando seu método `appendText()`.
5. Do menu principal, escolha Controlar > Testar filme para criar um arquivo SWF e exibir os resultados.

Como esta abordagem é para exibir os valores de variáveis, há duas formas de fazer isso facilmente ao testar os exemplos: escrever os valores em uma ocorrência de campo de texto no Palco ou usar a função `trace()` para imprimir os valores no painel Saída.

- A função `trace()`: a função `trace()` do ActionScript escreve os valores de qualquer parâmetro transmitido (variáveis ou expressões literais) no painel Saída. Muitas listagens de exemplo deste manual já incluem uma chamada de função `trace()`, por isso, para essas listagens, basta copiar o código para o documento e testar o projeto. Se quiser usar `trace()` para testar o valor de uma variável em uma listagem de código que não a inclua, apenas adicione uma chamada `trace()` à listagem de código, transmitindo a variável como um parâmetro. Por exemplo, se você encontrar uma listagem de código como esta no capítulo,

  ```
  var albumName:String = "Three for the money";
  ```

  deverá copiar o código no painel Ações e adicionar uma chamada à função `trace()` como esta para testar o resultado da listagem de código:

  ```
  var albumName:String = "Three for the money";
  trace("albumName =", albumName);
  ```

  Durante a execução do programa, esta linha será impressa:

```
albumName = Three for the money
```

Cada chamada de função `trace()` pode usar vários parâmetros, que são reunidos em uma string como uma única linha impressa. Uma quebra de linha é adicionada ao final de cada chamada de função `trace()`, de forma que chamadas `trace()` separadas são impressas em linhas separadas.

- Um campo de texto no Palco: se preferir não usar a função `trace()`, você pode adicionar um campo Texto dinâmico ao Palco usando a Ferramenta texto e escrever os valores nesse campo de texto para exibir os resultados de uma listagem de código. O método `appendText()` da classe TextField pode ser usado para adicionar um valor String ao final do conteúdo do campo de texto. Para acessar o campo de texto usando o ActionScript, atribua-lhe um nome de ocorrência no Inspetor de propriedades. Por exemplo, se o seu campo de texto tiver o nome de ocorrência `outputText`, o seguinte código poderá ser usado para verificar o valor da variável `albumName`:

```
var albumName:String = "Three for the money";
outputText.appendText("albumName = ");
outputText.appendText(albumName);
```

Esse código deve escrever o seguinte texto no campo de texto chamado `outputText`:

```
albumName = Three for the money
```

Como mostra o exemplo, o método `appendText()` adicionará o texto à mesma linha que o conteúdo anterior; portanto, diversos valores podem ser adicionados à mesma linha de texto usando várias chamadas `appendText()`. Para forçar o texto para a linha seguinte, você pode anexar um caractere de nova linha (`"\n"`):

```
outputText.appendText("\n"); // adds a line break to the text field
```

Diferentemente da função `trace()`, o método `appendText()` aceita apenas um valor como parâmetro. Esse valor deve ser uma seqüência de caracteres (uma ocorrência de String ou um literal de seqüência de caracteres). Para imprimir o valor de uma variável que não seja uma seqüência de caracteres, primeiro é necessário converter o valor em uma String. A maneira mais fácil de fazer isso é chamar o método `toString()` do objeto:

```
var albumYear:int = 1999;
outputText.appendText("albumYear = ");
outputText.appendText(albumYear.toString());
```

Teste dos exemplos do fim do capítulo

Como este, a maioria dos capítulos deste manual inclui um exemplo de fim de capítulo significativo relacionado aos diversos conceitos discutidos. Entretanto, diferentemente do exemplo Hello World deste capítulo, esses exemplos não serão apresentados em um formato de tutorial passo a passo. O código relevante do ActionScript 3.0 em cada exemplo será destacado e discutido, mas as instruções sobre a execução dos exemplos em ambientes de desenvolvimento específicos não serão fornecidas. Os arquivos de exemplo distribuídos com este manual, contudo, incluirão todos os arquivos necessários para compilar e executar os exemplos facilmente no ambiente de desenvolvimento de sua escolha.

Capítulo 4: Linguagem e sintaxe do ActionScript

O ActionScript 3.0 consiste na linguagem central do ActionScript e na API (Application Programming Interface) do Adobe Flash Player. A linguagem principal é a parte do ActionScript que define a sintaxe da linguagem, assim como os tipos de dados de nível superior. O ActionScript 3.0 oferece acesso programático ao Flash Player.

Este capítulo apresenta uma breve instrução à linguagem e sintaxe centrais do ActionScript. Depois de lê-lo, você terá uma noção básica de como trabalhar com tipos de dados e variáveis, como usar a sintaxe apropriada e como controlar o fluxo de dados no seu programa.

Visão geral da linguagem

Os objetos são a base da linguagem do ActionScript 3.0, seus os blocos de construção fundamentais. Cada variável declarada, cada função escrita e cada ocorrência de classe criada é um objeto. Pense em um programa do ActionScript 3.0 como um grupo de objetos que realizam tarefas, respondem a eventos e se comunicam.

Os programadores acostumados à OOP (Programação orientada a objetos) em Java ou C++ podem pensar nos objetos como módulos que contêm dois tipos de membros: dados armazenados em variáveis ou propriedades de membros e comportamento acessível por meio de métodos. O ActionScript 3.0 define objetos de modo similar, com pequenas particularidades. No ActionScript 3.0, os objetos são apenas conjuntos de propriedades. Essas propriedades são contêineres que podem manter não apenas dados, mas também funções ou outros objetos. Se uma função for anexada a um objeto dessa forma, ela será chamada de método.

Embora a definição do ActionScript 3.0 possa parecer um pouco estranha aos programadores com experiência em Java ou C++, na prática, a definição dos tipos de objeto com classes do ActionScript 3.0 é bastante semelhante à forma como as classes são definidas em Java ou C++. A distinção entre as duas definições de objeto é importante ao discutir o modelo de objeto do ActionScript e outros tópicos avançados, mas, na maioria das situações, o termo *propriedades* significa variáveis de membro de classe, e não métodos. A Referência dos componentes e da linguagem do ActionScript 3.0, por exemplo, usa o termo *propriedades* para se referir a variáveis ou propriedades getter-setter. Ela usa o termo *métodos* para se referir às funções que fazem parte de uma classe.

Uma diferença sutil entre as classes no ActionScript e as classes em Java ou C++ é que, no ActionScript, as classes não são apenas entidades abstratas. As classes do ActionScript são representadas por *objetos de classe* que armazenam propriedades e métodos da classe. Isso permite o uso de técnicas que podem parecer estranhas aos programadores de Java e C++, como incluir instruções ou código executável no nível superior de uma classe ou um pacote.

Outra diferença entre as classes do ActionScript e as de Java ou C++ é que toda classe do ActionScript tem o que chamamos de *objeto deprotótipo*. Nas versões anteriores do ActionScript, os objetos de protótipo, vinculados em *cadeias de protótipos*, serviam coletivamente como a base de toda a hierarquia de herança de classes. No ActionScript 3.0, contudo, os objetos de protótipo desempenham um papel secundário no sistema de herança. Apesar disso, eles poderão ser úteis como uma alternativa às propriedades e aos métodos estáticos se você quiser compartilhar uma propriedade e seu valor com todas as ocorrências de uma classe.

Anteriormente, os programadores de ActionScript avançados podiam manipular diretamente a cadeia de protótipos com elementos de linguagem embutidos especiais. Agora que essa linguagem fornece uma implementação mais madura de uma interface de programação baseada em classes, muitos desses elementos de linguagem especiais, como `__proto__` e `__resolve`, não fazem mais parte da linguagem. Além disso, as otimizações do mecanismo de herança interno que fornece melhorias de desempenho significativas no Flash Player e no Adobe AIR impedem o acesso direto ao mecanismo de herança.

Objetos e classes

No ActionScript 3.0, cada objeto é definido por uma classe. Uma classe pode ser entendida como um modelo ou uma cópia de um tipo de objeto. As definições de classe podem incluir variáveis e constantes, que mantêm valores de dados, e métodos, que são funções de encapsulamento de comportamento vinculadas à classe. Os valores armazenados em propriedades podem ser *valores primitivos* ou outros objetos. Os valores primitivos são números, seqüências de caracteres ou valores booleanos.

O ActionScript contém diversas classes embutidas que fazem parte da linguagem central. Algumas delas, como Number, Boolean e String, representam os valores primitivos disponíveis no ActionScript. Outras classes, como Array, Math e XML, definem objetos mais complexos.

Todas as classes, incorporadas ou definidas pelo usuário, derivam da classe Object. Para os programadores com experiência no ActionScript, é importante observar que o tipo de dados Object não é mais o tipo de dados padrão, muito embora todas as outras classes ainda derivem dessa. No ActionScript 2.0, as duas linhas de código a seguir eram equivalentes porque a falta de uma anotação de tipo significava que uma variável seria do tipo Object:

```
var someObj:Object;
var someObj;
```

O ActionScript 3.0, porém, apresenta o conceito de variáveis sem tipo, que podem ser designadas destas duas formas:

```
var someObj:*;
var someObj;
```

Uma variável sem tipo não é igual a uma variável do tipo Object. A principal diferença é que as variáveis sem tipo podem manter o valor especial `undefined`, enquanto que uma variável do tipo Object não pode.

Você pode definir suas próprias classes usando a palavra-chave `class`. As propriedades de classe podem ser declaradas de três formas: as constantes podem ser definidas com a palavra-chave `const`, as variáveis são definidas com a palavra-chave `var` e as propriedades getter e setter são definidas usando os atributos `get` e `set` em uma declaração de método. Os métodos podem ser declarados com a palavra-chave `function`.

Uma ocorrência de uma classe é criada usando o operador `new`. O exemplo a seguir cria uma ocorrência da classe Date chamada `myBirthday`.

```
var myBirthday:Date = new Date();
```

Pacotes e espaços para nomes

Os pacotes e espaços para nomes são conceitos relacionados. Os pacotes permitem compactar definições de classe juntas de uma forma que facilita o compartilhamento de dados e minimiza conflitos de nome. Os espaços para nomes permitem controlar a visibilidade de identificadores, como nomes de propriedades e métodos, e podem ser aplicados ao código quer ele resida dentro ou fora de um pacote. Os pacotes permitem organizar os arquivos de classe, e os espaços para nomes permitem gerenciar a visibilidade de propriedades e métodos individuais.

Pacotes

Os pacotes no ActionScript 3.0 são implementados com espaços para nomes, mas eles não são sinônimos. Ao declarar um pacote, você cria implicitamente um tipo especial de espaço para nomes que será conhecido em tempo de compilação. Os espaços para nomes, quando criados explicitamente, não são necessariamente conhecidos em tempo de compilação.

O seguinte exemplo usa a diretiva `package` para criar um pacote simples contendo uma classe:

```
package samples
{
    public class SampleCode
    {
        public var sampleGreeting:String;
        public function sampleFunction()
        {
            trace(sampleGreeting + " from sampleFunction()");
        }
    }
}
```

O nome da classe neste exemplo é SampleCode. Como a classe está dentro do pacote de amostras, o compilador automaticamente qualifica o nome da classe em tempo de compilação em seu nome totalmente qualificado: samples.SampleCode. O compilador também qualifica os nomes de quaisquer propriedades e métodos, para que `sampleGreeting` e `sampleFunction()` se tornem `samples.SampleCode.sampleGreeting` e `samples.SampleCode.sampleFunction()`, respectivamente.

Muitos desenvolvedores, especialmente aqueles com experiência em programação Java, podem optar por colocar apenas classes no nível superior de um pacote. O ActionScript 3.0, no entanto, oferece suporte não apenas a classes no nível superior de um pacote, mas também a variáveis, funções e até mesmo instruções. Uma utilização avançada desse recurso é definir um espaço para nomes no nível superior de um pacote de forma que fique disponível para todas as classes desse pacote. Observe, porém, que somente dois especificadores de acesso, `public` e `internal`, são permitidos no nível superior de um pacote. Diferentemente de Java, que permite declarar classes aninhadas como particulares, o ActionScript 3.0 não oferece suporte a classes aninhadas nem a particulares.

Entretanto, de muitas outras formas, os pacotes do ActionScript 3.0 são semelhantes aos pacotes na linguagem de programação Java. Como você pode ver no exemplo anterior, as referências aos pacotes totalmente qualificados são expressas usando o operador dot (`.`), da mesma forma que em Java. Você pode usar pacotes para organizar seu código em uma estrutura hierárquica intuitiva para ser usada por outros programadores. Isso facilita o compartilhamento de código, permitindo criar seu próprio pacote para compartilhar com outros e usar pacotes criados por outros em seu código.

O uso de pacotes também ajuda a garantir que os nomes de identificador usados sejam exclusivos e não entrem em conflito com outros. Na verdade, alguns acham que essa é a maior vantagem dos pacotes. Por exemplo, dois programadores que desejam compartilhar código entre si criam uma classe chamada SampleCode. Sem pacotes, ela cria um conflito de nome e a única solução seria renomear uma das classes. Com pacotes, porém, o conflito de nomes é facilmente evitado colocando uma classe, ou de preferência as duas, em pacotes com nomes exclusivos.

Também é possível incluir pontos incorporados no nome do pacote para criar pacotes aninhados. Isso permite criar uma organização hierárquica de pacotes. Um bom exemplo disso é o pacote flash.xml fornecido pelo ActionScript 3.0. O pacote flash.xml é aninhado dentro do pacote flash.

O pacote flash.xml contém o analisador XML herdado que era usado em versões anteriores do ActionScript. Um dos motivos para que agora ele resida no pacote flash.xml é que o nome da classe XML herdada entra em conflito com o nome da nova classe XML que implementa a funcionalidade XML da especificação ECMAScript (E4X) disponível no ActionScript 3.0.

Embora o ato de mover a classe XML herdada para um pacote seja uma boa medida inicial, a maioria dos usuários das classes XML herdadas importa o pacote flash.xml, o que irá gerar o conflito de nomes a menos que você se lembre de sempre usar o nome totalmente qualificado da classe XML herdada (flash.xml.XML). Para evitar essa situação, a classe XML herdada agora se chama XMLDocument, como mostra o seguinte exemplo:

```
package flash.xml
{
    class XMLDocument {}
    class XMLNode {}
    class XMLSocket {}
}
```

Grande parte do ActionScript 3.0 é organizado com base no pacote flash. Por exemplo, o pacote flash.display contém a API de lista de exibição, e o pacote flash.events contém o novo modelo de eventos.

Criação de pacotes

O ActionScript 3.0 fornece uma flexibilidade significativa na forma de organizar pacotes, classes e arquivos de origem. As versões anteriores do ActionScript permitiam somente uma classe por arquivo de origem e exigiam que o nome do arquivo de origem correspondesse ao nome da classe. O ActionScript 3.0 permite incluir diversas classes em um único arquivo de origem, mas somente uma classe em cada arquivo pode ser disponibilizada para um código externo ao arquivo. Em outras palavras, somente uma classe em cada arquivo pode ser declarada dentro de uma declaração de pacote. As classes adicionais devem ser declaradas fora da definição do pacote, o que torna as classes invisíveis ao código fora do arquivo de origem. O nome da classe declarada dentro da definição do pacote deve corresponder ao nome do arquivo de origem.

O ActionScript 3.0 também oferece mais flexibilidade na forma de declarar pacotes. Nas versões anteriores do ActionScript, os pacotes simplesmente representavam diretórios nos quais os arquivos de origem eram colocados e os pacotes não eram declarados com a instrução `package`, mas incluíam o nome do pacote como parte do nome da classe totalmente qualificada na sua declaração de classe. Embora ainda representem diretórios no ActionScript 3.0, os pacotes podem conter mais do que apenas classes. No ActionScript 3.0, a instrução `package` é usada para declarar um pacote, o que significa que você também pode declarar variáveis, funções e espaços para nomes no nível superior de um pacote. É possível até incluir instruções executáveis no nível superior de um pacote. Se você declarar variáveis, funções ou espaços para nomes no nível superior de um pacote, os únicos atributos disponíveis nesse nível serão `public` e `internal`, e somente uma declaração de nível de pacote por arquivo poderá usar o atributo `public`, quer a declaração seja uma classe, variável, função ou um espaço para nomes.

Os pacotes são úteis para organizar o código e evitar conflitos de nome. Não confunda o conceito de pacotes com o conceito não relacionado de herança de classe. Duas classes que residem no mesmo pacote têm um espaço para nomes em comum, mas não estão necessariamente relacionadas de outra forma. Da mesma forma, um pacote aninhado pode não ter nenhuma relação semântica com o pacote pai.

importação de pacotes

Para usar uma classe que está dentro de um pacote, você deve importar o pacote ou a classe específica. Isso difere do ActionScript 2.0, em que a importação de classes era opcional.

Por exemplo, considere o exemplo de classe SampleCode apresentado anteriormente neste capítulo. Se a classe residir em um pacote chamado sample, você deverá usar uma das seguintes instruções de importação usando a classe SampleCode:

```
import samples.*;
```

ou

```
import samples.SampleCode;
```

Em geral, as instruções `import` devem ser tão específicas quanto possível. Se você pretende usar apenas a classe SampleCode do pacote samples, deverá importar somente a classe SampleCode e não o pacote inteiro ao qual ela pertence. A importação de pacotes inteiros pode gerar conflitos de nome inesperados.

Você também deve colocar o código-fonte que define o pacote ou a classe no *caminho de classe*. O caminho de classe é uma lista definida pelo usuário de caminhos de diretório locais que determina onde o compilador pesquisará as classes e os pacotes importados. O caminho de classe, às vezes, é chamado de *caminho de criação* ou *caminho de origem*.

Depois de importar adequadamente a classe ou o pacote, você pode usar o nome totalmente qualificado da classe (samples.SampleCode) ou apenas o nome da classe em si (SampleCode).

Os nomes totalmente qualificados são úteis quando classes, métodos ou propriedades com nomes idênticos geram código ambíguo, mas podem ser difíceis de gerenciar se usados para todos os identificadores. Por exemplo, o uso do nome totalmente qualificado gera um código detalhado ao instanciar uma ocorrência da classe SampleCode:

```
var mySample:samples.SampleCode = new samples.SampleCode();
```

Conforme os níveis de pacotes aninhados crescem, a legibilidade do código diminui. Nas situações em que certamente não haverá identificadores ambíguos, você pode tornar seu código mais fácil de ler usando identificadores simples. Por exemplo, a instanciação de uma nova ocorrência da classe SampleCode será bem menos detalhada se você usar somente o identificador de classe:

```
var mySample:SampleCode = new SampleCode();
```

Se você tentar usar os nomes de identificador sem primeiro importar o pacote ou a classe apropriados, o compilador não conseguirá encontrar as definições de classe. Entretanto, se você importar um pacote ou uma classe, qualquer tentativa de definir um nome que entre em conflito com um nome importado irá gerar um erro.

Durante a criação de um pacote, o especificador de acesso padrão para todos os seus membros é `internal`, o que significa que, por padrão, os membros do pacote são visíveis apenas por outros membros do mesmo pacote. Para que uma classe fique disponível para o código fora do pacote, é necessário declará-la como `public`. Por exemplo, o seguinte pacote contém duas classes, SampleCode e CodeFormatter:

```
// SampleCode.as file
package samples
{
    public class SampleCode {}
}

// CodeFormatter.as file
package samples
{
    class CodeFormatter {}
}
```

A classe SampleCode é visível fora do pacote porque é declarada como uma classe `public`. A classe CodeFormatter, porém, é visível somente dentro do próprio pacote de amostras. Se você tentar acessar a classe CodeFormatter fora do pacote de amostras, irá gerar um erro, como mostra o exemplo a seguir:

```
import samples.SampleCode;
import samples.CodeFormatter;
var mySample:SampleCode = new SampleCode(); // okay, public class
var myFormatter:CodeFormatter = new CodeFormatter(); // error
```

Para que as duas classes fiquem disponíveis fora do pacote, é necessário declará-las como `public`. Você não pode aplicar o atributo `public` à declaração do pacote.

Os nomes totalmente qualificados são úteis para resolver conflitos de nome que podem ocorrer durante o uso de pacotes. Esse cenário pode surgir na importação de dois pacotes que definem classes com o mesmo identificador. Por exemplo, considere o seguinte pacote, que também tem uma classe chamada SampleCode:

```
package langref.samples
{
    public class SampleCode {}
}
```

Se você importar as duas classes, como a seguir, terá um conflito de nomes ao fazer referência à classe SampleCode:

```
import samples.SampleCode;
import langref.samples.SampleCode;
var mySample:SampleCode = new SampleCode(); // name conflict
```

O compilador não tem como saber qual classe SampleCode deve usar. Para resolver o conflito, você deve usar o nome totalmente qualificado de cada classe, como a seguir:

```
var sample1:samples.SampleCode = new samples.SampleCode();
var sample2:langref.samples.SampleCode = new langref.samples.SampleCode();
```

Nota: *Os programadores com experiência em C++ costumam confundir a instrução* `import` *com* `#include`*. A diretiva* `#include` *é necessária em C++ porque os compiladores de C++ processam um arquivo por vez e não pesquisam definições de classes em outros arquivos a menos que um arquivo de cabeçalho seja incluído explicitamente. O ActionScript 3.0 tem uma diretiva* `include`*, mas não foi criado para importar classes e pacotes. Para importar classes ou pacotes no ActionScript 3.0, é necessário usar a instrução* `import` *e colocar o arquivo de origem que contém o pacote no caminho da classe.*

Espaços para nomes

Os espaços para nomes fornecem controle sobre a visibilidade das propriedades e dos métodos criados. Pense nos especificadores de controle de acesso `public`, `private`, `protected` e `internal` como espaços para nomes embutidos. Se esses especificadores de controle de acesso predefinidos não atenderem às suas necessidades, você poderá definir seus próprios espaços para nomes.

Se você está familiarizado com espaços para nomes XML, boa parte desta discussão não será novidade, embora a sintaxe e os detalhes da implementação do ActionScript sejam ligeiramente diferentes do XML. Se nunca trabalhou com espaços para nomes antes, o conceito em si é simples, mas a implementação tem uma terminologia específica que você deverá aprender.

Para entender como os espaços para nomes funcionam, é bom saber que o nome de uma propriedade ou método sempre contém duas partes: um identificador e um espaço para nomes. O identificador é o que normalmente entendemos como um nome. Por exemplo, os identificadores na seguinte definição de classe são `sampleGreeting` e `sampleFunction()`:

```
class SampleCode
{
    var sampleGreeting:String;
    function sampleFunction () {
        trace(sampleGreeting + " from sampleFunction()");
    }
}
```

Sempre que as definições não forem precedidas por um atributo de espaço para nomes, seus nomes serão qualificados pelo espaço para nomes `internal` padrão, o que significa que ficam visíveis apenas para os chamadores no mesmo pacote. Se o compilador estiver definido no modo estrito, o compilador emitirá um aviso de que o espaço para nomes `internal` se aplica a qualquer identificador sem um atributo de espaço para nomes. Para garantir que um identificador fique disponível em todo lugar, é necessário que seu nome especificamente seja precedido pelo atributo `public`. No código anterior, `sampleGreeting` e `sampleFunction()` têm um valor de espaço para nomes `internal`.

Há três etapas básicas que devem ser seguidas ao usar espaços para nomes: Primeiro, defina o espaço para nomes usando a palavra-chave `namespace`. Por exemplo, o código a seguir define o espaço para nomes `version1`:

```
namespace version1;
```

Em segundo lugar, você deve aplicar o espaço para nomes usando-o no lugar de um especificador de controle de acesso em uma declaração de propriedade ou método. O exemplo a seguir coloca uma função chamada `myFunction()` no espaço para nomes `version1`:

```
version1 function myFunction() {}
```

Por último, depois de aplicar o espaço para nomes, você pode fazer referência a ele com a diretiva `use` ou qualificando o nome de um identificador com um espaço para nomes. O exemplo a seguir faz referência à função `myFunction()` por meio da diretiva `use`:

```
use namespace version1;
myFunction();
```

Também é possível usar um nome qualificado para fazer referência à função `myFunction()`, como mostra o exemplo a seguir:

```
version1::myFunction();
```

Definição de espaços para nomes

Os espaços para nomes contêm um valor, o URI (Localizador uniforme de recursos), que às vezes é chamado de *nome do espaço para nomes*. Um URI permite garantir que a definição do espaço para nomes seja exclusiva.

Você cria um espaço para nomes declarando uma definição para ele de duas formas. Você pode definir um espaço para nomes com um URI explícito, assim como definiria um espaço para nomes XML, ou pode omitir o URI. O exemplo a seguir mostra como um espaço para nomes pode ser definido usando um URI:

```
namespace flash_proxy = "http://www.adobe.com/flash/proxy";
```

O URI funciona como uma seqüência de caracteres de identificação exclusiva para o espaço para nomes. Se você omitir o URI, como no exemplo a seguir, o compilador criará uma seqüência de caracteres de identificação interna exclusiva no lugar do URI. Você não possui acesso a essa seqüência de caracteres de identificação interna.

```
namespace flash_proxy;
```

Depois de definido, com ou sem um URI, o espaço para nomes não poderá ser redefinido no mesmo escopo. A tentativa de definir um espaço para nomes definido anteriormente no mesmo escopo irá gerar um erro de compilação.

Se for definido dentro de um pacote ou uma classe, talvez o espaço para nomes não fique visível para o código fora do pacote ou da classe, a menos que seja usado o especificador de controle de acesso apropriado. Por exemplo, o seguinte código mostra o espaço para nomes `flash_proxy` definido com o pacote flash.utils. No exemplo a seguir, a falta de um especificador de controle de acesso significa que o espaço para nomes `flash_proxy` deve ser visível apenas para o código dentro do pacote flash.utils e não para qualquer outro código fora do pacote:

```
package flash.utils
{
    namespace flash_proxy;
}
```

O código a seguir usa o atributo `public` para tornar o espaço para nomes `flash_proxy` visível para o código fora do pacote:

```
package flash.utils
{
    public namespace flash_proxy;
}
```

Aplicação de espaços para nomes

Aplicar um espaço para nomes significa colocar uma definição em um espaço para nomes. As definições que podem ser colocadas em espaços para nomes incluem funções, variáveis e constantes (não é possível colocar uma classe em um espaço para nomes personalizado).

Considere, por exemplo, uma função declarada usando o espaço para nomes de controle de acesso `public`. O uso do atributo `public` em uma definição de função coloca a função no espaço para nomes público, tornando-a disponível para todo o código. Depois de definir um espaço para nomes, você pode usá-lo da mesma forma que usa o atributo `public`, e a definição ficará disponível para o código que pode referenciar o seu espaço para nomes personalizado. Por exemplo, se você definir um espaço para nomes `example1`, poderá adicionar um método chamado `myFunction()` usando `example1` como um atributo, como mostra este exemplo:

```
namespace example1;
class someClass
{
    example1 myFunction() {}
}
```

A declaração do método `myFunction()` usando o espaço para nomes `example1` como um atributo significa que o método pertence ao espaço para nomes `example1`.

Tenha em mente o seguinte durante a aplicação de espaços para nomes:

- Você só pode aplicar um espaço para nomes por declaração.
- Não há como aplicar um atributo de espaço para nomes a mais de uma definição por vez. Em outras palavras, se você quiser aplicar seu espaço para nomes a dez funções diferentes, deverá adicioná-lo como um atributo a cada uma das dez definições de função.
- Se aplicar um espaço para nomes, também não será possível definir um especificador de controle de acesso porque espaços para nomes e especificadores de acesso são mutuamente exclusivos. Ou seja, não é possível declarar uma função ou propriedade como `public`, `private`, `protected` ou `internal` e aplicar o espaço para nomes.

Referência a espaços para nomes

Não é necessário fazer referência a um espaço para nomes explicitamente durante o uso de um método ou uma propriedade declarados com qualquer um dos espaços para nomes de controle de acesso, como `public`, `private`, `protected` e `internal`. Isso porque o acesso a esses espaços para nomes especiais é controlado por contexto. Por exemplo, as definições colocadas no espaço para nomes `private` ficam disponíveis automaticamente para o código dentro da mesma classe. Para os espaços para nomes que você definir, porém, essa diferenciação de contexto não existe. Para usar um método ou uma propriedade colocados em um espaço para nomes personalizado, é necessário fazer referência ao espaço para nomes.

Você pode fazer referência a espaços para nomes com a diretiva `use namespace` ou pode qualificar o nome com o espaço para nomes usando o pontuador do qualificador de nome (`::`). A referência a um espaço para nomes com a diretiva `use namespace` "abre" o espaço para nomes, para que ele possa ser aplicado a quaisquer identificadores não qualificados. Por exemplo, se definir o espaço para nomes `example1`, você poderá acessar seus nomes usando `use namespace example1`:

```
use namespace example1;
myFunction();
```

É possível abrir mais de um espaço para nomes por vez. Quando aberto com `use namespace`, o espaço para nomes permanece aberto em todo o bloco de código no qual se encontra. Não há como fechar explicitamente um espaço para nomes.

O uso de mais de um espaço para nomes, contudo, aumenta a probabilidade de conflitos de nome. Se preferir não abrir um espaço para nomes, você poderá evitar a diretiva `use namespace` qualificando o nome do método ou da propriedade com o espaço para nomes e o pontuador do qualificador de nome. Por exemplo, o seguinte código mostra como qualificar o nome `myFunction()` com o espaço para nomes `example1`:

```
example1::myFunction();
```

Uso de espaços para nomes

Um exemplo real de um espaço para nomes usado para evitar conflitos de nome é a classe flash.utils.Proxy que faz parte do ActionScript 3.0. A classe Proxy, que é a substituição para a propriedade `Object.__resolve` do ActionScript 2.0, permite interceptar diferenças em propriedades ou métodos não definidos antes da ocorrência de um erro. Todos os métodos da classe Proxy residem no espaço para nomes `flash_proxy` para evitar conflitos de nome.

Para entender melhor como o espaço para nomes `flash_proxy` é usado, é preciso entender como usar a classe Proxy. A funcionalidade da classe Proxy está disponível somente para suas classes herdadas. Em outras palavras, se quiser usar os métodos da classe Proxy em um objeto, a definição de classe do objeto deve estender a classe Proxy. Por exemplo, para interceptar as tentativas de chamar um método não definido, é necessário estender a classe Proxy e substituir seu método `callProperty()`.

Você deve se lembrar de que a implementação de espaços para nomes, em geral, é um processo de três etapas: definir, aplicar e referenciar um espaço para nomes. Como os métodos da classe Proxy nunca são chamados explicitamente, o espaço para nomes `flash_proxy` é definido e aplicado, mas não referenciado. O ActionScript 3.0 define o espaço para nomes `flash_proxy` e o aplica na classe Proxy. O código precisa apenas aplicar o espaço para nomes `flash_proxy` às classes que estendem a classe Proxy.

O espaço para nomes `flash_proxy` é definido no pacote flash.utils de forma semelhante à seguinte:

```
package flash.utils
{
    public namespace flash_proxy;
}
```

O espaço para nomes é aplicado aos métodos da classe Proxy como mostrado no seguinte trecho dessa classe:

```
public class Proxy
{
    flash_proxy function callProperty(name:*, ... rest):*
    flash_proxy function deleteProperty(name:*):Boolean
    ...
}
```

Como mostra o código a seguir, primeiro você deve importar a classe Proxy e o espaço para nomes `flash_proxy`. Depois, deve declarar sua classe de forma que estenda a classe Proxy (você também deve adicionar o atributo `dynamic` se estiver compilando no modo estrito). Ao substituir o método `callProperty()`, o espaço para nomes `flash_proxy` deve ser usado.

```
package
{
    import flash.utils.Proxy;
    import flash.utils.flash_proxy;

    dynamic class MyProxy extends Proxy
    {
        flash_proxy override function callProperty(name:*, ...rest):*
        {
            trace("method call intercepted: " + name);
        }
    }
}
```

Se criar uma ocorrência da classe MyProxy e chamar um método não definido, tal como o método `testing()` chamado no exemplo a seguir, seu objeto Proxy irá interceptar a chamada de método e executar as instruções dentro do método `callProperty()` (neste caso, uma instrução `trace()` simples).

```
var mySample:MyProxy = new MyProxy();
mySample.testing(); // method call intercepted: testing
```

Há duas vantagens em ter os métodos da classe Proxy dentro do espaço para nomes `flash_proxy`. Primeiro, ter um espaço para nomes separado reduz a desordem na interface pública de qualquer classe que estende a classe Proxy. (Há aproximadamente uma dúzia de métodos na classe Proxy que podem ser substituídos e nenhum foi criado para ser chamado diretamente. Colocá-los no espaço para nomes público poderia gerar confusão.) Em segundo lugar, o uso do espaço para nomes `flash_proxy` evita os conflitos de nome caso a subclasse Proxy contenha métodos de ocorrência com nomes que correspondem a qualquer método da classe Proxy. Por exemplo, você pode querer chamar um de seus métodos de `callProperty()`. O código a seguir é aceitável, porque sua versão do método `callProperty()` está em um espaço para nomes diferente:

```
dynamic class MyProxy extends Proxy
{
    public function callProperty() {}
    flash_proxy override function callProperty(name:*, ...rest):*
    {
        trace("method call intercepted: " + name);
    }
}
```

Os espaços para nomes também podem ser úteis para fornecer acesso a métodos ou propriedades de uma forma que não seria possível com os quatro especificadores de controle de acesso (`public`, `private`, `internal` e `protected`). Por exemplo, você pode ter alguns métodos utilitários espalhados em vários pacotes. Você quer disponibilizar esses métodos para todos os pacotes, mas não quer que sejam públicos. Para fazer isso, você pode criar um novo espaço para nomes e usá-lo como seu próprio especificador de controle de acesso especial.

O exemplo a seguir usa um espaço para nomes definido pelo usuário para agrupar duas funções que residem em pacotes diferentes. Ao agrupá-las no mesmo espaço para nomes, você pode tornar as duas funções visíveis para uma classe ou um pacote por meio de uma única instrução `use namespace`.

Este exemplo usa quatro arquivos para demonstrar a técnica. Todos os arquivos devem estar no caminho de classe. O primeiro deles, myInternal.as, é usado para definir o espaço para nomes `myInternal`. Como o arquivo está em um pacote chamado example, você deve colocá-lo em uma pasta chamada example. O espaço para nomes é marcado como `public` para que possa ser importado para outros pacotes.

```
// myInternal.as in folder example
package example
{
    public namespace myInternal = "http://www.adobe.com/2006/actionscript/examples";
}
```

O segundo e terceiro arquivos, Utility.as e Helper.as, definem as classes que contêm os métodos que devem estar disponíveis para outros pacotes. A classe Utility é um pacote example.alpha, o que significa que o arquivo deve ser colocado dentro de uma pasta chamada alpha, que é uma subpasta da pasta example. A classe Helper é um pacote example.beta, o que significa que o arquivo deve ser colocado dentro de uma pasta chamada beta, que também é uma subpasta da pasta example. Os dois pacotes, example.alpha e example.beta, devem importar o espaço para nomes antes de usá-lo.

```
// Utility.as in the example/alpha folder
package example.alpha
{
    import example.myInternal;

    public class Utility
    {
        private static var _taskCounter:int = 0;

        public static function someTask()
        {
            _taskCounter++;
        }

        myInternal static function get taskCounter():int
        {
            return _taskCounter;
        }
    }
}

// Helper.as in the example/beta folder
package example.beta
{
    import example.myInternal;

    public class Helper
    {
        private static var _timeStamp:Date;

        public static function someTask()
        {
            _timeStamp = new Date();
        }

        myInternal static function get lastCalled():Date
        {
            return _timeStamp;
        }
    }
}
```

O quarto arquivo, NamespaceUseCase.as, é a classe de aplicativo principal e deve ser uma irmã da pasta example. No Adobe Flash CS4 Professional, essa classe deveria ser usada como a classe de documento para o FLA. A classe NamespaceUseCase também importa o espaço para nomes `myInternal` e o usa para chamar os dois métodos estáticos que residem nos outros pacotes. O exemplo usa métodos estáticos apenas para simplificar o código. Os métodos estáticos e de ocorrência podem ser colocados no espaço para nomes `myInternal`.

```
// NamespaceUseCase.as
package
{
    import flash.display.MovieClip;
    import example.myInternal; // import namespace
    import example.alpha.Utility;// import Utility class
    import example.beta.Helper;// import Helper class

    public class NamespaceUseCase extends MovieClip
    {
        public function NamespaceUseCase()
        {
            use namespace myInternal;

            Utility.someTask();
            Utility.someTask();
            trace(Utility.taskCounter); // 2

            Helper.someTask();
            trace(Helper.lastCalled); // [time someTask() was last called]
        }
    }
}
```

Variáveis

As variáveis permitem armazenar valores usados no programa. Para declarar uma variável, você deve usar a instrução `var` com o nome da variável. No ActionScript 2.0, o uso da instrução `var` só é necessário se você usar anotações de tipo. No ActionScript 3.0, o uso da instrução `var` é sempre necessário. Por exemplo, a seguinte linha do ActionScript declara uma variável chamada `i`:

```
var i;
```

Se omitir a instrução `var` ao declarar uma variável, você obterá um erro de compilador no modo estrito e um erro de tempo de execução no modo padrão. Por exemplo, a seguinte linha de código resultará em um erro se a variável `i` não for definida antes:

```
i; // error if i was not previously defined
```

A associação de uma variável a um tipo de dados deve ser feita durante a declaração da variável. A declaração de uma variável sem designar seu tipo é legal, mas gera um aviso do compilador no modo estrito. Um tipo de variável é designado anexando o nome da variável ao caractere dois-pontos (:) seguido do tipo da variável. Por exemplo, o seguinte código declara uma variável `i` que é do tipo int:

```
var i:int;
```

Você atribui um valor à variável usando o operador de atribuição (=). Por exemplo, o seguinte código declara uma variável `i` e lhe atribui o valor 20:

```
var i:int;
i = 20;
```

Pode ser mais conveniente atribuir um valor a uma variável ao mesmo que em que ela é declarada, como no exemplo a seguir:

```
var i:int = 20;
```

A técnica de atribuir um valor a uma variável no momento em que ela é declarada é comumente usado não apenas para atribuir valores primitivos, como inteiros e seqüências de caracteres, mas também para criar uma matriz ou instanciação de uma ocorrência de uma classe. O exemplo a seguir mostra a declaração e a atribuição de um valor a uma matriz usando uma única linha de código:

```
var numArray:Array = ["zero", "one", "two"];
```

É possível criar uma ocorrência de uma classe usando o operador `new`. O exemplo a seguir cria uma ocorrência de uma classe chamada `CustomClass` e atribui uma referência para a ocorrência de classe recém-criada à variável chamada `customItem`:

```
var customItem:CustomClass = new CustomClass();
```

Se tiver mais de uma variável a declarar, você poderá declará-las em uma única linha de código usando o operador vírgula (`,`) para separar as variáveis. Por exemplo, o seguinte código declara três variáveis em uma única linha de código:

```
var a:int, b:int, c:int;
```

Você também pode atribuir valores a cada variável na mesma linha de código. Por exemplo, o seguinte código declara três variáveis (`a`, `b` e `c`), e atribui um valor a cada uma:

```
var a:int = 10, b:int = 20, c:int = 30;
```

Embora você possa usar o operador vírgula para agrupar declarações de variáveis em uma instrução, isso pode reduzir a legibilidade do código.

Noções básicas sobre o escopo de variáveis

O *escopo* de uma variável é a área do código em que a variável pode ser acessada por uma referência léxica. Uma variável *global* é aquela definida em todas as áreas do seu código, enquanto que uma variável *local* é aquela definida apenas em uma parte dele. No ActionScript 3.0, às variáveis é sempre atribuído o escopo da função ou classe em que elas são declaradas. Uma variável global é aquela especificada fora de qualquer definição de função ou classe. Por exemplo, o seguinte código cria uma variável global `i` declarando-a fora de qualquer função: O exemplo mostra que uma variável global está disponível tanto dentro quanto fora da definição da função.

```
var strGlobal:String = "Global";
function scopeTest()
{
    trace(strGlobal); // Global
}
scopeTest();
trace(strGlobal); // Global
```

A variável local é declarada dentro de uma definição de função. A menor área de código para a qual é possível definir uma variável local é uma definição de função. Uma variável local declarada dentro de uma função existirá somente nessa função. Por exemplo, se você declarar uma variável chamada `str2` dentro de uma função chamada `localScope()`, essa variável não ficará disponível fora da função.

```
function localScope()
{
    var strLocal:String = "local";
}
localScope();
trace(strLocal); // error because strLocal is not defined globally
```

Se o nome de variável usado para a variável local já estiver declarado como uma variável global, a definição local ocultará (ou obscurecerá) a definição global enquanto a variável local estiver no escopo. A variável global ainda existirá fora da função. Por exemplo, o código a seguir cria uma variável de seqüência de caracteres global chamada `str1` e uma variável local de mesmo nome dentro da função `scopeTest()`. A instrução `trace` dentro da função gera o valor local da variável, mas a instrução `trace` fora da função gera o valor global da variável.

```
var str1:String = "Global";
function scopeTest ()
{
    var str1:String = "Local";
    trace(str1); // Local
}
scopeTest();
trace(str1); // Global
```

As variáveis do ActionScript, diferentemente de C++ e Java, não possuem escopo em nível de bloqueio. Um código de bloqueio é qualquer grupo de instruções entre uma chave de abertura ({) e uma de fechamento (}). Em algumas linguagens de programação, como C++ e Java, as variáveis declaradas dentro de um bloco de código não ficam disponíveis fora dele. Essa restrição de escopo é chamada de escopo em nível de bloqueio e não existe no ActionScript. Se você declarar uma variável dentro de um bloco de código, ela ficará disponível não apenas nesse bloco, mas também em outras partes da função à qual o bloco pertence. Por exemplo, a seguinte função contém variáveis que são definidas em vários escopos de bloco. Todas as variáveis estão disponíveis na função.

```
function blockTest (testArray:Array)
{
    var numElements:int = testArray.length;
    if (numElements > 0)
    {
        var elemStr:String = "Element #";
        for (var i:int = 0; i < numElements; i++)
        {
            var valueStr:String = i + ": " + testArray[i];
            trace(elemStr + valueStr);
        }
        trace(elemStr, valueStr, i); // all still defined
    }
    trace(elemStr, valueStr, i); // all defined if numElements > 0
}

blockTest(["Earth", "Moon", "Sun"]);
```

Uma implicação interessante da falta de escopo em nível de bloco é que você pode ler ou gravar em uma variável antes que ela seja declarada, contanto que ela seja declarada antes que a função termine. Isso é possível por causa de uma técnica chamada *içamento*, que significa que o compilador move todas as declarações de variável para o início da função. Por exemplo, o código a seguir é compilado muito embora a função inicial `trace()` para a variável `num` ocorra antes que a variável `num` seja declarada:

```
trace(num); // NaN
var num:Number = 10;
trace(num); // 10
```

O compilador, porém, não içará nenhuma instrução de atribuição. Isso explica por que o `trace()` inicial de `num` resulta em `NaN` (e não um número), que é o valor padrão para as variáveis do tipo de dados Number. Isso significa que você pode atribuir valores a variáveis mesmo antes que elas sejam declaradas, como mostra o seguinte exemplo:

```
num = 5;
trace(num); // 5
var num:Number = 10;
trace(num); // 10
```

Valores padrão

Um *valor padrão* é o valor que uma variável contém antes que seu valor seja definido. Uma variável é *inicializada* quando seu valor é definido pela primeira vez. Se você declarar uma variável, mas não definir seu valor, ela será uma variável *não inicializada*. O valor de uma variável não inicializada depende de seu tipo de dados. A tabela a seguir descreve os valores padrão de variáveis, organizados por tipo de dados:

Tipo de dados	**Valor padrão**
Boolean	`false`
int	0
Number	`NaN`
Object	`null`
String	null
uint	0
Não declarado (equivalente à anotação de tipo `*`)	`undefined`
Todas as outras classes, inclusive classes definidas pelo usuário.	`null`

Para variáveis do tipo Number, o valor padrão é `NaN` (e não um número), que é um valor especial definido pelo padrão IEEE-754 para indicar um valor que não representa um número.

Se você declarar uma variável, mas não seu tipo de dados, o tipo de dados padrão `*` será aplicado, o que significa que, na verdade, a variável é sem tipo. Se você também não inicializar uma variável sem tipo com um valor, seu valor padrão será `undefined`.

Para tipos de dados que não forem Boolean, Number, int e uint, o valor padrão de qualquer variável não inicializada será `null`. Isso se aplica a todas as classes definidas pelo ActionScript 3.0, bem como a quaisquer classes personalizadas que você criar.

O valor `null` não é um valor válido para variáveis do tipo Boolean, Number, int ou uint. Se você tentar atribuir um valor `null` a esse tipo de variável, o valor será convertido no valor padrão para esse tipo de dados. Para variáveis do tipo Object, é possível atribuir um valor `null`. Se você tentar atribuir um valor `undefined` a uma variável do tipo Object, o valor será convertido em `null`.

Para variáveis do tipo Number, existe uma função especial de nível superior chamada `isNaN()` que retorna o valor booleano `true` se a variável não for um número e `false` se for.

Tipos de dados

Um *tipo de dados* define um conjunto de valores. Por exemplo, o tipo de dados Boolean é o conjunto de exatamente dois valores: `true` e `false`. Além do tipo de dados Boolean, o ActionScript 3.0 define vários tipos de dados mais comumente usados, como String, Number e Array. Você pode escolher seus próprios tipos de dados usando classes ou interfaces para definir um conjunto de valores personalizado. Todos os valores no ActionScript 3.0, sejam primitivos ou complexos, são objetos.

Um *valor primitivo* é aquele que pertence a um dos seguintes tipos de dados: Boolean, int, Number, String e uint. Trabalhar com valores primitivos, em geral, é mais rápido do que trabalhar com valores complexos, porque o ActionScript armazena valores primitivos de uma forma especial que torna as otimizações de memória e velocidade possíveis.

Nota: *Para os leitores interessados nos detalhes técnicos, o ActionScript armazena valores primitivos internamente como objetos imutáveis. O fato de serem armazenados como objetos imutáveis significa que transmitir por referência, na prática, é o mesmo que transmitir por valor. Isso reduz o uso de memória e aumenta a velocidade de execução, porque as referências são significativamente menores do que os valores em si.*

Um *valor complexo* é um valor que não é primitivo. Os tipos de dados que definem conjuntos de valores complexos incluem Array, Date, Error, Function, RegExp, XML e XMLList.

Muitas linguagens de programação distinguem os valores primitivos dos objetos delimitadores. Java, por exemplo, tem um primitivo int e a classe java.lang.Integer que o delimita. Os primitivos Java não são objetos, mas seus delimitadores são, o que torna os primitivos úteis para algumas operações e os objetos delimitadores mais adequados para outras operações. No ActionScript 3.0, os valores primitivos e seus objetos delimitadores são, na prática, indistinguíveis. Todos os valores, mesmo os primitivos, são objetos. O Flash Player e o Adobe AIR tratam esses tipos primitivos como casos especiais que se comportam como objetos mas não exigem a sobrecarga normal associada à criação de objetos. Isso significa que as duas linhas de código a seguir são equivalentes:

```
var someInt:int = 3;
var someInt:int = new int(3);
```

Todos os tipos de dados primitivos e complexos listados acima são definidos pelas classes centrais do ActionScript 3.0. As classes centrais permitem criar objetos usando os valores literais em vez do operador `new`. Por exemplo, você pode criar uma matriz usando um valor literal ou o construtor de classe Array, como a seguir:

```
var someArray:Array = [1, 2, 3]; // literal value
var someArray:Array = new Array(1,2,3); // Array constructor
```

Verificação de tipos

A verificação de tipos pode ocorrer em tempo de compilação ou de execução. Linguagens tipificadas estatisticamente, como C++ e Java, executam a verificação de tipos em tempo de compilação. Linguagens tipificadas dinamicamente, como Smalltalk e Python, manipulam a verificação de tipos em tempo de execução. Como uma linguagem tipificada dinamicamente, o ActionScript 3.0 tem uma verificação de tipos em tempo de execução, mas também oferece suporte à verificação de tipos em tempo de compilação com um modo de compilador especial chamado *modo estrito*. No modo estrito, a verificação de tipos ocorre em tempo de compilação e de execução, mas no modo padrão, ela ocorre apenas em tempo de execução.

Linguagens tipificadas dinamicamente oferecem grande flexibilidade na estruturação do código, mas às custas de permitir que erros de tipo se manifestem em tempo de execução. Linguagens tipificadas estatisticamente relatam erros de tipo em tempo de compilação, mas exigem que as informações de tipo sejam conhecidas em tempo de compilação.

Verificação de tipos em tempo de compilação

A verificação de tipos em tempo de compilação é mais vantajosa em projetos grandes porque, conforme o tamanho de um projeto aumenta, a flexibilidade do tipo de dados se torna menos importante do que a rápida detecção de erros de tipo. É por isso que, por padrão, o compilador do ActionScript no Adobe Flash CS4 Professional e no Adobe Flex Builder é definido para ser executado no modo restrito.

Para fornecer a verificação de tipos em tempo de compilação, o compilador precisa conhecer as informações de tipo de dados para as variáveis ou expressões no seu código. Para declarar explicitamente um tipo de dados para uma variável, adicione o operador dois-pontos (:) seguido do tipo de dados como um sufixo para o nome da variável. Para associar um tipo de dados a um parâmetro, use o operador dois-pontos seguido do tipo de dados. Por exemplo, o seguinte código adiciona informações de tipo de dados ao parâmetro `xParam` e declara uma variável `myParam` com um tipo de dados explícito:

```
function runtimeTest(xParam:String)
{
    trace(xParam);
}
var myParam:String = "hello";
runtimeTest(myParam);
```

No modo estrito, o compilador do ActionScript relata incompatibilidades de tipos como erros do compilador. Por exemplo, o código a seguir declara um parâmetro de função `xParam`, do tipo Object, mas depois tenta atribuir valores do tipo String e Number ao parâmetro. Isso gera um erro do compilador no modo estrito.

```
function dynamicTest(xParam:Object)
{
    if (xParam is String)
    {
        var myStr:String = xParam; // compiler error in strict mode
        trace("String: " + myStr);
    }
    else if (xParam is Number)
    {
        var myNum:Number = xParam; // compiler error in strict mode
        trace("Number: " + myNum);
    }
}
```

Mesmo no modo estrito, porém, é possível optar seletivamente pela verificação de tipos em tempo de compilação deixando o lado direito de uma instrução de atribuição sem tipos. Você pode marcar uma variável ou expressão como sem tipo omitindo uma anotação de tipo ou usando a anotação de tipo especial de asterisco (`*`). Por exemplo, se o parâmetro `xParam` no exemplo anterior for modificado de forma que não tenha mais uma anotação de tipo, o código compilará no modo estrito:

```
function dynamicTest(xParam)
{
    if (xParam is String)
    {
        var myStr:String = xParam;
        trace("String: " + myStr);
    }
    else if (xParam is Number)
    {
        var myNum:Number = xParam;
        trace("Number: " + myNum);
    }
}
dynamicTest(100)
dynamicTest("one hundred");
```

Verificação de tipos em tempo de execução

A verificação de tipos em tempo de execução ocorrerá no ActionScript 3.0 se você compilar em modo restrito ou em modo de padrão. Considere uma situação em que o valor 3 é transmitido como um argumento para uma função que espera uma matriz. No modo estrito, o compilador irá gerar um erro, porque o valor 3 não é compatível com o tipo de dados Array. Se você desabilitar o modo estrito e executar no modo padrão, o compilador não reclamará sobre incompatibilidade de tipos, mas a verificação em tempo de execução do Flash Player e do Adobe AIR resultará em um erro em tempo de execução.

O exemplo a seguir mostra uma função chamada `typeTest()` que espera um argumento Array mas tem um valor transmitido de 3. Isso gera um erro em tempo de execução no modo padrão, porque o valor 3 não é um membro do tipo de dados (Array) declarado do parâmetro.

```
function typeTest(xParam:Array)
{
    trace(xParam);
}
var myNum:Number = 3;
typeTest(myNum);
// run-time error in ActionScript 3.0 standard mode
```

Também pode haver situações em que há um erro de tipo em tempo de execução mesmo durante a operação no modo estrito. Isso é possível se você usar o modo estrito, mas optar pela verificação de tipos em tempo de compilação, usando uma variável sem tipo. O uso de uma variável sem tipo não elimina a verificação de tipos, mas a suspende até o tempo de execução. Por exemplo, se a variável `myNum` do exemplo anterior não tiver um tipo de dados declarado, o compilador não poderá detectar a incompatibilidade de tipos, mas o Flash Player e o Adobe AIR irão gerar um erro de tempo de execução porque comparam o valor do tempo de execução de `myNum`, que está definido como 3 como resultado da instrução de atribuição, com o tipo de `xParam`, que é definido com o tipo de dados Array.

```
function typeTest(xParam:Array)
{
    trace(xParam);
}
var myNum = 3;
typeTest(myNum);
// run-time error in ActionScript 3.0
```

A verificação de tipos em tempo de execução também permite um uso mais flexível de herança que a verificação em tempo de compilação. Com a suspensão da verificação de tipos para o tempo de execução, o modo padrão permite referenciar as propriedades de uma subclasse mesmo que você a *eleve*. Uma elevação ocorre quando você usa uma classe base para declarar o tipo de uma ocorrência de classe mas usa uma subclasse para instanciá-la. Por exemplo, você pode criar uma classe chamada ClassBase que pode ser estendida (classes com o atributo `final` não podem ser estendidas):

```
class ClassBase
{
}
```

Depois, você pode criar uma subclasse de uma ClassBase chamada ClassExtender, que tem uma propriedade chamada `someString`, como a seguir:

```
class ClassExtender extends ClassBase
{
    var someString:String;
}
```

Usando as duas classes, é possível criar uma ocorrência de classe que é declarada usando o tipo de dados ClassBase, mas instanciada usando o construtor ClassExtender. Uma elevação é considerada uma operação segura, porque a classe base não contém nenhuma propriedade ou método que não esteja na subclasse.

```
var myClass:ClassBase = new ClassExtender();
```

Uma subclasse, porém, contém propriedades ou métodos que sua classe base não contém. Por exemplo, a classe ClassExtender contém a propriedade `someString`, que não existe na classe ClassBase. No modo padrão do ActionScript 3.0, é possível referenciar essa propriedade usando a ocorrência `myClass` sem gerar um erro de tempo de compilação, como mostra o seguinte exemplo:

```
var myClass:ClassBase = new ClassExtender();
myClass.someString = "hello";
// no error in ActionScript 3.0 standard mode
```

O operador is

O operador `is`, que é novo no ActionScript 3.0, permite testar se uma variável ou expressão é um membro de um determinado tipo de dados. Nas versões anteriores do ActionScript, o operador `instanceof` fornecia essa funcionalidade, mas, no ActionScript 3.0, o operador `instanceof` não deve ser usado para testar a associação de tipo de dados. O operador `is` deve ser usado no lugar do operador `instanceof` para verificação de tipos manual, porque a expressão `x instanceof y` apenas verifica a existência de `x` na cadeia de protótipos `y` (e, no ActionScript 3.0, a cadeia de protótipos não fornece um retrato completo da hierarquia de herança).

O operador `is` examina a hierarquia de herança apropriada e pode ser usado para verificar não apenas se um objeto é uma ocorrência de uma classe específica, mas também de uma classe que implementa uma determinada interface. O exemplo a seguir cria uma ocorrência da classe Sprite, chamada `mySprite` e usa o operador `is` para testar se `mySprite` é uma ocorrência das classes Sprite e DisplayObject e se implementa a interface IEventDispatcher:

```
var mySprite:Sprite = new Sprite();
trace(mySprite is Sprite); // true
trace(mySprite is DisplayObject);// true
trace(mySprite is IEventDispatcher); // true
```

O operador `is` verifica a hierarquia de herança e relata adequadamente que `mySprite` é compatível com as classes Sprite e DisplayObject (a classe Sprite é uma subclasse da classe DisplayObject). O operador `is` também verifica se `mySprite` é herdada de alguma classe que implementa a interface IEventDispatcher. Como a classe Sprite é herdada da classe EventDispatcher, que implementa a interface IEventDispatcher, o operador `is` relata corretamente que `mySprite` implementa a mesma interface.

O exemplo a seguir mostra os mesmos testes do exemplo anterior, mas com `instanceof` em vez do operador `is`. O operador `instanceof` identifica corretamente que `mySprite` é uma ocorrência de Sprite ou DisplayObject, mas retorna `false` quando usado para testar se `mySprite` implementa a interface IEventDispatcher.

```
trace(mySprite instanceof Sprite); // true
trace(mySprite instanceof DisplayObject);// true
trace(mySprite instanceof IEventDispatcher); // false
```

O operador as

O operador `as`, que é novo no ActionScript 3.0, também permite verificar se uma expressão é um membro de um determinado tipo de dados. Diferentemente do operador `is`, porém, o operador `as` não retorna um valor booleano. Em vez disso, o operador `as` retorna o valor da expressão em vez de `true` e `null` em vez de `false`. O exemplo a seguir mostra os resultados do uso do operador `as` em vez de `is` no caso simples de verificar se uma ocorrência de Sprite é um membro dos tipos de dados DisplayObject, IEventDispatcher e Number.

```
var mySprite:Sprite = new Sprite();
trace(mySprite as Sprite); // [object Sprite]
trace(mySprite as DisplayObject); // [object Sprite]
trace(mySprite as IEventDispatcher); // [object Sprite]
trace(mySprite as Number);          // null
```

Durante o uso do operador `as`, o operando à direita deve ser um tipo de dados. Uma tentativa de usar uma expressão diferente de um tipo de dados como operando à direita resultará em um erro.

Classes dinâmicas

Uma classe *dynamic* define um objeto que pode ser alterado em tempo de execução adicionando ou alterando propriedades e métodos. Uma classe que não é dinâmica, como a classe String, é uma classe *selada*. Não é possível adicionar propriedades ou métodos a uma classe selada em tempo de execução.

As classes dinâmicas são criadas com o uso do atributo `dynamic` ao declarar uma classe. Por exemplo, o código a seguir cria uma classe dinâmica chamada `Protean`:

```
dynamic class Protean
{
    private var privateGreeting:String = "hi";
    public var publicGreeting:String = "hello";
    function Protean()
    {
        trace("Protean instance created");
    }
}
```

Se, posteriormente, você instanciar uma ocorrência da classe `Protean`, poderá adicionar propriedades ou métodos a ela fora da definição da classe. Por exemplo, o código a seguir cria uma ocorrência da classe `Protean` e adiciona uma propriedade chamada `aString` e outra chamada `aNumber` à ocorrência:

```
var myProtean:Protean = new Protean();
myProtean.aString = "testing";
myProtean.aNumber = 3;
trace(myProtean.aString, myProtean.aNumber); // testing 3
```

As propriedades adicionadas a uma ocorrência de uma classe dinâmica são entidades de tempo de execução, por isso qualquer tipo de verificação é feito em tempo de execução. Não é possível adicionar uma anotação de tipo a uma propriedade adicionada dessa forma.

Você também pode adicionar um método à ocorrência `myProtean` definindo uma função e anexando-a a uma propriedade da ocorrência `myProtean`. O código a seguir move a instrução de rastreamento para um método chamado `traceProtean()`:

```
var myProtean:Protean = new Protean();
myProtean.aString = "testing";
myProtean.aNumber = 3;
myProtean.traceProtean = function ()
{
    trace(this.aString, this.aNumber);
};
myProtean.traceProtean(); // testing 3
```

Os métodos criados dessa forma, entretanto, não têm acesso a qualquer propriedade ou método particular da classe Protean. Além disso, mesmo as referências às propriedades ou métodos públicos da classe `Protean` devem ser qualificados com a palavra-chave `this` ou o nome da classe. O exemplo a seguir mostra a tentativa do método `traceProtean()` de acessar as variáveis particulares e públicas da classe `Protean`.

```
myProtean.traceProtean = function ()
{
    trace(myProtean.privateGreeting); // undefined
    trace(myProtean.publicGreeting); // hello
};
myProtean.traceProtean();
```

Descrições de tipos de dados

Os tipos nativos primitivos incluem Boolean, int, Null, Number, String, uint e void. As classes base do ActionScript também definem os seguintes tipos de dados complexos: Object, Array, Date, Error, Function, RegExp, XML e XMLList.

Tipo de dados Boolean

O tipo de dados Boolean compreende dois valores: `true` e `false`. Nenhum outro valor é válido para variáveis do tipo Boolean. O valor padrão de uma variável Boolean que foi declarada mas não inicializada é `false`.

Tipo de dados int

O tipo de dados int é armazenado internamente como um inteiro de 32 bits e compreende o conjunto de inteiros de

-2,147,483,648 (-2^{31}) a 2,147,483,647 ($2^{31} - 1$), inclusive. As versões anteriores do ActionScript ofereciam apenas o tipo de dados Number, que era usado para números inteiros e de ponto flutuante. No ActionScript 3.0, agora você tem acesso a tipos de computador de nível baixo para inteiros de 32 bits assinados e não assinados. Se a sua variável não tiver números de ponto flutuante, o uso do tipo de dados int em vez do tipo de dados Number deverá ser mais rápido e eficiente.

Para valores inteiros fora do intervalo dos valores int mínimo e máximo, use o tipo de dados Number, que pode manipular valores entre positivo e negativo 9,007,199,254,740,992 (valores inteiros de 53 bits). O valor padrão para variáveis do tipo de dados int é 0.

Tipo de dados Null

O tipo de dados Null contém apenas um valor, `null`. É o valor padrão para o tipo de dados String e todas as classes que definem tipos de dados complexos, inclusive a classe Object. Nenhum outro tipo de dados primitivo, como Boolean, Number, int e uint, contém o valor `null`. O Flash Player e o Adobe AIR converterão o valor `null` no valor padrão apropriado se você tentar atribuir `null` a variáveis do tipo Boolean, Number, int ou uint. Você não pode usar esse tipo de dados como uma anotação de tipo.

Tipo de dados Number

No ActionScript 3.0, o tipo de dados Number pode representar inteiros, inteiros não assinados e números de ponto flutuante. Entretanto, para maximizar o desempenho, você deve usar o tipo de dados Number somente para valores inteiros maiores do que os tipos `int` e `uint`de 32 bits podem armazenar ou para números de ponto flutuante. Para armazenar um número de ponto flutuante, inclua um ponto decimal no número. Se você omitir um ponto decimal, o número será armazenado como um inteiro.

O tipo de dados Number usa o formato de precisão dupla de 64 bits conforme especificado pelo Padrão IEEE para Aritmética de Ponto Flutuante Binário (IEEE-754). Esse padrão determina como os números de ponto flutuante são armazenados usando os 64 bits disponíveis. Um bit é usado para designar se o número é positivo ou negativo. Onze bits são usados para o expoente, que é armazenado como base 2. Os 52 bits restantes são usados para armazenar o *significando* (também chamado de *mantissa*), que é o número elevado à potência indicada pelo expoente.

Com o uso de alguns bits para armazenar um expoente, o tipo de dados Number pode armazenar números de ponto flutuante significativamente maiores do que se usasse todos os bits para o significando. Por exemplo, se o tipo de dados Number usasse os 64 bits para armazenar o significando, ele armazenaria um número tão grande quanto $2^{65} - 1$. Com o uso de 11 bits para armazenar um expoente, o tipo de dados Number pode elevar seu significando à potência de 2^{1023}.

Os valores máximo e mínimo que o tipo Number pode representar são armazenados em propriedades estáticas da classe Number chamadas `Number.MAX_VALUE` e `Number.MIN_VALUE`.

```
Number.MAX_VALUE == 1.79769313486231e+308
Number.MIN_VALUE == 4.940656458412467e-324
```

Embora esse intervalo de números seja enorme, seu custo é a precisão. O tipo de dados Number usa 52 bits para armazenar o significando, por isso os números que exigem mais de 52 bits para fazer uma representação precisa, como a fração 1/3, são apenas aproximações. Se o seu aplicativo exibir precisão absoluta com números decimais, será necessário usar um software que implemente a aritmética de ponto flutuante decimal em vez da aritmética de ponto flutuante binário.

Durante o armazenamento de valore inteiros com o tipo de dados Number, somente os 52 bits do significando são usados. O tipo de dados Number usa esses 52 bits e um bit oculto especial para representar inteiros de -9,007,199,254,740,992 (-2^{53}) a 9,007,199,254,740,992 (2^{53}).

O Flash Player e o Adobe AIR usam o valor `NaN` não apenas como o valor padrão para variáveis do tipo `Number`, mas também como resultado de qualquer operação que deve retornar um número e não retorna. Por exemplo, se você tentar calcular a raiz quadrada de um número negativo, o resultado será `NaN`. Outros valores Number especiais incluem *infinito positivo* e *infinito negativo* .

Nota: *O resultado da divisão por `0` será apenas `NaN` se o divisor também for `0`. A divisão por `0` produz `infinity` quando o dividendo é positivo ou `-infinity` quando o dividendo é negativo.*

Tipo de dados String

O tipo de dados String representa uma seqüência de caracteres de 16 bits. Os Strings são armazenados internamente como caracteres Unicode, usando o formato UTF-16, Eles são valores imutáveis, assim como na linguagem de programação Java. Uma operação sobre um valor String retorna uma nova ocorrência da seqüência de caracteres. O valor padrão para uma variável declarada com o tipo de dados String é `null`. O valor `null` não é o mesmo que a seqüência de caracteres vazia (`""`), muito embora ambos representem a ausência de caracteres.

Tipo de dados uint

O tipo de dados int é armazenado internamente como um inteiro não assinado de 32 bits e compreende o conjunto de inteiros de 0 a 4,294,967,295 (2^{32} - 1), inclusive. Use o tipo de dados uint para circunstâncias especiais que exigem inteiros não negativos. Por exemplo, você deve usar o tipo de dados uint para representar os valores de cor de pixel, porque o tipo de dados int tem um bit de sinal interno que não é apropriado para manipular valores de cor. Para valores inteiros maiores do que o valor uint máximo, use o tipo de dados Number, que pode manipular valores inteiros de 53 bits. O valor padrão para variáveis do tipo de dados uint é 0.

Tipo de dados void

O tipo de dados void contém apenas um valor, `undefined`. Nas versões anteriores do ActionScript, `undefined` era o valor padrão para ocorrências da classe Object. No ActionScript 3.0, o valor padrão para ocorrências de Object é `null`. Se você tentar atribuir um valor `undefined` a uma ocorrência da classe Object, o Flash Player ou o Adobe AIR converterão o valor em `null`. É possível atribuir apenas um valor de `undefined` a variáveis sem tipo. Variáveis sem tipo são aquelas que não possuem nenhuma anotação de tipo ou usam o símbolo asterisco (`*`) para a anotação de tipo. Você pode usar `void` apenas como uma anotação de tipo de retorno.

Tipo de dados Object

O tipo de dados Object é definido pela classe Object. A classe Object serve de classe base para todas as definições de classe no ActionScript. A versão do ActionScript 3.0 do tipo de dados Object difere das versões anteriores de três formas. Primeiro, o tipo de dados Object não é mais o tipo de dados padrão atribuído a variáveis sem nenhuma anotação de tipo. Em segundo lugar, o tipo de dados Object não inclui mais o valor `undefined`, que costumava ser o valor padrão das ocorrências Object. Em terceiro lugar, no ActionScript 3.0, o valor padrão para ocorrências da classe Object é `null`.

Nas versões anteriores do ActionScript, uma variável sem nenhuma anotação de tipo era automaticamente atribuída ao tipo de dados Object. Isso não acontece mais no ActionScript 3.0, que agora inclui a idéia de uma variável realmente sem tipo. As variáveis sem nenhuma anotação de tipo agora são consideradas sem tipo. Se preferir deixar mais claro para os leitores do código que sua intenção é deixar uma variável sem tipo, você pode usar o novo símbolo de asterisco (`*`) para a anotação de tipo, que é equivalente a omitir uma anotação de tipo. O exemplo a seguir mostra duas instruções equivalentes, que declaram uma variável sem tipo `x`:

```
var x
var x:*
```

Somente variáveis sem tipo podem manter o valor `undefined`. Se você tentar atribuir o valor `undefined` a uma variável que possui um tipo de dados, o Flash Player ou o Adobe AIR converterão o valor `undefined` no valor padrão desse tipo de dados. Para ocorrências do tipo de dados Object, o valor padrão é `null`, o que significa que o Flash Player ou o Adobe AIR converterão o valor `undefined` em `null` se você tentar atribuir `undefined` a uma ocorrência de Object.

Conversões de tipo

Uma conversão de tipo ocorre quando um valor é transformado em um valor de um tipo de dados diferente. As conversões de tipo podem ser *implícitas* ou *explícitas*. A conversão implícita, que também é chamada de *coerção*, às vezes, é executada pelo Flash Player ou Adobe AIR em tempo de execução. Por exemplo, se o valor 2 for atribuído a uma variável do tipo de dados Boolean, o Flash Player ou o Adobe AIR converterá o valor 2 no valor booleano `true` antes de atribuir o valor à variável. A conversão explícita, também chamada de *projeção*, ocorre quando o código instrui o compilador a tratar uma variável de um tipo de dados como se pertencesse a um tipo de dados diferente. Quando os valores primitivos estão envolvidos, a projeção realmente converte os valores de um tipo de dados para outro. Para projetar um objeto em um tipo diferente, use parênteses para delimitar o nome do objeto e preceda-o com o nome do novo tipo. Por exemplo, o seguinte código usa um valor booleano e projeta-o em um inteiro:

```
var myBoolean:Boolean = true;
var myINT:int = int(myBoolean);
trace(myINT); // 1
```

Conversões implícitas

As conversões implícitas acontecem em tempo de execução em diversos contextos:

- Em instruções de atribuição
- Quando os valores são transmitidos como argumentos de função
- Quando os valores são retornados de funções
- Em expressões que usam determinados operadores, como o operador de adição (+)

  Para tipos definidos pelo usuário, as conversões implícitas são bem-sucedidas quando o valor a ser convertido é uma ocorrência da classe de destino ou de uma classe derivada dela. Se uma conversão implícita não for bem-sucedida, ocorrerá um erro. Por exemplo, o seguinte código contém uma conversão implícita bem-sucedida e outra malsucedida:

  ```
  class A {}
  class B extends A {}

  var objA:A = new A();
  var objB:B = new B();
  var arr:Array = new Array();

  objA = objB; // Conversion succeeds.
  objB = arr; // Conversion fails.
  ```

  Para os tipos primitivos, as conversões implícitas são tratadas chamando os mesmos algoritmos de conversão internos que são chamados pelas funções de conversão explícitas. As seções a seguir discutem essas conversões de tipo primitivas em detalhes.

Conversões explícitas

É útil usar conversões explícitas, ou projeção, ao compilar no modo estrito, porque pode haver situações em que você não deseja que uma incompatibilidade de tipos gere um erro em tempo de compilação. Pode ser o caso de quando você sabe que a coerção converterá os valores corretamente em tempo de execução. Por exemplo, ao trabalhar com dados recebidos de um formulário, você pode querer contar com a coerção para converter determinados valores de seqüência de caracteres em valores numéricos. O código a seguir gera um erro em tempo de compilação, muito embora o código seja executado corretamente no modo padrão:

```
var quantityField:String = "3";
var quantity:int = quantityField; // compile time error in strict mode
```

Se quiser continuar a usar o modo estrito, mas preferir converter a seqüência de caracteres em um inteiro, você pode usar a conversão explícita da seguinte forma:

```
var quantityField:String = "3";
var quantity:int = int(quantityField); // Explicit conversion succeeds.
```

Projeção para int, uint e Number

É possível projetar qualquer tipo de dados em um dos três tipos de números: int, uint e Number. Se o Flash Player ou o Adobe AIR não conseguir converter o número por algum motivo, o valor padrão de 0 será atribuído aos tipos de dados int e uint, e o valor padrão de `NaN` será atribuído para o tipo de dados Number. Se você converter um valor Boolean em um número, `true` se tornará o valor 1 e `false` se tornará o valor 0.

```
var myBoolean:Boolean = true;
var myUINT:uint = uint(myBoolean);
var myINT:int = int(myBoolean);
var myNum:Number = Number(myBoolean);
trace(myUINT, myINT, myNum); // 1 1 1
myBoolean = false;
myUINT = uint(myBoolean);
myINT = int(myBoolean);
myNum = Number(myBoolean);
trace(myUINT, myINT, myNum); // 0 0 0
```

Os valores String que contêm apenas dígitos pode ser convertidos com êxito em um dos tipos de número. Os tipos de número também podem converter seqüências de caracteres que parecem números negativos ou que representam um valor hexadecimal (por exemplo, `0x1A`). O processo de conversão ignora os caracteres de espaço em branco à esquerda e à direita no valor da seqüência de caracteres. Também é possível projetar seqüências de caracteres que se parecem com números de ponto flutuante usando `Number()`. A inclusão de um ponto decimal faz com que `uint()` e `int()` retornem um inteiro, truncando o decimal e os caracteres seguintes. Por exemplo, os seguintes valores de seqüência de caracteres podem ser projetados em números:

```
trace(uint("5")); // 5
trace(uint("-5")); // 4294967291. It wraps around from MAX_VALUE
trace(uint(" 27 ")); // 27
trace(uint("3.7")); // 3
trace(int("3.7")); // 3
trace(int("0x1A")); // 26
trace(Number("3.7")); // 3.7
```

Os valores String que contêm caracteres não numéricos retornam 0 quando projetados com `int()` ou `uint()` e `NaN` quando projetados com `Number()`. O processo de conversão ignora espaço em branco à esquerda e à direita, mas retorna 0 ou `NaN` se uma seqüência de caracteres tiver espaço em branco separando dois números.

```
trace(uint("5a")); // 0
trace(uint("ten")); // 0
trace(uint("17 63")); // 0
```

No ActionScript 3.0, a função `Number()` não suporta mais números octais, ou de base 8. Se você fornecer uma seqüência de caracteres com um zero à esquerda para a função `Number()` do ActionScript 2.0, o número será interpretado como um número octal e convertido em seu equivalente decimal. Isso não acontece com a função `Number()` no ActionScript 3.0, que em vez disso ignora o zero à esquerda. Por exemplo, o seguinte código gera uma saída diferente quando compilado usando versões diferentes do ActionScript:

```
trace(Number("044"));
// ActionScript 3.0 44
// ActionScript 2.0 36
```

A projeção não é necessária quando um valor de um tipo numérico é atribuído a uma variável de um tipo numérico diferente. Mesmo no modo estrito, os tipos numéricos são implicitamente convertidos em outros tipos numéricos. Isso significa que, em alguns casos, quando o intervalo de um tipo for excedido, o resultado poderá gerar valores inesperados. Os seguintes exemplos compilam no modo estrito, embora alguns gerem valores inesperados:

```
var myUInt:uint = -3; // Assign int/Number value to uint variable
trace(myUInt); // 4294967293

var myNum:Number = sampleUINT; // Assign int/uint value to Number variable
trace(myNum) // 4294967293

var myInt:int = uint.MAX_VALUE + 1; // Assign Number value to uint variable
trace(myInt); // 0

myInt = int.MAX_VALUE + 1; // Assign uint/Number value to int variable
trace(myInt); // -2147483648
```

A tabela a seguir resume os resultados da projeção para os tipos de dados Number, int ou uint a partir de outros tipos de dados.

Tipo de dados ou valor	Resultado da conversão em Number, int ou uint
Boolean	Se o valor for `true`, 1; caso contrário, 0.
Date	A representação interna do objeto Date, que é o número de milésimos de segundo desde a meia-noite de 1º de janeiro de 1970, hora universal.
`null`	0
Object	Se a ocorrência for `null` e convertida para Number, `NaN`; caso contrário, 0.
String	Um número se o Flash Player ou o Adobe AIR puderem converter a seqüência de caracteres em um número; caso contrário, `NaN` se convertida em Number ou 0 se convertida em int ou uint.
`undefined`	Se convertido em Number, `NaN`; se convertido em int ou uint, 0.

Projeção para Boolean

A projeção para Boolean a partir de qualquer tipo de dados numérico (uint, int e Number) resultará em `false` se o valor numérico for 0 e em `true` se não for. Para o tipo de dados Number, o valor `NaN` também resulta em `false`. O exemplo a seguir mostra os resultados da projeção dos números em -1, 0 e 1:

```
var myNum:Number;
for (myNum = -1; myNum<2; myNum++)
{
    trace("Boolean(" + myNum +") is " + Boolean(myNum));
}
```

A saída do exemplo mostra que, dos três números, somente 0 retorna um valor `false`:

```
Boolean(-1) is true
Boolean(0) is false
Boolean(1) is true
```

A projeção para Boolean de um valor String retornará `false` se a seqüência de caracteres for `null` ou vazia (`""`). Do contrário, retornará `null`.

```
var str1:String; // Uninitialized string is null.
trace(Boolean(str1)); // false

var str2:String = ""; // empty string
trace(Boolean(str2)); // false

var str3:String = " "; // white space only
trace(Boolean(str3)); // true
```

A projeção para Boolean de uma ocorrência da classe Object retornará `false` se a ocorrência for `null`; do contrário, retornará `true`:

```
var myObj:Object; // Uninitialized object is null.
trace(Boolean(myObj)); // false

myObj = new Object(); // instantiate
trace(Boolean(myObj)); // true
```

As variáveis Boolean obtém tratamento especial no modo estrito no que se refere a atribuir valores de qualquer tipo de dados a uma variável Boolean sem projeção. A coerção implícita de todos os tipos de dados para o tipo de dados Boolean ocorre mesmo no modo estrito. Em outras palavras, diferentemente de quase todos os outros tipos de dados, a projeção para Boolean não é necessária para evitar erros no modo estrito. Os seguintes exemplos compilam no modo estrito e se comportam conforme o esperado em tempo de execução:

```
var myObj:Object = new Object(); // instantiate
var bool:Boolean = myObj;
trace(bool); // true
bool = "random string";
trace(bool); // true
bool = new Array();
trace(bool); // true
bool = NaN;
trace(bool); // false
```

A tabela a seguir resume os resultados da projeção para o tipo de dados Boolean a partir de outros tipos de dados:

Tipo de dados ou valor	Resultado da conversão em Boolean
String	`false` se o valor for `null` ou uma seqüência de caracteres vazia (`""`); caso contrário, `true`.
`null`	`false`
Number, int ou uint	`false` se o valor for `NaN` ou 0; caso contrário, `true`.
Object	`false` se a ocorrência for `null`; caso contrário, `true`.

Projeção para String

A projeção para o tipo de dados String de qualquer tipo de dados numérico retorna uma representação de seqüência de caracteres do número. A projeção para o tipo de dados String de um valor Boolean retornará a seqüência de caracteres `"true"` se o valor for `true` e retornará a seqüência de caracteres `"false"` se o valor for `false`.

A projeção para String de uma ocorrência da classe Object retornará a seqüência de caracteres `"null"` se a ocorrência for `null`. Caso contrário, a projeção para o tipo String da classe Object retornará a seqüência de caracteres `"[object Object]"` .

A projeção para String de uma ocorrência da classe Array retorna uma seqüência de caracteres incluindo uma lista delimitada por vírgula de todos os elementos da matriz. Por exemplo, a seguinte projeção para o tipo de dados String retorna uma seqüência de caracteres contendo os três elementos da matriz:

```
var myArray:Array = ["primary", "secondary", "tertiary"];
trace(String(myArray)); // primary,secondary,tertiary
```

A projeção para String de uma ocorrência da classe Date retorna uma representação da seqüência de caracteres da data que a ocorrência contém. Por exemplo, o seguinte exemplo retorna uma representação da seqüência de caracteres da ocorrência da classe Date (a saída mostra o resultado para o Horário de Verão do Pacífico):

```
var myDate:Date = new Date(2005,6,1);
trace(String(myDate)); // Fri Jul 1 00:00:00 GMT-0700 2005
```

A tabela a seguir resume os resultados da projeção para o tipo de dados String a partir de outros tipos de dados:

Tipo de dados ou valor	Resultado da conversão em String
Array	Uma seqüência de caracteres incluindo todos os elementos de matriz.
Boolean	`"true"` ou `"false"`
Date	Uma representação da seqüência de caracteres do objeto Date.
`null`	`"null"`
Number, int ou uint	Uma representação da seqüência de caracteres do número.
Object	Se a ocorrência for null, `"null"`; caso contrário, `"[object Object]"`.

Sintaxe

A sintaxe de uma linguagem define um conjunto de regras que deve ser seguido durante a escrita de código executável.

Diferenciação entre maiúsculas e minúsculas

O ActionScript 3.0 é uma linguagem que diferencia maiúsculas e minúsculas. Os identificadores que diferem somente em maiúsculas e minúsculas são considerados identificadores diferentes. Por exemplo, o código a seguir cria duas variáveis diferentes:

```
var num1:int;
var Num1:int;
```

Sintaxe de pontos

O operador dot (.) fornece uma maneira de acessar as propriedades e os métodos de um objeto. Com o uso da sintaxe de pontos, é possível fazer referência a uma propriedade ou um método de classe usando um nome de ocorrência, seguido do operador dot e do nome da propriedade ou do método. Por exemplo, considere a seguinte definição de classe:

```
class DotExample
{
    public var prop1:String;
    public function method1():void {}
}
```

Com o uso da sintaxe de pontos, é possível acessar a propriedade `prop1` e o método `method1()` usando o nome de ocorrência criado no seguinte código:

```
var myDotEx:DotExample = new DotExample();
myDotEx.prop1 = "hello";
myDotEx.method1();
```

É possível usar a sintaxe de pontos para definir pacotes. O operador dot é usado para fazer referência a pacotes aninhados. Por exemplo, a classe EventDispatcher reside em um pacote chamado eventos que é aninhado dentro do pacote chamado flash. Você pode fazer referência aos pacotes de eventos usando a seguinte expressão:

```
flash.events
```

Você também pode fazer referência à classe EventDispatcher usando esta expressão:

```
flash.events.EventDispatcher
```

Sintaxe de barras

A sintaxe de barras não é suportada no ActionScript 3.0. Ela foi usada nas versões anteriores do ActionScript para indicar o caminho de um clipe de filme ou variável.

Literais

Um *literal* é um valor que aparece diretamente em seu código. Os seguintes exemplos são de literais:

```
17
"hello"
-3
9.4
null
undefined
true
false
```

Os literais também podem ser agrupados para formar literais compostos. Os literais de matriz são colocados entre colchetes (`[]`) e usam a vírgula para separar elementos de matriz.

Um literal de matriz pode ser usado para inicializar uma matriz. Os exemplos a seguir mostram duas matrizes que são inicializadas usando literais de matriz. É possível usar a instrução `new` e transmitir o literal composto como um parâmetro para o construtor de classe Array, mas você também pode atribuir valores literais diretamente ao instanciar ocorrências das seguintes classes centrais do ActionScript: Object, Array, String, Number, int, uint, XML, XMLList e Boolean.

```
// Use new statement.
var myStrings:Array = new Array(["alpha", "beta", "gamma"]);
var myNums:Array = new Array([1,2,3,5,8]);

// Assign literal directly.
var myStrings:Array = ["alpha", "beta", "gamma"];
var myNums:Array = [1,2,3,5,8];
```

Os literais também podem ser usado para inicializar um objeto genérico. Um objeto genérico é uma ocorrência da classe Object. Os literais Object são colocados entre chaves (`{}`) e usam a vírgula para separar propriedades de objetos. Cada propriedade é declarada com o caractere dois-pontos (`:`), que separa o nome da propriedade do valor da propriedade.

É possível criar um objeto genérico usando a instrução `new` e transmitir o literal de objeto como um parâmetro para o construtor de classe Objeto ou atribuir o literal de objeto diretamente à ocorrência que você está declarando. O exemplo a seguir demonstra duas formas alternativas de criar um novo objeto genérico e inicializar o objeto com três propriedades (`propA`, `propB` e `propC`), cada uma com valores definidos como 1, 2, e 3, respectivamente:

```
// Use new statement and add properties.
var myObject:Object = new Object();
myObject.propA = 1;
myObject.propB = 2;
myObject.propC = 3;

// Assign literal directly.
var myObject:Object = {propA:1, propB:2, propC:3};
```

Para obter mais informações, consulte "Noções básicas de strings" na página 143, "Noções básicas de expressões regulares:" na página 209 e "Inicialização de variáveis XML" na página 238.

Ponto-e-vírgula

Você pode usar o caractere ponto-e-vírgula (`;`) para encerrar uma instrução. Opcionalmente, se omitir o caractere ponto-e-vírgula, o compilador presumirá que cada linha de código representa uma única instrução. Como muitos programadores estão acostumados a usar o ponto-e-vírgula para denotar o fim de uma instrução, seu código poderá ser mais fácil de ler se você usar consistentemente ponto-e-vírgula para encerrar as instruções.

O uso de um ponto-e-vírgula para encerrar uma instrução permite colocar mais de uma instrução em uma única linha, mas isso pode tornar o código mais difícil de ler.

Parênteses

Você pode usar parênteses (`()`) de três formas no ActionScript 3.0. Primeiro, você pode usar parênteses para alterar a ordem das operações em uma expressão. As operações que são agrupadas dentro de parênteses são sempre executadas primeiro. Por exemplo, os parênteses são usados para alterar a ordem das operações no seguinte código:

```
trace(2 + 3 * 4); // 14
trace((2 + 3) * 4); // 20
```

Em segundo lugar, você pode usar parênteses com o operador vírgula (`,`) para avaliar uma série de expressões e retornar o resultado da expressão final, como mostra o seguinte exemplo:

```
var a:int = 2;
var b:int = 3;
trace((a++, b++, a+b)); // 7
```

Em terceiro lugar, você pode usar parênteses para transmitir um ou mais parâmetros para funções ou métodos, como mostra o exemplo a seguir, que transmite um valor String para a função `trace()`:

```
trace("hello"); // hello
```

Comentários

O código do ActionScript 3.0 oferece suporte a dois tipos de comentários: comentários de uma única linha e de várias linhas. Esses mecanismos de comentários são semelhantes aos do C++ e Java. O compilador irá ignorar o texto marcado com um comentário.

Os comentários de uma única linha começam com dois caracteres de barra inclinada (`//`) e continuam até o fim da linha. Por exemplo, o seguinte código contém um comentário de uma única linha:

```
var someNumber:Number = 3; // a single line comment
```

Os comentários de várias linhas começam com uma barra inclinada e um asterisco (`/*`) e terminam com um asterisco e uma barra inclinada (`*/`).

```
/* This is multiline comment that can span
more than one line of code. */
```

Palavras-chave e palavras reservadas

As *palavras reservadas* são palavras que não podem ser usadas como identificadores no código porque são para uso do ActionScript. Elas incluem *palavras-chave léxicas*, que são removidas do espaço para nomes do programa pelo compilador. O compilador relatará um erro se você usar uma palavra-chave léxica como identificador. A seguinte tabela lista as palavras-chave léxicas do ActionScript 3.0.

as	break	case	catch
class	const	continue	default
delete	do	else	extends
false	finally	for	function
if	implements	import	in
instanceof	interface	internal	is
native	new	null	package
private	protected	public	return
super	switch	this	throw
to	true	try	typeof
use	var	void	while
with			

Há um pequeno conjunto de palavras-chave, chamado *palavras-chave sintáticas*, que pode ser usado como identificador, mas têm um significado especial em determinados contextos. A tabela a seguir lista as palavras-chave sintáticas do ActionScript 3.0.

each	get	set	namespace
include	dynamic	final	native
override	static		

Também há vários identificadores que, às vezes, são referidos como *palavras reservadas futuras*. Eles não são reservados pelo ActionScript 3.0, embora alguns sejam tratados como palavras-chave pelo software que incorpora o ActionScript 3.0. Você pode usar vários desses identificadores no seu código, mas a Adobe não recomenda essa prática porque eles podem aparecer como palavras-chave em uma versão subseqüente da linguagem.

abstract	boolean	byte	cast
char	debugger	double	enum
export	float	goto	intrinsic
long	prototype	short	synchronized
throws	to	transient	type
virtual	volatile		

Constantes

O ActionScript 3.0 oferece suporte à instrução `const`, que você pode usar para criar constantes. As constantes são propriedades com um valor fixo que não pode ser alterado. Você pode atribuir um valor a uma constante apenas uma vez, e a atribuição deve ocorrer próxima à declaração da constante. Por exemplo, se uma constante for declarada como um membro de uma classe, você poderá atribuir-lhe um valor somente como parte da declaração ou dentro do construtor de classe.

O seguinte código declara duas constantes. A primeira, `MINIMUM`, tem um valor atribuído como parte da instrução de declaração. A segunda, `MAXIMUM`, tem um valor atribuído no construtor. Observe que este exemplo é compilado apenas no modo padrão porque o modo estrito só permite que um valor de constante seja atribuído em tempo de inicialização.

```
class A
{
    public const MINIMUM:int = 0;
    public const MAXIMUM:int;

    public function A()
    {
        MAXIMUM = 10;
    }
}

var a:A = new A();
trace(a.MINIMUM); // 0
trace(a.MAXIMUM); // 10
```

Um erro será gerado se você tentar atribuir um valor inicial a uma constante de qualquer outra forma. Por exemplo, se você tentar definir o valor inicial de `MAXIMUM` fora da classe, ocorrerá um erro de tempo de execução.

```
class A
{
public const MINIMUM:int = 0;
public const MAXIMUM:int;
}

var a:A = new A();
a["MAXIMUM"] = 10; // run-time error
```

O ActionScript 3.0 define uma ampla gama de constantes para sua conveniência. Por convenção, as constantes do ActionScript usam tudo em letras maiúsculas, com as palavras separadas pelo caractere de sublinhado (_). Por exemplo, a definição de classe MouseEvent usa essa convenção de nomenclatura para suas constantes, cada uma representando um evento relacionado à entrada do mouse:

```
package flash.events
{
    public class MouseEvent extends Event
    {
    public static const CLICK:String = "click";
    public static const DOUBLE_CLICK:String = "doubleClick";
    public static const MOUSE_DOWN:String = "mouseDown";
    public static const MOUSE_MOVE:String = "mouseMove";
    ...
    }
}
```

Operadores

Os operadores são funções especiais que usam um ou mais operandos e retornam um valor. Um *operando* é um valor, normalmente um literal, uma variável ou uma expressão, que um operador usa como entrada. Por exemplo, no código a seguir, os operadores de adição (`+`) e de multiplicação (`*`) são usados com três operandos literais (`2`, `3`, e `4`) para retornar um valor. Esse valor é usado pelo operador de atribuição (`=`) para atribuir o valor retornado, 14, para a variável `sumNumber`.

```
var sumNumber:uint = 2 + 3 * 4; // uint = 14
```

Os operadores podem ser unários, binários ou ternários. Um operador *unário* usa um operando. Por exemplo, o operador de incremento (`++`) é um operador unário, porque usa apenas um operando. Um operador *binário* usa dois operandos. Por exemplo, o operador de divisão (`/`) usa dois operandos. Um operador *ternário* usa três operandos. Por exemplo, o operador condicional (`/`) usa três operandos.

Alguns operadores são *sobrecarregados*, o que significa que se comportam de forma diferente dependendo do tipo ou da quantidade de operandos que lhes são transmitidos. O operador de adição (`+`) é um exemplo de um operador sobrecarregado que se comporta de maneira diferente dependendo do tipo de dados dos operandos. Se os dois operandos forem números, o operador de adição retornará a soma dos valores. Se os dois operandos forem seqüências de caracteres, o operador de adição retornará a concatenação dos dois operandos. O código de exemplo a seguir mostra como o operador se comporta de forma diferente dependendo dos operandos:

```
trace(5 + 5); // 10
trace("5" + "5"); // 55
```

Os operadores também podem se comportar de forma diferente com base no número de operandos fornecidos. O operador de subtração (`-`) é um operador unário e binário. Quando fornecido com apenas um operando, o operador de subtração nega o operando e retorna o resultado. Quando fornecido com dois operandos, o operador de subtração retorna a diferença entre os operandos. O exemplo a seguir mostra o operador de subtração usado primeiro como um operador unário e depois como binário.

```
trace(-3); // -3
trace(7 - 2); // 5
```

Precedência e associatividade de operadores

A precedência e a associatividade de operadores determina a ordem na qual os operadores são processados. Embora possa parecer natural aos que estão familiarizados com aritmética que o compilador processe o operador de multiplicação (`*`) antes do operador de adição (`+`), o compilador precisa de instruções explícitas sobre os operadores que deve processar primeiro. Essas instruções são denominadas coletivamente de *precedência de operador*. O ActionScript define a precedência de um operador padrão que você pode alterar usando o operador parênteses (`()`). Por exemplo, o seguinte código altera a precedência padrão no exemplo anterior para forçar o compilador a processar o operador de adição antes do operador de multiplicação:

```
var sumNumber:uint = (2 + 3) * 4; // uint == 20
```

Você pode encontrar situações em que dois ou mais operadores da mesma precedência aparecem na mesma expressão. Nesses casos, o compilador usa as regras de *associatividade* para determinar qual operador será processado primeiro. Todos os operadores binários, exceto os de atribuição, são *associativos à esquerda*, o que significa que os operadores à esquerda são processados antes dos que estão à direita. Todos os operadores de atribuição e o operador condicional (`?:`) são *associativos à direita*, o que significa que os operadores à direita são processados antes dos que estão à esquerda.

Por exemplo, considere os operadores "menor do que" (<) e "maior do que" (>), que têm a mesma precedência. Se os dois forem usados na mesma expressão, o operador à esquerda será processado primeiro porque os dois operadores são associativos à esquerda. Isso significa que as duas instruções a seguir produzem a mesma saída:

```
trace(3 > 2 < 1); // false
trace((3 > 2) < 1); // false
```

O operador maior do que é processado primeiro, o que resulta em um valor `true`, porque o operando 3 é maior do que o operando 2. O valor `true` é transmitido para o operador menor do que junto com o operando 1. O seguinte código representa esse estado intermediário:

```
trace((true) < 1);
```

O operador menor do que converte o valor `true` no valor numérico 1 e compara esse valor numérico com o segundo operando 1 para retornar o valor `false` (o valor 1 não é menor que 1).

```
trace(1 < 1); // false
```

É possível alterar a associatividade à esquerda padrão com o operador parênteses. Você pode instruir o compilador a processar o operador menor do que primeiro, colocando esse operador e seus operandos entre parênteses. O exemplo a seguir usa o operador parênteses para produzir uma saída diferente usando os mesmos números que o exemplo anterior:

```
trace(3 > (2 < 1)); // true
```

O operador menor do que é processado primeiro, o que resulta em um valor `false`, porque o operando 2 não é menor que o operando 1. O valor `false` é transmitido para o operador maior do que junto com o operando 3. O seguinte código representa esse estado intermediário:

```
trace(3 > (false));
```

O operador maior do que converte o valor `false` no valor numérico 0 e compara esse valor numérico com o outro operando 3 para retornar o valor `true` (o valor 3 é maior que 0).

```
trace(3 > 0); // true
```

A tabela a seguir lista os operadores para o ActionScript 3.0 em ordem decrescente de precedência. Cada linha da tabela contém operadores de mesma precedência. Cada linha de operadores tem precedência sobre a linha que aparece abaixo dela na tabela.

Grupo	**Operadores**
Primário	`[] {x:y} () f(x) new x.y x[y] <></> @ :: ..`
Sufixo	`x++ x--`
Unário	`++x --x + - ~ ! delete typeof void`
Multiplicativo	`* / %`
Aditivo	`+ -`
Desvio em nível de bits	`<< >> >>>`
Relacional	`< > <= >= as in instanceof is`
Igualdade	`== != === !==`
AND em nível de bits	`&`
XOR em nível de bits	`^`
OR em nível de bits	`|`

Grupo	Operadores
AND lógico	`&&`
OR lógico	`||`
Condicional	`?:`
Atribuição	`= *= /= %= += -= <<= >>= >>>= &= ^= |=`
Vírgula	`,`

Operadores primários

Os operadores primários incluem aqueles usados para criar literais Array e Object, agrupar expressões, chamar funções, instanciar ocorrências de classes e acessar propriedades.

Todos os operadores primários, conforme listados na tabela a seguir, têm a mesma precedência. Os operadores que fazem parte da especificação E4X são indicados pela notação (E4X).

Operador	Operação executada
`[]`	Inicializa uma matriz
`{x:y}`	Inicializa um objeto
`()`	Agrupa expressões
`f(x)`	Chama uma função
`new`	Chama um construtor
`x.y x[y]`	Acessa uma propriedade
`<></>`	Inicializa um objeto XMLList (E4X)
`@`	Acessa um atributo (E4X)
`::`	Qualifica um nome (E4X)
`..`	Acessa um elemento XML descendente (E4X)

Operadores de sufixo

Os operadores de sufixo usam um operador e incrementam ou decrementam o valor. Embora esses operadores sejam unários, eles são classificados separadamente do resto dos operadores unários por causa de sua maior precedência e seu comportamento especial. Quando um operador de sufixo é usado como parte de uma expressão maior, o valor da expressão é retornado antes que o operador de sufixo seja processado. Por exemplo, o seguinte código mostra como o valor da expressão `xNum++` é retornado antes de ser incrementado:

```
var xNum:Number = 0;
trace(xNum++); // 0
trace(xNum); // 1
```

Todos os operadores de sufixo, conforme listados na tabela a seguir, têm a mesma precedência:

Operador	Operação executada
`++`	Incrementos (sufixo)
`--`	Decrementos (sufixo)

Operadores unários

Os operadores unários usam um operando. Os operadores de incremento (`++`) e de decremento (`--`) deste grupo são *operadores deprefixo*, o que significa que aparecem antes do operando em uma expressão. Os operadores de prefixo diferem dos de sufixo na operação de incremento ou decremento que é executada antes que o valor da expressão geral seja retornado. Por exemplo, o seguinte código mostra como o valor da expressão `++xNum` é retornado depois de ser incrementado:

```
var xNum:Number = 0;
trace(++xNum); // 1
trace(xNum); // 1
```

Todos os operadores unários, conforme listados na tabela a seguir, têm a mesma precedência:

Operador	Operação executada
`++`	Incrementos (prefixo)
`--`	Decrementos (prefixo)
`+`	Unário +
`-`	Unário - (negativa)
`!`	NOT lógico
`~`	NOT em nível de bits
`delete`	Exclui uma propriedade
`typeof`	Retorna informações de tipo
`void`	Retorna um valor indefinido

Operadores multiplicativos

Os operadores multiplicativos usam dois operandos e executam cálculos de multiplicação, divisão ou módulo.

Todos os operadores multiplicativos, conforme listados na tabela a seguir, têm a mesma precedência:

Operador	Operação executada
`*`	Multiplicação
`/`	Divisão
`%`	Módulo

Operadores aditivos

Os operadores aditivos usam dois operandos e executam cálculos de adição ou subtração. Todos os operadores aditivos, conforme listados na tabela a seguir, têm a mesma precedência:

Operador	Operação executada
`+`	Adição
`-`	Subtração

Operadores de desvio em nível de bits

Os operadores de desvio em nível de bits usam dois operandos e desviam os bits do primeiro até o ponto especificado pelo segundo operando. Todos os operadores de desvio em nível de bits, conforme listados na tabela a seguir, têm a mesma precedência:

Operador	Operação executada
`<<`	Desvio à esquerda em nível de bits
`>>`	Desvio à direita em nível de bits
`>>>`	Desvio à direita não assinado em nível de bits

Operadores relacionais

Os operadores relacionais usam dois operandos, comparam seus valores e retornam um valor booleano. Todos os operadores relacionais, conforme listados na tabela a seguir, têm a mesma precedência:

Operador	Operação executada
`<`	Menor do que
`>`	Maior do que
`<=`	Menor do que ou igual a
`>=`	Maior do que ou igual a
`as`	Verifica tipo de dados
`in`	Verifica propriedades de objeto
`instanceof`	Verifica cadeia de protótipos
`is`	Verifica tipo de dados

Operadores de igualdade

Os operadores de igualdade usam dois operandos, comparam seus valores e retornam um valor booleano. Todos os operadores de igualdade, conforme listados na tabela a seguir, têm a mesma precedência:

Operador	Operação executada
`==`	Igualdade
`!=`	Desigualdade
`===`	Igualdade estrita
`!==`	Desigualdade estrita

Operadores lógicos em nível de bits

Os operadores lógicos em nível de bits usam dois operandos e executam operações lógicas em nível de bits. Os operadores lógicos em nível de bits diferem na precedência e são listados na tabela a seguir em ordem decrescente de precedência:

Operador	Operação executada
`&`	AND em nível de bits
`^`	XOR em nível de bits
`|`	OR em nível de bits

Operadores lógicos

Os operadores lógicos usam dois operandos e retornam um valor booleano. Os operadores lógicos diferem em precedência e são listados na tabela a seguir em ordem decrescente de precedência:

Operador	Operação executada
`&&`	AND lógico
`||`	OR lógico

Operador condicional

O operador condicional é um operador ternário, o que significa que usa três operandos. Ele é um método útil para aplicar a instrução condicional `if..else` .

Operador	Operação executada
`?:`	Condicional

Operadores de atribuição

Os operadores de atribuição usam dois operandos e atribuem um valor a um deles, com base no valor do outro. Todos os operadores de atribuição, conforme listados na tabela a seguir, têm a mesma precedência:

Operador	Operação executada
`=`	Atribuição
`*=`	Atribuição de multiplicação
`/=`	Atribuição de divisão
`%=`	Atribuição de módulo
`+=`	Atribuição de adição
`-=`	Atribuição de subtração
`<<=`	Atribuição de desvio à esquerda em nível de bits
`>>=`	Atribuição de desvio à direita em nível de bits
`>>>=`	Atribuição de desvio à direita não assinado em nível de bits
`&=`	Atribuição AND em nível de bits
`^=`	Atribuição XOR em nível de bits
`|=`	Atribuição OR em nível de bits

Condicionais

O ActionScript 3.0 fornece três instruções condicionais básicas que você pode usar para controlar o fluxo de programa.

if..else

A instrução condicional `if..else` permite testar uma condição e executar um bloco de código se essa condição existir, ou executar um bloco de código alternativo se ela não existir. Por exemplo, o seguinte código testa se o valor `x` excede 20, gera uma função `trace()` em caso afirmativo ou gera uma função `trace()` diferente em caso negativo:

```
if (x > 20)
{
    trace("x is > 20");
}
else
{
    trace("x is <= 20");
}
```

Se não quiser executar um bloco de código alternativo, você poderá usar a instrução `if` sem a instrução `else`.

if..else if

É possível testar mais de uma condição usando a instrução condicional `if..else if`. Por exemplo, o código a seguir não apenas testa se o valor `x` excede 20, mas também se o valor `x` é negativo:

```
if (x > 20)
{
    trace("x is > 20");
}
else if (x < 0)
{
    trace("x is negative");
}
```

Se uma instrução `if` ou `else` for seguida de apenas uma instrução, a instrução não precisa ficar entre chaves. Por exemplo, o código a seguir não usa chaves:

```
if (x > 0)
    trace("x is positive");
else if (x < 0)
    trace("x is negative");
else
    trace("x is 0");
```

Entretanto, a Adobe recomenda que você sempre use chaves, porque poderá ocorrer um comportamento inesperado se instruções forem adicionadas posteriormente a uma instrução condicional sem chaves. Por exemplo, no código a seguir, o valor `positiveNums` é aumentado em 1 quer a condição seja ou não avaliada como `true`:

```
var x:int;
var positiveNums:int = 0;

if (x > 0)
    trace("x is positive");
    positiveNums++;

trace(positiveNums); // 1
```

switch

A instrução `switch` será útil se você tiver vários caminhos de execução que dependam da mesma expressão de condição. Ela fornece uma funcionalidade semelhante a uma longa série de instruções `if..else if`, mas é mais fácil de ler. Em vez de testar uma condição quanto a um valor booleano, a instrução `switch` avalia uma expressão e usa o resultado para determinar qual bloco de código será executado. Os blocos de código começam com uma instrução `case` e terminam com uma instrução `break`. Por exemplo, a seguinte instrução `switch` imprime o dia da semana, com base no número de dias retornado pelo método `Date.getDay()`:

```
var someDate:Date = new Date();
var dayNum:uint = someDate.getDay();
switch(dayNum)
{
    case 0:
        trace("Sunday");
        break;
    case 1:
        trace("Monday");
        break;
    case 2:
        trace("Tuesday");
        break;
    case 3:
        trace("Wednesday");
        break;
    case 4:
        trace("Thursday");
        break;
    case 5:
        trace("Friday");
        break;
    case 6:
        trace("Saturday");
        break;
    default:
        trace("Out of range");
        break;
}
```

Repetição

As instruções de repetição permitem executar um bloco específico de código repetidamente usando uma série de valores ou variáveis. A Adobe recomenda que o bloco de código seja sempre colocado entre chaves ({}). Embora seja possível omitir as chaves caso o bloco de código contenha apenas uma instrução, essa prática não é recomendada pelo mesmo motivo que para as condicionais: ela aumenta as chances de que as instruções adicionadas em um momento posterior sejam excluídas inadvertidamente do bloco de código. Se, mais tarde, você adicionar uma instrução que deseja incluir no bloco de código, mas esquecer de adicionar as chaves necessárias, a instrução não será executada como parte da repetição.

for

A repetição `for` permite fazer a iteração por meio de uma variável para um intervalo específico de valores. Você deve fornecer três expressões em uma instrução `for`: uma variável que é definida com um valor inicial, uma instrução condicional que determina quando a repetição termina e uma expressão que altera o valor da variável a cada repetição. Por exemplo, o código a seguir é repetido cinco vezes. O valor da variável `i` começa com 0 e termina com 4, e a saída será os números de 0 a 4, cada um em sua própria linha.

```
var i:int;
for (i = 0; i < 5; i++)
{
    trace(i);
}
```

for..in

A repetição `for..in` itera por meio das propriedades de um objeto ou dos elementos de uma matriz. Por exemplo, você pode usar uma repetição `for..in` para iterar por meio das propriedades de um objeto genérico (as propriedades de objeto não são mantidas em uma ordem específica, por isso elas podem aparecer em uma ordem aparentemente aleatória):

```
var myObj:Object = {x:20, y:30};
for (var i:String in myObj)
{
    trace(i + ": " + myObj[i]);
}
// output:
// x: 20
// y: 30
```

Também é possível iterar por meio dos elementos de uma matriz:

```
var myArray:Array = ["one", "two", "three"];
for (var i:String in myArray)
{
    trace(myArray[i]);
}
// output:
// one
// two
// three
```

O que você não pode fazer é iterar por meio das propriedades de um objeto se ele for uma ocorrência de uma classe definida pelo usuário, a menos que a classe seja dinâmica. Mesmo com ocorrências de classes dinâmicas, será possível iterar somente por meio de propriedades adicionadas dinamicamente.

for each..no

A repetição `for each..in` itera por meio dos itens de um conjunto, que podem ser tags em um objeto XML ou XMLList, os valores mantidos pelas propriedades do objeto ou os elementos de uma matriz. Por exemplo, como mostra o trecho a seguir, você pode usar uma repetição `for each..in` para iterar por meio das propriedades de um objeto genérico, mas diferentemente da repetição `for..in`, a variável do iterador em uma repetição `for each..in` contém o valor mantido pela propriedade em vez do nome da propriedade:

```
var myObj:Object = {x:20, y:30};
for each (var num in myObj)
{
    trace(num);
}
// output:
// 20
// 30
```

Você pode iterar por meio de um objeto XML ou XMLList, como mostra o seguinte exemplo:

```
var myXML:XML = <users>
    <fname>Jane</fname>
    <fname>Susan</fname>
    <fname>John</fname>
</users>;

for each (var item in myXML.fname)
{
    trace(item);
}
/* output
Jane
Susan
John
*/
```

Também é possível iterar por meio dos elementos de uma matriz, como mostra este exemplo:

```
var myArray:Array = ["one", "two", "three"];
for each (var item in myArray)
{
    trace(item);
}
// output:
// one
// two
// three
```

Você não poderá iterar por meio das propriedades de um objeto se o objeto for uma ocorrência de uma classe selada. Mesmo para ocorrências de classes dinâmicas, não é possível iterar por meio de uma propriedade fixa, que é a propriedade especificada como parte da definição de classe.

while

A repetição `while` é como uma instrução `if` que é repetida desde que a condição seja `true`. Por exemplo, o código a seguir produz a mesma saída que o exemplo da repetição `for`:

```
var i:int = 0;
while (i < 5)
{
    trace(i);
    i++;
}
```

Uma desvantagem do uso de uma repetição `while` em vez de `for` é que repetições infinitas são mais fáceis de escrever com repetições `while`. O exemplo da repetição `for` não será compilado se você omitir a expressão que incrementa a variável do contador, mas o exemplo da repetição `while` será compilado se essa etapa for omitida. Sem a expressão que incrementa `i`, a repetição se torna infinita.

do..while

A repetição `do..while` é uma repetição `while` que garante que o bloco de código seja executado pelo menos uma vez, porque a condição é verificada depois que o bloco é executado. O código a seguir mostra um exemplo simples de uma repetição `do..while` que gera saída mesmo que a condição não seja atendida:

```
var i:int = 5;
do
{
    trace(i);
    i++;
} while (i < 5);
// output: 5
```

Funções

As *funções* são blocos de código que executam tarefas específicas e podem ser reutilizados no seu programa. Há dois tipos de funções no ActionScript 3.0: *métodos* e *fechamentos de função*. O fato de uma função ser uma chamada a um método ou um fechamento de função depende do contexto na qual ela é definida. Uma função é chamada de método quando é especificada como parte de uma definição de classe ou anexada a uma ocorrência de um objeto. Uma função é chamada de fechamento de função quando é definida de qualquer outra forma.

As funções sempre foram extremamente importantes no ActionScript. No ActionScript 1.0, por exemplo, a palavra-chave `class` não existir, por isso as "classes" eram definidas pelas funções do construtor. Embora a palavra-chave `class` tenha sido adicionada à linguagem, ainda é importante ter um bom entendimento das funções para aproveitar todas as vantagens que ela tem a oferecer. Isso pode ser um desafio para os programadores que esperam que as funções do ActionScript se comportem da mesma forma que as funções em linguagens como C++ ou Java. Embora a definição e a chamada de funções básicas não apresentem um desafio aos programadores experientes, alguns recursos mais avançados das funções do ActionScript exigem uma explicação.

Conceitos de funções básicas

Esta seção discute as técnicas de definição e chamada de funções básicas.

Chamada de funções

Uma função é chamada usando seu identificador seguido de um operador parênteses (`()`). O operador parênteses delimita qualquer parâmetro de função que você deseja enviar para a função. Por exemplo, a função `trace()`, que é uma função de nível superior no ActionScript 3.0, é usada em todo este manual:

```
trace("Use trace to help debug your script");
```

Se estiver chamando uma função sem nenhum parâmetro, você deverá usar parênteses vazios. Por exemplo, você pode usar o método `Math.random()`, que não usa nenhum parâmetro, para gerar um número aleatório:

```
var randomNum:Number = Math.random();
```

Definição de suas próprias funções

Há duas maneiras de definir uma função no ActionScript 3.0: você pode usar uma instrução de função ou uma expressão de função. A escolha da técnica depende de sua preferência por um estilo de programação mais estático ou dinâmico. Defina suas funções com instruções de função se preferir uma programação de modo estático ou estrito. Defina as funções com expressões de função se tiver uma necessidade específica para isso. As expressões de função são usadas com mais freqüência em programação de modo dinâmico ou padrão.

Instruções de função

A instrução de função é a técnica preferida para definir funções no modo estrito. Uma instrução de função começa com a palavra-chave `function` e, em seguida, vem:

- O nome da função
- Os parâmetros, em uma lista delimitada por vírgula e entre parênteses
- O corpo da função, ou seja, o código do ActionScript a ser executado quando a função é chamada, delimitado por chaves

  Por exemplo, o código a seguir cria uma função que define um parâmetro e chama a função usando a seqüência de caracteres `"hello"` como valor do parâmetro:

```
function traceParameter(aParam:String)
{
    trace(aParam);
}

traceParameter("hello"); // hello
```

Expressões de função

A segunda forma de declarar uma função é usar uma instrução de atribuição com uma expressão de função, que às vezes também é chamada de literal de função ou função anônima. Este método é mais detalhado e amplamente usado nas versões anteriores do ActionScript.

Uma instrução de atribuição com uma expressão de função começa com a palavra-chave `var` e, em seguida, vem:

- O nome da função
- O operador dois-pontos (`:`)
- A classe `Function` para indicar o tipo de dados
- O operador de atribuição (`=`)
- A palavra-chave `function`
- Os parâmetros, em uma lista delimitada por vírgula e entre parênteses
- O corpo da função, ou seja, o código do ActionScript a ser executado quando a função é chamada, delimitado por chaves

  Por exemplo, o seguinte código declara a função `traceParameter` usando uma expressão de função:

```
var traceParameter:Function = function (aParam:String)
{
    trace(aParam);
};
traceParameter("hello"); // hello
```

Observe que um nome de função não é especificado da mesma forma que em uma instrução de função. Outra diferença importante entre a expressão de função e a instrução de função é que a primeira é uma expressão e não uma instrução. isso significa que uma expressão de função não é suficiente como uma instrução de função. Ela só pode ser usada como parte de uma instrução, em geral, de atribuição. O exemplo a seguir mostra uma expressão de função atribuída a um elemento de matriz:

```
var traceArray:Array = new Array();
traceArray[0] = function (aParam:String)
{
    trace(aParam);
};
traceArray[0]("hello");
```

Escolha entre instruções e expressões

Como regra geral, use uma instrução de função a menos que circunstâncias específicas exijam o uso de uma expressão. As instruções de função são menos detalhadas e fornecem uma experiência mais consistente entre o modo estrito e o padrão do que as expressões de função.

As instruções de função são mais fáceis de ler do que as instruções de atribuição que contêm expressões de função. As instruções de função tornam o código mais conciso; elas são menos confusas do que as expressões de função, que requerem o uso das palavras-chave `var` e `function`.

As instruções de função fornecem uma experiência mais consistente entre os dois modos de compilação já que é possível usar a sintaxe de pontos nos modos estrito e padrão para chamar um método declarado usando uma instrução de função. Isso não é necessariamente verdadeiro para métodos declarados com uma expressão de função. Por exemplo, o código a seguir define uma classe chamada Example com dois métodos: `methodExpression()`, que é declarado com uma expressão de função, e `methodStatement()`, que é declarado com uma instrução de função. No modo estrito, não é possível usar a sintaxe de pontos para chamar o método `methodExpression()`.

```
class Example
{
var methodExpression = function() {}
function methodStatement() {}
}

var myEx:Example = new Example();
myEx.methodExpression(); // error in strict mode; okay in standard mode
myEx.methodStatement(); // okay in strict and standard modes
```

As expressões de função são consideradas mais adequadas para a programação direcionada ao comportamento de tempo de execução ou dinâmico. Se preferir usar o modo estrito, mas também precisar chamar um método declarado com uma expressão de função, você poderá usar qualquer uma destas duas técnicas. Primeiro, você pode chamar o método usando colchetes (`[]`) em vez do operador ponto (`.`). A chamada de método a seguir é bem-sucedida nos modos estrito e padrão:

```
myExample["methodLiteral"]();
```

Em segundo lugar, você pode declarar a classe inteira como uma classe dinâmica. Embora isso permita chamar o método usando o operador dot, o lado negativo é que você sacrifica algumas funcionalidades do modo estrito para todas as ocorrências dessa classe. Por exemplo, o compilador não irá gerar um erro se você tentar acessar uma propriedade indefinida em uma ocorrência de uma classe dinâmica.

Há algumas circunstâncias nas quais as expressões de função são úteis. Um uso comum de expressões de função é para funções que são usadas uma única vez e, depois, são descartadas. Outro uso menos comum é para anexar uma função a uma propriedade de protótipo. Para obter mais informações, consulte “Objeto de protótipo” na página 123.

Há duas diferenças sutis entre as instruções de função e as expressões de função que devem ser levadas em conta ao escolher a técnica usada. A primeira diferença é que as expressões de função não existem independentemente como objetos em relação ao gerenciamento de memória e à coleta de lixo. Em outras palavras, durante a atribuição de uma expressão de função a outro objeto, como um elemento de matriz ou uma propriedade de objeto, você cria a única referência a essa expressão de função no código. Se a matriz ou o objeto ao qual a expressão de função é anexada sair do escopo ou não estiver disponível, não será mais possível acessar a expressão de função. Se a matriz ou o objeto forem excluídos, a memória usada pela expressão de função se tornará qualificada para a coleta de lixo, o que significa que a memória é qualificada para ser reivindicada e reutilizada para outros fins.

O exemplo a seguir mostra que, para uma expressão de função, assim que a propriedade com a expressão atribuída for excluída, a função não ficará mais disponível. A classe Test é dinâmica, o que significa que é possível adicionar uma propriedade chamada `functionExp` que mantém uma expressão de função. A função `functionExp()` pode ser chamada com o operador dot, mas assim que a propriedade `functionExp` for excluída, a função ficará inacessível.

```
dynamic class Test {}
var myTest:Test = new Test();

// function expression
myTest.functionExp = function () { trace("Function expression") };
myTest.functionExp(); // Function expression
delete myTest.functionExp;
myTest.functionExp(); // error
```

Se, no entanto, for definida com uma instrução de função, a função existirá como seu próprio objeto e continuará a existir mesmo depois que a propriedade à qual está anexada for excluída. O operador `delete` funciona apenas em propriedades de objetos, por isso até mesmo uma chamada para excluir a função `stateFunc()` em si não funciona.

```
dynamic class Test {}
var myTest:Test = new Test();

// function statement
function stateFunc() { trace("Function statement") }
myTest.statement = stateFunc;
myTest.statement(); // Function statement
delete myTest.statement;
delete stateFunc; // no effect
stateFunc();// Function statement
myTest.statement(); // error
```

A segunda diferença entre as instruções de função e as expressões de função é que as primeiras existem em todo o escopo no qual são definidas, incluindo em instruções que aparecem antes da instrução de função. As expressões de função, porém, são definidas somente para instruções subseqüentes. Por exemplo, o seguinte código chama com êxito a função `scopeTest()` antes que ela seja definida:

```
statementTest(); // statementTest

function statementTest():void
{
    trace("statementTest");
}
```

As expressões de função não estão disponíveis antes de serem definidas, por isso o seguinte código resulta em um erro de tempo de execução:

```
expressionTest(); // run-time error

var expressionTest:Function = function ()
{
    trace("expressionTest");
}
```

Retorno de valores de funções

Para retornar um valor de sua função, use a instrução `return` seguida pela expressão ou pelo valor literal que deseja retornar. Por exemplo, o seguinte código retorna uma expressão representando o parâmetro:

```
function doubleNum(baseNum:int):int
{
    return (baseNum * 2);
}
```

Observe que a instrução `return` encerra a função, de forma que as instruções abaixo de uma instrução `return` não são executadas, como a seguir:

```
function doubleNum(baseNum:int):int {
    return (baseNum * 2);
    trace("after return"); // This trace statement will not be executed.
}
```

No modo estrito, você deve retornar um valor do tipo apropriado se decidir especificar um tipo de retorno. Por exemplo, o código a seguir gera um erro no modo estrito, porque não retorna um valor válido:

```
function doubleNum(baseNum:int):int
{
    trace("after return");
}
```

Funções aninhadas

É possível aninhar funções, o que significa que as funções podem ser declaradas dentro de outras. Uma função aninhada está disponível apenas dentro da função pai, a menos que uma referência a ela seja transmitida ao código externo. Por exemplo, o seguinte código declara duas funções aninhadas dentro da função `getNameAndVersion()`:

```
function getNameAndVersion():String
{
    function getVersion():String
    {
        return "10";
    }
    function getProductName():String
    {
        return "Flash Player";
    }
    return (getProductName() + " " + getVersion());
}
trace(getNameAndVersion()); // Flash Player 10
```

Quando funções aninhadas são transmitidas ao código externo, elas são transmitidas como fechamentos de função, o que significa que a função retém as definições que estão no escopo quando a função é definida. Para obter mais informações, consulte “Escopo da função” na página 90.

Parâmetros de função

O ActionScript 3.0 fornece algumas funcionalidades para parâmetros de função que podem parecer novas aos programadores que não conhecem a linguagem. Embora a idéia de transmitir parâmetros por valor ou referência não deva ser nova para a maioria dos programadores, o objeto `arguments` e o parâmetro ... (rest) talvez sejam uma novidade para muitos.

Transmissão de argumentos por valor ou referência

Em muitas linguagens de programação, é importante entender a distinção entre transmitir argumentos por valor ou por referência, pois ela pode afetar a forma como o código é criado.

Ser transmitido por valor significa que o valor do argumento é copiado em uma variável local para uso dentro da função. Ser transmitido por referência significa que apenas uma referência ao argumento é transmitido, em vez do valor real. Não é feita nenhuma cópia do argumento real. Em vez disso, uma referência à variável transmitida como um argumento é criada e atribuída a uma variável local para uso dentro da função. Como uma referência a uma variável fora da função, a variável local fornece a capacidade de alterar o valor da variável original.

No ActionScript 3.0, todos os argumentos são transmitidos por referência, porque todos os valores são armazenados como objetos. Entretanto, os objetos que pertencem aos tipos de dados primitivos, que incluem Boolean, Number, int, uint e String, têm operadores especiais que fazem com que se comportem como se fossem transmitidos por valor. Por exemplo, o seguinte código cria uma função chamada `passPrimitives()` que define dois parâmetros chamados `xParam` e `yParam`, ambos do tipo int. Esses parâmetros são semelhantes às variáveis locais declaradas no corpo da função `passPrimitives()`. Quando a função é chamada com os argumentos `xValue` e `yValue`, os parâmetros `xParam` e `yParam` são inicializados com referências aos objetos int representados por `xValue` e `yValue`. Como são primitivos, os argumentos se comportam como se fossem transmitidos por valor. Embora `xParam` e `yParam` inicialmente contenham apenas referências aos objetos `xValue` e `yValue`, quaisquer alterações às variáveis dentro do corpo da função gera novas cópias dos valores na memória.

```
function passPrimitives(xParam:int, yParam:int):void
{
    xParam++;
    yParam++;
    trace(xParam, yParam);
}

var xValue:int = 10;
var yValue:int = 15;
trace(xValue, yValue);// 10 15
passPrimitives(xValue, yValue); // 11 16
trace(xValue, yValue);// 10 15
```

Dentro da função `passPrimitives()`, os valores de `xParam` e `yParam` são incrementados, mas isso não afeta os valores de `xValue` e `yValue`, como mostra a última instrução `trace`. Isso seria válido mesmo que os parâmetros fossem nomeados de forma idêntica às variáveis, `xValue` e `yValue`, porque `xValue` e `yValue` dentro da função apontariam para novos locais na memória que existem separadamente das variáveis de mesmo nome fora da função.

Todos os outros objetos, ou seja, objetos que não pertencem aos tipos de dados primitivos, são transmitidos por referência, o que fornece a capacidade de alterar o valor das variáveis originais. Por exemplo, o código a seguir cria um objeto chamado `objVar` com duas propriedades, `x` e `y`. O objeto é transmitido como um argumento para a função `passByRef()`. Como não é de um tipo primitivo, o objeto não é apenas transmitido por referência, mas também permanece como uma. Isso significa que as alterações feitas aos parâmetros dentro da função afetarão as propriedades do objeto fora da função.

```
function passByRef(objParam:Object):void
{
    objParam.x++;
    objParam.y++;
    trace(objParam.x, objParam.y);
}
var objVar:Object = {x:10, y:15};
trace(objVar.x, objVar.y); // 10 15
passByRef(objVar); // 11 16
trace(objVar.x, objVar.y); // 11 16
```

O parâmetro `objParam` referencia o mesmo objeto que a variável global `objVar`. Como você pode ver nas instruções `trace` do exemplo, as alterações nas propriedades `x` e `y` do objeto `objParam` são refletidas no objeto `objVar`.

Valores de parâmetro padrão

Nova no ActionScript 3.0 é a capacidade de declarar *valores de parâmetro padrão* para uma função. Se uma chamada a uma função com valores de parâmetro padrão omitir um parâmetro com valores padrão, será usado o valor especificado na definição de função para esse parâmetro. Todos os parâmetros com valores padrão devem ser colocados no final da lista de parâmetros. Os valores atribuídos como padrão devem ser constantes de tempo de compilação. A existência de um valor padrão para um parâmetro efetivamente o torna um *parâmetro opcional*. Um parâmetro sem um valor padrão é considerado um *parâmetro necessário*.

Por exemplo, o seguinte código cria uma função com três parâmetros, dois dos quais têm valores padrão. Quando a função é chamada com apenas um parâmetro, os valores padrão para os parâmetros são usados.

```
function defaultValues(x:int, y:int = 3, z:int = 5):void
{
    trace(x, y, z);
}
defaultValues(1); // 1 3 5
```

O objeto arguments

Quando parâmetros são transmitidos a uma função, é possível usar o objeto `arguments` para acessar informações sobre esses parâmetros. Alguns aspectos importantes do objeto `arguments` incluem o seguinte:

- O objeto `arguments` é uma matriz que inclui todos os parâmetros transmitidos à função.
- A propriedade `arguments.length` relata o número de argumentos transmitidos à função.
- A propriedade `arguments.callee` fornece uma referência à função em si, que é útil para chamadas recursivas às expressões de função.

  Nota: *O objeto `arguments` não estará disponível se qualquer parâmetro for chamado de `arguments` ou se for usado o parâmetro ... (rest).*

  Se o objeto de `argumentos` estiver referenciado no corpo de uma função, o ActionScript 3.0 permitirá que as chamadas de função incluam mais parâmetros do que os especificados na definição de função, mas irá gerar um erro no compilador no modo restrito, se o número de parâmetros não corresponder ao número de parâmetros necessários (e, opcionalmente, quaisquer parâmetros opcionais). É possível usar o aspecto de matriz do objeto `arguments` para acessar qualquer parâmetro transmitido à função, independentemente de ele ser especificado na definição de função. O exemplo a seguir, que é compilado apenas no modo padrão, usa a matriz `arguments` com a propriedade `arguments.length` para rastrear todos os parâmetros transmitidos para a função `traceArgArray()`:

```
function traceArgArray(x:int):void
{
    for (var i:uint = 0; i < arguments.length; i++)
    {
        trace(arguments[i]);
    }
}

traceArgArray(1, 2, 3);

// output:
// 1
// 2
// 3
```

A propriedade `arguments.callee`, em geral, é usada em funções anônimas para criar recursão. Você pode usá-la para adicionar flexibilidade ao seu código. Se o nome de uma função recursiva for alterado durante o ciclo de desenvolvimento, não será necessário se preocupar em alterar a chamada recursiva no corpo da função se usar `arguments.callee` em vez do nome de função. A propriedade `arguments.callee` é usada na seguinte expressão de função para permitir recursão:

```
var factorial:Function = function (x:uint)
{
    if(x == 0)
    {
        return 1;
    }
    else
    {
        return (x * arguments.callee(x - 1));
    }
}

trace(factorial(5)); // 120
```

Caso você use o parâmetro ... (rest) na declaração de função, o objeto `arguments` não ficará disponível. Em vez disso, você terá de acessar os parâmetros usando os nomes de parâmetro declarados para eles.

Você também deve ter o cuidado de evitar o uso da seqüência de caracteres `"arguments"` como parâmetro, porque ela irá obscurecer o objeto `arguments`. Por exemplo, se a função `traceArgArray()` for reescrita de forma que um parâmetro `arguments` seja adicionado, as referências a `arguments` no corpo da função irão se referir ao parâmetro e não ao objeto `arguments`. O seguinte código não produz nenhuma saída:

```
function traceArgArray(x:int, arguments:int):void
{
    for (var i:uint = 0; i < arguments.length; i++)
    {
        trace(arguments[i]);
    }
}

traceArgArray(1, 2, 3);

// no output
```

O objeto `arguments` nas versões anteriores do ActionScript também continham uma propriedade chamada `caller`, que é uma referência á função que chamava a função atual. A propriedade `caller` não está presente no ActionScript 3.0, mas, se uma referência à função de chamada for necessária, você poderá alterar a função de chamada para que ela transmita um parâmetro extra que faça referência a si mesmo.

O parâmetro ... (rest)

O ActionScript 3.0 apresenta uma nova declaração de parâmetro chamada de parâmetro ... (rest). Esse parâmetro permite especificar um parâmetro de matriz que aceita uma grande quantidade de argumentos delimitados por vírgula. O parâmetro pode ter qualquer nome que não seja uma palavra reservada. Essa declaração de parâmetro deve ser o último parâmetro especificado. O uso desse parâmetro torna o objeto `arguments` indisponível. Embora o parâmetro ... (rest) ofereça a mesma funcionalidade que a matriz `arguments` e a propriedade `arguments.length`, ele não fornece uma funcionalidade semelhante à fornecida por `arguments.callee`. Você deve se certificar de que não será preciso usar `arguments.callee` antes de usar o parâmetro ... (rest).

O exemplo a seguir reescreve a função `traceArgArray()` usando o parâmetro ... (rest) em vez do objeto `arguments`:

```
function traceArgArray(... args):void
{
    for (var i:uint = 0; i < args.length; i++)
    {
        trace(args[i]);
    }
}

traceArgArray(1, 2, 3);

// output:
// 1
// 2
// 3
```

O parâmetro ... (rest) também pode ser usado com outros parâmetros, contanto que o último parâmetro seja listado. O exemplo a seguir modifica a função `traceArgArray()` para que seu primeiro parâmetro, `x`, seja do tipo int e o segundo use o parâmetro ... (rest). A saída ignora o primeiro valor, porque o primeiro parâmetro não faz mais parte da matriz criada pelo parâmetro ... (rest).

```
function traceArgArray(x: int, ... args)
{
    for (var i:uint = 0; i < args.length; i++)
    {
        trace(args[i]);
    }
}

traceArgArray(1, 2, 3);

// output:
// 2
// 3
```

Funções como objetos

As funções no ActionScript 3.0 são objetos. Durante a criação de uma função, você cria um objeto que não apenas pode ser transmitido como um parâmetro para outra função, mas também possui propriedades e métodos anexados.

As funções transmitidas como argumentos para outra função são transmitidas por referência e não por valor. Durante a transmissão de uma função como um argumento, é usado apenas o identificador e não o operador parênteses usado para chamar o método. Por exemplo, o código a seguir transmite uma função chamada `clickListener()` como um argumento para o método `addEventListener()`:

```
addEventListener(MouseEvent.CLICK, clickListener);
```

O método `Array.sort()` também define um parâmetro que aceita uma função. Para obter um exemplo de uma função de classificação personalizada que é usada como um argumento para a função `Array.sort()`, consulte "Classificação de uma matriz" na página 165.

Embora possa parecer estranho aos programadores que não conhecem o ActionScript, as funções podem ter propriedades e métodos, assim como qualquer outro objeto. Na verdade, cada função possui uma propriedade somente leitura chamada `length` que armazena o número de parâmetros definido para a função. Isso é diferente na propriedade `arguments.length`, que relata o número de argumentos enviados à função. Lembre-se de que, no ActionScript, o número de argumentos enviados a uma função pode exceder o número de parâmetros definido para ela. O exemplo a seguir, que é compilado somente no modo padrão porque o modo estrito requer uma correspondência exata entre o número de argumentos transmitidos e o número de parâmetros definido, mostra a diferença entre as duas propriedades:

```
// Compiles only in standard mode
function traceLength(x:uint, y:uint):void
{
    trace("arguments received: " + arguments.length);
    trace("arguments expected: " + traceLength.length);
}

traceLength(3, 5, 7, 11);
/* output:
arguments received: 4
arguments expected: 2 */
```

No modo padrão, é possível especificar suas próprias propriedades de função definindo-as fora do corpo da função. As propriedades de função podem servir como propriedades quase estáticas que permitem salvar o estado de uma variável relacionada à função. Por exemplo, você pode querer controlar o número de vezes que uma função específica é chamada. Essa funcionalidade pode ser útil quando você escreve um jogo e deseja controlar o número de vezes que um usuário usa um comando especifico, embora também seja possível usar uma propriedade de classe estática para isso. O exemplo a seguir, que é compilado somente no modo padrão porque o modo estrito não permite adicionar propriedades dinâmicas às funções, cria uma propriedade de função fora da declaração de função e incrementa a propriedade sempre que a função é chamada:

```
// Compiles only in standard mode
var someFunction:Function = function ():void
{
    someFunction.counter++;
}

someFunction.counter = 0;

someFunction();
someFunction();
trace(someFunction.counter); // 2
```

Escopo da função

Um escopo de função determina não apenas o local em um programa no qual a função pode ser chamada, mas também as definições que ela pode acessar. As mesmas regras de escopo válidas para os identificadores de variável se aplicam aos identificadores de função. Uma função declarada no escopo global está disponível em todo o código. Por exemplo, o ActionScript 3.0 contém funções globais, tais como `isNaN()` e `parseInt()`, que estão disponíveis em qualquer lugar no seu código. Uma função aninhada (uma função declarada dentro de outra função) pode ser usada em qualquer lugar na função na qual ela foi declarada.

A cadeia do escopo

Sempre que uma função começa com a execução, vários objetos e propriedades são criados. Primeiro, é criado um objeto especial chamado *objeto de ativação* que armazena os parâmetros e quaisquer variáveis locais ou funções declaradas no corpo da função. Não é possível acessar o objeto de ativação diretamente, porque ele é um mecanismo interno. Em segundo lugar, é criada uma *cadeia do escopo* que contém uma lista ordenada de objetos na qual o Flash Player ou o Adobe AIR verifica as declarações de identificador. Cada função executada tem uma cadeia de escopo que é armazenada em uma propriedade interna. Para uma função aninhada, a cadeia do escopo começa com seu próprio objeto de ativação, seguido pelo objeto de ativação de sua função pai. A cadeia continua assim até atingir o objeto global. O objeto global é criado quando um programa do ActionScript começa e contém todas as variáveis e funções globais.

Fechamentos de função

Um *fechamento de função* é um objeto que contém um instantâneo de uma função e seu *ambiente léxico*. O ambiente léxico de uma função inclui todas as variáveis, propriedades, métodos e objetos na cadeia do escopo da função, além de seus valores. Os fechamentos de função são criados sempre que uma função é executada independentemente de um objeto ou uma classe. O fato de que os fechamentos de função mantêm o escopo no qual eles foram definidos cria resultados interessantes quando uma função é transmitida como um argumento ou um valor de retorno em um escopo diferente.

Por exemplo, o código a seguir cria duas funções: `foo()`, que retorna uma função aninhada chamada `rectArea()` que calcula a área de um retângulo, e `bar()`, que chama `foo()` e armazena o fechamento de função retornado em uma variável chamada `myProduct`. Muito embora a função `bar()` defina sua própria variável local `x` (com um valor 2), quando o fechamento de função `myProduct()` é chamado, ela mantém a variável `x` (com um valor 40) definida na função `foo()`. A função `bar()`, portanto, retorna o valor `160` em vez de `8`.

```
function foo():Function
{
    var x:int = 40;
    function rectArea(y:int):int // function closure defined
    {
        return x * y
    }
    return rectArea;
}
function bar():void
{
    var x:int = 2;
    var y:int = 4;
    var myProduct:Function = foo();
    trace(myProduct(4)); // function closure called
}
bar(); // 160
```

Os métodos se comportam de forma semelhante pois também mantêm as informações sobre o ambiente léxico no qual são criados. Essa característica é a mais notável quando um método é extraído de sua ocorrência, que cria um método vinculado. A principal diferença entre um fechamento de função e um método vinculado é que o valor da palavra-chave `this` em um método vinculado sempre se refere à ocorrência à qual ele foi inicialmente anexado, enquanto que, em um fechamento de função, o valor da palavra-chave `this` pode ser alterado. Para obter mais informações, consulte "Métodos" na página 100.

Capítulo 5: Programação orientada a objetos no ActionScript

Este capítulo descreve os elementos do ActionScript que oferecem suporte à OOP (Programação orientada a objetos). O capítulo não descreve princípios de OOP gerais, como design de objeto, abstração, encapsulamento, herança e polimorfismo. Ele se concentra em como aplicar esses princípios usando o ActionScript 3.0.

Devido a raízes do ActionScript como uma linguagem de script, o suporte ao OOP do ActionScript 3.0 é opcional. Isso fornece aos programadores flexibilidade para escolherem a melhor abordagem para projetos de vários escopos e complexidades. Para tarefas pequenas, você pode chegar à conclusão de que usar o ActionScript com um paradigma de programação de procedimento é tudo o que você precisa. Para projetos maiores, aplicar princípios de OOP pode fazer com que o código fique mais fácil de ser compreendido, mantido e estendido.

Noções básicas de programação orientada a objetos

Introdução à programação orientada a objetos

OOP (Programação orientada a objetos) é uma maneira de organizar o código em um programa agrupando-o em objetos, elementos individuais que incluem informações (valores de dados) e funcionalidade. O uso de uma abordagem orientada a objetos para organizar um programa permite agrupar partes específicas de informações (por exemplo, informações sobre música, como título do álbum, título da faixa ou nome do artista) juntamente com a funcionalidade comum ou com as ações associadas a essas informações (como "adicionar faixa à lista de reprodução" ou "reproduzir todas as canções desse artista"). Esses itens são combinados em único item, um objeto (por exemplo, um "Album" ou "MusicTrack"). Vários benefícios são obtidos quando se pode agrupar esses valores e funções em conjunto, incluindo apenas a necessidade de manter o rastreio de uma única variável, em vez de várias, organizar a funcionalidade relacionada em conjunto, e poder estruturar programas de maneiras muito correspondentes ao mundo real.

Tarefas de programação orientadas a objetos comuns

Na prática, a programação orientada a objetos tem duas partes. Uma parte são as estratégias e técnicas para criar um programa (freqüentemente chamada de *design orientado a objetos*). Esse é um assunto abrangente e não é discutido neste capítulo. A outra parte da OOP são as estruturas reais de programação que estão disponíveis em uma determinada linguagem de programação para criar um programa usando uma abordagem orientada a objetos. Este capítulo abrange as seguintes tarefas comuns na OOP:

- Definição de classes
- Criação de propriedades, métodos e obtenção e definição de acessor (métodos de acessor)
- Controle do acesso a classes, propriedades, métodos e acessores
- Criação de propriedades e métodos estáticos
- Criação de estruturas semelhantes a enumeração
- Definição e uso de interfaces
- Trabalho com herança, incluindo substituição de elementos de classes

Conceitos e termos importantes

A lista de referência a seguir contém termos importantes usados neste capítulo:

- Atributo: Uma característica atribuída a um elemento de classe (como uma propriedade ou método) na definição de classe. Os atributos são usados normalmente para definir se a propriedade ou método estará disponível para acesso por código em outras partes do programa. Por exemplo, `private` e `public` são atributos. Um método particular pode ser chamado apenas pelo código dentro da classe, enquanto um método público pode ser chamado por qualquer código no programa.
- Classe: A definição da estrutura e o comportamento de objetos de um determinado tipo (como um modelo ou plano gráfico de objetos desse tipo de dados).
- Hierarquia de classes: A estrutura de várias classes relacionadas, especificando a classes que herdam funcionalidade de outras classes.
- Construtor: Um método especial que você pode definir em uma classe, que é chamado quando uma ocorrência da classe é criada. Um construtor é usado normalmente para especificar valores padrão ou, de outra forma, executar operações de configuração do objeto.
- Tipo de dados: O tipo de informações que uma variável específica pode armazenar. Em geral, *tipo de dados* tem o mesmo significado que *classe*.
- Operador ponto: O sinal de ponto (.), que no ActionScript (e em muitas outras linguagens de programação) é usado para indicar que um nome faz referência a um elemento filho de um objeto (como uma propriedade ou método). Por exemplo, na expressão `myObject.myProperty`, o operador ponto indica que o termo `myProperty` está fazendo referência a algum valor que é um elemento do objeto denominado `myObject`.
- Enumeração: Um conjunto de valores constantes relacionados, agrupados em conjunto por conveniência como propriedades de uma única classe.
- Herança: O mecanismo da OOP que permite que uma definição de classe inclua toda a funcionalidade de uma definição de classe diferente (e geralmente seja adicionada a essa funcionalidade).
- Ocorrência: Um objeto real criado em um programa.
- Espaço para nomes: Essencialmente um atributo personalizado que permite controle mais refinado sobre qual código pode acessar outro código.

Teste dos exemplos do capítulo

Talvez você queira testar algumas das listagens de código de exemplo por si próprio, durante a leitura deste capítulo. Como as listagens de código neste capítulo tratam principalmente com a definição e a manipulação de tipos de dados, testar os exemplos envolve a criação de uma ocorrência da classe que está sendo definida, a manipulação dessa ocorrência usando suas propriedades ou métodos e a exibição dos valores das propriedades dessa ocorrência. Para exibir esses valores, grave-os em uma ocorrência do campo de texto no Palco ou use a função `trace()` para imprimir valores no painel Saída. Essas técnicas são descritas em detalhes em "Teste de listagens de código de exemplo dos capítulos" na página 36.

Classes

Uma classe é uma representação abstrata de um objeto. Uma classe armazena informações sobre os tipos de dados que um objeto pode manter e os comportamentos que um objeto pode exibir. A utilidade dessa abstração pode não ser aparente quando você grava pequenos scripts que contêm apenas alguns objetos que interagem entre si. No entanto, conforme o escopo de um programa aumenta, assim como o número de objetos que precisam ser gerenciados, você pode descobrir que as classes permitem um controle melhor de como os objetos são criados e como eles interagem entre si.

Já no ActionScript 1.0, os programadores do ActionScript podiam usar objetos Function para criar construções semelhantes a classes. O ActionScript 2.0 adicionou suporte formal para classes com palavras-chave, como `class` e `extends`. O ActionScript 3.0 não apenas continua a oferecer suporte às palavras-chave introduzidas no ActionScript 2.0, mas também adiciona alguns novos recursos, como controle de acesso aprimorado com os atributos `protected` e `internal`, e melhor controle sobre a herança com as palavras-chave `final` e `override`.

Se você já tiver criado classes em linguagens de programação, como Java, C++ ou C#, descobrirá que o ActionScript fornece uma experiência familiar. O ActionScript compartilha muitas das mesmas palavras-chave e nomes de atributos, como `class`, `extends` e `public`, que são discutidos nas seções a seguir.

Nota: *Neste capítulo, o termo propriedade significa qualquer membro de um objeto ou classe, incluindo variáveis, constantes e métodos. Além disso, embora os termos classe e estática sejam sempre usados alternadamente, neste capítulo esses termos são distintos. Por exemplo, neste capítulo a expressão propriedades de classes fazem referência a todos os membros de uma classe, em vez de apenas a membros estáticos.*

Definições de classes

As definições de classes do ActionScript 3.0 usam sintaxe semelhante a usada no ActionScript 2.0. A sintaxe apropriada para uma definição de classe requer a palavra-chave `class` seguida pelo nome da classe. O corpo da classe que está entre chaves ({}) segue o nome da classe. Por exemplo, o código a seguir cria uma classe denominada Shape que contém uma variável denominada `visible`:

```
public class Shape
{
    var visible:Boolean = true;
}
```

Uma alteração significativa na sintaxe envolve definições de classes que estão dentro de um pacote. No ActionScript 2.0, se uma classe estiver dentro de um pacote, o nome do pacote deverá estar incluído na declaração da classe. No ActionScript 3.0, que introduz a instrução `package`, o nome do pacote deve estar incluído na declaração do pacote em vez de na declaração da classe. Por exemplo, as seguintes declarações de classe mostram como a classe BitmapData, que faz parte do pacote flash.display, é definida no ActionScript 2.0 e no ActionScript 3.0:

```
// ActionScript 2.0
class flash.display.BitmapData {}

// ActionScript 3.0
package flash.display
{
    public class BitmapData {}
}
```

Atributos de classes

O ActionScript 3.0 permite modificar definições de classe usando um dos quatro seguintes atributos:

Atributo	Definição
`dinâmico`	Permitir que propriedades sejam adicionadas a ocorrências em tempo de execução.
`final`	Não deve ser estendido por outra classe.
`internal` (padrão)	Visível para referências dentro do pacote atual.
`public`	Visível para referências em todos os lugares.

Para cada um desses atributos, exceto para `internal`, você deve incluir explicitamente o atributo para obter o comportamento associado. Por exemplo, se você não incluir o atributo `dynamic` ao definir uma classe, não poderá adicionar propriedades a uma ocorrência da classe em tempo de execução. Você atribui explicitamente um atributo colocando-o no início da definição de classe, conforme demonstrado no código a seguir:

```
dynamic class Shape {}
```

Observe que a lista não inclui um atributo denominado `abstract`. Isso é porque as classes abstratas não têm suporte no ActionScript 3.0. Observe também que a lista não inclui atributos denominados `private` e `protected`. Esses atributos têm significado apenas dentro de uma definição de classe e não podem ser aplicados às próprias classes. Se você não desejar que uma classe seja publicamente visível fora de um pacote, coloque-a no pacote e marque-a com o atributo `internal`. Como alternativa, você pode omitir os atributos `internal` e `public` e o compilador adicionará automaticamente o atributo `internal` para você. Para que uma classe seja visível fora do arquivo de origem no qual ela está definida, coloque a classe na parte inferior do arquivo de origem, abaixo das chaves de fechamento de definição do pacote.

Corpo da classe

O corpo da classe, que é delimitado por chaves, é usado para definir as variáveis, constantes e métodos da classe. O exemplo a seguir mostra a declaração da classe Accessibility na API do Adobe Flash Player:

```
public final class Accessibility
{
    public static function get active():Boolean;
    public static function updateProperties():void;
}
```

Você também pode definir um espaço para nomes dentro de um corpo de classe. O exemplo a seguir mostra como um espaço para nomes pode ser definido dentro de um corpo de classe e usado como atributo de um método naquela classe:

```
public class SampleClass
{
    public namespace sampleNamespace;
    sampleNamespace function doSomething():void;
}
```

O ActionScript 3.0 permite incluir não apenas definições em um corpo da classe, mas também instruções. Instruções que estão dentro de um corpo de classe, mas fora de uma definição do método, são executadas exatamente uma vez, quando a definição da classe é encontrada primeiro e o objeto de classe associado é criado. O exemplo a seguir inclui uma chamada para uma função externa, `hello()`, e uma instrução `trace` que produz uma mensagem de confirmação quando a classe é definida:

```
function hello():String
{
    trace("hola");
}
class SampleClass
{
    hello();
    trace("class created");
}
// output when class is created
hola
class created
```

Em contraste com versões anteriores do ActionScript, no ActionScript 3.0 é permitido definir uma propriedade estática e uma propriedade de ocorrência com o mesmo nome no mesmo corpo da classe. Por exemplo, o código a seguir declara uma variável estática denominada `message` e uma variável da ocorrência do mesmo nome:

```
class StaticTest
{
    static var message:String = "static variable";
    var message:String = "instance variable";
}
// In your script
var myST:StaticTest = new StaticTest();
trace(StaticTest.message); // output: static variable
trace(myST.message); // output: instance variable
```

Atributos de propriedade de classes

Em discussões do modelo de objeto do ActionScript, o termo *property* significa qualquer coisa que possa ser um membro de uma classe, incluindo variáveis, constantes e métodos. Isso é diferente da maneira como o termo é usado na Referência dos componentes e da linguagem do ActionScript 3.0, em que o termo é usado de maneira mais limitada e inclui apenas membros de classes que são variáveis ou são definidos por um método getter ou setter. No ActionScript 3.0, há um conjunto de atributos que pode ser usado com qualquer propriedade de uma classe. A tabela a seguir lista esse conjunto de atributos.

Atributo	**Definição**
`internal` (padrão)	Visível para referências dentro do mesmo pacote.
`private`	Visível para referências na mesma classe.
`protected`	Visível para referências na mesma classe e em classes derivadas.
`public`	Visível para referências em todos os lugares.
`static`	Especifica que uma propriedade pertence à classe, ao contrário das ocorrências da classe.
`UserDefinedNamespace`	Nome do espaço para nomes personalizado definido pelo usuário.

Atributos de espaço para nomes de controle de acesso

O ActionScript 3.0 fornece quatro atributos especiais que controlam o acesso às propriedades definidas dentro de uma classe: `public`, `private`, `protected` e `internal`.

O atributo `public` torna a propriedade visível em qualquer lugar no script. Por exemplo, para disponibilizar um método para o código fora de seu pacote, declare o método com o atributo `public`. Isso é verdadeiro para qualquer propriedade, se ela for declarada usando as palavras-chave `var`, `const` ou `function`.

O atributo `private` torna a propriedade visível apenas para chamadores dentro da classe de definição da propriedade. Esse comportamento é diferente daquele do atributo `private` no ActionScript 2.0, que permite que uma subclasse acesse uma propriedade private em uma superclasse. Outra alteração significativa no comportamento que tem a ver com acesso em tempo de execução. No ActionScript 2.0, a palavra-chave `private` proibia o acesso apenas em tempo de compilação e era facilmente contornada em tempo de execução. No ActionScript 3.0, isso não é mais verdade. Propriedades que estão marcadas como `private` não estão disponíveis em tempo de compilação e em tempo de execução.

Por exemplo, o seguinte código cria uma classe simples denominada PrivateExample com uma variável private, e tenta acessar a variável private de fora da classe. No ActionScript 2.0, o acesso em tempo de compilação era proibido, mas a proibição era facilmente contornada usando o operador de acesso de propriedade (`[]`), que faz a pesquisa de propriedades em tempo de execução em vez de em tempo de compilação.

```
class PrivateExample
{
    private var privVar:String = "private variable";
}

var myExample:PrivateExample = new PrivateExample();
trace(myExample.privVar);// compile-time error in strict mode
trace(myExample["privVar"]); // ActionScript 2.0 allows access, but in ActionScript 3.0, this
is a run-time error.
```

No ActionScript 3.0, uma tentativa de acessar uma propriedade private usando o operador ponto (`myExample.privVar`) resultará em um erro em tempo de compilação se você estiver usando modo estrito. Caso contrário, o erro será relatado em tempo de execução, exatamente como quando você usa o operador de acesso de propriedade (`myExample["privVar"]`).

A tabela a seguir resume os resultados da tentativa de acessar uma propriedade private que pertence a uma classe selada (não dinâmica):

	modo Estrito	**modo Padrão**
operador ponto (`.`)	erro em tempo de compilação	erro em tempo de execução
operador colchete (`[]`)	erro em tempo de execução	erro em tempo de execução

Em classes declaradas com o atributo `dynamic`, as tentativas de acessar uma variável private não resultará em um erro em tempo de execução. Em vez disso, a variável simplesmente não é visível, portanto, o Flash Player ou o Adobe® AIR™ retorna o valor `undefined`. No entanto, ocorrerá um erro em tempo de compilação, se você usar o operador ponto no modo estrito. O exemplo a seguir é o mesmo do exemplo anterior, exceto que a classe PrivateExample é declarada como uma classe dinâmica:

```
dynamic class PrivateExample
{
    private var privVar:String = "private variable";
}

var myExample:PrivateExample = new PrivateExample();
trace(myExample.privVar);// compile-time error in strict mode
trace(myExample["privVar"]); // output: undefined
```

As classes dinâmicas geralmente retornam o valor `undefined` em vez de gerar um erro quando o código externo a uma classe tenta acessar uma propriedade private. A tabela a seguir mostra que um erro é gerado apenas quando o operador ponto é usado para acessar uma propriedade private no modo estrito:

	modo Estrito	**modo Padrão**
operador ponto (`.`)	erro em tempo de compilação	`undefined`
operador colchete (`[]`)	`undefined`	`undefined`

O atributo `protected`, que é novo para o ActionScript 3.0, torna uma propriedade visível a chamadores dentro de sua própria classe ou em uma subclasse. Em outras palavras, uma propriedade protected está disponível dentro de sua própria classe ou a classes que estão em qualquer lugar abaixo dela na hierarquia de heranças. Isso será verdadeiro se a subclasse estiver no mesmo pacote ou em um pacote diferente.

Para quem está familiarizado com o ActionScript 2.0, essa funcionalidade é semelhante ao atributo `private` no ActionScript 2.0. O atributo `protected` do ActionScript 3.0 também é semelhante ao atributo `protected` no Java, mas difere já que a versão do Java também permite acesso a chamadores dentro do mesmo pacote. O atributo `protected` é útil quando você tem uma variável ou método que as subclasses precisam, mas que você deseja ocultar do código que está fora da cadeia de heranças.

O atributo `internal`, que é novo para o ActionScript 3.0, torna uma propriedade visível a chamadores dentro de seu próprio pacote. Este é o atributo padrão do código dentro de um pacote e se aplica a qualquer propriedade que não tenha nenhum dos seguintes atributos:

- `public`
- `private`
- `protected`
- um espaço para nomes definido pelo usuário

O atributo `internal` é semelhante ao controle de acesso padrão no Java, embora no Java não haja nenhum nome explícito para este nível de acesso, e ele pode ser alcançado apenas por meio da omissão de qualquer outro modificador de acesso. O atributo `internal` está disponível no ActionScript 3.0 para fornecer a opção de indicar explicitamente a intenção tornar a propriedade visível apenas para chamadores dentro de seu próprio pacote.

atributo static

O atributo `static`, que pode ser usado com propriedades declaradas com as palavras-chave `var`, `const` ou `function`, permite anexar uma propriedade à classe em vez de às ocorrências da classe. O código externo à classe deve chamar propriedades estáticas usando o nome da classe em vez do nome de uma ocorrência.

As propriedades estáticas não são herdadas pelas subclasses, mas fazem parte da cadeia de escopos de uma subclasse. Isso significa que dentro do corpo de uma subclasse, uma variável ou método estático pode ser usado sem fazer referência à classe na qual ele foi definido. Para obter mais informações, consulte "Propriedades estáticas não herdadas" na página 116.

Atributos de espaço para nomes definidos pelo usuário

Como alternativa aos atributos de controle de acesso predefinidos, você pode criar um espaço para nomes personalizado para uso como um atributo. Apenas um atributo de espaço para nomes pode ser usado por definição, e você não pode usar um atributo de espaço para nomes em combinação com qualquer um dos atributos de controle de acesso (`public`, `private`, `protected`, `internal`). Para obter mais informações sobre como usar espaços para nomes, consulte "Espaços para nomes" na página 43.

Variáveis

As variáveis podem ser declaradas com as palavras-chave `var` ou `const`. Variáveis declaradas com a palavra-chave `var` podem ter seus valores alterados várias vezes durante a execução de um script. Variáveis declaradas com a palavra-chave `const` são chamadas *constants* e valores podem ser atribuídos a elas apenas uma vez. Uma tentativa de atribuir um novo valor a uma constante inicializada resulta em um erro. Para obter mais informações, consulte "Constantes" na página 69.

Variáveis estáticas

As variáveis estáticas são declaradas usando uma combinação da palavra-chave `static` e da instrução `var` ou `const`. As variáveis estáticas que são anexadas a uma classe em vez de a uma ocorrência de a uma classe são úteis para armazenar e compartilhar informações que se aplicam a uma classe inteira de objetos. Por exemplo, uma variável estática será apropriada se você desejar manter uma contagem do número de vezes que uma classe é instanciada ou se desejar armazenar o número máximo de ocorrências da classe que são permitidas.

O exemplo a seguir cria uma variável `totalCount` para rastrear o número de instanciações de classes e uma constante `MAX_NUM` para armazenar o número máximo de instanciações. As variáveis `totalCount` e `MAX_NUM` são estáticas porque contêm valores que se aplicam à classe como um todo em vez de a uma ocorrência específica.

```
class StaticVars
{
    public static var totalCount:int = 0;
    public static const MAX_NUM:uint = 16;
}
```

O código que é externo à classe StaticVars e qualquer uma de suas subclasses pode fazer referência às propriedades `totalCount` e `MAX_NUM` apenas por meio da própria classe. Por exemplo, o código a seguir funciona:

```
trace(StaticVars.totalCount); // output: 0
trace(StaticVars.MAX_NUM); // output: 16
```

Não é possível acessar variáveis estáticas por meio de uma ocorrência da classe, portanto, o código a seguir retorna erros:

```
var myStaticVars:StaticVars = new StaticVars();
trace(myStaticVars.totalCount); // error
trace(myStaticVars.MAX_NUM); // error
```

Variáveis que são declaradas com as palavras-chave `static` e `const` devem ser inicializadas ao mesmo tempo em que a constante é declarada, como faz a classe StaticVars para `MAX_NUM`. Você não pode atribuir um valor a `MAX_NUM` dentro do construtor ou de um método da ocorrência. O código a seguir gerará um erro, porque não é uma maneira válida de inicializar uma constante estática:

```
// !! Error to initialize static constant this way
class StaticVars2
{
    public static const UNIQUESORT:uint;
    function initializeStatic():void
    {
        UNIQUESORT = 16;
    }
}
```

Variáveis de ocorrência

As variáveis de ocorrência incluem propriedades declaradas com as palavras-chave `var` e `const`, mas sem a palavra-chave `static`. As variáveis de ocorrência que são anexadas às ocorrências da classe em vez de a uma classe inteira são úteis para armazenar valores específicos a uma ocorrência. Por exemplo, a classe Array tem uma propriedade de ocorrência denominada `length` que armazena o número de elementos da matriz que é mantida por uma ocorrência específica da classe Array.

Variáveis de ocorrência, se declaradas como `var` ou `const`, não podem ser substituídas em uma subclasse. No entanto, você pode alcançar funcionalidade semelhante substituindo variáveis pelos métodos getter e setter. Para obter mais informações, consulte "Métodos de acessor get e set" na página 103.

Métodos

Métodos são funções que fazem parte de uma definição de classe. Depois que uma ocorrência da classe é criada, um método é ligado a essa ocorrência. Ao contrário de uma função declarada fora de uma classe, um método não pode ser usado à parte da ocorrência à qual ele está anexado.

Os métodos são definidos usando a palavra-chave `function`. Assim como em qualquer propriedade de classe, você pode aplicar atributos de propriedade de classe a métodos, incluindo um espaço para nome privado, protegido, público, interno, estático ou personalizado. Você pode usar uma instrução de função da seguinte maneira:

```
public function sampleFunction():String {}
```

Ou pode usar uma variável à qual atribuir uma expressão de função, da seguinte maneira:

```
public var sampleFunction:Function = function () {}
```

Na maior parte dos casos convém usar uma instrução function em vez de uma expressão function pelas seguintes razões:

- As instruções function são mais concisas e mais fáceis de ler.
- As instruções function permitem usar as palavras-chave `override` e `final`. Para obter mais informações, consulte "Substituição de métodos" na página 114.
- As instruções function criam uma ligação mais forte entre o identificador, isto é, o nome da função, e o código dentro do corpo do método. Como o valor de uma variável pode ser alterado com uma instrução de atribuição, a conexão entre uma variável e sua expressão de função pode ser desfeita a qualquer momento. Embora você possa solucionar esse problema declarando a variável com `const` em vez de `var`, essa técnica não é considerada uma prática recomendada, porque ela torna código difícil de ler e impede o uso das palavras-chave `override` e `final`.

Um caso em que você deve usar uma expressão function é ao optar por anexar uma função ao objeto de protótipo. Para obter mais informações, consulte "Objeto de protótipo" na página 123.

Métodos do construtor

Os métodos de construtor, às vezes chamados simplesmente de *construtores*, são funções que compartilham o mesmo nome da classe na qual eles são definidos. Qualquer código incluído em um método de construtor é executado todas as vezes que uma ocorrência da classe é criada com a palavra-chave `new`. Por exemplo, o código a seguir define uma classe simples denominada Example que contém uma única propriedade denominada `status`. O valor inicial da variável `status` é definido dentro da função de construtor.

```
class Example
{
    public var status:String;
    public function Example()
    {
        status = "initialized";
    }
}

var myExample:Example = new Example();
trace(myExample.status); // output: initialized
```

Os métodos de construtor podem ser apenas públicos, mas o uso do atributo `public` é opcional. Você não pode usar nenhum dos outros especificadores de controle de acesso, inclusive `private`, `protected` ou `internal`, em um construtor. Você também não pode usar um espaço para nomes definido pelo usuário com um método de construtor.

Um construtor pode fazer uma chamada explícita para o construtor de sua superclasse direta usando a instrução `super()`. Se o construtor da superclasse não for chamado explicitamente, o compilador inserirá automaticamente uma chamada antes da primeira instrução no corpo do construtor. Você também pode chamar métodos da superclasse usando o prefixo `super` como uma referência à superclasse. Se você decidir usar `super()` e `super` no mesmo corpo do construtor, verifique se `super()` é chamado primeiro. Caso contrário, a referência `super` não se comportará conforme esperado. O construtor `super()` deve ser chamado também antes de qualquer instrução `throw` ou `return`.

O exemplo a seguir demonstra o que acontecerá se você tentar usar a referência `super` antes de chamar o construtor `super()`. Uma nova classe, ExampleEx, estende a classe Example. O construtor ExampleEx tenta acessar a variável status definida em sua superclasse, mas faz isso antes de chamar `super()`. A instrução `trace()` dentro do construtor ExampleEx produz o valor `null`, porque a variável `status` não está disponível até que o construtor `super()` seja executado.

```
class ExampleEx extends Example
{
    public function ExampleEx()
    {
        trace(super.status);
        super();
    }
}

var mySample:ExampleEx = new ExampleEx(); // output: null
```

Embora seja válido usar a instrução `return` dentro de um construtor, não é permitido retornar um valor. Em outras palavras, a instrução `return` não deve ter expressões ou valores associados. De forma correspondente, os métodos do construtor não têm permissão para retornar valores, o que significa que nenhum tipo de retorno pode ser especificado.

Se você não definir um método de construtor em sua classe, o compilador criará automaticamente um construtor vazio. Se a classe estender outra classe, o compilador incluirá uma chamada `super()` no construtor gerado.

Métodos estáticos

Os métodos estáticos, também chamados de *métodos de classe*, são métodos que são declarados com a palavra-chave `static`. Os métodos estáticos que são anexados a uma classe e não em uma ocorrência de uma classe, são úteis para encapsular a funcionalidade que afeta algo diferente do estado de uma ocorrência individual. Como os métodos estáticos são anexados a uma classe como um todo, eles podem ser acessados apenas por meio de uma classe e não por meio de uma ocorrência da classe.

Os métodos estáticos são úteis para encapsular funcionalidade que não está limitada a afetar o estado das ocorrências da classe. Em outras palavras, um método deverá ser estático se fornecer funcionalidade que não afete diretamente o valor de uma ocorrência de classe. Por exemplo, a classe Date tem um método estático denominado `parse()`, que obtém uma string e converte-a em um número. O método é estático porque não afeta uma ocorrência individual da classe. Em vez disso, o método `parse()` obtém uma string que representa um valor de data, analisa-a e retorna um número em um formato compatível com a representação interna de um objeto Date. Esse método não é um método da ocorrência, porque não faz sentido aplicar o método a uma ocorrência da classe Date.

Compare o método estático `parse()` com um dos métodos da ocorrência da classe Date, como `getMonth()`. O método `getMonth()` é um método de ocorrência, porque ele opera diretamente no valor de uma ocorrência, recuperando um componente específico, o mês, de uma ocorrência Date.

Como os métodos estáticos não estão ligados a ocorrências individuais, você não pode usar as palavras-chave `this` ou `super` dentro do corpo de um método estático. As referências `this` e `super` têm significado apenas dentro do contexto de um método de ocorrência.

Em comparação com algumas outras linguagens de programação com base em classe, os métodos estáticos no ActionScript 3.0 não são herdados. Para obter mais informações, consulte "Propriedades estáticas não herdadas" na página 116.

Métodos de ocorrência

Métodos de ocorrência são métodos declarados sem a palavra-chave `static`. Os métodos de ocorrência que são anexados a ocorrências de uma classe e não à classe como um todo, são úteis para implementar a funcionalidade que afeta ocorrências individuais de uma classe. Por exemplo, a classe Array contém um método de ocorrência denominado `sort()` que opera diretamente em ocorrências de Array.

Dentro do corpo de um método de ocorrência, as variáveis de ocorrência e estáticas estão no escopo, o que significa que as variáveis definidas na mesma classe podem ser referenciadas usando um identificador simples. Por exemplo, a seguinte classe, CustomArray, estende a classe Array. A classe CustomArray define uma variável estática denominada `arrayCountTotal` para rastrear o número total de ocorrências da classe, uma variável de ocorrência denominada `arrayNumber` que rastreia a ordem na qual as ocorrências foram criadas e o método de ocorrência denominado `getPosition()` que retorna os valores dessas variáveis.

```
public class CustomArray extends Array
{
    public static var arrayCountTotal:int = 0;
    public var arrayNumber:int;

    public function CustomArray()
    {
        arrayNumber = ++arrayCountTotal;
    }

    public function getArrayPosition():String
    {
         return ("Array " + arrayNumber + " of " + arrayCountTotal);
    }
}
```

Embora o código externo à classe deva fazer referência à variável estática `arrayCountTotal` por meio do objeto de classe, usando `CustomArray.arrayCountTotal`, o código que reside dentro do corpo do método `getPosition()` pode fazer referência diretamente à variável estática `arrayCountTotal`. Isso é verdadeiro mesmo para variáveis estáticas em superclasses. Apesar das propriedades estáticas não serem herdadas no ActionScript 3.0, as propriedades estáticas em superclasses estão no escopo. Por exemplo, a classe Array tem algumas variáveis estáticas, uma das quais é uma constante denominada `DESCENDING`. O código que reside em uma subclasse Array pode fazer referência à constante estática `DESCENDING` usando um identificador simples:

```
public class CustomArray extends Array
{
    public function testStatic():void
    {
        trace(DESCENDING); // output: 2
    }
}
```

O valor da referência `this` dentro do corpo de um método da ocorrência é uma referência à ocorrência à qual o método está anexado. O código a seguir demonstra que a referência `this` aponta para a ocorrência que contém o método:

```
class ThisTest
{
    function thisValue():ThisTest
    {
        return this;
    }
}

var myTest:ThisTest = new ThisTest();
trace(myTest.thisValue() == myTest); // output: true
```

A herança de métodos de ocorrência pode ser controlada com as palavras-chave `override` e `final`. Você pode usar o atributo `override` para redefinir um método herdado e o atributo `final` para impedir que as subclasses substituam um método. Para obter mais informações, consulte "Substituição de métodos" na página 114.

Métodos de acessor get e set

As funções de acessor get e set, também chamadas de *getters* e *setters*, permitem aderir aos princípios de programação de ocultação de informações e encapsulamento enquanto fornece uma interface de programação fácil de usar para as classes criadas. As funções get e set permitem manter as propriedades de classe privadas para a classe, mas permitem que usuários da classe acessem essas propriedades como se fossem acessar uma variável de classe em vez de chamar um método de classe.

A vantagem dessa abordagem é que ela permite evitar as funções de acessor tradicional com nomes inadequados, como `getPropertyName()` e `setPropertyName()`. Outra vantagem de getters e setters é que você pode evitar ter duas funções voltadas ao público para cada propriedade que permitam acesso de leitura e gravação.

O seguinte exemplo de classe, denominado GetSet, inclui funções do acessor get e set denominadas `publicAccess()` que fornecem acesso à variável privada denominada `privateProperty`:

```
class GetSet
{
    private var privateProperty:String;

    public function get publicAccess():String
    {
        return privateProperty;
    }

    public function set publicAccess(setValue:String):void
    {
        privateProperty = setValue;
    }
}
```

Se você tentar acessar a propriedade `privateProperty` diretamente, ocorrerá um erro, da seguinte maneira:

```
var myGetSet:GetSet = new GetSet();
trace(myGetSet.privateProperty); // error occurs
```

Em vez disso, um usuário da classe GetSet usa alguma coisa que parece ser uma propriedade denominada `publicAccess`, mas que realmente é um par de funções de acessor get e set que operam na propriedade privada denominada `privateProperty`. O exemplo a seguir instancia a classe GetSet e, em seguida, define o valor da `privateProperty` usando o acessor público denominado `publicAccess`:

```
var myGetSet:GetSet = new GetSet();
trace(myGetSet.publicAccess); // output: null
myGetSet.publicAccess = "hello";
trace(myGetSet.publicAccess); // output: hello
```

As funções getter e setter também permitem substituir propriedades que são herdadas de uma superclasse, algo que não é possível ao usar variáveis de membros de classe normal. As variáveis de membros de classe que são declaradas usando a palavra-chave `var` não podem ser substituídas em uma subclasse. No entanto as propriedades criadas usando as funções getter e setter não têm essa restrição. É possível usar o atributo `override` nas funções getter e setter que são herdadas de uma superclasse.

Métodos vinculados

Um método vinculado, às vezes chamado *fechamento de método*, é simplesmente um método extraído de sua ocorrência. Os exemplos de métodos vinculados incluem métodos que são passados como argumentos para uma função ou retornados de uma função como valores. Novo no ActionScript 3.0, um método vinculado é semelhante a um fechamento de função já que ele retém seu ambiente léxico mesmo quando extraído de sua ocorrência. No entanto a diferença principal entre um método vinculado e um fechamento de função é que a referência `this` de um método vinculado permanece vinculada, ou ligada, à ocorrência que implementa o método. Em outras palavras, a referência `this` em um método vinculado sempre aponta para o objeto original que implementou o método. Para fechamentos de funções, a referência `this` é genérica, o que significa que ela aponta para qualquer objeto com o qual a função está associada no momento em que é chamada.

É importante compreender os métodos vinculados ao usar a palavra-chave `this`. Lembre-se de que a palavra-chave `this` fornece uma referência ao objeto pai de um método. A maioria dos programadores do ActionScript espera que a palavra-chave `this` sempre faça referência ao objeto ou classe que contém a definição de um método. No entanto, sem a vinculação do método, isso não é sempre verdadeiro. Em versões anteriores do ActionScript, por exemplo, a referência `this` não se referia sempre à ocorrência que implementou o método. Quando os métodos são extraídos de

uma ocorrência no ActionScript 2.0, não somente a referência `this` não está vinculada à ocorrência original, mas também os métodos as variáveis de membros da classe da ocorrência não estão disponíveis. Esse não é um problema no ActionScript 3.0 porque os métodos vinculados são criados automaticamente quando você transmite um método como um parâmetro. Os métodos vinculados garantem que a palavra-chave `this` sempre faça referência ao objeto ou à classe na qual um método está definido.

O código a seguir define uma classe denominada ThisTest que contém um método denominado `foo()` que define o método vinculado e um método denominado `bar()` que retorna o método vinculado. O código externo à classe cria uma ocorrência da classe ThisTest, chama o método `bar()` e armazena o valor de retorno em uma variável denominada `myFunc`.

```
class ThisTest
{
    private var num:Number = 3;
    function foo():void // bound method defined
    {
        trace("foo's this: " + this);
        trace("num: " + num);
    }
    function bar():Function
    {
        return foo; // bound method returned
    }
}

var myTest:ThisTest = new ThisTest();
var myFunc:Function = myTest.bar();
trace(this); // output: [object global]
myFunc();
/* output:
foo's this: [object ThisTest]
output: num: 3 */
```

As duas últimas linhas do código mostram que a referência `this` no método vinculado `foo()` ainda aponta para uma ocorrência da classe ThisTest, mesmo que a referência `this` na linha imediatamente antes dela aponte para o objeto global. Além disso, o método vinculado armazenado na variável `myFunc` ainda tem acesso às variáveis de membros da classe ThisTest. Se esse mesmo código estiver em execução no ActionScript 2.0, as referências `this` corresponderão, e a variável `num` será `undefined`.

Uma área em que a adição de métodos vinculados é mais perceptível é a de identificadores de eventos, porque o método `addEventListener()` exige que você passe uma função ou método como um argumento. Para obter mais informações, consulte a Função de ouvinte definida como um método de classe em "Ouvintes de evento" na página 261.

Enumerações com classes

Enumerações são tipos de dados personalizados criados para encapsular um pequeno conjunto de valores. O ActionScript 3.0 não oferece suporte a um recurso de enumeração específico, ao contrário do C++ com sua palavra-chave `enum` ou do Java com sua interface de Enumeração. No entanto, você pode criar enumerações usando classes e constantes estáticas. Por exemplo, a classe PrintJob no ActionScript 3.0 usa uma enumeração denominada PrintJobOrientation para armazenar o conjunto de valores que compõem `"landscape"` e `"portrait"`, conforme mostrado no código a seguir:

```
public final class PrintJobOrientation
{
    public static const LANDSCAPE:String = "landscape";
    public static const PORTRAIT:String = "portrait";
}
```

Por convenção, uma classe de enumeração é declarada com o atributo `final`, porque não há nenhuma necessidade de estender a classe. A classe inclui apenas membros estáticos, o que significa que você não cria ocorrências da classe. Em vez disso, você acessa os valores de enumeração diretamente por meio do objeto de classe, conforme mostrado no trecho de código a seguir:

```
var pj:PrintJob = new PrintJob();
if(pj.start())
{
    if (pj.orientation == PrintJobOrientation.PORTRAIT)
    {
        ...
    }
    ...
}
```

Todas as classes de enumeração no ActionScript 3.0 contêm apenas variáveis do tipo String, int ou uint. A vantagem de usar enumerações em vez de string literal ou valores numéricos é que os erros tipográficos são mais fáceis de encontrar com enumerações. Se você digitar incorretamente o nome de uma enumeração, o compilador do ActionScript gerará um erro. Se você usar valores literais, o compilador não reclamará se você digitar uma palavra incorretamente ou usar um número incorreto. No exemplo anterior, o compilador gerará um erro se o nome da constante de enumeração estiver incorreto, conforme mostrado no trecho a seguir:

```
    if (pj.orientation == PrintJobOrientation.PORTRAI) // compiler error
```

No entanto o compilador não gerará um erro se você digitar de forma incorreta um valor literal da string, da seguinte maneira:

```
    if (pj.orientation == "portrai") // no compiler error
```

A segunda técnica para criar enumerações também envolve a criação de uma classe separada com propriedades estáticas para a enumeração. No entanto essa técnica difere já que cada uma das propriedades estáticas contém uma ocorrência da classe em vez de uma string ou um valor inteiro. Por exemplo, o código a seguir cria uma classe de enumeração para os dias da semana:

```
public final class Day
{
    public static const MONDAY:Day = new Day();
    public static const TUESDAY:Day = new Day();
    public static const WEDNESDAY:Day = new Day();
    public static const THURSDAY:Day = new Day();
    public static const FRIDAY:Day = new Day();
    public static const SATURDAY:Day = new Day();
    public static const SUNDAY:Day = new Day();
}
```

Essa técnica não é usada pelo ActionScript 3.0, mas é usada por muitos desenvolvedores que preferem a verificação de tipo aprimorada que a técnica fornece. Por exemplo, um método que retorna um valor de enumeração pode restringir o valor de retorno para o tipo de dados de enumeração. O código a seguir mostra não apenas uma função que retorna um dia da semana, mas também uma chamada de função que usa o tipo de enumeração como uma anotação de tipo:

```
function getDay():Day
{
    var date:Date = new Date();
    var retDay:Day;
    switch (date.day)
    {
        case 0:
            retDay = Day.MONDAY;
            break;
        case 1:
            retDay = Day.TUESDAY;
            break;
        case 2:
            retDay = Day.WEDNESDAY;
            break;
        case 3:
            retDay = Day.THURSDAY;
            break;
        case 4:
            retDay = Day.FRIDAY;
            break;
        case 5:
            retDay = Day.SATURDAY;
            break;
        case 6:
            retDay = Day.SUNDAY;
            break;
    }
    return retDay;
}

var dayOfWeek:Day = getDay();
```

Você também pode aprimorar a classe Day para que ela associe um inteiro a cada dia da semana, e forneça um método `toString()` que retorne uma representação da string do dia. Você pode desejar aprimorar a classe Day dessa maneira como um exercício.

Classes de ativos incorporados

O ActionScript 3.0 usa classes especiais, chamadas *classes de ativos incorporados*, para representar ativos incorporados. Um *ativo incorporado* é um ativo, como um som, imagem ou fonte, que é incluído em um arquivo SWF no momento da compilação. Incorporar um ativo, em vez de carregá-lo dinamicamente, garante que ele estará disponível em tempo de execução, mas ao custo do tamanho do arquivo SWF aumentado.

Uso de classes de ativos incorporados no Flash

Para incorporar um ativo, coloque primeiro o ativo em uma biblioteca do arquivo FLA. Em seguida, use a propriedade de ligação do ativo para fornecer um nome para a classe de ativo incorporado. Se uma classe por esse nome não puder ser encontrada no caminho de classe, ela será gerada automaticamente para você. Portanto você pode criar uma ocorrência da classe de ativo incorporado e usar todas as propriedades e métodos definidos ou herdados por essa classe. Por exemplo, o código a seguir pode ser usado para reproduzir um som incorporado que está vinculado a uma classe de ativo incorporado denominada PianoMusic:

```
var piano:PianoMusic = new PianoMusic();
var sndChannel:SoundChannel = piano.play();
```

Interfaces

Uma interface é uma coleção de declarações de métodos que permite que objetos não relacionados se comuniquem. Por exemplo, o ActionScript 3.0 define a interface IEventDispatcher que contém declarações de métodos que uma classe pode usar para manipular objetos de eventos. A interface IEventDispatcher estabelece uma maneira padrão para os objetos passarem objetos de eventos entre si. O código a seguir mostra a definição da interface IEventDispatcher:

```
public interface IEventDispatcher
{
    function addEventListener(type:String, listener:Function,
            useCapture:Boolean=false, priority:int=0,
            useWeakReference:Boolean = false):void;
    function removeEventListener(type:String, listener:Function,
            useCapture:Boolean=false):void;
    function dispatchEvent(event:Event):Boolean;
    function hasEventListener(type:String):Boolean;
    function willTrigger(type:String):Boolean;
}
```

As interfaces são baseadas na distinção entre uma interface do método e sua implementação. Uma interface do método inclui todas as informações necessárias para chamá-lo, inclusive o nome do método, todos os seus parâmetros e seu tipo de retorno. Uma implementação do método inclui não apenas as informações da interface, mas também as instruções executáveis que executam o comportamento do método. Uma definição de interface contém apenas interfaces do método, e qualquer classe que implemente a interface é responsável por definir as implementações do método.

No ActionScript 3.0, a classe EventDispatcher implementa a interface IEventDispatcher definindo todos os métodos da interface IEventDispatcher e adicionando corpos de métodos a cada um dos métodos. O exemplo a seguir é um trecho da definição da classe EventDispatcher:

```
public class EventDispatcher implements IEventDispatcher
{
    function dispatchEvent(event:Event):Boolean
    {
        /* implementation statements */
    }

    ...
}
```

A interface IEventDispatcher serve como um protocolo que as ocorrências do EventDispatcher usam para processar objetos de eventos e passá-los para outros objetos que também implementaram a interface IEventDispatcher.

Outra maneira de descrever uma interface é dizer que ela define um tipo de dados exatamente como faz uma classe. Conseqüentemente, uma interface pode ser usada como uma anotação de tipo, exatamente como uma classe. Como um tipo de dados, uma interface pode ser usada também com operadores, como os operadores `is` e `as` que exigem um tipo de dados. No entanto, ao contrário de uma classe, uma interface não pode ser instanciada. Essa distinção levou muitos programadores a considerar interfaces como tipos de dados abstratos e as classes como tipos de dados concretos.

Definição de uma interface

A estrutura de uma definição de interface é semelhante à da definição de uma classe, exceto que uma interface pode conter apenas métodos sem nenhum corpo de método. As interfaces não podem incluir variáveis ou constantes, mas podem incluir getters e setters. Para definir uma interface, use a palavra-chave `interface`. Por exemplo, a seguinte interface, IExternalizable, faz parte do pacote flash.utils no ActionScript 3.0. A interface IExternalizable define um protocolo para serializar um objeto, o que significa converter um objeto em um formato adequado para armazenamento em um dispositivo ou para transporte pela rede.

```
public interface IExternalizable
{
    function writeExternal(output:IDataOutput):void;
    function readExternal(input:IDataInput):void;
}
```

Observe que a interface IExternalizable é declarada com o modificador de controle de acesso `public`. As definições de interface podem ser modificadas apenas pelos especificadores de controle de acesso `public` e `internal`. As declarações de método dentro de uma definição de interface não podem ter nenhum especificador de controle de acesso.

O ActionScript 3.0 segue uma convenção na qual os nomes de interface começam com um `I` maiúsculo, mas você pode usar qualquer identificador válido como o nome de uma interface. As definições de interface são sempre colocadas no nível superior de um pacote. As definições de interface não podem ser colocadas dentro de uma definição de classe ou dentro da definição de outra interface.

As interfaces podem estender uma ou mais interfaces. Por exemplo, a seguinte interface, IExample, estende a interface IExternalizable:

```
public interface IExample extends IExternalizable
{
    function extra():void;
}
```

Qualquer classe que implemente a interface IExample deve incluir implementações não apenas para o método `extra()`, mas também para os métodos `writeExternal()` e `readExternal()` herdados da interface IExternalizable.

Implementação de uma interface em uma classe

Uma classe é o único elemento de linguagem do ActionScript 3.0 que pode implementar uma interface. Use a palavra-chave `implements` em uma declaração de classe para implementar uma ou mais interfaces. O exemplo a seguir define duas interfaces, IAlpha e IBeta, e uma classe, Alpha, que implementa as duas:

```
interface IAlpha
{
    function foo(str:String):String;
}

interface IBeta
{
    function bar():void;
}

class Alpha implements IAlpha, IBeta
{
    public function foo(param:String):String {}
    public function bar():void {}
}
```

Em uma classe que implementa uma interface, os métodos implementados devem fazer o seguinte:

- Usar o identificador de controle de acesso `public`.
- Usar o mesmo nome do método da interface.
- Ter o mesmo número de parâmetros, cada um com tipos de dados que correspondam aos tipos de dados do parâmetro do método da interface.
- Usar o mesmo tipo de retorno.

```
public function foo(param:String):String {}
```

No entanto, você realmente tem alguma flexibilidade quanto a como nomear os parâmetros de métodos implementados. Embora o número de parâmetros e o tipo de dados de cada parâmetro no método implementado devam corresponder àquele do método da interface, os nomes de parâmetros não precisam corresponder. Por exemplo, no exemplo anterior o parâmetro do método `Alpha.foo()` é denominado `param`:

Mas o parâmetro é denominado `str` no método da interface `IAlpha.foo()`:

```
function foo(str:String):String;
```

Você tem também alguma flexibilidade com valores de parâmetro padrão. Uma definição de interface pode incluir declarações de função com valores de parâmetro padrão. Um método que implementa um declaração de função desse tipo deve ter um valor de parâmetro padrão que seja membro do mesmo tipo de dados que o valor especificado na definição da interface, mas o valor real não precisa corresponder. Por exemplo, o código a seguir define uma interface que contém um método com um valor de parâmetro padrão 3:

```
interface IGamma
{
    function doSomething(param:int = 3):void;
}
```

A seguinte definição de classe implementa a interface Igamma, mas usa um valor do parâmetro padrão diferente:

```
class Gamma implements IGamma
{
    public function doSomething(param:int = 4):void {}
}
```

O motivo dessa flexibilidade é que as regras para implementação de uma interface são projetadas especificamente para garantir a compatibilidade do tipo de dados, e exigir nomes de parâmetros idênticos e valores de parâmetros padrão não é necessário para atingir esse objetivo.

Herança

Herança é uma forma de reutilização de código que permite que programadores desenvolvam novas classes com base em classes existentes. As classes existentes são sempre conhecidas como *classes base* ou *superclasses*, enquanto as novas classes normalmente são chamadas de *subclasses*. A vantagem principal da herança é que ela permite reutilizar código de uma classe base e ainda deixar o código existente inalterado. Além disso, a herança não requer nenhuma alteração no modo como as outras classes interagem com a classe base. Em vez de modificar uma classe existente que pode ter sido completamente testada ou já estar em uso, usando herança você pode tratar essa classe como um módulo integrado que pode ser estendido com propriedades ou métodos adicionais. De forma correspondente, você usa a palavra-chave `extends` para indicar que uma classe herda de outra classe.

A herança também permite que você se beneficie com o *polimorfismo* do código. Polimorfismo é a habilidade de usar um único nome de método para um método que se comporta de maneira diferente ao ser aplicado a diferentes tipos de dados. Um exemplo simples é uma classe base denominada Shape com duas subclasses denominadas Circle e Square. A classe Shape define um método denominado `area()`, que retorna a área da forma. Se o polimorfismo estiver implementado, você poderá chamar o método `area()` em objetos do tipo Circle e Square e fazer com que os cálculos corretos sejam feitos para você. A herança ativa o polimorfismo permitindo que as subclasses sejam herdadas e redefinidas, ou *substituam*, métodos da classe base. No exemplo a seguir, o método `area()` é redefinido pelas classes Circle e Square:

```
class Shape
{
    public function area():Number
    {
        return NaN;
    }
}

class Circle extends Shape
{
    private var radius:Number = 1;
    override public function area():Number
    {
        return (Math.PI * (radius * radius));
    }
}

class Square extends Shape
{
    private var side:Number = 1;
    override public function area():Number
    {
        return (side * side);
    }
}

var cir:Circle = new Circle();
trace(cir.area()); // output: 3.141592653589793
var sq:Square = new Square();
trace(sq.area()); // output: 1
```

Como cada classe define um tipo de dados, o uso da herança cria um relacionamento especial entre a classe base e a classe que a estende. Uma subclasse garantidamente possui todas as propriedades de sua classe base, o que significa que uma ocorrência de uma subclasse pode ser sempre substituída por uma ocorrência da classe base. Por exemplo, se um método definir um parâmetro do tipo Shape, é válido passar um argumento do tipo Circle, porque Circle estende Shape, da seguinte maneira:

```
function draw(shapeToDraw:Shape) {}

var myCircle:Circle = new Circle();
draw(myCircle);
```

Propriedades da ocorrência e herança

Uma propriedade da ocorrência, se definida com as palavras-chave `function`, `var` ou `const`, será herdada por todas as subclasses desde que a propriedade não seja declarada com o atributo `private` na classe base. Por exemplo, a classe Event no ActionScript 3.0 tem várias subclasses que herdam propriedades comuns a todos os objetos de eventos.

Para alguns tipos de eventos, a classe Event contém todas as propriedades necessárias para definir o evento. Esses tipos de eventos não exigem propriedades da ocorrência além daquelas definidas na classe Event. Exemplos desses eventos são o evento `complete`, que ocorre quando os dados foram carregados com êxito, e o evento `connect`, que ocorre quando uma conexão de rede foi estabelecida.

O exemplo a seguir é um fragmento da classe Event que mostra algumas das propriedades e métodos herdados por subclasses. Como as propriedades são herdadas, uma ocorrência de qualquer subclasse pode acessar essas propriedades.

```
public class Event
{
    public function get type():String;
    public function get bubbles():Boolean;
    ...

    public function stopPropagation():void {}
    public function stopImmediatePropagation():void {}
    public function preventDefault():void {}
    public function isDefaultPrevented():Boolean {}
    ...
}
```

Outros tipos de eventos exigem propriedades exclusivas não estão disponíveis na classe Event. Esses eventos são definidos usando subclasses da classe Event para que novas propriedades possam ser adicionadas às propriedades definidas na classe Event. Um exemplo dessa subclasse é a classe MouseEvent, que adiciona propriedades exclusivas a eventos associados a movimento ou a cliques do mouse, como os eventos `mouseMove` e `click`. O exemplo a seguir é um fragmento da classe MouseEvent que mostra a definição de propriedades que existem na subclasse, mas não na classe base:

```
public class MouseEvent extends Event
{
    public static const CLICK:String= "click";
    public static const MOUSE_MOVE:String = "mouseMove";
    ...

    public function get stageX():Number {}
    public function get stageY():Number {}
    ...
}
```

Especificadores de controle de acesso e herança

Se uma propriedade for declarada com a palavra-chave `public`, ela será visível ao código em qualquer lugar. Isso significa que a palavra-chave `public`, ao contrário das palavras-chave `private`, `protected` e `internal`, não coloca nenhuma restrição sobre a herança da propriedade.

Se uma propriedade for declarada com a palavra-chave `private`, ela será visível apenas na classe que a define, o que significa que não será herdada por nenhuma subclasse. Esse comportamento é diferente nas versões anteriores do ActionScript, em que a palavra-chave `private` se comportava de maneira mais semelhante à palavra-chave `protected` do ActionScript 3.0.

A palavra-chave `protected` indica que uma propriedade é visível não apenas dentro da classe que a define, mas também a todas as subclasses. Ao contrário da palavra-chave `protected` na linguagem de programação Java, a palavra-chave `protected` no ActionScript 3.0 não torna a propriedade visível para todas as outras classes no mesmo pacote. No ActionScript 3.0, apenas as subclasses podem acessar uma propriedade declarada com a palavra-chave `protected`. Além disso, uma propriedade protegida será visível para uma subclasse, se a subclasse estiver no mesmo pacote da classe base ou em um pacote diferente.

Para limitar a visibilidade de uma propriedade para o pacote no qual ela está definida, use a palavra-chave `internal` ou não use nenhum especificador de controle de acesso. O especificador de controle de acesso `internal` é o especificador padrão aplicado quando um não está especificado. Uma propriedade marcada como `internal` será herdada apenas por uma subclasse que reside no mesmo pacote.

Você pode usar o exemplo a seguir para ver como cada um dos especificadores de controle de acesso afeta a herança entre limites de pacotes. O código a seguir define uma classe de aplicativo principal denominada AccessControl e duas outras classes denominadas Base e Extender. A classe Base está em um pacote denominado foo e a classe Extender, que é uma subclasse da classe Base, está em um pacote denominado bar. A classe AccessControl importa apenas a classe Extender e cria uma ocorrência de Extender que tenta acessar uma variável denominada `str` definida na classe Base. A variável `str` é declarada como `public` para que o código seja compilado e executado, conforme mostrado no seguinte trecho:

```
// Base.as in a folder named foo
package foo
{
    public class Base
    {
        public var str:String = "hello"; // change public on this line
    }
}

// Extender.as in a folder named bar
package bar
{
    import foo.Base;
    public class Extender extends Base
    {
        public function getString():String {
            return str;
        }
    }
}

// main application class in file named AccessControl.as
package
{
    import flash.display.MovieClip;
    import bar.Extender;
    public class AccessControl extends MovieClip
    {
        public function AccessControl()
        {
            var myExt:Extender = new Extender();
            trace(myExt.str);// error if str is not public
            trace(myExt.getString()); // error if str is private or internal
        }
    }
}
```

Para ver como os outros especificadores de controle de acesso afetam a compilação e a execução do exemplo anterior, altere o especificador de controle de acesso da variável `str` para `private`, `protected` ou `internal` após excluir ou comentar a linha seguinte da classe `AccessControl`:

```
trace(myExt.str);// error if str is not public
```

Substituição de variáveis não permitidas

As propriedades declaradas com as palavras-chave `var` ou `const` são herdadas, mas não podem ser substituídas. Substituir uma propriedade significa redefini-la em uma subclasse. O único tipo de propriedade que pode ser substituído são os métodos, isto é, propriedades declaradas com a palavra-chave `function`. Embora não seja possível substituir uma variável de ocorrência, você pode alcançar funcionalidade semelhante criando os métodos getter e setter para a variável de ocorrência e substituindo os métodos. Para obter mais informações, consulte "Substituição de métodos" na página 114.

Substituição de métodos

Substituir um método significa redefinir o comportamento de um método herdado. Métodos estáticos não são herdados e não podem ser substituídos. No entanto métodos de ocorrência são herdados por subclasses e podem ser substituídos desde que os dois seguintes critérios sejam atendidos:

- O método da ocorrência não é declarado com a palavra-chave `final` na classe base. Quando usada com um método da ocorrência, a palavra-chave `final` indica a intenção do programador de impedir que as subclasses substituam o método.
- O método da ocorrência não é declarado com o especificador de controle de acesso `private` na classe base. Se um método estiver marcado como `private` na classe base, não haverá necessidade de usar a palavra-chave `override` ao definir um método nomeado de maneira idêntica na subclasse, porque o método da classe base não será visível para a subclasse.

  Para substituir um método da ocorrência que atenda a esses critérios, a definição do método na subclasse deve usar a palavra-chave `override` e deve corresponder à versão da superclasse do método das seguintes maneiras:

- O método de substituição deve ter o mesmo nível de controle de acesso do método da classe base. Métodos marcados como internos têm o mesmo nível de controle de acesso que os métodos que não têm nenhum especificador de controle de acesso.
- O método de substituição deve ter o mesmo número de parâmetros que o método da classe base.
- Os parâmetros do método de substituição devem ter as mesmas anotações de tipo de dados que os parâmetros do método da classe base.
- O método de substituição deve ter o mesmo tipo de retorno que o método da classe base.

No entanto os nomes dos parâmetros no método de substituição não precisam corresponder aos nomes dos parâmetros na classe base, desde que o número de parâmetros e o tipo de dados de cada parâmetro correspondam.

A instrução super

Ao substituir um método, os programadores sempre querem aumentar o comportamento do método da superclasse que estão substituindo, em vez de substituir completamente o comportamento. Isso requer um mecanismo que permita que um método em uma subclasse chame a versão da superclasse de si próprio. A instrução `super` fornece esse mecanismo, já que ela contém uma referência à superclasse imediata. O exemplo a seguir define uma classe denominada Base que contém um método denominado `thanks()` e uma subclasse da classe Base denominada Extender que substitui o método `thanks()`. O método `Extender.thanks()` usa a instrução `super` para chamar `Base.thanks()`.

```
package {
    import flash.display.MovieClip;
    public class SuperExample extends MovieClip
    {
        public function SuperExample()
        {
            var myExt:Extender = new Extender()
            trace(myExt.thanks()); // output: Mahalo nui loa
        }
    }
}

class Base {
    public function thanks():String
    {
        return "Mahalo";
    }
}

class Extender extends Base
{
    override public function thanks():String
    {
        return super.thanks() + " nui loa";
    }
}
```

Substituição de getters e setters

Embora não seja possível substituir variáveis definidas em uma superclasse, você pode substituir getters e setters. Por exemplo, o código a seguir substitui um getter denominado `currentLabel` que é definido na classe MovieClip no ActionScript 3.0:

```
package
{
    import flash.display.MovieClip;
    public class OverrideExample extends MovieClip
    {
        public function OverrideExample()
        {
            trace(currentLabel)
        }
        override public function get currentLabel():String
        {
            var str:String = "Override: ";
            str += super.currentLabel;
            return str;
        }
    }
}
```

A saída da instrução `trace()` no construtor da classe OverrideExample é `Override: null`, que mostra que o exemplo pôde substituir a propriedade `currentLabel` herdada.

Propriedades estáticas não herdadas

Propriedades estáticas não são herdadas por subclasses. Isso significa que as propriedades estáticas não podem ser acessadas por meio de uma ocorrência de uma subclasse. Uma propriedade estática pode ser acessada apenas por meio do objeto da classe no qual ela é definida. Por exemplo, o código a seguir define uma classe base denominada Base e uma subclasse denominada Extender que estende a Base. Uma variável estática denominada `test` é definida na classe Base. O código conforme escrito no fragmento a seguir, não é compilado no modo estrito e gera um erro em tempo de execução no modo padrão.

```
package {
    import flash.display.MovieClip;
    public class StaticExample extends MovieClip
    {
        public function StaticExample()
        {
            var myExt:Extender = new Extender();
            trace(myExt.test);// error
        }
    }
}

class Base {
    public static var test:String = "static";
}

class Extender extends Base { }
```

A única maneira de acessar a variável estática `test` é por meio do objeto da classe, conforme mostrado no código a seguir:

```
Base.test;
```

No entanto é permitido definir uma propriedade da ocorrência usando o mesmo nome de uma propriedade estática. Essa propriedade da ocorrência pode ser definida na mesma classe que a propriedade estática ou em uma subclasse. Por exemplo, a classe Base no exemplo anterior podia ter uma propriedade da ocorrência denominada `test`. O código a seguir é compilado e executado porque a propriedade da ocorrência é herdada pela classe Extender. O código também será compilado e executado, se a definição da variável da ocorrência de teste for movida, mas não copiada, para a classe Extender.

```
package
{
    import flash.display.MovieClip;
    public class StaticExample extends MovieClip
    {
        public function StaticExample()
        {
            var myExt:Extender = new Extender();
            trace(myExt.test);// output: instance
        }
    }
}

class Base
{
    public static var test:String = "static";
    public var test:String = "instance";
}

class Extender extends Base {}
```

Propriedades estáticas e a cadeia de escopos

Embora as propriedades estáticas não sejam herdadas, elas estão dentro da cadeia do escopo da classe que as define e em qualquer subclasse dessa classe. Como tal, diz-se que as propriedades estáticas estão *in scope* da classe na qual elas são definidas e em qualquer subclasse. Isso significa que uma propriedade estática pode ser acessada diretamente dentro do corpo da classe que a define e em qualquer subclasse dessa classe.

O exemplo a seguir modifica as classes definidas no exemplo anterior para mostrar que a variável estática `test` definida na classe Base está no escopo da classe Extender. Em outras palavras, a classe Extender pode acessar a variável estática `test` sem prefixar a variável com o nome da classe que define `test`.

```
package
{
    import flash.display.MovieClip;
    public class StaticExample extends MovieClip
    {
        public function StaticExample()
        {
            var myExt:Extender = new Extender();
        }
    }
}

class Base {
    public static var test:String = "static";
}

class Extender extends Base
{
    public function Extender()
    {
        trace(test); // output: static
    }

}
```

Se for definida uma propriedade de ocorrência que usa o mesmo nome que uma propriedade estática na mesma classe ou em uma superclasse, a propriedade de ocorrência terá precedência mais alta na cadeia do escopo. Diz-se que a propriedade da ocorrência *sombreia* a propriedade estática, o que significa que o valor da propriedade da ocorrência é usado no lugar do valor da propriedade estática. Por exemplo, o código a seguir mostra que se a classe Extender definir uma variável da ocorrência denominada `test`, a instrução `trace()` usará o valor da variável da ocorrência em vez do valor da variável estática:

```
package
{
    import flash.display.MovieClip;
    public class StaticExample extends MovieClip
    {
        public function StaticExample()
        {
            var myExt:Extender = new Extender();
        }
    }
}

class Base
{
    public static var test:String = "static";
}

class Extender extends Base
{
    public var test:String = "instance";
    public function Extender()
    {
        trace(test); // output: instance
    }

}
```

Tópicos avançados

Esta seção é iniciada com um breve histórico do ActionScript e do OOP e continua com uma discussão do modelo de objeto do ActionScript 3.0 e como ele permite que o novo AVM2 (ActionScript Virtual Machine) execute de maneira significativamente mais rápida do que em versões anteriores do Flash Player que contêm o AVM1 (ActionScript Virtual Machine) antigo.

Histórico do suporte da OOP ao ActionScript

Como o ActionScript 3.0 foi criado sobre versões anteriores do ActionScript, é útil compreender como o modelo de objeto do ActionScript evoluiu. O ActionScript começou como um mecanismo de script simples para versões anteriores da ferramenta de autoria do Flash. Subseqüentemente, os programadores começaram a criar aplicativos cada vez mais complexos com o ActionScript. Em resposta às necessidades desses programadores, cada versão subseqüente adicionou recursos de linguagem que facilitam a criação de aplicativos complexos.

ActionScript 1.0

O ActionScript 1.0 faz referência à versão da linguagem usada no Flash Player 6 e anterior. Mesmo na primeira fase de desenvolvimento, o modelo de objeto do ActionScript era baseado no conceito do objeto como um tipo de dados fundamental. Um objeto do ActionScript é um tipo de dados composto por um grupo de *propriedades*. Ao discutir o modelo de objeto, o termo *propriedades* inclui tudo o que está conectado a um objeto, como variáveis, funções ou métodos.

Embora essa primeira geração do ActionScript não ofereça suporte à definição de classes com uma palavra-chave `class`, é possível definir uma classe usando um tipo especial de objeto chamado objeto de protótipo. Em vez de usar uma palavra-chave `class` para criar uma definição de classe abstrata que você instancia em objetos concretos, como o faz em linguagens baseadas em classe, como Java e C++, as linguagens baseadas em protótipo como o ActionScript 1.0 usam um objeto existente como um modelo (ou protótipo) para outros objetos. Enquanto objetos em uma linguagem baseada em classe podem apontar para uma classe que serve como seu modelo, objetos em uma linguagem baseada em protótipo apontam para outro objeto, seu protótipo, que serve como seu modelo.

Para criar uma classe no ActionScript 1.0, defina uma função de construtor para essa classe. No ActionScript, as funções são objetos reais, não apenas definições abstratas. A função de construtor que você cria serve como o objeto de protótipo para ocorrências dessa classe. O código a seguir cria uma classe denominada Shape e define uma propriedade denominada `visible` que, por padrão, é definida como `true`:

```
// base class
function Shape() {}
// Create a property named visible.
Shape.prototype.visible = true;
```

Essa função de construtor define uma classe Shape que você pode instanciar com o operador `new`, da seguinte maneira:

```
myShape = new Shape();
```

Do mesmo modo como o objeto de função de construtor `Shape()` serve como protótipo para ocorrências da classe Shape, ele também pode servir como o protótipo para subclasses de Shape, isto é, outras classes que estendem a classe Shape.

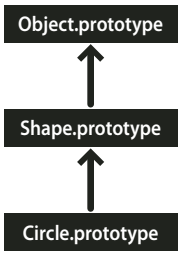
A criação de uma classe que é uma subclasse da classe Shape é um processo de duas etapas. Primeiro, crie a classe definindo uma função de construtor para a classe, da seguinte maneira:

```
// child class
function Circle(id, radius)
{
this.id = id;
this.radius = radius;
}
```

Segundo, use o operador `new` para declarar que a classe Shape é o protótipo para a classe Circle. Por padrão, qualquer classe criada usa a classe Object como seu protótipo, o que significa que `Circle.prototype` atualmente contém um objeto genérico (uma ocorrência da classe Object). Para especificar que o protótipo de Circle é Shape em vez de Object, use o código a seguir para alterar o valor de `Circle.prototype` para que ele contenha um objeto Shape em vez de um objeto genérico:

```
// Make Circle a subclass of Shape.
Circle.prototype = new Shape();
```

As classes Shape e Circle estão agora vinculadas em conjunto em um relacionamento de herança que é conhecido como a *cadeia de protótipos*. O diagrama ilustra os relacionamentos em uma cadeia de protótipos:



A classe base no final de cada cadeia de protótipos é a classe Object. A classe Object contém uma propriedade estática denominada `Object.prototype` que aponta para o objeto de protótipo base de todos os objetos criados no ActionScript 1.0. O próximo objeto em nossa cadeia de protótipos de exemplo é o objeto Shape. Isso ocorre porque a propriedade `Shape.prototype` nunca foi definida explicitamente, portanto ela ainda mantém um objeto genérico (uma ocorrência da classe Object). O link final nessa cadeia é a classe Circle que está vinculada a seu protótipo, a classe Shape (a propriedade `Circle.prototype` mantém um objeto Shape).

Se criarmos uma ocorrência da classe Circle, como no seguinte exemplo, a ocorrência herdará a cadeia de protótipos da classe Circle:

```
// Create an instance of the Circle class.
myCircle = new Circle();
```

Lembre-se de que criamos uma propriedade denominada `visible` como um membro da classe Shape. Em nosso exemplo, a propriedade `visible` não existe como parte do objeto `myCircle`, apenas como membro do objeto Shape, apesar da linha seguinte do código produzir `true`:

```
trace(myCircle.visible); // output: true
```

O Flash Player pode verificar se o objeto `myCircle` herda a propriedade `visible` percorrendo a cadeia de protótipos. Ao executar este código, o Flash Player primeiro pesquisa a propriedades do objeto `myCircle` por uma propriedade denominada `visible`, mas não encontra essa propriedade. Em seguida, o Flash Player verifica o objeto `Circle.prototype`, mas ainda não encontra uma propriedade denominada `visible`. Continuando na cadeia de protótipos, o Flash Player finalmente encontra a propriedade `visible` definida no objeto `Shape.prototype` e fornece o valor daquela propriedade.

Pelo bem da simplicidade, esta seção omite muitos dos detalhes e complexidades da cadeia de protótipos e tem o objetivo de fornecer informações suficientes para ajudar a compreender o modelo de objeto do ActionScript 3.0.

ActionScript 2.0

O ActionScript 2.0 introduziu novas palavras-chave, como `class`, `extends`, `public`e `private`, que permitiram definir classes de uma maneira que seja familiar a qualquer pessoa que trabalhe com linguagens baseadas em classes como Java e C++. É importante compreender que o mecanismo da herança subjacente não foi alterado entre o ActionScript 1.0 e o ActionScript 2.0. O ActionScript 2.0 simplesmente adicionou uma nova sintaxe para definir classes. A cadeia de protótipos funciona da mesma maneira nas duas versões da linguagem.

A nova sintaxe introduzida pelo ActionScript 2.0, mostrada no trecho a seguir, permite definir classes de uma maneira que é considerada intuitiva por muitos programadores:

```
// base class
class Shape
{
var visible:Boolean = true;
}
```

Observe que o ActionScript 2.0 também introduziu anotações de tipo para uso com verificação de tipos em tempo de compilação. Isso permite declarar que a propriedade `visible` no exemplo anterior deve conter apenas um valor booleano. A nova palavra-chave `extends` também simplifica o processo de criação de uma subclasse. No exemplo a seguir, o processo em duas etapas necessário no ActionScript 1.0 é executado em uma etapa com a palavra-chave `extends`:

```
// child class
class Circle extends Shape
{
    var id:Number;
    var radius:Number;
    function Circle(id, radius)
    {
        this.id = id;
        this.radius = radius;
        }
}
```

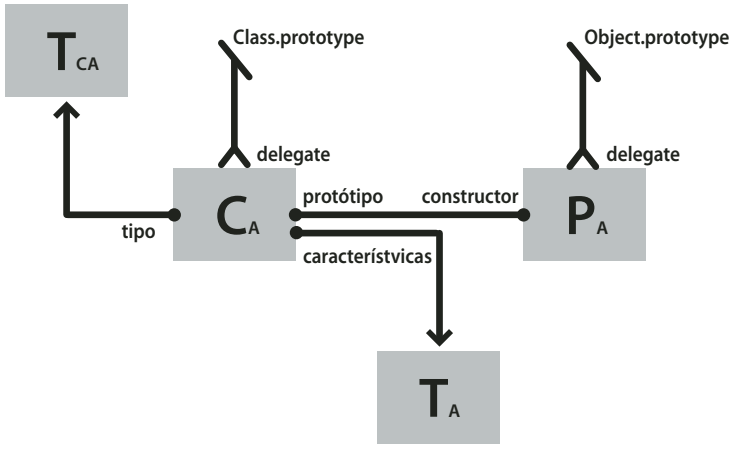
O construtor está agora declarado como parte da definição da classe e as propriedades de classes `id` e `radius` também devem ser declaradas explicitamente.

O ActionScript 2.0 também adicionou suporte para a definição de interfaces, o que permite refinar ainda mais os programas orientados a objetos com protocolos definidos formalmente para comunicação entre objetos.

Objeto de classe do ActionScript 3.0

Um paradigma comum da programação orientada a objetos, mais freqüentemente associado a Java e C++, usa classes para definir tipos de objetos. Linguagens de programação que adotam esse paradigma também tendem a usar classes para construir ocorrências do tipo de dados que a classe define. O ActionScript usa classes para duas dessas finalidades, mas suas raízes como uma linguagem com base em protótipo adicionam uma característica interessante. O ActionScript cria para cada definição de classe um objeto de classe especial que permite o compartilhamento do comportamento e do estado. No entanto, para muitos programadores do ActionScript, esta distinção não pode ter nenhuma implicação de codificação prática. O ActionScript 3.0 é projetado de forma que você possa criar aplicativos do ActionScript orientados a objetos sofisticados sem usar, ou mesmo compreender, esses objetos de classes especiais. Para programadores avançados que desejam tirar proveito de objetos de classes, esta seção discute os problemas em profundidade.

O diagrama a seguir mostra a estrutura de um objeto de classe que representa uma classe simples denominada A que é definida com a instrução `class A {}`:



Cada retângulo no diagrama representa um objeto. Cada objeto no diagrama tem um caractere subscrito A para representar que ele pertence à classe A. O objeto de classe (CA) contém referências a vários outros objetos importantes. Um objeto com características da ocorrência (TA) armazena as propriedades da ocorrência que são definidas dentro de uma definição de classe. Um objeto de características da classe (TCA) representa o tipo interno da classe e armazena as propriedades estáticas definidas pela classe (o caractere subscrito C representa a "classe"). O objeto de protótipo (PA) sempre faz referência ao objeto da classe ao qual ele era originalmente anexado por meio da propriedade `constructor`.

Objeto de características

O objeto de características, novo no ActionScript 3.0, foi implementado tendo em mente o desempenho. Em versões anteriores do ActionScript, a pesquisa de nome era um processo demorado pois o Flash Player percorria a cadeia de protótipos. No ActionScript 3.0, a pesquisa de nome é muito mais eficiente e menos demorada, porque as propriedades herdadas são copiadas das superclasses no objeto de características de subclasses.

O objeto de características não pode ser acessado diretamente pelo código do programador, mas sua presença pode ser sentida pelos aprimoramentos no desempenho e no uso de memória. O objeto de características fornece ao AVM2 informações detalhadas sobre o layout e o conteúdo de uma classe. Com esse conhecimento, o AVM2 pode reduzir significativamente o tempo de execução, porque pode gerar freqüentemente instruções de máquina diretas para acessar propriedades ou chamar métodos diretamente sem uma pesquisa de nome demorada.

Graças ao objeto de características, uma superfície de memória do objeto pode ser significativamente menor do que a de um objeto semelhante em versões anteriores do ActionScript. Por exemplo, se uma classe estiver selada (isto é, a classe não está declarada dinâmica), uma ocorrência da classe não precisará de uma tabela hash para propriedades adicionadas dinamicamente, e poderá manter um pouco mais do que um ponteiro para os objetos de características e alguns slots para as propriedades fixas definidas na classe. Como resultado, um objeto que exigia 100 bytes de memória no ActionScript 2.0 pode exigir apenas 20 bytes de memória no ActionScript 3.0.

Nota: *O objeto de características é um detalhe da implementação interna, e não há nenhuma garantia de que ele não seja alterado ou mesmo que desapareça em versões futuras do ActionScript.*

Objeto de protótipo

Cada objeto de classe do ActionScript tem uma propriedade denominada `prototype`, que é uma referência ao objeto de protótipo da classe. O objeto de protótipo é um herança das raízes do ActionScript como linguagem com base em protótipo. Para obter mais informações, consulte "Histórico do suporte da OOP ao ActionScript" na página 118.

A propriedade `prototype` é somente leitura, o que significa que não pode ser modificada para apontar para objetos diferentes. Ela é diferente da propriedade `prototype` da classe em versões anteriores do ActionScript, em que o protótipo podia ser reatribuído para que apontasse para uma classe diferente. Embora a propriedade `prototype` seja somente leitura, o objeto de protótipo ao qual ela faz referência não é. Em outras palavras, novas propriedades podem ser adicionadas ao objeto de protótipo. Propriedades adicionadas ao objeto de protótipo são compartilhadas entre todas as ocorrências da classe.

A cadeia de protótipos, que era o único mecanismo de herança em versões anteriores do ActionScript, serve apenas uma função secundária no ActionScript 3.0. O mecanismo de herança principal, herança de propriedade fixa, é manipulado internamente pelo objeto de características. Uma propriedade fixa é uma variável ou método que é definida como parte de uma definição de classe. A herança de propriedade fixa também é de chamada herança de classe, porque ela é o mecanismo de herança associado a palavras-chave, como `class`, `extends` e `override`.

A cadeia de protótipos fornece um mecanismo de herança alternativa que é mais dinâmico do que a herança de propriedade fixa. Você pode adicionar propriedades a um objeto de protótipo de classe não apenas como parte da definição da classe, mas também em tempo de execução por meio da propriedade `prototype` do objeto de classe. No entanto, observe que se você definir o compilador como modo estrito, talvez não seja possível acessar propriedades adicionadas a um objeto de protótipo, a não ser que você declare uma classe com a palavra-chave `dynamic`.

Um bom exemplo de uma classe com várias propriedades anexadas ao objeto de protótipo é a classe Object. Os métodos `toString()` e `valueOf()` da classe Object são realmente funções atribuídas às propriedades do objeto de protótipo da classe Object. A seguir está um exemplo de como pode ser a aparência da declaração desses métodos, em teoria, (a implementação real é um pouco diferente, por causa dos detalhes da implementação):

```
public dynamic class Object
{
    prototype.toString = function()
    {
        // statements
    };
    prototype.valueOf = function()
    {
        // statements
    };
}
```

Conforme mencionado anteriormente, é possível anexar uma propriedade a um objeto de protótipo de classe fora da definição de classe. Por exemplo, o método `toString()` também pode ser definido fora da definição da classe Object, da seguinte maneira:

```
Object.prototype.toString = function()
{
    // statements
};
```

No entanto, ao contrário da herança de propriedade fixa, a herança de protótipo não exigirá a palavra-chave `override`, se você desejar redefinir um método em uma subclasse. Por exemplo: se você desejar redefinir o método `valueOf()` em uma subclasse da classe Object, terá três opções. Primeiro, você pode definir um método `valueOf()` no objeto de protótipo de subclasse dentro da definição de classe. O código a seguir cria uma subclasse de Object denominada Foo e redefine o método `valueOf()` no objeto de protótipo de Foo como parte da definição de classe. Como cada classe é herdada de Object, não é necessário usar a palavra-chave `extends`.

```
dynamic class Foo
{
    prototype.valueOf = function()
    {
        return "Instance of Foo";
    };
}
```

Segundo, você pode definir um método `valueOf()` no objeto de protótipo de Foo fora da definição de classe, conforme mostrado no código a seguir:

```
Foo.prototype.valueOf = function()
{
    return "Instance of Foo";
};
```

Terceiro, você pode definir uma propriedade fixa denominada `valueOf()` como parte da classe Foo. Essa técnica é diferente das outras já que ela mescla herança de propriedade fixa com herança de protótipo. Qualquer subclasse de Foo que precise redefinir `valueOf()` deve usar a palavra-chave `override`. O código a seguir mostra `valueOf()` definido como uma propriedade fixa em Foo:

```
class Foo
{
    function valueOf():String
    {
        return "Instance of Foo";
    }
}
```

Espaço para nomes AS3

A existência de dois mecanismos de herança separados, herança de propriedade fixa e herança de protótipo, cria uma desafio de compatibilidade interessante em relação às propriedades e métodos das classes principais. A compatibilidade com a especificação de linguagem do ECMAScript na qual o ActionScript é baseado exige o uso de herança de protótipo, o que significa que as propriedades e métodos de uma classe principal são definidos no objeto de protótipo dessa classe. Por outro lado, a compatibilidade com o ActionScript 3.0 exige o uso de herança de propriedade fixa, o que significa que as propriedades e métodos de uma classe principal são definidos na definição da classe usando as palavras-chave `const`, `var` e `function`. Além disso, o uso de propriedades fixas, em vez das versões do protótipo pode levar a aumentos significativos no desempenho em tempo de execução.

O ActionScript 3.0 resolve esse problema usando herança de protótipo e herança de propriedade fixa para as classes principais. Cada classe principal contém dois conjuntos de propriedades e métodos. Um conjunto é definido no objeto de protótipo para compatibilidade com a especificação do ECMAScript, e o outro conjunto é definido com propriedades fixas e o espaço para nomes AS3 para compatibilidade com o ActionScript 3.0.

O espaço para nomes AS3 fornece um mecanismo conveniente para escolher entre os dois conjuntos de propriedades e métodos. Se você não usar o espaço para nomes AS3, uma ocorrência de uma classe principal herdará as propriedades e métodos definidos no objeto de protótipo da classe principal. Se você decidir usar o espaço para nomes AS3, uma ocorrência de uma classe principal herdará as versões do AS3, porque as propriedades fixas são sempre preferidas sobre as propriedades de protótipo. Em outras palavras, sempre que uma propriedade fixa estiver disponível, ela será sempre usada no lugar de uma propriedade de protótipo nomeada de forma idêntica.

Você pode usar seletivamente a versão do espaço para nomes AS3 de uma propriedade ou método qualificando-a com o espaço para nomes AS3. Por exemplo, o código a seguir usa a versão do AS3 do método `Array.pop()`:

```
var nums:Array = new Array(1, 2, 3);
nums.AS3::pop();
trace(nums); // output: 1,2
```

Como alternativa, você pode usar a diretiva `use namespace` para abrir o espaço para nomes AS3 para todas as definições dentro de um bloco de código. Por exemplo, o código a seguir usa a diretiva `use namespace` para abrir o espaço para nomes AS3 para os métodos `pop()` e `push()`:

```
use namespace AS3;

var nums:Array = new Array(1, 2, 3);
nums.pop();
nums.push(5);
trace(nums) // output: 1,2,5
```

O ActionScript 3.0 também fornece opções de compilador para cada conjunto de propriedades para que você possa aplicar o espaço para nomes AS3 ao programa inteiro. A opção de compilador `-as3` representa o espaço para nomes AS3, e a opção de compilador `-es` representa a opção de herança de protótipo (`es` representa o ECMAScript). Para abrir o espaço para nomes AS3 para o programa inteiro, defina a opção de compilador `-as3` como `true` e a opção de compilador `-es` como `false`. Para usar as versões de protótipo, defina as opções do compilador como os valores opostos. As configurações do compilador padrão do Adobe Flex Builder 3 e do Adobe Flash CS4 Professional são `-as3 = true` e `-es = false`.

Se você planejar estender qualquer uma das classes principais e substituir qualquer método, deverá compreender como o espaço para nomes AS3 pode afetar o modo como você deve declarar um método substituído. Se você estiver usando o espaço para nomes AS3, qualquer substituição de método de um método da classe principal também deve usar o espaço para nomes AS3 juntamente com o atributo `override`. Se não estiver usando o espaço para nomes AS3 e desejar redefinir um método da classe principal em uma subclasse, você não deverá usar o espaço para nomes AS3 ou a palavra-chave `override`.

Exemplo: GeometricShapes

O aplicativo de amostra GeometricShapes mostra como vários conceitos e recursos orientados a objetos podem ser aplicados usando o ActionScript 3.0, inclusive:

- Definição de classes
- Extensão de classes
- Polimorfismo e a palavra-chave `override`
- Definição, extensão e implementação de interfaces

Ele também inclui um "método de fábrica" que cria ocorrências de classes, mostrando como declarar um valor de retorno como uma ocorrência de uma interface, e usar esse objeto retornado de uma maneira genérica.

Para obter os arquivos do aplicativo desta amostra, consulte www.adobe.com/go/learn_programmingAS3samples_flash_br. Os arquivos do aplicativo GeometricShapes podem ser encontrados na pasta Amostras/GeometricShapes. O aplicativo consiste nos seguintes arquivos:

Arquivo	Descrição
GeometricShapes.mxml ou GeometricShapes.fla	O arquivo principal do aplicativo no Flash (FLA) ou no Flex (MXML).
com/example/programmingas3/geometricshapes/IGeometricShape.as	A interface base que define métodos a serem implementados por todas as classes do aplicativo GeometricShapes.
com/example/programmingas3/geometricshapes/IPolygon.as	Uma interface que define métodos a serem implementados pelas classes do aplicativo GeometricShapes que têm vários lados.
com/example/programmingas3/geometricshapes/RegularPolygon.as	Um tipo de forma geométrica que tem lados de comprimento igual prolongados simetricamente em torno do centro da forma.
com/example/programmingas3/geometricshapes/Circle.as	Um tipo de forma geométrica que define um círculo.
com/example/programmingas3/geometricshapes/EquilateralTriangle.as	Uma subclasse de RegularPolygon que define um triângulo com todos os lados com o mesmo comprimento.
com/example/programmingas3/geometricshapes/Square.as	Uma subclasse de RegularPolygon que define um retângulo com os quatro lados com o mesmo comprimento.
com/example/programmingas3/geometricshapes/GeometricShapeFactory.as	Uma classe que contém um método de fábrica para criar formas com tamanho e tipo especificados.

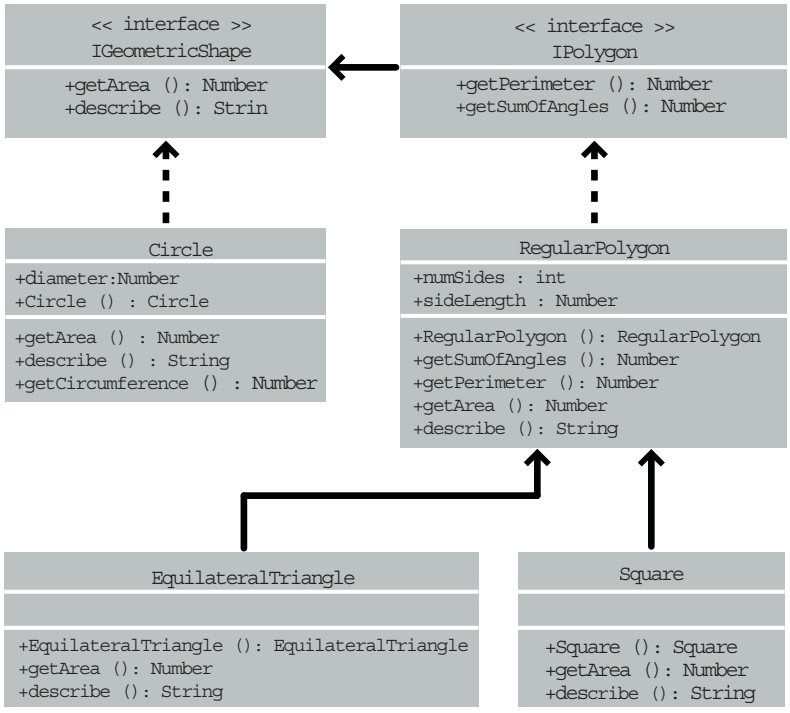
Definição das classes GeometricShapes

O aplicativo GeometricShapes permite que o usuário especifique um tipo de forma geométrica e um tamanho. Em seguida, ele responde com uma descrição da forma, sua área e a distância em torno de seu perímetro.

A interface de usuário do aplicativo é trivial, incluindo alguns controles para seleção do tipo de forma, configuração do tamanho e exibição da descrição. A parte mais interessante desse aplicativo está sob a superfície, na estrutura das classes e das próprias interfaces.

Esse aplicativo trata de formas geométricas, mas não as exibe graficamente. Ele fornece uma pequena biblioteca de classes e interfaces que serão reutilizadas no exemplo de um exemplo capítulo posterior (consulte “Exemplo: SpriteArranger” na página 316). O exemplo SpriteArranger exibe as formas graficamente e permite que o usuário as manipule, com base na estrutura da classe fornecida aqui no aplicativo GeometricShapes.

As classes e interfaces que definem as formas geométricas neste exemplo são mostradas no diagrama a seguir que usa a notação UML (Linguagem de modelação unificada):



Classes de exemplo do GeometricShapes

Definição do comportamento comum com interfaces

Este aplicativo GeometricShapes trata de três tipos de formas: círculos, quadrados e triângulos eqüiláteros. A estrutura de classe GeometricShapes começa com uma interface muito simples, IGeometricShape, que lista métodos comuns para todos os três tipos de formas:

```
package com.example.programmingas3.geometricshapes
{
    public interface IGeometricShape
    {
        function getArea():Number;
        function describe():String;
    }
}
```

A interface define dois métodos: o método `getArea()`, que calcula e retorna a área da forma, e o método `describe()`, que monta uma descrição de texto das propriedades da forma.

Desejamos saber também a distância em torno do perímetro de cada forma. No entanto, o perímetro de um círculo é chamado de circunferência, e é calculado de uma maneira exclusiva, portanto o comportamento diverge daquele de um triângulo ou de um quadrado. Ainda há semelhança suficiente entre triângulos, quadrados e outros polígonos, portanto faz sentido definir uma nova classe de interface só para eles: IPolygon. A interface IPolygon também é muito simples, conforme mostrado aqui:

```
package com.example.programmingas3.geometricshapes
{
    public interface IPolygon extends IGeometricShape
    {
        function getPerimeter():Number;
        function getSumOfAngles():Number;
    }
}
```

Essa interface define dois métodos comuns a todos os polígonos: o método `getPerimeter()` que mede a distância combinada de todos os lados e o método `getSumOfAngles()` que adiciona todos os ângulos internos.

A interface IPolygon estende a interface IGeometricShape, o que significa que qualquer classe que implemente a interface IPolygon deve declarar os quatro métodos, dois da interface IGeometricShape e dois da interface IPolygon.

Definição das classes Shape

Depois que você tiver uma boa idéia sobre os métodos comuns a cada tipo de forma, você pode definir as próprias classes. Em termos da quantidade dos métodos precisam ser implementados, a forma mais simples é a da classe Circle, mostrada aqui:

```
package com.example.programmingas3.geometricshapes
{
    public class Circle implements IGeometricShape
    {
        public var diameter:Number;

        public function Circle(diam:Number = 100):void
        {
            this.diameter = diam;
        }

        public function getArea():Number
        {
            // The formula is Pi * radius * radius.
            var radius:Number = diameter / 2;
            return Math.PI * radius * radius;
        }

        public function getCircumference():Number
        {
            // The formula is Pi * diameter.
            return Math.PI * diameter;
        }

        public function describe():String
        {
            var desc:String = "This shape is a Circle.\n";
            desc += "Its diameter is " + diameter + " pixels.\n";
            desc += "Its area is " + getArea() + ".\n";
            desc += "Its circumference is " + getCircumference() + ".\n";
            return desc;
        }
    }
}
```

A classe Circle implementa a interface IGeometricShape, portanto ela deve fornecer código para os métodos `getArea()` e `describe()`. Além disso, ela define o método `getCircumference()` que é exclusivo à classe Circle. A classe Circle também declara uma propriedade, `diameter` que não é encontrada nas outras classes de polígonos.

Os dois outros tipos de formas, quadrados e triângulos eqüiláteros, têm algumas outras coisas em comum: cada um deles têm lados com o mesmo comprimento e há fórmulas comuns que você pode usar para calcular o perímetro e a soma dos ângulos internos dos dois. De fato, essas fórmulas comuns são aplicadas a todos os outros polígonos regulares que você precisar definir no futuro.

A classe RegularPolygon é a superclasse das classes Square e EquilateralTriangle. Uma superclasse permite definir métodos comuns em um lugar, portanto você não precisa defini-los separadamente em cada subclasse. Este é o código da classe RegularPolygon:

```
package com.example.programmingas3.geometricshapes
{
    public class RegularPolygon implements IPolygon
    {
        public var numSides:int;
        public var sideLength:Number;

        public function RegularPolygon(len:Number = 100, sides:int = 3):void
        {
            this.sideLength = len;
            this.numSides = sides;
        }

        public function getArea():Number
        {
            // This method should be overridden in subclasses.
            return 0;
        }

        public function getPerimeter():Number
        {
            return sideLength * numSides;
        }

        public function getSumOfAngles():Number
        {
            if (numSides >= 3)
```

```
            {
                return ((numSides - 2) * 180);
            }
            else
            {
                return 0;
            }
        }

        public function describe():String
        {
            var desc:String = "Each side is " + sideLength + " pixels long.\n";
            desc += "Its area is " + getArea() + " pixels square.\n";
            desc += "Its perimeter is " + getPerimeter() + " pixels long.\n";
            desc += "The sum of all interior angles in this shape is " + getSumOfAngles() + "
degrees.\n";
            return desc;
        }
    }
}
```

Primeiro, a classe RegularPolygon declara duas propriedades comuns a todos os polígonos regulares: o comprimento de cada lado (a propriedade `sideLength`) e o número de lados (a propriedade `numSides`).

A classe RegularPolygon implementa a interface IPolygon e declara os quatro métodos da interface IPolygon. Ela implementa dois desses, os métodos `getPerimeter()` e `getSumOfAngles()`, usando fórmulas comuns.

Como a fórmula do método `getArea()` é diferente de forma para forma, a versão da classe base do método não pode incluir a lógica comum que pode ser herdada pelos métodos da subclasse. Em vez disso, ele simplesmente retorna um valor padrão 0 para indicar que a área não foi calculada. Para calcular a área de cada forma corretamente, as próprias subclasses da classe RegularPolygon precisam substituir o método `getArea()`.

O seguinte código da classe EquilateralTriangle mostra como o método `getArea()` é substituído:

```
package com.example.programmingas3.geometricshapes
{
    public class EquilateralTriangle extends RegularPolygon
    {
        public function EquilateralTriangle(len:Number = 100):void
        {
            super(len, 3);
        }

        public override function getArea():Number
        {
        // The formula is ((sideLength squared) * (square root of 3)) / 4.
        return ( (this.sideLength * this.sideLength) * Math.sqrt(3) ) / 4;
        }

        public override function describe():String
        {
                /* starts with the name of the shape, then delegates the rest
                of the description work to the RegularPolygon superclass */
        var desc:String = "This shape is an equilateral Triangle.\n";
        desc += super.describe();
        return desc;
        }
    }
}
```

A palavra-chave `override` indica que o método `EquilateralTriangle.getArea()` substitui intencionalmente o método `getArea()` da superclasse RegularPolygon. Quando o método `EquilateralTriangle.getArea()` é chamado, ele calcula a área usando a fórmula do código anterior, e o código no método `RegularPolygon.getArea()` nunca é executado.

Em comparação, a classe EquilateralTriangle não define sua própria versão do método `getPerimeter()`. Quando o método `EquilateralTriangle.getPerimeter()` é chamado, a chamada percorre a cadeia de herança e executa o código no método `getPerimeter()` da superclasse RegularPolygon.

O construtor `EquilateralTriangle()` usa a instrução `super()` para chamar explicitamente o construtor `RegularPolygon()` de sua superclasse. Se os dois construtores tivessem o mesmo conjunto de parâmetros, você poderia omitir completamente o construtor `EquilateralTriangle()` e o construtor `RegularPolygon()` seria executado. No entanto, o construtor `RegularPolygon()` precisa de um parâmetro extra, `numSides`. Portanto o construtor `EquilateralTriangle()` chama `super(len, 3)` que passa o parâmetro de entrada `len` e o valor 3 para indicar que o triângulo terá 3 lados.

O método `describe()` também usa a instrução `super()`, mas de uma maneira diferente, para chamar a versão da superclasse RegularPolygon do método `describe()`. O método `EquilateralTriangle.describe()` define primeiro a variável da string `desc` como uma instrução sobre o tipo da forma. Em seguida, ele obtém os resultados do método `RegularPolygon.describe()` chamando `super.describe()` e anexa esse resultado à string `desc`.

A classe Square não será descrita em detalhes aqui, mas é semelhante à classe EquilateralTriangle, fornecendo um construtor e suas próprias implementações dos métodos `getArea()` e `describe()`.

Polimorfismo e o método de fábrica

Um conjunto de classes que faz bom uso de interfaces e herança pode ser usado de muitas maneiras interessantes. Por exemplo, todas essas classes de formas descritas até agora implementam a interface IGeometricShape ou estendem uma superclasse que o faz. Portanto, se você definir uma variável como sendo uma ocorrência de IGeometricShape, não precisará saber se ela é realmente uma ocorrência das classes Circle ou Square para chamar seu método `describe()`.

O código a seguir mostra como isso funciona:

```
var myShape:IGeometricShape = new Circle(100);
trace(myShape.describe());
```

Quando `myShape.describe()` é chamado, ele executa o método `Circle.describe()`, porque embora a variável esteja definida como uma ocorrência da interface IGeometricShape, Circle é sua classe subjacente.

Este exemplo mostra o princípio do polimorfismo em ação: a mesma chamada de método exata resulta na execução de código diferente, dependendo da classe do objeto cujo método está sendo chamado.

O aplicativo GeometricShapes aplica esse tipo de polimorfismo com base em interface usando uma versão simplificada de um padrão de design conhecido como método de fábrica. O termo *método de fábrica* faz referência a uma função que retorna um objeto cujo tipo de dados subjacentes ou conteúdo pode ser diferente, dependendo do contexto.

A classe GeometricShapeFactory mostrada aqui define um método de fábrica denominado `createShape()`:

```
package com.example.programmingas3.geometricshapes
{
    public class GeometricShapeFactory
    {
        public static var currentShape:IGeometricShape;

        public static function createShape(shapeName:String,

len:Number):IGeometricShape
        {
            switch (shapeName)
            {
                case "Triangle":
                    return new EquilateralTriangle(len);

                case "Square":
                    return new Square(len);

                case "Circle":
                    return new Circle(len);
            }
            return null;
        }

        public static function describeShape(shapeType:String, shapeSize:Number):String
        {
            GeometricShapeFactory.currentShape =
                GeometricShapeFactory.createShape(shapeType, shapeSize);
            return GeometricShapeFactory.currentShape.describe();
        }
    }
}
```

O método de fábrica `createShape()` permite que construtores de subclasses de formas definam os detalhes das ocorrências que eles criam, enquanto retornam os novos objetos como ocorrências de IGeometricShape para que eles possam ser manipulados pelo aplicativo de maneira mais geral.

O método `describeShape()` no exemplo anterior mostra como um aplicativo pode usar o método de fábrica para obter uma referência genérica para um objeto mais específico. O aplicativo pode obter a descrição de um objeto Circle criado recentemente, como este:

```
GeometricShapeFactory.describeShape("Circle", 100);
```

Em seguida, o método `describeShape()` chama o método de fábrica `createShape()` com os mesmo parâmetros, armazenando o novo objeto Circle em uma variável estática denominada `currentShape` que foi digitada como um objeto IGeometricShape. Em seguida, o método `describe()` é chamado no objeto `currentShape`, e essa chamada é resolvida automaticamente para executar o método `Circle.describe()` retornando uma descrição detalhada do círculo.

Aprimoramento do aplicativo de amostra

O poder real das interfaces e da herança torna-se aparente quando você aprimora ou altera o aplicativo.

Suponha que você deseja adicionar uma nova forma, um pentágono, a este aplicativo de amostra. Você criaria uma nova classe Pentagon que estende a classe RegularPolygon e define suas próprias versões dos métodos `getArea()` e `describe()`. Em seguida, adicionaria uma nova opção de Pentagon à caixa de combinação na interface de usuário do aplicativo. Mas isso é tudo. A classe Pentagon obteria automaticamente a funcionalidade dos métodos `getPerimeter()` e `getSumOfAngles()` da classe RegularPolygon por herança. Como ela é herdada de uma classe que implementa a interface IGeometricShape, uma ocorrência Pentagon também pode ser tratada como uma ocorrência IGeometricShape. Isso significa que para adicionar um novo tipo de forma, não é preciso alterar a assinatura de nenhum dos métodos na classe GeometricShapeFactory (e, conseqüentemente, também não é preciso alterar nenhum código que use a classe GeometricShapeFactory).

Talvez você queira adicionar uma classe Pentagon ao exemplo de Geometric Shapes como um exercício, para ver como as interfaces e a herança podem facilitar a carga de trabalho da adição de novos recursos a um aplicativo.

Capítulo 6: Trabalho com datas e horas

Data e hora talvez não sejam tudo, mas normalmente são um fator chave nos aplicativos de software. O ActionScript 3.0 oferece formas úteis para gerenciar datas de calendário, horas e intervalos de tempo. Duas classes principais permitem gerenciar data e hora: a classe Date e a nova classe Timer do pacote flash.utils.

Noções básicas de data e hora

Introdução ao trabalho com datas e horas

Datas e horas são informações comuns usadas nos programas ActionScript. Por exemplo, você talvez precise saber o dia da semana atual ou medir o tempo que o usuário gasta em uma tela específica, entre muitas outras possibilidades. No ActionScript, você pode usar a classe Date para representar um único ponto temporal, incluindo informações de data e hora. Em uma ocorrência de Date estão incluídos valores de unidades individuais de data e hora, como ano, mês, data, dia da semana, hora, minutos, segundos, milissegundos e fuso horário. Para usos mais avançados, o ActionScript também inclui a classe Timer, que pode ser usada para realizar ações após um determinado atraso ou em intervalos repetidos.

Tarefas comuns de data e hora

Este capítulo descreve as seguintes tarefas comuns para trabalhar com informações de data e hora:

- Trabalho com os objetos de Date
- Obtenção da data e hora atuais
- Acesso a unidades individuais de data e hora (dias, anos, horas, minutos e assim por diante)
- Cálculos aritméticos com datas e horas
- Conversão entre fusos horários
- Realização de ações repetitivas
- Realização de ações após um intervalo de tempo definido

Conceitos e termos importantes

A lista de referência a seguir contém termos importantes usados neste capítulo:

- Horário UTC: Horário universal coordenado - o fuso horário de referência de “zero hora”. Todos os outros fusos horários são definidos como um número de horas (a mais ou a menos) em relação ao horário UTC.

Teste dos exemplos do capítulo

Talvez você queira testar algumas das listagens de código de exemplo por si próprio, durante a leitura deste capítulo. Como as listagens de código deste capítulo tratam principalmente de objetos de Date, o teste dos exemplos envolve a visualização dos valores das variáveis usadas nos exemplos, por meio da inserção de valores em uma ocorrência de campo de texto no palco ou pelo uso da função `trace()` para imprimir valores no painel Saída. Essas técnicas são descritas em detalhes em “Teste de listagens de código de exemplo dos capítulos” na página 36.

Gerenciamento de datas de calendário e horas

Todas as funções de gerenciamento de datas de calendário e horas do ActionScript 3.0 estão concentradas na classe Date de nível superior. A classe Date contém métodos e propriedades que permitem manipular datas e horas no horário UTC ou em um fuso horário específico local. O UTC é uma definição de horário padrão basicamente igual ao horário do Meridiano de Greenwich (GMT).

Criação de objetos de Date

A classe Date tem um dos métodos de construtor mais versáteis de todas as classes básicas. É possível chamá-la de quatro formas diferentes.

Primeiro, se não houver nenhum parâmetro, o construtor `Date()` retornará um objeto de Date que contém data e hora atuais, com base no seu fuso horário local. Eis um exemplo:

```
var now:Date = new Date();
```

Segundo, se houver um único parâmetro numérico, o construtor `Date()` irá tratá-lo como o número de milissegundos desde 1º de janeiro de 1970 e retornará um objeto de Date correspondente. Observe que o valor de milissegundos fornecido é tratado como o número de milissegundos desde 1º de janeiro de 1970, em UTC. No entanto, o objeto Date mostra valores no seu fuso horário local, a não ser que você use métodos específicos de UTC para recuperá-los e exibi-los. Se você criar um novo objeto Date usando um único parâmetro de milissegundos, leve em consideração a diferença de fuso entre o horário local e o UTC. As seguintes instruções criam um objeto Date definido como meia-noite do dia 1º de janeiro de 1970, em UTC:

```
var millisecondsPerDay:int = 1000 * 60 * 60 * 24;
// gets a Date one day after the start date of 1/1/1970
var startTime:Date = new Date(millisecondsPerDay);
```

Terceiro, é possível transmitir vários parâmetros numéricos para o construtor `Date()`. Esses parâmetros são tratados como ano, mês, dia, hora, minuto, segundo e milissegundo, respectivamente, e um objeto Date correspondente é retornado. Supõe-se que esses parâmetros de entrada estão no horário local, não no UTC. As seguintes instruções obtêm um objeto Date definido como meia-noite do dia 1º de janeiro de 2000, no horário local:

```
var millenium:Date = new Date(2000, 0, 1, 0, 0, 0, 0);
```

Quarto, é possível transmitir um parâmetro com uma única string para o construtor `Date()`. O construtor tentará analisar os componentes de data ou hora dessa string e retornará um objeto Date correspondente. Se você usar essa abordagem, é recomendado colocar o construtor `Date()` entre um bloco `try..catch` para detectar qualquer erro de análise. O construtor `Date()` aceita diversos formatos de string diferentes, conforme descrito no Guia de referência de componentes e linguagem do ActionScript 3.0. A instrução a seguir inicializa um novo objeto Date usando um valor de string:

```
var nextDay:Date = new Date("Mon May 1 2006 11:30:00 AM");
```

Se o construtor `Date()` não conseguir analisar o parâmetro de string, não lançará uma exceção. No entanto, o objeto Date resultante terá um valor de data inválido.

Obtenção de valores de unidade de tempo

Você pode extrair valores para várias unidades de tempo de um objeto Date usando propriedades ou métodos da classe Date. Cada propriedade a seguir fornece o valor de uma unidade de tempo no objeto Date:

- A propriedade `fullYear`
- A propriedade `month`, que está em um formato numérico que vai de 0 para janeiro até 11 para dezembro

- A propriedade `date`, que é o número do calendário do dia do mês, no intervalo de 1 a 31
- A propriedade `day`, que é o dia da semana em formato numérico, com 0 para domingo
- A propriedade `hours`, no intervalo de 0 a 23
- A propriedade `minutes`
- A propriedade `seconds`
- A propriedade `milliseconds`

  Na realidade, a classe Date fornece diversas maneiras para obter cada um desses valores. Você pode obter, por exemplo, o valor do mês de um objeto Date de quatro formas diferentes:

- A propriedade `month`
- O método `getMonth()`
- A propriedade `monthUTC`
- O método `getMonthUTC()`

  Os quatro métodos são praticamente iguais em termos de eficiência, de modo que você pode usar a abordagem mais adequada para seu aplicativo.

  Todas as propriedades listadas acima representam componentes do valor de data total. Por exemplo, a propriedade milliseconds nunca será maior do que 999, pois, quando atingir o número 1000, o valor da propriedade seconds aumentará em uma unidade e a propriedade milliseconds será zerada.

  Se desejar obter o valor do objeto Date em termos de milissegundos desde 1º de janeiro de 1970 (UTC), use o método `getTime()`. Seu correspondente, o método `setTime()`, permite alterar o valor de um objeto Date existente usando os milissegundos desde 1º de janeiro de 1970 (UTC).

Execução de cálculos aritméticos de data e hora

É possível realizar operações de adição e subtração em datas e horas com a classe Date. Os valores de Date são mantidos internamente em termos de milissegundos, de modo que você deve converter outros valores em milissegundos antes de adicioná-los ou subtraí-los dos objetos Date.

Se o seu aplicativo efetua vários cálculos aritméticos de data e hora, crie constantes que armazenam valores comuns de unidade de tempo em milissegundos, conforme mostrado a seguir:

```
public static const millisecondsPerMinute:int = 1000 * 60;
public static const millisecondsPerHour:int = 1000 * 60 * 60;
public static const millisecondsPerDay:int = 1000 * 60 * 60 * 24;
```

Agora é fácil efetuar cálculos aritméticos de data usando unidades de tempo padrão. O código a seguir define um valor de data como uma hora do horário atual usando os métodos `getTime()` e `setTime()`:

```
var oneHourFromNow:Date = new Date();
oneHourFromNow.setTime(oneHourFromNow.getTime() + millisecondsPerHour);
```

Outra maneira de definir um valor de data é criar um novo objeto Date usando um único parâmetro de milissegundos. Por exemplo, o código a seguir adiciona 30 dias a uma data para calcular outra:

```
// sets the invoice date to today's date
var invoiceDate:Date = new Date();

// adds 30 days to get the due date
var dueDate:Date = new Date(invoiceDate.getTime() + (30 * millisecondsPerDay));
```

Em seguida, a constante `millisecondsPerDay` é multiplicada por 30 para representar o período de 30 dias e o resultado é adicionado ao valor `invoiceDate` e usado para definir o valor `dueDate`.

Conversão entre fusos horários

Cálculos aritméticos de data e hora são úteis quando você deseja converter datas de um fuso horário para outro. O método `getTimezoneOffset()` serve para isso, pois retorna o valor como a diferença de minutos entre o fuso horário do objeto Date e o UTC. Ele retorna um valor em minutos porque nem todos os fusos horários são definidos como incrementos de horas inteiras - alguns têm diferenças de meia hora em relação às zonas vizinhas.

O exemplo a seguir usa a diferença de fuso para converter uma data do horário local em UTC. Para fazer a conversão, primeiro calcula-se o valor do fuso horário em milissegundos e, depois, ajusta-se o valor de Date de acordo com o resultado:

```
// creates a Date in local time
var nextDay:Date = new Date("Mon May 1 2006 11:30:00 AM");

// converts the Date to UTC by adding or subtracting the time zone offset
var offsetMilliseconds:Number = nextDay.getTimezoneOffset() * 60 * 1000;
nextDay.setTime(nextDay.getTime() + offsetMilliseconds);
```

Controle de intervalos de tempo

Ao desenvolver aplicativos usando o Adobe Flash CS4 Professional, você tem acesso à linha do tempo, que oferece uma progressão uniforme quadro a quadro do seu aplicativo. No entanto, em projetos exclusivamente do ActionScript, é necessário usar outros mecanismos de controle de tempo.

Loops versus timers

Em algumas linguagens de programação, você deve desenvolver seus próprios esquemas de controle de tempo usando instruções de loop como `for` ou `do..while`.

As instruções de loop geralmente são executadas conforme permitido pela máquina local, ou seja, o aplicativo é executado mais rapidamente em algumas máquinas e mais lentamente em outras. Se o seu aplicativo precisa de um intervalo de tempo consistente, associe-o a um calendário ou relógio real. Muitos aplicativos, como jogos, animações e controladores em tempo real, precisam de mecanismos de tempo regulares que se adaptem a cada máquina.

A classe Timer do ActionScript 3.0 fornece uma solução incrível. Usando o modelo de eventos do ActionScript 3.0, a classe Timer envia eventos de tempo sempre que um intervalo especificado é atingido.

A classe Timer

O melhor modo de manipular funções de controle de tempo no ActionScript 3.0 é usar a classe Timer (flash.utils.Timer) para enviar eventos sempre que um intervalo for atingido.

Para iniciar um timer, você precisa criar primeiro uma ocorrência da classe Timer, informando com que freqüência um evento de tempo deve ser gerado e quantas vezes isso deve ser feito antes de parar.

Por exemplo, o código a seguir cria uma ocorrência de Timer que envia um evento por segundo e continua durante 60 segundos:

```
var oneMinuteTimer:Timer = new Timer(1000, 60);
```

O objeto Timer envia um objeto TimerEvent sempre que o intervalo especificado é atingido. Um tipo de evento do objeto TimerEvent é `timer` (definido pela constante `TimerEvent.TIMER`). Um objeto TimerEvent contém as mesmas propriedades de um objeto Event padrão.

Se a ocorrência de Timer for definida como um número fixo de intervalos, um evento `timerComplete` (definido pela constante `TimerEvent.TIMER_COMPLETE`) também será enviado quando o intervalo final for atingido.

Veja um pequeno aplicativo de exemplo que mostra a classe Timer em ação:

```
package
{
    import flash.display.Sprite;
    import flash.events.TimerEvent;
    import flash.utils.Timer;

    public class ShortTimer extends Sprite
    {
        public function ShortTimer()
        {
            // creates a new five-second Timer
            var minuteTimer:Timer = new Timer(1000, 5);

            // designates listeners for the interval and completion events
            minuteTimer.addEventListener(TimerEvent.TIMER, onTick);
            minuteTimer.addEventListener(TimerEvent.TIMER_COMPLETE, onTimerComplete);

            // starts the timer ticking
            minuteTimer.start();
        }

        public function onTick(event:TimerEvent):void
        {
            // displays the tick count so far
            // The target of this event is the Timer instance itself.
            trace("tick " + event.target.currentCount);
        }

        public function onTimerComplete(event:TimerEvent):void
        {
            trace("Time's Up!");
        }
    }
}
```

Ao ser criada, a classe ShortTimer cria uma ocorrência de Timer que será acionada uma vez por segundo durante cinco segundos. Em seguida, são adicionados dois ouvintes ao timer: um que ouve cada acionamento e outro que ouve o evento `timerComplete`.

Em seguida, começa o acionamento do timer e, a partir desse ponto, o método `onTick()` é executado em intervalos de um segundo.

O método `onTick()` simplesmente exibe a contagem de acionamentos atual. Depois de cinco segundos, o método `onTimerComplete()` é executado, informando que o tempo acabou.

Ao executar este exemplo, você deve ver as seguintes linhas na janela do console ou rastreamento, na velocidade de uma linha por segundo:

```
tick 1
tick 2
tick 3
tick 4
tick 5
Time's Up!
```

Funções de controle de tempo do pacote flash.utils

O ActionScript 3.0 contém várias funções de controle de tempo similares às que estavam disponíveis no ActionScript 2.0. Essas funções são fornecidas no nível do pacote flash.utils e funcionam da mesma maneira como no ActionScript 2.0.

Função	Descrição
`clearInterval(id:uint):void`	Cancela uma chamada `setInterval()` especificada.
`clearTimeout(id:uint):void`	Cancela uma chamada `setTimeout()` especificada.
`getTimer():int`	Retorna o número de milissegundos desde que o Adobe® Flash® Player ou o Adobe® AIR™ foi inicializado.
`setInterval(closure:Function, delay:Number, ... arguments):uint`	Executa uma função em um intervalo especificado (em milissegundos).
`setTimeout(closure:Function, delay:Number, ... arguments):uint`	Executa uma função especificada após um atraso especificado (em milissegundos).

Essas funções estão incluídas no ActionScript 3.0 por questões de compatibilidade com versões anteriores. A Adobe não recomenda utilizá-las em novos aplicativos do ActionScript 3.0. Em geral, é mais fácil e mais eficiente usar a classe Timer em seus aplicativos.

Exemplo: relógio analógico simples

O exemplo de um relógio analógico simples ilustra dois conceitos de data e hora discutidos neste capítulo:

- Obtenção de data e hora atuais e extração de valores para horas, minutos e segundos
- Utilização de Timer para definir o ritmo de um aplicativo

  Para obter os arquivos de aplicativo desse exemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_br. Os arquivos do aplicativo SimpleClock estão localizados na pasta Amostras/SimpleClock. O aplicativo consiste nos seguintes arquivos:

Arquivo	Descrição
SimpleClockApp.mxml ou SimpleClockApp.fla	O arquivo principal do aplicativo no Flash (FLA) ou no Flex (MXML).
com/example/programmingas3/simpleclock/SimpleClock.as	O arquivo de aplicativo principal.
com/example/programmingas3/simpleclock/AnalogClockFace.as	Desenha a superfície de um relógio redondo e os ponteiros de horas, minutos e segundos com base na hora.

Definição da classe SimpleClock

O exemplo do relógio é simples, mas é uma boa idéia para organizar até mesmo aplicativos simples que podem ser expandidos com facilidade no futuro. Para tanto, o aplicativo SimpleClock usa a classe SimpleClock para manipular as tarefas de inicialização e controle de tempo e usa outra classe chamada AnalogClockFace para exibir realmente a hora.

Veja o código que define e inicializa a classe SimpleClock (observe que, na versão Flash, SimpleClock estende a classe Sprite):

```
public class SimpleClock extends UIComponent
{
    /**
     * The time display component.
     */
    private var face:AnalogClockFace;

    /**
     * The Timer that acts like a heartbeat for the application.
     */
    private var ticker:Timer;
```

A classe tem duas propriedades importantes:

- A propriedade `face`, que é uma ocorrência da classe AnalogClockFace
- A propriedade `ticker`, que é uma ocorrência da classe Timer

  A classe SimpleClock usa um construtor padrão. O método `initClock()` realiza o trabalho de configuração real, criando a superfície do relógio e iniciando o acionamento da ocorrência de Timer.

Criação da superfície do relógio

As próximas linhas do código SimpleClock criam a superfície do relógio que é usada para exibir a hora:

```
    /**
     * Sets up a SimpleClock instance.
     */
    public function initClock(faceSize:Number = 200)
    {
        // creates the clock face and adds it to the display list
        face = new AnalogClockFace(Math.max(20, faceSize));
        face.init();
        addChild(face);

        // draws the initial clock display
        face.draw();
```

O tamanho da superfície pode ser transmitido para o método `initClock()`. Se nenhum valor de `faceSize` for transmitido, será usado o tamanho padrão de 200 pixels.

Em seguida, o aplicativo inicializa a superfície e a adiciona à lista de exibição usando o método `addChild()` herdado da classe DisplayObject. O método `AnalogClockFace.draw()` é chamado para exibir a superfície do relógio uma vez, mostrando a hora atual.

Início do timer

Assim que a superfície do relógio é criada, o método `initClock()` configura um timer:

```
// creates a Timer that fires an event once per second
ticker = new Timer(1000);

// designates the onTick() method to handle Timer events
ticker.addEventListener(TimerEvent.TIMER, onTick);

// starts the clock ticking
ticker.start();
```

Primeiro, esse método percorre uma ocorrência de Timer que enviará um evento por segundo (a cada 1000 milissegundos). Como nenhum outro parâmetro `repeatCount` é transmitido para o construtor `Timer()`, o Timer será repetido indefinidamente.

O método `SimpleClock.onTick()` será executado uma vez por segundo quando o evento `timer` for recebido:

```
public function onTick(event:TimerEvent):void
{
    // updates the clock display
    face.draw();
}
```

O método `AnalogClockFace.draw()` simplesmente desenha a superfície e os ponteiros do relógio.

Exibição do horário atual

A maior parte do código na classe AnalogClockFace envolve a configuração dos elementos de exibição da superfície do relógio. Ao ser inicializado, o AnalogClockFace desenha um contorno circular, coloca um rótulo de texto numérico em cada marcação de hora e cria três objetos Shape, um para cada ponteiro do relógio (horas, minutos e segundos).

Assim que o aplicativo SimpleClock é executado, o método `AnalogClockFace.draw()` é chamado a cada segundo do seguinte modo:

```
/**
 * Called by the parent container when the display is being drawn.
 */
public override function draw():void
{
    // stores the current date and time in an instance variable
    currentTime = new Date();
    showTime(currentTime);
}
```

Este método salva a hora atual em uma variável para que o horário não mude no meio do desenho dos ponteiros do relógio. Em seguida, o método `showTime()` é chamado para exibir os ponteiros do seguinte modo:

```
/**
 * Displays the given Date/Time in that good old analog clock style.
 */
public function showTime(time:Date):void
{
    // gets the time values
    var seconds:uint = time.getSeconds();
    var minutes:uint = time.getMinutes();
    var hours:uint = time.getHours();

    // multiplies by 6 to get degrees
    this.secondHand.rotation = 180 + (seconds * 6);
    this.minuteHand.rotation = 180 + (minutes * 6);

    // Multiply by 30 to get basic degrees, then
    // add up to 29.5 degrees (59 * 0.5)
    // to account for the minutes.
    this.hourHand.rotation = 180 + (hours * 30) + (minutes * 0.5);
}
```

Primeiro, este método extrai os valores para horas, minutos e segundos do horário atual. Depois, esses valores são usados para calcular o ângulo de cada ponteiro. Como faz uma rotação completa em 60 segundos, o ponteiro dos segundos gira 6 graus a cada segundo (360/60). O ponteiro dos minutos gira do mesmo modo em cada minuto.

O ponteiro das horas também é atualizado a cada minuto e pode avançar um pouco à medida que os minutos passam. Ele gira 30 graus por hora (360/12), mas também gira meio grau por minuto (30 graus dividido por 60 minutos).

Capítulo 7: Trabalho com strings

A classe String contém métodos que permitem trabalhar com strings de texto. Strings são importantes ao trabalhar com muitos objetos. Os métodos descritos neste capítulo são úteis para trabalhar com strings usadas em objetos, como TextField, StaticText, XML, ContextMenu e FileReference.

Strings são seqüências de caracteres. O ActionScript 3.0 oferece suporte a caracteres ASCII e Unicode.

Noções básicas de strings

Introdução ao trabalho com strings

Em linguagem de programação, uma string é um valor de texto, uma seqüência de letras, números ou outros caracteres combinados em um único valor. Por exemplo, esta linha de código cria uma variável com o tipo de dados String e atribui um valor de string literal àquela variável:

```
var albumName:String = "Three for the money";
```

Conforme mostrado nesse exemplo, no ActionScript é possível denotar um valor de string incluindo o texto entre aspas duplas ou simples. Estes são vários exemplos adicionais de strings:

```
"Hello"
"555-7649"
"http://www.adobe.com/"
```

Ao manipular uma parte de texto no ActionScript, você está trabalhando com um valor de string. A classe String do ActionScript é o tipo de dados que pode ser usado para trabalhar com valores de texto. As ocorrências de strings são usadas com freqüência para propriedades, parâmetros de métodos e assim por diante em muitas outras classes ActionScript.

Tarefas comuns do trabalho com strings

As seguintes são tarefas comuns relacionadas a strings que são exploradas neste capítulo:

- Criação de objetos String
- Trabalho com caracteres especiais, como retorno de carro, tabulação e caracteres que não fazem parte do teclado.
- Medição do comprimento da string
- Isolamento de caracteres individuais em uma string
- Junção de strings
- Comparação de strings
- Localização, extração e substituição de partes de uma string
- Colocação de strings em maiúsculas ou minúsculas

Conceitos e termos importantes

A lista de referência a seguir contém termos importantes usados neste capítulo:

- ASCII: Um sistema que representa caracteres de texto e símbolos em programas de computador. O sistema ASCII oferece suporte ao alfabeto inglês de 26 letras, mais um conjunto limitado de caracteres adicionais.
- Caractere: A menor unidade de dados de texto (uma única letra ou símbolo).
- Concatenação: Junção de vários valores de strings com a adição de um ao final do outro, criando um novo valor de string.
- string vazia: Uma string que não contém nenhum texto, espaço em branco ou outros caracteres, escrita como `""`. Um valor de string vazia é diferente de uma variável String com um valor nulo, uma variável String nula é uma variável que não tem uma ocorrência de String atribuída a ela, enquanto que uma string vazia tem uma ocorrência com um valor que não contém nenhum caractere.
- String: Um valor textual (seqüência de caracteres).
- string literal (ou "literal de string"): Um valor de string escrito explicitamente em código como um valor de texto incluído entre aspas duplas ou aspas simples.
- Substring: Uma string que faz parte de outra string.
- Unicode: Um sistema padrão que representa caracteres de texto e símbolos em programas de computador. O sistema Unicode permite usar qualquer caractere em qualquer sistema de gravação.

Teste dos exemplos do capítulo

Talvez você queira testar algumas das listagens de código de exemplo por si próprio, durante a leitura deste capítulo. Como as listagens de código deste capítulo tratam principalmente de manipulação de texto, o teste dos exemplos envolve a exibição dos valores das variáveis usadas nos exemplos, seja escrevendo valores em uma ocorrência de campo de texto no Palco ou usando a função `trace()` para imprimir valores no painel Saída. Essas técnicas são descritas em detalhes em "Teste de listagens de código de exemplo dos capítulos" na página 36.

Criação de strings

A classe String é usada para representar dados de string (textuais) no ActionScript 3.0. As strings do ActionScript oferecem suporte a caracteres ASCII e Unicode. A maneira mais simples de criar uma string é usar uma string literal. Para declarar um string literal, use caracteres de aspas duplas retas (`"`) ou de aspas simples (`'`). Por exemplo, as duas strings a seguir são equivalentes:

```
var str1:String = "hello";
var str2:String = 'hello';
```

Também é possível declarar uma string usando o operador `new`, da seguinte maneira:

```
var str1:String = new String("hello");
var str2:String = new String(str1);
var str3:String = new String();       // str3 == ""
```

As duas strings a seguir são equivalentes:

```
var str1:String = "hello";
var str2:String = new String("hello");
```

Para usar aspas simples (') em uma string literal definida com delimitadores de aspas simples ('), use o caractere de escape barra invertida (\). De maneira semelhante, para usar aspas duplas (") em uma string literal definida com delimitadores de aspas duplas ("), use o caractere de escape barra invertida (\). As duas strings a seguir são equivalentes:

```
var str1:String = "That's \"A-OK\"";
var str2:String = 'That\'s "A-OK"';
```

É possível escolher o uso de aspas simples ou de aspas duplas com base em quaisquer aspas simples ou duplas existentes em uma string literal, como no exemplo a seguir:

```
var str1:String = "ActionScript <span class='heavy'>3.0</span>";
var str2:String = '<item id="155">banana</item>';
```

Lembre-se de que o ActionScript diferencia aspas simples retas (') e aspas simples esquerda ou direita (' ou '). O mesmo é verdadeiro para aspas duplas. Use aspas retas para delinear strings literais. Ao colar texto de outra origem no ActionScript, use os caracteres corretos.

Conforme mostrado na tabela a seguir, é possível usar o caractere de escape de barra invertida (\) para definir outros caracteres em strings literais:

Seqüência de escape	Caractere
\b	Backspace
\f	Feed de formulário
\n	Nova linha
\r	Retorno de carro
\t	Tabulação
\u***nnnn***	O caractere Unicode com o código de caractere especificado pelo número hexadecimal *nnnn*; por exemplo, \u263a é o caractere smiley.
\\x***nn***	O caractere ASCII com o código de caractere especificado pelo número hexadecimal *nn*
\'	Aspas simples
\"	Aspas duplas
\\	Caractere de barra invertida simples

A propriedade length

Cada string tem uma propriedade `length` que é igual ao número de caracteres da string:

```
var str:String = "Adobe";
trace(str.length);            // output: 5
```

Uma string vazia e uma string nula têm um comprimento de 0, conforme mostrado no exemplo a seguir:

```
var str1:String = new String();
trace(str1.length);           // output: 0

str2:String = '';
trace(str2.length);           // output: 0
```

Trabalho com caracteres em strings

Cada caractere em uma string tem uma posição de índice na string (um inteiro). A posição do índice do primeiro caractere é 0. Por exemplo, na seguinte string, o caractere `y` está na posição 0 e o caractere `w` está na posição 5:

```
"yellow"
```

É possível examinar caracteres individuais em várias posições em uma string usando os métodos `charAt()` e `charCodeAt()`, como neste exemplo:

```
var str:String = "hello world!";
for (var i:int = 0; i < str.length; i++)
{
    trace(str.charAt(i), "-", str.charCodeAt(i));
}
```

Quando esse código é executado, a seguinte saída é produzida:

```
h - 104
e - 101
l - 108
l - 108
o - 111
- 32
w - 119
o - 111
r - 114
l - 108
d - 100
! - 33
```

Também é possível usar códigos de caracteres para definir uma string usando o método `fromCharCode()`, como no exemplo a seguir:

```
var myStr:String = String.fromCharCode(104,101,108,108,111,32,119,111,114,108,100,33);
        // Sets myStr to "hello world!"
```

Comparação de strings

É possível usar os seguintes operadores para comparar strings: `<`, `<=`, `!=`, `==`, `=>` e `>`. Esses operadores podem ser usados com declarações condicionais, como `if` e`while`, como no exemplo a seguir:

```
var str1:String = "Apple";
var str2:String = "apple";
if (str1 < str2)
{
    trace("A < a, B < b, C < c, ...");
}
```

Ao usar esses operadores com strings, o ActionScript considera o valor de código de cada caractere na string, comparando da esquerda para a direita, como no seguinte exemplo:

```
trace("A" < "B"); // true
trace("A" < "a"); // true
trace("Ab" < "az"); // true
trace("abc" < "abza"); // true
```

Use os operadores == e != para comparar strings umas com as outras e para comparar strings com outros tipos de objetos, conforme mostrado no exemplo a seguir:

```
var str1:String = "1";
var str1b:String = "1";
var str2:String = "2";
trace(str1 == str1b); // true
trace(str1 == str2); // false
var total:uint = 1;
trace(str1 == total); // true
```

Obtenção de representações de strings de outros objetos

É possível obter uma representação de String de qualquer tipo de objeto. Todos os objetos têm um método `toString()` para essa finalidade:

```
var n:Number = 99.47;
var str:String = n.toString();
    // str == "99.47"
```

Ao usar o operador de concatenação + com uma combinação de objetos String que não são strings, não é necessário usar o método `toString()`. Para obter detalhes sobre concatenação, consulte a seção a seguir.

A função global `String()` retorna o mesmo valor para um determinado objeto que o valor retornado pelo objeto de chamada do método `toString()`.

Concatenação de strings

A concatenação de strings significa utilizar duas strings e uni-las seqüencialmente em uma. Por exemplo, é possível usar o operador + para concatenar duas strings:

```
var str1:String = "green";
var str2:String = "ish";
var str3:String = str1 + str2; // str3 == "greenish"
```

Também é possível usar o operador += para produzir o mesmo resultado, conforme mostrado no exemplo a seguir:

```
var str:String = "green";
str += "ish"; // str == "greenish"
```

Além disso, a classe String inclui um método `concat()` que pode ser usado da seguinte maneira:

```
var str1:String = "Bonjour";
var str2:String = "from";
var str3:String = "Paris";
var str4:String = str1.concat(" ", str2, " ", str3);
// str4 == "Bonjour from Paris"
```

Se você usar o operador + (ou o operador +=) com o objeto String e um objeto *não*-String, o ActionScript converterá automaticamente o objeto não-String em um objeto String para avaliar a expressão, conforme mostrado neste exemplo:

```
var str:String = "Area = ";
var area:Number = Math.PI * Math.pow(3, 2);
str = str + area; // str == "Area = 28.274333882308138"
```

No entanto é possível usar parênteses para agrupamento para fornecer contexto para o operador +, conforme mostrado no exemplo a seguir:

```
trace("Total: $" + 4.55 + 1.45); // output: Total: $4.551.45
trace("Total: $" + (4.55 + 1.45)); // output: Total: $6
```

Localização de substrings e padrões em strings

Substrings são strings dentro de uma string. Por exemplo, a string `"abc"` tem as seguinte substrings: `""`, `"a"`, `"ab"`, `"abc"`, `"b"`, `"bc"`, `"c"`. É possível usar os métodos do ActionScript para localizar substrings de uma string.

Padrões são definidos no ActionScript por strings ou por expressões regulares. Por exemplo, a expressão regular a seguir define um padrão específico, as letras A, B e C seguidas por um caractere de dígito (as barras são delimitadores de expressões regulares):

```
/ABC\d/
```

O ActionScript inclui métodos para localização de padrões em strings e para substituir correspondências localizadas com substrings de substituição. Esses métodos são descritos nas seções a seguir.

Expressões regulares podem definir padrões intrincados. Para obter mais informações, consulte "Uso de expressões regulares" na página 209.

Localização de uma substring pela posição do caractere

Os métodos `substr()` e `substring()` são semelhantes. Os dois retornam uma substring de uma string. Os dois utilizam dois parâmetros. Nos dois métodos, o primeiro parâmetro é a posição do caractere inicial na string fornecida. No entanto, no método `substr()`, o segundo parâmetro é o *length* da substring a ser retornada e, no método `substring()`, o segundo parâmetro é a posição do caractere no *end* da substring (que não é incluída na string retornada). Este exemplo mostra a diferença entre esses dois métodos:

```
var str:String = "Hello from Paris, Texas!!!";
trace(str.substr(11,15)); // output: Paris, Texas!!!
trace(str.substring(11,15)); // output: Pari
```

O método `slice()` funciona de maneira semelhante ao método `substring()`. Quando recebe dois inteiros não negativos como parâmetros, ele funciona exatamente da mesma forma. No entanto o método `slice()` pode utilizar inteiros negativos como parâmetros e nesse caso a posição do caractere é utilizada a partir do final da string, conforme mostrado no exemplo a seguir:

```
var str:String = "Hello from Paris, Texas!!!";
trace(str.slice(11,15)); // output: Pari
trace(str.slice(-3,-1)); // output: !!
trace(str.slice(-3,26)); // output: !!!
trace(str.slice(-3,str.length)); // output: !!!
trace(str.slice(-8,-3)); // output: Texas
```

É possível combinar inteiros não negativos e negativos como os parâmetros do método `slice()`.

Localização da posição do caractere de uma substring correspondente

É possível usar os métodos `indexOf()` e `lastIndexOf()` para localizar substrings dentro de uma string, conforme mostrado no exemplo a seguir:

```
var str:String = "The moon, the stars, the sea, the land";
trace(str.indexOf("the")); // output: 10
```

Observe que o método `indexOf()` faz distinção entre maiúsculas e minúsculas.

É possível especificar um segundo parâmetro para indicar a posição do índice na string na qual iniciar a pesquisa, da seguinte maneira:

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.indexOf("the", 11)); // output: 21
```

O método `lastIndexOf()` localiza a última ocorrência de uma substring na string:

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.lastIndexOf("the")); // output: 30
```

Se você incluir um segundo parâmetro com o método `lastIndexOf()`, a pesquisa será conduzida naquela posição do índice na string operando retroativamente (da direita para a esquerda):

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.lastIndexOf("the", 29)); // output: 21
```

Criação de uma matriz de substrings segmentadas por um delimitador

É possível usar o método `split()` para criar uma matriz de substrings que é dividida com base em um delimitador. Por exemplo, é possível segmentar uma string delimitada por vírgula ou por tabulação em várias strings.

O exemplo a seguir mostra como dividir uma matriz em substrings com o caractere e comercial (&) como o delimitador:

```
var queryStr:String = "first=joe&last=cheng&title=manager&StartDate=3/6/65";
var params:Array = queryStr.split("&", 2); // params == ["first=joe","last=cheng"]
```

O segundo parâmetro do método `split()`, que é opcional, define o tamanho máximo da matriz retornada.

Também é possível usar uma expressão regular como o caractere delimitador:

```
var str:String = "Give me\t5."
var a:Array = str.split(/\s+/); // a == ["Give","me","5."]
```

Para obter mais informações, consulte "Uso de expressões regulares" na página 209 e a Referência dos componentes e da linguagem do ActionScript 3.0.

Localização de padrões em strings e substituição de substrings

A classe String inclui os seguintes métodos para trabalhar com padrões em strings:

- Use os métodos `match()` e `search()` para localizar substrings que correspondem a um padrão.
- Use o método `replace()` para localizar substrings que correspondem a um padrão e substituí-las por uma substring especificada.

Esses métodos são descritos nas seções a seguir.

É possível usar strings ou expressões regulares para definir padrões usados nesses métodos. Para obter mais informações sobre expressões regulares, consulte "Uso de expressões regulares" na página 209.

Localização de substrings correspondentes

O método `search()` retorna a posição do índice da primeira substring que corresponde a um determinado padrão, conforme mostrado neste exemplo:

```
var str:String = "The more the merrier.";
// (This search is case-sensitive.)
trace(str.search("the")); // output: 9
```

Também é possível usar expressões regulares para definir o padrão a ser correspondido, conforme mostrado no exemplo a seguir:

```
var pattern:RegExp = /the/i;
var str:String = "The more the merrier.";
trace(str.search(pattern)); // 0
```

A saída do método `trace()` é 0, porque o primeiro caractere na string é a posição 0 do índice. O sinalizador `i` é definido na expressão regular, portanto a pesquisa não faz distinção entre maiúsculas e minúsculas.

O método `search()` localiza apenas uma correspondência e retorna sua posição inicial no índice, mesmo que o sinalizador `g` (global) esteja definido na expressão regular.

O exemplo a seguir mostra uma expressão regular mais intrincada que corresponde a uma string entre aspas duplas:

```
var pattern:RegExp = /"[^"]*"/;
var str:String = "The \"more\" the merrier.";
trace(str.search(pattern)); // output: 4

str = "The \"more the merrier.";
trace(str.search(pattern)); // output: -1
// (Indicates no match, since there is no closing double quotation mark.)
```

O método `match()` funciona de maneira semelhante. Ele pesquisa uma substring correspondente. No entanto quando você usa o sinalizador global em um padrão de expressão regular, como no exemplo a seguir, `match()` retorna uma matriz de substrings correspondentes:

```
var str:String = "bob@example.com, omar@example.org";
var pattern:RegExp = /\w*@\w*\.[org|com]+/g;
var results:Array = str.match(pattern);
```

A matriz `results` é definida como o seguinte:

```
["bob@example.com","omar@example.org"]
```

Para obter mais informações sobre expressões regulares, consulte “Uso de expressões regulares” na página 209“Uso de expressões regulares” na página 209.

Substituição de substrings correspondentes

É possível usar o método `replace()` para pesquisar um padrão especificado em uma string e substituir correspondências pela string de substituição especificada, conforme mostrado no exemplo a seguir:

```
var str:String = "She sells seashells by the seashore.";
var pattern:RegExp = /sh/gi;
trace(str.replace(pattern, "sch"));
    //sche sells seaschells by the seaschore.
```

Observe que, neste exemplo, as strings correspondentes não fazem distinção entre maiúsculas e minúsculas porque o sinalizador `i` (`ignoreCase`) está definido na expressão regular, e várias correspondências são substituídas porque o sinalizador `g` (`global`) está definido. Para obter mais informações, consulte “Uso de expressões regulares” na página 209.

É possível incluir os seguintes códigos de substituição `$`na string de substituição. O texto de substituição mostrado na tabela a seguir é inserido no lugar do código de substituição `$`:

$ Code	Texto de substituição
`$$`	$
`$&`	A substring correspondida.
`$``	A parte da string que precede a substring correspondida. Esse código usa o caractere aspas simples retas esquerdas (`), não aspas simples retas (') ou aspas simples inglesas esquerdas (').
`$'`	A parte da string que segue a substring correspondida. Esse código usa as aspas simples retas (').
$n	A *n*ª correspondência de grupo entre parênteses capturado, em que n é um único dígito, 1 a 9, e $n não é seguido por um dígito decimal.
$nn	A *nn*ª correspondência de grupo entre parênteses capturado, em que *nn* é um número decimal de dois dígitos (01 a 99). Se a *nn*ª captura estiver indefinida, o texto de substituição será uma string vazia.

Por exemplo, veja a seguir o uso dos códigos de substituição `$2` e `$1` que representam o primeiro e o segundo grupos de captura correspondidos:

```
var str:String = "flip-flop";
var pattern:RegExp = /(\w+)-(\w+)/g;
trace(str.replace(pattern, "$2-$1")); // flop-flip
```

Também é possível usar uma função como o segundo parâmetro do método `replace()`. O texto correspondente é substituído pelo valor retornado da função.

```
var str:String = "Now only $9.95!";
var price:RegExp = /\$([\d,]+.\d+)+/i;
trace(str.replace(price, usdToEuro));

function usdToEuro(matchedSubstring:String,  capturedMatch1:String,  index:int,
str:String):String
{
    var usd:String = capturedMatch1;
    usd = usd.replace(",", "");
    var exchangeRate:Number = 0.853690;
    var euro:Number = parseFloat(usd) * exchangeRate;
    const euroSymbol:String = String.fromCharCode(8364);
    return euro.toFixed(2) + " " + euroSymbol;
}
```

Ao usar uma função como o segundo parâmetro do método `replace()`, os seguintes argumentos são passados para a função:

- A parte correspondente da string.
- Quaisquer correspondências de grupos entre parênteses capturadas. O número de argumentos transmitidos dessa maneira irá variar dependendo do número de correspondências parentéticas. É possível determinar o número de correspondências entre parênteses verificando `arguments.length - 3` no código da função.
- A posição de índice na string em que a correspondência começa.
- A string completa.

Conversão de strings entre maiúsculas e minúsculas

Conforme mostrado no exemplo a seguir, os métodos `toLowerCase()` e `toUpperCase()` convertem caracteres alfabéticos da string em minúsculas e maiúsculas, respectivamente:

```
var str:String = "Dr. Bob Roberts, #9."
trace(str.toLowerCase()); // dr. bob roberts, #9.
trace(str.toUpperCase()); // DR. BOB ROBERTS, #9.
```

Após a execução desses métodos, a string de origem permanece inalterada. Para transformar a string de origem, use o seguinte código:

```
str = str.toUpperCase();
```

Esses métodos funcionam com caracteres estendidos, não simplesmente com a a z e A a Z:

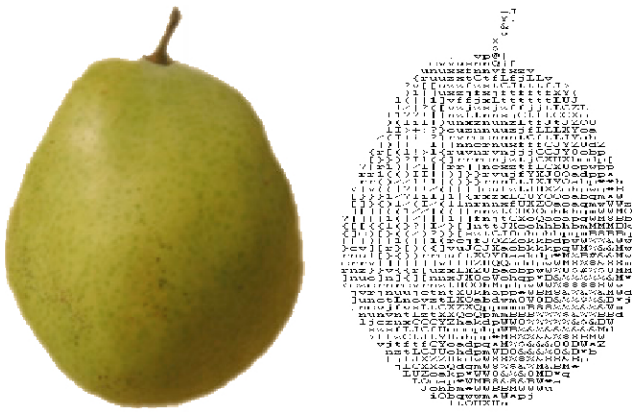
```
var str:String = "José Barça";
trace(str.toUpperCase(), str.toLowerCase()); // JOSÉ BARÇA josé barça
```

Exemplo: arte ASCII

Este exemplo de Arte ASCII mostra vários recursos para trabalhar com a classe String no ActionScript 3.0, incluindo o seguinte:

- O método `split()` da classe String é usado para extrair valores de uma string delimitada por caracteres (informações de imagem em um arquivo de texto delimitado por tabulação).
- Várias técnicas de manipulação de string, incluindo `split()`, concatenação e extração de uma parte da string usando `substring()` e `substr()`, são usadas para colocar a primeira letra de cada palavra em maiúscula nos títulos de imagem.
- O método `getCharAt()` é usado para obter um único caractere de uma string (para determinar o caractere ASCII correspondente a um valor de bitmap em escala de cinza).
- A concatenação de string é usada para criar a representação de arte ASCII de uma imagem com um caractere de cada vez.

O termo *arte ASCII* faz referência a representações de texto de uma imagem, na qual uma grade de caracteres de fonte monoespaçada, como caracteres Courier New, plotam a imagem. A imagem a seguir mostra um exemplo de arte ASCII produzida pelo aplicativo:



A versão da arte ASCII do gráfico é mostrada à direita.

Para obter os arquivos de aplicativo deste exemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_br. Os arquivos do aplicativo ASCIIArt podem ser encontrados na pasta Amostras/AsciiArt. O aplicativo consiste nos seguintes arquivos:

Arquivo	Descrição
AsciiArtApp.mxml ou AsciiArtApp.fla	O arquivo principal do aplicativo no Flash (FLA) ou no Flex (MXML).
com/example/programmingas3/asciiArt/AsciiArtBuilder.as	A classe que fornece a funcionalidade principal do aplicativo, incluindo a extração de metadados da imagem de um arquivo de texto, carregamento de imagens e gerenciamento do processo de conversão de imagem para texto.
com/example/programmingas3/asciiArt/BitmapToAsciiConverter.as	A classe que fornece o método `parseBitmapData()` para conversão de dados de imagem em uma versão String.
com/example/programmingas3/asciiArt/Image.as	Uma classe que representa uma imagem de bitmap carregada.
com/example/programmingas3/asciiArt/ImageInfo.as	Uma classe que representa metadados para uma imagem arte ASCII (como título, URL do arquivo de imagem, etc.)
imagem/	Uma pasta que contém imagens usadas pelo aplicativo.
txt/ImageData.txt	O arquivo de texto delimitado por tabulação que contém informações sobre as imagens a serem carregadas pelo aplicativo.

Extração de valores delimitados por tabulação

Este exemplo usa a prática comum de armazenar dados do aplicativo separados do próprio aplicativo. Dessa maneira, se os dados forem alterados (por exemplo, se outra imagem for adicionada ou um título da imagem for alterado) não haverá necessidade de recriar o arquivo SWF. Nesse caso, os metadados da imagem, incluindo o título da imagem, a URL do arquivo real da imagem e alguns valores que são usados para manipular a imagem são armazenados em um arquivo de texto (o arquivo txt\ImageData.ttx do projeto). O conteúdo do arquivo de texto é o seguinte:

```
FILENAMETITLEWHITE_THRESHHOLDBLACK_THRESHHOLD
FruitBasket.jpgPear, apple, orange, and bananad810
Banana.jpgA picture of a bananaC820
Orange.jpgorangeFF20
Apple.jpgpicture of an apple6E10
```

O arquivo usa um formato delimitado por tabulação específico. A primeira linha é uma linha de título. As linhas restantes contêm os seguintes dados para cada bitmap a ser carregado:

- O nome do arquivo do bitmap.
- O nome de exibição do bitmap.
- Os valores de limite de branco e de limite de preto dos bitmaps. Esses são valores hexadecimais acima e abaixo dos quais um pixel deve ser considerado completamente branco ou completamente preto.

Assim que o aplicativo é iniciado, a classe AsciiArtBuilder carrega e analisa o conteúdo do arquivo de texto para criar a "pilha" de imagens que irá exibir usando o seguinte código do método `parseImageInfo()` da classe AsciiArtBuilder:

```
var lines:Array = _imageInfoLoader.data.split("\n");
var numLines:uint = lines.length;
for (var i:uint = 1; i < numLines; i++)
{
    var imageInfoRaw:String = lines[i];
    ...
    if (imageInfoRaw.length > 0)
    {
        // Create a new image info record and add it to the array of image info.
        var imageInfo:ImageInfo = new ImageInfo();

        // Split the current line into values (separated by tab (\t)
        // characters) and extract the individual properties:
        var imageProperties:Array = imageInfoRaw.split("\t");
        imageInfo.fileName = imageProperties[0];
        imageInfo.title = normalizeTitle(imageProperties[1]);
        imageInfo.whiteThreshold = parseInt(imageProperties[2], 16);
        imageInfo.blackThreshold = parseInt(imageProperties[3], 16);
        result.push(imageInfo);
    }
}
```

Todo o conteúdo do arquivo de texto é contido em uma única ocorrência de String, a propriedade `_imageInfoLoader.data`. Usando o método `split()` com o caractere de nova linha (`"\n"`) como um parâmetro, a ocorrência de String é dividida em uma matriz (`linhas`) cujos elementos são as linhas individuais do arquivo de texto. Em seguida, o código usa um loop para trabalhar com cada uma das linhas (exceto a primeira, porque ela contém apenas cabeçalhos em vez do conteúdo real). Dentro do loop, o método `split()` é usado uma vez novamente para dividir o conteúdo da única linha em um conjunto de valores (o objeto Array denominado `imageProperties`). O parâmetro usado com o método `split()` nesse caso é o caractere de tabulação (`"\t"`), porque os valores de cada linha estão delineados por caracteres de tabulação.

Uso de método String para normalizar títulos de imagens

Uma da decisões de design desse aplicativo é que todos os títulos de imagens são exibidos usando um formato padrão, com a primeira letra de cada palavra colocada em maiúscula (exceto por algumas palavras que normalmente não são colocadas em maiúsculas em títulos em inglês). Em vez de assumir que o arquivo de texto contém títulos formatados de maneira apropriada, o aplicativo formata os títulos enquanto eles estão sendo extraídos do arquivo de texto.

Na listagem de código anterior, como parte da extração de valores individuais de metadados da imagem, a seguinte linha de código é usada:

```
imageInfo.title = normalizeTitle(imageProperties[1]);
```

Nesse código, o título da imagem do arquivo de texto é passado pelo método `normalizeTitle()` antes de ser armazenado no objeto ImageInfo:

```
private function normalizeTitle(title:String):String
{
    var words:Array = title.split(" ");
    var len:uint = words.length;
    for (var i:uint; i < len; i++)
    {
        words[i] = capitalizeFirstLetter(words[i]);
    }

    return words.join(" ");
}
```

Esse método usa o método `split()` para dividir o título em palavras individuais (separadas pelo caractere espaço), passa cada palavra pelo método `capitalizeFirstLetter()` e, em seguida, usa o método `join()` da classe Array para combinar as palavras em uma única string novamente.

Como o nome sugere, o método `capitalizeFirstLetter()` realmente faz o trabalho de colocar a primeira letra de cada palavra em maiúscula:

```
    /**
     * Capitalizes the first letter of a single word, unless it's one of
     * a set of words that are normally not capitalized in English.
     */
    private function capitalizeFirstLetter(word:String):String
    {
        switch (word)
        {
            case "and":
            case "the":
            case "in":
            case "an":
            case "or":
            case "at":
            case "of":
            case "a":
                // Don't do anything to these words.
                break;
            default:
                // For any other word, capitalize the first character.
                var firstLetter:String = word.substr(0, 1);
                firstLetter = firstLetter.toUpperCase();
                var otherLetters:String = word.substring(1);
                word = firstLetter + otherLetters;
        }
        return word;
    }
```

Em inglês, o caractere inicial de cada palavra em um título *não* será colocado em maiúscula se a palavra for uma das seguintes: “and”, “the”, “in”, “an”, “or”, “at”, “of” ou “a” (essa é uma versão simplificada das regras). Para executar essa lógica, o código primeiro usa uma declaração `switch` para verificar se a palavra é uma das palavras que não devem ser colocadas em letra maiúscula. Nesse caso, o código simplesmente ignora a declaração `switch`. Por outro lado, se a palavra precisar ser colocada em maiúscula, isso será feito em várias etapas, da seguinte maneira:

1 A primeira letra da palavra é extraída usando `substr(0, 1)` que extrai uma substring a partir do caractere no índice 0 (a primeira letra da string, conforme indicado pelo primeiro parâmetro `0`). A substring terá um caractere de comprimento (indicado pelo segundo parâmetro `1`).

2 Esse caractere é colocado em maiúscula usando o método `toUpperCase()`.

3 Os caracteres restantes da palavra original são extraídos usando `substring(1)` que extrai uma substring no índice 1 (a segunda letra) até o final da string (indicado pela omissão do segundo parâmetro do método `substring()`).

4 A palavra final é criada combinando a primeira letra recém colocada em maiúscula com as letras restantes usando concatenação de strings: `firstLetter + otherLetters`.

Geração do texto de arte ASCII

A classe BitmapToAsciiConverter fornece a funcionalidade de converter uma imagem de bitmap em sua representação de texto ASCII. Esse processo é executado pelo método `parseBitmapData()` que é parcialmente mostrado aqui:

```
var result:String = "";

// Loop through the rows of pixels top to bottom:
for (var y:uint = 0; y < _data.height; y += verticalResolution)
{
    // Within each row, loop through pixels left to right:
    for (var x:uint = 0; x < _data.width; x += horizontalResolution)
    {
        ...

        // Convert the gray value in the 0-255 range to a value
        // in the 0-64 range (since that's the number of "shades of
        // gray" in the set of available characters):
        index = Math.floor(grayVal / 4);
        result += palette.charAt(index);
    }
    result += "\n";
}
return result;
```

Esse código primeiro define uma ocorrência de String denominada `result` que será usada para criar a versão de arte ASCII da imagem de bitmap. Em seguida, ela executa loop pelos pixels individuais da imagem do bitmap de origem. Usando várias técnicas de manipulação de cores (omitidas aqui para resumir), ela converte os valores das cores vermelho, verde e azul de um pixel individual em um valor de escala de cinza único (um número de 0 a 255). Em seguida, o código divide esse valor por 4 (conforme mostrado) para convertê-lo em um valor na escala de 0 a 63 que é armazenado na variável `index`. (A escala de 0 a 63 é usada porque o "palete" de caracteres ASCII disponíveis usado por esse aplicativo contém 64 valores.) O palete de caracteres é definido como uma ocorrência de String na classe BitmapToAsciiConverter:

```
// The characters are in order from darkest to lightest, so that their
// position (index) in the string corresponds to a relative color value
// (0 = black).
private static const palette:String =
"@#$%&8BMW*mwqpdbkhaoQ0OZXYUJCLtfjzxnuvcr[]{}1()|/?Il!i><+_~-;,. ";
```

Como a variável `index` define qual caractere ASCII no palete corresponde ao pixel atual na imagem de bitmap, esse caractere é recuperado da String `palette` usando o método `charAt()`. Em seguida, ele é anexado à ocorrência da String `result` usando o operador de atribuição de concatenação (`+=`). Além disso, no final de cada linha de pixels, um caractere de nova linha é concatenado ao final da String `result`, forçando uma quebra de linha para criar uma nova linha de "pixels" de caracteres.

Capítulo 8: Trabalho com matrizes

As matrizes permitem que você armazene vários valores em uma única estrutura de dados. É possível usar matrizes indexadas simples que armazenam valores usando índices de números ordinais inteiros e fixos ou matrizes associativas complexas que armazenam valores usando chaves arbitrárias. As matrizes também podem ser multidimensionais, contendo elementos que já são matrizes propriamente ditas. Finalmente, você pode usar um Vetor para uma matriz cujos elementos são ocorrências do mesmo tipo de dados. Este capítulo discute como criar e manipular vários tipos de matriz.

Noções básicas sobre matrizes

Introdução ao trabalho com matrizes

Em programação é comum o trabalho com uma série de itens, e não com um único objeto. Por exemplo, talvez você queira ter uma lista de músicas pronta para ser reproduzida em um aplicativo de player de música. Você certamente não optaria por criar uma variável específica para cada música da lista. Seria melhor juntar todos os objetos Song em um pacote e trabalhar com eles como se fossem um grupo.

Uma matriz é um elemento de programação que funciona como contêiner para um conjunto de itens, como uma lista de músicas. De modo geral, todos os itens da matriz são ocorrências da mesma classe, mas isso não é uma exigência no ActionScript. Os itens individuais de uma matriz são conhecidos como *elementos* da matriz. A matriz pode ser considerada como um arquivador de variáveis. As variáveis podem ser adicionadas como elementos na matriz, como quando você coloca uma pasta em seu arquivador. Você pode trabalhar com a matriz como uma única variável, como carregar todo o seu arquivo para um outro local. Você pode trabalhar com as variáveis como um grupo, como analisar as pastas uma a uma para buscar informações. Você também pode acessá-las individualmente, como se estivesse abrindo o arquivo e selecionando uma única pasta.

Por exemplo, imagine que está criando um aplicativo de player de música no qual o usuário pode selecionar várias músicas e adicioná-las a uma lista de reprodução. No código do ActionScript, há um método chamado `addSongsToPlaylist()`, que aceita uma única matriz como parâmetro. Independentemente da quantidade de músicas que forem adicionadas à lista (poucas, muitas ou apenas uma), você pode chamar o método `addSongsToPlaylist()` apenas uma vez, transmitindo a matriz que contém os objetos Song. Dentro do método `addSongsToPlaylist()`, você pode usar um loop para navegar entre os elementos da matriz (as músicas) uma a uma e adicioná-las à lista de reprodução.

A *matriz indexada* é o tipo de matriz mais comum do ActionScript. Em uma matriz indexada, cada item é armazenado em um slot numerado (conhecido como *índice*). Os itens são acessados por meio de números, como em endereços. As matrizes indexadas atendem a maior parte das exigências de programação. A classe Array é uma das classes mais comuns utilizada para representar uma matriz indexada.

Em geral, uma matriz indexada é usada para armazenar vários itens do mesmo tipo (objetos que são ocorrências da mesma classe). A classe Array não tem meios de restringir o tipo de itens que ela contém. A classe Vector é o tipo de matriz indexada no qual todos os itens de uma única matriz são do mesmo tipo. O uso da ocorrência Vector em vez de Array também pode proporcionar melhorias no desempenho, além de outras vantagens. A classe Vector está disponível a partir do Flash Player 10.

A *matriz multidimensional* representa um caso especial de utilização de matrizes indexadas. Uma matriz multidimensional é uma matriz indexada cujos elementos são matrizes indexadas que, por sua vez, contêm outros elementos.

A *matriz associativa* é um outro tipo de matriz, que usa uma string *key* em vez do índice numérico para identificar elementos individuais. Por fim, o ActionScript 3.0 também possui a classe Dictionary, que representa um *dicionário*. Um dicionário é uma matriz que permite o uso de qualquer tipo de objeto como uma chave de distinção entre elementos.

Tarefas comuns de matrizes

As seguintes atividades comuns para se trabalhar com matrizes são descritas neste capítulo:

- Criação de matrizes indexadas usando as classes Array e Vector
- Adição e remoção de elementos de matriz
- Classificação de elementos de matriz
- Extração de partes de uma matriz
- Trabalho com matrizes associativas e dicionários
- Trabalho com matrizes multidimensionais
- Cópia de elementos de matriz
- Criação de uma subclasse de matriz

Conceitos e termos importantes

A lista de referência a seguir contém termos importantes usados neste capítulo:

- Matriz: é um objeto que serve como contêiner para agrupar vários objetos
- Operador de acesso à matriz (`[]`): é um par de colchetes que circundam um índice ou uma chave e identifica exclusivamente um elemento de matriz. Essa sintaxe é usada após um nome de variável de matriz para especificar um único elemento da matriz, em vez de especificá-la inteira.
- Matriz associativa: é uma matriz que usa chaves de string para identificar elementos individuais
- Tipo base: é o tipo de dados dos objetos que uma ocorrência de Vector pode armazenar
- Dicionário: é uma matriz cujos itens consistem em pares de objetos, conhecidos como chaves e valores. A chave é usada no lugar de um índice numérico para identificar um único elemento.
- Elemento: é um item único de uma matriz
- Índice: é o "endereço" numérico usado para identificar um único elemento em uma matriz indexada
- Matriz indexada: é o tipo padrão da matriz que armazena cada elemento em uma posição numerada e usa o número (índice) para identificar elementos individuais
- Chave: é a string ou o objeto usado para identificar um único elemento em uma matriz associativa ou em um dicionário
- Matriz multidimensional: é uma matriz que contém itens que são matrizes, em vez de valores únicos
- `T`: é a convenção padrão usada nesta documentação para representar o tipo base de uma ocorrência de Vector, independentemente do tipo base. A convenção T é usada para representar um nome de classe, conforme exibido na descrição do parâmetro Type. (“T” corresponde a “tipo”, como em “tipo de dados”).

- Parâmetro Type: é a sintaxe usada com o nome da classe Vector para especificar o tipo base do vetor (o tipo de dados dos objetos que ele armazena). A sintaxe consiste em um ponto (`.`), seguido do nome do tipo de dados entre colchetes angulares (`<>`). Resumindo, teremos algo como: `Vector.<T>`. Nessa documentação, a classe especificada no parâmetro type é representada genericamente como `T`.
- Vector: é um tipo de matriz cujos elementos são todos ocorrências do mesmo tipo de dados

Teste dos exemplos do capítulo

Durante a leitura deste capítulo, talvez você queira testar algumas listagens de código de exemplo sozinho. Basicamente, todas as listagens de código deste capítulo incluem a chamada de função `trace()` apropriada. Para testar as listagens de código deste capítulo:

1. Crie um documento vazio com a ferramenta de autoria do Flash
2. Selecione um quadro-chave na linha de tempo.
3. Abra o painel Ações e copie a listagem de código no painel Script.
4. Execute o programa usando o comando Controlar > Testar filme.

   Você verá os resultados da função `trace()` no painel Saída.

Essa e outras técnicas para testar as listagens de código de exemplo estão descritas detalhadamente em "Teste de listagens de código de exemplo dos capítulos" na página 36.

Matrizes indexadas

As matrizes indexadas armazenam uma série de um ou mais valores organizados de tal maneira que cada valor pode ser acessado com um valor inteiro sem sinal. O primeiro índice é sempre o número 0 e o índice aumenta em incrementos de 1 para cada elemento subseqüente adicionado à matriz. No ActionScript 3.0, duas classes são usadas como matrizes indexadas: classes Array e Vector.

As matrizes indexadas usam um inteiro de 32 bits sem sinal como número do índice. O tamanho máximo de uma matriz indexada é 2^{32} - 1 ou 4.294.967.295. Uma tentativa de criar uma matriz maior que o tamanho máximo resulta em um erro de tempo de execução.

Para acessar um elemento individual de uma matriz indexada, use o operador de acesso à matriz (`[]`) para especificar a posição de índice do elemento que deseja acessar. Por exemplo, o código a seguir representa o primeiro elemento (o elemento no índice 0) em uma matriz indexada chamada `songTitles`:

```
songTitles[0]
```

A combinação do nome da variável de matriz seguido pelo índice entre colchetes funciona como um identificador único (em outras palavras, ele pode ser usado da mesma forma que um nome de variável). Você pode atribuir um valor para um elemento de matriz indexada usando o nome e o índice à esquerda de uma instrução de atribuição:

```
songTitles[1] = "Symphony No. 5 in D minor";
```

Da mesma forma, você pode recuperar o valor de um elemento de matriz indexada usando o nome e o índice à direita de uma instrução de atribuição:

```
var nextSong:String = songTitles[2];
```

Você também pode usar uma variável entre colchetes em vez de fornecer um valor explícito (a variável deve conter um valor inteiro não-negativo como uint, um número inteiro positivo ou uma ocorrência de Number de um número inteiro positivo). Geralmente, essa técnica é usada para fazer um loop dos elementos de uma matriz indexada e executar uma operação em alguns ou todos os elementos. A lista de códigos a seguir demonstra essa técnica: O código usa um loop para acessar cada valor em um objeto Array denominado `oddNumbers`. Ele usa a instrução `trace()` para imprimir cada valor da fórmula "oddNumber[*index*] = *value*":

```
var oddNumbers:Array = [1, 3, 5, 7, 9, 11];
var len:uint = oddNumbers.length;
for (var i:uint = 0; i < len; i++)
{
    trace("oddNumbers[" + i.toString() + "] = " + oddNumbers[i].toString());
}
```

Classe Array

A classe Array é o primeiro tipo de matriz indexada. Uma ocorrência de Array pode armazenar valores de qualquer tipo de dados. O mesmo objeto Array pode armazenar objetos de diferentes tipos de dados. Por exemplo, uma única ocorrência de Array pode ter um valor String no índice 0, uma ocorrência de Number no índice 1 e um objeto XML no índice 2.

Classe Vector

A classe Vector corresponde a outro tipo de matriz indexada que está disponível no ActionScript 3.0. Uma ocorrência de Vector é uma *matriz tipificada*, o que significa que todos os elementos dessa ocorrência possuem o mesmo tipo de dados.

Nota: *A classe Vector está disponível a partir do Flash Player 10.*

Quando você declara uma variável Vector ou cria ocorrência de um objeto Vector, está especificando explicitamente os tipos de dados dos objetos que o vetor pode conter. O tipo de dados especificado é conhecido como o *tipo base* do vetor. Nos tempos de execução e de compilação (no modo restrito), qualquer código que define o valor de um elemento Vector ou recupera um valor de um vetor é verificado. Se o tipo de dados do objeto que está sendo adicionado ou recuperado não corresponder ao tipo base do vetor, ocorrerá um erro.

Além da restrição de tipo de dados, a classe Vector tem outras restrições que a diferenciam da classe Array:

- Um vetor é uma matriz densa. Um objeto Array pode ter valores nos índices 0 e 7, mesmo se não tiver valores nas posições de 1 a 6. Entretanto, um vetor deve ter um valor (ou `null`) em cada índice.
- Um vetor pode ter tamanho fixo, opcionalmente. Isso significa que o número de elementos contidos no vetor não pode ser alterado.
- Os limites de acesso aos elementos de um vetor são verificados. Não é possível ler um valor de um índice maior que o elemento final (`tamanho - 1`). Nunca defina um valor com um índice que exceda o índice final atual (em outras palavras, você só pode definir um valor em um índice existente ou no índice `[length]`).

Como resultado das restrições, um vetor tem duas vantagens principais sobre uma ocorrência de Array cujos elementos são ocorrências de uma única classe:

- Desempenho: a iteração e o acesso ao elemento da matriz são muito mais rápidos ao usar uma ocorrência de Vector do que ao usar uma de Array.

- Segurança do tipo: o compilador pode identificar erros de tipo de dados no modo restrito. Exemplos de erros desse tipo incluem a atribuição de valor de um tipo de dados incorreto ao um vetor ou a espera do tipo de dados errado na leitura de um valor de um vetor. Em tempo de execução, os tipos de dados também são verificados durante a adição ou leitura de dados de um objeto Vector. Observe, entretanto, que ao usar o método `push()` ou `unshift()` para adicionar valores a um vetor, os tipos de dados dos argumentos não são verificados durante a compilação. Quando usar esses métodos, os valores serão verificados no tempo de execução.

Com exceção das restrições adicionais e vantagens, a classe Vector é muito parecida com a classe Array. As propriedades e os métodos de um objeto Vector são semelhantes — em grande parte, idênticos — aos de um Array. De qualquer forma, quando você usa uma Matriz em que todos os elementos têm o mesmo tipo de dados, uma ocorrência de Vector tem preferência.

Criação de matrizes

É possível usar diversas técnicas para criar uma ocorrência de Array ou Vector. No entanto, as técnicas para criação de cada tipo de matriz apresentam diferenças entre si.

Criação de uma ocorrência de Array

É possível criar um objeto Array ao chamar o construtor `Array()` ou ao usar a sintaxe literal de matriz.

A função de construtor `Array()` pode ser usada de três modos: Primeiro, se você chama o construtor sem nenhum argumento, obtém uma matriz vazia. Você pode usar a propriedade `length` da classe Array para verificar se a matriz não tem nenhum elemento. Por exemplo, o código a seguir chama o construtor `Array()` sem nenhum argumento:

```
var names:Array = new Array();
trace(names.length); // output: 0
```

Segundo, se você usar um número como o único parâmetro do construtor `Array()`, uma matriz com esse comprimento será criada, com o valor de cada elemento definido como `undefined`. O argumento deve ser um número inteiro sem sinal entre os valores 0 e 4.294.967.295. Por exemplo, o código a seguir chama o construtor `Array()` com um único argumento numérico:

```
var names:Array = new Array(3);
trace(names.length); // output: 3
trace(names[0]); // output: undefined
trace(names[1]); // output: undefined
trace(names[2]); // output: undefined
```

Terceiro, se você chama o construtor e transmite uma lista de elementos como parâmetros, uma matriz com elementos correspondem a cada parâmetro é criada. O código a seguir transmite três argumentos para o construtor `Array()`:

```
var names:Array = new Array("John", "Jane", "David");
trace(names.length); // output: 3
trace(names[0]); // output: John
trace(names[1]); // output: Jane
trace(names[2]); // output: David
```

Você também pode criar matrizes com literais de matriz. Um literal de matriz pode ser atribuído diretamente a uma variável de matriz, como mostra o exemplo a seguir:

```
var names:Array = ["John", "Jane", "David"];
```

Criação de uma ocorrência de Vector

Você pode criar uma ocorrência de Vector ao chamar o construtor `Vector.<T>()`. Você também pode criar um vetor ao chamar a função global `Vector.<T>()`. Essa função converte um objeto especificado em uma ocorrência de Vector. O ActionScript não contém vetores equivalentes à sintaxe do literal de matriz.

Quando você declara uma variável de vetor (ou do mesmo modo, um parâmetro de método de vetor ou um tipo de retorno de método), está especificando o tipo base da variável de vetor. O tipo base também é especificado quando você cria uma ocorrência de Vector ao chamar o construtor `Vector.<T>()`. Em outras palavras, sempre que usar o termo `Vector` no ActionScript, ele estará acompanhado por um tipo base.

O tipo base do vetor é especificado por meio de uma sintaxe de parâmetro de tipo. O parâmetro de tipo segue imediatamente a palavra `Vector` no código. Esse parâmetro consiste em um ponto (.), seguido do nome de classe base entre colchetes angulares (`<>`), conforme apresentado no exemplo a seguir:

```
var v:Vector.<String>;
v = new Vector.<String>();
```

Na primeira linha do exemplo, a variável `v` está declarada como uma ocorrência de `Vector.<String>`. Em outras palavras, ela representa uma matriz indexada que pode conter apenas ocorrências de String. A segunda linha chama o construtor `Vector()` para criar uma ocorrência do mesmo tipo de vetor, ou seja, um vetor cujos elementos são objetos String. Esse construtor atribui o objeto a `v`.

Uso do construtor Vector.<T>()

Caso você use o construtor `Vector.<T>()` sem quaisquer argumentos, uma ocorrência de Vector vazia será criada. Você pode verificar se um vetor está vazio ao conferir sua propriedade `length`. Por exemplo, o código a seguir chama o construtor `Vector.<T>()` sem argumentos:

```
var names:Vector.<String> = new Vector.<String>();
trace(names.length); // output: 0
```

Você pode predefinir a quantidade de elementos que um vetor precisa ter inicialmente, se tiver essa informação antecipadamente. Para criar um vetor com um determinado número de elementos, transfira o número de elementos como o primeiro parâmetro (o parâmetro `length`). Os elementos Vector são preenchidos com ocorrências do tipo base, já que não podem ficar vazios. Se o tipo base é um tipo de referência que permite valores `null`, todos os elementos conterão `null`. Caso contrário, os elementos conterão o valor padrão para a classe. Por exemplo, uma variável uint não pode ser `null`. Conseqüentemente, o código a seguir que lista o vetor chamado `ages` é criado com sete elementos contendo o valor 0:

```
var ages:Vector.<uint> = new Vector.<uint>(7);
trace(ages); // output: 0,0,0,0,0,0,0
```

Por fim, ao usar o construtor `Vector.<T>()`, você também poderá criar um vetor de tamanho fixo ao passar `true` para o segundo parâmetro (o parâmetro `fixed`). Nesse caso o vetor é criado com o número de elementos especificado e esse número não pode ser alterado. Note, no entanto, que ainda será possível mudar os valores dos elementos de um vetor de tamanho fixo.

Diferentemente do que ocorre na classe Array, não é possível passar a lista de valores para o construtor `Vector.<T>()` para especificar os valores iniciais do vetor.

Uso da função global Vector.<T>()

Além do construtor `Vector.<T>()`, você também pode usar a função global `Vector.<T>()` para criar um objeto Vector. A função global `Vector.<T>()` é uma função de conversão. Quando a função global `Vector.<T>()` é chamada, você está especificando o tipo base do vetor que o método retorna. Com isso, uma única matriz indexada (ocorrência de Array ou Vector) é passada como um argumento. Em seguida, o método retorna um vetor com o tipo base especificado, contendo os valores do argumento da matriz de origem. A listagem de código a seguir exibe a sintaxe necessária para chamar a função global `Vector.<T>()`:

```
var friends:Vector.<String> = Vector.<String>(["Bob", "Larry", "Sarah"]);
```

A função global `Vector.<T>()` executa a conversão do tipo de dados em dois níveis. Primeiro, uma ocorrência de Vector é retornada quando uma ocorrência de Array é passada para a função. Segundo, quando a matriz de origem é uma ocorrência de Array ou Vector, a função tenta converter os elementos da matriz de origem em valores do tipo base. A conversão usa regras de conversão de tipo de dados padrão do ActionScript. Por exemplo, a listagem de código a seguir converte os valores de string na matriz de origem para números inteiros no vetor resultante. A parte decimal do primeiro valor (`"1.5"`) está truncada e o terceiro valor não-numérico (`"Waffles"`) é convertido para 0 no resultado:

```
var numbers:Vector.<int> = Vector.<int>("1.5", "17", "Waffles"]);
trace(numbers); // output: 1,17,0
```

Se não for possível converter nenhum dos elementos de origem, ocorrerá um erro.

Quando o código chama a função global `Vector.<T>()`, se um elemento da matriz de origem é uma ocorrência de uma subclasse do tipo base especificado, o elemento é adicionado ao vetor resultante (não há ocorrência de erros). O uso da função global `Vector.<T>()` é o único modo de converter um vetor com tipo base `T` em um vetor com tipo base que seja uma superclasse de `T`.

Inserção de elementos de matriz

A maneira mais simples de adicionar um elemento a uma matriz indexada é usar o operador de acesso à matriz (`[]`). Para definir um valor de um elemento de matriz indexada, use o nome de objeto Array ou Vector e o número de índice à esquerda de uma instrução de atribuição:

```
songTitles[5] = "Happy Birthday";
```

Se a matriz ou o vetor ainda não tiver um elemento no índice, esse índice será criado e o valor será armazenado nele. Se houver um valor no índice, o novo valor substituirá o existente.

Um objeto Array permite a criação de um elemento em qualquer índice. Entretanto, com um objeto Vector você somente poderá atribuir um valor a um índice existente ou ao próximo índice disponível. O próximo índice disponível corresponde à propriedade `length` do objeto Vector. A maneira mais segura de adicionar um novo elemento a um objeto Vector é usar um código como na listagem a seguir:

```
myVector[myVector.length] = valueToAdd;
```

Três métodos da classe Array e Vector - `push()`, `unshift()` e `splice()` - permitem inserir elementos em uma matriz indexada. O método `push()` anexa um ou mais elementos ao final de uma matriz. Em outras palavras, o último elemento inserido na matriz com o método `push()` terá o maior número de índice. O método `unshift()` insere um ou mais elementos no início de uma matriz, sempre no número de índice 0. O método `splice()` insere qualquer número de itens em um índice especificado na matriz.

O exemplo a seguir demonstra os três métodos. Uma matriz chamada `planetas` é criada para armazenar os nomes dos planetas de acordo com a proximidade ao Sol. Primeiro, o método `push()` é chamado para adicionar o item inicial, `Marte`. Segundo, o método `unshift()` é chamado para inserir o item que pertence ao início da matriz, `Mercúrio`. Finalmente, o método `splice()` é chamado para inserir os itens `Vênus` e `Terra` depois de `Mercúrio`, mas antes de `Marte`. O primeiro argumento enviado para `splice()`, o inteiro 1, direciona a inserção para começar no índice 1. O segundo argumento enviado para `splice()`, o inteiro 0, indica que nenhum item deve ser excluído. Finalmente, o terceiro e quarto argumentos enviados para `splice()`, `Vênus` e `Terra`, são os itens que devem ser inseridos.

```
var planets:Array = new Array();
planets.push("Mars"); // array contents: Mars
planets.unshift("Mercury"); // array contents: Mercury,Mars
planets.splice(1, 0, "Venus", "Earth");
trace(planets); // array contents: Mercury,Venus,Earth,Mars
```

Os métodos `push()` e `unshift()` retornam um inteiro sem sinal que representa o comprimento da matriz modificada. O método `splice()` retorna uma matriz vazia quando é usado para inserir elementos, o que pode parecer estranho, mas faz mais sentido de acordo com a versatilidade do método `splice()`. Você pode usar o método `splice()` não só para inserir elementos em uma matriz, mas também para remover elementos de uma matriz. Quando usado para remover elementos, o método `splice()` retorna uma matriz que contém os elementos removidos.

Nota: *Se a propriedade `fixed` do objeto Vector for `true`, o número total de elementos do vetor não poderá ser alterado. Se você tentar adicionar um novo elemento a um vetor de tamanho fixo por meio de técnicas aqui descritas, ocorrerá um erro.*

Recuperação de valores e remoção de elementos de matriz

A maneira mais simples de recuperar o valor de um elemento de uma matriz indexada é usar o operador de acesso à matriz (`[]`). Para recuperar o valor de um elemento de matriz indexada, use o nome de objeto Array ou Vector e o número de índice à direita de uma instrução de atribuição:

```
var myFavoriteSong:String = songTitles[3];
```

Você pode tentar recuperar um valor de uma matriz ou vetor usando um índice onde não houver elementos. Nesse caso, um objeto Array retorna um valor indefinido e um vetor emite uma exceção RangeError.

Três métodos das classes Array e Vector - `pop()`, `shift()` e `splice()` - permitem a remoção de elementos. O método `pop()` remove um elemento do final da matriz. Em outras palavras, ele remove o elemento com o número de índice mais alto. O método `shift()` remove um elemento do início da matriz, ou seja, sempre remove o elemento com o número de índice 0. O método `splice()`, que também pode ser usado para inserir elementos, remove um número arbitrário de elementos começando no número de índice especificado pelo primeiro argumento enviado ao método.

O exemplo a seguir usa os três métodos para remover elementos de uma ocorrência de Array. Uma matriz chamada `oceanos` é criada para armazenar os nomes dos grandes corpos de água. Alguns nomes da matriz referem-se a lagos, não a oceanos, e precisam ser removidos.

Primeiro, o método `splice()` é usado para remover os itens `Aral` e `Superior`, e inserir os itens `Atlântico` e `Índico`. O primeiro argumento enviado para `splice()`, o inteiro 2, indica que a operação deve começar com o terceiro item da lista, que está no índice 2. O segundo argumento, 2, indica que dois itens devem ser removidos. Os argumentos restantes, `Atlântico` e `Índico`, são os valores que devem ser inseridos no índice 2.

Segundo, o método `pop()` é usado para remover o último elemento da matriz, `Huron`. Finalmente, o método `shift()` é usado para remover o primeiro item da matriz, `Vitória`.

```
var oceans:Array = ["Victoria", "Pacific", "Aral", "Superior", "Indian", "Huron"];
oceans.splice(2, 2, "Arctic", "Atlantic"); // replaces Aral and Superior
oceans.pop(); // removes Huron
oceans.shift(); // removes Victoria
trace(oceans);// output: Pacific,Arctic,Atlantic,Indian
```

Os métodos `pop()` e `shift()` retornam o item que foi removido. Para uma ocorrência de Array, o tipo de dados do valor de retorno é Object porque as matrizes podem armazenar valores de qualquer tipo de dados. Para uma ocorrência de Vector, o tipo de dados do valor de retorno é o tipo base do vetor. O método `splice()` retorna uma matriz ou um vetor que contém os valores removidos. Você pode alterar o exemplo de matriz `oceanos` para que a chamada do método `splice()` atribua a matriz retornada a uma nova variável Array, como mostra o exemplo a seguir:

```
var lakes:Array = oceans.splice(2, 2, "Arctic", "Atlantic");
trace(lakes); // output: Aral,Superior
```

Talvez apareça um código que usa o operador `delete` em um elemento de objeto Array. O operador `delete` define o valor de um elemento Array como `undefined`, mas não remove o elemento da matriz. Por exemplo, o código a seguir usa o operador `delete` no terceiro elemento da matriz `oceanos`, mas o comprimento da matriz continua sendo 5:

```
var oceans:Array = ["Arctic", "Pacific", "Victoria", "Indian", "Atlantic"];
delete oceans[2];
trace(oceans);// output: Arctic,Pacific,,Indian,Atlantic
trace(oceans[2]); // output: undefined
trace(oceans.length); // output: 5
```

Você pode truncar uma matriz ou um vetor usando uma propriedade de matriz `length`. Se a propriedade `length` de uma matriz indexada for definida como um comprimento menor do que o atual, a matriz será truncada, o que remove os elementos armazenados nos números de índice maiores do que o novo valor de `length` menos 1. Por exemplo, se a matriz `oceanos` fosse classificada de modo que todas as entradas válidas estivessem no início da matriz, a propriedade `length` poderia ser usada para remover as entradas no final da matriz, como mostra o código a seguir:

```
var oceans:Array = ["Arctic", "Pacific", "Victoria", "Aral", "Superior"];
oceans.length = 2;
trace(oceans); // output: Arctic,Pacific
```

Nota: *Se a propriedade `fixed` do objeto Vector for `true`, o número total de elementos do vetor não poderá ser alterado. Se você tentar remover um elemento ou truncar um vetor de tamanho fixo usando as técnicas aqui descritas, ocorrerá um erro.*

Classificação de uma matriz

Existem três métodos - `reverse()`, `sort()` e `sortOn()` - que permitem alterar a ordem de uma matriz indexada, classificando-a ou invertendo-a. Todos esses métodos modificam a matriz existente. A tabela a seguir resume esses métodos e seus respectivos comportamentos para objetos Array e Vector:

Método	Comportamento de Array	Comportamento de Vector
`reverse()`	Altera a ordem dos elementos de modo de que o último elemento se transforma no primeiro, o penúltimo no segundo e assim por diante.	Idêntico ao comportamento de Array
`sort()`	Permite a classificação dos elementos da matriz de diversos modos predefinidos, como ordem alfabética ou numérica. Também é possível especificar um algoritmo de classificação personalizada.	Classifica os elementos de acordo com o algoritmo de classificação personalizada especificado
`sortOn()`	Permite a classificação de objetos que possuem uma ou mais propriedades em comum, especificando a(s) propriedade(s) que podem ser usadas como chaves de classificação.	Indisponível na classe Vector.

O método reverse()

O método `reverse()` não assume nenhum parâmetro e não retorna um valor, mas permite alternar a ordem da matriz do estado atual para a ordem inversa. O exemplo a seguir inverte a ordem dos oceanos listados na matriz `oceanos`:

```
var oceans:Array = ["Arctic", "Atlantic", "Indian", "Pacific"];
oceans.reverse();
trace(oceans); // output: Pacific,Indian,Atlantic,Arctic
```

Classificação básica com o método sort() (somente para a classe Array)

Para uma ocorrência de Array, o método `sort()` reorganiza os elementos de uma matriz usando a *ordem de classificação padrão*. A ordem de classificação padrão deve ter as seguintes características:

- A classificação diferencia maiúsculas de minúsculas, isto é, os caracteres maiúsculos precedem os minúsculos. Por exemplo, a letra D precede a letra b.
- A classificação é crescente, ou seja, os códigos de caracteres menores (como A) precedem os maiores (como B).
- A classificação coloca valores idênticos perto um do outro, mas não em uma ordem específica.

- A classificação baseia-se em strings, ou seja, os elementos são convertidos em strings antes de serem comparados (por exemplo, 10 vem antes de 3 porque a string `"1"` tem um código de caractere menor do que o da string `"3"`).

Talvez você precise classificar uma matriz sem diferenciar maiúsculas de minúsculas, em ordem decrescente ou sua matriz pode conter elementos que devem ser classificados numericamente em vez de em ordem alfabética. O método `sort()` de classe Array tem um parâmetro `options` que permite a alteração de cada característica da ordem de classificação padrão. As opções são definidas por um conjunto de constantes estáticas da classe Array, como mostra a lista a seguir:

- `Array.CASEINSENSITIVE`: essa opção não diferencia maiúsculas de minúsculas durante a classificação. Por exemplo, a letra minúscula b precede a letra maiúscula D.
- `Array.DESCENDING`: essa opção inverte a classificação crescente padrão. Por exemplo, a letra B precede a letra A.
- `Array.UNIQUESORT`: essa opção interrompe a classificação quando dois valores idênticos são encontrados.
- `Array.NUMERIC`: essa opção faz a classificação numérica, de modo que 3 vem antes de 10.

O exemplo a seguir destaca algumas dessas opções. Uma matriz chamada `poetas` é criada e classificada com diversas opções diferentes.

```
var poets:Array = ["Blake", "cummings", "Angelou", "Dante"];
poets.sort(); // default sort
trace(poets); // output: Angelou,Blake,Dante,cummings

poets.sort(Array.CASEINSENSITIVE);
trace(poets); // output: Angelou,Blake,cummings,Dante

poets.sort(Array.DESCENDING);
trace(poets); // output: cummings,Dante,Blake,Angelou

poets.sort(Array.DESCENDING | Array.CASEINSENSITIVE); // use two options
trace(poets); // output: Dante,cummings,Blake,Angelou
```

Classificação padrão com o método sort() (somente para as classes Array e Vector)

Além da classificação básica disponível para um objeto Array, também é possível definir uma regra de classificação personalizada. Essa técnica é a única forma do método `sort()` disponível para a classe Vector. Para definir uma classificação padrão, escreva uma função de classificação personalizada e passe-a como um argumento para o método `sort()`.

Por exemplo, se houver uma lista de nomes e cada elemento da lista tiver o nome completo da pessoa, mas você quiser classificar a lista pelo sobrenome, use uma função de classificação personalizada para analisar cada elemento e use o sobrenome na função de classificação. O código a seguir mostra como isso pode ser feito com uma função personalizada que é usada como um parâmetro para o método `Array.sort()`:

```
var names:Array = new Array("John Q. Smith", "Jane Doe", "Mike Jones");
function orderLastName(a, b):int
{
    var lastName:RegExp = /\b\S+$/;
    var name1 = a.match(lastName);
    var name2 = b.match(lastName);
    if (name1 < name2)
    {
        return -1;
    }
    else if (name1 > name2)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
trace(names); // output: John Q. Smith,Jane Doe,Mike Jones
names.sort(orderLastName);
trace(names); // output: Jane Doe,Mike Jones,John Q. Smith
```

A função de classificação personalizada `orderLastName()` usa uma expressão regular para extrair o sobrenome de cada elemento a ser usado para a operação de comparação. O identificador da função `orderLastName` é usado como o único parâmetro ao chamar o método `sort()` na matriz `nomes`. A função de classificação aceita dois parâmetros, `a` e `b`, porque atua em dois elementos de matriz ao mesmo tempo. O valor de retorno da função de classificação indica como os elementos devem ser classificados:

- O valor de retorno -1 indica que o primeiro parâmetro, `a`, precede o segundo, `b`.
- O valor de retorno 1 indica que o segundo parâmetro, `b`, precede o primeiro, `a`.
- O valor de retorno 0 indica que os elementos têm a mesma precedência de classificação.

O método sortOn() (somente para a classe Array)

O método `sortOn()` foi desenvolvido para objetos Array com elementos que contêm objetos. Esses objetos devem ter pelo menos uma propriedade comum que pode ser usada como a chave de classificação. O uso do método `sortOn()` para outros tipos de matriz gera resultados inesperados.

Nota: *A classe Vector não inclui um método `sortOn()`. Esse método está disponível apenas para objetos Array.*

O exemplo a seguir revisa a matriz `poetas` para que cada elemento seja um objeto, em vez de uma string. Cada objeto armazena o sobrenome e o ano de nascimento do poeta.

```
var poets:Array = new Array();
poets.push({name:"Angelou", born:"1928"});
poets.push({name:"Blake", born:"1757"});
poets.push({name:"cummings", born:"1894"});
poets.push({name:"Dante", born:"1265"});
poets.push({name:"Wang", born:"701"});
```

Você pode usar o método `sortOn()` para classificar a matriz pela propriedade `born`. O método `sortOn()` define dois parâmetros, `fieldName` e `options`. O argumento `fieldName` deve ser especificado como uma string. No exemplo a seguir, `sortOn()` é chamado com dois argumentos, "`born`" e `Array.NUMERIC`. O argumento `Array.NUMERIC` é usado para assegurar que a classificação seja feita numericamente, não em ordem alfabética. Isso é útil mesmo quando todos os números têm o mesmo número de dígitos porque garante que a classificação continuará apresentando o comportamento esperado se um número com mais ou menos dígitos for adicionado posteriormente à matriz.

```
poets.sortOn("born", Array.NUMERIC);
for (var i:int = 0; i < poets.length; ++i)
{
    trace(poets[i].name, poets[i].born);
}
/* output:
Wang 701
Dante 1265
Blake 1757
cummings 1894
Angelou 1928
*/
```

Classificação sem modificação da matriz original (somente para a classe Array)

Geralmente, os métodos `sort()` e `sortOn()` modificam uma matriz. Se desejar classificar uma matriz sem modificar a existente, transmita a constante `Array.RETURNINDEXEDARRAY` como parte do parâmetro `options`. Essa opção instrui os métodos a retornar uma nova matriz que reflete a classificação e deixar a matriz original inalterada. A matriz retornada pelos métodos é uma simples matriz de números de índice que reflete a nova ordem de classificação e não contém nenhum elemento da matriz original. Por exemplo, para classificar a matriz `poetas` por ano de nascimento sem modificá-la, inclua a constante `Array.RETURNINDEXEDARRAY` como parte do argumento transmitido para o parâmetro `options`.

O exemplo a seguir armazena as informações de índice retornadas em uma matriz chamada `índices` e usa a matriz `índices` junto com a matriz `poetas` inalterada para classificar os poetas por ano de nascimento:

```
var indices:Array;
indices = poets.sortOn("born", Array.NUMERIC | Array.RETURNINDEXEDARRAY);
for (var i:int = 0; i < indices.length; ++i)
{
    var index:int = indices[i];
    trace(poets[index].name, poets[index].born);
}
/* output:
Wang 701
Dante 1265
Blake 1757
cummings 1894
Angelou 1928
*/
```

Consulta de uma matriz

Quatro métodos das classes Array e Vector - `concat()`, `join()`, `slice()` e `toString()` - consultam a matriz em busca de informações, mas não a modificam. Os métodos `concat()` e `slice()` retornam novas matrizes, enquanto os métodos `join()` e `toString()` retornam strings. O método `concat()` pega uma nova matriz ou lista de elementos como argumentos e a combina com a matriz existente para criar uma nova. O método `slice()` tem dois parâmetros, devidamente chamados de `startIndex` e `endIndex`, e retorna uma nova matriz que contém uma cópia dos elementos "fatiados" a partir da matriz existente. A fatia começa com o elemento em `startIndex` e termina com o elemento bem antes de `endIndex`. A máxima é a mesma: O elemento em `endIndex` não é incluído no valor de retorno.

O exemplo a seguir usa `concat()` e `slice()` para criar novas matrizes com elementos de outras matrizes:

```
var array1:Array = ["alpha", "beta"];
var array2:Array = array1.concat("gamma", "delta");
trace(array2); // output: alpha,beta,gamma,delta

var array3:Array = array1.concat(array2);
trace(array3); // output: alpha,beta,alpha,beta,gamma,delta

var array4:Array = array3.slice(2,5);
trace(array4); // output: alpha,beta,gamma
```

Você pode usar os métodos `join()` e `toString()` para consultar a matriz e retornar seu conteúdo como uma string. Se nenhum parâmetro for usado para o método `join()`, os dois métodos terão o mesmo comportamento: eles retornarão uma string que contém uma lista delimitada por vírgulas de todos os elementos da matriz. O método `join()`, diferente do método `toString()`, aceita um parâmetro chamado `delimiter`, que permite escolher o símbolo a ser usado como separador entre cada elemento na string retornada.

O exemplo a seguir cria uma matriz chamada `rios` e chama `join()` e `toString()` para retornar os valores na matriz como uma string. O método `toString()` é usado para retornar valores separados por vírgula (`riverCSV`), enquanto o método `join()` é usado para retornar valores separados pelo caractere `+`.

```
var rivers:Array = ["Nile", "Amazon", "Yangtze", "Mississippi"];
var riverCSV:String = rivers.toString();
trace(riverCSV); // output: Nile,Amazon,Yangtze,Mississippi
var riverPSV:String = rivers.join("+");
trace(riverPSV); // output: Nile+Amazon+Yangtze+Mississippi
```

Um problema do método `join()` que deve ser levado em consideração é o fato de as ocorrências de Array ou Vector aninhadas sempre serem retornadas com valores separados por vírgula, independentemente do separador especificado para os elementos principais da matriz, como mostra o exemplo a seguir:

```
var nested:Array = ["b","c","d"];
var letters:Array = ["a",nested,"e"];
var joined:String = letters.join("+");
trace(joined); // output: a+b,c,d+e
```

Matrizes associativas

Uma matriz associativa, às vezes chamadas de *hash* ou *mapa*, usa *chaves* em vez de índices numéricos para organizar os valores armazenados. Cada chave de uma matriz associativa é uma string exclusiva que é usada para acessar um valor armazenado. Uma matriz associativa é uma ocorrência da classe Object, o que indica que cada chave corresponde a um nome de propriedade. As matrizes associativas são coleções não ordenadas de pares de chave e valor. Seu código não deve esperar que as chaves de uma matriz associativa estejam em uma ordem específica.

O ActionScript 3.0 também inclui um tipo avançado de matriz associativa chamado *dicionário*. Os dicionários, que são ocorrências da classe Dictionary no pacote flash.utils, usa chaves que podem ser de qualquer tipo de dados. Em outras palavras, as chaves de dicionário não estão limitadas aos valores do tipo String.

Matrizes associativas com chaves de string

Há duas maneiras de criar matrizes associativas no ActionScript 3.0. A primeira alternativa é usar uma ocorrência de Object. Ao usar essa ocorrência, será possível inicializar a matriz com um literal de objeto. Uma ocorrência da classe Object, também chamada de *objeto genérico*, tem a mesma funcionalidade de uma matriz associativa. Cada nome de propriedade do objeto genérico serve como a chave que fornece acesso a um valor armazenado.

O exemplo a seguir cria uma matriz associativa chamada `monitorInfo`, usando um literal de objeto para inicializar a matriz com dois pares de chave e valor:

```
var monitorInfo:Object = {type:"Flat Panel", resolution:"1600 x 1200"};
trace(monitorInfo["type"], monitorInfo["resolution"]);
// output: Flat Panel 1600 x 1200
```

Se não for necessário inicializar a matriz no tempo da declaração, use o construtor Object para criar a matriz da seguinte maneira:

```
var monitorInfo:Object = new Object();
```

Depois que a matriz é criada com um literal de objeto ou um construtor da classe Object, você pode adicionar novos valores à matriz usando o operador de acesso à matriz (`[]`) ou o operador de ponto (`.`). O exemplo a seguir adiciona dois novos valores a `monitorArray`:

```
monitorInfo["aspect ratio"] = "16:10"; // bad form, do not use spaces
monitorInfo.colors = "16.7 million";
trace(monitorInfo["aspect ratio"], monitorInfo.colors);
// output: 16:10 16.7 million
```

Observe que a chave chamada `aspect ratio` contém um caractere de espaço. Isso é possível com o operador de acesso à matriz (`[]`), mas gera um erro se a tentativa for feita com o operador ponto. Não é recomendado usar espaços em nomes de chave.

A segunda maneira de criar uma matriz associativa é usar o construtor Array (ou o construtor de qualquer classe dinâmica) e usar o operador de acesso à matriz (`[]`) ou o operador de ponto (`.`) para adicionar os pares de chave e valor à matriz. Se declarar que a matriz associativa é do tipo Array, você não poderá usar um literal de objeto para inicializar a matriz. O exemplo a seguir cria uma matriz associativa chamada `monitorInfo` usando o construtor Array e adiciona uma chave chamada `type` e uma chave chamada `resolution`, junto com seus valores:

```
var monitorInfo:Array = new Array();
monitorInfo["type"] = "Flat Panel";
monitorInfo["resolution"] = "1600 x 1200";
trace(monitorInfo["type"], monitorInfo["resolution"]);
// output: Flat Panel 1600 x 1200
```

Não há nenhuma vantagem em usar o construtor Array para criar uma matriz associativa. Você não pode usar a propriedade `Array.length` ou algum método da classe Array com matrizes associativas, mesmo se o construtor Array ou o tipo de dados Array for usado. O uso do construtor Array é mais adequado para a criação de matrizes indexadas.

Matrizes associativas com chaves de objeto (Dicionários)

É possível usar a classe Dictionary para criar uma matriz associativa que usa objetos para chaves em vez de strings. Essas matrizes às vezes são chamadas de dicionários, hashes ou mapas. Pense, por exemplo, em um aplicativo que determina o local de um objeto Sprite com base em sua associação com um recipiente específico. Você pode usar um objeto Dictionary para mapear cada objeto Sprite para um recipiente.

O código a seguir cria três ocorrências da classe Sprite que servem como chaves para o objeto Dictionary. Cada chave recebe o valor `GroupA` ou `GroupB`. Os valores podem ser de qualquer tipo de dados, mas, nesse exemplo, `GroupA` e `GroupB` são ocorrências da classe Object. Posteriormente, você poderá acessar o valor associado a cada chave com o operador de acesso à matriz (`[]`), como mostra o código a seguir:

```
import flash.display.Sprite;
import flash.utils.Dictionary;

var groupMap:Dictionary = new Dictionary();

// objects to use as keys
var spr1:Sprite = new Sprite();
var spr2:Sprite = new Sprite();
var spr3:Sprite = new Sprite();

// objects to use as values
var groupA:Object = new Object();
var groupB:Object = new Object();

// Create new key-value pairs in dictionary.
groupMap[spr1] = groupA;
groupMap[spr2] = groupB;
groupMap[spr3] = groupB;

if (groupMap[spr1] == groupA)
{
    trace("spr1 is in groupA");
}
if (groupMap[spr2] == groupB)
{
    trace("spr2 is in groupB");
}
if (groupMap[spr3] == groupB)
{
    trace("spr3 is in groupB");
}
```

Iteração com chaves de objeto

É possível percorrer o conteúdo de um objeto Dictionary com um loop `for..in` ou um loop `for each..in`. O loop `for..in` permite percorrer o conteúdo com base nas chaves, enquanto o loop `for each..in` baseia-se nos valores associados a cada chave.

Use o loop `for..in` para acessar diretamente as chaves de um objeto Dictionary. Você também pode acessar os valores do objeto Dictionary com o operador de acesso à matriz (`[]`). O código a seguir usa o exemplo anterior do dicionário `groupMap` para mostrar como percorrer um objeto Dictionary com o loop `for..in`:

```
for (var key:Object in groupMap)
{
    trace(key, groupMap[key]);
}
/* output:
[object Sprite] [object Object]
[object Sprite] [object Object]
[object Sprite] [object Object]
*/
```

Use o loop `for each..in` para acessar diretamente os valores de um objeto Dictionary. O código a seguir também usa o dicionário `groupMap` para mostrar como percorrer um objeto Dictionary com o loop `for each..in`:

```
for each (var item:Object in groupMap)
{
    trace(item);
}
/* output:
[object Object]
[object Object]
[object Object]
*/
```

Gerenciamento de memória e chaves de objeto

O Adobe® Flash® Player e o Adobe® AIR™ usam um sistema de coleta de lixo para recuperar a memória que não é mais usada. Quando um objeto não tem nenhuma referência apontada para ele, está qualificado para a coleta de lixo e a memória é recuperada na próxima vez em que o sistema é executado. Por exemplo, o código a seguir cria um novo objeto e atribui uma referência do objeto à variável `myObject`:

```
var myObject:Object = new Object();
```

Contanto que exista alguma referência ao objeto, o sistema de coleta de lixo não recuperará a memória ocupada pelo objeto. Se o valor de `myObject` for alterado de modo que aponte para um objeto diferente ou seja definido como o valor `null`, a memória ocupada pelo objeto original se qualificará para a coleta de lixo, mas somente se não houver nenhuma outra referência ao objeto original.

Se `myObject` for usado como uma chave em um objeto Dictionary, outra referência ao objeto original estará sendo criado. Por exemplo, o código a seguir cria duas referências a um objeto: a variável `myObject` e a chave no objeto `myMap`:

```
import flash.utils.Dictionary;

var myObject:Object = new Object();
var myMap:Dictionary = new Dictionary();
myMap[myObject] = "foo";
```

Para que o objeto mencionado por `myObject` se qualifique para a coleta de lixo, remova todas as referências a ele. Nesse caso, altere o valor de `myObject` e exclua a chave `myObject` de `myMap`, como mostra o código a seguir:

```
myObject = null;
delete myMap[myObject];
```

Como alternativa, você pode usar o parâmetro `useWeakReference` do construtor Dictionary para que todas as chaves de dicionário sejam *referências fracas*. O sistema de coleta de lixo ignora as referências fracas e, desse modo, um objeto que tem apenas referências fracas se qualifica para a coleta de lixo. Por exemplo, no código a seguir, não é necessário excluir a chave `myObject` de `myMap` para que o objeto se qualifique para coleta de lixo:

```
import flash.utils.Dictionary;

var myObject:Object = new Object();
var myMap:Dictionary = new Dictionary(true);
myMap[myObject] = "foo";
myObject = null; // Make object eligible for garbage collection.
```

Matrizes multidimensionais

As matrizes multidimensionais contêm outras matrizes como elementos. Por exemplo, considere uma lista de tarefas armazenada como uma matriz indexada de strings:

```
var tasks:Array = ["wash dishes", "take out trash"];
```

Se desejar armazenar uma lista separada de tarefas para cada dia da semana, você poderá criar uma matriz multidimensional com um elemento para cada dia da semana. Cada elemento contém uma matriz indexada, similar à matriz `tarefas`, que armazena a lista de tarefas. É possível usar qualquer combinação de matrizes indexadas ou associativas em matrizes multidimensionais. Os exemplos das seções a seguir usam duas matrizes indexadas ou uma matriz associativa de matrizes indexadas. Se desejar, experimente outras combinações para praticar.

Duas matrizes indexadas

Ao usar duas matrizes indexadas, você pode visualizar o resultado como uma tabela ou uma planilha. Os elementos da primeira matriz representam as linhas da tabela, enquanto os elementos da segunda matriz representam as colunas.

Por exemplo, a seguinte matriz multidimensional usa duas matrizes indexadas para rastrear as listas de tarefas de cada dia da semana. A primeira matriz, `masterTaskList`, é criada com o construtor da classe Array. Cada elemento da matriz representa um dia da semana; o índice 0 representa segunda-feira e o índice 6 representa domingo. Esses elementos podem ser considerados linhas da tabela. Você pode criar a lista de tarefas de cada dia atribuindo um literal de matriz a cada um dos sete elementos que podem ser criados na matriz `masterTaskList`. Os literais de matriz representam as colunas da tabela.

```
var masterTaskList:Array = new Array();
masterTaskList[0] = ["wash dishes", "take out trash"];
masterTaskList[1] = ["wash dishes", "pay bills"];
masterTaskList[2] = ["wash dishes", "dentist", "wash dog"];
masterTaskList[3] = ["wash dishes"];
masterTaskList[4] = ["wash dishes", "clean house"];
masterTaskList[5] = ["wash dishes", "wash car", "pay rent"];
masterTaskList[6] = ["mow lawn", "fix chair"];
```

Você pode acessar itens individuais em qualquer lista de tarefas, usando o operador de acesso à matriz (`[]`). O primeiro conjunto de colchetes representa o dia da semana e o segundo representa a lista de tarefas desse dia. Por exemplo, para recuperar a segunda tarefa da lista de quarta-feira, use primeiro o índice 2 para quarta-feira e, em seguida, use o índice 1 para a segunda tarefa da lista.

```
trace(masterTaskList[2][1]); // output: dentist
```

Para recuperar a primeira tarefa da lista de domingo, use o índice 6 para domingo e o índice 0 para a primeira tarefa da lista.

```
trace(masterTaskList[6][0]); // output: mow lawn
```

Matriz associativa com uma matriz indexada

Para facilitar o acesso a matrizes individuais, você pode usar uma matriz associativa para os dias da semana e uma matriz indexada para as listas de tarefas. A matriz associativa permite usar a sintaxe de ponto ao fazer referência a um dia da semana específico, mas aumenta o tempo de processamento de tempo de execução para acessar cada elemento da matriz associativa. O exemplo a seguir usa uma matriz associativa como base de uma lista de tarefas, com um par de chave e valor para cada dia da semana:

```
var masterTaskList:Object = new Object();
masterTaskList["Monday"] = ["wash dishes", "take out trash"];
masterTaskList["Tuesday"] = ["wash dishes", "pay bills"];
masterTaskList["Wednesday"] = ["wash dishes", "dentist", "wash dog"];
masterTaskList["Thursday"] = ["wash dishes"];
masterTaskList["Friday"] = ["wash dishes", "clean house"];
masterTaskList["Saturday"] = ["wash dishes", "wash car", "pay rent"];
masterTaskList["Sunday"] = ["mow lawn", "fix chair"];
```

A sintaxe de ponto deixa o código mais legível, evitando vários conjuntos de colchetes.

```
trace(masterTaskList.Wednesday[1]); // output: dentist
trace(masterTaskList.Sunday[0]);// output: mow lawn
```

É possível percorrer a lista de tarefas usando um loop `for..in`, mas é necessário usar o operador de acesso à matriz (`[]` em vez da sintaxe de ponto para acessar o valor associado a cada chave. Como `masterTaskList` é uma matriz associativa, os elementos não são recuperados necessariamente na ordem esperada, como mostra o exemplo a seguir:

```
for (var day:String in masterTaskList)
{
    trace(day + ": " + masterTaskList[day])
}
/* output:
Sunday: mow lawn,fix chair
Wednesday: wash dishes,dentist,wash dog
Friday: wash dishes,clean house
Thursday: wash dishes
Monday: wash dishes,take out trash
Saturday: wash dishes,wash car,pay rent
Tuesday: wash dishes,pay bills
*/
```

Clonagem de matrizes

A classe Array não tem nenhum método incorporado para fazer cópias de matrizes. Você pode criar uma *cópiasuperficial* de uma matriz chamando os métodos `concat()` ou `slice()` sem nenhum argumento. Em uma cópia superficial, se a matriz original tiver elementos que são objetos, somente as referências aos objetos serão copiadas, não os objetos propriamente ditos. A cópia aponta para os mesmos objetos da original. Qualquer alteração feita nos objetos é refletida nas duas matrizes.

Em uma *cópia profunda*, todos os objetos encontrados na matriz original também são copiados para que a nova matriz não aponte para os mesmos objetos da matriz original. A cópia profunda requer mais de uma linha de código, que normalmente chama uma função de criação. Essa função pode ser criada como uma função do utilitário global ou como um método de uma subclasse de Array.

O exemplo a seguir define uma função chamada `clone()` que faz a cópia profunda. O algoritmo é emprestado de uma técnica comum de programação Java. A função cria uma cópia profunda serializando a matriz em uma ocorrência da classe ByteArray e lendo a matriz de volta em uma nova matriz. Essa função aceita um objeto para que possa ser usada com matrizes indexadas e associativas, como mostra o código a seguir:

```
import flash.utils.ByteArray;

function clone(source:Object):*
{
    var myBA:ByteArray = new ByteArray();
    myBA.writeObject(source);
    myBA.position = 0;
    return(myBA.readObject());
}
```

Tópicos avançados

Extensão da classe Array

A classe Array é uma das poucas classes principais que não é final, ou seja, você pode criar sua própria subclasse de Array. Esta seção mostra um exemplo de como criar uma subclasse de Array e discute alguns problemas que podem ocorrer durante o processo.

Como mencionado anteriormente, as matrizes no ActionScript não são tipificadas, mas é possível criar uma subclasse de Array que aceita elementos somente de um tipo de dados específico. O exemplo das seções a seguir define uma subclasse de Array chamada TypedArray que limita seus elementos aos valores do tipo de dados especificado no primeiro parâmetro. A classe TypedArray é apresentada simplesmente como um exemplo que mostra como estender a classe Array e talvez não seja adequada para fins de produção por diversos motivos. Primeiro, a verificação de tipo ocorre durante o tempo de execução, não no tempo de compilação. Segundo, quando um método TypedArray encontra uma discordância, esta é ignorada e nenhuma exceção é lançada, embora os métodos possam ser facilmente modificados para lançar exceções. Terceiro, a classe não pode impedir o uso do operador de acesso à matriz para inserir valores de qualquer tipo na matriz. Quarto, o estilo de codificação favorece a simplicidade por meio da otimização do desempenho.

Nota: *Você pode usar a técnica descrita aqui para criar uma matriz tipificada. No entanto, recomenda-se o uso de um objeto Vector. Uma ocorrência de Vector é uma matriz true type e fornece desempenho e outros aprimoramentos em relação à classe Array ou qualquer subclasse. O objetivo desta discussão é demonstrar como criar uma subclasse Array.*

Declaração da subclasse

Use a palavra-chave `extends` para indicar que uma classe é subclasse de Array. Uma subclasse de Array deve usar o atributo `dynamic`, assim como a classe Array. Caso contrário, a subclasse não funcionará adequadamente.

O código a seguir mostra a definição da classe TypedArray, que contém uma constante para armazenar o tipo de dados, um método de construtor e os quatro métodos que permitem adicionar elementos à matriz. O código de cada método é omitido neste exemplo, mas é descrito e explicado em detalhes nas seções a seguir:

```
public dynamic class TypedArray extends Array
{
    private const dataType:Class;

    public function TypedArray(...args) {}

    AS3 override function concat(...args):Array {}

    AS3 override function push(...args):uint {}

    AS3 override function splice(...args) {}

    AS3 override function unshift(...args):uint {}
}
```

Os quatro métodos de substituição usam o espaço para nomes AS3 em vez do atributo `public` porque este exemplo supõe que a opção `-as3` do compilador está definida como `true` e a opção `-es` está definida como `false`. Essas são as configurações padrão do Adobe Flex Builder 3 e do Adobe® Flash® CS4 Professional. Para obter mais informações, consulte "Espaço para nomes AS3" na página 124.

Se você for um desenvolvedor experiente que prefere usar protótipos herdados, faça duas pequenas alterações na classe TypedArray para compilá-la com a opção `-es` do compilador definida como `true`. Primeiro, remova todas as ocorrências do atributo `override` e substitua o espaço para nomes AS3 pelo atributo `public`. Segundo, substitua `Array.prototype` nas quatro ocorrências de `super`.

construtor TypedArray

O construtor de subclasse cria um desafio interessante porque aceita uma lista de argumentos de comprimento arbitrário. O desafio agora é transmitir os argumentos para o superconstrutor a fim de criar a matriz. Se você transmitir a lista de argumentos como uma matriz, o superconstrutor irá considerá-la como um único argumento do tipo Array e a matriz resultante terá sempre um elemento. O modo tradicional de manipular a transmissão de listas de argumentos é usar o método `Function.apply()`, que trata uma matriz de argumentos como seu segundo parâmetro, mas a converte em uma lista de argumentos ao executar a função. Infelizmente, o método `Function.apply()` não pode ser usado com construtores.

A única opção restante é recriar a lógica do construtor Array no construtor TypedArray. O código a seguir mostra o algoritmo usado no construtor da classe Array, que pode ser reutilizado no construtor da subclasse de Array:

```
public dynamic class Array
{
    public function Array(...args)
    {
        var n:uint = args.length
        if (n == 1 && (args[0] is Number))
        {
            var dlen:Number = args[0];
            var ulen:uint = dlen;
            if (ulen != dlen)
            {
                throw new RangeError("Array index is not a 32-bit unsigned integer ("+dlen+")");
            }
            length = ulen;
        }
        else
        {
            length = n;
            for (var i:int=0; i < n; i++)
            {
                this[i] = args[i]
            }
        }
    }
}
```

O construtor TypedArray compartilha a maior parte do código do construtor Array, com apenas quatro alterações no código. Primeiro, a lista de parâmetros inclui um novo parâmetro obrigatório de classe de tipo que permite especificar o tipo de dados da matriz. Segundo, a variável `dataType` é atribuída ao tipo de dados transmitido ao construtor. Terceiro, na instrução `else`, o valor da propriedade `length` é atribuído depois do loop `for` para que `length` inclua somente os argumentos do tipo correto. Quarto, o corpo do loop `for` usa a versão de substituição do método `push()` para que apenas os argumentos do tipo de dados correto sejam adicionados à matriz. O exemplo a seguir mostra a função do construtor TypedArray:

```
public dynamic class TypedArray extends Array
{
    private var dataType:Class;
    public function TypedArray(typeParam:Class, ...args)
    {
        dataType = typeParam;
        var n:uint = args.length
        if (n == 1 && (args[0] is Number))
        {
            var dlen:Number = args[0];
            var ulen:uint = dlen
            if (ulen != dlen)
            {
                throw new RangeError("Array index is not a 32-bit unsigned integer ("+dlen+")")
            }
            length = ulen;
        }
        else
        {
            for (var i:int=0; i < n; i++)
            {
                // type check done in push()
                this.push(args[i])
            }
            length = this.length;
        }
    }
}
```

Métodos de substituição de TypedArray

A classe TypedArray substitui os quatro métodos da classe Array que permitem adicionar elementos a uma matriz. Em cada caso, o método de substituição adiciona uma verificação de tipo que impede a adição de elementos que não são do tipo de dados correto. Posteriormente, cada método chama sua própria versão de superclasse.

O método `push()` percorre a lista de argumentos com um loop `for..in` e faz uma verificação de tipo em cada argumento. Todos os argumentos que não são do tipo correto são removidos da matriz `args` com o método `splice()`. Quando o loop `for..in` termina, a matriz `args` contém valores somente do tipo `dataType`. A versão de superclasse de `push()` é chamada com a matriz `args` atualizada, como mostra o código a seguir:

```
AS3 override function push(...args):uint
{
    for (var i:* in args)
    {
        if (!(args[i] is dataType))
        {
            args.splice(i,1);
        }
    }
    return (super.push.apply(this, args));
}
```

O método `concat()` cria uma matriz TypedArray temporária chamada `passArgs` para armazenar os argumentos aprovados na verificação de tipo. Isso permite a reutilização do código de verificação de tipo existente no método `push()`. O loop `for..in` percorre a matriz `args` e chama `push()` em cada argumento. Como `passArgs` é tipificada como TypedArray, a versão TypedArray de `push()` é executada. Em seguida, o método `concat()` chama sua própria versão de superclasse, como mostra o código a seguir:

```
    AS3 override function concat(...args):Array
    {
        var passArgs:TypedArray = new TypedArray(dataType);
        for (var i:* in args)
        {
            // type check done in push()
            passArgs.push(args[i]);
        }
        return (super.concat.apply(this, passArgs));
    }
```

O método `splice()` usa uma lista arbitrária de argumentos, mas os dois primeiros argumentos sempre fazem referência a um número de índice e ao número de elementos a serem excluídos. É por esse motivo que o método de substituição `splice()` executa a verificação de tipo somente para os elementos da matriz `args` nas posições de índice 2 ou superior. É interessante observar que o código parece ser uma chamada recursiva para `splice()` no loop `for`, mas não é porque `args` é do tipo Array e não TypedArray, o que significa que a chamada para `args.splice()` é uma chamada para a versão de superclasse do método. Quando o loop `for..in` termina, a matriz `args` contém apenas os valores do tipo correto nas posições de índice 2 ou superior e `splice()` chama sua própria versão de superclasse, como mostra o código a seguir:

```
    AS3 override function splice(...args):*
    {
        if (args.length > 2)
        {
            for (var i:int=2; i< args.length; i++)
            {
                if (!(args[i] is dataType))
                {
                    args.splice(i,1);
                }
            }
        }
        return (super.splice.apply(this, args));
    }
```

O método `unshift()`, que adiciona elementos ao início de uma matriz, também aceita uma lista arbitrária de argumentos. O método de substituição `unshift()` usa um algoritmo muito parecido com o usado pelo método `push()`, como mostra o exemplo a seguir:

```
    AS3 override function unshift(...args):uint
    {
        for (var i:* in args)
        {
            if (!(args[i] is dataType))
            {
                args.splice(i,1);
            }
        }
        return (super.unshift.apply(this, args));
    }
}
```

Exemplo: lista de reprodução

O exemplo Lista de reprodução demonstra técnicas de trabalho com matrizes, no âmbito de um aplicativo de lista de reprodução que gerencia uma lista de músicas. Estas técnicas são:

- Criação de uma matriz indexada
- Adição de itens a uma matriz indexada
- Classificação de uma matriz de objetos por propriedades diferentes, usando opções de classificação diferentes
- Conversão de uma matriz em uma string delimitada por caracteres

Para obter os arquivos de aplicativo desse exemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_br. Os arquivos do aplicativo Lista de reprodução estão localizados na pasta Amostras/Lista de reprodução. O aplicativo consiste nos seguintes arquivos:

Arquivo	Descrição
PlayList.mxml ou PlayList.fla	O arquivo principal do aplicativo no Flash (FLA) ou no Flex (MXML).
com/example/programmingas3/playlist/Song.as	O objeto de valor que representa informações sobre uma única música. Os itens gerenciados pela classe PlayList são ocorrências de Song.
com/example/programmingas3/playlist/SortProperty.as	Uma pseudo enumeração cujos valores disponíveis representam as propriedades da classe Song por meio da qual uma lista de objetos Song pode ser classificada.

Visão geral da classe PlayList

A classe PlayList gerencia um conjunto de objetos Song. Ela tem métodos públicos e adiciona uma música à lista de reprodução (o método `addSong()`), além de classificar as músicas na lista (o método `sortList()`). Além disso, a classe inclui uma propriedade de acesso somente leitura, `songList`, que fornece acesso ao conjunto real de músicas da lista de reprodução. Internamente, a classe PlayList mantém um registro das músicas usando uma variável de Array privada:

```
public class PlayList
{
    private var _songs:Array;
    private var _currentSort:SortProperty = null;
    private var _needToSort:Boolean = false;
    ...
}
```

Além da variável `_songs` de Array, que é usada pela classe PlayList para manter um registro da lista de músicas, duas outras classes privadas mantém um registro que indica se a lista precisa ser classificada (`_needToSort`) e qual propriedade classifica a lista de músicas em um determinado momento (`_currentSort`).

Assim como com todos os objetos, declarar uma ocorrência de Array é apenas metade do trabalho de criação de uma matriz. Antes de acessar propriedades ou métodos de uma ocorrência de Array, essa classe deve ser instanciada, o que é feito no construtor da classe PlayList.

```
public function PlayList()
{
    this._songs = new Array();
    // Set the initial sorting.
    this.sortList(SortProperty.TITLE);
}
```

A primeira linha do construtor instancia a variável `_songs`, que fica pronta para ser usada. Além disso, o método `sortList()` é chamado para definir a propriedade sort-by inicial.

Adição de uma música à lista

Quando o usuário insere uma nova música do aplicativo, o código no formulário de entrada de dados chama o método `addSong()` da classe PlayList.

```
/**
 * Adds a song to the playlist.
 */
public function addSong(song:Song):void
{
    this._songs.push(song);
    this._needToSort = true;
}
```

Em `addSong()`, o método `push()` da matriz `_songs` é chamado e adiciona o objeto Song que foi transmitido para `addSong()` como um novo elemento dessa matriz. Com o método `push()`, o novo elemento é adicionado ao final da matriz, independentemente de alguma classificação que tenha sido aplicada anteriormente. Isso significa que, depois que o método `push()` é chamado, a lista de músicas provavelmente não será mais classificada corretamente, de modo que a variável `_needToSort` será definida como `true`. Teoricamente, o método `sortList()` poderia ser chamado imediatamente, não sendo mais necessário manter o registro que indica se a lista deve ou não ser classificada em um determinado momento. No entanto, na prática, não é necessário classificar a lista de músicas imediatamente antes que ela seja recuperada. Adiando a operação de classificação, o aplicativo não executará uma classificação desnecessária se, por exemplo, várias músicas forem adicionadas à lista antes de ser recuperada.

Classificação da lista de músicas

Como as ocorrências de Song gerenciadas pela lista de reprodução são objetos complexos, os usuários do aplicativo talvez queiram classificar a lista de reprodução de acordo com propriedades diferentes, como o título da música ou o ano de publicação. No aplicativo PlayList, a tarefa de classificar a lista de músicas tem três partes: identificar a propriedade que classifica a lista, indicar as opções de classificação que devem ser usadas ao classificar de acordo com essa propriedade e executar a operação de classificação real.

Propriedades de classificação

Um objeto Song mantém o registro de várias propriedades, incluindo o título da música, o artista, o ano de publicação, o nome do arquivo e um conjunto de gêneros definido pelo usuário ao qual a música pertence. Entre essas propriedades, apenas as três primeiras são práticas para classificação. Para facilitar o entendimento dos desenvolvedores, o exemplo inclui a classe SortProperty, que age como uma enumeração com valores que representam as propriedades disponíveis para classificação.

```
public static const TITLE:SortProperty = new SortProperty("title");
public static const ARTIST:SortProperty = new SortProperty("artist");
public static const YEAR:SortProperty = new SortProperty("year");
```

A classe SortProperty contém três constantes (`TITLE`, `ARTIST` e `YEAR`), que armazenam uma string com o nome real da propriedade associada da classe Song que pode ser usada para classificação. No restante do código, sempre que uma propriedade de classificação for indicada, o membro de enumeração será usado. Por exemplo, no construtor PlayList, a lista é classificada inicialmente chamando o método `sortList()` do seguinte modo:

```
// Set the initial sorting.
this.sortList(SortProperty.TITLE);
```

Como a propriedade de classificação é especificada como `SortProperty.TITLE`, as músicas são classificadas de acordo com o título.

Classificação por propriedade e especificação das opções de classificação

A classificação real da lista de músicas é realizada pela classe PlayList no método `sortList()` do seguinte modo:

```
/**
 * Sorts the list of songs according to the specified property.
 */
public function sortList(sortProperty:SortProperty):void
{
    ...
    var sortOptions:uint;
    switch (sortProperty)
    {
        case SortProperty.TITLE:
            sortOptions = Array.CASEINSENSITIVE;
            break;
        case SortProperty.ARTIST:
            sortOptions = Array.CASEINSENSITIVE;
            break;
        case SortProperty.YEAR:
            sortOptions = Array.NUMERIC;
            break;
    }

    // Perform the actual sorting of the data.
    this._songs.sortOn(sortProperty.propertyName, sortOptions);

    // Save the current sort property.
    this._currentSort = sortProperty;

    // Record that the list is sorted.
    this._needToSort = false;
}
```

Quando a classificação é feita por título ou artista, faz sentido classificar em ordem alfabética, mas, quando a classificação é feita por ano, é mais lógico realizar uma classificação numérica. A instrução `switch` é usada para definir a opção de classificação adequada, armazenada na variável `sortOptions`, de acordo com o valor especificado no parâmetro `sortProperty`. Aqui, mais uma vez, os membros de enumeração nomeados são usados para diferenciar as propriedades, em vez de valores codificados.

Quando a propriedade e as opções de classificação são determinadas, a matriz `_songs` é realmente classificada chamando o método `sortOn()` e transmitindo esses dois valores como parâmetros. A propriedade de classificação atual é registrada, o que indica que a lista de músicas está classificada no momento.

Combinação de elementos de matriz em uma string delimitada por caracteres

Além de usar uma matriz para manter a lista de músicas na classe PlayList, neste exemplo, as matrizes também são usadas na classe Song para ajudar a gerenciar a lista de gêneros aos quais pertencem as músicas. Veja este snippet da definição da classe Song:

```
private var _genres:String;

public function Song(title:String, artist:String, year:uint, filename:String, genres:Array)
{
    ...
    // Genres are passed in as an array
    // but stored as a semicolon-separated string.
    this._genres = genres.join(";");
}
```

Ao criar uma nova ocorrência de Song, o parâmetro `genres` que é usado para especificar os gêneros das músicas é definido como uma ocorrência de Array. Desse modo, é fácil agrupar vários gêneros em uma única variável que pode ser transmitida para o construtor. No entanto, internamente, a classe Song mantém os gêneros na variável `_genres` privada como uma ocorrência de String separada por ponto-e-vírgula. O parâmetro Array é convertido em uma string separada por ponto-e-vírgula chamando seu método `join()` com o valor de string literal `";"` como o delimitador especificado.

Com o mesmo token, os acessadores de `genres` permitem que os gêneros sejam definidos ou recuperados como uma matriz:

```
    public function get genres():Array
    {
        // Genres are stored as a semicolon-separated String,
        // so they need to be transformed into an Array to pass them back out.
        return this._genres.split(";");
    }
    public function set genres(value:Array):void
    {
        // Genres are passed in as an array,
        // but stored as a semicolon-separated string.
        this._genres = value.join(";");
    }
```

O acessador `genresset` se comporta exatamente como o construtor: aceita uma matriz e chama o método `join()` para convertê-la em uma string separada por ponto-e-vírgula. O acessador `get` executa a operação oposta: o método `split()` da variável `_genres` é chamado, dividindo a string em uma matriz de valores que usa o delimitador especificado (o valor de string literal `";"` como antes).

Capítulo 9: Manipulação de erros

"Manipular" um erro significa criar uma lógica no aplicativo que responda a, ou corrija, um erro gerado quando um aplicativo é compilado ou quando um aplicativo compilado é executado. Quando o seu aplicativo manipula erros e o erro é encontrado, acontece *algo* em resposta, em vez de ficar sem resposta, independentemente do processo que criou a falha do erro silenciosamente. Usado corretamente, a manipulação de erros ajuda a proteger o aplicativos e seus usuários de um comportamento inesperado.

Contudo, trata-se de uma categoria ampla que inclui a resposta a vários tipos de erros que são lançados durante a compilação ou em tempo de execução. Este capítulo descreve como manipular erros de tempo de execução, os diferentes tipos de erros que podem ser gerados e as vantagens do novo sistema de manipulação de erros no ActionScript 3.0. Além disso, explica como implementar suas próprias estratégias personalizadas de manipulação de erros para seus aplicativos.

Noções básicas da manipulação de erros

Introdução à manipulação de erros

Um erro de tempo de execução acontece quando há algo de errado no código do ActionScript que impede a execução de seu conteúdo no Adobe® Flash® Player ou no Adobe® AIR™. Para garantir que o seu código do ActionScript seja executado corretamente para os usuários, você deve escrevê-lo no aplicativo que manipula o erro (que o corrige, contorna ou pelo menos deixa o usuário saber o que aconteceu). Esse processo é chamado de *manipulação de erros*.

A manipulação de erros é uma categoria ampla que inclui a resposta a vários tipos de erros que são lançados durante a compilação ou em tempo de execução. Os erros que acontecem em tempo de compilação, em geral, são mais fáceis de identificar; é necessário corrigi-los para concluir o processo de criação de um arquivo SWF. Este capítulo não discute os erros em tempo de compilação; para obter mais informações sobre a escrita de código que não contém erros de tempo de compilação, consulte “Linguagem e sintaxe do ActionScript” na página 38 e “Programação orientada a objetos no ActionScript” na página 92. Este capítulo enfoca os erros de tempo de execução.

Os erros de tempo de execução podem ser mais difíceis de detectar, porque, para que ocorram, o código com falha deve realmente ser executado. Se um segmento do seu programa tiver várias ramificações de código, como uma instrução `if..then..else`, será necessário testar cada condição possível, com todos os valores de entrada possíveis que os usuários reais podem usar, para confirmar que o seu código não possui erros.

Os erros de tempo de execução podem ser divididos em duas categorias: *erros de programa* são erros no seu código do ActionScript, como especificar o tipo de dados errado para o parâmetro de um método; *erros lógicos* são erros na lógica (na verificação de dados e manipulação de valores) do seu programa, tal como usar a fórmula errada para calcular taxas de juros em um aplicativo bancário. Novamente, os dois tipos de erros, em geral, podem ser detectados e corrigidos com antecedência, testando prontamente o aplicativo.

Provavelmente, você desejará identificar e remover todos os erros do aplicativo antes de lançá-lo para os usuários finais. Entretanto, nem todos os erros podem ser previstos ou evitados. Por exemplo, suponha que o seu aplicativo do ActionScript carregue informações de um site específico que esteja fora do seu controle. Se, em algum momento, esse site não estiver disponível, a parte do seu aplicativo que depende desses dados externos não se comportará corretamente. O aspecto mais importante da manipulação de erros envolve a preparação para esses casos desconhecidos e sua manipulação de forma apropriada para que os usuários possam continuar a usar o seu aplicativo ou, pelo menos, recebam uma mensagem de erro amigável explicando por que ele não está funcionando.

Os erros de tempo de execução são representados de duas formas no ActionScript:

- Classes de erro: muitos erros possuem uma classe de erro associada. Quando ocorre um erro, o Flash Player ou o Adobe AIR cria uma ocorrência da classe de erro específica associada a esse erro em particular. O seu código pode usar as informações contidas nesse objeto de erro para dar uma resposta apropriada ao erro.
- Eventos de erro: às vezes, ocorre um erro quando o Flash Player ou o Adobe AIR normalmente acionaria um evento. Em vez disso, o Flash Player e o Adobe AIR acionam um evento de erro nesses casos. Como outros eventos, cada evento de erro possui uma classe associada, e o Flash Player e o Adobe AIR transmitem uma ocorrência dessa classe para os métodos que são inscritos no evento de erro.

Para determinar se um método específico pode acionar um erro ou evento de erro, consulte a entrada do método na Referência dos componentes e da linguagem do ActionScript 3.0.

Tarefas comuns da manipulação de erros

As tarefas comuns relacionadas a erros que você precisa executar com o seu código são:

- Escrever código para manipular erros
- Testar, detectar e relançar erros
- Definir sua própria classe de erro
- Responder a eventos de status e erro

Conceitos e termos importantes

A lista de referência a seguir contém termos importantes usados neste capítulo:

- Assíncrono: um comando de programa, como uma chamada de método, que não fornece um resultado imediato e, em vez disso, apresenta um resultado (ou erro) na forma de um evento.
- Catch: quando ocorre uma exceção (um erro de tempo de execução) e seu código obtém essa informação, ele recebe a instrução *catch* para detectar a exceção. Assim que a exceção é detectada, o Flash Player e o Adobe AIR param de notificar outro código do ActionScript a respeito dela.
- Versão de depurador: uma versão especial do Flash Player ou do Adobe AIR (ADL) que contém código para notificar os usuários de erros de tempo de execução. Na versão padrão do Flash Player ou do Adobe AIR (aquela que a maioria dos usuários possui), os erros que não são manipulados pelo código do ActionScript são ignorados. Nas versões de depurador (que são incluídas no Adobe Flash CS4 Professional e no Adobe Flex), é exibida uma mensagem de aviso quando acontece um erro não manipulado.
- Exceção: um erro que ocorre quando um programa é executado e o ambiente de tempo de execução (ou seja, o Flash Player ou Adobe AIR) não consegue resolvê-lo sozinho.
- Relançar: quando o seu código detecta uma exceção, o Flash Player e o Adobe AIR param de notificar os outros objetos a respeito dela. Se for importante que os outros objetos sejam notificados a seu respeito, o seu código deverá *relançar* a exceção para reiniciar o processo de notificação.
- Síncrono: um comando de programa, como uma chamada de método, que fornece um resultado imediato (ou imediatamente lança um erro), o que significa que a resposta pode ser usada dentro do mesmo bloco de código.
- Lançar: o ato de notificar o Flash Player ou o Adobe AIR (e conseqüentemente notificar outros objetos e código do ActionScript) de que um erro ocorreu é conhecido como *lançar* um erro.

Teste dos exemplos do capítulo

Durante a leitura deste capítulo, talvez você queira testar algumas listagens de código de exemplo sozinho. Basicamente, todas as listagens de código deste capítulo incluem a chamada de função `trace()` apropriada. Para testar as listagens de código deste capítulo:

1. Crie um documento Flash vazio.
2. Selecione um quadro-chave na linha de tempo.
3. Abra o painel Ações e copie a listagem de código no painel Script.
4. Execute o programa usando Controlar > Testar filme.

   Você verá os resultados das funções `trace()` da listagem de código no painel Saída.

Algumas das últimas listagens de código são mais complexas e são escritas como uma classe. Para testar esses exemplos:

1. Crie um documento Flash e salve-o no seu computador.
2. Crie e salve um novo arquivo do ActionScript no mesmo diretório do documento Flash. O nome do arquivo deve corresponder ao nome da classe na listagem de código. Por exemplo, se a listagem de código definir uma classe chamada ErrorTest, use o nome ErrorTest.as para salvar o arquivo do ActionScript.
3. Copie a listagem de código no arquivo do ActionScript e salve o arquivo.
4. No documento Flash, clique em uma parte em branco do Palco ou espaço de trabalho para ativar o Inspetor de propriedades do documento.
5. No Inspetor de propriedades no campo Classe do documento, digite o nome da classe do ActionScript que você copiou do texto.
6. Execute o programa usando Controlar > Testar filme.

   Você verá os resultados do exemplo no painel Saída (se o exemplo usar a função `trace()`) ou um campo de texto criado pelo código de exemplo.

Essas técnicas para testar as listagens de código de exemplo são descritas com mais detalhes em "Teste de listagens de código de exemplo dos capítulos" na página 36.

Tipos de erros

Quando desenvolver e executar aplicativos, você encontrará diferentes tipos de erros e terminologia de erros. A lista a seguir apresenta os principais termos e tipos de erros:

- *Erros de tempo de compilação* são gerados pelo compilador do ActionScript durante a compilação de código. Os erros de compilação ocorrem quando problemas sintáticos no seu código impedem a criação do aplicativo.
- *Erros de tempo de execução* ocorrem quando você executa o aplicativo depois de compilá-lo. Os erros de tempo de execução representam os erros causados quando um arquivo SWF é reproduzido no Adobe Flash Player ou no Adobe AIR. Na maioria dos casos, você poderá manipular os erros de tempo de execução assim que eles ocorrerem, relatando-os ao usuário e tomando medidas para manter o aplicativo em execução. Se o erro for fatal, por exemplo, não for possível conectar a um site remoto ou carregar os dados necessários, você poderá usar a manipulação de erros para permitir que o aplicativo seja concluído normalmente.

- *Erros síncronos* são erros de tempo de execução que ocorrem no momento em que uma função é chamada; por exemplo, quando você tenta usar um método específico e o argumento que transmite para o método é inválido, o Flash Player ou o Adobe AIR lança uma exceção. A maioria dos erros ocorre de forma síncrona, no momento em que a instrução é executada, e o fluxo de controle transmite imediatamente para a instrução `catch` mais aplicável.

  Por exemplo, o trecho de código a seguir lança um erro de tempo de execução porque o método `browse()` não é chamado antes de o programa tentar carregar um arquivo:

  ```
  var fileRef:FileReference = new FileReference();
  try
  {
      fileRef.upload("http://www.yourdomain.com/fileupload.cfm");
  }
  catch (error:IllegalOperationError)
  {
      trace(error);
      // Error #2037: Functions called in incorrect sequence, or earlier
      // call was unsuccessful.
  }
  ```

  Nesse caso, um erro de tempo de execução é lançado de forma síncrona porque o Flash Player determinou que o método `browse()` não foi chamado antes da tentativa de upload do arquivo.

  Para obter informações detalhadas sobre manipulação de erros síncronos, consulte "Manipulação de erros síncronos em um aplicativo" na página 191.

- *Errosassíncronos* são erros de tempo de execução que ocorrem em vários momentos durante o tempo de execução; eles geram eventos e são detectados por ouvintes de eventos. Uma operação assíncrona é aquela na qual uma função inicia uma operação, mas não espera ela ser concluída. Você pode criar um ouvinte de eventos de erro que espere o aplicativo ou usuário tentar alguma operação e, se a operação falhar, detecte o erro com um ouvinte de eventos e responda ao evento de erro. Depois, o ouvinte de eventos chama uma função de manipulador de eventos para responder ao evento de erro da melhor maneira. Por exemplo, o manipulador de eventos pode iniciar uma caixa de diálogo que solicite que o usuário resolva o erro.

  Considere o exemplo de erro síncrono de carregamento de arquivo apresentado anteriormente. Se você chamar com êxito o método `browse()` antes de começar a carregar um arquivo, o Flash Player irá despachar vários eventos. Por exemplo, quando um upload é iniciado, o evento `open` é despachado. Quando a operação de upload de arquivo é concluída com êxito, o evento `complete` é despachado. Como a manipulação de eventos é assíncrona (isto é, não ocorre em momentos específicos, conhecidos e predeterminados), é necessário usar o método `addEventListener()` para ouvir esses eventos específicos, como mostra o seguinte código:

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.OPEN, openHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
fileRef.browse();

function selectHandler(event:Event):void
{
    trace("...select...");
    var request:URLRequest = new URLRequest("http://www.yourdomain.com/fileupload.cfm");
    request.method = URLRequestMethod.POST;
    event.target.upload(request.url);
}
function openHandler(event:Event):void
{
    trace("...open...");
}
function completeHandler(event:Event):void
{
    trace("...complete...");
}
```

Para obter informações detalhadas sobre manipulação de erros assíncronos, consulte "Resposta a eventos e status de erros" na página 196.

- *Exceções não detectadas* são erros lançados sem nenhuma lógica correspondente (como uma instrução `catch`) em resposta a elas. Se o seu aplicativo lançar um erro, e nenhuma instrução `catch` apropriada ou manipulador de eventos puderem ser encontrados no nível atual ou superior para manipular o erro, o erro será considerado uma exceção não detectada.

  Em tempo de execução, o Flash Player ignora, por design, os erros não detectados e tenta continuar a reprodução quando o erro não interrompe o arquivo SWF atual, porque os usuários não podem necessariamente resolver um erro sozinhos. O processo de ignorar um erro não detectado é chamado de "falha silenciosa" e pode complicar a depuração de aplicativos. A versão de depurador do Flash Player responde a um erro não detectado finalizando o script atual e exibindo o erro não detectado na saída da instrução `trace` ou escrevendo a mensagem de erro em um arquivo de log. Se o objeto de exceção for uma ocorrência da classe Error ou uma de suas subclasses, o método `getStackTrace()` será chamado e as informações de rastreamento de pilha também serão exibidas na saída da instrução de rastreamento ou em um arquivo de log. Para obter mais informações sobre como usar a versão de depurador do Flash Player, consulte "Trabalho com as versões de depurador do Flash Player e do AIR" na página 190.

Manipulação de erros no ActionScript 3.0

Como muitos aplicativos podem ser executados sem construir a lógica para manipular erros, os desenvolvedores ficam tentados a adiar a criação da manipulação de erros em seus aplicativos. Entretanto, sem isso, um aplicativo pode parar facilmente ou frustrar o usuário caso algo não funcione conforme o esperado. O ActionScript 2.0 possui uma classe Error que permite criar a lógica dentro de funções personalizadas e lançar uma exceção com uma mensagem específica. Como a manipulação de erros é essencial para tornar um aplicativo amigável, o ActionScript 3.0 inclui uma arquitetura expandida para detectar erros.

Nota: *Embora a Referência dos componentes e da linguagem do ActionScript 3.0 documente as exceções lançadas por vários métodos, ela pode não incluir todas as exceções possíveis para cada método. Um método pode lançar uma exceção para erros de sintaxe ou outros problemas que não são observados explicitamente na descrição do método, mesmo quando a descrição lista algumas das exceções que um método lança.*

Elementos de manipulação de erros do ActionScript 3.0

O ActionScript 3.0 inclui várias ferramentas para manipulação de erros, incluindo:

- Classes de erro. O ActionScript 3.0 inclui uma ampla variedade de classes Error para expandir o escopo de situações que podem produzir objetos de erro. Cada classe Error ajuda os aplicativos a manipular e responder a condições de erro específicas, quer estejam relacionadas a erros de sistema (como uma condição MemoryError), erros de código (como uma condição ArgumentError), erros de rede e comunicação (como uma condição URIError) ou a outras situações. Para obter mais informações sobre cada classe, consulte "Comparação das classes Error" na página 199.
- Menos falhas silenciosas. Nas versões anteriores do Flash Player, os erros eram gerados e relatados somente se você usasse a instrução `throw` explicitamente. Para o Flash Player 9 e suas versões posteriores e o Adobe AIR, métodos e propriedades nativos do ActionScript lançam erros de tempo de execução que permitem manipular essas exceções com mais eficiência quando elas ocorrem e reagir individualmente a cada uma delas.
- Limpar mensagens de erro exibidas durante a depuração. Ao usar a versão de depurador do Flash Player ou do Adobe AIR, situações ou códigos problemáticos geram mensagens de erro robustas, o que ajuda a identificar facilmente os motivos da falha de um bloco de código específico. Com isso, a correção de erros torna-se mais eficiente. Para obter mais informações, consulte "Trabalho com as versões de depurador do Flash Player e do AIR" na página 190.
- Erros precisos permitem a exibição de claras mensagens de erro aos usuários em tempo de execução. Nas versões anteriores do Flash Player, o método `FileReference.upload()` retornava um valor booleano de `false` se a chamada `upload()` não fosse bem-sucedida, indicando um de cinco erros possíveis. Se ocorrer um erro ao chamar o método `upload()` no ActionScript 3.0, você poderá lançar um de quatro erros específicos, o que ajuda a exibir mensagens de erro mais precisas aos usuários finais.
- Manipulação de erros refinada. Erros distintos são lançados para várias situações comuns. Por exemplo, no ActionScript 2.0, antes de um objeto FileReference ser preenchido, a propriedade `name` tem o valor `null` (portanto, antes de usar ou exibir a propriedade `name`, é necessário garantir que o valor seja definido e não seja `null`). No ActionScript 3.0, se você tentar acessar a propriedade `name` antes que ela seja preenchida, o Flash Player ou o AIR lançará IllegalOperationError, informando que o valor não foi definido, e você pode usar blocos `try..catch..finally` para manipular o erro. Para obter mais informações, consulte, "Uso das instruções try..catch..finally" na página 191.
- Nenhuma desvantagem de desempenho significativa. O uso de blocos `try..catch..finally` para manipular erros exige poucos (ou nenhum dos) recursos adicionais comparado às versões anteriores do ActionScript.
- Uma classe ErrorEvent que permite criar ouvintes para eventos de erro assíncronos específicos. Para obter mais informações, consulte "Resposta a eventos e status de erros" na página 196.

Estratégias de manipulação de erros

Contanto que seu aplicativo não encontre uma condição problemática, ele poderá ser executado com êxito se você não criar uma lógica de manipulação de erros no seu código. Entretanto, se você não manipular os erros ativamente e seu aplicativo encontrar um problema, seus usuários nunca saberão por que ocorreu uma falha.

Há diferentes maneiras de abordar a manipulação de erros no seu aplicativo. A lista a seguir resume as três principais opções para manipular erros:

- Usar instruções `try..catch..finally`. Elas detectarão os erros síncronos que ocorrerem. Você pode aninhar essas instruções em uma hierarquia para detectar exceções em diversos níveis de execução de código. Para obter mais informações, consulte, "Uso das instruções try..catch..finally" na página 191.
- Criar seus próprios objetos de erro personalizados. Você pode usar a classe Error para criar seus próprios objetos de erro personalizados a fim de controlar operações específicas no seu aplicativo que não estejam incluídas nos tipos de erro embutidos. Depois, você pode usar as instruções `try..catch..finally` nos seus objetos de erro personalizados. Para obter mais informações, consulte "Criação de classes de erros personalizadas" na página 195.
- Escreva ouvintes e manipuladores de eventos para responder aos eventos de erro. Usando essa estratégia, você pode criar manipuladores de erro globais que permitem manipular eventos semelhantes sem duplicar uma grande quantidade de código nos blocos `try..catch..finally`. Você tem muito mais chances de detectar erros assíncronos usando essa abordagem. Para obter mais informações, consulte "Resposta a eventos e status de erros" na página 196.

Trabalho com as versões de depurador do Flash Player e do AIR

A Adobe fornece aos desenvolvedores edições especiais do Flash Player e do Adobe AIR para ajudar nos esforços de depuração. Ao instalar o Adobe Flash CS4 Professional ou o Adobe Flex Builder 3, você obtém uma cópia da versão de depurador do Flash Player. Ao instalar uma dessas ferramentas, você obterá uma versão de depurador do Adobe AIR chamada ADL, também fornecida como parte do Adobe AIR SDK.

Há uma grande diferença na forma como as versões de depurador e de lançamento do Flash Player e do Adobe AIR indicam os erros. As versões de depurador mostram o tipo de erro (como Error, IOError ou EOFError genéricos), o número de erros e uma mensagem de erro legível. As versões de lançamento mostram apenas o tipo de erro e o número de erros. Por exemplo, considere o seguinte código:

```
try
{
    tf.text = myByteArray.readBoolean();
}
catch (error:EOFError)
{
    tf.text = error.toString();
}
```

Se o método `readBoolean()` lançasse um EOFError na versão de depurador do Flash Player, seria exibida a seguinte mensagem no campo de texto `tf`: "EOFError: Erro 2030: Fim do arquivo localizado."

O mesmo código em uma versão de lançamento do Flash Player ou do Adobe AIR exibiria o seguinte texto: "EOFError: Erro 2030."

Para manter os recursos e o tamanho mínimos nas versões de lançamento, as seqüências de caracteres de mensagem de erro não estão presentes. Você pode pesquisar o número do erro na documentação (os apêndices da Referência dos componentes e da linguagem do ActionScript 3.0) para correlacionar uma mensagem de erro. Se preferir, você pode reproduzir o erro usando as versões de depurador do Flash Player e do AIR para ver a mensagem inteira.

Manipulação de erros síncronos em um aplicativo

A manipulação de erros mais comum é a lógica de manipulação de erros síncrona, em que você insere instruções no código para detectar erros síncronos em tempo de execução. Esse tipo de manipulação de erros permite ao aplicativo notar e se recuperar de erros de tempo de execução quando as funções falham. A lógica para detectar um erro síncrono inclui as instruções `try..catch..finally`, que literalmente tentam uma operação, detectam qualquer resposta de erro do Flash Player ou do Adobe AIR e finalmente executam outra operação para manipular a operação que falhou.

Uso das instruções try..catch..finally

Durante o trabalho com erros de tempo de execução síncronos, use as instruções `try..catch..finally` para detectar erros. Quando um erro de tempo de execução ocorre, o Flash Player ou o Adobe AIR lança uma exceção, o que significa que ele suspende a execução normal e cria um objeto especial do tipo Error. O objeto Error é lançado para o primeiro bloco `catch` disponível.

A instrução `try` delimita instruções com potencial para criar erros. A instrução `catch` é sempre usada com uma instrução `try`. Se for detectado um erro em uma das instruções no bloco `try`, as instruções `catch` anexadas a `try` serão executadas.

A instrução `finally` delimitará as instruções que serão executadas caso ocorra ou não um erro no bloco `try`. Se não houver nenhum erro, as instruções no bloco `finally` serão executadas depois que as instruções do bloco `try` forem concluídas. Se houver um erro, a instrução `catch` apropriada será executada primeiro, seguida pelas instruções no bloco `finally`.

O código a seguir demonstra a sintaxe para usar as instruções `try..catch..finally`:

```
try
{
    // some code that could throw an error
}
catch (err:Error)
{
    // code to react to the error
}
finally
{
    // Code that runs whether or not an error was thrown. This code can clean
    // up after the error, or take steps to keep the application running.
}
```

Cada instrução `catch` identifica um tipo específico de exceção que ela manipula. A instrução `catch` pode especificar apenas as classes de erro que forem subclasses da classe Error. Cada instrução `catch` é verificada por ordem. Somente a primeira instrução `catch` que corresponder ao tipo de erro lançado será executada. Em outras palavras, se você verificar primeiro a classe Error de alto nível e depois uma subclasse dela, somente a classe Error de alto nível será correspondente. O seguinte código ilustra essa questão:

```
try
{
    throw new ArgumentError("I am an ArgumentError");
}
catch (error:Error)
{
    trace("<Error> " + error.message);
}
catch (error:ArgumentError)
{
    trace("<ArgumentError> " + error.message);
}
```

O código anterior exibe a seguinte saída:

```
<Error> I am an ArgumentError
```

Para detectar corretamente o ArgumentError, os tipos de erro mais específicos devem ser listados primeiro e os mais genéricos por último, como mostra o seguinte código:

```
try
{
    throw new ArgumentError("I am an ArgumentError");
}
catch (error:ArgumentError)
{
    trace("<ArgumentError> " + error.message);
}
catch (error:Error)
{
    trace("<Error> " + error.message);
}
```

Vários métodos e propriedades na API do Flash Player lançam erros de tempo de execução quando encontram erros ao serem executados. Por exemplo, o método `close()` na classe Sound lançará um IOError se o método não conseguir fechar o fluxo de áudio, conforme demonstrado no seguinte código:

```
var mySound:Sound = new Sound();
try
{
    mySound.close();
}
catch (error:IOError)
{
    // Error #2029: This URLStream object does not have an open stream.
}
```

Conforme se familiarizar mais com a Referência dos componentes e da linguagem do ActionScript 3.0, você notará quais métodos lançam exceções, conforme detalhado em cada descrição de método.

A instrução throw

O Flash Player e o Adobe AIR lançam exceções quando encontram erros no aplicativo em tempo de execução. Além disso, você mesmo pode lançar exceções explicitamente usando a instrução `throw`. Ao lançar erros explicitamente, a Adobe recomenda que você lance ocorrências da classe Error ou de suas subclasses. O código a seguir demonstra uma instrução `throw` que lança uma ocorrência da classe Error, `MyErr` e finalmente chama uma função, `myFunction()`, para responder depois que o erro é lançado:

```
var MyError:Error = new Error("Encountered an error with the numUsers value", 99);
var numUsers:uint = 0;
try
{
    if (numUsers == 0)
    {
        trace("numUsers equals 0");
    }
}
catch (error:uint)
{
    throw MyError; // Catch unsigned integer errors.
}
catch (error:int)
{
    throw MyError; // Catch integer errors.
}
catch (error:Number)
{
    throw MyError; // Catch number errors.
}
catch (error:*)
{
    throw MyError; // Catch any other error.
}
finally
{
    myFunction(); // Perform any necessary cleanup here.
}
```

Observe que as instruções `catch` são ordenadas de forma que os tipos de dados mais específicos sejam listados primeiro. Se a instrução `catch` do tipo de dados Number fosse listada primeiro, a instrução `catch` para o tipo de dados uint e a instrução catch para o tipo de dados int nunca seriam executadas.

Nota: *Na linguagem de programação Java, cada função que pode lançar uma exceção deve declarar esse fato, listando as classes de exceção que pode lançar em uma cláusula* `throws` *anexada à declaração de função. O ActionScript não requer que você declare as exceções que podem ser lançadas por uma função.*

Exibição de uma mensagem de erro simples

Uma das grandes vantagens do novo modelo de eventos de exceção e erro é que ele permite dizer aos usuários quando e por que uma ação falhou. Cabe a você escrever o código para exibir a mensagem e oferecer opções em resposta.

O código a seguir mostra uma instrução `try..catch` simples para exibir o erro em um campo de texto:

```
package
{
    import flash.display.Sprite;
    import flash.text.TextField;

    public class SimpleError extends Sprite
    {
        public var employee:XML =
            <EmpCode>
                <costCenter>1234</costCenter>
                <costCenter>1-234</costCenter>
            </EmpCode>;

        public function SimpleError()
        {
            try
            {
                if (employee.costCenter.length() != 1)
                {
                    throw new Error("Error, employee must have exactly one cost center assigned.");
                }
            }
            catch (error:Error)
            {
                var errorMessage:TextField = new TextField();
                errorMessage.autoSize = TextFieldAutoSize.LEFT;
                errorMessage.textColor = 0xFF0000;
                errorMessage.text = error.message;
                addChild(errorMessage);
            }
        }
    }
}
```

Usando uma ampla variedade de classes de erro e erros do compilador embutidos, o ActionScript 3.0 oferece mais informações do que as versões anteriores do ActionScript sobre o motivo de uma falha. Isso permite criar aplicativos mais estáveis com uma manipulação de erros melhor.

Relançamento de erros

Ao criar aplicativos, há várias ocasiões em que pode ser preciso relançar um erro quando não é possível manipulá-lo adequadamente. Por exemplo, o código a seguir mostra um bloco `try..catch` aninhado, que relança um ApplicationError personalizado quando o bloco `catch` não consegue manipular o erro:

```
try
{
    try
    {
        trace("<< try >>");
        throw new ArgumentError("some error which will be rethrown");
    }
    catch (error:ApplicationError)
    {
        trace("<< catch >> " + error);
        trace("<< throw >>");
        throw error;
    }
    catch (error:Error)
    {
        trace("<< Error >> " + error);
    }
}
catch (error:ApplicationError)
{
    trace("<< catch >> " + error);
}
```

O resultado do snippet anterior seria o seguinte:

```
<< try >>
<< catch >> ApplicationError: some error which will be rethrown
<< throw >>
<< catch >> ApplicationError: some error which will be rethrown
```

O bloco `try` aninhado lança um erro ApplicationError personalizado que é detectado pelo bloco `catch` subseqüente. Esse bloco `catch` aninhado pode tentar manipular o erro e, se for bem-sucedido, lançará o objeto ApplicationError ao bloco `try..catch` delimitador.

Criação de classes de erros personalizadas

Você pode estender uma das classes Error padrão para criar suas próprias classes de erro especializadas no ActionScript. Há vários motivos para criar suas próprias classes de erro:

- Para identificar erros ou grupos de erros específicos que são exclusivos do seu aplicativo.

  Por exemplo, você pode querer executar ações diferentes para erros lançados por seu próprio código, além daqueles capturados pelo Flash Player ou pelo Adobe AIR. Você pode criar uma subclasse da classe Error para controlar o novo tipo de dados de erro em blocos `try..catch`.

- Para fornecer recursos de exibição de erros exclusivos para os erros gerados pelo seu aplicativo.

  Por exemplo, você pode criar um novo método `toString()` que formata suas mensagens de erro de uma determinada maneira. Você também pode definir o método `lookupErrorString()` que obtém um código de erro e recupera a mensagem apropriada com base na preferência de idioma do usuário.

Uma classe de erro especializada deve estender a classe Error principal do ActionScript. Veja um exemplo de uma classe AppError especializada que estende a classe Error:

```
public class AppError extends Error
{
    public function AppError(message:String, errorID:int)
    {
        super(message, errorID);
    }
}
```

O exemplo a seguir mostra o uso de AppError em seu projeto:

```
try
{
    throw new AppError("Encountered Custom AppError", 29);
}
catch (error:AppError)
{
    trace(error.errorID + ": " + error.message)
}
```

Nota: *Para substituir o método `Error.toString()` na sua subclasse, você deve atribuir-lhe um parâmetro `...(rest)`. A especificação da linguagem ECMAScript na qual o ActionScript 3.0 é baseado define o método `Error.toString()` desse modo, enquanto o ActionScript 3.0 adota a mesma definição para garantir a compatibilidade com versões anteriores. Portanto, ao substituir o método `Error.toString()`, deve haver uma correspondência exata de parâmetros. Os parâmetros não devem ser transmitidos para o método `toString()` em tempo de execução, porque serão ignorados.*

Resposta a eventos e status de erros

Um dos aprimoramentos mais notáveis da manipulação de erros no ActionScript 3.0 é o suporte à manipulação de eventos de erro para responder a erros de tempo de execução assíncronos. (Para obter uma definição de erros assíncronos, consulte "Tipos de erros" na página 186.)

Você pode criar ouvintes e manipuladores de eventos para responder aos eventos de erro. Muitas classes despacham os eventos de erro da mesma forma o fazem com outros eventos. Por exemplo, uma instância da classe XMLSocket normalmente despacha três tipos de eventos: `Event.CLOSE`, `Event.CONNECT` e `DataEvent.DATA`. Entretanto, quando ocorre um problema, a classe XMLSocket pode despachar o `IOErrorEvent.IOError` ou o `SecurityErrorEvent.SECURITY_ERROR`. Para obter mais informações sobre ouvintes e manipuladores de eventos, consulte"Manipulação de eventos" na página 251.

Os eventos de erro se enquadram em duas categorias:

- Eventos de erro que estendem a classe ErrorEvent

  A classe flash.events.ErrorEvent contém as propriedades e os métodos para gerenciar erros de tempo de execução relacionados às operações de rede e comunicação. As classes AsyncErrorEvent, IOErrorEvent e SecurityErrorEvent estendem a classe ErrorEvent. Se você estiver usando a versão de depurador do Flash Player ou do Adobe AIR, uma caixa de diálogo o informará, em tempo de execução, de qualquer evento de erro sem as funções de ouvinte que o player encontrar.

- Eventos de erro baseados no status

  Os eventos de erro baseados no status estão relacionados às propriedades `netStatus` e `status` das classes de rede e comunicação. Se o Flash Player ou o Adobe AIR encontrar um problema ao ler ou gravar dados, o valor das propriedades `netStatus.info.level` ou `status.level` (dependendo do objeto de classe usado) será definido com o valor `"error"`. A resposta a esse erro é verificar se a propriedade `level` contém o valor `"error"` na função do manipulador de eventos.

Trabalho com eventos de erro

A classe ErrorEvent e suas subclasses contêm tipos de erro para manipular erros que o Flash Player e Adobe AIR despacham quando tentam ler ou gravar dados.

O exemplo a seguir usa uma instrução `try..catch` e manipuladores de evento de erro para exibir os erros detectados ao tentar ler um arquivo local. Você pode adicionar código de manipulação mais sofisticado para fornecer opções a um usuário ou manipular o erro automaticamente nos locais indicados pelo comentário "seu código de manipulação de erros aqui":

```
package
{
    import flash.display.Sprite;
    import flash.errors.IOError;
    import flash.events.IOErrorEvent;
    import flash.events.TextEvent;
    import flash.media.Sound;
    import flash.media.SoundChannel;
    import flash.net.URLRequest;
    import flash.text.TextField;
    import flash.text.TextFieldAutoSize;

    public class LinkEventExample extends Sprite
    {
        private var myMP3:Sound;
        public function LinkEventExample()
        {
            myMP3 = new Sound();
            var list:TextField = new TextField();
            list.autoSize = TextFieldAutoSize.LEFT;
            list.multiline = true;
            list.htmlText = "<a href=\"event:track1.mp3\">Track 1</a><br>";
            list.htmlText += "<a href=\"event:track2.mp3\">Track 2</a><br>";
            addEventListener(TextEvent.LINK, linkHandler);
            addChild(list);
        }

        private function playMP3(mp3:String):void
        {
            try
            {
                myMP3.load(new URLRequest(mp3));
                myMP3.play();
            }
            catch (err:Error)
```

```
            {
                trace(err.message);
                // your error-handling code here
            }
            myMP3.addEventListener(IOErrorEvent.IO_ERROR, errorHandler);
        }

        private function linkHandler(linkEvent:TextEvent):void
        {
            playMP3(linkEvent.text);
            // your error-handling code here
        }

        private function errorHandler(errorEvent:IOErrorEvent):void
        {
            trace(errorEvent.text);
            // your error-handling code here
        }
    }
}
```

Trabalho com eventos de alteração de status

O Flash Player e o Adobe AIR alteram dinamicamente o valor das propriedades `netStatus.info.level` ou `status.level` para as classes que oferecem suporte à propriedade `level`. As classes que possuem a propriedade `netStatus.info.level` são: NetConnection, NetStream e SharedObject. As classes que possuem a propriedade `status.level` são: HTTPStatusEvent, Camera, Microphone e LocalConnection. Você pode escrever uma função de manipulador para responder à alteração no valor `level` e controlar os erros de comunicação.

O exemplo a seguir usa uma função `netStatusHandler()` para testar o valor da propriedade `level`. Se a propriedade `level` indicar que um erro foi encontrado, o código rastreia a mensagem "Falha no fluxo de vídeo".

```
package
{
    import flash.display.Sprite;
    import flash.events.NetStatusEvent;
    import flash.events.SecurityErrorEvent;
    import flash.media.Video;
    import flash.net.NetConnection;
    import flash.net.NetStream;

    public class VideoExample extends Sprite
    {
        private var videoUrl:String = "Video.flv";
        private var connection:NetConnection;
        private var stream:NetStream;

        public function VideoExample()
        {
            connection = new NetConnection();
            connection.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
            connection.addEventListener(SecurityErrorEvent.SECURITY_ERROR, securityErrorHandler);
            connection.connect(null);
        }

        private function netStatusHandler(event:NetStatusEvent):void
```

```
        {
            if (event.info.level == "error")
            {
                trace("Video stream failed")
            }
            else
            {
                connectStream();
            }
        }

        private function securityErrorHandler(event:SecurityErrorEvent):void
        {
            trace("securityErrorHandler: " + event);
        }

        private function connectStream():void
        {
            var stream:NetStream = new NetStream(connection);
            var video:Video = new Video();
            video.attachNetStream(stream);
            stream.play(videoUrl);
            addChild(video);
        }
    }
}
```

Comparação das classes Error

O ActionScript fornece várias classes Error predefinidas. Muitas delas são usadas pelo Flash Player e Adobe AIR, mas você também pode usar as mesmas classes Error no seu próprio código. Há dois tipos principais de classes Error no ActionScript 3.0: As classes Error principais do ActionScript e as classes Error do pacote flash.error. As classes Error principais são prescritas pela especificação de linguagem do ECMAScript (ECMA-262) edição 3. O pacote flash.error contém classes adicionais introduzidas para auxiliar no desenvolvimento e na depuração de aplicativos do ActionScript 3.0.

ECMAScript, classes Error principais

As classes de erro principais do ECMAScript incluem as classes Error, EvalError, RangeError, ReferenceError, SyntaxError, TypeError e URIError. Cada uma dessas classes está localizada no espaço para nomes de nível superior.

Nome da classe	Descrição	Observações
Error	A classe Error pode ser usada para lançar exceções e é a classe base para as outras classes de exceção definidas no ECMAScript: EvalError, RangeError, ReferenceError, SyntaxError, TypeError e URIError.	A classe Error serve de classe base para todos os erros de tempo de execução lançados pelo Flash® Player e Adobe® AIR® e é a classe base recomendada para qualquer classe de erro personalizada.
EvalError	Uma exceção EvalError é lançada se qualquer parâmetro for transmitido para o construtor da classe Function ou se o código do usuário chamar a função `eval()`.	No ActionScript 3.0, o suporte à função `eval()` foi removido e as tentativas de usar a função causam o lançamento de um erro. As versões anteriores do Flash Player usavam a função `eval()` para acessar variáveis, propriedades, objetos ou clipes de filme pelo nome.
RangeError	Uma exceção RangeError é lançada quando um valor numérico fica fora de uma faixa aceitável.	Por exemplo, um RangeError seria lançado pela classe Timer se um atraso fosse negativo ou não fosse finito. Um RangeError também poderia ser lançado se você tentasse adicionar um objeto de exibição em uma profundidade inválida.
ReferenceError	Uma exceção de ReferenceError é lançada quando há uma tentativa de referência a uma propriedade não definida em um objeto selado (não dinâmico). As versões de compilador do ActionScript antes do ActionScript 3.0 não lançavam um erro quando havia uma tentativa de acesso a uma propriedade `não definida`. No entanto, o ActionScript 3.0 lança a exceção ReferenceError nessa condição.	As exceções para variáveis não definidas apontam para possíveis bugs, ajudando a melhorar a qualidade do software. Entretanto, se você não está acostumado a inicializar as variáveis, esse novo comportamento do ActionScript pode exigir algumas alterações nos seus hábitos de criação de código.

Nome da classe	Descrição	Observações
SyntaxError	Uma exceção SyntaxError é lançada quando ocorre um erro de análise no código do ActionScript. Para obter mais informações, consulte a Seção 15.11.6.4 da especificação de linguagem do ECMAScript (ECMA-262) edição 3 em www.ecma-international.org/publications/standards/Ecma-262.htm, bem como a Seção 10.3.1 da especificação do ECMAScript para XML (E4X) (ECMA-357 edição 2) em www.ecma-international.org/publications/standards/Ecma-357.htm.	Um SyntaxError pode ser lançado sob as seguintes circunstâncias: • O ActionScript lança exceções SyntaxError quando uma expressão regular inválida é analisada pela classe RegExp. • O ActionScript lança exceções SyntaxError quando um XML inválido é analisado pela classe XMLDocument.
TypeError	A exceção TypeError é lançada quando o tipo real de operando é diferente do tipo esperado. Para obter mais informações, consulte a Seção 15.11.6.5 da especificação ECMAScript em www.ecma-international.org/publications/standards/Ecma-262.htm, bem como a Seção 10.3 da especificação E4X em www.ecma-international.org/publications/standards/Ecma-357.htm.	Um TypeError pode ser lançado sob as seguintes circunstâncias: • Um parâmetro real para uma função ou um método não pôde ser forçado ao tipo de parâmetro formal. • Um valor é atribuído a uma variável e não pode ser forçado ao tipo da variável. • O lado direito do operador `is` ou `instanceof` não é um tipo válido. • A palavra-chave `super` é usada ilegalmente. • Uma pesquisa de propriedades resulta em mais de uma ligação e, portanto, é ambígua. • Um método é chamado em um objeto incompatível. Por exemplo, uma exceção TypeError será lançada se um método na classe RegExp for "enxertado" em um objeto genérico e, depois, chamado.
URIError	Uma exceção URIError é lançada quando uma das funções de manipulação de URI global é usada de forma incompatível com sua definição. Para obter mais informações, consulte a Seção 15.11.6.6 da especificação ECMAScript em www.ecma-international.org/publications/standards/Ecma-262.htm.	Um URIError pode ser lançado sob as seguintes circunstâncias: Um URI inválido é especificado para uma função da API do Flash Player que espera um URI válido, como `Socket.connect()`.

ActionScript, classes Error principais

Além das classes Error ECMAScript principais, o ActionScript adiciona várias de suas próprias classes para as condições de erro e funcionalidades de manipulação de erros específicas do ActionScript.

Como essas classes eram extensões da linguagem do ActionScript para a especificação de linguagem do ECMAScript edição 3 que podiam ser adições interessantes para uma futura versão da especificação, elas foram mantidas no nível superior em vez de serem colocadas em um pacote como flash.error.

Nome da classe	Descrição	Observações
ArgumentError	A classe ArgumentError representa um erro que ocorre quando os valores de parâmetro fornecidos durante uma chamada de função não correspondem aos definidos para essa função.	Alguns exemplos de erros de argumento incluem: • Foram fornecidos argumentos insuficientes ou excedentes para um método. • Um argumento que deveria ser um membro de uma enumeração não é.
SecurityError	A exceção SecurityError é lançada quando ocorre uma violação de segurança e o acesso é negado.	Alguns exemplos de erros de segurança incluem: • Um acesso a propriedade ou uma chamada de método não autorizado é feito no outro lado de um limite de caixa de proteção. • Foi feita uma tentativa de acessar uma URL não permitida pela caixa de proteção. • Foi feita uma tentativa de conexão de soquete a uma porta, mas o arquivo de política de soquete necessário não estava disponível. • Foi feita uma tentativa de acessar a câmera ou o microfone do usuário, e a solicitação de acesso ao dispositivo foi negada pelo usuário.
VerifyError	Uma exceção VerifyError é lançada quando é encontrado um arquivo SWF malformado ou corrompido.	Quando um arquivo SWF carrega outro arquivo SWF, o SWF pai pode detectar um VerifyError gerado pelo SWF carregado.

flash.error, classes Error do pacote

O pacote flash.error contém classes de Error que são consideradas parte da API do Flash Player. Ao contrário das classes Error que acabamos de descrever, o pacote flash.error comunica eventos de erro específicos do Flash Player ou do Adobe AIR.

Nome da classe	Descrição	Observações
EOFError	Uma exceção EOFError é lançada quando você tenta ler além do fim dos dados disponíveis.	Por exemplo, um EOFError é mostrado quando um dos métodos de leitura na interface IDataInput é chamado e não há dados suficientes para atender à solicitação de leitura.
IllegalOperationError	Uma exceção IllegalOperationError é lançada quando um método não é implementado ou quando a implementação não abrange o uso atual.	Exemplos de exceções de erro de operação ilegal incluem: • Uma classe base, como DisplayObjectContainer, fornece mais funcionalidade do que o Palco pode suportar. Por exemplo, se você tentar obter ou definir uma máscara no Palco (usando `stage.mask`), o Flash Player e o Adobe AIR lançarão um IllegalOperationError com a mensagem: "A classe Stage não implementa essa propriedade ou esse método". • Uma subclasse herda um método que não requer e para o qual não deseja oferecer suporte. • Determinados métodos de acessibilidade são chamados quando o Flash Player é compilado sem o suporte de acessibilidade. • Recursos somente de autoria são chamados de uma versão de tempo de execução do Flash Player. • Você tenta definir o nome de um objeto colocado na linha de tempo.
IOError	Uma exceção IOError é lançada quando ocorre algum tipo de exceção de E/S.	Por exemplo, você obtém este erro quando uma operação de leitura/gravação é tentada em um soquete que não está conectado ou se desconectou.
MemoryError	Uma exceção MemoryError é lançada quando há uma falha na solicitação de alocação de memória.	Por padrão, a ActionScript Virtual Machine 2 não impõe um limite sobre a quantidade de memória que um programa do ActionScript pode alocar. Em um PC desktop, as falhas de alocação de memória não são freqüentes. Um erro é lançado quando o sistema não consegue alocar a memória necessária para uma operação. Por isso, em um PC desktop, essa exceção é rara, a menos que uma solicitação de alocação seja extremamente grande; por exemplo, uma solicitação de 3 bilhões de bytes é impossível porque um programa Microsoft® Windows® de 32 bits só pode acessar 2 GB de espaço de endereçamento.
ScriptTimeoutError	Uma exceção ScriptTimeoutError é lançada quando um intervalo de tempo limite do script de 15 segundos é atingido. Ao detectar uma exceção ScriptTimeoutError, é possível manipular melhor o tempo limite do script. Se não houver um manipulador de exceções, a exceção não capturada exibirá uma caixa de diálogo com uma mensagem de erro.	Para impedir que um desenvolvedor mal-intencionado detecte a exceção e permaneça em um loop infinito, somente a primeira exceção ScriptTimeoutError lançada durante um determinado script pode ser detectada. Uma exceção ScriptTimeoutError subseqüente não poderá ser detectada pelo seu código e irá imediatamente para o manipulador de exceções não detectadas.
StackOverflowError	A exceção StackOverflowError é lançada quando a pilha disponível para o script é esgotada.	Uma exceção StackOverflowError pode indicar que ocorreu recursão infinita.

Exemplo: Aplicativo CustomErrors

O aplicativo CustomErrors demonstra técnicas para trabalhar com erros personalizados ao criar um aplicativo. Estas técnicas são:

- Validar um pacote XML
- Escrever um erro personalizado
- Lançar erros personalizados
- Notificar os usuários quando um erro é lançado

Para obter os arquivos do aplicativo para este exemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_br. Os arquivos do aplicativo CustomErrors podem ser encontrados na pasta Samples/CustomError. O aplicativo consiste nos seguintes arquivos:

Arquivo	Descrição
CustomErrors.mxml ou CustomErrors.fla	O arquivo principal do aplicativo no Flash (FLA) ou no Flex (MXML).
com/example/programmingas3/errors/ApplicationError.as	Uma classe que serve de classe de erro base para as classes FatalError e WarningError.
com/example/programmingas3/errors/FatalError.as	Uma classe que define um erro FatalError que pode ser lançado pelo aplicativo. Essa classe estende a classe ApplicationError personalizada.
com/example/programmingas3/errors/Validator.as	Uma classe que define um único método que valida um pacote XML funcionário fornecido pelo usuário.
com/example/programmingas3/errors/WarningError.as	Uma classe que define um erro WarningError que pode ser lançado pelo aplicativo. Essa classe estende a classe ApplicationError personalizada.

Visão geral do aplicativo CustomErrors

Quando o aplicativo é carregado, o método `initApp()` é chamado no Flex ou o código (não-função) de linha de tempo é executado no Flash. Esse código define um pacote XML de exemplo que será verificado pela classe Validator. O seguinte código é executado:

```
employeeXML =
    <employee id="12345">
        <firstName>John</firstName>
        <lastName>Doe</lastName>
        <costCenter>12345</costCenter>
        <costCenter>67890</costCenter>
    </employee>;
}
```

O pacote XML é exibido posteriormente em uma ocorrência do componente TextArea no Palco. Isso permite modificar o pacote XML antes de tentar validá-lo novamente.

Quando o usuário clica no botão Validate, o método `validateData()` é chamado. Esse método valida o pacote XML funcionário usando o método `validateEmployeeXML()` na classe Validator. O código a seguir mostra o método `validateData()`:

```
function validateData():void
{
    try
    {
        var tempXML:XML = XML(xmlText.text);
        Validator.validateEmployeeXML(tempXML);
        status.text = "The XML was successfully validated.";
    }
    catch (error:FatalError)
    {
        showFatalError(error);
    }
    catch (error:WarningError)
    {
        showWarningError(error);
    }
    catch (error:Error)
    {
        showGenericError(error);
    }
}
```

Primeiro, um objeto XML temporário é criado usando o conteúdo da ocorrência do componente TextArea `xmlText`. Em seguida, o método `validateEmployeeXML()` na classe Validator personalizada (com.example.programmingas3/errors/Validator.as) é chamado e transmite o objeto XML temporário como um parâmetro. Se o pacote XML for válido, a ocorrência do componente Label `status` exibirá uma mensagem de êxito e o aplicativo será encerrado. Se o método `validateEmployeeXML()` lançar um erro personalizado (isto é, se ocorrer um FatalError, WarningError ou um Error genérico), a instrução `catch` apropriada executará e chamará o método `showFatalError()`, `showWarningError()` ou `showGenericError()`. Cada um desses métodos exibe uma mensagem apropriada em uma área de texto chamada `statusText` para notificar o usuário do erro específico que ocorreu. Cada método também atualiza a ocorrência do componente Label `status` com uma mensagem específica.

Se ocorrer um erro fatal durante a tentativa de validar o pacote XML funcionário, a mensagem de erro será exibida na área de texto `statusText` e as ocorrências dos componentes TextArea `xmlText` e Button `validateBtn` serão desativadas, como mostra o seguinte código:

```
function showFatalError(error:FatalError):void
{
    var message:String = error.message + "\n\n";
    var title:String = error.getTitle();
    statusText.text = message + " " + title + "\n\nThis application has ended.";
    this.xmlText.enabled = false;
    this.validateBtn.enabled = false;
    hideButtons();
}
```

Se ocorrer um erro de aviso, em vez de um erro fatal, a mensagem de erro será exibida na ocorrência `statusText` TextArea, mas as ocorrências dos componentes Button e TextField `xmlText` não serão desativadas. O método `showWarningError()` exibe uma mensagem de erro personalizada na área de texto `statusText`. A mensagem também solicita que o usuário decida se deseja prosseguir com a validação do XML ou abortar o script. O seguinte trecho mostra o método `showWarningError()`:

```
function showWarningError(error:WarningError):void
{
    var message:String = error.message + "\n\n" + "Do you want to exit this application?";
    showButtons();
    var title:String = error.getTitle();
    statusText.text = message;
}
```

Quando o usuário clica no botão Yes ou No, o método `closeHandler()` é chamado. O seguinte trecho mostra o método `closeHandler()`:

```
function closeHandler(event:CloseEvent):void
{
    switch (event.detail)
    {
        case yesButton:
            showFatalError(new FatalError(9999));
            break;
        case noButton:
            statusText.text = "";
            hideButtons();
            break;
    }
}
```

Se o usuário decidir abortar o script clicando em Yes, um FatalError será lançado fazendo com que o aplicativo seja encerrado.

Criação de um validador personalizado

A classe Validator personalizada contém um único método, `validateEmployeeXML()`. O método `validateEmployeeXML()` usa um único argumento, `employee`, que é o pacote XML que você deseja validar. O método `validateEmployeeXML()` é o seguinte:

```
public static function validateEmployeeXML(employee:XML):void
{
    // checks for the integrity of items in the XML
    if (employee.costCenter.length() < 1)
    {
        throw new FatalError(9000);
    }
    if (employee.costCenter.length() > 1)
    {
        throw new WarningError(9001);
    }
    if (employee.ssn.length() != 1)
    {
        throw new FatalError(9002);
    }
}
```

Para ser validado, um funcionário deve pertencer a um (e apenas um) centro de custos. Se o funcionário não pertencer a nenhum centro de custos, o método lançará um FatalError, que é emitido ao método `validateData()` no arquivo do aplicativo principal. Se o funcionário pertencer a mais de um centro de custos, será lançado um WarningError. A verificação final no validador XML é que o usuário tem exatamente um número de seguro social (o nó `ssn` no pacote XML). Se não houver exatamente um nó `ssn`, será lançado um erro FatalError.

Você pode adicionar outras verificações ao método `validateEmployeeXML()`, por exemplo, para garantir que o nó `ssn` contenha um número válido ou que o funcionário tenha pelo menos um número de telefone e um endereço de email definido, e os dois valores sejam válidos. Também é possível modificar o XML de forma que cada funcionário tenha uma ID exclusiva e especifique a ID do seu gerente.

Definição da classe ApplicationError

A classe ApplicationError serve de classe base para as classes FatalError e WarningError. A classe ApplicationError estende a classe Error e define seus próprios métodos e propriedades personalizados, incluindo a definição de ID e gravidade de um erro, e um objeto XML que contém os códigos e as mensagens de erro personalizados. Essa classe também define duas constantes estáticas que são usadas para definir a gravidade de cada tipo de erro.

O método do construtor da classe ApplicationError é o seguinte:

```
public function ApplicationError()
{
    messages =
        <errors>
            <error code="9000">
                <![CDATA[Employee must be assigned to a cost center.]]>
            </error>
            <error code="9001">
                <![CDATA[Employee must be assigned to only one cost center.]]>
            </error>
            <error code="9002">
                <![CDATA[Employee must have one and only one SSN.]]>
            </error>
            <error code="9999">
                <![CDATA[The application has been stopped.]]>
            </error>
        </errors>;
}
```

Cada nó de erro no objeto XML contém um código numérico exclusivo e uma mensagem de erro. As mensagens de erro podem ser facilmente pesquisadas pelo seu código de erro usando o E4X, como mostra o seguinte método `getMessageText()`:

```
public function getMessageText(id:int):String
{
    var message:XMLList = messages.error.(@code == id);
    return message[0].text();
}
```

O método `getMessageText()` usa um único argumento inteiro, `id`, e retorna uma seqüência de caracteres. O argumento `id` é o código de erro para o erro a ser pesquisado. Por exemplo, transmitir um `id` de 9001 recupera o erro dizendo que os funcionários devem ser atribuídos apenas a um centro de custos. Se houver mais de um erro com o mesmo código, o ActionScript retornará a mensagem de erro apenas para o primeiro resultado encontrado (`message[0]` no objeto XMLList retornado).

O método seguinte nessa classe, `getTitle()`, não usa nenhum parâmetro e retorna um valor de seqüência de caracteres que contém a ID de erro para esse erro específico. Esse valor é usado para ajudar a identificar com facilidade o erro exato que ocorreu durante a validação do pacote XML. O seguinte trecho mostra o método `getTitle()`:

```
public function getTitle():String
{
    return "Error #" + id;
}
```

O método final na classe ApplicationError é `toString()`. Esse método substitui a função definida na classe Error para que você possa personalizar a apresentação da mensagem de erro. O método retorna uma seqüência de caracteres que identifica o número do erro específico e a mensagem que ocorreu.

```
public override function toString():String
{
    return "[APPLICATION ERROR #" + id + "] " + message;
}
```

Definição da classe FatalError

A classe FatalError estende a classe ApplicationError personalizada e define três métodos: o construtor FatalError, `getTitle()` e `toString()`. O primeiro método, o construtor FatalError, usa um único argumento inteiro, `errorID`, define a gravidade do erro usando os valores constantes estáticos definidos na classe ApplicationError e obtém a mensagem de erro específica, chamando o método `getMessageText()` na classe ApplicationError. O construtor FatalError é o seguinte:

```
public function FatalError(errorID:int)
{
    id = errorID;
    severity = ApplicationError.FATAL;
    message = getMessageText(errorID);
}
```

O método seguinte na classe FatalError, `getTitle()`, substitui o método `getTitle()` definido anteriormente na classe ApplicationError e anexa o texto "-- FATAL" no título para informar ao usuário que um erro fatal ocorreu. O método `getTitle()` é o seguinte:

```
public override function getTitle():String
{
    return "Error #" + id + " -- FATAL";
}
```

O método final nessa classe, `toString()`, substitui o método `toString()` definido na classe ApplicationError. O método `toString()` é:

```
public override function toString():String
{
    return "[FATAL ERROR #" + id + "] " + message;
}
```

Definição da classe WarningError

A classe WarningError estende a classe ApplicationError e é quase idêntica à classe FatalError, exceto por duas pequenas alterações de seqüência de caracteres e por definir a gravidade do erro como ApplicationError.WARNING, em vez de ApplicationError.FATAL, como mostra o seguinte código:

```
public function WarningError(errorID:int)
{
    id = errorID;
    severity = ApplicationError.WARNING;
    message = super.getMessageText(errorID);
}
```

Capítulo 10: Uso de expressões regulares

Uma expressão regular descreve um padrão que é utilizado para localizar e manipular texto correspondente em strings. As expressões regulares parecem ser strings, mas elas podem incluir códigos especiais para descrever padrões e repetição. Por exemplo, a expressão regular a seguir corresponde a uma string que começa com o caractere A seguido por um ou mais dígitos seqüenciais.

```
/A\d+/
```

Este capítulo descreve a sintaxe básica para construir expressões regulares. Entretanto, as expressões regulares podem ter muita complexidade e nuances. Você pode encontrar informações detalhadas sobre expressões regulares na Web e nas livrarias. Lembre-se de que diferentes ambientes de programação implementam expressões regulares de modos diferentes. O ActionScript 3.0 implementa expressões regulares como definido na especificação de idioma do ECMAScript Edição 3 (ECMA-262).

Noções básicas de expressões regulares:

Introdução ao uso de expressões regulares

Uma expressão regular descreve um padrão de caracteres. As expressões regulares são normalmente usadas para verificar se um valor de texto está em conformidade com um determinado padrão (como a verificação para saber se o número de telefone digitado pelo usuário tem o número de dígitos adequado) ou para substituir partes de um valor de texto que corresponde a um determinado padrão.

As expressões regulares podem ser simples. Por exemplo, suponha que você queira confirmar se uma determinada string corresponde a "ABC" ou substituir cada ocorrência de "ABC" em uma string por algum outro texto. Nesse caso, você pode usar a seguinte expressão regular, que define o padrão composto pelas letras A, B e C em seqüência:

```
/ABC/
```

Observe que o literal de uma expressão regular é delineado com o caractere de barra (`/`).

Os padrões de expressões regulares também podem ser complexos e, às vezes, críptico na aparência, como a expressão a seguir, para corresponder a uma endereço de email válido:

```
/([0-9a-zA-Z]+[-._+&])*[0-9a-zA-Z]+@([-0-9a-zA-Z]+[.])+[a-zA-Z]{2,6}/
```

Com mais freqüência, você utilizará expressões regulares para pesquisar padrões em strings e substituir caracteres. Nesses casos, você criará um objeto de expressão regular e o usará como um parâmetro para um dos vários métodos da classe String. Os seguintes métodos da classe String usam as expressões regulares como parâmetros: `match()`, `replace()`, `search()` e`split()`. Para obter mais informações sobre esses métodos, consulte “Localização de padrões em strings e substituição de substrings” na página 149.

A classe RegExp inclui as seguintes opções: `test()` e `exec()`. Para obter mais informações, consulte “Métodos para usar expressões regulares com strings” na página 224.

Tarefas comuns de expressões regulares

Há vários usos comuns para as expressões regulares, descritos em detalhes neste capítulo:

- Criação de uma expressão regular padrão
- Uso de caracteres especiais nos padrões
- Identificação das seqüências de vários caracteres (como "um número de dois dígitos" ou "entre sete e dez letras")
- Identificação de qualquer caractere em um intervalo de letras ou números (como “qualquer letra de *a* a *m*”)
- Identificação de um caractere em um conjunto de possíveis caracteres
- Identificação de subseqüências (segmentos dentro de um padrão)
- Correspondência e substituição de texto com base em padrões

Conceitos e termos importantes

A lista de referência a seguir contém termos importantes usados neste capítulo:

- Caractere Escape: indica que o caractere seguinte deve ser tratado como um metacaractere em vez de um caractere literal. Na sintaxe de expressões regulares, o caractere de barra invertida (\) é o caractere escape; portanto uma barra invertida seguida por outro caractere é um código especial e não apenas o próprio caractere.
- Sinalizador: especifica algumas opções sobre como o padrão de expressão regular deve ser utilizado, como distinguir entre caracteres maiúsculos e minúsculos.
- Metacaractere: um caractere que tem um significado especial em um padrão de expressão regular, em oposição à representação literal do caractere no padrão.
- Quantificador: um caractere (ou vários) indicando quantas vezes uma parte do padrão deve se repetir. Por exemplo, um quantificador é utilizado para designar que o código postal dos Estados Unidos deve conter cinco ou nove números.
- Expressão regular: Uma instrução do programa que define um padrão de caracteres que podem ser usados para confirmar se outras strings correspondem àquele padrão ou substituir partes de uma string.

Teste dos exemplos do capítulo

Talvez você queira testar algumas das listagens de código de exemplo por si próprio, durante a leitura deste capítulo. Como as listagens de código neste capítulo consistem principalmente de padrões de expressões regulares, o teste dos exemplos envolve algumas etapas:

1. Crie um novo documento Flash.
2. Selecione um quadro-chave e abra o painel Ações
3. Crie uma variável RegExp (expressão regular) como esta:

```
var pattern:RegExp = /ABC/;
```

4. Copie o padrão do exemplo e atribua-o como o valor da variável RegExp. Por exemplo, na linha de código anterior, o padrão é a parte do código à direita do sinal de igual, sem incluir o ponto-e-vírgula (`/ABC/`).
5. Crie uma ou mais variáveis String contendo strings apropriadas para testar sua expressão regular. Por exemplo, se você estiver criando uma expressão regular para testar endereços de email válidos, crie algumas variáveis String contendo endereços de email válidos e incorretos.

```
var goodEmail:String = "bob@example.com";
var badEmail:String = "5@$2.99";
```

6 Adicione linhas de código para testar as variáveis String para determinar se elas correspondem ao padrão da expressão regular. Esses serão os valores que serão exibidos na tela utilizando a função `trace()` ou serão gravados em um campo de texto no Palco.

```
trace(goodEmail, " is valid:", pattern.test(goodEmail));
trace(badEmail, " is valid:", pattern.test(badEmail));
```

Por exemplo, considerando que `pattern` define o padrão da expressão regular para um endereço de email válido, as linhas de código anteriores gravam esse texto no painel Saída:

```
bob@example.com is valid: true
5@$2.99 is valid: false
```

Para obter mais informações sobre valores de teste, gravando os valores em uma ocorrência de campo de texto no Palco ou utilizando a função `trace()` para imprimir valores no painel Saída, consulte "Teste de listagens de código de exemplo dos capítulos" na página 36.

Sintaxe da expressão regular

Esta seção descreve todos os elementos da sintaxe de expressão regular do ActionScript. Como você verá, as expressões regulares podem ter muita complexidade e nuances. Você pode encontrar informações detalhadas sobre expressões regulares na Web e nas livrarias. Lembre-se de que diferentes ambientes de programação implementam expressões regulares de modos diferentes. O ActionScript 3.0 implementa expressões regulares como definido na especificação de idioma do ECMAScript Edição 3 (ECMA-262).

Normalmente, você usa expressões regulares que correspondem a padrões mais complicados do que uma string de caracteres simples. Por exemplo, a seguinte expressão regular define o padrão composto pelas letras A, B e C em seqüência seguida por qualquer dígito:

```
/ABC\d/
```

O código `\d` representa "qualquer dígito". O caractere de barra invertida (`\`) é chamado de caractere escape e é combinado ao caractere que o segue (nesse caso a letra d), tendo um significado especial na expressão regular. Este capítulo descreve essas seqüências de caracteres escape e outros recursos de sintaxe de expressões regulares.

A expressão regular a seguir define o padrão das letras ABC seguido por qualquer número de dígitos (observe o asterisco):

```
/ABC\d*/
```

O caractere asterisco (`*`) é um *metacaractere*. Um metacaractere é um caractere que tem significado especial nas expressões regulares. O asterisco é um tipo específico de metacaractere chamado de *quantificador* que é usado para quantificar o número de repetição de um caractere ou de um grupo de caracteres. Para obter mais informações, consulte "Quantificadores" na página 216.

Além desse padrão, uma expressão regular pode conter sinalizadores, que especificam como ela deve ser correspondida. Por exemplo, a seguinte expressão regular usa o sinalizador `i`, que especifica que a expressão regular não diferencia maiúsculas de minúsculas na correspondência de strings:

```
/ABC\d*/i
```

Para obter mais informações, consulte "Sinalizadores e propriedades" na página 221.

Você pode usar expressões regulares com os seguintes métodos da classe String: `match()`, `replace()` e `search()`. Para obter mais informações sobre esses métodos, consulte "Localização de padrões em strings e substituição de substrings" na página 149.

Criação de uma ocorrência de uma expressão regular

Há duas maneiras de criar uma ocorrência de expressão regular: Um modo usa caracteres de barra (/) para delinear a expressão regular; o outro usa o construtor `new`. Por exemplo, as expressões regulares a seguir são equivalentes:

```
var pattern1:RegExp = /bob/i;
var pattern2:RegExp = new RegExp("bob", "i");
```

As barras delineiam um literal da expressão regular da mesma forma que as aspas delineiam um literal de string. A parte da expressão regular entre as barras define o *padrão.* A expressão regular também pode incluir *sinalizadores* depois da barra delineadora final. Esses sinalizadores são considerados como parte da expressão regular, mas são separados do padrão.

Ao usar o construtor `new`, você usa duas strings para definir a expressão regular. A primeira string define o padrão e a segunda string define os sinalizadores como no exemplo a seguir:

```
var pattern2:RegExp = new RegExp("bob", "i");
```

Ao incluir uma barra *em* uma expressão regular que é definida utilizando os delineadores de barra, você deve preceder a barra com um caractere escape de barra invertida (\). Por exemplo, as expressões regulares a seguir correspondem ao padrão `1/2`:

```
var pattern:RegExp = /1\/2/;
```

Para incluir aspas *em* uma expressão regular que é definida com o construtor `new`, você deve adicionar uma caractere escape de barra invertida (\) antes das aspas (assim como faria ao definir qualquer literal String). Por exemplo, as expressões regulares a seguir correspondem ao padrão `eat at "joe's"`:

```
var pattern1:RegExp = new RegExp("eat at \"joe's\"", "");
var pattern2:RegExp = new RegExp('eat at "joe\'s"', "");
```

Não use o caractere escape de barra invertida com aspas em expressões regulares que são definidas utilizando os delineadores de barra. Da mesma forma, não use o caractere escape com barras em expressões regulares que são definidas utilizando o construtor `new`. As expressões regulares a seguir são equivalentes e definem o padrão `1/2 "joe's"`:

```
var pattern1:RegExp = /1\/2 "joe's"/;
var pattern2:RegExp = new RegExp("1/2 \"joe's\"", "");
var pattern3:RegExp = new RegExp('1/2 "joe\'s"', '');
```

Em uma expressão regular definida com o construtor `new`, para usar uma metaseqüência que comece com o caractere de barra invertida (\) , como `\d` (que corresponde a qualquer dígito), digite duas vezes esse caractere de barra invertida:

```
var pattern:RegExp = new RegExp("\\d+", ""); // matches one or more digits
```

É necessário digitar o caractere de barra invertida duas vezes nesse caso, pois o primeiro parâmetro do método do construtor `RegExp()` é uma string, e, em um literal de string, é necessário digitar um caractere de barra invertida duas vezes para que ele seja reconhecido como um único caractere de barra invertida.

As seções a seguir descrevem a sintaxe para definir os padrões de expressões regulares.

Para obter mais informações sobre sinalizadores, consulte "Sinalizadores e propriedades" na página 221.

Caracteres, metacaracteres e metaseqüências

A expressão regular mais simples é aquela que corresponde a uma seqüência de caracteres, como no exemplo a seguir:

```
var pattern:RegExp = /hello/;
```

Entretanto, os seguintes caracteres, conhecidos como metacaracteres , têm significado especial nas expressões regulares:

```
^ $ \ . * + ? ( ) [ ] { } |
```

Por exemplo, a expressão regular a seguir corresponde à letra A seguida por nenhuma ou mais ocorrências da letra B (o metacaractere asterisco indica essa repetição), seguida pela letra C:

```
/AB*C/
```

Para incluir um metacaractere sem seu significado especial em um padrão de expressão regular, você deve usar o caractere escape de barra invertida (\). Por exemplo, a seguinte expressão regular corresponde à letra A seguida pela letra B, seguida por um asterisco e pela letra C:

```
var pattern:RegExp = /AB\*C/;
```

Uma *metaseqüência*, como um metacaractere, tem um significado especial em uma expressão regular. Uma metaseqüência é composta por mais de um caractere. As seções a seguir fornecem detalhes sobre o uso de metacaracteres e metaseqüências.

Sobre metacaracteres

A tabela a seguir resume os metacaracteres que você pode usar em expressões regulares:

Metacaractere	Descrição
^ (circunflexo)	Correspondente ao início da string. Com o sinalizador `m` (`multiline`) definido, o circunflexo também corresponde ao início de uma linha (consulte "Sinalizadores e propriedades" na página 221). Observe que, quando utilizado no início de uma classe de caracteres, o circunflexo indica negação e não o início de uma string. Para obter mais informações, consulte "Classes de caracteres" na página 215.
$(sinal de dólar)	Correspondente ao fim da string. Com o conjunto de sinalizadores `m` (`multilinha`), `$` também corresponde à posição antes de um caractere de nova linha (`\n`). Para obter mais informações, consulte "Sinalizadores e propriedades" na página 221.
(barra invertida) \	Elimina o significado do metacaractere especial dos caracteres especiais. Além disso, use o caractere de barra invertida se você quiser utilizar o caractere de barra em um literal de expressão regular, como em `/1\/2/` (para corresponder ao caractere 1, seguido pelo caractere de barra e pelo caractere 2).
. (ponto)	Corresponde a qualquer caractere único. Um ponto corresponde a um caractere de nova linha (`\n`) apenas se o sinalizador `s` (`dotall`) está definido. Para obter mais informações, consulte "Sinalizadores e propriedades" na página 221.
* (estrela)	Corresponde ao item anterior repetido nenhuma ou várias vezes. Para obter mais informações, consulte "Quantificadores" na página 216.
+ (adição)	Corresponde ao item anterior repetido uma ou várias vezes. Para obter mais informações, consulte "Quantificadores" na página 216.
? (ponto de interrogação)	Corresponde ao item anterior repetido nenhuma ou uma vezes. Para obter mais informações, consulte "Quantificadores" na página 216.

Metacaractere	Descrição
`(` e `)`	Define grupos dentro de uma expressão regular. Use os grupos para: • Confinar o escopo do alternador |: `/(a|b|c)d/` • Definir o escopo de um quantificador: `/(walla.){1,2}/` • Em referências anteriores: Por exemplo, `\1` na expressão regular a seguir corresponde àquilo que correspondeu ao primeiro grupo entre parênteses do padrão: • `/(\w*) é repetido: \1/` Para obter mais informações, consulte "Grupos" na página 218.
`[` e `]`	Define uma classe de caracteres que define possíveis correspondências para um único caractere: `/[aeiou]/` corresponde a qualquer um dos caracteres especificados. Nas classes de caracteres, use o hífen (**-**) para designar um intervalo de caracteres: `/[A-Z0-9]/` corresponde às letras maiúsculas de A a Z ou de 0 a 9. Nas classes de caracteres, insira uma barra invertida para eliminar os caracteres] e -: `/[+\-]\d+/` corresponde a `+` ou `-` antes de um ou mais dígitos. Nas classes de caracteres, outros caracteres - normalmente metacaracteres -, não tratados como caracteres normais (e não metacaracteres), sem a necessidade de usar uma barra invertida: `/[$]/`£ corresponde a `$`ou £. Para obter mais informações, consulte "Classes de caracteres" na página 215.
| *(pipe)*	Utilizado para alternação, para corresponder à parte do lado esquerdo ou do lado direito: `/abc|xyz/` corresponde a `abc` ou`xyz`.

Sobre metaseqüências

As metaseqüências são seqüências de caracteres que têm significado especial em um padrão de expressão regular. A tabela a seguir descreve essas metaseqüências:

Metaseqüência	Descrição
{***n***} {***n***,} e {***n***,***n***}	Especifica um quantificador numérico ou intervalo de quantificador para o item anterior: `/A{27}/` corresponde ao caractere`A` repetido `27` vezes. `/A{3,}/` corresponde ao caractere`A` repetido `3` vezes. `/A{3,5}/` corresponde ao caractere `A` repetido de `3` a `5` vezes. Para obter mais informações, consulte "Quantificadores" na página 216.
`\b`	Corresponde à posição entre um caractere de palavra e um diferente de palavra. Se o primeiro ou o último caractere na string for um caractere de palavra, ele também corresponderá ao início ou ao fim da string.
`\B`	Corresponde à posição entre dois caracteres de palavra. Também corresponde à posição entre dois caracteres diferentes de palavra.
`\d`	Corresponde a um dígito decimal.
`\D`	Corresponde a qualquer caractere diferente de dígito.
`\f`	Corresponde ao caractere feed de formulário.
`\n`	Corresponde ao caractere de nova linha.

Metaseqüência	Descrição
\r	Corresponde ao caractere de retorno.
\s	Corresponde a qualquer caractere de espaço em branco (um caractere de espaço, tabulação, nova linha ou retorno).
\S	Corresponde a qualquer caractere diferente de um caractere de espaço em branco.
\t	Corresponde ao caractere de tabulação.
\u*nnnn*	Corresponde ao caractere Unicode com o código de caractere especificado pelo número hexadecimal *nn*. Por exemplo, \u263a é um caractere smiley.
\v	Corresponde ao caractere feed vertical.
\w	Corresponde a um caractere de palavra (A–Z, a–z, 0–9 ou _). Observe que \w não corresponde a caracteres diferentes de inglês, como é , ñ ou ç .
\W	Corresponde a qualquer caractere diferente de um caractere de palavra.
\\x*nn*	Corresponde ao caractere com o valor ASCII especificado, como definido pelo número hexadecimal *nn*.

Classes de caracteres

Você usa as classes de caracteres para especificar uma lista de caracteres que correspondem a uma posição na expressão regular. Você define as classes de caracteres com colchetes ([e]). Por exemplo, a expressão regular a seguir define uma classe de caracteres que corresponde a bag, beg, big, bog oubug:

```
/b[aeiou]g/
```

Seqüências de eliminação nas classes de caracteres

A maioria dos metacaracteres e das metaseqüências que normalmente tem significados especiais em uma expressão regular*não* tem esses mesmos significados em uma classe de caractere. Por exemplo, em uma expressão regular, o asterisco é usado para repetição, mas esse não é o caso quando o asterisco aparece em uma classe de caracteres. A classe de caracteres a seguir corresponde ao asterisco literalmente, junto com qualquer outro caractere listado:

```
/[abc*123]/
```

Entretanto, os três caracteres listados na tabela a seguir funcionam como metacaracteres, com significado especial, nas classes de caracteres:

Metacaractere	Significado nas classes de caracteres
]	Define o final de uma classe de caracteres.
-	Define um intervalo de caracteres (consulte a próxima seção, "Intervalos de caracteres nas classes de caracteres").
\	Define metaseqüências e elimina o significação especial dos metacaracteres.

Para que qualquer um desses caracteres seja reconhecido como caracteres literais (sem o significado do metacaractere especial), você deve precedê-lo com o caractere escape de barra invertida. Por exemplo, a expressão regular a seguir inclui uma classe de caracteres que corresponde a qualquer um dos quatro símbolos ($, \,] ou -):

```
/[$\\\]\-]/
```

Além dos metacaracteres que mantêm seu significado especial, as metaseqüências a seguir funcionam como metaseqüências nas classes de caracteres:

Metaseqüência	Significado nas classes de caracteres
`\n`	Corresponde a um caractere de nova linha.
`\r`	Corresponde a um caractere de retorno.
`\t`	Corresponde a um caractere de tabulação.
`\u`***nnnn***	Corresponde ao caractere com o valor do ponto de código Unicode especificado (como definido pelo número hexadecimal *nnnn*).
`\\x`***nn***	Corresponde ao caractere com o valor ASCII especificado (como definido pelo número hexadecimal *nn*).

Outros metacaracteres e metaseqüências de expressões regulares são tratados como caracteres normais em uma classe de caracteres.

Intervalos de caracteres nas classes de caracteres

Use o hífen para especificar um intervalo de caracteres, como `A-Z`, `a-z` ou`0-9`. Esses caracteres devem constituir um intervalo válido no conjunto de caracteres. Por exemplo, a classe de caracteres a seguir corresponde a qualquer um dos caracteres no intervalo de `a-z` ou qualquer dígito:

```
/[a-z0-9]/
```

Você também pode usar o código de caractere ASCII `\\x`*nn* para especificar um intervalo por valor ASCII. Por exemplo, a classe de caracteres a seguir corresponde a qualquer caractere de um conjunto de caracteres ASCII estendido (como `é` e `ê`):

```
\\x
```

Classes de caracteres negadas

Quando você usa um caractere circunflexo (`^`) no início de uma classe de caracteres, ele nega aquela classe — qualquer caractere não listado é considerado uma correspondência. A classe de caracteres a seguir corresponde a qualquer caractere *exceto* a uma letra minúscula (`az`–) ou um dígito:

```
/[^a-z0-9]/
```

Você deve digitar o caractere circunflexo (`^`) no *início* de uma classe de caracteres para indicar a negação. Caso contrário, você estará simplesmente adicionando o caractere circunflexo aos caracteres na classe de caracteres. Por exemplo, a classe de caracteres a seguir corresponde a qualquer um dos caracteres de símbolo, incluindo o circunflexo:

```
/[!.,#+*%$&^]/
```

Quantificadores

Você usa quantificadores para especificar repetições de caracteres ou seqüências nos padrões, como se segue:

Metacaractere de quantificador	Descrição
`*` (estrela)	Corresponde ao item anterior repetido nenhuma ou várias vezes.

Metacaractere de quantificador	**Descrição**
+ (adição)	Corresponde ao item anterior repetido uma ou várias vezes.
? (ponto de interrogação)	Corresponde ao item anterior repetido nenhuma ou uma vezes.
{*n*} {*n*,} e {*n*,*n*}	Especifica um quantificador numérico ou intervalo de quantificador para o item anterior: /A{27}/ corresponde ao caractere A repetido 27 vezes. /A{3,}/ corresponde ao caractere A repetido 3 ou mais vezes. /A{3,5}/ corresponde ao caractere A repetido de 3 a 5 vezes.

Você pode aplicar um quantificador a um único caractere, a uma classe de caracteres ou a um grupo:

- /a+/ corresponde ao caractere a repetido uma ou mais vezes.
- /\d+/ corresponde a um ou mais dígitos.
- /[abc]+/ corresponde a uma repetição de um ou mais caracteres, cada um é a, b ou c.
- /(very,)*/ corresponde à palavra very seguida por uma vírgula e um espaço repetido nenhuma ou várias vezes.

Você pode usar quantificadores em grupos entre parênteses que têm quantificadores aplicados a eles. Por exemplo, o quantificador a seguir corresponde a strings como word e word-word-word:

```
/\w+(-\w+)*/
```

Por padrão, as expressões regulares executam o que é conhecido como correspondência *greedy*. Qualquer subpadrão na expressão regular (como .*) tenta corresponder o máximo possível de caracteres na string antes de avançar para a próxima parte da expressão regular. Por exemplo, considere a seguinte expressão regular e string:

```
var pattern:RegExp = /<p>.*<\/p>/;
str:String = "<p>Paragraph 1</p> <p>Paragraph 2</p>";
```

A expressão regular corresponde à string inteira:

```
<p>Paragraph 1</p> <p>Paragraph 2</p>
```

Considere, entretanto, que você deseja corresponder apenas um grupo ... . É possível fazer isso da seguinte forma:

```
<p>Paragraph 1</p>
```

Adicione um ponto de interrogação (?) depois de qualquer quantificador para alterá-lo para o que conhecido como *quantificador lazy*. Por exemplo, a seguinte expressão regular, que usa o quantificador lazy *?, corresponde a seguido pelo número mínimo de caracteres possíveis (lazy) e por :

```
/<p>.*?<\/p>/
```

Lembre os seguintes pontos sobre quantificadores:

- Os quantificadores {0} e {0,0} não excluem um item de uma correspondência.
- Não combine vários quantificadores, como em /abc+*/.
- O ponto (.) não estende linhas a menos que o sinalizador s (dotall) esteja definido, mesmo se for seguido por um quantificador *. Por exemplo, considere o seguinte código:

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*<\/p>/;
trace(str.match(re)); // null;

re = /<p>.*<\/p>/s;
trace(str.match(re));
    // output: <p>Test
    //                 Multiline</p>
```

Para obter mais informações, consulte "Sinalizadores e propriedades" na página 221.

Alternação

Use o caractere | (pipe) em uma expressão regular para que o mecanismo de expressões regulares considere as alternativas para uma correspondência. Por exemplo, as expressões regulares a seguir correspondem a qualquer uma das palavras `cat, dog, pig, rat`:

```
var pattern:RegExp = /cat|dog|pig|rat/;
```

Você pode usar parênteses para definir grupos a fim de restringir o escopo do alternador |. A expressão regular a seguir corresponde a `cat` seguido por `nap` ou`nip`:

```
var pattern:RegExp = /cat(nap|nip)/;
```

Para obter mais informações, consulte "Grupos" na página 218.

As duas expressões regulares a seguir, uma usando o alternador `|` e a outra usando uma classe de caracteres (definida com `[` e `]`), são equivalentes:

```
/1|3|5|7|9/
/[13579]/
```

Para obter mais informações, consulte "Classes de caracteres" na página 215.

Grupos

Você pode especificar um grupo em uma expressão regular utilizando parênteses, como se segue:

```
/class-(\d*)/
```

Um grupo é uma subseção de um padrão. Você pode usar grupos para fazer as seguintes atividades:

- Aplicar um quantificador a mais de um caractere.
- Delinear subpadrões a serem aplicados com alternação (utilizando o caractere `|`).
- Capturar correspondências de substring (por exemplo, utilizando `\1` em uma expressão regular para corresponder um grupo com correspondência anterior ou utilizando `$1` de modo semelhante no método `replace()` da classe String).

As seções a seguir fornecem detalhes sobre o uso de grupos.

Uso de grupos com quantificadores

Se você não usar um grupo, um quantificador se aplicará ao caractere ou classe de caracteres que o precede, como mostrado a seguir:

```
var pattern:RegExp = /ab*/ ;
// matches the character a followed by
// zero or more occurrences of the character b

pattern = /a\d+/;
// matches the character a followed by
// one or more digits

pattern = /a[123]{1,3}/;
// matches the character a followed by
// one to three occurrences of either 1, 2, or 3
```

Entretanto, você pode usar um grupo para aplicar um quantificador a mais de um caractere ou classe de caracteres:

```
var pattern:RegExp = /(ab)*/;
// matches zero or more occurrences of the character a
// followed by the character b, such as abababab

pattern = /(a\d)+/;
// matches one or more occurrences of the character a followed by
// a digit, such as a1a5a8a3

pattern = /(spam ){1,3}/;
// matches 1 to 3 occurrences of the word spam followed by a space
```

Para obter mais informações sobre quantificadores, consulte "Quantificadores" na página 216.

Uso de grupos com o caractere alternador (|)

Você pode usar grupos para definir o grupo de caracteres aos quais deseja aplicar um caractere alternador (`|`), como se segue:

```
var pattern:RegExp = /cat|dog/;
// matches cat or dog

pattern = /ca(t|d)og/;
// matches catog or cadog
```

Uso de grupos para capturar correspondências de substrings

Quando terminar de definir um grupo entre parênteses padrão, poderá se referir a ele posteriormente na expressão regular. Isso é conhecido como *referência anterior*, e essas classificações de grupos são conhecidas como *capturas de grupos*. Por exemplo, na expressão regular a seguir, a seqüência `\1` corresponde a qualquer substring que correspondeu à captura do grupo entre parênteses:

```
var pattern:RegExp = /(\d+)-by-\1/;
// matches the following: 48-by-48
```

Você pode especificar até 99 dessas referências anteriores em uma expressão regular digitando `\1`, `\2`, ... , `\99`.

De modo semelhante, no método `replace()` da classe String, você pode usar `$1$99`– para inserir correspondência de substring de grupos capturados na string de substituição:

```
var pattern:RegExp = /Hi, (\w+)\./;
var str:String = "Hi, Bob.";
trace(str.replace(pattern, "$1, hello."));
    // output: Bob, hello.
```

Além disso, se você usar a captura de grupo, o método `exec()` da classe RegExp e o método `match()` da classe String retornarão substrings que correspondem à captura de grupos:

```
var pattern:RegExp = /(\w+)@(\w+).(\w+)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
    // bob@example.com,bob,example,com
```

Uso de grupos de não captura e grupos lookahead

Um grupo de não captura é aquele que é usado para agrupamento apenas; ele não é "coletado" e não corresponde a referências anteriores numeradas. Use `(?: e )` para definir grupos de não captura, como se segue:

```
var pattern = /(?:com|org|net);
```

Por exemplo, observe a diferença entre colocar `(com|org)` em uma captura versus um grupo de não captura (o método `exec()` lista os grupos de captura depois da conclusão da correspondência):

```
var pattern:RegExp = /(\w+)@(\w+).(com|org)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
// bob@example.com,bob,example,com

//noncapturing:
var pattern:RegExp = /(\w+)@(\w+).(?:com|org)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
    // bob@example.com,bob,example
```

Um tipo especial de grupo de não captura é o *grupo lookahead*, composto por dois tipos: o *grupo lookahead positivo* e o *grupo lookahead negativo*.

Use `(?=` e `)` para definir um grupo lookahead positivo, que especifica que o subpadrão no grupo deve corresponder à posição. Entretanto, a porção da string que corresponde ao grupo lookahead positivo pode corresponder aos padrões restantes na expressão regular. Por exemplo, como `(?=e)` é um grupo lookahead positivo no código a seguir, o caractere `e` ao qual ele corresponde pode ser correspondido por uma parte subseqüente da expressão regular — nesse caso, o grupo de captura, `\w*)`:

```
var pattern:RegExp = /sh(?=e)(\w*)/i;
var str:String = "Shelly sells seashells by the seashore";
trace(pattern.exec(str));
// Shelly,elly
```

Use `(?!` e `)` para definir um grupo lookahead negativo, que especifica que o subpadrão no grupo *não* deve corresponder à posição. Por exemplo:

```
var pattern:RegExp = /sh(?!e)(\w*)/i;
var str:String = "She sells seashells by the seashore";
trace(pattern.exec(str));
// shore,ore
```

Uso de grupos nomeados

Um grupo nomeado é um tipo de grupo em uma expressão regular que tem um identificador nomeado. Use `(?P<name>` e `)` para definir um grupo nomeado. Por exemplo, as expressões regulares a seguir incluem um grupo nomeado com os `dígitos` nomeados do identificador:

```
var pattern = /[a-z]+(?P<digits>\d+)[a-z]+/;
```

Quando você use o método `exec()`, uma correspondência de grupo nomeado é adicionada como uma propriedade da matriz `result`:

```
var myPattern:RegExp = /([a-z]+)(?P<digits>\d+)[a-z]+/;
var str:String = "a123bcd";
var result:Array = myPattern.exec(str);
trace(result.digits); // 123
```

Aqui está outro exemplo, que usa dois grupos nomeados, com os identificadores `name` e `dom`:

```
var emailPattern:RegExp =
    /(?P<name>(\w|[_.\-])+)@(?P<dom>((\w|-)+))+\.\w{2,4}+/;
var address:String = "bob@example.com";
var result:Array = emailPattern.exec(address);
trace(result.name); // bob
trace(result.dom); // example
```

Nota: *Os grupos nomeados não fazem parte da especificação de linguagem ECMAScript. Eles são um recurso adicionado no ActionScript 3.0.*

Sinalizadores e propriedades

A tabela a seguir lista os cinco sinalizadores que você pode definir para expressões regulares: Cada sinalizador pode ser acessado como uma propriedade do objeto da expressão regular.

Sinalizador	Propriedade	Descrição
`g`	`global`	Corresponde a mais de uma correspondência.
`i`	`ignoreCase`	Correspondência que não faz distinção entre maiúsculas e minúsculas. Aplica-se aos caracteres `A`—`Z` e `a`—`z`, mas não a caracteres estendidos como `É` e `é`.
`m`	`multiline`	Com esse sinalizador definido, `$` e `^` pode corresponder ao início e ao fim de uma linha, respectivamente.
`s`	`dotall`	Com esse sinalizador definido, `.` (ponto) pode corresponder ao caractere de nova linha (`\n`).
`x`	`extended`	Permite expressões regulares estendidas. Você pode digitar espaços na expressão regular, que serão ignorados como parte do padrão. Isso permite digitar código de expressão regular de modo mais legível.

Observe que essas propriedade são somente leitura. Você pode definir os sinalizadores (`g`, `i`, `m`, `s`, `x`) quando definir uma variável de expressão regular, como se segue:

```
var re:RegExp = /abc/gimsx;
```

Entretanto, não é possível definir diretamente as propriedades nomeadas. Por exemplo, o código a seguir resulta em um erro:

```
var re:RegExp = /abc/;
re.global = true; // This generates an error.
```

Por padrão, a menos que você os especifique na declaração de expressão regular, os sinalizadores não serão definidos e as propriedades de correspondência também são definidas como `false`.

De modo adicional, há duas outras propriedades de uma expressão regular:

- A propriedade `lastIndex` especifica a posição de índice na string para a próxima chamada do método `exec()` or `test()` de uma expressão regular.
- A propriedade `source` especifica a string que define a parte padrão da expressão regular.

O sinalizador g (global)

Quando o sinalizador `g` (`global`) *não* é incluído, uma expressão regular tem não mais do que uma correspondência. Por exemplo, com o sinalizador `g` não incluso na expressão regular, o método `String.match()` retorna somente uma substring de correspondência:

```
var str:String = "she sells seashells by the seashore.";
var pattern:RegExp = /sh\w*/;
trace(str.match(pattern)) // output: she
```

Quando o sinalizador `g` é definido, o método `Sting.match()` retorna várias correspondências, como se segue:

```
var str:String = "she sells seashells by the seashore.";
var pattern:RegExp = /sh\w*/g;
// The same pattern, but this time the g flag IS set.
trace(str.match(pattern)); // output: she,shells,shore
```

O sinalizador i (ignoreCase)

Por padrão, as correspondências de expressões regulares fazem distinção entre maiúsculas e minúsculas. Quando você define o sinalizador `i` (`ignoreCase`), a distinção entre maiúsculas e minúsculas é ignorada. Por exemplo, o `s` minúsculo na expressão regular não corresponde ao `S` maiúsculo, o primeiro caractere da string:

```
var str:String = "She sells seashells by the seashore.";
trace(str.search(/sh/)); // output: 13 -- Not the first character
```

Com o sinalizador `i` definido, entretanto, a expressão regular corresponde ao `S` maiúsculo:

```
var str:String = "She sells seashells by the seashore.";
trace(str.search(/sh/i)); // output: 0
```

O sinalizador `i` ignora a distinção entre maiúsculas e minúsculas somente para os caracteres `A–Z` e `a–z`, mas não para caracteres estendidos como `É` e `é`.

O sinalizador m (multiline)

Se o sinalizador `m` (`multiline`) não estiver definido, `^` corresponde ao início da string e `$` corresponde ao fim da string. Se o sinalizador `m` estiver definido, esses caracteres corresponderão ao início e ao fim de uma linha, respectivamente. Considere a seguinte string, que inclui um caractere de nova linha:

```
var str:String = "Test\n";
str += "Multiline";
trace(str.match(/^\w*/g)); // Match a word at the beginning of the string.
```

Mesmo que o sinalizador `g` (`global`) esteja definido na expressão regular, o método `match()` corresponde a apenas uma substring, pois há apenas uma correspondência para `^` — o início da string. A saída é:

```
Test
```

Aqui está o mesmo código com o sinalizador `m` definido:

```
var str:String = "Test\n";
str += "Multiline";
trace(str.match(/^\w*/gm)); // Match a word at the beginning of lines.
```

Neste momento, a saída inclui as palavras no início das linhas:

```
Test,Multiline
```

Observe que apenas o caractere `\n` sinaliza o fim de uma linha. Os caracteres a seguir não:

- Caractere de retorno (`\r`)

- Caractere Unicode separador de linha (\u2028)
- Caractere Unicode separador de parágrafo (\u2028)

O sinalizador s (dotall)

Se o sinalizador s (dotall ou "dot all") não estiver definido, um ponto (.) em um padrão de expressão regular não corresponde a um caractere de nova linha (\n). Portanto para o exemplo a seguir, não há nenhuma correspondência:

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*?<\/p>/;
trace(str.match(re));
```

Entretanto, se o sinalizador s estiver definido, o ponto corresponderá ao caractere de nova linha:

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*?<\/p>/s;
trace(str.match(re));
```

Nesse caso, a correspondência é a substring inteira dentro das tags <p>, incluindo o caractere de nova linha:

```
<p>Test
Multiline</p>
```

O sinalizador x (extended)

As expressões regulares podem ser difíceis de ler, especialmente quando incluem muitos metasímbolos e metaseqüências. Por exemplo:

```
/<p(>|(\s*[^>]*>)).*?<\/p>/gi
```

Quando você usa o sinalizador x (extended) em uma expressão regular, qualquer espaço em branco digitado no padrão será ignorado. Por exemplo, a expressão regular a seguir é idêntica ao exemplo anterior:

```
/     <p    (>  |  (\s*  [^>]*  >))    .*?    <\/p>  /gix
```

Se o sinalizador x estiver definido e você quiser uma correspondência com o caractere de espaço em branco, preceda o espaço em branco com uma barra invertida. Por exemplo, as duas expressões regulares a seguir são equivalentes:

```
/foo bar/
/foo \ bar/x
```

A propriedade lastIndex

A propriedade lastIndex especifica a posição de índice na string no qual a próxima pesquisa será iniciada. Essa propriedade afeta os métodos exec() e test() chamados em uma expressão regular que tem o sinalizador g definido como true. Por exemplo, considere o seguinte código:

```
var pattern:RegExp = /p\w*/gi;
var str:String = "Pedro Piper picked a peck of pickled peppers.";
trace(pattern.lastIndex);
var result:Object = pattern.exec(str);
while (result != null)
{
    trace(pattern.lastIndex);
    result = pattern.exec(str);
}
```

A propriedade `lastIndex` é definida como 0 por padrão (para iniciar a pesquisa no início da string). Depois de cada correspondência, ela é definida para a posição de índice seguindo a correspondência. Portanto, a saída para o código precedente é a seguinte:

```
0
5
11
18
25
36
44
```

Se o sinalizador `global` estiver definido como `false`, os métodos `exec()` e `test()` não usam nem definem a propriedade `lastIndex`.

Os métodos `match()`, `replace()` e `search()` da classe String iniciam todas as pesquisas no início da string, independentemente da configuração da propriedade `lastIndex` da expressão regular utilizada na chamada do método. (Contudo, o método `match()` define `lastIndex` como 0.)

Você pode definir a propriedade `lastIndex` para ajustar a posição inicial na string para a correspondência da expressão regular.

A propriedade source

A propriedade `source` especifica a string que define a parte padrão de uma expressão regular. Por exemplo:

```
var pattern:RegExp = /foo/gi;
trace(pattern.source); // foo
```

Métodos para usar expressões regulares com strings

A classe RegExp inclui dois métodos: `exec()` e`test()`.

Além dos métodos `exec()` e `test()` da classe RegExp, a classe String inclui os seguintes métodos que permitem corresponder expressões regulares em strings: `match()`, `replace()`, `search()` e`splice()`.

O método test()

O método `test()` da classe RegExp verifica simplesmente a string fornecida para ver se ela contém uma correspondência para a expressão regular, como mostra o exemplo a seguir:

```
var pattern:RegExp = /Class-\w/;
var str = "Class-A";
trace(pattern.test(str)); // output: true
```

O método exec()

O método `exec()` da classe RegExp verifica a string fornecida quanto a uma correspondência da expressão regular e retorna uma matriz com o seguinte:

- A substring de correspondência
- Correspondência de substring para qualquer grupo entre parênteses na expressão regular

A matriz também inclui uma propriedade `index`, indicando a posição de índice do início da correspondência de substring.

Por exemplo, considere o seguinte código:

```
var pattern:RegExp = /\d{3}\-\d{3}-\d{4}/; //U.S phone number
var str:String = "phone: 415-555-1212";
var result:Array = pattern.exec(str);
trace(result.index, " - ", result);
// 7-415-555-1212
```

Use o método `exec()` várias vezes para corresponder várias substrings quando o sinalizador `g` (`global`) está definido para uma expressão regular:

```
var pattern:RegExp = /\w*sh\w*/gi;
var str:String = "She sells seashells by the seashore";
var result:Array = pattern.exec(str);

while (result != null)
{
    trace(result.index, "\t", pattern.lastIndex, "\t", result);
    result = pattern.exec(str);
}
//output:
// 0  3  She
// 10  19  seashells
// 27  35  seashore
```

Métodos String que usam parâmetros RegExp

Os seguintes métodos da classe String usam as expressões regulares como parâmetros: `match()`, `replace()`, `search()` e`split()`. Para obter mais informações sobre esses métodos, consulte "Localização de padrões em strings e substituição de substrings" na página 149.

Exemplo: Um analisador Wiki

Esse exemplo simples de conversão de texto Wiki ilustra vários usos para expressões regulares:

- Conversão de linhas de texto que correspondem o padrão Wiki de origem às strings de saída HTML apropriadas.
- Uso de uma expressão regular para converter padrões de URL para tags de hiperlinks HTML `<a>`.
- Uso de uma expressão regular para converter strings com dólares norte-americanos (como `"$9,95"`) em strings com euros (como `"8,24 €"`).

Para obter os arquivos de aplicativo deste exemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_br. Os arquivos de aplicativo WikiEditor podem ser encontrados na pasta Samples/WikiEditor. O aplicativo consiste nos seguintes arquivos:

Arquivo	Descrição
WikiEditor.mxml ou WikiEditor.fla	O arquivo principal do aplicativo no Flash (FLA) ou no Flex (MXML).
com/example/programmingas3/regExpExamples/WikiParser.as	Uma classe que inclui métodos que usam expressões regulares para converter padrões de texto de entrada Wiki em saída HTML equivalente.
com/example/programmingas3/regExpExamples/URLParser.as	Uma classe que inclui métodos que usam expressões regulares para converter strings URL para tags de hiperlinks HTML `<a>`.
com/example/programmingas3/regExpExamples/CurrencyConverter.as	Uma classe que inclui métodos que usam expressões regulares para converter strings de dólar americano em strings de euro.

Definição da classe WikiParser

A classe WikiParser inclui métodos que convertem texto de entrada Wiki em saída HTML equivalente. Esse não é um aplicativo de conversão Wiki muito robusto, mas ele ilustra alguns bons usos de expressões regulares para correspondência de padrão e conversão de strings.

A função de construtor, junto com o método `setWikiData()`, simplesmente inicializa uma string de amostra do texto de entrada Wiki, como se segue:

```
public function WikiParser()
{
    wikiData = setWikiData();
}
```

Quando o usuário clica no botão Testar no aplicativo de amostra, o aplicativo chama o método `parseWikiString()` do objeto WikiParser. Esse método chama vários outros métodos, que por sua vez montam a string HTML resultante.

```
public function parseWikiString(wikiString:String):String
{
    var result:String = parseBold(wikiString);
    result = parseItalic(result);
    result = linesToParagraphs(result);
    result = parseBullets(result);
    return result;
}
```

Cada um dos métodos chamados — `parseBold()`, `parseItalic()`, `linesToParagraphs()` e `parseBullets()` — usa o método `replace()` da string para substituir os padrões de correspondência, definidos por uma expressão regular, para transformar o texto de entrada Wiki em texto no formato HTML.

Conversão de padrões negrito e itálico

O método `parseBold()` procura padrão de texto negrito Wiki (como `'''foo'''`) e o transforma em seu equivalente em HTML (como `<b>foo</b>`), como se segue:

```
private function parseBold(input:String):String
{
    var pattern:RegExp = /'''(.*?)'''/g;
    return input.replace(pattern, "<b>$1</b>");
}
```

Observe que a parte `(.?*)` de uma pressão regular corresponde a vários caracteres (`*`) entre os dois padrões de definição`'''`. O quantificador `?` torna a correspondência não greedy, para que uma string como `'''aaa''' bbb '''ccc'''`, a primeira string correspondida será `'''aaa'''` e não a string inteira (que começa e termina com o padrão `'''`).

Os parênteses na expressão regular definem um grupo de captura, e o método `replace()` se refere a esse grupo utilizando o código `$1` na string de substituição. O sinalizador `g` (`global`) na expressão regular garante que o método `replace()` substitua todas as correspondências na string (não simplesmente a primeira).

O método `parseItalic()` funciona de forma semelhante ao método `parseBold()`, exceto pelo fato de ele verificar dois apóstrofes (`''`) como o delimitador para texto itálico (não três):

```
private function parseItalic(input:String):String
{
    var pattern:RegExp = /''(.*?)''/g;
    return input.replace(pattern, "<i>$1</i>");
}
```

Conversão de padrões de bullet

Como mostra o exemplo a seguir, o método `parseBullet()` procura o padrão de linha de bullet Wiki (como `* foo`) e o transforma em seu equivalente HTML (como `<li>foo</li>`):

```
private function parseBullets(input:String):String
{
    var pattern:RegExp = /^\*(.*)/gm;
    return input.replace(pattern, "<li>$1</li>");
}
```

O símbolo `^` no início de uma expressão regular corresponde ao início de uma linha. O sinalizador `m` (`multiline`) na expressão regular faz com que essa expressão corresponda o símbolo `^` ao início de uma linha, e não simplesmente ao início da string.

O padrão `\*` corresponde a um caractere asterisco (a barra invertida é usada para sinalizar um asterisco literal em vez de um quantificador `*`).

Os parênteses na expressão regular definem um grupo de captura, e o método `replace()` se refere a esse grupo utilizando o código `$1` na string de substituição. O sinalizador `g` (`global`) na expressão regular garante que o método `replace()` substitua todas as correspondências na string (não simplesmente a primeira).

Conversão de padrões Wiki de parágrafos

O método `linesToParagraphs()` converte cada linha na string Wiki de entrada em uma tag de parágrafo HTML `<p>`. Essas linhas no método retiram linhas vazias da string Wiki de entrada:

```
var pattern:RegExp = /^$/gm;
var result:String = input.replace(pattern, "");
```

Os símbolos `^` e `$` de uma expressão regular correspondem ao início e ao fim de uma linha. O sinalizador `m` (`multiline`) na expressão regular faz com que essa expressão corresponda o símbolo ^ ao início de uma linha, e não simplesmente ao início da string.

O método `replace()` substitui todas as substrings correspondentes (linhas vazias) por uma string vazia (`""`). O sinalizador `g` (`global`) na expressão regular garante que o método `replace()` substitua todas as correspondências na string (não simplesmente a primeira).

Conversão de URLs para tags HTML <a>

Quando o usuário clica no botão Testar no aplicativo de amostra, se ele marcou a caixa de seleção `urlToATag`, o aplicativo chama o método estático `URLParser.urlToATag()` para converter as strings URL da string Wiki de entrada em tags HTML `<a>`.

```
var protocol:String = "((?:http|ftp)://)";
var urlPart:String = "([a-z0-9_-]+\.[a-z0-9_-]+)";
var optionalUrlPart:String = "(\.[a-z0-9_-]*)";
var urlPattern:RegExp = new RegExp(protocol + urlPart + optionalUrlPart, "ig");
var result:String = input.replace(urlPattern, "<a href='$1$2$3'><u>$1$2$3</u></a>");
```

A função do construtor `RegExp()` é usada para montar uma expressão regular (`urlPattern`) a partir de inúmeras partes constituintes. Essas partes constituintes são cada string que define parte do padrão da expressão regular.

A primeira parte do padrão da expressão regular, definida pela string `protocol`, define um protocolo de URL: `http://` ou `ftp://`. Os parênteses definem um grupo de não captura, indicado pelo símbolo `?`. Isso significa que os parênteses são usados simplesmente para definir um grupo para o padrão de alternação `|`; o grupo não corresponderá a códigos de referência anterior (`$1`, `$2`, `$3`) na string de substituição do método `replace()`.

As outras partes constituintes da expressão regular usam grupos de captura (indicado por parênteses no padrão), que são usados nos códigos de referência anterior (`$1`, `$2`, `$3`) na string de substituição do método `replace()`.

A parte do padrão definido pela string `urlPart` corresponde a *pelo menos* um destes caracteres: `a-z`, `0-9`, _ ou `-`. O quantificador `+` indica que pelo menos um caractere tem correspondência. `\.` indica um caractere de ponto (`.`) exigido. E o restante corresponde a outra string de pelo menos um destes caracteres: `a-z`, `0-9`, _ ou `-`.

A parte do padrão definido pela string `optionalUrlPart` corresponde a *nenhum ou mais* destes caracteres: um ponto (`.`) seguido por qualquer número de caracteres alfanuméricos (incluindo _ e `-`). O quantificador `*` indica que nenhum ou mais caracteres têm correspondência.

A chamada do método `replace()` aplica a expressão regular e monta a string HTML de substituição, utilizando referências anteriores.

O método `urlToATag()` chama o método `emailToATag()`, que usa técnicas semelhantes para substituir padrões de email por string de hiperlinks HTML `<a>`. As expressões regulares utilizadas para corresponder HTTP, FTP e URLs de email nesse arquivo de amostra são muito simples, com o objetivo de exemplificação; há expressões regulares muito mais complicadas para correspondência com esses URLs.

Conversão de strings de dólar americano para strings de euro

Quando o usuário clica no botão Testar do aplicativo de exemplo, se ele marcou a caixa de seleção `dollarToEuro`, o aplicativo chama o método estático `CurrencyConverter.usdToEuro()` para converter as strings com dólares norte-americanos (como `"$9,95"`) em strings com euros (como `"8,24 €"`), da seguinte maneira:

```
var usdPrice:RegExp = /\$([\d,]+.\d+)+/g;
return input.replace(usdPrice, usdStrToEuroStr);
```

A primeira linha define um padrão simples para correspondência de strings de dólar americano. Observe que o caractere `$` é precedido por um caractere escape de barra invertida (`\`).

O método `replace()` usa a expressão regular como o correspondente padrão e chama a função `usdStrToEuroStr()` para determinar a string de substituição (um valor em euros).

Quando um nome de função for utilizado como o parâmetro secundário do método `replace()`, o seguinte será transmitido como parâmetros para a função chamada:

- A parte correspondente da string.

- Qualquer correspondência de grupo em parênteses capturado. O número de argumentos transmitidos dessa maneira irá variar dependendo do número de correspondências de grupo entre parênteses capturado. É possível determinar o número de correspondências de grupo entre parênteses capturado, verificando `arguments.length -3` no código da função.
- A posição de índice na string em que a correspondência começa.
- A string completa.

O método `usdStrToEuroStr()` converte padrões de string de dólar americano para string de euro, como se segue:

```
private function usdToEuro(...args):String
{
    var usd:String = args[1];
    usd = usd.replace(",", "");
    var exchangeRate:Number = 0.828017;
    var euro:Number = Number(usd) * exchangeRate;
    trace(usd, Number(usd), euro);
    const euroSymbol:String = String.fromCharCode(8364); // €
    return euro.toFixed(2) + " " + euroSymbol;
}
```

Observe que `args[1]` representa o grupo entre parênteses capturado, correspondido pela expressão regular `usdPrice`. Essa é uma parte numérica da string de dólar americano: isto é, a quantidade de dólar sem o sinal `$`. O método aplica uma conversão de taxa de câmbio e retorna a string resultante (com o símbolo € à direita em vez do símbolo $ à esquerda).

Capítulo 11: Trabalho com XML

O ActionScript 3.0 inclui um grupo de classes com base na especificação ECMAScript para XML (E4X) (ECMA-357 edição 2). Essas classes incluem recursos avançados e fáceis de usar para trabalhar com dados XML. Usando o E4X, você poderá desenvolver códigos com dados XML mais rápido do que com as técnicas de programação anteriores. Além disso, o código produzido será mais fácil de ler.

Este capítulo descreve como usar o E4X para processar dados XML.

Noções básicas sobre XML

Introdução ao trabalho com XML

XML é uma maneira padrão de representar informações estruturadas com a qual os computadores devem trabalhar com facilidade e que deve ser relativamente fácil para as pessoas gravarem e entenderem. XML é uma abreviação de eXtensible Markup Language (Linguagem de markup extensível). O padrão XML está disponível em www.w3.org/XML/.

O XML oferece um modo prático e padrão de classificar dados, facilitando a leitura, o acesso e a manipulação. O XML usa estruturas de árvore e de tag similares às do HTML. Veja um exemplo simples de dados XML:

```
<song>
    <title>What you know?</title>
    <artist>Steve and the flubberblubs</artist>
    <year>1989</year>
    <lastplayed>2006-10-17-08:31</lastplayed>
</song>
```

Os dados XML também podem ser mais complexos, com tags aninhadas em outras tags, bem como em atributos e outros componentes estruturais. Veja um exemplo mais complexo de dados XML:

```
<album>
    <title>Questions, unanswered</title>
    <artist>Steve and the flubberblubs</artist>
    <year>1989</year>
    <tracks>
        <song tracknumber="1" length="4:05">
            <title>What do you know?</title>
            <artist>Steve and the flubberblubs</artist>
            <lastplayed>2006-10-17-08:31</lastplayed>
        </song>
        <song tracknumber="2" length="3:45">
            <title>Who do you know?</title>
            <artist>Steve and the flubberblubs</artist>
            <lastplayed>2006-10-17-08:35</lastplayed>
        </song>
        <song tracknumber="3" length="5:14">
            <title>When do you know?</title>
            <artist>Steve and the flubberblubs</artist>
            <lastplayed>2006-10-17-08:39</lastplayed>
        </song>
        <song tracknumber="4" length="4:19">
            <title>Do you know?</title>
            <artist>Steve and the flubberblubs</artist>
            <lastplayed>2006-10-17-08:44</lastplayed>
        </song>
    </tracks>
</album>
```

Observe que esse documento XML contém outras estruturas XML completas (como as tags `song` e seus filhos). Ele também demonstra outras estruturas XML como atributos (`tracknumber` e `length` nas tags `song`) e tags que contêm outras tags em vez de dados (como a tag `tracks`).

Como começar a usar o XML

Se você tiver pouca ou nenhuma experiência com XML, veja uma breve descrição dos aspectos mais comuns dos dados XML. Os dados XML são gravados em texto sem formatação, com uma sintaxe específica para organizar as informações em um formato estruturado. Em geral, um único conjunto de dados XML é conhecido como *documento XML*. No formato XML, os dados são organizados em *elementos* (que podem ser itens de dados únicos ou contêineres de outros elementos) usando uma estrutura hierárquica. Cada documento XML tem um único elemento como item de nível superior ou principal; dentro desse elemento raiz, pode existir uma única informação, embora provavelmente haja outros elementos que, por sua vez, contêm outros elementos e assim por diante. Por exemplo, esse documento XML contém as informações sobre um álbum de música:

```
<song tracknumber="1" length="4:05">
    <title>What do you know?</title>
    <artist>Steve and the flubberblubs</artist>
    <mood>Happy</mood>
    <lastplayed>2006-10-17-08:31</lastplayed>
</song>
```

Cada elemento é diferenciado por um conjunto de *tags* - o nome do elemento entre os sinais de menor do que e maior do que. A tag de abertura, que indica o início do elemento, tem o nome do elemento:

```
<title>
```

A tag de fechamento, que marca o final do elemento, tem uma barra antes do nome do elemento:

```
</title>
```

Se um elemento não tiver nenhum conteúdo, poderá ser gravado como um elemento vazio (às vezes chamado de elemento de fechamento automático). Em XML, esse elemento:

```
<lastplayed/>
```

é idêntico a este elemento:

```
<lastplayed></lastplayed>
```

Além do conteúdo do elemento contido entre as tags de abertura e fechamento, um elemento também pode incluir outros valores, conhecidos como *atributos*, definidos na tag de abertura. Por exemplo, este elemento XML define um único atributo chamado `length`, com o valor `"4:19"` :

```
<song length="4:19"></song>
```

Cada elemento XML tem conteúdo, que pode ser um único valor, um ou mais elementos XML ou nada (para um elemento vazio).

Mais informações sobre XML

Para saber mais sobre como trabalhar com XML, existem diversos outros livros e recursos, incluindo estes sites:

- Tutorial W3Schools XML: http://w3schools.com/xml/
- XML.com: http://www.xml.com/
- Tutoriais da XMLpitstop, listas de discussão e muito mais: http://xmlpitstop.com/

Classes do ActionScript para trabalhar com XML

O ActionScript 3.0 inclui várias classes que são usadas para trabalhar com informações estruturadas como XML. As duas classes principais são as seguintes:

- XML: representa um único elemento XML, que pode ser um documento XML com vários filhos ou um elemento com um único valor em um documento.
- XMLList: representa um conjunto de elementos XML. Um objeto XMLList é usado quando existem vários elementos XML que são "irmãos" (no mesmo nível e contidos pelo mesmo pai na hierarquia de documento XML). Por exemplo, uma ocorrência de XMLList seria o modo mais fácil de trabalhar com este conjunto de elementos XML (supostamente contidos em um documento XML):

```
<artist type="composer">Fred Wilson</artist>
<artist type="conductor">James Schmidt</artist>
<artist type="soloist">Susan Harriet Thurndon</artist>
```

Para usos mais avançados que envolvem espaços para nomes XML, o ActionScript também inclui as classes Namespace e QName. Para obter mais informações, consulte “Uso de espaços para nomes XML” na página 245.

Além das classes internas para trabalhar com XML, o ActionScript 3.0 também inclui vários operadores que fornecem recursos específicos para acessar e manipular dados XML. Essa abordagem de trabalhar com XML usando essas classes e operadores é conhecida como ECMAScript para XML (E4X), conforme definido pela especificação ECMA-357 edição 2.

Tarefas comuns de XML

Ao trabalhar com XML no ActionScript, você provavelmente realizará as seguintes tarefas:

- Criação de documentos XML (adição de elementos e valores)
- Acesso a elementos, valores e atributos XML
- Filtragem (pesquisa) de elementos XML

- Consulta de um conjunto de elementos XML
- Conversão de dados entre classes XML e a classe String
- Trabalho com espaços para nomes XML
- Carregamento de arquivos XML externos

Conceitos e termos importantes

A lista de referência a seguir contém termos importantes usados neste capítulo:

- Elemento: um único item em um documento XML, identificado como o conteúdo contido entre uma tag inicial e uma tag final (incluindo as tags). Os elementos XML podem conter dados de texto ou outros elementos, ou podem ser vazios.
- Elemento vazio: um elemento XML que não contém nenhum elemento filho. Os elementos vazios geralmente são gravados como tags de fechamento (como `<element/>`).
- Documento: uma única estrutura XML. Um documento XML pode conter qualquer número de elementos (ou ser constituído por apenas um único elemento vazio); no entanto, um documento XML deve ter um elemento de nível superior que contém todos os outros elementos do documento.
- Nó: outro nome para um elemento XML.
- Atributo: um valor nomeado associado a um elemento que está gravado na tag de abertura do elemento no formato `attributename="value"`, em vez de estar gravado como um elemento filho separado aninhado no elemento.

Teste dos exemplos do capítulo

Talvez você queira testar algumas das listagens de código de exemplo por si próprio, durante a leitura deste capítulo. Basicamente, todas as listagens de código deste capítulo já incluem a chamada da função `trace()` adequada. Para testar as listagens de código deste capítulo:

1. Crie um documento do Flash vazio.
2. Selecione um quadro-chave na linha de tempo.
3. Abra o painel Ações e copie a listagem de código no painel Script.
4. Execute o programa usando o comando Controlar > Testar filme.

   Você verá os resultados da função `trace()` no painel Saída.

Essa e outras técnicas para testar as listagens de código de exemplo estão descritas em mais detalhes em "Teste de listagens de código de exemplo dos capítulos" na página 36.

A abordagem E4X em relação ao processamento de XML

A especificação ECMAScript para XML define um conjunto de classes e recursos para trabalhar com dados XML. Em conjunto, essas classes e recursos são conhecidos como *E4X*. O ActionScript 3.0 inclui as seguintes classes E4X: XML, XMLList, QName e Namespace.

Os métodos, as propriedades e os operadores das classes E4X foram desenvolvidos com os seguintes objetivos:

- Simplicidade - Sempre que possível, o E4X facilita a gravação e a compreensão do código para trabalhar com dados XML.
- Consistência - Os métodos e princípios por trás do E4X são consistentes internamente e com outras partes do ActionScript.

- Familiaridade - Você manipula os dados XML com operadores conhecidos, como o operador de ponto (.).

Nota: *Havia uma classe XML no ActionScript 2.0. No ActionScript 3.0, ela foi renomeada como XMLDocument, de modo que não entra em conflito com a classe XML do ActionScript 3.0 que faz parte do E4X. No ActionScript 3.0, as classes herdadas (XMLDocument, XMLNode, XMLParser e XMLTag) são incluídas no pacote flash.xml principalmente para dar suporte a versões anteriores. As novas classes do E4X são classes básicas; não é necessário importar um pacote para utilizá-las. Este capítulo não descreve em detalhes as classes XML herdadas do ActionScript 2.0. Para obter informações sobre elas, consulte o pacote flash.xml na Referência de componentes e linguagem do ActionScript 3.0.*

Veja um exemplo de manipulação dos dados com E4X:

```
var myXML:XML =
    <order>
        <item id='1'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
        <item id='2'>
            <menuName>fries</menuName>
            <price>1.45</price>
        </item>
    </order>
```

Normalmente, seu aplicativo carregará dados XML a partir de uma fonte externa, como um serviço da Web ou um feed RSS. No entanto, para simplificar, os exemplos deste capítulo atribuem dados XML como literais.

Como mostra o código a seguir, o E4X inclui alguns operadores intuitivos, como os operadores de ponto (.) e de identificador de atributo (@), para acessar propriedades e atributos no XML:

```
trace(myXML.item[0].menuName); // Output: burger
trace(myXML.item.(@id==2).menuName); // Output: fries
trace(myXML.item.(menuName=="burger").price); // Output: 3.95
```

Use o método `appendChild()` para atribuir um novo nó filho ao XML, como mostra o snippet a seguir:

```
var newItem:XML =
    <item id="3">
        <menuName>medium cola</menuName>
        <price>1.25</price>
    </item>

myXML.appendChild(newItem);
```

Use os operadores `@` e `.` não só para ler, mas também para atribuir dados do seguinte modo:

```
myXML.item[0].menuName="regular burger";
myXML.item[1].menuName="small fries";
myXML.item[2].menuName="medium cola";

myXML.item.(menuName=="regular burger").@quantity = "2";
myXML.item.(menuName=="small fries").@quantity = "2";
myXML.item.(menuName=="medium cola").@quantity = "2";
```

Use um loop `for` para percorrer os nós do XML do seguinte modo:

```
var total:Number = 0;
for each (var property:XML in myXML.item)
{
    var q:int = Number(property.@quantity);
    var p:Number = Number(property.price);
    var itemTotal:Number = q * p;
    total += itemTotal;
    trace(q + " " + property.menuName + " $" + itemTotal.toFixed(2))
}
trace("Total: $", total.toFixed(2));
```

Objetos XML

Um objeto XML pode representar um elemento, atributo, comentários, instrução de processamento ou elemento de texto XML.

Um objeto XML pode ter *conteúdo simples* ou *conteúdo complexo*. Um objeto XML que tem nós filho tem conteúdo complexo. Um objeto XML terá conteúdo simples se contiver um dos seguintes itens: um atributo, um comentário, uma instrução de processamento ou um nó de texto.

Por exemplo, o objeto XML a seguir tem conteúdo complexo, incluindo um comentário e uma instrução de processamento:

```
XML.ignoreComments = false;
XML.ignoreProcessingInstructions = false;
var x1:XML =
    <order>
        <!--This is a comment. -->
        <?PROC_INSTR sample ?>
        <item id='1'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
        <item id='2'>
            <menuName>fries</menuName>
            <price>1.45</price>
        </item>
    </order>
```

Como mostra o exemplo a seguir, agora você pode usar os métodos `comments()` e `processingInstructions()` para criar novos objetos XML, um comentário e uma instrução de processamento:

```
var x2:XML = x1.comments()[0];
var x3:XML = x1.processingInstructions()[0];
```

propriedades XML

A classe XML tem cinco propriedades estáticas:

- As propriedades `ignoreComments` e `ignoreProcessingInstructions` determinam se comentários ou instruções de processamento devem ser ignorados quando o objeto XML é analisado.
- A propriedade `ignoreWhitespace` determina se os caracteres de espaço em branco devem ser ignorados em tags de elemento e expressões incorporadas que são separadas somente por caracteres de espaço em branco.

- As propriedades `prettyIndenteprettyPrinting` são usadas para formatar o texto que é retornado pelos métodos `toString()` e `toXMLString()` da classe XML.

Para obter detalhes sobre essas propriedades, consulte a Referência de componentes e linguagem do ActionScript 3.0.

métodos XML

Os métodos a seguir permitem trabalhar com a estrutura hierárquica dos objetos XML:

- `appendChild()`
- `child()`
- `childIndex()`
- `children()`
- `descendants()`
- `elements()`
- `insertChildAfter()`
- `insertChildBefore()`
- `parent()`
- `prependChild()`

Os métodos a seguir permitem trabalhar com atributos de objetos XML:

- `attribute()`
- `attributes()`

Os métodos a seguir permitem trabalhar com propriedades de objetos XML:

- `hasOwnProperty()`
- `propertyIsEnumerable()`
- `replace()`
- `setChildren()`

Os métodos a seguir permitem trabalhar com nomes e espaços para nomes qualificados:

- `addNamespace()`
- `inScopeNamespaces()`
- `localName()`
- `name()`
- `namespace()`
- `namespaceDeclarations()`
- `removeNamespace()`
- `setLocalName()`
- `setName()`
- `setNamespace()`

Os métodos a seguir permitem trabalhar e determinar tipos específicos de conteúdo XML:

- `comments()`

- `hasComplexContent()`
- `hasSimpleContent()`
- `nodeKind()`
- `processingInstructions()`
- `text()`

Os métodos a seguir servem para a conversão em strings e a formatação de objetos XML:

- `defaultSettings()`
- `setSettings()`
- `settings()`
- `normalize()`
- `toString()`
- `toXMLString()`

Existem alguns métodos adicionais:

- `contains()`
- `copy()`
- `valueOf()`
- `length()`

Para obter detalhes sobre esses métodos, consulte a Referência de componentes e linguagem do ActionScript 3.0

Objetos XMLList

Uma ocorrência de XMLList representa uma coleção arbitrária de objetos XML. Ela contém documentos XML completos, fragmentos de XML ou os resultados de uma consulta XML.

Os métodos a seguir permitem trabalhar com a estrutura hierárquica dos objetos XMLList:

- `child()`
- `children()`
- `descendants()`
- `elements()`
- `parent()`

Os métodos a seguir permitem trabalhar com atributos de objetos XMLList:

- `attribute()`
- `attributes()`

Os métodos a seguir permitem trabalhar com propriedades XMLList:

- `hasOwnProperty()`
- `propertyIsEnumerable()`

Os métodos a seguir permitem trabalhar e determinar tipos específicos de conteúdo XML:

- `comments()`
- `hasComplexContent()`
- `hasSimpleContent()`
- `processingInstructions()`
- `text()`

Os métodos a seguir servem para a conversão em strings e a formatação do objeto XMLList:

- `normalize()`
- `toString()`
- `toXMLString()`

Existem alguns métodos adicionais:

- `contains()`
- `copy()`
- `length()`
- `valueOf()`

Para obter detalhes sobre esses métodos, consulte a Referência de componentes e linguagem do ActionScript 3.0

Para um objeto XMLList que contém exatamente um elemento XML, você pode usar todos os métodos e propriedades da classe XML porque um XMLList com um elemento XML é tratado do mesmo modo como um objeto XML. Por exemplo, no código a seguir, como `doc.div` é um objeto XMLList que contém um elemento, você pode usar o método `appendChild()` da classe XML:

```
var doc:XML =
        <body>
            <div>
                <p>Hello</p>
            </div>
        </body>;
doc.div.appendChild(<p>World</p>);
```

Para obter uma lista de propriedades e métodos XML, consulte "Objetos XML" na página 235.

Inicialização de variáveis XML

Você pode atribuir um literal XML a um objeto XML do seguinte modo:

```
var myXML:XML =
    <order>
        <item id='1'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
        <item id='2'>
            <menuName>fries</menuName>
            <price>1.45</price>
        </item>
    </order>
```

Como mostra o snippet a seguir, também é possível usar o construtor `new` para criar uma ocorrência de um objeto XML a partir de uma string que contém dados XML:

```
var str:String = "<order><item id='1'><menuName>burger</menuName>"
                            + "<price>3.95</price></item></order>";
var myXML:XML = new XML(str);
```

Se os dados XML da string não estiverem bem formados (por exemplo, se estiver faltando uma tag de fechamento), ocorrerá um erro de tempo de execução.

Você também pode transmitir os dados por referência (de outras variáveis) em um objeto XML, como mostra o exemplo a seguir:

```
var tagname:String = "item";
var attributename:String = "id";
var attributevalue:String = "5";
var content:String = "Chicken";
var x:XML = <{tagname} {attributename}={attributevalue}>{content}</{tagname}>;
trace(x.toXMLString())
    // Output: <item id="5">Chicken</item>
```

Para carregar os dados XML a partir de uma URL, use a classe URLLoader, como mostra o exemplo a seguir:

```
import flash.events.Event;
import flash.net.URLLoader;
import flash.net.URLRequest;

var externalXML:XML;
var loader:URLLoader = new URLLoader();
var request:URLRequest = new URLRequest("xmlFile.xml");
loader.load(request);
loader.addEventListener(Event.COMPLETE, onComplete);

function onComplete(event:Event):void
{
    var loader:URLLoader = event.target as URLLoader;
    if (loader != null)
    {
        externalXML = new XML(loader.data);
        trace(externalXML.toXMLString());
    }
    else
    {
        trace("loader is not a URLLoader!");
    }
}
```

Para ler dados XML a partir de uma conexão de soquete, use a classe XMLSocket. Para obter mais informações, consulte a entrada classe XMLSocket na Referência de componentes e linguagem do ActionScript 3.0.

Montagem e transformação de objetos XML

Use o método `prependChild()` ou o método `appendChild()` para adicionar uma propriedade ao início ou ao final de uma lista de propriedades do objeto XML, como mostra o exemplo a seguir:

```
var x1:XML = <p>Line 1</p>
var x2:XML = <p>Line 2</p>
var x:XML = <body></body>
x = x.appendChild(x1);
x = x.appendChild(x2);
x = x.prependChild(<p>Line 0</p>);
    // x == <body><p>Line 0</p><p>Line 1</p><p>Line 2</p></body>
```

Use o método `insertChildBefore()` ou o método `insertChildAfter()` para adicionar uma propriedade antes ou depois de uma propriedade especificada, do seguinte modo:

```
var x:XML =
    <body>
        <p>Paragraph 1</p>
        <p>Paragraph 2</p>
    </body>
var newNode:XML = <p>Paragraph 1.5</p>
x = x.insertChildAfter(x.p[0], newNode)
x = x.insertChildBefore(x.p[2], <p>Paragraph 1.75</p>)
```

Como mostra o exemplo a seguir, você também pode usar os operadores de chaves ({ e }) para transmitir dados por referência (de outras variáveis) ao criar objetos XML:

```
var ids:Array = [121, 122, 123];
var names:Array = [["Murphy","Pat"], ["Thibaut","Jean"], ["Smith","Vijay"]]
var x:XML = new XML("<employeeList></employeeList>");

for (var i:int = 0; i < 3; i++)
{
    var newnode:XML = new XML();
    newnode =
        <employee id={ids[i]}>
            <last>{names[i][0]}</last>
            <first>{names[i][1]}</first>
        </employee>;

    x = x.appendChild(newnode)
}
```

É possível atribuir propriedades e atributos a um objeto XML usando o operador =, conforme mostrado a seguir:

```
var x:XML =
    <employee>
        <lastname>Smith</lastname>
    </employee>
x.firstname = "Jean";
x.@id = "239";
```

Isso define o objeto XML `x` como o seguinte:

```
<employee id="239">
    <lastname>Smith</lastname>
    <firstname>Jean</firstname>
</employee>
```

Você pode usar os operadores + e += para concatenar objetos XMLList:

```
var x1:XML = <a>test1</a>
var x2:XML = <b>test2</b>
var xList:XMLList = x1 + x2;
xList += <c>test3</c>
```

Isso define o objeto XMLList `xList` como o seguinte:

```
<a>test1</a>
<b>test2</b>
<c>test3</c>
```

Como percorrer estruturas XML

Um dos recursos avançados do XML é fornecer dados aninhados complexos por meio de uma string linear de caracteres de texto. Ao carregar dados em um objeto XML, o ActionScript analisa os dados e carrega sua estrutura hierárquica na memória (ou envia um erro de tempo de execução se os dados XML não estiverem bem formados).

Os operadores e métodos dos objetos XML e XMLList facilitam o percurso pela estrutura de dados XML.

Use o operador de ponto (.) e o operador de acessador do descendente (..) para acessar as propriedades filho de um objeto XML. Considere o objeto XML a seguir:

```
var myXML:XML =
    <order>
        <book ISBN="0942407296">
            <title>Baking Extravagant Pastries with Kumquats</title>
            <author>
                <lastName>Contino</lastName>
                <firstName>Chuck</firstName>
            </author>
            <pageCount>238</pageCount>
        </book>
        <book ISBN="0865436401">
            <title>Emu Care and Breeding</title>
            <editor>
                <lastName>Case</lastName>
                <firstName>Justin</firstName>
            </editor>
            <pageCount>115</pageCount>
        </book>
    </order>
```

O objeto `myXML.book` é um objeto XMLList que contém as propriedades filho do objeto `myXML` chamado `book`. Esses dois objetos XML correspondem às duas propriedades `book` do objeto `myXML`.

O objeto `myXML..lastName` é um objeto XMLList que contém todas as propriedades do descendente com o nome `lastName`. Esses dois objetos XML correspondem às duas propriedades `lastName` do objeto `myXML`.

O objeto `myXML.book.editor.lastName` é um objeto XMLList que contém todos os filhos com o nome `lastName` dos filhos com o nome `editor` dos filhos com o nome `book` do objeto `myXML`: nesse caso, um objeto XMLList que contém apenas um objeto XML (a propriedade `lastName` com o valor "`Case`").

Acesso a nós pai e filho

O método `parent()` retorna o pai de um objeto XML.

Você pode usar os valores ordinais de índice de uma lista de filhos para acessar objetos filho específicos. Por exemplo, considere um objeto XML `myXML` que tem duas propriedades filho chamadas `book`. Cada propriedade filho chamada `book` tem um número de índice associado:

```
myXML.book[0]
myXML.book[1]
```

Para acessar um neto específico, você pode indicar números de índice para os nomes do filho e do neto:

```
myXML.book[0].title[0]
```

No entanto, se houver apenas um filho de `x.book[0]` com o nome `title`, você poderá omitir a referência de índice do seguinte modo:

```
myXML.book[0].title
```

Do mesmo modo, se houver apenas um filho de book do objeto `x`, e se esse objeto filho tiver apenas um objeto title, você poderá omitir as duas referências de índice assim:

```
myXML.book.title
```

É possível usar o método `child()` para navegar até os filhos com nomes baseados em uma variável ou expressão, como mostra o exemplo a seguir:

```
var myXML:XML =
        <order>
            <book>
                <title>Dictionary</title>
            </book>
        </order>;

var childName:String = "book";

trace(myXML.child(childName).title) // output: Dictionary
```

Acesso a atributos

Use o símbolo `@` (operador de identificador de atributo) para acessar atributos em um objeto XML ou XMLList, como mostra o código a seguir:

```
var employee:XML =
    <employee id="6401" code="233">
        <lastName>Wu</lastName>
        <firstName>Erin</firstName>
    </employee>;
trace(employee.@id); // 6401
```

Você pode usar o símbolo de caractere curinga `*` com o símbolo `@` para acessar todos os atributos de um objeto XML ou XMLList, como no código a seguir:

```
var employee:XML =
    <employee id="6401" code="233">
        <lastName>Wu</lastName>
        <firstName>Erin</firstName>
    </employee>;
trace(employee.@*.toXMLString());
// 6401
// 233
```

Você pode usar o método `attribute()` ou `attributes()` para acessar um atributo específico ou todos os atributos de um objeto XML ou XMLList, como no código a seguir:

```
var employee:XML =
    <employee id="6401" code="233">
        <lastName>Wu</lastName>
        <firstName>Erin</firstName>
    </employee>;
trace(employee.attribute("id")); // 6401
trace(employee.attribute("*").toXMLString());
// 6401
// 233
trace(employee.attributes().toXMLString());
// 6401
// 233
```

Também é possível usar a sintaxe a seguir para acessar atributos, como mostra o seguinte exemplo:

```
employee.attribute("id")
employee["@id"]
employee.@["id"]
```

Cada um é equivalente a `employee.@id`. No entanto, a sintaxe `employee.@id` é recomendada.

Filtragem por valor de elemento ou atributo

Você pode usar os operadores de parênteses - `(` e `)` - para filtrar elementos com um nome de elemento ou valor de atributo específico. Considere o objeto XML a seguir:

```
var x:XML =
    <employeeList>
        <employee id="347">
            <lastName>Zmed</lastName>
            <firstName>Sue</firstName>
            <position>Data analyst</position>
        </employee>
        <employee id="348">
            <lastName>McGee</lastName>
            <firstName>Chuck</firstName>
            <position>Jr. data analyst</position>
        </employee>
    </employeeList>
```

As seguintes expressões são válidas:

- `x.employee.(lastName == "McGee")` - É o segundo nó `employee`.
- `x.employee.(lastName == "McGee").firstName` - É a propriedade `firstName` do segundo nó `employee`.
- `x.employee.(lastName == "McGee").@id` - É o valor do atributo `id` do segundo nó `employee`.
- `x.employee.(@id == 347)` - O primeiro nó `employee`.
- `x.employee.(@id == 347).lastName` - É a propriedade `lastName` do primeiro nó `employee`.
- `x.employee.(@id > 300)` - É um objeto XMLList com as duas propriedades `employee`.
- `x.employee.(position.toString().search("analyst") > -1)` - É um objeto XMLList com as duas propriedades `position`.

Se você tentar filtrar atributos ou elementos que não existem, o Flash® Player e o Adobe® AIR™ lançarão uma exceção. Por exemplo, a linha final do código a seguir gera um erro porque não existe nenhum atributo `id` no segundo elemento `p`:

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(@id == '123'));
```

Do mesmo modo, a linha final do código a seguir gera um erro porque não existe nenhuma propriedade `b` do segundo elemento `p`:

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(b == 'Bob'));
```

Para evitar esses erros, você pode identificar as propriedades que têm atributos ou elementos correspondentes usando os métodos `attribute()` e `elements()`, assim como no código a seguir:

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(attribute('id') == '123'));
trace(doc.p.(elements('b') == 'Bob'));
```

Também é possível usar o método `hasOwnProperty()`, como no seguinte código:

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(hasOwnProperty('@id') && @id == '123'));
trace(doc.p.(hasOwnProperty('b') && b == 'Bob'));
```

Uso de for..in e for each..em instruções

O ActionScript 3.0 inclui a instrução `for..in` e a instrução `for each..in` para percorrer objetos XMLList. Por exemplo, considere o seguinte objeto XML, `myXML`, e o objeto XMLList, `myXML.item`. O objeto XMLList, `myXML.item`, consiste em dois nós `item` do objeto XML.

```
var myXML:XML =
    <order>
        <item id='1' quantity='2'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
        <item id='2' quantity='2'>
            <menuName>fries</menuName>
            <price>1.45</price>
        </item>
    </order>;
```

A instrução `for..in` permite percorrer um conjunto de nomes de propriedades em um objeto XMLList:

```
var total:Number = 0;
for (var pname:String in myXML.item)
{
    total += myXML.item.@quantity[pname] * myXML.item.price[pname];
}
```

A instrução `for each..in` permite percorrer as propriedades em um objeto XMLList:

```
var total2:Number = 0;
for each (var prop:XML in myXML.item)
{
    total2 += prop.@quantity * prop.price;
}
```

Uso de espaços para nomes XML

Os espaços para nomes em um objeto (ou documento) XML identificam o tipo de dados contido no objeto. Por exemplo, ao enviar e fornecer dados XML para um serviço da Web que usa o protocolo SOAP, você declara o espaço para nomes na tag de abertura do XML:

```
var message:XML =
    <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <soap:Body xmlns:w="http://www.test.com/weather/">
            <w:getWeatherResponse>
                <w:tempurature >78</w:tempurature>
            </w:getWeatherResponse>
        </soap:Body>
    </soap:Envelope>;
```

O espaço para nomes tem um prefixo, `soap`, e uma URI que define o espaço para nomes, `http://schemas.xmlsoap.org/soap/envelope/`.

O ActionScript 3.0 inclui a classe Namespace para trabalhar com espaços para nomes XML. Para o objeto XML do exemplo anterior, você pode usar a classe Namespace do seguinte modo:

```
var soapNS:Namespace = message.namespace("soap");
trace(soapNS); // Output: http://schemas.xmlsoap.org/soap/envelope/

var wNS:Namespace = new Namespace("w", "http://www.test.com/weather/");
message.addNamespace(wNS);
var encodingStyle:XMLList = message.@soapNS::encodingStyle;
var body:XMLList = message.soapNS::Body;

message.soapNS::Body.wNS::GetWeatherResponse.wNS::tempurature = "78";
```

A classe XML inclui os seguintes métodos para trabalhar com espaços para nomes: `addNamespace()`, `inScopeNamespaces()`, `localName()`, `name()`, `namespace()`, `namespaceDeclarations()`, `removeNamespace()`, `setLocalName()`, `setName()` e `setNamespace()`.

A diretiva de `espaço para nomes XML padrão` permite atribuir um espaço para nomes padrão para objetos XML. Por exemplo, a seguir, `x1` e `x2` têm o mesmo espaço para nomes padrão:

```
var ns1:Namespace = new Namespace("http://www.example.com/namespaces/");
default xml namespace = ns1;
var x1:XML = <test1 />;
var x2:XML = <test2 />;
```

Conversão de tipo XML

Você pode converter objetos XML e XMLList em valores de string. Do mesmo modo, é possível converter strings em objetos XML e XMLList. Além disso, tenha em mente que todos os valores de atributo, nomes e valores de texto XML são strings. As seções a seguir discutem todas essas formas de conversão de tipo XML.

Conversão de objetos XML e XMLList em strings

As classes XML e XMLList incluem um método `toString()` e um método `toXMLString()`. O método `toXMLString()` retorna uma string que inclui todas as tags, atributos, instruções de espaço para nomes e conteúdo do objeto XML. Para objetos XML com conteúdo complexo (elementos filho), o método `toString()` é exatamente igual ao método `toXMLString()`. Para objetos XML com conteúdo simples (aqueles que contêm apenas um elemento de texto), o método `toString()` retorna somente o conteúdo de texto do elemento, como mostra o exemplo a seguir:

```
var myXML:XML =
    <order>
        <item id='1' quantity='2'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
    <order>;

trace(myXML.item[0].menuName.toXMLString());
    // <menuName>burger</menuName>
trace(myXML.item[0].menuName.toString());
    // burger
```

Se o método `trace()` é usado sem especificar `toString()` ou `toXMLString()`, os dados são convertidos usando o método `toString()` por padrão, como mostra este código:

```
var myXML:XML =
    <order>
        <item id='1' quantity='2'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
    <order>;

trace(myXML.item[0].menuName);
    // burger
```

Ao usar o método `trace()` para depurar o código, você normalmente usará o método `toXMLString()` para que `trace()` gere dados mais completos.

Conversão de strings em objetos XML

É possível usar o construtor `new XML()` para criar um objeto XML a partir de uma string, do seguinte modo:

```
var x:XML = new XML("<a>test</a>");
```

Se você tentar converter uma string em XML a partir de uma string que representa um XML inválido ou mal formado, ocorrerá um erro de tempo de execução do seguinte modo:

```
var x:XML = new XML("<a>test"); // throws an error
```

Conversão de valores de atributo, nomes e valores de texto a partir de strings

Todos os valores de atributo, nomes e valores de texto XML são tipos de dados String e talvez seja necessário convertê-los em outros tipos de dados. Por exemplo, o código a seguir usa a função `Number()` para converter valores de texto em números:

```
var myXML:XML =
                        <order>
                            <item>
                                <price>3.95</price>
                            </item>
                            <item>
                                <price>1.00</price>
                            </item>
                        </order>;

var total:XML = <total>0</total>;
myXML.appendChild(total);

for each (var item:XML in myXML.item)
{
    myXML.total.children()[0] = Number(myXML.total.children()[0])
                                                + Number(item.price.children()[0]);
}
trace(myXML.total); // 4.35;
```

Se esse código não tivesse usado a função `Number()`, o código interpretaria o operador + como o operador de concatenação de string e o método `trace()` na última linha seria o seguinte:

```
01.003.95
```

Leitura de documentos XML externos

Você pode usar a classe URLLoader para carregar dados XML a partir de uma URL. Para usar o código a seguir em seus aplicativos, substitua o valor `XML_URL` do exemplo por uma URL válida:

```
var myXML:XML = new XML();
var XML_URL:String = "http://www.example.com/Sample3.xml";
var myXMLURL:URLRequest = new URLRequest(XML_URL);
var myLoader:URLLoader = new URLLoader(myXMLURL);
myLoader.addEventListener("complete", xmlLoaded);

function xmlLoaded(event:Event):void
{
    myXML = XML(myLoader.data);
    trace("Data loaded.");
}
```

Você também pode usar a classe XMLSocket para configurar uma conexão de soquete XML assíncrona com um servidor. Para obter mais informações, consulte a Referência de componentes e linguagem do ActionScript 3.0.

Exemplo: carregamento de dados RSS a partir da Internet

O aplicativo de exemplo RSSViewer mostra diversos recursos para trabalhar com XML no ActionScript, incluindo os seguintes:

- Uso dos métodos XML para percorrer dados XML em forma de um feed RSS.
- Uso dos métodos XML para montar dados XML em forma de HTML a ser usado em um campo de texto.

O formato RSS é muito utilizado para distribuir notícias via XML. Um arquivo de dados RSS simples pode ser parecido com o seguinte:

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0" xmlns:dc="http://purl.org/dc/elements/1.1/">
<channel>
    <title>Alaska - Weather</title>
    <link>http://www.nws.noaa.gov/alerts/ak.html</link>
    <description>Alaska - Watches, Warnings and Advisories</description>

    <item>
        <title>
            Short Term Forecast - Taiya Inlet, Klondike Highway (Alaska)
        </title>
        <link>
            http://www.nws.noaa.gov/alerts/ak.html#A18.AJKNK.1900
        </link>
        <description>
            Short Term Forecast Issued At: 2005-04-11T19:00:00
            Expired At: 2005-04-12T01:00:00 Issuing Weather Forecast Office
            Homepage: http://pajk.arh.noaa.gov
        </description>
</item>
    <item>
        <title>
            Short Term Forecast - Haines Borough (Alaska)
        </title>
            <link>
            http://www.nws.noaa.gov/alerts/ak.html#AKZ019.AJKNOWAJK.190000
        </link>
        <description>
            Short Term Forecast Issued At: 2005-04-11T19:00:00
            Expired At: 2005-04-12T01:00:00 Issuing Weather Forecast Office
            Homepage: http://pajk.arh.noaa.gov
        </description>
    </item>
</channel>
</rss>
```

O aplicativo SimpleRSS lê os dados RSS na Internet, analisa os dados em busca de cabeçalhos (títulos), links e descrições e retorna esses dados. A classe SimpleRSSUI fornece a interface de usuário e chama a classe SimpleRSS, que faz todo o processamento XML.

Para obter os arquivos de aplicativo deste exemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_br. Os arquivos do aplicativo RSSViewer estão localizados na pasta Amostras/RSSViewer. O aplicativo consiste nos seguintes arquivos:

Arquivo	Descrição
RSSViewer.mxml ou RSSViewer.fla	O arquivo principal do aplicativo no Flash (FLA) ou no Flex (MXML).
com/example/programmingas3/rssViewer/RSSParser.as	Uma classe que contém métodos que usam o E4X para percorrer dados RSS (XML) e gerar uma representação em HTML correspondente.
RSSData/ak.rss	Um arquivo RSS de exemplo. O aplicativo é configurado para ler dados RSS na Web, em um feed RSS do Flex hospedado pela Adobe. No entanto, você pode alterar o arquivo com facilidade para ler dados RSS neste documento, que usa um esquema ligeiramente diferente do feed RSS do Flex.

Leitura e análise de dados XML

A classe RSSParser inclui um método `xmlLoaded()` que converte os dados RSS de entrada, armazenados na variável `rssXML`, em uma string que contém a saída em formato HTML, `rssOutput`.

Logo do início do método, o código define o espaço para nomes XML padrão se os dados RSS de origem incluírem um espaço para nomes padrão:

```
if (rssXML.namespace("") != undefined)
{
    default xml namespace = rssXML.namespace("");
}
```

As próximas linhas percorrem o conteúdo dos dados XML de origem, examinando cada propriedade de descendente chamada `item`:

```
for each (var item:XML in rssXML..item)
{
    var itemTitle:String = item.title.toString();
    var itemDescription:String = item.description.toString();
    var itemLink:String = item.link.toString();
    outXML += buildItemHTML(itemTitle,
                            itemDescription,
                            itemLink);
}
```

As três primeiras linhas simplesmente definem variáveis de string para representar as propriedades de título, descrição e link da propriedade `item` dos dados XML. Em seguida, a próxima linha chama o método `buildItemHTML()` para obter os dados HTML em forma de um objeto XMLList, usando as três novas variáveis de string como parâmetros.

Montagem de dados XMLList

Os dados HTML (um objeto XMLList) têm uma das seguintes formas:

```
<b>itemTitle</b>
<p>
    itemDescription
    <br />
    <a href="link">
        <font color="#008000">More...</font>
    </a>
</p>
```

As primeiras linhas do método apagam o espaço para nomes XML padrão:

```
default xml namespace = new Namespace();
```

A diretiva de `espaço para nomes XML padrão` tem o escopo do nível de bloqueio da função. Isso significa que os escopo dessa instrução é o método `buildItemHTML()`.

As próximas linhas montam o XMLList, com base nos argumentos de string transmitidos para a função:

```
var body:XMLList = new XMLList();
body += new XML("<b>" + itemTitle + "</b>");
var p:XML = new XML("<p>" + itemDescription + "</p>");

var link:XML = <a></a>;
link.@href = itemLink; // <link href="itemLinkString"></link>
link.font.@color = "#008000";
        // <font color="#008000"></font></a>
        // 0x008000 = green
link.font = "More...";

p.appendChild(<br/>);
p.appendChild(link);
body += p;
```

Esse objeto XMLList representa dados de string adequados para um campo de texto HTML do ActionScript.

O método `xmlLoaded()` usa o valor de retorno do método `buildItemHTML()` e o converte em uma string:

```
XML.prettyPrinting = false;
rssOutput = outXML.toXMLString();
```

Extração do título do feed RSS e envio de um evento personalizado

O método `xmlLoaded()` define uma variável de string `rssTitle`, com base nas informações dos dados XML RSS de origem:

```
rssTitle = rssXML.channel.title.toString();
```

Finalmente, o método `xmlLoaded()` gera um evento, que informa ao aplicativo que os dados estão analisados e disponíveis:

```
            dataWritten = new Event("dataWritten", true);
```

Capítulo 12: Manipulação de eventos

Um sistema de manipulação de eventos permite que os programadores respondam à entrada do usuário e aos eventos do sistema de modo prático. Além de prático, o modelo de evento do ActionScript 3.0 está em conformidade com os padrões e está bem integrado às listas de exibição do Adobe® Flash® Player e do Adobe® AIR™. Com base na especificação de eventos DOM nível 3 e em uma arquitetura de manipulação de eventos padrão do setor, o novo modelo de evento fornece uma ferramenta poderosa e intuitiva para os programadores do ActionScript.

Este capítulo está organizado em cinco seções. As duas primeiras seções fornecem informações básicas sobre a manipulação de eventos no ActionScript. As três últimas seções descrevem os principais conceitos do modelo de evento: fluxo, objeto e ouvintes de evento. O sistema de manipulação de eventos do ActionScript 3.0 interage de perto com a lista de exibição. Este capítulo supõe que você tem noções básicas sobre a lista de exibição. Para obter mais informações, consulte “Programação de exibição” na página 274.

Noções básicas sobre a manipulação de eventos

Introdução à manipulação de eventos

Pense nos eventos como ocorrências de qualquer tipo no arquivo SWF que interessam a você como programador. Por exemplo, a maioria dos arquivos SWF oferece suporte a algum tipo de interação do usuário - seja algo simples, como responder ao clique do mouse, ou algo mais complexo, como aceitar e processar dados inseridos em um formulário. Toda interação do usuário com seu arquivo SWF é considerada um evento. Os eventos também podem ocorrer sem nenhuma interação direta do usuário, como quando os dados terminam de ser carregados a partir de um servidor ou quando uma câmera acoplada é ativada.

No ActionScript 3.0, cada evento é representado por um objeto, que é uma ocorrência da classe Event ou uma de suas subclasses. Um objeto de evento não só armazena informações sobre um evento específico, mas também contém métodos que facilitam a manipulação do objeto. Por exemplo, quando detecta um clique do mouse, o Flash Player ou o AIR cria um objeto de evento (uma ocorrência da classe MouseEvent) para representar esse evento específico de clique do mouse.

Depois de criar um objeto de evento, o Flash Player ou o AIR o *envia*, ou seja, o objeto de evento é transmitido para o objeto que é destino do evento. O objeto que é o destino de um objeto de evento enviado é chamado de *destino do evento*. Por exemplo, quando uma câmera acoplada é ativada, o Flash Player envia um objeto de evento diretamente ao destino que, nesse caso, é o objeto que representa a câmera. No entanto, se o destino do evento estiver na lista de exibição, o objeto será transmitido pela hierarquia da lista de exibição até atingir o destino do evento. Em alguns casos, o objeto de evento forma "bolhas" na hierarquia da lista de exibição ao longo da mesma rota. Essa profundidade da hierarquia da lista de exibição é chamada de *fluxo de evento*.

Você pode “ouvir” objetos de evento no seu código usando ouvintes de evento. *Ouvintes de evento* são as funções ou os métodos gravados para responder a eventos específicos. Para garantir que seu programa responda a eventos, adicione ouvintes ao destino do evento ou a qualquer objeto da lista de exibição que faça parte do fluxo de um objeto de evento.

Sempre que é gravado, o código do ouvinte de evento segue essa estrutura básica (os elementos em negrito são alocadores de espaço preenchidos de acordo com suas necessidades):

```
function eventResponse(eventObject:EventType):void
{
    // Actions performed in response to the event go here.
}

eventTarget.addEventListener(EventType.EVENT_NAME, eventResponse);
```

Esse código faz duas coisas. Primeiro, ele define uma função, que é a maneira de especificar as ações que serão executadas em resposta ao evento. Em seguida, o método `addEventListener()` do objeto de origem é chamado, basicamente "inscrevendo" a função do evento especificado para que, quando o evento acontecer, as ações da função sejam executadas. Quando o evento realmente acontece, o destino do evento verifica a lista de todos os métodos e funções registrados como ouvintes de evento. Cada um deles é chamado e o objeto de evento é transmitido como um parâmetro.

Para criar seu próprio ouvinte de evento, é necessário alterar quatro coisas nesse código. Primeiro, você deve dar à função o nome que deseja usar (essa alteração deve ser feita em dois lugares, onde aparece **eventResponse** no código). Segundo, você deve especificar o nome de classe adequado do objeto que é enviado pelo evento que deseja ouvir (**EventType** no código) e também deve especificar a constante correta para o evento em questão (**EVENT_NAME** na listagem). Terceiro, você deve chamar o método `addEventListener()` no objeto que enviará o evento (**eventTarget** neste código). Se desejar, altere o nome da variável usada como parâmetro da função (**eventObject** neste código).

Tarefas comuns de manipulação de eventos

As seguintes tarefas comuns de manipulação de eventos são descritas neste capítulo:

- Gravação do código para responder a eventos
- Interrupção do código em resposta aos eventos
- Trabalho com os objetos de evento
- Trabalho com o fluxo de evento:
  - Identificação de informações do fluxo de evento
  - Interrupção do fluxo de evento
  - Como evitar o comportamento padrão
- Envio de eventos a partir de suas classes
- Criação de um tipo de evento personalizado

Conceitos e termos importantes

A lista de referência a seguir contém termos importantes usados neste capítulo:

- Comportamento padrão: alguns eventos incluem um comportamento que normalmente ocorre ao longo do evento e é conhecido como comportamento padrão. Por exemplo, quando um usuário digita em um campo de texto, ocorre um evento de entrada de texto. O comportamento padrão desse evento é exibir o caractere que realmente foi digitado no campo de texto, mas você pode substituí-lo (se, por algum motivo, não desejar exibir o caractere digitado).
- Envio: para notificar o evento ocorrido para os ouvintes de evento.
- Evento: algo que acontece em um objeto e que pode ser informado para outros objetos.

- Fluxo de evento: quando os eventos acontecem em um objeto na lista de exibição (um objeto exibido na tela), todos os objetos que contêm o objeto em questão são informados sobre o evento e notificam seus ouvintes. Esse processo começa com o palco e continua na lista de exibição até o objeto real onde ocorreu o evento e, em seguida, retorna ao palco. Esse processo também é conhecido como fluxo de evento.
- Objeto de evento: um objeto que contém informações sobre a ocorrência de um evento específico, que é enviado para todos os ouvintes assim que o evento acontece.
- Destino do evento: o objeto que realmente envia um evento. Por exemplo, se o usuário clica em um botão que está dentro de uma entidade gráfica que, por sua vez, está no palco, todos esses objetos enviam eventos, mas o destino do evento é o local onde o evento realmente aconteceu - nesse caso, o botão clicado.
- Ouvinte: um objeto ou uma função que foi registrada com um objeto para indicar que deve ser notificado quando ocorrer um evento específico.

Teste dos exemplos do capítulo

Talvez você queira testar algumas das listagens de código de exemplo por si próprio, durante a leitura deste capítulo. Basicamente, todas as listagens de código deste capítulo incluem uma chamada da função `trace()` para testar os resultados do código. Para testar as listagens de código deste capítulo:

1. Crie um documento vazio usando a ferramenta de autoria do Flash.
2. Selecione um quadro-chave na linha de tempo.
3. Abra o painel Ações e copie a listagem de código no painel Script.
4. Execute o programa usando o comando Controlar > Testar filme.

   Você verá os resultados das funções `trace()` da listagem de código no painel Saída.

Algumas listagens de código são mais complexas e gravadas como uma classe. Para testar esses exemplos:

1. Crie um documento vazio usando a ferramenta de autoria do Flash e salve-o no computador.
2. Crie um novo arquivo ActionScript e salve-o no mesmo diretório em que o documento criado na etapa 1. O nome do arquivo deve corresponder ao nome da classe na listagem de código. Por exemplo, se a listagem de código define uma classe chamada EventTest, use o nome EventTest.as para salvar o arquivo do ActionScript.
3. Copie a listagem de código no arquivo do ActionScript e salve o arquivo.
4. No documento, clique em uma parte branca do Palco ou espaço de trabalho para ativar o Inspetor de propriedades do documento.
5. No Inspetor de propriedades, no campo Classe do documento, digite o nome da classe ActionScript que você copiou do texto.
6. Execute o programa usando o comando Controlar > Testar filme.

   Você verá os resultados do exemplo no painel Saída.

Essas técnicas para testar listagens de código de exemplo são detalhadas em "Teste de listagens de código de exemplo dos capítulos" na página 36.

Como a manipulação de eventos do ActionScript 3.0 é diferente das versões anteriores

A diferença mais notável entre a manipulação de eventos no ActionScript 3.0 e nas versões anteriores do ActionScript é o fato de que, no ActionScript 3.0, existe apenas um sistema para manipulação de eventos, enquanto nas versões anteriores existem diversos sistemas diferentes de manipulação de eventos. Esta seção começa com uma visão geral de como era a manipulação de eventos nas versões anteriores do ActionScript e, em seguida, discute como a manipulação de eventos foi alterada para o ActionScript 3.0.

Manipulação de eventos nas versões anteriores do ActionScript

As versões anteriores ao ActionScript 3.0 forneciam diversas maneiras diferentes para manipular eventos:

- Manipuladores de eventos `on()` que podem ser colocados diretamente nas ocorrências de Button e de MovieClip
- Manipuladores `onClipEvent()` que podem ser colocados diretamente nas ocorrências de MovieClip
- Propriedades de função de retorno de chamada, como `XML.onload` e `Camera.onActivity`
- Ouvintes de evento registrados ao usar o método `addListener()`
- A classe UIEventDispatcher que implementou parcialmente o modelo de evento DOM.

Cada um desses mecanismos tem vantagens e desvantagens. Os manipuladores `on()` e `onClipEvent()` são fáceis de usar, mas dificultam a manutenção posterior dos projetos porque o código colocado diretamente nos botões e clipes de filme pode ser difícil de localizar. As funções de retorno de chamada também são simples de implementar, mas você só pode usar uma função por evento. Os ouvintes de evento são mais difíceis de implementar porque requerem não só a criação de um objeto de ouvinte e de uma função, mas também o registro do ouvinte com o objeto que gera o evento. No entanto, esse aumento da sobrecarga permite criar vários objetos de ouvinte e registrar todos eles para o mesmo evento.

O desenvolvimento de componentes para o ActionScript 2.0 gerou mais um modelo de evento. Esse novo modelo, incorporado na classe UIEventDispatcher, foi baseado em um subconjunto da especificação de eventos DOM. Os desenvolvedores familiarizados com a manipulação de eventos de componente acharão a transição para o modelo de evento do ActionScript 3.0 relativamente fácil.

Infelizmente, a sintaxe usada pelos diversos modelos de evento é diferente em alguns pontos. Por exemplo, no ActionScript 2.0, algumas propriedades, como `TextField.onChanged`, podem ser usadas como uma função de retorno de chamada ou um ouvinte de evento. No entanto, a sintaxe para registrar objetos de ouvinte é diferente, dependendo do uso de uma das seis classes que oferecem suporte aos ouvintes ou da classe UIEventDispatcher. Para as classes Key, Mouse, MovieClipLoader, Selection, Stage e TextField, use o método `addListener()`, mas, para a manipulação de eventos de componentes, use o método chamado `addEventListener()`.

Outra complexidade decorrente dos diferentes modelos de manipulação de eventos é o escopo da função do manipulador de eventos, que variava muito dependendo do mecanismo usado. Em outras palavras, o significado da palavra-chave `this` não era consistente entre os sistemas de manipulação de eventos.

Manipulação de eventos no ActionScript 3.0

O ActionScript 3.0 apresenta um único modelo de manipulação de eventos que substitui os diversos mecanismos diferentes que existiam nas versões anteriores da linguagem. O novo modelo de evento baseia-se na especificação de eventos DOM nível 3. Embora o formato de arquivo SWF não estejam em conformidade especificamente com o padrão DOM, existem similaridades suficientes entre a lista de exibição e a estrutura do DOM que permitem a implementação do modelo de evento DOM. Um objeto na lista de exibição é equivalente a um nó na estrutura hierárquica DOM e os termos *objeto da lista de exibição* e *nó* são usados alternadamente nesta discussão.

A implementação do Flash Player e do AIR do modelo de evento DOM inclui um conceito chamado comportamento padrão. Um *comportamento padrão* é uma ação executada pelo Flash Player ou AIR como conseqüência normal de determinados eventos.

Comportamentos padrão

Os desenvolvedores normalmente são responsáveis por gravar o código que responde aos eventos. No entanto, em alguns casos, como um comportamento normalmente é associado a um evento, o Flash Player ou o AIR o executa automaticamente a não ser que o desenvolvedor adicione algum código para cancelá-lo. Como o Flash Player ou o AIR exibe o comportamento automaticamente, tais comportamentos são chamados de padrão.

Por exemplo, quando um usuário insere um texto em um objeto TextField, a expectativa de que o texto será exibido nesse objeto TextField é tão comum que o comportamento é incorporado no Flash Player e no AIR. Se não desejar que esse comportamento padrão ocorra, cancele-o usando o novo sistema de manipulação de eventos. Quando o usuário insere o texto em um objeto TextField, o Flash Player ou o AIR cria uma ocorrência da classe TextEvent para representar essa entrada do usuário. Para impedir que o Flash Player ou o AIR exiba o texto no objeto TextField, acesse essa ocorrência específica de TextEvent e chame o método `preventDefault()` dessa ocorrência.

Não é possível impedir todos os comportamentos padrão. Por exemplo, o Flash Player e o AIR geram um objeto MouseEvent quando o usuário clica duas vezes em uma palavra em um objeto TextField. O comportamento padrão, que não pode ser evitado, é a palavra realçada quando o cursor passa.

Muitos tipos de objetos de evento não têm comportamentos padrão associados. Por exemplo, o Flash Player envia um objeto de evento connect quando uma conexão de rede é estabelecida, mas não há nenhum comportamento padrão associado a esse objeto. A documentação da API da classe Event e de suas subclasses relaciona cada tipo de evento e descreve os comportamentos padrão associados, além de indicar se esse comportamento pode ser evitado.

É importante mencionar que os comportamentos padrão são associados apenas aos objetos de evento enviados pelo Flash Player ou AIR, e não existem para objetos de evento enviados de modo programático pelo ActionScript. Por exemplo, você pode usar os métodos da classe EventDispatcher para enviar um objeto de evento do tipo `textInput`, mas esse objeto não terá nenhum comportamento padrão associado. Em outras palavras, o Flash Player e o AIR não exibirão um caractere em um objeto TextField em decorrência de um evento `textInput` enviado programaticamente.

Novidades dos ouvintes de evento do ActionScript 3.0

Para desenvolvedores familiarizados com o método `addListener()` do ActionScript 2.0, talvez seja útil descrever as diferenças entre o modelo de ouvinte de evento do ActionScript 2.0 e o modelo de evento do ActionScript 3.0. A lista a seguir descreve as principais diferenças entre os dois modelos de evento:

- Para incluir ouvintes de evento no ActionScript 2.0, você usa `addListener()` em alguns casos e `addEventListener()` em outros, enquanto no ActionScript 3.0, `addEventListener()` é usado em todas as situações.
- Não existe nenhum fluxo de evento no ActionScript 2.0, ou seja, o método `addListener()` pode ser chamado somente no objeto que transmite o evento. Já no ActionScript 3.0, o método `addEventListener()` pode ser chamado em qualquer objeto que faça parte do fluxo de evento.

- No ActionScript 2.0, os ouvintes de evento podem ser funções, métodos ou objetos e, no ActionScript 3.0, apenas funções ou métodos podem ser ouvintes de evento.

O fluxo de evento

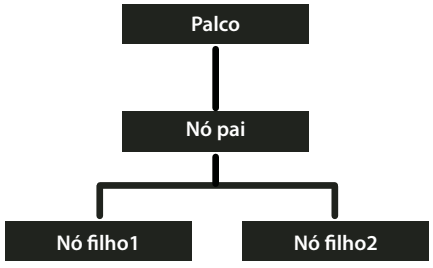
O Flash Player ou o AIR envia objetos de evento sempre que ocorre um evento. Se o destino do evento não estiver na lista de exibição, o Flash Player ou o AIR enviará o objeto diretamente para o destino. Por exemplo, o Flash Player envia o objeto de evento progress diretamente para um objeto URLStream. No entanto, se o destino do evento estiver na lista de exibição, o Flash Player enviará o objeto para a lista de exibição e esse objeto irá percorrer a lista até chegar ao destino.

O *fluxo de evento* descreve como um objeto de evento se move pela lista de exibição. A lista de exibição está organizada em uma hierarquia que pode ser descrita como uma árvore. No topo da hierarquia da lista de exibição está o palco, que é um contêiner especial de objeto de exibição que serve como raiz da lista de exibição. O palco é representado pela classe flash.display.Stage e só pode ser acessado por meio de um objeto de exibição. Cada objeto de exibição tem uma propriedade chamada `stage` que faz referência ao palco desse aplicativo.

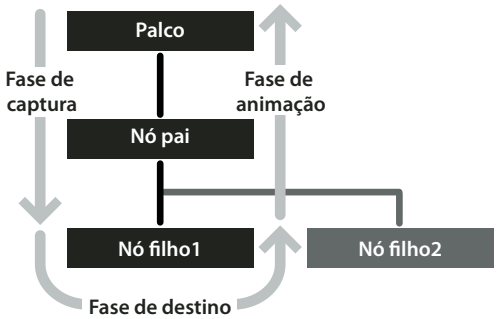
Quando o Flash Player ou o AIR envia um objeto para um evento relacionado à lista de exibição, esse objeto de evento faz uma viagem de ida e volta do palco ao *nó de destino*. A especificação de eventos DOM define o nó de destinocomo o nó que representa o destino do evento. Em outras palavras, o nó de destino é o objeto da lista de exibição onde ocorreu o evento. Por exemplo, se o usuário clicar em um objeto da lista de exibição chamado `child1`, o Flash Player ou o AIR enviará um objeto de evento usando `child1` como nó de destino.

O fluxo de evento é dividido conceitualmente em três partes. A primeira parte é chamada de fase de captura; essa fase compreende todos os nós do palco ao pai do nó de destino. A segunda parte é chamada de fase de destino e consiste apenas no nó de destino. A terceira parte é chamada de fase de bubbling. A fase de bubbling é composta pelos nós encontrados na viagem de retorno do pai do nó de destino ao palco.

Os nomes das fases farão mais sentido se você pensar na lista de exibição como uma hierarquia vertical com o palco no topo, como mostra o diagrama a seguir:



Se o usuário clicar no `nó Child1`, o Flash Player ou o AIR enviará um objeto ao fluxo de evento. Como mostra a imagem a seguir, a jornada do objeto começa no `palco`, vai até o `nó pai`, segue para o `nó Child1` e volta ao `palco`, movendo-se pelo `nó pai` novamente até voltar ao `palco`.



Neste exemplo, a fase de captura inclui o `palco` e o `nó pai` durante a viagem de ida inicial. A fase de destino consiste no tempo gasto no `nó Child1`. A fase de bubbling inclui o `nó pai` e o `palco`, pois ambos estão presentes na viagem de volta ao nó raiz.

O fluxo de evento contribui com um sistema de manipulação de eventos mais avançado do que o anteriormente disponível para os programadores no ActionScript. Nas versões anteriores do ActionScript, não existe o fluxo de evento, ou seja, os ouvintes de evento podem ser adicionados somente ao objeto que gera o evento. No ActionScript 3.0, é possível adicionar ouvintes de evento não só a um nó de destino, mas também a qualquer nó ao longo do fluxo de evento.

A possibilidade de adicionar ouvintes ao longo do fluxo de evento é útil quando um componente da interface do usuário tem mais de um objeto. Por exemplo, um objeto de botão normalmente contém um objeto de texto que serve como rótulo do botão. Sem poder adicionar um ouvinte ao fluxo de evento, seria necessário adicionar um ouvinte ao objeto de botão e ao objeto de texto para garantir o recebimento de notificações sobre eventos de clique que ocorrem em qualquer lugar no botão. No entanto, a existência do fluxo de evento permite colocar um único ouvinte no objeto de botão que manipula eventos de clique ocorridos no objeto de texto ou nas áreas do objeto de botão que não são obscurecidas pelo objeto de texto.

Porém, nem todos os objetos de evento participam das três fases do fluxo de evento. Alguns tipos de eventos, como `enterFrame` e `init`, são enviados diretamente para o nó de destino e não participam da fase de captura, nem da fase de bubbling. Outros eventos podem ser direcionados para objetos que não estão na lista de exibição, como eventos enviados para uma ocorrência da classe Socket. Esses objetos de evento também vão diretamente para o objeto de destino, sem participar das fases de captura e de bubbling.

Para descobrir como se comporta um tipo de evento específico, consulte a documentação da API ou examine as propriedades do objeto de evento. O exame das propriedades do objeto de evento está descrito na próxima seção.

Objetos de evento

Os objetos de evento têm duas finalidades principais no novo sistema de manipulação de eventos. Primeiro, esses objetos representam eventos reais, armazenando informações sobre eventos específicos em um conjunto de propriedades. Segundo, os objetos de evento contêm um conjunto de métodos que permitem manipular objetos de evento e afetam o comportamento do sistema de manipulação de eventos.

Para facilitar o acesso a esses métodos e propriedades, a API do Flash Player define uma classe Event que serve como base para todos os objetos de evento. A classe Event define um conjunto básico de métodos e propriedades comuns a todos os objetos de evento.

Esta seção começa com uma discussão sobre as propriedades da classe Event, continua com uma descrição dos métodos da classe Event e termina explicando por que existem subclasses de Event.

Compreensão das propriedades da classe Event

A classe Event define diversas propriedades e constantes somente leitura que fornecem informações importantes sobre um objeto de evento. As seguintes propriedades são especialmente importantes:

- Os tipos de objeto de evento são representados por constantes e armazenados na propriedade `Event.type`.
- Se for necessário impedir o comportamento padrão de um evento, isso será representado por um valor booleano e armazenado na propriedade `Event.cancelable`.
- As informações de fluxo de evento são contidas nas propriedades restantes.

Tipos de objeto de evento

Cada objeto de evento tem um tipo associado. Os tipos de evento são armazenados na propriedade `Event.type` como valores de string. É útil saber o tipo de um objeto de evento para que o código possa diferenciar os objetos de tipos diferentes. Por exemplo, o código a seguir especifica que a função do ouvinte `clickHandler()` deve responder a qualquer objeto de evento de clique de mouse transmitido para `myDisplayObject`:

```
myDisplayObject.addEventListener(MouseEvent.CLICK, clickHandler);
```

Vinte e quatro tipos de evento são associados à classe Event propriamente dita e representados por constantes da classe Event; alguns deles são mostrados no trecho a seguir da definição da classe Event:

```
package flash.events
{
    public class Event
    {
        // class constants
        public static const ACTIVATE:String = "activate";
        public static const ADDED:String= "added";
        // remaining constants omitted for brevity
    }
}
```

Essas constantes facilitam a referência a tipos de evento específicos. Use essas constantes em vez das strings que representam. Se você digitar o nome de uma constante incorretamente no código, o compilador detectará o erro, mas, se as strings forem utilizadas, um erro tipográfico talvez não se manifeste no tempo de compilação e gere um comportamento inesperado difícil de depurar. Por exemplo, ao adicionar um ouvinte de evento, use o seguinte código:

```
myDisplayObject.addEventListener(MouseEvent.CLICK, clickHandler);
```

em vez de:

```
myDisplayObject.addEventListener("click", clickHandler);
```

Informações sobre o comportamento padrão

Seu código pode verificar se o comportamento padrão de um determinado objeto de evento pode ser impedido acessando a propriedade `cancelable`. A propriedade `cancelable` tem um valor booleano que indica se um comportamento padrão pode ou não ser impedido. Você pode impedir ou cancelar o comportamento padrão associado a um pequeno número de eventos usando o método `preventDefault()`. Para obter mais informações, consulte Cancelamento do comportamento padrão do evento em "Compreensão dos métodos da classe Event" na página 260.

Informações sobre o fluxo de evento

As demais propriedades da classe Event contêm informações importantes sobre um objeto de evento e sua relação com o fluxo de evento, conforme descrito na lista a seguir:

- A propriedade `bubbles` contém informações sobre as partes do fluxo nas quais o objeto de evento participa.
- A propriedade `eventPhase` indica a fase atual no fluxo de evento.
- A propriedade `target` armazena uma referência ao destino do evento.
- A propriedade `currentTarget` armazena uma referência ao objeto da lista de exibição que está processando o objeto de evento no momento.

A propriedade bubbles

Um evento é animado se seu objeto participa da fase de bubbling do fluxo de evento, o que significa que o objeto de evento volta do nó de destino para seu ancestral até chegar ao palco. A propriedade `Event.bubbles` armazena um valor booleano que indica se o objeto de evento participa na fase de bubbling. Como todos os eventos que são animados também participam nas fases de captura e de destino, qualquer evento animado participa das três fases do fluxo de evento. Se o valor for `true`, o objeto de evento participará das três fases. Se o valor for `false`, o objeto de evento não participará na fase de bubbling.

A propriedade eventPhase

Você pode determinar a fase de qualquer objeto de evento investigando a propriedade `eventPhase`. A propriedade `eventPhase` contém um valor inteiro sem sinal que representa uma das três fases do fluxo de evento. A API do Flash Player define uma classe EventPhase separada que contém três constantes que correspondem a três valores inteiros sem sinal, como mostra o seguinte trecho de código:

```
package flash.events
{
    public final class EventPhase
    {
        public static const CAPTURING_PHASE:uint = 1;
        public static const AT_TARGET:uint = 2;
        public static const BUBBLING_PHASE:uint= 3;
    }
}
```

Essa constantes correspondem a três valores válidos da propriedade `eventPhase`. Você pode usar essas constantes para deixar seu código mais legível. Por exemplo, se desejar assegurar que uma função `myFunc()` seja chamada somente se o destino do evento estiver no palco de destino, use o código a seguir para testar essa condição:

```
if (event.eventPhase == EventPhase.AT_TARGET)
{
    myFunc();
}
```

A propriedade target

A propriedade `target` armazena uma referência ao objeto que é o destino do evento. Em alguns casos, isso é simples, como quando um microfone é ativado; o destino do objeto de evento é o objeto Microphone. No entanto, se o destino estiver na lista de exibição, a hierarquia da lista deve ser levada em consideração. Por exemplo, se o usuário inserir um clique de mouse em um ponto que inclui objetos sobrepostos da lista de exibição, o Flash Player e o AIR sempre escolherão o objeto mais distante do palco como destino do evento.

Para arquivos SWF complexos, especialmente aqueles nos quais os botões são decorados regularmente com objetos filho menores, a propriedade `target` talvez não seja usada com freqüência porque, em geral, apontará para o objeto filho de um botão, não para o botão. Nesses casos, a prática comum é adicionar ouvintes de evento ao botão e usar a propriedade `currentTarget` que aponta para o botão, já que a propriedade `target` pode apontar para um filho do botão.

A propriedade currentTarget

A propriedade `currentTarget` contém uma referência ao objeto que está processando o objeto de evento no momento. Embora possa parecer estranho não saber qual nó está processando o objeto de evento que você está examinando no momento, é possível adicionar uma função de ouvinte a qualquer objeto de exibição do fluxo desse objeto de evento; a função de ouvinte pode ser colocada em qualquer lugar. Além disso, a mesma função de ouvinte pode ser adicionada a diferentes objetos de exibição. À medida que aumenta o tamanho e a complexidade de um projeto, a propriedade `currentTarget` fica cada vez mais útil.

Compreensão dos métodos da classe Event

Existem três categorias de métodos da classe Event:

- Métodos de utilitário, que podem criar cópias de um objeto de evento ou convertê-lo em uma string
- Métodos de fluxo de evento, que removem objetos do fluxo de evento
- Métodos de comportamento padrão, que impedem o comportamento padrão ou verificam se ele foi impedido

Métodos de utilitário da classe Event

Existem dois métodos de utilitário na classe Event. O método `clone()` permite criar cópias de um objeto de evento. O método `toString()` permite gerar uma representação de string das propriedades de um objeto de evento, junto com seus valores. Esses dois métodos são usados internamente pelo sistema de modelo de evento, mais são expostos aos desenvolvedores para uso geral.

Para desenvolvedores avançados que criam subclasses de Event, é necessário substituir e implementar versões dos dois métodos de utilitário para garantir que a subclasse funcione corretamente.

Interrupção do fluxo de evento

Você pode chamar o método `Event.stopPropagation()` ou o método `Event.stopImmediatePropagation()` para impedir que um objeto de evento continue seu percurso no fluxo. Os dois métodos são praticamente idênticos e diferem apenas quanto aos ouvintes de evento do nó atual que podem ser executados:

- O método `Event.stopPropagation()` impede que o objeto de evento se mova para o próximo nó, mas só depois que algum outro ouvinte de evento do nó atual tenha permissão para ser executado.
- O método `Event.stopImmediatePropagation()` também impede que o objeto de evento se mova para o próximo nó, mas não permite que outros ouvintes de evento do nó atual sejam executados.

Chamar um desses métodos não afeta a ocorrência do comportamento padrão associado a um evento. Use os métodos do comportamento padrão da classe Event para impedir tal comportamento.

Cancelamento do comportamento padrão do evento

Os dois métodos envolvidos no cancelamento do comportamento padrão são `preventDefault()` e `isDefaultPrevented()`. Chame o método `preventDefault()` para cancelar o comportamento padrão associado a um evento. Para verificar se `preventDefault()` já foi chamado em um objeto de evento, chame o método `isDefaultPrevented()`, que retorna o valor `true` se o método já tiver sido chamado; caso contrário, retornará `false`.

O método `preventDefault()` só funcionará se o comportamento padrão do evento puder ser cancelado. Para verificar isso, consulte a documentação da API para esse tipo de evento ou use o ActionScript para examinar a propriedade `cancelable` do objeto de evento.

O cancelamento do comportamento padrão não afeta o progresso de um objeto no fluxo de evento. Use os métodos do fluxo de evento da classe Event para remover um objeto do fluxo.

Subclasses de Event

Para muitos eventos, o conjunto comum de propriedades definido na classe Event é suficiente. No entanto, outros eventos têm características exclusivas que não podem ser capturadas pelas propriedades disponíveis na classe Event. Para esses eventos, o ActionScript 3.0 define várias subclasses da classe Event.

Cada subclasse fornece propriedades e tipos de evento adicionais que são exclusivos dessa categoria de eventos. Por exemplo, os eventos relacionados à entrada do mouse têm várias características exclusivas que não podem ser capturadas pelas propriedades definidas na classe Event. A classe MouseEvent estende a classe Event adicionando dez propriedades que contêm informações como o local do evento de mouse e que indicam se teclas específicas foram pressionadas durante o evento de mouse.

Uma subclasse de Event também contém constantes que representam os tipos de evento associados à subclasse. Por exemplo, a classe MouseEvent define constantes para vários tipos de evento de mouse, incluindo `click`, `doubleClick`, `mouseDown` e `mouseUp`.

Conforme descrito na seção de métodos de utilitário da classe Event em"Objetos de evento" na página 257, é preciso substituir os métodos `clone()` e `toString()` para oferecer funcionalidade específica para a subclasse.

Ouvintes de evento

Os ouvintes de evento, também chamados de manipuladores de evento, são funções executadas pelo Flash Player e pelo AIR em resposta a eventos específicos. A adição de um ouvinte de evento é um processo de duas etapas. Primeiro, crie um método de função ou classe a ser executado pelo Flash Player ou AIR em resposta ao evento. Às vezes, isso é chamado de função de ouvinte ou de manipulador de evento. Segundo, use o método `addEventListener()` para registrar a função de ouvinte no destino do evento ou em qualquer objeto da lista de exibição ao longo do fluxo de evento adequado.

Criação de uma função de ouvinte

A criação de funções de ouvinte é uma das diferenças entre o modelo de evento do ActionScript 3.0 e o modelo de evento DOM. No modelo de evento DOM, existe uma distinção clara entre um ouvinte de evento e uma função de ouvinte: um ouvinte de evento é uma ocorrência de uma classe que implementa a interface EventListener, enquanto a função de ouvinte é um método dessa classe chamado `handleEvent()`. No modelo de evento DOM, a ocorrência da classe que contém a função de ouvinte é registrada, não a função de ouvinte propriamente dita.

No modelo de evento do ActionScript 3.0, não existe diferença entre um ouvinte de evento e uma função de ouvinte. O ActionScript 3.0 não tem uma interface EventListener e as funções de ouvinte podem ser definidas fora de uma classe ou como parte dela. Além disso, as funções de ouvinte não precisam ser chamadas de `handleEvent()` - elas podem ser chamadas com qualquer identificador válido. No ActionScript 3.0, o nome da função de ouvinte real é registrado.

Função de ouvinte definida fora de uma classe

O código a seguir cria um arquivo SWF simples que exibe um quadrado vermelho. Uma função de ouvinte chamada `clickHandler()`, que não faz parte de uma classe, ouve os eventos de clique de mouse no quadrado vermelho.

```
package
{
    import flash.display.Sprite;

    public class ClickExample extends Sprite
    {
        public function ClickExample()
        {
            var child:ChildSprite = new ChildSprite();
            addChild(child);
        }
    }
}

import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
    public function ChildSprite()
    {
        graphics.beginFill(0xFF0000);
        graphics.drawRect(0,0,100,100);
        graphics.endFill();
        addEventListener(MouseEvent.CLICK, clickHandler);
    }
}

function clickHandler(event:MouseEvent):void
{
    trace("clickHandler detected an event of type: " + event.type);
    trace("the this keyword refers to: " + this);
}
```

Quando um usuário interage com o arquivo SWF resultante clicando no quadrado, o Flash Player ou AIR gera a seguinte saída de traço:

```
clickHandler detected an event of type: click
the this keyword refers to: [object global]
```

Observe que o objeto de evento é transmitido como um argumento para `clickHandler()`. Isso permite que a função de ouvinte examine o objeto de evento. Neste exemplo, use a propriedade `type` do objeto de evento para certificar-se de que o evento é um evento de clique.

O exemplo também verifica o valor da palavra-chave `this`. Nesse caso, `this` representa o objeto global, o que faz sentido porque a função é definida fora de um objeto ou classe personalizado.

Função de ouvinte definida como um método de classe

O exemplo a seguir é idêntico ao anterior que define a classe ClickExample, mas a função `clickHandler()` é definida como um método da classe ChildSprite:

```
package
{
    import flash.display.Sprite;

    public class ClickExample extends Sprite
    {
        public function ClickExample()
        {
            var child:ChildSprite = new ChildSprite();
            addChild(child);
        }
    }
}

import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
    public function ChildSprite()
    {
        graphics.beginFill(0xFF0000);
        graphics.drawRect(0,0,100,100);
        graphics.endFill();
        addEventListener(MouseEvent.CLICK, clickHandler);
    }
    private function clickHandler(event:MouseEvent):void
    {
        trace("clickHandler detected an event of type: " + event.type);
        trace("the this keyword refers to: " + this);
    }
}
```

Quando um usuário interage com o arquivo SWF resultante clicando no quadrado vermelho, o Flash Player ou AIR gera a seguinte saída de traço:

```
clickHandler detected an event of type: click
the this keyword refers to: [object ChildSprite]
```

Observe que a palavra-chave `this` faz referência à ocorrência de ChildSprite chamada `child`. Este comportamento é diferente do ActionScript 2.0. Caso já tenha usado componentes no ActionScript 2.0, talvez se lembre que, quando um método de classe era transmitido para `UIEventDispatcher.addEventListener()`, o escopo do método ia até o componente que transmite o evento em vez da classe na qual o método de ouvinte era definido. Em outras palavras, se você utilizou essa técnica no ActionScript 2.0, a palavra-chave `this` fazia referência ao componente que transmite o evento, não à ocorrência de ChildSprite.

Isso foi um problema significativo para alguns programadores, pois não permitia o acesso a outros métodos e propriedades da classe que contém o método de ouvinte. Como solução temporária, os programadores do ActionScript 2.0 podiam usar a classe `mx.util.Delegate` para alterar o escopo do método de ouvinte. No entanto, isso não é mais necessário porque o ActionScript 3.0 cria um método vinculado quando `addEventListener()` é chamado. Em resultado disso, a palavra-chave `this` faz referência à ocorrência de ChildSprite chamada `child`, e o programador tem acesso a outros métodos e propriedades da classe ChildSprite.

Ouvinte de evento que não deve ser usado

Existe uma terceira técnica que permite criar um objeto genérico com uma propriedade que aponta para uma função de ouvinte atribuída dinamicamente, mas essa técnica não é recomendada. Ela será discutida aqui porque foi muito usada no ActionScript 2.0, mas não deve ser usada no ActionScript 3.0. Essa técnica não é recomendada porque a palavra-chave `this` fará referência ao objeto global em vez do objeto de ouvinte.

O exemplo a seguir é idêntico ao exemplo anterior da classe ClickExample, mas a função de ouvinte é definida como parte de um objeto genérico chamado `myListenerObj`:

```
package
{
    import flash.display.Sprite;

    public class ClickExample extends Sprite
    {
        public function ClickExample()
        {
            var child:ChildSprite = new ChildSprite();
            addChild(child);
        }
    }
}

import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
    public function ChildSprite()
    {
        graphics.beginFill(0xFF0000);
        graphics.drawRect(0,0,100,100);
        graphics.endFill();
        addEventListener(MouseEvent.CLICK, myListenerObj.clickHandler);
    }
}

var myListenerObj:Object = new Object();
myListenerObj.clickHandler = function (event:MouseEvent):void
{
        trace("clickHandler detected an event of type: " + event.type);
        trace("the this keyword refers to: " + this);
}
```

Os resultados do traço serão similares a:

```
clickHandler detected an event of type: click
the this keyword refers to: [object global]
```

Você talvez pensasse que `this` faria referência a `myListenerObj` e que a saída de traço seria `[object Object]`, mas a referência foi feita ao objeto global. Ao transmitir o nome de uma propriedade dinâmica como um argumento para `addEventListener()`, o Flash Player ou AIR não consegue criar um método vinculado. Isso ocorre porque, quando está transmitindo o parâmetro `listener`, você não está transmitindo nada mais do que o endereço de memória da função de ouvinte, e o Flash Player e AIR não conseguem vincular esse endereço de memória com a ocorrência de `myListenerObj`.

Gerenciamento de ouvintes de evento

Você pode gerenciar suas funções de ouvinte usando os métodos da interface IEventDispatcher. A interface IEventDispatcher é a versão ActionScript 3.0 da interface EventTarget do modelo de evento DOM. Embora o nome IEventDispatcher possa transmitir a idéia de que o objetivo principal é enviar (ou despachar) objetos de evento, os métodos dessa classe são usados com mais freqüência para registrar, verificar e remover ouvintes de evento. A interface IEventDispatcher define cinco métodos, como mostra o código a seguir:

```
package flash.events
{
    public interface IEventDispatcher
    {
        function addEventListener(eventName:String,
                        listener:Object,
                        useCapture:Boolean=false,
                        priority:Integer=0,
                        useWeakReference:Boolean=false):Boolean;

        function removeEventListener(eventName:String,
                    listener:Object,
                    useCapture:Boolean=false):Boolean;

        function dispatchEvent(eventObject:Event):Boolean;

        function hasEventListener(eventName:String):Boolean;
        function willTrigger(eventName:String):Boolean;
    }
}
```

A API do Flash Player implementa a interface IEventDispatcher com a classe EventDispatcher, que serve como base para todas as classes que podem ser destinos de evento ou fazer parte de um fluxo de evento. Por exemplo, a classe DisplayObject é herdada da classe EventDispatcher. Isso significa que todos os objetos da lista de exibição têm acesso aos métodos da interface IEventDispatcher.

Adição de ouvintes de evento

O método `addEventListener()` é a força motriz da interface IEventDispatcher. Você pode usá-lo para registrar suas funções de ouvinte. Os dois parâmetros necessários são `type` e `listener`. Use o parâmetro `type` para especificar o tipo de evento. Use o parâmetro `listener` para especificar a função de ouvinte que será executada quando o evento ocorrer. O parâmetro `listener` pode ser uma referência a um método de classe ou função.

Nota: *Não use parênteses ao especificar o parâmetro `listener`. Por exemplo, a função `clickHandler()` é especificada sem parênteses na seguinte chamada do método `addEventListener()`:*

Nota: *`addEventListener(MouseEvent.CLICK, clickHandler)`.*

O parâmetro `useCapture` do método `addEventListener()` permite controlar a fase do fluxo de evento na qual o ouvinte estará ativo. Se `useCapture` for definido como `true`, o ouvinte estará ativo durante a fase de captura do fluxo de evento. Se `useCapture` for definido como `false`, o ouvinte estará ativo durante as fases de destinos e de bubbling do fluxo de evento. Para ouvir um evento durante todas as fases do fluxo de evento, chame `addEventListener()` duas vezes, uma com `useCapture` definido como `true` e outra com `useCapture` definido como `false`.

O parâmetro `priority` do método `addEventListener()` não é uma parte oficial do modelo de evento DOM nível 3. Ele está incluído no ActionScript 3.0 para fornecer mais flexibilidade para organizar os ouvintes de evento. Ao chamar `addEventListener()`, você pode definir a prioridade desse ouvinte de evento transmitindo um valor inteiro como o parâmetro `priority`. O valor padrão é 0, mas é possível definir valores inteiros negativos ou positivos. Quanto maior o número, mais rapidamente o ouvinte de evento será executado. Os ouvintes de evento com a mesma prioridade são executados na ordem em que foram adicionados, de modo que o ouvinte adicionado primeiro será executado antes.

O parâmetro `useWeakReference` permite especificar se a referência à função de ouvinte será fraca ou normal. Se esse parâmetro for definido como `true`, você poderá evitar situações nas quais as funções de ouvinte persistem na memória mesmo quando não são mais necessárias. O Flash Player e o AIR usam uma técnica chamada *coleta de lixo* para apagar da memória os objetos que não são mais usados. Um objeto não é mais necessário quando não existe nenhuma referência a ele. O coletor de lixo ignora as referências fracas e, desse modo, uma função de ouvinte que tem apenas uma referência fraca se qualifica para a coleta de lixo.

Remoção de ouvintes de evento

Você pode usar o método `removeEventListener()` para remover um ouvinte de evento que não é mais necessário. É recomendado remover todos os ouvintes que não serão mais usados. Os parâmetros necessários incluem `eventName` e `listener`, que são os mesmos parâmetros obrigatórios para o método `addEventListener()`. Não se esqueça que é possível ouvir eventos durante todas as fases chamando `addEventListener()` duas vezes, uma com `useCapture` definido como `true` e outra definido como `false`. Para remover os dois ouvintes de evento, seria necessário chamar `removeEventListener()` duas vezes, uma com `useCapture` definido como `true` e outra definido como `false`.

Envio de eventos

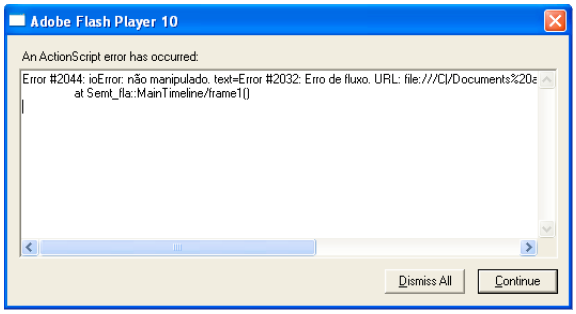
O método `dispatchEvent()` pode ser usado por programadores avançados para enviar um objeto de evento personalizado para o fluxo de evento. O único parâmetro aceito por esse método é uma referência a um objeto de evento, que deve ser uma ocorrência da classe Event ou uma subclasse de Event. Assim que é enviada, a propriedade `target` do objeto de evento é definida como o objeto no qual `dispatchEvent()` foi chamado.

Verificação de ouvintes de evento existentes

Os dois métodos finais da interface IEventDispatcher fornecem informações úteis sobre a existência de ouvintes de evento. O método `hasEventListener()` retornará `true` se um ouvinte de evento for encontrado para um tipo de evento específico em um determinado objeto da lista de exibição. O método `willTrigger()` também retornará `true` se um ouvinte for encontrado para um objeto específico da lista de exibição, mas `willTrigger()` procurará ouvintes nesse objeto de exibição e em todos os ancestrais desse objeto em todas as fases do fluxo de evento.

Eventos de erro sem ouvintes

As exceções, em vez dos eventos, são o principal mecanismo de manipulação de erros no ActionScript 3.0, mas a manipulação de exceções não funciona para operações assíncronas, como o carregamento de arquivos. Se ocorrer um erro durante uma operação assíncrona, o Flash Player e o AIR enviarão um objeto de evento de erro. Se nenhum ouvinte for criado para o evento de erro, as versões do depurador do Flash Player e do AIR exibirão uma caixa de diálogo com informações sobre o erro. Por exemplo, a versão do depurador do Flash Player exibe a caixa de diálogo a seguir, descrevendo o erro quando o aplicativo tentar carregar arquivos a partir de URLs inválidas:



A maioria dos eventos de erro baseia-se na classe ErrorEvent e, desse modo, tem uma propriedade chamada `text` que é usada para armazenar a mensagem de erro exibida pelo Flash Player ou AIR. As duas exceções são as classes StatusEvent e NetStatusEvent. Essas duas classes têm uma propriedade `level` (`StatusEvent.level` e `NetStatusEvent.info.level`). Quando o valor da propriedade `level` é "`error`", esses tipos de evento são considerados eventos de erro.

Um evento de erro não interrompe a execução do arquivo SWF. Ele será manifestado somente como uma caixa de diálogo nas versões do depurador dos plug-ins do navegador e de players dedicados, como uma mensagem no painel de saída do player de criação e como uma entrada no arquivo de log do Adobe Flex Builder 3. Ele não será manifesto de modo algum nas versões do Flash Player ou AIR.

Exemplo: Alarm Clock

O exemplo do Alarm Clock consiste em um relógio que permite que o usuário especifique um horário no qual o alarme deve ser desativado, bem como uma mensagem a ser exibida nesse horário. O exemplo do Alarm Clock baseia-se no aplicativo SimpleClock da seção "Trabalho com datas e horas" na página 134 e ilustra vários aspectos do trabalho com eventos ActionScript 3.0, incluindo:

- Como ouvir e responder a um evento
- Notificação de um evento aos ouvintes
- Criação de um tipo de evento personalizado

Para obter os arquivos de aplicativo desse exemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_br. Os arquivos do aplicativo Alarm Clock estão localizados na pasta Amostras/AlarmClock. O aplicativo inclui estes arquivos:

Arquivo	Descrição
AlarmClockApp.mxml ou AlarmClockApp.fla	O arquivo principal do aplicativo no Flash (FLA) ou no Flex (MXML).
com/example/programmingas3/clock/AlarmClock.as	Uma classe que estende a classe SimpleClock, adicionando a funcionalidade do despertador.
com/example/programmingas3/clock/AlarmEvent.as	Uma classe de evento personalizada (uma subclasse de flash.events.Event) que serve como objeto do evento `alarm` da classe AlarmClock.
com/example/programmingas3/clock/AnalogClockFace.as	Desenha a superfície de um relógio redondo e os ponteiros de horas, minutos e segundos com base na hora (descrito no exemplo de SimpleClock).
com/example/programmingas3/clock/SimpleClock.as	Um componente da interface do relógio com um recurso simples de marcação da hora (descrito no exemplo de SimpleClock).

Visão geral do Alarm Clock

O principal recurso do relógio neste exemplo, incluindo o controle da hora e a exibição da superfície do relógio, reutiliza o código do aplicativo SimpleClock, descrito em "Exemplo: relógio analógico simples" na página 139. A classe AlarmClock estende a classe SimpleClock desse exemplo, adicionando a funcionalidade necessária para um despertador, incluindo a definição da hora do alarme e do envio da notificação quando o alarme for "desativado".

Os eventos são gerados para enviar a notificação sobre algo ocorrido. A classe AlarmClock expõe o evento Alarm, que outros objetos podem ouvir para executar as ações desejadas. Além disso, a classe AlarmClock usa uma ocorrência da classe Timer para determinar quando o alarme deve ser acionado. Assim como AlarmClock, a classe Timer gera um evento para notificar outros objetos (neste caso, uma ocorrência de AlarmClock) após um determinado período. Assim como a maioria dos aplicativos do ActionScript, os eventos formam uma parte importante da funcionalidade do aplicativo Alarm Clock de exemplo.

Acionamento do alarme

Conforme mencionado anteriormente, a única funcionalidade da classe AlarmClock está relacionada com a definição e o acionamento do alarme. A classe incorporada Timer (flash.utils.Timer) permite que o desenvolvedor defina o código que será executado após o período especificado. A classe AlarmClock usa uma ocorrência de Timer para determinar quando o alarme deve ser desativado.

```
import flash.events.TimerEvent;
import flash.utils.Timer;

/**
 * The Timer that will be used for the alarm.
 */
public var alarmTimer:Timer;
...
/**
 * Instantiates a new AlarmClock of a given size.
 */
public override function initClock(faceSize:Number = 200):void
{
    super.initClock(faceSize);
    alarmTimer = new Timer(0, 1);
    alarmTimer.addEventListener(TimerEvent.TIMER, onAlarm);
}
```

A ocorrência de Timer definida na classe AlarmClock chama-se `alarmTimer`. O método `initClock()`, que executa as operações de configuração necessárias para a ocorrência de AlarmClock, faz duas coisas com a variável `alarmTimer`. Primeiro, a variável é percorrida com os parâmetros que instruem a ocorrência de Timer a aguardar 0 milissegundos e acionar o evento timer apenas uma vez. Depois de percorrer `alarmTimer`, o código chama o método `addEventListener()` da variável para indicar se deseja ouvir o evento `timer` dessa variável. Uma ocorrência de Timer funciona enviando o evento `timer` após um período especificado. A classe AlarmClock não precisa saber quando o evento `timer` é enviado para desativar seu próprio alarme. Chamando `addEventListener()`, o código de AlarmClock se auto-registra como ouvinte em `alarmTimer`. Os dois parâmetros indicam que a classe AlarmClock deseja ouvir o evento `timer` (indicado pela constante `TimerEvent.TIMER`) e que, quando o evento ocorrer, o método `onAlarm()` da classe AlarmClock deve ser chamado em resposta a esse evento.

Para definir o alarme realmente, o método `setAlarm()` da classe AlarmClock é chamado do seguinte modo:

```
/**
 * Sets the time at which the alarm should go off.
 * @param hour The hour portion of the alarm time.
 * @param minutes The minutes portion of the alarm time.
 * @param message The message to display when the alarm goes off.
 * @return The time at which the alarm will go off.
 */
public function setAlarm(hour:Number = 0, minutes:Number = 0, message:String = "Alarm!"):Date
{
    this.alarmMessage = message;
    var now:Date = new Date();
    // Create this time on today's date.
    alarmTime = new Date(now.fullYear, now.month, now.date, hour, minutes);

    // Determine if the specified time has already passed today.
    if (alarmTime <= now)
    {
        alarmTime.setTime(alarmTime.time + MILLISECONDS_PER_DAY);
    }

    // Stop the alarm timer if it's currently set.
    alarmTimer.reset();
    // Calculate how many milliseconds should pass before the alarm should
    // go off (the difference between the alarm time and now) and set that
    // value as the delay for the alarm timer.
    alarmTimer.delay = Math.max(1000, alarmTime.time - now.time);
    alarmTimer.start();

    return alarmTime;
}
```

Esse método faz várias coisas, como armazenar a mensagem do alarme e criar um objeto Date (`alarmTime`) que representa o momento real em que o alarme é desativado. O mais importante nisto tudo, nas várias linhas finais do método, é o fato de o objeto timer da variável `alarmTimer` ser definido e ativado. Primeiro, o método `reset()` é chamado, interrompendo o cronômetro e redefinindo-o caso já esteja em execução. Em seguida, o horário atual (representado pela variável `now`) é subtraído do valor da variável `alarmTime` para determinar quantos milissegundos devem se passar até o alarme ser desativado. A classe Timer não aciona o evento `timer` em um tempo absoluto, de modo que essa diferença relativa de tempo é atribuída à propriedade `delay` de `alarmTimer`. Finalmente, o método `start()` é chamado para iniciar o cronômetro realmente.

Assim que o período especificado termina, `alarmTimer` envia o evento `timer`. Com a classe AlarmClock registrou o método `onAlarm()` como um ouvinte desse evento, quando o evento `timer` acontece, `onAlarm()` é chamado.

```
/**
 * Called when the timer event is dispatched.
 */
public function onAlarm(event:TimerEvent):void
{
    trace("Alarm!");
    var alarm:AlarmEvent = new AlarmEvent(this.alarmMessage);
    this.dispatchEvent(alarm);
}
```

Um método que é registrado como ouvinte de evento deve ser definido com a assinatura apropriada (isto é, o conjunto de parâmetros e o tipo de retorno do método). Para ser ouvinte do evento `timer` da classe Timer, um método deve definir um parâmetro cujo tipo de dados seja TimerEvent (flash.events.TimerEvent), uma subclasse de Event. Quando chama seus ouvintes de evento, a ocorrência de Timer transmite uma ocorrência de TimerEvent como objeto de evento.

Notificação do alarme a outros

Assim como a classe Timer, a classe AlarmClock fornece um evento que permite que outros códigos recebam notificações quando o alarme é desativado. Para que uma classe use a estrutura de manipulação de eventos incorporada no ActionScript, essa classe deve implementar a interface flash.events.IEventDispatcher. Normalmente, isso é feito por meio da extensão da classe flash.events.EventDispatcher, que fornece uma implementação padrão de IEventDispatcher (ou por meio da extensão de uma das subclasses de EventDispatcher). Conforme descrito anteriormente, a classe AlarmClock estende a classe SimpleClock que, por sua vez, estende a classe Sprite, que (por meio de uma cadeia de herança) estende a classe EventDispatcher. Tudo isso significa que a classe AlarmClock já tem uma funcionalidade interna para fornecer seus próprios eventos.

Outro código pode ser registrado para ser notificado sobre o evento `alarm` da classe AlarmClock chamando o método `addEventListener()` que AlarmClock herda de EventDispatcher. Quando uma ocorrência de AlarmClock está pronta para notificar outro código sobre a geração do evento `alarm`, o método `dispatchEvent()`, que também é herdado de EventDispatcher, é chamado.

```
var alarm:AlarmEvent = new AlarmEvent(this.alarmMessage);
this.dispatchEvent(alarm);
```

Essas linhas de código foram obtidas do método `onAlarm()` da classe AlarmClock (mostrada por completo antes). O método `dispatchEvent()` da ocorrência de AlarmClock é chamado e notifica todos os ouvintes registrados sobre o acionamento do evento `alarm` da ocorrência de AlarmClock. O parâmetro transmitido para `dispatchEvent()` é o objeto de evento que será transmitido junto com os métodos de ouvinte. Nesse caso, é uma ocorrência da classe AlarmEvent, uma subclasse de Event criada especificamente para este exemplo.

Fornecimento de um evento de alarme personalizado

Todos os ouvintes de evento recebem um parâmetro de objeto de evento com informações sobre o evento específico que está sendo acionado. Em muitos casos, o objeto de evento é uma ocorrência da classe Event. No entanto, em alguns casos é útil fornecer informações adicionais aos ouvintes de evento. Conforme descrito anteriormente neste capítulo, uma maneira comum de fazer isso é definir uma nova classe, uma subclasse de Event, e usar uma ocorrência dessa classe como objeto de evento. Neste exemplo, uma ocorrência de AlarmEvent é usada como objeto quando o evento `alarm` da classe AlarmClock é enviado. A classe AlarmEvent, mostrada aqui, fornece informações adicionais sobre o evento `alarm`, especificamente a mensagem de alarme:

```
import flash.events.Event;

/**
 * This custom Event class adds a message property to a basic Event.
 */
public class AlarmEvent extends Event
{
    /**
     * The name of the new AlarmEvent type.
     */
    public static const ALARM:String = "alarm";

    /**
     * A text message that can be passed to an event handler
     * with this event object.
     */
    public var message:String;

    /**
     *Constructor.
     *@param message The text to display when the alarm goes off.
     */
    public function AlarmEvent(message:String = "ALARM!")
    {
        super(ALARM);
        this.message = message;
    }
    ...
}
```

A melhor maneira de criar uma classe de objeto de evento personalizada é definir uma classe que estende a classe Event, como mostrou o exemplo anterior. Para complementar a funcionalidade herdada, a classe AlarmEvent define uma propriedade `message` que contém o texto da mensagem de alarme associada ao evento; o valor de `message` é transmitido como um parâmetro no construtor AlarmEvent. A classe AlarmEvent também define a constante `ALARM`, que pode ser usada para fazer referência ao evento específico (`alarm`) ao chamar o método `addEventListener()` da classe AlarmClock.

Além de adicionar a funcionalidade personalizada, cada subclasse de Event deve substituir o método `clone()` herdado como parte da estrutura de manipulação de eventos do ActionScript. As subclasses de Event também podem substituir o método `toString()` herdado para incluir as propriedades do evento personalizado no valor retornado quando o método `toString()` é chamado.

```
    /**
     * Creates and returns a copy of the current instance.
     * @return A copy of the current instance.
     */
    public override function clone():Event
    {
        return new AlarmEvent(message);
    }

    /**
     * Returns a String containing all the properties of the current
     * instance.
     * @return A string representation of the current instance.
     */
    public override function toString():String
    {
        return formatToString("AlarmEvent", "type", "bubbles", "cancelable", "eventPhase",
"message");
    }
```

O método `clone()` substituído precisa retornar uma nova ocorrência da subclasse personalizada de Event, com todas as propriedades personalizadas definidas para corresponder à ocorrência atual. No método `toString()` substituído, o método de utilitário `formatToString()` (herdado de Event) é usado para fornecer uma string com o nome do tipo personalizado, bem como nomes e valores de todas as propriedades.

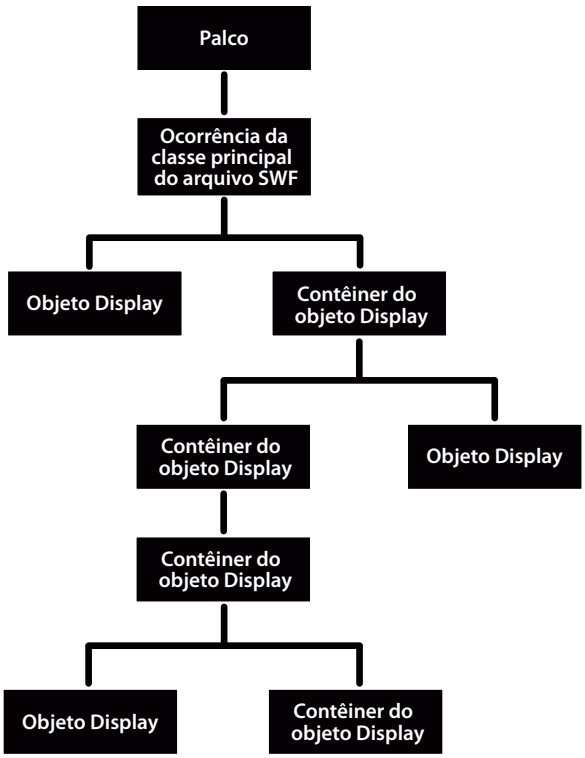
Capítulo 13: Programação de exibição

A programação de exibição do Adobe® ActionScript® 3.0 permite que você trabalhe com elementos que aparecem no palco do Adobe® Flash® Player ou do Adobe®AIR™. Este capítulo descreve os conceitos básicos do trabalho com elementos da tela. Você aprenderá detalhes sobre como organizar os elementos visuais de modo programático. Você também aprenderá a criar suas próprias classes personalizadas para objetos de exibição.

Noções básicas sobre a programação de exibição

Introdução à programação de exibição

Cada aplicativo desenvolvido no ActionScript 3.0 tem uma hierarquia de objetos exibidos, conhecida como *lista de exibição*, apresentada a seguir. A lista de exibição contém todos os elementos visíveis do aplicativo.



Conforme mostrado na ilustração, os elementos de exibição podem pertencer a um ou mais dos seguintes grupos:

- O Palco

  O Palco é o contêiner básico de objetos de exibição. Cada aplicativo tem um objeto Stage, que contém todos os objetos de exibição da tela. O Palco é o contêiner de nível superior e está no topo da hierarquia da lista de exibição:

  Cada arquivo SWF tem uma classe ActionScript associada, conhecida como *a classe principal do arquivo SWF*. Quando abre um arquivo SWF em uma página HTML, o Flash Player chama a função do construtor dessa classe e a ocorrência criada (que sempre é um tipo do objeto de exibição) é adicionada como filho do objeto Stage. A classe principal de um arquivo SWF sempre amplia a classe Sprite (para obter mais informações, consulte "Vantagens da abordagem da lista de exibição" na página 279).

  Você pode acessar o palco por meio da propriedade `stage` de qualquer ocorrência de DisplayObject. Para obter mais informações, consulte "Configuração de propriedades do palco" na página 287.

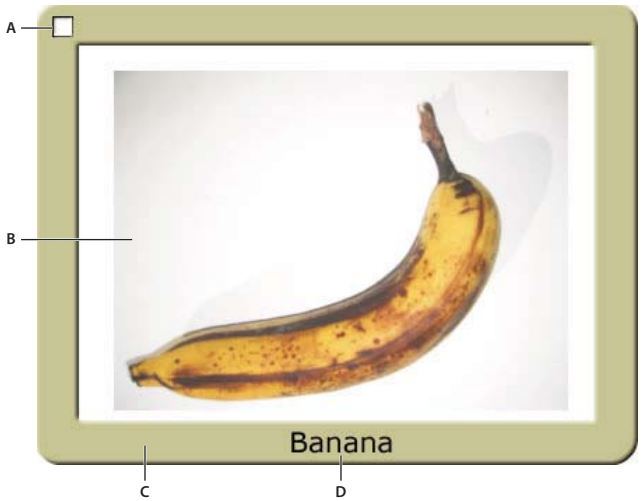
- Objetos de exibição

  No ActionScript 3.0, todos os elementos que aparecem na tela de um aplicativo são tipos de *objetos de exibição*. O pacote flash.display contém uma classe DisplayObject, que é uma classe base estendida por uma série de outras classes. Essas classes diferentes representam tipos diferentes de objetos de exibição, como formas vetoriais, clipes de filme e campos de texto, entre outros. Para obter uma visão geral dessas classes, consulte "Vantagens da abordagem da lista de exibição" na página 279.

- Contêineres de objeto de exibição

  Os contêineres são tipos especiais de objeto de exibição que, além de ter sua própria representação visual, podem conter objetos filho que também são objetos de exibição.

  A classe DisplayObjectContainer é uma subclasse da classe DisplayObject. Um objeto DisplayObjectContainer pode conter vários objetos de exibição em sua *lista defilhos*. Por exemplo, a ilustração a seguir mostra um tipo de objeto DisplayObjectContainer conhecido como Sprite que contém vários objetos de exibição:



A. *Um objeto SimpleButton. Esse tipo de objeto de exibição tem estados "up", "down" e "over" diferentes.* ***B.*** *Um objeto Bitmap. Nesse caso, o objeto Bitmap foi carregado a partir de um JPEG externo por meio de um objeto Loader.* ***C.*** *Um objeto Shape. O "quadro de imagem" contém um retângulo arredondado que é desenhado no ActionScript. Esse objeto Shape tem o filtro Sombra projetada aplicado.* ***D.*** *Um objeto TextField.*

No âmbito da discussão sobre objetos de exibição, os objetos DisplayObjectContainer também são conhecidos como *contêineres de objeto de exibição* ou simplesmente *contêineres*. Conforme mencionado anteriormente, o palco é um contêiner de objetos de exibição.

Embora todos os objetos de exibição visíveis sejam herdados da classe DisplayObject, o tipo de cada um é de uma subclasse específica da classe DisplayObject. Por exemplo, existe uma função de construtor para as classes Shape ou Video, mas não existe nenhum função assim para a classe DisplayObject.

Tarefas comuns de programação de exibição

Como grande parte da programação do ActionScript envolve a criação e a manipulação de elementos visuais, existem diversas tarefas relacionadas à programação de exibição. Este capítulo descreve as tarefas comuns que se aplicam a todos os objetos de exibição, incluindo:

- Trabalho com a lista de exibição e os contêineres de objeto de exibição
  - Adição de objetos à lista de exibição
  - Remoção de objetos da lista de exibição
  - Movimentação de objetos entre contêineres de exibição
  - Movimentação de objetos na frente ou atrás de outros objetos
- Trabalho com o palco
  - Configuração da taxa de quadros
  - Controle do dimensionamento do palco
  - Trabalho com o modo de tela cheia
- Manipulação de eventos de objeto de exibição
- Posicionamento de objetos de exibição, incluindo a criação da interação de arrastar e soltar
- Redimensionamento, dimensionamento e rotação dos objetos de exibição
- Aplicação de modos de mesclagem, transformações de cor e transparência em objetos de exibição
- Mascaramento de objetos de exibição
- Animação de objetos de exibição
- Carregamento de conteúdo de exibição externo (como arquivos SWF ou imagens)

Os últimos capítulos deste manual descrevem tarefas adicionais para trabalhar com objetos de exibição. Essas tarefas incluem tarefas gerais e tarefas associadas a tipos específicos de objetos de exibição:

- Desenho de gráficos vetoriais com o ActionScript nos objetos de exibição, descrito em "Uso de objetos visuais" na página 322
- Aplicação de transformações geométricas em objetos de exibição, descrito em "Trabalho com geometria" na página 342
- Aplicação de efeitos de filtro gráfico como desfoque, brilho e sombra projetada em objetos de exibição, descrito em "Filtro de objetos de exibição" na página 354
- Trabalho com características específicas de clipes de filme, descrito em "Trabalho com clipes de filme" na página 408
- Trabalho com objetos TextField, descrito em "Trabalho com texto" na página 434
- Trabalho com gráficos de bitmap, descrito em "Trabalho com bitmaps" na página 485
- Trabalho com elementos de vídeo, descrito em "Trabalho com vídeo" na página 527

Conceitos e termos importantes

A lista de referência a seguir contém termos importantes usados neste capítulo:

- Alfa: o valor de cor que representa a quantidade de transparência (melhor dizendo, a quantidade de opacidade) de uma cor. Por exemplo, uma cor com um canal alfa de 60% mostra apenas 60% de sua intensidade total e é 40% transparente.
- Gráfico de bitmap: um elemento gráfico que é definido no computador como uma grade (linhas e colunas) de pixels coloridos. Em geral, os gráficos de bitmap incluem fotos digitais e imagens similares.
- Modo de mesclagem: uma especificação de como o conteúdo de duas imagens sobrepostas deve interagir. Normalmente, uma imagem opaca em cima de outra imagem simplesmente bloqueia a imagem subjacente para que não fique visível; no entanto, modos de mesclagem diferentes fazem com que as cores das imagens se misturem de formas diferentes, de modo que o conteúdo resultante é alguma combinação das duas imagens.
- Lista de exibição: a hierarquia de objetos de exibição que será renderizada como o conteúdo visível na tela pelo Flash Player e pelo AIR. O palco é a raiz da lista de exibição e todos os objetos de exibição anexados ao palco ou a um de seus filhos formam a lista de exibição (mesmo que o objeto não seja realmente renderizado, por exemplo, se estiver fora dos limites do palco).
- Objeto de exibição: um objeto que representa algum tipo de conteúdo visual no Flash Player ou no AIR. Apenas objetos de exibição podem ser incluídos na lista de exibição e todas as classes desses objetos são subclasses de DisplayObject.
- Contêiner de objeto de exibição: um tipo especial de objeto de exibição que pode conter objetos de exibição filhos além de (geralmente) ter sua própria representação visual.
- Classe principal do arquivo SWF: classe que define o comportamento do objeto de exibição mais externo de um arquivo SWF e que, conceitualmente, é a classe do próprio arquivo SWF. Por exemplo, um SWF criado no Flash tem uma "linha do tempo principal" que contém todas as outras linhas do tempo; a classe principal do arquivo SWF é a classe da qual a linha do tempo principal é uma ocorrência.
- Mascaramento: técnica para ocultar algumas partes de uma imagem (ou de exibir apenas algumas partes da imagem). As partes mascaradas da imagem ficam transparentes, de modo que o conteúdo subjacente é exibido. O termo está relacionado à fita isolante do pintor que é usada para impedir que algumas áreas sejam pintadas.
- Palco: o contêiner visual que é a base ou o plano de fundo de todo o conteúdo visual de um SWF.
- Transformação: ajuste de uma característica visual de um gráfico como a rotação do objeto, a alteração da escala, a inclinação ou distorção da forma ou a alteração da cor.
- Gráfico vetorial: elemento gráfico que é definido no computador como linhas e formas desenhadas com características específicas (como espessura, comprimento, tamanho, ângulo e posição).

Teste dos exemplos do capítulo

Talvez você queira testar algumas das listagens de código de exemplo por si próprio, durante a leitura deste capítulo. Como este capítulo trata da criação e da manipulação do conteúdo visual, basicamente todas as listagens de código mencionadas criam objetos visuais e os exibem na tela; o teste do exemplo envolve a visualização do resultado no Flash Player ou no AIR, em vez da visualização de valores de variáveis como nos capítulos anteriores. Para testar as listagens de código deste capítulo:

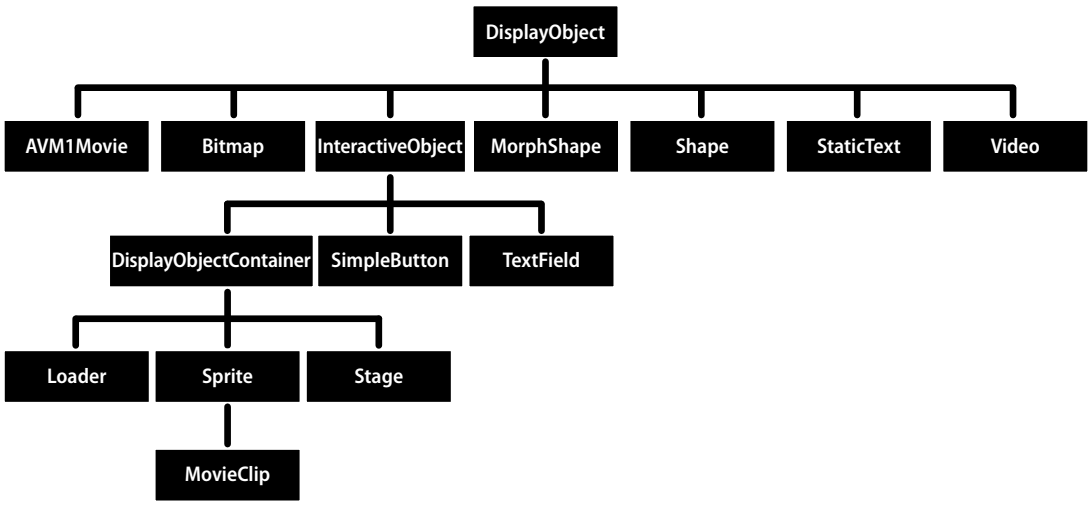
1. Criar um documento vazio usando a ferramenta de autoria do Flash
2. Selecione um quadro-chave na Linha de tempo.
3. Abra o painel Ações e copie a listagem de código no painel Script.
4. Execute o programa usando o comando Controlar > Testar filme.

Você verá os resultados do código exibidos na tela e todas as chamadas da função `trace()` serão exibidas no painel Saída.

As técnicas para testar listagens de código de exemplo são detalhadas em "Teste de listagens de código de exemplo dos capítulos" na página 36.

Principais classes de exibição

O pacote flash.display do ActionScript 3.0 inclui classes para objetos visuais que podem aparecer no Flash Player ou no AIR. A ilustração a seguir mostra as relações de subclasse dessas classes principais de objeto de exibição.



A ilustração mostra a herança das classes de objeto de exibição. Algumas dessas classes, especificamente StaticText, TextField e Video, não estão no pacote flash.display, mas ainda são herdadas da classe DisplayObject.

Todas as classes derivadas de DisplayObject herdam seus métodos e propriedades. Para obter mais informações, consulte "Propriedades e métodos da classe DisplayObject" na página 282.

É possível percorrer os objetos das seguintes classes contidas no pacote flash.display:

- Bitmap - A classe Bitmap pode ser usada para definir objetos de bitmap, carregados a partir de arquivos externos ou renderizados pelo ActionScript. É possível carregar bitmaps a partir de arquivos externos por meio da classe Loader. É possível carregar arquivos GIF, JPG ou PNG. Além disso, você pode criar um objeto BitmapData com dados personalizados e criar um objeto Bitmap que usa esses dados. Você pode usar os métodos da classe BitmapData para alterar bitmaps, sejam eles carregados ou criados no ActionScript. Para obter mais informações, consulte "Carregamento de objetos de exibição" na página 313 e "Trabalho com bitmaps" na página 485.
- Loader - A classe Loader pode ser usada para carregar ativos externos (arquivos SWF ou elementos gráficos). Para obter mais informações, consulte "Carregamento dinâmico do conteúdo da exibição" na página 313.
- Shape - A classe Shape pode ser usada para criar gráficos vetoriais, como retângulos, linhas, círculos e assim por diante. Para obter mais informações, consulte "Uso de objetos visuais" na página 322.
- SimpleButton - Um objeto SimpleButton é a representação do ActionScript do símbolo de um botão criado com a ferramenta de autoria do Flash. Uma ocorrência de SimpleButton tem quatro estados de botão: para cima, para baixo, por cima e teste de ocorrência (a área que responde aos eventos de mouse e de teclado).

- Sprite - Um objeto Sprite pode conter gráficos próprios e objetos de exibição filho. A classe Sprite estende a classe DisplayObjectContainer. Para obter mais informações, consulte "Trabalho com contêineres de objeto de exibição" na página 282 e "Uso de objetos visuais" na página 322.
- MovieClip - Um objeto MovieClip é a forma atribuída pelo ActionScript a um símbolo de clipe de filme criado com a ferramenta de autoria do Flash. Na prática, o objeto MovieClip é similar a um objeto Sprite, mas também tem uma linha do tempo. Para obter mais informações, consulte "Trabalho com clipes de filme" na página 408.

As classes a seguir, que não estão no pacote flash.display, são subclasses de DisplayObject:

- A classe TextField, incluída no pacote flash.text, é um objeto de exibição para entrada e exibição de texto. Para obter mais informações, consulte "Trabalho com texto" na página 434.
- A classe Video, incluída no pacote flash.media, é um objeto de exibição usado para exibir arquivos de vídeo. Para obter mais informações, consulte "Trabalho com vídeo" na página 527.

As seguintes classes do pacote flash.display estendem a classe DisplayObject, mas não é possível criar ocorrências delas. Em vez disso, elas servem como classes pai para outros objetos de exibição, combinando funcionalidades comuns em uma única classe.

- AVM1Movie - A classe AVM1Movie é usada para representar arquivos SWF carregados que são criados no ActionScript 1.0 e 2.0.
- DisplayObjectContainer - As classes Loader, Stage, Sprite e MovieClip estendem a classe DisplayObjectContainer. Para obter mais informações, consulte "Trabalho com contêineres de objeto de exibição" na página 282.
- InteractiveObject - InteractiveObject é a classe base de todos os objetos usados para interagir com o mouse e o teclado. Os objetos SimpleButton, TextField, Loader, Sprite, Stage e MovieClip são subclasses de InteractiveObject. Para obter mais informações sobre como criar a interação do mouse e do teclado, consulte "Captura da entrada do usuário" na página 600.
- MorphShape - Esses objetos são criados quando você cria uma interpolação de forma na ferramenta de criação do Flash. Não é possível percorrê-los usando o ActionScript, mas é possível acessá-los a partir da lista de exibição.
- Stage - A classe Stage estende a classe DisplayObjectContainer. Existe uma ocorrência de Stage em cada aplicativo, que fica no topo da hierarquia da lista de exibição. Para acessar o palco, use a propriedade `stage` de qualquer ocorrência de DisplayObject. Para obter mais informações, consulte "Configuração de propriedades do palco" na página 287.

Além disso, a classe StaticText do pacote flash.text estende a classe DisplayObject, mas não é possível criar uma ocorrência dela no código. Os campos de texto estático só podem ser criados no Flash.

Vantagens da abordagem da lista de exibição

No ActionScript 3.0, existem classes separadas para diferentes tipos de objeto de exibição. No ActionScript 1.0 e 2.0, muitos objetos do mesmo tipo são incluídos em uma classe: a classe MovieClip.

Essa individualização de classes e a estrutura hierárquica das listas de exibição têm as seguintes vantagens:

- Renderização mais eficiente e menos uso da memória
- Melhor gerenciamento de profundidade
- Profundidade completa da lista de exibição
- Objetos de exibição fora da lista
- Facilidade de subclassificação dos objetos de exibição

Renderização mais eficiente e arquivos menores

No ActionScript 1.0 e 2.0, você pode desenhar formas somente em um objeto MovieClip. No ActionScript 3.0, existem classes de objeto de exibição mais simples nas quais é possível desenhar formas. Como essas classes de objeto de exibição do ActionScript 3.0 não incluem um conjunto completo de métodos e propriedades como o objeto MovieClip, consomem menos recursos da memória e do processador.

Por exemplo, cada objeto MovieClip inclui propriedades para a linha do tempo do clipe de filme, enquanto o objeto Shape não faz isso. As propriedades de gerenciamento da linha do tempo podem usar muitos recursos da memória e do processador. No ActionScript 3.0, usar o objeto Shape melhora o desempenho. O objeto Shape tem uma sobrecarga menor do que o objeto MovieClip mais complexo. O Flash Player e o AIR não precisam gerenciar propriedades MovieClip não utilizadas, o que melhora a velocidade e reduz a memória usada pelo objeto.

Melhor gerenciamento de profundidade

No ActionScript 1.0 e 2.0, a profundidade era gerenciada por um esquema linear e métodos como `getNextHighestDepth()`.

O ActionScript 3.0 inclui a classe DisplayObjectContainer, que tem métodos e propriedades mais práticos para gerenciar a profundidade dos objetos de exibição.

No ActionScript 3.0, ao mover um objeto de exibição para uma nova posição da lista de filhos de uma ocorrência de DisplayObjectContainer, os outros filhos do contêiner de objeto de exibição são reposicionados automaticamente e designados nas posições de índice secundárias apropriadas no contêiner.

Além disso, no ActionScript 3.0, sempre é possível detectar todos os objetos filho de qualquer contêiner de objeto de exibição. Cada ocorrência de DisplayObjectContainer tem uma propriedade `numChildren`, que lista o número de filhos no contêiner de objeto de exibição. Como a lista de filhos de um contêiner de objeto de exibição sempre é uma lista indexada, você pode examinar todos os objetos da lista desde a posição 0 até a última posição de índice (`numChildren - 1`). Isso não era possível com os métodos e as propriedades de um objeto MovieClip no ActionScript 1.0 e no 2.0.

No ActionScript 3.0, você pode percorrer com facilidade a lista de exibição na seqüência; não existe nenhuma falha nos números de índice de uma lista de filhos de um contêiner de objeto de exibição. Percorrer a lista de exibição e gerenciar a profundidade dos objetos é muito mais fácil do que o que era permitido no ActionScript 1.0 e 2.0. No ActionScript 1.0 e 2.0, um clipe de filme podia ter objetos com lacunas intermitentes na ordem de profundidade, o que dificultava percorrer a lista de objetos. No ActionScript 3.0, cada lista de filhos de um contêiner de objeto de exibição é armazenada em cache internamente como uma matriz, resultando em consultas muito rápidas (por índice). Percorrer todos os filhos de um contêiner de objeto de exibição também é muito rápido.

No ActionScript 3.0, você também pode acessar os filhos de um contêiner de objeto de exibição usando o método `getChildByName()` da classe DisplayObjectContainer.

Profundidade completa da lista de exibição

No ActionScript 1.0 e 2.0, não era possível acessar alguns objetos, como formas vetoriais, que eram desenhados na ferramenta de criação do Flash. No ActionScript 3.0, você pode acessar todos os objetos da lista de exibição - os criados com o ActionScript e todos os objetos criados na ferramenta de criação do Flash. Para obter detalhes, consulte “Como percorrer a lista de exibição” na página 286.

Objetos de exibição fora da lista

No ActionScript 3.0, você pode criar objetos que não estão na lista de exibição visível. São os chamados objetos de exibição *fora da lista*. Um objeto de exibição é adicionado à lista visível somente quando você chama o método `addChild()` ou `addChildAt()` de uma ocorrência de DisplayObjectContainer que já foi adicionada à lista de exibição.

Você pode usar objetos de exibição fora da lista para montar objetos complexos, como os que têm vários contêineres com vários objetos de exibição. Mantendo os objetos de exibição fora da lista, você pode montar objetos complicados sem usar o tempo de processamento para renderizar esses objetos de exibição. Em seguida, você pode adicionar um objeto fora da lista à lista de exibição quando necessário. Além disso, é possível mover um filho de um contêiner de objeto de exibição na lista e fora dela e para qualquer posição desejada.

Facilidade de subclassificação dos objetos de exibição

No ActionScript 1.0 e 2.0, você tinha que adicionar com freqüência novos objetos MovieClip a um arquivo SWF para criar formas básicas ou exibir bitmaps. No ActionScript 3.0, a classe DisplayObject inclui muitas subclasses internas, como Shape e Bitmap. Como as classes do ActionScript 3.0 são mais especializadas para tipos específicos de objetos, é mais fácil criar subclasses básicas das classes internas.

Por exemplo, para desenhar um círculo no ActionScript 2.0, era necessário criar uma classe CustomCircle que estende a classe MovieClip quando um objeto da classe personalizada é percorrido. No entanto, essa classe também incluía diversos métodos e propriedades da classe MovieClip (como `totalFrames`) não apropriados. No ActionScript 3.0, entretanto, é possível criar uma classe CustomCircle que estende o objeto Shape e, desse modo, não inclui métodos e propriedades não relacionados que estão contidos na classe MovieClip. O código a seguir mostra um exemplo de uma classe CustomCircle:

```
import flash.display.*;

public class CustomCircle extends Shape
{
    var xPos:Number;
    var yPos:Number;
    var radius:Number;
    var color:uint;
    public function CustomCircle(xInput:Number,
                                 yInput:Number,
                                 rInput:Number,
                                 colorInput:uint)
    {
        xPos = xInput;
        yPos = yInput;
        radius = rInput;
        color = colorInput;
        this.graphics.beginFill(color);
        this.graphics.drawCircle(xPos, yPos, radius);
    }
}
```

Trabalho com os objetos de exibição

Agora que você já conhece os conceitos básicos de palco, objetos de exibição, contêineres de objeto de exibição e lista de exibição, esta seção fornece informações mais específicas sobre como trabalhar com objetos de exibição no ActionScript 3.0.

Propriedades e métodos da classe DisplayObject

Todos os objetos de exibição são subclasses de DisplayObject e, como tal, herda as propriedades e os métodos da classe DisplayObject. As propriedades herdadas são básicas e se aplicam a todos os objetos de exibição. Por exemplo, cada objeto de exibição tem uma propriedade `x` e uma propriedade `y` que especifica a posição do objeto no contêiner.

Não é possível criar uma ocorrência de DisplayObject usando o construtor da classe DisplayObject. Você deve criar outro tipo de objeto (que seja uma subclasse de DisplayObject), como Sprite, para percorrer um objeto com o operador `new`. Além disso, se desejar criar uma classe personalizada de objeto de exibição, você deve criar uma subclasse de uma das subclasses que têm uma função de construtor útil (como a classe Shape ou a classe Sprite). Para obter mais informações, consulte a descrição da classe DisplayObject na Referência dos componentes e da linguagem do ActionScript 3.0.

Adição de objetos à lista de exibição

Ao ser percorrido, o objeto de exibição não aparece na tela (no palco) até que você adicione a ocorrência desse objeto a um contêiner que está na lista de exibição. Por exemplo, no código a seguir, o objeto TextField `myText` não ficaria visível se a última linha do código fosse omitida. Na última linha do código, a palavra-chave `this` deve fazer referência a um contêiner já adicionado à lista de exibição.

```
import flash.display.*;
import flash.text.TextField;
var myText:TextField = new TextField();
myText.text = "Buenos dias.";
this.addChild(myText);
```

Quando algum elemento visual é adicionado ao palco, esse elemento se transforma em *filho* do objeto Stage. O primeiro arquivo SWF carregado em um aplicativo (por exemplo, aquele incorporado em uma página HTML) é adicionado automaticamente como filho do objeto Stage. Pode ser um objeto de qualquer tipo que estende a classe Sprite.

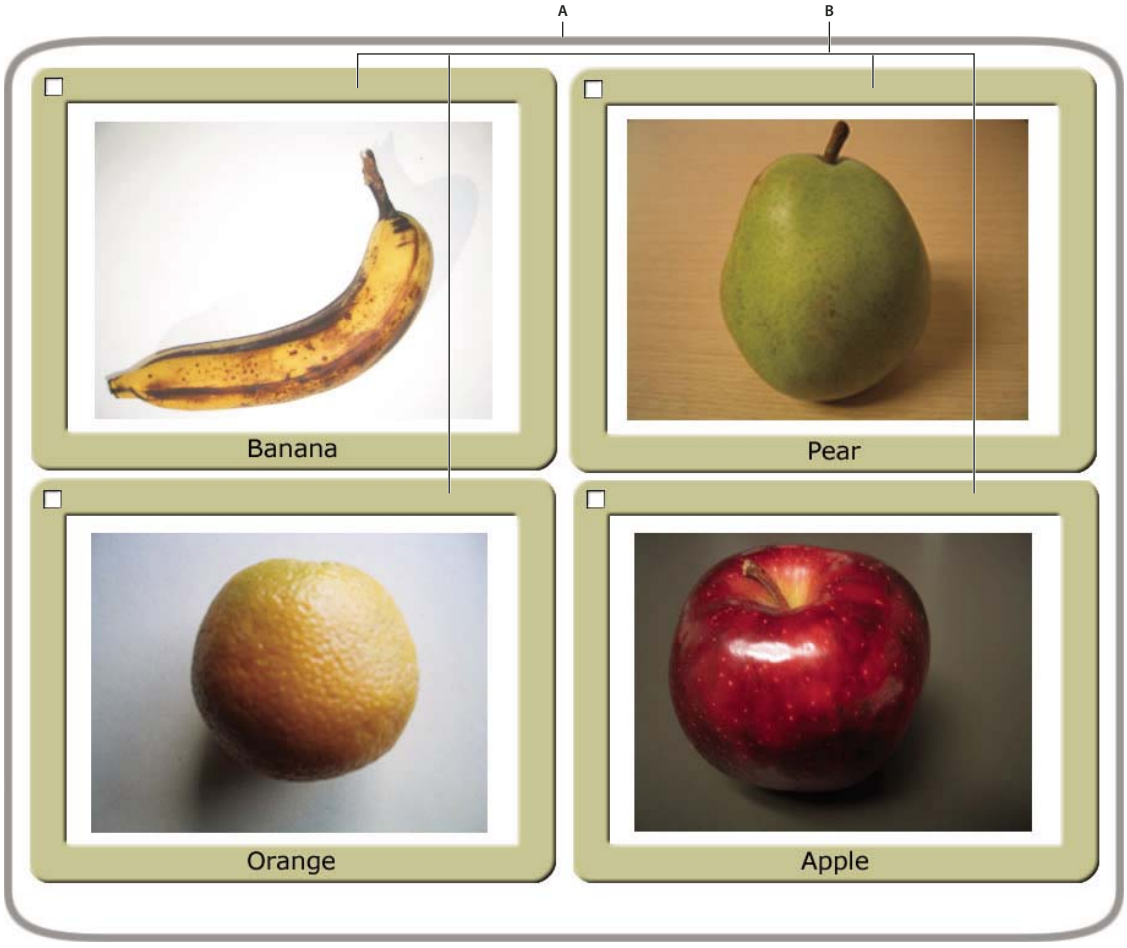
Todos os objetos de exibição criados *sem* o ActionScript (por exemplo, por meio da inserção de uma tag MXML no Adobe Flex Builder 3 ou da colocação de um item no palco do Flash) são adicionados à lista de exibição. Embora esses objetos de exibição não sejam adicionados por meio do ActionScript, é possível acessá-los usando o ActionScript. Por exemplo, o código a seguir ajusta a largura de um objeto chamado `button1` que foi inserido na ferramenta de criação (não por meio do ActionScript):

```
button1.width = 200;
```

Trabalho com contêineres de objeto de exibição

Se um objeto DisplayObjectContainer for excluído da lista de exibição, ou movido ou transformado de algum modo, cada objeto de exibição de DisplayObjectContainer também será excluído, movido ou transformado.

Um contêiner é um tipo de objeto de exibição propriamente dito e pode ser adicionado a outro contêiner. Por exemplo, a imagem a seguir mostra um contêiner de objeto de exibição, `pictureScreen`, que tem uma forma de contorno e outros quatro contêineres (do tipo PictureFrame):



A. *Uma forma que define a borda do contêiner de objeto de exibição pictureScreen* ***B.*** *Quatro contêineres de objeto de exibição que são filhos do objeto pictureScreen*

Para que um objeto apareça na lista de exibição, você deve adicioná-lo a um contêiner que esteja na lista. Para fazer isso, use o método `addChild()` ou o método `addChildAt()` do objeto de contêiner. Por exemplo, sem a linha final do código a seguir, o objeto `myTextField` não seria exibido:

```
var myTextField:TextField = new TextField();
myTextField.text = "hello";
this.root.addChild(myTextField);
```

Nesse exemplo de código, `this.root` aponta para o contêiner de objeto de exibição MovieClip que tem o código. No seu código real, você pode especificar um contêiner diferente.

Use o método `addChildAt()` para adicionar o filho a uma posição específica na lista de filhos do contêiner de objeto de exibição. Essas posições de índice baseadas em zero da lista de filhos estão relacionadas à disposição em camadas (ordem da frente para trás) dos objetos de exibição. Por exemplo, considere os três objetos de exibição a seguir. Cada objeto foi criado a partir de uma classe personalizada chamada Ball.

A disposição em camadas desses objetos de exibição no contêiner pode ser ajustada com o método `addChildAt()`. Por exemplo, considere o seguinte código:

```
ball_A = new Ball(0xFFCC00, "a");
ball_A.name = "ball_A";
ball_A.x = 20;
ball_A.y = 20;
container.addChild(ball_A);

ball_B = new Ball(0xFFCC00, "b");
ball_B.name = "ball_B";
ball_B.x = 70;
ball_B.y = 20;
container.addChild(ball_B);

ball_C = new Ball(0xFFCC00, "c");
ball_C.name = "ball_C";
ball_C.x = 40;
ball_C.y = 60;
container.addChildAt(ball_C, 1);
```

Depois que este código é executado, os objetos de exibição são posicionados do seguinte modo no `objeto` DisplayObjectContainer do contêiner. Observe a disposição em camadas dos objetos.

Para reposicionar um objeto na parte superior da lista de exibição, basta adicioná-lo novamente à lista. Por exemplo, depois do código anterior, para mover `ball_A` até a parte superior da pilha, use esta linha do código:

```
container.addChild(ball_A);
```

Este código remove `ball_A` por completo do seu local na lista de exibição do `contêiner` e o adicionam novamente à parte superior da lista, que equivale a movê-lo até a parte superior da pilha.

Você pode usar o método `getChildAt()` para verificar a ordem de camadas dos objetos de exibição. O método `getChildAt()` retorna objetos filho de um contêiner com base no número de índice fornecido. Por exemplo, o código a seguir revela nomes de objetos de exibição em posições diferentes da lista de filhos do objeto DisplayObjectContainer do `contêiner`:

```
trace(container.getChildAt(0).name); // ball_A
trace(container.getChildAt(1).name); // ball_C
trace(container.getChildAt(2).name); // ball_B
```

Se você remover um objeto de exibição da lista de filhos do contêiner pai, os elementos superiores da lista serão movidos para uma posição inferior no índice filho. Por exemplo, continuando com o código anterior, o código a seguir mostra como o objeto de exibição que estava na posição 2 no DisplayObjectContainer do `contêiner` será movido para a posição 1 se um objeto de exibição inferior da lista de filhos for removido:

```
container.removeChild(ball_C);
trace(container.getChildAt(0).name); // ball_A
trace(container.getChildAt(1).name); // ball_B
```

Os métodos `removeChild()` e `removeChildAt()` não excluem uma ocorrência de objeto de exibição por completo. Eles simplesmente a removem da lista de filhos do contêiner. A ocorrência ainda pode ser mencionada por outra variável. Use o operador `delete` para remover um objeto por completo.

Como um objeto de exibição tem apenas um contêiner pai, você pode adicionar uma ocorrência de um objeto de exibição a somente um contêiner. Por exemplo, o código a seguir mostra que o objeto de exibição `tf1` pode existir em apenas um contêiner (nesse caso, um Sprite, que estende a classe DisplayObjectContainer):

```
tf1:TextField = new TextField();
tf2:TextField = new TextField();
tf1.name = "text 1";
tf2.name = "text 2";

container1:Sprite = new Sprite();
container2:Sprite = new Sprite();

container1.addChild(tf1);
container1.addChild(tf2);
container2.addChild(tf1);

trace(container1.numChildren); // 1
trace(container1.getChildAt(0).name); // text 2
trace(container2.numChildren); // 1
trace(container2.getChildAt(0).name); // text 1
```

Se você adicionar um objeto de exibição que está contido em um contêiner a outro contêiner, esse objeto será removido da lista de filhos do primeiro contêiner.

Além dos métodos descritos acima, a classe DisplayObjectContainer define vários métodos para trabalhar com objetos de exibição filho, incluindo os seguintes:

- `contains()`: determina se um objeto de exibição é filho de DisplayObjectContainer.
- `getChildByName()`: recupera um objeto de exibição por nome.
- `getChildIndex()`: retorna a posição de índice de um objeto de exibição.
- `setChildIndex()`: altera a posição de um objeto de exibição filho.
- `swapChildren()`: alterna a ordem de frente para trás de dois objetos de exibição.

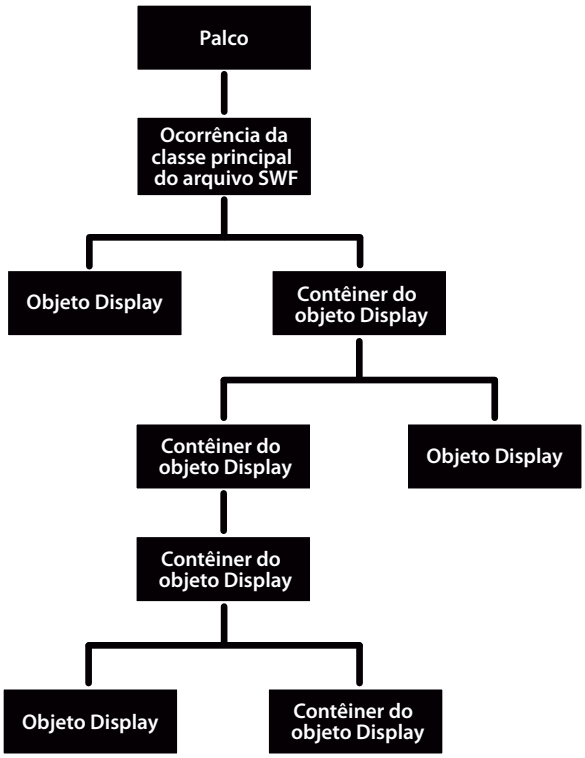
- `swapChildrenAt()`: alterna a ordem de frente para trás de dois objetos de exibição, especificados pelos valores de índice.

Para obter mais informações, consulte as entradas relevantes na Referência dos componentes e da linguagem do ActionScript 3.0.

Lembre-se de que um objeto de exibição que está fora da lista (não incluído em um contêiner de objeto de exibição que é filho do objeto Stage) é conhecido como objeto de exibição *fora da lista*.

Como percorrer a lista de exibição

Como você observou, a lista de exibição é uma estrutura de árvore. Na parte superior da árvore está o palco, que pode conter vários objetos de exibição. Esses objetos que são contêineres propriamente ditos podem ter outros objetos de exibição ou contêineres.



A classe DisplayObjectContainer inclui propriedades e métodos para percorrer a lista de exibição por meio das listas de filhos dos contêineres de objeto de exibição. Por exemplo, considere o código a seguir, que adiciona dois objetos de exibição, `title` e `pict`, ao objeto `container` (que é um Sprite; a classe Sprite estende a classe DisplayObjectContainer):

```
var container:Sprite = new Sprite();
var title:TextField = new TextField();
title.text = "Hello";
var pict:Loader = new Loader();
var url:URLRequest = new URLRequest("banana.jpg");
pict.load(url);
pict.name = "banana loader";
container.addChild(title);
container.addChild(pict);
```

O método `getChildAt()` retorna o filho da lista de exibição em uma posição de índice específica:

```
trace(container.getChildAt(0) is TextField); // true
```

Você também pode acessar objetos filho por nome. Cada objeto de exibição tem uma propriedade de nome e, se você não designá-la, o Flash Player ou AIR atribuirá um valor padrão, como `"instance1"`. Por exemplo, o código a seguir mostra como usar o método `getChildByName()` para acessar um objeto de exibição filho com o nome `"banana loader"`:

```
trace(container.getChildByName("banana loader") is Loader); // true
```

O método `getChildByName()` piora mais o desempenho do que o método `getChildAt()`.

Como um contêiner de objeto de exibição pode ter outros contêineres como objetos filho em sua lista de exibição, é possível percorrer a lista inteira do aplicativo como uma árvore. Por exemplo, no trecho de código mostrado anteriormente, assim que a operação de carregamento do objeto `pict` Loader for concluída, o objeto `pict` terá um objeto de exibição filho, que é o bitmap carregado. Para acessar esse objeto de exibição de bitmap, você pode gravar `pict.getChildAt(0)`. Você também pode gravar `container.getChildAt(0).getChildAt(0)` (visto que `container.getChildAt(0) == pict`).

A função a seguir fornece uma saída `trace()` pretendida da lista de exibição a partir de um contêiner:

```
function traceDisplayList(container:DisplayObjectContainer,indentString:String = ""):void
{
    var child:DisplayObject;
    for (var i:uint=0; i < container.numChildren; i++)
    {
        child = container.getChildAt(i);
        trace(indentString, child, child.name);
        if (container.getChildAt(i) is DisplayObjectContainer)
        {
            traceDisplayList(DisplayObjectContainer(child), indentString + "")
        }
    }
}
```

Configuração de propriedades do palco

A classe Stage substitui a maioria das propriedades e dos métodos da classe DisplayObject. Se um desses métodos ou propriedades substituídos for chamado, o Flash Player e o AIR lançarão uma exceção. Por exemplo, o objeto Stage não tem as propriedades `x` ou `y`, pois sua posição é fixa como o principal contêiner do aplicativo. As propriedades `x` e `y` fazem referência à posição de um objeto de exibição com relação ao seu contêiner e, como o objeto Stage não está contido em nenhum outro contêiner, essas propriedades não são aplicáveis.

Nota: *Alguns métodos e propriedades da classe Stage estão disponíveis somente para objetos de exibição que estão na mesma caixa de proteção de segurança do primeiro arquivo SWF carregado. Para obter detalhes, consulte "segurança de Palco" na página 722.*

Controle da taxa de quadros de reprodução

A propriedade `framerate` da classe Stage é usada para definir a taxa de quadros de todos os arquivos SWF carregados no aplicativo. Para obter mais informações, consulte a Referência dos componentes e da linguagem do ActionScript 3.0.

Controle do dimensionamento do palco

Quando uma parte da tela de representação do Flash Player ou do AIR é redimensionada, o Flash Player ou o AIR ajusta o conteúdo do palco automaticamente para fazer uma compensação. A propriedade `scaleMode` da classe Stage determina como o conteúdo do palco é ajustado. Essa propriedade pode ter quatro valores diferentes, definidos como constantes na classe flash.display.StageScaleMode.

Para três valores de `scaleMode` (`StageScaleMode.EXACT_FIT`, `StageScaleMode.SHOW_ALL` e `StageScaleMode.NO_BORDER)`, o Flash Player e o AIR dimensionam o conteúdo do palco para ajustá-lo dentro dos limites. O modo de dimensionamento é diferente nessas três opções:

- `StageScaleMode.EXACT_FIT` dimensiona o SWFproporcionalmente.
- `StageScaleMode.SHOW_ALL` determina se uma borda deve aparecer, como as barras pretas que são exibidas ao visualizar um filme em uma televisão de tela plana padrão.
- `StageScaleMode.NO_BORDER` determinase o conteúdo pode ser parcialmente cortado ou não.

Como alternativa, se `scaleMode` estiver definido como `StageScaleMode.NO_SCALE`, o conteúdo do palco mantém o tamanho definido quando o visualizador redimensiona a janela do Flash Player ou do AIR. Somente neste modo de escala, as propriedades stage`Width` e stage`Height` da classe Stage podem ser usadas para determinar as dimensões de pixel reais da janela redimensionada do Flash Player. Nos outros modos de escala, as propriedades `stageWidth` e `stageHeight` sempre refletem a largura e a altura originais do SWF. Além disso, quando `scaleMode` é definido como `StageScaleMode.NO_SCALE` e o arquivo SWF é redimensionado, o evento `resize` da classe Stage é enviado e permite fazer os ajustes adequados.

Conseqüentemente, definir `scaleMode` como `StageScaleMode.NO_SCALE` permite que você tenha mais controle sobre o ajuste do conteúdo da tela para redimensionar a janela se desejar. Por exemplo, em um arquivo SWF que contém um vídeo e uma barra de controle, você talvez queira que a barra de controle permaneça do mesmo tamanho quando o palco for redimensionado e apenas o tamanho da janela do vídeo seja alterado para acomodar o novo tamanho do palco. Isso é demonstrado no exemplo a seguir:

```
// videoScreen is a display object (e.g. a Video instance) containing a
// video; it is positioned at the top-left corner of the Stage, and
// it should resize when the SWF resizes.

// controlBar is a display object (e.g. a Sprite) containing several
// buttons; it should stay positioned at the bottom-left corner of the
// Stage (below videoScreen) and it should not resize when the SWF
// resizes.

import flash.display.Stage;
import flash.display.StageAlign;
import flash.display.StageScaleMode;
import flash.events.Event;

var swfStage:Stage = videoScreen.stage;
swfStage.scaleMode = StageScaleMode.NO_SCALE;
swfStage.align = StageAlign.TOP_LEFT;

function resizeDisplay(event:Event):void
{
    var swfWidth:int = swfStage.stageWidth;
    var swfHeight:int = swfStage.stageHeight;

    // Resize the video window.
    var newVideoHeight:Number = swfHeight - controlBar.height;
    videoScreen.height = newVideoHeight;
    videoScreen.scaleX = videoScreen.scaleY;

    // Reposition the control bar.
    controlBar.y = newVideoHeight;
}

swfStage.addEventListener(Event.RESIZE, resizeDisplay);
```

Trabalho com o modo de tela cheia

O modo de tela cheia permite que você configure o palco de um filme para preencher todo o monitor do visualizador, sem nenhuma borda ou menu de contêiner. A propriedade `displayState` da classe Stage é usada para ativar e desativar o modo de tela cheia para um SWF. A propriedade `displayState` pode ser configurada como um dos valores definidos pelas constantes da classe flash.display.StageDisplayState. Para ativar o modo de tela cheia, defina `displayState` como `StageDisplayState.FULL_SCREEN`:

```
// Send the stage to full-screen in ActionScript 3.0
stage.displayState = StageDisplayState.FULL_SCREEN;

// Exit full-screen mode in ActionScript 3.0
stage.displayState = StageDisplayState.NORMAL;

// Send the stage to full-screen in ActionScript 2.0
Stage.displayState = "fullScreen";

// Exit full-screen mode in ActionScript 2.0
Stage.displayState = "normal";
```

Além disso, o usuário pode optar por sair do modo de tela cheia alternando o foco para uma janela diferente ou usando uma de várias combinações de tecla: a tecla Esc (todas as plataformas), Ctrl-W (Windows), Command-W (Mac) ou Alt-F4 (Windows).

O comportamento de dimensionamento do palco para o modo de tela cheia é igual ao de um modo normal; o dimensionamento é controlado pela propriedade `scaleMode` da classe Stage. Se a propriedade `scaleMode` estiver definida como `StageScaleMode.NO_SCALE`, as propriedades `stageWidth` e `stageHeight` da classe Stage serão alteradas para refletir o tamanho da área da tela ocupado pelo SWF (a tela inteira, nesse caso); se for visualizado no navegador, o parâmetro HTML controlará a configuração.

Você pode usar o evento `fullScreen` da classe Stage para detectar e responder quando o modo de tela cheia está ativado ou desativado. Por exemplo, você talvez queira reposicionar, adicionar ou remover itens da tela ao acessar ou sair do modo de tela cheia, como mostra este exemplo:

```
import flash.events.FullScreenEvent;

function fullScreenRedraw(event:FullScreenEvent):void
{
    if (event.fullScreen)
    {
        // Remove input text fields.
        // Add a button that closes full-screen mode.
    }
    else
    {
        // Re-add input text fields.
        // Remove the button that closes full-screen mode.
    }
}

mySprite.stage.addEventListener(FullScreenEvent.FULL_SCREEN, fullScreenRedraw);
```

Como mostra este código, o objeto do evento `fullScreen` é uma ocorrência da classe flash.events.FullScreenEvent, que inclui uma propriedade `fullScreen` que indica se o modo de tela cheia está ativado (`true`) ou não (`false`).

Ao trabalhar com o modo de tela cheia no ActionScript, considere o seguinte:

- No Flash Player, o modo de tela cheia só pode ser iniciado por meio do ActionScript em resposta a um clique do mouse (incluindo o clique com o botão direito) ou pressionamento de tecla. O conteúdo do AIR em execução na caixa de proteção de segurança do aplicativo não requer a ativação do modo de tela cheia em resposta a um gesto do usuário.
- Para usuários com vários monitores, o conteúdo do SWF será expandido para preencher apenas um monitor. O Flash Player e o AIR usam uma métrica para determinar qual monitor contém a maior parte do SWF e usam esse monitor para o modo de tela cheia.
- Para um arquivo SWF incorporado em uma página HTML, o código HTML a ser incorporado no Flash Player deve incluir uma tag `param` e o atributo `embed` com o nome `allowFullScreen` e o valor `true`, do seguinte modo:

  ```
  <object>
      ...
      <param name="allowFullScreen" value="true" />
      <embed ... allowfullscreen="true" />
  </object>
  ```

  Se estiver usando JavaScript em uma página da Web para gerar as tags incorporadas no SWF, altere o JavaScript para adicionar a tag `allowFullScreen param` e o atributo. Por exemplo, se a página HTML usa a função `AC_FL_RunContent()` (que é usada pelas páginas HTML geradas pelo Flex Builder e pelo Flash), adicione o parâmetro `allowFullScreen` a essa chamada de função do seguinte modo:

```
AC_FL_RunContent(
    ...
    'allowFullScreen','true',
    ...
    ); //end AC code
```

Isso não se aplica aos arquivos SWF em execução no Flash Player autônomo.

- Todo ActionScript relacionado ao teclado, como eventos de teclado e entrada de texto nas ocorrências de TextField, é desativado no modo de tela cheia. Os atalhos de teclado que fecham o modo de tela cheia são a exceção.

Existem também algumas restrições adicionais relacionadas à segurança que devem ser consideradas. Elas estão descritas em "Caixas de proteção de segurança" na página 706.

Dimensionamento em hardware

Você pode usar a propriedade `fullScreenSourceRect` da classe Stage para configurar o Flash Player ou o AIR para dimensionar uma região específica do palco no modo de tela cheia. O Flash Player e o AIR são dimensionados em hardware, se disponível, usando os gráficos e a placa de vídeo no computador de um usuário, e geralmente exibem o conteúdo mais rapidamente do que no dimensionamento em software.

Para tirar vantagem do dimensionamento em hardware, defina o palco inteiro ou parte dele para o modo de tela cheia. O código ActionScript 3.0 a seguir define o palco inteiro para o modo de tela cheia:

```
import flash.geom.*;
{
stage.fullScreenSourceRect = new Rectangle(0,0,320,240);
stage.displayState = StageDisplayState.FULL_SCREEN;
}
```

Quando essa propriedade é definida como um retângulo válido e a propriedade `displayState` é definida como o modo de tela cheia o Flash Player e o AIR dimensionam a área especificada. O tamanho real do Palco em pixels no ActionScript não é alterado. O Flash Player e o AIR forçam um limite mínimo para o tamanho do retângulo de forma a acomodar a mensagem padrão "Pressione Esc para sair do modo de tela cheia". Em geral, esse limite está em torno de 260 por 30 pixels, mas pode variar de acordo com a plataforma e a versão do Flash Player.

A propriedade `fullScreenSourceRect` só pode ser definida quando o Flash Player ou AIR não está no modo de tela cheia. Para usá-la corretamente, defina-a primeiro e depois defina a propriedade `displayState` como o modo de tela cheia, como mostram os exemplos de código.

Para ativar o dimensionamento, defina a propriedade `fullScreenSourceRect` como um objeto de retângulo.

```
stage.fullScreenSourceRect = new Rectangle(0,0,320,240); // Valid, will enable hardware
scaling.
```

Para desativar o dimensionamento, defina a propriedade `fullScreenSourceRect` como `null` no ActionScript 3.0 e como `undefined` no ActionScript 2.0.

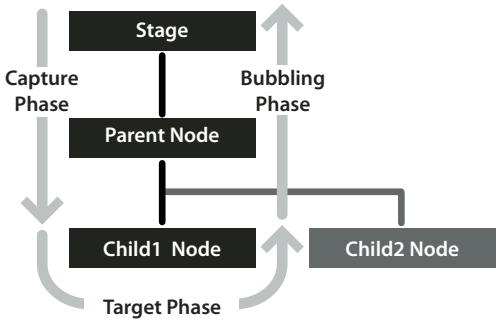
```
stage.fullScreenSourceRect = null;
```

Manipulação de eventos de objetos de exibição

A classe DisplayObject é herdada da classe EventDispatcher. Desse modo, todos os objetos de exibição podem participar plenamente no modelo de evento (descrito em "Manipulação de eventos" na página 251). Todos os objetos de exibição podem usar seu método `addEventListener()` (herdado da classe EventDispatcher) para ouvir um evento específico, mas somente se o objeto ouvinte fizer parte do fluxo desse evento.

Quando o Flash Player ou o AIR envia um objeto de evento, esse objeto faz uma viagem de ida e volta do palco até o objeto de exibição onde ocorreu o evento. Por exemplo, se o usuário clicar em um objeto de exibição chamado `child1`, o Flash Player enviará um objeto de evento do palco, por meio da hierarquia da lista de exibição, até o objeto de exibição `child1`.

O fluxo do evento é conceitualmente dividido em três fases, como mostra este diagrama:



Para obter mais informações, consulte "Manipulação de eventos" na página 251.

Uma questão importante que deve ser considerada ao trabalhar com eventos de objeto de exibição é o efeito que os ouvintes de evento podem ter quando os objetos de exibição removidos automaticamente da memória (lixo coletado) forem removidos da lista de exibição. Se um objeto de exibição tiver objetos inscritos como ouvintes de eventos, esse objeto não será removido da memória mesmo quando for removido da lista de exibição, pois ainda terá referências a esses objetos de ouvinte. Para obter mais informações, consulte "Gerenciamento de ouvintes de evento" na página 265.

Escolha de uma subclasse de DisplayObject

Com tantas opções disponíveis, ao trabalhar com objetos de exibição, uma das decisões importantes é definir qual objeto será usado para cada finalidade. Veja algumas diretrizes que podem ajudá-lo a tomar essa decisão. Essas mesmas sugestões podem ser usadas quando você precisa de uma ocorrência de uma classe ou está escolhendo uma base para criar uma classe:

- Se você não precisar de um objeto que possa ser contêiner de outros objetos de exibição (isto é, precisar de apenas um que sirva como um elemento de tela autônomo), escolha uma dessas subclasses de DisplayObject ou InteractiveObject, dependendo do uso pretendido:
  - Bitmap para exibir uma imagem de bitmap.
  - TextField para adicionar texto.
  - Video para exibir vídeo.
  - Shape para uma "tela" de desenho do conteúdo na tela. Especificamente, se desejar criar uma ocorrência para desenhar formas na tela que não será contêiner de outros objetos de exibição, use Shape em vez de Sprite ou MovieClip para melhorar o desempenho significativamente.
  - MorphShape, StaticText ou SimpleButton para itens criados com a ferramenta de autoria do Flash. Não é possível criar ocorrências dessas classes de modo programático, mas você pode criar variáveis com esses tipos de dados para fazer referência aos itens criados com a ferramenta da autoria do Flash.
- Se precisar de uma variável para fazer referência ao palco principal, use a classe Stage como tipo de dados.

- Se precisar de um contêiner para carregar um arquivo SWF ou de imagem externo, use uma ocorrência de Loader. O conteúdo carregado será adicionado à lista de exibição como filho da ocorrência de Loader. O tipo de dados depende da natureza do conteúdo carregado, do seguinte modo:
  - Uma imagem carregada será uma ocorrência do Bitmap.
  - Um arquivo SWF carregado gravado no ActionScript 3.0 será uma ocorrência de Sprite ou MovieClip (ou uma ocorrência de uma subclasse dessas classes, conforme especificado pelo criador do conteúdo).
  - Um arquivo SWF carregado gravado no ActionScript 1.0 ou no ActionScript 2.0 será uma ocorrência de AVM1Movie.
- Se precisar de um objeto para servir como contêiner de outros objetos de exibição (esteja você desenhando ou não no objeto de exibição com o ActionScript), escolha uma das subclasses de DisplayObjectContainer:
  - Sprite se o objeto for criado apenas com o ActionScript ou como a classe base de um objeto de exibição personalizado que será criado e manipulado somente com o ActionScript.
  - MovieClip se estiver criando uma variável para fazer referência a um símbolo de clipe de filme criado na ferramenta de criação do Flash.
- Se estiver criando uma classe que será associada a um símbolo de clipe de filme na biblioteca do Flash, escolha uma destas subclasses de DisplayObjectContainer como sua classe base:
  - MovieClip se o símbolo de clipe de filme associado tiver conteúdo em mais de um quadro
  - Sprite se o símbolo de clipe de filme associado tiver conteúdo somente no primeiro quadro

Manipulação de objetos de exibição

Independentemente do objeto de exibição utilizado, existem diversas manipulações comuns a todos os objetos de exibição que servem como elementos exibidos na tela. Por exemplo, todos podem ser posicionados na tela, movidos para frente ou para trás na ordem de empilhamento dos objetos de exibição, dimensionados, girados e assim por diante. Como todos os objetos de exibição herdam essa funcionalidade da classe base comum (DisplayObject), tal funcionalidade apresenta o mesmo comportamento demonstrado na manipulação de uma ocorrência de TextField, de Video, de Shape ou de qualquer outro objeto de exibição. As seções a seguir descrevem em detalhes diversas manipulações comuns de objeto de exibição.

Alteração da posição

A manipulação mais básica de qualquer objeto de exibição é seu posicionamento na tela. Para definir a posição de um objeto de exibição, altere as propriedades `x` e `y` do objeto.

```
myShape.x = 17;
myShape.y = 212;
```

O sistema de posicionamento de objetos de exibição trata o palco como um sistema de coordenadas cartesianas (o sistema de grade comum com um eixo x horizontal e um eixo y vertical). A origem do sistema de coordenadas (a coordenada 0,0 onde os eixos x e y se encontram) está no canto superior esquerdo do palco. A partir desse ponto, os valores de x são positivos para a direita e negativos para a esquerda, enquanto (diferente dos sistemas gráficos típicos) os valores de y são positivos para baixo e negativos para cima. Por exemplo, as linhas anteriores do código movem o objeto `myShape` até a coordenada 17 do eixo x (17 pixel à direita da origem) e a coordenada 212 do eixo y (212 pixels abaixo da origem).

Por padrão, quando um objeto de exibição é criado com o ActionScript, as propriedades `x` e `y` são definidas como 0, colocando o objeto no canto superior esquerdo do conteúdo pai.

Alteração da posição em relação ao palco

É importante lembrar que as propriedades `x` e `y` sempre fazem referência à posição do objeto de exibição em relação à coordenada 0,0 dos eixos do objeto de exibição pai. Assim, para uma ocorrência de Shape (como um círculo) contida em uma ocorrência de Sprite, se você definir as propriedades `x` e `y` do objeto Shape como 0, o círculo será colocado no canto superior esquerdo de Sprite, que não é necessariamente o canto superior esquerdo do palco. Para posicionar um objeto em relação às coordenadas globais do palco, você pode usar o método `globalToLocal()` de qualquer objeto de exibição para converter coordenadas globais (palco) em locais (contêiner do objeto de exibição), do seguinte modo:

```
// Position the shape at the top-left corner of the Stage,
// regardless of where its parent is located.

// Create a Sprite, positioned at x:200 and y:200.
var mySprite:Sprite = new Sprite();
mySprite.x = 200;
mySprite.y = 200;
this.addChild(mySprite);

// Draw a dot at the Sprite's 0,0 coordinate, for reference.
mySprite.graphics.lineStyle(1, 0x000000);
mySprite.graphics.beginFill(0x000000);
mySprite.graphics.moveTo(0, 0);
mySprite.graphics.lineTo(1, 0);
mySprite.graphics.lineTo(1, 1);
mySprite.graphics.lineTo(0, 1);
mySprite.graphics.endFill();

// Create the circle Shape instance.
var circle:Shape = new Shape();
mySprite.addChild(circle);

// Draw a circle with radius 50 and center point at x:50, y:50 in the Shape.
circle.graphics.lineStyle(1, 0x000000);
circle.graphics.beginFill(0xff0000);
circle.graphics.drawCircle(50, 50, 50);
circle.graphics.endFill();

// Move the Shape so its top-left corner is at the Stage's 0, 0 coordinate.
var stagePoint:Point = new Point(0, 0);
var targetPoint:Point = mySprite.globalToLocal(stagePoint);
circle.x = targetPoint.x;
circle.y = targetPoint.y;
```

Do mesmo modo, você pode usar o método `localToGlobal()` da classe DisplayObject para converter coordenadas locais em coordenadas do palco.

Criação da interação de arrastar e soltar

Um objeto de exibição normalmente é movido para criar uma interação de arrastar e soltar para que, quando o usuário clicar, o objeto se mova junto com o movimento do mouse até que o botão do mouse seja solto. A interação de arrastar e soltar pode ser criada de duas formas no ActionScript. Em qualquer uma delas, dois eventos de mouse são usados: quando o botão do mouse é pressionado, o objeto é acionado para seguir o cursor do mouse e, quando é solto, o objeto deve parar de segui-lo.

A primeira forma, usando o método `startDrag()`, é mais simples, porém mais limitada. Quando o botão do mouse é pressionado, o método `startDrag()` é chamado para arrastar o objeto de exibição. Quando o botão do mouse é solto, o método `stopDrag()` é chamado.

```
// This code creates a drag-and-drop interaction using the startDrag()
// technique.
// square is a DisplayObject (e.g. a MovieClip or Sprite instance).

import flash.events.MouseEvent;

// This function is called when the mouse button is pressed.
function startDragging(event:MouseEvent):void
{
    square.startDrag();
}

// This function is called when the mouse button is released.
function stopDragging(event:MouseEvent):void
{
    square.stopDrag();
}

square.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
square.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

Essa técnica tem uma limitação bem significativa: somente um item por vez pode ser arrastado com `startDrag()`. Se um objeto de exibição estiver sendo arrastado e o método `startDrag()` for chamado em outro objeto de exibição, o primeiro objeto deixará de seguir o mouse imediatamente. Por exemplo, se a função `startDragging()` for alterada conforme mostrado aqui, somente o objeto `circle` será arrastado, apesar da chamada do método `square.startDrag()`:

```
function startDragging(event:MouseEvent):void
{
    square.startDrag();
    circle.startDrag();
}
```

Como apenas um objeto pode ser arrastado por vez com `startDrag()`, o método `stopDrag()` pode ser chamado em qualquer objeto de exibição e pára sempre que o objeto está sendo arrastado.

Se precisar arrastar mais de um objeto de exibição ou para evitar a possibilidade de conflitos quando houver mais de um objeto que provavelmente use `startDrag()`, é melhor usar a técnica de acompanhamento do mouse para criar o efeito de desenho. Com essa técnica, quando o botão do mouse é pressionado, uma função é inscrita como ouvinte no evento `mouseMove` do palco. Essa função, que é chamada sempre que o mouse se move, faz com que o objeto arrastado passe para a coordenada x, y do mouse. Assim que o botão do mouse é solto, a função deixa de ser ouvinte, ou seja, não será mais chamada quando o mouse se movimentar, e o objeto pára de acompanhar o cursor do mouse. Veja um código que demonstra essa técnica:

```
// This code creates a drag-and-drop interaction using the mouse-following
// technique.
// circle is a DisplayObject (e.g. a MovieClip or Sprite instance).

import flash.events.MouseEvent;

var offsetX:Number;
var offsetY:Number;

// This function is called when the mouse button is pressed.
function startDragging(event:MouseEvent):void
{
    // Record the difference (offset) between where
    // the cursor was when the mouse button was pressed and the x, y
    // coordinate of the circle when the mouse button was pressed.
    offsetX = event.stageX - circle.x;
    offsetY = event.stageY - circle.y;

    // tell Flash Player to start listening for the mouseMove event
    stage.addEventListener(MouseEvent.MOUSE_MOVE, dragCircle);
}

// This function is called when the mouse button is released.
function stopDragging(event:MouseEvent):void
{
    // Tell Flash Player to stop listening for the mouseMove event.
    stage.removeEventListener(MouseEvent.MOUSE_MOVE, dragCircle);
}

// This function is called every time the mouse moves,
// as long as the mouse button is pressed down.
function dragCircle(event:MouseEvent):void
{
    // Move the circle to the location of the cursor, maintaining
    // the offset between the cursor's location and the
    // location of the dragged object.
    circle.x = event.stageX - offsetX;
    circle.y = event.stageY - offsetY;

    // Instruct Flash Player to refresh the screen after this event.
    event.updateAfterEvent();
}

circle.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
circle.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

Além de fazer com que um objeto de exibição acompanhe o cursor do mouse, uma parte comum da interação de arrastar e soltar inclui a movimentação do objeto arrastado até a frente da exibição para que apareça flutuando acima de todos os outros objetos. Por exemplo, imagine que você tenha dois objetos, um círculo e um quadrado, e que os dois têm uma interação de arrastar e soltar. Se o círculo ficar abaixo do quadrado na lista de exibição e você clicar e arrastar o círculo para que o cursor fique por cima do quadrado, irá parecer que o círculo desliza ao lado do quadrado, quebrando a ilusão de arrastar e soltar. Em vez disso, você pode definir que, quando for clicado, o círculo deve se mover para cima da lista de exibição e, assim, sempre aparecer em cima de qualquer outro conteúdo.

O código a seguir (adaptado do exemplo anterior) cria uma interação de arrastar e soltar para dois objetos de exibição: um círculo e um quadrado. Sempre que o botão do mouse é pressionado em um deles, esse item é movido para a parte superior da lista de exibição do palco para que o item arrastado sempre apareça por cima. O código novo ou alterado a partir da listagem anterior aparece em negrito.

```
// This code creates a drag-and-drop interaction using the mouse-following
// technique.
// circle and square are DisplayObjects (e.g. MovieClip or Sprite
// instances).

import flash.display.DisplayObject;
import flash.events.MouseEvent;

var offsetX:Number;
var offsetY:Number;
var draggedObject:DisplayObject;

// This function is called when the mouse button is pressed.
function startDragging(event:MouseEvent):void
{
    // remember which object is being dragged
    draggedObject = DisplayObject(event.target);

    // Record the difference (offset) between where the cursor was when
    // the mouse button was pressed and the x, y coordinate of the
    // dragged object when the mouse button was pressed.
    offsetX = event.stageX - draggedObject.x;
    offsetY = event.stageY - draggedObject.y;

    // move the selected object to the top of the display list
    stage.addChild(draggedObject);

    // Tell Flash Player to start listening for the mouseMove event.
    stage.addEventListener(MouseEvent.MOUSE_MOVE, dragObject);
}

// This function is called when the mouse button is released.
function stopDragging(event:MouseEvent):void
{
    // Tell Flash Player to stop listening for the mouseMove event.
    stage.removeEventListener(MouseEvent.MOUSE_MOVE, dragObject);
}
```

```
// This function is called every time the mouse moves,
// as long as the mouse button is pressed down.
function dragObject(event:MouseEvent):void
{
    // Move the dragged object to the location of the cursor, maintaining
    // the offset between the cursor's location and the location
    // of the dragged object.
    draggedObject.x = event.stageX - offsetX;
    draggedObject.y = event.stageY - offsetY;

    // Instruct Flash Player to refresh the screen after this event.
    event.updateAfterEvent();
}

circle.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
circle.addEventListener(MouseEvent.MOUSE_UP, stopDragging);

square.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
square.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

Para estender esse efeito ainda mais, como para um jogo onde pinos ou cartas são movidos entre pilhas, você pode adicionar o objeto arrastado à lista de exibição do palco quando for "tirado" e adicioná-lo a outra lista de exibição (como a "pilha" onde é solto) quando o botão do mouse for liberado.

Finalmente, para aprimorar o efeito, você poderia aplicar um filtro de sombra projetada no objeto de exibição quando for clicado (quando começar a ser arrastado) e remover a sombra projetada quando o objeto for solto. Para obter detalhes sobre como usar o filtro de sombra projetada e outros filtros de objeto de exibição no ActionScript, consulte "Filtro de objetos de exibição" na página 354.

Visão panorâmica e rolagem de objetos de exibição

Se houver um objeto de exibição muito grande para a área na qual deseja exibi-lo, você pode usar a propriedade `scrollRect` para definir a área visível do objeto de exibição. Além disso, alterando a propriedade `scrollRect` em resposta à entrada do usuário, você pode obter uma visão panorâmica do conteúdo à esquerda e à direita e percorrer para cima e para baixo.

A propriedade `scrollRect` é uma ocorrência da classe Rectangle, que combina os valores necessários para definir uma área retangular como um único objeto. Para definir inicialmente a área visível do objeto de exibição, crie uma nova ocorrência de Rectangle e a atribua à propriedade `scrollRect` do objeto de exibição. Posteriormente, para percorrer ou obter a visão panorâmica, leia a propriedade `scrollRect` em uma variável separada de Rectangle e altere a propriedade desejada (por exemplo, altere a propriedade `x` da ocorrência de Rectangle para obter a visão panorâmica ou a propriedade `y` para percorrer). Em seguida, atribua novamente essa ocorrência de Rectangle à propriedade `scrollRect` para notificar o objeto de exibição do valor alterado.

Por exemplo, o código a seguir define a área visível de um objeto TextField chamado `bigText` que é muito pequeno para se adaptar nos limites do arquivo SWF. Quando são clicados, os dois botões chamados `up` e `down` chamam funções que fazem com que o conteúdo do objeto TextField se movimente para cima ou para baixo, modificando a propriedade `y` da ocorrência `scrollRect` de Rectangle.

```
import flash.events.MouseEvent;
import flash.geom.Rectangle;

// Define the initial viewable area of the TextField instance:
// left: 0, top: 0, width: TextField's width, height: 350 pixels.
bigText.scrollRect = new Rectangle(0, 0, bigText.width, 350);

// Cache the TextField as a bitmap to improve performance.
bigText.cacheAsBitmap = true;

// called when the "up" button is clicked
function scrollUp(event:MouseEvent):void
{
    // Get access to the current scroll rectangle.
    var rect:Rectangle = bigText.scrollRect;
    // Decrease the y value of the rectangle by 20, effectively
    // shifting the rectangle down by 20 pixels.
    rect.y -= 20;
    // Reassign the rectangle to the TextField to "apply" the change.
    bigText.scrollRect = rect;
}

// called when the "down" button is clicked
function scrollDown(event:MouseEvent):void
{
    // Get access to the current scroll rectangle.
    var rect:Rectangle = bigText.scrollRect;
    // Increase the y value of the rectangle by 20, effectively
    // shifting the rectangle up by 20 pixels.
    rect.y += 20;
    // Reassign the rectangle to the TextField to "apply" the change.
    bigText.scrollRect = rect;
}

up.addEventListener(MouseEvent.CLICK, scrollUp);
down.addEventListener(MouseEvent.CLICK, scrollDown);
```

Como este exemplo ilustra, ao trabalhar com a propriedade `scrollRect` de um objeto de exibição, é melhor especificar que o Flash Player ou o AIR deve armazenar em cache o conteúdo do objeto de exibição como um bitmap, usando a propriedade `cacheAsBitmap`. Ao fazer isso, o Flash Player e o AIR não precisam redesenhar o conteúdo inteiro do objeto de exibição sempre que for movido e, assim, podem usar o bitmap em cache para renderizar a parte necessária diretamente na tela. Para obter detalhes, consulte "Armazenamento em cache de objetos de exibição" na página 302.

Manipulação do tamanho e dimensionamento de objetos

É possível medir e manipular o tamanho de um objeto de exibição de duas formas, usando as propriedades de dimensão (`width` e `height`) ou as propriedades de escala (`scaleX` e `scaleY`).

Todo objeto de exibição tem uma propriedade `width` e uma propriedade `height`, que são definidas inicialmente como o tamanho do objeto em pixels. É possível ler os valores dessas propriedades para medir o tamanho do objeto de exibição. Você também pode especificar novos valores para alterar o tamanho do objeto, do seguinte modo:

```
// Resize a display object.
square.width = 420;
square.height = 420;

// Determine the radius of a circle display object.
var radius:Number = circle.width / 2;
```

Se você alterar a `altura` ou a `largura` de um objeto de exibição, esse objeto será dimensionado, ampliando ou reduzindo seu conteúdo para ajustá-lo na nova área. Caso o objeto de exibição tenha apenas formas vetoriais, essas formas serão redesenhadas na nova escala, sem prejudicar a qualidade. Todos os elementos gráficos de bitmap do objeto de exibição serão dimensionados, não redesenhados. Assim, por exemplo, uma foto digital cuja largura e altura são aumentadas além das dimensões reais das informações de pixels na imagem será pixelizada, ficando irregular.

Quando as propriedades `width` ou `height` de um objeto de exibição são alteradas, o Flash Player e o AIR também atualizam as propriedades `scaleX` e `scaleY`.

Nota: *Os objetos TextField são uma exceção a este comportamento de dimensionamento. Os campos de texto devem realizar o dimensionamento automático para suportar textos com quebra e tamanhos de fonte variados, fazendo com que os valores scaleX ou scaleY sejam redefinidos para 1 após o redimensionamento. No entanto, se ajustar os valores scaleX ou scaleY de um objeto TextField, os valores de largura e altura serão alterados para comportar os valores de dimensionamento fornecidos.*

Essas propriedades representam o tamanho relativo do objeto de exibição em comparação com o tamanho original. As propriedades `scaleX` e `scaleY` usam valores fracionários (decimais) para representar porcentagens. Por exemplo, se a `largura` de um objeto de exibição for alterada para ter metade do tamanho original, a propriedade `scaleX` terá o valor `.5`, que indica 50%. Se a altura for dobrada, a propriedade `scaleY` terá o valor `2`, que indica 200%.

```
// circle is a display object whose width and height are 150 pixels.
// At original size, scaleX and scaleY are 1 (100%).
trace(circle.scaleX); // output: 1
trace(circle.scaleY); // output: 1

// When you change the width and height properties,
// Flash Player changes the scaleX and scaleY properties accordingly.
circle.width = 100;
circle.height = 75;
trace(circle.scaleX); // output: 0.6622516556291391
trace(circle.scaleY); // output: 0.4966887417218543
```

As alterações de tamanho não são proporcionais. Em outras palavras, e você alterar a `altura` de um quadrado, mas não a `largura`, suas proporções não serão mais as mesmas e o resultado será um retângulo, em vez de um quadrado. Se desejar fazer alterações relacionadas ao tamanho de um objeto de exibição, defina os valores das propriedades `scaleX` e `scaleY` para redimensionar o objeto, em vez de definir as propriedades `width` ou `height`. Por exemplo, este código altera a `largura` do objeto de exibição chamado `square` e, em seguida, altera a escala vertical (`scaleY`) para corresponder à escala horizontal, mantendo o tamanho proporcional do quadrado.

```
// Change the width directly.
square.width = 150;

// Change the vertical scale to match the horizontal scale,
// to keep the size proportional.
square.scaleY = square.scaleX;
```

Controle da distorção durante o dimensionamento

Normalmente, quando um objeto de exibição é dimensionado (por exemplo, ampliado na horizontal), a distorção resultante é distribuída uniformemente no objeto, para que cada parte seja ampliada do mesmo modo. Para elementos gráficos e de design, isso é provavelmente o que se espera. No entanto, às vezes é melhor ter controle sobre as partes do objeto de exibição que são ampliadas e as partes que permanecem inalteradas. Um exemplo comum disso é um botão retangular com cantos arredondados. Com o dimensionamento normal, os cantos do botão são ampliados, alterando o raio do canto à medida que o botão é redimensionado.

No entanto, nesse caso seria melhor ter controle sobre o dimensionamento — conseguir designar algumas áreas que devem ser dimensionadas (lados retos e o centro) e outras que não devem (os cantos) — para que o dimensionamento ocorra sem nenhuma distorção visível.

Você pode usar o dimensionamento de 9 fatias (Escala 9) para criar objetos de exibição cujo dimensionamento pode ser controlado. Com o dimensionamento de 9 fatias, o objeto de exibição é dividido em 9 retângulos separados (uma grade 3 x 3, como a grade do jogo da velha). Os retângulos não são necessariamente do mesmo tamanho; você pode desenhar onde as linhas da grade são colocadas. Todo conteúdo que estiver nos retângulos dos quatro cantos (como os cantos arredondados de um botão) não será ampliado ou reduzido quando o objeto de exibição for dimensionado. Os retângulos centrais superior e inferior serão dimensionados na horizontal, não na vertical, enquanto os retângulos centrais esquerdo e direito serão dimensionados na vertical, não na horizontal. O retângulo central será dimensionado tanto na horizontal quanto na vertical.

Com isso em mente, se estiver criando um objeto de exibição e desejar que um determinado conteúdo nunca seja dimensionado, verifique se as linhas divisórias da grade de dimensionamento de 9 fatias estão colocadas de modo que o conteúdo fique em um dos retângulos do canto.

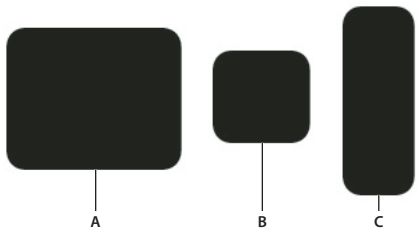
No ActionScript, definir um valor para a propriedade `scale9Grid` de um objeto de exibição ativa o dimensionamento de 9 fatias do objeto e define o tamanho dos retângulos na grade de escala 9 do objeto. Use uma ocorrência da classe Rectangle como valor da propriedade `scale9Grid`, do seguinte modo:

```
myButton.scale9Grid = new Rectangle(32, 27, 71, 64);
```

Os quatro parâmetros do construtor de retângulo são a coordenada x, a coordenada y, a largura e a altura. Neste exemplo, o canto superior esquerdo do retângulo é colocado no ponto x: 32, y: 27 no objeto de exibição chamado `myButton`. O retângulo tem 71 pixels de largura e 64 pixels de altura (de modo que a borda direita está na coordenada 103 do eixo x e a borda inferior está na coordenada 92 do eixo y no objeto de exibição).

A área real contida na região definida pela ocorrência de Rectangle representa o retângulo central da grade de escala 9. Os outros retângulos são calculados pelo Flash Player e AIR, estendendo os lados da ocorrência de Rectangle, conforme mostrado aqui:

Nesse caso, à medida que o botão é dimensionado para cima ou para baixo, os cantos arredondados não são ampliados ou reduzidos, mas as outras áreas se ajustam de acordo com o dimensionamento.



A. *myButton.width = 131;myButton.height = 106;* ***B.*** *myButton.width = 73;myButton.height = 69;* ***C.*** *myButton.width = 54;myButton.height = 141;*

Armazenamento em cache de objetos de exibição

Sempre que estiver aumentando o tamanho no Flash, seja para criar um aplicativo ou animações complexas com script, você precisa levar em conta o desempenho e a otimização. O Flash Player e o AIR não otimizam conteúdo que permanece estático (como uma ocorrência de Shape retangular). Desse modo, quando você altera a posição do retângulo, o Flash Player ou o AIR redesenha a ocorrência de Shape inteira.

Você pode armazenar os objetos de exibição especificados em cache para melhorar o desempenho do arquivo SWF. O objeto de exibição é uma *superfície*, basicamente uma versão de bitmap dos dados vetoriais da ocorrência, que são os dados que não devem mudar muito durante o fluxo do arquivo SWF. Portanto, as ocorrências com o cache ativado não são redesenhadas continuamente à medida que o arquivo SWF é reproduzido, o que aumenta a velocidade da renderização.

Nota: *Você pode atualizar os dados vetoriais e, quando isso é feito, a superfície é recriada. Assim, os dados vetoriais armazenados em cache na superfície não precisam permanecer iguais para todo o arquivo SWF.*

Se você definir a propriedade `cacheAsBitmap` do objeto de exibição como `true`, o cache do objeto de exibição será uma representação em bitmap do próprio objeto. O Flash Player ou o AIR criam um objeto de superfície para a ocorrência, que é um bitmap armazenado em cache em vez de dados vetoriais. Se os limites do objeto de exibição forem alterados, a superfície será recriada em vez de ser redimensionada. As superfícies podem ser aninhadas com outras superfícies. A superfície filho copia seu bitmap na superfície pai. Para obter mais informações, consulte "Ativação do armazenamento em cache de bitmaps" na página 304.

A propriedade `opaqueBackground` e a propriedade `scrollRect` da classe DisplayObject estão relacionadas ao armazenamento em cache de bitmaps realizado com a propriedade `cacheAsBitmap`. Embora essas três propriedades sejam independentes entre si, as propriedades `opaqueBackground` e `scrollRect` funcionam melhor quando um objeto é armazenado em cache como um bitmap; você verá a melhora de desempenho das propriedades `opaqueBackground` e `scrollRect` somente quando `cacheAsBitmap` for definido como `true`. Para obter mais informações sobre como percorrer o conteúdo do objeto de exibição, consulte "Visão panorâmica e rolagem de objetos de exibição" na página 298. Para obter mais informações sobre como configurar um plano de fundo opaco, consulte "Definição de uma cor de fundo opaca" na página 304.

Para obter informações sobre o mascaramento do canal alfa, que requer a definição da propriedade `cacheAsBitmap` como `true`, consulte "Mascaramento de objetos de exibição" na página 309.

Quando ativar o armazenamento em cache

A ativação do cache para um objeto de exibição cria uma superfície, o que tem diversas vantagens, como ajudar na renderização rápida de animações vetoriais complexas. Existem diversos cenários nos quais é necessário ativar o cache. Você talvez pense que sempre que o cache é ativado, o desempenho dos arquivos SWF melhora; no entanto, existem situações nas quais a ativação do cache não melhora o desempenho ou pode inclusive piorá-lo. Esta seção descreve cenários nos quais o cache deve ser usado e quando objetos de exibição regulares devem ser usados.

O desempenho global de dados em cache depende da complexidade dos dados vetoriais das ocorrências, quanto foram mudados os dados e se foi ou não definida a propriedade `opaqueBackground`. Se você estiver mudando regiões pequenas, a diferença entre o uso de superfície e o uso de dados vetoriais pode ser desprezível. Teste o seu trabalho das duas formas, antes de implantar o aplicativo.

Quando usar o armazenamento em cache de bitmaps

A seguir, alguns casos comuns nos quais podem ser vistos benefícios significativos quando se ativa o armazenamento em cache de bitmaps.

- Imagem de fundo complexa: um aplicativo que contém uma imagem de fundo complexa e detalhada dos dados vetoriais (talvez uma imagem na qual o comando Traçar bitmap tenha sido aplicado ou a arte final criada no Adobe Illustrator®). Você pode animar caracteres no plano de fundo, o que deixa a animação mais lenta porque o plano de fundo precisa gerar regularmente os dados vetoriais mais uma vez. Para melhorar o desempenho, você pode definir a propriedade `opaqueBackground` do objeto de exibição de fundo como `true`. O fundo é renderizado como um bitmap e pode se redesenhado rapidamente, de modo que a execução da animação seja muito mais rápida.
- Rolagem de campo de texto: um aplicativo que exibe uma grande quantidade de texto na rolagem do campo de texto. Você pode colocar o campo de texto em um objeto de exibição definido como rolável com limites de rolagem (a propriedade `scrollRect`). Isso agiliza a rolagem de pixel para a ocorrência especificada. Quando o usuário rola a ocorrência do objeto de exibição, o Flash Player ou o AIR move os pixels rolados para cima e gera a região recém-exposta, em vez de gerar novamente todo o campo de texto.
- Sistema de janelas: um aplicativo com um sistema complexo de janelas sobrepostas. Cada janela pode ser aberta ou fechada (por exemplo, as janelas de navegador da web). Se você marcar cada janela como uma superfície (definindo a propriedade `cacheAsBitmap` como `true`), cada janela é isolada e colocada em cache. Os usuários podem arrastar as janelas de modo que se sobreponham, e as janelas não precisam gerar novamente o conteúdo vetorial.
- Mascaramento do canal alfa: ao usar o mascaramento do canal alfa, você deve definir a propriedade `cacheAsBitmap` como `true`. Para obter mais informações, consulte "Mascaramento de objetos de exibição" na página 309.

A ativação do cache de bitmaps em todos esses cenários melhora a resposta e a interatividade do aplicativo, otimizando os gráficos vetoriais.

Além disso, sempre que um filtro é aplicado em um objeto de exibição, `cacheAsBitmap` é definido automaticamente como `true`, mesmo que esteja explicitamente definido como `false`. Se todos os filtros forem desativados do objeto de exibição, a propriedade `cacheAsBitmap` retornará ao valor definido pela última vez.

Quando evitar o uso do armazenamento em cache de bitmaps

O mau uso desse recurso pode afetar negativamente o arquivo SWF. Ao usar o cache de bitmaps, lembre-se das seguintes orientações:

- Não use em exagero superfícies (objetos de exibição com o cache ativado). Cada superfície usa mais memória do que um objeto de exibição normal, o que indica que você deve ativar as superfícies apenas quando precisar melhorar o desempenho da renderização.

  Um bitmap em cache pode usar significativamente mais memória do que um objeto de exibição normal. Por exemplo, se uma ocorrência de Sprite no palco tem 250 pixels por 250 pixels de tamanho, pode usar 250 KB em vez de 1 KB em cache quando for uma ocorrência normal de Sprite (não armazenada em cache).

- Evite o zoom em superfícies em cache. Se usar exageradamente cache de bitmaps, é consumida uma grande quantidade de memória (veja observação anterior) se fizer o zoom do conteúdo.
- Use superfícies para ocorrências de objeto de exibição em grande parte estáticas (sem animação). Você pode arrastar ou mover a ocorrência, mas o conteúdo da ocorrência não deve ser animado ou mudado muito. A animação ou alteração do conteúdo tem maior probabilidade de acontecer com uma ocorrência de MovieClip que contém animação ou uma ocorrência de Video. Por exemplo, se você girar ou transformar uma ocorrência, ela muda entre a superfície e os dados vetoriais, o que é difícil de processar e afeta negativamente o arquivo SWF.
- Se misturar superfícies com dados vetoriais, aumenta a quantidade de processamento a ser feita pelo Flash Player e pelo AIR (e algumas vezes o computador). Agrupe as superfícies o máximo possível, por exemplo, quando criar aplicativos em janelas.

Ativação do armazenamento em cache de bitmaps

Para ativar o armazenamento em cache de bitmaps para um objeto de exibição, defina a propriedade `cacheAsBitmap` como `true`:

```
mySprite.cacheAsBitmap = true;
```

Depois de definir a propriedade `cacheAsBitmap` como `true`, você deve perceber que o objeto de exibição realiza o encaixe de pixels automaticamente em coordenadas inteiras. Ao testar o arquivo SWF, você verá que qualquer animação executada em uma imagem vetorial complexa é processada muito mais rápido.

Uma superfície (bitmap em cache) não será criada, mesmo que `cacheAsBitmap` esteja definido como `true`, se ocorrer uma ou mais das seguintes situações:

- O bitmap é muito maior do que 2880 pixels de altura ou largura.
- O bitmap não consegue ser alocado (produzindo erro de falta de memória).

Definição de uma cor de fundo opaca

Você pode definir um plano de fundo opaco para um objeto de exibição. Por exemplo, quando o SWF tem um plano de fundo que contém elementos vetoriais complexos, você pode definir a propriedade `opaqueBackground` como uma cor especificada (normalmente a mesma cor do palco). A cor é especificada como um número (em geral, um valor de cor hexadecimal). O plano de fundo é tratado como um bitmap, o que ajuda a otimizar o desempenho.

Quando você define `cacheAsBitmap` como `true` e também define a propriedade `opaqueBackground` como uma cor especificada, a propriedade `opaqueBackground` permite que o bitmap interno seja opaco e renderizado mais rapidamente. Se `cacheAsBitmap` não for definido como `true`, a propriedade `opaqueBackground` adicionará uma forma vetorial quadrada opaca ao plano de fundo do objeto de exibição. Isso não cria um bitmap automaticamente.

O exemplo a seguir mostra como definir o plano de fundo de um objeto de exibição para otimizar o desempenho:

```
myShape.cacheAsBitmap = true;
myShape.opaqueBackground = 0xFF0000;
```

Nesse caso, a cor de fundo da forma chamada `myShape` é definida como vermelha (`0xFF0000`). Supondo que a ocorrência de Shape contém um desenho de um triângulo verde, em um palco com fundo branco, seria mostrado um triângulo verde com vermelho no espaço vazio da caixa delimitadora da ocorrência de Shape (o retângulo que envolve a forma por completo).

Obviamente, isso faria mais sentido se fosse usado com um palco com fundo vermelho sólido. Em outro fundo colorido, essa cor seria especificada. Por exemplo, em um SWF com fundo branco, a propriedade `opaqueBackground` provavelmente seria definida como `0xFFFFFF`, ou branco puro.

Aplicação de modos de mesclagem

Os modos de mesclagem envolvem a combinação das cores de uma imagem (a imagem base) com as cores de outra imagem (a imagem de mesclagem) para produzir uma terceira imagem; a imagem resultante é aquela realmente exibida na tela. Cada valor de pixel em uma imagem é processado com o valor de pixel correspondente da outra imagem para produzir um valor de pixel para a mesma posição no resultado.

Todos os objetos de exibição têm uma propriedade `blendMode` que pode ser definida como um dos seguintes modos de mesclagem. Esses modos são constantes definidas na classe BlendMode. Se preferir, você pode usar os valores de String (entre parênteses) que são os reais valores das constantes.

- `BlendMode.ADD` (`"add"`): normalmente usado para criar um efeito animado de dissolução de iluminação entre duas imagens.
- `BlendMode.ALPHA` (`"alpha"`): normalmente usado para aplicar a transparência do primeiro plano no plano de fundo.
- `BlendMode.DARKEN` (`"darken"`): normalmente usado para sobrepor tipos.
- `BlendMode.DIFFERENCE` (`"difference"`): normalmente usado para criar cores mais vibrantes.
- `BlendMode.ERASE` (`"erase"`): normalmente usado para cortar (apagar) parte do plano de fundo usando o alfa do primeiro plano.
- `BlendMode.HARDLIGHT` (`"hardlight"`): normalmente usado para criar efeitos de sombra.
- `BlendMode.INVERT` (`"invert"`): usado para inverter o plano de fundo.
- `BlendMode.LAYER` (`"layer"`): usado para forçar a criação de um buffer temporário para pré-composição de um objeto de exibição específico.
- `BlendMode.LIGHTEN` (`"lighten"`): normalmente usado para sobrepor tipos.
- `BlendMode.MULTIPLY` (`"multiply"`): normalmente usado para criar sombras e efeitos de profundidade.

- `BlendMode.NORMAL` (`"normal"`): usado para especificar que os valores de pixel da imagem de mesclagem substituem os da imagem base.
- `BlendMode.OVERLAY` (`"overlay"`): normalmente usado para criar efeitos de sombra.
- `BlendMode.SCREEN` (`"screen"`): normalmente usado para criar realces e manchas de luz.
- `BlendMode.SHADER` (`"shader"`): usado para especificar que um sombreador Pixel Bender é usado para criar um efeito de mesclagem personalizado. Para obter mais informações sobre como usar sombreadores, consulte "Trabalho com sombreadores Pixel Bender" na página 386.
- `BlendMode.SUBTRACT` (`"subtract"`): normalmente usado para criar um efeito animado de dissolução de escurecimento entre duas imagens.

Ajuste das cores de DisplayObject

Você pode usar os métodos da classe ColorTransform (flash.geom.ColorTransform) para ajustar a cor de um objeto de exibição. Cada objeto de exibição tem uma propriedade `transform`, que é uma ocorrência da classe Transform, e contém informações sobre várias transformações que são aplicadas no objeto de exibição (como rotação, alterações na escala ou posição e assim por diante). Além de informações sobre transformações geométricas, a classe Transform também inclui uma propriedade `colorTransform`, que é uma ocorrência da classe ColorTransform e fornece acesso para fazer ajustes de cor no objeto de exibição. Para acessar as informações de transformação de cor de um objeto de exibição, você pode usar um código como esse:

```
var colorInfo:ColorTransform = myDisplayObject.transform.colorTransform;
```

Depois de criar uma ocorrência de ColorTransform, você pode ler os valores de propriedade para descobrir quais transformações de cor já foram aplicadas ou definir esses valores para alterar cores no objeto de exibição. Para atualizar o objeto de exibição depois de fazer alterações, atribua novamente a ocorrência de ColorTransform à propriedade `transform.colorTransform`.

```
var colorInfo:ColorTransform = my DisplayObject.transform.colorTransform;

// Make some color transformations here.

// Commit the change.
myDisplayObject.transform.colorTransform = colorInfo;
```

Definição de valores de cor com código

A propriedade `color` da classe ColorTransform pode ser usada para atribuir um valor de cor RGB (vermelho, verde e azul) específico ao objeto de exibição. O exemplo a seguir usa a propriedade `color` para alterar a cor do objeto de exibição chamado `square` como azul quando o usuário clicar no botão `blueBtn`:

```
// square is a display object on the Stage.
// blueBtn, redBtn, greenBtn, and blackBtn are buttons on the Stage.

import flash.events.MouseEvent;
import flash.geom.ColorTransform;

// Get access to the ColorTransform instance associated with square.
var colorInfo:ColorTransform = square.transform.colorTransform;

// This function is called when blueBtn is clicked.
function makeBlue(event:MouseEvent):void
{
    // Set the color of the ColorTransform object.
    colorInfo.color = 0x003399;
    // apply the change to the display object
    square.transform.colorTransform = colorInfo;
}

blueBtn.addEventListener(MouseEvent.CLICK, makeBlue);
```

Observe que, ao alterar a cor de um objeto de exibição usando a propriedade `color`, a cor do objeto inteiro é alterada, independentemente de o objeto ter várias cores anteriormente. Por exemplo, se houver um objeto de exibição que contém um círculo verde com texto preto na parte superior, definir a propriedade `color` da ocorrência de ColorTransform associada desse objeto como uma sombra vermelha faz com que o objeto inteiro, círculo e texto, fique vermelho (de modo que o texto não será mais diferenciado do restante do objeto).

Alteração de efeitos de brilho e cor com código

Digamos que você tenha um objeto de exibição com várias cores (por exemplo, uma foto digital) e não queira colorir o objeto inteiro novamente; você quer apenas ajustar a cor de um objeto de exibição com base nas cores existentes. Nesse cenário, a classe ColorTransform inclui uma série de propriedades de multiplicador e deslocamento que podem ser usadas para fazer esse tipo de ajuste. As propriedades de multiplicador, chamadas `redMultiplier`, `greenMultiplier`, `blueMultiplier` e `alphaMultiplier`, funcionam como filtros fotográficos coloridos (ou óculos de sol coloridos), intensificando ou ofuscando algumas cores no objeto de exibição. As propriedades de deslocamento (`redOffset`, `greenOffset`, `blueOffset` e `alphaOffset`) podem ser usadas para aumentar a quantidade de uma determinada cor no objeto ou para especificar o valor mínimo que uma cor específica pode ter.

Essas propriedades de multiplicador e deslocamento são idênticas às configurações de cor avançadas que estão disponíveis para símbolos de clipe de filme na ferramenta de criação do Flash quando você escolhe Avançado no menu pop-up Cor no Inspetor de propriedades.

O código a seguir carrega uma imagem JPEG e aplica uma transformação de cor nela, ajustando os canais vermelho e verde à medida que o ponteiro do mouse se move ao longo dos eixos x e y. Nesse caso, como nenhum valor de deslocamento foi especificado, o valor de cada canal de cor exibido na tela será uma porcentagem do valor de cor original na imagem - a maior parte de vermelho ou verde exibida em um pixel é a quantidade original de vermelho ou verde nesse pixel.

```
import flash.display.Loader;
import flash.events.MouseEvent;
import flash.geom.Transform;
import flash.geom.ColorTransform;
import flash.net.URLRequest;

// Load an image onto the Stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image1.jpg");
loader.load(url);
this.addChild(loader);

// This function is called when the mouse moves over the loaded image.
function adjustColor(event:MouseEvent):void
{
    // Access the ColorTransform object for the Loader (containing the image)
    var colorTransformer:ColorTransform = loader.transform.colorTransform;

    // Set the red and green multipliers according to the mouse position.
    // The red value ranges from 0% (no red) when the cursor is at the left
    // to 100% red (normal image appearance) when the cursor is at the right.
    // The same applies to the green channel, except it's controlled by the
    // position of the mouse in the y axis.
    colorTransformer.redMultiplier = (loader.mouseX / loader.width) * 1;
    colorTransformer.greenMultiplier = (loader.mouseY / loader.height) * 1;

    // Apply the changes to the display object.
    loader.transform.colorTransform = colorTransformer;
}

loader.addEventListener(MouseEvent.MOUSE_MOVE, adjustColor);
```

Rotação de objetos

Os objetos de exibição podem ser girados com a propriedade `rotation`. Você pode ler esse valor para saber se um objeto foi girado ou, para girar o objeto, defina essa propriedade como um número (em graus) que representa o valor de rotação a ser aplicado no objeto. Por exemplo, essa linha do código gira o objeto chamado `square` 45 graus (1/8 de uma revolução completa):

```
square.rotation = 45;
```

Se preferir, gire o objeto de exibição usando uma matriz de transformação, conforme descrito em "Trabalho com geometria" na página 342.

Desaparecimento de objetos

Você pode controlar a transparência de um objeto de exibição para deixá-lo parcial ou completamente transparente ou alterar a transparência para realçar ou ofuscar o objeto. A propriedade `alpha` da classe DisplayObject define a transparência (ou, mais precisamente, a opacidade) de um objeto de exibição. A propriedade `alpha` pode ser definida como qualquer valor entre 0 e 1, onde 0 é completamente transparente e 1 é completamente opaco. Por exemplo, essas linhas de código deixam o objeto chamado `myBall` parcialmente transparente (50%) quando é clicado com o mouse:

```
function fadeBall(event:MouseEvent):void
{
    myBall.alpha = .5;
}
myBall.addEventListener(MouseEvent.CLICK, fadeBall);
```

Você também pode alterar a transparência de um objeto de exibição usando os ajustes de cor disponíveis pela classe ColorTransform. Para obter mais informações, consulte "Ajuste das cores de DisplayObject" na página 306.

Mascaramento de objetos de exibição

Você pode usar um objeto de exibição como uma máscara para criar um orifício por meio do qual o conteúdo de outro objeto de exibição é visualizado.

Definição de uma máscara

Para indicar que um objeto de exibição será a máscara de outro objeto, defina o objeto de máscara como a propriedade `mask` do objeto de exibição a ser mascarado:

```
// Make the object maskSprite be a mask for the object mySprite.
mySprite.mask = maskSprite;
```

O objeto de exibição mascarado é revelado em todas as áreas opacas (não transparentes) do objeto de exibição que age como a máscara. Por exemplo, o código a seguir cria uma ocorrência de Shape que contém um quadrado vermelho de 100 x 100 pixels e uma ocorrência de Sprite que contém um círculo azul com raio de 25 pixels. Assim que é clicado, o círculo é definido como a máscara do quadrado para que a única parte exibida do quadrado seja a parte coberta pela parte sólida do círculo. Em outras palavras, somente um círculo vermelho ficará visível.

```
// This code assumes it's being run within a display object container
// such as a MovieClip or Sprite instance.

import flash.display.Shape;

// Draw a square and add it to the display list.
var square:Shape = new Shape();
square.graphics.lineStyle(1, 0x000000);
square.graphics.beginFill(0xff0000);
square.graphics.drawRect(0, 0, 100, 100);
square.graphics.endFill();
this.addChild(square);

// Draw a circle and add it to the display list.
var circle:Sprite = new Sprite();
circle.graphics.lineStyle(1, 0x000000);
circle.graphics.beginFill(0x0000ff);
circle.graphics.drawCircle(25, 25, 25);
circle.graphics.endFill();
this.addChild(circle);

function maskSquare(event:MouseEvent):void
{
    square.mask = circle;
    circle.removeEventListener(MouseEvent.CLICK, maskSquare);
}

circle.addEventListener(MouseEvent.CLICK, maskSquare);
```

O objeto de exibição que atua como máscara pode ser arrastado, animado, redimensionado dinamicamente e pode usar formas separadas em uma única máscara. O objeto de exibição de máscara não precisa ser necessariamente adicionado à lista de exibição. No entanto, se desejar que o objeto de máscara seja dimensionado quando o palco for dimensionado ou se desejar permitir a interação do usuário com a máscara (como as operações de arrastar e redimensionar controladas pelo usuário), o objeto de máscara deve ser adicionado à lista de exibição. O índice z real (ordem da frente para trás) dos objetos de exibição não importam, desde que o objeto de máscara seja adicionado à lista de exibição. O objeto de máscara será exibido na tela apenas como uma máscara. Se o objeto de máscara for uma ocorrência de MovieClip com vários quadros, todos os quadros da linha do tempo desse objeto serão reproduzidos, do mesmo modo como se não estivesse agindo como máscara. Para remover uma máscara, defina a propriedade `mask` como `null`:

```
// remove the mask from mySprite
mySprite.mask = null;
```

Você não pode usar uma máscara para mascarar outra. A propriedade `alpha` de um objeto de exibição de máscara não pode ser definida. Somente os preenchimentos são usados em um objeto de exibição usado como máscara; os traçados são ignorados.

Sobre o mascaramento de fontes de dispositivo

Você pode usar um objeto de exibição para mascarar o texto que está definido em uma fonte de dispositivo. Quando um objeto de exibição é usado para mascarar o texto definido em uma fonte de dispositivo, a caixa delimitadora retangular da máscara é usada como a forma de mascaramento. Isso significa que, se você criar uma máscara não retangular de objeto de exibição para o texto da fonte de dispositivo, a máscara exibida no arquivo SWF terá a forma da caixa delimitadora retangular da máscara, não a forma da máscara propriamente dita.

Mascaramento do canal alfa

O mascaramento do canal alfa é permitido se a máscara e os objetos de exibição mascarados usarem o cache de bitmaps, conforme mostrado aqui:

```
// maskShape is a Shape instance which includes a gradient fill.
mySprite.cacheAsBitmap = true;
maskShape.cacheAsBitmap = true;
mySprite.mask = maskShape;
```

Por exemplo, o mascaramento do canal alfa usa um filtro no objeto de máscara diferente do filtro que é aplicado no objeto de exibição mascarado.

No exemplo a seguir, um arquivo de imagem externo é carregado no palco. Essa imagem (ou, mais precisamente, a ocorrência de Loader na qual é carregada) será o objeto de exibição mascarado. Um elemento oval de gradiente (centro preto sólido que fica transparente nas bordas) é desenhado na imagem; esta será a máscara alfa. Os dois objetos de exibição têm o cache de bitmaps ativado. O elemento oval é definido como uma máscara para a imagem e pode ser arrastado.

```
// This code assumes it's being run within a display object container
// such as a MovieClip or Sprite instance.

import flash.display.GradientType;
import flash.display.Loader;
import flash.display.Sprite;
import flash.geom.Matrix;
import flash.net.URLRequest;

// Load an image and add it to the display list.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image1.jpg");
loader.load(url);
this.addChild(loader);

// Create a Sprite.
var oval:Sprite = new Sprite();
// Draw a gradient oval.
var colors:Array = [0x000000, 0x000000];
var alphas:Array = [1, 0];
var ratios:Array = [0, 255];
var matrix:Matrix = new Matrix();
matrix.createGradientBox(200, 100, 0, -100, -50);
oval.graphics.beginGradientFill(GradientType.RADIAL,
                                colors,
                                alphas,
                                ratios,
                                matrix);
oval.graphics.drawEllipse(-100, -50, 200, 100);
oval.graphics.endFill();
// add the Sprite to the display list
this.addChild(oval);

// Set cacheAsBitmap = true for both display objects.
loader.cacheAsBitmap = true;
oval.cacheAsBitmap = true;
// Set the oval as the mask for the loader (and its child, the loaded image)
loader.mask = oval;

// Make the oval draggable.
oval.startDrag(true);
```

Animação de objetos

Animação é o processo de fazer algo se movimentar ou, também, de fazer algo mudar com o passar do tempo. A animação com script é uma parte fundamental dos jogos de vídeo e normalmente é usada para adicionar dicas úteis de interação a outros aplicativos.

A idéia básica por trás da animação por script é de que uma alteração deve ocorrer e essa alteração precisa ser dividida em incrementos com o passar do tempo. É fácil fazer algo se repetir no ActionScript, usando uma instrução de consulta comum. No entanto, uma consulta será executada em todas as iterações antes da atualização da exibição. Para criar animação com script, você precisa gravar um ActionScript que execute alguma ação repetidas vezes com o passar do tempo e também atualize a tela sempre que essa ação for executada.

Por exemplo, imagine que você quer criar uma animação simples, como uma bola que percorre a tela. O ActionScript inclui um mecanismo fácil que permite acompanhar a passagem do tempo e atualizar a tela conforme necessário, ou seja, você pode gravar o código que move a bola um pouco por vez até atingir o destino. Após cada movimentação, a tela é atualizada e o usuário pode visualizar o movimento no palco.

Do ponto de vista prático, faz sentido sincronizar a animação com script com a taxa de quadros do arquivo SWF (em outras palavras, fazer uma alteração de animação sempre que um novo quadro for exibido), pois tal procedimento define a freqüência de atualização de tela do Flash Player. Cada objeto de exibição tem um evento `enterFrame` que é enviado de acordo com a taxa de quadros do arquivo SWF - um evento por quadro. A maioria dos desenvolvedores que cria animações com script usa o evento `enterFrame` para criar ações que se repetem com o passar do tempo. Você pode gravar um código que ouve o evento `enterFrame`, movendo a bola animada um pouco em cada quadro e, à medida que a tela é atualizada (cada quadro), a bola seria redesenhada em seu novo local, criando o movimento.

Nota: *Uma ação que se repete com o passar do tempo também pode ser criada com a classe Timer. Uma ocorrência de Timer aciona uma notificação de evento sempre que um período especificado passa. Você poderia gravar um código que executa a animação manipulando os eventos de tempo da classe Timer, definindo um intervalo de tempo pequeno (alguma fração de um segundo). Para obter mais informações sobre como usar a classe Timer, consulte "Controle de intervalos de tempo" na página 137.*

No exemplo a seguir, uma ocorrência circular de Sprite, chamada `circle`, é criada no palco. Quando o usuário clica no círculo, uma seqüência de animação com script é iniciada, fazendo com que o `círculo` desapareça (a propriedade `alpha` é diminuída) até ficar completamente transparente:

```
import flash.display.Sprite;
import flash.events.Event;
import flash.events.MouseEvent;

// draw a circle and add it to the display list
var circle:Sprite = new Sprite();
circle.graphics.beginFill(0x990000);
circle.graphics.drawCircle(50, 50, 50);
circle.graphics.endFill();
addChild(circle);

// When this animation starts, this function is called every frame.
// The change made by this function (updated to the screen every
// frame) is what causes the animation to occur.
function fadeCircle(event:Event):void
{
    circle.alpha -= .05;

    if (circle.alpha <= 0)
    {
        circle.removeEventListener(Event.ENTER_FRAME, fadeCircle);
    }
}

function startAnimation(event:MouseEvent):void
{
    circle.addEventListener(Event.ENTER_FRAME, fadeCircle);
}

circle.addEventListener(MouseEvent.CLICK, startAnimation);
```

Quando o usuário clica no círculo, a função `fadeCircle()` é inscrita como ouvinte do evento `enterFrame`, indicando que começará a ser chamada uma vez por quadro. Essa função desbota o `círculo` alterando sua propriedade `alpha` para que, uma vez por quadro, o `alfa` do círculo diminua 0,05 (5%) e a tela seja atualizada. Eventualmente, quando o valor de `alpha` é 0 (`círculo` completamente transparente), a função `fadeCircle()` é removida como ouvinte de eventos, encerrando a animação.

O mesmo código poderia ser usado, por exemplo, para criar um movimento animado em vez de desbotar o objeto. Substituindo uma propriedade diferente para `alpha` na função que é um ouvinte de eventos `enterFrame`, essa propriedade será animada. Por exemplo, alterar esta linha

```
circle.alpha -= .05;
```

para este código

```
circle.x += 5;
```

animará a propriedade `x`, fazendo com que o círculo se mova para a direita no palco. A condição que encerra a animação poderia ser alterada para finalizar a animação (isto é, cancelar a inscrição do ouvinte `enterFrame`) quando a coordenada x desejada for atingida.

Carregamento dinâmico do conteúdo da exibição

Você pode carregar qualquer uma dos seguintes ativos de exibição externos em um aplicativo ActionScript 3.0:

- Um arquivo SWF criado no ActionScript 3.0 - Esse arquivo pode ser uma classe Sprite, MovieClip ou qualquer classe que estende Sprite.
- Um arquivo de imagem - Isso inclui arquivos JPG, PNG e GIF.
- Um arquivo SWF AVM1 - Arquivo SWF gravado no ActionScript 1.0 ou 2.0.

Carregue esses ativos usando a classe Loader.

Carregamento de objetos de exibição

Os objetos Loader são usados para carregar arquivos SWF e de imagem em um aplicativo. A classe Loader é uma subclasse de DisplayObjectContainer. Um objeto Loader pode conter apenas um objeto de exibição filho na lista de exibição, o objeto que representa o arquivo SWF ou de imagem carregado. Quando você adiciona um objeto Loader à lista de exibição, como no código a seguir, também pode adicionar o objeto filho carregado à lista de exibição após o carregamento:

```
var pictLdr:Loader = new Loader();
var pictURL:String = "banana.jpg"
var pictURLReq:URLRequest = new URLRequest(pictURL);
pictLdr.load(pictURLReq);
this.addChild(pictLdr);
```

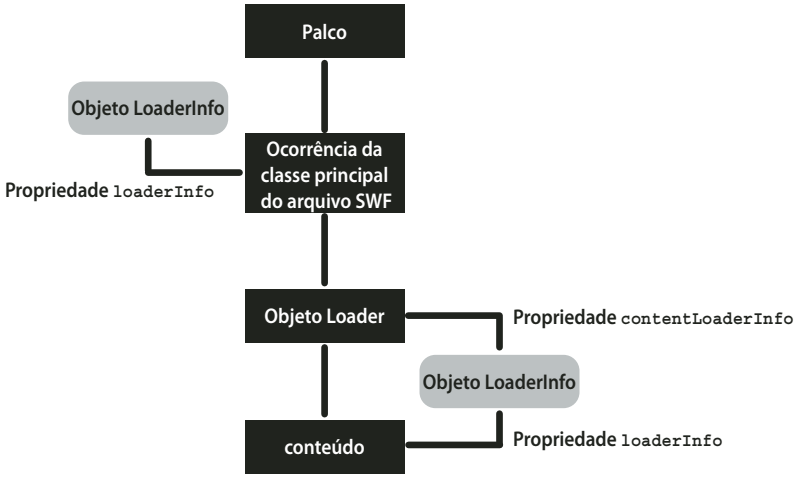
Assim que o arquivo SWF ou imagem é carregado, você pode mover o objeto de exibição carregado para outro contêiner, como o objeto `container` DisplayObjectContainer neste exemplo:

```
import flash.display.*;
import flash.net.URLRequest;
import flash.events.Event;
var container:Sprite = new Sprite();
addChild(container);
var pictLdr:Loader = new Loader();
var pictURL:String = "banana.jpg"
var pictURLReq:URLRequest = new URLRequest(pictURL);
pictLdr.load(pictURLReq);
pictLdr.contentLoaderInfo.addEventListener(Event.COMPLETE, imgLoaded);
function imgLoaded(event:Event):void
{
    container.addChild(pictLdr.content);
}
```

Monitoramento do progresso do carregamento

Assim que o arquivo começa a ser carregado, um objeto LoaderInfo é criado. Um objeto LoaderInfo fornece informações como o progresso do carregamento, os URLs do carregador e do conteúdo carregado, o número de bytes totais para a mídia e a altura e largura nominais da mídia. O objeto LoaderInfo também envia eventos para o monitoramento do progresso do carregamento.

O diagrama a seguir mostra os diferentes usos do objeto LoaderInfo - para a ocorrência da classe principal do arquivo SWF, para um objeto Loader e para um objeto carregado por Loader:



O objeto LoaderInfo pode ser acessado como uma propriedade do objeto Loader e do objeto de exibição carregado. Assim que o carregamento começa, o objeto LoaderInfo pode ser acessado por meio da propriedade `contentLoaderInfo` do objeto Loader. Quando o carregamento do objeto de exibição termina, o objeto LoaderInfo também pode ser acessado como uma propriedade do objeto de exibição carregado pela propriedade `loaderInfo`. A propriedade `loaderInfo` do objeto de exibição carregado refere-se ao mesmo objeto LoaderInfo da propriedade `contentLoaderInfo` do objeto Loader. Em outras palavras, um objeto LoaderInfo é compartilhado entre um objeto carregado e o objeto Loader que o carregou (entre o carregador e o carregado).

Para acessar as propriedades do conteúdo carregado, adicione um ouvinte de eventos ao objeto LoaderInfo, assim como no código a seguir:

```
import flash.display.Loader;
import flash.display.Sprite;
import flash.events.Event;

var ldr:Loader = new Loader();
var urlReq:URLRequest = new URLRequest("Circle.swf");
ldr.load(urlReq);
ldr.contentLoaderInfo.addEventListener(Event.COMPLETE, loaded);
addChild(ldr);

function loaded(event:Event):void
{
    var content:Sprite = event.target.content;
    content.scaleX = 2;
}
```

Para obter mais informações, consulte "Manipulação de eventos" na página 251.

Especificação do contexto do carregamento

Quando você carrega um arquivo externo no Flash Player ou no AIR com o método `load()` ou `loadBytes()` da classe Loader, pode especificar, se desejar, um parâmetro `context`. Este parâmetros é um objeto LoaderContext.

A classe LoaderContext inclui três propriedades que permitem definir o contexto de como o conteúdo carregado pode ser usado:

- `checkPolicyFile`: Use essa propriedade apenas ao carregar um arquivo de imagem (não um arquivo SWF). Se você definir a propriedade como `true`, o Loader verificará o servidor de origem de um determinado arquivo de política (consulte "Controles de site (arquivos de política)" na página 711). Isso é necessário apenas para o conteúdo originado de domínios diferentes dos domínios do arquivo SWF que contém o objeto Loader. Se o servidor conceder permissão ao domínio de Loader, o ActionScript dos arquivos SWF do domínio de Loader poderá acessar os dados na imagem carregada; em outras palavras, você pode usar o comando `BitmapData.draw()` para acessar os dados na imagem carregada.

  Observe que um arquivo SWF de outros domínios que não são os do objeto Loader pode chamar `Security.allowDomain()` para permitir um domínio específico.

- `securityDomain`: Só use essa propriedade ao carregar um arquivo SWF (não uma imagem). Especifique-a para um arquivo SWF de um domínio diferente daquele do arquivo que contém o objeto Loader. Quando essa opção é especificada, o Flash Player verifica a existência de um arquivo de política e, se existir algum, os arquivos SWF dos domínios permitidos no arquivo de política poderão cruzar o script do conteúdo SWF carregado. Você pode especificar `flash.system.SecurityDomain.currentDomain` como este parâmetro.

- `applicationDomain`: Use essa propriedade apenas ao carregar um arquivo SWF gravado no ActionScript . (não em uma imagem ou arquivo SWF gravado no ActionScript 1.0 ou 2.0). Ao carregar o arquivo, você pode especificar que o arquivo seja incluído no mesmo domínio de aplicativo do objeto Loader, definindo o parâmetro `applicationDomain` como `flash.system.ApplicationDomain.currentDomain`. Colocando o arquivo SWF carregado no mesmo domínio de aplicativo, é possível acessar suas classes diretamente. Isso pode ser útil se estiver carregando um arquivo SWF que contém mídia incorporada, que pode ser acessada por meio dos nomes de classe associados. Para obter mais informações, consulte "Uso da classe ApplicationDomain" na página 657.

Veja um exemplo de busca de um arquivo de política ao carregar um bitmap de outro domínio:

```
var context:LoaderContext = new LoaderContext();
context.checkPolicyFile = true;
var urlReq:URLRequest = new URLRequest("http://www.[your_domain_here].com/photo11.jpg");
var ldr:Loader = new Loader();
ldr.load(urlReq, context);
```

Veja um exemplo de busca de um arquivo de política ao carregar um SWF de outro domínio para colocar o arquivo na mesma caixa de proteção do objeto Loader. Além disso, o código adiciona as classes do arquivo SWF carregado ao mesmo domínio de aplicativo do objeto Loader:

```
var context:LoaderContext = new LoaderContext();
context.securityDomain = SecurityDomain.currentDomain;
context.applicationDomain = ApplicationDomain.currentDomain;
var urlReq:URLRequest = new URLRequest("http://www.[your_domain_here].com/library.swf");
var ldr:Loader = new Loader();
ldr.load(urlReq, context);
```

Para obter mais informações, consulte a classe LoaderContext na Referência dos componentes e da linguagem do ActionScript 3.0.

Exemplo: SpriteArranger

O aplicativo de amostra SpriteArranger é criado com base no aplicativo de exemplo Geometric Shapes descrito separadamente (consulte “Exemplo: GeometricShapes” na página 125).

O aplicativo de amostra SpriteArranger ilustra diversos conceitos de manipulação de objetos de exibição:

- Extensão de classes de objeto de exibição
- Adição de objetos à lista de exibição
- Disposição em camadas de objetos de exibição e trabalho com contêineres de objeto de exibição
- Resposta a eventos de objeto de exibição
- Uso de propriedades e métodos de objetos de exibição

Para obter os arquivos de aplicativo desse exemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_br. Os arquivos do aplicativo SpriteArranger estão localizados na pasta Exemplos/SpriteArranger. O aplicativo consiste nos seguintes arquivos:

Arquivo	Descrição
SpriteArranger.mxml ou SpriteArranger.fla	O arquivo principal do aplicativo no Flash (FLA) ou no Flex (MXML).
com/example/programmingas3/SpriteArranger/CircleSprite.as	Uma classe que define um tipo de objeto Sprite que processa um círculo na tela.
com/example/programmingas3/SpriteArranger/DrawingCanvas.as	Uma classe que define a tela, que é um contêiner de objeto de exibição que contém objetos GeometricSprite.
com/example/programmingas3/SpriteArranger/SquareSprite.as	Uma classe que define um tipo de objeto Sprite que processa um quadrado na tela.
com/example/programmingas3/SpriteArranger/TriangleSprite.as	Uma classe que define um tipo de objeto Sprite que processa um triângulo na tela.

Arquivo	Descrição
com/example/programmingas3/SpriteArranger/GeometricSprite.as	Uma classe que estende o objeto Sprite, usado para definir uma forma na tela. CircleSprite, SquareSprite e TriangleSprite estendem essa classe.
com/example/programmingas3/geometricshapes/IGeometricShape.as	A interface básica que define os métodos a serem implementados por todas as classes de forma geométrica.
com/example/programmingas3/geometricshapes/IPolygon.as	Uma interface que define os métodos a serem implementados pelas classes de forma geométrica que têm vários lados.
com/example/programmingas3/geometricshapes/RegularPolygon.as	Um tipo de forma geométrica que tem lados com o mesmo comprimento posicionados simetricamente ao redor do centro da forma.
com/example/programmingas3/geometricshapes/Circle.as	Um tipo de forma geométrica que define um círculo.
com/example/programmingas3/geometricshapes/EquilateralTriangle.as	Uma subclasse de RegularPolygon que define um triângulo com todos os lados com o mesmo comprimento.
com/example/programmingas3/geometricshapes/Square.as	Uma subclasse de RegularPolygon que define um retângulo com os quatro lados com o mesmo comprimento.
com/example/programmingas3/geometricshapes/GeometricShapeFactory.as	Uma classe que contém um "método de fábrica" para criar formas com tamanho e tipo especificados.

Definição das classes SpriteArranger

O aplicativo SpriteArranger permite que o usuário adicione vários objetos de exibição à "tela" exibida.

A classe DrawingCanvas define uma área de desenho, um tipo de contêiner de objeto de exibição ao qual o usuário pode adicionar formas na tela. Essas formas na tela são ocorrências de uma das subclasses de GeometricSprite.

A classe DrawingCanvas

A classe DrawingCanvas estende a classe Sprite e sua herança é definida na declaração da classe DrawingCanvas, do seguinte modo:

```
public class DrawingCanvas extends Sprite
```

A classe Sprite é uma subclasse de DisplayObjectContainer e de DisplayObject, e a classe DrawingCanvas usa métodos e propriedades dessas classes.

O método do construtor `DrawingCanvas()` configura um objeto de Rectangle, `bounds`, que é a propriedade usada posteriormente no desenho do contorno da tela. Em seguida, o método `initCanvas()` é chamado do seguinte modo:

```
this.bounds = new Rectangle(0, 0, w, h);
initCanvas(fillColor, lineColor);
```

Como mostra o exemplo a seguir, o método `initCanvas()` define várias propriedades do objeto DrawingCanvas, que foram transmitidas como argumentos para a função do construtor:

```
this.lineColor = lineColor;
this.fillColor = fillColor;
this.width = 500;
this.height = 200;
```

O método `initCanvas()` chama o método `drawBounds()`, que desenha a tela usando a propriedade `graphics` da classe DrawingCanvas. A propriedade `graphics` é herdada da classe Shape.

```
this.graphics.clear();
this.graphics.lineStyle(1.0, this.lineColor, 1.0);
this.graphics.beginFill(this.fillColor, 1.0);
this.graphics.drawRect(bounds.left - 1,
                       bounds.top - 1,
                       bounds.width + 2,
                       bounds.height + 2);
this.graphics.endFill();
```

Os métodos adicionais a seguir da classe DrawingCanvas são invocados com base nas interações do usuário com o aplicativo:

- Os métodos `addShape()` e `describeChildren()`, que são descritos em "Adição de objetos de exibição à tela" na página 319
- Os métodos `moveToBack()`, `moveDown()`, `moveToFront()` e `moveUp()`, que são descritos em "Reorganização da disposição em camadas do objeto de exibição" na página 321
- O método `onMouseUp()`, que é descrito em "Clique e arrasto de objetos de exibição" na página 320

A classe GeometricSprite e suas subclasses

Cada objeto de exibição que pode ser adicionado à tela pelo usuário é uma ocorrência de uma das seguintes subclasses de GeometricSprite:

- CircleSprite
- SquareSprite
- TriangleSprite

A classe GeometricSprite estende a classe flash.display.Sprite:

```
public class GeometricSprite extends Sprite
```

A classe GeometricSprite inclui diversas propriedades comuns a todos os objetos GeometricSprite. Essas propriedades são definidas na função do construtor com base nos parâmetros transmitidos para a função. Por exemplo:

```
this.size = size;
this.lineColor = lColor;
this.fillColor = fColor;
```

A propriedade `geometricShape` da classe GeometricSprite define uma interface IGeometricShape, que define as propriedades matemáticas, mas não as visuais, da forma. As classes que implementam a interface IGeometricShape são definidas no aplicativo de amostra GeometricShapes (consulte "Exemplo: GeometricShapes" na página 125).

A classe GeometricSprite define o método `drawShape()`, que é refinado ainda mais nas definições de substituição em cada subclasse de GeometricSprite. Para obter mais informações, consulte a seção "Adição de objetos de exibição à tela", apresentada a seguir.

A classe GeometricSprite também fornece os seguintes métodos:

- Os métodos `onMouseDown()` e `onMouseUp()`, que são descritos em "Clique e arrasto de objetos de exibição" na página 320

- Os métodos `showSelected()` e `hideSelected()`, que são descritos em "Clique e arrasto de objetos de exibição" na página 320

Adição de objetos de exibição à tela

Quando o usuário clica no botão Adicionar forma, o aplicativo chama o método `addShape()` da classe DrawingCanvas. Esse método percorre um novo GeometricSprite chamando a função do construtor adequada de uma das subclasses de GeometricSprite, como mostra o exemplo a seguir:

```
public function addShape(shapeName:String, len:Number):void
{
    var newShape:GeometricSprite;
    switch (shapeName)
    {
        case "Triangle":
            newShape = new TriangleSprite(len);
            break;

        case "Square":
            newShape = new SquareSprite(len);
            break;

        case "Circle":
            newShape = new CircleSprite(len);
            break;
    }
    newShape.alpha = 0.8;
    this.addChild(newShape);
}
```

Cada método do construtor chama o método `drawShape()`, que usa a propriedade `graphics` da classe (herdada da classe Sprite) para desenhar o gráfico vetorial adequado. Por exemplo, o método `drawShape()` da classe CircleSprite inclui o seguinte código:

```
this.graphics.clear();
this.graphics.lineStyle(1.0, this.lineColor, 1.0);
this.graphics.beginFill(this.fillColor, 1.0);
var radius:Number = this.size / 2;
this.graphics.drawCircle(radius, radius, radius);
```

Da segunda à última linha da função `addShape()`, é definida a propriedade `alpha` do objeto de exibição (herdada da classe DisplayObject) para que cada objeto de exibição adicionado à tela seja ligeiramente transparente, deixando o usuário ver os objetos que estão por trás.

A linha final do método `addChild()` adiciona o novo objeto de exibição à lista de filhos da ocorrência da classe DrawingCanvas, que já está na lista de exibição. Desse modo, o novo objeto de exibição aparece no palco.

A interface do aplicativo inclui dois campos de texto, `selectedSpriteTxt` e `outputTxt`. As propriedades de texto desses campos são atualizados com informações sobre os objetos GeometricSprite que foram adicionados à tela ou selecionados pelo usuário. A classe GeometricSprite manipula essa tarefa de registro de informações substituindo o método `toString()` da seguinte maneira:

```
public override function toString():String
{
    return this.shapeType + " of size " + this.size + " at " + this.x + ", " + this.y;
}
```

A propriedade `shapeType` é definida como o valor adequado no método do construtor de cada subclasse de GeometricSprite. Por exemplo, o método `toString()` poderia retornar o seguinte valor para uma ocorrência de CircleSprite adicionada recentemente à ocorrência de DrawingCanvas:

```
Circle of size 50 at 0, 0
```

O método `describeChildren()` da classe DrawingCanvas percorre a lista de filhos da tela, usando a propriedade `numChildren` (herdada da classe DisplayObjectContainer), para definir o limite do loop `for`. É gerada uma string que lista cada filho, do seguinte modo:

```
var desc:String = "";
var child:DisplayObject;
for (var i:int=0; i < this.numChildren; i++)
{
child = this.getChildAt(i);
desc += i + ": " + child + '\n';
}
```

A string resultante é usada para definir a propriedade `text` do campo de texto `outputTxt`.

Clique e arrasto de objetos de exibição

Quando o usuário clica em uma ocorrência de GeometricSprite, o aplicativo chama o manipulador de eventos `onMouseDown()`. Conforme mostrado a seguir, esse manipulador de eventos é definido para ouvir eventos de mouse na função do construtor da classe GeometricSprite:

```
this.addEventListener(MouseEvent.MOUSE_DOWN, onMouseDown);
```

O método `onMouseDown()` chama o método `showSelected()` do objeto GeometricSprite. Se for a primeira vez que esse método foi chamado para o objeto, o método criará um novo objeto Shape chamado `selectionIndicator` e usará a propriedade `graphics` do objeto Shape para desenhar um retângulo de realce vermelho, do seguinte modo:

```
this.selectionIndicator = new Shape();
this.selectionIndicator.graphics.lineStyle(1.0, 0xFF0000, 1.0);
this.selectionIndicator.graphics.drawRect(-1, -1, this.size + 1, this.size + 1);
this.addChild(this.selectionIndicator);
```

Se não for a primeira vez que o método `onMouseDown()` é chamado, o método simplesmente definirá a propriedade `visible` da forma `selectionIndicator` (herdada da classe DisplayObject), do seguinte modo:

```
this.selectionIndicator.visible = true;
```

O método `hideSelected()` oculta a forma `selectionIndicator` do objeto selecionado anteriormente definindo a propriedade `visible` como `false`.

O manipulador de eventos `onMouseDown()` também chama o método `startDrag()` (herdado da classe Sprite), que inclui o seguinte código:

```
var boundsRect:Rectangle = this.parent.getRect(this.parent);
boundsRect.width -= this.size;
boundsRect.height -= this.size;
this.startDrag(false, boundsRect);
```

Isso permite que o usuário arraste o objeto selecionado em torno da tela, dentro dos limites definidos pelo retângulo `boundsRect`.

Quando o usuário solta o botão do mouse, o evento `mouseUp` é enviado. O método do construtor de DrawingCanvas configura o seguinte ouvinte de eventos:

```
this.addEventListener(MouseEvent.MOUSE_UP, onMouseUp);
```

Esse ouvinte de eventos é definido para o objeto DrawingCanvas, não para objetos GeometricSprite individuais. Isso ocorre porque quando o objeto GeometricSprite é arrastado, pode terminar atrás de outro objeto de exibição (outro objeto GeometricSprite) assim que o mouse é solto. O objeto de exibição em primeiro plano receberia o evento de mouse, mas o objeto de exibição que está sendo arrastado pelo usuário não. A adição do ouvinte ao objeto DrawingCanvas garante que o evento seja sempre manipulado.

O método `onMouseUp()` chama o método `onMouseUp()` do objeto GeometricSprite, que, por sua vez, chama o método `stopDrag()` do objeto GeometricSprite.

Reorganização da disposição em camadas do objeto de exibição

A interface de usuário do aplicativo inclui os botões Mover para trás, Mover para baixo, Mover para cima e Mover para frente. Quando o usuário clica em um desses botões, o aplicativo chama o método correspondente da classe DrawingCanvas: `moveToBack()`, `moveDown()`, `moveUp()` ou `moveToFront()`. Por exemplo, o método `moveToBack()` inclui o seguinte código:

```
public function moveToBack(shape:GeometricSprite):void
{
    var index:int = this.getChildIndex(shape);
    if (index > 0)
    {
        this.setChildIndex(shape, 0);
    }
}
```

O método `setChildIndex()` (herdado da classe DisplayObjectContainer) é usado para colocar o objeto de exibição na posição de índice 0 na lista de filhos da ocorrência de DrawingCanvas (`this`).

O método `moveDown()` funciona de modo similar, mas diminui a posição de índice do objeto de exibição em incrementos de 1 na lista de filhos da ocorrência de DrawingCanvas:

```
public function moveDown(shape:GeometricSprite):void
{
    var index:int = this.getChildIndex(shape);
    if (index > 0)
    {
        this.setChildIndex(shape, index - 1);
    }
}
```

Os métodos `moveUp()` e `moveToFront()` funcionam de modo similar aos métodos `moveToBack()` e `moveDown()`.

Capítulo 14: Uso de objetos visuais

Embora as imagens e a arte final importadas sejam importantes, a funcionalidade conhecida como API de desenho, que permite desenhar linhas e formas no ActionScript, dá a você a liberdade de iniciar um aplicativo com o equivalente computacional de uma tela em branco, na qual é possível criar as imagens desejadas. A habilidade de criar seus próprios gráficos abre muitas possibilidades para seus aplicativos. Com as técnicas discutidas neste capítulo, você pode criar um programa de desenho, fazer imagens animadas e interativas ou criar de modo programático seus próprios elementos de interface do usuário, entre outras possibilidades.

Noções básicas do uso da API de desenho

Introdução ao uso da API de desenho

API de desenho é o nome da funcionalidade incorporada ao ActionScript que permite criar gráficos vetoriais (linhas, curvas, formas, preenchimentos e gradientes) e exibi-los na tela usando o ActionScript. A classe flash.display.Graphics oferece esta funcionalidade. É possível desenhar com o ActionScript em qualquer ocorrência de Shape, Sprite ou MovieClip usando a propriedade `graphics` definida em cada uma dessas classes. (A propriedade `graphics` de cada uma dessas classes é, na verdade, uma ocorrência da classe Graphics.)

Se você está apenas começando a desenhar com código, a classe Graphics inclui diversos métodos que ajudam a desenhar formas comuns, como círculos, elipses, retângulos e retângulos com cantos arredondados. É possível desenhá-las como linhas vazias ou formas preenchidas. Quando você precisar de funcionalidade mais avançada, a classe Graphics também inclui métodos para desenhar linhas e curvas Bézier quadráticas, que podem ser usadas junto com as funções de trigonometria da classe Math para criar qualquer forma desejada.

O Flash Player 10 adiciona mais uma API de desenho, que permite desenhar formas inteiras de modo programático usando um único comando. Depois que você estiver familiarizado com a classe Graphics e as tarefas incluídas em "Noções básicas do uso da API de desenho", passe para "Uso avançado da API de desenho" na página 334 para saber mais sobre esses recursos da API de desenho.

Tarefas comuns da API de desenho

As seguintes tarefas são operações que provavelmente você executará usando a API de desenho do ActionScript e que estão descritas neste capítulo:

- Definição de estilos de linha e estilos de preenchimento para desenhar formas
- Desenho de linhas retas e curvas
- Uso de métodos para desenhar formas como círculos, elipses e retângulos
- Desenho com linhas e preenchimentos de gradiente
- Definição de matrizes para criar gradientes
- Uso de trigonometria com a API de desenho
- Incorporação da API de desenho em animações

Conceitos e termos importantes

A lista de referência a seguir contém termos importantes usados neste capítulo:

- Ponto de ancoragem: uma das duas extremidades de uma curva Bézier quadrática.
- Ponto de controle: o ponto que define a direção e o grau da curva de uma curva Bézier quadrática. A linha curva nunca atinge o ponto de controle, mas a linha faz uma curva como se estivesse sendo desenhada perto do ponto de controle.
- Espaço de coordenadas: o gráfico de coordenadas contido em um objeto de exibição, no qual seus elementos-filho são posicionados.
- Preenchimento: a parte interna sólida de uma forma que tem uma linha preenchida com cor ou uma forma inteira que não tem contorno.
- Gradiente: cor que consiste em uma transição gradual de uma cor para uma ou mais cores (ao contrário de uma cor sólida).
- Ponto: um local isolado em um espaço de coordenadas. No sistema de coordenadas bidimensional usado no ActionScript, um ponto é definido por seu local no eixo x e no eixo y (as coordenadas do ponto).
- Curva Bézier quadrática: tipo de curva definido por uma fórmula matemática específica. Neste tipo de curva, a forma de uma curva é calculada com base nas posições dos pontos de ancoragem (as extremidades da curva) e em um ponto de controle que define o grau e a direção da curva.
- Escala: o tamanho de um objeto em relação ao seu tamanho original. Quando usado como um verbo, dimensionar um objeto significa alterar seu tamanho, aumentando ou diminuindo o objeto.
- Traçado: a parte do contorno de uma forma que tem uma linha preenchida com cor ou as linhas de uma forma sem preenchimento.
- Transpor: alterar as coordenadas de um ponto de um espaço de coordenadas para outro.
- Eixo X: o eixo horizontal no sistema de coordenadas bidimensional usado no ActionScript.
- Eixo Y: o eixo vertical no sistema de coordenadas bidimensional usado no ActionScript.

Teste dos exemplos do capítulo

Talvez você queira testar algumas das listagens de código de exemplo durante a leitura deste capítulo. Como este capítulo aborda o desenho de conteúdo visual, o teste das listagens de código envolve a execução do código e a visualização dos resultados no SWF que é criado. Para testar as listagens de código:

1. Crie um documento do Flash vazio.
2. Selecione um quadro-chave na Linha de tempo.
3. Abra o painel Ações e copie a listagem de código no painel Script.
4. Execute o programa usando Controlar > Testar filme.

   Você verá os resultados da listagem de código no arquivo SWF que é criado.

Compreensão da classe Graphics

Cada objeto Shape, Sprite e MovieClip tem uma propriedade `graphics`, que é uma ocorrência da classe Graphics. A classe Graphics inclui propriedades e métodos para desenhar linhas, preenchimentos e formas. Se você quiser que o objeto display seja usado apenas como uma tela para o conteúdo do desenho, use uma ocorrência Shape. Uma ocorrência Shape será executada de modo melhor do que outros objetos display para desenho, porque ela não tem a sobrecarga da funcionalidade adicional nas classes Sprite e MovieClip. Se quiser um objeto display no qual seja possível desenhar conteúdo gráfico e no qual outros objetos display estejam contidos, use uma ocorrência Sprite. Para obter mais informações sobre a determinação dos objetos display que serão usados para várias tarefas, consulte "Escolha de uma subclasse de DisplayObject" na página 292.

Desenho de linhas e curvas

Todos os desenhos feitos com uma ocorrência Graphics baseiam-se em desenhos básicos com linhas e curvas. Conseqüentemente, todos os desenhos do ActionScript devem ser executados utilizando a mesma série de etapas:

- Definir estilos de linha e preenchimento
- Definir posição inicial do desenho
- Desenhar linhas, curvas e formas (opcionalmente movendo o ponto de desenho)
- Concluir a criação de um preenchimento, se necessário

Definição de estilos de linha e preenchimento

Para desenhar com a propriedade `graphics` de uma ocorrência Shape, Sprite ou MovieClip, defina primeiro o estilo (tamanho e cor da linha, cor do preenchimento) a ser utilizado durante o desenho. Da mesma forma que acontece quando você usa as ferramentas de desenho do Adobe® Flash® CS4 Professional ou outro aplicativo de desenho, ao utilizar o ActionScript para desenhar, você pode desenhar com ou sem traçado e com ou sem uma cor de preenchimento. Você especifica a aparência do traçado utilizando o método `lineStyle()` ou `lineGradientStyle()`. Para criar uma linha sólida, use o método `lineStyle()`. Quando você chamar esse método, os valores mais comuns que você especificará serão os três primeiros parâmetros: espessura da linha, cor e alfa. Por exemplo, essa linha de código instrui Shape chamada `myShape` a desenhar linhas com espessura de 2 pixels, vermelhas (0x990000) e 75% opacas:

```
myShape.graphics.lineStyle(2, 0x990000, .75);
```

O valor padrão para o parâmetro alpha é 1.0 (100%), então você pode deixá-lo desativado, se quiser uma linha totalmente opaca. O método `lineStyle()` também aceita dois parâmetros adicionais para dica de pixel e modo de escala; para obter mais informações sobre o uso desses parâmetros, consulte a descrição do método `Graphics.lineStyle()` em Referência dos componentes e da linguagem do ActionScript 3.0.

Para criar uma linha gradiente, use o método `lineGradientStyle()`. Esse método é descrito em "Criação de linhas e preenchimentos gradientes" na página 327.

Se você quiser criar uma forma preenchida, chame os métodos `beginFill()`, `beginGradientFill()`, `beginBitmapFill()` ou `beginShaderFill()` antes de iniciar o desenho. O mais básico deles, o método `beginFill()`, aceita dois parâmetros: a cor de preenchimento e (opcionalmente) um valor alfa para a cor de preenchimento. Por exemplo, se você quiser desenhar uma forma com um preenchimento verde sólido, utilize o seguinte código (considerando que está desenhando em um objeto chamado `myShape`):

```
myShape.graphics.beginFill(0x00FF00);
```

Chamar qualquer método de preenchimento implicitamente encerra qualquer preenchimento anterior antes de iniciar um novo. Chamar qualquer método que especifique um estilo de traçado substitui o traçado anterior, mas não altera o preenchimento especificado anteriormente e vice-versa.

Depois de especificar as propriedades style e fill da linha, a próxima etapa é indicar o ponto de início para o desenho. A ocorrência Graphics tem um ponto de desenho, como a ponta de uma caneta em um pedaço de papel. Sempre que o ponto de desenho for localizado, esse será o local em que a próxima ação de desenho será iniciada. Inicialmente um objeto Graphics começa com seu ponto de desenho no ponto 0, 0 no espaço de coordenadas do objeto em que está o desenho. Para iniciar o desenho em um ponto diferente, você pode primeiro chamar o método `moveTo()` antes de chamar um dos métodos de desenho. Isso é semelhante a tirar do papel a ponta da caneta e movê-la para uma nova posição.

Com o ponto de desenho no local, desenhe usando uma série de chamadas de métodos de desenho `lineTo()` (para linhas retas) e `curveTo()` (para linhas curvas).

Enquanto estiver desenhando, chame o método `moveTo()` a qualquer momento para mover o ponto de desenho para uma nova posição sem desenhar.

Enquanto desenha, se você especificou uma cor de preenchimento, poderá orientar o Adobe Flash Player ou o Adobe® AIR™ a encerrar o preenchimento chamando o método `endFill()`. Se você não desenhou uma forma fechada (em outras palavras, se no momento em que você chama `endFill()`, o ponto de desenho não está no ponto inicial da forma), quando chamou o método `endFill()`, o Flash Player ou o AIR fecha automaticamente a forma desenhando uma linha reta do ponto de desenho atual até o local especificado na chamada mais recente de `moveTo()`. Se você iniciou um preenchimento e não chamou `endFill()`, chamar `beginFill()` (ou qualquer um dos outros métodos de preenchimento) fecha o preenchimento atual e inicia um novo.

Desenho de linhas retas

Quando você chama o método `lineTo()`, o objeto Graphics desenha uma linha reta do ponto de desenho atual até as coordenadas especificadas como os dois parâmetros na chamada do método, com o estilo de linha também especificado. Por exemplo, essa linha de código coloca o ponto de desenho no ponto 100, 100 e desenha uma linha até o ponto 200, 200:

```
myShape.graphics.moveTo(100, 100);
myShape.graphics.lineTo(200, 200);
```

O exemplo a seguir desenha triângulos vermelho e verde com uma altura de 100 pixels:

```
var triangleHeight:uint = 100;
var triangle:Shape = new Shape();

// red triangle, starting at point 0, 0
triangle.graphics.beginFill(0xFF0000);
triangle.graphics.moveTo(triangleHeight / 2, 0);
triangle.graphics.lineTo(triangleHeight, triangleHeight);
triangle.graphics.lineTo(0, triangleHeight);
triangle.graphics.lineTo(triangleHeight / 2, 0);

// green triangle, starting at point 200, 0
triangle.graphics.beginFill(0x00FF00);
triangle.graphics.moveTo(200 + triangleHeight / 2, 0);
triangle.graphics.lineTo(200 + triangleHeight, triangleHeight);
triangle.graphics.lineTo(200, triangleHeight);
triangle.graphics.lineTo(200 + triangleHeight / 2, 0);

this.addChild(triangle);
```

Desenho de curvas

O método `curveTo()` desenha uma curva Bézier quadrática. Isso desenha um arco que conecta dois pontos (chamados pontos de ancoragem, enquanto se curva a um terceiro ponto (chamado ponto de controle). O objeto Graphics usa a posição atual do desenho como o primeiro ponto de ancoragem. Quando você chama o método `curveTo()`, passa quatro parâmetros: as coordenadas x e y do ponto de controle, seguido pelas coordenadas x e y do segundo ponto de ancoragem. Por exemplo, o seguinte código desenha uma curva começando no ponto 100, 100 e terminando no ponto 200, 200. Como o ponto de controle está no ponto 175, 125, isso cria uma curva que se move para a direita e para baixo:

```
myShape.graphics.moveTo(100, 100);
myShape.graphics.curveTo(175, 125, 200, 200);
```

O exemplo a seguir desenha objetos circulares vermelho e verde com uma largura e altura de 100 pixels. Observe que, devido à natureza da equação de Bézier de segundo grau, esses círculos não são perfeitos.

```
var size:uint = 100;
var roundObject:Shape = new Shape();

// red circular shape
roundObject.graphics.beginFill(0xFF0000);
roundObject.graphics.moveTo(size / 2, 0);
roundObject.graphics.curveTo(size, 0, size, size / 2);
roundObject.graphics.curveTo(size, size, size / 2, size);
roundObject.graphics.curveTo(0, size, 0, size / 2);
roundObject.graphics.curveTo(0, 0, size / 2, 0);

// green circular shape
roundObject.graphics.beginFill(0x00FF00);
roundObject.graphics.moveTo(200 + size / 2, 0);
roundObject.graphics.curveTo(200 + size, 0, 200 + size, size / 2);
roundObject.graphics.curveTo(200 + size, size, 200 + size / 2, size);
roundObject.graphics.curveTo(200, size, 200, size / 2);
roundObject.graphics.curveTo(200, 0, 200 + size / 2, 0);

this.addChild(roundObject);
```

Desenho de formas utilizando os métodos incorporados

Para sua conveniência, ao desenhar formas comuns como círculo, elipse, retângulos e retângulas com cantos arredondados, o ActionScript 3.0 tem métodos que desenham essas formas comuns para você. Esses são os métodos `drawCircle()`, `drawEllipse()`, `drawRect()`, `drawRoundRect()` e`drawRoundRectComplex()` da classe Graphics. Esses métodos também podem ser utilizados no lugar dos métodos `lineTo()` e `curveTo()`. Observe, entretanto, que você ainda deve especificar os estilos de linha e preenchimento antes de chamar esses métodos.

O exemplo a seguir recria o exemplo de desenho de quadrados vermelho, verde e azul com largura e altura de 100 pixels. Esse código usa o método `drawRect()` e, além disso, especifica que a cor de preenchimento tem um alfa de 50% (0,5):

```
var squareSize:uint = 100;
var square:Shape = new Shape();
square.graphics.beginFill(0xFF0000, 0.5);
square.graphics.drawRect(0, 0, squareSize, squareSize);
square.graphics.beginFill(0x00FF00, 0.5);
square.graphics.drawRect(200, 0, squareSize, squareSize);
square.graphics.beginFill(0x0000FF, 0.5);
square.graphics.drawRect(400, 0, squareSize, squareSize);
square.graphics.endFill();
this.addChild(square);
```

Em um objeto Sprite ou MovieClip, o conteúdo de desenho criado com a propriedade `graphics` sempre aparece atrás de todos os objetos display filhos contidos no objeto. Além disso, o conteúdo da propriedade `graphics` não é um objeto display separado; portanto ele não aparece na lista de filhos do objeto Sprite ou MovieClip. Por exemplo, o seguinte objeto Sprite tem um círculo desenhado com sua propriedade `graphics` e um objeto TextField em sua lista de objetos display filhos:

```
var mySprite:Sprite = new Sprite();
mySprite.graphics.beginFill(0xFFCC00);
mySprite.graphics.drawCircle(30, 30, 30);
var label:TextField = new TextField();
label.width = 200;
label.text = "They call me mellow yellow...";
label.x = 20;
label.y = 20;
mySprite.addChild(label);
this.addChild(mySprite);
```

Observe que TextField aparece na parte superior do círculo desenhado com o objeto graphics.

Criação de linhas e preenchimentos gradientes

O objeto graphics também pode desenhar traçados e preenchimentos com gradientes em vez de cores sólidas. Um traçado gradiente é criado com o método `lineGradientStyle()` e um preenchimento de gradiente é criado com o método `beginGradientFill()`.

Os dois métodos aceitam os mesmos parâmetros. Os primeiros quatro são obrigatórios: tipo, cores, alfas e proporções. Os outros quatro são opcionais, mas úteis para personalização avançada.

- O primeiro parâmetro especifica o tipo de gradiente que está criando. Os valores aceitáveis são `GradientFill.LINEAR` ou `GradientFill.RADIAL`.
- O segundo parâmetro especifica a matriz dos valores de cor que serão utilizados. Em um gradiente linear, as cores serão organizadas da esquerda para a direita. Em um gradiente radial, as cores serão organizadas de dentro para fora. A ordem das cores da matriz representa a ordem em que elas serão desenhadas no gradiente.
- O terceiro parâmetro especifica os valores de transparência alfa das cores correspondentes no parâmetro anterior.
- O quarto parâmetro especifica as proporções ou a ênfase que cada cor tem no gradiente. A faixa de valores aceitável é de 0 a 255. Esses valores não representam a largura ou a altura, mas a posição no gradiente; 0 representa o início do gradiente e 255 representa o final do gradiente. A matriz de proporções deve aumentar seqüencialmente e tem o mesmo número de entradas que as matrizes de cores e alfa especificadas no segundo e no terceiro parâmetros.

Embora o quinto parâmetro, a matriz de transformação, seja opcional, ele é normalmente usado porque fornece um modo fácil e eficiente de controlar a aparência do gradiente. Esse parâmetro aceita uma ocorrência Matrix. O modo mais fácil de criar um objeto Matrix para um gradiente é usar o método `createGradientBox()` da classe Matrix.

Definição de um objeto Matrix para usar com um gradiente

Você usa os métodos `beginGradientFill()` e `lineGradientStyle()` da classe flash.display.Graphics para definir gradientes para usar em formas. Ao definir um gradiente, você fornece uma matriz como um dos parâmetros desses métodos.

A maneira mais fácil de definir a matriz é utilizando o método `createGradientBox()` da classe Matrix, que cria uma matriz utilizada para definir o gradiente. Você define a escala, a rotação e a posição do gradiente utilizando os parâmetros passados para o método `createGradientBox()`. O método `createGradientBox()` aceita os seguintes parâmetros:

- Largura da caixa de gradientes: a largura (em pixels) para a qual o gradiente será ampliado.
- Altura da caixa de gradientes: a altura (em pixels) para a qual o gradiente será ampliado.
- Rotação da caixa de gradientes: a rotação (em radianos) que será aplicada ao gradiente.
- Movimentação horizontal: a distância (em pixels) que o gradiente se deslocará horizontalmente.
- Movimentação vertical: a distância (em pixels) que o gradiente se deslocará verticalmente.

Por exemplo, considere um gradiente com as seguintes características:

- `GradientType.LINEAR`
- Duas cores, verde e azul, com a matriz de `proporções` definidas para `[0, 255]`.
- `SpreadMethod.PAD`
- `InterpolationMethod.LINEAR_RGB`

Os seguintes exemplos mostram gradientes nos quais o parâmetro `rotation` do método `createGradientBox()` é diferente conforme indicado, mas todas as configurações permanecem as mesmas:

`width = 100;`
`height = 100;`
`rotation = 0;`
`tx = 0;`
`ty = 0;`
	

`width = 100;`
`height = 100;`
`rotation = Math.PI/4; // 45°`
`tx = 0;`
`ty = 0;`
	

`width = 100;`
`height = 100;`
`rotation = Math.PI/2; // 90°`
`tx = 0;`
`ty = 0;`
	

Os seguintes exemplos mostram os efeitos de um gradiente linear verde para azul no qual os parâmetros `rotation`, `tx` e `ty` do método `createGradientBox()` são diferentes conforme indicado, mas todas as outras configurações permanecem as mesmas:

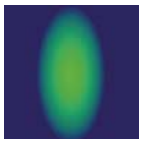
`width = 50;`
`height = 100;`
`rotation = 0;`
`tx = 0;`
`ty = 0;`
	

`width = 50;`
`height = 100;`
`rotation = 0`
`tx = 50;`
`ty = 0;`
	

`width = 100;`
`height = 50;`
`rotation = Math.PI/2; // 90°`
`tx = 0;`
`ty = 0;`
	

`width = 100;`
`height = 50;`
`rotation = Math.PI/2; // 90°`
`tx = 0;`
`ty = 50;`
	

Os parâmetros `width`, `height`, `tx` e `ty` do método `createGradientBox()` também afetam o tamanho e a posição de um preenchimento de gradiente *radial*, conforme mostram os exemplos a seguir:

`width = 50;`
`height = 100;`
`rotation = 0;`
`tx = 25;`
`ty = 0;`
	

O código a seguir produz o último gradiente radial ilustrado:

```
import flash.display.Shape;
import flash.display.GradientType;
import flash.geom.Matrix;

var type:String = GradientType.RADIAL;
var colors:Array = [0x00FF00, 0x000088];
var alphas:Array = [1, 1];
var ratios:Array = [0, 255];
var spreadMethod:String = SpreadMethod.PAD;
var interp:String = InterpolationMethod.LINEAR_RGB;
var focalPtRatio:Number = 0;

var matrix:Matrix = new Matrix();
var boxWidth:Number = 50;
var boxHeight:Number = 100;
var boxRotation:Number = Math.PI/2; // 90°
var tx:Number = 25;
var ty:Number = 0;
matrix.createGradientBox(boxWidth, boxHeight, boxRotation, tx, ty);

var square:Shape = new Shape;
square.graphics.beginGradientFill(type,
                           colors,
                           alphas,
                           ratios,
                           matrix,
                           spreadMethod,
                           interp,
                           focalPtRatio);
square.graphics.drawRect(0, 0, 100, 100);
addChild(square);
```

Observe que a largura e a altura do preenchimento de gradiente são determinadas pela largura e altura da matriz de gradientes em vez da largura e altura desenhadas utilizando o objeto Graphics. Ao desenhar com objetos Graphics, você desenha o que existe naquelas coordenadas na matriz de gradiente. Mesmo que você use um dos métodos shape de um objeto Graphics como `drawRect()`, o gradiente não se amplia para o tamanho da forma que é desenhada — o tamanho do gradiente deve ser especificado na própria matrix de gradiente.

A seguir está uma ilustração da diferença visual entre as dimensões da matrix de gradiente e das dimensões do próprio desenho.

```
var myShape:Shape = new Shape();
var gradientBoxMatrix:Matrix = new Matrix();
gradientBoxMatrix.createGradientBox(100, 40, 0, 0, 0);
myShape.graphics.beginGradientFill(GradientType.LINEAR, [0xFF0000, 0x00FF00, 0x0000FF], [1,
1, 1], [0, 128, 255], gradientBoxMatrix);
myShape.graphics.drawRect(0, 0, 50, 40);
myShape.graphics.drawRect(0, 50, 100, 40);
myShape.graphics.drawRect(0, 100, 150, 40);
myShape.graphics.endFill();
this.addChild(myShape);
```

Esse código desenha três gradientes com o mesmo estilo de preenchimento, especificado com uma distribuição igual de vermelho, verde e azul. Os gradientes são desenhados utilizando o método `drawRect()` com larguras de pixel de 50, 100 e 150 respectivamente. A matriz de gradiente que é especificada no método `beginGradientFill()` é criada com uma largura de 100 pixels. Isso significa que o primeiro gradiente engloba apenas meio espectro de gradiente, o segundo engloba todo ele e o terceiro engloba todo ele e tem 50 pixels adicionais de azul estendidos para a direita.

O método `lineGradientStyle()` funciona de modo semelhante a `beginGradientFill()` exceto pelo fato de que ao definir o gradiente, você deve especificar a espessura do traço utilizando o método `lineStyle()` antes do desenho. O código a seguir desenha uma caixa com um traçado gradiente vermelho, verde e azul:

```
var myShape:Shape = new Shape();
var gradientBoxMatrix:Matrix = new Matrix();
gradientBoxMatrix.createGradientBox(200, 40, 0, 0, 0);
myShape.graphics.lineStyle(5, 0);
myShape.graphics.lineGradientStyle(GradientType.LINEAR, [0xFF0000, 0x00FF00, 0x0000FF], [1,
1, 1], [0, 128, 255], gradientBoxMatrix);
myShape.graphics.drawRect(0, 0, 200, 40);
this.addChild(myShape);
```

Para obter mais informações sobre a classe Matrix, consulte "Uso de objetos Matrix" na página 349.

Uso da classe Math com métodos de desenho

O objeto Graphics desenha círculos e quadrados, mas também pode desenhar formas mais complexas, especialmente quando os métodos de desenho são utilizados junto com as propriedades e os métodos da classe Math. A classe Math contém constantes matemáticas comuns, como `Math.PI` (aproximadamente 3.14159265...), uma constante para proporção de circunferência de um círculo em relação ao seu diâmetro. Ela também contém métodos para funções de trigonometria, incluindo `Math.sin()`, `Math.cos()` e `Math.tan()` entre outros. Desenhar formas utilizando esses métodos e constantes cria efeitos visuais mais dinâmicos, especialmente quando utilizados com repetição ou recursão.

Muitos métodos da classe Math esperam medidas circulares em unidades de radianos em vez de graus. A conversão entre esses dois tipos de unidades é um uso comum da classe Math:

```
var degrees = 121;
var radians = degrees * Math.PI / 180;
trace(radians) // 2.111848394913139
```

O exemplo a seguir cria uma onda senoidal e uma cosenoidal entre os métodos `Math.sin()` e `Math.cos()` para um determinado valor.

```
var sinWavePosition = 100;
var cosWavePosition = 200;
var sinWaveColor:uint = 0xFF0000;
var cosWaveColor:uint = 0x00FF00;
var waveMultiplier:Number = 10;
var waveStretcher:Number = 5;

var i:uint;
for(i = 1; i < stage.stageWidth; i++)
{
    var sinPosY:Number = Math.sin(i / waveStretcher) * waveMultiplier;
    var cosPosY:Number = Math.cos(i / waveStretcher) * waveMultiplier;

    graphics.beginFill(sinWaveColor);
    graphics.drawRect(i, sinWavePosition + sinPosY, 2, 2);
    graphics.beginFill(cosWaveColor);
    graphics.drawRect(i, cosWavePosition + cosPosY, 2, 2);
}
```

Animação com a API de desenho

Uma vantagem da criação de conteúdo com a API de desenho é que você não está limitado a posicionar o conteúdo apenas uma vez. O que você desenha pode ser modificado pela manutenção e modificação das variáveis que usa ao desenhar. Você pode transmitir a animação alterando as variáveis e redesenhando, em um período de quadros ou com um temporizador.

Por exemplo, o código a seguir altera a exibição com cada quadro transmitido (atendendo ao evento `Event.ENTER_FRAME`), incrementando a contagem de graus atual, direciona o objeto graphics para limpar e redesenhar na posição atualizada.

```
stage.frameRate = 31;

var currentDegrees:Number = 0;
var radius:Number = 40;
var satelliteRadius:Number = 6;

var container:Sprite = new Sprite();
container.x = stage.stageWidth / 2;
container.y = stage.stageHeight / 2;
addChild(container);
var satellite:Shape = new Shape();
container.addChild(satellite);

addEventListener(Event.ENTER_FRAME, doEveryFrame);

function doEveryFrame(event:Event):void
{
    currentDegrees += 4;
    var radians:Number = getRadians(currentDegrees);
    var posX:Number = Math.sin(radians) * radius;
    var posY:Number = Math.cos(radians) * radius;
    satellite.graphics.clear();
    satellite.graphics.beginFill(0);
    satellite.graphics.drawCircle(posX, posY, satelliteRadius);
}
function getRadians(degrees:Number):Number
{
return degrees * Math.PI / 180;
}
```

Para produzir um resultado significativamente diferente, você pode modificar as variáveis base iniciais no início do código, `currentDegrees`, `radius` e`satelliteRadius`. Por exemplo, tente reduzir a variável radius e/ou aumentar a variável totalSatellites. Esse é apenas um exemplo de como a API de desenho pode criar uma exibição visual cuja complexidade oculta a simplicidade de sua criação.

Exemplo: Gerador visual algorítmico

O exemplo do Gerador visual algorítmico desenha dinamicamente no palco vários "satélites" ou círculos que se movem em uma órbita circular. Entre os recursos explorados estão:

- Uso da API de desenho para desenhar uma forma básica com aparências dinâmicas
- Conexão da interação do usuário com as propriedades utilizadas em um desenho

- Transmissão de animação por meio da limpeza do palco em cada quadro e redesenho

O exemplo na subseção anterior animou um "satélite" solitário utilizando o evento `Event.ENTER_FRAME`. Esse exemplo vai além disso, criando um painel de controle com séries de controles deslizantes que atualizam imediatamente a exibição visual de vários satélites. Esse exemplo formaliza o código em classes externas e agrupa o código de criação de satélites em um loop, armazenando uma referência para cada satélite em uma matriz `satellites`.

Para obter os arquivos de aplicativo deste exemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_br. Os arquivos do aplicativo podem ser encontrados na pasta Amostras/AlgorithmicVisualGenerator. Essa pasta contém os seguintes arquivos:

Arquivo	Descrição
AlgorithmicVisualGenerator.fla	O arquivo principal do aplicativo no Flash (FLA).
com/example/programmingas3/algorithmic/AlgorithmicVisualGenerator.as	A classe que fornece a funcionalidade principal do aplicativo, que inclui desenho de satélites no palco e resposta aos eventos do painel de controle para atualizar as variáveis que afetam o desenho dos satélites.
com/example/programmingas3/algorithmic/ControlPanel.as	A classe que gerencia a interação do usuário com vários controles deslizantes e eventos de despacho quando eles ocorrem.
com/example/programmingas3/algorithmic/Satellite.as	Uma classe que representa o objeto display que gira em uma órbita ao redor de um ponto central e contém propriedades relacionadas ao estados atual do seu desenho.

Definição de ouvintes

O aplicativo primeiro cria três ouvintes. O primeiro atende um evento despachado do painel de controle informando que é necessário recriar os satélites. O segundo atende às alterações de tamanho do palco do arquivo SWF. O terceiro atende a cada transmissão de quadro no arquivo SWF e para redesenhar utilizando a função `doEveryFrame()`.

Criação de satélites

Assim que os ouvintes estiverem definidos, a função`build()` é chamada. Essa função chama primeiro a função `clear()`, que esvazia a matriz `satellites` e apaga qualquer desenho anterior no palco. Isso é necessário pois a função `build()` pode ser chamada novamente sempre que o painel de controle envia um evento para isso, como quando as configurações de cor são alteradas. Nesse caso, os satélites devem ser removidos e recriados.

A função então cria os satélites, configurando as propriedades iniciais necessárias para a criação, como a variável `position`, que começa em uma posição aleatória na órbita, e a variável `color`, que neste exemplo não é alterada pois o satélite foi criado.

Como cada satélite é criado, uma referência a ele é incluída na matriz `satellites`. Quando a função `doEveryFrame()` é chamada, ela será atualizada para todos os satélites nessa matriz.

Atualização da posição do satélite

A função `doEveryFrame()` é o coração do processo de animação do aplicativo. Ela é chamada para cada quadro, em uma taxa igual à taxa de quadro do arquivo SWF. As pequenas alterações das variáveis do desenho transmitem a aparência da animação.

A função primeiro apaga todos os desenhos anteriores e redesenha o plano de fundo. Em seguida, ela percorre cada contêiner de satélite, incrementa a propriedade `position` de cada satélite e atualiza as propriedades `radius` e `orbitRadius` que podem ter sido alteradas na interação do usuário com o painel de controle. Por fim, o satélite atualiza-se para a sua nova posição chamando o método `draw()` da classe Satellite.

Observe que o contador, `i`, incrementa apenas até a variável `visibleSatellites`. Isso porque se o usuário limitar a quantidade de satélites que são exibidos no painel de controle, os satélites restantes no loop não deverão ser redesenhados, mas deverão ficar ocultos. Isso ocorre em um loop que segue imediatamente o loop responsável pelo desenho.

Quando a função doEveryFrame() é concluída, o número de `visibleSatellites` é atualizado na posição na tela.

Resposta à interação do usuário

A interação do usuário ocorre por meio do painel de controle, que é gerenciado pela classe ControlPanel. Essa classe define um ouvinte junto com os valores individuais mínimo, máximo e padrão de cada controle deslizante. À medida que o usuário move esses controles, a função `changeSetting()` é chamada. Essa função atualiza as propriedades do painel de controle. Se a alteração exigir a criação da exibição, um evento é despachado e então tratado no arquivo principal do aplicativo. À medida que as configurações do painel de controle são alteradas, a função `doEveryFrame()` desenha cada satélite com as variáveis atualizadas.

Personalização posterior

Esse exemplo é apenas um esquema básico de como gerar visuais utilizando a API de desenho. Ele usa relativamente poucas linhas de código para criar uma experiência interativa que parece muito complexa. Apesar disso, esse exemplo pode ser estendido com pequenas alterações Algumas idéias:

- A função `doEveryFrame()` pode incrementar o valor de cor do satélite.
- A função `doEveryFrame()` pode reduzir ou expandir o raio do satélite com o tempo.
- O raio do satélite não precisa ser circular; ele pode usar a classe Math para se mover de acordo com uma onda senoidal, por exemplo.
- Os satélites podem usar a detecção de pressionamento com outros satélites.

A API de desenho pode ser utilizada como uma alternativa para criar efeitos visuais no ambiente de autoria do Flash, desenhando formas básicas no tempo de execução. Mas ela também pode ser utilizada para criar efeitos visuais diversos e com escopo diferente que não podem ser criados manualmente. Com a utilização da API de desenho e um pouco de matemática, o autor do ActionScript pode dar vida a várias criações inesperadas.

Uso avançado da API de desenho

Introdução ao uso da API de desenho avançada

O Flash Player 10 introduz suporte para um conjunto avançado de recursos de desenho. A nova API de desenho expande os métodos de desenho de versões anteriores, por isso é possível estabelecer conjuntos de dados para gerar formas, alterar formas em tempo de execução e criar efeitos tridimensionais. A nova API de desenho consolida métodos existentes em novos comandos. Os novos comandos utilizam matrizes de vetor e classes de enumeração para fornecer conjuntos de dados aos métodos de desenho. O uso de matrizes de vetor permite que formas mais complexas sejam renderizadas rapidamente e que os desenvolvedores alterem os valores de matriz de modo programático para a renderização dinâmica de formas em tempo de execução.

Os recursos de desenho introduzidos no Flash Player 10 estão descritos nas seguintes seções: "Caminhos de desenho" na página 335, "Definição de regras de contorno" na página 337, "Uso de classes de dados gráficos" na página 339 e "Sobre o uso de drawTriangles()" na página 341.

Tarefas comuns da API de desenho avançada

As seguintes tarefas são operações que provavelmente você executará usando a API de desenho avançada do ActionScript:

- Uso de objetos Vector para armazenar dados para métodos de desenho
- Definição de caminhos para desenhar formas de modo programático
- Definição de regras de contorno para determinar como preencher formas sobrepostas
- Uso de classes de dados gráficos
- Uso de triângulos e métodos de desenho para efeitos tridimensionais

Conceitos e termos importantes

A lista de referência a seguir contém termos importantes usados nesta seção:

- Vector: uma matriz de valores, todos com o mesmo tipo de dados. Um objeto Vector pode armazenar uma matriz de valores que os métodos de desenho usam para construir linhas e formas com um único comando. Para obter mais informações sobre objetos Vector, consulte "Matrizes indexadas" na página 159.
- Caminho: um caminho é formado por um ou mais segmentos retos ou curvos. O início e o final de cada segmento são marcados por coordenadas, que funcionam como alfinetes que prendem um esboço. Um caminho pode ser fechado (por exemplo, um círculo) ou aberto, com extremidades distintas (como uma linha ondulada).
- Contorno: a direção de um caminho conforme interpretada pelo renderizador, seja positiva (sentido horário) ou negativa (sentido anti-horário).
- GraphicsStroke: classe usada para definir o estilo da linha. Embora o termo "traçado" não seja novo no ActionScript, o uso de uma classe para designar um estilo de linha com sua propriedade de preenchimento é novo no ActionScript. É possível ajustar o estilo de uma linha dinamicamente usando a classe GraphicsStroke.
- Objeto Fill: objetos criados usando novas classes de exibição, como flash.display.GraphicsBitmapFill e flash.display.GraphicsGradientFill, que são passadas para o comando de desenho Graphics.drawGraphicsData(). Os objetos Fill e os novos comandos de desenho introduzem uma abordagem de programação mais orientada a objetos para replicar `Graphics.beginBitmapFill()` e `Graphics.beginGradientFill()`.

Caminhos de desenho

A seção sobre como desenhar linhas e curvas (consulte"Desenho de linhas e curvas" na página 324) apresentou os comandos usados para desenhar uma única linha (`Graphics.lineTo()`) ou curva (`Graphics.curveTo()`) e mover a linha até outro ponto (`Graphics.moveTo()`) para obter uma forma. O Flash Player 10 introduz suporte para novas APIs de desenho do ActionScript, como `Graphics.drawPath()` e `Graphics.drawTriangles()`, que utilizam os comandos de desenho existentes como parâmetros. Por isso, uma série de comandos `Graphics.lineTo()`, `Graphics.curveTo()` ou `Graphics.moveTo()` são executados em uma única instrução.

Duas novas inclusões feitas na API de desenho permitem que `Graphics.drawPath()` e `Graphics.drawTriangles()` consolidem os comandos existentes:

- A classe de enumeração GraphicsPathCommand: a classe GraphicsPathCommand associa vários comandos de desenho a valores de constante. Use uma série desses valores como parâmetros para o método `Graphics.drawPath()`. Em seguida, com um único comando, você pode renderizar a forma inteira ou várias formas. Também é possível alterar dinamicamente os valores passados para esses métodos a fim de alterar uma forma existente.
- Matrizes de vetor: as matrizes de vetor contêm uma série de valores de um tipo de dados específico. Assim, você pode armazenar um grupo de constantes GraphicsPathCommand em um objeto Vector e uma série de coordenadas em outro objeto Vector. `Graphics.drawPath()` ou `Graphics.drawTriangles()` atribui esses valores juntos para gerar um caminho de desenho ou uma forma.

Não é mais preciso separar comandos para cada segmento de uma forma. Por exemplo, o método `Graphics.drawPath()` consolida `Graphics.moveTo()`, `Graphics.lineTo()` e `Graphics.curveTo()` em um único método. Em vez de cada método ser chamado separadamente, eles são abstraídos em identificadores numéricos, conforme definido na classe GraphicsPathCommand. Uma operação `moveTo()` é representada por um `1`, enquanto uma operação `lineTo()` é um `2`. Armazene uma matriz desses valores em um objeto Vector.<int> para uso no parâmetro `commands`. Em seguida, crie outra matriz que contenha coordenadas em um objeto Vector.<Number> para o parâmetro `data`. Cada valor GraphicsPathCommand corresponde aos valores de coordenada armazenados no parâmetro de dados em que dois números consecutivos definem um local no espaço de coordenadas de destino.

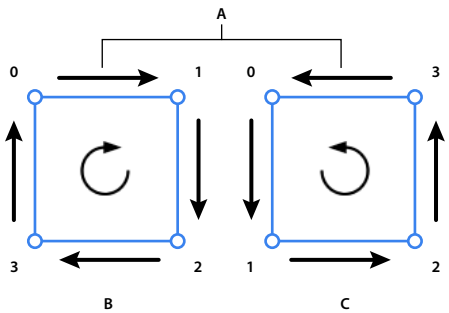
Nota: *Os valores do vetor não são objetos Point; o vetor consiste em uma série de números em que cada grupo de dois números representa um par de coordenadas x/y.*

O método `Graphics.drawPath()` compara cada comando com seus respectivos valores de ponto (uma coleção de dois ou quatro números) para gerar um caminho no objeto Graphics:

```
package{
import flash.display.*;

public class DrawPathExample extends Sprite {
    public function DrawPathExample(){

    var square_commands:Vector.<int> = new Vector.<int>(5,true);
    square_commands[0] = 1;//moveTo
    square_commands[1] = 2;//lineTo
    square_commands[2] = 2;
    square_commands[3] = 2;
    square_commands[4] = 2;

    var square_coord:Vector.<Number> = new Vector.<Number>(10,true);
    square_coord[0] = 20; //x
    square_coord[1] = 10; //y
    square_coord[2] = 50;
    square_coord[3] = 10;
    square_coord[4] = 50;
    square_coord[5] = 40;
    square_coord[6] = 20;
    square_coord[7] = 40;
    square_coord[8] = 20;
    square_coord[9] = 10;

    graphics.beginFill(0x442266);//set the color
    graphics.drawPath(square_commands, square_coord);
    }
}
}
```

Definição de regras de contorno

O Flash Player 10 também introduz o conceito de "contorno" de caminho: a direção de um caminho. O contorno de um caminho é tanto positivo (sentido horário) quanto negativo (sentido anti-horário). A ordem em que o renderizador interpreta as coordenadas fornecidas pelo vetor para o parâmetro de dados determina o contorno.



Contorno positivo e negativo
***A.** Setas indicam a direção do desenho **B.** Rotação positiva (sentido horário) **C.** Rotação negativa (sentido anti-horário)*

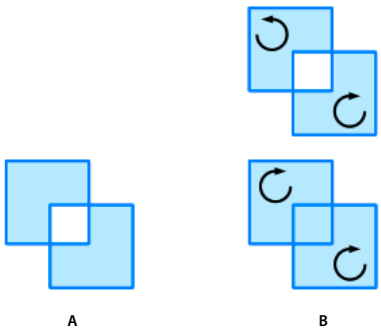
Além disso, observe que o método `Graphics.drawPath()` tem um terceiro parâmetro opcional, chamado "winding":

```
drawPath(commands:Vector.<int>, data:Vector.<Number>, winding:String = "evenOdd"):void
```

Neste contexto, o terceiro parâmetro é uma string ou uma constante que especifica a regra de contorno ou de preenchimento para intersecção de caminhos. (Os valores de constante são definidos na classe GraphicsPathWinding como `GraphicsPathWinding.EVEN_ODD` ou `GraphicsPathWinding.NON_ZERO`.) A regra de contorno é importante quando ocorre intersecção de caminhos.

A regra par-ímpar é a regra de contorno padrão, sendo utilizada por todas as APIs de desenho anteriores ao Flash Player 10. Par-ímpar também é a regra padrão para o método `Graphics.drawPath()`. Com a regra de contorno par-ímpar, qualquer caminho de intersecção alterna entre preenchimentos abertos e fechados. Se houver intersecção de dois quadrados desenhados com o mesmo preenchimento, a área em que ocorre a intersecção será preenchida. Geralmente, as áreas adjacentes não são ambas preenchidas nem ambas não preenchidas.

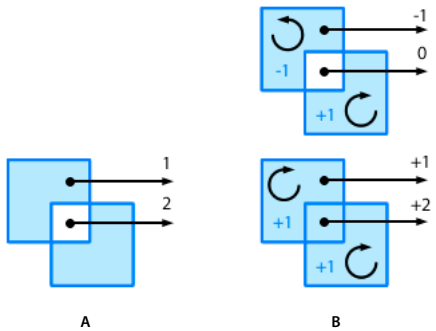
A regra diferente de zero, por outro lado, depende do contorno (direção do desenho) para determinar se áreas definidas por caminhos de intersecção são preenchidas. Quando caminhos de contorno opostos se cruzam, a área definida não é preenchida, bem parecido com o que ocorre na regra par-ímpar. Para caminhos com o mesmo contorno, a área que não seria preenchida é preenchida:



Regras de contorno para áreas de intersecção
***A.** Regra de contorno par-ímpar **B.** Regra de contorno diferente de zero*

Nomes de regras de contorno

Os nomes referem-se a uma regra mais específica que define como esses preenchimentos são gerenciados. Caminhos de rotação positiva recebem um valor de +1; caminhos de rotação negativa recebem um valor de -1. Começando em um ponto dentro de uma área fechada de uma forma, desenhe uma linha a partir desse ponto que se estenda indefinidamente. O número de vezes que a linha cruza um caminho e os valores combinados desses caminhos são usados para determinar o preenchimento. Para contorno par-ímpar, é usado o número de vezes em que a linha cruza um caminho. Quando a contagem é ímpar, a área é preenchida. Para contagens pares, a área não é preenchida. Para contorno diferente de zero, são usados os valores atribuídos aos caminhos. Quando os valores combinados do caminho não são 0, a área é preenchida. Quando os valores combinados são 0, a área não é preenchida.



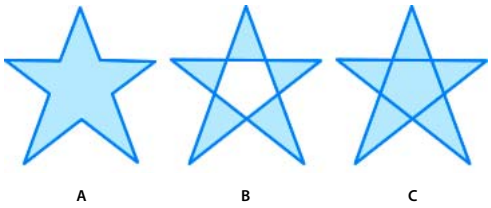
Preenchimentos e contagens de regras de contorno
***A.** Regra de contorno par-ímpar **B.** Regra de contorno diferente de zero*

Uso de regras de contorno

Essas regras de preenchimento são complicadas, mas em algumas situações elas são necessárias. Por exemplo, pense no desenho de uma forma de estrela. Com a regra par-ímpar padrão, a forma exigiria dez linhas diferentes. Com a regra de contorno diferente de zero, essas dez linhas são reduzidas a cinco. Este é o ActionScript de uma estrela com cinco linhas e uma regra de contorno diferente de zero:

```
fill.graphics.beginFill(0x60A0FF);graphics.drawPath( Vector.<int>([1,2,2,2,2]),
Vector.<Number>([66,10, 23,127, 122,50, 10,49, 109,127]),   GraphicsPathWinding.NON_ZERO);
```

E esta é a forma de estrela:



Uma forma de estrela usando regras de contorno diferente
***A.** 10 linhas pares-ímpares **B.** 5 linhas pares-ímpares **C.** 5 linhas diferentes de zero*

E, à medida que imagens são animadas ou usadas como texturas em objetos tridimensionais e se sobrepõem, as regras de contorno tornam-se mais importantes.

Uso de classes de dados gráficos

O Flash Player 10 introduz uma nova coleção de classes no pacote flash.display do tipo IGraphicsData (uma interface implementada por cada uma das classes). As classes que implementam a interface IGraphicsData funcionam como contêineres de dados para os métodos da API de desenho.

As seguintes classes implementam a interface IGraphicsData:

- GraphicsBitmapFill
- GraphicsEndFill
- GraphicsGradientFill
- GraphicsPath
- GraphicsShaderFill
- GraphicsSolidFill
- GraphicsStroke
- GraphicsTrianglePath

Com elas, é possível armazenar desenhos completos em uma matriz de objeto Vector do tipo IGraphicsData (Vector.<IGraphicsData>) que pode ser reutilizada como fonte de dados para outras ocorrências da forma ou para armazenar informações para uso posterior.

Observe que existem várias classes de preenchimento para cada estilo de preenchimento, mas somente uma classe de traçado. O ActionScript tem apenas uma classe de traçado IGraphicsData, pois a classe de traçado usa as classes de preenchimento para definir seu estilo. Assim, cada traçado é, na verdade, a classe de traçado e uma classe de preenchimento. Do contrário, a API dessas classes de dados gráficos espelha os métodos que elas representam na classe flash.display.Graphics:

Método de Graphics	Classe de dados
beginBitmapFill()	GraphicsBitmapFill
beginFill()	GraphicsSolidFill
beginGradientFill()	GraphicsGradientFill
beginShaderFill()	GraphicsShaderFill
lineBitmapStyle()	GraphicsStroke + GraphicsBitmapFill
lineGradientStyle()	GraphicsStroke + GraphicsGradientFill
lineShaderStyle()	GraphicsStroke + GraphicsShaderFill
lineStyle()	GraphicsStroke + GraphicsSolidFill
moveTo() lineTo() curveTo() drawPath()	GraphicsPath
drawTriangles()	GraphicsTrianglePath

Além disso, a classe GraphicsPath tem seus próprios métodos de utilitário `GraphicsPath.moveTo()`, `GraphicsPath.lineTo()`, `GraphicsPath.curveTo()`, `GraphicsPath.wideLineTo()` e `GraphicsPath.wideMoveTo()` que facilitam a definição desses comandos para uma ocorrência de GraphicsPath. Esses métodos de utilitário facilitam a definição ou a atualização direta dos comandos e valores de dados.

Uma vez que você tem uma coleção de ocorrências de IGraphicsData, use o método `Graphics.drawGraphicsData()` para renderizar os gráficos. O método `Graphics.drawGraphicsData()` executa um vetor de ocorrências de IGraphicsData através da API de desenho em ordem seqüencial:

```
// stroke object
var stroke:GraphicsStroke = new GraphicsStroke(3);
stroke.joints = JointStyle.MITER;
stroke.fill = new GraphicsSolidFill(0x102020);// solid stroke

// fill object
var fill:GraphicsGradientFill = new GraphicsGradientFill();
fill.colors = [0x0000FF, 0xEEFFEE];
fill.matrix = new Matrix();
fill.matrix.createGradientBox(70,70, Math.PI/2);
// path object
var path:GraphicsPath = new GraphicsPath(new Vector.<int>(), new Vector.<Number>());
path.commands.push(1,2,2);
path.data.push(125,0, 50,100, 175,0);

// combine objects for complete drawing
var drawing:Vector.<IGraphicsData> = new Vector.<IGraphicsData>();
drawing.push(stroke, fill, path);

// draw the drawing
graphics.drawGraphicsData(drawing);
```

Modificando-se um valor no caminho usado pelo desenho do exemplo, é possível redesenhar a forma várias vezes para uma imagem mais complexa:

```
// draw the drawing multiple times
// change one value to modify each variation
graphics.drawGraphicsData(drawing);
path.data[2] += 200;
graphics.drawGraphicsData(drawing);
path.data[2] -= 150;
graphics.drawGraphicsData(drawing);
path.data[2] += 100;
graphics.drawGraphicsData(drawing);
path.data[2] -= 50;graphicsS.drawGraphicsData(drawing);
```

Embora objetos IGraphicsData possam definir estilos de preenchimento e de traçado, esses estilos não são obrigatórios. Em outras palavras, os métodos da classe Graphics podem ser usados para definir estilos, enquanto os objetos IGraphicsData podem ser usados para desenhar uma coleção de caminhos salvos ou vice-versa.

Nota: *Use o método Graphics.clear() para remover um desenho anterior antes de começar um novo, a menos que você esteja aumentando o desenho original, como visto no exemplo acima. Quando alterar uma única parte de um caminho ou de uma coleção de objetos IGraphicsData, redesenhe o desenho inteiro para ver as alterações.*

Quando são usadas classes de dados gráficos, o preenchimento é renderizado sempre que três ou mais pontos são desenhados porque a forma é inerentemente fechada nesse ponto. Embora o preenchimento seja fechado, o traçado não é, e esse comportamento é diferente de quando são usados vários comandos `Graphics.lineTo()` ou `Graphics.moveTo()`.

Sobre o uso de drawTriangles()

Outro método avançado incluído no Flash Player 10, `Graphics.drawTriangles()`, é parecido com o método `Graphics.drawPath()`. O método Graphics.drawTriangles() também usa um objeto Vector.<Number> para especificar localizações de pontos para desenhar um caminho.

No entanto, a verdadeira finalidade do método `Graphics.drawTriangles()` é facilitar os efeitos tridimensionais através do ActionScript. Para obter informações sobre como usar `Graphics.drawTriangles()` para produzir efeitos tridimensionais, consulte "Uso de triângulos para obter efeitos 3D" na página 519.

Capítulo 15: Trabalho com geometria

O pacote flash.geom contém classes que definem objetos geométricos como pontos, retângulos e matrizes de transformação. Essas classes são usadas para definir as propriedades dos objetos utilizados em outras classes.

Noções básicas de geometria

Introdução ao trabalho com geometria

A geometria, em geral, é uma matéria escolar em que as pessoas querem ser aprovadas aprendendo o mínimo possível, mas um pouco de conhecimento no assunto pode ser muito útil durante o uso do ActionScript.

O pacote flash.geom contém classes que definem objetos geométricos como pontos, retângulos e matrizes de transformação. Essas classes necessariamente não fornecem uma funcionalidade em si, mas são usadas para definir as propriedades dos objetos utilizados em outras classes.

Todas as classes de geometria giram em torno da noção de que os locais na tela são representados como um plano bidimensional. A tela é tratada como um gráfico plano com um eixo horizontal (x) e um vertical (y). Qualquer local (ou *ponto*) da tela pode ser representado como um par de valores x e y — as *coordenadas* desse local.

Cada objeto de exibição, incluindo o Palco, possui seu próprio *espaço de coordenadas* — basicamente seu próprio gráfico para marcar os locais de objetos de exibição-filho, desenhos etc. Em geral, a *origem* (o local com a coordenada 0, 0 em que os eixos x e y se encontram) é colocada no canto esquerdo superior do objeto de exibição. Embora isso seja sempre válido para o Palco, não é necessariamente válido para qualquer outro objeto de exibição. Assim como nos sistemas de coordenadas bidimensionais, os valores no eixo x aumentam para a direita e diminuem para a esquerda; para os locais à esquerda da origem, a coordenada x é negativa. Entretanto, diferentemente dos sistemas de coordenadas tradicionais, no ActionScript, os valores no eixo y aumentam para baixo e diminuem para cima (os valores acima da origem têm uma coordenada y negativa). Como o canto esquerdo superior do Palco é a origem de seu espaço de coordenadas, qualquer objeto nele terá uma coordenada x maior do que 0 e menor do que a largura do Palco, e terá uma coordenada y maior do que 0 e menor do que a altura do Palco.

Você pode usar ocorrências da classe Point para representar pontos individuais em um espaço de coordenadas. É possível criar uma ocorrência de Rectangle para representar uma região retangular em um espaço de coordenadas. Para usuários avançados, você pode usar uma ocorrência de Matrix para aplicar várias transformações ou transformações complexas em um objeto de exibição. Muitas transformações simples, como rotação, posição e alterações de escala, podem ser aplicadas diretamente a um objeto de exibição usando as propriedades desse objeto. Para obter informações sobre como aplicar transformações usando propriedades de objeto de exibição, consulte “Manipulação de objetos de exibição” na página 293.

Tarefas comuns de geometria

As seguintes tarefas são as mais prováveis de ser executadas usando classes de geometria no ActionScript:

- Calcular a distância entre dois pontos
- Determinar coordenadas de um ponto em espaços de coordenadas diferentes
- Mover um objeto de exibição usando ângulo e distância

- Trabalhar com ocorrências de Rectangle:
  - Reposicionar uma ocorrência de Rectangle
  - Redimensionar uma ocorrência de Rectangle
  - Determinar áreas combinadas de tamanho ou sobreposição de ocorrências de Rectangle
- Criar objetos Matrix
- Usar um objeto Matrix para aplicar transformações em um objeto de exibição

Conceitos e termos importantes

A lista de referência a seguir contém termos importantes usados neste capítulo:

- Coordenadas cartesianas: coordenas comumente escritas como um par de números (como 5, 12 ou 17, -23). Os dois números são as coordenadas x e y, respectivamente.
- Espaço de coordenadas: o gráfico de coordenadas contido em um objeto de exibição, no qual seus elementos-filho são posicionados.
- Origem: o ponto em um espaço de coordenadas no qual o eixo x encontra o eixo y. Esse ponto tem a coordenada 0,0.
- Ponto: um local isolado em um espaço de coordenadas. No sistema de coordenadas bidimensional usado no ActionScript, um ponto é definido por seu local no eixo x e no eixo y (as coordenadas do ponto).
- Ponto de registro: em um objeto de exibição, a origem (coordenada 0,0) de seu espaço coordenado.
- Escala: o tamanho de um objeto em relação ao seu tamanho original. Quando usado como um verbo, dimensionar um objeto significa alterar seu tamanho, aumentando ou diminuindo o objeto.
- Transpor: alterar as coordenadas de um ponto de um espaço de coordenadas para outro.
- Transformação: ajuste de uma característica visual de um gráfico como a rotação do objeto, a alteração da escala, a inclinação ou distorção da forma ou a alteração da cor.
- Eixo X: o eixo horizontal no sistema de coordenadas bidimensional usado no ActionScript.
- Eixo Y: o eixo vertical no sistema de coordenadas bidimensional usado no ActionScript.

Teste dos exemplos do capítulo

Muitos exemplos deste capítulo demonstram cálculos ou valores de alteração; a maioria deles inclui as chamadas da função `trace()` apropriadas para demonstrar os resultados do código. Para testar esses exemplos, faça o seguinte:

1. Crie um documento vazio usando a ferramenta de autoria do Flash.
2. Selecione um quadro-chave na Linha de tempo.
3. Abra o painel Ações e copie a listagem de código no painel Script.
4. Execute o programa usando Controlar > Testar filme.

   Você verá os resultados das funções `trace()` da listagem de código no painel Saída.

Alguns exemplos do capítulo demonstram a aplicação de transformações em objetos de exibição. Para esses exemplos, os resultados do exemplo serão verificados visualmente e não por meio de saída de texto. Para testar esses exemplos de transformações, faça o seguinte:

1. Crie um documento vazio usando a ferramenta de autoria do Flash.
2. Selecione um quadro-chave na Linha de tempo.
3. Abra o painel Ações e copie a listagem de código no painel Script.

4 Crie uma ocorrência de símbolo de clipe de filme no Palco. Por exemplo, desenhe uma forma, selecione-a, escolha Modificar > Converter em símbolo e dê um nome ao símbolo.

5 Com o clipe de filme do Palco selecionado, no Inspetor de propriedades, dê um nome de ocorrência à ocorrência. O nome deve corresponder ao nome usado para o objeto de exibição na listagem de código de exemplo — por exemplo, se a listagem de código aplicar uma transformação a um objeto chamado `myDisplayObject`, o nome da ocorrência do clipe de filme também deverá ser `myDisplayObject`.

6 Execute o programa usando Controlar > Testar filme.

   Na tela, você verá os resultados das transformações aplicadas ao objeto conforme especificado na listagem de código.

As técnicas para testar listagens de código de exemplo são detalhadas em "Teste de listagens de código de exemplo dos capítulos" na página 36.

Uso de objetos Point

Um objeto Point define um par cartesiano de coordenadas. Ele representa um local em um sistema de coordenadas bidimensional, em que *x* representa o eixo horizontal e *y*, o vertical.

Para definir um objeto Point, defina as propriedades de `x` e `y` da seguinte forma:

```
import flash.geom.*;
var pt1:Point = new Point(10, 20); // x == 10; y == 20
var pt2:Point = new Point();
pt2.x = 10;
pt2.y = 20;
```

Cálculo da distância entre dois pontos

Você pode usar o método `distance()` da classe Point para calcular a distância entre dois pontos em um espaço de coordenadas. Por exemplo, o código a seguir calcula a distância entre os pontos de registro de dois objetos de exibição, `circle1` e `circle2`, no mesmo contêiner de objetos de exibição:

```
import flash.geom.*;
var pt1:Point = new Point(circle1.x, circle1.y);
var pt2:Point = new Point(circle2.x, circle2.y);
var distance:Number = Point.distance(pt1, pt2);
```

Transposição de espaços de coordenadas

Se dois objetos de exibição estiverem em contêineres de objetos de exibição diferentes, talvez estejam em espaços de coordenadas diferentes. Você pode usar o método `localToGlobal()` da classe DisplayObject para transpor as coordenadas para o mesmo espaço de coordenadas (global) que o Palco. Por exemplo, o código a seguir calcula a distância entre os pontos de registro de dois objetos de exibição, `circle1` e `circle2`, em contêineres de objetos de exibição diferentes:

```
import flash.geom.*;
var pt1:Point = new Point(circle1.x, circle1.y);
pt1 = circle1.localToGlobal(pt1);
var pt2:Point = new Point(circle2.x, circle2.y);
pt2 = circle2.localToGlobal(pt2);
var distance:Number = Point.distance(pt1, pt2);
```

Da mesma forma, para calcular a distância do ponto de registro de um objeto de exibição chamado `target` a partir de um ponto específico no Palco, você pode usar o método `localToGlobal()` da classe DisplayObject:

```
import flash.geom.*;
var stageCenter:Point = new Point();
stageCenter.x = this.stage.stageWidth / 2;
stageCenter.y = this.stage.stageHeight / 2;
var targetCenter:Point = new Point(target.x, target.y);
targetCenter = target.localToGlobal(targetCenter);
var distance:Number = Point.distance(stageCenter, targetCenter);
```

Mover um objeto de exibição usando um ângulo e uma distância especificados

Você pode usar o método `polar()` da classe Point para mover um objeto de exibição a uma distância específica em um ângulo específico. Por exemplo, o código a seguir move o objeto `myDisplayObject` 100 pixels em 60 graus:

```
import flash.geom.*;
var distance:Number = 100;
var angle:Number = 2 * Math.PI * (90 / 360);
var translatePoint:Point = Point.polar(distance, angle);
myDisplayObject.x += translatePoint.x;
myDisplayObject.y += translatePoint.y;
```

Outros usos da classe Point

É possível usar os objetos Point com os seguintes métodos e propriedades:

Classe	Métodos ou propriedades	Descrição
DisplayObjectContainer	`areInaccessibleObjectsUnderPoint()getObjectsUnderPoint()`	Usada para retornar uma lista de objetos sob um ponto em um contêiner de objetos de exibição.
BitmapData	`hitTest()`	Usada para definir o pixel no objeto BitmapData bem como o ponto cuja ocorrência você está verificando.

BitmapData	`applyFilter()`
	`copyChannel()`
	`merge()`
	`paletteMap()`
	`pixelDissolve()`
	`threshold()`
		Usada para definir as posições de retângulos que definem as operações.

Matrix	`deltaTransformPoint()` `transformPoint()`	Usada para definir os pontos para os quais você deseja aplicar uma transformação.

Retângulo	`bottomRight`
	`size`
	`topLeft`
		Usada para definir essas propriedades.

Uso de objetos Rectangle

Um objeto Rectangle define uma área retangular. Um objeto Rectangle tem uma posição, definida pelas coordenadas *x* e *y* de seu canto superior esquerdo, uma propriedade `width` e uma propriedade `height` . Você pode definir essas propriedades para um novo objeto Rectangle chamando a função de construtor `Rectangle()` da seguinte forma:

```
import flash.geom.Rectangle;
var rx:Number = 0;
var ry:Number = 0;
var rwidth:Number = 100;
var rheight:Number = 50;
var rect1:Rectangle = new Rectangle(rx, ry, rwidth, rheight);
```

Redimensionamento e reposicionamento de objetos Rectangle

Há várias formas de redimensionar e reposicionar objetos Rectangle.

Você pode reposicionar o objeto Rectangle alterando suas propriedades `x` e `y`. Isso não afeta a largura ou a altura do objeto Rectangle.

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.x = 20;
rect1.y = 30;
trace(rect1); // (x=20, y=30, w=100, h=50)
```

Como mostra o código a seguir, se você alterar a propriedade `left` ou `top` de um objeto Rectangle, ele também é reposicionado, com suas propriedades `x` e `y` correspondendo às propriedades `left` e `top`, respectivamente. Entretanto, a posição do canto inferior esquerdo do objeto Rectangle não é alterada, por isso ele é redimensionado.

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.left = 20;
rect1.top = 30;
trace(rect1); // (x=20, y=30, w=80, h=20)
```

Da mesma forma, como mostra o exemplo, se você alterar a propriedade `bottom` ou `right` de um objeto Rectangle, a posição de seu canto superior esquerdo não é alterada, por isso ele é redimensionado de acordo:

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.right = 60;
trect1.bottom = 20;
trace(rect1); // (x=0, y=0, w=60, h=20)
```

Você também pode reposicionar um objeto Rectangle usando o método `offset()` da seguinte maneira:

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.offset(20, 30);
trace(rect1); // (x=20, y=30, w=100, h=50)
```

O método `offsetPt()` funciona da mesma forma, exceto que assume um objeto Point como parâmetro, em vez dos valores de deslocamento *x* e *y*.

Você também pode redimensionar um objeto Rectangle usando o método `inflate()`, que inclui dois parâmetros, `dx` e `dy`. O parâmetro `dx` representa o número de pixels que os lados esquerdo e direito do retângulo moverão do centro, e o parâmetro `dy` representa o número de pixels que o lado superior e inferior do retângulo moverão do centro:

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.inflate(6,4);
trace(rect1); // (x=-6, y=-4, w=112, h=58)
```

O método `inflatePt()` funciona da mesma forma, exceto que assume um objeto Point como parâmetro, em vez dos valores de deslocamento `dx` e `dy`.

Localização de uniões e interseções de objetos Rectangle

Você usa o método `union()` para localizar a região retangular formada pelos limites de dois retângulos:

```
import flash.display.*;
import flash.geom.Rectangle;
var rect1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(rect1); // (x=0, y=0, w=100, h=100)
var rect2:Rectangle = new Rectangle(120, 60, 100, 100);
trace(rect2); // (x=120, y=60, w=100, h=100)
trace(rect1.union(rect2)); // (x=0, y=0, w=220, h=160)
```

Você usa o método `intersection()` para localizar a região retangular formada pela região sobreposta de dois retângulos:

```
import flash.display.*;
import flash.geom.Rectangle;
var rect1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(rect1); // (x=0, y=0, w=100, h=100)
var rect2:Rectangle = new Rectangle(80, 60, 100, 100);
trace(rect2); // (x=120, y=60, w=100, h=100)
trace(rect1.intersection(rect2)); // (x=80, y=60, w=20, h=40)
```

O método `intersects()` é usado para descobrir se há interseção de dois retângulos. O método `intersects()` também pode ser usado para descobrir se um objeto de exibição está em uma determinada região do Palco. Por exemplo, no código a seguir, suponha que o espaço de coordenadas do contêiner de objetos de exibição que inclui o objeto `circle` seja o mesmo do Palco. O exemplo mostra como usar o método `intersects()` para determinar se há interseção de um objeto de exibição, `circle`, com regiões especificadas do Palco, definidas pelos objetos Rectangle `target1` e `target2`:

```
import flash.display.*;
import flash.geom.Rectangle;
var circle:Shape = new Shape();
circle.graphics.lineStyle(2, 0xFF0000);
circle.graphics.drawCircle(250, 250, 100);
addChild(circle);
var circleBounds:Rectangle = circle.getBounds(stage);
var target1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(circleBounds.intersects(target1)); // false
var target2:Rectangle = new Rectangle(0, 0, 300, 300);
trace(circleBounds.intersects(target2)); // true
```

Da mesma forma, o método `intersects()` pode ser usado para descobrir se os retângulos delimitadores de dois objetos de exibição se sobrepõem. Você pode usar o método `getRect()` da classe DisplayObject para incluir qualquer espaço adicional que os traçados de um objeto de exibição possam adicionar a uma região delimitadora.

Outros usos de objetos Rectangle

Os objetos Rectangle são usados nos seguintes métodos e propriedades:

Classe	Métodos ou propriedades	Descrição
BitmapData	`applyFilter()`, `colorTransform()`, `copyChannel()`, `copyPixels()`, `draw()`, `fillRect()`, `generateFilterRect()`, `getColorBoundsRect()`, `getPixels()`, `merge()`, `paletteMap()`, `pixelDissolve()`, `setPixels()` e `threshold()`	Usada como o tipo para alguns parâmetros a fim de definir uma região do objeto BitmapData.
DisplayObject	`getBounds()`, `getRect()`, `scrollRect`, `scale9Grid`	Usada como o tipo de dados para a propriedade ou o tipo de dados retornado.
PrintJob	`addPage()`	Usada para definir o parâmetro `printArea`.
Sprite	`startDrag()`	Usada para definir o parâmetro `bounds`.
TextField	`getCharBoundaries()`	Usada como um tipo de valor de retorno.
Transform	`pixelBounds`	Usada como o tipo de dados.

Uso de objetos Matrix

A classe Matrix representa uma matriz de transformação que determina como mapear pontos de um espaço de coordenadas para outro. É possível realizar várias transformações gráficas em um objeto de exibição definindo as propriedades de um objeto Matrix, aplicando esse objeto Matrix à propriedade `matrix` de um objeto Transform, e depois aplicando esse objeto Transform como a propriedade `transform` do objeto de exibição. Essas funções de transformação incluem conversão (reposicionamento de *x* e *y*), rotação, dimensionamento e inclinação.

Definição de objetos Matrix

Embora seja possível definir uma matriz ajustando diretamente as propriedades (`a`, `b`, `c`, `d`, `tx`, `ty`) de um objeto Matrix, é mais fácil usar o método `createBox()`. Esse método inclui parâmetros que permitem definir diretamente os efeitos de dimensionamento, rotação e transposição da matriz resultante. Por exemplo, o código a seguir cria um objeto Matrix que possui o efeito de dimensionar um objeto horizontalmente em 2,0, verticalmente em 3,0, girá-lo em 45 graus, mover (transpor) 10 pixels para a direita e 20 pixels para baixo:

```
var matrix:Matrix = new Matrix();
var scaleX:Number = 2.0;
var scaleY:Number = 3.0;
var rotation:Number = 2 * Math.PI * (45 / 360);
var tx:Number = 10;
var ty:Number = 20;
matrix.createBox(scaleX, scaleY, rotation, tx, ty);
```

Também é possível ajustar os efeitos de dimensionamento, rotação e transposição de um objeto Matrix usando os métodos `scale()`, `rotate()` e `translate()` . Observe que esses métodos combinam com os valores do objeto Matrix existente. Por exemplo, o código a seguir define um objeto Matrix que dimensiona um objeto por um fator de 4 e o gira 60 graus, desde que os métodos `scale()` e `rotate()` sejam chamados duas vezes:

```
var matrix:Matrix = new Matrix();
var rotation:Number = 2 * Math.PI * (30 / 360); // 30°
var scaleFactor:Number = 2;
matrix.scale(scaleFactor, scaleFactor);
matrix.rotate(rotation);
matrix.scale(scaleX, scaleY);
matrix.rotate(rotation);

myDisplayObject.transform.matrix = matrix;
```

Para aplicar uma transformação de inclinação em um objeto Matrix, ajuste sua propriedade `b` ou `c`. O ajuste da propriedade `b` inclina a matriz verticalmente e o ajuste da propriedade`c` inclina a matriz horizontalmente. O código a seguir inclina o objeto Matrix `myMatrix` verticalmente por um fator de 2:

```
var skewMatrix:Matrix = new Matrix();
skewMatrix.b = Math.tan(2);
myMatrix.concat(skewMatrix);
```

Você pode aplicar uma transformação de Matrix à propriedade `transform` de um objeto de exibição. Por exemplo, o código a seguir aplica uma transformação de matriz em um objeto de exibição chamado `myDisplayObject`:

```
var matrix:Matrix = myDisplayObject.transform.matrix;
var scaleFactor:Number = 2;
var rotation:Number = 2 * Math.PI * (60 / 360); // 60°
matrix.scale(scaleFactor, scaleFactor);
matrix.rotate(rotation);

myDisplayObject.transform.matrix = matrix;
```

A primeira linha define um objeto Matrix como a matriz de transformação existente usada pelo objeto de exibição `myDisplayObject` (a propriedade `matrix` da propriedade `transformation` do objeto de exibição `myDisplayObject`). Dessa forma, os métodos da classe Matrix que você chama terão um efeito cumulativo na posição, dimensão e rotação existentes do objeto de exibição.

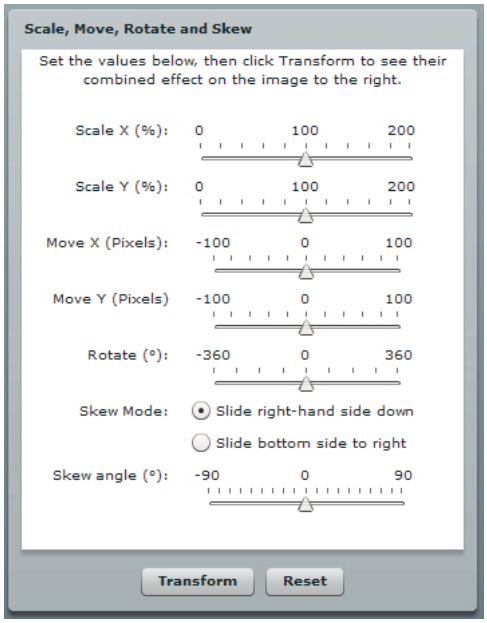
Nota: *A classe ColorTransform também está incluída no pacote flash.geometry. Essa classe é usada para definir a propriedade* `colorTransform` *de um objeto Transform. Como não se aplica a nenhum tipo de transformação geométrica, ela não será discutida neste capítulo. Para obter mais informações, consulte a classe ColorTransform na Referência dos componentes e da linguagem do ActionScript 3.0.*

Exemplo: Aplicação de uma transformação de matriz em um objeto de exibição

O aplicativo de exemplo DisplayObjectTransformer mostra vários recursos de uso da classe Matrix para transformar um objeto de exibição, incluindo o seguinte:

- Girar o objeto de exibição
- Dimensionar o objeto de exibição
- Transpor (reposicionar) o objeto de exibição
- Inclinar o objeto de exibição

O aplicativo fornece uma interface para ajustar os parâmetros de transformação da matriz, como a seguir:



Quando o usuário clica no botão Transformar, o aplicativo aplica a transformação apropriada.

O objeto de exibição original e o objeto de exibição com um giro de - 45° e um dimensionamento de 50%

Para obter os arquivos do aplicativo para este exemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_br. Os arquivos do aplicativo DisplayObjectTransformer podem ser encontrados na pasta Samples/DisplayObjectTransformer. O aplicativo consiste nos seguintes arquivos:

Arquivo	Descrição
DisplayObjectTransformer.mxml ou DisplayObjectTransformer.fla	O arquivo principal do aplicativo no Flash (FLA) ou no Flex (MXML).
com/example/programmingas3/geometry/MatrixTransformer.as	Uma classe que contém métodos para aplicar transformações de matriz.
img/	Um diretório contendo arquivos de imagem de exemplo usados pelo aplicativo.

Definição da classe MatrixTransformer

A classe MatrixTransformer inclui métodos estáticos que aplicam transformações geométricas de objetos Matrix.

O método transform()

O método `transform()` inclui parâmetros para cada um dos itens a seguir:

- `sourceMatrix` — a matriz de entrada, que o método transforma
- `xScale` e `yScale` — o fator de dimensionamento *x* e *y*
- `dx` e `dy` — os valores de transposição *x* e *y*, em pixels
- `rotation` — o valor de rotação, em graus
- `skew` — o fator de inclinação, em porcentagem
- `skewType` — a direção da inclinação `"right"` ou `"left"`

O valor de retorno é a matriz resultante.

O método `transform()` chama os seguintes métodos estáticos da classe:

- `skew()`
- `scale()`
- `translate()`
- `rotate()`

Cada um retorna a matriz de origem com a transformação aplicada.

O método skew()

O método `skew()` inclina a matriz, ajustando as propriedades `b` e `c` da matriz. Um parâmetro opcional, `unit`, determina as unidades usadas para definir o ângulo de inclinação e, se necessário, o método converte o valor `angle` em radianos:

```
if (unit == "degrees")
{
    angle = Math.PI * 2 * angle / 360;
}
if (unit == "gradients")
{
    angle = Math.PI * 2 * angle / 100;
}
```

Um objeto Matrix `skewMatrix` é criado e ajustado para aplicar a transformação de inclinação. Inicialmente, é a matriz de identidade, como a seguir:

```
var skewMatrix:Matrix = new Matrix();
```

O parâmetro `skewSide` determina o lado para o qual a inclinação é aplicada. Se for definida como `"right"`, o seguinte código define a propriedade `b` da matriz:

```
skewMatrix.b = Math.tan(angle);
```

Caso contrário, o lado inferior é inclinado ajustando a propriedade `c` de Matrix, como a seguir:

```
skewMatrix.c = Math.tan(angle);
```

A inclinação resultante é aplicada à matriz existente, concatenando as duas matrizes, como mostra o seguinte exemplo:

```
sourceMatrix.concat(skewMatrix);
return sourceMatrix;
```

O método scale()

Como mostra o exemplo a seguir, o método `scale()` primeiro ajusta o fator de dimensionamento, caso seja fornecido como uma porcentagem, e depois usa o método `scale()` do objeto da matriz:

```
if (percent)
{
    xScale = xScale / 100;
    yScale = yScale / 100;
}
sourceMatrix.scale(xScale, yScale);
return sourceMatrix;
```

O método translate()

O método `translate()` apenas aplica os fatores de transposição `dx` e `dy` chamando o método `translate()` do objeto da matriz, como a seguir:

```
sourceMatrix.translate(dx, dy);
return sourceMatrix;
```

O método rotate()

O método `rotate()` converte o fator de rotação de entrada em radianos (se fornecido em graus ou gradientes) e chama o método `rotate()` do objeto da matriz:

```
if (unit == "degrees")
{
    angle = Math.PI * 2 * angle / 360;
}
if (unit == "gradients")
{
    angle = Math.PI * 2 * angle / 100;
}
sourceMatrix.rotate(angle);
return sourceMatrix;
```

Chamada do método MatrixTransformer.transform() no aplicativo

O aplicativo contém uma interface do usuário para obter os parâmetros de transformação do usuário. Ele os transmite, junto com a propriedade `matrix` da propriedade `transform` do objeto de exibição, para o método `Matrix.transform()`, como a seguir:

```
tempMatrix = MatrixTransformer.transform(tempMatrix,
    xScaleSlider.value,
    yScaleSlider.value,
    dxSlider.value,
    dySlider.value,
    rotationSlider.value,
    skewSlider.value,
    skewSide );
```

O aplicativo aplica o valor de retorno à propriedade `matrix` da propriedade `transform` do objeto de exibição, acionando a transformação:

```
img.content.transform.matrix = tempMatrix;
```

Capítulo 16: Filtro de objetos de exibição

Historicamente, a aplicação de filtros em imagens de bitmap tem sido feita por softwares especiais de edição de imagens como o Adobe Photoshop® e o Adobe Fireworks®. O ActionScript 3.0 inclui o pacote flash.filters, que contém uma série de classes de filtro de bitmap que permite aos desenvolvedores aplicar filtros de modo programático em bitmaps e objetos de exibição para atingir os mesmos efeitos que estão disponíveis em aplicativos de manipulação de elementos gráficos.

Noções básicas sobre filtragem de objetos de exibição

Introdução à filtragem de objetos de exibição

Uma das maneiras de refinar um aplicativo é adicionar efeitos gráficos simples, como uma sombra projetada atrás de uma foto para criar a ilusão 3D ou um brilho em torno de um botão para mostrar que ele está ativo. O ActionScript 3.0 inclui nove filtros que podem ser aplicados em qualquer objeto de exibição ou em uma ocorrência de BitmapData. Esses filtros variam desde filtros básicos, como os de sombra projetada e brilho, até filtros complexos para criar diversos efeitos, como o deslocamento do filtro de mapa e do filtro de convolução.

Tarefas comuns de filtragem

As tarefas a seguir provavelmente serão realizadas ao utilizar filtros no ActionScript:

- Criação de um filtro
- Aplicação de um filtro em um objeto de exibição
- Remoção de um filtro de um objeto de exibição
- Aplicação de um filtro nos dados de imagem em uma ocorrência de BitmapData
- Remoção de filtros de um objeto
- Criação de diversos filtros, como brilho, desfoque, sombra projetada, nitidez, deslocamento, detecção de borda, entalhe e outros efeitos

Conceitos e termos importantes

A lista de referência a seguir contém termos importantes usados neste capítulo:

- Bisel: uma borda criada por meio do clareamento de pixels em dois lados e do escurecimento dos pixels nos dois lados opostos, criando um efeito de borda tridimensional normalmente usado para botões realçados ou recuados e efeitos gráficos similares.
- Convolução: distorção de pixels em uma imagem por meio da combinação do valor de cada pixel com os valores de alguns ou todos os pixels ao redor, usando diversas proporções.
- Deslocamento: mudança ou movimentação de pixels em uma imagem para uma nova posição.
- Matriz: uma grade de números usada para realizar alguns cálculos matemáticos aplicando os números da grade em diversos valores e combinando os resultados.

Teste dos exemplos do capítulo

Talvez você queira testar as listagens de código de exemplo fornecidas durante a leitura deste capítulo. Como este capítulo trata da criação e da manipulação de conteúdo visual, o teste do código envolve a execução do código e a visualização dos resultados no SWF criado. Quase todos os exemplos criam o conteúdo usando a API de desenho ou carregam imagens nas quais os filtros são aplicados.

Para testar o código deste capítulo:

1. Crie um documento vazio usando a ferramenta de autoria do Flash.
2. Selecione um quadro-chave na Linha de tempo.
3. Abra o painel Ações e copie o código no painel Script.
4. Execute o programa usando o comando Controlar > Testar filme.

   Você verá os resultados do código no arquivo SWF criado.

Quase todos os exemplos incluem um código que cria uma imagem de bitmap, de modo que você pode testar apenas o código, sem precisar fornecer nenhum conteúdo de bitmap. Se preferir, altere as listagens de código para carregar suas próprias imagens e use-as em vez das dos exemplos.

Criação e aplicação de filtros

Os filtros permitem aplicar diversos efeitos em objetos de exibição e bitmap, variando desde sombras projetadas até biséis e desfoques. Cada filtro é definido como uma classe, de modo que a aplicação de filtros envolve a criação de ocorrências de objetos de filtro, o que equivale a criar qualquer outro objeto. Depois de criar uma ocorrência de um objeto de filtro, é possível aplicá-la com facilidade em um objeto de exibição usando a propriedade `filters` do objeto ou, no caso de um objeto BitmapData, usando o método `applyFilter()`.

Criação de um novo filtro

Para criar um novo objeto de filtro, basta chamar o método de construtor da classe de filtro selecionada. Por exemplo, para criar um novo objeto DropShadowFilter, use o seguinte código:

```
import flash.filters.DropShadowFilter;
var myFilter:DropShadowFilter = new DropShadowFilter();
```

Embora não seja mostrado aqui, o construtor `DropShadowFilter()` (como todos os construtores de classes de filtro) aceita vários parâmetros opcionais que podem ser usados para personalizar a aparência do efeito do filtro.

Aplicação de um filtro

Depois de criar um objeto de filtro, você pode aplicá-lo em um objeto de exibição ou em um objeto BitmapData; o modo de aplicação do filtro depende do objeto no qual ele será aplicado.

Aplicação de um filtro em um objeto de exibição

Use a propriedade `filters` para aplicar efeitos de filtro em um objeto de exibição. A propriedade `filters` de um objeto de exibição é uma ocorrência de Array, cujos elementos são objetos de filtro aplicados no objeto de exibição. Para aplicar um único filtro em um objeto de exibição, crie a ocorrência de filtro, adicione-a a uma ocorrência de Array e atribua esse objeto Array à propriedade `filters` do objeto de exibição:

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.filters.DropShadowFilter;

// Create a bitmapData object and render it to screen
var myBitmapData:BitmapData = new BitmapData(100,100,false,0xFFFF3300);
var myDisplayObject:Bitmap = new Bitmap(myBitmapData);
addChild(myDisplayObject);

// Create a DropShadowFilter instance.
var dropShadow:DropShadowFilter = new DropShadowFilter();

// Create the filters array, adding the filter to the array by passing it as
// a parameter to the Array() constructor.
var filtersArray:Array = new Array(dropShadow);

// Assign the filters array to the display object to apply the filter.
myDisplayObject.filters = filtersArray;
```

Se desejar atribuir vários filtros ao objeto, basta adicionar todos os filtros à ocorrência de Array antes de atribuí-la à propriedade `filters`. Para adicionar vários objetos a uma matriz, transmita-os como parâmetros para o construtor. Por exemplo, esse código aplica um filtro de bisel e um filtro de brilho ao objeto de exibição criado anteriormente:

```
import flash.filters.BevelFilter;
import flash.filters.GlowFilter;

// Create the filters and add them to an array.
var bevel:BevelFilter = new BevelFilter();
var glow:GlowFilter = new GlowFilter();
var filtersArray:Array = new Array(bevel, glow);

// Assign the filters array to the display object to apply the filter.
myDisplayObject.filters = filtersArray;
```

Ao criar a matriz que contém os filtros, você pode usar o construtor `new Array()` (conforme mostrado nos exemplos anteriores) ou a sintaxe do literal de matriz, colocando os filtros entre colchetes (`[]`). Por exemplo, esta linha de código:

```
var filters:Array = new Array(dropShadow, blur);
```

faz a mesma coisa que esta linha de código:

```
var filters:Array = [dropShadow, blur];
```

Se vários filtros forem aplicados em objetos de exibição, a aplicação será feita de modo cumulativo e seqüencial. Por exemplo, se uma matriz de filtros tiver dois elementos, um filtro de bisel adicionado primeiro e um filtro de sombra projetada adicionado depois, o filtro de sombra projetada será aplicado no filtro de bisel e no objeto de exibição. Isso ocorre devido à posição do filtro de sombra projetada em segundo lugar na matriz de filtros. Se desejar aplicar filtros de modo não cumulativo, aplique cada filtro em uma nova cópia do objeto de exibição.

Se estiver atribuindo apenas um ou alguns filtros a um objeto de exibição, crie a ocorrência do filtro e atribua-a ao objeto em uma única instrução. Por exemplo, a linha de código a seguir aplica um filtro de desfoque em um objeto de exibição chamado `myDisplayObject`:

```
myDisplayObject.filters = [new BlurFilter()];
```

O código anterior cria uma ocorrência de Array usando a sintaxe do literal de matriz (entre colchetes), cria uma nova ocorrência de BlurFilter como um elemento da matriz e atribui a essa matriz a propriedade `filters` do objeto de exibição chamado `myDisplayObject`.

Remoção de filtros de um objeto de exibição

A remoção de todos os filtros de um objeto de exibição é tão simples quanto atribuir um valor nulo à propriedade `filters`:

```
myDisplayObject.filters = null;
```

Se tiver aplicado vários filtros em um objeto e desejar remover apenas um dos filtros, realize as etapas para alterar a matriz da propriedade `filters`. Para obter mais informações, consulte "Possíveis problemas resultantes do trabalho com filtros" na página 357.

Aplicação de um filtro em um objeto BitmapData

A aplicação de um filtro em um objeto BitmapData requer o uso do método `applyFilter()` do objeto BitmapData:

```
var rect:Rectangle = new Rectangle();
var origin:Point = new Point();
myBitmapData.applyFilter(sourceBitmapData, rect, origin, new BlurFilter());
```

O método `applyFilter()` aplica um filtro em um objeto BitmapData de origem, produzindo uma nova imagem filtrada. Esse método não modifica a imagem de origem inicial; em vez disso, o resultado da aplicação do filtro na imagem de origem é armazenado na ocorrência de BitmapData na qual o método `applyFilter()` é chamado.

Como funcionam os filtros

A filtragem de objetos de exibição funciona armazenando uma cópia do objeto original em cache como um bitmap transparente.

Assim que um filtro é aplicado em um objeto de exibição, o Adobe Flash Player ou o Adobe® AIR™ armazena o objeto em cache como um bitmap enquanto o objeto tiver uma lista de filtros válida. O bitmap de origem é usado como imagem original para todos os efeitos de filtro aplicados posteriormente.

Cada objeto de exibição geralmente contém dois bitmaps: um com o objeto original de exibição de origem e outro para a imagem final, após a filtragem. A imagem final é usada no momento da renderização. Desde que o objeto de exibição não seja alterado, a imagem final não precisa de atualização.

Possíveis problemas resultantes do trabalho com filtros

Existem várias fontes de possíveis problemas que devem ser consideradas ao trabalhar com filtros. Elas são descritas nas seções a seguir.

Cache de bitmaps e filtros

Para aplicar um filtro em um objeto de exibição, o cache de bitmaps deve ser ativado para esse objeto. Ao aplicar um filtro em um objeto de exibição cuja propriedade `cacheAsBitmap` é definida como `false`, o Flash Player ou o AIR define automaticamente o valor da propriedade `cacheAsBitmap` do objeto como `true`. Se todos os filtros forem removidos do objeto de exibição posteriormente, o Flash Player ou o AIR redefinirá a propriedade `cacheAsBitmap` como o valor definido pela última vez.

Alteração de filtros em tempo de execução

Se um objeto de exibição já tiver um ou mais filtros aplicados, não será possível alterar o conjunto de filtros adicionando mais filtros ou removendo filtros da matriz da propriedade `filters`. Em vez disso, para adicionar ou alterar o conjunto de filtros aplicados, faça as alterações em uma matriz separada e, em seguida, atribua essa matriz à propriedade filters do objeto de exibição para que os filters sejam aplicados no objeto. A maneira mais simples de fazer isso é ler a matriz da propriedade `filters` em uma variável Array e fazer as modificações nessa matriz temporária. Depois, atribua essa matriz novamente à propriedade `filters` do objeto de exibição. Em casos mais complexos, talvez seja necessário manter uma matriz de filtros principal separada. Faça as alterações nessa matriz de filtros principal e atribua-a novamente à propriedade `filters` do objeto de exibição depois de cada alteração.

Adição de mais um filtro

O código a seguir demonstra o processo de adição de mais um filtro a um objeto de exibição que já tem um ou mais filtros aplicados. Inicialmente, um filtro de brilho é aplicado no objeto de exibição chamado `myDisplayObject`; posteriormente, quando o objeto de exibição é clicado, a função `addFilters()` é chamada. Nesta função, dois filtros adicionais são aplicados em `myDisplayObject`:

```
import flash.events.MouseEvent;
import flash.filters.*;

myDisplayObject.filters = [new GlowFilter()];

function addFilters(event:MouseEvent):void
{
    // Make a copy of the filters array.
    var filtersCopy:Array = myDisplayObject.filters;

    // Make desired changes to the filters (in this case, adding filters).
    filtersCopy.push(new BlurFilter());
    filtersCopy.push(new DropShadowFilter());

    // Apply the changes by reassigning the array to the filters property.
    myDisplayObject.filters = filtersCopy;
}

myDisplayObject.addEventListener(MouseEvent.CLICK, addFilters);
```

Remoção de um filtro do conjunto de filtros

Se um objeto de exibição tiver vários filtros aplicados e você desejar remover um dos filtros enquanto os outros continuam a ser aplicados no objeto, copie os filtros em uma matriz separada, remova o filtro indesejado dessa matriz e atribua a matriz temporária novamente à propriedade `filters` do objeto de exibição. Várias maneiras de remover um ou mais elementos de qualquer matriz estão descritas em "Recuperação de valores e remoção de elementos de matriz" na página 164.

O mais fácil é remover o filtro de nível superior do objeto (o último filtro aplicado no objeto). Use o método `pop()` da classe Array para remover o filtro da matriz:

```
// Example of removing the top-most filter from a display object
// named "filteredObject".

var tempFilters:Array = filteredObject.filters;

// Remove the last element from the Array (the top-most filter).
tempFilters.pop();

// Apply the new set of filters to the display object.
filteredObject.filters = tempFilters;
```

Similarmente, para remover o filtro de nível inferior (o primeiro aplicado no objeto), use o mesmo código, usando o método `shift()` da matriz Array em vez do método `pop()`.

Para remover um filtro do meio de uma matriz de filtros (supondo que a matriz tem mais de dois filtros), use o método `splice()`. Você deve saber o índice (a posição na matriz) do filtro que deseja remover. Por exemplo, o código a seguir remove o segundo filtro (no índice 1) de um objeto de exibição:

```
// Example of removing a filter from the middle of a stack of filters
// applied to a display object named "filteredObject".

var tempFilters:Array = filteredObject.filters;

// Remove the second filter from the array. It's the item at index 1
// because Array indexes start from 0.
// The first "1" indicates the index of the filter to remove; the
// second "1" indicates how many elements to remove.
tempFilters.splice(1, 1);

// Apply the new set of filters to the display object.
filteredObject.filters = tempFilters;
```

Determinação do índice de um filtro

Você precisa saber qual filtro deve ser removido da matriz para determinar o índice do filtro. Você deve saber (em virtude da designação do aplicativo) ou calcular o índice do filtro a ser removido.

A melhor maneira é designar o aplicativo de modo que o filtro a ser removido sempre fique na mesma posição no conjunto de filtros. Por exemplo, se houver apenas um objeto de exibição com um filtro de convolução e um filtro de sombra projetada aplicados (nessa ordem) e você desejar remover o filtro de sombra projetada, mas manter o outro, o filtro estará em uma posição conhecida (o filtro de nível superior) para que você saiba com antecedência qual método Array deve ser usado (neste caso, `Array.pop()` para remover o filtro de sombra projetada).

Se o filtro que deseja remover for sempre do mesmo tipo, mas não estiver necessariamente na mesma posição do conjunto de filtros todas as vezes, verifique o tipo de dados de cada filtro da matriz para determinar qual deve ser removido. Por exemplo, o código a seguir determina qual é o filtro de brilho em um conjunto de filtros e remove esse filtro do conjunto.

```
// Example of removing a glow filter from a set of filters, where the
//filter you want to remove is the only GlowFilter instance applied
// to the filtered object.

var tempFilters:Array = filteredObject.filters;

// Loop through the filters to find the index of the GlowFilter instance.
var glowIndex:int;
var numFilters:int = tempFilters.length;
for (var i:int = 0; i < numFilters; i++)
{
    if (tempFilters[i] is GlowFilter)
    {
        glowIndex = i;
        break;
    }
}

// Remove the glow filter from the array.
tempFilters.splice(glowIndex, 1);

// Apply the new set of filters to the display object.
filteredObject.filters = tempFilters;
```

Em um caso mais complexo, como se o filtro a ser removido for selecionado em tempo de execução, o melhor é manter uma cópia separada permanente da matriz de filtros que serve como a lista principal de filtros. Sempre que você alterar o conjunto de filtros (adicionar ou remover um filtro), altere a lista principal e aplique essa matriz de filtros como a propriedade `filters` do objeto de exibição.

Por exemplo, na listagem de código a seguir, vários filtros de convolução são aplicados em um objeto de exibição para criar efeitos visuais diferentes e, posteriormente, um desses filtros é removido enquanto os outros são mantidos. Neste caso, o código mantém uma cópia principal da matriz de filtros, bem como uma referência ao filtro a ser removido. Localizar e remover o filtro específico é similar à abordagem anterior, mas, em vez de criar uma cópia temporária da matriz de filtros, a cópia principal é manipulada e aplicada no objeto de exibição.

```
// Example of removing a filter from a set of
// filters, where there may be more than one
// of that type of filter applied to the filtered
// object, and you only want to remove one.

// A master list of filters is stored in a separate,
// persistent Array variable.
var masterFilterList:Array;

// At some point, you store a reference to the filter you
// want to remove.
var filterToRemove:ConvolutionFilter;

// ... assume the filters have been added to masterFilterList,
// which is then assigned as the filteredObject.filters:
filteredObject.filters = masterFilterList;

// ... later, when it's time to remove the filter, this code gets called:

// Loop through the filters to find the index of masterFilterList.
var removeIndex:int = -1;
var numFilters:int = masterFilterList.length;
for (var i:int = 0; i < numFilters; i++)
{
    if (masterFilterList[i] == filterToRemove)
    {
        removeIndex = i;
        break;
    }
}

if (removeIndex >= 0)
{
    // Remove the filter from the array.
    masterFilterList.splice(removeIndex, 1);

    // Apply the new set of filters to the display object.
    filteredObject.filters = masterFilterList;
}
```

Nesta abordagem (quando estiver comparando uma referência de filtro armazenada com os itens da matriz de filtros para determinar qual filtro deve ser removido), você *deve* manter uma cópia separada da matriz de filtros - o código não funcionará se a referência de filtro armazenada for comparada com os elementos de uma matriz temporária copiada da propriedade `filters` do objeto de exibição. Isso ocorre porque internamente, quando uma matriz é atribuída à propriedade `filters`, o Flash Player ou o AIR faz uma cópia de cada objeto de filtro na matriz. Essas cópias (não os objetos originais) são aplicadas no objeto de exibição e, quando a propriedade `filters` é lida em uma matriz temporária, a matriz temporária contém referências aos objetos de filtro copiados, em vez de referências aos objetos de filtro originais. Conseqüentemente, se no exemplo anterior você tentar determinar o índice de `filterToRemove` comparando-o com os filtros de uma matriz temporária, nenhuma correspondência será encontrada.

Transformações de objeto e filtros

Nenhuma região filtrada (uma sombra projetada, por exemplo) fora do retângulo delimitador do objeto de exibição é considerada como parte da superfície para fins de detecção de ocorrências (determinar se uma ocorrência se sobrepõe ou faz interseção com outra). Como os métodos de detecção de ocorrências da classe DisplayObject são baseados em vetores, não é possível detectar ocorrências no resultado de bitmap. Por exemplo, se um filtro de bisel for aplicado em uma ocorrência de botão, a detecção de ocorrências não estará disponível na parte do bisel da ocorrência.

Os filtros não permitem dimensionar, girar e inclinar objetos; se o objeto de exibição filtrado for dimensionado (se `scaleX` e `scaleY` não forem 100%), o efeito de filtro não será dimensionado com a ocorrência. Isso significa que a forma original da ocorrência é girada, dimensionada ou inclinada; no entanto, o filtro não é girado, dimensionado ou inclinado com a ocorrência.

Você pode animar uma ocorrência com um filtro para criar efeitos realistas ou aninhar ocorrências e usar a classe BitmapData para animar os filtros e atingir esse efeito.

Objetos de bitmap e filtros

Quando algum filtro é aplicado em um objeto BitmapData, a propriedade `cacheAsBitmap` é definida automaticamente como `true`. Desse modo, o filtro é realmente aplicado na cópia do objeto, não no original.

Em seguida, essa cópia é colocada na exibição principal (no objeto original) o mais perto possível do pixel mais próximo. Se os limites do bitmap original forem alterados, o bitmap de cópia filtrado será recriado a partir do original, em vez de ser esticado ou distorcido.

Se você apagar todos os filtros de um objeto de exibição, a propriedade `cacheAsBitmap` será redefinida como o valor original antes do filtro ser aplicado.

Filtros de exibição disponíveis

O ActionScript 3.0 inclui dez classes de filtro que podem ser aplicadas em objetos de exibição e em objetos BitmapData:

- Filtro de bisel (classe BevelFilter)
- Filtro de desfoque (classe BlurFilter)
- Filtro de sombra projetada (classe DropShadowFilter)
- Filtro de brilho (classe GlowFilter)
- Filtro de bisel de gradiente (classe GradientBevelFilter)
- Filtro de brilho de gradiente (classe GradientGlowFilter)
- Filtro de matriz de cor (classe ColorMatrixFilter)
- Filtro de convolução (classe ConvolutionFilter)
- Filtro de mapa de deslocamento (classe DisplacementMapFilter)
- Filtro de sombreador (classe ShaderFilter)

Os seis primeiros filtros são simples e podem ser usados para criar um efeito específico, com alguns recursos de personalização disponíveis. Esses seis filtros podem ser aplicados usando o ActionScript e também em objetos no Adobe Flash CS4 Professional usando o painel Filtros. Conseqüentemente, mesmo que os filtros sejam aplicados com ActionScript, se você tiver a ferramenta de criação do Flash, poderá usar a interface visual para experimentar rapidamente filtros e configurações diferentes para saber como criar o efeito desejado.

Os quatro últimos filtros estão disponíveis somente no ActionScript. Esses filtros (matriz de cor, convolução, mapa de deslocamento e sombreador) são muito mais flexíveis nos tipos de efeitos que podem criar. Em vez de serem otimizados para um único efeito, eles fornecem poder e flexibilidade. Por exemplo, selecionando valores diferentes para sua matriz, o filtro de convolução pode ser usado para criar efeitos como desfoque, entalhe, nitidez, localização de bordas de cor, transformações e muito mais.

Cada filtro, simples ou complexo, pode ser personalizado com suas propriedades. Geralmente, existem duas opções para configurar propriedades de filtro. Todos os filtros permitem definir as propriedades por meio da transmissão de valores de parâmetro ao construtor do objeto de filtro. Se preferir, independentemente de configurar as propriedades de filtro transmitindo parâmetros, você pode ajustar os filtros posteriormente definindo valores para as propriedades do objeto de filtro. A maioria das listagens de código de exemplo define as propriedades diretamente para facilitar o acompanhamento do exemplo. Entretanto, você normalmente atinge o mesmo resultado em menos linhas de código transmitindo os valores como parâmetros no construtor de objeto de filtro. Para obter mais detalhes sobre as especificidades de cada filtro e suas respectivas propriedades e parâmetros de construtor, consulte as listagens do pacote flash.filters na Referência de componentes e linguagem do ActionScript 3.0.

Filtro de bisel

A classe BevelFilter permite a adição de uma borda 3D chanfrada ao objeto filtrado. Esse filtro deixa os cantos ou bordas pontudos do objeto talhados ou chanfrados.

As propriedades da classe BevelFilter permitem personalizar a aparência do bisel. É possível definir cores de realce e sombra, desfoques de borda chanfrada, ângulos de bisel e colocação da borda chanfrada; você pode inclusive criar um efeito vazado.

O exemplo a seguir carrega uma imagem externa e aplica um filtro de bisel nela.

```
import flash.display.*;
import flash.filters.BevelFilter;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.net.URLRequest;

// Load an image onto the Stage.
var imageLoader:Loader = new Loader();
var url:String = "http://www.helpexamples.com/flash/images/image3.jpg";
var urlReq:URLRequest = new URLRequest(url);
imageLoader.load(urlReq);
addChild(imageLoader);

// Create the bevel filter and set filter properties.
var bevel:BevelFilter = new BevelFilter();

bevel.distance = 5;
bevel.angle = 45;
bevel.highlightColor = 0xFFFF00;
bevel.highlightAlpha = 0.8;
bevel.shadowColor = 0x666666;
bevel.shadowAlpha = 0.8;
bevel.blurX = 5;
bevel.blurY = 5;
bevel.strength = 5;
bevel.quality = BitmapFilterQuality.HIGH;
bevel.type = BitmapFilterType.INNER;
bevel.knockout = false;

// Apply filter to the image.
imageLoader.filters = [bevel];
```

Filtro de desfoque

A classe BlurFilter mancha ou desfoca objetos de exibição e seu respectivo conteúdo. Os efeitos de desfoque são úteis para dar a impressão de que um objeto está fora do foco ou para simular um deslocamento rápido, como em um desfoque de movimento. Defina a propriedade `quality` do filtro de desfoque como baixa para simular um efeito de lente ligeiramente fora do foco. Definir a propriedade `quality` como alta resulta em um efeito de desfoque suave similar a um desfoque de Gauss.

O exemplo a seguir cria um objeto circular usando o método `drawCircle()` da classe Graphics e aplica um filtro de desfoque nele:

```
import flash.display.Sprite;
import flash.filters.BitmapFilterQuality;
import flash.filters.BlurFilter;

// Draw a circle.
var redDotCutout:Sprite = new Sprite();
redDotCutout.graphics.lineStyle();
redDotCutout.graphics.beginFill(0xFF0000);
redDotCutout.graphics.drawCircle(145, 90, 25);
redDotCutout.graphics.endFill();

// Add the circle to the display list.
addChild(redDotCutout);

// Apply the blur filter to the rectangle.
var blur:BlurFilter = new BlurFilter();
blur.blurX = 10;
blur.blurY = 10;
blur.quality = BitmapFilterQuality.MEDIUM;
redDotCutout.filters = [blur];
```

Filtro de sombra projetada

As sombras projetadas dão a impressão de que existe uma fonte de luz separada situada acima do objeto de destino. A posição e a intensidade dessa fonte de luz podem ser modificadas para produzir diversos efeitos de sombra projetada diferentes.

A classe DropShadowFilter usa um algoritmo similar ao do filtro de desfoque. A principal diferença é que o filtro de sombra projetada tem mais propriedades que podem ser modificadas para simular outros atributos de fonte de luz (como alfa, cor, deslocamento e brilho).

O filtro de sombra projetada também permite aplicar opções de transformação personalizadas no estilo da sombra projetada, incluindo a sombra interna ou externa e o modo vazado (também conhecido como recorte).

O código a seguir cria uma caixa quadrada e aplica um filtro de sombra projetada nela:

```
import flash.display.Sprite;
import flash.filters.DropShadowFilter;

// Draw a box.
var boxShadow:Sprite = new Sprite();
boxShadow.graphics.lineStyle(1);
boxShadow.graphics.beginFill(0xFF3300);
boxShadow.graphics.drawRect(0, 0, 100, 100);
boxShadow.graphics.endFill();
addChild(boxShadow);

// Apply the drop shadow filter to the box.
var shadow:DropShadowFilter = new DropShadowFilter();
shadow.distance = 10;
shadow.angle = 25;

// You can also set other properties, such as the shadow color,
// alpha, amount of blur, strength, quality, and options for
// inner shadows and knockout effects.

boxShadow.filters = [shadow];
```

Filtro de brilho

A classe GlowFilter aplica um efeito de iluminação aos objetos de exibição, como se o objeto fosse iluminado por baixo, criando um leve brilho.

Assim como o filtro de sombra projetada, o filtro de brilho inclui propriedades para modificar a distância, o ângulo e a cor da fonte de luz para produzir efeitos variados. O GlowFilter também tem várias opções para modificar o estilo do brilho, incluindo o brilho interno ou externo e o modo vazado.

O código a seguir cria uma transversal usando a classe Sprite e aplica um filtro de brilho nela:

```
import flash.display.Sprite;
import flash.filters.BitmapFilterQuality;
import flash.filters.GlowFilter;

// Create a cross graphic.
var crossGraphic:Sprite = new Sprite();
crossGraphic.graphics.lineStyle();
crossGraphic.graphics.beginFill(0xCCCC00);
crossGraphic.graphics.drawRect(60, 90, 100, 20);
crossGraphic.graphics.drawRect(100, 50, 20, 100);
crossGraphic.graphics.endFill();
addChild(crossGraphic);

// Apply the glow filter to the cross shape.
var glow:GlowFilter = new GlowFilter();
glow.color = 0x009922;
glow.alpha = 1;
glow.blurX = 25;
glow.blurY = 25;
glow.quality = BitmapFilterQuality.MEDIUM;

crossGraphic.filters = [glow];
```

Filtro de bisel de gradiente

A classe GradientBevelFilter permite a aplicação de um efeito de bisel aprimorado em objetos de exibição ou objetos BitmapData. Usar uma cor de gradiente no bisel melhora muito a profundidade espacial do bisel, dando às bordas uma aparência 3D mais realista.

O código a seguir cria um objeto retangular usando o método `drawRect()` da classe Shape e aplica um filtro de bisel de gradiente nele.

```
import flash.display.Shape;
import flash.filters.BitmapFilterQuality;
import flash.filters.GradientBevelFilter;

// Draw a rectangle.
var box:Shape = new Shape();
box.graphics.lineStyle();
box.graphics.beginFill(0xFEFE78);
box.graphics.drawRect(100, 50, 90, 200);
box.graphics.endFill();

// Apply a gradient bevel to the rectangle.
var gradientBevel:GradientBevelFilter = new GradientBevelFilter();

gradientBevel.distance = 8;
gradientBevel.angle = 225; // opposite of 45 degrees
gradientBevel.colors = [0xFFFFCC, 0xFEFE78, 0x8F8E01];
gradientBevel.alphas = [1, 0, 1];
gradientBevel.ratios = [0, 128, 255];
gradientBevel.blurX = 8;
gradientBevel.blurY = 8;
gradientBevel.quality = BitmapFilterQuality.HIGH;

// Other properties let you set the filter strength and set options
// for inner bevel and knockout effects.

box.filters = [gradientBevel];

// Add the graphic to the display list.
addChild(box);
```

Filtro de brilho de gradiente

A classe GradientGlowFilter permite a aplicação de um efeito de brilho aprimorado em objetos de exibição ou objetos BitmapData. Os efeitos permitem que você tenha mais controle sobre o brilho e produza um efeito mais realista. Além disso, o filtro de brilho de gradiente permite aplicar um brilho gradiente nas bordas interna, externa ou superior de um objeto.

O exemplo a seguir desenha um círculo no palco e aplica um filtro de brilho de gradiente a ele. À medida que você move o mouse mais para a direita e para baixo, a quantidade de desfoque aumenta nas direções horizontal e vertical, respectivamente. Além disso, sempre que você clicar no palco, a intensidade do desfoque aumentará.

```
import flash.events.MouseEvent;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.filters.GradientGlowFilter;

// Create a new Shape instance.
var shape:Shape = new Shape();

// Draw the shape.
shape.graphics.beginFill(0xFF0000, 100);
shape.graphics.moveTo(0, 0);
shape.graphics.lineTo(100, 0);
shape.graphics.lineTo(100, 100);
shape.graphics.lineTo(0, 100);
shape.graphics.lineTo(0, 0);
shape.graphics.endFill();

// Position the shape on the Stage.
addChild(shape);
shape.x = 100;
shape.y = 100;

// Define a gradient glow.
var gradientGlow:GradientGlowFilter = new GradientGlowFilter();
gradientGlow.distance = 0;
gradientGlow.angle = 45;
gradientGlow.colors = [0x000000, 0xFF0000];
gradientGlow.alphas = [0, 1];
gradientGlow.ratios = [0, 255];
gradientGlow.blurX = 10;
gradientGlow.blurY = 10;
gradientGlow.strength = 2;
gradientGlow.quality = BitmapFilterQuality.HIGH;
gradientGlow.type = BitmapFilterType.OUTER;

// Define functions to listen for two events.
function onClick(event:MouseEvent):void
{
    gradientGlow.strength++;
    shape.filters = [gradientGlow];
}

function onMouseMove(event:MouseEvent):void
{
    gradientGlow.blurX = (stage.mouseX / stage.stageWidth) * 255;
    gradientGlow.blurY = (stage.mouseY / stage.stageHeight) * 255;
    shape.filters = [gradientGlow];
}
stage.addEventListener(MouseEvent.CLICK, onClick);
stage.addEventListener(MouseEvent.MOUSE_MOVE, onMouseMove);
```

Exemplo: combinação de filtros básicos

O código a seguir usa vários filtros básicos, combinados com um cronômetro, para criar ações repetidas e simular um farol animado.

```
import flash.display.Shape;
import flash.events.TimerEvent;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.filters.DropShadowFilter;
import flash.filters.GlowFilter;
import flash.filters.GradientBevelFilter;
import flash.utils.Timer;

var count:Number = 1;
var distance:Number = 8;
var angleInDegrees:Number = 225; // opposite of 45 degrees
var colors:Array = [0xFFFFCC, 0xFEFE78, 0x8F8E01];
var alphas:Array = [1, 0, 1];
var ratios:Array = [0, 128, 255];
var blurX:Number = 8;
var blurY:Number = 8;
var strength:Number = 1;
var quality:Number = BitmapFilterQuality.HIGH;
var type:String = BitmapFilterType.INNER;
var knockout:Boolean = false;

// Draw the rectangle background for the traffic light.
var box:Shape = new Shape();
box.graphics.lineStyle();
box.graphics.beginFill(0xFEFE78);
box.graphics.drawRect(100, 50, 90, 200);
box.graphics.endFill();

// Draw the 3 circles for the three lights.
var stopLight:Shape = new Shape();
stopLight.graphics.lineStyle();
stopLight.graphics.beginFill(0xFF0000);
stopLight.graphics.drawCircle(145,90,25);
stopLight.graphics.endFill();

var cautionLight:Shape = new Shape();
cautionLight.graphics.lineStyle();
cautionLight.graphics.beginFill(0xFF9900);
cautionLight.graphics.drawCircle(145,150,25);
cautionLight.graphics.endFill();

var goLight:Shape = new Shape();
goLight.graphics.lineStyle();
goLight.graphics.beginFill(0x00CC00);
goLight.graphics.drawCircle(145,210,25);
goLight.graphics.endFill();

// Add the graphics to the display list.
addChild(box);
addChild(stopLight);
addChild(cautionLight);
addChild(goLight);

// Apply a gradient bevel to the traffic light rectangle.
var gradientBevel:GradientBevelFilter = new GradientBevelFilter(distance, angleInDegrees,
colors, alphas, ratios, blurX, blurY, strength, quality, type, knockout);
```

```
box.filters = [gradientBevel];

// Create the inner shadow (for lights when off) and glow
// (for lights when on).
var innerShadow:DropShadowFilter = new DropShadowFilter(5, 45, 0, 0.5, 3, 3, 1, 1, true,
false);
var redGlow:GlowFilter = new GlowFilter(0xFF0000, 1, 30, 30, 1, 1, false, false);
var yellowGlow:GlowFilter = new GlowFilter(0xFF9900, 1, 30, 30, 1, 1, false, false);
var greenGlow:GlowFilter = new GlowFilter(0x00CC00, 1, 30, 30, 1, 1, false, false);

// Set the starting state of the lights (green on, red/yellow off).
stopLight.filters = [innerShadow];
cautionLight.filters = [innerShadow];
goLight.filters = [greenGlow];

// Swap the filters based on the count value.
function trafficControl(event:TimerEvent):void
{
    if (count == 4)
    {
        count = 1;
    }

    switch (count)
    {
        case 1:
            stopLight.filters = [innerShadow];
            cautionLight.filters = [yellowGlow];
            goLight.filters = [innerShadow];
            break;
        case 2:
            stopLight.filters = [redGlow];
            cautionLight.filters = [innerShadow];
            goLight.filters = [innerShadow];
            break;
        case 3:
            stopLight.filters = [innerShadow];
            cautionLight.filters = [innerShadow];
            goLight.filters = [greenGlow];
            break;
    }

    count++;
}

// Create a timer to swap the filters at a 3 second interval.
var timer:Timer = new Timer(3000, 9);
timer.addEventListener(TimerEvent.TIMER, trafficControl);
timer.start();
```

Filtro de matriz de cor

A classe ColorMatrixFilter é usada para manipular a cor e os valores alfa do objeto filtrado. Isso permite criar alterações de saturação, rotação de matizes (alterando uma paleta de uma gama de cores para outra), alterações de luminância para alfa e outros efeitos de manipulação de cor usando valores de um canal de cor e possivelmente aplicando-os em outros canais.

Conceitualmente, o filtro percorre cada pixel da imagem de origem e separa-os nos componentes vermelho, verde, azul e alfa. Em seguida, os valores fornecidos na matriz de cor são multiplicados por cada um desses valores, somando os resultados para determinar o valor de cor resultante que será exibido na tela para o pixel em questão. A propriedade `matrix` do filtro é uma matriz de 20 números usada no cálculo da cor final. Para obter informações sobre o algoritmo específico usado para calcular os valores de cor, consulte a entrada que descreve a propriedade de matriz da classe ColorMatrixFilter na Referência de componentes e linguagem do ActionScript 3.0.

Outras informações e exemplos do filtro de matriz de cor estão disponíveis no artigo "Uso de matrizes para transformações, ajustes de cor e efeitos de convolução no Flash" no site do Centro de desenvolvedores da Adobe.

Filtro de convolução

A classe ConvolutionFilter pode ser usada para aplicar diversas transformações de imagem em objetos BitmapData e objetos de exibição, como desfoque, detecção de borda, nitidez, entalhe e bisel.

Conceitualmente, o filtro de convolução percorre cada pixel da imagem de origem e determina a cor final desse pixel usando o valor do pixel e dos pixels ao redor. Uma matriz, especificada como uma matriz de valores numéricos, indica até que ponto o valor de cada pixel ao redor afeta o valor resultante final.

Considere o tipo de matriz utilizado com mais freqüência, que é uma matriz 3 x 3. A matriz inclui nove valores:

```
N   N   N
N   P   N
N   N   N
```

Quando o Flash Player ou o AIR está aplicando o filtro de convolução em um determinado pixel, é considerado o valor de cor do pixel propriamente dito ("P" no exemplo), bem como os valores dos pixels ao redor ("N" no exemplo). No entanto, definindo valores na matriz, você especifica a prioridade de determinados pixels na imagem resultante.

Por exemplo, a matriz a seguir, que tem um filtro de convolução aplicado, deixará uma imagem exatamente como era:

```
0   0   0
0   1   0
0   0   0
```

A imagem não é alterada porque o valor do pixel original tem uma intensidade relativa igual a 1 para determinar a cor final do pixel, enquanto os valores dos pixels ao redor têm a intensidade relativa igual a 0 - suas cores não afetam a imagem final.

Similarmente, essa matriz fará com que os pixels de uma imagem mudem um pixel para a esquerda:

```
0   0   0
0   0   1
0   0   0
```

Observe que, nesse caso, o pixel propriamente dito não afeta o valor final do pixel exibido nesse local na imagem final; apenas o valor do pixel à direita é usado para determinar o valor resultante do pixel.

No ActionScript, você cria a matriz como uma combinação de uma ocorrência de Array que contém os valores e as duas propriedades que especificam o número de linhas e colunas da matriz. O exemplo a seguir carrega uma imagem e, quando o carregamento termina, aplica um filtro de convolução na imagem usando a matriz da listagem anterior:

```
// Load an image onto the Stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image1.jpg");
loader.load(url);
this.addChild(loader);

function applyFilter(event:MouseEvent):void
{
    // Create the convolution matrix.
    var matrix:Array = [0, 0, 0,
                                    0, 0, 1,
                                    0, 0, 0];

    var convolution:ConvolutionFilter = new ConvolutionFilter();
    convolution.matrixX = 3;
    convolution.matrixY = 3;
    convolution.matrix = matrix;
    convolution.divisor = 1;

    loader.filters = [convolution];
}

loader.addEventListener(MouseEvent.CLICK, applyFilter);
```

O que não fica óbvio nesse código é o efeito do uso de valores diferentes de 1 ou 0 na matriz. Por exemplo, a mesma matriz, com o número 8 em vez de 1 na posição direita, executa a mesma ação (movimentando os pixels para a esquerda). Além disso, ela afeta as cores da imagem, deixando-as oito vezes mais brilhantes. Isso ocorre porque os valores finais de cor do pixel são calculados pela multiplicação dos valores da matriz pelas cores do pixel original, pela soma dos valores e pela divisão pelo valor da propriedade `divisor` do filtro. Observe que, no código de exemplo, a propriedade `divisor` é definida como 1. Como regra geral, se desejar que o brilho das cores permaneça quase igual ao brilho da imagem original, o divisor deve ser igual à soma dos valores da matriz. Desse modo, se uma matriz tiver a soma de seus valores igual a 8 e o divisor igual a 1, a imagem resultante será aproximadamente 8 vezes mais brilhante do que a imagem original.

Embora o efeito nessa matriz não seja muito notável, outros valores de matriz podem ser usados para criar diversos efeitos. Veja alguns conjuntos padrão de valores de matriz para diferentes efeitos usando uma matriz 3 x 3:

- Desfoque básico (divisor 5):

```
0 1 0
1 1 1
0 1 0
```

- Nitidez (divisor 1):

```
 0, -1, 0
-1, 5, -1
 0, -1, 0
```

- Detecção de borda (divisor 1):

```
 0, -1, 0
-1, 4, -1
 0, -1, 0
```

- Efeito de entalhe (divisor 1):

```
-2, -1, 0
-1, 1, 1
 0, 1, 2
```

Observe que na maioria desses efeitos o divisor é igual a 1. Isso ocorre porque a adição de valores de matriz negativos a valores de matriz positivos resulta em 1 (ou 0 no caso da detecção de borda, mas o valor da propriedade `divisor` não pode ser 0).

Filtro de mapa de deslocamento

A classe DisplacementMapFilter usa valores de pixel de um objeto BitmapData (conhecido como a imagem do mapa de deslocamento) para realizar um efeito de deslocamento em um novo objeto. A imagem do mapa de deslocamento normalmente é diferente do objeto de exibição ou da ocorrência de BitmapData real no qual o filtro está sendo aplicado. Um efeito de deslocamento envolve a movimentação de pixels nas imagem filtrada - em outras palavras, deslocá-los do local original para alguma posição. Esse filtro pode ser usado para criar um efeito alterado, distorcido ou matizado.

O local e a quantidade de deslocamento aplicados em um determinado pixel são especificados pelo valor de cor da imagem do mapa de deslocamento. Ao trabalhar com o filtro, além de especificar a imagem do mapa, você especifica os seguintes valores para controlar como o deslocamento é calculado a partir da imagem do mapa:

- Ponto do mapa: o local da imagem filtrada na qual o canto superior esquerdo do filtra de deslocamento será aplicado. Use esse ponto apenas se desejar aplicar o filtro em parte de uma imagem.
- Componente X: canal de cor da imagem do mapa que afeta a posição x dos pixels.
- Componente Y: canal de cor da imagem do mapa que afeta a posição y dos pixels.
- Escala X: um valor de multiplicador que especifica a intensidade do deslocamento do eixo x.
- Escala Y: um valor de multiplicador que especifica a intensidade do deslocamento do eixo y.
- Modo de filtro: determina o que o Flash Player ou o AIR deve fazer nos espaços vazios criados pelos pixels alterados. As opções, definidas como constantes na classe DisplacementMapFilterMode, devem exibir os pixels originais (modo de filtro `IGNORE`), colocar os pixels em torno do outro lado da imagem (modo de filtro `WRAP`, que é o padrão), usar o pixel alterado mais próximo (modo de filtro `CLAMP`) ou preencher os espaços com uma cor (modo de filtro `COLOR`).

Para entender o funcionamento do filtro do mapa de deslocamento, considere um exemplo simples. No código a seguir, uma imagem é carregada e, em seguida, centralizada no palco. Um filtro de mapa de deslocamento é aplicado, fazendo com que os pixels da imagem inteira mudem horizontalmente para a esquerda.

```
import flash.display.BitmapData;
import flash.display.Loader;
import flash.events.MouseEvent;
import flash.filters.DisplacementMapFilter;
import flash.geom.Point;
import flash.net.URLRequest;

// Load an image onto the Stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image3.jpg");
loader.load(url);
this.addChild(loader);

var mapImage:BitmapData;
var displacementMap:DisplacementMapFilter;

// This function is called when the image finishes loading.
function setupStage(event:Event):void
{
    // Center the loaded image on the Stage.
    loader.x = (stage.stageWidth - loader.width) / 2;
    loader.y = (stage.stageHeight - loader.height) / 2;

    // Create the displacement map image.
    mapImage = new BitmapData(loader.width, loader.height, false, 0xFF0000);

    // Create the displacement filter.
    displacementMap = new DisplacementMapFilter();
    displacementMap.mapBitmap = mapImage;
    displacementMap.mapPoint = new Point(0, 0);
    displacementMap.componentX = BitmapDataChannel.RED;
    displacementMap.scaleX = 250;
    loader.filters = [displacementMap];
}

loader.contentLoaderInfo.addEventListener(Event.COMPLETE, setupStage);
```

As propriedades usadas para definir o deslocamento são as seguintes:

- Bitmap de mapa: o bitmap de deslocamento é uma nova ocorrência de BitmapData criada pelo código. Suas dimensões coincidem com as dimensões da imagem carregada (de modo que o deslocamento é aplicado na imagem inteira). Ela é preenchida com pixels vermelhos sólidos.
- Ponto do mapa: esse valor é definido como o ponto 0, 0 - mais uma vez, o deslocamento será aplicado na imagem inteira.
- Componente X: esse valor é definido como a constante `BitmapDataChannel.RED`, o que significa que o valor vermelho do bitmap de mapa determinará o deslocamento dos pixels (o quanto se movimentam) ao longo do eixo x.
- Escala X: esse valor é definido como 250. O valor total de deslocamento (a partir da imagem do mapa que está completamente vermelha) desloca a imagem apenas um pouco (aproximadamente metade de um pixel), de modo que, se esse valor fosse definido como 1, a imagem seria movida apenas 0,5 pixel na horizontal. Se esse valor for definido como 250, a imagem será deslocada aproximadamente 125 pixels.

Essas configurações fazem com que os pixels da imagem filtrada sejam deslocados 250 pixels para a esquerda. A direção (esquerda ou direita) e o valor do deslocamento baseiam-se no valor de cor dos pixels da imagem do mapa. Conceitualmente, o Flash Player ou o AIR percorre cada pixel da imagem filtrada (pelo menos os pixels da região onde o filtro é aplicado que, nesse caso, são todos os pixels) e faz o seguinte com cada pixel:

1. Localiza o pixel correspondente na imagem do mapa. Por exemplo, quando o Flash Player ou o AIR está calculando o valor de deslocamento para o pixel no canto superior esquerdo da imagem filtrada, será observado o pixel no canto superior esquerdo da imagem do mapa.
2. Determina o valor do canal de cor especificado no pixel do mapa. Nesse caso, o canal de cor do componente x é o canal vermelho, de modo que o Flash Player e o AIR observam qual é o valor do canal vermelho da imagem do mapa no pixel em questão. Como a imagem do mapa é vermelho sólido, o canal vermelho do pixel é 0xFF ou 255. Esse valor é usado como valor de deslocamento.
3. Compara o valor de deslocamento com o valor "central" (127, que é o valor médio entre 0 e 255). Se o valor de deslocamento for inferior ao valor médio, o pixel será deslocado em uma direção positiva (para a direita no deslocamento x; para baixo no deslocamento y). Por outro lado, se o valor de deslocamento for maior do que o valor médio (como neste exemplo), o pixel será deslocado em uma direção negativa (para a esquerda no deslocamento x; para cima no deslocamento y). Para ser mais preciso, o Flash Player e o AIR subtraem o valor de deslocamento de 127 e o resultado (positivo ou negativo) é o valor relativo do deslocamento que é aplicado.
4. Finalmente, ele determina o valor real de deslocamento definindo qual porcentagem de deslocamento completo é representado pelo valor de deslocamento relativo. Nesse caso, vermelho total indica 100% de deslocamento. Essa porcentagem é multiplicada pelo valor da escala x ou y para determinar o número de pixels do deslocamento que será aplicado. Neste exemplo, 100% vezes um multiplicador igual a 250 determina o valor de deslocamento - aproximadamente 125 pixels para a esquerda.

Como nenhum valor foi especificado para o componente e a escala y, os padrões (que não provocam nenhum deslocamento) foram usados; é por esse motivo que a imagem não é alterada na vertical.

A configuração padrão do modo de filtro, `WRAP`, é usada no exemplo, de modo que os pixels se movem para a esquerda e o espaço vazio à direita é preenchido pelos pixels que se deslocaram da margem esquerda da imagem. Você pode experimentar esse valor para ver os diferentes efeitos. Por exemplo, se você adicionar a linha a seguir à parte do código onde as propriedades de deslocamento estão sendo definidas (antes da linha `loader.filters = [displacementMap]`), a imagem parecerá ter sido manchada ao longo do palco:

```
displacementMap.mode = DisplacementMapFilterMode.CLAMP;
```

Em exemplos mais complexos, a listagem a seguir usa um filtro de mapa de deslocamento para criar um efeito de lupa em uma imagem:

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.display.BitmapDataChannel;
import flash.display.GradientType;
import flash.display.Loader;
import flash.display.Shape;
import flash.events.MouseEvent;
import flash.filters.DisplacementMapFilter;
import flash.filters.DisplacementMapFilterMode;
import flash.geom.Matrix;
import flash.geom.Point;
import flash.net.URLRequest;

// Create the gradient circles that will together form the
// displacement map image
var radius:uint = 50;

var type:String = GradientType.LINEAR;
var redColors:Array = [0xFF0000, 0x000000];
var blueColors:Array = [0x0000FF, 0x000000];
var alphas:Array = [1, 1];
var ratios:Array = [0, 255];
var xMatrix:Matrix = new Matrix();
xMatrix.createGradientBox(radius * 2, radius * 2);
var yMatrix:Matrix = new Matrix();
yMatrix.createGradientBox(radius * 2, radius * 2, Math.PI / 2);

var xCircle:Shape = new Shape();
xCircle.graphics.lineStyle(0, 0, 0);
xCircle.graphics.beginGradientFill(type, redColors, alphas, ratios, xMatrix);
xCircle.graphics.drawCircle(radius, radius, radius);

var yCircle:Shape = new Shape();
yCircle.graphics.lineStyle(0, 0, 0);
yCircle.graphics.beginGradientFill(type, blueColors, alphas, ratios, yMatrix);
yCircle.graphics.drawCircle(radius, radius, radius);

// Position the circles at the bottom of the screen, for reference.
this.addChild(xCircle);
xCircle.y = stage.stageHeight - xCircle.height;
this.addChild(yCircle);
yCircle.y = stage.stageHeight - yCircle.height;
yCircle.x = 200;

// Load an image onto the Stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image1.jpg");
loader.load(url);
this.addChild(loader);

// Create the map image by combining the two gradient circles.
var map:BitmapData = new BitmapData(xCircle.width, xCircle.height, false, 0x7F7F7F);
map.draw(xCircle);
var yMap:BitmapData = new BitmapData(yCircle.width, yCircle.height, false, 0x7F7F7F);
yMap.draw(yCircle);
map.copyChannel(yMap, yMap.rect, new Point(0, 0), BitmapDataChannel.BLUE,
BitmapDataChannel.BLUE);
```

```
yMap.dispose();

// Display the map image on the Stage, for reference.
var mapBitmap:Bitmap = new Bitmap(map);
this.addChild(mapBitmap);
mapBitmap.x = 400;
mapBitmap.y = stage.stageHeight - mapBitmap.height;

// This function creates the displacement map filter at the mouse location.
function magnify():void
{
    // Position the filter.
    var filterX:Number = (loader.mouseX) - (map.width / 2);
    var filterY:Number = (loader.mouseY) - (map.height / 2);
    var pt:Point = new Point(filterX, filterY);
    var xyFilter:DisplacementMapFilter = new DisplacementMapFilter();
    xyFilter.mapBitmap = map;
    xyFilter.mapPoint = pt;
    // The red in the map image will control x displacement.
    xyFilter.componentX = BitmapDataChannel.RED;
    // The blue in the map image will control y displacement.
    xyFilter.componentY = BitmapDataChannel.BLUE;
    xyFilter.scaleX = 35;
    xyFilter.scaleY = 35;
    xyFilter.mode = DisplacementMapFilterMode.IGNORE;
    loader.filters = [xyFilter];
}

// This function is called when the mouse moves. If the mouse is
// over the loaded image, it applies the filter.
function moveMagnifier(event:MouseEvent):void
{
    if (loader.hitTestPoint(loader.mouseX, loader.mouseY))
    {
        magnify();
    }
}
loader.addEventListener(MouseEvent.MOUSE_MOVE, moveMagnifier);
```

Primeiro, o código gera dois círculos de gradiente, que são combinados para formar a imagem do mapa de deslocamento. O círculo vermelho cria o deslocamento do eixo x (`xyFilter.componentX = BitmapDataChannel.RED`) e o círculo azul cria o deslocamento do eixo y (`xyFilter.componentY = BitmapDataChannel.BLUE`). Para ajudar você a entender como se parece a imagem do mapa de deslocamento, o código adiciona os círculos originais, bem como o círculo combinado que serve como a imagem do mapa, à parte inferior da tela.

Em seguida, o código carrega uma imagem e, à medida que o mouse se move, aplica o filtro de deslocamento na parte da imagem que está embaixo do mouse. Os círculos de gradiente usados como a imagem do mapa de deslocamento fazem com que a região deslocada se distancie do ponteiro do mouse. Observe que as regiões de cinza da imagem do mapa não provocam nenhum deslocamento. A cor de cinza é `0x7F7F7F`. Os canais azul e vermelho dessa sombra de cinza correspondem exatamente à sombra central desses canais de cor, de modo que não há nenhum deslocamento em uma área de cinza da imagem do mapa. Do mesmo modo, não há nenhum deslocamento no centro do círculo. Embora a cor não seja cinza, os canais azul e vermelho dessa cor são idênticos aos canais azul e vermelho do cinza médio e, como azul e vermelho são as cores que provocam o deslocamento, não ocorre nenhum deslocamento aqui.

Filtro de sombreador

A classe ShaderFilter permite o uso de um efeito de filtro personalizado definido como um sombreador Pixel Bender. Como é gravado como um sombreador Pixel Bender, o efeito de filtro pode ser completamente personalizado. O conteúdo filtrado é transmitido para o sombreador como uma entrada de imagem e o resultado da operação de shader se transforma no resultado de filtro.

Para aplicar um filtro de sombreador em um objeto, crie uma ocorrência de Shader que representa o sombreador Pixel Bender utilizado. Para obter informações sobre o procedimento de criação de uma ocorrência de Shader e sobre como especificar a imagem de entrada e valores de parâmetro, consulte “Trabalho com sombreadores Pixel Bender” na página 386.

Ao usar um sombreador como filtro, tenha em mente três coisas importantes:

- O sombreador deve ser definido para aceitar pelo menos uma imagem de entrada.
- O objeto filtrado (o objeto de exibição ou o objeto BitmapData no qual o filtro é aplicado) é transmitido para o sombreador como o primeiro valor da imagem de entrada. Devido a isso, você não deve especificar manualmente um valor para a primeira entrada de imagem.

- Se o sombreador definir mais de uma imagem de entrada, as entradas adicionais deverão ser especificadas manualmente (isto é, será necessário definir a propriedade `input` de todas as ocorrências de ShaderInput que pertencem à ocorrência de Shader).

Assim que um objeto Shader tiver sido definido para o sombreador, crie uma ocorrência de ShaderFilter. Este é o objeto de filtro real usado como qualquer outro filtro. Para criar um ShaderFilter que usa um objeto Shader, chame o construtor `ShaderFilter()` e transmita o objeto Shader como um argumento, como mostra esta listagem:

```
var myFilter:ShaderFilter = new ShaderFilter(myShader);
```

Para obter um exemplo completo de como utilizar um filtro de sombreador, consulte "Uso de um sombreador como filtro" na página 404.

Exemplo: Filter Workbench

O Filter Workbench fornece uma interface de usuário para aplicar filtros diferentes em imagens e outros conteúdos visuais e para visualizar o código resultante que pode ser usado para gerar o mesmo efeito no ActionScript. Além de fornecer uma ferramenta para experimentar filtros, esse aplicativo demonstra as seguintes técnicas:

- Criação de ocorrências de vários filtros
- Aplicação de vários filtros em um objeto de exibição

Para obter os arquivos de aplicativo desse exemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_br. Os arquivos do aplicativo Filter Workbench estão localizados na pasta Amostras/FilterWorkbench. O aplicativo consiste nos seguintes arquivos:

Arquivo	Descrição
com/example/programmingas3/filterWorkbench/FilterWorkbenchController.as	Classe que fornece a principal funcionalidade do aplicativo, incluindo a alternância do conteúdo para os filtros aplicados e a aplicação de filtros no conteúdo.
com/example/programmingas3/filterWorkbench/IFilterFactory.as	Interface que define métodos comuns que são implementados por cada classe padrão de filtro. Essa interface define a funcionalidade comum que a classe FilterWorkbenchController usa para interagir com classes individuais de filtro padrão.
Na pasta com/example/programmingas3/filterWorkbench/: BevelFactory.as BlurFactory.as ColorMatrixFactory.as ConvolutionFactory.as DropShadowFactory.as GlowFactory.as GradientBevelFactory.as GradientGlowFactory.as	Conjunto de classes, cada uma delas implementa a interface IFilterFactory. Cada uma dessas classes permite criar e definir valores para um único tipo de filtro. Os painéis de propriedade de filtro do aplicativo usam essas classes padrão para criar ocorrências de filtros específicos, que a classe FilterWorkbenchController recupera e aplica no conteúdo da imagem.
com/example/programmingas3/filterWorkbench/IFilterPanel.as	Interface que define métodos comuns implementados pelas classes que definem os painéis da interface de usuário usados para manipular valores de filtro no aplicativo.

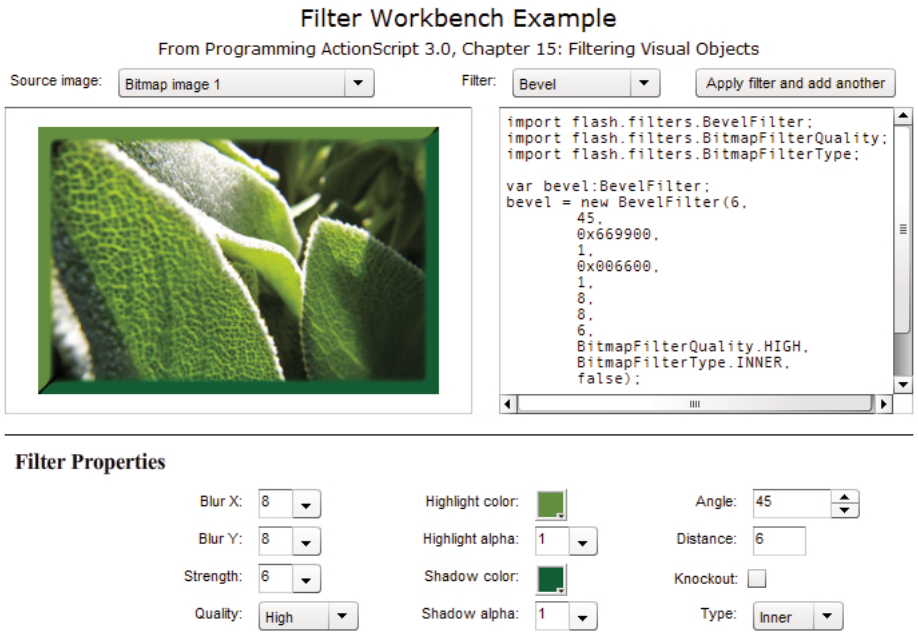
Arquivo	**Descrição**
com/example/programmingas3/filterWorkbench/ColorStringFormatter.as	Classe de utilitário que inclui um método para converter um valor de cor numérico em um formato de string hexadecimal
com/example/programmingas3/filterWorkbench/GradientColor.as	Classe que serve como objeto de valor, combinando em um único objeto os três valores (cor, alfa e proporção) associados a cada cor em GradientBevelFilter e GradientGlowFilter
Interface de usuário (Flash)	
FilterWorkbench.fla	O arquivo principal que define a interface de usuário do aplicativo.
flashapp/FilterWorkbench.as	Classe que gera a funcionalidade da interface de usuário do aplicativo principal; essa classe é usada como a classe document do arquivo FLA do aplicativo.
Na pasta flashapp/filterPanels: BevelPanel.as BlurPanel.as ColorMatrixPanel.as ConvolutionPanel.as DropShadowPanel.as GlowPanel.as GradientBevelPanel.as GradientGlowPanel.as	Conjunto de classes que geram a funcionalidade de cada painel usado para definir opções para um único filtro. Para cada classe, também existe um símbolo MovieClip associado na biblioteca do arquivo FLA do aplicativo principal, cujo nome corresponde ao nome da classe (por exemplo, o símbolo "BlurPanel" está vinculado à classe definida em BlurPanel.as). Os componentes que fazem parte da interface de usuário são posicionados e nomeados nesses símbolos.
flashapp/ImageContainer.as	Um objeto de exibição que serve como contêiner da imagem carregada na tela
flashapp/ButtonCellRenderer.as	Renderizador de célula personalizado usado para incluir um componente de botão em uma célula no componente DataGrid
Conteúdo da imagem filtrada	
com/example/programmingas3/filterWorkbench/ImageType.as	Essa classe serve como um objeto de valor que contém o tipo e a URL de um único arquivo de imagem no qual o aplicativo pode carregar e aplicar filtros. A classe também inclui um conjunto de constantes que representam os arquivos de imagem reais disponíveis.
images/sampleAnimation.swf, images/sampleImage1.jpg, images/sampleImage2.jpg	Imagens e outros conteúdos visuais nos quais os filtros são aplicados no aplicativo.

Como experimentar filtros do ActionScript

O aplicativo Filter Workbench foi desenvolvido para ajudar você a experimentar os diversos efeitos de filtro e gerar o código do ActionScript relevante para esse efeito. O aplicativo permite selecionar entre três arquivos diferentes que possuem conteúdo visual, incluindo imagens de bitmap e uma animação criada no Flash, além de permitir aplicar oito filtros diferentes do ActionScript na imagem selecionada, individualmente ou em combinação com outros filtros. O aplicativo inclui estes filtros:

- Bisel (flash.filters.BevelFilter)
- Desfoque (flash.filters.BlurFilter)
- Matrix de cor (flash.filters.ColorMatrixFilter)
- Convolução (flash.filters.ConvolutionFilter)
- Sombra projetada (flash.filters.DropShadowFilter)
- Brilho (flash.filters.GlowFilter)
- Bisel de gradiente (flash.filters.GradientBevelFilter)
- Brilho de gradiente (flash.filters.GradientGlowFilter)

Depois que o usuário seleciona uma imagem e um filtro a ser aplicado nessa imagem, o aplicativo exibe um painel com controles para definir as propriedades específicas do filtro selecionado. Por exemplo, a imagem a seguir mostra o aplicativo com o filtro de bisel selecionado:



À medida que o usuário ajusta as propriedades de filtro, a visualização é atualizada em tempo real. O usuário também pode aplicar vários filtros personalizando um filtro, clicando no botão Aplicar, personalizando outro filtro, clicando no botão Aplicar e assim por diante.

Existem alguns recursos e limitações nos painéis de filtro do aplicativo:

- O filtro de matriz de cor inclui um conjunto de controles para manipular diretamente propriedades comuns de imagem, como brilho, contrastes, saturação e matiz. Além disso, é possível especificar valores personalizados de matriz de cor.
- O filtro de convolução, que está disponível somente no ActionScript, inclui no conjunto de valores de matriz de convolução usados normalmente ou valores personalizados podem ser especificados. No entanto, enquanto a classe ConvolutionFilter aceita qualquer tamanho de matriz, o aplicativo Filter Workbench usa uma matriz 3 x 3 fixa, o tamanho de filtro usado com mais freqüência.
- Os filtros de mapa de deslocamento e de sombreador, disponíveis somente no ActionScript, não estão presentes no aplicativo Filter Workbench. Um filtro de mapa de deslocamento requer uma imagem do mapa além do conteúdo da imagem filtrada. A imagem do mapa do filtro de mapa de deslocamento é a entrada principal que determina o resultado do filtro de modo que, sem poder carregar ou criar uma imagem do mapa, seria extremamente limitado experimentar o filtro do mapa de deslocamento. Do mesmo modo, um filtro de sombreador requer um arquivo Pixel Bender de código de bytes além do conteúdo da imagem filtrada. Sem poder carregar o código de bytes de sombreador, é impossível usar um filtro de sombreador.

Criação de ocorrências de filtro

O aplicativo Filter Workbench inclui um conjunto de classes, uma para cada filtro disponível, que são usadas por painéis individuais para criar os filtros. Quando o usuário seleciona um filtro, o código do ActionScript associado ao painel de filtro cria uma ocorrência da classe de filtro de fábrica apropriada. Essas classes são conhecidas como *classes de fábrica* porque sua finalidade é criar ocorrências de outros objetos, assim como uma fábrica real cria produtos individuais.

Sempre que o usuário altera um valor de propriedade no painel, o código do painel chama o método adequado na classe de fábrica. Cada classe de fábrica inclui métodos específicos usados pelo painel para criar a ocorrência de filtro adequada. Por exemplo, se o usuário selecionar o filtro de desfoque, o aplicativo criará uma ocorrência de BlurFactory. A classe BlurFactory inclui um método `modifyFilter()` que aceita três parâmetros: `blurX`, `blurY` e `quality`, que são usados em conjunto para criar a ocorrência de BlurFilter desejada:

```
private var _filter:BlurFilter;

public function modifyFilter(blurX:Number = 4, blurY:Number = 4, quality:int = 1):void
{
    _filter = new BlurFilter(blurX, blurY, quality);
    dispatchEvent(new Event(Event.CHANGE));
}
```

Por outro lado, se o usuário selecionar o filtro de convolução, esse filtro dará uma flexibilidade muito maior e, conseqüentemente, terá um conjunto maior de propriedades para controlar. Na classe ConvolutionFactory, o código a seguir é chamado quando o usuário seleciona um valor diferente no painel de filtro:

```
private var _filter:ConvolutionFilter;

public function modifyFilter(matrixX:Number = 0,
                                                matrixY:Number = 0,
                                                matrix:Array = null,
                                                divisor:Number = 1.0,
                                                bias:Number = 0.0,
                                                preserveAlpha:Boolean = true,
                                                clamp:Boolean = true,
                                                color:uint = 0,
                                                alpha:Number = 0.0):void
{
    _filter = new ConvolutionFilter(matrixX, matrixY, matrix, divisor, bias, preserveAlpha,
clamp, color, alpha);
    dispatchEvent(new Event(Event.CHANGE));
}
```

Observe que, em cada caso, quando os valores de filtro são alterados, o objeto de fábrica envia um evento `Event.CHANGE` para notificar os ouvintes que os valores do filtro foram alterados. A classe FilterWorkbenchController, que realmente aplica os filtros no conteúdo filtrado, ouve esse evento para recuperar uma nova cópia do filtro e reaplicá-lo no conteúdo filtrado quando necessário.

A classe FilterWorkbenchController não precisa conhecer os detalhes específicos de cada classe de fábrica de filtro - ela só precisa saber que o filtro foi alterado e acessar uma cópia do filtro. Para dar suporte a isso, o aplicativo inclui uma interface, IFilterFactory, que define o comportamento que uma classe de fábrica de filtro precisa fornecer para que a ocorrência de FilterWorkbenchController do aplicativo possa fazer seu trabalho. A classe IFilterFactory define o método `getFilter`() que é usado na classe FilterWorkbenchController:

```
function getFilter():BitmapFilter;
```

Observe que a definição do método da interface `getFilter()` especifica o retorno de uma ocorrência de BitmapFilter, em vez de um tipo específico de filtro. A classe BitmapFilter não define um tipo específico de filtro. Em vez disso, BitmapFilter é a classe básica a partir da qual todas as classes de filtro são criadas. Cada classe de fábrica de filtro define uma implementação específica do método `getFilter()` que retorna uma referência ao objeto de filtro criado. Por exemplo, veja uma versão abreviada do código-fonte da classe ConvolutionFactory:

```
public class ConvolutionFactory extends EventDispatcher implements IFilterFactory
{
    // ------- Private vars -------
    private var _filter:ConvolutionFilter;
    ...
    // ------- IFilterFactory implementation -------
    public function getFilter():BitmapFilter
    {
        return _filter;
    }
    ...
}
```

Na implementação da classe ConvolutionFactory do método `getFilter()`, é retornada uma ocorrência de ConvolutionFilter, embora nenhum objeto que chame `getFilter()` saiba necessariamente disso - de acordo com a definição do método `getFilter()` seguida por ConvolutionFactory, qualquer ocorrência de BitmapFilter deve ser retornada, a qual pode ser uma ocorrência de qualquer classe de filtro do ActionScript.

Aplicação de filtros em objetos de exibição

Conforme explicado na seção anterior, o aplicativo Filter Workbench usa uma ocorrência da classe FilterWorkbenchController (a partir daqui mencionada como "ocorrência do controlador"), que realmente aplica os filtros no objeto visual selecionado. Antes de aplicar um filtro, a ocorrência do controlador precisa saber em qual imagem ou conteúdo visual o filtro deve ser aplicado. Quando o usuário seleciona uma imagem, o aplicativo chama o método `setFilterTarget()` na classe FilterWorkbenchController, transmitindo uma das constantes definidas na classe ImageType:

```
public function setFilterTarget(targetType:ImageType):void
{
    ...
    _loader = new Loader();
    ...
    _loader.contentLoaderInfo.addEventListener(Event.COMPLETE, targetLoadComplete);
    ...
}
```

Usando essas informações, a ocorrência do controlador carrega o arquivo designado, armazenando-o em uma variável da ocorrência chamada `_currentTarget` assim que o carregamento termina:

```
private var _currentTarget:DisplayObject;

private function targetLoadComplete(event:Event):void
{
    ...
    _currentTarget = _loader.content;
    ...
}
```

Quando o usuário seleciona um filtro, o aplicativo chama o método `setFilter()` da ocorrência do controlador, dando ao controlador uma referência ao objeto de fábrica de filtro relevante, armazenado em uma variável da ocorrência chamada `_filterFactory`.

```
private var _filterFactory:IFilterFactory;

public function setFilter(factory:IFilterFactory):void
{
    ...

    _filterFactory = factory;
    _filterFactory.addEventListener(Event.CHANGE, filterChange);
}
```

Observe que, conforme descrito anteriormente, a ocorrência do controlador não sabe o tipo de dados específico da ocorrência de fábrica de filtro fornecida; ela só sabe que o objeto implementa a ocorrência de IFilterFactory, indicando que tem um método `getFilter()` e envia um evento `change` (`Event.CHANGE`) quando o filtro é alterado.

Quando o usuário altera as propriedades no painel do filtro, a ocorrência do controlador descobre que o filtro foi alterado por meio do evento `change` da fábrica de filtro, que chama o método `filterChange()` da ocorrência do controlador. Esse método, por sua vez, chama o método `applyTemporaryFilter()`:

```
private function filterChange(event:Event):void
{
    applyTemporaryFilter();
}

private function applyTemporaryFilter():void
{
    var currentFilter:BitmapFilter = _filterFactory.getFilter();

    // Add the current filter to the set temporarily
    _currentFilters.push(currentFilter);

    // Refresh the filter set of the filter target
    _currentTarget.filters = _currentFilters;

    // Remove the current filter from the set
    // (This doesn't remove it from the filter target, since
    // the target uses a copy of the filters array internally.)
    _currentFilters.pop();
}
```

A aplicação do filtro no objeto de exibição ocorre no método `applyTemporaryFilter()`. Primeiro, o controlador recupera uma referência ao objeto de filtro chamando o método `getFilter()` da fábrica de filtro.

```
var currentFilter:BitmapFilter = _filterFactory.getFilter();
```

A ocorrência do controlador tem uma variável da ocorrência de Array chamada `_currentFilters`, na qual são armazenados todos os filtros que foram aplicados no objeto de exibição. A próxima etapa é adicionar o filtro recém atualizado a essa matriz:

```
_currentFilters.push(currentFilter);
```

Em seguida, o código atribui a matriz dos filtros à propriedade `filters` do objeto de exibição, que realmente aplica os filtros na imagem:

```
_currentTarget.filters = _currentFilters;
```

Finalmente, como esse filtro adicionado recentemente ainda é o filtro de "trabalho", ele não deve ser aplicado permanentemente no objeto de exibição e, por isso, é removido da matriz `_currentFilters`:

```
_currentFilters.pop();
```

A remoção desse filtro da matriz não afeta o objeto de exibição filtrado porque um objeto de exibição cria uma cópia da matriz de filtros quando esta é atribuída à propriedade `filters` e usa essa matriz interna em vez da original. Desse modo, todas as alterações feitas na matriz dos filtros não afetam o objeto de exibição até que a matriz seja novamente atribuída à propriedade `filters` do objeto de exibição.

Capítulo 17: Trabalho com sombreadores Pixel Bender

O Adobe Pixel Bender Toolkit permite que os desenvolvedores gravem sombreadores para criar efeitos gráficos e executar outro processamento de imagem e dados. O código de bytes do Pixel Bender pode ser executado no ActionScript para aplicar o efeito aos dados da imagem ou ao conteúdo visual. O uso de sombreadores Pixel Bender no ActionScript fornece a capacidade de criar efeitos visuais personalizados e de executar processamento de dados além dos recursos internos do ActionScript.

Noções básicas de sombreadores Pixel Bender

Introdução ao trabalho com sombreadores Pixel Bender

O Adobe Pixel Bender é uma linguagem de programação utilizada para criar ou manipular conteúdo de imagem. Usando o Pixel Bender você cria um kernel, também chamado de sombreador neste documento. O sombreador define uma única função executada individualmente em cada pixel de uma imagem. O resultado de cada chamada à função é a cor gerada nessa coordenada de pixel da imagem. É possível especificar valores de imagens e parâmetros de entrada para personalizar a operação. Em uma única execução de um sombreador, os valores de entrada e parâmetro são constantes. A única coisa variável é a coordenada do pixel cuja cor é o resultado da chamada à função.

Quando possível, a função de sombreador é chamada para várias coordenadas de pixel de saída em paralelo. Isso melhora o desempenho do sombreador e pode oferecer processamento de alto desempenho.

No Flash Player e no Adobe AIR, há três tipos de efeitos que podem ser facilmente criados usando-se um sombreador:

- preenchimento de desenho
- modo de mesclagem
- filtro

Também é possível executar um sombreador no modo autônomo. No modo autônomo, o resultado de um sombreador é acessado diretamente, sem a necessidade de especificar previamente seu uso pretendido. É possível acessar o resultado como dados de imagem ou como dados binários ou numéricos. Não é necessário que os dados sejam dados de imagem. Dessa forma, você pode atribuir a um sombreador um conjunto de dados como entrada. O sombreador processa os dados, e você pode acessar o resultado retornado por ele.

Tarefas comuns do sombreador Pixel Bender

As tarefas a seguir provavelmente serão realizadas ao utilizar filtros no ActionScript:

- Carregar um sombreador em um aplicativo SWF em execução ou incorporá-lo em tempo de compilação e acessá-lo em tempo de execução
- Acesso aos metadados de sombreador
- Identificar e especificar valores para entradas de sombreador (normalmente imagens)
- Identificar e especificar valores para parâmetros de sombreador

- Usar um sombreador das seguintes maneiras:
  - Como preenchimento de desenho
  - Como modo de mesclagem
  - Como filtro
  - No modo autônomo

Conceitos e termos importantes

A lista de referência a seguir contém termos importantes usados neste capítulo:

- Kernel: No Pixel Bender, um kernel é o mesmo que um sombreador. Usando o Pixel Bender, seu código define um kernel, que por sua vez define uma única função que é executada individualmente em cada pixel de uma imagem.
- Código de bytes do Pixel Bender: quando compilado, um kernel do Pixel Bender é transformado em código de bytes do Pixel Bender. O código de bytes é acessado e executado pelo Flash Player ou Adobe AIR em tempo de execução.
- Linguagem Pixel Bender: a linguagem de programação usada para criar um kernel do Pixel Bender.
- Pixel Bender Toolkit: o aplicativo usado para criar um arquivo de código de bytes do Pixel Bender com base no código-fonte do Pixel Bender. O Toolkit permite escrever, testar e compilar o código-fonte do Pixel Bender.
- Sombreador: neste documento, um sombreador é um conjunto de funcionalidades criado na linguagem Pixel Bender. O código de um sombreador cria um efeito visual ou executa um cálculo. Em qualquer um dos casos, o sombreador retorna um conjunto de dados (normalmente, os pixels de uma imagem). O sombreador executa a mesma operação em cada ponto de dados, a única diferença são as coordenadas do pixel de saída.

  O sombreador não é criado no ActionScript. Ele é criado na linguagem Pixel Bender e compilado no código de bytes do Pixel Bender. Ele pode ser incorporado a um arquivo SWF em tempo de compilação ou carregado como arquivo externo em tempo de execução. Nos dois casos, ele é acessado no ActionScript através da criação de um objeto Shader e da vinculação deste ao código de bytes do sombreador.
- Entrada do sombreador: uma entrada complexa, geralmente dados de imagem de bitmap, fornecida para um sombreador para uso em seus cálculos. Para cada variável de entrada definida em um sombreador, é usado um único valor (isto é, uma única imagem ou um conjunto de dados binários) para a execução inteira do sombreador.
- Parâmetro de sombreador: um único valor (ou conjunto limitado de valores) que é fornecido para um sombreador usá-lo em seus cálculos. Cada valor de parâmetro é definido para uma única execução do sombreador, e o mesmo valor é usado durante toda a execução.

Teste dos exemplos do capítulo

Talvez você queira testar as listagens de código de exemplo fornecidas durante a leitura deste capítulo. Como este capítulo trata da criação e da manipulação de conteúdo visual, o teste do código envolve a execução do código e a visualização dos resultados no SWF criado. Todos os exemplos criam conteúdo usando a API de desenho que utiliza ou é modificada pelo efeito de sombreador.

A maioria das listagens de código de exemplo têm duas partes. Uma parte é o código-fonte do Pixel Bender referente ao sombreador usado no exemplo. Primeiro você deve usar o Pixel Bender Toolkit para compilar o código-fonte em um arquivo de código de bytes do Pixel Bender. Siga estas etapas para criar o arquivo de código de bytes do Pixel Bender:

1. Abra o Adobe Pixel Bender Toolkit. Se necessário, no menu Criar, escolha "Ativar avisos e erros do Flash Player".
2. Copie a listagem de código do Pixel Bender e cole-a no painel do editor de código do Pixel Bender Toolkit.
3. No menu Arquivo, escolha "Exportar filtro de kernel do Flash Player".

4 Salve o arquivo do código de bytes do Pixel Bender no mesmo diretório que o documento do Flash. O nome do arquivo deve ser igual ao nome especificado na descrição do exemplo.

A parte do ActionScript de cada exemplo é escrita como arquivo de classe. Para testar o exemplo:

1 Crie um documento Flash vazio e salve-o no seu computador.

2 Crie e salve um novo arquivo do ActionScript no mesmo diretório do documento Flash. O nome do arquivo deve corresponder ao nome da classe na listagem de código. Por exemplo, se a listagem de código define uma classe chamada MyApplication, use o nome MyApplication.as para salvar o arquivo do ActionScript.

3 Copie a listagem de código no arquivo do ActionScript e salve o arquivo.

4 No documento Flash, clique em uma parte em branco do Palco ou espaço de trabalho para ativar o Inspetor de propriedades do documento.

5 No Inspetor de propriedades no campo Classe do documento, digite o nome da classe do ActionScript que você copiou do texto.

6 Execute o programa usando Controlar > Testar filme.

Você verá os resultados do exemplo na janela de visualização.

Essas técnicas para testar listagens de código de exemplo são detalhadas em "Teste de listagens de código de exemplo dos capítulos" na página 36.

Carregamento ou incorporação de um sombreador

A primeira etapa do uso de um sombreador Pixel Bender no ActionScript é obter acesso ao sombreador no seu código ActionScript. Como um sombreador é criado usando o Adobe Pixel Bender Toolkit e gravado na linguagem Pixel Bender, ele não pode ser acessado diretamente no ActionScript. Em vez disso, crie uma ocorrência da classe Shader que represente o sombreador Pixel Bender para o ActionScript. O objeto Shader permite localizar informações sobre o sombreador, como por exemplo, se ele espera parâmetros ou valores da imagem de entrada. Você passa o objeto Shader para outros objetos para realmente usar o sombreador. Por exemplo, para usar o sombreador como um filtro, você atribui o objeto Shader à propriedade `shader` de um objeto ShaderFilter. Como alternativa, para usar o sombreador como um preenchimento de desenho, você passa o objeto Shader como um argumento para o método `Graphics.beginShaderFill()`.

Seu código ActionScript pode acessar um sombreador criado pelo Adobe Pixel Bender Toolkit (um arquivo .pbj) de duas maneiras:

- Carregado em tempo de execução: o arquivo sombreador pode ser carregado como um ativo externo usando um objeto URLLoader. Essa técnica é semelhante ao carregamento de um ativo externo, como um arquivo de texto. O exemplo a seguir demonstra o carregamento de um arquivo de código de bytes do sombreador em tempo de execução e sua vinculação com uma ocorrência de Shader:

```
var loader:URLLoader = new URLLoader();
loader.dataFormat = URLLoaderDataFormat.BINARY;
loader.addEventListener(Event.COMPLETE, onLoadComplete);
loader.load(new URLRequest("myShader.pbj"));

var shader:Shader;

function onLoadComplete(event:Event):void {
    // Create a new shader and set the loaded data as its bytecode
    shader = new Shader();
    shader.byteCode = loader.data;

    // You can also pass the bytecode to the Shader() constructor like this:
    // shader = new Shader(loader.data);

     // do something with the shader
}
```

- Incorporado no arquivo SWF: o arquivo sombreador pode ser incorporado no arquivo SWF em tempo de compilação usando a tag de metadados `[Embed]`. A tag de metadados `[Embed]` estará disponível apenas se você usar o SDK do Flex para compilar o arquivo SWF. O parâmetro `source` da tag `[Embed]` aponta para o arquivo sombreador, e seu parâmetro `mimeType` é o `"application/octet-stream"`, como neste exemplo:

```
[Embed(source="myShader.pbj", mimeType="application/octet-stream)]
var MyShaderClass:Class;

// ...

// create a shader and set the embedded shader as its bytecode
var shaderShader = new Shader();
shader.byteCode = new MyShaderClass();

// You can also pass the bytecode to the Shader() constructor like this:
// var shader:Shader = new Shader(new MyShaderClass());

// do something with the shader
```

Nos dois casos, você vincula o código de bytes brutos do sombreador (a propriedade `URLLoader.data` ou uma ocorrência da classe de dados `[Embed]`) com a ocorrência de Shader. Como demonstram os exemplos anteriores, você pode atribuir o código de bytes à ocorrência de Shader de duas maneiras. Você pode passar o código de bytes do sombreador como um argumento para o construtor `Shader()`. Você também pode defini-lo como a propriedade `byteCode` da ocorrência de Shader.

Depois que um sombreador Pixel Bender foi criado e vinculado com um objeto Shader, você pode usá-lo para criar efeitos de várias maneiras. Pode usá-lo como um filtro, um modo de mesclagem, um preenchimento bitmap ou para processamento autônomo de bitmap ou de outros dados. Você pode usar também a propriedade `data` do objeto Shader para acessar os metadados de sombreador, especificar imagens de entrada e definir valores de parâmetros.

Acesso aos metadados do sombreador

Ao criar um kernel de sombreador Pixel Bender, o autor pode especificar metadados sobre o sombreador no código fonte Pixel Bender. Ao usar um sombreador no ActionScript, você pode examiná-lo e extrair seus metadados.

Ao criar uma ocorrência de Shader e vinculá-la a um sombreador Pixel Bender, um objeto ShaderData que contém dados sobre o sombreador é criado e armazenado na propriedade `data` do objeto Shader. A classe ShaderData não define nenhuma propriedade própria. No entanto, em tempo de execução uma propriedade é adicionada dinamicamente ao objeto ShaderData para cada valor de metadados definido no código fonte do sombreador. O nome fornecido a cada propriedade é igual ao nome especificado nos metadados. Por exemplo, suponha que o código fonte de um sombreador Pixel Bender inclua a seguinte definição de metadados:

```
namespace : "Adobe::Example";
vendor : "Bob Jones";
version : 1;
description : "Creates a version of the specified image with the specified brightness.";
```

O objeto ShaderData criado para esse sombreador é criado com as seguintes propriedades e valores:

- `namespace` (String): `"Adobe::Example"`
- `vendor` (String): `"Bob Jones"`
- `version` (String): `"1"`
- `description` (String): `"Cria uma versão da imagem especificada com o brilho especificado"`

Como as propriedades de metadados são adicionadas dinamicamente ao objeto ShaderData, você pode usar um loop `for..in` para examinar o objeto ShaderData. Usando essa técnica é possível descobrir se o sombreador tem algum metadado e quais são os seus valores. Além das propriedades de metadados, um objeto ShaderData pode ter propriedades que representam entradas e parâmetros definidos no sombreador. Ao usar um loop `for..in` para examinar um objeto ShaderData, verifique o tipo de dados de cada propriedade para determinar se a propriedade é uma entrada (uma ocorrência de ShaderInput), um parâmetro (uma ocorrência de ShaderParameter) ou um valor de metadados (uma ocorrência de String). O exemplo a seguir mostra como usar um loop `for..in` para examinar as propriedades dinâmicas de uma propriedade `data` do sombreador. Cada valor de metadados é adicionado a uma ocorrência de Vector denominada `metadata`. Observe que este exemplo assume que uma ocorrência de Shader denominada `myShader` já foi criada:

```
var shaderData:ShaderData = myShader.data;
var metadata:Vector.<String> = new Vector.<String>();

for (var prop:String in shaderData)
{
    if (!(shaderData[prop] is ShaderInput) && !(shaderData[prop] is ShaderParameter))
    {
        metadata[metadata.length] = shaderData[prop];
    }
}

// do something with the metadata
```

Para obter uma versão desse exemplo que também extrai entradas e parâmetros de sombreador, consulte "Identificação de entradas e parâmetros de sombreador" na página 391. Para obter mais informações sobre propriedades de entrada e parâmetro, consulte "Especificação de valores de entrada e de parâmetro de sombreador" na página 391.

Especificação de valores de entrada e de parâmetro de sombreador

Muitos sombreadores Pixel Bender são definidos para usar uma ou mais imagens de entrada que são usadas no processamento do sombreador. Por exemplo, é comum um sombreador aceitar uma imagem de origem e produzi-la com um efeito específico aplicado a ela. Dependendo de como o sombreador é usado, o valor de entrada pode ser especificado automaticamente ou você pode precisar fornecer um valor explicitamente. De modo semelhante, muitos sombreadores especificam parâmetros que são usados para personalizar a saída do sombreador. Você também deve definir explicitamente um valor para cada parâmetro antes de usar o sombreador.

Você usa a propriedade `data` do objeto Shader para definir entradas e parâmetros do sombreador e para determinar se um sombreador específico espera entradas ou parâmetros. A propriedade `data` é uma ocorrência de ShaderData.

Identificação de entradas e parâmetros de sombreador

A primeira etapa da especificação de valores de entrada e de parâmetro de sombreador é descobrir se o sombreador específico que você está usando espera qualquer imagem ou parâmetro de entrada. Cada ocorrência de Shader tem uma propriedade `data` que contém um objeto ShaderData. Se o sombreador definir quaisquer entradas ou parâmetros, eles serão acessados como propriedades desse objeto ShaderData. Os nomes de propriedades correspondem aos nomes especificados para as entradas e parâmetros no código fonte do sombreador. Por exemplo, se um sombreador definir uma entrada denominada `src`, o objeto ShaderData terá uma propriedade denominada `src` que representa essa entrada. Cada propriedade que representa uma entrada é uma ocorrência de ShaderInput, e cada propriedade que representa um parâmetro é uma ocorrência de ShaderParameter.

O ideal é que o autor do sombreador forneça documentação do sombreador, indicando quais valores de imagens de entrada e de parâmetros o sombreador espera, o que eles representam, os valores apropriados, e assim por diante.

No entanto, se o sombreador não estiver documentado (e você não tiver seu código fonte), você poderá inspecionar os dados do sombreador para identificar as entradas e parâmetros. As propriedades que representam entradas e parâmetros são adicionadas dinamicamente ao objeto ShaderData. Conseqüentemente, você pode usar um loop `for..in` para examinar o objeto ShaderData para descobrir se o sombreador associado define quaisquer entradas ou parâmetros. Conforme descrito em "Acesso aos metadados do sombreador" na página 389, qualquer valor de metadados definido para um sombreador também é acessado como uma propriedade dinâmica adicionada à propriedade `Shader.data`. Ao usar essa técnica para identificar entradas e parâmetros de sombreador, verifique o tipo de dados das propriedades dinâmicas. Se uma propriedade for uma ocorrência de ShaderInput, ela representará uma entrada. Se ela for uma ocorrência de ShaderParameter, ela representará um parâmetro. Caso contrário, ela será um valor de metadados. O exemplo a seguir mostra como usar um loop `for..in` para examinar as propriedades dinâmicas de uma propriedade `data` do sombreador. Cada entrada (objeto ShaderInput) é adicionada a uma ocorrência de Vector denominada `inputs`. Cada parâmetro (objeto ShaderParameter) é adicionado a uma ocorrência de Vector denominada `parameters`. Finalmente, quaisquer propriedades de metadados são adicionadas a uma ocorrência de Vector denominada `metadata`. Observe que este exemplo assume que uma ocorrência de Shader denominada `myShader` já foi criada:

```
var shaderData:ShaderData = myShader.data;
var inputs:Vector.<ShaderInput> = new Vector.<ShaderInput>();
var parameters:Vector.<ShaderParameter> = new Vector.<ShaderParameter>();
var metadata:Vector.<String> = new Vector.<String>();

for (var prop:String in shaderData)
{
    if (shaderData[prop] is ShaderInput)
    {
        inputs[inputs.length] = shaderData[prop];
    }
    else if (shaderData[prop] is ShaderParameter)
    {
        parameters[parameters.length] = shaderData[prop];
    }
    else
    {
        metadata[metadata.length] = shaderData[prop];
    }
}

// do something with the inputs or properties
```

Especificação de valores de entrada de sombreador

Muitos sombreadores esperam uma ou mais imagens de entrada que são usadas no processamento do sombreador. No entanto, em muitos casos, uma entrada é especificada automaticamente quando o objeto Shader é usado. Por exemplo, suponha que um sombreador exija uma entrada e seja usado como um filtro. Quando o filtro é aplicado a um objeto de exibição ou a um objeto BitmapData, esse objeto é definido automaticamente como a entrada. Nesse caso você não define explicitamente um valor de entrada.

No entanto, em alguns casos, especialmente se um sombreador definir várias entradas, você definirá explicitamente um valor para uma entrada. Cada entrada definida em um sombreador é representada no ActionScript por um objeto ShaderInput. O objeto ShaderInput é uma propriedade da ocorrência de ShaderData na propriedade `data` do objeto Shader, conforme descrito em "Identificação de entradas e parâmetros de sombreador" na página 391. Por exemplo, suponha que um sombreador defina uma entrada denominada `src` e que o sombreador esteja vinculado a um objeto Shader denominado `myShader`. Nesse caso, você acessa o objeto ShaderInput que corresponde à entrada `src` usando o seguinte identificador:

```
myShader.data.src
```

Cada objeto ShaderInput tem uma propriedade `input` usada para definir o valor da entrada. Você define a propriedade `input` como uma ocorrência de BitmapData para especificar dados da imagem. Você também pode definir a propriedade `input` como uma ocorrência de BitmapData ou Vector.<Número> para especificar dados binários ou numéricos. Para obter detalhes e restrições sobre o uso de uma ocorrência de BitmapData ou Vector.<Número> como uma entrada, consulte a listagem `ShaderInput.input` na referência de linguagem.

Além da propriedade `input`, um objeto ShaderInput tem propriedades que podem ser usadas para determinar qual tipo de imagem a entrada espera. Essas propriedades incluem as propriedades `width`, `height` e `channels`. Cada objeto ShaderInput também tem uma propriedade `index` que é útil para determinar se um valor explícito deve ser fornecido para a entrada. Se um sombreador esperar mais entradas do que o número definido automaticamente, defina valores para essas entradas. Para obter detalhes sobre as diferentes maneiras de uso de um sombreador, e se os valores de entrada são definidos automaticamente, consulte "Uso de um sombreador" na página 396.

Especificação de valores de parâmetros de sombreador

Alguns sombreadores definem valores de parâmetros que o sombreador usa para criar seu resultado. Por exemplo, um sombreador que altera o brilho de uma imagem pode especificar um parâmetro de brilho que determina o quanto a operação afeta o brilho. Um único parâmetro definido em um sombreador pode esperar um único valor ou vários valores, de acordo com a definição do parâmetro no sombreador. Cada parâmetro definido em um sombreador é representado no ActionScript por um objeto ShaderParameter. O objeto ShaderParameter é uma propriedade da ocorrência de ShaderData na propriedade de dados do objeto Shader, conforme descrito em "Identificação de entradas e parâmetros de sombreador" na página 391. Por exemplo, suponha que um sombreador defina um parâmetro denominado `brightness` e que o sombreador esteja representado por um objeto Shader denominado `myShader`. Nesse caso, você acessa o ShaderParameter que corresponde ao parâmetro `brightness` usando o seguinte identificador:

```
myShader.data.brightness
```

Para definir um valor (ou valores) para o parâmetro, crie uma matriz do ActionScript que contenha o valor ou valores e atribua essa matriz à propriedade `value` do objeto ShaderParameter. A propriedade `value` é definida como uma ocorrência de Array porque é possível que um único parâmetro do sombreador exija vários valores. Mesmo que o parâmetro do sombreador espere apenas um único valor, você deve delimitar o valor em um objeto Array para atribuí-lo à propriedade `ShaderParameter.value`. A listagem a seguir demonstra como configurar um único valor como a propriedade `value`:

```
myShader.data.brightness.value = [75];
```

Se o código fonte de Pixel Bender do sombreador definir um valor padrão para o parâmetro, uma matriz que contém o valor ou valores padrão será criada e atribuída à propriedade `value` do objeto ShaderParameter quando o objeto Shader for criado. Depois que a matriz foi atribuída à propriedade `value` (inclusive se ela for a matriz padrão), o valor do parâmetro poderá ser alterado com a alteração do valor do elemento de matriz. Você não precisa criar uma nova matriz e atribuí-la à propriedade `value`.

O exemplo a seguir demonstra a configuração de um valor do parâmetro do sombreador no ActionScript. Nesse exemplo o sombreador define um parâmetro denominado `color`. O parâmetro `color` é declarado como uma variável `float4` no código fonte do Pixel Bender, o que significa que ela é uma matriz de quatro números de ponto flutuante. No exemplo, o valor do parâmetro `color` é alterado continuamente e, a cada vez que ele é alterado, o sombreador é usado para desenhar um retângulo colorido na tela. O resultado é uma alteração de cor animada.

Nota: *O código para este exemplo foi escrito por Ryan Taylor. Obrigado por compartilhar este exemplo, Ryan. Para ver o portfólio de Ryan e ler seus artigos, acesse www.boostworthy.com/.*

O código ActionScript está centralizado em torno de três métodos:

- `init()`: No método `init()`, o código carrega o arquivo de código de bytes do Pixel Bender que contém o sombreador. Quando o arquivo é carregado, o método `onLoadComplete()` é chamado.
- `onLoadComplete()`: No método `onLoadComplete()`, o código cria o objeto Shader chamado `shader`. Ele também cria uma ocorrência de Sprite chamada `texture`. No método `renderShader()`, o código desenha o resultado do sombreador em `texture` uma vez por quadro.
- `onEnterFrame()`: O método `onEnterFrame()` é chamado uma vez por quadro criando o efeito de animação. Nesse método, o código define o valor do parâmetro do sombreador como a nova cor e, em seguida, chama o método `renderShader()` para desenhar o resultado do sombreador como um retângulo.
- `renderShader()`: No método `renderShader()`, o código chama o método `Graphics.beginShaderFill()` para especificar um preenchimento do sombreador. Em seguida, ele desenha um retângulo cujo preenchimento é definido pela saída do sombreador (a cor gerada). Para obter mais informações sobre como usar um sombreador desta maneira, consulte "Uso de um sombreador como um preenchimento de desenho" na página 396.

Veja abaixo o código ActionScript para este exemplo. Use esta classe com a classe de aplicativo principal em um projeto somente ActionScript no Flex ou como a classe do documento para o arquivo FLA na Ferramenta de autoria do Flash:

```
package
{
    import flash.display.Shader;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class ColorFilterExample extends Sprite
    {
        private const DELTA_OFFSET:Number = Math.PI * 0.5;
        private var loader:URLLoader;
        private var shader:Shader;
        private var texture:Sprite;
        private var delta:Number = 0;

        public function ColorFilterExample()
        {
            init();
        }

        private function init():void
        {
            loader = new URLLoader();
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("ColorFilter.pbj"));
        }

        private function onLoadComplete(event:Event):void
        {
            shader = new Shader(loader.data);

            shader.data.point1.value = [topMiddle.x, topMiddle,y];
            shader.data.point2.value = [bottomLeft.x, bottomLeft.y];
            shader.data.point3.value = [bottomRight.x, bottomRight.y];
            texture = new Sprite();

            addChild(texture);

            addEventListener(Event.ENTER_FRAME, onEnterFrame);
        }
        private function onEnterFrame(event:Event):void
        {
```

```
            shader.data.color.value[0] = 0.5 + Math.cos(delta - DELTA_OFFSET) * 0.5;
            shader.data.color.value[1] = 0.5 + Math.cos(delta) * 0.5;
            shader.data.color.value[2] = 0.5 + Math.cos(delta + DELTA_OFFSET) * 0.5;
            // The alpha channel value (index 3) is set to 1 by the kernel's default
            // value. This value doesn't need to change.

            delta += 0.1;

            renderShader();
        }

        private function renderShader():void
        {
            texture:graphics.clear();
            texture.graphics.beginShaderFill(shader);
            texture.graphics.drawRect(0, 0, stage.stageWidth, stage.stageHeight);
            texture.graphics.endFill();
        }
    }
}
```

Este é o código-fonte para o kernel de sombreador ColorFilter, usado para criar o arquivo de código de bytes do Pixel Bender "ColorFilter.pbj":

```
<languageVersion : 1.0;>
kernel ColorFilter
<
    namespace : "boostworthy::Example";
    vendor : "Ryan Taylor";
    version : 1;
    description : "Creates an image where every pixel has the specified color value.";
>
{
    output pixel4 result;

    parameter float4 color
    <
        minValue:float4(0, 0, 0, 0);
        maxValue:float4(1, 1, 1, 1);
        defaultValue:float4(0, 0, 0, 1);
    >;

    void evaluatePixel()
    {
        result = color;
    }
}
```

Se você estiver usando um sombreador cujos parâmetros não estão documentados, poderá calcular quantos elementos de qual tipo devem ser incluídos na matriz, marcando a propriedade `type` do objeto ShaderParameter. A propriedade `type` indica o tipo de dados do parâmetro, conforme definido no próprio sombreador. Para obter uma lista de números e tipos de elementos esperados por cada tipo de parâmetro, consulte as listagens de propriedades `ShaderParameter.value` na referência de linguagem.

Cada objeto ShaderParameter também tem uma propriedade `index` que indica onde o parâmetro se ajusta na ordem dos parâmetros do sombreador. Além dessas propriedades, um objeto ShaderParameter pode ter propriedades adicionais que contêm valores de metadados fornecidos pelo autor do sombreador. Por exemplo, o autor pode especificar valores de metadados, como valores mínimos, máximos e padrão para um parâmetro. Todos os valores de metadados especificados pelo autor são adicionados ao objeto ShaderParameter como propriedades dinâmicas. Para examinar essas propriedades, use um loop `for..in` para executar um loop nas propriedades dinâmicas do objeto ShaderParameter para identificar seus metadados. O exemplo a seguir mostra como usar um loop `for..in` para identificar metadados de um objeto ShaderParameter. Cada valor de metadados é adicionado a uma ocorrência de Vector denominada `metadata`. Observe que este exemplo assume que uma ocorrência de Shader denominada `myShader` já foi criada, e que é conhecida como tendo um parâmetro denominado `brightness`:

```
var brightness:ShaderParameter = myShader.data.brightness;
var metadata:Vector.<String> = new Vector.<String>();

for (var prop:String in brightness)
{
    if (brightness[prop] is String)
    {
        metadata[metadata.length] = brightness[prop];
    }
}

// do something with the metadata
```

Uso de um sombreador

Depois que o sombreador Pixel Bender está disponível no ActionScript como um objeto Shader, ele pode ser usado de várias maneiras:

- Preenchimento de desenho do sombreador: O sombreador define a parte de preenchimento de uma forma desenhada usando a API de desenho
- Modo de mesclagem: O sombreador define a mesclagem entre dois objetos de exibição sobrepostos
- Filtro: O sombreador define um filtro que modifica a aparência do conteúdo visual
- Processamento autônomo do sombreador: O processamento do sombreador é executado sem especificar o uso pretendido da saída. Como opção, o sombreador pode ser executado no plano de fundo, com o resultado disponível quando o processamento é concluído. Essa técnica pode ser usada para gerar dados de bitmap e também para processar dados não visuais.

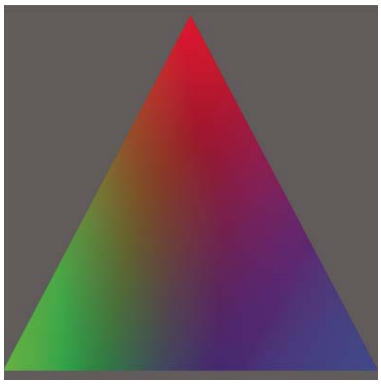
Uso de um sombreador como um preenchimento de desenho

Ao usar um sombreador para criar um preenchimento de desenho, você deve utilizar os métodos de API de desenho para criar uma forma vetorial. A saída do sombreador é usada para preencher a forma, da mesma maneira que se pode usar qualquer imagem de bitmap como preenchimento de bitmap com a API de desenho. Para criar um preenchimento de desenho, no ponto do código em que você deseja começar a desenhar a forma, chame o método `beginShaderFill()` do objeto Graphics. Passe o objeto Shader como o primeiro argumento para o método `beginShaderFill()`, conforme mostrado nesta listagem:

```
var canvas:Sprite = new Sprite();
canvas.graphics.beginShaderFill(myShader);
canvas.graphics.drawRect(10, 10, 150, 150);
canvas.graphics.endFill();
// add canvas to the display list to see the result
```

Quando você usa um sombreador como preenchimento de desenho, deve definir quaisquer valores de imagem de entrada e valores de parâmetro exigidos pelo sombreador.

O exemplo a seguir demonstra o uso de um sombreador como preenchimento de desenho. Neste exemplo, o sombreador cria um gradiente de três pontas. Esse gradiente tem três cores, cada uma na ponta de um triângulo, com uma mesclagem de gradiente entre elas. Além disso, as cores se revezam para criar um efeito animado de cores girando.



Nota: *O código deste exemplo foi escrito por Petri Leskinen. Obrigado por compartilhar este exemplo, Petri. Para ver mais exemplos e tutoriais de Petri, acesse http://pixelero.wordpress.com/.*

O código ActionScript está baseado em três métodos:

- `init()`: O método `init()` é chamado quando o aplicativo é carregado. Neste método, o código define os valores iniciais para os objetos Point que representam as pontas do triângulo. O código também cria uma ocorrência de Sprite chamada `canvas`. Posteriormente, no método `updateShaderFill()`, o código desenha o resultado do sombreador em `canvas` uma vez por quadro. Por último, o código carrega o arquivo de código de bytes do sombreador.
- `onLoadComplete()`: No método `onLoadComplete()`, o código cria o objeto Shader chamado `shader`. Ele também define os valores de parâmetro iniciais. Para terminar, o código adiciona o método `updateShaderFill()` como um ouvinte do evento `enterFrame`, o que significa que ele é chamado uma vez por quadro para criar um efeito de animação.
- `updateShaderFill()`: O método `updateShaderFill()` é chamado uma vez por quadro, criando o efeito de animação. Neste método, o código calcula e define os valores dos parâmetros do sombreador. O código então chama o método `beginShaderFill()` para criar um preenchimento de sombreador e chama outros métodos de API de desenho para desenhar o resultado do sombreador em um triângulo.

Veja abaixo o código ActionScript para este exemplo. Use esta classe com a classe de aplicativo principal em um projeto somente ActionScript no Flex ou como a classe do documento para o arquivo FLA na Ferramenta de autoria do Flash:

```
package
{
    import flash.display.Shader;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.geom.Point;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class ThreePointGradient extends Sprite
    {
        private var canvas:Sprite;
        private var shader:Shader;
        private var loader:URLLoader;

        private var topMiddle:Point;
        private var bottomLeft:Point;
        private var bottomRight:Point;

        private var colorAngle:Number = 0.0;
        private const d120:Number = 120 / 180 * Math.PI; // 120 degrees in radians


        public function ThreePointGradient()
        {
            init();
        }

        private function init():void
        {
            canvas = new Sprite();
            addChild(canvas);

            var size:int = 400;
            topMiddle = new Point(size / 2, 10);
            bottomLeft = new Point(0, size - 10);
            bottomRight = new Point(size, size - 10);

            loader = new URLLoader();
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("ThreePointGradient.pbj"));
        }

        private function onLoadComplete(event:Event):void
        {
            shader = new Shader(loader.data);

            shader.data.point1.value = [topMiddle.x, topMiddle,y];
            shader.data.point2.value = [bottomLeft.x, bottomLeft.y];
            shader.data.point3.value = [bottomRight.x, bottomRight.y];

            addEventListener.Event.ENTER_FRAME, updateShaderFill);
        }

        private function updateShaderFill(event:Event):void
```

```
        {
            colorAngle += .06;

            var c1:Number = 1 / 3 + 2 / 3 * Math.cos(colorAngle);
            var c2:Number = 1 / 3 + 2 / 3 * Math.cos(colorAngle + d120);
            var c3:Number = 1 / 3 + 2 / 3 * Math.cos(colorAngle - d120;

            shader.data.color1.value = [c1, c2, c3, 1.0];
            shader.data.color2.value = [c3, c1, c2, 1.0];
            shader.data.color3.value = [c2, c3, c1, 1.0];

            canvas.graphics.clear();
            canvas.graphics.beginShaderFill(shader);

            canvas.graphics.moveTo(topMiddle.x, topMiddle.y);
            canvas.graphics.lineTo(bottomLeft.x, bottomLeft.y);
            canvas.graphics.lineTo(bottomRight.x, bottomLeft.y);

            canvas.graphics.endFill();
        }
    }
}
```

Este é o código-fonte para o kernel de sombreador ThreePointGradient, usado para criar o arquivo de código de bytes do Pixel Bender "ThreePointGradient.pbj":

```
<languageVersion : 1.0;>
kernel ThreePointGradient
<
    namespace : "Petri Leskinen::Example";
    vendor : "Petri Leskinen";
    version : 1;
    description : "Creates a gradient fill using three specified points and colors.";
>
{
    parameter float2 point1 // coordinates of the first point
    <
        minValue:float2(0, 0);
        maxValue:float2(4000, 4000);
    >;

    parameter float4 color1 // color at the first point, opaque red by default
    <
        defaultValue:float4(1.0, 0.0, 0.0, 1.0);
    >;

    parameter float2 point2 // coordinates of the second point
    <
        minValue:float2(0, 0);
        maxValue:float2(4000, 4000);
    >;

    parameter float4 color2 // color at the second point, opaque green by default
    <
        defaultValue:float4(0.0, 1.0, 0.0, 1.0);
    >;
```

```
    parameter float2 point3 // coordinates of the third point
    <
        minValue:float2(0, 0);
        maxValue:float2(4000, 4000);
    >;

    parameter float4 color3 // color at the third point, opaque blue by default
    <
        defaultValue:float4(0.0, 0.0, 1.0, 1.0);
    >;

    output pixel4 dst;

    void evaluatePixel()
    {
        float d2 = point2 - point1;
        float d3 = point3 - point1;

        // transformation to a new coordinate system
        // transforms point 1 to origin, point2 to (1, 0), and point3 to (0, 1)
        float2x2 mtrx = float2x2(d3.y, -d2.y, -d3.x, d2.x) / (d2.x * d3.y - d3.x * d2.y);
        float2 pNew = mtrx * (outCoord() - point1);

        // repeat the edge colors on the outside
        pNex.xy = clamp(pNew.xy, 0.0, 1.0); // set the range to 0.0 ... 1.0

        // interpolating the output color or alpha value
        dst = mix(mix(color1, color2, pNex.x), color3, pNew.y);
    }
}
```

Para obter mais informações sobre como desenhar formas usando a API de desenho, consulte "Uso de objetos visuais" na página 322.

Uso de um sombreador como modo de mesclagem

O uso de um sombreador como modo de mesclagem é parecido com o uso de outros modos de mesclagem. O sombreador define a aparência resultante da mesclagem visual de dois objetos de exibição. Para usar um sombreador como modo de mesclagem, atribua o objeto Shader à propriedade `blendShader` do objeto de exibição em primeiro plano. A atribuição de um valor diferente de `null` à propriedade `blendShader` automaticamente define a propriedade `blendMode` do objeto de exibição como `BlendMode.SHADER`. A listagem a seguir demonstra o uso de um sombreador como modo de mesclagem. Observe que este exemplo pressupõe a existência de um objeto de exibição chamado `foreground` contido no mesmo pai na lista de exibição que outro conteúdo de exibição, com `foreground` se sobrepondo ao outro conteúdo:

```
foreground.blendShader = myShader;
```

Quando você usa um sombreador como modo de mesclagem, ele deve ser definido com pelo menos duas entradas. Como mostra o exemplo, você não define os valores de entrada no código. Em vez disso, as duas imagens mescladas são usadas automaticamente como entradas do sombreador. A imagem em primeiro plano é definida como a segunda imagem. (Este é o objeto de exibição ao qual o modo de mesclagem é aplicado.) Uma imagem em segundo plano é criada colocando-se a composição de todos os pixels atrás da caixa delimitadora da imagem em primeiro plano. Essa imagem em segundo plano é definida como a primeira imagem de entrada. Se usar um sombreador que espera mais de duas entradas, especifique um valor para qualquer entrada além das duas primeiras.

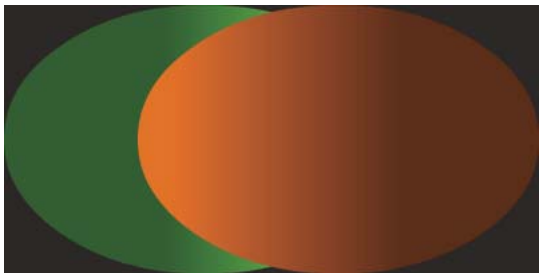
O exemplo a seguir demonstra o uso de um sombreador como modo de mesclagem. Este exemplo utiliza um modo de mesclagem de clareamento baseado na luminosidade. O resultado da mesclagem é que o valor de pixel mais claro de qualquer um dos objetos mesclados se torna o pixel que é exibido.

Nota: *O código deste exemplo foi escrito por Mario Klingemann. Obrigado por compartilhar este exemplo, Mario. Para conhecer mais sobre o trabalho de Mario e ler seus artigos, acesse www.quasimondo.com/.*

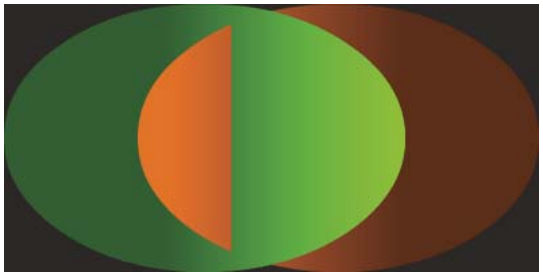
O código ActionScript importante está nestes dois métodos:

- `init()`: O método `init()` é chamado quando o aplicativo é carregado. Neste método, o código carrega o arquivo de código de bytes do sombreador.
- `onLoadComplete()`: No método `onLoadComplete()`, o código cria o objeto Shader chamado `shader`. Em seguida, ele desenha três objetos. O primeiro, `backdrop`, é um plano de fundo cinza escuro atrás de objetos mesclados. O segundo, `backgroundShape`, é uma elipse gradiente verde. O terceiro objeto, `foregroundShape`, é uma elipse gradiente cor de laranja.

  A elipse `foregroundShape` é o objeto de primeiro plano da mesclagem. A imagem de fundo da mesclagem é formada pela parte de `backdrop` e a parte de `backgroundShape` que são sobrepostas pela caixa delimitadora do objeto `foregroundShape`. O objeto `foregroundShape` é o objeto no início da lista de exibição. Ele sobrepõe `backgroundShape` parcialmente e `backdrop` completamente. Por causa dessa sobreposição, sem um modo de mesclagem aplicado, a elipse laranja (`foregroundShape`) é exibida por completo e parte da elipse verde (`backgroundShape`) é ocultada por ela:



  No entanto, com o modo de mesclagem aplicado, a parte mais brilhante da elipse verde "transparece" porque é mais clara do que a parte de `foregroundShape` que se sobrepõe a ela:



Veja abaixo o código ActionScript para este exemplo. Use esta classe com a classe de aplicativo principal em um projeto somente ActionScript no Flex ou como a classe do documento para o arquivo FLA na Ferramenta de autoria do Flash:

```
package
{
    import flash.display.BlendMode;
    import flash.display.GradientType;
    import flash.display.Graphics;
    import flash.display.Shader;
    import flash.display.Shape;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.geom.Matrix;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class LumaLighten extends Sprite
    {
        private var shader:Shader;
        private var loader:URLLoader;

        public function LumaLighten()
        {
            init();
        }

        private function init():void
        {
            loader = new URLLoader();
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("LumaLighten.pbj"));
        }


        private function onLoadComplete(event:Event):void
        {
            shader = new Shader(loader.data);

            var backdrop:Shape = new Shape();
            var g0:Graphics = backdrop.graphics;
            g0.beginFill(0x303030);
            g0.drawRect(0, 0, 400, 200);
            g0.endFill();
            addChild(backdrop);

            var backgroundShape:Shape = new Shape();
            var g1:Graphics = backgroundShape.graphics;
            var c1:Array = [0x336600, 0x80ff00];
            var a1:Array = [255, 255];
            var r1:Array = [100, 255];
            var m1:Matrix = new Matrix();
            m1.createGradientBox(300, 200);
            g1.beginGradientFill(GradientType.LINEAR, c1, a1, r1, m1);
            g1.drawEllipse(0, 0, 300, 200);
```

```
            g1.endFill();
            addChild(backgroundShape);

            var foregroundShape:Shape = new Shape();
            var g2:Graphics = foregroundShape.graphics;
            var c2:Array = [0xff8000, 0x663300];
            var a2:Array = [255, 255];
            var r2:Array = [100, 255];
            var m2:Matrix = new Matrix();
            m2.createGradientBox(300, 200);
            g2.beginGradientFill(GradientType.LINEAR, c2, a2, r2, m2);
            g2.drawEllipse(100, 0, 300, 200);
            g2.endFill();
            addChild(foregroundShape);

            foregroundShape.blendShader = shader;
            foregroundShape.blendMode = BlendMode.SHADER;
        }
    }
}
```

Este é o código-fonte para o kernel de sombreador LumaLighten, usado para criar o arquivo de código de bytes do Pixel Bender "LumaLighten.pbj":

```
<languageVersion : 1.0;>
kernel LumaLighten
<
    namespace : "com.quasimondo.blendModes";
    vendor : "Quasimondo.com";
    version : 1;
    description : "Luminance based lighten blend mode";
>
{
    input image4 background;
    input image4 foreground;

    output pixel4 dst;

    const float3 LUMA = float3(0.212671, 0.715160, 0.072169);

    void evaluatePixel()
    {
        float4 a = sampleNearest(foreground, outCoord());
        float4 b = sampleNearest(background, outCoord());
        float luma_a = a.r * LUMA.r + a.g * LUMA.g + a.b * LUMA.b;
        float luma_b = b.r * LUMA.r + b.g * LUMA.g + b.b * LUMA.b;

        dst = luma_a > luma_b ? a : b;
    }
}
```

Para obter mais informações sobre como usar modos de mesclagem, consulte "Aplicação de modos de mesclagem" na página 305.

Uso de um sombreador como filtro

Usar um sombreador como filtro é parecido com usar qualquer um dos outros filtros do ActionScript. Quando você utiliza um sombreador como filtro, a imagem filtrada (um objeto de exibição ou objeto BitmapData) é passada para ele. O sombreador usa a imagem de entrada para criar a saída do filtro, que geralmente é uma versão modificada da imagem original. Se o objeto filtrado é um objeto de exibição, a saída do sombreador é exibida na tela no lugar do objeto de exibição filtrado. Se o objeto filtrado é um objeto BitmapData, a saída do sombreador torna-se o conteúdo do objeto BitmapData cujo método `applyFilter()` é chamado.

Para usar um sombreador como filtro, primeiro você deve criar o objeto Shader conforme descrito em "Carregamento ou incorporação de um sombreador" na página 388. Em seguida, crie um objeto ShaderFilter vinculado ao objeto Shader. O objeto ShaderFilter é o filtro aplicado ao objeto filtrado. Você deve aplicá-lo a um objeto da mesma maneira que aplica qualquer filtro. Passe-o para a propriedade `filters` de um objeto de exibição ou chame o método `applyFilter()` em um objeto BitmapData. Por exemplo, o código a seguir cria um objeto ShaderFilter e aplica o filtro a um objeto de exibição chamado `homeButton`.

```
var myFilter:ShaderFilter = new ShaderFilter(myShader);
homeButton.filters = [myFilter];
```

Quando você usa um sombreador como filtro, o filtro deve ser definido com pelo menos uma entrada. Como mostra o exemplo, você não define o valor de entrada no código. Em vez disso, o objeto de exibição filtrado ou o objeto BitmapData é definido como a imagem de entrada. Se usar um sombreador que espera mais de uma entrada, especifique um valor para qualquer entrada além da primeira.

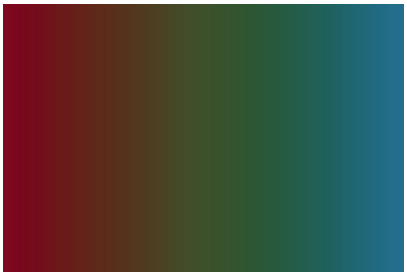
Em alguns casos, um filtro altera as dimensões da imagem original. Por exemplo, um típico efeito de sombra adiciona pixels extra contendo a sombra que é acrescentada à imagem. Quando usar um sombreador que altera as dimensões da imagem, defina as propriedades `leftExtension`, `rightExtension`, `topExtension` e `bottomExtension` para indicar em quanto você deseja que o tamanho da imagem seja alterado.

O exemplo a seguir demonstra o uso de um sombreador como filtro. O filtro deste exemplo inverte os valores de canal de vermelho, verde e azul de uma imagem. O resultado é a versão "negativa" da imagem.

Nota: *O sombreador usado neste exemplo é o kernel do Pixel Bender invertRGB.pbk fornecido com o Pixel Bender Toolkit. É possível carregar o código-fonte do kernel a partir do diretório de instalação do Pixel Bender Toolkit. Compile o código-fonte e salve o arquivo de código de bytes no mesmo diretório do código-fonte.*

O código ActionScript importante está nestes dois métodos:

- `init()`: O método `init()` é chamado quando o aplicativo é carregado. Neste método, o código carrega o arquivo de código de bytes do sombreador.
- `onLoadComplete()`: No método `onLoadComplete()`, o código cria o objeto Shader chamado `shader`. Em seguida, ele cria e desenha o conteúdo de um objeto chamado `target`. O objeto `target` é um retângulo preenchido com uma cor gradiente linear que é vermelho na esquerda, verde e amarelo no meio e azul claro na direita. O objeto não filtrado é semelhante a este:



Com o filtro aplicado, as cores são invertidas, e o retângulo fica parecido com o seguinte:

O sombreador usado neste exemplo é o kernel do Pixel Bender de exemplo "invertRGB.pbk" fornecido com o Pixel Bender Toolkit. O código-fonte está disponível no arquivo "invertRGB.pbk" localizado no diretório de instalação do Pixel Bender Toolkit. Compile o código-fonte e salve o arquivo de código de bytes com o nome "invertRGB.pbj" no mesmo diretório que o seu código-fonte do ActionScript.

Veja abaixo o código ActionScript para este exemplo. Use esta classe com a classe de aplicativo principal em um projeto somente ActionScript no Flex ou como a classe do documento para o arquivo FLA na Ferramenta de autoria do Flash:

```
package
{
    import flash.display.GradientType;
    import flash.display.Graphics;
    import flash.display.Shader;
    import flash.display.Shape;
    import flash.display.Sprite;
    import flash.filters.ShaderFilter;
    import flash.events.Event;
    import flash.geom.Matrix;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class InvertRGB extends Sprite
    {
        private var shader:Shader;
        private var loader:URLLoader;

        public function InvertRGB()
        {
            init();
        }

        private function init():void
        {
            loader = new URLLoader();
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("invertRGB.pbj"));
        }


        private function onLoadComplete(event:Event):void
        {
```

```
            shader = new Shader(loader.data);

            var target:Shape = new Shape();
            addChild(target);

            var g:Graphics = target.graphics;
            var c:Array = [0x990000, 0x445500, 0x007799];
            var a:Array = [255, 255, 255];
            var r:Array = [0, 127, 255];
            var m:Matrix = new Matrix();
            m.createGradientBox(w, h);
            g.beginGradientFill(GradientType.LINEAR, c, a, r, m);
            g.drawRect(10, 10, w, h);
            g.endFill();

            var invertFilter:ShaderFilter = new ShaderFilter(shader);
            target.filters = [invertFilter];
        }
    }
}
```

Para obter mais informações sobre como aplicar filtros, consulte "Criação e aplicação de filtros" na página 355.

Uso de um sombreador no modo autônomo

Quando você usa um sombreador no modo autônomo, o processamento do sombreador é executado independentemente do modo como você pretende usar a saída. Você especifica o sombreador a ser executado, define valores de entrada e de parâmetro e designa um objeto em que são colocados os dados resultantes. Você pode usar um sombreador no modo autônomo por dois motivos:

- Processamento de dados não de imagem: no modo autônomo, você pode optar por passar dados binários ou numéricos arbitrários para o sombreador em vez de dados de imagem de bitmap. Você pode determinar que o resultado do sombreador seja retornado como dados binários ou numéricos além dos dados de imagem de bitmap.
- Processamento em segundo plano: Quando você executa um sombreador no modo autônomo, por padrão ele é executado de maneira assíncrona. Isso significa que o sombreador é executado em segundo plano enquanto o aplicativo continua a executar, e o código recebe uma notificação quando o processamento do sombreador é concluído. Você pode usar um sombreador que demora bastante tempo para ser executado e não congela a interface de usuário do aplicativo ou outro processamento enquanto o sombreador está em execução.

Use um objeto ShaderJob para executar um sombreador no modo autônomo. Primeiro você deve criar o objeto ShaderJob e vinculá-lo ao objeto Shader que representa o sombreador a ser executado:

```
var job:ShaderJob = new ShaderJob(myShader);
```

Em seguida, defina quaisquer valores de entrada ou de parâmetro esperados pelo sombreador. Se estiver executando o sombreador em segundo plano, você também deverá registrar um ouvinte para o evento `complete` do objeto ShaderJob. O ouvinte será chamado quando o sombreador concluir seu trabalho:

```
function completeHandler(event:ShaderEvent):void
{
    // do something with the shader result
}

job.addEventListener(ShaderEvent.COMPLETE, completeHandler);
```

Em seguida, crie um objeto no qual o resultado da operação do sombreador será gravado quando a operação for concluída. Atribua esse objeto à propriedade `target` do objeto ShaderJob:

```
var jobResult:BitmapData = new BitmapData(100, 75);
job.target = jobResult;
```

Atribua uma ocorrência de BitmapData à propriedade `target` caso você esteja usando ShaderJob para executar o processamento da imagem. Se estiver processando dados binários ou numéricos, atribua um objeto ByteArray ou uma ocorrência de Vector.<Number> à propriedade `target`. Nesse caso, você deve definir as propriedades `width` e `height` do objeto ShaderJob para especificar o volume de dados a ser gerado para o objeto `target`.

Nota: *é possível definir as propriedades `shader`, `target`, `width` e `height` do objeto ShaderJob em uma única etapa passando argumentos para o construtor `ShaderJob()`, da seguinte forma:`var job:ShaderJob = new ShaderJob(myShader, myTarget, myWidth, myHeight);`*

Quando estiver pronto para executar o sombreador, chame o método `start()` do objeto ShaderJob:

```
job.start();
```

Por padrão, chamar `start()` faz com que ShaderJob seja executado de maneira assíncrona. Nesse caso, a execução do programa prossegue imediatamente com a próxima linha de código, em vez de esperar a conclusão do sombreador. Quando a operação do sombreador é concluída, o objeto ShaderJob chama seus ouvintes de evento `complete`, notificando-os sobre o término da execução. Nesse momento (isto é, no corpo do ouvinte de evento `complete`), o objeto `target` contém o resultado da operação do sombreador.

Nota: *Em vez de usar o objeto da propriedade `target`, você pode recuperar o resultado do sombreador diretamente do objeto do evento que é passado para o método do ouvinte. O objeto do evento é uma ocorrência de ShaderEvent. O objeto ShaderEvent tem três propriedades que podem ser usadas para acessar o resultado, dependendo do tipo de dados do objeto definido como a propriedade `target`: `ShaderEvent.bitmapData`, `ShaderEvent.byteArray` e `ShaderEvent.vector`.*

Se preferir, você pode passar um argumento `true` para o método `start()`. Nesse caso, a operação do sombreador será executada de maneira síncrona. Todo o código (inclusive a interação com a interface de usuário e quaisquer outros eventos) é pausado enquanto o sombreador é executado. Quando o sombreador é concluído, o objeto `target` contém o resultado do sombreador e o programa prossegue com a próxima linha de código.

```
job.start(true);
```

Capítulo 18: Trabalho com clipes de filme

MovieClip é a principal classe de símbolos de animação e clipe de filme criada no Adobe® Flash® CS4 Professional. Ela possui todos os comportamentos e funcionalidades dos objetos de exibição, mas com propriedades e métodos adicionais para controlar a linha de tempo de um clipe de filme. Este capítulo explica como usar o ActionScript para controlar a reprodução de clipe de filme e criar um clipe de filme dinamicamente.

Noções básicas de clipes de filme

Introdução ao trabalho com clipes de filme

Os clipes de filme são elementos essenciais para quem cria conteúdo animado com a ferramenta de autoria do Flash e deseja controlar esse conteúdo com o ActionScript. Sempre que você cria um símbolo de clipe de filme no Flash, o Flash adiciona o símbolo à biblioteca desse documento Flash. Por padrão, esse símbolo se torna uma ocorrência da classe MovieClip, que contém as propriedades e métodos da classe MovieClip.

Quando uma ocorrência de um símbolo de clipe de filme é colocada no Palco, o clipe de filme avança automaticamente na linha de tempo (se tiver mais de um quadro), a menos que a reprodução seja alterada usando o ActionScript. É essa linha de tempo que distingue a classe MovieClip, permitindo que você crie a animação por meio de interpolações de movimento ou forma com a ferramenta de autoria do Flash. Entretanto, com um objeto de exibição que é uma ocorrência da classe Sprite, você pode criar a animação apenas alterando os valores do objeto de modo programático.

Nas versões anteriores do ActionScript, a classe MovieClip era a base para todas as ocorrências no Palco. No ActionScript 3.0, um clipe de filme é apenas um dos muitos objetos que podem aparecer na tela. Se uma linha de tempo não for necessária para a função de um objeto de exibição, o uso da classe Shape ou Sprite no lugar da classe MovieClip poderá melhorar o desempenho de renderização. Para obter informações sobre a escolha do objeto de exibição apropriado para uma tarefa, consulte “Escolha de uma subclasse de DisplayObject” na página 292.

Tarefas de clipe de filme comuns

As seguintes tarefas de clipes de filme comuns são descritas neste capítulo:

- Fazer com que os clipes de filme sejam reproduzidos e parem
- Reproduzir clipes de filme no sentido inverso
- Mover o indicador de reprodução para pontos específicos da linha de tempo de um clipe de filme
- Trabalhar com rótulos de quadro no ActionScript
- Acessar informações de cena no ActionScript
- Criar ocorrências de símbolos da biblioteca de clipe de filme usando o ActionScript
- Carregar e controlar arquivos SWF externos, incluindo arquivos criados para versões anteriores do Flash Player
- Construir um sistema do ActionScript para criar ativos gráficos a serem carregados e usados em tempo de execução

Conceitos e termos importantes

A lista de referência a seguir contém termos importantes usados neste capítulo:

- SWF AVM1: um arquivo SWF criado usando o ActionScript 1.0 ou o ActionScript 2.0, em geral, para ser executado no Flash Player 8 ou anterior.
- SWF AVM2: um arquivo SWF criado usando o ActionScript 3.0 para Adobe Flash Player 9 ou posterior ou Adobe AIR.
- SWF externo: um arquivo SWF que é criado separadamente do arquivo SWF de projeto e serve para ser carregado no arquivo SWF de projeto e reproduzido nesse arquivo SWF.
- Quadro: a menor divisão de tempo na linha de tempo. Assim como em um fotograma de filme de cinema, cada quadro é como um instantâneo da animação no tempo e, quando os quadros são reproduzidos rapidamente em seqüência, o efeito da animação é criado.
- Linha de tempo: a representação metafórica da série de quadros forma a seqüência de animação de um clipe de filme. A linha de tempo de um objeto MovieClip é equivalente à da ferramenta de autoria do Flash.
- Indicador de reprodução: um marcador identificando o local (quadro) na linha de tempo que está sendo exibido em um determinado momento.

Teste dos exemplos do capítulo

Ao trabalhar em um capítulo, talvez você queira testar algumas listagens de código de exemplo sozinho. Como este capítulo descreve como trabalhar com clipes de filme no ActionScript, quase todas as listagens de código foram escritas com a idéia de manipular um símbolo de clipe de filme que foi criado e colocado no Palco. O teste da amostra envolverá a exibição do resultado no Flash Player ou no AIR para ver os efeitos do código no símbolo. Para testar as listagens de código deste capítulo:

1. Crie um documento Flash vazio.
2. Selecione um quadro-chave na linha de tempo.
3. Abra o painel Ações e copie a listagem de código no painel Script.
4. Crie uma ocorrência de símbolo de clipe de filme no Palco. Por exemplo, desenhe uma forma, selecione-a, escolha Modificar > Converter em símbolo e dê um nome ao símbolo.
5. Com o clipe de filme selecionado, no Inspetor de propriedades, dê um nome de ocorrência a ele. O nome deve corresponder ao nome usado para o clipe de filme na listagem de código de exemplo — por exemplo, se a listagem de código manipular um clipe de filme chamado `myMovieClip`, o nome da ocorrência do clipe de filme também deverá ser `myMovieClip`.
6. Execute o programa usando Controlar > Testar filme.

   Na tela, você verá os resultados do código manipulando o clipe de filme como especificado na listagem de código.

Outras técnicas para testar listagens de código de exemplo são explicadas mais detalhadamente em"Teste de listagens de código de exemplo dos capítulos" na página 36.

Trabalho com objetos MovieClip

Ao publicar um arquivo SWF, o Flash converte todas as ocorrências do símbolo de clipe de filme no Palco para objetos MovieClip. Para disponibilizar um símbolo de clipe de filme para o ActionScript, dê a ele um nome de ocorrência no campo Nome da ocorrência do Inspetor de propriedades. Quando um arquivo SWF é criado, o Flash gera o código que cria a ocorrência de MovieClip no Palco e declara uma variável usando o nome da ocorrência. Se você nomeou clipes de filme que estão aninhados em outros clipes de filme nomeados, esses clipes de filme filho serão tratados como propriedades do clipe de filme pai (você pode acessar o clipe de filme filho usando a sintaxe de pontos). Por exemplo, se um clipe de filme com o nome de ocorrência `childClip` estiver aninhado em outro clipe com o nome de ocorrência `parentClip`, você poderá reproduzir a animação de linha de tempo do clipe filho chamando este código:

```
parentClip.childClip.play();
```

Nota: *: Ocorrências-filho inseridas no Palco, na ferramenta de autoria do Flash, não podem ser acessadas pelo código a partir do construtor de uma ocorrência-pai, já que não foram criadas nesse ponto da execução do código. Antes de acessar o filho, o pai deve, em vez disso, criar a ocorrência-filho por código ou atrasar o acesso a uma função de retorno de chamada que ouve o filho de modo a despachar seu evento* `Event.ADDED_TO_STAGE`*.*

Embora alguns métodos e propriedades herdados da classe MovieClip do ActionScript 2.0 permaneçam os mesmos, outros foram alterados. Todas as propriedades prefixadas com um sublinhado foram renomeadas. Por exemplo, as propriedades `_width` e `_height` agora são acessadas como `width` e `height`, enquanto que `_xscale` e `_yscale` agora são acessadas como `scaleX` e `scaleY`. Para obter uma lista completa das propriedades e dos métodos da classe MovieClip, consulte a Referência dos componentes e da linguagem do ActionScript 3.0.

Controle de reprodução de clipe de filme

O Flash usa a metáfora de uma linha de tempo para transmitir uma animação ou uma alteração de estado. Qualquer elemento visual que empregue uma linha de tempo deve ser um objeto MovieClip ou uma extensão da classe MovieClip. Embora o ActionScript possa instruir qualquer clipe de filme a parar, reproduzir ou ir para outro ponto na linha de tempo, ele não pode ser usado para criar dinamicamente uma linha de tempo ou adicionar conteúdo a quadros específicos; isso só é possível usando a ferramenta de autoria do Flash.

Ao ser reproduzido, o MovieClip avança na linha de tempo em uma velocidade controlada pela velocidade de projeção do arquivo SWF. Se desejar, você pode substituir essa configuração definindo a propriedade `Stage.frameRate` no ActionScript.

Reproduzir clipes de filme e parar a reprodução

Os métodos `play()` e `stop()` permitem o controle básico de um clipe de filme na linha de tempo. Por exemplo, suponha que você tenha um símbolo de clipe de filme no Palco que contém uma animação de uma bicicleta se movendo pela tela, com o nome de ocorrência definido como `bicycle`. Se o seguinte código for anexado a um quadro-chave na linha de tempo principal,

```
bicycle.stop();
```

a bicicleta não se moverá (a animação não será reproduzida). O movimento da bicicleta pode começar por meio de outra interação do usuário. Por exemplo, se você tivesse um botão chamado `startButton`, o seguinte código em um quadro-chave na linha de tempo principal faria a animação ser reproduzida com um clique no botão:

```
// This function will be called when the button is clicked. It causes the
// bicycle animation to play.
function playAnimation(event:MouseEvent):void
{
    bicycle.play();
}
// Register the function as a listener with the button.
startButton.addEventListener(MouseEvent.CLICK, playAnimation);
```

Avançar e retroceder

Os métodos `play()` e `stop()` não são a única forma de controlar a reprodução em um clipe de filme. Você também pode mover o indicador de reprodução para frente ou para trás na linha de tempo manualmente, usando os métodos `nextFrame()` e `prevFrame()`. Quando qualquer um desses métodos é chamado, a reprodução pára e move o indicador de reprodução um quadro para frente ou para trás, respectivamente.

Usar o método `play()` é como chamar `nextFrame()` sempre que o evento `enterFrame` do objeto do clipe de filme é disparado. Com essas linhas, você pode fazer a reprodução do clipe de filme `bicycle` retroceder, criando um ouvinte de eventos para o evento `enterFrame` e instruindo `bicycle` a ir para o quadro anterior na função do ouvinte, desta forma:

```
// This function is called when the enterFrame event is triggered, meaning
// it's called once per frame.
function everyFrame(event:Event):void
{
    if (bicycle.currentFrame == 1)
    {
        bicycle.gotoAndStop(bicycle.totalFrames);
    }
    else
    {
        bicycle.prevFrame();
    }
}
bicycle.addEventListener(Event.ENTER_FRAME, everyFrame);
```

Na reprodução normal, se um clipe de filme tiver mais de um quadro, ele fará loop indefinidamente ao ser reproduzido, ou seja, ele retornará ao Quadro 1 quando ultrapassar o quadro final. Ao usar `prevFrame()` ou `nextFrame()`, esse comportamento não acontece automaticamente (chamar `prevFrame()` quando o indicador de reprodução está no Quadro 1 não o move para o último quadro). A condição `if` no exemplo anterior verifica se o indicador de reprodução retrocedeu para o primeiro quadro e o define à frente de seu quadro final, criando assim um loop contínuo do clipe de filme cuja reprodução retrocede.

Salto para um quadro diferente e uso de rótulos de quadro

O envio de um clipe de filme para um novo quadro é simples. A chamada de `gotoAndPlay()` ou `gotoAndStop()` fará o clipe de filme saltar para o número de quadro especificado como parâmetro. Se preferir, você pode transmitir uma seqüência de caracteres que corresponda ao nome de um rótulo de quadro. É possível atribuir um rótulo a qualquer quadro na linha de tempo. Para fazer isso, selecione um quadro na linha de tempo e digite um nome no campo Rótulo do quadro no Inspetor de propriedades.

As vantagens de usar rótulos de quadro em vez de números ficam mais evidentes ao criar um clipe de filme complexo. Quando o número de quadros, camadas e interpolações em uma animação for grande, considere a rotulagem de quadros importantes com descrições explicativas que representem mudanças de comportamento no clipe de filme (por exemplo, "off", "walking", "running"). Isso melhora a confiabilidade do código e também fornece flexibilidade, já que as chamadas do ActionScript que vão para um quadro rotulado são ponteiros para uma única referência, o rótulo, em vez de um número de quadro específico. Se mais tarde você decidir mover um segmento específico da animação para um quadro diferente, não será necessário alterar o código do ActionScript contanto que mantenha o mesmo rótulo para os quadros no novo local.

Para representar os rótulos de quadro no código, o ActionScript 3.0 inclui a classe FrameLabel. Cada ocorrência dessa classe representa um único rótulo de quadro e tem uma propriedade `name` representando o nome do rótulo de quadro conforme especificado no Inspetor de propriedades e uma propriedade`frame` representando o número do quadro com rótulo colocado na linha de tempo.

Para obter acesso às ocorrências de FrameLabel associadas a uma ocorrência de clipe de filme, a classe MovieClip inclui duas propriedades que retornam objetos FrameLabel diretamente. A propriedade `currentLabels` retorna uma matriz que consiste em todos os objetos FrameLabel na linha de tempo inteira de um clipe de filme. A propriedade `currentLabel` retorna uma seqüência de caracteres contendo o nome do rótulo de quadro encontrado mais recentemente na linha de tempo.

Suponha que você crie um clipe de filme chamado `robot` e rotulou os diversos estados dessa animação. Você pode configurar uma condição que verifica a propriedade `currentLabel` para acessar o estado atual de `robot`, como no seguinte código:

```
if (robot.currentLabel == "walking")
{
    // do something
}
```

Trabalho com cenas

No ambiente de autoria do Flash, você pode usar cenas para demarcar uma série de linhas de tempo pelas quais um arquivo SWF pode avançar. Com o uso do segundo parâmetro dos métodos `gotoAndPlay()` ou `gotoAndStop()`, é possível especificar uma cena à qual enviar o indicador de reprodução. Todos os arquivos FLA começam apenas com a cena inicial, mas você pode criar novas cenas.

O uso de cenas nem sempre é a melhor abordagem porque as cenas apresentam várias desvantagens. Um documento Flash contendo várias cenas pode ser difícil de manter, principalmente em ambientes de vários autores. Várias cenas também podem ser ineficientes em largura de banda, porque o processo de publicação mescla todas as cenas em uma única linha de tempo. Isso provoca um download progressivo de todas as cenas, mesmo que elas nunca sejam reproduzidas. Por esses motivos, o uso de várias cenas, muitas vezes, não é recomendado para organizar várias animações longas baseadas na linha de tempo.

A propriedade `scenes` da classe MovieClip é uma matriz de objetos Scene que representa todas as cenas no arquivo SWF. A propriedade `currentScene` retorna um objeto Scene que representa a cena que está em execução no momento.

A classe Scene possui várias propriedades que fornecem informações sobre uma cena. A propriedade `labels` retorna uma matriz de objetos FrameLabel representando os rótulos de quadro dessa cena. A propriedade `name` retorna o nome da cena como uma seqüência de caracteres. A propriedade `numFrames` retorna um int representando o número total de quadro na cena.

Criação de objetos MovieClip com o ActionScript

Uma maneira de adicionar conteúdo à tela no Flash é arrastando ativos da biblioteca para o Palco, mas esse não é o único fluxo de trabalho. Para projetos complexos, os desenvolvedores experientes em geral preferem criar clipes de filme de modo programático. Essa abordagem apresenta várias vantagens: facilidade de reutilização de código, velocidade de tempo de compilação mais rápida e modificações mais sofisticadas que estão disponíveis apenas para o ActionScript.

A API da lista de exibição do ActionScript 3.0 simplifica o processo de criar dinamicamente os objetos MovieClip. A capacidade de instanciar uma ocorrência de MovieClip diretamente, separada do processo de adicioná-la á lista de exibição, fornece flexibilidade e simplicidade sem sacrificar o controle.

No ActionScript 3.0, quando você cria uma ocorrência de clipe de filme (ou qualquer outro objeto de exibição) de modo programático, ela só aparece na tela quando é adicionada à lista de exibição, chamando `addChild()` ou o método `addChildAt()` em um contêiner de objetos de exibição. Isso permite criar um clipe de filme, definir suas propriedades e até chamar métodos antes que ele seja renderizado na tela. Para obter informações sobre como trabalhar com a lista de exibição, consulte "Trabalho com contêineres de objeto de exibição" na página 282.

Exportação de símbolos da biblioteca para o ActionScript

Por padrão, as ocorrências de símbolos de clipe de filme em uma biblioteca de documentos Flash não podem ser criadas dinamicamente (ou seja, usando apenas o ActionScript). Isso porque cada símbolo que é exportado para ser usado no ActionScript aumenta o tamanho do arquivo SWF, e talvez não haja a intenção de usar alguns símbolos no palco. Por isso, para que um símbolo se torne disponível, é necessário especificar que ele seja exportado para o ActionScript.

Para exportar um símbolo para o ActionScript:

1. Selecione o símbolo no painel Biblioteca e abra a caixa de diálogo Propriedades do símbolo.
2. Se necessário, ative as Configurações avançadas.
3. Na seção de ligação, marque a caixa de seleção Exportar para ActionScript.

   Isso ativará os campos Classe e Classe base.

   Por padrão, o campo Classe é preenchido com o nome do símbolo, com os espaços removidos (por exemplo, um símbolo chamado "Tree House" se tornaria "TreeHouse"). Para especificar que o símbolo use uma classe personalizada para seu comportamento, digite o nome completo da classe incluindo seu pacote neste campo. Se pretende criar ocorrências do símbolo no ActionScript, mas não precisa adicionar nenhum comportamento adicional, você pode deixar o nome da classe como está.

   O valor da Classe base assume `flash.display.MovieClip` como padrão. Se quiser estender a funcionalidade do símbolo para outra classe personalizada, você poderá especificar o nome dessa classe, contanto que ela estenda a classe Sprite (ou MovieClip).
4. Pressione o botão OK para salvar as alterações.

   Neste ponto, se o Flash não puder encontrar um arquivo ActionScript externo com uma definição para a classe especificada (por exemplo, se não for preciso adicionar outro comportamento ao símbolo), será exibido um aviso:

   Não foi possível localizar uma definição para esta classe no caminho de classe. Portanto, uma definição será automaticamente gerada no arquivo SWF após a exportação..

   Desconsidere esse aviso se o símbolo da biblioteca não exigir funcionalidade exclusiva além daquela da classe MovieClip.

Se você não fornecer uma classe para o símbolo, o Flash criará uma para ele equivalente a esta:

```
package
{
    import flash.display.MovieClip;

    public class ExampleMovieClip extends MovieClip
    {
        public function ExampleMovieClip()
        {
        }
    }
}
```

Se quiser adicionar outra funcionalidade ActionScript ao símbolo, adicione as propriedades e métodos adequados à estrutura de código abaixo. Por exemplo, suponha que você tenha um símbolo de clipe de filme contendo um círculo com 50 pixels de largura e 50 de altura, e o símbolo seja especificado para ser exportado para o ActionScript com uma classe chamada Circle. O código a seguir, quando colocado em um arquivo Circle.as, estende a classe MovieClip e fornece ao símbolo os métodos adicionais `getArea()` e `getCircumference()`:

```
package
{
    import flash.display.MovieClip;

    public class Circle extends MovieClip
    {
        public function Circle()
        {
        }

        public function getArea():Number
        {
        // The formula is Pi times the radius squared.
        return Math.PI * Math.pow((width / 2), 2);
        }

        public function getCircumference():Number
        {
        // The formula is Pi times the diameter.
        return Math.PI * width;
        }
    }
}
```

O seguinte código, colocado em um quadro-chave no Quadro 1 do documento Flash, criará uma ocorrência do símbolo e a exibirá na tela:

```
var c:Circle = new Circle();
addChild(c);
trace(c.width);
trace(c.height);
trace(c.getArea());
trace(c.getCircumference());
```

Esse código demonstra a instanciação baseada no ActionScript como uma alternativa para arrastar ativos individuais ao Palco. Isso cria um círculo com todas as propriedades de um clipe de filme e com os métodos personalizados definidos na classe Circle. Este exemplo é bem básico; o símbolo da biblioteca pode especificar um número qualquer de propriedades e métodos em sua classe.

A instanciação baseada no ActionScript é muito eficiente, porque permite criar dinamicamente grandes quantidades de ocorrências, um trabalho que seria cansativo se feito manualmente. Ela também é flexível, porque você pode personalizar as propriedades de cada ocorrência na sua criação. Para ter uma idéia dessas duas vantagens, use um loop para criar dinamicamente várias ocorrências de Circle. Com o símbolo e a classe Circle descritos anteriormente na sua biblioteca de documentos Flash, coloque o seguinte código em um quadro-chave do Quadro 1:

```
import flash.geom.ColorTransform;

var totalCircles:uint = 10;
var i:uint;
for (i = 0; i < totalCircles; i++)
{
    // Create a new Circle instance.
    var c:Circle = new Circle();
    // Place the new Circle at an x coordinate that will space the circles
    // evenly across the Stage.
    c.x = (stage.stageWidth / totalCircles) * i;
    // Place the Circle instance at the vertical center of the Stage.
    c.y = stage.stageHeight / 2;
    // Change the Circle instance to a random color
    c.transform.colorTransform = getRandomColor();
    // Add the Circle instance to the current timeline.
    addChild(c);
}

function getRandomColor():ColorTransform
{
    // Generate random values for the red, green, and blue color channels.
    var red:Number = (Math.random() * 512) - 255;
    var green:Number = (Math.random() * 512) - 255;
    var blue:Number = (Math.random() * 512) - 255;

    // Create and return a ColorTransform object with the random colors.
    return new ColorTransform(1, 1, 1, 1, red, green, blue, 0);
}
```

Isso demonstra como você pode criar e personalizar várias ocorrências de um símbolo com rapidez usando código. Cada ocorrência é posicionada com base na contagem atual dentro do loop e recebe uma cor aleatória definindo sua propriedade `transform` (que Circle herda ao estender a classe MovieClip).

Carregamento de um arquivo SWF externo

No ActionScript 3.0, os arquivos SWF são carregados usando a classe Loader. Para carregar um arquivo SWF externo, o ActionScript precisa fazer quatro coisas:

1. Criar um novo objeto URLRequest com a url do arquivo.
2. Criar um novo objeto Loader.
3. Chamar o método `load()` do objeto Loader, transmitindo a ocorrência de URLRequest como um parâmetro.
4. Chame o método `addChild()` em um contêiner de objetos de exibição (como a linha de tempo principal de um documento Flash) para adicionar a ocorrência de Loader à lista de exibição.

Por fim, o código fica semelhante a este:

```
var request:URLRequest = new URLRequest("http://www.[yourdomain].com/externalSwf.swf");
var loader:Loader = new Loader()
loader.load(request);
addChild(loader);
```

Esse mesmo código pode ser usado para carregar um arquivo de imagem externo, como uma imagem JPEG, GIF ou PNG, especificando a url do arquivo de imagem em vez da url de um arquivo SWF. Um arquivo SWF, diferentemente de um arquivo de imagem, pode conter o ActionScript. Por isso, embora o processo de carregar um arquivo SWF seja idêntico ao de carregar uma imagem, no carregamento de um arquivo SWF externo, o arquivo SWF que faz o carregamento e o arquivo SWF que é carregado residem na mesma caixa de proteção caso o Flash Player ou o AIR executem o SWF e você pretenda usar o ActionScript para se comunicar de alguma forma com arquivo SWF. Além disso, se o arquivo SWF contiver classes que compartilhem o mesmo espaço para nomes das classes no arquivo SWF de carregamento, talvez seja necessário criar um novo domínio de aplicativo para o arquivo SWF carregado a fim de evitar conflitos de espaço para nomes. Para obter mais informações sobre considerações de segurança e domínios de aplicativo, consulte "Uso da classe ApplicationDomain" na página 657 e "Carregamento de arquivos SWF e de imagens" na página 718.

Quando carregado com êxito, o arquivo SWF externo pode ser acessado por meio da propriedade `Loader.content`. Se for publicado no ActionScript 3.0, o arquivo SWF externo será um clipe de filme ou uma entidade gráfica, dependendo da classe que ele estender.

Considerações sobre o carregamento de um arquivo SWF antigo

Se o arquivo SWF externo for publicado com uma versão mais antiga do ActionScript, há importantes limitações a serem consideradas. Diferentemente de um arquivo SWF do ActionScript 3.0 que é executado com AVM2 (ActionScript Virtual Machine 2), um arquivo SWF publicado para o ActionScript 1.0 ou no 2.0 é executado com AVM1 (ActionScript Virtual Machine 1).

Quando um arquivo SWF AVM1 é carregado com êxito, o objeto carregado (a propriedade `Loader.content`) é um objeto AVM1Movie. Uma ocorrência de AVM1Movie não é igual a uma ocorrência de MovieClip. É um objeto de exibição, mas, diferentemente de um clipe de filme, não inclui métodos ou propriedades relacionadas à linha de tempo. O arquivo SWF AVM2 pai não terá acesso a propriedades, métodos ou objetos do objeto AVM1Movie carregado.

Existem várias restrições adicionais em um arquivo SWF AVM1 carregado por um arquivo SWF AVM2. Para obter detalhes, consulte a listagem da classe AVM1Movie na Referência dos componentes e da linguagem do ActionScript 3.0.

Exemplo: RuntimeAssetsExplorer

A funcionalidade Exportar para ActionScript pode ser especialmente vantajosa para bibliotecas que podem ser úteis em mais de um projeto. Se o Flash Player ou o AIR executar um arquivo SWF, os símbolos exportados para o ActionScript ficarão disponíveis para qualquer arquivo SWF dentro da mesma caixa de proteção de segurança que o SWF que o carregar. Dessa forma, um único documento Flash pode gerar um arquivo SWF exclusivamente designado para manter ativos gráficos. Essa técnica é especialmente útil para projetos grandes nos quais os designers que lidam com ativos visuais podem trabalhar em paralelo com os desenvolvedores que criam um arquivo SWF "empacotador" que carrega o arquivo SWF de ativos gráficos em tempo de execução. Você pode usar esse método para manter uma série de arquivos de versão nos quais os ativos gráficos não dependem do progresso do desenvolvimento de programação.

O aplicativo RuntimeAssetsExplorer carrega qualquer arquivo SWF que for uma subclasse de RuntimeAsset e permite navegar pelos ativos disponíveis desse arquivo SWF. O exemplo ilustra o seguinte:

- Carregamento de um arquivo SWF externo usando `Loader.load()`

- Criação dinâmica de um símbolo de biblioteca exportado para o ActionScript
- Controle do ActionScript de reprodução de MovieClip

Antes de começar, observe que cada arquivo SWF a ser executado no Flash Player deve estar localizado na mesma caixa de proteção de segurança. Para obter mais informações, consulte "Caixas de proteção de segurança" na página 706.

Para obter os arquivos de aplicativo para esta amostra, consulte www.adobe.com/go/learn_programmingAS3samples_flash_br. Os arquivos de aplicativo RuntimeAssetsExplorer podem ser encontrados na pasta Samples/RuntimeAssetsExplorer. O aplicativo consiste nos seguintes arquivos:

Arquivo	Descrição
RuntimeAssetsExample.mxml ou RuntimeAssetsExample.fla	A interface do usuário do aplicativo para Flex (MXML) ou Flash (FLA).
GeometricAssets.as	Uma classe de exemplo que implementa a interface RuntimeAsset.
GeometricAssets.fla	Um arquivo FLA vinculado à classe GeometricAssets (a classe de documento do FLA) contendo símbolos que são exportados para o ActionScript.
com/example/programmingas3/runtimeassetsexplorer/RuntimeLibrary.as	Uma interface que define os métodos necessários esperados de todos os arquivos SWF de ativos de tempo de execução que serão carregados no contêiner do explorador.
com/example/programmingas3/runtimeassetsexplorer/AnimatingBox.as	A classe do símbolo de biblioteca na forma de uma caixa de rotação.
com/example/programmingas3/runtimeassetsexplorer/AnimatingStar.as	A classe do símbolo de biblioteca na forma de uma estrela de rotação.

Estabelecimento de uma interface de biblioteca de tempo de execução

Para que o explorador interaja adequadamente com uma biblioteca SWF, a estrutura das bibliotecas de ativos de tempo de execução deve ser formalizada. Faremos isso criando uma interface, que é semelhante a uma classe pois trata-se de um projeto de métodos que demarca uma estrutura esperada, mas, diferentemente de uma classe, não inclui nenhum corpo de método. A interface fornece um meio de comunicação ente a biblioteca de tempo de execução e o explorador. Cada SWF de ativos de tempo de execução que for carregado em nosso navegador implementará essa interface. Para obter informações sobre interfaces e como elas podem ser úteis, consulte "Interfaces" na página 108.

A interface RuntimeLibrary será bem simples: precisamos apenas de uma função que forneça ao explorador uma matriz de caminhos de classe para os símbolos a serem exportados e disponibilizados na biblioteca de tempo de execução. Para esse fim, a interface possui um único método: `getAssets()`.

```
package com.example.programmingas3.runtimeassetexplorer
{
    public interface RuntimeLibrary
    {
        function getAssets():Array;
    }
}
```

Criação do arquivo SWF da biblioteca de ativos

Ao definir a interface RuntimeLibrary, é possível criar vários arquivos SWF de biblioteca de ativos que podem ser carregados em outro arquivo SWF. A criação de uma biblioteca SWF individual de ativos envolve quatro tarefas:

- Criação de uma classe para o arquivo SWF da biblioteca de ativos
- Criação de classes para ativos individuais contidos na biblioteca
- Criação dos ativos gráficos em si
- Associação de elementos gráficos a classes e publicação do SWF da biblioteca

Criação de uma classe para implementar a interface RuntimeLibrary

Em seguida, criaremos a classe GeometricAssets que implementará a interface RuntimeLibrary. Ela será a classe de documento do FLA. O código para essa classe é muito semelhante à interface RuntimeLibrary; a diferença entre eles é que, na definição da classe, o método `getAssets()` possui um corpo de método.

```
package
{
    import flash.display.Sprite;
    import com.example.programmingas3.runtimeassetexplorer.RuntimeLibrary;

    public class GeometricAssets extends Sprite implements RuntimeLibrary
    {
        public function GeometricAssets() {

        }
        public function getAssets():Array {
            return [ "com.example.programmingas3.runtimeassetexplorer.AnimatingBox",
                     "com.example.programmingas3.runtimeassetexplorer.AnimatingStar" ];
        }
    }
}
```

Se fôssemos criar uma segunda biblioteca de tempo de execução, criaríamos outro FLA com base em outra classe (por exemplo, AnimationAssets) que fornece sua própria implementação de `getAssets()`.

Criação de classes para cada ativo MovieClip

Para este exemplo, iremos apenas estender a classe MovieClip sem adicionar nenhuma funcionalidade aos ativos personalizados. O seguinte código para AnimatingStar é análogo ao de AnimatingBox:

```
package com.example.programmingas3.runtimeassetexplorer
{
    import flash.display.MovieClip;

    public class AnimatingStar extends MovieClip
    {
        public function AnimatingStar() {
        }
    }
}
```

Publicação da biblioteca

Agora, iremos conectar os ativos baseados no MovieClip à nova classe, criando um novo FLA e inserindo GeometricAssets no campo Classe do documento do Inspetor de propriedades. Para fins deste exemplo, criaremos duas formas básicas que suam uma interpolação de linha de tempo para fazer uma rotação no sentido horário em 360 quadros. Os símbolos `animatingBox` e `animatingStar` são definidos para Exportar para ActionScript e o campo Classe é definido com os respectivos caminhos de classe especificados na implementação de `getAssets()`. A classe base padrão de `flash.display.MovieClip` permanece, pois desejamos subclassificar os métodos MovieClip padrão.

Depois de definir as configurações de exportação do símbolo, publique o FLA. Agora, sua primeira biblioteca de tempo de execução está pronta. Se esse arquivo SWF fosse carregado em outro arquivo SWF AVM2, os símbolos AnimatingBox e AnimatingStar ficariam disponíveis para o arquivo SWF.

Carregamento da biblioteca em outro arquivo SWF

A última questão funcional a ser resolvida é a interface do usuário para o explorador de ativos. Neste exemplo, o caminho para a biblioteca de tempo de execução é codificado como uma variável chamada `ASSETS_PATH`. Se preferir, você pode usar a classe FileReference, por exemplo, para criar uma interface que navegue para um arquivo SWF específico do seu disco rígido.

Quando a biblioteca de tempo de execução é carregada com êxito, o Flash Player chama o método `runtimeAssetsLoadComplete()`:

```
private function runtimeAssetsLoadComplete(event:Event):void
{
    var rl:* = event.target.content;
    var assetList:Array = rl.getAssets();
    populateDropdown(assetList);
    stage.frameRate = 60;
}
```

Nesse método, a variável rl representa o arquivo SWF carregado. O código chama o método `getAssets()` do arquivo SWF carregado, obtendo uma lista de ativos que estão disponíveis e os usa para preencher um componente ComboBox com uma lista de ativos disponíveis, chamando o método `populateDropDown()`. Esse método por sua vez armazena o caminho de classe completo de cada ativo. Ao ser clicado, o botão Adicionar da interface do usuário dispara o método `addAsset()`:

```
private function addAsset():void
{
    var className:String = assetNameCbo.selectedItem.data;
    var AssetClass:Class = getDefinitionByName(className) as Class;
    var mc:MovieClip = new AssetClass();
    ...
}
```

que obtém o caminho de classe do ativo atualmente selecionado no ComboBox (`assetNameCbo.selectedItem.data`) e usa a função `getDefinitionByName()` (do pacote flash.utils) para obter uma referência real para a classe do ativo a fim de criar uma nova ocorrência desse ativo.

Capítulo 19: Trabalho com interpolações de movimento

"Animação de objetos" na página 311 descreve como implementar animações com script no ActionScript.

Neste capítulo descreveremos uma técnica diferente de criação de animações: a interpolação de movimento. Essa técnica permite a criação de movimento ao configurar o movimento interativamente em um arquivo FLA usando o Adobe® Flash® CS4 Professional. Você poderá usar o movimento recém-criado em sua animação dinâmica com base em ActionScript no tempo de execução.

O Flash CS4 gera automaticamente o código ActionScript que implementa a interpolação de movimento e a disponibiliza para cópia e reutilização.

Para criar interpolações de movimento, é preciso ter a licença do Adobe Flash CS4 Professional.

Noções básicas de interpolações de movimento

Introdução a interpolações de movimento no ActionScript

As interpolações de movimento facilitam a criação de animações.

Uma interpolação de movimento modifica as propriedades do objeto de exibição, tais como posição ou rotação, eu uma base quadro a quadro. Uma interpolação de movimento também pode mudar a aparência de um objeto de exibição durante a movimentação, ao aplicar vários filtros e outras propriedades. A interpolação de movimento pode ser criada interativamente com o Flash, que gera o seu respectivo código ActionScript. No Flash, use Copiar movimento como o comando do ActionScript 3.0 para copiar o código que gerou a interpolação de movimento. Em seguida, será possível reutilizar o código ActionScript para gerar movimento em sua própria animação dinâmica no tempo de execução.

Consulte a seção Interpolações de movimento em *Uso do Flash CS4 Professional* para obter informações sobre a criação de interpolações de movimento.

Tarefas comuns de interpolação de movimento

O código ActionScript gerado automaticamente que implementa a interpolação de movimento executa as seguintes operações:

- Cria uma ocorrência de objeto de movimento para a interpolação de movimento
- Define a duração da interpolação de movimento
- Adiciona as propriedades da interpolação de movimento
- Adiciona filtros à interpolação de movimento
- Associa a interpolação de movimento com seu objeto de exibição ou objetos comuns

Termos e conceitos importantes

O termo a seguir é importante e é usado neste capítulo:

- Interpolação de movimento: é uma construção que gera quadros intermediários de um objeto de exibição em diferentes estados e momentos, criando o efeito de que o primeiro estado evolui para o segundo estado suavemente. Ela é usada para movimentar um objeto de exibição no palco, além de fazê-lo aumentar, diminuir, rotacionar, atenuar ou mudar de cor no decorrer do tempo.

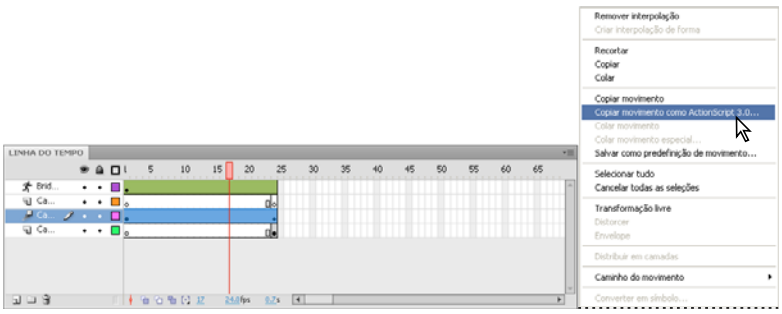
Cópia de scripts de interpolações de movimento

Uma interpolação gera quadros intermediários que mostram um objeto de exibição em diferentes estados em dois quadros diferentes de uma linha do tempo. Ela cria o efeito de que a imagem do primeiro quadro evolui suavemente até se tornar a imagem do segundo. Em uma interpolação de movimento, a mudança de aspecto geralmente envolve a mudança de posição do objeto de exibição, criando assim o movimento. Além do reposicionamento do objeto de exibição, uma interpolação de movimento também pode fazê-lo girar, inclinar, redimensionar ou ainda aplicar filtros a ele.

Crie uma interpolação de movimento no Flash, movendo um objeto de exibição entre os quadros-chave da linha do tempo. O Flash gera automaticamente o código ActionScript que descreve a interpolação, que pode ser copiado e salvo em um arquivo. Consulte a seção Interpolações de movimento em *Uso do Flash CS4 Professional* para obter informações sobre a criação de interpolações de movimento.

Você pode acessar Copiar movimento como um comando do ActionScript 3.0 no Flash de duas maneiras. A primeira é a partir de um menu de contexto de interpolação no palco:

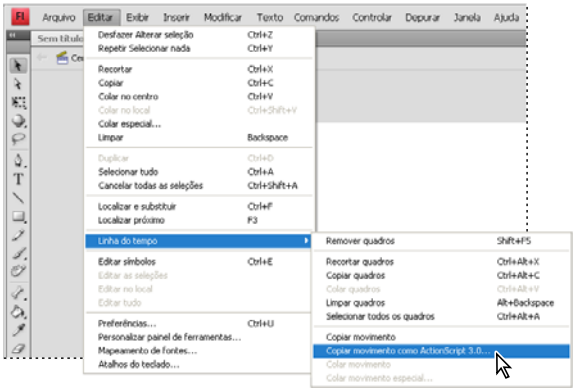
1. Selecione a interpolação de movimento no palco.
2. Clique sobre ela com o botão direito (Windows) ou mantenha a tecla Ctrl pressionada (Macintosh).
3. Selecione Copiar movimento como o comando do ActionScript 3.0. . .



A segunda maneira é selecionar o comando diretamente no menu Editar do Flash:

1. Selecione a interpolação de movimento no palco.

2 Selecione Editar > Linha do tempo > Copiar movimento como ActionScript 3.0.



Após copiar o script, cole-o em um arquivo e salve-o.

Após criar uma interpolação de movimento e copiar e salvar o script, você pode reutilizá-lo ou modificá-lo na animação dinâmica com base em ActionScript que criou.

Incorporação de scripts de interpolações de movimento

O cabeçalho do código ActionScript copiado do Flash lista todos os módulos necessários ao suporte da interpolação de movimento.

Classes de interpolação de movimento

As classes essenciais são: AnimatorFactory, MotionBase e as classes Motion contidas no pacote `fl.motion`. Talvez você precise de classes adicionais, dependendo das propriedades manipuladas pela interpolação de movimento. Por exemplo, se a interpolação de movimento transforma ou rotaciona o objeto de exibição, importe as classes `flash.geom` adequadas. Se ela aplicar filtros, importe as classes `flash.filter`. No ActionScript, uma interpolação de movimento é uma ocorrência da classe Motion. A classe Motion armazena uma seqüência de animação de quadros-chave que pode ser aplicada a um objeto visual. Os dados da animação abrangem posição, escala, rotação, inclinação, cor, filtros e atenuação.

O código ActionScript a seguir foi copiado de uma interpolação de movimento criada para animar um objeto de exibição cujo nome de ocorrência é `Symbol1_2`. Ele declara uma variável para um objeto MotionBase chamado `__motion_Symbol1_2`. A classe MotionBase é pai da classe Motion.

```
var __motion_Symbol1_2:MotionBase;
```

Em seguida, o script cria o objeto Motion:

```
__motion_Symbol1_2 = new Motion();
```

nomes de objetos Motion

No caso anterior, o Flash gerou automaticamente o nome `__motion_Symbol1_2` para o objeto Motion. O prefixo `__motion_` é adicionado ao nome do objeto de exibição. Desse modo, o nome gerado automaticamente é baseado no nome de ocorrência do objeto de destino da interpolação de movimento na ferramenta de autoria do Flash. A propriedade `duration` do objeto Motion indica o número total de quadros na interpolação de movimento:

```
__motion_Symbol1_2.duration = 200;
```

Quando você reutiliza esse código ActionScript na animação que criou, é possível manter o nome que o Flash gera automaticamente para a interpolação ou substituir por um outro de sua preferência. Por padrão, o Flash nomeia automaticamente a ocorrência do objeto de exibição cuja interpolação de movimento está sendo copiada, caso ainda não tenha um nome de ocorrência. Se o nome da interpolação for alterado, certifique-se de alterá-lo em todo o script. Opcionalmente, é possível atribuir um nome de sua preferência ao objeto de destino da interpolação de movimento no Flash. Em seguida, crie a interpolação de movimento e copie o script. Independentemente do nome que utilizar, certifique-se de que cada objeto Motion de seu código ActionScript tenha um nome exclusivo.

Descrição da animação

O método `addPropertyArray()` da classe MotionBase adiciona uma matriz de valores para descrever cada propriedade interpolada.

Potencialmente, a matriz contém um item para cada quadro-chave na interpolação de movimento. Em geral, algumas dessas matrizes contêm uma quantidade de itens menor do que o número total de quadros-chave na interpolação de movimento. Essa situação ocorre quando o último valor da matriz não é alterado nos quadros restantes.

Se o comprimento do argumento da matriz for maior do que a propriedade `duration` do objeto Motion, `addPropertyArray()` fará o ajuste do valor da propriedade `duration` de modo adequado. Ela não adiciona quadros-chave às propriedades que foram adicionadas anteriormente. Os quadros-chave recém-adicionados permanecem nos quadros extras da animação.

As propriedades `x` e `y` do objeto Motion descrevem a mudança de posição do objeto interpolado no decorrer da execução da animação. Essas coordenadas são os valores que apresentam maior possibilidade de mudança em cada quadro-chave, se a posição do objeto de exibição for alterada. Você pode adicionar propriedades de movimento adicionais com o método `addPropertyArray()`. Por exemplo, adicione os valores `scaleX` e `scaleY` se o objeto interpolado estiver redimensionado. Adicione os valores `cewX` e `skewY` se ele estiver inclinado. Adicione a propriedade `rotationConcat` se ele estiver em rotação.

Use o método `addPropertyArray()` para definir as seguintes propriedades de interpolação:

`x`	Posição horizontal do ponto de transformação do objeto no espaço coordenado de seu objeto pai.
`y`	Posição vertical do ponto de transformação do objeto no espaço coordenado de seu objeto pai.
`z`	Posição de profundidade (eixo z) do ponto de transformação do objeto no espaço coordenado de seu objeto pai.
`scaleX`	Escala horizontal como uma porcentagem do objeto, como aplicada a partir do ponto de transformação.
`scaleY`	Escala vertical como porcentagem do objeto, como aplicada a partir do ponto de transformação.
`skewX`	Ângulo de inclinação horizontal do objeto em graus, como aplicado a partir do ponto de transformação.
`skewY`	Ângulo de inclinação vertical do objeto em graus, como aplicado a partir do ponto de transformação.
`rotationX`	Rotação do objeto ao redor do eixo x, a partir de sua orientação original.
`rotationY`	Rotação do objeto ao redor do eixo y, a partir de sua orientação original.
`rotationConcat`	Valores de rotação (eixo z) do objeto no movimento relacionado à orientação anterior, como aplicados a partir do ponto de transformação.
`useRotationConcat`	Se definido, faz com que o objeto de destino gire quando `addPropertyArray()` fornecer dados de movimento.

`blendMode`	Valor da classe BlendMode especificando a mistura de cores de um objeto com gráficos abaixo.
`matrix3D`	Propriedade matrix3D, caso haja alguma no quadro-chave (usada em interpolações em 3D). Se usada, todas as propriedades de transformação anteriores serão ignoradas.
`rotationZ`	Rotação do eixo z do objeto, em graus, a partir da sua orientação original referente ao contêiner 3D pai. Usado em interpolações em 3D no lugar de rotationConcat.

As propriedades adicionadas ao script gerado automaticamente dependem das propriedades que foram atribuídas à interpolação de movimento no Flash. Você pode adicionar, remover ou modificar algumas dessas propriedades ao personalizar a sua versão do script.

O código a seguir atribui valores às propriedades de uma interpolação de movimento chamada `__motion_Wheel`. Nesse caso, o objeto de exibição interpolado não muda de lugar, mas gira na mesma posição nos 29 quadros da interpolação de movimento. Os valores múltiplos atribuídos à matriz `rotationConcat` definem a rotação. Os outros valores de propriedades dessa interpolação de movimento não são alterados.

```
__motion_Wheel = new Motion();
__motion_Wheel.duration = 29;
__motion_Wheel.addPropertyArray("x", [0]);
__motion_Wheel.addPropertyArray("y", [0]);
__motion_Wheel.addPropertyArray("scaleX", [1.00]);
__motion_Wheel.addPropertyArray("scaleY", [1.00]);
__motion_Wheel.addPropertyArray("skewX", [0]);
__motion_Wheel.addPropertyArray("skewY", [0]);
__motion_Wheel.addPropertyArray("rotationConcat",
    [
        0,-13.2143,-26.4285,-39.6428,-52.8571,-66.0714,-79.2857,-92.4999,-105.714,
        -118.929,-132.143,-145.357,-158.571,-171.786,-185,-198.214,-211.429,-224.643,
        -237.857,-251.071,-264.286,-277.5,-290.714,-303.929,-317.143,-330.357,
        -343.571,-356.786,-370
    ]
);
__motion_Wheel.addPropertyArray("blendMode", ["normal"]);
```

No exemplo a seguir, o objeto de exibição chamado `Leaf_1` se move no palco. Suas matrizes de propriedade `x` e `y` contêm valores diferentes para cada um dos 100 quadros da animação. Além disso, o objeto gira em torno de seu eixo `z` ao mesmo tempo em que se move pelo palco. Os itens múltiplos da matriz de propriedade `rotationZ` determinam a rotação.

```
__motion_Leaf_1 = new MotionBase();
__motion_Leaf_1.duration = 100;
__motion_Symbol1_4.addPropertyArray("y",
    [
        0,5.91999,11.84,17.76,23.68,29.6,35.52,41.44,47.36,53.28,59.2,65.12,71.04,
        76.96,82.88,88.8,94.72,100.64,106.56,112.48,118.4,124.32,130.24,136.16,142.08,
        148,150.455,152.909,155.364,157.818,160.273,162.727,165.182,167.636,170.091,
        172.545,175,177.455,179.909,182.364,184.818,187.273,189.727,192.182,194.636,
        197.091,199.545,202,207.433,212.865,218.298,223.73,229.163,234.596,240.028,
        245.461,250.893,256.326,261.759,267.191,272.624,278.057,283.489,
        288.922,294.354,299.787,305.22,310.652,316.085,321.517,326.95,330.475,334,
        337.525,341.05,344.575,348.1,351.625,355.15,358.675,362.2,365.725,369.25,
        372.775,376.3,379.825,383.35,386.875,390.4,393.925,397.45,400.975,404.5,
        407.5,410.5,413.5,416.5,419.5,422.5,425.5
    ]
);
__motion_Symbol1_4.addPropertyArray("scaleX", [1.00]);
__motion_Symbol1_4.addPropertyArray("scaleY", [1.00]);
__motion_Symbol1_4.addPropertyArray("skewX", [0]);
__motion_Symbol1_4.addPropertyArray("skewY", [0]);
__motion_Symbol1_4.addPropertyArray("z",
    [
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ]
);
__motion_Symbol1_4.addPropertyArray("rotationX", [64.0361]);
__motion_Symbol1_4.addPropertyArray("rotationY", [41.9578]);
__motion_Symbol1_4.addPropertyArray("rotationZ",
    [
        -18.0336,-17.5536,-17.0736,-16.5936,-16.1136,-15.6336,-15.1536,-14.6736,
        -14.1936,-13.7136,-13.2336,-12.7536,-12.2736,-11.7936,-11.3136,-10.8336,
        -10.3536,-9.8736,-9.3936,-8.9136,-8.4336,-7.9536,-7.4736,-6.9936,-6.5136,
        -6.0336,-7.21542,-8.39723,-9.57905,-10.7609,-11.9427,-13.1245,-14.3063,
        -15.4881,-16.67,-17.8518,-19.0336,-20.2154,-21.3972,-22.5791,-23.7609,
        -24.9427,-26.1245,-27.3063,-28.4881,-29.67,-30.8518,-32.0336,-31.0771,
        -30.1206,-29.164,-28.2075,-27.251,-26.2945,-25.338,-24.3814,-23.4249,
        -22.4684,-21.5119,-20.5553,-19.5988,-18.6423,-17.6858,-16.7293,-15.7727
        -14.8162,-13.8597,-12.9032,-11.9466,-10.9901,-10.0336,-10.9427,-11.8518,
        -12.7609,-13.67,-14.5791,-15.4881,-16.3972,-17.3063,-18.2154,-19.1245,
        -20.0336,-20.9427,-21.8518,-22.7609,-23.67,-24.5791,-25.4881,-26.3972,
        -27.3063,-28.2154,-29.1245,-30.0336,-28.3193,-26.605,-24.8907,-23.1765,
        -21.4622,-19.7479,-18.0336
    ]
);
__motion_Symbol1_4.addPropertyArray("blendMode", ["normal"]);
```

Adição de filtros

Se o objeto de destino de uma interpolação de movimento contiver filtros, esses filtros serão adicionados usando os métodos `initFilters()` e `addFilterPropertyArray()` da classe Motion.

Inicialização da matriz de filtros

O método `initFilters()` inicializa os filtros. O seu primeiro argumento é uma matriz dos nomes de classe totalmente qualificados de todos os filtros aplicados ao objeto de exibição. Essa matriz de nomes de filtros é gerada a partir da lista de filtros da interpolação de movimento no Flash. Em sua cópia do script, é possível remover ou adicionar quaisquer filtros no pacote `flash.filters` dessa ou a essa matriz. A chamada a seguir inicializa a lista de filtros para o objeto de exibição de destino. Ela aplica os filtros `DropShadowFilter`, `GlowFilter` e `BevelFilter` e copia a lista para cada quadro-chave no objeto Motion.

```
__motion_Box.initFilters(["flash.filters.DropShadowFilter", "flash.filters.GlowFilter",
"flash.filters.BevelFilter"], [0, 0, 0]);
```

Adição de filtros

O método `addFilterPropertyArray()` descreve as propriedades de um filtro inicializado com os seguintes argumentos:

1. O primeiro argumento identifica um filtro por índice. O índice refere-se à posição do nome do filtro na matriz de nomes de classes de filtro transferidas em uma chamada anterior de `initFilters()`.
2. O segundo argumento é a propriedade de filtro para armazenamento desse filtro em cada quadro-chave.
3. O terceiro argumento é o valor da propriedade de filtro especificada.

Dada a chamada anterior de `initFilters()`, as chamadas seguintes de `addFilterPropertyArray()` atribuem um valor 5 às propriedades `blurX` e `blurY` de `DropShadowFilter`. `DropShadowFilter` é o primeiro (índice 0) item da matriz dos filtros inicializados:

```
__motion_Box.addFilterPropertyArray(0, "blurX", [5]);
__motion_Box.addFilterPropertyArray(0, "blurY", [5]);
```

As três chamadas a seguir atribuem valores às propriedades quality, alpha e color de `GlowFilter`, o segundo item (índice 1) da matriz do filtro inicializado:

```
__motion_Box.addFilterPropertyArray(1, "quality", [BitmapFilterQuality.LOW]);
__motion_Box.addFilterPropertyArray(1, "alpha", [1.00]);
__motion_Box.addFilterPropertyArray(1, "color", [0xff0000]);
```

As quatro chamadas a seguir atribuem valores a `shadowAlpha`, `shadowColor`, `highlightAlpha` e `highlightColor` de `BevelFilter`, o terceiro (índice 2) item da matriz de filtros inicializados:

```
__motion_Box.addFilterPropertyArray(2, "shadowAlpha", [1.00]);
__motion_Box.addFilterPropertyArray(2, "shadowColor", [0x000000]);
__motion_Box.addFilterPropertyArray(2, "highlightAlpha", [1.00]);
__motion_Box.addFilterPropertyArray(2, "highlightColor", [0xffffff]);
```

Ajuste da cor com ColorMatrixFilter

Após a inicialização de `ColorMatrixFilter`, você poderá atribuir as propriedades `AdjustColor` adequadas para ajustar o brilho, o contraste, a saturação e a matiz do objeto de exibição interpolado. Geralmente aplica-se o filtro `AdjustColor` no Flash para possibilitar o ajuste de sua cópia do ActionScript. O exemplo a seguir transforma a matiz e a saturação do objeto de exibição à medida que ele se movimenta.

```
__motion_Leaf_1.initFilters(["flash.filters.ColorMatrix"], [0], -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorBrightness", [0], -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorContrast", [0], -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorSaturation",
    [
        0,-0.589039,1.17808,-1.76712,-2.35616,-2.9452,-3.53424,-4.12328,
        -4.71232,-5.30136,-5.89041, 6.47945,-7.06849,-7.65753,-8.24657,
        -8.83561,-9.42465,-10.0137,-10.6027,-11.1918,11.7808,-12.3699,
        -12.9589,-13.5479,-14.137,-14.726,-15.3151,-15.9041,-16.4931,
        17.0822,-17.6712,-18.2603,-18.8493,-19.4383,-20.0274,-20.6164,
        -21.2055,-21.7945,22.3836,-22.9726,-23.5616,-24.1507,-24.7397,
        -25.3288,-25.9178,-26.5068,-27.0959,27.6849,-28.274,-28.863,-29.452,
        -30.0411,-30.6301,-31.2192,-31.8082,-32.3973,32.9863,-33.5753,
        -34.1644,-34.7534,-35.3425,-35.9315,-36.5205,-37.1096,-37.6986,
        38.2877,-38.8767,-39.4657,-40.0548,-40.6438,-41.2329,-41.8219,
        -42.411,-43
    ],
    -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorHue",
    [
        0,0.677418,1.35484,2.03226,2.70967,3.38709,4.06451,4.74193,5.41935,
        6.09677,6.77419,7.45161,8.12903,8.80645,9.48387,10.1613,10.8387,11.5161,
        12.1935,12.871,13.5484,14.2258,14.9032,15.5806,16.2581,16.9355,17.6129,
        18.2903,18.9677,19.6452,20.3226,21,22.4286,23.8571,25.2857,26.7143,28.1429,
        29.5714,31,32.4286,33.8571,35.2857,36.7143,38.1429,39.5714,41,42.4286,43.8571,
        45.2857,46.7143,48.1429,49.5714,51,54,57,60,63,66,69,72,75,78,81,84,87,
        90,93,96,99,102,105,108,111,114
    ],
    -1, -1);
```

Associação de uma interpolação de movimento com seus objetos de exibição

A última tarefa é associar a interpolação de movimento ao objeto de exibição ou aos objetos que ele manipula.

A classe AnimatorFactory gerencia a associação entre uma interpolação de movimento e seus objetos de exibição de destino. O argumento para o construtor AnimatorFactory é o objeto Motion:

```
var __animFactory_Wheel:AnimatorFactory = new AnimatorFactory(__motion_Wheel);
```

Use o método `addTarget()` da classe AnimatorFactory para associar o objeto de exibição de destino à sua interpolação de movimento. O ActionScript copiado do Flash gera um comentário para a linha `addTarget()` e não especifica um nome de ocorrência:

```
// __animFactory_Wheel.addTarget(<instance name goes here>, 0);
```

Em sua cópia, especifique o objeto de exibição para associá-lo à interpolação de movimento. No exemplo a seguir, os destinos são especificados como `greenWheel` e `redWheel`:

```
__animFactory_Wheel.AnimatorFactory.addTarget(greenWheel, 0);
__animFactory_Wheel.AnimationFactory.addTarget(redWheel, 0);
```

Você pode associar vários objetos de exibição com a mesma interpolação de movimento usando várias chamadas de `addTarget()`.

Capítulo 20: Trabalho com cinemática inversa

A IK (Cinemática inversa) é uma ótima técnica de criação de movimentos realistas.

A IK permite a criação de movimentos coordenados dentro de uma cadeia de trechos conectados conhecida como armadura IK, para que as partes apresentem movimentos sincronizados realistas. As partes da armadura são os bones e as junções. Dado o ponto final da armadura, a IK calcula os ângulos das junções que devem alcançar esse ponto.

Fazer os cálculos de tais ângulos manualmente seria desafiador. O mérito desse recurso é que você pode criar armaduras interativamente usando o Adobe® Flash® CS4 Professional. Em seguida, é só animá-las usando o ActionScript. O mecanismo IK, incluído na ferramenta de autoria do Flash, realiza os cálculos que descrevem o movimento da armadura. É possível limitar o movimento a determinados parâmetros contidos em seu código ActionScript.

Para criar armaduras de cinemática inversa, é preciso ter a licença do Adobe Flash CS4 Professional.

Noções básicas de cinemática inversa

Introdução à IK

A IK permite a criação de animações realistas por meio da junção de partes que se movimentam entre si.

Por exemplo, com o uso da IK você pode mover uma perna para uma determinada posição ao articular o movimento de suas respectivas junções para alcançar a posição desejada. A IK usa uma estrutura de bones encadeados em uma estrutura chamada armadura IK. O pacote `fl.ik` ajuda a criar animações que remetem a movimentos naturais. Ele permite que você anime várias armaduras IK diretamente sem precisar saber muito sobre o funcionamento dos algoritmos IK.

Crie a armadura IK com seus respectivos bones e junções auxiliares no Flash. Em seguida, acesse as classes IK para animá-las no tempo de execução.

Consulte a seção Uso de cinemática inversa em *Uso do Flash CS4 Professional* para obter instruções detalhadas sobre como criar uma armadura IK.

Tarefas comuns de IK

Geralmente, o código ActionScript realiza as operações a seguir para iniciar e controlar o movimento de uma armadura IK no tempo de execução:

- Declara variáveis para as armaduras, bones e junções envolvidas no movimento
- Recupera as ocorrências de armaduras, bones e junções
- Cria ocorrência do objeto IKMover
- Define limites para o movimento
- Move a armadura para um ponto de destino

Termos e conceitos importantes

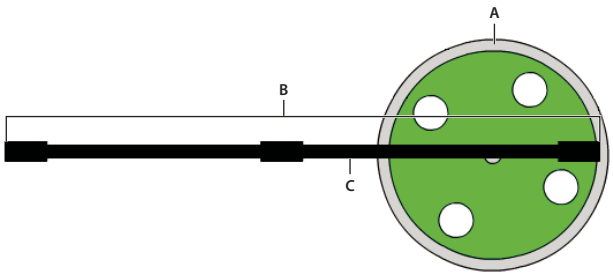
A lista de referência a seguir contém termos importantes usados neste capítulo:

- Armadura: é uma cadeia cinemática, formada por bones e junções, utilizada em animações computadorizadas para simular movimentos reais
- Bone: é um segmento rígido de uma armadura, que corresponde a um dos ossos de um esqueleto
- IK (Cinemática inversa): é um processo de determinação dos parâmetros de um objeto com junções flexíveis, que pode ser uma cadeia cinemática ou uma armadura
- Junção: é o local em que ocorre a união de dois bones, construída para permitir o movimento dos bones, que corresponde a uma articulação de um ser vivo

Visão geral da animação de armaduras IK

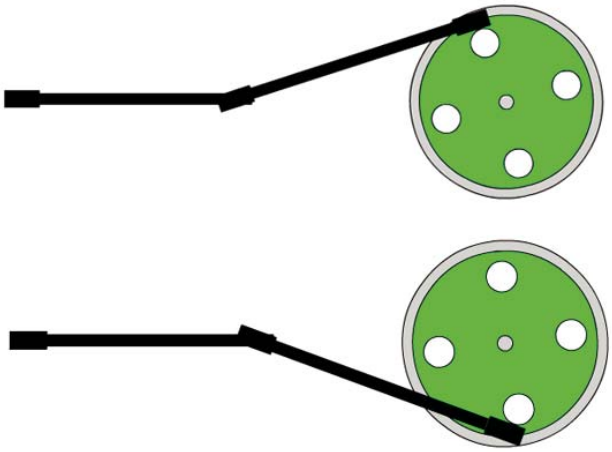
Após criar uma armadura IK, use as classes `fl.ik` para restringir seus movimentos, rastrear seus eventos e animá-la no tempo de execução.

A figura a seguir apresenta um clipe de filme chamado `Roda`. O eixo é uma ocorrência de uma IKArmature chamada `Eixo`. A classe IKMover move a armadura de modo sincronizado com a rotação da roda. O IKBone, `ikBone2`, da armadura é anexado à roda, como junção de sua parte inferior.



***A.** Roda **B.** Eixo **C.** ikBone2*

No tempo de execução, a roda gira em associação com a interpolação de movimento `__motion_Wheel` descrita em "Descrição da animação" na página 423, no capítulo Trabalho com interpolações de movimento. Um objeto IKMover inicia e controla o movimento do eixo. A figura a seguir exibe dois instantâneos da armadura do eixo fixos à roda em diferentes quadros durante o movimento de rotação.



No tempo de execução, o ActionScript a seguir:

- Obtém informações sobre a armadura e seus componentes
- Cria ocorrência de um objeto IKMover
- Move o eixo de acordo com a rotação da roda

```
import fl.ik.*

var tree:IKArmature = IKManager.getArmatureByName("Axle");
var bone:IKBone = tree.getBoneByName("ikBone2");
var endEffector:IKJoint = bone.tailJoint;
var pos:Point = endEffector.position;

var ik:IKMover = new IKMover(endEffector, pos);
ik.limitByDistance = true;
ik.distanceLimit = 0.1;
ik.limitByIteration = true;
ik.iterationLimit = 10;

Wheel.addEventListener(Event.ENTER_FRAME, frameFunc);

function frameFunc(event:Event)
{
    if (Wheel != null)
    {
        var mat:Matrix = Wheel.transform.matrix;
        var pt = new Point(90, 0);
        pt = mat.transformPoint(pt);

        ik.moveTo(pt);
    }
}
```

As classes IK usadas para mover o eixo são:

- IKArmature: descreve a armadura, uma estrutura em árvore formada por bones e junções, que deve ser criada no Flash
- IKManager: classe de contêiner para todas as armaduras IK do documento, que deve ser criada no Flash
- IKBone: um segmento de uma armadura IK
- IKJoint: uma conexão entre dois bones IK
- IKMover: inicia e controla o movimento IK das armaduras

Para obter descrições completas e detalhadas dessas classes, consulte o pacote IK.

Obtenção de informações sobre uma armadura IK

Primeiro, declare variáveis para a armadura, para o bone e para a junção que une as partes que deseja movimentar.

O código a seguir usa o método `getArmatureByName()` da classe IKManager para atribuir o valor da armadura Eixo à variável `tree` da IKArmature. A armadura Eixo foi criada anteriormente no Flash.

```
var tree:IKArmature = IKManager.getArmatureByName("Axle");
```

De modo semelhante, o código a seguir usa o método `getBoneByName()` da classe IKArmature para atribuir a variável IKBone ao valor do bone `ikBone2`.

```
var bone:IKBone = tree.getBoneByName("ikBone2");
```

A junção da parte inferior do bone `ikBone2` é a parte da armadura que a conecta à roda.

A linha a seguir declara a variável `endEffector` e atribui a ela a propriedade `tailjoint` do bone `ikBone2`:

```
var endEffector:IKJoint = home.tailjoint;
```

A variável `pos` é um ponto que armazena a posição correta da junção `endEffector`.

```
var pos:Point = endEffector.position;
```

Nesse exemplo, `pos` é a posição da junção na extremidade do eixo conectada à roda. O valor original dessa variável é obtida a partir da propriedade `position` do IKJoint.

Ocorrência de IKMover e limitação de seu movimento

Uma ocorrência da classe IKMover movimenta o eixo.

A linha a seguir cria ocorrência do objeto `ik` do IKMover, transmitindo ao seu construtor o elemento a ser movimentado e o ponto de início do movimento:

```
var ik:IKMover = new IKMover(endEffector, pos);
```

As propriedades da classe IKMover permitem a limitação do movimento de uma armadura. É possível limitar o movimento, tendo como base a distância, as iterações e o tempo de duração do movimento.

Os pares de propriedades a seguir reforçam esses limites. Os pares consistem em um `Boolean` que indica se o movimento é limitado, além dos números que especificam o limite:

`limitByDistance:Boolean`	`distanceLimit:int`	Define a distância máxima em pixels que o mecanismo IK se desloca para cada iteração.
`limitByIteration:Boolean`	`iterationLimit:int`	Define o número máximo de iterações que o mecanismo IK executa para cada movimento.
`limitByTime:Boolean`	`timeLimit:int`	Define o tempo máximo em milissegundos designado para que o mecanismo IK execute o movimento.

Define a propriedade `Boolean` apropriada como `true` para aplicar o limite. Por padrão, todas as propriedades `Booleans` são definidas como `false`, para que o movimento não seja limitado, a menos que você as tenha definido de modo explícito. Se você definir o limite para um valor sem configurar seu `Boolean` correspondente, o limite será ignorado. Nesse caso, o mecanismo IK continua a movimentação do objeto até outro limite ou até que a posição de destino do IKMover seja alcançada.

No exemplo a seguir, a distância máxima do movimento da armadura está definido como 0,1 pixels por iteração. O número máximo de iterações para cada movimento está definido como 10.

```
ik.limitByDistance = true;
ik.distanceLimit = 0.1;
ik.limitByIteration = true;
ik.iterationLimit = 10;
```

Movimentação de uma armadura IK

O IKMover movimenta o eixo interno do ouvinte de eventos para a roda. Em cada evento enterFrame da roda, uma nova posição de destino para a armadura é calculada. Usando seu método `moveTo()` , o IKMover movimenta a junção da parte inferior para sua respectiva posição de destino ou o mais próximo possível, dentro dos limites definidos pelas suas propriedades `limitByDistance`, `limitByIteration` e `limitByTime`.

```
Wheel.addEventListener(Event.ENTER_FRAME, frameFunc);

function frameFunc(event:Event)
{
    if (Wheel != null)
    {
        var mat:Matrix = Wheel.transform.matrix;
        var pt = new Point(90,0);
        pt = mat.transformPoint(pt);

        ik.moveTo(pt);
    }
}
```

Uso de eventos IK

A classe IKEvent permite a criação de um objeto de evento que contém informações sobre eventos IK. As informações IKEvent descrevem o movimento que foi interrompido porque o tempo especificado, a distância ou o limite de iteração foi excedido.

O código a seguir mostra um ouvinte de eventos e um manipulador para rastreamento de eventos de limite de tempo. Esse manipulador de eventos indica o tempo, a distância, a contagem de iterações e as propriedades de junções de um evento que dispara quando o limite de tempo do IKMover é excedido.

```
var ikmover:IKMover = new IKMover(endjoint, pos);
ikMover.limitByTime = true;
ikMover.timeLimit = 1000;

ikmover.addEventListener(IKEvent.TIME_LIMIT, timeLimitFunction);

function timeLimitFunction(evt:IKEvent):void
{
    trace("timeLimit hit");
    trace("time is " + evt.time);
    trace("distance is " + evt.distance);
    trace("iterationCount is " + evt.iterationCount);
    trace("IKJoint is " + evt.joint.name);
}
```

Capítulo 21: Trabalho com texto

Para exibir texto na tela no Adobe® Flash® Player ou Adobe® AIR™, use uma ocorrência da classe TextField ou use as classes do Mecanismo de texto do Flash. Essas classes permitem criar, exibir e formatar texto.

É possível estabelecer conteúdo específico para campos de texto ou designar a origem do texto e, em seguida, definir a aparência desse texto. Também é possível responder a eventos do usuário conforme o usuário insere texto ou clica em um link de hipertexto.

Noções básicas do trabalho com texto

Introdução ao trabalho com texto

Tanto a classe TextField quanto as classes do Mecanismo de texto do Flash permitem exibir e gerenciar texto no Flash Player e no AIR.

Você pode usar a classe TextField para criar objetos de texto para fins de exibição e entrada. A classe TextField funciona como a base para outros componentes baseados em texto, como TextArea e TextInput, disponíveis no Flash e no Adobe Flex. Você pode usar a classe TextFormat para definir a formatação de caracteres e parágrafos para objetos TextField e pode aplicar CSS (Folhas de estilo em cascata) usando a propriedade Textfield.styleSheet e a classe StyleSheet. É possível atribuir texto com formatação HTML, que pode conter mídia incorporada (clipes de filme e arquivos SWF, GIF, PNG e JPEG), diretamente a um campo de texto.

O Mecanismo de texto do Flash, disponível no Flash Player 10, oferece suporte de baixo nível para controle sofisticado de métricas de texto, formatação e texto bidirecional. Ele também oferece fluxo de texto aprimorado e suporte a idioma avançado. Embora você possa usar o Mecanismo de texto do Flash para criar e gerenciar elementos de texto, a finalidade principal dele é servir de base para a criação de componentes de manipulação de texto e requer maior conhecimento em programação.

Tarefas comuns do trabalho com texto

São abordadas as seguintes tarefas comuns relacionadas à classe TextField:

- Modificação de conteúdo de campo de texto
- Uso de HTML em campos de texto
- Uso de imagens em campos de texto
- Seleção de texto e trabalho com texto selecionado pelo usuário
- Captura da entrada de texto
- Restrição de entrada de texto
- Aplicação de formatação e de estilos CSS ao texto
- Ajuste fino da exibição do texto com nitidez, espessura e suavização de borda
- Acesso e trabalho com campos de texto estáticos no ActionScript

São abordadas as seguintes tarefas comuns relacionadas às classes do Mecanismo de texto do Flash:

- Criação e exibição de texto

- Manipulação de eventos
- Formatação de texto
- Trabalho com fontes
- Controle de texto

Conceitos e termos importantes

A lista de referências a seguir contém termos importantes usados neste capítulo:

- Folhas de estilo em cascata: Uma sintaxe padrão para especificar estilos e formatação para conteúdo estruturado em formato XML (ou HTML).
- Fonte do dispositivo: Uma fonte instalada na máquina do usuário.
- Campo de texto dinâmico: Um campo de texto cujo conteúdo pode ser alterado pelo ActionScript, mas não por entrada do usuário.
- Fonte incorporada: Uma fonte que tem seus dados de contorno de caractere armazenados no arquivo SWF do aplicativo.
- Texto HTML: Conteúdo de texto inserido em um campo de texto usando o ActionScript, que inclui tags de formatação HTML juntamente com conteúdo de texto real.
- Campo de texto de entrada: Um campo de texto cujo conteúdo pode ser alterado pela entrada do usuário ou pelo ActionScript.
- Kerning: um ajuste do espaçamento entre pares de caracteres para tornar o espaçamento das palavras mais proporcional e facilitar a leitura do texto.
- Campo de texto estático: um campo de texto criado na Ferramenta de autoria cujo conteúdo não pode ser alterado quando o arquivo SWF está em execução.
- Métricas de linha de texto: Medidas do tamanho de várias partes do conteúdo de texto em um campo de texto, como a linha de base do texto, a altura da parte superior dos caracteres, o tamanho dos descendentes (a parte de algumas letras minúsculas que se estende abaixo da linha de base) e assim por diante.
- Rastreamento: um ajuste do espaçamento entre grupos de letras ou blocos de texto para aumentar ou diminuir a densidade e tornar o texto mais legível.

Teste dos exemplos do capítulo

Ao trabalhar em um capítulo, talvez você queira testar algumas listagens de código de exemplo sozinho. Basicamente todas as listagens de código envolvem a manipulação de um objeto de texto, seja um objeto criado e colocado no Palco da Ferramenta auditoria do Flash ou um objeto que tenha sido criado no ActionScript. O teste de um exemplo envolve exibir o resultado no Flash Player ou no AIR para verificar os efeitos do código no texto ou no objeto TextField.

Os exemplos deste tópico podem ser divididos em dois grupos. Um tipo de exemplo manipula um objeto TextField sem a criação explícita do objeto. Para testar essas listagens de código, siga estas etapas:

1. Crie um documento Flash vazio.
2. Selecione um quadro-chave na linha de tempo.
3. Abra o painel Ações e copie a listagem de código no painel Script.
4. Usando a ferramenta Texto, crie um campo de texto dinâmico no Palco.

5 Com o campo de texto selecionado, no Inspetor de propriedades, dê a ele um nome de ocorrência. O nome deve corresponder ao nome usado para o campo de texto na listagem de código de exemplo. Por exemplo, se a listagem de código manipular um campo de texto denominado`myTextField`, você deverá nomear o campo de texto também como `myTextField`.

6 Execute o programa usando Controlar > Testar filme.

Na tela, você vê os resultados do código manipulando o campo de texto conforme especificado na listagem de código.

O outro tipo de listagem de código de exemplo consiste em uma definição de classe que deve ser usada como a classe do documento para o arquivo SWF. Nessas listagens, os objetos de texto são criados pelo código de exemplo, portanto não é necessário criá-los separadamente. Para testar esse tipo de listagem de código:

1 Crie um documento Flash vazio e salve-o no seu computador.

2 Crie e salve um novo arquivo do ActionScript no mesmo diretório do documento Flash. O nome do arquivo deve corresponder ao nome da classe na listagem de código. Por exemplo, se a listagem de código definir uma classe denominada TextFieldTest, use o nome TextFieldTest.as para salvar o arquivo ActionScript.

3 Copie a listagem de código no arquivo do ActionScript e salve o arquivo.

4 No documento do Flash, selecione Janela -> Propriedades para ativar o Inspetor de propriedades do documento.

5 No Inspetor de propriedades, no campo Classe, digite o nome da classe ActionScript que você copiou do texto.

6 Execute o programa usando Controlar > Testar filme.

Veja os resultados do exemplo exibidos na tela.

Outras técnicas para testar listagens de código de exemplo são explicadas mais detalhadamente em "Teste de listagens de código de exemplo dos capítulos" na página 36.

Uso da classe TextField

A classe TextField é a base de outros componentes baseados em texto, como os componentes TextArea ou TextInput, que são fornecidos na estrutura do Adobe Flex e no ambiente de autoria do Flash. Para obter mais informações sobre como usar componentes de texto no ambiente de autoria do Flash, consulte "Sobre controles de texto" em *Uso do Flash*.

O conteúdo do campo de texto pode ser pré-especificado no arquivo SWF, carregado de um arquivo de texto ou de um banco de dados ou inserido por um usuário em interação com o aplicativo. Dentro de um campo de texto, o texto pode aparecer como conteúdo HTML renderizado, com imagens incorporadas no HTML renderizado. Depois de criar uma ocorrência de um campo de texto, é possível usar as classes flash.text, como TextFormat e StyleSheet, para controlar a aparência do texto. O pacote flash.text contém praticamente todas as classes relacionadas à criação, ao gerenciamento e à formatação de texto no ActionScript.

É possível formatar texto definindo a formatação com um objeto TextFormat e atribuindo esse objeto ao campo de texto. Se o campo de texto contiver texto HTML, é possível aplicar um objeto StyleSheet ao campo de texto para atribuir estilos a partes específicas do conteúdo do campo de texto. O objeto TextFormat ou StyleSheet contém propriedades que definem a aparência do texto, como cor, tamanho e peso. O objeto TextFormat atribui as propriedades a todo o conteúdo dentro de um campo de texto ou a uma faixa de texto. Por exemplo, dentro do mesmo campo de texto, uma sentença pode estar em negrito vermelho e a próxima em itálico azul.

Para obter mais informações sobre formatos de texto, consulte "Atribuição de formatos de texto" na página 443.

Para obter mais informações sobre texto HTML em campos de texto, consulte “Exibição de texto HTML” na página 438.

Para obter mais informações sobre folhas de estilo, consulte“Aplicação de folhas de estilo em cascata” na página 444.

Além das classes do pacote flash.text, é possível usar a classe flash.events.TextEvent para responder a ações do usuário relacionadas a texto.

Exibição de texto

Apesar das ferramentas de autoria, como o Adobe Flex Builder e a ferramenta de autoria do Flash, fornecerem várias opções para exibição de texto, incluindo componentes relacionados a texto ou ferramentas de texto, a principal maneira de exibir texto programaticamente é por meio de um campo de texto.

Tipos de texto

O tipo de texto dentro de um campo de texto é caracterizado por sua origem:

- Texto dinâmico

  O texto dinâmico inclui conteúdo carregado de uma origem externa, como um arquivo de texto, um arquivo XML ou mesmo um serviço remoto da Web.

- Texto de entrada

  Texto de entrada é qualquer texto inserido por um usuário ou texto dinâmico que pode ser editado pelo usuário. É possível configurar uma folha de estilos para formatar texto de entrada ou usar a classe flash.text.TextFormat para atribuir propriedades ao campo de texto para o conteúdo de entrada. Para obter mais informações, consulte “Captura da entrada de texto” na página 441.

- Texto estático

  O texto estático é criado somente através da Ferramenta de autoria. Não é possível criar uma ocorrência de texto estático usando o ActionScript 3.0. No entanto, é possível usar as classes ActionScript, como StaticText e TextSnapshot, para manipular uma ocorrência de texto estático existente. Para obter mais informações, consulte “Trabalho com texto estático” na página 448.

Modificação de conteúdo de campo de texto

É possível definir texto dinâmico atribuindo uma string à propriedade `flash.text.TextField.text`. Você atribui uma string diretamente à propriedade, da seguinte maneira:

```
myTextField.text = "Hello World";
```

Também é possível atribuir à propriedade `text` um valor de uma variável definida no script, como no exemplo a seguir:

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class TextWithImage extends Sprite
    {
        private var myTextBox:TextField = new TextField();
        private var myText:String = "Hello World";

        public function TextWithImage()
        {
            addChild(myTextBox);
            myTextBox.text = myText;
        }
    }
}
```

Como alternativa, é possível atribuir à propriedade `text` um valor de uma variável remota. Você tem três opções para carregar valores de texto de origens remotas:

- As classes flash.net.URLLoader e flash.net.URLRequest carregam variáveis para o texto de uma localização local ou remota.
- O atributo `FlashVars` é incorporado na página HTML que hospeda o arquivo SWF e pode conter valores para variáveis de texto.
- A classe flash.net.SharedObject gerencia armazenamento persistente de valores. Para obter mais informações, consulte"Armazenamento de dados locais" na página 628.

Exibição de texto HTML

A classe flash.text.TextField tem uma propriedade `htmlText` que pode ser usada para identificar a string de texto como aquela que contém tags HTML para formatar o conteúdo. Como no exemplo a seguir, você deve atribuir o valor da string à propriedade `htmlText` (não à propriedade `text`) para que o Flash Player ou o AIR renderizem o texto como HTML:

```
var myText:String = "<p>This is <b>some</b> content to <i>render</i> as <u>HTML</u> text.</p>";
myTextBox.htmlText = myText;
```

O Flash Player e o AIR oferecem suporte a um subconjunto de tags e entidades HTML para a propriedade `htmlText`. A descrição da propriedade `flash.text.TextField.htmlText` na Referência dos componentes e da linguagem do ActionScript 3.0 fornece informações detalhadas sobre as tags e entidades HTML suportadas.

Depois de designar seu conteúdo usando a propriedade `htmlText`, é possível usar folhas de estilo ou a tag `textformat` para gerenciar a formatação do conteúdo. Para obter mais informações, consulte "Formatação de texto" na página 443.

Uso de imagens em campos de texto

Outra vantagem da exibição do conteúdo como texto HTML é que você pode incluir imagens no campo de texto. É possível fazer referência a uma imagem, local ou remota, usando a tag`img` e fazer com que ela apareça dentro do campo de texto associado.

O exemplo a seguir cria um campo de texto denominado `myTextBox` e inclui uma imagem JPG de um olho, armazenada no mesmo diretório que o arquivo SWF, dentro do texto exibido:

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class TextWithImage extends Sprite
    {
        private var myTextBox:TextField;
        private var myText:String = "<p>This is <b>some</b> content to <i>test</i> and
<i>see</i></p><p><img src='eye.jpg' width='20' height='20'></p><p>what can be
rendered.</p><p>You should see an eye image and some <u>HTML</u> text.</p>";

        public function TextWithImage()
        {
            myTextBox.width = 200;
            myTextBox.height = 200;
            myTextBox.multiline = true;
            myTextBox.wordWrap = true;
            myTextBox.border = true;

            addChild(myTextBox);
            myTextBox.htmlText = myText;
        }
    }
}
```

A tag `img` oferece suporte aos arquivos JPEG, GIF, PNG e SWF.

Rolagem de texto em um campo de texto

Em muitos casos, o texto pode ser mais longo do que o campo de texto que o exibe. Ou você pode ter um campo de texto que permite que um usuário insira mais texto do que o que pode ser exibido de uma vez. É possível usar as propriedades relacionadas à rolagem da classe flash.text.TextField para gerenciar conteúdo longo, tanto na vertical quanto na horizontal.

As propriedades relacionadas a rolagem incluem `TextField.scrollV`, `TextField.scrollH`, `maxScrollV` e `maxScrollH`. Use essas propriedades para responder a eventos, com um clique do mouse ou um pressionamento de tecla.

O exemplo a seguir cria um campo de texto que é de um tamanho definido e contém mais texto do que o campo pode exibir de uma vez. Conforme o usuário clica no campo de texto, o texto rola verticalmente.

```
package
{
    import flash.display.Sprite;
    import flash.text.*;
    import flash.events.MouseEvent;

    public class TextScrollExample extends Sprite
    {
        private var myTextBox:TextField = new TextField();
        private var myText:String = "Hello world and welcome to the show. It's really nice to
meet you. Take your coat off and stay a while. OK, show is over. Hope you had fun. You can go
home now. Don't forget to tip your waiter. There are mints in the bowl by the door. Thank you.
Please come again.";

        public function TextScrollExample()
        {
            myTextBox.text = myText;
            myTextBox.width = 200;
            myTextBox.height = 50;
            myTextBox.multiline = true;
            myTextBox.wordWrap = true;
            myTextBox.background = true;
            myTextBox.border = true;

            var format:TextFormat = new TextFormat();
            format.font = "Verdana";
            format.color = 0xFF0000;
            format.size = 10;

            myTextBox.defaultTextFormat = format;
            addChild(myTextBox);
            myTextBox.addEventListener(MouseEvent.MOUSE_DOWN, mouseDownScroll);
        }

        public function mouseDownScroll(event:MouseEvent):void
        {
            myTextBox.scrollV++;
        }
    }
}
```

Seleção e manipulação de texto

É possível selecionar texto dinâmico ou de entrada. Como as propriedades e os métodos de seleção de texto da classe TextField usam posições do índice para definir a faixa de texto a ser manipulada, é possível selecionar programaticamente texto dinâmico ou de entrada mesmo que você não conheça o conteúdo.

Nota: *Na ferramenta de autoria do Flash, se você escolher a opção selecionável em um campo de texto estático, o campo de texto que é exportado e colocado na lista de exibição será um campo de texto dinâmico regular.*

Seleção de texto

Por padrão, a propriedade `flash.text.TextField.selectable` é `true`, e é possível selecionar texto programaticamente usando o método `setSelection()`.

Por exemplo, é possível definir texto específico em um campo de texto a ser selecionado quando o usuário clica no campo de texto:

```
var myTextField:TextField = new TextField();
myTextField.text = "No matter where you click on this text field the TEXT IN ALL CAPS is selected.";
myTextField.autoSize = TextFieldAutoSize.LEFT;
addChild(myTextField);
addEventListener(MouseEvent.CLICK, selectText);

function selectText(event:MouseEvent):void
{
    myTextField.setSelection(49, 65);
}
```

De maneira semelhante, se você desejar que o texto do campo de texto seja selecionado como o texto é exibido inicialmente, crie uma função de manipulador de eventos que é chamada quando o campo de texto é adicionado à lista de exibição.

Captura de texto selecionado pelo usuário

As propriedades `selectionBeginIndex` e `selectionEndIndex` de TextField que são "somente leitura" e, portanto, não podem ser definidas como texto selecionado programaticamente, podem ser usadas para capturar qualquer coisa que o usuário tenha selecionado no momento. Além disso, campos de texto de entrada podem usar a propriedade `caretIndex`.

Por exemplo, o código a seguir rastreia os valores do índice do texto selecionado pelo usuário:

```
var myTextField:TextField = new TextField();
myTextField.text = "Please select the TEXT IN ALL CAPS to see the index values for the first
and last letters.";
myTextField.autoSize = TextFieldAutoSize.LEFT;
addChild(myTextField);
addEventListener(MouseEvent.MOUSE_UP, selectText);

function selectText(event:MouseEvent):void
{
    trace("First letter index position: " + myTextField.selectionBeginIndex);
    trace("Last letter index position: " + myTextField.selectionEndIndex);
}
```

É possível aplicar uma coleção de propriedades do objeto TextFormat à seleção para alterar a aparência do texto. Para obter mais informações sobre como aplicar uma coleção de propriedades de TextFormat ao texto selecionado, consulte "Formatação de faixas de texto em um campo de texto" na página 446.

Captura da entrada de texto

Por padrão, a propriedade `type` de um campo de texto é definida como `dynamic`. Se você definir a propriedade `type` como `input` usando a classe TextFieldType, poderá coletar a entrada do usuário e salvar o valor para uso em outras partes do aplicativo. Os campos de texto de entrada são úteis para formulários e para qualquer aplicativo que queira que o usuário defina um valor de texto para uso em outro lugar do programa.

Por exemplo, o código a seguir cria um campo de texto de entrada chamado `myTextBox`. Conforme o usuário insere texto no campo, o evento `textInput` é acionado. Um manipulador de eventos chamado `textInputCapture` captura a string de texto inserida e atribui uma variável a ela. O Flash Player ou o AIR exibe o novo texto em outro campo de texto chamado `myOutputBox`.

```
package
{
    import flash.display.Sprite;
    import flash.display.Stage;
    import flash.text.*;
    import flash.events.*;

    public class CaptureUserInput extends Sprite
    {
        private var myTextBox:TextField = new TextField();
        private var myOutputBox:TextField = new TextField();
        private var myText:String = "Type your text here.";

        public function CaptureUserInput()
        {
            captureText();
        }

        public function captureText():void
        {
            myTextBox.type = TextFieldType.INPUT;
            myTextBox.background = true;
            addChild(myTextBox);
            myTextBox.text = myText;
            myTextBox.addEventListener(TextEvent.TEXT_INPUT, textInputCapture);
        }

        public function textInputCapture(event:TextEvent):void
        {
            var str:String = myTextBox.text;
            createOutputBox(str);
        }

        public function createOutputBox(str:String):void
        {
            myOutputBox.background = true;
            myOutputBox.x = 200;
            addChild(myOutputBox);
            myOutputBox.text = str;
        }

    }
}
```

Restrição de entrada de texto

Como os campos de texto de entrada normalmente são usados para formulários ou caixas de diálogo, você pode desejar limitar os tipos de caracteres que um usuário pode inserir em um campo de texto ou mesmo manter o texto oculto, por exemplo, no caso de uma senha. A classe flash.text.TextField tem uma propriedade `displayAsPassword` e uma propriedade `restrict` que podem ser definidas para controlar a entrada do usuário.

A propriedade `displayAsPassword` simplesmente oculta o texto (exibindo-o como uma série de asteriscos) conforme o usuário o digita. Quando `displayAsPassword` está definida como `true`, os comandos Recortar e Copiar e seus atalhos de teclado correspondentes não funcionam. Como mostra o exemplo a seguir, você atribui a propriedade `displayAsPassword` exatamente como o faz com outras propriedades, como background e color:

```
myTextBox.type = TextFieldType.INPUT;
myTextBox.background = true;
myTextBox.displayAsPassword = true;
addChild(myTextBox);
```

A propriedade `restrict` é um pouco mais complicada, pois você precisa especificar quais caracteres o usuário pode digitar em um campo de texto de entrada. Você pode especificar letras, números específicos ou faixas de letras, números e caracteres. O código a seguir permite que o usuário insira apenas letras maiúsculas (e não números ou caracteres especiais) no campo de texto:

```
myTextBox.restrict = "A-Z";
```

O ActionScript 3.0 usa hífens para definir faixas e circunflexos para definir caracteres excluídos. Para obter mais informações sobre como definir o que é restringido em um campo de texto de entrada, consulte a entrada da propriedade `flash.text.TextField.restrict` na Referência dos componentes e da linguagem do ActionScript 3.0.

Formatação de texto

Há várias opções para formatar a exibição de texto programaticamente. É possível definir propriedades diretamente na ocorrência TextField, por exemplo, as propriedades `TextFIeld.thickness`, `TextField.textColor` e `TextField.textHeight` .Ou é possível designar o conteúdo do campo de texto usando a propriedade `htmlText` e usar tags HTML suportadas, como `b`, `i` e `u`. Mas também é possível aplicar objetos TextFormat a campos de texto que contêm texto simples ou objetos StyleSheet a campos de texto que contêm a propriedade `htmlText`. O uso dos objetos TextFormat e StyleSheet fornece o máximo em controle e consistência sobre a aparência do texto em todo o aplicativo. É possível definir um objeto TextFormat ou StyleSheet e aplicá-lo a muitos ou a todos os campos de texto no aplicativo.

Atribuição de formatos de texto

É possível usar a classe TextFormat para definir um número de diferentes propriedades de exibição de texto e aplicá-las a todo o conteúdo de um objeto TextField ou a uma faixa de texto.

O exemplo a seguir aplica um objeto TextFormat a um objeto TextField inteiro e aplica um segundo objeto TextFormat a uma faixa de texto naquele objeto TextField:

```
var tf:TextField = new TextField();
tf.text = "Hello Hello";

var format1:TextFormat = new TextFormat();
format1.color = 0xFF0000;

var format2:TextFormat = new TextFormat();
format2.font = "Courier";

tf.setTextFormat(format1);
var startRange:uint = 6;
tf.setTextFormat(format2, startRange);

addChild(tf);
```

O método `TextField.setTextFormat()` afeta apenas texto que já está exibido no campo texto. Se o conteúdo do TextField for alterado, o aplicativo poderá precisar chamar o método `TextField.setTextFormat()` novamente para reaplicar a formatação. Você também pode definir a propriedade `defaultTextFormat` de TextField para especificar o formato a ser usado para texto inserido pelo usuário.

Aplicação de folhas de estilo em cascata

Campos de texto podem conter texto simples ou texto formatado por HTML. Texto simples é armazenado na propriedade `text` da ocorrência e texto HTML é armazenado na propriedade `htmlText`.

É possível usar declarações de estilo CSS para definir estilos de texto que podem ser aplicados a muitos diferentes campos de texto. Declarações de estilo CSS podem ser criadas no código do aplicativo ou carregadas em tempo de execução de um arquivo CSS externo.

A classe flash.text.StyleSheet manipula estilos CSS. A classe StyleSheet reconhece um conjunto limitado de propriedades CSS. Para obter uma lista detalhada das propriedades de estilo suportadas pela classe StyleSheet, consulte a entrada flash.textStylesheet na Referência dos componentes e da linguagem do ActionScript 3.0.

Conforme mostrado no exemplo a seguir, é possível criar CSS no código e aplicar esses estilos ao texto HTML usando um objeto StyleSheet:

```
var style:StyleSheet = new StyleSheet();

var styleObj:Object = new Object();
styleObj.fontSize = "bold";
styleObj.color = "#FF0000";
style.setStyle(".darkRed", styleObj);

var tf:TextField = new TextField();
tf.styleSheet = style;
tf.htmlText = "<span class = 'darkRed'>Red</span> apple";

addChild(tf);
```

Depois de criar um objeto StyleSheet, o código de exemplo cria um objeto simples para manter um conjunto de propriedades de declaração de estilo. Em seguida, ele chama o método `StyleSheet.setStyle()` que adiciona um novo estilo à folha de estilos com o nome ".darkred". Em seguida, ele aplica a formatação da folha de estilos atribuindo o objeto StyleSheet à propriedade `styleSheet` de TextField.

Para que os estilos CSS tenham efeito, a folha de estilos deve ser aplicada ao objeto TextField antes da propriedade `htmlText` ser definida.

Por design, não é possível editar um campo de texto com uma folha de estilos. Se você tiver um campo de texto de entrada e atribuir uma folha de estilos a ele, o campo de texto mostrará as propriedades da folha de estilos, mas o campo de texto não permitirá que os usuários insiram novo texto nela. Além disso, também não é possível usar as seguintes APIs do ActionScript em um campo de texto com uma folha de estilos atribuída.

- O método `TextField.replaceText()`
- O método `TextField.replaceSelectedText()`
- A propriedade `TextField.defaultTextFormat`
- O método `TextField.setTextFormat()`

Se um campo de texto tiver uma folha de estilos atribuída a ele, mas a propriedade `TextField.styleSheet` for definida posteriormente como `null`, o conteúdo das duas propriedades `TextField.text` e `TextField.htmlText` adicionará tags e atributos a seu conteúdo para incorporar a formatação a partir da folha de estilos atribuída anteriormente. Para preservar a propriedade `htmlText` original, salve-a em uma variável antes de definir a folha de estilos como `null`.

Carregamento de uma arquivo CSS externo

A abordagem CSS para formatação é mais poderosa quando você pode carregar informações CSS de um arquivo externo em tempo de execução. Quando os dados CSS são externos ao próprio aplicativo, é possível alterar o estilo visual do texto no aplicativo sem precisar alterar o código-fonte do ActionScript 3.0. Depois da implantação do aplicativo, é possível alterar um arquivo CSS externo para alterar a aparência do aplicativo, sem precisar implantar novamente o arquivo SWF do aplicativo.

O método `StyleSheet.parseCSS()` converte uma string que contém dados CSS em declarações de estilo no objeto StyleSheet. O exemplo a seguir mostra como ler um arquivo CSS externo e aplicar suas declarações de estilo a um objeto TextField.

Primeiro, este é o conteúdo do arquivo CSS a ser carregado que é denominado example.css:

```
p {
    font-family: Times New Roman, Times, _serif;
    font-size: 14;
}

h1 {
    font-family: Arial, Helvetica, _sans;
    font-size: 20;
    font-weight: bold;
}

.bluetext {
    color: #0000CC;
}
```

Em seguida, está o código ActionScript para uma classe que carrega o arquivo example.css e aplica os estilos ao conteúdo de TextField:

```
package
{
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.URLLoader;
    import flash.net.URLRequest;
    import flash.text.StyleSheet;
    import flash.text.TextField;
    import flash.text.TextFieldAutoSize;

    public class CSSFormattingExample extends Sprite
    {
        var loader:URLLoader;
        var field:TextField;
        var exampleText:String = "<h1>This is a headline</h1>" +
            "<p>This is a line of text. <span class='bluetext'>" +
            "This line of text is colored blue.</span></p>";

        public function CSSFormattingExample():void
        {
            field = new TextField();
            field.width = 300;
```

```
            field.autoSize = TextFieldAutoSize.LEFT;
            field.wordWrap = true;
            addChild(field);

            var req:URLRequest = new URLRequest("example.css");

            loader = new URLLoader();
            loader.addEventListener(Event.COMPLETE, onCSSFileLoaded);
            loader.load(req);
        }

        public function onCSSFileLoaded(event:Event):void
        {
            var sheet:StyleSheet = new StyleSheet();
            sheet.parseCSS(loader.data);
            field.styleSheet = sheet;
            field.htmlText = exampleText;
        }
    }
}
```

Quando os dados CSS são carregados, o método `onCSSFileLoaded()` executa e chama o método `StyleSheet.parseCSS()` para transferir as declarações de estilo para o objeto StyleSheet.

Formatação de faixas de texto em um campo de texto

Um método especialmente útil da classe flash.text.TextField é o método `setTextFormat()`. Usando o `setTextFormat()`, é possível atribuir propriedades específicas ao conteúdo de uma parte de um campo de texto para responder à entrada do usuário, como formulários que precisam lembrar os usuários de que determinadas entradas são necessárias ou alterar a ênfase de uma subseção de uma passagem de texto dentro de um campo de texto conforme o usuário seleciona partes do texto.

O exemplo a seguir usa `TextField.setTextFormat()` em uma faixa de caracteres para alterar a aparência de parte do conteúdo de `myTextField` quando o usuário clica no campo de texto:

```
var myTextField:TextField = new TextField();
myTextField.text = "No matter where you click on this text field the TEXT IN ALL CAPS changes
format.";
myTextField.autoSize = TextFieldAutoSize.LEFT;
addChild(myTextField);
addEventListener(MouseEvent.CLICK, changeText);

var myformat:TextFormat = new TextFormat();
myformat.color = 0xFF0000;
myformat.size = 18;
myformat.underline = true;

function changeText(event:MouseEvent):void
{
    myTextField.setTextFormat(myformat, 49, 65);
}
```

Renderização avançada de texto

O ActionScript 3.0 fornece várias classes no pacote flash.text para controlar as propriedades do texto exibido, incluindo fontes incorporadas, configurações de suavização de borda, controle de canal alfa e outras configurações específicas. A Referência dos componentes e da linguagem do ActionScript 3.0 fornece descrições detalhadas dessas classes e propriedades, incluindo as classes CSMSettings, Font e TextRenderer.

Uso de fontes incorporadas

Quando você especifica uma fonte específica para um TextField no aplicativo, o Flash Player ou o AIR procuram uma fonte de dispositivo (uma fonte que reside no computador do usuário) com o mesmo nome. Se ele não localizar aquela fonte no sistema, ou se o usuário tiver uma versão levemente diferente de uma fonte com aquele nome, a exibição do texto poderá ter uma aparência muito diferente da pretendida.

Para ter certeza de que o usuário vê exatamente a fonte correta, é possível incorporar aquela fonte no arquivo SWF do aplicativo. As fontes incorporadas têm vários benefícios:

- As bordas dos caracteres das fontes incorporadas são suavizadas, principalmente para texto grande.
- É possível girar texto que usa fontes incorporadas.
- O texto da fonte incorporada pode ser transformado em transparente ou semi-transparente.
- É possível usar o estilo da CSS `kerning` com fontes incorporadas.

A maior limitação do uso de fontes incorporadas é que elas aumentam o tamanho do arquivo ou o tamanho do download do aplicativo.

O método exato a ser usado para incorporar um arquivo de fonte no arquivo SWF do aplicativo varia de acordo com o ambiente de desenvolvimento.

Depois de incorporar uma fonte, você pode verificar se um TextField usa a fonte incorporada correta:

- Defina a propriedade `embedFonts` do TextField como `true`.
- Crie um objeto TextFormat, defina sua propriedade `fontFamily` como o nome da fonte incorporada e aplique o objeto TextFormat ao TextField. Ao especificar uma fonte incorporada, a propriedade`fontFamily` deve conter apenas um único nome. Ela não pode usar uma lista delimitada por vírgulas de vários nomes de fontes.
- Se estiver usando estilos CSS para definir fontes para TextFields ou componentes, defina a propriedade CSS `font-family` como o nome da fonte incorporada. A propriedade `font-family` deve conter um único nome e não uma lista de nomes se você desejar especificar uma fonte incorporada.

Incorporação de uma fonte no Flash

A ferramenta de autoria do Flash permite incorporar quase qualquer fonte instalada no sistema, incluindo fontes TrueType e fontes Postscript Tipo 1.

É possível incorporar fontes em um aplicativo de muitas maneiras, incluindo:

- Definir a fonte e as propriedades de estilo de um TextField no Palco e clicar na caixa de seleção Incorporar fontes.
- Criar e referenciar um símbolo de fonte.
- Criar e usar uma biblioteca compartilhada em tempo de execução contendo símbolos de fontes incorporadas.

Para obter mais detalhes sobre como incorporar fontes em aplicativos, consulte “Fontes incorporadas para campos de texto dinâmicos ou de entrada” em *Uso do Flash*.

Controle da nitidez, da espessura e da suavização de borda

Por padrão, o Flash Player ou o AIR determina as configurações dos controles de exibição de texto, como nitidez, espessura e suavização de borda, conforme o texto é redimensionado, muda de cor ou é exibido em vários fundos. Em alguns casos, como quando tem um texto muito pequeno ou muito grande, ou texto em vários fundos exclusivos, talvez você queira manter controle sobre essas configurações. É possível substituir as configurações do Flash Player ou do AIR usando a classe `flash.text.TextRenderer` e as classes associadas, como CSMSettings. Essas classes fornecem um controle preciso sobre a qualidade da renderização de texto incorporado. Para obter mais informações sobre fontes incorporadas, consulte "Uso de fontes incorporadas" na página 447.

Nota: *A propriedade `flash.text.TextField.antiAliasType` deve ter o valor `AntiAliasType.ADVANCED` para que você defina a nitidez, a espessura e a propriedade gridFitType ou use o método `TextRenderer.setAdvancedAntiAliasingTable()`.*

O exemplo a seguir aplica formatação e propriedades de CSM (Modulação de traçado contínua) personalizadas a texto exibido usando uma fonte incorporada chamada `myFont`. Quando o usuário clica no texto exibido, o Flash Player ou o Adobe AIR aplica as configurações personalizadas:

```
var format:TextFormat = new TextFormat();
format.color = 0x336699;
format.size = 48;
format.font = "myFont";

var myText:TextField = new TextField();
myText.embedFonts = true;
myText.autoSize = TextFieldAutoSize.LEFT;
myText.antiAliasType = AntiAliasType.ADVANCED;
myText.defaultTextFormat = format;
myText.selectable = false;
myText.mouseEnabled = true;
myText.text = "Hello World";
addChild(myText);
myText.addEventListener(MouseEvent.CLICK, clickHandler);

function clickHandler(event:Event):void
{
    var myAntiAliasSettings = new CSMSettings(48, 0.8, -0.8);
    var myAliasTable:Array = new Array(myAntiAliasSettings);
    TextRenderer.setAdvancedAntiAliasingTable("myFont", FontStyle.ITALIC,
TextColorType.DARK_COLOR, myAliasTable);
}
```

Trabalho com texto estático

O texto estático é criado somente na Ferramenta autoria. Não é possível instanciar programaticamente texto estático usando o ActionScript. O texto estático será útil se o texto for curto e não precisar ser alterado (como é possível no caso de texto dinâmico). Considere o texto estático como sendo semelhante a um elemento gráfico, como um círculo ou um quadrado desenhado no Palco na ferramenta de autoria do Flash. Embora o texto estático seja mais limitado do que o texto dinâmico, o ActionScript 3.0 não permite que você leia os valores das propriedades do texto estático usando a classe StaticText. É possível usar a classe TextSnapshot para ler valores fora do texto estático.

Acesso a campos de texto estático com a classe StaticText

Normalmente, você usa a classe flash.text.StaticText no painel Ações da ferramenta de autoria do Flash para interagir com uma ocorrência de texto estático posicionada no Palco. Você também pode trabalhar em arquivos ActionScript que interagem com um arquivo SWF que contém texto estático. Em qualquer um dos casos, não é possível instanciar uma ocorrência de texto estático de maneira programática. O texto estático é criado na Ferramenta de autoria.

Para criar uma referência a um campo de texto estático existente, repita os itens na lista de exibição e atribua uma variável. Por exemplo:

```
for (var i = 0; i < this.numChildren; i++) {
var displayitem:DisplayObject = this.getChildAt(i);
if (displayitem instanceof StaticText) {
trace("a static text field is item " + i + " on the display list");
        var myFieldLabel:StaticText = StaticText(displayitem);
        trace("and contains the text: " + myFieldLabel.text);
}
}
```

Quando há uma referência a um campo de texto estático, é possível usar as propriedades daquele campo no ActionScript 3.0. O código a seguir é anexado a um quadro na linha de tempo e assume que uma variável denominada `myFieldLabel` está atribuída a uma referência de texto estático. Um campo de texto dinâmico denominado `myField` é posicionado próximo aos valores `x` e `y` de `myFieldLabel` e exibe o valor de `myFieldLabel` novamente.

```
var myField:TextField = new TextField();
addChild(myField);
myField.x = myFieldLabel.x;
myField.y = myFieldLabel.y + 20;
myField.autoSize = TextFieldAutoSize.LEFT;
myField.text = "and " + myFieldLabel.text
```

Uso da classe TextSnapshot

Para trabalhar programaticamente com uma ocorrência de texto estático, é possível usar a classe flash.text.TextSnapshot para trabalhar com a propriedade `textSnapshot` de um flash.display.DisplayObjectContainer. Ou seja, você cria uma ocorrência de TextSnapshot a partir da propriedade `DisplayObjectContainer.textSnapshot`. Em seguida, é possível aplicar métodos a essa ocorrência para recuperar valores ou selecionar partes do texto estático.

Por exemplo, posicionar um campo de texto estático que contém o texto "TextSnapshot Example" no Palco. Adicionar o seguinte ActionScript ao Quadro 1 da linha de tempo:

```
var mySnap:TextSnapshot = this.textSnapshot;
var count:Number = mySnap.charCount;
mySnap.setSelected(0, 4, true);
mySnap.setSelected(1, 2, false);
var myText:String = mySnap.getSelectedText(false);
trace(myText);
```

A classe TextSnapshot é útil para remover o texto dos campos de texto estático em um arquivo SWF carregado, se você desejar usar o texto como um valor em outra parte de um aplicativo.

Exemplo de TextField: Formatação de texto em estilo jornal

O exemplo de Layout de jornal formata texto para que tenha a aparência de um artigo em um jornal impresso. O texto de entrada pode conter um título, um subtítulo e o corpo do artigo. Com uma largura e uma altura determinadas, este exemplo de Layout de jornal formata o título e o subtítulo para usarem a largura total da área de exibição. O texto do artigo é distribuído em duas ou mais colunas.

Este exemplo ilustra as seguintes técnicas de programação do ActionScript:

- Extensão da classe TextField
- Carregamento e aplicação de um arquivo CSS externo
- Conversão de estilos CSS em objetos TextFormat
- Uso da classe TextLineMetrics para obter informações sobre o tamanho da exibição do texto

Para obter os arquivos de aplicativo deste exemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_br. Os arquivos do aplicativo Layout de notícias podem ser encontrados na pasta Amostras/NewsLayout. O aplicativo consiste nos seguintes arquivos:

Arquivo	Descrição
NewsLayout.mxml ou NewsLayout.fla	A interface do usuário do aplicativo para Flex (MXML) ou Flash (FLA).
StoryLayoutComponent.as	Uma classe UIComponent do Flex que coloca a ocorrência StoryLayout.
StoryLayout.as	A classe principal do ActionScript que organiza todos os componentes de um artigo de jornal para exibição.
FormattedTextField.as	Uma subclasse da classe TextField que gerencia seu próprio objeto TextFormat.
HeadlineTextField.as	Uma subclasse da classe FormattedTextField que ajusta tamanhos de fontes a uma largura desejada.
MultiColumnTextField.as	Uma classe ActionScript que divide texto em duas ou mais colunas.
story.css	Um arquivo CSS que define estilos de texto para o layout.

Leitura do arquivo CSS externo

O aplicativo Layout de jornal começa lendo o texto do artigo de um arquivo XML local. Em seguida, ele lê um arquivo CSS externo que fornece as informações de formatação do título, do subtítulo e do texto principal.

O arquivo CSS define três estilos, um estilo de parágrafo padrão para o artigo e os estilos h1 e h2 do título e do subtítulo respectivamente.

```
p {
    font-family: Georgia, "Times New Roman", Times, _serif;
    font-size: 12;
    leading: 2;
    text-align: justify;
    indent: 24;
}

h1 {
    font-family: Verdana, Arial, Helvetica, _sans;
    font-size: 20;
    font-weight: bold;
    color: #000099;
    text-align: left;
}

h2 {
    font-family: Verdana, Arial, Helvetica, _sans;
    font-size: 16;
    font-weight: normal;
    text-align: left;
}
```

A técnica usada para ler o arquivo CSS externo é a mesma descrita em "Carregamento de uma arquivo CSS externo" na página 445. Quando o arquivo CSS foi carregado, o aplicativo executa o método `onCSSFileLoaded()` mostrado a seguir.

```
public function onCSSFileLoaded(event:Event):void
{
    this.sheet = new StyleSheet();
    this.sheet.parseCSS(loader.data);

    h1Format = getTextStyle("h1", this.sheet);
    if (h1Format == null)
    {
        h1Format = getDefaultHeadFormat();
    }
    h2Format = getTextStyle("h2", this.sheet);
    if (h2Format == null)
    {
        h2Format = getDefaultHeadFormat();
        h2Format.size = 16;
    }
    pFormat = getTextStyle("p", this.sheet);
    if (pFormat == null)
    {
        pFormat = getDefaultTextFormat();
        pFormat.size = 12;
    }
    displayText();
}
```

O método `onCSSFileLoaded()` cria um objeto StyleSheet e faz com que ele analise os dados CSS de entrada. O texto principal do artigo é exibido em um objeto MultiColumnTextField que pode usar um objeto StyleSheet diretamente. No entanto, os campos de título usam a classe HeadlineTextField que usa um objeto TextFormat para sua formatação.

O método `onCSSFileLoaded()` chama o método `getTextStyle()` duas vezes para converter uma declaração de estilo CSS em um objeto TextFormat para uso com cada um dos dois objetos HeadlineTextField.

```
public function getTextStyle(styleName:String, ss:StyleSheet):TextFormat
{
    var format:TextFormat = null;

    var style:Object = ss.getStyle(styleName);
    if (style != null)
    {
        var colorStr:String = style.color;
        if (colorStr != null && colorStr.indexOf("#") == 0)
        {
            style.color = colorStr.substr(1);
        }
        format = new TextFormat(style.fontFamily,
                        style.fontSize,
                        style.color,
                        (style.fontWeight == "bold"),
                        (style.fontStyle == "italic"),
                        (style.textDecoration == "underline"),
                        style.url,
                        style.target,
                        style.textAlign,
                        style.marginLeft,
                        style.marginRight,
                        style.indent,
                        style.leading);

        if (style.hasOwnProperty("letterSpacing"))
        {
            format.letterSpacing = style.letterSpacing;
        }
    }
    return format;
}
```

Os nomes das propriedades e o significado dos valores das propriedades diferem entre declarações de estilo CSS e objetos TextFormat. O método `getTextStyle()` converte valores das propriedades CSS em valores esperados pelo objeto TextFormat.

Organização dos elementos do artigo na página

A classe StoryLayout formata e dispõe o título, o subtítulo e os campos de texto principais em uma organização em estilo jornal. O método `displayText()` inicialmente cria e posiciona os vários campos.

```
public function displayText():void
{
    headlineTxt = new HeadlineTextField(h1Format);
    headlineTxt.wordWrap = true;
    headlineTxt.x = this.paddingLeft;
    headlineTxt.y = this.paddingTop;
    headlineTxt.width = this.preferredWidth;
    this.addChild(headlineTxt);

    headlineTxt.fitText(this.headline, 1, true);

    subtitleTxt = new HeadlineTextField(h2Format);
    subtitleTxt.wordWrap = true;
    subtitleTxt.x = this.paddingLeft;
    subtitleTxt.y = headlineTxt.y + headlineTxt.height;
    subtitleTxt.width = this.preferredWidth;
    this.addChild(subtitleTxt);

    subtitleTxt.fitText(this.subtitle, 2, false);

    storyTxt = new MultiColumnText(this.numColumns, 20,
                        this.preferredWidth, 400, true, this.pFormat);
    storyTxt.x = this.paddingLeft;
    storyTxt.y = subtitleTxt.y + subtitleTxt.height + 10;
    this.addChild(storyTxt);

    storyTxt.text = this.content;
...
```

Cada campo é posicionado abaixo do campo anterior configurando sua propriedade `y` como igual à propriedade `y` do campo anterior mais sua altura. Esse cálculo de posicionamento dinâmico é necessário porque os objetos HeadlineTextField e MultiColumnTextField podem ter a altura alterada para se ajustarem ao conteúdo.

Alteração do tamanho da fonte para ajuste ao tamanho do campo

Determinados uma largura em pixels e um número máximo de linhas a serem exibidos, o HeadlineTextField altera o tamanho da fonte para ajustar o texto ao campo. Se o texto for curto, o tamanho da fonte será muito grande, criando um título em estilo tablóide. Se o texto for longo, o tamanho da fonte será menor.

O método `HeadlineTextField.fitText()` mostrado a seguir faz o trabalho de redimensionamento da fonte:

```
public function fitText(msg:String, maxLines:uint = 1, toUpper:Boolean = false,
targetWidth:Number = -1):uint
{
    this.text = toUpper ? msg.toUpperCase() : msg;

    if (targetWidth == -1)
    {
        targetWidth = this.width;
    }

    var pixelsPerChar:Number = targetWidth / msg.length;

    var pointSize:Number = Math.min(MAX_POINT_SIZE, Math.round(pixelsPerChar * 1.8 * maxLines));

    if (pointSize < 6)
    {
        // the point size is too small
        return pointSize;
    }

    this.changeSize(pointSize);

    if (this.numLines > maxLines)
    {
        return shrinkText(--pointSize, maxLines);
    }
    else
    {
        return growText(pointSize, maxLines);
    }
}

public function growText(pointSize:Number, maxLines:uint = 1):Number
{
    if (pointSize >= MAX_POINT_SIZE)
    {
        return pointSize;
    }

    this.changeSize(pointSize + 1);

    if (this.numLines > maxLines)
    {
        // set it back to the last size
        this.changeSize(pointSize);
        return pointSize;
    }
    else
    {
        return growText(pointSize + 1, maxLines);
    }
```

```
}

public function shrinkText(pointSize:Number, maxLines:uint=1):Number
{
    if (pointSize <= MIN_POINT_SIZE)
    {
        return pointSize;
    }

    this.changeSize(pointSize);

    if (this.numLines > maxLines)
    {
        return shrinkText(pointSize - 1, maxLines);
    }
    else
    {
        return pointSize;
    }
}
```

O método `HeadlineTextField.fitText()` usa uma técnica recursiva simples para redimensionar a fonte. Primeiro ele estima um número médio de pixels por caractere no texto e a partir daí calcula um tamanho de ponto inicial. Em seguida, ele altera o tamanho da fonte e verifica se houve quebra de linha no texto para criar mais do que o número máximo de linhas de texto. Se houver linhas em excesso, ele chamará o método `shrinkText()` para reduzir o tamanho da fonte e tentar novamente. Se não houver muitas linhas, ele chamará o método `growText()` para aumentar o tamanho da fonte e tentar novamente. O processo pára no ponto em que se o tamanho da fonte for incrementado em mais um ponto, linhas em excesso serão criadas.

Divisão do texto em várias colunas

A classe MultiColumnTextField propaga texto entre vários objetos TextField que, em seguida, são organizados como colunas de jornal.

O construtor `MultiColumnTextField()` primeiro cria uma matriz de objetos TextField, um para cada coluna, conforme mostrado aqui:

```
for (var i:int = 0; i < cols; i++)
{
    var field:TextField = new TextField();
    field.multiline = true;
    field.autoSize = TextFieldAutoSize.NONE;
    field.wordWrap = true;
    field.width = this.colWidth;
    field.setTextFormat(this.format);
    this.fieldArray.push(field);
    this.addChild(field);
}
```

Cada objeto TextField é adicionado à matriz e adicionado à lista de exibição com o método `addChild()`.

Sempre que a propriedade `text` ou a propriedade `styleSheet` de StoryLayout é alterada, ele chama o método `layoutColumns()` para re-exibir o texto. O método `layoutColumns()` chama o método `getOptimalHeight()` para calcular a altura correta do pixel que é necessária para ajustar todo o texto à largura do layout fornecido.

```
public function getOptimalHeight(str:String):int
{
    if (field.text == "" || field.text == null)
    {
        return this.preferredHeight;
    }
    else
    {
        this.linesPerCol = Math.ceil(field.numLines / this.numColumns);

        var metrics:TextLineMetrics = field.getLineMetrics(0);
        this.lineHeight = metrics.height;
        var prefHeight:int = linesPerCol * this.lineHeight;

        return prefHeight + 4;
    }
}
```

Primeiro, o método `getOptimalHeight()` calcula a largura de cada coluna. Em seguida, ele define a largura e a propriedade `htmlText` do primeiro objeto TextField da matriz. O método `getOptimalHeight()` usa esse primeiro objeto TextField para descobrir o número total de quebras de linha no texto, e a partir daí identifica quantas linhas devem estar em cada coluna. Em seguida, ele chama o método `TextField.getLineMetrics()` para recuperar um objeto TextLineMetrics que contém detalhes sobre o tamanho do texto na primeira linha. A propriedade `TextLineMetrics.height` representa a altura total de uma linha de texto, em pixels, incluindo a elevação, a descendência e a entrelinha. A altura ideal do objeto MultiColumnTextField é portanto a altura da linha multiplicada pelo número de linhas por coluna, mais 4 para considerar a borda de dois pixels na parte superior e inferior do objeto TextField.

Este é o código do método `layoutColumns()` completo:

```
public function layoutColumns():void
{
    if (this._text == "" || this._text == null)
    {
        return;
    }

    var field:TextField = fieldArray[0] as TextField;
    field.text = this._text;
    field.setTextFormat(this.format);

    this.preferredHeight = this.getOptimalHeight(field);

    var remainder:String = this._text;
    var fieldText:String = "";
    var lastLineEndedPara:Boolean = true;

    var indent:Number = this.format.indent as Number;

    for (var i:int = 0; i < fieldArray.length; i++)
    {
        field = this.fieldArray[i] as TextField;

        field.height = this.preferredHeight;
        field.text = remainder;

        field.setTextFormat(this.format);
```

```
var lineLen:int;
if (indent > 0 && !lastLineEndedPara && field.numLines > 0)
{
    lineLen = field.getLineLength(0);
    if (lineLen > 0)
    {
        field.setTextFormat(this.firstLineFormat, 0, lineLen);
        }
    }

field.x = i * (colWidth + gutter);
field.y = 0;

remainder = "";
fieldText = "";

var linesRemaining:int = field.numLines;
var linesVisible:int = Math.min(this.linesPerCol, linesRemaining);

for (var j:int = 0; j < linesRemaining; j++)
{
    if (j < linesVisible)
    {
        fieldText += field.getLineText(j);
    }
    else
    {
        remainder +=field.getLineText(j);
    }
}

field.text = fieldText;

field.setTextFormat(this.format);

if (indent > 0 && !lastLineEndedPara)
{
    lineLen = field.getLineLength(0);
    if (lineLen > 0)
    {
        field.setTextFormat(this.firstLineFormat, 0, lineLen);
    }
}

var lastLine:String = field.getLineText(field.numLines - 1);
var lastCharCode:Number = lastLine.charCodeAt(lastLine.length - 1);
```

```
        if (lastCharCode == 10 || lastCharCode == 13)
        {
        lastLineEndedPara = true;
        }
        else
        {
        lastLineEndedPara = false;
        }

        if ((this.format.align == TextFormatAlign.JUSTIFY) &&
                (i < fieldArray.length - 1))
        {
        if (!lastLineEndedPara)
        {
            justifyLastLine(field, lastLine);
        }
    }
    }
}
```

Após a propriedade `preferredHeight` ser definida chamando o método `getOptimalHeight()`, o método `layoutColumns()` repete os objetos TextField definindo a altura de cada um como o valor de `preferredHeight`. Em seguida, o método `layoutColumns()` distribui apenas linhas de texto suficientes para cada campo, de maneira que nenhuma rolagem ocorra em nenhum campo individual e o texto de cada campo sucessivo comece onde o texto no campo anterior terminou. Se o estilo de alinhamento do texto tiver sido definido como "justificar" o método `justifyLastLine()` será chamado para justificar a linha final do texto em um campo. Caso contrário, essa última linha será tratada como uma linha de final de parágrafo e não será justificada.

Uso do Mecanismo de texto do Flash

O FTE (Mecanismo de texto do Flash) oferece suporte de baixo nível para controle sofisticado de métricas de texto, formatação e texto bidirecional. Ele oferece fluxo de texto aprimorado e suporte a idioma avançado. Embora possa ser usado para criar e gerenciar elementos de texto simples, o FTE foi desenvolvido principalmente como uma base para os desenvolvedores criarem componentes de manipulação de texto. Como tal, o Mecanismo de texto do Flash assume um nível mais avançado de experiência de programação. Para exibir elementos de texto simples, consulte as seções anteriores que descrevem o uso do `TextField` e dos objetos relacionados.

Criação e exibição de texto

As classes que compõem o Mecanismo de texto do Flash permitem criar, formatar e controlar texto. As seguintes classes são os blocos básicos de construção para criar e exibir texto com o Mecanismo de texto do Flash:

- TextElement/GraphicElement/GroupElement - incluem o conteúdo de uma ocorrência de TextBlock.
- ElementFormat - especifica atributos de formatação para o conteúdo de uma ocorrência de TextBlock.
- TextBlock - a fábrica para criação de um parágrafo de texto.
- TextLine - uma linha de texto criada no TextBlock.

Para exibir texto, crie um objeto TextElement a partir de um String e de um objeto ElementFormat, que especifica as características de formatação, e atribua o TextElement à propriedade `content` de um objeto TextBlock. Crie as linhas de texto para exibição chamando o método `TextBlock.createTextLine()`. O código a seguir, por exemplo, usa essas classes FTE para exibir "Hello World! This is Flash Text Engine!" usando o formato e os valores de fonte padrão.

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class HelloWorldExample extends Sprite
    {
        public function HelloWorldExample()
        {
            var str = "Hello World! This is Flash Text Engine!";
            var format:ElementFormat = new ElementFormat();
            var textElement:TextElement = new TextElement(str, format);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = textElement;

            var textLine1:TextLine = textBlock.createTextLine(null, 300);
            addChild(textLine1);
            textLine1.x = 30;
            textLine1.y = 30;
        }
    }
}
```

Os parâmetros de `createTextLine()` especificam a linha da qual iniciar a nova linha e a largura da linha em pixels. A linha da qual iniciar a nova linha normalmente é a linha anterior, mas, no caso da primeira linha, ela é `null`.

Adição de objetos GraphicElement e GroupElement

É possível atribuir um objeto GraphicElement a um objeto TextBlock para exibir uma imagem ou um elemento gráfico. Basta criar uma ocorrência da classe GraphicElement de um gráfico ou de uma imagem e atribuir a ocorrência à propriedade `TextBlock.content`. Crie a linha de texto chamando `TextBlock.createTextline()` como normalmente o faz. O seguinte exemplo cria duas linhas de texto, uma com um objeto GraphicElement e uma com um objeto TextElement.

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.display.Graphics;

    public class GraphicElementExample extends Sprite
    {
        public function GraphicElementExample()
        {
            var str:String = "Beware of Dog!";

            var triangle:Shape = new Shape();
            triangle.graphics.beginFill(0xFF0000, 1);
            triangle.graphics.lineStyle(3);
            triangle.graphics.moveTo(30, 0);
            triangle.graphics.lineTo(60, 50);
            triangle.graphics.lineTo(0, 50);
            triangle.graphics.lineTo(30, 0);
            triangle.graphics.endFill();

            var format:ElementFormat = new ElementFormat();
            format.fontSize = 20;

            var graphicElement:GraphicElement = new GraphicElement(triangle, triangle.width,
triangle.height, format);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = graphicElement;
            var textLine1:TextLine = textBlock.createTextLine(null, triangle.width);
            textLine1.x = 50;
            textLine1.y = 110;
            addChild(textLine1);

            var textElement:TextElement = new TextElement(str, format);
            textBlock.content = textElement;
            var textLine2 = textBlock.createTextLine(null, 300);
            addChild(textLine2);
            textLine2.x = textLine1.x - 30;
            textLine2.y = textLine1.y + 15;
        }
    }
}
```

Você pode criar um objeto GroupElement para criar um grupo de objetos TextElement, GraphicElement e outro GroupElement para atribuição à propriedade `content` de um objeto TextBlock. O parâmetro para o construtor `GroupElement()` é um Vetor que aponta para o texto, o gráfico e os elementos de grupo que compõem o grupo. O exemplo a seguir agrupa dois elementos gráficos e um elemento de texto e os atribui como uma unidade a um bloco de texto.

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.display.Graphics;

    public class GroupElementExample extends Sprite
    {
        public function GroupElementExample()
        {
            var str:String = "Beware of Alligators!";

            var triangle1:Shape = new Shape();
            triangle1.graphics.beginFill(0xFF0000, 1);
            triangle1.graphics.lineStyle(3);
            triangle1.graphics.moveTo(30, 0);
            triangle1.graphics.lineTo(60, 50);
            triangle1.graphics.lineTo(0, 50);
            triangle1.graphics.lineTo(30, 0);
            triangle1.graphics.endFill();

            var triangle2:Shape = new Shape();
            triangle2.graphics.beginFill(0xFF0000, 1);
            triangle2.graphics.lineStyle(3);
            triangle2.graphics.moveTo(30, 0);
            triangle2.graphics.lineTo(60, 50);
            triangle2.graphics.lineTo(0, 50);
            triangle2.graphics.lineTo(30, 0);
            triangle2.graphics.endFill();

            var format:ElementFormat = new ElementFormat();
            format.fontSize = 20;
            var graphicElement1:GraphicElement = new GraphicElement(triangle1,
triangle1.width, triangle1.height, format);
            var textElement:TextElement = new TextElement(str, format);
            var graphicElement2:GraphicElement = new GraphicElement(triangle2,
triangle2.width, triangle2.height, format);
            var groupVector:Vector.<ContentElement> = new Vector.<ContentElement>();
            groupVector.push(graphicElement1, textElement, graphicElement2);
            var groupElement = new GroupElement(groupVector);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = groupElement;
            var textLine:TextLine = textBlock.createTextLine(null, 800);
            addChild(textLine);
            textLine.x = 100;
            textLine.y = 200;
        }
    }
}
```

Substituição de texto

É possível substituir texto em uma ocorrência de TextBlock chamando `TextElement.replaceText()` para substituir texto no TextElement atribuído à propriedade `TextBlock.content`. O exemplo a seguir usa `repaceText()` para inserir texto no início da linha, acrescentar texto no final da linha e substituir texto no meio da linha.

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class ReplaceTextExample extends Sprite
    {
        public function ReplaceTextExample()
        {

            var str:String = "Lorem ipsum dolor sit amet";
            var fontDescription:FontDescription = new FontDescription("Arial");
            var format:ElementFormat = new ElementFormat(fontDescription);
            format.fontSize = 14;
            var textElement:TextElement = new TextElement(str, format);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = textElement;
            createLine(textBlock, 10);
            textElement.replaceText(0, 0, "A text fragment: ");
            createLine(textBlock, 30);
            textElement.replaceText(43, 43, "...");
            createLine(textBlock, 50);
            textElement.replaceText(23, 28, "(ipsum)");
            createLine(textBlock, 70);
        }

        function createLine(textBlock:TextBlock, y:Number):void {
            var textLine:TextLine = textBlock.createTextLine(null, 300);
            textLine.x = 10;
            textLine.y = y;
            addChild(textLine);
        }
    }
}
```

O método `replaceText()` substitui o texto especificado pelos parâmetros `beginIndex` e`endIndex` pelo texto especificado pelo parâmetro `newText`. Se os valores dos parâmetros `beginIndex` e `endIndex` forem iguais, `replaceText()` inserirá o texto especificado naquele local. Caso contrário, ele substituirá os caracteres especificados por`beginIndex` e `endIndex` pelo novo texto.

Manipulação de eventos no FTE

É possível adicionar ouvintes de eventos a uma ocorrência de TextLine exatamente como em outros objetos de exibição. Por exemplo, é possível detectar quando um usuário rola o mouse sobre uma linha de texto ou clica na linha. O exemplo a seguir detecta esses dois eventos. Quando o mouse é rolado sobre a linha, o cursor é alterado para um cursor de botão e quando você clica na linha, ela muda de cor.

```
package
{
    import flash.text.engine.*;
    import flash.ui.Mouse;
    import flash.display.Sprite
    import flash.events.MouseEvent;
    import flash.events.EventDispatcher;

    public class EventHandlerExample extends Sprite
    {
        var textBlock:TextBlock = new TextBlock();

        public function EventHandlerExample():void
        {
            var str:String = "I'll change color if you click me.";
            var fontDescription:FontDescription = new FontDescription("Arial");
            var format:ElementFormat = new ElementFormat(fontDescription, 18);
            var textElement = new TextElement(str, format);
            textBlock.content = textElement;
            createLine(textBlock);
        }

        private function createLine(textBlock:TextBlock):void
        {
            var textLine:TextLine = textBlock.createTextLine(null, 500);
            textLine.x = 30;
            textLine.y = 30;
            addChild(textLine);
            textLine.addEventListener("mouseOut", mouseOutHandler);
            textLine.addEventListener("mouseOver", mouseOverHandler);
            textLine.addEventListener("click", clickHandler);
        }

        private function mouseOverHandler(event:MouseEvent):void
        {
            Mouse.cursor = "button";
        }

        private function mouseOutHandler(event:MouseEvent):void
        {
            Mouse.cursor = "arrow";
        }

        function clickHandler(event:MouseEvent):void {
            if(textBlock.firstLine)
                removeChild(textBlock.firstLine);
            var newFormat:ElementFormat = textBlock.content.elementFormat.clone();
```

```
            switch(newFormat.color)
            {
                case 0x000000:
                    newFormat.color = 0xFF0000;
                    break;
                case 0xFF0000:
                    newFormat.color = 0x00FF00;
                    break;
                case 0x00FF00:
                    newFormat.color = 0x0000FF;
                    break;
                case 0x0000FF:
                    newFormat.color = 0x000000;
                    break;
            }
            textBlock.content.elementFormat = newFormat;
            createLine(textBlock);
        }
    }
}
```

Espelhamento de eventos

Também é possível espelhar eventos em um bloco de texto ou em uma parte de um bloco de texto para um dispatcher de eventos. Primeiro, crie uma ocorrência de EventDispatcher e, em seguida, atribua-a à propriedade `eventMirror` de uma ocorrência de TextElement. Se o bloco de texto consistir em um único elemento de texto, o mecanismo de texto espelhará eventos para todo o bloco de texto. Se o bloco de texto consistir em vários elementos de texto, o mecanismo de texto espelhará eventos apenas para as ocorrências de TextElement que tiverem a propriedade `eventMirror` definida. O texto no exemplo a seguir consiste em três elementos: a palavra "Click", a palavra "here" e a string "to see me in italic". O exemplo atribui um despachador de eventos ao segundo elemento de texto, a palavra "here", e adiciona um ouvinte de eventos, o método `clickHandler()`. O método `clickHandler()` altera o texto para itálico. Ele também substitui o conteúdo do terceiro elemento de texto para que seja "Click here to see me in normal font!".

```
package
{
    import flash.text.engine.*;
    import flash.ui.Mouse;
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    import flash.events.EventDispatcher;

    public class EventMirrorExample extends Sprite
    {
        var fontDescription:FontDescription = new FontDescription("Helvetica", "bold");
        var format:ElementFormat = new ElementFormat(fontDescription, 18);
        var textElement1 = new TextElement("Click ", format);
        var textElement2 = new TextElement("here ", format);
        var textElement3 = new TextElement("to see me in italic! ", format);
        var textBlock:TextBlock = new TextBlock();

        public function EventMirrorExample()
        {
            var myEvent:EventDispatcher = new EventDispatcher();

            myEvent.addEventListener("click", clickHandler);
            myEvent.addEventListener("mouseOut", mouseOutHandler);
```

```
            myEvent.addEventListener("mouseOver", mouseOverHandler);

            textElement2.eventMirror=myEvent;

            var groupVector:Vector.<ContentElement> = new Vector.<ContentElement>;
            groupVector.push(textElement1, textElement2, textElement3);
            var groupElement:GroupElement = new GroupElement(groupVector);

            textBlock.content = groupElement;
            createLines(textBlock);
        }

        private function clickHandler(event:MouseEvent):void
        {
            var newFont:FontDescription = new FontDescription();
            newFont.fontWeight = "bold";

            var newFormat:ElementFormat = new ElementFormat();
            newFormat.fontSize = 18;
            if(textElement3.text == "to see me in italic! ") {
                newFont.fontPosture = FontPosture.ITALIC;
                textElement3.replaceText(0,21, "to see me in normal font! ");
            }
            else {
                newFont.fontPosture = FontPosture.NORMAL;
                textElement3.replaceText(0, 26, "to see me in italic! ");
            }
            newFormat.fontDescription = newFont;
            textElement1.elementFormat = newFormat;
            textElement2.elementFormat = newFormat;
            textElement3.elementFormat = newFormat;
            createLines(textBlock);
        }

        private function mouseOverHandler(event:MouseEvent):void
        {
            Mouse.cursor = "button";
        }

        private function mouseOutHandler(event:MouseEvent):void
        {
                Mouse.cursor = "arrow";
        }

        private function createLines(textBlock:TextBlock):void
        {
            if(textBlock.firstLine)
                removeChild (textBlock.firstLine);
            var textLine:TextLine = textBlock.createTextLine (null, 300);
            textLine.x = 15;
            textLine.y = 20;
            addChild (textLine);
        }
    }
}
```

As funções `mouseOverHandler()` e `mouseOutHandler()` definem o cursor como um cursor de botão quando está sobre a palavra "here" e novamente como uma seta quando não está.

Formatação de texto

Um objeto `TextBlock` é uma fábrica para criação de linhas de texto. O conteúdo de um `TextBlock` é atribuído por meio do objeto `TextElement`. Um objeto `ElementFormat` manipula a formatação do texto. A classe ElementFormat define propriedades, como alinhamento de linha de base, kerning, rastreamento, rotação de texto e tamanho, cor e maiúsculas e minúsculas da fonte. Ela também inclui um objeto `FontDescription` discutido mais detalhadamente em "Trabalho com fontes" na página 469.

Uso do objeto ElementFormat

O construtor do objeto `ElementFormat` utiliza qualquer parâmetro de uma longa lista de parâmetros, incluindo um `FontDescription`. Também é possível definir essas propriedades fora do construtor. O exemplo a seguir mostra o relacionamento dos vários objetos para definir e exibir uma linha de texto simples:

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class ElementFormatExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef:ElementFormat;
        private var fd:FontDescription = new FontDescription();
        private var str:String;
        private var tl:TextLine;

        public function ElementFormatExample()
        {
            fd.fontName = "Garamond";
            ef = new ElementFormat(fd);
            ef.fontSize = 30;
            ef.color = 0xFF0000;
            str = "This is flash text";
            te = new TextElement(str, ef);
            tb.content = te;
            tl = tb.createTextLine(null,600);
            addChild(tl);
        }
    }
}
```

Cor e transparência da fonte (alfa)

A propriedade `color` do objeto `ElementFormat` define a cor da fonte. O valor é um inteiro que representa os componentes RGB da cor. Por exemplo, 0xFF0000 para vermelho e 0x00FF00 para verde. O padrão é preto (0x000000).

A propriedade `alpha` define o valor da transparência alfa de um elemento (`TextElement` e `GraphicElement`). Os valores podem variar de 0 (totalmente transparente) a 1 (totalmente opaco, que é o padrão). Os elementos com um `alpha` igual a 0 são invisíveis, mas ainda ativos. Esse valor é multiplicado por qualquer valor alfa herdado, o que torna o elemento mais transparente.

```
var ef:ElementFormat = new ElementFormat();
ef.alpha = 0.8;
ef.color = 0x999999;
```

Alinhamento e deslocamento da linha de base

A fonte e o tamanho do maior texto em uma linha determina sua linha de base dominante. É possível substituir esses valores definindo `TextBlock.baselineFontDescription` e`TextBlock.baselineFontSize`. É possível alinhar a linha de base dominante com uma das várias linhas de base existentes no texto. Essas linhas de base incluem a linha elevada e a linha descendente ou a parte superior, a parte central ou a parte inferior da ideografia.

***A.** Elevação **B.** Linha de base **C.** Descida **D.** altura de x*

No objeto `ElementFormat`, três propriedades determinam as características da linha de base e do alinhamento. A propriedade `alignmentBaseline` define a linha de base principal de um`TextElement` ou `GraphicElement`. Essa linha de base é a linha de "encaixe" do elemento e é nessa posição que a linha de base dominante de todo o texto se alinha.

A propriedade `dominantBaseline` especifica qual das várias linhas de base do elemento deverá ser usada, o que determina a posição vertical do elemento na linha. O valor padrão é `TextBaseline.ROMAN`, mas ele também pode ser definido para fazer com que as linhas de base `IDEOGRAPHIC_TOP` ou `IDEOGRAPHIC_BOTTOM` sejam dominantes.

A propriedade `baselineShift` move a linha de base em um determinado número de pixels no eixo y. Em texto normal (não girado), um valor positivo move a linha de base para baixo e um valor negativo a move para cima.

Maiúsculas/minúsculas tipográficas

A propriedade `TypographicCase` de `ElementFormat` especifica maiúsculas/minúsculas do texto, como maiúsculas, minúsculas ou versalete.

```
var ef_Upper:ElementFormat = new ElementFormat();
ef_Upper.typographicCase = TypographicCase.UPPERCASE;

var ef_SmallCaps:ElementFormat = new ElementFormat();
ef_SmallCaps.typographicCase = TypographicCase.SMALL_CAPS;
```

Rotação do texto

É possível girar um bloco de texto ou os glifos dentro de um segmento de texto em incrementos de 90 graus. A classe TextRotation define as seguintes constantes para definir a rotação do bloco de texto e do glifo:

Constante	Valor	Descrição
AUTO	"automático"	Especifica rotação de 90 gruas no sentido anti-horário. Usado normalmente com texto asiático vertical para girar apenas glifos que precisam de rotação.
ROTATE_0	"rotate_0"	Especifica que não haverá rotação.

Constante	Valor	Descrição
ROTATE_180	"rotate_180"	Especifica rotação de 180 graus.
ROTATE_270	"rotate_270"	Especifica rotação de 270 graus.
ROTATE_90	"rotate_90"	Especifica rotação de 90 graus no sentido horário.

Para girar as linhas de texto em um bloco de texto, defina a propriedade `TextBlock.lineRotation` antes de chamar o método `TextBlock.createTextLine()` para criar a linha de texto.

Para girar os glifos dentro de um bloco de texto ou de um segmento, defina a propriedade `ElementFormat.textRotation` como o número de graus desejados para a rotação dos glifos. Um glifo é a forma que compõe um caractere ou uma parte de um caractere que consiste em vários glifos. A letra a e o ponto em um i, por exemplo, são glifos.

A rotação de glifos é relevante em alguns idiomas asiáticos em que você queira girar as linhas na vertical, mas não os caracteres dentro das linhas. Para obter mais informações sobre rotação de texto asiático, consulte "Justificação de texto do Leste Asiático" na página 473.

Este é um exemplo de rotação de um bloco de texto e dos glifos constituintes, como em texto asiático. O exemplo também usa fonte japonesa:

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class RotationExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef:ElementFormat;
        private var fd:FontDescription = new FontDescription();
        private var str:String;
        private var tl:TextLine;

        public function RotationExample()
        {
            fd.fontName = "MS Mincho";
            ef = new ElementFormat(fd);
            ef.textRotation = TextRotation.AUTO;
            str = "This is rotated Japanese text";
            te = new TextElement(str, ef);
            tb.lineRotation = TextRotation.ROTATE_90;
            tb.content = te;
            tl = tb.createTextLine(null,600);
            addChild(tl);
        }
    }
}
```

Bloqueio e clonagem de ElementFormat

Quando um objeto `ElementFormat` é atribuído a qualquer tipo de `ContentElement`, sua propriedade `locked` é automaticamente definida como `true`. A tentativa de modificar um objeto `ElementFormat` bloqueado lança um `IllegalOperationError`. A prática recomendada é definir completamente um objeto desse tipo antes de atribuir-lhe uma ocorrência de `TextElement`.

Para modificar uma ocorrência de `ElementFormat` existente, primeiro verifique sua propriedade `locked`. Se ela for `true`, use o método `clone()` para criar uma cópia não bloqueada do objeto. As propriedades desse objeto desbloqueado podem ser alteradas e, em seguida, ele pode ser atribuído à ocorrência de `TextElement`. Quaisquer novas linhas criadas a partir dele terão a nova formatação. As linhas anteriores criadas a partir desse mesmo objeto e que usam o formato antigo não serão alteradas.

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class ElementFormatCloneExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef1:ElementFormat;
        private var ef2:ElementFormat;
        private var fd:FontDescription = new FontDescription();

        public function ElementFormatCloneExample()
        {
            fd.fontName = "Garamond";
            ef1 = new ElementFormat(fd);
            ef1.fontSize = 24;
            var str:String = "This is flash text";
            te = new TextElement(str, ef);
            tb.content = te;
            var tx1:TextLine = tb.createTextLine(null,600);
            addChild(tx1);

            ef2 = (ef1.locked) ? ef1.clone() : ef1;
            ef2.fontSize = 32;
            tb.content.elementFormat = ef2;
            var tx2:TextLine = tb.createTextLine(null,600);
            addChild(tx2);
        }
    }
}
```

Trabalho com fontes

O objeto `FontDescription` é usado em conjunto com `ElementFormat` para identificar um tipo de fonte e definir algumas de suas características. Essas características incluem o nome, o peso, a postura e a renderização da fonte e como localizá-la (de dispositivo versus incorporada).

Nota: *O FTE não oferece suporte a fontes de Tipo 1 ou fontes de bitmap, como Tipo 3, ATC, sfnt-wrapped CID ou Naked CID.*

Definição de características de fontes (objeto FontDescription)

A propriedade `fontName` do objeto `FontDescription` pode ser um único nome ou uma lista de nomes separados por vírgula. Por exemplo, em uma lista como "Arial, Helvetica, _sans", primeiro o Flash Player ou o AIR procura por "Arial", depois por "Helvetica" e, por último, por "_sans" se não conseguir encontrar nenhuma das primeiras duas fontes. O conjunto de nomes de fontes inclui três nomes de fontes de dispositivo genéricos: "_sans", "_serif" e "_typewriter". Eles mapeiam para fontes de dispositivo específicas, dependendo do sistema de reprodução. É uma boa prática especificar nomes padrão, como esses, em todas as descrições de fonte que usam fontes de dispositivo. Se nenhum `fontName` for especificado, "_serif" será usado como padrão.

A propriedade `fontPosture` pode ser definida como a padrão (`FontPosture.NORMAL`) ou itálico (`FontPosture.ITALIC`). A propriedade `fontWeight` pode ser definida como a padrão (`FontWeight.NORMAL`) ou como negrito (`FontWeight.BOLD`).

```
var fd1:FontDescription = new FontDescription();
fd1.fontName = "Arial, Helvetica, _sans";
fd1.fontPosture = FontPosture.NORMAL;
fd1.fontWeight = FontWeight.BOLD;
```

Fontes de dispositivo versus incorporadas

A propriedade `fontLookup` do objeto `FontDescription` especifica se o Flash Player e o AIR devem procurar uma fonte de dispositivo ou uma fonte incorporada para renderizar texto. Se for especificada uma fonte de dispositivo (`FontLookup.DEVICE`), o tempo de execução procurará pela fonte no sistema de reprodução. Especificar uma fonte incorporada (`FontLookup.EMBEDDED_CFF`) faz com que o tempo de execução procure uma fonte incorporada com o nome especificado no arquivo SWF. Somente fontes CFF (Formato de fonte compacta) funcionam com essa configuração. Se a fonte especificada não for localizada, uma fonte de dispositivo de fallback será utilizada.

Fontes de dispositivo resultam em um tamanho menor do arquivo SW. Fontes incorporadas conferem maior fidelidade entre plataformas.

```
var fd1:FontDescription = new FontDescription();
fd1.fontLookup = FontLookup.EMBEDDED_CFF;
fd1.fontName = "Garamond, _serif";
```

Modo de renderização e referência

A renderização do CFF (Formato de fonte compacta) é um novo recurso do Flash Player 10. Esse tipo de renderização de fonte torna o texto mais legível e permite exibição de melhor qualidade de fontes em tamanhos pequenos. Essa configuração aplica-se apenas a fontes incorporadas. `FontDescription` é padronizada para essa configuração (`RenderingMode.CFF`) para a propriedade `renderingMode`. É possível definir essa propriedade como `RenderingMode.NORMAL` para corresponder ao tipo de renderização usado pelo Flash Player 7 ou por versões anteriores.

Quando a renderização de CFF é selecionada, uma segunda propriedade, `cffHinting`, controla como os troncos horizontais de uma fonte se ajustam à grade de subpixels. O valor padrão, `CFFHinting.HORIZONTAL_STEM`, usa referência de CFF. A configuração dessa propriedade como `CFFHinting.NONE` remove a referência, o que é apropriado para animação ou para grandes tamanhos de fonte.

```
var fd1:FontDescription = new FontDescription();
fd1.renderingMode = RenderingMode.CFF;
fd1.cffHinting = CFFHinting.HORIZONTAL_STEM;
```

Bloqueio e clonagem de FontDescription

Quando um objeto `FontDescription` é atribuído a um `ElementFormat`, sua propriedade `locked` é automaticamente definida como `true`. A tentativa de modificar um objeto `FontDescription` bloqueado lança um `IllegalOperationError`. A prática recomendada é definir completamente um objeto desse tipo antes de atribuir-lhe um `ElementFormat`.

Para modificar um `FontDescription` existente, primeiro verifique sua propriedade `locked`. Se ela for `true`, use o método `clone()` para criar uma cópia não bloqueada do objeto. As propriedades desse objeto desbloqueado podem ser alteradas e, em seguida, ele pode ser atribuído a `ElementFormat`. Quaisquer novas linhas criadas a partir desse `TextElement` terão a nova formatação. Linhas anteriores criadas a partir desse mesmo objeto não serão alteradas.

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class FontDescriptionCloneExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef1:ElementFormat;
        private var ef2:ElementFormat;
        private var fd1:FontDescription = new FontDescription();
        private var fd2:FontDescription;

        public function FontDescriptionCloneExample()
        {
            fd1.fontName = "Garamond";
            ef1 = new ElementFormat(fd);
            var str:String = "This is flash text";
            te = new TextElement(str, ef);
            tb.content = te;
            var tx1:TextLine = tb.createTextLine(null,600);
            addChild(tx1);

            fd2 = (fd1.locked) ? fd1.clone() : fd1;
            fd2.fontName = "Arial";
            ef2 = (ef1.locked) ? ef1.clone() : ef1;
            ef2.fontDescription = fd2;
            tb.content.elementFormat = ef2;
            var tx2:TextLine = tb.createTextLine(null,600);
            addChild(tx2);
        }
    }
}
```

Controle de texto

O FTE fornece um novo conjunto de controles de formatação de texto para manipular a justificação e o espaçamento de caracteres (kerning e rastreamento). Também existem propriedades para controlar a maneira como as linhas são quebradas e para definir paradas de tabulação dentro das linhas.

Justificação de texto

A justificação de texto faz com que todas as linhas em um parágrafo tenham o mesmo comprimento ajustando o espaçamento entre palavras e, algumas vezes, entre letras. O efeito é alinhar o texto nos dois lados, enquanto o espaçamento entre palavras e letras varia. As colunas de texto em jornais e revistas normalmente são justificadas.

A propriedade `lineJustfication` na classe SpaceJustifier permite controlar a justificação de linhas em um bloco de texto. A classe LineJustification define constantes que podem ser usadas para especificar uma opção de justificação: `ALL_BUT_LAST` justifica todas, menos a última linha de texto; `ALL_INCLUDING_LAST` justifica todo o texto, inclusive a última linha; `UNJUSTIFIED`, que é o padrão, deixa o texto sem justificação.

Para justificar texto, defina a propriedade `lineJustification` para uma ocorrência da classe SpaceJustifier e atribua essa ocorrência à propriedade `textJustifier` de uma ocorrência de TextBlock. O exemplo a seguir cria um parágrafo no qual todas as linhas de texto, menos a última, são justificadas.

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class JustifyExample extends Sprite
    {
        public function JustifyExample()
        {
            var str:String = "Lorem ipsum dolor sit amet, consectetur adipisicing elit, " +
            "sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut " +
            "enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut " +
            "aliquip ex ea commodo consequat.";

            var format:ElementFormat = new ElementFormat();
            var textElement:TextElement=new TextElement(str,format);
            var spaceJustifier:SpaceJustifier=new
SpaceJustifier("en",LineJustification.ALL_BUT_LAST);

            var textBlock:TextBlock = new TextBlock();
            textBlock.content=textElement;
            textBlock.textJustifier=spaceJustifier;
            createLines(textBlock);
        }

        private function createLines(textBlock:TextBlock):void {
            var yPos=20;
            var textLine:TextLine=textBlock.createTextLine(null,150);

            while (textLine) {
                addChild(textLine);
                textLine.x=15;
                yPos+=textLine.textHeight+2;
                textLine.y=yPos;
                textLine=textBlock.createTextLine(textLine,150);
            }
        }
    }
}
```

Para variar o espaçamento entre letras, bem como entre palavras, defina a propriedade `SpaceJustifier.letterspacing` como `true`. A ativação de letterspacing pode reduzir as ocorrências de lacunas desagradáveis entre palavras, o que pode, algumas vezes, ocorrer com a justificação simples.

Justificação de texto do Leste Asiático

A justificação de texto do Leste Asiático envolve considerações adicionais. Ele pode ser escrito de cima para baixo e alguns caracteres, conhecidos como kinsoku, não podem aparecer no início ou no fim de uma linha. A classe JustificationStyle define as seguintes constantes, que especificam as opções para manipulação desses caracteres. `PRIORITIZE_LEAST_ADJUSTMENT` baseia a justificação na expansão ou na compactação da linha, dependendo de qual delas produz o resultado mais desejado. `PUSH_IN_KINSOKU`baseia a justificação na compactação kinsoku ao final da linha ou na expansão caso não ocorra kinsoku ou o espaço seja insuficiente.

`PUSH_OUT_ONLY` baseia a justificação na expansão da linha. Para criar um bloco de texto asiático vertical, defina a propriedade `TextBlock.lineRotation` como `TextRotation.ROTATE_90` e a propriedade `ElementFormat.textRotation` como `TextRotation.AUTO`, que é a padrão. A configuração da propriedade `textRotation` como `AUTO` faz com que os glifos do texto permaneçam verticais em vez girar suas laterais quando a linha é girada. A configuração `AUTO` especifica uma rotação anti-horária de 90 graus apenas para glifos de largura completa e glifos largos, conforme determinado pelas propriedades Unicode do glifo. O exemplo a seguir exibe um bloco vertical de texto japonês e o justifica usando a opção `PUSH_IN_KINSOKU`. i

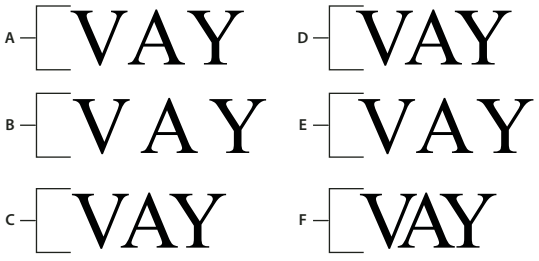
```
package
{
    import flash.text.engine.*;
    import flash.display.Stage;
    import flash.display.Sprite;
    import flash.system.Capabilities;

    public class EastAsianJustifyExample extends Sprite
    {
        public function EastAsianJustifyExample()
        {
            var Japanese_txt:String = String.fromCharCode(
            0x5185, 0x95A3, 0x5E9C, 0x304C, 0x300C, 0x653F, 0x5E9C, 0x30A4,
            0x30F3, 0x30BF, 0x30FC, 0x30CD, 0x30C3, 0x30C8, 0x30C6, 0x30EC,
            0x30D3, 0x300D, 0x306E, 0x52D5, 0x753B, 0x914D, 0x4FE1, 0x5411,
            0x3051, 0x306B, 0x30A2, 0x30C9, 0x30D3, 0x30B7, 0x30B9, 0x30C6,
            0x30E0, 0x30BA, 0x793E, 0x306E)
            var textBlock:TextBlock = new TextBlock();
            var font:FontDescription = new FontDescription();
            var format:ElementFormat = new ElementFormat();
            format.fontSize = 12;
            format.color = 0xCC0000;
            format.textRotation = TextRotation.AUTO;
            textBlock.baselineZero = TextBaseline.IDEOGRAPHIC_CENTER;
            var eastAsianJustifier:EastAsianJustifier = new EastAsianJustifier("ja",
LineJustification.ALL_BUT_LAST);
            eastAsianJustifier.justificationStyle = JustificationStyle.PUSH_IN_KINSOKU;
            textBlock.textJustifier = eastAsianJustifier;
            textBlock.lineRotation = TextRotation.ROTATE_90;
            var linePosition:Number = this.stage.stageWidth - 75;
            if (Capabilities.os.search("Mac OS") > -1)
                // set fontName: Kozuka Mincho Pro R
```

```
                font.fontName = String.fromCharCode(0x5C0F, 0x585A, 0x660E, 0x671D) + " Pro R";
            else
                font.fontName = "Kozuka Mincho Pro R";
            textBlock.content = new TextElement(Japanese_txt, format);
            var previousLine:TextLine = null;

            while (true)
            {
                var textLine:TextLine = textBlock.createTextLine(previousLine, 200);
                if (textLine == null)
                    break;
                textLine.y = 20;
                textLine.x = linePosition;
                linePosition -= 25;
                addChild(textLine);
                previousLine = textLine;
            }
        }
    }
}
```

Kerning e rastreamento

O kerning e o rastreamento afetam a distância entre pares adjacentes de caracteres em um bloco de texto. O kerning controla como pares de caracteres "se ajustam" em conjunto, como os pares "WA" ou "Va". O kerning é definido no objeto `ElementFormat`. Ele é ativado por padrão (`Kerning.ON`) e pode ser definido como OFF ou AUTO, nesse caso sendo aplicado somente entre caracteres se eles não forem caracteres Kanji, Hiragana ou Katakana.

O rastreamento adiciona ou subtrai um número definido de pixels entre todos os caracteres em um bloco de texto e também é definido no objeto `ElementFormat`. Ele funciona com fontes de dispositivo e incorporadas. O FTE oferece suporte a duas propriedades de rastreamento, `trackingLeft`, que adiciona/subtrai pixels do lado esquerdo de um caractere, e `trackingRight`, que adiciona/subtrai do lado direito. Se o kerning estiver sendo usado, o valor do rastreamento será adicionado ou subtraído dos valores de kerning para cada par de caracteres.



***A.** Kerning.OFF **B.** TrackingRight=5, Kerning.OFF **C.** TrackingRight=-5, Kerning.OFF **D.** Kerning.ON **E.** TrackingRight=-5, Kerning.ON **F.** TrackingRight=-5, Kerning.ON*

```
var ef1:ElementFormat = new ElementFormat();
ef1.kerning = Kerning.OFF;

var ef2:ElementFormat = new ElementFormat();
ef2.kerning = Kerning.ON;
ef2.trackingLeft = 0.8;
ef2.trackingRight = 0.8;

var ef3:ElementFormat = new ElementFormat();
ef3.trackingRight = -0.2;
```

Quebras de linha para quebra de texto

A propriedade `breakOpportunity` do objeto `ElementFormat` determina quais caracteres podem ser usados para quebra ocorre quebra de texto em várias linhas. O padrão, `BreakOpportunity.AUTO`, usa propriedades padrão do Unicode, como quebra entre palavras e em hífens. O uso de `BreakOpportunity.ALL` permite que qualquer caractere seja tratado com uma oportunidade de quebra de linha, o que é útil para criar efeitos como texto ao longo de um caminho.

```
var ef:ElementFormat = new ElementFormat();
ef.breakOpportunity = BreakOpportunity.ALL;
```

Paradas de tabulação

Para definir paradas de tabulação em um bloco de texto, defina as paradas de tabulação criando ocorrências da classe TabStop. Os parâmetros para o construtor `TabStop()` especificam como o texto se alinha com a parada de tabulação. Esses parâmetros especificam a posição da parada de tabulação e, para alinhamento decimal, o valor no qual alinhar, expresso como uma string. Normalmente, esse valor é um ponto decimal, mas também pode ser uma vírgula, um símbolo de dólar ou o símbolo do Iene ou do Euro, por exemplo. A linha de código a seguir cria uma parada de tabulação denominada tab1.

```
var tab1:TabStop = new TabStop(TabAlignment.DECIMAL, 50, ".");
```

Depois de criar as paradas de tabulação para um bloco de texto, atribua-as à propriedade `tabStops` da ocorrência de TextBlock. Mas como a propriedade `tabStops` requer um Vetor, primeiro crie um Vetor e adicione as paradas de tabulação a ele. O Vetor permite atribuir um conjunto de paradas de tabulação ao bloco de texto. O exemplo a seguir cria uma ocorrência de `Vector<TabStop>` e adiciona um conjunto de objetos TabStop a ele. Em seguida, ele atribui as paradas de tabulação à propriedade `tabStops` de uma ocorrência de TextBlock.

```
var tabStops:Vector.<TabStop> = new Vector.<TabStop>();
tabStops.push(tab1, tab2, tab3, tab4);
textBlock.tabStops = tabStops
```

Para obter mais informações sobre vetores, consulte “Trabalho com matrizes” na página 157.

O exemplo a seguir mostra o efeito de cada uma das opções de alinhamento de TabStop.

```
package {

    import flash.text.engine.*;
    import flash.display.Sprite;

    public class TabStopExample extends Sprite
    {
        public function TabStopExample()
        {
            var format:ElementFormat = new ElementFormat();
            format.fontDescription = new FontDdscription("Arial");
            format.fontSize = 16;

            var tabStops:Vector.<TabStop> = new Vector.<TabStop>();
            tabStops.push(
                new TabStop(TabAlignment.START, 20),
                new TabStop(TabAlignment.CENTER, 140),
                new TabStop(TabAlignment.DECIMAL, 260, "."),
                new TabStop(TabAlignment.END, 380));
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = new TextElement(
                "\tt1\tt2\tt3\tt4\n" +
                "\tThis line aligns on 1st tab\n" +
                "\t\t\t\tThis is the end\n" +
                "\tThe following fragment centers on the 2nd tab:\t\t\n" +
                "\t\tit's on me\t\t\n" +
                "\tThe following amounts align on the decimal point:\n" +
                "\t\t\t45.00\t\n" +
                "\t\t\t75,320.00\t\n" +
                "\t\t\t6,950.00\t\n" +
                "\t\t\t7.01\t\n", format);

            textBlock.tabStops = tabStops;
            var yPosition:Number = 60;
            var previousTextLine:TextLine = null;
            var textLine:TextLine;
            var i:int;
            for (i = 0; i < 10; i++) {
                textLine = textBlock.createTextLine(previousTextLine, 1000, 0);
                textLine.x = 20;
                textLine.y = yPosition;
                addChild(textLine);
                yPosition += 25;
                previousTextLine = textLine;
            }
        }
    }
}
```

Exemplo do FTE - Layout de jornal

Este exemplo de programação mostra o Mecanismo de texto do Flash sendo usado no layout de uma página de jornal simples. A página inclui um título grande, um subtítulo e uma seção body de várias colunas.

Primeiro, crie um novo arquivo FLA e acrescente o seguinte código ao quadro 2 da camada padrão:

```
import com.example.programmingas3.newslayout.StoryLayout ;
// frame sc ript - create  a 3-columned arti cle layout
var story:StoryLayout = new StoryLayout(720, 500, 3, 10);
story.x = 20;
story.y = 80;
addChild(story);
stop();
```

StoryLayout.as é o script controlador deste exemplo. Ele define o conteúdo, lê informações de estilo em uma folha de estilo externa e atribui esses estilos a objetos ElementFormat. Em seguida, ele cria o título, o subtítulo e elementos de texto com várias colunas.

```
package com.example.programmingas3.newslayout
{
    import flash.display.Sprite;
    import flash.text.StyleSheet;
    import flash.text.engine.*;

    import flash.events.Event;
    import flash.net.URLRequest;
    import flash.net.URLLoader;
    import flash.display.Sprite;
    import flash.display.Graphics;

    public class StoryLayout extends Sprite
    {
        public var headlineTxt:HeadlineTextField;
        public var subtitleTxt:HeadlineTextField;
        public var storyTxt:MultiColumnText;
        public var sheet:StyleSheet;
        public var h1_ElFormat:ElementFormat;
        public var h2_ElFormat:ElementFormat;
        public var p_ElFormat:ElementFormat;

        private var loader:URLLoader;

        public var paddingLeft:Number;
        public var paddingRight:Number;
        public var paddingTop:Number;
        public var paddingBottom:Number;

        public var preferredWidth:Number;
        public var preferredHeight:Number;

        public var numColumns:int;

        public var bgColor:Number = 0xFFFFFF;

        public var headline:String = "News Layout Example";
        public var subtitle:String = "This example formats text like a newspaper page using the
Flash Text Engine API. ";

        public var rawTestData:String =
        "From the part Mr. Burke took in the American Revolution, it was natural that I should
consider him a friend to mankind; and as our acquaintance commenced on that ground, it would
have been more agreeable to me to have had cause to continue in that opinion than to change it. " +
        "At the time Mr. Burke made his violent speech last winter in the English Parliament
```

```
against the French Revolution and the National Assembly, I was in Paris, and had written to him
but a short time before to inform him how prosperously matters were going on. Soon after this I
saw his advertisement of the Pamphlet he intended to publish: As the attack was to be made in a
language but little studied, and less understood in France, and as everything suffers by
translation, I promised some of the friends of the Revolution in that country that whenever Mr.
Burke's Pamphlet came forth, I would answer it. This appeared to me the more necessary to be
done, when I saw the flagrant misrepresentations which Mr. Burke's Pamphlet contains; and that
while it is an outrageous abuse on the French Revolution, and the principles of Liberty, it is
an imposition on the rest of the world. " +
        "I am the more astonished and disappointed at this conduct in Mr. Burke, as (from the
circumstances I am going to mention) I had formed other expectations. " +
        "I had seen enough of the miseries of war, to wish it might never more have existence
in the world, and that some other mode might be found out to settle the differences that should
occasionally arise in the neighbourhood of nations. This certainly might be done if Courts were
disposed to set honesty about it, or if countries were enlightened enough not to be made the
dupes of Courts. The people of America had been bred up in the same prejudices against France,
which at that time characterised the people of England; but experience and an acquaintance with
the French Nation have most effectually shown to the Americans the falsehood of those prejudices;
and I do not believe that a more cordial and confidential intercourse exists between any two
countries than between America and France. ";

        public function StoryLayout(w:int = 400, h:int = 200, cols:int = 3, padding:int =
10):void
        {
            this.preferredWidth = w;
            this.preferredHeight = h;

            this.numColumns = cols;

            this.paddingLeft = padding;
            this.paddingRight = padding;
            this.paddingTop = padding;
            this.paddingBottom = padding;

            var req:URLRequest = new URLRequest("story.css");
            loader = new URLLoader();
            loader.addEventListener(Event.COMPLETE, onCSSFileLoaded);
            loader.load(req);
        }

        public function onCSSFileLoaded(event:Event):void
        {
            this.sheet = new StyleSheet();
            this.sheet.parseCSS(loader.data);

            // convert headline styles to ElementFormat objects
            h1_ElFormat = getElFormat("h1", this.sheet);
            h1_ElFormat.typographicCase = TypographicCase.UPPERCASE;
            h2_ElFormat = getElFormat("h2", this.sheet);
            p_ElFormat = getElFormat("p", this.sheet);
            displayText();
        }

        public function drawBackground():void
        {
            var h:Number = this.storyTxt.y + this.storyTxt.height +
                           this.paddingTop + this.paddingBottom;
```

```
    var g:Graphics = this.graphics;
    g.beginFill(this.bgColor);
    g.drawRect(0, 0, this.width + this.paddingRight + this.paddingLeft, h);
    g.endFill();
}

/**
 * Reads a set of style properties for a named style and then creates
 * a TextFormat object that uses the same properties.
 */
public function getElFormat(styleName:String, ss:StyleSheet):ElementFormat
{
var style:Object = ss.getStyle(styleName);
if (style != null)
{
var colorStr:String = style.color;
if (colorStr != null && colorStr.indexOf("#") == 0)
{
    style.color = colorStr.substr(1);
}
        var fd:FontDescription = new FontDescription(
                            style.fontFamily,
                            style.fontWeight,
                            FontPosture.NORMAL,
                            FontLookup.DEVICE,
                            RenderingMode.NORMAL,
                            CFFHinting.NONE);
var format:ElementFormat = new ElementFormat(fd,
                        style.fontSize,
                        style.color,
                        1,
                        TextRotation.AUTO,
                        TextBaseline.ROMAN,
                        TextBaseline.USE_DOMINANT_BASELINE,
                        0.0,
                        Kerning.ON,
                        0.0,
                        0.0,
                        "en",
                                BreakOpportunity.AUTO,
                                DigitCase.DEFAULT,
                                DigitWidth.DEFAULT,
                        LigatureLevel.NONE,
                                TypographicCase.DEFAULT);

if (style.hasOwnProperty("letterSpacing"))
{
    format.trackingRight = style.letterSpacing;
}
}
return format;
}

public function displayText():void
{
headlineTxt = new HeadlineTextField(h1_ElFormat,headline,this.preferredWidth);
headlineTxt.x = this.paddingLeft;
```

```
        headlineTxt.y = 40 + this.paddingTop;
            headlineTxt.fitText(1);
            this.addChild(headlineTxt);

        subtitleTxt = new HeadlineTextField(h2_ElFormat,subtitle,this.preferredWidth);
        subtitleTxt.x = this.paddingLeft;
        subtitleTxt.y = headlineTxt.y + headlineTxt.height;
            subtitleTxt.fitText(2);
        this.addChild(subtitleTxt);

        storyTxt = new MultiColumnText(rawTestData, this.numColumns,
                        20, this.preferredWidth, this.preferredHeight, p_ElFormat);
        storyTxt.x = this.paddingLeft;
        storyTxt.y = subtitleTxt.y + subtitleTxt.height + 10;
        this.addChild(storyTxt);

            drawBackground();
        }
    }
}
```

FormattedTextBlock.as é usado como uma classe de base para a criação de blocos de texto. Ele também inclui funções de utilitário para alterar o tamanho e o uso de maiúsculas/minúsculas nas fontes.

```
package com.example.programmingas3.newslayout
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class FormattedTextBlock extends Sprite
    {
        public var tb:TextBlock;
        private var te:TextElement;
        private var ef1:ElementFormat;
        private var textWidth:int;
        public var totalTextLines:int;
        public var blockText:String;
        public var leading:Number = 1.25;
        public var preferredWidth:Number = 720;
        public var preferredHeight:Number = 100;

        public function FormattedTextBlock(ef:ElementFormat,txt:String, colW:int = 0)
        {
            this.textWidth = (colW==0) ? preferredWidth : colW;
            blockText = txt;
            ef1 = ef;
            tb = new TextBlock();
            tb.textJustifier = new SpaceJustifier("en",LineJustification.UNJUSTIFIED,false);
            te = new TextElement(blockText,this.ef1);
            tb.content = te;
            this.breakLines();
        }

        private function breakLines()
        {
            var textLine:TextLine = null;
            var y:Number = 0;
```

```
            var lineNum:int = 0;
            while (textLine = tb.createTextLine(textLine,this.textWidth,0,true))
            {
                textLine.x = 0;
                textLine.y = y;
                y += this.leading*textLine.height;
                this.addChild(textLine);
            }
            for (var i:int = 0; i < this.numChildren; i++)
            {
                TextLine(this.getChildAt(i)).validity = TextLineValidity.STATIC;
            }
            this.totalTextLines = this.numChildren;
        }

        private function rebreakLines()
        {
            this.clearLines();
            this.breakLines();
        }

        private function clearLines()
        {
            while(this.numChildren)
            {
                this.removeChildAt(0);
            }
        }

        public function changeSize(size:uint=12):void
        {
            if (size > 5)
            {
                var ef2:ElementFormat = ef1.clone();
                ef2.fontSize = size;
                te.elementFormat = ef2;
                this.rebreakLines();
            }
        }

        public function changeCase(newCase:String = "default"):void
        {
            var ef2:ElementFormat = ef1.clone();
            ef2.typographicCase = newCase;
            te.elementFormat = ef2;
        }
    }
}
```

HeadlineTextBlock.as estende a classe FormattedTextBlock e é usado para criar títulos. Ele inclui uma função que filtra o texto em um espaço da página definido.

```
package com.example.programmingas3.newslayout
{
    import flash.text.engine.*;
    public class HeadlineTextField extends FormattedTextBlock
    {

        public static var MIN_POINT_SIZE:uint = 6;
        public static var MAX_POINT_SIZE:uint = 128;

        public function HeadlineTextField(te:ElementFormat,txt:String,colW:int = 0)
        {
            super(te,txt);
        }

        public function fitText(maxLines:uint = 1, targetWidth:Number = -1):uint
        {
            if (targetWidth == -1)
            {
                targetWidth = this.width;
            }

            var pixelsPerChar:Number = targetWidth / this.blockText.length;
            var pointSize:Number = Math.min(MAX_POINT_SIZE,
                           Math.round(pixelsPerChar * 1.8 * maxLines));

            if (pointSize < 6)
            {
                // the point size is too small
                return pointSize;
            }

            this.changeSize(pointSize);
            if (this.totalTextLines > maxLines)
            {
                return shrinkText(--pointSize, maxLines);
            }
            else
            {
                return growText(pointSize, maxLines);
            }
        }

        public function growText(pointSize:Number, maxLines:uint = 1):Number
        {
            if (pointSize >= MAX_POINT_SIZE)
            {
                return pointSize;
            }

            this.changeSize(pointSize + 1);
            if (this.totalTextLines > maxLines)
            {
                // set it back to the last size
                this.changeSize(pointSize);
                return pointSize;
            }
            else
```

```
            {
                return growText(pointSize + 1, maxLines);
            }
        }

        public function shrinkText(pointSize:Number, maxLines:uint=1):Number
        {
            if (pointSize <= MIN_POINT_SIZE)
            {
                return pointSize;
            }
            this.changeSize(pointSize);

            if (this.totalTextLines > maxLines)
            {
                return shrinkText(pointSize - 1, maxLines);
            }
            else
            {
                return pointSize;
            }
        }
    }
}
```

MultiColumnText.as manipula a formatação do texto em um design de várias colunas. Ele demonstra o uso flexível de um objeto TextBlock como uma fábrica para criar, formatar e colocar linhas de texto.

```
package com.example.programmingas3.newslayout
{
    import flash.display.Sprite;
    import flash.text.engine.*;

    public class MultiColumnText extends Sprite
    {
        private var tb:TextBlock;
        private var te:TextElement;
        private var numColumns:uint = 2;
        private var gutter:uint = 10;
        private var leading:Number = 1.25;
        private var preferredWidth:Number = 400;
        private var preferredHeight:Number = 100;
        private var colWidth:int = 200;

        public function MultiColumnText(txt:String = "",cols:uint = 2,
            gutter:uint = 10, w:Number = 400, h:Number = 100,
            ef:ElementFormat = null):void
        {
            this.numColumns = Math.max(1, cols);
            this.gutter = Math.max(1, gutter);

            this.preferredWidth = w;
            this.preferredHeight = h;

            this.setColumnWidth();

            var field:FormattedTextBlock = new FormattedTextBlock(ef,txt,this.colWidth);
```

```
            var totLines:int = field.totalTextLines;
            field = null;
            var linesPerCol:int = Math.ceil(totLines/cols);

            tb = new TextBlock();
            te = new TextElement(txt,ef);
            tb.content = te;
            var textLine:TextLine = null;
            var x:Number = 0;
            var y:Number = 0;
            var i:int = 0;
            var j:int = 0;
            while (textLine = tb.createTextLine(textLine,this.colWidth,0,true))
            {
                textLine.x = Math.floor(i/(linesPerCol+1))*(this.colWidth+this.gutter);
                textLine.y = y;
                y += this.leading*textLine.height;
                j++;
                if(j>linesPerCol)
                {
                    y = 0;
                    j = 0;
                }
                i++;

                this.addChild(textLine);
            }
        }

        private function setColumnWidth():void
        {
        this.colWidth = Math.floor( (this.preferredWidth -
            ((this.numColumns - 1) * this.gutter)) / this.numColumns);
        }

    }
}
```

Capítulo 22: Trabalho com bitmaps

Além dos recursos de desenho de vetores, o ActionScript 3.0 permite criar imagens de bitmap ou manipular dados em pixels de imagens de bitmap externas carregadas em um arquivo SWF. Com a habilidade de acessar e alterar valores de pixel individuais, você pode criar seus próprios efeitos de imagem como filtro e usar as funções internas de ruído para criar textura e ruídos aleatórios. Todas essas técnicas são descritas neste capítulo.

Noções básicas do trabalho com bitmaps

Introdução ao trabalho com bitmaps

Quando você trabalha com imagens digitais, provavelmente encontra dois tipos principais de gráficos: bitmap e vetor. Os gráficos bitmap, também conhecidos como gráficos raster, são compostos por pequenos quadrados (pixels) organizados em um formato de grade retangular. Os gráficos de vetor são compostos por formas geométricas geradas matematicamente, como linhas, curvas e polígonos.

As imagens de bitmap são definidas pela largura e altura da imagem, medidas em pixels, e pelo número de bits contido em cada pixel, que representa o número de cores que um pixel pode conter. No caso das imagens de bitmap que utilizam o modelo de cor RGB, os pixels são compostos por três bytes: vermelho, verde e azul. Cada um desses bytes contém um valor que varia de 0 a 255. Quando os bytes são combinados em pixels, eles produzem uma cor semelhante à cor produzida quando um artista mistura cores de pintura. Por exemplo, um pixel que contém valores de byte de vermelho-255, verde-102 e azul-0 produz uma cor laranja vibrante.

A qualidade de uma imagem de bitmap é determinada pela combinação de resolução de imagem e o valor de bits da densidade de cor. *Resolução* relaciona-se ao número de pixels contido dentro de uma imagem. Quanto maior o número de pixels, mais alta é a resolução e melhor a imagem aparece. *Densidade de cor* relaciona-se à quantidade de informações que um pixel pode conter. Por exemplo, uma imagem que tem um valor de densidade de cor de 16 bits por pixel não pode representar o mesmo número de cores que uma imagem que tem uma densidade de cor de 48 bits. Conseqüentemente, a imagem de 48 bits terá graus de sombreamento mais suaves do que a contraparte de 16 bits.

Como os gráficos de bitmap dependem da resolução, eles não são dimensionados muito bem. Isso é mais facilmente notado quando imagens de bitmap são aumentadas em tamanho. Aumentar um bitmap normalmente resulta em perda de detalhes e qualidade.

Formatos de arquivo bitmap

As imagens bitmap são agrupadas em vários formatos de arquivo comuns. Esses formatos usam tipos diferentes de algoritmos de compactação para reduzir o tamanho do arquivo, bem como otimizar a qualidade da imagem com base no objeto final da imagem. Os formatos de imagem de bitmap suportados pelo Adobe Flash Player e pelo Adobe AIR são BMP, GIF, JPG, PNG e TIFF.

BMP

O formato BMP (bit mapped) é um formato de imagem padrão usado pelo sistema operacional Microsoft Windows. Por não utilizar algoritmos de compactação de nenhum tipo, o tamanho dos arquivos BMP é maior.

GIF

O GIF (Graphics Interchange Format) foi originalmente desenvolvido pela CompuServe em 1987 como uma forma de transmitir imagem com 256 cores (cor de 8 bits) O formato fornece tamanhos pequenos de arquivo e é ideal para imagens com base na Web. Devido a esse palete de cores limitado do formato, as imagens GIF não são normalmente para fotografias, que geralmente exigem mais graus de sombreamento e gradientes de cor. As imagens GIF permitem transparência de bit único, possibilitando que as cores sejam mapeadas como claras (ou transparentes). Isso resulta na cor de plano de fundo de uma página da Web mostrada através da imagem onde a transparência foi mapeada.

JPEG

Desenvolvido pelo Joint Photographic Experts Group (JPEG), o formato de imagem JPEG (também escrito como JPG) usa um algoritmo de compactação com perdas para permitir densidade de cores de 24 bits com tamanho de arquivo pequeno. Compactação com perdas significa que sempre que a imagem é salva, ela perde qualidade de dados, mas resulta em um tamanho de arquivo menor. O formato JPEG é ideal para fotografias porque ele é capaz de exibir milhões de cores. A habilidade de controlar o grau de compactação aplicada a uma imagem permite manipular a qualidade da imagem e o tamanho do arquivo.

PNG

O formato PNG (Portable Network Graphics) foi produzido como uma alternativa de fonte aberta para o formato de arquivo GIF patenteado. O formato PNG suporta densidade de cores de até 64 bits, permitindo até 16 milhões de cores. Como o PNG é relativamente um novo formato, alguns navegadores antigos não suportam arquivos PNG. Diferentemente dos JPGs, os PNGs usam compactação sem perda, o que significa que nenhum dado da imagem é perdido quando a imagem é salva. Os arquivos PNG também suportam transparência alfa, que permite até 256 níveis de transparência.

TIFF

O TIFF (Tagged Image File Format) foi o formato compatível com várias plataformas escolhido antes do surgimento do PNG. A desvantagem desse formato é que, devido à grande variedade de TIFFs, não há um único leitor que possa manipular cada versão. Além disso, não há navegadores da Web que suportam esse formato atualmente. Esse formato usa compactação com ou sem perdas, e pode manipular espaços de cor específicos do dispositivo (como CMYK).

Bitmaps transparentes e opacos

As imagens de bitmap que usam os formatos GIF ou PNG podem ter um byte extra (canal alfa) adicionado a cada pixel. O byte de pixel extra representa o valor da transparência do pixel.

As images GIF permitem transparência de único bit, o que significa que você pode especificar uma única cor, de um palete de 256 cores, para ser transparente. As imagens PNG, por outro lado, podem ter até 256 níveis de transparência. Essa função traz benefícios especialmente quando imagens ou texto devem ficar mesclados nos planos de fundo.

O ActionScript 3.0 replica esse byte de pixel de transparência extra dentro da classe BitmapData. Semelhante ao modelo de transparência do PNG, a constante `BitmapDataChannel.ALPHA` oferece até 256 níveis de transparência.

Tarefas comuns do trabalho com bitmaps

A lista a seguir descreve várias tarefas que você pode executar quando estiver trabalhando com imagens bitmap no ActionScript:

- Exibição de bitmaps na tela
- Recuperação e configuração de valores de cor de pixel

- Cópia de dados de bitmap:
  - Criação de uma cópia exata de um bitmap
  - Cópia de dados de um canal de cor de um bitmap para um canal de cor de outro bitmap
  - Cópia de um snapshot de um objeto de exibição de tela para um bitmap
- Criação de ruído e texturas nas imagens de bitmap
- Rolagem de bitmaps

Conceitos e termos importantes

A lista a seguir contém termos importantes usados neste capítulo:

- Alfa: O nível de transparência (ou mais precisamente, de opacidade) em uma cor ou uma imagem. O valor de alfa é freqüentemente descrito como o valor do *canal alfa*.
- Cor ARGB: Um esquema de cores em que cada cor de pixel é uma mistura de valores das cores vermelho, verde e azul, e sua transparência é especificada como um valor alfa.
- Canal de cores: Normalmente, as cores são representadas como uma mistura de algumas cores básicas - geralmente (para gráficos de computador) vermelho, verde e azul. Cada cor básica é considerada um canal de cor; a quantidade de cor em cada canal de cor, misturadas juntas, determina a cor final.
- Densidade de cores: Também conhecido como *densidade de bits*, isso se refere à quantidade de memória do computador que está atribuída a cada pixel, que por sua vez determina o número de cores possíveis que podem ser representadas na imagem.
- Pixel: A menor unidade de informação em uma imagem de bitmap - essencialmente um ponto de cor.
- Resolução: As dimensões em pixel de uma imagem, que determinam o nível de detalhes granulados contidos na imagem. A resolução é freqüentemente expressa em termos de largura e altura em número de pixels.
- Cor RGB: Um esquema de cores em que cada cor de pixel é representada como uma mistura de valores das cores vermelho, verde e azul.

Teste dos exemplos do capítulo

Talvez você queira testar o código de exemplo durante a leitura deste capítulo. Como esse capítulo lida com a criação e a manipulação de conteúdo visual, o teste de código envolve a execução de código e a visualização de resultados no SWF que é criado.

Para testar os exemplos de código deste capítulo:

1. Criar um documento vazio usando a ferramenta de autoria do Flash
2. Selecione um quadro-chave na Linha de tempo.
3. Abra o painel Ações e copie o código no painel Script.
4. Execute o programa usando Controlar > Testar filme.

   Você verá os resultados do código no arquivo SWF criado.

Quase todos os exemplos incluem um código que cria uma imagem de bitmap de modo que você pode testar apenas o código sem precisar fornecer nenhum conteúdo de bitmap. Como alternativa, se você quiser testar o código na sua própria imagem, poderá importar aquela imagem no Adobe Flash CS4 Professional ou carregar a imagem externa no SWF de teste e usar os dados de bitmap com o código de exemplo. Para obter mais informações sobre como carregar imagens externas, consulte “Carregamento dinâmico do conteúdo da exibição” na página 313.

Classes Bitmap e BitmapData

As classes principais do ActionScript 3.0 para trabalhar com imagens de bitmap são a classe Bitmap, que é usada para exibir imagens bitmap na tela e a classe BitmapData, que é usada para acessar e manipular os dados de imagem não processados de um bitmap.

Compreensão da classe Bitmap

Como uma subclasse da classe DisplayObject, a classe Bitmap é a principal classe do ActionScript 3.0 usada para exibir imagens de bitmap. É possível que essas imagens tenham sido carregadas no Flash Player ou no Adobe AIR por meio da classe flash.display.Loader ou criadas dinamicamente usando o construtor `Bitmap()`. Durante o carregamento de uma imagem a partir de uma fonte externa, um objeto Bitmap pode usar apenas imagens nos formatos GIF, JPEG ou PNG. Depois de instanciado, a ocorrência de Bitmap pode ser considerada um wrapper para um objeto BitmapData que precisa ser renderizado no Palco. Como uma ocorrência de bitmap é um objeto de exibição, todos os recursos e funcionalidade dos objetos de exibição também podem ser usados para manipular uma ocorrência de Bitmap. Para obter mais informações sobre como trabalhar com objetos de exibição, consulte "Programação de exibição" na página 274.

Encaixe e suavização de pixels

Além da funcionalidade comum para todos os objetos de exibição, a classe Bitmap fornece alguns recursos adicionais que são específicos para imagens bitmaps.

Semelhante ao recurso encaixe de pixel encontrada na ferramenta de autoria do Flash, a propriedade `pixelSnapping` da classe Bitmap determina se um objeto Bitmap é ou não encaixado no seu pixel mais próximo. Essa propriedade aceita uma das três constantes definidas na classe PixelSnapping: `ALWAYS`, `AUTO` e `NEVER`.

A sintaxe para aplicação de encaixe de pixel como se segue:

```
myBitmap.pixelSnapping = PixelSnapping.ALWAYS;
```

Freqüentemente, quando imagens de bitmap são escalada, elas ficam borradas e distorcidas. Para ajudar a reduzir essa distorção, use a propriedade `smoothing` da classe BitmapData. Essa propriedade Boolean, quando definida como `true`, suaviza os pixels dentro da imagem quando ela é escalada. Isso dá à imagem uma aparência mais clara e natural.

Compreensão da classe BitmapData

A classe BitmapData, que está no pacote flash.display, pode ser comparada a um snapshot fotográfico de pixels contido em um imagem de bitmap carregada ou criada dinamicamente. Esse snapshot é representado por uma matriz de dados de pixel dentro do objeto. A classe BitmapData também contém uma série de métodos incorporados que são utilizados para a criação e a manipulação de dados de pixel.

Para instanciar um objeto BitmapData, use o seguinte código:

```
var myBitmap:BitmapData = new BitmapData(width:Number, height:Number, transparent:Boolean,
fillColor:uinit);
```

Os parâmetros `width` e `height` especificam o tamanho do bitmap; o valor máximo de ambos é 2880 pixels. O parâmetro `transparent` especifica se os dados de bitmap incluem um canal alfa (`true`) ou não (`false`). O parâmetro `fillColor` é um valor de cor de 32 bits que especifica a cor de plano de fundo, bem como o valor de transparência (se ele tiver sido definido como `true`). O exemplo a seguir cria um objeto BitmapData com plano de fundo laranja que é 50% transparente.

```
var myBitmap:BitmapData = new BitmapData(150, 150, true, 0x80FF3300);
```

Para renderizar um objeto BitmapData criado recentemente na tela, atribua-o ou envolva-o em uma ocorrência de Bitmap. Para fazer isso, você pode transmitir o objeto BitmapData como um parâmetro do construtor de objeto Bitmap ou atribuí-lo à propriedade `bitmapData` de uma ocorrência de Bitmap existente. Você também deve adicionar a ocorrência de Bitmap à lista de exibição chamando os métodos `addChild()` ou `addChildAt()` do contêiner de objeto de exibição que conterá a ocorrência de Bitmap. Para obter mais informações sobre como trabalhar com a lista de exibição, consulte "Adição de objetos à lista de exibição" na página 282.

O exemplo a seguir cria um objeto BitmapData com preenchimento vermelho e exibe-o em uma ocorrência de Bitmap.

```
var myBitmapDataObject:BitmapData = new BitmapData(150, 150, false, 0xFF0000);
var myImage:Bitmap = new Bitmap(myBitmapDataObject);
addChild(myImage);
```

Manipulação de pixels

A classe BitmapData também contém um conjunto de métodos que permitem manipular valores de dados de pixels.

Manipulação de pixels individuais

Ao alterar a aparência de uma imagem de bitmap no nível de pixel, você primeiro precisa obter os valores de cores dos pixels contidos dentro da área que deseja manipular. Use o método `getPixel()` para ler esses valores de pixel.

O método `getPixel()` recupera um valor RGB a partir de um conjunto de coordenadas x, y (pixel) transmitidas como parâmetros. Se qualquer um dos pixels que você deseja manipular incluir informações sobre transparência (canal alfa), será necessário usar o método `getPixel32()`. Esse método também recupera um valor RGB, mas diferente de `getPixel()`, o valor retornado por `getPixel32()` contém dados adicionais que representam o valor do canal alfa (transparência) do pixel selecionado.

Como alternativa, se você quiser simplesmente alterar a cor ou a transparência de um pixel contido em um bitmap, poderá usar o método `setPixel()` ou `setPixel32()`. Para definir uma cor de pixels, transmita simplesmente as coordenadas x, y e o valor da cor para um desses métodos.

O exemplo a seguir usa o método `setPixel()` para desenhar uma cruz em um plano de fundo BitmapData verde: Ele então usa `getPixel()` para recuperar o valor da cor do pixel nas coordenadas 50, 50 e liga o valor retornado.

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmapData:BitmapData = new BitmapData(100, 100, false, 0x009900);

for (var i:uint = 0; i < 100; i++)
{
    var red:uint = 0xFF0000;
    myBitmapData.setPixel(50, i, red);
    myBitmapData.setPixel(i, 50, red);
}

var myBitmapImage:Bitmap = new Bitmap(myBitmapData);
addChild(myBitmapImage);

var pixelValue:uint = myBitmapData.getPixel(50, 50);
trace(pixelValue.toString(16));
```

Se você quiser ler o valor de um grupo de pixels, use o método `getPixels()`. Esse método gera uma matriz de bytes a partir de uma região retangular dos dados de pixels que são transmitidos como um parâmetro. Cada um dos elementos da matriz de bytes (em outra palavras, os valores de pixel) são valores de pixel inteiros não assinados de 32 bits e não multiplicados.

Inversamente, para alterar (ou definir) o valor de um grupo de pixels, use o método `setPixels()`. Esse método espera dois parâmetros (`rect` e `inputByteArray`), que são combinados para resultar em uma região retangular (`rect`) dos dados de pixel (`inputByteArray`).

Como os dados são lidos (e gravados) fora de `inputByteArray`, o método `ByteArray.readUnsignedInt()` é chamado para cada um dos pixels na matriz. Se, por alguma razão, `inputByteArray` não contiver pelo menos um retangular inteiro de dados de pixel, o método interromperá o processamento dos dados da imagem naquele ponto.

É importante lembrar que, para obter e configurar dados de pixel, a matriz de byte espera valores de pixel vermelho, verde, azul e alfa de 32 bits (ARGB).

O exemplo a seguir usa os métodos `getPixels()` e `setPixels()` para copiar um grupo de pixels de um objeto BitmapData para outro:

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.utils.ByteArray;
import flash.geom.Rectangle;

var bitmapDataObject1:BitmapData = new BitmapData(100, 100, false, 0x006666FF);
var bitmapDataObject2:BitmapData = new BitmapData(100, 100, false, 0x00FF0000);

var rect:Rectangle = new Rectangle(0, 0, 100, 100);
var bytes:ByteArray = bitmapDataObject1.getPixels(rect);

bytes.position = 0;
bitmapDataObject2.setPixels(rect, bytes);

var bitmapImage1:Bitmap = new Bitmap(bitmapDataObject1);
addChild(bitmapImage1);
var bitmapImage2:Bitmap = new Bitmap(bitmapDataObject2);
addChild(bitmapImage2);
bitmapImage2.x = 110;
```

Detecção de colisão de nível de pixel

O método `BitmapData.hitTest()` executa a detecção de colisão no nível de pixel entre os dados de bitmap e outro objeto ou ponto.

O método `BitmapData.hitTest()` aceita cinco parâmetros:

- `firstPoint` (Point): Esse parâmetro se refere à posição do pixel do canto superior esquerdo do primeiro BitmapData no qual o teste de ocorrência está sendo executado.
- `firstAlphaThreshold` (uint): Esse parâmetro especifica o valor de canal alfa mais alto que é considerado opaco para esse teste de ocorrência.
- `secondObject` (Object): Esse parâmetro representa a área de impacto. O objeto `secondObject` não pode ser um objeto Rectangle, Point, Bitmap ou BitmapData. Esse objeto representa a área de ocorrência na qual a detecção de colisão está sendo executada.

- `secondBitmapDataPoint` (Point): Esse parâmetro opcional é usado para definir uma localização de pixel no segundo objeto BitmapData. Use esse parâmetro apenas quando o valor de `secondObject` for um objeto BitmapData. O padrão é `null`.
- `secondAlphaThreshold` (uint): Esse parâmetro opcional representa o valor de canal alfa mais alto que é considerado opaco no segundo objeto BitmapData. O valor padrão é 1. Use esse parâmetro apenas quando o valor de `secondObject` for um objeto BitmapData e quando ambos os objetos BitmapData forem transparentes.

Ao executar a detecção de colisão em imagens opacas, lembre-se de que o ActionScript trata a imagem como se ela fosse um retângulo totalmente opaco (ou caixa delimitadora). Como alternativa, ao executar teste de ocorrência em nível de pixel em imagens transparentes, ambas as imagens devem ser transparentes. Além disso, o ActionScript usa os parâmetros de limitação de alfa para determinar em qual ponto os pixels são alterados de transparente para opaco.

O exemplo a seguir cria três imagens de bitmap e verifica se há colisão de pixel utilizando dois diferentes pontos de colisão (um retorna false e outro true):

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.geom.Point;

var bmd1:BitmapData = new BitmapData(100, 100, false, 0x000000FF);
var bmd2:BitmapData = new BitmapData(20, 20, false, 0x00FF3300);

var bm1:Bitmap = new Bitmap(bmd1);
this.addChild(bm1);

// Create a red square.
var redSquare1:Bitmap = new Bitmap(bmd2);
this.addChild(redSquare1);
redSquare1.x = 0;

// Create a second red square.
var redSquare2:Bitmap = new Bitmap(bmd2);
this.addChild(redSquare2);
redSquare2.x = 150;
redSquare2.y = 150;

// Define the point at the top-left corner of the bitmap.
var pt1:Point = new Point(0, 0);
// Define the point at the center of redSquare1.
var pt2:Point = new Point(20, 20);
// Define the point at the center of redSquare2.
var pt3:Point = new Point(160, 160);

trace(bmd1.hitTest(pt1, 0xFF, pt2)); // true
trace(bmd1.hitTest(pt1, 0xFF, pt3)); // false
```

Cópia de dados de bitmap

Para copiar dados de bitmap de uma imagem para outra, você pode usar vários métodos: `clone()`, `copyPixels()`, `copyChannel()` e `draw()`.

Como o nome sugere, o método `clone()` permite clonar, ou obter uma amostra, dos dados de bitmap de um objeto BitmapData para outro. Quando chamado, o método retorna um novo objeto BitmapData que é um clone exato da ocorrência original do qual foi copiado.

O exemplo a seguir clona uma cópia de um quadrado (pai) laranja e coloca o clone ao lado do quadrado pai original:

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myParentSquareBitmap:BitmapData = new BitmapData(100, 100, false, 0x00ff3300);
var myClonedChild:BitmapData = myParentSquareBitmap.clone();

var myParentSquareContainer:Bitmap = new Bitmap(myParentSquareBitmap);
this.addChild(myParentSquareContainer);

var myClonedChildContainer:Bitmap = new Bitmap(myClonedChild);
this.addChild(myClonedChildContainer);
myClonedChildContainer.x = 110;
```

O método `copyPixels()` é um modo fácil e rápido de copiar pixels de um objeto BitmapData para outro. O método obtém um snapshot retangular (definido pelo parâmetro `sourceRect`) da imagem de origem e a copia para outra área retangular (de tamanho igual). O local do retângulo "colado" recentemente é definido dentro do parâmetro `destPoint`.

O método `copyChannel()` fornece amostras de um valor de canal de cor predefinido (alfa, vermelho, verde ou azul) de um objeto BitmapData de origem e o copia para um canal de um objeto BitmapData de destino. Chamar esse método não afeta os outros canais no objeto BitmapData de destino.

O método `draw()` desenha, ou processa, o conteúdo gráfico de um sprite de origem, clipe de filme ou outro objeto de exibição em um novo bitmap. Usando `matrix`, `colorTransform`, `blendMode` e os parâmetros de destino`clipRect`, você pode modificar a forma na qual o novo bitmap é renderizado. Esse método usa o processador de vetor no Flash Player e no AIR para gerar os dados.

Quando você chama `draw()`, transmite o objeto de origem (sprite, clipe de filme ou outro objeto de exibição) como o primeiro parâmetro, como demonstrado aqui:

```
myBitmap.draw(movieClip);
```

Se o objeto de origem teve qualquer transformação (cor, matriz etc.) aplicada a ele depois que ele foi originalmente carregado, essas transformações não serão copiadas para o novo objeto. Se você quiser copiar as transformações para o novo bitmap, será necessário copiar o valor da propriedade `transform` do objeto original para a propriedade `transform` do objeto Bitmap que usa o novo objeto BitmapData.

Texturas com funções de ruído

Para modificar a aparência de um bitmap, você pode aplicar um efeito de ruído nele, usando o método `noise()` ou os métodos `perlinNoise()`. Um efeito de ruído pode ser comparado a um estático que aparece em uma tela de televisão não sintonizada.

Para aplicar um efeito de ruído a um bitmap, use o método `noise()`. Esse método aplica um valor de cor aleatório para pixels dentro de uma área especificada de uma imagem de bitmap.

Esse método aceita cinco parâmetros:

- `randomSeed` (int): O número base aleatório que determina o padrão. Apesar do nome, esse número cria realmente os mesmos resultados se o mesmo número é transmitido. Para obter um resultado aleatório verdadeiro, use o método `Math.random()` para transmitir um número aleatório para esse parâmetro.
- `low` (uint): Esse parâmetro se refere ao valor mais baixo a ser gerado para cada pixel (0 a 255). O valor padrão é 0. A configuração desse valor mais baixo resulta em um padrão de ruído mais escuro, enquanto a configuração de um valor mais alto resulta em um padrão mais claro.
- `high` (uint): Esse parâmetro se refere ao valor mais alto a ser gerado para cada pixel (0 a 255). O valor padrão é 255. A configuração desse valor mais baixo resulta em um padrão de ruído mais escuro, enquanto a configuração de um valor mais alto resulta em um padrão mais claro.
- `channelOptions` (uint): Esse parâmetro especifica a qual canal de cores do objeto Bitmap o padrão de ruído será aplicado. O número pode ser uma combinação de qualquer um dos quatro valores ARGB do canal de cores. O valor padrão é 7.
- `grayScale` (Boolean): Quando definido para `true`, esse parâmetro aplica o valor `randomSeed` aos pixels de bitmap, eliminando efetivamente todas as cores da imagem. O canal alfa não é afetado por esse parâmetro. O valor padrão é `false`.

O exemplo a seguir cria uma imagem de bitmap e aplica um padrão de ruído azul a ela:

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmap:BitmapData = new BitmapData(250, 250,false, 0xff000000);
myBitmap.noise(500, 0, 255, BitmapDataChannel.BLUE,false);
var image:Bitmap = new Bitmap(myBitmap);
addChild(image);
```

Se você quiser criar uma textura com a aparência mais orgânica, use o método `perlinNoise()` . O método `perlinNoise()` produz texturas orgânicas realistas que são ideais para fumaça, nuvens, água, fogo ou até mesmo explosões.

Como isso é gerado por um algoritmo, o método `perlinNoise()` usa menos memória do que as texturas com base em bitmap. Entretanto, ele pode ainda ter um impacto no uso do processador, tornando o conteúdo desenvolvido no Flash mais lento e fazendo com que a tela seja redesenhada mais lentamente do que a taxa de quadro, especialmente em computadores mais antigos. Isso deve-se principalmente aos cálculos de ponto de flutuação que precisam ocorrer para processar os algoritmos de ruído perlin.

O método aceita nove parâmetros (os primeiro seis são obrigatório):

- `baseX` (Number): Determina o valor de x (tamanho) dos padrões criados.
- `baseY` (Number): Determina o valor de y (tamanho) dos padrões criados.
- `numOctaves` (uint): Número de oitavas ou funções de ruído individuais a serem combinadas para criar esse ruído. Números maiores de oitavas criam imagens com mais detalhes, mas também exigem mais tempo de processamento.
- `randomSeed` (int): O número base aleatório funciona exatamente da mesma forma que na função `noise()`. Para obter um resultado aleatório verdadeiro, use o método `Math.random()` para transmitir um número aleatório para esse parâmetro.
- `stitch` (Boolean): Se definido para `true`, esse método tentará suavizar as bordas de transição da imagem para criar texturas contínuas para colocação lado a lado como um preenchimento de bitmap.

- `fractalNoise` (Boolean): Esse parâmetro relaciona as bordas de gradientes sendo geradas pelo método. Se definido para `true`, o método gerará ruído fractal que suaviza as bordas do efeito. Se definido como `false`, ocorre turbulência. Uma imagem com turbulência tem descontinuidades visíveis no gradiente que podem fazer com que ela aproxime melhor os efeitos visuais mais nítidos, como chamas e ondas do mar.
- `channelOptions` (uint): O parâmetro `channelOptions` funciona exatamente da mesma forma que no método `noise()`. Ele especifica a qual canal de cores (do bitmap) o padrão de ruído será aplicado. O número pode ser uma combinação de qualquer um dos quatro valores ARGB do canal de cores. O valor padrão é 7.
- `grayScale` (Boolean): O parâmetro `grayScale` funciona exatamente da mesma forma que no método `noise()`. Quando definido como `true`, esse parâmetro aplica o valor `randomSeed` aos pixels de bitmap, eliminando todas as cores da imagem de modo eficaz. O valor padrão é `false`.
- `deslocamentos` (Matriz): Uma matriz de pontos que corresponde a deslocamentos x e y para cada oitava. Ao manipular os valores de deslocamento, você pode rolar suavemente as camadas de uma imagem. Cada ponto na matriz de deslocamento afeta uma função de ruído de oitava específica. O valor padrão é `null`.

O exemplo a seguir cria um objeto BitmapData de 150 x 150 pixels que chama o método `perlinNoise()` para gerar um efeito de nuvem em verde e azul:

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmapDataObject:BitmapData = new BitmapData(150, 150, false, 0x00FF0000);

var seed:Number = Math.floor(Math.random() * 100);
var channels:uint = BitmapDataChannel.GREEN | BitmapDataChannel.BLUE
myBitmapDataObject.perlinNoise(100, 80, 6, seed, false, true, channels, false, null);

var myBitmap:Bitmap = new Bitmap(myBitmapDataObject);
addChild(myBitmap);
```

Rolagem de bitmaps

Imagine que você criou um aplicativo de mapeamento de ruas em que sempre que o usuário move o mapa você precisa atualizar a visualização (mesmo que o mapa tenha sido movido apenas alguns pixels).

Uma forma de criar essa funcionalidade seria reprocessar uma nova imagem contendo a visualização atualizada do mapa sempre que o usuário move o mapa. Como alternativa, você pode criar uma única imagem grande e o método `scroll()`.

O método `scroll()` copia um bitmap na tela e o cola em um local de deslocamento — especificado por parâmetros (`x`, `y`). Se acontecer de uma parte do bitmap ficar fora do palco, o efeito resultante indicará que a imagem foi deslocada. Quando combinado com uma função de temporizador (ou um evento `enterFrame`), você poderá fazer com a imagem pareça ser animada ou estar em rolagem.

O exemplo a seguir pega o exemplo de ruído perlin anterior e gera uma imagem de bitmap maior (três-quartos dela são renderizados fora do palco). O método `scroll()` é então aplicado junto com um ouvinte do evento `enterFrame` que desloca a imagem por um pixel na direção diagonal para baixo. Esse método é chamado sempre que o quadro é inserido e, conseqüentemente, as partes da imagem fora da tela são renderizadas no Palco à medida que usamos a rolagem na imagem.

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmapDataObject:BitmapData = new BitmapData(1000, 1000, false, 0x00FF0000);
var seed:Number = Math.floor(Math.random() * 100);
var channels:uint = BitmapDataChannel.GREEN | BitmapDataChannel.BLUE;
myBitmapDataObject.perlinNoise(100, 80, 6, seed, false, true, channels, false, null);

var myBitmap:Bitmap = new Bitmap(myBitmapDataObject);
myBitmap.x = -750;
myBitmap.y = -750;
addChild(myBitmap);

addEventListener(Event.ENTER_FRAME, scrollBitmap);

function scrollBitmap(event:Event):void
{
    myBitmapDataObject.scroll(1, 1);
}
```

Benefícios do mapeamento mip

Mapas MIP (também conhecido como *mipmaps*), são bitmaps agrupados juntos e associados a uma textura para aumentar a qualidade e o desempenho da renderização de tempo de execução. O Flash Player 9.0.115.0, as versões posteriores e o AIR implementam essa tecnologia (o processo é chamado *mipmapping*), por meio da criação de versões otimizadas de escala variável de cada bitmap (começando em 50%).

O Flash Player e o AIR criam mapas MIP para bitmaps (arquivos JPEG, GIF ou PNG) que você exibe utilizando a classe Loader do ActionScript 3.0, um bitmap na biblioteca de ferramentas de autoria do Flash ou um objeto BitmapData. O Flash Player cria mapas MIP para os bitmaps que você exibe utilizando a função `loadMovie()` do ActionScript 2.0.

Os mapas MIP não são aplicados a objetos filtrados nem a clipes de filme em cache de bitmap. Entretanto, os mapas MIP são aplicados se você tiver transformações de bitmap em um objeto de exibição filtrado, mesmo se o bitmap estiver dentro de conteúdo mascarado.

Os mapeamentos mip do Flash Player e do AIR ocorrem automaticamente, mas você pode seguir algumas orientações para certificar-se de que suas imagens usam essa otimização:

- Para reprodução de vídeo, defina a propriedade `smoothing` como `true` para o objeto Video (consulte a classe Video).
- Para bitmaps, a propriedade `smoothing` não tem que ser definida como `true`, mas os aprimoramentos de qualidade são mais visíveis quando os bitmaps usam suavização.
- Use tamanhos de bitmap que são divisíveis por 4 ou 8 para imagens bidimensionais (como 640 x 128, que pode ser reduzida como se segue: 320 x 64 > 160 x 32 > 80 x 16 > 40 x 8 > 20 x 4 > 10 x 2 > 5 x 1) e 2^n para texturas tridimensionais. Os mapas MIP são gerados a partir de bitmaps que têm largura e altura que são 2^n (como 256 x 256, 512 x 512, 1024 x 1024). O mapeamento mip é interrompido quando o Flash Player ou o AIR encontra uma largura ou altura ímpar.

Exemplo: Lua giratória animada

O exemplo de lua giratória animada demonstra técnicas para trabalhar com objetos Bitmap e dados de imagem de bitmap (objetos BitmapData). O exemplo cria uma animação de uma lua giratória e esférica utilizando uma imagem plana da superfície da lua como os dados de imagem não processados. As técnicas a seguir são demonstradas:

- Carregamento de uma imagem externa e acesso aos seus dados de imagem não processados
- Criação de animação por meio de cópia repetida de pixels de diferentes partes de uma imagem de origem
- Criação de uma imagem de bitmap pela definição de valores de pixel

Para obter os arquivos de aplicativo desse exemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_br. Os arquivos de aplicativo da lua giratório animada podem ser encontrados na pasta Samples/SpinningMoon. O aplicativo consiste nos seguintes arquivos:

Arquivo	Descrição
SpinningMoon.mxml ou SpinningMoon.fla	O arquivo principal do aplicativo no Flex (MXML) ou Flash (FLA).
com/example/programmingas3/moon/MoonSphere.as	A classe que executa a funcionalidade de carregamento, exibição e animação da lua.
moonMap.png	O arquivo de imagem que contém uma fotografia da superfície da lua, que é carregado e utilizado para criar a lua giratório e animada.

Carregamento de uma imagem externa como dados de bitmap

A primeira tarefa principal que essa amostra executa é o carregamento de um arquivo de imagem externa, que é uma fotografia da superfície da lua. A operação de carregamento é manipulada por dois métodos na classe MoonSphere: o construtor `MoonSphere()`, em que o processo de carregamento é iniciado, e o método `imageLoadComplete()`, que é chamado quando a imagem externa está totalmente carregada.

O carregamento de uma imagem externa é semelhante ao carregamento de um SWF externo; ambos utilizam uma ocorrência da classe flash.display.Loader para executar essa operação. O código real no método `MoonSphere()` que inicia o carregamento de uma imagem é o seguinte:

```
var imageLoader:Loader = new Loader();
imageLoader.contentLoaderInfo.addEventListener(Event.COMPLETE, imageLoadComplete);
imageLoader.load(new URLRequest("moonMap.png"));
```

A primeira linha declara que a ocorrência de Loader é chamada `imageLoader`. A terceira linha inicia realmente o processo de carregamento chamando o método `load()` do objeto Loader, transmitindo uma ocorrência de URLRequest que representa a URL da imagem a ser carregada. A segunda linha configura o ouvinte do evento que será acionado quando a imagem for totalmente carregada. Observe que o método `addEventListener()` não é chamado na própria ocorrência de Loader; em vez disso, ele é chamado na propriedade `contentLoaderInfo` do objeto Loader. A própria ocorrência de Loader não despacha eventos relacionados ao conteúdo sendo carregado. A propriedade `contentLoaderInfo`, entretanto, contém uma referência ao objeto LoaderInfo que é associado ao conteúdo sendo carregado no objeto Loader (a imagem externa nesse caso). Aquele objeto LoaderInfo fornece vários eventos relacionados ao progresso e à conclusão do carregamento de conteúdo externo, incluindo o evento `complete` (`Event.COMPLETE`) que acionará uma chamada para o método `imageLoadComplete()` quando a imagem for totalmente carregada.

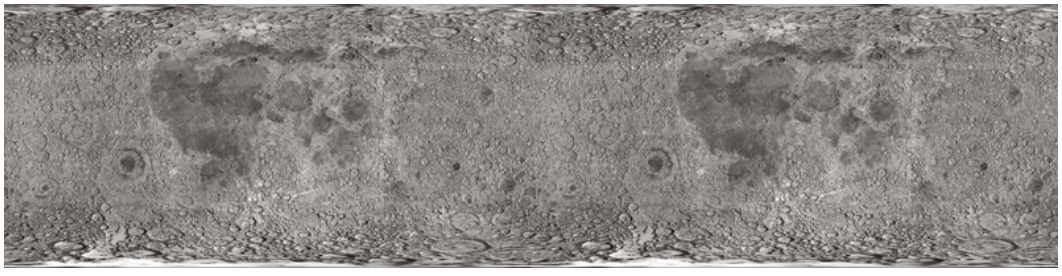
Embora o início do carregamento da imagem externa seja uma parte importante do processo, é igualmente importante saber o que fazer quando o carregamento for concluído. Como mostrado no código acima, a função `imageLoadComplete()` é chamada quando a imagem é carregada. Aquela função faz várias atividades com os dados de imagem carregados, descritas nas seções subseqüentes. Contudo, para usar os dados da imagem, é necessário acessar aqueles dados. Quando o objeto Loader for utilizado para carregar uma imagem externa, a imagem carregada torna-se uma ocorrência de bitmap que é anexada a um objeto de exibição filho do objeto Loader. Nesse caso, a ocorrência de Loader está disponível para o método do ouvinte do evento como parte do objeto de evento que é transmitido para o método como um parâmetro. As primeiras linhas do método `imageLoadComplete()` são como se segue:

```
private function imageLoadComplete(event:Event):void
{
    textureMap = event.target.content.bitmapData;
    ...
}
```

Observe que o parâmetro do objeto de evento é chamado `event`, e é uma ocorrência da classe Event. Cada ocorrência da classe Event tem uma propriedade `target`, que se refere ao objeto que está acionando o evento (nesse caso, a ocorrência de LoaderInfo na qual o método `addEventListener()` foi chamado, conforme descrito anteriormente). O objeto LoaderInfo, por sua vez, tem uma propriedade `content` que (uma vez concluído o processo de carregamento) contém a ocorrência de Bitmap com a imagem de bitmap carregada. Se você quiser exibir a imagem diretamente na tela, poderá anexar essa ocorrência de Bitmap (`event.target.content`) a um contêiner do objeto de exibição. (Você também pode anexar o objeto Loader a um contêiner do objeto de exibição). Entretanto, nessa amostra, o conteúdo carregado é utilizado como uma origem dos dados da imagem não processados que estão sendo exibidos na tela. Conseqüentemente, a primeira linha do método `imageLoadComplete()` lê a propriedade `bitmapData` da ocorrência de Bitmap carregada (`event.target.content.bitmapData`) e a armazena na variável de ocorrência chamada `textureMap`, que, como descrito na seção a seguir, é usada como uma origem de dados de imagem para criar a animação da lua giratória.

Criação da animação pela cópia de pixels

Uma definição básica da animação é a ilusão de movimentação, ou alteração, criada pela alteração de uma imagem ao longo do tempo. Nessa amostra, o objetivo é criar a ilusão de uma lua esférica girando ao redor de seu eixo vertical. Contudo, para o propósito de animação, você pode ignorar o aspecto de distorção esférica da amostra. Considere a imagem real que está carregada e é usada como a origem dos dados da imagem de lua:



Como você pode ver, a imagem não uma nem várias esferas; ela é uma fotografia retangular da superfície da lua. Como a foto foi tirada exatamente na linha do equador da lua, as partes da imagem próximas à parte superior e inferior da imagem são expandidas e distorcidas. Para remover a distorção da imagem e deixá-la com uma aparência esférica, utilizaremos um filtro de mapa de deslocamento, como descrito posteriormente. Entretanto, como essa imagem de origem é um retângulo, para criar a ilusão de que a esfera está girando, o código simplesmente precisa deslizar a foto da superfície da lua horizontalmente, como descrito nos parágrafos a seguir.

Observe que a imagem realmente contém duas cópias da fotografia da superfície da lua próximas uma da outra. Essa imagem é a imagem de origem da qual os dados de imagem foram copiados repetidas vezes para criar a aparência de movimento. Tendo duas cópias da imagem próximas uma da outra, um efeito de rolagem contínuo e ininterrupto pode ser criado com mais facilidade. Vamos avançar no processo de animação etapa por etapa para ver como ele funciona.

O processo realmente envolve dois objetos separados do ActionScript. Primeiro, existe a imagem de origem carregada, que no código é representada pela ocorrência de BitmapData chamada `textureMap`. Como descrito anteriormente, `textureMap` é preenchido com os dados de imagem assim que a imagem externa é carregada, utilizando este código:

```
textureMap = event.target.content.bitmapData;
```

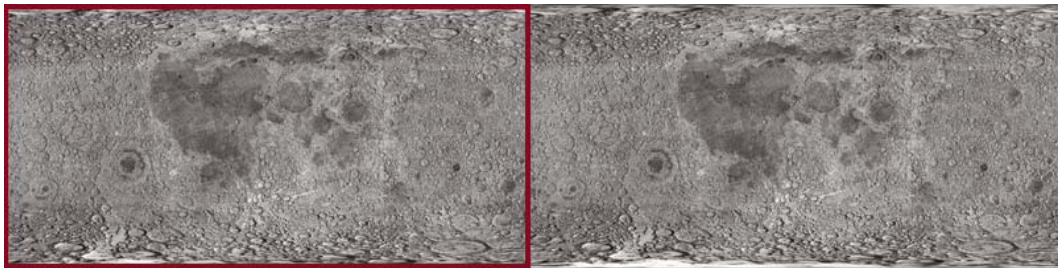
O conteúdo de `textureMap` é a imagem mostrada anteriormente. Além disso, para criar a rotação animada, a amostra usa uma ocorrência de Bitmap chamada `sphere`, que é o objeto de exibição real que mostra a imagem da lua na tela. Como `textureMap`, o objeto `sphere` é criado e preenchido com seus dados de imagem iniciais no método `imageLoadComplete()`, utilizando o seguinte código:

```
sphere = new Bitmap();
sphere.bitmapData = new BitmapData(textureMap.width / 2, textureMap.height);
sphere.bitmapData.copyPixels(textureMap,
                        new Rectangle(0, 0, sphere.width, sphere.height),
                        new Point(0, 0));
```

Como mostra o código, `sphere` é instanciado. Sua propriedade `bitmapData` (os dados de imagem não processados que são exibidos por `sphere`) é criada com a mesma altura e metade da largura de `textureMap`. Em outras palavras, o conteúdo de `sphere` será o tamanho de uma foto da lua (desde que a imagem `textureMap` contenha duas fotos da lua lado a lado). Em seguida, a propriedade `bitmapData` é preenchida com dados de imagem utilizando o método `copyPixels()`. Os parâmetros na chamada do método `copyPixels()` indicam várias coisas:

- O primeiro parâmetro indica que os dados da imagem são copiados de `textureMap`.
- O segundo parâmetro, uma nova ocorrência de Rectangle, especifica de qual parte de `textureMap` o snapshot da imagem deve ser tirado; nesse caso, o snapshot é um retângulo que começa no canto superior esquerdo de `textureMap` (indicado pelos dois primeiros parâmetros `Rectangle(): 0, 0`) e a largura e a altura do snapshot do retângulo corresponde às propriedades `width` e `height` de`sphere`.
- O terceiro parâmetro, uma nova ocorrência de Point com os valores x e y de `0`, define o destino dos dados de pixel — nesse caso, o canto superior esquerdo (0, 0) de `sphere.bitmapData`.

Representado visualmente, o código copia os pixels de `textureMap` contornados na imagem a seguir e os cola em `sphere`. Em outras palavras, o conteúdo de BitmapData de`sphere` é a parte de `textureMap` em destaque aqui:



Lembre-se, contudo, de que esse é apenas o estado inicial de `sphere` — o conteúdo da primeira imagem que é copiada na `sphere`.

Com a imagem de origem carregada e `sphere` criada, a tarefa final executada pelo método `imageLoadComplete()` é a configuração da animação. A animação é orientada pela ocorrência de Timer chamada `rotationTimer`, que é criada e iniciada pelo seguinte código:

```
var rotationTimer:Timer = new Timer(15);
rotationTimer.addEventListener(TimerEvent.TIMER, rotateMoon);
rotationTimer.start();
```

O código cria primeiro a ocorrência de Timer chamada `rotationTimer`; o parâmetro transmitido para o construtor `Timer()` indica que `rotationTimer` deve acionar seu evento `timer` a cada 15 milissegundos. Em seguida, o método `addEventListener()` é chamado, especificando que, quando o evento `timer` (`TimerEvent.TIMER`) ocorre, o método `rotateMoon()` é chamado. Por fim, o temporizado é realmente iniciado chamando seu método `start()`.

Devido ao modo com `rotationTimer` está definido, aproximadamente a cada 15 milissegundos, o Flash Player chama o método `rotateMoon()` na classe MoonSphere, que é onde a animação da lua acontece. O código-fonte do método rotateMoon() é o seguinte:

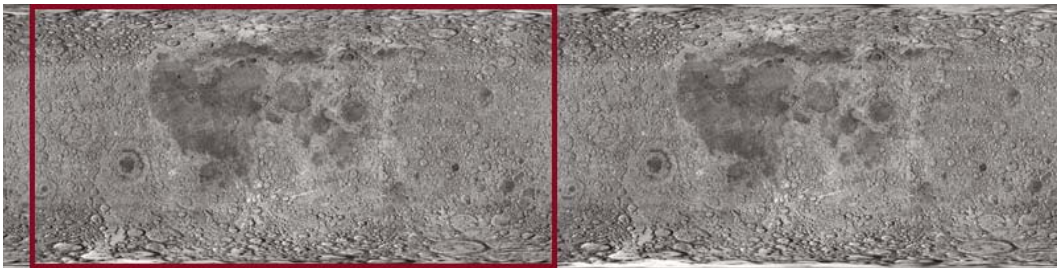
```
private function rotateMoon(event:TimerEvent):void
{
    sourceX += 1;
    if (sourceX > textureMap.width / 2)
    {
        sourceX = 0;
    }

    sphere.bitmapData.copyPixels(textureMap,
                                 new Rectangle(sourceX, 0, sphere.width, sphere.height),
                                 new Point(0, 0));

    event.updateAfterEvent();
}
```

O código faz três coisas:

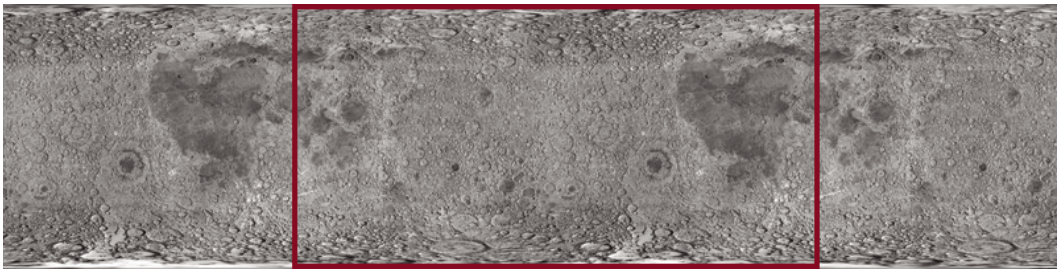
1 O valor da variável `sourceX` (inicialmente definido como 0) é incrementado em 1.

```
sourceX += 1;
```

Como você verá, `sourceX` é usado para determinar o local em `textureMap` do qual os pixels serão copiados para `sphere`; portanto esse código tem o efeito de mover o retângulo um pixel para a direita em `textureMap`. Voltando à representação visual, depois de vários ciclos de animação, o retângulo de origem terá se movido vários pixels para a direita, como se segue:

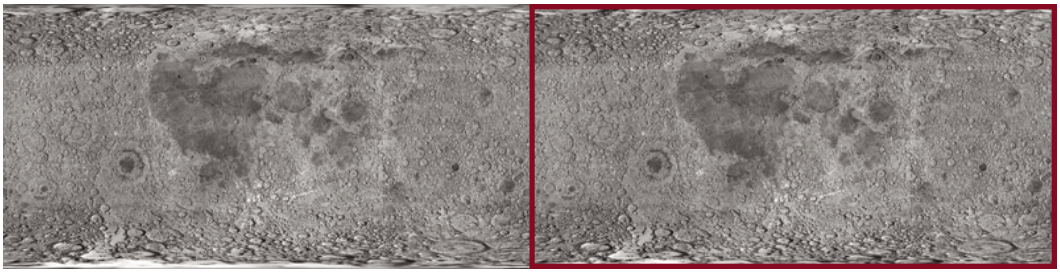


Depois de vários outros ciclos, o retângulo terá se movido para mais longe:



Esse deslocamento gradual, uniforme no local do qual os pixels são copiados é a chave da animação. Movendo lenta e continuamente o local de origem para a direita, a imagem que é exibida na tela em `sphere` parece deslizar continuamente para a esquerda. Essa é a razão pela qual a imagem de origem (`textureMap`) precisa ter duas cópias da foto da superfície da lua. Como o retângulo se move continuamente para a direita, a maioria das vezes ele não está sobre uma única foto da lua, mas sobrepõe as duas fotos da lua.

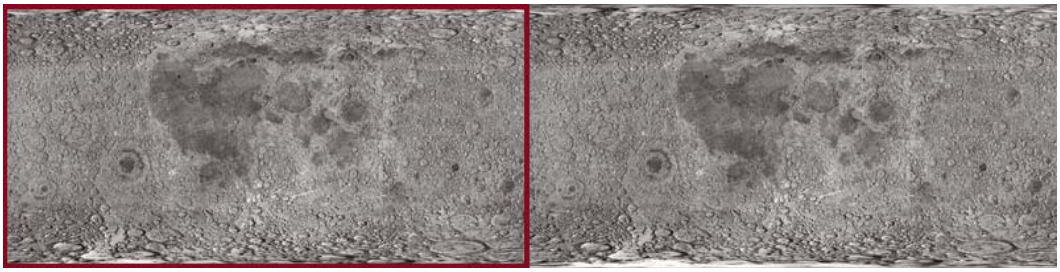
2 Com o retângulo de origem movendo-se lentamente para a direita, há um problema. Finalmente, o retângulo atingirá a borda direita de `textureMap` e ficará sem os pixels da foto da lua para copiar na `sphere`:



As linhas de código a seguir solucionam esse problema:

```
if (sourceX >= textureMap.width / 2)
{
    sourceX = 0;
}
```

O código verifica se `sourceX` (a borda esquerda do retângulo) atingiu o meio de `textureMap`. Em caso afirmativo, ele redefine `sourceX` para 0, movendo-o para a borda esquerda de `textureMap` e iniciando o ciclo novamente:



3 Com o valor `sourceX` apropriado calculado, a etapa final na criação da animação é copiar realmente os pixels do novo retângulo de origem para `sphere`. O código que faz isso é muito semelhante ao código que inicialmente preenche `sphere` (descrito anteriormente); a única diferença é que nesse caso, na chamada do construtor `new Rectangle()`, a borda esquerda do retângulo é colocada em `sourceX`:

```
sphere.bitmapData.copyPixels(textureMap,
                             new Rectangle(sourceX, 0, sphere.width, sphere.height),
                             new Point(0, 0));
```

Lembre-se de que esse código é chamada repetidamente a cada 15 milissegundos. Como o local do retângulo de origem é deslocado de forma contínua, e os pixels são copiados em `sphere`, a aparência na tela é a de que a imagem da foto da lua representada por `sphere` desliza continuamente. Em outras palavras, a lua parece girar continuamente.

Criação da aparência esférica

A lua, é claro, é uma esfera e não um retângulo. Conseqüentemente, a mostra precisa pegar a foto da superfície da lua retangular, à medida que é animada continuamente, e convertê-la em uma esfera. Isso envolve duas etapas separadas: uma máscara é usada para ocultar todo o conteúdo exceto uma região circular da foto da superfície da lua, e um filtro de mapa de deslocamento é usado para distorcer a aparência da foto da lua para que ela pareça tridimensional.

Primeiro, uma máscara no formato de círculo é usada para ocultar todo o conteúdo do objeto MoonSphere exceto a esfera criada pelo filtro. O código a seguir cria a máscara como uma ocorrência de Shape e a aplica como a máscara da ocorrência MoonSphere:

```
moonMask = new Shape();
moonMask.graphics.beginFill(0);
moonMask.graphics.drawCircle(0, 0, radius);
this.addChild(moonMask);
this.mask = moonMask;
```

Observe que, como MoonSphere é um objeto de exibição (ele é baseado na classe Sprite), a máscara pode ser aplicada diretamente à ocorrência de MoonSphere usando sua propriedade `mask` herdada.

Ocultar simplesmente partes da foto utilizando uma máscara em forma de círculo não é suficiente para criar um efeito de esfera giratória realista. Devido à forma como a foto da superfície da lua foi tirada, suas dimensões não são proporcionais; as partes da imagem que estão mais próximas à parte superior ou inferior da imagem são mais distorcidas e expandidas em comparação com as partes na linha do equador. Para distorcer a aparência da foto da lua e torná-la tridimensional, usaremos um filtro de mapa de deslocamento.

Um filtro de mapa de deslocamento é um tipo de filtro utilizado para distorcer uma imagem. Nesse caso, a foto da lua será "distorcida" para torná-la mais realista, comprimindo a parte superior e inferior da imagem horizontalmente, enquanto deixa o meio inalterado. Considerando que o filtro funciona em uma parte em formato de quadrado da foto, comprimir a parte superior e a inferior, mas não o meio, transformará o quadrado em um círculo. Um efeito colateral da animação dessa imagem distorcida é que o meio da imagem parece mover-se para mais longe em distância real de pixels do que as áreas próximas à parte superior e inferior, o que cria a ilusão de que o círculo é realmente um objeto tridimensional (uma esfera).

O código a seguir é usado para criar o filtro do mapa de deslocamento chamado `displaceFilter`:

```
var displaceFilter:DisplacementMapFilter;
displaceFilter = new DisplacementMapFilter(fisheyeLens,
                                new Point(radius, 0),
                                BitmapDataChannel.RED,
                                BitmapDataChannel.GREEN,
                                radius, 0);
```

Esse primeiro parâmetro, `fisheyeLens`, é conhecido como a imagem de mapa; nesse caso, um objeto BitmapData que é criado de modo programático. A criação daquela imagem é descrita abaixo na seção "Criação de uma imagem de bitmap pela definição de valores de pixel" na página 503. Os outros parâmetros descrevem a posição na imagem filtrada na qual o filtro deve ser aplicado, os canais de cor que serão utilizados para controlar o efeito do deslocamento e até qual extensão eles afetarão o deslocamento. Depois que o filtro de mapa de deslocamento é criado, ele é aplicado a `sphere`, ainda dentro do método `imageLoadComplete()`:

```
sphere.filters = [displaceFilter];
```

A imagem final, com a máscara e o filtro de mapa de deslocamento aplicados, se parece com:

Com cada ciclo de animação da lua giratória, o conteúdo de BitmapData da esfera é sobregravado por um novo snapshot dos dados da imagem de origem. Contudo, o filtro não precisa ser reaplicado sempre. Isso porque o filtro é aplicado à ocorrência de Bitmap (o objeto de exibição) em vez de aos dados do bitmap (as informações em pixels não processadas). Lembre-se, a ocorrência do Bitmap não são os dados reais de bitmap; ela é uma objeto de exibição que mostra os dados de bitmap na tela. Para usar uma analogia, uma ocorrência de Bitmap é como o projeto de slides que é usado para exibir slides fotográficos em uma tela; um objeto BitmapData é como o slide de fotografia real que pode ser apresentando por de um projetor de slide. Um filtro pode ser aplicado diretamente a um objeto BitmapData, que seria comparável a desenhar diretamente no slide fotográfico para alterar a imagem. Um filtro também pode ser

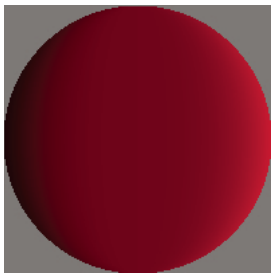
aplicado a qualquer objeto de exibição, inclusive uma ocorrência de Bitmap; isso seria como colocar um filtro na frente das lentes do projetor de slides para distorcer a saída mostrada na tela (sem alterar o slide original). Como os dados de bitmap não processados estão acessíveis por meio da propriedade bitmapData da ocorrência de Bitmap, o filtro pode ter sido aplicado diretamente nos dados de bitmap não processados. Entretanto, nesse caso, faz sentido aplicar o filtro ao objeto de exibição Bitmap em vez dos dados de bitmap.

Para obter informações detalhadas sobre o uso de filtro de mapas de deslocamento no ActionScript, consulte "Filtro de objetos de exibição" na página 354.

Criação de uma imagem de bitmap pela definição de valores de pixel

Um importante aspecto de um filtro de mapa de deslocamento é que ele realmente envolve duas imagens. Uma imagem, a imagem de origem, é a imagem que será realmente alterada pelo filtro. Nessa amostra, a imagem de origem é a ocorrência de Bitmap chamada `sphere`. A outra imagem usada pelo filtro é conhecida como a imagem de mapa. A imagem do mapa não é realmente exibida na ela. Em vez disso, a cor de cada um de seus pixels é usada como uma entrada para a função de deslocamento — a cor do pixel em uma determinada coordenada x, y na imagem de mapa determina quanto de deslocamento (deslocamento físico na posição) é aplicado ao pixel naquelas coordenadas x, y na imagem de origem.

Conseqüentemente, para usar o filtro de mapa de deslocamento para criar um efeito de esfera, a amostra precisa de imagem de mapa apropriada — aquela que tem um plano de fundo cinza e um círculo que é preenchido com um gradiente de uma única cor (vermelho) indo horizontalmente de escuro para claro, como mostrado aqui:



Como somente uma imagem e um filtro de mapa são utilizados nesta amostra, a imagem do mapa é criada apenas uma vez, no método `imageLoadComplete()` (em outras palavras, quando a imagem externa acabar de ser carregada. A imagem do mapa, chamada `fisheyeLens`, é criada chamando o método `createFisheyeMap()` da classe MoonSphere:

```
var fisheyeLens:BitmapData = createFisheyeMap(radius);
```

Dentro do método `createFisheyeMap()`, a imagem de mapa é desenhada um pixel por vez utilizando o método `setPixel()` da classe BitmapData. O código completo para o método `createFisheyeMap()` é listado aqui, seguido por uma discussão etapa por etapa de como ele funciona:

```
private function createFisheyeMap(radius:int):BitmapData
{
    var diameter:int = 2 * radius;

    var result:BitmapData = new BitmapData(diameter,
                                           diameter,
                                           false,
                                           0x808080);

    // Loop through the pixels in the image one by one
    for (var i:int = 0; i < diameter; i++)
    {
        for (var j:int = 0; j < diameter; j++)
        {
            // Calculate the x and y distances of this pixel from
            // the center of the circle (as a percentage of the radius).
            var pctX:Number = (i - radius) / radius;
            var pctY:Number = (j - radius) / radius;

            // Calculate the linear distance of this pixel from
            // the center of the circle (as a percentage of the radius).
            var pctDistance:Number = Math.sqrt(pctX * pctX + pctY * pctY);

            // If the current pixel is inside the circle,
            // set its color.
            if (pctDistance < 1)
            {
                // Calculate the appropriate color depending on the
                // distance of this pixel from the center of the circle.
                var red:int;
                var green:int;
                var blue:int;
                var rgb:uint;
                red = 128 * (1 + 0.75 * pctX * pctX * pctX / (1 - pctY * pctY));
                green = 0;
                blue = 0;
                rgb = (red << 16 | green << 8 | blue);
                // Set the pixel to the calculated color.
                result.setPixel(i, j, rgb);
            }
        }
    }
    return result;
}
```

Primeiro, quando o método é chamado, ele recebe um parâmetro, `radius`, indicando o raio da imagem em forma de círculo a ser criado. Em seguida, o código cria o objeto BitmapData no qual o círculo será desenhado. Aquele objeto, chamado `result`, é por fim transmitido de volta como o valor de retorno do método. Como mostrado no snippet de código a seguir, a ocorrência de BitmapData de `result` é criada com a largura e a altura tão grandes quando o diâmetro do círculo, sem transparência (`false` para o terceiro parâmetro), e preenchida com a cor `0x808080` (cinza médio):

```
var result:BitmapData = new BitmapData(diameter,
                                       diameter,
                                       false,
                                       0x808080);
```

Em seguida, o código usa loops para iterar cada pixel da imagem. O loop externo engloba cada coluna da imagem da esquerda para a direita (utilizando a variável `i` para representar a posição horizontal do pixel atualmente sendo manipulado), enquanto o loop interno engloba cada pixel da coluna atual de cima para baixo (com a variável `j` representando a posição vertical de pixels atual). O código para os loops (com o conteúdo do loop interno omitido) é mostrado aqui:

```
for (var i:int = 0; i < diameter; i++)
{
    for (var j:int = 0; j < diameter; j++)
    {
        ...
    }
}
```

À medida que o loop engloba os pixels um a um, em cada pixel um valor (o valor de cor daquele pixel na imagem de mapa) é calculado. Esse processo envolve quatro etapas:

1. O código calcula a distância do pixel atual do centro do círculo ao longo do eixo x (`i - radius`). Esse valor é dividido pelo raio para torná-lo um percentual de raio em vez de uma distância absoluta (`(i - radius) / radius`). Esse valor de percentual é armazenado em uma variável chamada `pctX`, e o valor equivalente para o eixo y é calculado e armazenado na variável `pctY`, como mostra este código:

   ```
   var pctX:Number = (i - radius) / radius;
   var pctY:Number = (j - radius) / radius;
   ```

2. Usando uma fórmula trigonométrica padrão, o Teorema de Pitágoras, a distância linear entre o cento do círculo e o ponto atual é calculado de `pctX` e `pctY`. Esse valor é armazenado em uma variável chamada `pctDistance`, como mostrado aqui:

   ```
   var pctDistance:Number = Math.sqrt(pctX * pctX + pctY * pctY);
   ```

3. Em seguida, o código verifica se o percentual de distância é menor do que 1 (significando 100% do raio, ou em outras palavras, se o pixel sendo considerado está dentro do raio do círculo). Se o pixel estiver dentro do círculo, ele recebe um valor de cor calculado (omitido aqui, mas descrito na etapa 4); em caso negativo, nada mais acontece com aquele pixel; sua cor é deixada como o padrão de cinza médio:

   ```
   if (pctDistance < 1)
   {
       ...
   }
   ```

4. Para aqueles pixels que estiverem dentro do círculo, um valor de cor é calculado para o pixel. A cor final será uma sombra de vermelho variando de preto (0% de vermelho) na borda esquerda do círculo até vermelho vivo (100%) na borda direita do círculo. O valor da cor é inicialmente calculado em três partes (vermelho, verde e azul), como mostrado aqui:BitmapData

   ```
   red = 128 * (1 + 0.75 * pctX * pctX * pctX / (1 - pctY * pctY));
   green = 0;
   blue = 0;
   ```

Observe que apenas a parte vermelha da cor (a variável `red`) tem realmente um valor. Os valores de verde e azul (as variáveis `green` e `blue`) são mostrados aqui para fins de explicação, mas podem ser omitidos. Como o propósito desse método é criar um círculo que contenha um gradiente vermelho, nenhum valor de verde ou azul é necessário.

Depois que os três valores de cores individuais estiverem determinados, eles serão combinados em um único valor de cor inteiro utilizando um algoritmo de deslocamento de bits padrão, mostrado neste código:

```
rgb = (red << 16 | green << 8 | blue);
```

Finalmente, com o valor de cor calculado, aquele valor é atribuído ao pixel atual utilizando o método `setPixel()` do objeto `result` BitmapData, mostrado aqui:

```
result.setPixel(i, j, rgb);
```

Capítulo 23: Trabalho em 3D (três dimensões)

Noções básicas de 3D

Introdução a 3D no ActionScript

A principal diferença entre um objeto bidimensional (2D) e um objeto tridimensional (3D) projetados em uma tela bidimensional é a adição de uma terceira dimensão ao objeto. A terceira dimensão permite que o objeto se aproxime ou se afaste do ponto de visão do usuário.

Quando você define explicitamente a propriedade `z` de um objeto de exibição com um valor numérico, o objeto automaticamente cria uma matriz de transformação 3D. É possível alterar essa matriz para modificar as configurações de transformação 3D do objeto.

Além disso, a rotação 3D é diferente da rotação 2D. Em 2D, o eixo de rotação está sempre perpendicular ao plano x/y - em outras palavras, ele está no eixo z. Em 3D, o eixo de rotação pode estar em torno de qualquer um dos eixos x, y ou z. Definir as propriedades de rotação e dimensionamento de um objeto de exibição permite que ele se mova no espaço 3D.

Tarefas comuns de 3D

As seguintes tarefas comuns relacionadas a 3D são exploradas neste capítulo:

- Criação de um objeto 3D
- Movimentação de um objeto no espaço 3D
- Rotação de um objeto no espaço 3D
- Representação da profundidade usando projeção em perspectiva
- Reordenação da lista de exibição para corresponder aos eixos z relacionados, de modo que os objetos apareçam na frente um dos outros corretamente para o usuário
- Transformação de objetos 3D usando matrizes 3D
- Uso de vetores para manipular objetos no espaço 3D
- Uso do método `Graphics.drawTriangles()` para criar perspectiva
- Uso de mapeamento UV para adicionar texturas de bitmap a um objeto 3D
- Definir o parâmetro de remoção do método `Graphics.drawTriangles()` para agilizar a renderização e ocultar partes de um objeto 3D opostas ao ponto de visão atual

Termos e conceitos importantes

A lista de referência a seguir contém termos importantes usados neste capítulo:

- perspectiva: em um plano 2D, a representação de linhas paralelas como convergentes em um ponto de fuga para dar a ilusão de profundidade e distância

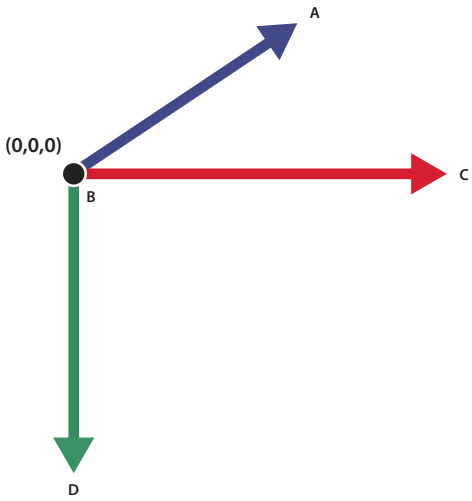
- projeção: a produção de uma imagem 2D de um objeto de dimensões maiores; a projeção 3D mapeia pontos 3D para um plano 2D
- rotação: alterar a orientação (e, muitas vezes, a posição) de um objeto movendo cada ponto incluído nele em movimento circular
- transformação: alterar pontos 3D ou conjuntos de pontos por translação, rotação, escala, inclinação ou uma combinação destas ações
- translação: alterar a posição de um objeto movendo cada ponto incluído nele da mesma forma e na mesma direção
- ponto de fuga: ponto em que linhas paralelas recuadas parecem se encontrar quando representadas em perspectiva linear
- vetor: um vetor 3D representa um ponto ou um local no espaço tridimensional com as coordenadas cartesianas x, y e z.
- vértice: um ponto de canto
- malha com textura: qualquer ponto que define um objeto no espaço 3D
- mapeamento UV: uma forma de aplicar textura ou bitmap a uma superfície 3D. O mapeamento UV atribui valores a coordenadas de uma imagem como porcentagens dos eixos horizontal (U) e vertical (V).
- valor T: o fator de dimensionamento para determinar o tamanho de um objeto 3D à medida que ele se aproxima, ou se afasta, do ponto de visão atual
- remoção: renderização, ou não, de superfícies com contorno específico. O uso da remoção pode ocultar superfícies que não estão visíveis para o ponto de visão atual.

Noções básicas sobre os recursos 3D do Flash Player e o runtime do AIR

Nas versões do Flash Player anteriores ao Flash Player 10, os objetos de exibição têm duas propriedades, `x` e `y`, para posicioná-los em um plano 2D. A partir do Flash Player 10, todo objeto de exibição do ActionScript tem uma propriedade `z` que permite posicioná-lo ao longo do eixo z, normalmente usado para indicar profundidade ou distância.

O Flash Player 10 introduz o suporte para efeitos 3D. No entanto, os objetos de exibição são inerentemente planos. Cada objeto de exibição, como MovieClip ou Sprite, basicamente renderiza-se em duas dimensões em um único plano. Os recursos 3D permitem colocar, movimentar, girar e transformar todos esses objetos planos em tridimensionais. Eles também permitem gerenciar pontos 3D e convertê-los em coordenadas x, y 2D para que você possa projetar objetos 3D em uma exibição 2D. É possível simular muitos tipos de experiências 3D usando esses recursos.

O sistema de coordenadas 3D usado pelo ActionScript difere do de outros sistemas. Quando são usadas coordenadas 2D no ActionScript, o valor de x aumenta à medida que você vai para a direita no eixo x, e o valor de y aumenta conforme você percorre o eixo y. O sistema de coordenadas 3D retém essas convenções e adiciona um eixo z cujo valor aumenta à medida que você se afasta do ponto de visão.



As direções positivas dos eixos x, y e z no sistema de coordenadas 3D do ActionScript.
***A.** Eixo +Z **B.** Origem **C.** Eixo +X **D.** Eixo +Y*

***Nota:** O Flash Player e o AIR sempre representam 3D em camadas. Isso significa que, se o objeto A está na frente do objeto B na lista de exibição, o Flash Player ou o AIR sempre renderiza A na frente de B, independentemente dos valores do eixo z dos dois objetos. Para resolver este conflito entre a ordem na lista de exibição e a ordem no eixo z, use o método* `transform.getRelativeMatrix3D()` *para salvar e reordenar as camadas de objetos de exibição 3D. Para obter mais informações, consulte "Uso de objetos Matrix3D para reordenar a exibição" na página 518.*

As seguintes classes do ActionScript dão suporte aos novos recursos relacionados a 3D:

1. A classe flash.display.DisplayObject contém a propriedade `z` e novas propriedades de rotação e dimensionamento para manipular objetos de exibição no espaço 3D. O método `DisplayObject.local3DToGlobal()` oferece uma maneira simples de projetar geometria 3D em um plano 2D.
2. A classe flash.geom.Vector3D pode ser usada como estrutura de dados para gerenciar pontos 3D. Ela também oferece suporte para matemática de vetores.
3. A classe flash.geom.Matrix3D dá suporte a transformações complexas de geometria 3D, como rotação, dimensionamento e translação.
4. A classe flash.geom.PerspectiveProjection controla os parâmetros para mapear geometria 3D em uma exibição 2D.

Existem duas abordagens distintas para simular imagens 3D no ActionScript:

1. Organizar e animar objetos planos no espaço 3D. Esta abordagem envolve animar objetos de exibição usando as propriedades x, y e z de objetos de exibição ou definir propriedades de rotação e dimensionamento utilizando a classe DisplayObject. É possível obter movimentos mais complexos usando o objeto DisplayObject.transform.matrix3D. O objeto DisplayObject.transform.perspectiveProjection personaliza a forma como os objetos de exibição são desenhados em perspectiva 3D. Use esta abordagem quando quiser animar objetos 3D formados principalmente por planos. Exemplos desta abordagem incluem galerias de imagens 3D ou objetos de animação 2D organizados no espaço 3D.
2. Gerar triângulos 2D a partir de geometria 3D e renderizá-los com texturas. Para usar esta abordagem, primeiro você deve definir e gerenciar dados sobre objetos 3D e, em seguida, converter esses dados em triângulos 2D para fins de renderização. É possível mapear texturas de bitmap para esses triângulos, e depois eles são desenhados em um objeto gráfico através do método `Graphics.drawTriangles()`. Exemplos desta abordagem incluem carregar dados de modelo 3D a partir de um arquivo e renderizar o modelo na tela ou gerar e desenhar um terreno 3D como malhas de triângulo.

Criação e movimentação de objetos 3D

Para converter um objeto de exibição 2D em um objeto de exibição 3D, você pode definir explicitamente a propriedade `z` correspondente com um valor numérico. Quando você atribui um valor para a propriedade `z`, é criado um novo objeto Transform para o objeto de exibição. Definir as propriedades `DisplayObject.rotationX` ou `DisplayObject.rotationY` também cria um novo objeto Transform. O objeto Transform contém uma propriedade `Matrix3D` que determina como o objeto de exibição é representado no espaço 3D.

O seguinte código define as coordenadas para um objeto de exibição chamado "leaf" (folha):

```
leaf.x = 100; leaf.y = 50; leaf.z = -30;
```

Você pode ver estes valores, bem como as propriedades derivadas deles, na propriedade `matrix3D` do objeto Transform da folha:

```
var leafMatrix:Matrix3D  = leaf.transform.matrix3D;

trace(leafMatrix.position.x);
trace(leafMatrix.position.y);
trace(leafMatrix.position.z);
trace(leafMatrix.position.length);
trace(leafMatrix.position.lengthSquared);
```

Para obter informações sobre as propriedades do objeto Transform, consulte a classe Transform. Para obter informações sobre as propriedades do objeto Matrix3D, consulte a classe Matrix3D.

Movimentação de um objeto no espaço 3D

É possível movimentar um objeto no espaço 3D alterando os valores de suas propriedades `x`, `y` ou `z`. Quando você altera o valor da propriedade `z`, o objeto parece aproximar-se ou afastar-se do visualizador.

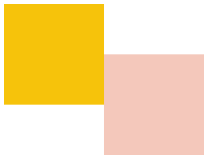
O código a seguir movimenta duas elipses para frente e para trás ao longo dos eixos z alterando o valor das respectivas propriedades `z` em resposta a um evento. `ellipse2` movimenta-se mais rápido do que `ellipse1`: sua propriedade `z` é aumentada por um múltiplo de 20 em cada evento Frame, enquanto a propriedade `z` de `ellipse1` é aumentada por um múltiplo de 10:

```
var depth:int = 1000;

function ellipse1FrameHandler(e:Event):void
{
    ellipse1Back = setDepth(e, ellipse1Back);
    e.currentTarget.z += ellipse1Back * 10;
}
function ellipse2FrameHandler(e:Event):void
{
    ellipse2Back = setDepth(e, ellipse1Back);
    e.currentTarget.z += ellipse1Back * 20;
}
function setDepth(e:Event, d:int):int
{
    if(e.currentTarget.z > depth)
    {
        e.currentTarget.z = depth;
        d = -1;
    }
    else if (e.currentTarget.z <  0)
    {
        e.currentTarget.z = 0;
        d = 1;
    }
}
```

Rotação de um objeto no espaço 3D

É possível girar um objeto de três formas diferentes, dependendo de como você definir as propriedades de rotação do objeto: `rotationX`, `rotationY` e `rotationZ`.

A figura abaixo mostra dois quadrados que não são girados:

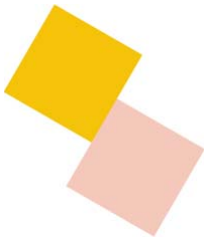


A próxima figura mostra os dois quadrados quando você incrementa a propriedade `rotationY` do contêiner dos quadrados para girá-los no eixo y. Girar o contêiner, ou o objeto de exibição pai, dos dois quadrados gira ambos os quadrados:

```
container.rotationY += 10;
```

A próxima figura mostra o que acontece quando você define a propriedade `rotationX` do contêiner dos quadrados. Isso gira os quadrados no eixo x.

A próxima figura mostra o que acontece quando você incrementa a propriedade `rotationZ` do contêiner dos quadrados. Isso gira os quadrados no eixo z.



Um objeto de exibição pode, ao mesmo tempo, se movimentar e girar no espaço 3D.

Projeção de objetos 3D em uma exibição 2D

A classe PerspectiveProjection do pacote `flash.geom` oferece uma maneira simples de aplicar perspectiva rudimentar quando você movimenta objetos de exibição pelo espaço 3D.

Caso você não crie uma projeção em perspectiva para o espaço 3D explicitamente, o mecanismo 3D usará um objeto PerspectiveProjection padrão que existe na raiz e é propagado para todos os seus filhos.

As três propriedades que definem como um objeto PerspectiveProjection exibe o espaço 3D são:

- `fieldOfView`
- `projectionCenter`
- `focalLength`

Modificar o valor de `fieldOfView` modifica automaticamente o valor de `focalLength` e vice-versa, uma vez que elas são interdependentes.

A fórmula usada para calcular a propriedade `focalLength` considerando-se o valor de `fieldOfView` é a seguinte:

```
focalLength = stageWidth/2 * (cos(fieldOfView/2) / sin(fieldOfView/2)
```

Normalmente você modificaria a propriedade `fieldOfView` de maneira explícita.

Campo de visão

Manipulando a propriedade `fieldOfView` da classe PerspectiveProjection, é possível fazer com que um objeto de exibição 3D que está se aproximando do visualizador pareça maior e com que um objeto que está se afastando do visualizador pareça menor.

A propriedade `fieldOfView` especifica um ângulo entre 0 e 180 graus que determina a intensidade da projeção em perspectiva. Quanto maior o valor, mais intensa a distorção aplicada a um objeto de exibição que se movimenta em seu eixo z. Um valor baixo de `fieldOfView` resulta em um dimensionamento muito pequeno e faz com que os objetos pareçam se movimentar apenas um pouco para trás no espaço. Um valor alto de `fieldOfView` gera mais distorção e o aspecto de maior movimento. O valor máximo de 180 graus gera um efeito de lente de câmera olho de peixe exagerado.

Centro da projeção

A propriedade `projectionCenter` representa o ponto de fuga na projeção em perspectiva. Ela é aplicada como um deslocamento até o ponto de registro padrão (0,0) no canto superior esquerdo do palco.

Conforme um objeto parece se afastar do visualizador, ele se movimenta em direção ao ponto de fuga e, por fim, desaparece. Imagine um corredor infinitamente longo. Quando você olha para o final dele, as extremidades das paredes se convergem em um ponto de fuga no final do corredor.

Se o ponto de fuga está no centro do palco, o corredor desaparece em direção a um ponto no centro. O valor padrão da propriedade `projectionCenter` é o centro do palco. Se, por exemplo, você quiser que os elementos apareçam no lado esquerdo do palco e que uma área 3D apareça no lado direito, defina `projectionCenter` como um ponto no lado direito do palco para torná-lo o ponto de fuga da sua área de exibição 3D.

Distância focal

A propriedade `focalLength` representa a distância entre a origem do ponto de visão (0,0,0) e a localização do objeto de exibição em seu eixo z.

Uma distância focal longa é como uma teleobjetiva com uma visualização limitada e distâncias reduzidas entre objetos. Uma distância focal curta é como uma lente com grande abertura angular, com a qual você tem uma visualização ampla com muita distorção. Uma distância focal média assemelha-se ao que os olhos humanos vêem.

Normalmente, a propriedade `focalLength` é recalculada dinamicamente durante a transformação de perspectiva à medida que o objeto de exibição se movimenta, mas é possível defini-la de forma explícita.

Valores padrão da projeção em perspectiva

O objeto PerspectiveProjection padrão criado na raiz tem os seguintes valores:

- `fieldOfView: 55`
- `perspectiveCenter: stagewidth/2, stageHeight/2`
- `focalLength: stageWidth/ 2 * ( cos(fieldOfView/2) / sin(fieldOfView/2) )`

Esses são os valores que serão utilizados se você não criar seu próprio objeto PerspectiveProjection.

É possível instanciar seu próprio objeto PerspectiveProjection com a intenção de você mesmo modificar `projectionCenter` e `fieldOfView`. Nesse caso, os valores padrão do objeto recém-criado são os seguintes, de acordo com um tamanho de palco padrão de 500 por 500:

- `fieldOfView: 55`
- `perspectiveCenter: 250,250`
- `focalLength: 480.24554443359375`

Exemplo: Projeção em perspectiva

O exemplo a seguir demonstra o uso da projeção em perspectiva para criar o espaço 3D. Ele mostra como você pode modificar o ponto de fuga e alterar a projeção em perspectiva do espaço através da propriedade `projectionCenter`. Essa modificação força o recálculo das propriedades `focalLength` e `fieldOfView` com a distorção concomitante do espaço 3D.

Este exemplo:

1. Cria uma entidade gráfica chamada `center`, como um círculo com cruzes
2. Atribui as coordenadas da entidade gráfica `center` à propriedade `projectionCenter` da propriedade`perspectiveProjection` da propriedade `transform` da raiz
3. Adiciona ouvintes de eventos para eventos de mouse que chamam manipuladores que modificam a propriedade `projectionCenter` para que ela siga a localização do objeto `center`
4. Cria quatro caixas com o estilo acordeão que formam as paredes do espaço em perspectiva

Quando você testar este exemplo, ProjectionDragger.swf, arraste o círculo para diferentes locais. O ponto de fuga segue o círculo, parando no lugar em que você soltá-lo. Observe as caixas que circundam o espaço se alongam e se tornam distorcidas quando você afasta o centro de projeção do centro do palco.

Para obter os arquivos de aplicativo deste exemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_br. Os arquivos do aplicativo ProjectionDragger estão na pasta Samples/ProjectionDragger.

```
package
{
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.geom.Point;
    import flash.events.*;
    public class ProjectionDragger extends Sprite
    {
        private var center : Sprite;
        private var boxPanel:Shape;
        private var inDrag:Boolean = false;

        public function ProjectionDragger():void
        {
            createBoxes();
            createCenter();
        }
        public function createCenter():void
        {
            var  centerRadius:int = 20;

            center = new Sprite();

            // circle
            center.graphics.lineStyle(1, 0x000099);
            center.graphics.beginFill(0xCCCCCC, 0.5);
            center.graphics.drawCircle(0, 0, centerRadius);
            center.graphics.endFill();
            // cross hairs
            center.graphics.moveTo(0, centerRadius);
            center.graphics.lineTo(0, -centerRadius);
            center.graphics.moveTo(centerRadius, 0);
```

```
            center.graphics.lineTo(-centerRadius, 0);
            center.x = 175;
            center.y = 175;
            center.z = 0;
            this.addChild(center);

            center.addEventListener(MouseEvent.MOUSE_DOWN, startDragProjectionCenter);
            center.addEventListener(MouseEvent.MOUSE_UP, stopDragProjectionCenter);
            center.addEventListener( MouseEvent.MOUSE_MOVE, doDragProjectionCenter);
            root.transform.perspectiveProjection.projectionCenter = new Point(center.x,
center.y);
        }
        public function createBoxes():void
        {
            // createBoxPanel();
            var boxWidth:int = 50;
            var boxHeight:int = 50;
            var numLayers:int = 12;
            var depthPerLayer:int = 50;

            // var boxVec:Vector.<Shape> = new Vector.<Shape>(numLayers);
            for (var i:int = 0; i < numLayers; i++)
            {
                this.addChild(createBox(150, 50, (numLayers - i) * depthPerLayer, boxWidth,
boxHeight, 0xCCCCFF));
                this.addChild(createBox(50, 150, (numLayers - i) * depthPerLayer, boxWidth,
boxHeight, 0xFFCCCC));
                this.addChild(createBox(250, 150, (numLayers - i) * depthPerLayer, boxWidth,
boxHeight, 0xCCFFCC));
                this.addChild(createBox(150, 250, (numLayers - i) * depthPerLayer, boxWidth,
boxHeight, 0xDDDDDD));
            }
        }

        public function createBox(xPos:int = 0, yPos:int = 0, zPos:int = 100, w:int = 50, h:int
= 50, color:int = 0xDDDDDD):Shape
        {
            var box:Shape = new Shape();
            box.graphics.lineStyle(2, 0x666666);
            box.graphics.beginFill(color, 1.0);
            box.graphics.drawRect(0, 0, w, h);
            box.graphics.endFill();
            box.x = xPos;
            box.y = yPos;
            box.z = zPos;
            return box;
        }
        public function startDragProjectionCenter(e:Event)
        {
            center.startDrag();
            inDrag = true;
```

```
        }

        public function doDragProjectionCenter(e:Event)
        {
            if (inDrag)
            {
                root.transform.perspectiveProjection.projectionCenter = new Point(center.x,
center.y);
            }
        }

        public function stopDragProjectionCenter(e:Event)
        {
            center.stopDrag();
            root.transform.perspectiveProjection.projectionCenter = new Point(center.x,
center.y);
            inDrag = false;
        }
    }
}
```

Para projeção em perspectiva mais complexa, use a classe Matrix3D.

Execução de transformações 3D complexas

A classe Matrix3D permite transformar pontos 3D dentro de um espaço de coordenadas ou mapear pontos 3D de um espaço de coordenadas para outro.

Você não precisa conhecer a matemática de matrizes para usar a classe Matrix3D. A maior parte das operações de transformação comuns pode ser feita usando-se os métodos da classe. Você não precisa se preocupar em definir ou calcular explicitamente os valores de cada elemento da matriz.

Depois de definir a propriedade `z` de um objeto de exibição como um valor numérico, você pode recuperar a matriz de transformação dele usando a propriedade Matrix3D do objeto Transform do objeto de exibição:

```
var leafMatrix:Matrix3D = this.transform.matrix3D;
```

Você pode usar os métodos do objeto Matrix3D para executar translação, rotação, dimensionamento e projeção em perspectiva no objeto de exibição.

Use a classe Vector3D com suas propriedades `x`, `y` e `z` para gerenciar pontos 3D. Ela também pode representar um vetor espacial na física, que tem uma direção e uma magnitude. Os métodos da classe Vector3D permitem executar cálculos comuns com vetores espaciais, como cálculos de adição, produto escalar e produto complementar.

Nota: *A classe Vector3D não está relacionada à classe Vector do ActionScript. A classe Vector3D contém propriedades e métodos para definir e manipular pontos 3D, enquanto a classe Vector oferece suporte a matrizes de objetos com tipo.*

Criação de objetos Matrix3D

Há três principais maneiras de criar ou recuperar objetos `Matrix3D`:

1 Use o método do construtor `Matrix3D()` para instanciar uma nova matriz. O construtor `Matrix3D()` toma um objeto `Vector` que contém 16 valores numéricos e coloca cada valor em uma célula da matriz. Por exemplo:

```
var rotateMatrix:Matrix3D = new Matrix3D(1,0,0,1, 0,1,0,1, 0,0,1,1, 0,0,0,1);
```

2 Defina o valor da propriedade `z` de um objeto de exibição. Em seguida, recupere a matriz de transformação da propriedade `transform.matrix3D` desse objeto.

3 Recupere o objeto Matrix3D que controla a exibição de objetos 3D no palco obtendo o valor da propriedade `perspectiveProjection.matrix3D` do objeto de exibição raiz.

Aplicação de várias transformações 3D

É possível aplicar muitas transformações 3D de uma só vez usando um objeto Matrix3D. Por exemplo, para girar, dimensionar e, depois, movimentar um cubo, você pode aplicar três transformações separadas a cada ponta do cubo. No entanto, é muito mais eficiente pré-calcular várias transformações em um objeto Matrix3D e, em seguida, executar uma transformação de matriz em cada uma das pontas.

Nota: *A ordem em que as transformações de matriz são aplicadas é importante. Os cálculos de matriz não são comutativos. Por exemplo, o resultado de aplicar uma rotação seguida de uma translação é diferente do resultado de aplicar a mesma translação seguida da mesma rotação.*

O exemplo a seguir mostra duas maneiras de executar várias transformações 3D.

```
package {
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.display.Graphics;
    import flash.geom.*;

public class Matrix3DTransformsExample extends Sprite
    {
        private var rect1:Shape;
        private var rect2:Shape;

public function Matrix3DTransformsExample():void
        {
            var pp:PerspectiveProjection = this.transform.perspectiveProjection;
            pp.projectionCenter = new Point(275,200);
            this.transform.perspectiveProjection = pp;

            rect1 = new Shape();
            rect1.x = -70;
            rect1.y = -40;
            rect1.z = 0;
            rect1.graphics.beginFill(0xFF8800);
            rect1.graphics.drawRect(0,0,50,80);
            rect1.graphics.endFill();
            addChild(rect1);

            rect2 = new Shape();
            rect2.x = 20;
            rect2.y = -40;
            rect2.z = 0;
            rect2.graphics.beginFill(0xFF0088);
            rect2.graphics.drawRect(0,0,50,80);
            rect2.graphics.endFill();
            addChild(rect2);
```

```
            doTransforms();
        }

        private function doTransforms():void
        {
            rect1.rotationX = 15;
            rect1.scaleX = 1.2;
            rect1.x += 100;
            rect1.y += 50;
            rect1.rotationZ = 10;

            var matrix:Matrix3D = rect2.transform.matrix3D;
            matrix.appendRotation(15, Vector3D.X_AXIS);
            matrix.appendScale(1.2, 1, 1);
            matrix.appendTranslation(100, 50, 0);
            matrix.appendRotation(10, Vector3D.Z_AXIS);
            rect2.transform.matrix3D = matrix;
        }
    }
}
```

No método `doTransforms()`, o primeiro bloco de código usa as propriedades DisplayObject para alterar a rotação, o dimensionamento e a posição de uma forma de retângulo. O segundo bloco de código usa os métodos da classe Matrix3D para fazer as mesmas transformações.

A principal vantagem de usar os métodos `Matrix3D` é que todos os cálculos são realizados primeiro na matriz. Em seguida, eles são aplicados ao objeto de exibição apenas uma vez, quando sua propriedade `transform.matrix3D` é definida. Configurar propriedades DisplayObject torna a leitura do código-fonte um pouco mais simples. Todavia, sempre que é definida uma propriedade de rotação ou dimensionamento, ela gera vários cálculos e altera diversas propriedades do objeto de exibição.

Se o seu código aplicará as mesmas transformações complexas a objetos de exibição mais de uma vez, salve o objeto Matrix3D como uma variável e, em seguida, aplique-o de novo repetidas vezes.

Uso de objetos Matrix3D para reordenar a exibição

Conforme mencionado anteriormente, a ordem em camadas dos objetos de exibição na lista de exibição determina a ordem em camadas da exibição, independentemente dos eixos x relacionados. Se a sua animação transforma as propriedades de objetos de exibição em uma ordem diferente da ordem da lista de exibição, o visualizador poderá ver os objetos de exibição em uma disposição em camadas que não corresponde à disposição em camadas no eixo z. Por isso, um objeto que deve parecer mais distante do visualizador pode aparecer na frente de um objeto que está mais perto do visualizador.

Para assegurar que a disposição em camadas dos objetos de exibição 3D corresponda às profundidades relativas dos objetos, use uma abordagem como a seguinte:

1. Use o método `getRelativeMatrix3D()` do objeto Transform para obter os `eixos z` relacionados dos objetos de exibição 3D filho.
2. Use o método `removeChild()` para remover os objetos da lista de exibição.
3. Classifique os objetos de exibição com base nos valores do eixo z relacionado.
4. Use o método `addChild()` para adicionar os filhos de volta à lista de exibição na ordem inversa.

Essa reordenação assegura que seus objetos serão exibidos de acordo com os eixos z relacionados.

O código a seguir força a exibição correta das seis faces de uma caixa 3D. Ele reordena as faces da caixa depois que rotações foram aplicadas a ela:

```
public var faces:Array; . . .

public function ReorderChildren()
{
    for(var ind:uint = 0; ind < 6; ind++)
    {
        faces[ind].z = faces[ind].child.transform.getRelativeMatrix3D(root).position.z;
        this.removeChild(faces[ind].child);
    }
    faces.sortOn("z", Array.NUMERIC | Array.DESCENDING);
    for (ind = 0; ind < 6; ind++)
    {
        this.addChild(faces[ind].child);
    }
}
```

Para obter os arquivos de aplicativo deste exemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_br. Os arquivos do aplicativo estão na pasta Samples/ReorderByZ.

Uso de triângulos para obter efeitos 3D

No ActionScript, você executa transformações de bitmap usando o método `Graphics.drawTriangles()`, pois os modelos 3D são representados por um conjunto de triângulos no espaço. (Contudo, o Flash Player e o AIR não dão suporte a um buffer de profundidade, por isso os objetos de exibição são inerentemente planos, ou 2D. Isto é descrito em "Noções básicas sobre os recursos 3D do Flash Player e o runtime do AIR" na página 508.) O método `Graphics.drawTriangles()` é parecido com o método `Graphics.drawPath()` no sentido de que usa um conjunto de coordenadas para desenhar um caminho de triângulo.

Para se familiarizar com o uso de `Graphics.drawPath()`, consulte "Caminhos de desenho" na página 335.

O método `Graphics.drawTriangles()` usa um Vector.<Number> para especificar as localizações das pontas para o caminho do triângulo:

```
drawTriangles(vertices:Vector.<Number>, indices:Vector.<int> = null, uvtData:Vector.<Number>
= null, culling:String = "none"):void
```

O primeiro parâmetro de `drawTriangles()` é o único necessário: o parâmetro `vertices`. Este parâmetro é um vetor de números que define as coordenadas através das quais os seus triângulos são desenhados. Cada três conjuntos de coordenadas (seis números) representa um caminho de triângulo. Sem o parâmetro `indices`, o comprimento do vetor deve sempre ser um fator de seis, uma vez que cada triângulo requer três pares de coordenadas (três conjuntos de dois valores x/y). Por exemplo:

```
graphics.beginFill(0xFF8000);
graphics.drawTriangles(
    Vector.<Number>([
        10,10,  100,10,  10,100,
        110,10, 110,100, 20,100]));
```

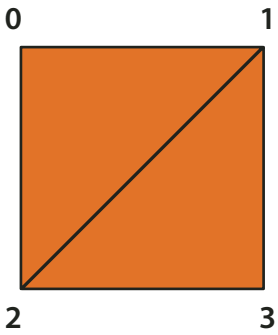
Nenhum desses triângulos compartilham pontas, mas, se compartilhassem, o segundo parâmetro de `drawTriangles()`, `indices`, poderia ser usado para reutilizar valores do vetor `vertices` para mais de um triângulo.

Quando usar o parâmetro `indices`, é importante que você saiba que os valores de `indices` são índices de pontos e não índices relacionados diretamente aos elementos de matriz `vertices`. Em outras palavras, um índice no vetor `vertices` conforme definido por `indices` é, na verdade, o índice real dividido por 2. Para a terceira ponta de um vetor `vertices`, por exemplo, use um valor de 2 para `indices`, mesmo que o primeiro valor numérico dessa ponta comece no índice de vetor de 4.

Por exemplo, mescle dois triângulos para que compartilhem a aresta diagonal usando o parâmetro `indices`:

```
graphics.beginFill(0xFF8000);
graphics.drawTriangles(
    Vector.<Number>([10,10, 100,10, 10,100, 100,100]),
    Vector.<int>([0,1,2, 1,3,2]));
```

Observe que, apesar de agora ter sido desenhado um quadrado usando-se dois triângulos, somente quatro pontas foram especificadas no vetor `vertices`. Usando `indices`, as duas pontas compartilhadas pelos dois triângulos são reutilizadas para cada triângulo. Isso diminui a contagem geral de vértices de 6 (12 números) para 4 (8 números):



Um quadrado desenhado com dois triângulos usando o parâmetro vertices

Esta técnica torna-se útil com malhas de triângulos maiores, onde a maioria das pontas são compartilhadas por vários triângulos.

Todos os preenchimentos podem ser aplicados a triângulos. Os preenchimentos são aplicados à malha de triângulos resultante, assim como ocorre com qualquer outra forma.

Transformação de bitmaps

As transformações de bitmap dão a ilusão de perspectiva ou "textura" em um objeto tridimensional. Especificamente, você pode distorcer um bitmap em direção a um ponto de fuga de modo que a imagem pareça diminuir à medida que se aproxima do ponto de fuga. Se preferir, você pode usar um bitmap bidimensional para criar uma superfície para um objeto tridimensional, dando a ilusão de textura ou “wrapping” nesse objeto 3D.

Uma superfície bidimensional usando um ponto de fuga e um objeto tridimensional delimitado com um bitmap.

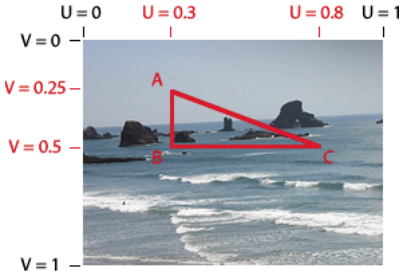
Mapeamento UV

Depois que começar a trabalhar com texturas, você desejará usar o parâmetro uvtData de `drawTriangles()`. Esse parâmetro permite configurar o mapeamento UV para preenchimentos de bitmap.

O mapeamento UV é um método de aplicar textura a objetos. Ele se baseia em dois valores: um valor horizontal U (x) e um valor vertical V (y). Em vez de serem baseados em valores de pixel, eles são baseados em porcentagens. 0 U e 0 V é o canto superior esquerdo de uma imagem, e 1 U e 1 V é o canto inferior direito:

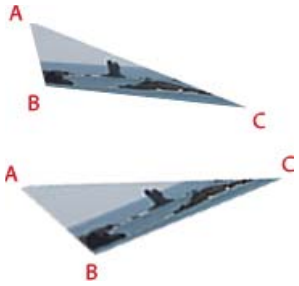
As localizações UV 0 e 1 em uma imagem de bitmap

Os vetores de um triângulo podem receber coordenadas UV para se associarem às respectivas localizações em uma imagem:



As coordenadas UV de uma área triangular de uma imagem de bitmap

Os valores UV permanecem consistentes com as pontas do triângulo:



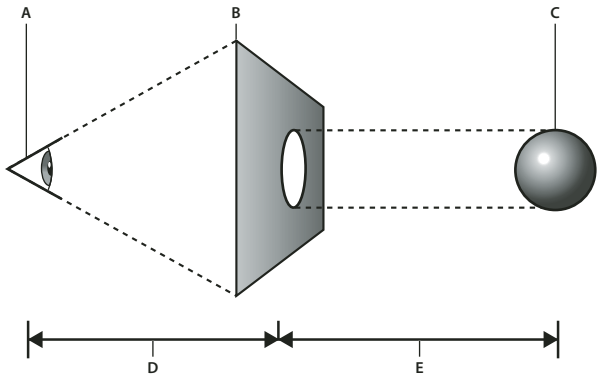
Os vértices do triângulo são movimentados e o bitmap é distorcido para manter iguais os valores UV de uma ponta individual

Conforme transformações 3D do ActionScript são aplicadas ao triângulo associado ao bitmap, a imagem do bitmap é aplicada ao triângulo com base nos valores UV. Portanto, em vez de usar cálculos de matriz, defina ou ajuste os valores UV para criar um efeito tridimensional.

O método `Graphics.drawTriangles()` também aceita uma informação opcional para transformações tridimensionais: o valor T. O valor T em uvtData representa a perspectiva 3D ou, mais especificamente, o fator de escala do vértice associado. O mapeamento UVT adiciona correção de perspectiva ao mapeamento UV. Por exemplo, se um objeto estiver posicionado no espaço 3D longe do ponto de visão, de modo que pareça ter 50% de seu tamanho "original", o valor T desse objeto será 0,5. Uma vez que triângulos são desenhados para representar objetos no espaço 3D, sua localização no eixo z determina seus valores T. A equação que determina o valor T é a seguinte:

```
T = focalLength/(focalLength + z);
```

Nesta equação, focalLength representa uma distância focal ou uma localização "na tela" calculada, que determina o grau da perspectiva observado na exibição.



A distância focal e o valor z
***A.** ponto de visão **B.** tela **C.** Objeto 3D **D.** valor de focalLength **E.** valor z*

O valor de T é usado para dimensionar formas básicas para fazer com que pareçam mais distantes. Normalmente é o valor usado para converter pontos 3D em pontos 2D. No caso de dados UVT, também é usado para dimensionar um bitmap entre as pontas de um triângulo com perspectiva.

Quando você define valores UVT, o valor T segue diretamente os valores UV definidos para um vértice. Com a inclusão de T, cada três valores no parâmetro `uvtData` (U, V e T) correspondem a cada dois valores no parâmetro `vertices` (x e y). Com valores UV apenas, uvtData.length == vertices.length. Com a inclusão de um valor T, uvtData.length = 1,5*vertices.length.

Exemplo: Um bitmap girando com perspectiva

O exemplo a seguir mostra um plano sendo girado no espaço 3D usando dados UVT. Este exemplo usa uma imagem chamada ocean.jpg e uma classe "auxiliar" (ImageLoader) para carregar a imagem de maneira que ela possa ser atribuída ao objeto BitmapData.

Este é o código-fonte da classe ImageLoader (salve-o em um arquivo chamado ImageLoader.as):

```
package {
    import flash.display.*
    import flash.events.*;
    import flash.net.URLRequest;
    public class ImageLoader extends Sprite {
        public var url:String;
        public var bitmap:Bitmap;
    public function ImageLoader(loc:String = null) {
        if (loc != null){
            url = loc;
            loadImage();
        }
    }
    public function loadImage():void{
        if (url != null){
            var loader:Loader = new Loader();
            loader.contentLoaderInfo.addEventListener(Event.COMPLETE, onComplete);
            loader.contentLoaderInfo.addEventListener(IOErrorEvent.IO_ERROR, onIoError);

                var req:URLRequest = new URLRequest(url);
                loader.load(req);
            }
        }

    private function onComplete(event:Event):void {
            var loader:Loader = Loader(event.target.loader);
            var info:LoaderInfo = LoaderInfo(loader.contentLoaderInfo);
            this.bitmap = info.content as Bitmap;
            this.dispatchEvent(new Event(Event.COMPLETE));
    }

    private function onIoError(event:IOErrorEvent):void {
            trace("onIoError: " + event);
    }
    }
}
```

E este é o ActionScript que usa triângulos, mapeamento UV e valores T para fazer com que a imagem pareça estar diminuindo à medida que se aproxima de um ponto de fuga e girando. Salve este código em um arquivo denominado Spinning3dOcean.as:

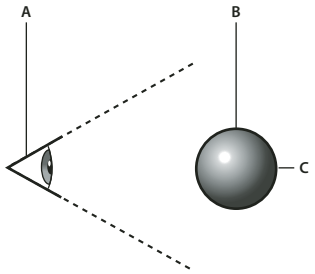
```
package {
    import flash.display.*
    import flash.events.*;
    import flash.utils.getTimer;

    public class Spinning3dOcean extends Sprite {
        // plane vertex coordinates (and t values)
        var x1:Number = -100,y1:Number = -100,z1:Number = 0,t1:Number = 0;
        var x2:Number = 100,y2:Number = -100,z2:Number = 0,t2:Number = 0;
        var x3:Number = 100,y3:Number = 100,z3:Number = 0,t3:Number = 0;
        var x4:Number = -100,y4:Number = 100,z4:Number = 0,t4:Number = 0;
        var focalLength:Number = 200;
        // 2 triangles for 1 plane, indices will always be the same
        var indices:Vector.<int>;

        var container:Sprite;

        var bitmapData:BitmapData; // texture
        var imageLoader:ImageLoader;
        public function Spinning3dOcean():void {
            indices =  new Vector.<int>();
            indices.push(0,1,3, 1,2,3);

            container = new Sprite(); // container to draw triangles in
            container.x = 200;
            container.y = 200;
            addChild(container);

            imageLoader = new ImageLoader("ocean.jpg");
            imageLoader.addEventListener(Event.COMPLETE, onImageLoaded);
        }
        function onImageLoaded(event:Event):void {
            bitmapData = imageLoader.bitmap.bitmapData;
            // animate every frame
            addEventListener(Event.ENTER_FRAME, rotatePlane);
        }
        function rotatePlane(event:Event):void {
            // rotate vertices over time
            var ticker = getTimer()/400;
            z2 = z3 = -(z1 = z4 = 100*Math.sin(ticker));
            x2 = x3 = -(x1 = x4 = 100*Math.cos(ticker));

            // calculate t values
```

```
            t1 = focalLength/(focalLength + z1);
            t2 = focalLength/(focalLength + z2);
            t3 = focalLength/(focalLength + z3);
            t4 = focalLength/(focalLength + z4);

            // determine triangle vertices based on t values
            var vertices:Vector.<Number> = new Vector.<Number>();
            vertices.push(x1*t1,y1*t1, x2*t2,y2*t2, x3*t3,y3*t3, x4*t4,y4*t4);
            // set T values allowing perspective to change
            // as each vertex moves around in z space
            var uvtData:Vector.<Number> = new Vector.<Number>();
            uvtData.push(0,0,t1, 1,0,t2, 1,1,t3, 0,1,t4);

            // draw
            container.graphics.clear();
            container.graphics.beginBitmapFill(bitmapData);
            container.graphics.drawTriangles(vertices, indices, uvtData);
        }
    }
}
```

Salve esses dois arquivos de classe no mesmo diretório de uma imagem chamada "ocean.jpg". Você pode ver como o bitmap original é transformado para dar a impressão de que ele está desaparecendo conforme aumenta a distância e girando no espaço 3D.

Remoção

Remoção é o processo que determina quais superfícies de um objeto tridimensional o renderizador não deve renderizar pelo fato de estarem ocultas do ponto de visão atual. No espaço 3D, a superfície na "parte de trás" de um objeto tridimensional fica oculta do ponto de visão:



A parte de trás de um objeto 3D fica oculta do ponto de visão.
***A.** ponto de visão **B.** objeto 3D **C.** a parte de trás de um objeto tridimensional*

Inerentemente, todos os triângulos sempre são renderizados, seja qual for o tamanho, a forma ou a posição. A remoção assegura que o Flash Player ou o AIR renderize o objeto 3D corretamente. Além disso, para ganhar tempo nos ciclos de renderização, às vezes você deseja que alguns triângulos sejam ignorados pelo renderizador. Pense em um cubo girando no espaço. Em um determinado momento, você não verá mais do que três lados desse cubo, uma vez que os lados que não aparecem na exibição estariam voltados para a outra direção no outro lado do cubo. Como esses lados não serão vistos, o renderizados não deve desenhá-los. Sem a remoção, o Flash Player ou o AIR renderiza os lados da frente e de trás.

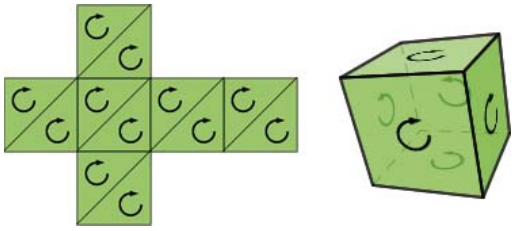
Um cubo tem lados que não ficam visíveis no ponto de visão atual

Por isso, o método `Graphics.drawTriangles()` tem um quarto parâmetro para estabelecer um valor de remoção:

```
public function drawTriangles(vertices:Vector.<Number>, indices:Vector.<int> = null,
uvtData:Vector.<Number> = null, culling:String = "none"):void
```

O parâmetro de remoção é um valor da classe de enumeração `TriangleCulling`: `TriangleCulling.NONE`, `TriangleCulling.POSITIVE` e `TriangleCulling.NEGATIVE`. Esses valores dependem da direção do caminho do triângulo que define a superfície do objeto. A API do ActionScript para determinar a remoção pressupõe que todos os triângulos voltados para fora de uma forma 3D sejam desenhados com a mesma direção de caminho. Uma vez que uma face do triângulo é virada, a direção do caminho também muda. Nesse ponto, o triângulo pode ser removido (excluído da renderização).

Portanto, um valor `POSITIVE` de`TriangleCulling` remove triângulos com direção de caminho positiva (no sentido horário). Um valor `NEGATIVE` de `TriangleCulling` remove triângulos com uma direção de caminho negativa (no sentido anti-horário). No caso de um cubo, enquanto as superfícies voltadas para frente têm uma direção de caminho positiva, as superfícies traseiras têm uma direção de caminho negativa:



Um cubo "aberto" para mostrar a direção do caminho. Quando o cubo está "fechado", a direção do caminho da parte traseira é invertida.

Para ver como funciona a remoção, comece com o exemplo anterior de "Mapeamento UV" na página 521, defina o parâmetro de remoção do método `drawTriangles()` como `TriangleCulling.NEGATIVE`:

```
container.graphics.drawTriangles(vertices, indices, uvtData, TriangleCulling.NEGATIVE);
```

Observe que o lado "de trás" da imagem não é renderizado porque o objeto gira.

Capítulo 24: Trabalho com vídeo

Os vídeos criados com o Adobe Flash são uma das tecnologias de destaque na Internet. No entanto, a apresentação tradicional do vídeo, em uma tela retangular com uma barra de progresso e botões de controle na parte inferior, é apenas um dos usos possíveis do vídeo. Por meio do ActionScript, você tem acesso e controle ajustados sobre o carregamento, a apresentação e a reprodução de vídeos.

Noções básicas sobre vídeo

Introdução ao trabalho com vídeo

Um recurso importante do Adobe® Flash® Player e do Adobe® AIR™ é a capacidade de exibir e manipular informações de vídeo com o ActionScript da mesma forma que você pode manipular outros conteúdos visuais, como imagens, animação, texto e assim por diante.

Ao criar um arquivo FLV (Flash Video) no Adobe Flash CS4 Professional, você tem a opção de selecionar uma capa que inclui controles de reprodução comuns. Todavia, não há motivo para se limitar às opções disponíveis. Usando o ActionScript, você tem controle ajustado sobre o carregamento, a exibição e a reprodução de vídeo, o que significa que é possível criar sua própria capa de player de vídeo ou usar o vídeo de qualquer maneira menos tradicional que você queira.

O trabalho com vídeo no ActionScript envolve o trabalho com uma combinação de várias classes:

- Classe Video: a caixa de conteúdo em vídeo propriamente dita no Palco é uma ocorrência da classe Video. A classe Video é um objeto de exibição, por isso pode ser manipulada usando as mesmas técnicas aplicáveis a outros objetos de exibição, como posicionamento, aplicação de transformações, aplicação de filtros e modos de mesclagem, entre outras.
- Classe NetStream: quando você está carregando um arquivo de vídeo a ser controlado pelo ActionScript, uma ocorrência de NetStream representa a origem do conteúdo do vídeo — neste caso, um fluxo de dados de vídeo. O uso de uma ocorrência de NetStream também inclui o uso de um objeto NetConnection, que é a conexão com o arquivo de vídeo — como o túnel pelo qual é carregado o conteúdo do vídeo.
- Classe Camera: quando você está trabalhando com dados de vídeo de uma câmera conectada ao computador do usuário, uma ocorrência de Camera representa a origem do conteúdo do vídeo — a câmera do usuário e os dados de vídeo disponibilizados por ela.

Quando você carrega vídeo externo, pode carregar o arquivo de um servidor Web padrão para download progressivo ou trabalhar com vídeo de fluxo contínuo entregue por um servidor especializado, como o Flash® Media Server da Adobe.

Tarefas comuns de vídeo

Este capítulo descreve as seguintes tarefas relacionadas a vídeo que provavelmente você desejará executar:

- Exibição e controle de vídeo na tela
- Carregamento de arquivos de vídeo externos
- Controle da reprodução de vídeo
- Uso de tela cheia

- Manipulação de metadados e informações de ponto de sinalização em um arquivo de vídeo
- Captura e exibição da entrada de vídeo da câmera de um usuário

Conceitos e termos importantes

- Ponto de sinalização: um marcador que pode ser colocado em um momento específico de um arquivo de vídeo, por exemplo, para funcionar como um marcador que localiza esse ponto no tempo ou para fornecer dados adicionais associados ao momento em questão.
- Codificação: o processo de pegar dados de vídeo em um formato e convertê-los em outro formato de dados de vídeo; por exemplo, você pode pegar um vídeo de uma origem de alta resolução e convertê-lo em um formato adequado para disponibilização na Internet.
- Quadro: um único segmento de informações de vídeo; cada quadro é como uma imagem estática que representa um instantâneo de um momento. A execução de quadros em seqüência em alta velocidade cria a ilusão de movimento.
- Quadro-chave: um quadro de vídeo que contém todas as informações do quadro. Outros quadros que vêm após um quadro-chave contêm apenas informações sobre no quê eles são diferentes do quadro-chave, e não todas as informações do quadro inteiro.
- Metadados: informações sobre um arquivo de vídeo que são incorporadas a ele e recuperadas quando o vídeo é carregado.
- Download progressivo: quando um arquivo de vídeo é disponibilizado de um servidor Web padrão, os dados de vídeo são carregados por meio de download progressivo, o que significa que as informações sobre o vídeo são carregadas em seqüência. Isso tem a vantagem de que a reprodução do vídeo pode começar antes de terminar o download do arquivo inteiro; no entanto, o download progressivo impede de você avançar para uma parte do vídeo que ainda não foi carregada.
- Streaming: como alternativa ao download progressivo, pode-se usar um servidor de vídeo especial para disponibilizar vídeo pela Internet usando uma técnica conhecida como streaming (também chamada de "true streaming"). Com o streaming, o computador do visualizador nunca baixa todo o vídeo de uma só vez. Para agilizar os tempos de download, em qualquer momento o computador só precisa de uma parte de todas as informações do vídeo. Como um servidor especial controla a entrega do conteúdo do vídeo, qualquer parte do vídeo pode ser acessada a qualquer momento, e não é necessário aguardar o término do download para acessá-lo.

Teste dos exemplos do capítulo

Ao trabalhar em um capítulo, talvez você queira testar algumas listagens de código de exemplo sozinho. Como este capítulo aborda o trabalho com vídeo no ActionScript, muitas das listagens de código aqui mencionadas envolvem o trabalho com um objeto de vídeo, que pode ser um objeto criado e colocado no Palco do Flash ou um objeto que tenha sido criado usando o ActionScript. O teste de uma amostra envolverá a exibição do resultado no Flash Player ou no AIR para ver os efeitos do código no vídeo.

A maioria das listagens de código de exemplo manipulam um objeto Video sem criá-lo explicitamente. Para testar essas listagens de código deste capítulo:

1. Crie um documento do Flash vazio.
2. Selecione um quadro-chave na Linha de tempo.
3. Abra o painel Ações e copie a listagem de código no painel Script.
4. Se necessário, abra o painel Biblioteca.
5. No menu do painel Biblioteca, clique em Novo vídeo.

6 Na caixa de diálogo Propriedades do vídeo, digite um nome para o símbolo do novo vídeo e clique em Vídeo (controlado pelo ActionScript) no campo Tipo. Clique em OK para criar o símbolo Video.

7 Arraste uma ocorrência do símbolo do seu vídeo do painel Biblioteca até o Palco.

8 Com a ocorrência do vídeo selecionada, no Inspetor de propriedades, dê a ela um nome de ocorrência. O nome deve corresponder ao nome usado para a ocorrência de Video na listagem de código de exemplo. Por exemplo, se a listagem de código manipula um objeto Video chamado `vid`, você deve nomear a ocorrência de Palco igualmente como `vid`.

9 Execute o programa usando Controlar > Testar filme.

Na tela, você verá os resultados do código manipulando o vídeo conforme especificado na listagem de código.

Algumas listagens de código de exemplo deste capítulo incluem uma definição de classe além do código de exemplo. Nessas listagens, além das etapas anteriores e antes de testar o SWF, você precisará criar a classe usada no exemplo. Para criar uma classe definida em uma listagem de código de exemplo:

1 Verifique se você salvou o arquivo FLA que será utilizado no teste.

2 No menu principal, clique em Arquivo > Novo.

3 Na seção Tipo da caixa de diálogo Novo documento, clique em Arquivo do ActionScript. Clique em OK para criar o novo arquivo do ActionScript.

4 Copie o código da definição de classe do exemplo para o documento do ActionScript.

5 No menu principal, clique em Arquivo > Salvar. Salve o arquivo no mesmo diretório que o documento do Flash. O nome do arquivo deve corresponder ao nome da classe na listagem de código. Por exemplo, se a listagem de código define uma classe chamada “VideoTest”, salve o arquivo do ActionScript como “VideoTest.as”.

6 Volte ao documento do Flash.

7 Execute o programa usando Controlar > Testar filme.

Os resultados do exemplo serão exibidos na tela.

Outras técnicas para testar listagens de código de exemplo são explicadas mais detalhadamente em “Teste de listagens de código de exemplo dos capítulos” na página 36.

Noções básicas sobre formatos de vídeo

Além do formato de vídeo Adobe FLV, o Flash Player e o Adobe AIR dão suporte a vídeo e áudio codificados em H.264 e HE-AAC a partir de formatos de arquivo com o padrão MPEG-4. Esses formatos transmitem vídeos de alta qualidade com taxas de bits mais baixas. Os desenvolvedores podem utilizar ferramentas padrão do setor, como o Adobe Premiere Pro e o Adobe After Effects, para criar e disponibilizar conteúdos em vídeo interessantes.

Tipo	Formato	Recipiente
Vídeo	H.264	MPEG-4: MP4, M4V, F4V, 3GPP
Vídeo	Arquivo FLV	Sorenson Spark
Vídeo	Arquivo FLV	ON2 VP6
Áudio	AAC+ / HE-AAC / AAC v1 / AAC v2	MPEG-4:MP4, M4V, F4V, 3GPP

Tipo	Formato	Recipiente
Áudio	Mp3	Mp3
Áudio	Nellymoser	Arquivo FLV
Áudio	Speex	Arquivo FLV

Compatibilidade do Flash Player e do AIR com arquivos de vídeo codificados

O Flash Player 7 oferece suporte a arquivos FLV codificados com o codec de vídeo Sorenson™ Spark™. O Flash Player 8 dá suporte a arquivos FLV codificados com o codificador Sorenson Spark ou On2 VP6 no Flash Professional 8. O codec de vídeo On2 VP6 oferece suporte a um canal alfa.

O Flash Player 9.0.115.0 e versões posteriores dão suporte a arquivos derivados do formato de recipiente MPEG-4 padrão. Esses arquivos incluem F4V, MP4, M4A, MOV, MP4V, 3GP e 3G2, se contêm vídeo H.264 ou áudio codificado em HE-AAC v2 ou ambos. O H.264 oferece vídeo de qualidade superior com taxas de bits inferiores se comparadas com o mesmo perfil de codificação no Sorenson ou On2. O HE-AAC v2 é uma extensão do AAC, um formato de áudio padrão definido no padrão de vídeo MPEG-4. O HE-AAC v2 usa as técnicas SBR (Spectral Band Replication) e PS (Parametric Stereo) para aumentar a eficiência da codificação em taxas de bits baixas.

A tabela a seguir lista os codecs suportados. Ela também mostra o formato de arquivo SWF correspondente e as versões do Flash Player e do AIR necessárias para reproduzi-los:

Codec	Versão do formato de arquivo SWF (versão de publicação mais antiga suportada)	Flash Player e AIR (versão mais antiga necessária para reprodução)
Sorenson Spark	6	FP 6, Flash Lite 3
On2 VP6	6	FP 8, Flash Lite 3. Apenas o Flash Player 8 e versões posteriores oferecem suporte à publicação e à reprodução de vídeo On2 VP6.
H.264 (MPEG-4 Parte 10)	9	FP 9 atualização 3, AIR 1.0
ADPCM	6	FP 6, Flash Lite 3
Mp3	6	FP 6, Flash Lite 3
AAC (MPEG-4 Parte 3)	9	FP 9 atualização 3, AIR 1.0
Speex (áudio)	10	FP 10, AIR 1.5
Nellymoser	6	FP 6

Noções básicas sobre os formatos de arquivo de vídeo Adobe F4V e FLV

A Adobe oferece os formatos de arquivo de vídeo F4V e FLV para transmitir conteúdo para o Flash Player e o AIR. Para ver uma descrição completa desses formatos de arquivo de vídeo, consulte www.adobe.com/go/video_file_format_br.

O formato de arquivo de vídeo F4V

A partir da atualização 3 do Flash Player (9.0.115.0) e do AIR 1.0, o Flash Player e o AIR dão suporte ao formato de vídeo Adobe F4V, que é baseado no formato ISO MP4. Subconjuntos do formato dão suporte a diferentes recursos. O Flash Player espera que um arquivo F4V válido comece com uma das seguintes caixas de nível superior:

- ftyp

  A caixa ftyp identifica os recursos para os quais um programa deve oferecer suporte para executar um determinado formato de arquivo.

- moov

  A caixa moov é o cabeçalho de um arquivo F4V. Ela contém uma ou mais outras caixas que, por sua vez, contêm outras caixas que definem a estrutura dos dados F4V. Um arquivo F4V deve conter somente uma caixa moov.

- mdat

  Uma caixa mdat contém a carga de dados do arquivo F4V. Um arquivo FV contém apenas uma caixa mdat. Também deve haver uma caixa moov no arquivo porque a caixa mdat não pode ser compreendida por si só.

Os arquivos F4V dão suporte a inteiros multibyte na ordem de bytes big-endian, pela qual o byte mais importante ocorre primeiro, no endereço mais baixo.

O formato de arquivo de vídeo FLV

O formato de arquivo Adobe FLV contém dados de áudio e vídeo codificados para entrega pelo Flash Player. É possível usar um codificador, como o Adobe Media Encoder ou o Sorenson™ Squeeze, para converter um arquivo de vídeo do QuickTime ou do Windows Media em um arquivo FLV.

Nota: *Você pode criar arquivos FLV importando vídeo para o Flash e o exportando como um arquivo FLV. Você pode usar o plug-in de exportação de FLV para exportar arquivos FLV de aplicativos de edição de vídeos suportados. Para carregar arquivos FLV de um servidor Web, registre a extensão do nome de arquivo e o tipo MIME no servidor Web. Consulte a documentação do seu servidor Web. O tipo MIME de arquivos FLV é* `video/x-flv`*. Para obter mais informações, consulte "Sobre a configuração de arquivos FLV para hospedagem em um servidor" na página 562.*

Para obter mais informações sobre arquivos FLV, consulte "Tópicos avançados sobre arquivos FLV" na página 562.

Vídeo externo versus incorporado

O uso de arquivos de vídeo externos oferece determinados recursos que não estão disponíveis quando você usa vídeo importado:

- Clipes de vídeo mais longos podem ser usados no seu aplicativo sem diminuir a velocidade da reprodução. Arquivos de vídeo externos usam a memória cache, o que significa que arquivos grandes são armazenados em pequenas partes e acessados dinamicamente. Por esse motivo, os arquivos F4V e FLV externos exigem menos memória do que os arquivos de vídeo incorporados.
- Um arquivo de vídeo externo pode ter uma taxa de quadros diferente do que o arquivo SWF no qual ele é reproduzido. Por exemplo, você pode definir a taxa de quadros do arquivo SWF como 30 quadros por segundo (fps) e a taxa de quadros de vídeo como 21 fps. Esta configuração lhe dá mais controle sobre o vídeo do que o vídeo incorporado e assegura a reprodução contínua. Ela também permite reproduzir arquivos de vídeo com taxas de quadro diferentes sem precisar alterar o conteúdo do arquivo SWF existente.
- Com arquivos de vídeo externos, a reprodução do conteúdo SWF não é interrompida enquanto o arquivo de vídeo está sendo carregado. Às vezes, os arquivos de vídeo importados podem interromper a reprodução de documentos para executar certas funções, como acessar uma unidade de CD-ROM. Os arquivos de vídeo podem executar funções independentemente do conteúdo SWF sem interromper a reprodução.

- É mais fácil legendar o conteúdo em vídeo com arquivos FLV externos porque você pode acessar os metadados do vídeo usando manipuladores de eventos.

Noções básicas sobre a classe Video

A classe Video permite exibir vídeo de fluxo contínuo ao vivo em um aplicativo sem incorporá-lo ao seu arquivo SWF. É possível capturar e reproduzir vídeo ao vivo usando o método `Camera.getCamera()`. Você também pode usar a classe Video para reproduzir arquivos de vídeo por HTTP ou a partir do sistema de arquivos local. Há várias maneiras diferentes de usar a classe Video em seus projetos:

- Carregue um arquivo de vídeo dinamicamente usando as classes NetConnection e NetStream e exiba o vídeo em um objeto Video.
- Capture a entrada da câmera do usuário. Para obter mais informações, consulte "Captura da entrada da câmera" na página 556.
- Use o componente FLVPlayback.

Nota: *As ocorrências de um objeto Video no Palco são ocorrências da classe Video.*

Embora a classe Video esteja no pacote flash.media, ela herda da classe flash.display.DisplayObject. Portanto, toda a funcionalidade do objeto de exibição, como transformações de matriz e filtros, também se aplica a ocorrências de Video.

Para obter mais informações, consulte "Manipulação de objetos de exibição" na página 293, "Trabalho com geometria" na página 342 e "Filtro de objetos de exibição" na página 354.

Carregamento de arquivos de vídeo

O processo de carregar vídeos usando as classes NetStream e NetConnection inclui várias etapas:

1. Crie um objeto NetConnection. Se você estiver se conectando a um arquivo de vídeo local ou a um arquivo que não está usando um servidor, como o Flash Media Server 2 da Adobe, passe `null` para o método `connect()` para reproduzir arquivos de vídeo de um endereço HTTP ou de uma unidade local. Se você estiver se conectando a um servidor, defina o parâmetro como o URI do aplicativo que contém o arquivo de vídeo no servidor.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);
```

2. Crie um objeto NetStream que use um objeto NetConnection como parâmetro e especifique o arquivo de vídeo que você deseja carregar. O seguinte snippet conecta um objeto NetStream à ocorrência especificada de NetConnection e carrega um arquivo de vídeo chamado video.mp4 no mesmo diretório do arquivo SWF:

```
var ns:NetStream = new NetStream(nc);
ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);
ns.play("video.mp4");
function asyncErrorHandler(event:AsyncErrorEvent):void
{
    // ignore error
}
```

3. Crie um novo objeto Video e anexe o objeto NetStream criado anteriormente usando o método `attachNetStream()` da classe Video. Em seguida, adicione o objeto de vídeo à lista de exibição usando o método `addChild()`, como visto neste snippet:

```
var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

À medida que o Flash Player executa esse código, ele tenta carregar o arquivo de vídeo video.mp4 do mesmo diretório em que está o seu arquivo SWF.

Controle da reprodução de vídeo

A classe NetStream oferece quatro principais métodos para controlar a reprodução de vídeos:

`pause()`: Pausa a reprodução de um fluxo de vídeo. Se o vídeo já estiver pausado, chamar este método não terá efeito.

resume(): Reinicia a reprodução de um fluxo de vídeo pausado. Se o vídeo já estiver em reprodução, chamar este método não terá efeito.

seek(): Busca o quadro-chave mais próximo do local especificado (um deslocamento, em segundos, a partir do início do fluxo).

togglePause(): Pausa ou reinicia a reprodução de um fluxo.

Nota: *Não há um método `stop()`. Para interromper um fluxo, você deve pausar a reprodução e fazer a busca até o início do fluxo de vídeo.*

Nota: *O método `play()` não reinicia a reprodução; ele é usado para carregar arquivos de vídeo.*

O exemplo a seguir demonstra como controlar um vídeo usando vários botões diferentes. Para executar o exemplo, crie um novo documento e adicione quatro ocorrências de botão ao seu espaço de trabalho (`pauseBtn`, `playBtn`, `stopBtn` e `togglePauseBtn`):

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);
ns.play("video.flv");
function asyncErrorHandler(event:AsyncErrorEvent):void
{
    // ignore error
}

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

pauseBtn.addEventListener(MouseEvent.CLICK, pauseHandler);
playBtn.addEventListener(MouseEvent.CLICK, playHandler);
stopBtn.addEventListener(MouseEvent.CLICK, stopHandler);
togglePauseBtn.addEventListener(MouseEvent.CLICK, togglePauseHandler);

function pauseHandler(event:MouseEvent):void
{
    ns.pause();
}
function playHandler(event:MouseEvent):void
{
    ns.resume();
}
function stopHandler(event:MouseEvent):void
{
    // Pause the stream and move the playhead back to
    // the beginning of the stream.
    ns.pause();
    ns.seek(0);
}
function togglePauseHandler(event:MouseEvent):void
{
    ns.togglePause();
}
```

Quando você clica na ocorrência do botão `pauseBtn` durante a reprodução do vídeo, o arquivo de vídeo é pausado. Se o vídeo já estiver pausado, clicar neste botão não terá efeito. Clicar na ocorrência do botão `playBtn` reinicia a reprodução do vídeo se a reprodução foi pausada anteriormente; caso contrário, o botão não terá efeito se o vídeo já estiver sendo reproduzido.

Detecção do final de um fluxo de vídeo

Para monitorar o começo ou o final de um fluxo de vídeo, você precisa adicionar um ouvinte de eventos à ocorrência de NetStream para monitorar o evento `netStatus`. O seguinte código demonstra como monitorar os diversos códigos durante a reprodução do vídeo:

```
ns.addEventListener(NetStatusEvent.NET_STATUS, statusHandler);
function statusHandler(event:NetStatusEvent):void
{
    trace(event.info.code)
}
```

O código anterior gera a seguinte saída:

```
NetStream.Play.Start
NetStream.Buffer.Empty
NetStream.Buffer.Full
NetStream.Buffer.Empty
NetStream.Buffer.Full
NetStream.Buffer.Empty
NetStream.Buffer.Full
NetStream.Buffer.Flush
NetStream.Play.Stop
NetStream.Buffer.Empty
NetStream.Buffer.Flush
```

Os dois códigos que você deseja monitorar especificamente são "NetStream.Play.Start" e "NetStream.Play.Stop", que sinalizam o início e o fim da reprodução do vídeo. O seguinte snippet usa uma instrução de opção para filtrar esses dois códigos e rastrear uma mensagem:

```
function statusHandler(event:NetStatusEvent):void
{
    switch (event.info.code)
    {
        case "NetStream.Play.Start":
            trace("Start [" + ns.time.toFixed(3) + " seconds]");
            break;
        case "NetStream.Play.Stop":
            trace("Stop [" + ns.time.toFixed(3) + " seconds]");
            break;
    }
}
```

Ao monitorar o evento `netStatus` (`NetStatusEvent.NET_STATUS`), você pode criar um player de vídeo que carregue o próximo vídeo de uma lista de exibição depois que a reprodução do vídeo atual for concluída.

Uso de tela cheia

O Flash Player e o AIR permitem dimensionar o vídeo em tela cheia através do ActionScript 3.0. As seguintes restrições são aplicáveis quando o Flash Player reproduz conteúdo em uma página HTML:

- As tags `object` e `embed` da página HTML que contém o arquivo SWF devem incluir o parâmetro `allowFullScreen` definido como true
- O modo de tela cheia é iniciado em resposta a um clique do mouse ou a um pressionamento de tecla
- Se você definir o Modo de janela (wmode no HTML) como Opaco sem janela (opaco) ou Transparente sem janela (transparente), a janela da tela cheia será sempre opaca

Essas restrições não se aplicam a conteúdo SWF em execução no Flash Player independente ou no AIR. O AIR dá suporte a um modo de tela cheia interativa que permite entrada do teclado.

Para conteúdo do AIR executado no modo de tela cheia, as opções de protetor de tela e de economia de energia do sistema são desativadas durante a reprodução até que a entrada de vídeo seja interrompida ou que o usuário saia do modo de tela cheia.

Ativação do modo de tela cheia do Flash Player em um navegador

Para que você possa implementar o modo de tela cheia do Flash Player em um navegador, ative-o através do modelo Publish do seu aplicativo. Os modelos que permitem o modo de tela cheia incluem as tags `<object>` e `<embed>` que contêm um parâmetro `allowFullScreen`. O exemplo a seguir mostra o parâmetro `allowFullScreen` em uma tag `<embed>`.

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
    id="fullScreen" width="100%" height="100%"
    codebase="http://fpdownload.macromedia.com/get/flashplayer/current/swflash.cab">
    ...
    <param name="allowFullScreen" value="true" />
    <embed src="fullScreen.swf" allowFullScreen="true" quality="high" bgcolor="#869ca7"
        width="100%" height="100%" name="fullScreen" align="middle"
        play="true"
        loop="false"
        quality="high"
        allowScriptAccess="sameDomain"
        type="application/x-shockwave-flash"
        pluginspage="http://www.adobe.com/go/getflashplayer">
    </embed>
    ...
</object>
```

No Flash, selecione Arquivo -> Configurações de publicação e, na aba HTML da caixa de diálogo Configurações de publicação, selecione o modelo Somente Flash - Permitir tela cheia.

No Flex, verifique se o modelo HTML inclui as tags `<object>` e `<embed>` que dão suporte a tela cheia.

Inicialização do modo de tela cheia

Para o Flash Player em um navegador, você inicia o modo de tela cheia para vídeo em resposta a um clique do mouse ou a um pressionamento de tecla. Por exemplo, você pode iniciar o modo de tela cheia quando o usuário clica em um botão chamado Tela cheia ou seleciona um comando Tela cheia em um menu de contexto. Para responder ao usuário, adicione um ouvinte de eventos ao objeto no qual ocorre a ação. O seguinte código adiciona um ouvinte de eventos a um botão no qual o usuário clica para entrar no modo de tela cheia:

```
var fullScreenButton:Button = new Button();
fullScreenButton.label = "Full Screen";
addChild(fullScreenButton);
fullScreenButton.addEventListener(MouseEvent.CLICK, fullScreenButtonHandler);

function fullScreenButtonHandler(event:MouseEvent)
{
    // add code here to enter full-screen mode

}
```

Inicie o modo de tela cheia definindo a propriedade `Stage.displayState` como `StageDisplayState.FULL_SCREEN`. O exemplo a seguir inicia o modo de tela cheia.

```
function fullScreenButtonHandler(event:MouseEvent)
{
    stage.displayState = StageDisplayState.FULL_SCREEN;
}
```

Este código dimensiona todo o palco em tela cheia, sendo que o vídeo é dimensionado proporcionalmente ao espaço que ocupa no palco.

A propriedade `fullScreenSourceRect` permite especificar uma determinada área do palco a ser dimensionada para tela cheia. Primeiro, defina o retângulo que você deseja dimensionar para tela cheia. Depois, atribua-o à propriedade `Stage.fullScreenSourceRect`. Esta versão da função `fullScreenButtonHandler()` adiciona duas linhas a mais de código que dimensionam somente o vídeo para tela cheia.

```
private function fullScreenButtonHandler(event:MouseEvent)
{
    var screenRectangle:Rectangle = new Rectangle(video.x, video.y, video.width, video.height);
    stage.fullScreenSourceRect = screenRectangle;
    stage.displayState = StageDisplayState.FULL_SCREEN;
}
```

Embora este exemplo chame um manipulador de eventos em resposta a um clique do mouse, a técnica de usar a tela cheia é a mesma para o Flash Player e o AIR. Defina o retângulo que você deseja dimensionar e configure a propriedade `Stage.displayState`. Para obter mais informações, consulte a Referência de componentes e linguagem do ActionScript 3.0.

O exemplo completo, mostrado a seguir, adiciona um código que cria a conexão e o objeto NetStream para o vídeo e começa a reproduzi-lo.

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    import flash.media.Video;
    import flash.display.StageDisplayState;
    import fl.controls.Button;
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    import flash.events.FullScreenEvent;
    import flash.geom.Rectangle;

    public class FullScreenVideoExample extends Sprite
    {
        var fullScreenButton:Button = new Button();
        var video:Video = new Video();

        public function FullScreenVideoExample()
        {
            var videoConnection:NetConnection = new NetConnection();
            videoConnection.connect(null);

            var videoStream:NetStream = new NetStream(videoConnection);
            videoStream.client = this;

            addChild(video);

            video.attachNetStream(videoStream);

            videoStream.play("http://www.helpexamples.com/flash/video/water.flv");

            fullScreenButton.x = 100;
```

```
            fullScreenButton.y = 270;
            fullScreenButton.label = "Full Screen";
            addChild(fullScreenButton);
            fullScreenButton.addEventListener(MouseEvent.CLICK, fullScreenButtonHandler);
        }

        private function fullScreenButtonHandler(event:MouseEvent)
        {
            var screenRectangle:Rectangle = new Rectangle(video.x, video.y, video.width,
video.height);
            stage.fullScreenSourceRect = screenRectangle;
            stage.displayState = StageDisplayState.FULL_SCREEN;
        }

        public function onMetaData(infoObject:Object):void
        {
            // stub for callback function
        }
    }
}
```

A função `onMetaData()` é uma função de retorno de chamada que manipula os metadados do vídeo, se houver. Uma função de retorno de chamada é a função que o tempo de execução chama em resposta a algum tipo de ocorrência ou evento. Neste exemplo, a função `onMetaData()` é um stub que atende ao requisito de fornecer a função. Para obter mais informações, consulte "Criação de métodos de retorno de chamada para metadados e pontos de sinalização" na página 542

fullScreenHeight e fullScreenWidth

As propriedades `Stage.fullScreenHeight` e `Stage.fullScreenWidth` retornam a altura e a largura do monitor que é utilizado quando se passa para o tamanho de tela cheia, caso a entrada nesse estado ocorra imediatamente. Esses valores poderão estar incorretos se o usuário tiver a oportunidade de mudar o navegador de um monitor para outro depois que você recuperá-los, mas antes de passar para o modo de tela cheia. Se você recuperar esses valores em um manipulador de eventos que define `Stage.displayState` como `StageDisplayState.FULL_SCREEN`, os valores estarão corretos.

Se o usuário tiver vários monitores, o monitor que essas propriedades refletirão será aquele que contém a maior parte do palco no momento.

Para obter mais informações, consulte `Stage.fullScreenHeight` e `Stage.fullScreenWidth` na Referência de componentes e linguagem do ActionScript 3.0.

Suporte para teclado

Quando o Flash Player é executado em um navegador, ele restringe a entrada por teclado no modo de tela cheia. As teclas aceitáveis incluem atalhos de teclado que encerram o modo de tela cheia e teclas que não são impressas, como as teclas de seta, espaço, Shift e Tab. Os atalhos de teclado que encerram o modo de tela cheia são os seguintes: Escape (Windows e Mac), Ctrl+W (Windows), Command+W (Mac) e Alt+F4.

Encerramento do modo de tela cheia

Um usuário pode encerrar o modo de tela cheia usando um dos atalhos de teclado, como a tecla Escape. Para obter mais informações sobre esses atalhos de teclado, consulte "Suporte para teclado" na página 538. É possível encerrar o modo de tela cheia no ActionScript configurando a propriedade `Stage.diplayState` como `StageDisplayState.NORMAL`. O código do exemplo a seguir encerra o modo de tela cheia quando ocorre o evento de status de rede NetStream.Play.Stop.

```
videoStream.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);

private function netStatusHandler(event:NetStatusEvent)
{
    if(event.info.code == "NetStream.Play.Stop")
        stage.displayState = StageDisplayState.NORMAL;
}
```

Desativação do modo de tela cheia do Flash Player

É possível desativar o suporte de tela cheia para todos os arquivos de vídeo adicionando-se um parâmetro ao arquivo de configuração do Flash Player (mms.cfg). Este tipo de operação costuma ser reservado para um administrador de TI. O arquivo mms.cfg pode ser encontrado nos seguintes locais:

- Windows XP, Vista -- C:\WINDOWS\System32\Macromed\Flash
- Mac -- \Application Support\Macromedia

O arquivo mms.cfg consiste em um arquivo de texto UTF-8 cujo formato é uma série de pares nome=valor separados por retornos de carro. O Flash Player assume o valor padrão para um parâmetro se ele não está definido no arquivo. Para desativar o modo de tela cheia do Flash Player, adicione o seguinte parâmetro ao arquivo mms.cfg:

```
FullScreenDisable=1
```

Nota: *As configurações contidas no arquivo mms.cfg substituem as configurações do usuário e este não pode alterá-las através do Gerenciador de configurações. Além disso, o modo de exibição do usuário do Gerenciador de configurações não reflete as configurações do arquivo mms.cfg.*

Aceleração de hardware

Quando você define a propriedade `Stage.fullScreenSourceRect` para dimensionar um segmento retangular do palco para o modo de tela cheia, o Flash Player ou o AIR usa aceleração de hardware caso ela esteja disponível e ativada. O tempo de execução usa o adaptador de vídeo do computador para acelerar o dimensionamento do vídeo ou uma parte do palco para o tamanho de tela cheia. A aceleração de hardware exige o Microsoft® DirectX 9 com 128 MB de VRAM no Windows e OpenGL para Apple® Macintosh®, Mac OS X v10.2 ou superior.

Para aproveitar as vantagens de todos os recursos da aceleração de hardware no Flash Player, ative-a na caixa de diálogo Configurações do Flash Player. Para carregar a caixa de diálogo, clique com o botão direito do mouse (Windows) ou pressione a tecla Ctrl enquanto clica (Mac) dentro do conteúdo Flash Player no navegador. Selecione a aba Exibir, que é a primeira, e clique na caixa de seleção: Habilitar aceleração de hardware.

Modos de janela direto e de composição GPU

O Flash Player 10 traz dois novos modos de janela, direto e composição GPU, que você pode implementar através das configurações de publicação do Flash. Esses modos não são suportados no AIR. Para aproveitar as vantagens desses modos, você deve ativar a aceleração de hardware para o Flash Player.

O modo direto usa o caminho mais rápido e direto para enviar gráficos para a tela, o que é vantajoso para a reprodução de vídeos.

A Composição GPU usa a unidade de processamento gráfico da placa de vídeo para acelerar a composição. A composição de vídeo é o processo de dispor várias imagens em camadas para criar uma única imagem de vídeo. Quando a composição é acelerada com a GPU, ela pode melhorar o desempenho de conversão YUV, correção de cores, rotação ou dimensionamento e mesclagem. Conversão YUV refere-se à conversão de cores de sinais analógicos compostos, que são usados para transmissão, no modelo de cores RGB (vermelho, verde, azul) usado por monitores e câmeras de vídeo. O uso da GPU para acelerar a composição reduz as demandas de memória e de computação que, de outro modo, seriam atribuídas à CPU. Ele também resulta em uma reprodução mais contínua para vídeo com definição padrão.

Seja cauteloso na implementação desses modos de janela. O uso da composição GPU pode ser dispendioso para a memória e os recursos da CPU. Se não for possível executar algumas operações (como modos de mesclagem, filtragem, corte ou mascaramento) na GPU, elas serão feitas pelo software. A Adobe recomenda que você se restrinja a um arquivo SWF por página HTML quando usar esses modos e que não ative esses modos para banners. O recurso Testar filme do Flash não utiliza aceleração de hardware, mas você pode usá-la através da opção Publicar visualização.

É inútil configurar uma taxa de quadros superior a 60 no arquivo SWF, que é a taxa máxima de atualização da tela. Configurar a taxa de quadros com um valor entre 50 e 55 viabiliza quadros descartados, que podem ocorrer por vários motivos de tempos em tempos.

O uso do modo direto exige o Microsoft® DirectX 9 com 128 MB de VRAM no Windows e OpenGL para Apple® Macintosh®, Mac OS X v10.2 ou superior. A composição GPU exige o Microsoft® DirectX 9 e suporte para Pixel Shader 2.0 no Windows com 128 MB de VRAM. No Apple Macintosh e no Linux, a composição GPU exige o OpenGL 1.5 e várias extensões OpenGL (objeto framebuffer, multitextura, objetos Shader, linguagem de sombreamento, sombreador de fragmentos).

É possível ativar os modos de aceleração `direto` e `gpu` por arquivo SWF na caixa de diálogo Configurações de publicação do Flash usando o menu Aceleração de hardware da aba Flash. Se você clicar em Nenhum, o modo de janela voltará para `padrão`, `transparente` ou `opaco`, conforme especificado pela configuração Modo de janela na aba HTML. O efeito da configuração Aceleração de hardware é definir o bit de cabeçalho estendido SWF, como segue:

```
swfFlagsUseAcceleratedBlit = 0x00000020, // this SWF enables direct acceleration
swfFlagsUseHardwareGPU = 0x00000040, // this SWF enables hardware gpu acceleration
```

Esta configuração afeta apenas a reprodução em projetores Windows e Mac e no Flash Player autônomo. Se o destino do player é uma página HTML, o Flash define o parâmetro HTML `wmode` como direto ou gpu.

Arquivos de vídeo em fluxo contínuo

Para reproduzir arquivos em fluxo contínuo no Flash Media Server, você pode usar as classes NetConnection e NetStream para se conectar a uma ocorrência do servidor remoto e reproduzir um fluxo especificado. Para especificar um servidor RTMP (Real-Time Messaging Protocol), passe a URL RTMP desejada, como "rtmp://localhost/appName/appInstance", para o método `NetConnection.connect()` em vez de passar `null`. Para reproduzir um determinado fluxo ao vivo ou gravado do Flash Media Server, passe um nome de identificação referente aos dados ao vivo publicados por `NetStream.publish()` ou um nome de arquivo gravado para reprodução ao método `NetStream.play()`. Para obter mais informações, consulte a documentação do Flash Media Server.

Noções básicas sobre pontos de sinalização

É possível incorporar pontos de sinalização a um arquivo de vídeo Adobe F4V ou FLV durante a codificação. Historicamente, os pontos de sinalização eram incorporados a filmes para dar ao projetista um sinal visual que indicava que o rolo de filme estava perto do fim. Nos formatos de vídeo Adobe F4V e FLV, um ponto de sinalização permite que você dê início a uma ou mais ações no aplicativo quando ele ocorrer no fluxo de vídeo.

Você pode usar vários tipos diferentes de pontos de sinalização com vídeos criados no Flash. Você pode usar o ActionScript para interagir com pontos de sinalização que incorpora a um arquivo de vídeo ao criá-lo.

- Pontos de sinalização de navegação: incorpore pontos de sinalização de navegação ao pacote de metadados e fluxo de vídeo quando codificar o arquivo de vídeo. Use pontos de sinalização de navegação para que os usuários possam fazer a busca até uma determinada parte de um arquivo.
- Pontos de sinalização de evento: incorpore pontos de sinalização de evento ao pacote de metadados e fluxo de vídeo quando codificar o arquivo de vídeo. Você pode criar um código para manipular os eventos que são acionados em pontos específicos durante a reprodução do vídeo.
- Pontos de sinalização do ActionScript: os pontos de sinalização do ActionScript só estão disponíveis para o componente FLVPlayback do Flash. Eles são pontos de sinalização externos criados e acessados com código ActionScript. Você pode criar um código para acionar esses pontos de sinalização em relação à reprodução do vídeo. Eles são menos precisos do que os pontos de sinalização incorporados (até um décimo de segundo), porque o player de vídeo os rastreia separadamente. Se você pretende criar um aplicativo em que os usuários poderão navegar até um ponto de sinalização, deve criar e incorporar pontos de sinalização quando codificar o arquivo em vez de usar pontos de sinalização do ActionScript. Você deve incorporar os pontos de sinalização ao arquivo FLV porque eles são mais precisos.

Os pontos de sinalização de navegação criam um quadro-chave no local especificado do ponto de sinalização, por isso você pode usar um código para deslocar o indicador de reprodução de um player de vídeo até esse local. É possível definir pontos específicos em um arquivo de vídeo no qual você deseja que os usuários façam uma busca. Por exemplo, o seu vídeo pode ter vários capítulos ou segmentos, e você pode controlá-lo incorporando pontos de sinalização de navegação ao arquivo de vídeo.

Para obter mais informações sobre como codificar arquivos de vídeo Adobe com pontos de sinalização, consulte "Incorporação de pontos de sinalização" em *Uso do Flash*.

É possível acessar parâmetros de ponto de sinalização criando código ActionScript. Os parâmetros de ponto de sinalização fazem parte do objeto de evento recebido pelo manipulador de retorno de chamada.

Para iniciar certas ações no seu código quando um arquivo FLV atingir um ponto de sinalização específico, use o manipulador de eventos `NetStream.onCuePoint`.

Para sincronizar uma ação para um ponto de sinalização em um arquivo de vídeo F4V, recupere os dados do ponto de sinalização usando as funções de retorno de chamada `onMetaData()` ou `onXMPData()` e acione o ponto de sinalização usando a classe Timer do ActionScript 3.0. Para obter mais informações sobre pontos de sinalização F4V, consulte "Uso de onXMPData()" na página 553.

Para obter mais informações sobre como manipular pontos de sinalização e metadados, consulte "Criação de métodos de retorno de chamada para metadados e pontos de sinalização" na página 542.

Criação de métodos de retorno de chamada para metadados e pontos de sinalização

É possível iniciar ações no seu aplicativo quando metadados específicos são recebidos pelo player ou quando se atingem determinados pontos de sinalização. Quando esses eventos ocorrem, você deve usar métodos de retorno de chamada específicos como manipuladores de eventos. A classe NetStream especifica os seguintes eventos de metadados, que podem ocorrer durante a reprodução: `onCuePoint` (somente em arquivos FLV), `onImageData`, `onMetaData`, `onPlayStatus`, `onTextData` e `onXMPData`.

Você deve criar métodos de retorno de chamada para esses manipuladores, caso contrário o Flash Player poderá gerar erros. Por exemplo, o código a seguir reproduz um arquivo FLV chamado video.flv na mesma pasta em que reside o arquivo SWF:

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);
ns.play("video.flv");
function asyncErrorHandler(event:AsyncErrorEvent):void
{
    trace(event.text);
}

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

O código anterior carrega um arquivo de vídeo local chamado video.flv e monitora o evento `asyncError` (`AsyncErrorEvent.ASYNC_ERROR`) a ser despachado. Esse evento é despachado quando o código assíncrono nativo gera uma exceção. Nesse caso, ele é despachado quando o arquivo de vídeo contém informações de metadados ou de ponto de sinalização e os ouvintes apropriados não foram definidos. O código anterior manipula o evento `asyncError` e ignora o erro se você não está interessado nas informações de metadados ou de ponto de sinalização do arquivo de vídeo. Se você tivesse um arquivo FLV com metadados e vários pontos de sinalização, a função `trace()` exibiria as seguintes mensagens de erro:

```
Error #2095: flash.net.NetStream was unable to invoke callback onMetaData.
Error #2095: flash.net.NetStream was unable to invoke callback onCuePoint.
Error #2095: flash.net.NetStream was unable to invoke callback onCuePoint.
Error #2095: flash.net.NetStream was unable to invoke callback onCuePoint.
```

Os erros ocorrem porque o objeto NetStream não conseguiu encontrar um método de retorno de chamada `onMetaData` ou `onCuePoint`. Existem diversas maneiras de definir esses métodos de retorno de chamada nos seus aplicativos:

Definir a propriedade client do objeto NetStream como Object

Ao definir a propriedade `client` como Object ou uma subclasse de NetStream, você pode redirecionar os métodos de retorno de chamada `onMetaData` e `onCuePoint` ou ignorá-los completamente. O exemplo a seguir demonstra como usar um Object vazio para ignorar os métodos de retorno de chamada sem monitorar o evento `asyncError`:

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var customClient:Object = new Object();

var ns:NetStream = new NetStream(nc);
ns.client = customClient;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

Para monitorar os métodos de retorno de chamada `onMetaData` ou `onCuePoint`, seria preciso definir métodos para manipular esses métodos de retorno de chamada, conforme mostrado no seguinte snippet:

```
var customClient:Object = new Object();
customClient.onMetaData = metaDataHandler;
function metaDataHandler(infoObject:Object):void
{
    trace("metadata");
}
```

O código anterior monitora o método de retorno de chamada `onMetaData` e chama o método `metaDataHandler()`, que rastreia uma string. Se o Flash Player encontrasse um ponto de sinalização, nenhum erro seria gerado, mesmo que não tenha sido definido um método de retorno de chamada `onCuePoint`.

Criar uma classe personalizada e definir métodos para manipular os métodos de retorno de chamada

O seguinte código define a propriedade `client` do objeto NetStream como uma classe personalizada, CustomClient, que define manipuladores para os métodos de retorno de chamada:

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = new CustomClient();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

Esta é a classe CustomClient:

```
package
{
    public class CustomClient
    {
        public function onMetaData(infoObject:Object):void
        {
            trace("metadata");
        }
    }
}
```

A classe CustomClient define um manipulador para o manipulador de retorno de chamada `onMetaData`. Se fosse encontrado um ponto de sinalização e o manipulador de retorno de chamada `onCuePoint` fosse chamado, seria despachado um evento `asyncError` (`AsyncErrorEvent.ASYNC_ERROR`) informando que flash.net.NetStream não pôde chamar o manipulador de retorno de chamada onCuePoint. Para evitar esse erro, seria necessário definir um método de retorno de chamada `onCuePoint` na classe CustomClient ou definir um manipulador de eventos para o evento `asyncError`.

Estender a classe NetStream e adicionar métodos para manipular os métodos de retorno de chamada

O código abaixo cria uma ocorrência da classe CustomNetStream, que é definida em uma listagem de código mais adiante:

```
var ns:CustomNetStream = new CustomNetStream();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

A seguinte listagem de código define a classe CustomNetStream que estende a classe NetStream, manipula a criação do objeto NetConnection necessário e manipula os métodos de manipulador de retorno de chamada `onMetaData` e `onCuePoint`:

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    public class CustomNetStream extends NetStream
    {
        private var nc:NetConnection;
        public function CustomNetStream()
        {
            nc = new NetConnection();
            nc.connect(null);
            super(nc);
        }
        public function onMetaData(infoObject:Object):void
        {
            trace("metadata");
        }
        public function onCuePoint(infoObject:Object):void
        {
            trace("cue point");
        }
    }
}
```

Para renomear os métodos `onMetaData()` e `onCuePoint()` na classe CustomNetStream, use o seguinte código:

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    public class CustomNetStream extends NetStream
    {
        private var nc:NetConnection;
        public var onMetaData:Function;
        public var onCuePoint:Function;
        public function CustomNetStream()
        {
            onMetaData = metaDataHandler;
            onCuePoint = cuePointHandler;
            nc = new NetConnection();
            nc.connect(null);
            super(nc);
        }
        private function metaDataHandler(infoObject:Object):void
        {
            trace("metadata");
        }
        private function cuePointHandler(infoObject:Object):void
        {
            trace("cue point");
        }
    }
}
```

Estender a classe NetStream e torná-la dinâmica

É possível estender a classe NetStream e tornar a subclasse dinâmica para que os manipuladores de retorno de chamada `onCuePoint` e `onMetaData` possam ser adicionados dinamicamente. Isso é demonstrado na listagem a seguir:

```
var ns:DynamicCustomNetStream = new DynamicCustomNetStream();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

Esta é a classe DynamicCustomNetStream:

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    public dynamic class DynamicCustomNetStream extends NetStream
    {
        private var nc:NetConnection;
        public function DynamicCustomNetStream()
        {
            nc = new NetConnection();
            nc.connect(null);
            super(nc);
        }
    }
}
```

Mesmo sem manipuladores para os manipuladores de retorno de chamada `onMetaData` e `onCuePoint`, não são gerados erros porque a classe DynamicCustomNetStream é dinâmica. Para definir métodos para os manipuladores de retorno de chamada `onMetaData` e `onCuePoint`, use o seguinte código:

```
var ns:DynamicCustomNetStream = new DynamicCustomNetStream();
ns.onMetaData = metaDataHandler;
ns.onCuePoint = cuePointHandler;
ns.play("http://www.helpexamples.com/flash/video/cuepoints.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

function metaDataHandler(infoObject:Object):void
{
    trace("metadata");
}
function cuePointHandler(infoObject:Object):void
{
    trace("cue point");
}
```

Definir a propriedade client do objeto NetStream como this

Quando se define a propriedade `client` como `this`, o aplicativo procura os métodos `onMetaData()` e `onCuePoint()` no escopo atual. Este exemplo demonstra isso:

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = this;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

Se os manipuladores de retorno de chamada `onMetaData` ou `onCuePoint` são chamados e não existem métodos para manipular o retorno de chamada, nenhum erro é gerado. Para manipular esses manipuladores de retorno de chamada, crie um método `onMetaData()` e `onCuePoint()` no seu código, como visto no snippet abaixo:

```
function onMetaData(infoObject:Object):void
{
    trace("metadata");
}
function onCuePoint(infoObject:Object):void
{
    trace("cue point");
}
```

Uso de pontos de sinalização e metadados

Use os métodos de retorno de chamada NetStream para capturar e processar eventos de metadados e de ponto de sinalização durante a reprodução do vídeo.

Uso de pontos de sinalização

A tabela a seguir descreve os métodos de retorno de chamada que você pode usar para capturar pontos de sinalização F4V e FLV no Flash Player e no AIR.

Tempo de execução	F4V	FLV
Flash Player 9/AIR1.0		OnCuePoint
		OnMetaData
Flash Player 10		OnCuePoint
	OnMetaData	OnMetaData
	OnXMPData	OnXMPData

O exemplo a seguir usa um loop `for..in` simples para se repetir em cada uma das propriedades do parâmetro `infoObject` que a função `onCuePoint()` recebe. Ele chama a função trace() para exibir uma mensagem quando recebe dados de ponto de sinalização:

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = this;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

function onCuePoint(infoObject:Object):void
{
    var key:String;
    for (key in infoObject)
    {
        trace(key + ": " + infoObject[key]);
    }
}
```

Aparece a seguinte saída:

```
parameters:
name: point1
time: 0.418
type: navigation
```

Este código usa uma das várias técnicas para definir o objeto no qual é executado o método de retorno de chamada. Você pode usar outras técnicas; para obter mais informações, consulte "Criação de métodos de retorno de chamada para metadados e pontos de sinalização" na página 542.

Uso de metadados de vídeo

É possível usar as funções `OnMetaData()` e `OnXMPData()` para acessar as informações de metadados do arquivo de vídeo, inclusive pontos de sinalização.

Uso de OnMetaData()

Os metadados incluem informações sobre o arquivo de vídeo, como duração, largura, altura e taxa de quadros. As informações dos metadados que são adicionadas ao arquivo de vídeo dependem do software utilizado para codificar o arquivo.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = this;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

function onMetaData(infoObject:Object):void
{
    var key:String;
    for (key in infoObject)
    {
        trace(key + ": " + infoObject[key]);
    }
}
```

O código anterior gera uma saída parecida com esta:

```
width: 320
audiodelay: 0.038
canSeekToEnd: true
height: 213
cuePoints: ,,
audiodatarate: 96
duration: 16.334
videodatarate: 400
framerate: 15
videocodecid: 4
audiocodecid: 2
```

Se o seu vídeo não tem áudio, as informações de metadados relacionadas a áudio (como `audiodatarate`) retornam `undefined`, pois nenhuma informação de áudio é adicionada aos metadados durante a codificação.

No código anterior, as informações de ponto de sinalização não eram exibidas. Para exibir os metadados de ponto de sinalização, use esta função, que exibe os itens recursivamente em um Object:

```
function traceObject(obj:Object, indent:uint = 0):void
{
    var indentString:String = "";
    var i:uint;
    var prop:String;
    var val:*;
    for (i = 0; i < indent; i++)
    {
        indentString += "\t";
    }
    for (prop in obj)
    {
        val = obj[prop];
        if (typeof(val) == "object")
        {
            trace(indentString + " " + prop + ": [Object]");
            traceObject(val, indent + 1);
        }
        else
        {
            trace(indentString + " " + prop + ": " + val);
        }
    }
}
```

O uso do snippet de código anterior para rastrear o parâmetro `infoObject` no método `onMetaData()` gera a seguinte saída:

```
width: 320
audiodatarate: 96
audiocodecid: 2
videocodecid: 4
videodatarate: 400
canSeekToEnd: true
duration: 16.334
audiodelay: 0.038
height: 213
framerate: 15
cuePoints: [Object]
    0: [Object]
        parameters: [Object]
            lights: beginning
        name: point1
        time: 0.418
        type: navigation
    1: [Object]
        parameters: [Object]
            lights: middle
        name: point2
        time: 7.748
        type: navigation
    2: [Object]
        parameters: [Object]
            lights: end
        name: point3
        time: 16.02
        type: navigation
```

O exemplo a seguir exibe os metadados de um vídeo MP4. Ele pressupõe que existe um objeto TextArea chamado `metaDataOut`, no qual grava os metadados.

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    import flash.events.NetStatusEvent;
    import flash.media.Video;
    import flash.display.StageDisplayState;
    import flash.display.Loader;
    import flash.display.Sprite;
    import flash.events.MouseEvent;

    public class onMetaDataExample extends Sprite
    {
        var video:Video = new Video();

        public function onMetaDataExample():void
        {
            var videoConnection:NetConnection = new NetConnection();
            videoConnection.connect(null);

            var videoStream:NetStream = new NetStream(videoConnection);
            videoStream.client = this;

            addChild(video);
```

```
            video.x = 185;
            video.y = 5;

            video.attachNetStream(videoStream);

            videoStream.play("video.mp4");

            videoStream.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
        }

        public function onMetaData(infoObject:Object):void
        {
            for(var propName:String in infoObject)
            {
                metaDataOut.appendText(propName + "=" + infoObject[propName] + "\n");
            }
        }

        private function netStatusHandler(event:NetStatusEvent):void
        {
            if(event.info.code == "NetStream.Play.Stop")
                stage.displayState = StageDisplayState.NORMAL;
        }
    }
}
```

A função `onMetaData()` gerou a seguinte saída para esse vídeo:

```
moovposition=731965
height=352
avclevel=21
videocodecid=avc1
duration=2.36
width=704
videoframerate=25
avcprofile=88
trackinfo=[object Object]
```

Uso do objeto de informação

A tabela a seguir mostra os possíveis valores para os metadados de vídeo que são passados à função de retorno de chamada `onMetaData()` no Object por eles recebido:

Parâmetro	Descrição
aacaot	Tipo de objeto de áudio AAC; são suportados os valores 0, 1 ou 2.
avclevel	Número de nível AVC IDC, como 10, 11, 20, 21 e assim por diante.
avcprofile	Número de perfil AVC, como 55, 77, 100 e assim por diante.
audiocodecid	String que indica o codec de áudio (técnica de codificação/decodificação) que foi usado; por exemplo ".Mp3" ou "mp4a"
audiodatarate	Número que indica a taxa de codificação de áudio, em KB por segundo.
audiodelay	Número que indica em que tempo do arquivo FLV está "time 0" do arquivo FLV original. O conteúdo do vídeo precisa ter um pequeno atraso para que o áudio seja sincronizado corretamente.

Parâmetro	Descrição
canSeekToEnd	Um valor booleano que é `true` se o arquivo FLV está codificado com um quadro-chave no último quadro, o que permite fazer a busca até o final de um arquivo de vídeo com download progressivo. É `falso` se o arquivo FLV não está codificado com um quadro-chave no último quadro.
cuePoints	Uma matriz de objetos, uma para cada ponto de sinalização incorporado no arquivo FLV. O valor é indefinido se o arquivo FLV não contiver pontos de sinalização. Cada objeto tem as seguintes propriedades: • `type`: string que especifica o tipo do ponto de sinalização como "navegação" ou "evento". • `name`: string que é o nome do ponto de sinalização. • `time`: número que é a hora do ponto de sinalização em segundos, com uma precisão de três casas decimais (milésimos de segundo). • `parameters`: objeto opcional que tem pares de nome/valor que o usuário designa ao criar os pontos de sinalização.
duration	Um número que representa a duração do arquivo de vídeo em segundos.
framerate	Um número que representa a taxa de quadros do arquivo FLV.
height	Um número que representa a altura do arquivo FLV em pixels.
seekpoints	Uma matriz que lista os quadros-chave disponíveis como carimbos de data/hora em milésimos de segundo. Opcional.
tags	Uma matriz de pares chave/valor que representam as informações no núcleo “ilst”, que é o equivalente de tags ID3 para arquivos MP4. O iTunes usa essas tags. Pode ser usado para exibir arte-final, se disponível.
trackinfo	Objeto que fornece informações sobre todas as faixas do arquivo MP4, inclusive o ID de descrição da amostra.
videocodecid	Uma string que representa a versão de codec que foi usada para codificar o vídeo. - por exemplo, “avc1” ou “VP6F”
videodatarate	Um número que representa a taxa de dados do vídeo do arquivo FLV.
videoframerate	Taxa de quadros do vídeo MP4.
width	Um número que representa a largura do arquivo FLV em pixels.

A tabela a seguir mostra os valores possíveis para o parâmetro `videocodecid`:

videocodecid	Nome do codec
2	Sorenson H.263
3	Vídeo de tela (somente SWF versão 7 e posteriores)
4	VP6 (somente SWF versão 8 e posteriores)
5	Vídeo VP6 com canal alfa (somente SWF versão 8 e posteriores)

A tabela a seguir mostra os valores possíveis para o parâmetro `audiocodecid`:

audiocodecid	Nome do codec
0	não compactado
1	ADPCM
2	Mp3
4	Nellymoser @ 16 kHz mono

audiocodecid	Nome do codec
5	Nellymoser, 8 kHz mono
6	Nellymoser
10	AAC
11	Speex

Uso de onXMPData()

A função de retorno de chamada onXMPData() recebe informações específicas da plataforma XMP (Extensible Metadata Platform) da Adobe que são incorporadas ao arquivo de vídeo Adobe F4V ou FLV. Os metadados XMP incluem pontos de sinalização e outros metadados de vídeo. Os metadados XMP foram introduzidos no Flash Player 10 e são suportados pelas versões subseqüentes do Flash Player e do Adobe AIR.

Este exemplo processa dados de ponto de sinalização nos metadados XMP:

```
package
{
    import flash.display.*;
    import flash.net.*;
    import flash.events.NetStatusEvent;
    import flash.media.Video;

    public class onXMPDataExample extends Sprite
    {
        public function onXMPDataExample():void
        {
            var videoConnection:NetConnection = new NetConnection();
            videoConnection.connect(null);

            var videoStream:NetStream = new NetStream(videoConnection);
            videoStream.client = this;
            var video:Video = new Video();

            addChild(video);

            video.attachNetStream(videoStream);

            videoStream.play("video.f4v");
        }

        public function onMetaData(info:Object):void {
            trace("onMetaData fired");
        }

        public function onXMPData(infoObject:Object):void
        {
            trace("onXMPData Fired\n");
            //trace("raw XMP =\n");
            //trace(infoObject.data);
            var cuePoints:Array = new Array();
            var cuePoint:Object;
            var strFrameRate:String;
            var nTracksFrameRate:Number;
            var strTracks:String = "";
```

```
            var onXMPXML = new XML(infoObject.data);
            // Set up namespaces to make referencing easier
            var xmpDM:Namespace = new Namespace("http://ns.adobe.com/xmp/1.0/DynamicMedia/");
            var rdf:Namespace = new Namespace("http://www.w3.org/1999/02/22-rdf-syntax-ns#");
            for each (var it:XML in onXMPXML..xmpDM::Tracks)
            {
                 var strTrackName:String =
it.rdf::Bag.rdf::li.rdf::Description.@xmpDM::trackName;
                 var strFrameRateXML:String =
it.rdf::Bag.rdf::li.rdf::Description.@xmpDM::frameRate;
                 strFrameRate = strFrameRateXML.substr(1,strFrameRateXML.length);

                 nTracksFrameRate = Number(strFrameRate);

                 strTracks += it;
            }
            var onXMPTracksXML:XML = new XML(strTracks);
            var strCuepoints:String = "";
            for each (var item:XML in onXMPTracksXML..xmpDM::markers)
            {
                strCuepoints += item;
            }
            trace(strCuepoints);
        }
    }
}
```

No caso de um arquivo de vídeo curto chamado startrekintro.f4v, este exemplo produz as linhas de rastreamento a seguir. As linhas mostram os dados de ponto de sinalização referentes a pontos de sinalização de navegação e de eventos nos metadados XMP:

```
onMetaData fired
onXMPData Fired

<xmpDM:markers xmlns:xmp="http://ns.adobe.com/xap/1.0/"
xmlns:xmpDM="http://ns.adobe.com/xmp/1.0/DynamicMedia/"
xmlns:stDim="http://ns.adobe.com/xap/1.0/sType/Dimensions#"
xmlns:xmpMM="http://ns.adobe.com/xap/1.0/mm/"
xmlns:stEvt="http://ns.adobe.com/xap/1.0/sType/ResourceEvent#"
xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#" xmlns:x="adobe:ns:meta/">
  <rdf:Seq>
    <rdf:li>
      <rdf:Description xmpDM:startTime="7695905817600" xmpDM:name="Title1"
xmpDM:type="FLVCuePoint" xmpDM:cuePointType="Navigation">
        <xmpDM:cuePointParams>
          <rdf:Seq>
            <rdf:li xmpDM:key="Title" xmpDM:value="Star Trek"/>
            <rdf:li xmpDM:key="Color" xmpDM:value="Blue"/>
          </rdf:Seq>
        </xmpDM:cuePointParams>
      </rdf:Description>
    </rdf:li>
    <rdf:li>
      <rdf:Description xmpDM:startTime="10289459980800" xmpDM:name="Title2"
xmpDM:type="FLVCuePoint" xmpDM:cuePointType="Event">
        <xmpDM:cuePointParams>
          <rdf:Seq>
            <rdf:li xmpDM:key="William Shatner" xmpDM:value="First Star"/>
            <rdf:li xmpDM:key="Color" xmpDM:value="Light Blue"/>
          </rdf:Seq>
        </xmpDM:cuePointParams>
      </rdf:Description>
    </rdf:li>
  </rdf:Seq>
</xmpDM:markers>
onMetaData fired
```

Nota: *Nos dados XMP, o tempo é armazenado como Tiques DVA e não como segundos. Para calcular o tempo do ponto de sinalização, divida a hora inicial pela taxa de quadros. Por exemplo, a hora inicial de 7695905817600 dividida por uma taxa de quadros de 254016000000 é igual a 30:30.*

Para ver os metadados XMP brutos completos, que incluem a taxa de quadros, remova os identificadores de comentário (//'s) que precedem a segunda e a terceira instruções `trace()` no início da função `onXMPData()`.

Para obter mais informações sobre a XMP, consulte:

- http://partners.adobe.com/public/developer/xmp/topic.html
- http://www.adobe.com/devnet/xmp/

Uso de metadados de imagem

O evento `onImageData` envia dados de imagem como uma matriz de bytes por um canal de dados AMF0. Os dados podem estar nos formatos JPEG, PNG ou GIF. Defina um método de retorno de chamada `onImageData()` para processar essas informações, da mesma forma que você definiria métodos de retorno de chamada para `onCuePoint` e `onMetaData`. Este exemplo acessa e exibe dados de imagem usando um método de retorno de chamada `onImageData()`:

```
public function onImageData(imageData:Object):void
{
    // display track number
    trace(imageData.trackid);
    var loader:Loader = new Loader();
    //imageData.data is a ByteArray object
    loader.loadBytes(imageData.data);
    addChild(loader);
}
```

Uso de metadados de texto

O evento `onTextData` envia dados de texto por meio de um canal de dados AMF0. Os dados de texto estão no formato UTF-8 e contêm informações adicionais sobre formatação com base na especificação de texto com tempo 3GP. Essa especificação define um formato de legenda padronizado. Defina um método de retorno de chamada `onTextData()` para processar essas informações, da mesma forma que você definiria métodos de retorno de chamada para `onCuePoint` ou `onMetaData`. No próximo exemplo, o método `onTextData()` exibe o número de identificação da faixa e o texto da faixa correspondente.

```
public function onTextData(textData:Object):void
{
    // display the track number
    trace(textData.trackid);
    // displays the text, which can be a null string, indicating old text
    // that should be erased
    trace(textData.text);
}
```

Captura da entrada da câmera

Além de arquivos de vídeo externos, uma câmera conectada ao computador de um usuário pode funcionar como fonte de dados de vídeo, que você pode exibir e manipular usando o ActionScript. A classe Camera é o mecanismo que foi incorporado ao ActionScript para trabalhar com uma câmera de computador.

Noções básicas sobre a classe Camera

O objeto Camera permite que você se conecte à câmera local do usuário e transmita o vídeo localmente (para o usuário) ou remotamente para um servidor (como o Flash Media Server).

Usando a classe Camera, é possível acessar os seguintes tipos de informações sobre a câmera do usuário:

- Quais câmeras instaladas no computador do usuário estão disponíveis para o Flash Player
- Se há uma câmera instalada
- Se o Flash Player tem acesso à câmera do usuário ou não
- Qual câmera está ativa
- A largura e a altura do vídeo que está sendo capturado

A classe Camera inclui vários métodos e propriedades úteis para trabalhar com objetos Camera. Por exemplo, a propriedade estática `Camera.names` contém uma matriz de nomes de câmeras que estão instaladas no computador do usuário. Você também pode usar a propriedade `name` para exibir o nome da câmera que está ativa.

Exibição do conteúdo da câmera na tela

A conexão a uma câmera pode exigir menos código do que usar as classes NetConnection e NetStream para carregar um vídeo. A classe Camera também pode rapidamente se tornar complicada porque, com o Flash Player, você precisa da permissão do usuário para se conectar à câmera dele antes de acessá-la.

Este código demonstra como usar a classe Camera para se conectar à câmera local de um usuário:

```
var cam:Camera = Camera.getCamera();
var vid:Video = new Video();
vid.attachCamera(cam);
addChild(vid);
```

Nota: *A classe Camera não tem um método construtor. Para criar uma nova ocorrência de Camera, use o método estático* `Camera.getCamera()`.

Desenvolvimento do aplicativo de câmera

Ao criar um aplicativo que se conecta à câmera de um usuário, considere os seguintes aspectos no seu código:

- Verifique se o usuário tem uma câmera instalada.
- Somente no Flash Player, verifique se o usuário permitiu acesso à câmera explicitamente. Por motivo de segurança, o player exibe a caixa de diálogo Configurações do Flash Player, onde o usuário pode conceder ou negar acesso à sua câmera. Isso impede que o Flash Player se conecte à câmera de um usuário e transmita um fluxo de vídeo sem a devida permissão. Se um usuário clicar em permitir, o aplicativo poderá se conectar à câmera. Se ele clicar em negar, o aplicativo não conseguirá ter acesso à câmera do usuário. Seus aplicativos sempre devem manipular os dois casos normalmente.

Conexão à câmera de um usuário

A primeira etapa para se conectar à câmera de um usuário é criar uma nova ocorrência de Camera criando uma variável do tipo Camera e a inicializando com o valor de retorno do método estático `Camera.getCamera()`.

A próxima etapa é criar um novo objeto de vídeo e conectar o objeto Camera a ele.

A terceira etapa é adicionar o objeto de vídeo à lista de exibição. Você precisa executar as etapas 2 e 3 porque a classe Camera não estende a classe DisplayObject, por isso não é possível adicioná-la diretamente à lista de exibição. Para exibir o vídeo capturado da câmera, crie um novo objeto de vídeo e chame o método `attachCamera()`.

O seguinte código mostra estas três etapas:

```
var cam:Camera = Camera.getCamera();
var vid:Video = new Video();
vid.attachCamera(cam);
addChild(vid);
```

Observe que, se o usuário não tiver uma câmera instalada, o aplicativo não exibirá nada.

Na prática, você precisa executar outras etapas que envolvem o seu aplicativo. Para obter mais informações, consulte "Verificação da instalação das câmeras" na página 558 e "Detecção de permissões de acesso à câmera" na página 558.

Verificação da instalação das câmeras

Antes de tentar usar qualquer um dos métodos ou propriedades em uma ocorrência de Camera, é recomendável verificar se o usuário tem uma câmera instalada. Há duas maneiras de fazer essa verificação:

- Verifique a propriedade estática `Camera.names` que contém uma matriz de nomes de câmeras que estão disponíveis. Normalmente, essa matriz terá no máximo 1 string, porque é provável que a maioria dos usuários não tenha mais de uma câmera instalada. Este código demonstra como verificar a propriedade `Camera.names` para saber se o usuário tem alguma câmera disponível:

```
if (Camera.names.length > 0)
{
    trace("User has at least one camera installed.");
    var cam:Camera = Camera.getCamera(); // Get default camera.
}
else
{
    trace("User has no cameras installed.");
}
```

- Verifique o valor de retorno do método estático `Camera.getCamera()`. Se não houver nenhuma câmera disponível ou instalada, o método retornará `null`; caso contrário, ele retornará uma referência a um objeto Camera. Este código demonstra como verificar o método `Camera.getCamera()` para saber se o usuário tem alguma câmera disponível:

```
var cam:Camera = Camera.getCamera();
if (cam == null)
{
    trace("User has no cameras installed.");
}
else
{
    trace("User has at least 1 camera installed.");
}
```

Como a classe Camera não estende a classe DisplayObject, não pode ser adicionada diretamente à lista de exibição usando o método `addChild()`. Para exibir o vídeo capturado da câmera, é preciso criar um novo objeto Video e chamar o método `attachCamera()` na ocorrência de Video.

Este snippet mostra como conectar a câmera, se disponível; se não houver uma câmera, o aplicativo não exibirá nada:

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
    var vid:Video = new Video();
    vid.attachCamera(cam);
    addChild(vid);
}
```

Detecção de permissões de acesso à câmera

Na caixa de proteção do aplicativo AIR, o aplicativo pode acessar qualquer câmera sem a permissão do usuário.

Para que o Flash Player possa exibir a saída de uma câmera, o usuário deve permitir o acesso do Flash Player à câmera explicitamente. Quando o método `attachCamera()` é chamado, o Flash Player exibe a caixa de diálogo Configurações do Flash Player, que pede para o usuário conceder ou negar acesso à câmera e ao microfone para o Flash Player. Se o usuário clicar no botão Permitir, o Flash Player exibirá a saída da câmera na ocorrência de Video no Palco. Se o usuário clicar no botão Negar, o Flash Player não conseguirá se conectar à câmera, e o objeto Video não exibirá nada.

Para detectar se o usuário concedeu acesso à câmera para o Flash Player, você pode monitorar o evento `status` da câmera (`StatusEvent.STATUS`), como visto neste código:

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
    cam.addEventListener(StatusEvent.STATUS, statusHandler);
    var vid:Video = new Video();
    vid.attachCamera(cam);
    addChild(vid);
}
function statusHandler(event:StatusEvent):void
{
    // This event gets dispatched when the user clicks the "Allow" or "Deny"
    // button in the Flash Player Settings dialog box.
    trace(event.code); // "Camera.Muted" or "Camera.Unmuted"
}
```

A função `statusHandler()` é chamada assim que o usuário clica em Permitir ou Negar. Você pode detectar em qual botão o usuário clicou usando um destes dois métodos:

- O parâmetro `event` da função `statusHandler()` contém uma propriedade de código que inclui a string "Camera.Muted" ou "Camera.Unmuted". Se o valor é "Camera.Muted", isso significa que o usuário clicou no botão Negar e o Flash Player não pode acessar a câmera. Este snippet mostra um exemplo disso:

```
function statusHandler(event:StatusEvent):void
{
    switch (event.code)
    {
        case "Camera.Muted":
            trace("User clicked Deny.");
            break;
        case "Camera.Unmuted":
            trace("User clicked Accept.");
            break;
    }
}
```

- A classe Camera contém uma propriedade somente leitura chamada `muted` que especifica se o usuário negou acesso à câmera (`true`) ou se permitiu acesso a ela (`false`) no painel Privacidade do Flash Player. Este snippet mostra um exemplo disso:

```
function statusHandler(event:StatusEvent):void
{
    if (cam.muted)
    {
        trace("User clicked Deny.");
    }
    else
    {
        trace("User clicked Accept.");
    }
}
```

Verificando o eventos de status a ser despachado, você pode criar um código que lide com a aceitação ou a negação do acesso à câmera pelo usuário e fazer as devidas exclusões. Por exemplo, se o usuário clicar no botão Negar, você poderá exibir uma mensagem informando que ele deve clicar em Permitir se deseja participar de um bate-papo com vídeo, ou você poderá verificar se o objeto Video da lista de exibição foi excluído para liberar recursos do sistema.

Maximização da qualidade do vídeo

Por padrão, as novas ocorrências da classe Video têm 320 pixels de largura por 240 pixels de altura. Para maximizar a qualidade dos vídeos, sempre verifique se o objeto de vídeo tem as mesmas dimensões do vídeo retornado pelo objeto de câmera. Para obter a largura e a altura do objeto de câmera, use as propriedades `width` e `height` da classe Camera; em seguida, você pode definir as propriedades `width` e `height` do objeto de vídeo de modo que correspondam às dimensões dos objetos de câmera, ou pode passar a largura e a altura da câmera para o método construtor da classe Video, como visto neste snippet:

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
    var vid:Video = new Video(cam.width, cam.height);
    vid.attachCamera(cam);
    addChild(vid);
}
```

Uma vez que o método `getCamera()` retorna uma referência a um objeto de câmera (ou `null` se não houver câmeras disponíveis), você poderá acessar os métodos e as propriedades da câmera mesmo que o usuário negue acesso a ela. Isso permite que você defina o tamanho da ocorrência de vídeo usando a altura e a largura nativas da câmera.

```
var vid:Video;
var cam:Camera = Camera.getCamera();

if (cam == null)
{
    trace("Unable to locate available cameras.");
}
else
{
    trace("Found camera: " + cam.name);
    cam.addEventListener(StatusEvent.STATUS, statusHandler);
    vid = new Video();
    vid.attachCamera(cam);
}
function statusHandler(event:StatusEvent):void
{
    if (cam.muted)
    {
        trace("Unable to connect to active camera.");
    }
    else
    {
        // Resize Video object to match camera settings and
        // add the video to the display list.
        vid.width = cam.width;
        vid.height = cam.height;
        addChild(vid);
    }
    // Remove the status event listener.
    cam.removeEventListener(StatusEvent.STATUS, statusHandler);
}
```

Para obter informações sobre o modo de tela cheia, consulte a seção Modo de tela cheia em "Configuração de propriedades do palco" na página 287.

Monitoramento das condições de reprodução

A classe Camera contém várias propriedades que permitem monitorar o status atual do objeto Camera. Por exemplo, este código exibe diversas propriedades da câmera usando um objeto Timer e uma ocorrência de campo de texto na lista de exibição:

```
var vid:Video;
var cam:Camera = Camera.getCamera();
var tf:TextField = new TextField();
tf.x = 300;
tf.autoSize = TextFieldAutoSize.LEFT;
addChild(tf);

if (cam != null)
{
    cam.addEventListener(StatusEvent.STATUS, statusHandler);
    vid = new Video();
    vid.attachCamera(cam);
}
function statusHandler(event:StatusEvent):void
{
    if (!cam.muted)
    {
        vid.width = cam.width;
        vid.height = cam.height;
        addChild(vid);
        t.start();
    }
    cam.removeEventListener(StatusEvent.STATUS, statusHandler);
}

var t:Timer = new Timer(100);
t.addEventListener(TimerEvent.TIMER, timerHandler);
function timerHandler(event:TimerEvent):void
{
    tf.text = "";
    tf.appendText("activityLevel: " + cam.activityLevel + "\n");
    tf.appendText("bandwidth: " + cam.bandwidth + "\n");
    tf.appendText("currentFPS: " + cam.currentFPS + "\n");
    tf.appendText("fps: " + cam.fps + "\n");
    tf.appendText("keyFrameInterval: " + cam.keyFrameInterval + "\n");
    tf.appendText("loopback: " + cam.loopback + "\n");
    tf.appendText("motionLevel: " + cam.motionLevel + "\n");
    tf.appendText("motionTimeout: " + cam.motionTimeout + "\n");
    tf.appendText("quality: " + cam.quality + "\n");
}
```

A cada 1/10 de segundo (100 milissegundos) o evento `timer` do objeto Timer é despachado, e a função `timerHandler()` atualiza o campo de texto na lista de exibição.

Envio de vídeo para um servidor

Se você deseja criar aplicativos mais complexos que envolvam objetos de vídeo ou câmera, o Flash Media Server oferece uma combinação de recursos de fluxo de mídia e um ambiente de desenvolvimento para criar e disponibilizar aplicativos de mídia para um grande público. Esta combinação permite que os desenvolvedores criem aplicativos como Vídeo sob Demanda, transmissões ao vivo de eventos pela Web e streaming Mp3, além de publicação de vídeos em blogs, envio de vídeos via mensagens instantâneas e ambientes de bate-papo com recursos multimídia. Para obter mais informações, consulte a documentação on-line do Flash Media Server em www.adobe.com/go/learn_fms_docs_br.

Tópicos avançados sobre arquivos FLV

Os tópicos a seguir abordam algumas questões especiais sobre o trabalho com arquivos FLV.

Sobre a configuração de arquivos FLV para hospedagem em um servidor

Quando você trabalha com arquivos FLV, pode ser necessário configurar o servidor para trabalhar com o formato de arquivo FLV. MIME (Multipurpose Internet Mail Extensions) é uma especificação de dados padronizada que permite enviar arquivos não-ASCII por conexões de Internet. Navegadores da Web e clientes de e-mail são configurados para interpretar inúmeros tipos MIME de maneira que possam enviar e receber vídeo, áudio, gráficos e texto formatado. Para carregar arquivos FLV de um servidor Web, talvez você precise registrar a extensão de arquivo e o tipo MIME no servidor Web; para isso, consulte a documentação do servidor Web. O tipo MIME de arquivos FLV é `video/x-flv`. As informações completas sobre o tipo de arquivo FLV são as seguintes:

- Tipo MIME: video/x-flv
- Extensão de arquivo: .flv
- Parâmetros necessários: nenhum
- Parâmetros opcionais: nenhum
- Considerações de codificação: os arquivos FLV são binários; alguns aplicativos podem exigir que o subtipo application/octet-stream seja definido
- Problemas de segurança: nenhum
- Especificação publicada: www.adobe.com/go/video_file_format_br

A Microsoft mudou a forma de tratamento de fluxos de mídia no servidor Web Microsoft Internet Information Services (IIS) 6.0 em relação a versões anteriores. As versões anteriores do IIS não exigem modificações para transmitir vídeo Flash em fluxo. No IIS 6.0, o servidor Web padrão fornecido com o Windows 2003, o servidor requer um tipo MIME para reconhecer que os arquivos FLV são fluxo de mídia.

Quando arquivos SWF que transmitem arquivos FLV externos em fluxo contínuo são colocados no Microsoft Windows Server® 2003 e visualizados em um navegador, o arquivo SWF é reproduzido corretamente, mas o vídeo FLV não é transmitido em fluxo contínuo. Este problema afeta todos os arquivos FLV colocados no Windows Server 2003, inclusive os arquivos criados com versões anteriores da Ferramenta de autoria do Flash e com o Macromedia Flash Video Kit para Dreamweaver MX 2004 da Adobe. Esses arquivos funcionarão corretamente se você testá-los em outros sistemas operacionais.

Para obter informações sobre como configurar o Microsoft Windows 2003 e o Microsoft IIS Server 6.0 para transmissão de vídeos FLV em fluxo contínuo, consulte www.adobe.com/go/tn_19439_br.

Sobre direcionamento de arquivos FLV locais no Macintosh

Se você tentar reproduzir um arquivo FLV local de uma unidade que não é do sistema em um computador com o Apple® Macintosh® usando um caminho que contém uma barra relativa (/), o vídeo não será reproduzido. Unidades que não são do sistema incluem mas não se limitam a CD-ROMs, discos rígidos particionados, mídia de armazenamento removíveis e dispositivos de armazenamento conectados.

Nota: *O motivo desta falha é uma limitação do sistema operacional e não do Flash Player ou do AIR.*

Para que um arquivo FLV seja reproduzido de uma unidade que não é do sistema em um Macintosh, faça referência a ela com um caminho absoluto usando uma notação baseada em dois pontos (:) em vez de barra (/). A lista a seguir mostra a diferença entre os dois tipos de notação:

- Notação baseada em barra**:** myDrive/myFolder/myFLV.flv
- Notação baseada em dois pontos**:** (Mac OS®) myDrive:myFolder:myFLV.flv

Também é possível criar um arquivo de projetor para um CD-ROM que você pretende usar para reprodução no Macintosh. Para obter informações mais recentes sobre CD-ROMs do Mac OS e arquivos FLV, consulte www.adobe.com/go/3121b301_br.

Exemplo: Jukebox de vídeo

O exemplo a seguir cria uma jukebox de vídeo simples que carrega uma lista de vídeos dinamicamente para reprodução em seqüência. Isso permite que você crie um aplicativo com o qual um usuário pode navegar por uma série de tutoriais em vídeo ou talvez especificar quais anúncios devem ser reproduzidos antes de reproduzir o vídeo solicitado pelo usuário. Este exemplo mostra os seguintes recursos do ActionScript 3.0:

- Atualização de um indicador de reprodução com base no progresso da reprodução de um arquivo de vídeo
- Monitoramento e análise dos metadados de um arquivo de vídeo
- Manipulação de códigos específicos em um fluxo de rede
- Carregamento, reprodução, pausa e interrupção de um FLV carregado dinamicamente
- Redimensionamento de um objeto de vídeo na lista de exibição com base nos metadados do fluxo de rede

Para obter os arquivos de aplicativo deste exemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_br. Os arquivos do aplicativo Jukebox de vídeo podem ser encontrados na pasta Samples/VideoJukebox. O aplicativo consiste nos seguintes arquivos:

Arquivo	Descrição
VideoJukebox.as	A classe que fornece a funcionalidade principal do aplicativo.
VideoJukebox.fla	O arquivo de aplicativo principal para Flash.
playlist.xml	Um arquivo que lista quais arquivos de vídeo serão carregados na jukebox de vídeo.

Carregamento de um arquivo externo de lista de reprodução de vídeo

O arquivo externo playlist.xml especifica os vídeos que devem ser carregados e a ordem em que serão reproduzidos. Para carregar o arquivo XML, você precisa usar um objeto URLLoader e um objeto URLRequest, como visto neste código:

```
uldr = new URLLoader();
uldr.addEventListener(Event.COMPLETE, xmlCompleteHandler);
uldr.load(new URLRequest(PLAYLIST_XML_URL));
```

Este código é colocado no construtor da classe VideoJukebox para que o arquivo seja carregado antes da execução de qualquer outro código. Assim que termina o carregamento do arquivo XML, é chamado o método `xmlCompleteHandler()`, que analisa o arquivo externo em um objeto XML, como visto neste código:

```
private function xmlCompleteHandler(event:Event):void
{
    playlist = XML(event.target.data);
    videosXML = playlist.video;
    main();
}
```

O objeto XML `playlist` contém o XML bruto do arquivo externo, enquanto videosXML é um objeto XMLList que contém apenas os nós de vídeo. Um arquivo playlist.xml de exemplo pode ser visto no seguinte snippet:

```
<videos>
    <video url="video/caption_video.flv" />
    <video url="video/cuepoints.flv" />
    <video url="video/water.flv" />
</videos>
```

Para finalizar, o método `xmlCompleteHandler()` chama o método `main()`, que configura as várias ocorrências de componente na lista de exibição, bem como os objetos NetConnection e NetStream que são usados para carregar os arquivos FLV externos.

Criação da interface de usuário

Para criar a interface de usuário, você precisa arrastar cinco ocorrências de Button até a lista de exibição e dar a elas os seguintes nomes de ocorrência: `playButton`, `pauseButton`, `stopButton`, `backButton` e `forwardButton`.

Para cada uma dessas ocorrências de Button, você terá de atribuir um manipulador para o evento `click`, como visto no seguinte snippet:

```
playButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
pauseButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
stopButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
backButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
forwardButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
```

O método `buttonClickHandler()` usa uma instrução de opção para determinar qual ocorrência de Button foi clicada, como visto neste código:

```
private function buttonClickHandler(event:MouseEvent):void
{
    switch (event.currentTarget)
    {
        case playButton:
            ns.resume();
            break;
        case pauseButton:
            ns.togglePause();
            break;
        case stopButton:
            ns.pause();
            ns.seek(0);
            break;
        case backButton:
            playPreviousVideo();
            break;
        case forwardButton:
            playNextVideo();
            break;
    }
}
```

Em seguida, adicione uma ocorrência de Slider à lista de exibição e dê a ela um nome de ocorrência de `volumeSlider`. O seguinte código define a propriedade `liveDragging` da ocorrência de Slider como `true` e um ouvinte de eventos para o evento `change` da ocorrência de Slider:

```
volumeSlider.value = volumeTransform.volume;
volumeSlider.minimum = 0;
volumeSlider.maximum = 1;
volumeSlider.snapInterval = 0.1;
volumeSlider.tickInterval = volumeSlider.snapInterval;
volumeSlider.liveDragging = true;
volumeSlider.addEventListener(SliderEvent.CHANGE, volumeChangeHandler);
```

Adicione uma ocorrência de ProgressBar à lista de exibição e dê a ela um nome de ocorrência de `positionBar`. Defina a propriedade `mode` como manual, conforme visto no seguinte snippet:

```
positionBar.mode = ProgressBarMode.MANUAL;
```

Para terminar, adicione uma ocorrência de Label à lista de exibição e dê a ela um nome de ocorrência de `positionLabel`. O valor dessa ocorrência de Label será definido pela ocorrência de Timer.

Monitoramento dos metadados de um objeto de vídeo

Quando o Flash Player encontra metadados de cada um dos vídeos carregados, o manipulador de retorno de chamada `onMetaData()` é chamado na propriedade `client` do objeto NetStream. O seguinte código inicializa um Object e configura o manipulador de retorno de chamada especificado:

```
client = new Object();
client.onMetaData = metadataHandler;
```

O método `metadataHandler()` copia seus dados para a propriedade meta definida anteriormente no código. Isso permite que você acesse os metadados do vídeo atual a qualquer momento em todo o aplicativo. Em seguida, o objeto de vídeo no Palco é redimensionado para corresponder às dimensões retornadas nos metadados. Por último, a ocorrência da barra de progresso positionBar é movida e redimensionada com base no tamanho do vídeo que está sendo reproduzido. O seguinte código contém o método `metadataHandler()` inteiro:

```
private function metadataHandler(metadataObj:Object):void
{
    meta = metadataObj;
    vid.width = meta.width;
    vid.height = meta.height;
    positionBar.move(vid.x, vid.y + vid.height);
    positionBar.width = vid.width;
}
```

Carregamento dinâmico de um vídeo Flash

Para carregar cada um dos vídeos Flash dinamicamente, o aplicativo usa um objeto NetConnection e NetStream. O código a seguir cria um objeto NetConnection e passa `null` para o método `connect()`. Especificando `null`, o Flash Player se conecta a um vídeo do servidor local em vez de se conectar a um servidor, como o Flash Media Server.

O seguinte código cria ocorrências de NetConnection e de NetStream, define um ouvinte de eventos para o evento `netStatus` e atribui Object `client` à propriedade `client`:

```
nc = new NetConnection();
nc.connect(null);

ns = new NetStream(nc);
ns.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
ns.client = client;
```

O método `netStatusHandler()` é chamado sempre que o status do vídeo é alterado. Isso inclui as ocasiões em que a reprodução de um vídeo é iniciada ou interrompida, quando o vídeo é armazenado em buffer ou quando um fluxo de vídeo não é encontrado. O seguinte código lista o evento `netStatusHandler()`:

```
private function netStatusHandler(event:NetStatusEvent):void
{
    try
    {
        switch (event.info.code)
        {
            case "NetStream.Play.Start":
                t.start();
                break;
            case "NetStream.Play.StreamNotFound":
            case "NetStream.Play.Stop":
                t.stop();
                playNextVideo();
                break;
        }
    }
    catch (error:TypeError)
    {
        // Ignore any errors.
    }
}
```

O código anterior avalia a propriedade de código do objeto de informações e filtra se o código é “NetStream.Play.Start”, “NetStream.Play.StreamNotFound” ou “NetStream.Play.Stop”. Todos os demais códigos serão ignorados. Se o fluxo de rede estiver iniciando, o código iniciará a ocorrência de Timer que atualiza o indicador de reprodução. Se o fluxo de rede não for encontrado ou se for interrompido, a ocorrência de Timer será interrompida e o aplicativo tentará reproduzir o próximo vídeo da lista de reprodução.

Sempre que Timer é executado, a ocorrência da barra de progresso `positionBar` atualiza sua posição atual chamando o método `setProgress()` da classe ProgressBar, e a ocorrência de Label `positionLabel` é atualizada com o tempo decorrido e o total de tempo do vídeo atual.

```
private function timerHandler(event:TimerEvent):void
{
    try
    {
        positionBar.setProgress(ns.time, meta.duration);
        positionLabel.text = ns.time.toFixed(1) + " of " meta.duration.toFixed(1) + " seconds";
    }
    catch (error:Error)
    {
        // Ignore this error.
    }
}
```

Controle do volume do vídeo

É possível controlar o volume do vídeo carregado dinamicamente definindo a propriedade `soundTransform` no objeto NetStream. O aplicativo de jukebox de vídeo permite modificar o nível de volume alterando o valor da ocorrência de Slider `volumeSlider`. Este código mostra como alterar o nível de volume atribuindo o valor do componente Slider a um objeto SoundTransform, que é definido com a propriedade `soundTransform` no objeto NetStream:

```
private function volumeChangeHandler(event:SliderEvent):void
{
    volumeTransform.volume = event.value;
    ns.soundTransform = volumeTransform;
}
```

Controle da reprodução de vídeo

O restante do aplicativo controla a reprodução de vídeos quando o vídeo chega ao final do fluxo ou quando o usuário acessa o vídeo anterior ou o próximo.

O seguinte método recupera a URL do vídeo no objeto XMLList relativo ao índice que está selecionado:

```
private function getVideo():String
{
    return videosXML[idx].@url;
}
```

O método `playVideo()` chama o método `play()` no objeto NetStream para carregar o vídeo que está selecionado:

```
private function playVideo():void
{
    var url:String = getVideo();
    ns.play(url);
}
```

O método `playPreviousVideo()` reduz o índice de vídeos atual, chama o método `playVideo()` para carregar o novo arquivo de vídeo e define a barra de progresso como visível:

```
private function playPreviousVideo():void
{
    if (idx > 0)
    {
        idx--;
        playVideo();
        positionBar.visible = true;
    }
}
```

O último método, `playNextVideo()`, aumenta o índice de vídeos e chama o método `playVideo()`. Se o vídeo atual é o último da lista de reprodução, o método `clear()` é chamado no objeto Video, e a propriedade `visible` da ocorrência da barra de progresso é definida como `false`:

```
private function playNextVideo():void
{
    if (idx < (videosXML.length() - 1))
    {
        idx++;
        playVideo();
        positionBar.visible = true;
    }
    else
    {
        idx++;
        vid.clear();
        positionBar.visible = false;
    }
}
```

Capítulo 25: Trabalho com som

O ActionScript foi desenvolvido para aplicativos interativos de imersão, e um dos elementos muitas vezes negligenciado nesses aplicativos é o som. É possível adicionar efeitos de som a um video game, feedback de áudio para a interface do usuário de um aplicativo ou mesmo criar um programa que analisa arquivos MP3 carregados pela Internet, com som na base do aplicativo.

Neste capítulo, você aprenderá como carregar arquivos de áudio externos e trabalhar com áudio incorporado em um SWF. Aprenderá a controlar o áudio, a criar representações visuais das informações de som e a capturar som de um microfone do usuário.

Noções básicas do trabalho com som

Introdução ao trabalho com som

Os computadores podem capturar e codificar áudio digital, que é a representação das informações de som no computador, bem como armazená-lo e recuperá-lo para reprodução pelos alto-falantes. É possível reproduzir som usando o Adobe® Flash® Player ou Adobe® AIR™ e o ActionScript.

Quando dados de som são convertidos em formato digital, eles têm várias características, como o volume e se o som é estéreo ou mono. Quando você reproduz um som no ActionScript, também pode ajustar essas características; por exemplo, você pode aumentar o som ou fazer parecer que ele está vindo de uma certa direção.

Para controlar um som no ActionScript, você precisa ter as informações do som carregadas no Flash Player ou no AIR. Há cinco maneiras de inserir dados de áudio no Flash Player ou AIR para que você possa trabalhar com eles no ActionScript. É possível carregar um arquivo de som externo, como um mp3, no SWF; incorporar as informações de som no arquivo SWF diretamente quando ele está sendo criado, obter entrada de áudio usando um microfone conectado ao computador do usuário, acessar dados de som transmitidos de um servidor e trabalhar com dados de som gerados de maneira dinâmica.

Quando você carrega dados de som a partir de um arquivo de som externo, pode começar a reproduzir o início do arquivo enquanto o restante dos dados ainda estão sendo carregados.

Embora existam vários formatos de arquivo de som usados para codificar áudio digital, o ActionScript 3.0, o Flash Player e o AIR dão suporte a arquivos de som armazenados no formato mp3. Eles não podem carregar ou reproduzir arquivos de som diretamente em outros formatos, como WAV ou AIFF.

Enquanto estiver trabalhando com som no ActionScript, provavelmente você desejará trabalhar com várias classes do pacote flash.media. A classe Sound é a classe que você usa para obter acesso a informações de áudio carregando um arquivo de som ou atribuindo uma função a um evento que tira amostras de dados de som e, em seguida, inicia a reprodução. Depois que você começa a reproduzir um som, o Flash Player e o AIR lhe dão acesso a um objeto SoundChannel. Como um arquivo de áudio carregado só pode ser um dos vários sons que você reproduz no computador de um usuário, cada som específico reproduzido usa seu próprio objeto SoundChannel; a saída combinada de todos os objetos SoundChannel misturados na verdade consiste no que é reproduzido pelos alto-falantes do computador. Use esta ocorrência de SoundChannel para controlar as propriedades do som e interromper a reprodução. Por último, se você deseja controlar o áudio combinado, a classe SoundMixer lhe dá controle sobre a saída combinada.

Também é possível usar várias outras classes para executar tarefas mais específicas quando você estiver trabalhando com som no ActionScript; para obter mais informações sobre todas as classes relacionadas a som, consulte "Compreensão da arquitetura do som" na página 571.

Tarefas comuns do trabalho com som

Este capítulo descreve as seguintes tarefas relacionadas a som que provavelmente você desejará executar:

- Carregamento de arquivos mp3 externos e controle do progresso da operação
- Reprodução, pausa, reinício e interrupção de sons
- Reprodução de fluxos de som durante seu carregamento
- Manipulação do volume e da panorâmica do som
- Recuperação de metadados ID3 de um arquivo mp3
- Uso de dados de onda de som brutos
- Geração dinâmica de som
- Captura e reprodução de entrada de som a partir do microfone de um usuário

Conceitos e termos importantes

A lista de referência a seguir contém termos importantes usados neste capítulo:

- Amplitude: a distância de um ponto na forma de onda de som a partir da linha zero ou de equilíbrio.
- Taxa de bits: a quantidade de dados que é codificada ou transmitida em fluxo para cada segundo de um arquivo de som. No caso de arquivos mp3, a taxa de bits normalmente é informada em milhares de bits por segundo (kbps). Uma taxa de bits mais alta geralmente significa uma onda de som de melhor qualidade.
- Armazenamento em buffer: o recebimento e o armazenamento de dados de som antes de eles serem reproduzidos.
- mp3: MPEG-1 Audio Layer 3, ou mp3, é um formato popular de compactação de som.
- Panorâmica: o posicionamento de um sinal de áudio entre os canais esquerdo e direito em um campo de som estéreo.
- Pico: o ponto mais alto em uma forma de onda.
- Taxa de amostragem: define o número de amostras por segundo extraídas de um sinal de áudio analógico para criar um sinal digital. A taxa de amostragem de áudio de um CD padrão é de 44,1 kHz ou 44.100 amostras por segundo.
- Streaming: o processo de reproduzir as partes iniciais de um arquivo de som ou de vídeo enquanto as partes finais ainda estão sendo carregadas de um servidor.
- Volume: o volume de um som.
- Forma de onda: a forma de um gráfico das variadas amplitudes de um sinal de som ao longo do tempo.

Teste dos exemplos do capítulo

Talvez você queira testar algumas das listagens de código de exemplo durante a leitura deste capítulo. Como este capítulo aborda o trabalho com som no ActionScript, muitos dos exemplos fazem algo que envolve o trabalho com um arquivo de som — reproduzi-lo, interromper a reprodução ou ajustar o som de alguma maneira. Para testar os exemplos deste capítulo:

1 Crie um novo documento do Flash e salve-o no seu computador.

2 Na Linha do tempo, selecione o primeiro quadro-chave e abra o painel Ações.

3 Copie a listagem de código de exemplo no painel Script.

4 Se o código envolver o carregamento de um arquivo de som externo, ele terá uma linha de código parecida com esta:

```
var req:URLRequest = new URLRequest("click.mp3");
var s:Sound = new Sound(req);
s.play();
```

onde "click.mp3" é o nome do arquivo de som que está sendo carregado. Para testar estes exemplos, você precisará de um arquivo mp3 para usá-lo. Coloque o arquivo mp3 na mesma pasta em que está o documento do Flash. Em seguida, altere o código para usar o nome do seu arquivo mp3 em vez do nome especificado na listagem de código (por exemplo, no código acima, você mudaria "click.mp3" para o nome do seu arquivo mp3).

5 No menu principal, selecione Controle > Testar filme para criar o arquivo SWF e visualizar (e ouvir) a saída do exemplo.

Além de reproduzir o áudio, alguns dos exemplos exibem valores usando a função `trace()`; quando estiver testando esses exemplos, você verá os resultados desses valores no painel Saída. Alguns exemplos também desenham conteúdo na tela, por isso, nesses exemplos, você também verá o conteúdo em uma janela do Flash Player ou do AIR.

Para obter mais informações sobre como testar as listagens de código de exemplo deste manual, consulte "Teste de listagens de código de exemplo dos capítulos" na página 36.

Compreensão da arquitetura do som

Seus aplicativos podem carregar dados de som de cinco origens principais:

- Arquivos de som externos carregados em tempo de execução
- Recursos de som incorporados no arquivo SWF do aplicativo
- Dados de som de um microfone conectado ao sistema do usuário
- Dados de som transmitidos de um servidor de mídia remoto, como o Flash Media Server
- Dados de som gerados dinamicamente por meio do uso do manipulador de eventos `sampleData`

Dados de som podem ser carregados completamente antes de o som ser reproduzido ou podem ser transmitidos em fluxo, o que significa que o som é reproduzido enquanto ainda está sendo carregado.

As classes de som do ActionScript 3.0 dão suporte a arquivos de som armazenados no formato mp3. Elas não podem carregar ou reproduzir arquivos de som diretamente em outros formatos, como WAV ou AIFF. No entanto, a partir do Flash Player 9.0.115.0, é possível carregar e reproduzir arquivos de áudio AAC usando a classe NetStream. Esta técnica é a mesma utilizada para carregar e reproduzir conteúdo de vídeo. Para obter mais informações sobre ela, consulte "Trabalho com vídeo" na página 527.

Com o Adobe Flash CS4 Professional, é possível importar arquivos de som WAV ou AIFF e incorporá-los aos arquivos SWF do aplicativo no formato mp3. A ferramenta de autoria do Flash também permite compactar arquivos de som incorporados para reduzir o tamanho do arquivo, embora essa redução de tamanho represente perda da qualidade do som. Para obter mais informações, consulte "Importação de sons" em *Uso do Flash*.

A arquitetura de som do ActionScript 3.0 usa as seguintes classes no pacote flash.media.

Classe	Descrição
flash.media.Sound	A classe Sound manipula o carregamento do som, gerencia propriedades básicas de som e inicia uma reprodução de som.
flash.media.SoundChannel	Quando um aplicativo reproduz um objeto Sound, um novo objeto SoundChannel é criado para controlar a reprodução. O objeto SoundChannel controla o volume dos canais de reprodução direito e esquerdo do som. Cada som reproduzido tem seu próprio objeto SoundChannel.
flash.media.SoundLoaderContext	A classe SoundLoaderContext especifica o número de segundos de buffer a ser usado ao carregar um som e se o Flash Player ou o AIR deve procurar um arquivo de política no servidor quando carregar um arquivo. Um objeto SoundLoaderContext é usado como um parâmetro para o método `Sound.load()`.
flash.media.SoundMixer	A classe SoundMixer controla as propriedades de reprodução e de segurança relativas a todos os sons em um aplicativo. Em vigor, vários canais de som são misturados por meio de um objeto SoundMixer comum, portanto valores de propriedades no objeto SoundMixer afetarão todos os objetos SoundChannel em execução no momento.
flash.media.SoundTransform	A classe SoundTransform contém valores que controlam o volume e o panorama do som. Um objeto SoundTransform pode ser aplicado a um objeto SoundChannel individual, ao objeto SoundMixer global ou a um objeto Microphone, entre outros.
flash.media.ID3Info	Um objeto ID3Info contém propriedades que representam informações de metadados ID3 que normalmente são armazenados em arquivos de som mp3.
flash.media.Microphone	A classe Microphone representa um microfone ou outro dispositivo de entrada de som conectado ao computador do usuário. A entrada de áudio de um microfone pode ser roteada para alto-falantes locais ou enviada a um servidor remoto. O objeto Microphone controla o ganho, a taxa de amostragem e outras características de seu próprio fluxo de som.

Cada som carregado e reproduzido precisa de sua própria ocorrência das classes Sound e SoundChannel. A saída das várias ocorrências de SoundChannel é então misturada pela classe global SoundMixer durante a reprodução.

As classes Sound, SoundChannel e SoundMixer não são usadas para dados de som obtidos de um microfone ou de um servidor de fluxo de mídia, como o Flash Media Server.

Carregamento de arquivos de som externos

Cada ocorrência da classe Sound existe para carregar e disparar a reprodução de um recurso de som específico. Um aplicativo não pode reutilizar um objeto Sound para carregar mais de um som. Para carregar um novo recurso de som, ele deve criar um novo objeto Sound.

Se estiver carregando um arquivo de som pequeno, como um som de clique a ser conectado a um botão, o aplicativo poderá criar um novo Sound e fazer com que ele carregue automaticamente o arquivo de som, conforme mostrado a seguir:

```
var req:URLRequest = new URLRequest("click.mp3");
var s:Sound = new Sound(req);
```

O construtor `Sound()` aceita um objeto URLRequest como seu primeiro parâmetro. Quando um valor é fornecido ao parâmetro URLRequest, o novo objeto Sound começa a carregar o recurso de som especificado automaticamente.

Em todos, menos nos casos mais simples, o aplicativo deve verificar se ocorrem erros durante o carregamento do som. Por exemplo, se o som de clique for razoavelmente grande, ele talvez não esteja completamente carregado quando o usuário clicar no botão que dispara o som. A tentativa de reproduzir um som não carregado pode provocar um erro em tempo de execução. É mais seguro aguardar que o carregamento do som seja concluído antes de permitir que os usuários executem ações que podem iniciar a reprodução de sons.

Um objeto Sound despacha vários eventos diferentes durante o processo de carregamento do som. O aplicativo pode ouvir esses eventos para controlar o progresso de carregamento e verificar se o som foi completamente carregado antes da reprodução. A tabela a seguir lista os eventos que podem ser despachados por um objeto Sound.

Evento	Descrição
abertura (`Event.OPEN`)	Despachado imediatamente antes do início da operação de carregamento do som.
progresso (`ProgressEvent.PROGRESS`)	Despachado periodicamente durante o processo de carregamento do som quando os dados são recebidos do arquivo ou do fluxo.
id3 (`Event.ID3`)	Despachado quando dados ID3 estão disponíveis para um som mp3.
concluído (`Event.COMPLETE`)	Despachado quando todos os dados do recurso de som foram carregados.
ioError (`IOErrorEvent.IO_ERROR`)	Despachado quando um arquivo de som não pode ser localizado ou quando o processo de carregamento é interrompido antes dos dados serem recebidos.

O código a seguir ilustra como reproduzir um som após a conclusão do carregamento:

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;

var s:Sound = new Sound();
s.addEventListener(Event.COMPLETE, onSoundLoaded);
var req:URLRequest = new URLRequest("bigSound.mp3");
s.load(req);

function onSoundLoaded(event:Event):void
{
    var localSound:Sound = event.target as Sound;
    localSound.play();
}
```

Primeiro, o código de amostra cria um novo objeto Sound sem dar a ele um valor inicial para o parâmetro URLRequest. Em seguida, ele ouve o evento `Event.COMPLETE` do objeto Sound, o que faz com que o método `onSoundLoaded()` seja executado quando todos os dados do som estão carregados. Em seguida, ele chama o método `Sound.load()` com um novo valor de URLRequest para o arquivo de som.

O método `onSoundLoaded()` é executado quando o carregamento do som é concluído. A propriedade `target` do objeto Event é uma referência ao objeto Sound. A chamada do método `play()` do objeto Sound inicia a reprodução do som.

Monitoramento do processo de carregamento do som

Arquivos de som podem ser muito grandes e levar muito tempo para serem carregados. Embora o Flash Player e o AIR permitam que o aplicativo reproduza sons mesmo antes de eles serem completamente carregados, talvez você queira dar ao usuário uma indicação da quantidade de dados de som carregada e da quantidade de som que já foi reproduzida.

A classe Sound despacha dois eventos que facilitam relativamente a exibição do progresso de carregamento de um som: `ProgressEvent.PROGRESS` e `Event.COMPLETE`. O exemplo a seguir mostra como usar esses eventos para exibir informações de progresso sobre o som que está sendo carregado:

```
import flash.events.Event;
import flash.events.ProgressEvent;
import flash.media.Sound;
import flash.net.URLRequest;

var s:Sound = new Sound();
s.addEventListener(ProgressEvent.PROGRESS, onLoadProgress);
s.addEventListener(Event.COMPLETE, onLoadComplete);
s.addEventListener(IOErrorEvent.IO_ERROR, onIOError);

var req:URLRequest = new URLRequest("bigSound.mp3");
s.load(req);

function onLoadProgress(event:ProgressEvent):void
{
    var loadedPct:uint = Math.round(100 * (event.bytesLoaded / event.bytesTotal));
    trace("The sound is " + loadedPct + "% loaded.");
}

function onLoadComplete(event:Event):void
{
    var localSound:Sound = event.target as Sound;
    localSound.play();
}
function onIOError(event:IOErrorEvent)
{
    trace("The sound could not be loaded: " + event.text);
}
```

Esse código primeiro cria um objeto Sound e, em seguida, adiciona ouvintes para esse objeto para os eventos `ProgressEvent.PROGRESS` e `Event.COMPLETE`. Após o método `Sound.load()` ter sido chamado e os primeiros dados terem sido recebidos do arquivo de som, ocorre um evento `ProgressEvent.PROGRESS` que dispara o método `onSoundLoadProgress()`.

A porcentagem dos dados de som que foi carregada é igual ao valor da propriedade `bytesLoaded` do objeto gressEvent dividido pelo valor da propriedade `bytesTotal`. As mesmas propriedades `bytesLoaded` e `bytesTotal` estão disponíveis no objeto Sound também. O exemplo acima simplesmente mostra mensagens sobre o progresso do carregamento do som, mas é possível usar facilmente os valores de `bytesLoaded` e `bytesTotal` para atualizar os componentes da barra de progresso, como os provenientes da estrutura do Adobe Flex 3 ou da Ferramenta de autoria do Flash.

Esse exemplo também mostra como um aplicativo pode reconhecer e responder a um erro ao carregar arquivos de som. Por exemplo, se um arquivo de som com o nome fornecido não puder ser localizado, um evento `Event.IO_ERROR` será despachado pelo objeto Sound. No código anterior, o método `onIOError()` executa e exibe uma breve mensagem de erro quando ocorre um erro.

Trabalho com sons incorporados

O uso de sons incorporados, em vez do carregamento de som de um arquivo externo, é mais útil para pequenos sons que são usados como indicadores dentro da interface do usuário do aplicativo, como sons que são reproduzidos quando botões são clicados.

Quando um arquivo de som é incorporado no aplicativo, o tamanho do arquivo SWF resultante aumenta pelo tamanho do arquivo de som. Em outras palavras, a incorporação de arquivos de som grandes no aplicativo pode aumentar o tamanho do arquivo SWF para um tamanho indesejado.

O método exato a ser usado para incorporar um arquivo de som no arquivo SWF do aplicativo varia de acordo com o ambiente de desenvolvimento.

Uso de um arquivo de som incorporado no Flash

A ferramenta de autoria do Flash permite importar sons de vários formatos e armazená-los como símbolos na Biblioteca. Em seguida, é possível atribuí-los a quadros na linha do tempo ou a quadros de um estado de botão, usá-los com Comportamentos ou usá-los diretamente no código do ActionScript. Esta seção descreve como usar sons incorporados no código ActionScript com a ferramenta de autoria do Flash. Para obter informações sobre as outras maneiras de usar sons incorporados no Flash, consulte "Importação de sons" em *Uso do Flash*.

Para incorporar um arquivo de som usando a Ferramenta de autoria do Flash:

1. Selecione Arquivo > Importar > Importar para biblioteca e, em seguida, selecione e importe um arquivo de som.
2. Clique com o botão direito do mouse no nome do arquivo importado no painel Biblioteca e selecione Propriedades. Clique na caixa de seleção Exportar para ActionScript.
3. No campo Classe, digite um nome a ser usado ao fazer referência a esse som incorporado no ActionScript. Por padrão, ele usará o nome do arquivo de som nesse campo. Se o nome do arquivo incluir um ponto, como no nome "DrumSound.mp3", você deverá alterá-lo para algo como "DrumSound". O ActionScript não permite um caractere ponto em um nome de classe. O campo Classe base ainda deve mostrar flash.media.Sound.
4. Clique em OK. Talvez você veja uma caixa de diálogo informando que uma definição dessa classe não pôde ser localizada no caminho de classe. Clique em OK e continue. Se você inseriu um nome de classe que não corresponde ao nome de qualquer uma das classes no caminho de classe do aplicativo, uma nova classe que herda da classe flash.media.Sound será gerada automaticamente para você.
5. Para usar o som incorporado, você faz referência ao nome da classe para aquele som no ActionScript. Por exemplo, o código a seguir começa criando uma nova ocorrência da classe DrumSound gerada automaticamente:

```
var drum:DrumSound = new DrumSound();
var channel:SoundChannel = drum.play();
```

DrumSound é uma subclasse da classe flash.media.Sound portanto ela herda os métodos e as propriedades da classe Sound, incluindo o método `play()` conforme mostrado acima.

Trabalho com arquivos de fluxo de som

A reprodução de um arquivo de som ou de vídeo enquanto seus dados ainda estão sendo carregados é conhecida como *streaming*. Arquivos de som externos que são carregados de um servidor remoto normalmente são transmitidos em fluxo para que o usuário não precise aguardar até que todos os dados do som sejam carregados para ouvir o som.

A propriedade `SoundMixer.bufferTime` representa o número de milissegundos de dados de som que o Flash Player ou o AIR deve coletar antes de permitir a reprodução do som. Em outras palavras, se a propriedade `bufferTime` estiver definida como 5000, o Flash Player ou o AIR carregarão pelo menos 5000 milissegundos de dados do arquivo de som antes de iniciar a reprodução do som. O valor padrão `SoundMixer.bufferTime` é 1000.

O aplicativo pode substituir o valor global `SoundMixer.bufferTime` de um som individual especificando explicitamente um novo valor de `bufferTime` ao carregar o som. Para substituir o tempo de buffer padrão, primeiro crie uma nova ocorrência da classe SoundLoaderContext, defina sua propriedade `bufferTime` e, em seguida, passe-a como um parâmetro para o método `Sound.load()`, conforme mostrado a seguir:

```
import flash.media.Sound;
import flash.media.SoundLoaderContext;
import flash.net.URLRequest;

var s:Sound = new Sound();
var req:URLRequest = new URLRequest("bigSound.mp3");
var context:SoundLoaderContext = new SoundLoaderContext(8000, true);
s.load(req, context);
s.play();
```

Conforme a reprodução continua, o Flash Player e o AIR tentam manter o buffer de som do mesmo tamanho ou maior. Se os dados do som forem carregados mais rapidamente do que a velocidade de reprodução, a reprodução continuará sem interrupção. No entanto, se a taxa de carregamento de dados for reduzida devido a limitações da rede, o indicador de reprodução poderá atingir o final do buffer de som. Nesse caso, a reprodução será suspensa, embora ela seja reiniciada automaticamente quando mais dados de som forem carregados.

Para descobrir se a reprodução está suspensa porque o Flash Player ou o AIR estão aguardando o carregamento dos dados, use a propriedade `Sound.isBuffering`.

Trabalho com áudio gerado dinamicamente

Em vez de carregar ou transmitir em fluxo um som existente, é possível gerar dados de áudio dinamicamente. Você pode gerar dados de áudio quando estiver atribuindo um ouvinte de eventos ao evento `sampleData` de um objeto Sound. (O evento `sampleData` é definido na classe SampleDataEvent do pacote flash.events.) Neste ambiente, o objeto Sound não carrega dados de som de um arquivo. Em vez disso, ele funciona como um soquete para dados de som que estão sendo transmitidos em fluxo através do uso da função que você atribui a esse evento.

Quando você adiciona um ouvinte de eventos `sampleData` a um objeto Sound, o objeto periodicamente solicita dados para adicioná-los ao buffer de som. Esse buffer contém dados para o objeto Sound reproduzir. Quando você chama o método `play()` do objeto Sound, ele despacha o evento `sampleData` ao solicitar novos dados de som. (Isto só se aplica quando o objeto Sound não carregou dados mp3 de um arquivo.)

O objeto SampleDataEvent inclui uma propriedade `data`. No ouvinte de eventos, você grava objetos ByteArray neste objeto `data`. As matrizes de bytes gravadas neste objeto somam-se aos dados de som em buffer reproduzidos pelo objeto Sound. A matriz de bytes no buffer é um fluxo de valores de ponto flutuante que variam de -1 a 1. Cada valor de ponto flutuante representa a amplitude de um canal (esquerdo ou direito) de uma amostra de som. A amostragem do som ocorre a 44.100 amostras por segundo. Cada amostra contém um canal esquerdo e direito, intercalados como dados de ponto flutuante na matriz de bytes.

Na função do manipulador, use o método `ByteArray.writeFloat()` para gravar na propriedade `data` do evento `sampleData`. Por exemplo, o seguinte código gera uma onda senoidal:

```
var mySound:Sound = new Sound();
mySound.addEventListener(SampleDataEvent.SAMPLE_DATA, sineWaveGenerator);
mySound.play();
function sineWaveGenerator(event:SampleDataEvent):void
{
    for (var i:int = 0; i < 8192; i++)
    {
        var n:Number = Math.sin((i + event.position) / Math.PI / 4);
        event.data.writeFloat(n);
        event.data.writeFloat(n);
    }
}
```

Quando você chama `Sound.play()`, o aplicativo começa a chamar seu manipulador de eventos, solicitando dados de amostra de som. O aplicativo continua enviando eventos à medida que o som é reproduzido, até você parar de fornecer dados ou chamar `SoundChannel.stop()`.

A latência do evento varia de uma plataforma para outra e pode mudar em versões futuras do Flash Player e do AIR. Não dependa de uma latência específica; em vez disso, você deve calculá-la. Para fazer esse cálculo, use a seguinte fórmula:

```
(SampleDataEvent.position / 44.1) - SoundChannelObject.position
```

Forneça de 2048 a 8192 amostras para a propriedade `data` do objeto SampleDataEvent (para cada chamada ao ouvinte de eventos). Para obter o melhor desempenho, forneça o máximo possível de amostras (até 8192). Quanto menos amostras forem fornecidas, maior será a probabilidade de ocorrerem cliques e estalos durante a reprodução. Esse comportamento pode diferir em várias plataformas e pode ocorrer em diversas situações; por exemplo, no redimensionamento do navegador. O código que funciona em uma plataforma quando você fornece apenas 2048 amostras pode não funcionar tão bem quando executado em outra plataforma. Se precisar da menor latência possível, pense na possibilidade de tornar o volume de dados selecionável pelo usuário.

Se você fornecer menos de 2048 amostras (por chamada ao ouvinte de eventos `sampleData`), o aplicativo será interrompido depois de reproduzir as amostras restantes. Em seguida, ele despachará um evento SoundComplete.

Modificação do som de dados mp3

Use o método `Sound.extract()` para extrair dados de um objeto Sound. Você pode usar (e modificar) esses dados para gravá-los no fluxo dinâmico de outro objeto Sound para fins de reprodução. Por exemplo, o código a seguir usa os bytes de um arquivo MP3 carregado e os passa usando uma função de filtro, `upOctave()`:

```
var mySound:Sound = new Sound();
var sourceSnd:Sound = new Sound();
var urlReq:URLRequest = new URLRequest("test.mp3");
sourceSnd.load(urlReq);
sourceSnd.addEventListener(Event.COMPLETE, loaded);
function loaded(event:Event):void
{
    mySound.addEventListener(SampleDataEvent.SAMPLE_DATA, processSound);
    mySound.play();
}
function processSound(event:SampleDataEvent):void
{
        var bytes:ByteArray = new ByteArray();
        sourceSnd.extract(bytes, 8192);
        event.data.writeBytes(upOctave(bytes));
}
function upOctave(bytes:ByteArray):ByteArray
{
    var returnBytes:ByteArray = new ByteArray();
    bytes.position = 0;
    while(bytes.bytesAvailable > 0)
    {
        returnBytes.writeFloat(bytes.readFloat());
        returnBytes.writeFloat(bytes.readFloat());
        if (bytes.bytesAvailable > 0)
        {
            bytes.position += 8;
        }
    }
    return returnBytes;
}
```

Limitações quanto aos sons gerados

Quando você usa um ouvinte de eventos `sampleData` com um objeto Sound, os outros únicos métodos Sound ativados são `Sound.extract()` e `Sound.play()`. Chamar qualquer outro método ou propriedade resultará em uma exceção. Todos os métodos e propriedades do objeto SoundChannel continuam ativados.

Reprodução de sons

A reprodução de um som carregado pode ser tão simples quanto chamar o método `Sound.play()` para um objeto Sound, da seguinte maneira:

```
var snd:Sound = new Sound(new URLRequest("smallSound.mp3"));
snd.play();
```

Ao reproduzir sons usando o ActionScript 3.0, é possível executar as seguintes operações:

- Reproduzir um som a partir de uma posição inicial específica.
- Pausar um som e reiniciar a reprodução a partir da mesma posição mais tarde.
- Saber exatamente quando a reprodução de um som é concluída.
- Rastrear o progresso da reprodução de um som.

- Alterar o volume ou a panorâmica enquanto o som é reproduzido.

Para executar essas operações durante a reprodução, use as classes SoundChannel, SoundMixer e SoundTransform.

A classe SoundChannel controle a reprodução de um único som. A propriedade `SoundChannel.position` pode ser considerada como um indicador de reprodução, indicando o ponto atual nos dados do som que está sendo reproduzido.

Quando um aplicativo chama o método `Sound.play()`, uma nova ocorrência da classe SoundChannel é criada para controlar a reprodução.

O aplicativo pode reproduzir um som a partir de uma posição inicial específica passando aquela posição, em termos de milissegundos, como o parâmetro `startTime` do método `Sound.play()`. Ele também pode especificar o número de vezes para repetição do som em sucessão rápida passando um valor numérico no parâmetro `loops` do método `Sound.play()`.

Quando o método `Sound.play()` é chamado com um parâmetro `startTime` e um parâmetro `loops`, o som é reproduzido repetidamente a partir do mesmo ponto inicial, conforme mostrado no código a seguir:

```
var snd:Sound = new Sound(new URLRequest("repeatingSound.mp3"));
snd.play(1000, 3);
```

Neste exemplo, o som é reproduzido a partir de um ponto um segundo após o início do som, três vezes em sucessão.

Pausa e reinício de um som

Se o aplicativo reproduzir sons longos, como canções ou podcasts, talvez você queira permitir que os usuários pausem e reiniciem a reprodução desses sons. Um som não pode ser pausado literalmente durante a reprodução no ActionScript. Ele pode apenas ser interrompido. No entanto um som pode ser reproduzido a partir de um ponto qualquer. É possível registrar a posição do som na hora em que foi interrompido e, em seguida, reproduzir o som mais tarde a partir daquela posição.

Por exemplo, suponha que o código carrega e reproduz um arquivo de som como este:

```
var snd:Sound = new Sound(new URLRequest("bigSound.mp3"));
var channel:SoundChannel = snd.play();
```

Enquanto o som é reproduzido, a propriedade `SoundChannel.position` indica o ponto no arquivo de som que está em reprodução no momento. O aplicativo pode armazenar o valor da posição antes de interromper a reprodução do som, da seguinte maneira:

```
var pausePosition:int = channel.position;
channel.stop();
```

Para reiniciar a reprodução do som, passe o valor da posição armazenado anteriormente para reiniciar o som a partir do mesmo ponto em que foi interrompido anteriormente.

```
channel = snd.play(pausePosition);
```

Monitoramento da reprodução

O aplicativo talvez queira saber quando a reprodução de um som é interrompida para que possa iniciar a reprodução de outro som ou limpar alguns recursos usados durante a reprodução anterior. A classe SoundChannel despacha um evento `Event.SOUND_COMPLETE` quando a reprodução do som é concluída. O aplicativo pode ouvir esse evento e tomar a ação apropriada, conforme mostrado a seguir:

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;

var snd:Sound = new Sound();
var req:URLRequest = new URLRequest("smallSound.mp3");
snd.load(req);

var channel:SoundChannel = snd.play();
channel.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

public function onPlaybackComplete(event:Event)
{
    trace("The sound has finished playing.");
}
```

A classe SoundChannel não despacha eventos de progresso durante a reprodução. Para relatar o progresso da reprodução, o aplicativo pode configurar seu próprio mecanismo de controle de tempo e rastrear a posição do indicador de reprodução do som.

Para calcular qual porcentagem de um som foi reproduzida, é possível dividir o valor da propriedade `SoundChannel.position` pelo comprimento dos dados do som que está sendo reproduzido:

```
var playbackPercent:uint = 100 * (channel.position / snd.length);
```

No entanto esse código relatará porcentagens exatas da reprodução apenas se os dados do som foram carregados completamente antes do início da reprodução. A propriedade `Sound.length` mostra o tamanho dos dados do som que estão sendo carregados no momento, não o tamanho eventual do arquivo de som inteiro. Para rastrear o progresso da reprodução de um fluxo de som que ainda está sendo carregado, o aplicativo deve estimar o tamanho eventual do arquivo de som inteiro e usar esse valor em seus cálculos. É possível estimar o comprimento eventual dos dados do som usando as propriedades `bytesLoaded` e `bytesTotal` do objeto Sound, da seguinte maneira:

```
var estimatedLength:int =
    Math.ceil(snd.length / (snd.bytesLoaded / snd.bytesTotal));
var playbackPercent:uint = 100 * (channel.position / estimatedLength);
```

O código a seguir carrega um arquivo de som maior e usa o evento `Event.ENTER_FRAME` como seu mecanismo de controle de tempo para mostrar o progresso da reprodução. Ele relata periodicamente a porcentagem de reprodução que é calculada como o valor da posição inicial dividido pelo comprimento total dos dados do som:

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;

var snd:Sound = new Sound();
var req:URLRequest = new
    URLRequest("http://av.adobe.com/podcast/csbu_dev_podcast_epi_2.mp3");
snd.load(req);

var channel:SoundChannel;
channel = snd.play();
addEventListener(Event.ENTER_FRAME, onEnterFrame);
channel.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

function onEnterFrame(event:Event):void
{
    var estimatedLength:int =
        Math.ceil(snd.length / (snd.bytesLoaded / snd.bytesTotal));
    var playbackPercent:uint =
        Math.round(100 * (channel.position / estimatedLength));
    trace("Sound playback is " + playbackPercent + "% complete.");
}

function onPlaybackComplete(event:Event)
{
    trace("The sound has finished playing.");
    removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

Após o início do carregamento dos dados do som, esse código chama o método `snd.play()` e armazena o objeto SoundChannel resultante na variável `channel`. Em seguida, ele adiciona um ouvinte de eventos ao aplicativo principal para o evento `Event.ENTER_FRAME` e outro ouvinte de eventos ao objeto SoundChannel para o evento `Event.SOUND_COMPLETE` que ocorre quando a reprodução é concluída.

A cada vez que o aplicativo atinge um novo quadro em sua animação, o método `onEnterFrame()` é chamado. O método `onEnterFrame()` estima o comprimento total do arquivo de som com base na quantidade de dados que já foi carregada e, em seguida, calcula e exibe a porcentagem de reprodução atual.

Quando todo o som foi reproduzido, o método `onPlaybackComplete()` é executado, removendo o ouvinte do evento `Event.ENTER_FRAME` de forma que ele não tente exibir atualizações do progresso após a conclusão da reprodução.

O evento `Event.ENTER_FRAME` pode ser despachado muitas vezes por segundo. Em alguns casos, não é conveniente exibir o progresso da reprodução com tanta freqüência. Nesses casos, o aplicativo pode configurar seu próprio mecanismo de controle do tempo usando a classe flash.util.Timer. Consulte “Trabalho com datas e horas” na página 134.

Interrupção de fluxos de som

Há uma peculiaridade no processo de reprodução de sons que estão sendo transmitidos em fluxo, isto é, sons que ainda estão sendo carregados enquanto estão sendo reproduzidos. Quando o aplicativo chama o método `SoundChannel.stop()` em uma ocorrência de SoundChannel que está reproduzindo um fluxo de som, a reprodução do som pára em um quadro e, em seguida, no próximo quadro, ela reinicia a partir do início do som. Isso ocorre porque o processo de carregamento do som ainda está em execução. Para interromper o carregamento e a reprodução de um fluxo de som, chame a o método `Sound.close()`.

Considerações sobre segurança ao carregar e reproduzir sons

A capacidade do aplicativo de acessar dados de som podem ser limitadas de acordo com o modelo de segurança do Flash Player ou do AIR. Cada som está sujeito a restrições de duas caixas de proteção de segurança, a caixa de proteção do próprio conteúdo (a "caixa de proteção do conteúdo") e a caixa de proteção do aplicativo ou do objeto que carrega e reproduz o som (a "caixa de proteção do proprietário"). Para conteúdo de aplicativo AIR na caixa de proteção do aplicativo, todos os sons, inclusive aqueles carregados de outros domínios, são acessíveis ao conteúdo na caixa de proteção de segurança do aplicativo. No entanto conteúdo em outras caixas de proteção de segurança observam as mesmas regras que o conteúdo em execução no Flash Player. Para obter mais informações sobre o modelo de segurança do Flash Player em geral e sobre a definição de caixas de proteção, consulte "Segurança do Flash Player" na página 704.

A caixa de proteção do conteúdo controla se dados de som detalhados podem ser extraídos do som usando a propriedade `id3` ou o método `SoundMixer.computeSpectrum()`. Ele não restringe o carregamento ou a reprodução do próprio arquivo de som.

O domínio de origem do arquivo de som define as limitações de segurança da caixa de proteção do conteúdo. Geralmente, se um arquivo de som está localizado no mesmo domínio ou pasta que o arquivo SWF do aplicativo ou do objeto que o carrega, o aplicativo ou o objeto terá acesso total àquele arquivo de som. Se o som for proveniente de um domínio diferente do domínio do aplicativo, ele ainda poderá ser trazido para a caixa de proteção do conteúdo usando um arquivo de política.

O aplicativo pode passar um objeto SoundLoaderContext com uma propriedade `checkPolicyFile` como parâmetro para o método `Sound.load()`. A configuração da propriedade `checkPolicyFile` como `true` instrui o Flash Player ou o AIR a procurar um arquivo de política no servidor onde o som é carregado. Se existir um arquivo de política e ele conceder acesso ao domínio do arquivo SWF que está sendo carregado, o arquivo SWF poderá carregar o arquivo de som, acessar a propriedade `id3` do objeto Sound e chamar o método `SoundMixer.computeSpectrum()` de sons carregados.

A caixa de proteção do proprietário controla a reprodução local dos sons. O aplicativo ou o objeto que inicia a reprodução de um som define a caixa de proteção do proprietário.

O método `SoundMixer.stopAll()` silencia os sons em todos os objetos SoundChannel que estão sendo reproduzidos no momento, desde que atendam aos seguintes critérios:

- Os sons foram iniciados por objetos dentro da mesma caixa de proteção do proprietário.
- Os sons são provenientes de uma origem com um arquivo de política que concede acesso ao domínio do aplicativo ou objeto que chama o método `SoundMixer.stopAll()`.

Contudo, em um aplicativo AIR, o conteúdo na caixa de proteção de segurança do aplicativo (conteúdo instalado com o aplicativo AIR) não é restringido por essas limitações de segurança.

Para descobrir se o método `SoundMixer.stopAll()` realmente interromperá todas os sons em reprodução, o aplicativo pode chamar o método `SoundMixer.areSoundsInaccessible()`. Se esse método retornar um valor de `true`, alguns dos sons que estão sendo reproduzidos estarão fora do controle da caixa de proteção do proprietário atual e não serão interrompidos pelo método `SoundMixer.stopAll()`.

O método `SoundMixer.stopAll()` também interrompe a continuação do indicador da reprodução para todos os sons que foram carregados de arquivos externos. No entanto sons que foram incorporados em arquivos FLA e anexados a quadros na linha do tempo usando a ferramenta de autoria do Flash talvez iniciem a reprodução novamente se a animação mover para um novo quadro.

Controle do volume e do panorama do som

Um objeto SoundChannel individual controla os canais estéreos esquerdo e direito para um som. Se um som mp3 for um som monofônico, os canais estéreos esquerdo e direito do objeto SoundChannel conterão formas de onda idênticas.

É possível descobrir a amplitude de cada canal estéreo do som que está sendo reproduzido usando as propriedades `leftPeak` e `rightPeak` do objeto SoundChannel. Essas propriedades mostram a amplitude de pico da própria forma de onda do som. Elas não representam o volume de reprodução real. O volume de reprodução real é uma função da amplitude da onda do som e dos valores de volume definidos no objeto SoundChannel e na classe SoundMixer.

A propriedade pan de um objeto SoundChannel pode ser usada para especificar um nível diferente de volume para cada um dos canais esquerdo e direito durante a reprodução. A propriedade pan pode ter um valor variando de -1 a 1, em que -1 significa que o canal esquerdo reproduz no volume máximo enquanto o canal direito está silencioso e 1 significa que o canal direito reproduz no volume máximo enquanto o canal esquerdo está silencioso. Valores numéricos entre -1 e 1 definem valores proporcionais para os valores dos canais esquerdo e direito e um valor igual a 0 significa que os dois canais são reproduzidos em um nível equilibrado de nível de volume médio.

O código de exemplo a seguir cria um objeto SoundTransform com um valor de volume de 0,6 e um valor de pan de -1 (volume máximo no canal esquerdo e nenhum volume no canal direito). Ele passa o objeto SoundTransform como um parâmetro para o método `play()` que aplica esse objeto SoundTransform ao novo objeto SoundChannel que é criado para controlar a reprodução.

```
var snd:Sound = new Sound(new URLRequest("bigSound.mp3"));
var trans:SoundTransform = new SoundTransform(0.6, -1);
var channel:SoundChannel = snd.play(0, 1, trans);
```

É possível alterar o volume e a panorâmica enquanto um som está sendo reproduzido definindo as propriedades `pan` ou `volume` de um objeto SoundTransform e aplicando esse objeto como a propriedade `soundTransform` de um objeto SoundChannel.

Também é possível definir os valores globais de volume e de pan para todos os sons de uma vez usando a propriedade `soundTransform` da classe SoundMixer, conforme mostrado no exemplo a seguir:

```
SoundMixer.soundTransform = new SoundTransform(1, -1);
```

Também é possível usar um objeto SoundTransform para definir os valores de volume e panorâmica para um objeto Microphone (consulte “Captura de entrada do som” na página 589) e para objetos Sprite e SimpleButton.

O exemplo a seguir alterna a panorâmica do som do canal esquerdo para o canal direito e vice-versa enquanto o som é reproduzido.

```
import flash.events.Event;
import flash.media.Sound;
import flash.media.SoundChannel;
import flash.media.SoundMixer;
import flash.net.URLRequest;

var snd:Sound = new Sound();
var req:URLRequest = new URLRequest("bigSound.mp3");
snd.load(req);

var panCounter:Number = 0;

var trans:SoundTransform;
trans = new SoundTransform(1, 0);
var channel:SoundChannel = snd.play(0, 1, trans);
channel.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

addEventListener(Event.ENTER_FRAME, onEnterFrame);

function onEnterFrame(event:Event):void
{
    trans.pan = Math.sin(panCounter);
    channel.soundTransform = trans; // or SoundMixer.soundTransform = trans;
    panCounter += 0.05;
}

function onPlaybackComplete(event:Event):void
{
    removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

Esse código é iniciado carregando um arquivo de som e criando um novo objeto SoundTransform com o volume definido como 1 (volume total) e panorâmica definida como 0 (equilibrada igualmente entre a esquerda e a direita). Em seguida, ele chama o método `snd.play()`, passando o objeto SoundTransform como um parâmetro.

Enquanto o som é reproduzido, o método `onEnterFrame()` é executado repetidamente. O método `onEnterFrame()` usa a função `Math.sin()` para gerar um valor entre -1 e 1, uma faixa que corresponde aos valores aceitáveis da propriedade `SoundTransform.pan`. A propriedade `pan` do objeto SoundTransform é definida como o novo valor e, em seguida, a propriedade `soundTransform` do canal é definida para usar o objeto SoundTransform alterado.

Para executar esse exemplo, substitua o nome do arquivo bigSound.mp3 pelo nome de um arquivo mp3 local. Em seguida, execute o exemplo. Você deve ouvir o volume do canal esquerdo se tornando mais alto enquanto o volume do canal direito se torna mais suave e vice-versa.

Nesse exemplo, o mesmo efeito pode ser obtido definindo a propriedade `soundTransform` da classe SoundMixer. No entanto isso afetará a panorâmica de todos os sons que estão sendo reproduzidos no momento, não apenas o único som que está sendo reproduzido pelo objeto SoundChannel.

Trabalho com metadados de som

Arquivos de som que usam o formato mp3 podem conter dados adicionais sobre o som na forma de tags ID3.

Nem todo arquivo mp3 contém metadados ID3. Quando um objeto Sound carrega um arquivo de som mp3, ele despachará um evento `Event.ID3` se o arquivo de som contiver metadados ID3. Para evitar erros em tempo de execução, o aplicativo deve aguardar até receber o evento `Event.ID3` antes de acessar a propriedade `Sound.id3` de um som carregado.

O código a seguir mostra como reconhecer quando os metadados ID3 de um arquivo de som foram carregados:

```
import flash.events.Event;
import flash.media.ID3Info;
import flash.media.Sound;

var s:Sound = new Sound();
s.addEventListener(Event.ID3, onID3InfoReceived);
s.load("mySound.mp3");

function onID3InfoReceived(event:Event)
{
    var id3:ID3Info = event.target.id3;

    trace("Received ID3 Info:");
    for (var propName:String in id3)
    {
        trace(propName + " = " + id3[propName]);
    }
}
```

Esse código começa criando um objeto Sound e indicando que ele ouça o evento `Event.ID3`. Quando os metadados ID3 do arquivo de som são carregados, o método `onID3InfoReceived()` é chamado. O destino do objeto Event que é passado para o método `onID3InfoReceived()` é o objeto Sound original, portanto o método obtém a propriedade `id3` do objeto Sound e percorre todas as suas propriedades nomeadas para rastrear seus valores.

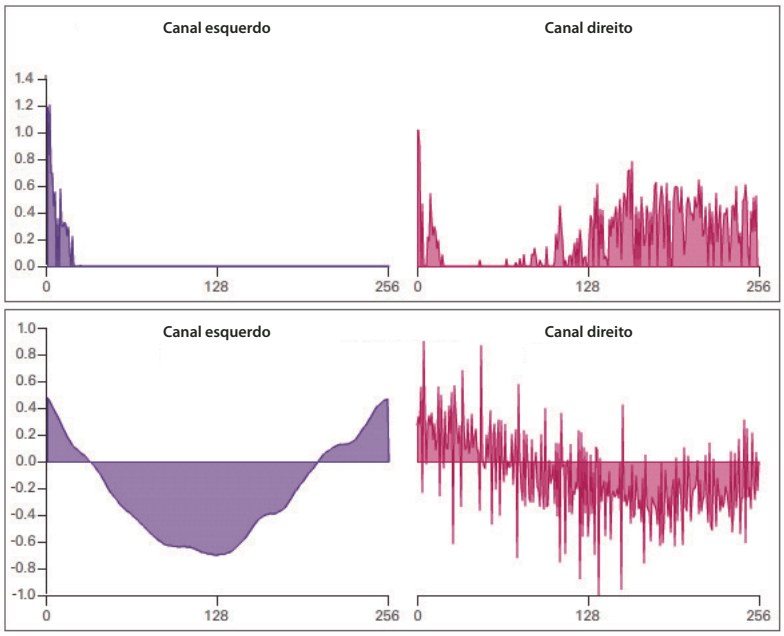
Acesso a dados de som brutos

O método `SoundMixer.computeSpectrum()` permite que um aplicativo leia os dados de som brutos para a forma de onda que está sendo reproduzida no momento. Se mais de um objeto SoundChannel estiver sendo reproduzido, o método `SoundMixer.computeSpectrum()` mostrará os dados de som combinados de cada objeto SoundChannel misturado.

Os dados de som são retornados como um objeto ByteArray que contém 512 bytes de dados, cada um deles contendo um valor de ponto flutuante entre -1 e 1. Esses valores representam a amplitude dos pontos na forma de onda do som que está sendo reproduzido. Os valores são entregues em dois grupos de 256, o primeiro grupo do canal estéreo esquerdo e o segundo grupo do canal estéreo direito.

O método `SoundMixer.computeSpectrum()` retornará os dados do espectro da freqüência em vez dos dados da forma de onda se o parâmetro `FFTMode` estiver definido como `true`. O espectro da freqüência mostra a amplitude organizada pela freqüência do som, da freqüência mais baixa para a mais alta. Uma FFT (Transformação rápida de Fourier) é usada para converter os dados da forma de onda nos dados do espectro da freqüência. Os valores do espectro da freqüência resultantes variam de 0 a aproximadamente 1,414 (a raiz quadrada de 2).

O diagrama a seguir compara os dados retornados do método `computeSpectrum()` quando o parâmetro `FFTMode` está definido como `true` e quando está definido como `false`. O som cujos dados foram usados para esse diagrama contém um som grave alto no canal esquerdo e um som de toque de tambor no canal direito.



Valores retornados pelo método SoundMixer.computeSpectrum()
***A.** fftMode=true **B.** fftMode=false*

O método `computeSpectrum()` também pode retornar dados que foram amostrados novamente em uma taxa de bits mais baixa. Geralmente, isso resulta em dados de forma de onda mais suaves ou em dados de freqüência em detrimento de detalhes. O parâmetro `stretchFactor` controla a taxa na qual os dados do método `computeSpectrum()` são amostrados. Quando o parâmetro `stretchFactor` está definido como 0, o padrão, os dados do som são amostrados a uma taxa de 44,1 kHz. A taxa é dividida em dois em cada valor sucessivo do parâmetro stretchFactor, portanto um valor de 1 especifica uma taxa de 22,05 kHz, um valor de 2 especifica uma taxa de 11,025 kHz e assim por diante. O método `computeSpectrum()` ainda retorna 256 bytes por canal estéreo quando um valor de `stretchFactor` mais alto é usado.

O método `SoundMixer.computeSpectrum()` tem algumas limitações:

- Como os dados do som de um microfone ou de fluxos RTMP não passam pelo objeto global SoundMixer, o método `SoundMixer.computeSpectrum()` não retorna dados dessas origens.
- Se um ou mais dos sons que estão sendo reproduzidos de origens externas à caixa de proteção do conteúdo atual, restrições de segurança farão com que o método `SoundMixer.computeSpectrum()` emita um erro. Para obter mais detalhes sobre limitações de segurança do método `SoundMixer.computeSpectrum()` consulte "Considerações sobre segurança ao carregar e reproduzir sons" na página 582 e"Acesso à mídia carregada como dados" na página 724.

Contudo, em um aplicativo AIR, o conteúdo na caixa de proteção de segurança do aplicativo (conteúdo instalado com o aplicativo AIR) não é restringido por essas limitações de segurança.

Criação de um único visualizador de som

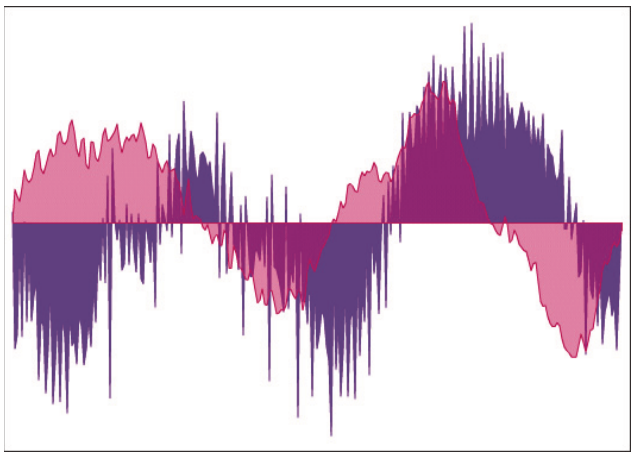
O exemplo a seguir usa o método `SoundMixer.computeSpectrum()` para mostrar um gráfico da forma de onda do som que é animado com cada quadro:

```
import flash.display.Graphics;
import flash.events.Event;
import flash.media.Sound;
import flash.media.SoundChannel;
import flash.media.SoundMixer;
import flash.net.URLRequest;

const PLOT_HEIGHT:int = 200;
const CHANNEL_LENGTH:int = 256;

var snd:Sound = new Sound();
var req:URLRequest = new URLRequest("bigSound.mp3");
snd.load(req);

var channel:SoundChannel;
channel = snd.play();
addEventListener(Event.ENTER_FRAME, onEnterFrame);
snd.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

var bytes:ByteArray = new ByteArray();

function onEnterFrame(event:Event):void
{
    SoundMixer.computeSpectrum(bytes, false, 0);

    var g:Graphics = this.graphics;

    g.clear();
    g.lineStyle(0, 0x6600CC);
    g.beginFill(0x6600CC);
    g.moveTo(0, PLOT_HEIGHT);

    var n:Number = 0;

    // left channel
    for (var i:int = 0; i < CHANNEL_LENGTH; i++)
    {
        n = (bytes.readFloat() * PLOT_HEIGHT);
        g.lineTo(i * 2, PLOT_HEIGHT - n);
    }
    g.lineTo(CHANNEL_LENGTH * 2, PLOT_HEIGHT);
```

```
    g.endFill();

    // right channel
    g.lineStyle(0, 0xCC0066);
    g.beginFill(0xCC0066, 0.5);
    g.moveTo(CHANNEL_LENGTH * 2, PLOT_HEIGHT);

    for (i = CHANNEL_LENGTH; i > 0; i--)
    {
        n = (bytes.readFloat() * PLOT_HEIGHT);
        g.lineTo(i * 2, PLOT_HEIGHT - n);
    }
    g.lineTo(0, PLOT_HEIGHT);
    g.endFill();
}

function onPlaybackComplete(event:Event)
{
    removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

Esse exemplo primeiro carrega e reproduz um arquivo de som e, em seguida, ouve o evento `Event.ENTER_FRAME` que disparará o método `onEnterFrame()` enquanto o som é reproduzido. O método `onEnterFrame()` é iniciado chamando o método `SoundMixer.computeSpectrum()` que armazena dos dados da onda do som no objeto ByteArray de `bytes`.

A forma de onda do som é plotada usando a API de desenho do vetor. Um loop `for` percorre os primeiros 256 valores de dados que representam o canal estéreo esquerdo e desenha uma linha de cada ponto até o próximo usando o método `Graphics.lineTo()`. Um segundo loop `for` percorre o próximo conjunto de 256 valores, plotando-os na ordem reversa desta vez, da direita para a esquerda. Os desenhos da forma de onda resultantes podem produzir um efeito interessante de imagem espelhada, conforme mostrado na imagem a seguir.



Captura de entrada do som

A classe Microphone permite que o aplicativo se conecte a um microfone ou a outro dispositivo de entrada de som no sistema do usuário e transmita o áudio de entrada para os alto-falantes daquele sistema ou envie os dados de áudio a um servidor remoto, como o Flash Media Server. Não é possível acessar dados de áudio brutos do microfone; você só pode enviar áudio para os alto-falantes do sistema ou enviar dados de áudio compactados para um servidor remoto. É possível usar o codec Speex ou Nellymoser para os dados enviados a um servidor remoto. (O Flash Player 10 vem com suporte para o codec Speex.)

Acesso a um microfone

A classe Microphone não tem um método construtor. Em vez disso, você usa o método estático `Microphone.getMicrophone()` para obter uma nova ocorrência de Microphone, conforme mostrado a seguir:

```
var mic:Microphone = Microphone.getMicrophone();
```

A chamada do método `Microphone.getMicrophone()` sem um parâmetro retorna o primeiro dispositivo de entrada de som descoberto no sistema do usuário.

Um sistema pode ter mais de um dispositivo de entrada de som conectado a ele. O aplicativo pode usar a propriedade `Microphone.names` para obter uma matriz dos nomes de todos os dispositivos de entrada de som disponíveis. Em seguida, ele pode chamar o método `Microphone.getMicrophone()` com um parâmetro `index` que corresponde ao valor de índice de um nome de dispositivo na matriz.

Um sistema talvez não tenha um microfone ou outro dispositivo de entrada de som conectado a ele. É possível usar a propriedade `Microphone.names` ou o método `Microphone.getMicrophone()` para verificar se o usuário tem um dispositivo de entrada de som instalado. Se o usuário não tiver um dispositivo de entrada de som instalado, a matriz `names` terá um comprimento de zero e o método `getMicrophone()` retornará um valor `null`.

Quando o aplicativo chama o método `Microphone.getMicrophone()`, o Flash Player exibe a caixa de diálogo Configurações do Flash Player que solicita que o usuário permita ou negue acesso do Flash Player à câmara e ao microfone no sistema. Depois que o usuário clicar no botão Permitir ou Negar neste diálogo, um StatusEvent será despachado. A propriedade `code` dessa ocorrência de StatusEvent indica se o acesso ao microfone foi permitido ou negado, conforme mostrado neste exemplo:

```
import flash.media.Microphone;

var mic:Microphone = Microphone.getMicrophone();
mic.addEventListener(StatusEvent.STATUS, this.onMicStatus);

function onMicStatus(event:StatusEvent):void
{
    if (event.code == "Microphone.Unmuted")
    {
        trace("Microphone access was allowed.");
    }
    else if (event.code == "Microphone.Muted")
    {
        trace("Microphone access was denied.");
    }
}
```

A propriedade `StatusEvent.code` conterá “Microphone.Unmuted”, se o acesso for permitido, ou “Microphone.Muted”, se o acesso for negado.

Nota: *A propriedade* `Microphone.muted` *é definida como* `true` *ou* `false` *quando o usuário permite ou nega acesso ao microfone, respectivamente. No entanto a propriedade* `muted` *não é definida na ocorrência de Microphone até que o StatusEvent tenha sido despachado, portanto o aplicativo também deve aguardar que o evento* `StatusEvent.STATUS` *seja despachado antes de verificar a propriedade* `Microphone.muted`*.*

Roteamento de áudio do microfone para alto-falantes locais

A entrada de áudio de um microfone pode ser roteada para os alto-falantes do sistema local chamando o método `Microphone.setLoopback()` com um valor de parâmetro de `true`.

Quando o som de um microfone local é roteado para alto-falantes locais, há um risco de criar um loop de feedback de áudio, o que pode provocar sons estridentes e danificar o hardware de som. A chamada do método `Microphone.setUseEchoSuppression()` com um valor de parâmetro `true` reduz, mas não elimina completamente, o risco de ocorrência de feedback de áudio. A Adobe recomenda sempre chamar `Microphone.setUseEchoSuppression(true)` antes de chamar `Microphone.setLoopback(true)`, a menos que você tenha certeza de que o usuário esteja reproduzindo o som usando fones de ouvido ou outra coisa que não seja um alto-falante.

O código a seguir mostra como rotear o áudio de um microfone local para os alto-falantes do sistema local:

```
var mic:Microphone = Microphone.getMicrophone();
mic.setUseEchoSuppression(true);
mic.setLoopBack(true);
```

Alteração do áudio do microfone

O aplicativo pode alterar os dados de áudio de um microfone de duas maneiras. Primeiro, ele pode alterar o ganho do som de entrada, o que efetivamente multiplica os valores de entrada por um valor especificado para criar um som mais baixo ou mais silencioso. A propriedade `Microphone.gain` aceita valores numéricos entre 0 e 100 inclusive. Um valor de 50 funciona como um multiplicador de um e especifica o volume normal. Um valor de zero funciona como um multiplicador de zero e silencia efetivamente o áudio de entrada. Valores acima de 50 especificam um volume mais alto do que o normal.

O aplicativo também pode alterar a taxa de amostragem do áudio de entrada. Taxas de amostragem mais altas aumentam a qualidade do som, mas elas também criam fluxos de dados mais densos que usam mais recursos para transmissão e armazenamento. A propriedade `Microphone.rate` representa a taxa de amostragem de áudio medida em quilohertz (kHz). A taxa de amostragem padrão é de 8 kHz. É possível definir a propriedade `Microphone.rate` como um valor mais alto do que 8 kHz se o microfone oferecer suporte à taxa mais alta. Por exemplo, configurar a propriedade `Microphone.rate` com um valor de 11 define a taxa de amostragem como 11 kHz. Configurá-la como 22 define a taxa de amostragem como 22 kHz e assim por diante. As taxas de amostra disponíveis dependem do codec selecionado. Quando você usa o codec Nellymoser, pode especificar 5, 8, 11, 16, 22 e 44 kHz como taxa de amostra. Quando você usa o codec Speex (adicionado ao Flash Player 10), só pode usar 16 kHz.

Detecção de atividade do microfone

Para conservar a largura de banda e os recursos de processamento, o Flash Player tenta detectar quando nenhum som está sendo transmitido por um microfone. Quando o nível de atividade do microfone permanece abaixo do limite do nível de silêncio por um certo período, o Flash Player pára de transmitir a entrada de áudio e despacha um simples ActivityEvent. Se você usar o codec Speex (adicionado ao Flash Player 10), defina o nível de silêncio como 0 para assegurar que o aplicativo transmita dados de áudio continuamente. A detecção da atividade de voz do Speex automaticamente reduz a largura de banda.

Três propriedades da classe Microphone monitoram e controlam a detecção de atividade:

- A propriedade somente leitura `activityLevel` indica a quantidade de som que o microfone está detectando em uma escala de 0 a 100.
- A propriedade `silenceLevel` especifica a quantidade de som necessária para ativar o microfone e despachar um evento `ActivityEvent.ACTIVITY`. A propriedade `silenceLevel` também usa uma escala de 0 a 100, e o valor padrão é 10.
- A propriedade `silenceTimeout` descreve o número de milissegundos que o nível de atividade deve permanecer abaixo do nível de silêncio, até que um evento `ActivityEvent.ACTIVITY` seja despachado para indicar que o microfone está silencioso. O valor padrão de `silenceTimeout` é 2000.

As propriedades `Microphone.silenceLevel` e `Microphone.silenceTimeout` são somente leitura, mas seus valores podem ser alterados usando o método `Microphone.setSilenceLevel()`.

Em alguns casos, o processo de ativação do microfone quando nova atividade é detectada pode provocar um pequeno atraso. Manter o microfone ativo durante todo o tempo pode remover esses atrasos na ativação. O aplicativo pode chamar o método `Microphone.setSilenceLevel()` com o parâmetro `silenceLevel` definido como zero para indicar ao Flash Player para manter o microfone ativo e continuar a coletar dados de áudio, mesmo quando nenhum som está sendo detectado. De modo oposto, a configuração do parâmetro `silenceLevel` como 100 impede que o microfone seja ativado de qualquer modo.

O exemplo a seguir exibe informações sobre o microfone e relata eventos de atividade e eventos de status despachados por um objeto Microphone:

```
import flash,events.ActivityEvent;
import flash,events.StatusEvent;
import flash.media.Microphone;

var deviceArray:Array = Microphone.names;
trace("Available sound input devices:");
for (var i:int = 0; i < deviceArray.length; i++)
{
    trace(" " + deviceArray[i]);
}

var mic:Microphone = Microphone.getMicrophone();
mic.gain = 60;
mic.rate = 11;
mic.setUseEchoSuppression(true);
mic.setLoopBack(true);
mic.setSilenceLevel(5, 1000);

mic.addEventListener(ActivityEvent.ACTIVITY, this.onMicActivity);
mic.addEventListener(StatusEvent.STATUS, this.onMicStatus);
```

```
var micDetails:String = "Sound input device name: " + mic.name + '\n';
micDetails += "Gain: " + mic.gain + '\n';
micDetails += "Rate: " + mic.rate + " kHz" + '\n';
micDetails += "Muted: " + mic.muted + '\n';
micDetails += "Silence level: " + mic.silenceLevel + '\n';
micDetails += "Silence timeout: " + mic.silenceTimeout + '\n';
micDetails += "Echo suppression: " + mic.useEchoSuppression + '\n';
trace(micDetails);

function onMicActivity(event:ActivityEvent):void
{
    trace("activating=" + event.activating + ", activityLevel=" +
        mic.activityLevel);
}

function onMicStatus(event:StatusEvent):void
{
    trace("status: level=" + event.level + ", code=" + event.code);
}
```

Ao executar o exemplo acima, fale ou faça ruídos no microfone do sistema e observe as instruções de rastreamento resultantes serem exibidas em um console ou janela de depuração.

Envio e recebimento de áudio de um servidor de mídia

Recursos adicionais de áudio também estão disponíveis ao usar o ActionScript com um servidor de mídia de fluxo contínuo, como o Flash Media Server.

Em particular, seu aplicativo pode conectar um objeto Microphone a um objeto NetStream e transmitir dados diretamente do microfone do usuário para o servidor. Os dados de áudio também podem ser transmitidos em fluxo do servidor para um aplicativo criado com o Flash ou o Flex e reproduzidos como parte de um MovieClip ou usando um objeto Video.

O Flash Player 10 introduz o suporte para o codec Speex. Para definir o codec usado para áudio compactado enviado ao servidor de mídia, defina a propriedade `codec` do objeto Microphone. Esta propriedade pode ter dois valores, que são enumerados na classe SoundCodec. Definir a propriedade do codec como `SoundCodec.SPEEX` seleciona o codec Speex para compactação de áudio. Definir a propriedade como `SoundCodec.NELLYMOSER` (o padrão) seleciona o codec Nellymoser para compactação de áudio.

Para obter mais informações, consulte a documentação do Flash Media Server online em http://livedocs.adobe.com.

Exemplo: Podcast Player

Um podcast é um arquivo de som distribuído pela Internet sob demanda ou por assinatura. Podcasts normalmente são publicados como parte de uma série, o que também é chamado de canal de podcast. Como episódios de podcast podem durar de um minuto a muitas horas, eles normalmente são transmitidos enquanto estão sendo carregados. Episódios de podcast, que também são chamados de itens, normalmente são entregues no formato de arquivo mp3. Os podcasts de vídeo também são populares, mas esse aplicativo de amostra reproduz apenas podcasts de áudio que usam arquivos mp3.

Este exemplo não é de um aplicativo agregador de podcast completo. Por exemplo, ele não gerencia assinaturas para podcasts específicos ou lembra quais podcasts foram ouvidos pelo usuário na próxima vez que o aplicativo é executado. Ele pode funcionar como um ponto de partida para um agregador de podcast mais completo.

Este exemplo de Podcast Player ilustra as seguintes técnicas de programação do ActionScript:

- Leitura de um RSS feed externo e análise de seu conteúdo XML
- Criação de uma classe SoundFacade para simplificar o carregamento e a reprodução de arquivos de som.
- Exibição do progresso da reprodução do som.
- Pausa e reinício da reprodução do som.

Para obter os arquivos de aplicativo deste exemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_br. Os arquivos do aplicativo Podcast Player podem ser encontrados na pasta Amostras/PodcastPlayer. O aplicativo consiste nos seguintes arquivos:

Arquivo	Descrição
PodcastPlayer.mxml ou PodcastPlayer.fla	A interface do usuário do aplicativo para Flex (MXML) ou Flash (FLA).
SoundPlayer.mxml	Um componente MXML que exibe botões de reprodução e barras de progresso e controla a reprodução de som, apenas para Flex.
RSSBase.as	A classe base que fornece propriedades e métodos comuns para a classe RSSChannel e a classe RSSItem.
RSSChannel.as	Uma classe ActionScript que mantém dados sobre um canal RSS.
RSSItem.as	Uma classe ActionScript que mantém dados sobre um item RSS.
SoundFacade.as	A classe principal do ActionScript para o aplicativo. Ela encapsula os métodos e eventos da classe Sound e da classe SoundChannel e adiciona suporte para pausar e reiniciar a reprodução.
URLService.as	Uma classe ActionScript que recupera dados de uma URL remota.
playerconfig.xml	Um arquivo XML que contém uma lista de RSS feeds que representam canais de podcast.

Leitura de dados RSS para um canal de podcast

O aplicativo Podcast Player começa lendo informações sobre vários canais de podcast e seus episódios:

1. Primeiro, o aplicativo lê um arquivo de configuração XML que contém uma lista de canais de podcast e exibe a lista de canais para o usuário.

2. Quando o usuário seleciona um dos canais de podcast, ele lê o RSS feed do canal e exibe uma lista de episódios de canal.

Este exemplo usa a classe do utilitário URLLoader para recuperar dados com base em texto de um local remoto ou de um arquivo local. O Podcast Player primeiro cria um objeto URLLoader para obter uma lista de RSS feeds em formato XML do arquivo playerconfig.xml. Em seguida, quando o usuário seleciona um feed específico da lista, um novo objeto URLLoader é criado para ler os dados RSS daquela URL do feed.

Simplificação do carregamento e da reprodução de som usando a classe SoundFacade

A arquitetura de som do ActionScript 3.0 é poderosa, mas complexa. Aplicativos que precisam apenas de recursos básicos de carregamento e reprodução de som podem usar uma classe que oculte um pouco da complexidade fornecendo um conjunto mais simples de chamadas e eventos de métodos. No mundo de padrões de design de software, essa classe é chamada de *facade*.

A classe SoundFacade apresenta uma única interface para executar as seguintes tarefas:

- Carregamento de arquivos de som usando um objeto Sound, um objeto SoundLoaderContext e uma classe SoundMixer.
- Reprodução de arquivos de som usando os objetos Sound e SoundChannel.
- Despacho de eventos de progresso da reprodução.
- Pausa e reinício da reprodução do som usando os objetos Sound e SoundChannel.

A classe SoundFacade tenta oferecer mais da funcionalidade das classes de som do ActionScript com menos complexidade.

O código a seguir mostra a declaração da classe, as propriedades da classe e o método construtor `SoundFacade()`:

```
public class SoundFacade extends EventDispatcher
{
    public var s:Sound;
    public var sc:SoundChannel;
    public var url:String;
    public var bufferTime:int = 1000;

    public var isLoaded:Boolean = false;
    public var isReadyToPlay:Boolean = false;
    public var isPlaying:Boolean = false;
    public var isStreaming:Boolean = true;
    public var autoLoad:Boolean = true;
    public var autoPlay:Boolean = true;

    public var pausePosition:int = 0;

    public static const PLAY_PROGRESS:String = "playProgress";
    public var progressInterval:int = 1000;
    public var playTimer:Timer;

    public function SoundFacade(soundUrl:String, autoLoad:Boolean = true,
                                autoPlay:Boolean = true, streaming:Boolean = true,
                                bufferTime:int = -1):void
    {
        this.url = soundUrl;
```

```
        // Sets Boolean values that determine the behavior of this object
        this.autoLoad = autoLoad;
        this.autoPlay = autoPlay;
        this.isStreaming = streaming;

        // Defaults to the global bufferTime value
        if (bufferTime < 0)
        {
            bufferTime = SoundMixer.bufferTime;
        }

        // Keeps buffer time reasonable, between 0 and 30 seconds
        this.bufferTime = Math.min(Math.max(0, bufferTime), 30000);

        if (autoLoad)
        {
            load();
        }
    }
```

A classe SoundFacade estende a classe EventDispatcher para poder despachar seus próprios eventos. O código da classe primeiro declara propriedades de um objeto Sound e de um objeto SoundChannel. A classe também armazena o valor da URL do arquivo de som e uma propriedade `bufferTime` a ser usada ao transmitir o som em fluxo. Além disso, ele aceita alguns valores de parâmetros booleanos que afetam o comportamento do carregamento e da reprodução:

- O parâmetro `autoLoad` indica ao objeto que o carregamento do som deve iniciar assim que esse objeto é criado.
- O parâmetro `autoPlay` indica que a reprodução do som deve iniciar assim que dados de som suficientes estiverem carregados. Se esse for um fluxo de som, a reprodução será iniciada assim que dados suficientes, conforme especificado pela propriedade `bufferTime`, estiverem carregados.
- O parâmetro `streaming` indica que a reprodução desse arquivo pode ser iniciada antes do carregamento ser concluído.

O parâmetro `bufferTime` é padronizado como um valor de -1. Se o método construtor detectar um valor negativo no parâmetro `bufferTime`, ele definirá a propriedade `bufferTime` como o valor de `SoundMixer.bufferTime`. Isso permite que o aplicativo seja padronizado para o valor global `SoundMixer.bufferTime` conforme desejado.

Se o parâmetro `autoLoad` for definido como `true`, o método construtor chamará imediatamente o método `load()` seguinte para iniciar o carregamento do arquivo de som:

```
public function load():void
{
    if (this.isPlaying)
    {
        this.stop();
        this.s.close();
    }
    this.isLoaded = false;

    this.s = new Sound();

    this.s.addEventListener(ProgressEvent.PROGRESS, onLoadProgress);
    this.s.addEventListener(Event.OPEN, onLoadOpen);
    this.s.addEventListener(Event.COMPLETE, onLoadComplete);
    this.s.addEventListener(Event.ID3, onID3);
    this.s.addEventListener(IOErrorEvent.IO_ERROR, onIOError);
    this.s.addEventListener(SecurityErrorEvent.SECURITY_ERROR, onIOError);

    var req:URLRequest = new URLRequest(this.url);

    var context:SoundLoaderContext = new SoundLoaderContext(this.bufferTime, true);
    this.s.load(req, context);
}
```

O método `load()` cria um novo objeto Sound e adiciona ouvintes para todos os eventos de som importantes. Em seguida, ele indica ao objeto Sound para carregar o arquivo de som usando um objeto SoundLoaderContext para passar o valor de `bufferTime`.

Como a propriedade `url` pode ser alterada, uma ocorrência de SoundFacade pode ser usada para reproduzir arquivos de som diferentes em sucessão: simplesmente altere a propriedade `url` e chame o método `load()` e o novo arquivo de som será carregado.

Os três métodos ouvintes de eventos a seguir mostram como o objeto SoundFacade rastreia o progresso do carregamento e decide quando iniciar a reprodução do som:

```
public function onLoadOpen(event:Event):void
{
    if (this.isStreaming)
    {
        this.isReadyToPlay = true;
        if (autoPlay)
        {
            this.play();
        }
    }
    this.dispatchEvent(event.clone());
}

public function onLoadProgress(event:ProgressEvent):void
{
    this.dispatchEvent(event.clone());
}

public function onLoadComplete(event:Event):void
{
    this.isReadyToPlay = true;
    this.isLoaded = true;
    this.dispatchEvent(evt.clone());

    if (autoPlay && !isPlaying)
    {
        play();
    }
}
```

O método `onLoadOpen()` é executado quando o carregamento do som é iniciado. Se o som puder ser reproduzido em modo de streaming, o método `onLoadComplete()` definirá o sinalizador `isReadyToPlay` como `true` imediatamente. O sinalizador `isReadyToPlay` determina se o aplicativo pode iniciar a reprodução do som, talvez em resposta a uma ação do usuário, como um clique em um botão de reprodução. A classe SoundChannel gerencia o buffer de dados de som, portanto não há necessidade de verificar explicitamente se dados suficientes foram carregados antes de chamar o método `play()`.

O método `onLoadProgress()` é executado periodicamente durante o processo de carregamento. Ele simplesmente despacha um clone de seu objeto ProgressEvent para uso pelo código que usa esse objeto SoundFacade.

Quando os dados do som estiverem totalmente carregados, o método `onLoadComplete()` é executado, chamando o método `play()` para sons que não sejam de fluxo, se necessário. O próprio método `play()` é mostrado a seguir.

```
public function play(pos:int = 0):void
{
    if (!this.isPlaying)
    {
        if (this.isReadyToPlay)
        {
            this.sc = this.s.play(pos);
            this.sc.addEventListener(Event.SOUND_COMPLETE, onPlayComplete);
            this.isPlaying = true;

            this.playTimer = new Timer(this.progressInterval);
            this.playTimer.addEventListener(TimerEvent.TIMER, onPlayTimer);
            this.playTimer.start();
        }
    }
}
```

O método `play()` chamará o método `Sound.play()` se o som estiver pronto para execução. O objeto SoundChannel resultante é armazenado na propriedade `sc`. Em seguida, o método `play()` cria um objeto Timer que será utilizado para despachar eventos de progresso da reprodução em intervalos regulares.

Exibição do progresso da reprodução

A criação de um objeto Timer para acionar o monitoramento da reprodução é uma operação complexa que você deve codificar apenas uma vez. O encapsulamento dessa lógica de Timer em uma classe reutilizável, como a classe SoundFacade, permite que os aplicativos ouçam alguns tipos de eventos de progresso quando um som está sendo carregado e quando ele está sendo reproduzido.

O objeto Timer criado pelo método `SoundFacade.play()` despacha uma ocorrência de TimerEvent a cada segundo. O método `onPlayTimer()` a seguir é executado sempre que um novo TimerEvent é recebido:

```
public function onPlayTimer(event:TimerEvent):void
{
    var estimatedLength:int =
        Math.ceil(this.s.length / (this.s.bytesLoaded / this.s.bytesTotal));
    var progEvent:ProgressEvent =
        new ProgressEvent(PLAY_PROGRESS, false, false, this.sc.position, estimatedLength);
    this.dispatchEvent(progEvent);
}
```

O método `onPlayTimer()` implementa a técnica de estimativa de tamanho descrita na seção "Monitoramento da reprodução" na página 579. Em seguida, ele cria uma nova ocorrência de ProgressEvent com um tipo de evento de `SoundFacade.PLAY_PROGRESS`, com a propriedade `bytesLoaded` definida como a posição atual do objeto SoundChannel e a propriedade `bytesTotal` definida como o comprimento estimado dos dados do som.

Pausa e reinício da reprodução

O método `SoundFacade.play()` mostrado anteriormente aceita um parâmetro `pos` correspondente a uma posição inicial nos dados de som. Se o valor `pos` for zero, a reprodução do som será executada do início.

O método `SoundFacade.stop()` também aceita um parâmetro `pos` conforme mostrado aqui:

```
public function stop(pos:int = 0):void
{
    if (this.isPlaying)
    {
        this.pausePosition = pos;
        this.sc.stop();
        this.playTimer.stop();
        this.isPlaying = false;
    }
}
```

Sempre que o método `SoundFacade.stop()` é chamado, ele define a propriedade `pausePosition` de forma que o aplicativo saiba onde posicionar o indicador de reprodução se o usuário desejar retomar a reprodução do mesmo som.

Os métodos `SoundFacade.pause()` e `SoundFacade.resume()` mostrados a seguir chamam os métodos `SoundFacade.stop()` e `SoundFacade.play()` respectivamente, passando um valor de parâmetro `pos` a cada vez.

```
public function pause():void
{
    stop(this.sc.position);
}

public function resume():void
{
    play(this.pausePosition);
}
```

O método `pause()` passa o valor atual de `SoundChannel.position` para o método `play()` que armazena o valor na propriedade `pausePosition`. O método `resume()` inicia a reprodução do mesmo som novamente usando o valor de `pausePosition` como o ponto de início.

Extensão do exemplo do Podcast Player

Este exemplo apresenta um Podcast Player reduzido ao essencial que demonstra o uso da classe SoundFacade reutilizável. É possível adicionar outros recursos para aprimorar a utilidade deste aplicativo, incluindo o seguinte:

- Armazenar a lista de feeds e informações de uso sobre cada episódio em uma ocorrência de SharedObject que pode ser usada na próxima vez que o usuário executar o aplicativo.
- Permitir que o usuário adicione seus próprios RSS feeds à lista de canais de podcast.
- Memorizar a posição do indicador de reprodução quando o usuário interromper ou sair de um episódio, para que ele possa ser reiniciado daquele ponto na próxima vez que o usuário executar o aplicativo.
- Baixar arquivos mp3 de episódios para ouvir offline, quando o usuário não estiver conectado à Internet.
- Adicionar recursos de assinatura que verificam periodicamente novos episódios em um canal de podcast e atualizar a lista de episódios automaticamente.
- Adicionar funcionalidade de pesquisa e de procura de podcast usando uma API de um serviço de host de podcast, como o Odeo.com.

Capítulo 26: Captura da entrada do usuário

Este capítulo descreve como criar interatividade usando o ActionScript 3.0 para responder à atividade do usuário. São discutidos eventos de teclado e de mouse e, em seguida, há tópicos mais avançados, incluindo a personalização do menu de contexto e o gerenciamento do foco. Este capítulo encerra com o WordSearch, exemplo de um aplicativo que responde à entrada do mouse.

Esse capítulo supõe que você já está familiarizado com o modelo de eventos do ActionScript 3.0. Para obter mais informações, consulte “Manipulação de eventos” na página 251.

Noções básicas sobre a entrada do usuário

Introdução à captura da entrada do usuário

A interação do usuário, por meio do teclado, do mouse, da câmera ou de uma combinação desses dispositivos, é a base da interatividade. No ActionScript 3.0, identificar e responder à interação do usuário envolve principalmente ouvir eventos.

A classe InteractiveObject, uma subclasse de DisplayObject, fornece a estrutura comum dos eventos e a funcionalidade necessária para manipular a interação do usuário. Você não cria diretamente uma ocorrência da classe InteractiveObject. Em vez disso, objetos de exibição como SimpleButton, Sprite, TextField e vários componentes da ferramenta de autoria do Flash e do Flex herdam o modelo de interação do usuário dessa classe e, desse modo, compartilham uma estrutura comum. Isso significa que as técnicas aprendidas e o código que você grava para manipular a interação do usuário em um objeto derivado de InteractiveObject são aplicáveis a todos os outros usuários.

As seguintes tarefas típicas de interação do usuário são descritas neste capítulo:

- Captura da entrada do teclado no âmbito do aplicativo
- Captura da entrada do teclado para um objeto de exibição específico
- Captura de ações do mouse no âmbito do aplicativo
- Captura da entrada do mouse para um objeto de exibição específico
- Criação da interatividade de arrastar e soltar
- Criação de um cursor personalizado (ponteiro do mouse)
- Adição de novos comportamentos ao menu de contexto
- Gerenciamento do foco

Conceitos e termos importantes

É importante que você se familiarize com os seguintes termos-chave de interação do usuário antes de continuar:

- Código de caractere: código numérico que representa um caractere no conjunto de caracteres atual (associado a uma tecla que está sendo pressionada no teclado). Por exemplo, “D” e “d” têm códigos de caractere diferentes embora tenham sido criados pela mesma tecla do teclado do alfabeto inglês.

- Menu de contexto: o menu que aparece quando um usuário clica com o botão direito do mouse ou usa uma combinação específica do teclado-mouse. Os comandos do menu de contexto em geral são aplicados diretamente no que foi clicado. Por exemplo, um menu de contexto de uma imagem pode conter um comando para mostrar a imagem em uma janela separada e um comando para fazer download dessa imagem.
- Foco: indicação de que um elemento selecionado está ativo e é o destino da interação do teclado ou do mouse.
- Código de tecla: código numérico que corresponde a uma tecla física no teclado.

Teste dos exemplos do capítulo

À medida que lê este capítulo, você talvez queira testar algumas listagens de código de amostra por conta própria. Como este capítulo trata da entrada do usuário no ActionScript, todas as listagens de código de exemplo envolvem basicamente a manipulação de algum tipo de objeto de exibição — em geral um campo de texto ou qualquer subclasse de InteractiveObject. Nos exemplos, o objeto de exibição pode ser um objeto criado e colocado no Palco no Adobe® Flash® CS4 Professional ou pode ser um objeto criado com o ActionScript. Testar um exemplo envolve visualizar o resultado no Flash Player ou no Adobe® AIR™ e interagir com o exemplo para ver os efeitos do código.

Para testar as listagens de código deste capítulo:

1. Criar um documento vazio usando a ferramenta de autoria do Flash
2. Selecione um quadro-chave na Linha de tempo.
3. Abra o painel Ações e copie a listagem de código no painel Script.
4. Crie uma ocorrência no palco:
   - Se o código fizer referência a um campo de texto, use a ferramenta Texto para criar um campo de texto dinâmico no palco.
   - Caso contrário, crie uma ocorrência de símbolo de botão ou clipe de filme no palco.
5. Selecione o campo de texto, o botão ou o clipe de filme e dê um nome de ocorrência no Inspetor de propriedades. O nome deve coincidir com o nome do objeto de exibição no código de exemplo; se o código manipular, por exemplo, um objeto chamado `myDisplayObject`, seu objeto de palco também deverá ser chamado de `myDisplayObject`.
6. Execute o programa usando o comando Controlar > Testar filme.

   Na tela, o objeto de exibição é manipulado conforme especificado no código.

Captura da entrada do teclado

Os objetos de exibição que herdam o modelo de interação da classe InteractiveObject podem responder aos eventos de teclado usando ouvintes de eventos. Você pode colocar, por exemplo, um ouvinte de eventos no palco para ouvir e responder à entrada do teclado. No exemplo a seguir, um ouvinte de eventos captura um pressionamento de tecla e as propriedades de código e nome da tecla são exibidos:

```
function reportKeyDown(event:KeyboardEvent):void
{
    trace("Key Pressed: " + String.fromCharCode(event.charCode) + " (character code: " +
event.charCode + ")");
}
stage.addEventListener(KeyboardEvent.KEY_DOWN, reportKeyDown);
```

Algumas teclas, como a tecla Ctrl, geram eventos embora não tenha nenhuma representação de glifo.

No exemplo de código anterior, o ouvinte de eventos de teclado capturou a entrada do teclado para todo o palco. Você também pode gravar um ouvinte de eventos para um objeto de exibição específico no palco; esse ouvinte de eventos é acionado quando o objeto está em foco.

No exemplo a seguir, os pressionamentos de tecla são refletidos no painel Saída somente quando o usuário digita dentro da ocorrência TextField. Manter a tecla Shift temporariamente pressionada faz com que a cor da borda de TextField mude para vermelho.

Esse código supõe a existência de uma ocorrência TextField chamada `tf` no palco.

```
tf.border = true;
tf.type = "input";
tf.addEventListener(KeyboardEvent.KEY_DOWN,reportKeyDown);
tf.addEventListener(KeyboardEvent.KEY_UP,reportKeyUp);

function reportKeyDown(event:KeyboardEvent):void
{
    trace("Key Pressed: " + String.fromCharCode(event.charCode) + " (key code: " +
event.keyCode + " character code: " + event.charCode + ")");
    if (event.keyCode == Keyboard.SHIFT) tf.borderColor = 0xFF0000;
}

function reportKeyUp(event:KeyboardEvent):void
{
    trace("Key Released: " + String.fromCharCode(event.charCode) + " (key code: " +
event.keyCode + " character code: " + event.charCode + ")");
    if (event.keyCode == Keyboard.SHIFT)
    {
        tf.borderColor = 0x000000;
    }
}
```

A classe TextField também registra um evento `textInput` que pode ser ouvido quando um usuário insere algum texto. Para obter mais informações, consulte "Captura da entrada de texto" na página 441.

Identificação de códigos de tecla e de códigos de caractere

Você pode acessar as propriedades `keyCode` e `charCode` de um evento de teclado para determinar qual tecla foi pressionada e, em seguida, acionar outras ações. A propriedade `keyCode` é um valor numérico que corresponde ao valor de uma tecla no teclado. A propriedade `charCode` é o valor numérico dessa tecla no conjunto de caracteres atual. O conjunto de caracteres padrão é UTF-8, que oferece suporte para ASCII.

A principal diferença entre o código de tecla e os valores de caractere está no fato de o código de tecla representar uma tecla específica do teclado (o valor 1 de um teclado é diferente do valor 1 na linha superior, mas a tecla que gera "1" e a que gera "!" são iguais), enquanto o valor de caractere representa um caractere específico (os caracteres R e r são diferentes).

Nota: *Em caso de mapeamentos entre teclas e seus respectivos valores de código de caracteres em ASCII, consulte a classe flash.ui.Keyboard na referência de linguagem do ActionScript.*

O mapeamento entre as teclas e os códigos de tecla dependem do dispositivo e do sistema operacional. Devido a isso, não use mapeamentos de tecla para acionar comandos. Em vez disso, use os valores de constantes predefinidos fornecidos pela classe Keyboard para fazer referência às propriedades `keyCode` adequadas. Por exemplo, em vez de usar o mapeamento para a tecla Shift, use a constante `Keyboard.SHIFT` (como mostra o código de exemplo anterior).

Compreensão da precedência de KeyboardEvent

Assim como outros eventos, a seqüência de eventos de teclado é determinada pela hierarquia de objetos de exibição, não pela ordem em que os métodos `addEventListener()` são atribuídos no código.

Suponha, por exemplo, que você coloque um campo de texto chamado `tf` em um clipe de filme chamado `container` e adicione um ouvinte de eventos para um evento de teclado nas duas ocorrências, como mostra o exemplo a seguir:

```
container.addEventListener(KeyboardEvent.KEY_DOWN,reportKeyDown);
container.tf.border = true;
container.tf.type = "input";
container.tf.addEventListener(KeyboardEvent.KEY_DOWN,reportKeyDown);

function reportKeyDown(event:KeyboardEvent):void
{
    trace(event.currentTarget.name + " hears key press: " + String.fromCharCode(event.charCode)
+ " (key code: " + event.keyCode + " character code: " + event.charCode + ")");
}
```

Como há um ouvinte no campo de texto e no recipiente pai, a função `reportKeyDown()` é chamada duas vezes para cada pressionamento de tecla em TextField. Para cada tecla pressionada, o campo de texto envia um evento antes que o clipe de filme `container` faça isso.

O sistema operacional e o navegador da Web processam eventos de teclado antes do Adobe Flash Player ou AIR. No Microsoft Internet Explorer, por exemplo, pressionar Ctrl+W fecha a janela do navegador antes que qualquer arquivo SWF contido envie um evento de teclado.

Captura da entrada do mouse

Os cliques criam eventos de mouse que podem ser usados para acionar funções interativas. Você pode adicionar um ouvinte de eventos ao palco para ouvir eventos de mouse que ocorrem em qualquer lugar no arquivo SWF. Também é possível adicionar ouvintes de eventos a objetos no palco que herdam modelos de InteractiveObject (por exemplo, Sprite ou MovieClip); esses ouvintes são acionados quando o objeto é clicado.

Assim como os eventos de teclado, os eventos de mouse aparecem em balões. No exemplo a seguir, como `square` é filho de Stage, o evento é enviado a partir do objeto `square` da entidade gráfica, bem como do objeto Stage quando se clica em square:

```
var square:Sprite = new Sprite();
square.graphics.beginFill(0xFF0000);
square.graphics.drawRect(0,0,100,100);
square.graphics.endFill();
square.addEventListener(MouseEvent.CLICK, reportClick);
square.x =
square.y = 50;
addChild(square);

stage.addEventListener(MouseEvent.CLICK, reportClick);

function reportClick(event:MouseEvent):void
{
    trace(event.currentTarget.toString() + " dispatches MouseEvent. Local coords [" +
event.localX + "," + event.localY + "] Stage coords [" + event.stageX + "," + event.stageY + "]");
}
```

No exemplo anterior, observe que o evento de mouse contém informações de posição sobre o clique. As propriedades `localX` e `localY` contêm o local do clique no filho mais inferior da cadeia de exibição. Por exemplo, clicar no canto superior esquerdo de `square` registra as coordenadas de local [0,0] porque este é o ponto de registro de `square`. Como alternativo, as propriedades `stageX` e `stageY` referem-se às coordenadas globais do clique no palco. O mesmo clique registra [50,50] para essas coordenadas, pois `square` foi movido até elas. Esses dois pares de coordenadas podem ser úteis dependendo do modo como deseja responder à interação do usuário.

O objeto MouseEvent também contém as propriedades booleanas `altKey`, `ctrlKey` e `shiftKey`. Você pode usar essas propriedades para verificar se a tecla Alt, Ctrl ou Shift também está sendo pressionada ao mesmo tempo em que o clique do mouse.

Criação da funcionalidade arrastar e soltar

A funcionalidade arrastar e soltar permite que os usuários selecionem um objeto enquanto pressionam o botão esquerdo do mouse, movam o objeto até um novo local na tela e soltem-no nesse novo local soltando o botão esquerdo do mouse. O código a seguir mostra um exemplo disso:

```
import flash.display.Sprite;
import flash.events.MouseEvent;

var circle:Sprite = new Sprite();
circle.graphics.beginFill(0xFFCC00);
circle.graphics.drawCircle(0, 0, 40);

var target1:Sprite = new Sprite();
target1.graphics.beginFill(0xCCFF00);
target1.graphics.drawRect(0, 0, 100, 100);
target1.name = "target1";

var target2:Sprite = new Sprite();
target2.graphics.beginFill(0xCCFF00);
target2.graphics.drawRect(0, 200, 100, 100);
target2.name = "target2";

addChild(target1);
addChild(target2);
addChild(circle);

circle.addEventListener(MouseEvent.MOUSE_DOWN, mouseDown)

function mouseDown(event:MouseEvent):void
{
    circle.startDrag();
}
circle.addEventListener(MouseEvent.MOUSE_UP, mouseReleased);

function mouseReleased(event:MouseEvent):void
{
    circle.stopDrag();
    trace(circle.dropTarget.name);
}
```

Para obter mais detalhes, consulte a seção sobre a criação da interação de arrastar e soltar em "Alteração da posição" na página 293.

Personalização do cursor do mouse

O cursor do mouse (ponteiro) pode ser ocultado ou alternado para qualquer objeto de exibição no palco. Para ocultar o cursor do mouse, chame o método `Mouse.hide()`. Personalize o cursor chamando `Mouse.hide()`, ouvindo o evento `MouseEvent.MOUSE_MOVE` no palco e definindo as coordenadas de um objeto de exibição (seu cursor personalizado) com as propriedades `stageX` e `stageY` do evento. O exemplo a seguir ilustra uma execução básica dessa tarefa:

```
var cursor:Sprite = new Sprite();
cursor.graphics.beginFill(0x000000);
cursor.graphics.drawCircle(0,0,20);
cursor.graphics.endFill();
addChild(cursor);

stage.addEventListener(MouseEvent.MOUSE_MOVE,redrawCursor);
Mouse.hide();

function redrawCursor(event:MouseEvent):void
{
    cursor.x = event.stageX;
    cursor.y = event.stageY;
}
```

Personalização do menu de contexto

Cada objeto herdado a partir da classe InteractiveObject pode ter um menu de contexto exclusivo, que é exibido quando o usuário clica com o botão direito do mouse no arquivo SWF. Vários comandos são incluídos por padrão, como Avançar, Voltar, Imprimir, Qualidade e Zoom.

Você pode remover todos os comandos padrão do menu, exceto Configurações e Sobre. Configurar a propriedade `showDefaultContextMenu` do palco como `false` remove esses comandos do menu de contexto.

Para criar um menu de contexto personalizado para um objeto de exibição específico, crie uma nova ocorrência da classe ContextMenu, chame o método `hideBuiltInItems()` e atribua essa ocorrência à propriedade `contextMenu` dessa instância DisplayObject. O exemplo a seguir fornece um quadrado desenhado dinamicamente com um comando de menu de contexto para alterá-lo para uma cor aleatória:

```
var square:Sprite = new Sprite();
square.graphics.beginFill(0x000000);
square.graphics.drawRect(0,0,100,100);
square.graphics.endFill();
square.x =
square.y = 10;
addChild(square);

var menuItem:ContextMenuItem = new ContextMenuItem("Change Color");
menuItem.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT,changeColor);
var customContextMenu:ContextMenu = new ContextMenu();
customContextMenu.hideBuiltInItems();
customContextMenu.customItems.push(menuItem);
square.contextMenu = customContextMenu;

function changeColor(event:ContextMenuEvent):void
{
    square.transform.colorTransform = getRandomColor();
}
function getRandomColor():ColorTransform
{
    return new ColorTransform(Math.random(), Math.random(), Math.random(),1,(Math.random() *
512) - 255, (Math.random() * 512) -255, (Math.random() * 512) - 255, 0);
}
```

Gerenciamento do foco

Um objeto interativo pode receber o foco, programaticamente ou por meio de uma ação do usuário. Nos dois casos, definir o foco altera a propriedade `focus` do objeto para `true`. Além disso, se a propriedade `tabEnabled` estiver definida como `true`, o usuário poderá passar o foco de um objeto para outro pressionando a tecla Tab. Observe que o valor de `tabEnabled` é `false` por padrão, exceto nos seguintes casos:

- Para um objeto SimpleButton, o valor é `true`.
- Para um campo de texto de entrada, o valor é `true`.
- Para um objeto Sprite ou MovieClip com `buttonMode` definido como `true`, o valor é `true`.

Em cada uma dessas situações, você pode adicionar um ouvinte para `FocusEvent.FOCUS_IN` ou `FocusEvent.FOCUS_OUT` a fim de fornecer outros comportamentos quando o foco for alterado. Isso é útil especialmente para campos de texto e formulários, mas também pode ser usado em entidades gráficas, clipes de filme ou qualquer objeto herdado da classe InteractiveObject. O exemplo a seguir mostra como ativar o ciclo do foco com a tecla Tab e como responder ao evento de foco subseqüente. Nesse caso, cada quadrado muda de cor assim que recebe o foco.

Nota: *A ferramenta de criação do Flash usa atalhos do teclado para gerenciar o foco; desse modo, para simular o gerenciamento do foco corretamente, os arquivos SWF devem ser testados em um navegador ou AIR, não no Flash.*

```
var rows:uint = 10;
var cols:uint = 10;
var rowSpacing:uint = 25;
var colSpacing:uint = 25;
var i:uint;
var j:uint;
for (i = 0; i < rows; i++)
{
    for (j = 0; j < cols; j++)
    {
        createSquare(j * colSpacing, i * rowSpacing, (i * cols) + j);
    }
}

function createSquare(startX:Number, startY:Number, tabNumber:uint):void
{
    var square:Sprite = new Sprite();
    square.graphics.beginFill(0x000000);
    square.graphics.drawRect(0, 0, colSpacing, rowSpacing);
    square.graphics.endFill();
    square.x = startX;
    square.y = startY;
    square.tabEnabled = true;
    square.tabIndex = tabNumber;
    square.addEventListener(FocusEvent.FOCUS_IN, changeColor);
    addChild(square);
}
function changeColor(event:FocusEvent):void
{
    event.target.transform.colorTransform = getRandomColor();
}
function getRandomColor():ColorTransform
{
    // Generate random values for the red, green, and blue color channels.
    var red:Number = (Math.random() * 512) - 255;
    var green:Number = (Math.random() * 512) - 255;
    var blue:Number = (Math.random() * 512) - 255;

    // Create and return a ColorTransform object with the random colors.
    return new ColorTransform(1, 1, 1, 1, red, green, blue, 0);
}
```

Exemplo: WordSearch

Este exemplo demonstra a interação do usuário por meio da manipulação de eventos de mouse. Os usuários criam o máximo de palavras possível a partir de uma grade aleatória de letras, movendo-se na horizontal ou vertical na grade, mas nunca usando a mesma letra duas vezes. Este exemplo demonstra os seguintes recursos do ActionScript 3.0:

- Criação dinâmica de uma grade de componentes
- Resposta a eventos de mouse
- Manutenção da pontuação com base na interação do usuário

Para obter os arquivos de aplicativo desse exemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_br. Os arquivos do aplicativo WordSearch estão localizados na pasta Amostras/WordSearch. O aplicativo consiste nos seguintes arquivos:

Arquivo	Descrição
WordSearch.as	A classe que fornece a funcionalidade principal do aplicativo.
WordSearch.fla	O arquivo de aplicativo principal para Flash.
dictionary.txt	Um arquivo usado para determinar se as palavras criadas podem ser pontuadas e foram escritas corretamente.

Carregamento de um dicionário

Para criar um jogo que envolve a busca de palavras, um dicionário é necessário. O exemplo inclui um arquivo de texto chamado dictionary.txt que contém uma lista de palavras separadas por retornos de carro. Depois que uma matriz chamada `words` é criada, a função `loadDictionary()` solicita esse arquivo que, ao ser carregado com êxito, se transforma em uma longa string. É possível analisar essa string em uma matriz de palavras usando o método `split()`, quebrando-a em cada ocorrência de um retorno de carro (código de caractere 10) ou de uma nova linha (código de caractere 13). Essa análise ocorre na função `dictionaryLoaded()`:

```
words = dictionaryText.split(String.fromCharCode(13, 10));
```

Criação da interface de usuário

Depois de armazenar as palavras, você pode configurar a interface de usuário. Crie duas ocorrências de botão: uma para enviar uma palavra e outra para apagar uma palavra que esteja com erros ortográficos no momento. Em cada caso, responda à entrada do usuário ouvindo o evento `MouseEvent.CLICK` que o botão transmite e, em seguida, chamando uma função. Na função `setupUI()`, esse código cria os ouvintes nos dois botões:

```
submitWordButton.addEventListener(MouseEvent.CLICK,submitWord);
clearWordButton.addEventListener(MouseEvent.CLICK,clearWord);
```

Geração de uma placa de jogo

A placa de jogo é uma grade de letras aleatórias. Na função `generateBoard()`, uma grade bidimensional é criada, aninhando um loop em outro. O primeiro loop aumenta as linhas e o segundo aumenta o número total de colunas por linha. Cada célula criada por essas linhas e colunas contém um botão que representa uma letra na placa.

```
private function generateBoard(startX:Number, startY:Number, totalRows:Number,
totalCols:Number, buttonSize:Number):void
{
    buttons = new Array();
    var colCounter:uint;
    var rowCounter:uint;
    for (rowCounter = 0; rowCounter < totalRows; rowCounter++)
    {
        for (colCounter = 0; colCounter < totalCols; colCounter++)
        {
            var b:Button = new Button();
            b.x = startX + (colCounter*buttonSize);
            b.y = startY + (rowCounter*buttonSize);
            b.addEventListener(MouseEvent.CLICK, letterClicked);
            b.label = getRandomLetter().toUpperCase();
            b.setSize(buttonSize,buttonSize);
            b.name = "buttonRow"+rowCounter+"Col"+colCounter;
            addChild(b);

            buttons.push(b);
        }
    }
}
```

Embora um ouvinte seja adicionado para um evento `MouseEvent.CLICK` apenas em uma linha, por estar em um loop `for`, ele é atribuído a cada ocorrência de botão. Além disso, cada botão recebe um nome derivado de sua posição de linha e coluna, o que facilita a referência à linha e à coluna de cada botão posteriormente no código.

Criação de palavras a partir da entrada do usuário

Para criar palavras, é possível selecionar letras que estão próximas na vertical ou na horizontal, mas nunca se deve usar a mesma letra duas vezes. Cada clique gera um evento de mouse e, nesse momento, a palavra fornecida pelo usuário deve ser verificada para assegurar que esteja sendo corretamente formada a partir das letras clicadas antes. Em caso negativo, a palavra anterior é removida e uma nova é iniciada. Essa verificação ocorre no método `isLegalContinuation()`.

```
private function isLegalContinuation(prevButton:Button, currButton:Button):Boolean
{
    var currButtonRow:Number = Number(currButton.name.charAt(currButton.name. indexOf("Row") +
3));
    var currButtonCol:Number = Number(currButton.name.charAt(currButton.name.indexOf("Col") +
3));
    var prevButtonRow:Number = Number(prevButton.name.charAt(prevButton.name.indexOf("Row") +
3));
    var prevButtonCol:Number = Number(prevButton.name.charAt(prevButton.name.indexOf("Col") +
3));

    return ((prevButtonCol == currButtonCol && Math.abs(prevButtonRow - currButtonRow) <= 1) ||
            (prevButtonRow == currButtonRow && Math.abs(prevButtonCol - currButtonCol) <= 1));
}
```

Os métodos `charAt()` e `indexOf()` da classe String recuperam as linhas e colunas apropriadas do botão clicado atualmente e do botão clicado anteriormente. O método `isLegalContinuation()` retornará `true` se a linha ou coluna estiver inalterada e se a linha ou coluna que foi alterada estiver em um único incremento da anterior. Se desejar alterar as regras do jogo e permitir a formação de palavras na diagonal, remova as verificações de linhas ou colunas inalteradas; a linha final é similar a:

```
return (Math.abs(prevButtonRow - currButtonRow) <= 1) && Math.abs(prevButtonCol -
currButtonCol) <= 1));
```

Verificação do envio de palavras

Para concluir o código do jogo, os mecanismos de verificação do envio de palavras e de cálculo da pontuação são necessários. O método `searchForWord()` contém os dois:

```
private function searchForWord(str:String):Number
{
    if (words && str)
    {
        var i:uint = 0
        for (i = 0; i < words.length; i++)
        {
            var thisWord:String = words[i];
            if (str == words[i])
            {
                return i;
            }
        }
        return -1;
    }
    else
    {
        trace("WARNING: cannot find words, or string supplied is null");
    }
    return -1;
}
```

Essa função percorre todas as palavras do dicionário. Se a palavra do usuário coincidir com a do dicionário, sua posição no dicionário será retornada. O método `submitWord()` verifica a resposta e atualiza a pontuação se a posição for válida.

Personalização

No início da classe, existem várias constantes. Você pode modificar esse jogo modificando essas variáveis. É possível, por exemplo, alterar a quantidade de tempo disponível para jogar aumentando a variável `TOTAL_TIME`. Você também pode aumentar ligeiramente a variável `PERCENT_VOWELS` para aumentar a probabilidade de busca das palavras.

Capítulo 27: Rede e comunicação

Este capítulo explica como ativar o arquivo SWF para se comunicar com arquivos externos e outras ocorrências do Adobe Flash® Player 9 e do Adobe® AIR™. Ele também explica como carregar dados de fontes externas, enviar mensagens entre um servidor Java e o Flash Player e fazer uploads e downloads de arquivos usando as classes FileReference e FileReferenceList.

Noções básicas de rede e comunicação

Introdução à rede e comunicação

Freqüentemente, ao criar aplicativos do ActionScript mais complexos, você necessita fazer a comunicação com scripts de servidor ou carregar dados de arquivos XML ou de texto externos. O pacote flash.net contém classes para enviar e receber dados pela Internet (por exemplo, para carregar conteúdo de URLs remotas, comunicar-se com outras ocorrências do Flash Player ou do AIR, além de conectar-se a sites remotos.

No ActionScript 3.0, você pode carregar arquivos externos com as classes URLLoader e URLRequest. Assim, você pode usar uma classe específica para acessar os dados, dependendo do tipo de dados carregado. Por exemplo, se o conteúdo remoto é formatado como pares de nome e valor, use a classe URLVariables para analisar os resultados do servidor. Opcionalmente, se o arquivo carregado por meio das classes URLLoader e URLRequest for um documento XML remoto, você poderá analisá-lo usando o construtor de classes XML, o construtor de classe XMLDocument ou o método `XMLDocument.parseXML()`. Isso permite simplificar o código ActionScript, pois o código para carregar arquivos externos é o mesmo, independentemente de você usar URLVariables, XML ou alguma outra classe para analisar e trabalhar com os dados remotos.

O pacote flash.net também contém classes para outros tipos de comunicação remota. Entre elas estão a classe FileReference, para carregar e baixar arquivos de um servidor, as classes Socket e XMLSocket, que permitem a comunicação direta com computadores remotos via conexões de soquete e as classes NetConnection e NetStream, utilizadas para comunicação com um servidor de recursos específicos do Flash, como o Flash Media Server e servidores Flash Remoting, bem como para o carregamento de arquivos de vídeo.

Por fim, o pacote flash.net inclui classes para comunicação no computador local dos usuários. Entre elas está a classe LocalConnection, que permite a comunicação entre dois ou mais arquivos SWF em execução em um único computador, e a classe SharedObject, que permite o armazenamento de dados no computador do usuário e os recupera no momento em que forem retornados para o aplicativo.

Tarefas comuns de comunicação e rede

A lista a seguir descreve as tarefas mais comuns, que talvez você queira realizar, relacionadas à comunicação externa do ActionScript; estas tarefas são descritas neste capítulo:

- Carregar dados a partir de um arquivo externo ou um script de servidor
- Enviar dados para um script de servidor
- Estabelecer a comunicação com outros arquivos SWF locais
- Trabalhar com conexões de soquete binário
- Estabelecer a comunicação com soquetes XML

- Armazenar dados locais persistentes
- Upload de arquivos em um servidor
- Baixar arquivos de um servidor para a máquina do usuário

Conceitos e termos importantes

A lista de referência a seguir contém termos importantes usados neste capítulo:

- Dados externos: são dados armazenados de algum modo fora do arquivo SWF e carregados nesse mesmo arquivo quando necessário. Esses dados podem ser armazenados em um arquivo que é carregado diretamente, em um banco de dados ou de algum modo que permita recuperá-los ao chamar scripts ou programas em execução no servidor.
- Variáveis codificadas por URL: o formato codificado por URL permite representar diversas variáveis (pares de nomes e valores de variáveis) em uma única string de texto. Variáveis individuais são escritas no formato `nome=valor`. Cada variável (ou seja, cada par de nome e valor) é separada por caracteres E comercial, desta maneira: `variável1=valor1&variável2=valor2`. Desse modo, é possível enviar um número indefinido de variáveis como uma só mensagem.
- Tipo MIME: é um código padrão usado para identificar o tipo de um determinado arquivo em comunicação via Internet. Qualquer tipo de arquivo possui um código específico, usado para identificá-lo. Ao enviar um arquivo ou uma mensagem, um computador (como um servidor Web ou uma ocorrência do Flash Player ou do AIR de usuário) especificará o tipo de arquivo que está sendo enviado.
- HTTP: forma abreviada do termo em inglês Hypertext Transfer Protocol, um formato padrão para apresentação de páginas da Web e vários outros tipos de conteúdo enviados pela Internet.
- Método de solicitação: quando um programa como o Flash Player ou um navegador da Web envia uma mensagem (chamada de solicitação HTTP) para um servidor Web, qualquer dado que está sendo enviado pode ser incorporado à solicitação de duas maneiras, por meio dos *métodos de solicitação* GET e POST. No servidor, o programa que recebe a solicitação precisará localizar os dados no trecho adequado da solicitação, por isso o método de solicitação usado para enviar dados do ActionScript deve corresponder ao método de solicitação usado para ler esses dados no servidor.
- Conexão de soquete: é uma conexão persistente para comunicação entre dois computadores.
- Upload: é o envio de arquivos para outro computador.
- Download: é usado para recuperar um arquivo contido em outro computador.

Trabalho com endereços IPv6

O Flash Player 9.0.115.0 e suas versões posteriores oferecem suporte a IPv6 (Internet Protocol versão 6). O IPv6 é uma versão do Internet Protocol que oferece suporte a endereços de 128 bits (um aprimoramento do protocolo IPv4 anterior, que oferece suporte a endereços de 32 bits). Talvez seja necessário ativar o IPv6 nas suas interfaces de rede. Para obter mais informações, consulte a Ajuda do sistema operacional que está hospedando os dados.

Se o IPv6 for suportado no sistema de host, será possível especificar endereços literais IPv6 numéricos em URLs, contidos entre parênteses ([]), como apresentado seguir:

```
rtmp://[2001:db8:ccc3:ffff:0:444d:555e:666f]:1935/test
```

O Flash Player retorna valores IPv6 literais, de acordo com as regras a seguir:

- O Flash Player retorna o formato longo da string para endereços IPv6.
- O valor de IP não possui abreviações com dois pontos.

- Os dígitos hexadecimais são expressos apenas em letras minúsculas.
- Os endereços IPv6 são apresentados entre parênteses ([]).
- Cada quarteto de endereço é transformado em dígitos hexadecimais de 0 a 4, com os zeros à esquerda omitidos.
- Um quarteto de endereço composto por zeros é transformado em um único zero (e não em dois-pontos duplos), exceto se mencionado na lista de exceções a seguir.

Os valores IPv6 retornados pelo Flash Player têm as seguintes exceções:

- Um endereço IPv6 não especificado (somente zeros) é transformado em [::].
- O endereço IPv6 de retorno ou de host local é transformado em [::1].
- Os endereços IPv4 mapeados (convertido para IPv6) são transformados em [::ffff:a.b.c.d], em que a.b.c.d corresponde a um típico valor decimal IPv4 separado por pontos.
- Os endereços IPv4 compatíveis são transformados em [::a.b.c.d], em que a.b.c.d corresponde a um típico valor decimal IPv4 separado por pontos.

Teste dos exemplos do capítulo

Talvez você queira testar as listagens de código de exemplo durante a leitura deste capítulo. Muitas das listagens de códigos desse capítulo carregam dados externos ou realizam algum tipo de comunicação. Entre esses exemplos, em geral, estão incluídas as chamadas da função `trace()`, para que os resultados da execução do exemplo sejam exibidos no Painel de saída. Outros exemplos executam determinadas funções, como o upload de um arquivo para um servidor. O teste desses exemplos envolverá a interação com o arquivo SWF e a confirmação da ação que eles estão incumbidos de executar.

Os exemplos de código são classificados em duas categorias. Algumas das listagens de exemplo são escritas considerando que o código esteja em um script autônimo, que pode estar anexado a um quadro-chave de um documento Flash, por exemplo. Para testar os exemplos:

1. Crie um novo documento Flash.
2. Selecione o quadro-chave no Quadro 1 da Linha de tempo e abra o painel Ações.
3. Copie a listagem de código no painel Script.
4. No menu principal, selecione Controle > Testar filme para criar o arquivo SWF e testar o exemplo.

Outras listagens de códigos de exemplo são escritas como uma classe; espera-se que a classe de exemplo sirva como a classe do documento para o documento Flash. Para testar os exemplos:

1. Crie um documento Flash vazio e salve-o no seu computador.
2. Crie e salve um novo arquivo do ActionScript no mesmo diretório do documento Flash. O nome do arquivo deve corresponder ao nome da classe na listagem de código. Por exemplo, se a listagem de código define uma classe chamada “UploadTest”, salve o arquivo ActionScript como “UploadTest.as”.
3. Copie a listagem de código no arquivo do ActionScript e salve o arquivo.
4. No documento Flash, clique em uma parte em branco do Palco ou da área de trabalho para ativar o Inspetor de propriedades do documento.
5. No Inspetor de propriedades, no campo Classe do documento, digite o nome da classe ActionScript que você copiou do texto.
6. Execute o programa usando o comando Controlar > Testar filme e teste o exemplo.

Por fim, alguns dos exemplos deste capítulo envolvem a interação com um programa em execução em um servidor. Esses exemplos incluem um código que pode ser usado para criar o programa de servidor necessário para testar o exemplo. Para isso, será necessário configurar os aplicativos apropriados em um computador definido como servidor Web para testar os exemplos.

Trabalho com dados externos

O ActionScript 3.0 inclui mecanismos para carregamento de dados de fontes externas. Essas fontes podem ser conteúdo estático, como arquivos de texto ou conteúdo dinâmico, como um script da Web que recupera dados de um banco de dados. Os dados podem ser formatados de várias maneiras, e o ActionScript fornece funcionalidade para decodificação e acesso aos dados. Você também pode enviar dados ao servidor externo como parte do processo de recuperação de dados.

Uso das classes URLLoader e URLVariables

O ActionScript 3.0 usa as classes URLLoader e URLVariables para carregar dados externos. A classe URLLoader baixa dados de uma URL como texto, dados binários ou variáveis codificadas de URL. Ela é útil para download de arquivos de texto, XML ou outras informações a serem usadas em aplicativos ActionScript dinâmicos controlados por dados. A classe URLLoader beneficia-se do modelo de tratamento de eventos avançado do ActionScript 3.0 que permite monitorar esses eventos como `complete`, `httpStatus`, `ioError`, `open`, `progress` e `securityError`. O novo modelo de tratamento de eventos representa uma melhoria significativa no suporte do ActionScript 2.0 para os manipuladores de eventos `LoadVars.onData`, `LoadVars.onHTTPStatus` e `LoadVars.onLoad` porque permite tratar erros e eventos de maneira mais eficiente. Para obter mais informações sobre como tratar eventos, consulte "Manipulação de eventos" na página 251

Muito semelhante às classes XML e LoadVars em versões anteriores do ActionScript, os dados da URL da URLLoader não estão disponíveis até que o download seja concluído. Você pode monitorar o andamento do download (bytes carregados e total de bytes) monitorando o evento `flash.events.ProgressEvent.PROGRESS` a ser despachado, se bem que se o arquivo for carregado muito rapidamente, um evento `ProgressEvent.PROGRESS` talvez não seja despachado. Quando um arquivo é baixado com êxito, o evento `flash.events.Event.COMPLETE` é despachado. Os dados carregados são decodificados da codificação UTF-8 ou UTF-16 em uma string.

Nota: *Se nenhum valor estiver definido para `URLRequest.contentType`, os valores serão enviados como `aplicativo/x-www-form-urlencoded`.*

O método `URLLoader.load()` (e opcionalmente o construtor da classe URLLoader) utiliza um único parâmetro, `request`, que é uma ocorrência de URLRequest. Uma ocorrência de URLRequest contém todas as informações para uma única solicitação HTTP, como a URL de destino, método de solicitação (`GET` ou `POST`), informações de cabeçalho adicionais e do tipo MIME (por exemplo, quando você carrega o conteúdo XML).

Por exemplo, para carregar um pacote XML em um script do lado do servidor, use o seguinte código do ActionScript 3.0:

```
var secondsUTC:Number = new Date().time;
var dataXML:XML =
    <login>
        <time>{secondsUTC}</time>
        <username>Ernie</username>
        <password>guru</password>
    </login>;
var request:URLRequest = new URLRequest("http://www.yourdomain.com/login.cfm");
request.contentType = "text/xml";
request.data = dataXML.toXMLString();
request.method = URLRequestMethod.POST;
var loader:URLLoader = new URLLoader();
try
{
    loader.load(request);
}
catch (error:ArgumentError)
{
    trace("An ArgumentError has occurred.");
}
catch (error:SecurityError)
{
    trace("A SecurityError has occurred.");
}
```

O snippet anterior cria uma ocorrência XML denominada `dataXML` que contém um pacote XML a ser enviado ao servidor. Em seguida, você define a propriedade `contentType` de URLRequest como `"text/xml"` e define a propriedade `data` de URLRequest como o conteúdo do pacote XML, que são convertidas em uma string usando o método `XML.toXMLString()`. Finalmente, você cria uma nova ocorrência de URLLoader e envia a solicitação ao script remoto usando o método `URLLoader.load()`.

Há três maneiras possíveis para especificar parâmetros a serem transmitidos em uma solicitação de URL:

- De dentro do construtor URLVariables
- De dentro do método `URLVariables.decode()`
- Como propriedades específicas dentro do próprio objeto URLVariables

Quando você define variáveis dentro do construtor URLVariables ou de dentro do método `URLVariables.decode()`, é necessário verificar se codificou o caractere E comercial em URL porque ele tem um significado especial e funciona como um delimitador. Por exemplo, ao transmitir um E comercial, você precisa codificá-lo em URL alterando-o de `&` para `%26` porque o E comercial funciona como um delimitador de parâmetros.

Carregamento de dados de documentos externos

Ao criar aplicativos dinâmicos com o ActionScript 3.0, é bom carregar os dados de arquivos externos ou de scripts do lado do servidor. Isso permite criar aplicativos dinâmicos sem precisar editar ou recompilar os arquivos ActionScript. Por exemplo, se você criar um aplicativo "tip of the day", poderá gravar um script do lado do servidor que recupera uma dica aleatória de um banco de dados e salva-a em um arquivo de texto uma vez por dia. Em seguida, o aplicativo ActionScript pode carregar o conteúdo de um arquivo de texto estático em vez de consultar o banco de dados a cada vez.

O seguinte snippet cria um objeto URLRequest e URLLoader que carrega o conteúdo de um arquivo de texto externo, params.txt:

```
var request:URLRequest = new URLRequest("params.txt");
var loader:URLLoader = new URLLoader();
loader.load(request);
```

É possível simplificar o snippet anterior para o seguinte:

```
var loader:URLLoader = new URLLoader(new URLRequest("params.txt"));
```

Por padrão, se você não definir um método de solicitação, o Flash Player e o Adobe AIR carregarão o conteúdo usando o método `GET` HTTP. Para enviar os dados usando o método `POST`, é necessário definir a propriedade `request.method` como `POST` usando a constante estática `URLRequestMethod.POST`, conforme mostrado pelo código a seguir:

```
var request:URLRequest = new URLRequest("sendfeedback.cfm");
request.method = URLRequestMethod.POST;
```

O documento externo, params.txt, que é carregado no tempo de execução, contém os seguintes dados:

```
monthNames=January,February,March,April,May,June,July,August,September,October,November,Dece
mber&dayNames=Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,Saturday
```

O arquivo contém dois parâmetros, `monthNames` e `dayNames`. Cada parâmetro contém uma lista separada por vírgula que é analisada como strings. É possível dividir essa lista em uma matriz usando o método `String.split()`.

Evite usar palavras reservadas ou construções de linguagem como nomes de variáveis em arquivos de dados externos, porque isso dificulta a leitura e a depuração do código.

Depois que os dados são carregados, o evento `Event.COMPLETE` é despachado e o conteúdo do documento externo está disponível para uso na propriedade `data` do URLLoader, conforme mostrado no código a seguir:

```
private function completeHandler(event:Event):void
{
    var loader2:URLLoader = URLLoader(event.target);
    trace(loader2.data);
}
```

Se o documento remoto contiver pares de nome e valor, você poderá analisar os dados usando a classe URLVariables transmitindo o conteúdo do arquivo carregado, da seguinte maneira:

```
private function completeHandler(event:Event):void
{
    var loader2:URLLoader = URLLoader(event.target);
    var variables:URLVariables = new URLVariables(loader2.data);
    trace(variables.dayNames);
}
```

Cada par de nome e valor do arquivo externo é criado como uma propriedade no objeto URLVariables. Cada propriedade dentro do objeto de variáveis na amostra do código anterior é tratada como uma string. Se o valor do par de nome e valor for uma lista de itens, você poderá converter a string em uma matriz chamando o método `String.split()`, da seguinte maneira:

```
var dayNameArray:Array = variables.dayNames.split(",");
```

Se você estiver carregando dados numéricos de arquivos de texto externos, será necessário converter os valores em valores numéricos usando a função de nível superior, como `int()`, `uint()` ou `Number()`.

Em vez de carregar o conteúdo do arquivo remoto como uma string e criar um novo objeto URLVariables, você pode definir a propriedade `URLLoader.dataFormat` como uma das propriedades estáticas encontradas na classe URLLoaderDataFormat. Os três valores possíveis para a propriedade `URLLoader.dataFormat` são os seguintes:

- `URLLoaderDataFormat.BINARY` — a propriedade `URLLoader.data` contém dados binários armazenados em um objeto ByteArray.
- `URLLoaderDataFormat.TEXT` — a propriedade `URLLoader.data` contém texto em um objeto String.

- `URLLoaderDataFormat.VARIABLES` — a propriedade `URLLoader.data` contém variáveis codificadas em URL armazenadas em um objeto URLVariables.

O código a seguir demonstra como a configuração da propriedade `URLLoader.dataFormat` como `URLLoaderDataFormat.VARIABLES` permite analisar automaticamente dados carregados em um objeto URLVariables:

```
package
{
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class URLLoaderDataFormatExample extends Sprite
    {
        public function URLLoaderDataFormatExample()
        {
            var request:URLRequest = new URLRequest("http://www.[yourdomain].com/params.txt");
            var variables:URLLoader = new URLLoader();
            variables.dataFormat = URLLoaderDataFormat.VARIABLES;
            variables.addEventListener(Event.COMPLETE, completeHandler);
            try
            {
                variables.load(request);
            }
            catch (error:Error)
            {
                trace("Unable to load URL: " + error);
            }
        }
        private function completeHandler(event:Event):void
        {
        var loader:URLLoader = URLLoader(event.target);
        trace(loader.data.dayNames);
        }
    }
}
```

Nota: *O valor padrão de `URLLoader.dataFormat` é `URLLoaderDataFormat.TEXT`.*

Conforme mostrado no exemplo a seguir, carregar XML de um arquivo externo é o mesmo que carregar URLVariables. É possível criar uma ocorrência de URLRequest e uma de URLLoader e usá-las para baixar um documento XML remoto. Quando o arquivo é baixado completamente, o evento `Event.COMPLETE` é despachado e o conteúdo do arquivo externo é convertido para uma ocorrência XML, que pode ser analisada usando métodos e propriedades XML.

```
package
{
    import flash.display.Sprite;
    import flash.errors.*;
    import flash.events.*;
    import flash.net.URLLoader;
    import flash.net.URLRequest;

    public class ExternalDocs extends Sprite
    {
        public function ExternalDocs()
        {
            var request:URLRequest = new URLRequest("http://www.[yourdomain].com/data.xml");
            var loader:URLLoader = new URLLoader();
            loader.addEventListener(Event.COMPLETE, completeHandler);
            try
            {
                loader.load(request);
            }
            catch (error:ArgumentError)
            {
                trace("An ArgumentError has occurred.");
            }
            catch (error:SecurityError)
            {
                trace("A SecurityError has occurred.");
            }
        }
        private function completeHandler(event:Event):void
        {
            var dataXML:XML = XML(event.target.data);
            trace(dataXML.toXMLString());
        }
    }
}
```

Comunicação com scripts externos

Além de carregar arquivos de dados externos, você pode usar também a classe URLVariables para enviar variáveis a um script do lado do servidor e processar a resposta do servidor. Isso será útil, por exemplo, se você estiver programando um jogo e desejar enviar a pontuação do usuário a um servidor para calcular se ela deve ser adicionada à lista de pontuações altas ou mesmo enviar informações de logon do usuário a um servidor para validação. Um script do lado do servidor pode processar o nome do usuário e a senha, validá-la em relação ao banco de dados e retornar a confirmação de que as credenciais fornecidas pelo usuário são válidas.

O seguinte snippet cria um objeto URLVariables denominado `variables` que cria uma nova variável chamada `name`. Em seguida, é criado um objeto URLRequest que especifica a URL do script do lado do servidor para o qual as variáveis são enviadas. Em seguida, você define a propriedade `method` do objeto URLRequest para enviar as variáveis como uma solicitação `POST` HTTP. Para adicionar o objeto URLVariables à solicitação de URL, defina a propriedade `data` do objeto URLRequest como o objeto URLVariables criado anteriormente. Finalmente, a ocorrência de URLLoader é criada e o método `URLLoader.load()` é chamado, o que inicia a solicitação.

```
var variables:URLVariables = new URLVariables("name=Franklin");
var request:URLRequest = new URLRequest();
request.url = "http://www.[yourdomain].com/greeting.cfm";
request.method = URLRequestMethod.POST;
request.data = variables;
var loader:URLLoader = new URLLoader();
loader.dataFormat = URLLoaderDataFormat.VARIABLES;
loader.addEventListener(Event.COMPLETE, completeHandler);
try
{
    loader.load(request);
}
catch (error:Error)
{
    trace("Unable to load URL");
}

function completeHandler(event:Event):void
{
    trace(event.target.data.welcomeMessage);
}
```

O código a seguir contém o conteúdo do documento greeting.cfm do Adobe ColdFusion® usado no exemplo anterior:

```
<cfif NOT IsDefined("Form.name") OR Len(Trim(Form.Name)) EQ 0>
    <cfset Form.Name = "Stranger" />
</cfif>
<cfoutput>welcomeMessage=#UrlEncodedFormat("Welcome, " & Form.name)#
</cfoutput>
```

Conexão com outras ocorrências do Flash Player e AIR

A classe LocalConnection permite que você se comunique entre diferentes ocorrências do Flash Player e do AIR, como um SWF em um contêiner HTML ou em um player autônomo ou incorporado. Isso permite criar aplicativos muito versáteis que podem compartilhar dados entre ocorrências do Flash Player e do AIR, como arquivos SWF executados em um navegador da Web ou incorporados em aplicativos da área de trabalho.

Classe LocalConnection

A classe LocalConnection permite desenvolver arquivos SWF que podem enviar instruções a outros arquivos SWF sem o uso do método `fscommand()` ou do JavaScript. Objetos LocalConnection podem se comunicar apenas entre arquivos SWF que estão em execução no mesmo computador cliente, mas podem ser executados em diferentes aplicativos. Por exemplo, um arquivo SWF em execução em um navegador e um arquivo SWF em execução em um projetor podem compartilhar informações, com o projetor mantendo informações locais e o SWF com base em navegador conectado remotamente. (Um projetor é um arquivo SWF salvo em um formato que pode ser executado como um aplicativo autônomo, isto é, o projetor não requer que o Flash Player seja instalado porque está incorporado dentro do executável.)

Objetos LocalConnection podem ser usados para comunicação entre SWFs usando diferentes versões do ActionScript:

- Objetos LocalConnection do ActionScript 3.0 podem se comunicar com objetos LocalConnection criados no ActionScript 1.0 ou 2.0.

- Objetos LocalConnection do ActionScript 1.0 ou 2.0 podem se comunicar com objetos LocalConnection criados no ActionScript 3.0.

O Flash Player manipula automaticamente essa comunicação entre objetos LocalConnection de diferentes versões.

A maneira mais simples de usar um objeto LocalConnection é permitir a comunicação apenas entre objetos LocalConnection localizados no mesmo domínio. Desse modo, não é necessário se preocupar com problemas de segurança. No entanto, se for necessário permitir a comunicação entre domínios, há várias maneiras de implementar medidas de segurança. Para obter mais informações, consulte a discussão do parâmetro `connectionName` do método `send()` e das entradas `allowDomain()` e `domain` na listagem de classes LocalConnection na Referência dos componentes e da linguagem do ActionScript 3.0.

É possível usar objetos LocalConnection para enviar e receber dados dentro de um arquivo SWF único, mas a Adobe não recomenda esse procedimento. Em vez disso, você deve usar objetos compartilhados.

Há três maneiras de adicionar métodos de retorno de chamada aos objetos LocalConnection:

- Criar subclasse da classe LocalConnection e adicionar métodos.
- Definir a propriedade `LocalConnection.client` como um objeto que implementa os métodos.
- Criar uma classe dinâmica que estende o LocalConnection e anexar métodos dinamicamente.

A primeira maneira de adicionar métodos de retorno de chamada é estender a classe LocalConnection. Você define os métodos dentro da classe personalizada em vez de adicioná-los dinamicamente à ocorrência de LocalConnection. Essa abordagem é demonstrada no código a seguir:

```
package
{
    import flash.net.LocalConnection;
    public class CustomLocalConnection extends LocalConnection
    {
        public function CustomLocalConnection(connectionName:String)
        {
            try
            {
                connect(connectionName);
            }
            catch (error:ArgumentError)
            {
                // server already created/connected
            }
        }
        public function onMethod(timeString:String):void
        {
            trace("onMethod called at: " + timeString);
        }
    }
}
```

Para criar uma nova ocorrência da classe DynamicLocalConnection, use o seguinte código:

```
var serverLC:CustomLocalConnection;
serverLC = new CustomLocalConnection("serverName");
```

A segunda maneira de adicionar métodos de retorno de chamada é usar a propriedade `LocalConnection.client`. Isso envolve a criação de uma classe personalizada e a atribuição de uma nova ocorrência para a propriedade `cliente`, conforme mostrado no código a seguir:

```
var lc:LocalConnection = new LocalConnection();
lc.client = new CustomClient();
```

A propriedade `LocalConnection.client` indica os métodos de retorno de chamada do objeto que devem ser chamados. No código anterior, a propriedade `client` foi definida como uma nova ocorrência de uma classe personalizada, CustomClient. O valor padrão da propriedade `client` é a ocorrência LocalConnection atual. Você poderá usar a propriedade `client`, se tiver dois manipuladores de dados que tenham o mesmo conjunto de métodos, mas funcionam de modo diferente, por exemplo, em um aplicativo em que um botão em uma janela alterna a exibição em uma segunda janela.

Para criar a classe CustomClient, você pode usar o seguinte código:

```
package
{
    public class CustomClient extends Object
    {
        public function onMethod(timeString:String):void
        {
            trace("onMethod called at: " + timeString);
        }
    }
}
```

A terceira maneira de adicionar métodos de retorno de chamada criando uma classe dinâmica e anexando os métodos dinamicamente, é muito semelhante ao uso da classe LocalConnection em versões anteriores do ActionScript, conforme mostrado no código a seguir:

```
import flash.net.LocalConnection;
dynamic class DynamicLocalConnection extends LocalConnection {}
```

Os métodos de retorno de chamada podem ser adicionados dinamicamente a essa classe usando o seguinte código:

```
var connection:DynamicLocalConnection = new DynamicLocalConnection();
connection.onMethod = this.onMethod;
// Add your code here.
public function onMethod(timeString:String):void
{
    trace("onMethod called at: " + timeString);
}
```

A maneira anterior de adicionar métodos de retorno de chamada não é recomendada porque o código não é muito portátil. Além disso, o uso desse método de criação de conexões locais pode criar problemas de desempenho, porque o acesso a propriedades dinâmicas é drasticamente mais lento do que o acesso a propriedades seladas.

Envio de mensagens entre duas ocorrências do Flash Player

Você usa a classe LocalConnection para se comunicar entre diferentes ocorrências do Flash Player e do Adobe AIR. Por exemplo, você pode ter várias ocorrências do Flash Player em uma página da Web ou fazer com que uma ocorrência do Flash Player recupere dados de uma ocorrência do Flash Player em uma janela pop-up.

O código a seguir define um objeto de conexão local que funciona como um servidor e aceita chamadas de entrada de outras ocorrências do Flash Player:

```
package
{
    import flash.net.LocalConnection;
    import flash.display.Sprite;
    public class ServerLC extends Sprite
    {
        public function ServerLC()
        {
            var lc:LocalConnection = new LocalConnection();
            lc.client = new CustomClient1();
            try
            {
                lc.connect("conn1");
            }
            catch (error:Error)
            {
                trace("error:: already connected");
            }
        }
    }
}
```

Esse código cria primeiro um objeto LocalConnection denominado `lc` e define a propriedade `client` como uma classe personalizada, CustomClient1. Quando outra ocorrência do Flash Player chama um método nessa ocorrência de conexão local, o Flash Player procura esse método na classe CustomClient1.

Sempre que uma ocorrência do Flash Player se conecta a esse arquivo SWF e tenta chamar qualquer método para a conexão local especificada, a solicitação é enviada para a classe especificada pela propriedade `client`, definida como classe CustomClient1:

```
package
{
    import flash.events.*;
    import flash.system.fscommand;
    import flash.utils.Timer;
    public class CustomClient1 extends Object
    {
        public function doMessage(value:String = ""):void
        {
            trace(value);
        }
        public function doQuit():void
        {
            trace("quitting in 5 seconds");
            this.close();
            var quitTimer:Timer = new Timer(5000, 1);
            quitTimer.addEventListener(TimerEvent.TIMER, closeHandler);
        }
        public function closeHandler(event:TimerEvent):void
        {
            fscommand("quit");
        }
    }
}
```

Para criar um servidor LocalConnection, chame o método `LocalConnection.connect()` e forneça um nome de conexão exclusivo. Se você já tiver uma conexão com o nome especificado, um erro ArgumentError será gerado, indicando que houve falha durante a tentativa de conexão, porque o objeto já está conectado.

O seguinte snippet demonstra como criar uma nova conexão de soquete com o nome `conn1`:

```
try
{
    connection.connect("conn1");
}
catch (error:ArgumentError)
{
    trace("Error! Server already exists\n");
}
```

Nota:

A conexão com o arquivo SWF primário a partir de um arquivo SWF secundário requer que você crie um novo objeto LocalConnection no objeto LocalConnection de envio e chame o método `LocalConnection.send()` com o nome da conexão e o nome do método a ser executado. Por exemplo, para conectar-se ao objeto LocalConnection criado anteriormente, use o seguinte código:

```
sendingConnection.send("conn1", "doQuit");
```

Esse código conecta-se a um objeto LocalConnection existente com o nome de conexão `conn1` e chama o método `doQuit()` no arquivo SWF remoto. Para enviar parâmetros ao arquivo SWF remoto, especifique argumentos adicionais após o nome do método no método `send()`, conforme mostrado no seguinte snippet:

```
sendingConnection.send("conn1", "doMessage", "Hello world");
```

Conexão a documentos SWF em domínios diferentes

Para permitir comunicações apenas de domínios específicos, chame o método `allowDomain()` ou `allowInsecureDomain()` da classe LocalConnection e transmita uma lista de um ou mais domínios que têm permissão para acessar esse objeto LocalConnection.

Em versões anteriores do ActionScript, `LocalConnection.allowDomain()` e `LocalConnection.allowInsecureDomain()` eram métodos de retorno de chamada que precisavam ser implementados por desenvolvedores e retornar um valor booleano. No ActionScript 3.0, `LocalConnection.allowDomain()` e `LocalConnection.allowInsecureDomain()` são métodos internos que os desenvolvedores podem chamar exatamente como `Security.allowDomain()` e `Security.allowInsecureDomain()`, transmitindo um ou mais nomes de domínios a serem permitidos.

Há dois valores especiais que você pode transmitir para os métodos `LocalConnection.allowDomain()` e `LocalConnection.allowInsecureDomain()`: `*` e `host local`. O valor de asterisco `(*)` permite o acesso de todos os domínios. A string `localhost` permite chamadas para o arquivo SWF de arquivos SWF que estão instalados localmente.

O Flash Player 8 introduziu restrições de segurança em arquivos SWF locais. Um arquivo SWF com acesso permitido à Internet não pode ter acesso também ao sistema de arquivos local. Se você especificar `localhost`, qualquer arquivo SWF local poderá acessar o arquivo SWF. Se o método `LocalConnection.send()` tentar se comunicar com um arquivo SWF a partir de uma caixa de proteção de segurança à qual o código de chamada não possui acesso, um evento `securityError` (`SecurityErrorEvent.SECURITY_ERROR`) será despachado. Para solucionar esse erro, especifique o domínio do chamador no método `LocalConnection.allowDomain()` do receptor.

Se você implementar a comunicação apenas entre arquivos SWF no mesmo domínio, poderá especificar um parâmetro `connectionName` que não comece com um sublinhado (_) e que não especifique um nome de domínio (por exemplo, `myDomain:connectionName`). Use a mesma string no comando `LocalConnection.connect(connectionName)`.

Se você implementar a comunicação entre arquivos SWF em domínios diferentes, especifique um parâmetro `connectionName` que comece com um sublinhado. Especificar um sublinhado faz com que o arquivo SWF com o objeto LocalConnection recebido seja mais portátil entre domínios. Estes são os dois casos possíveis:

- Se a string de `connectionName` não começar com um sublinhado, o Flash Player adicionará um prefixo com o nome do superdomínio e um sinal de dois pontos (por exemplo, `myDomain:connectionName`). Embora isso garanta que a conexão não entre em conflito com as conexões de nome idêntico em outros domínios, todos os objetos LocalConnection de envio devem especificar esse superdomínio (por exemplo, `myDomain:connectionName`). Se você mover o arquivo SWF com o objeto LocalConnection de recebimento para outro domínio, o Flash Player alterará o prefixo para refletir o novo superdomínio (por exemplo, `anotherDomain:connectionName`). Todos os objetos LocalConnection de envio precisam ser editados manualmente para apontar para o novo superdomínio.
- Se a string do `connectionName` começar com um sublinhado (por exemplo, `_connectionName`), o Flash Player não adicionará um prefixo à string. Isso significa que os objetos LocalConnection de recebimento e envio usam strings idênticas para `connectionName`. Se o objeto de recebimento usar `LocalConnection.allowDomain()` para especificar que as conexões de qualquer domínio são aceitas, você poderá mover o arquivo SWF com o objeto LocalConnection de recebimento para outro domínio sem alterar nenhum objeto LocalConnection de envio.

conexões de soquete

Há dois tipos diferentes de conexões de soquete possíveis no ActionScript 3.0: Conexões de soquete XML e conexões de soquete binário. Um soquete XML permite que você se conecte com um servidor remoto e crie uma conexão de servidor que permaneça aberta até que seja fechada explicitamente. Isso permite a troca de dados XML entre um servidor e um cliente sem necessidade de abrir continuamente novas conexões de servidor. Outro benefício do uso de um servidor de soquete XML é que o usuário não precisa solicitar dados explicitamente. É possível enviar dados do servidor sem solicitações e enviar dados a cada cliente conectado com o servidor de soquete XML.

As conexões de soquete XML requerem a presença de uma política de soquete no servidor de destino. Para obter mais informações, consulte “Controles de site (arquivos de política)” na página 711 e“Conexão a soquetes” na página 727.

Uma conexão de soquete binário é semelhante a um soquete XML, exceto que o cliente e o servidor não precisam trocar pacotes XML especificamente. Em vez disso, a conexão pode transferir dados como informações binárias. Isso permite conectar-se a uma ampla faixa de serviços, incluindo servidores de email (POP3, SMTP e IMAP) e novos servidores (NNTP).

classe Socket

Introduzida no ActionScript 3.0, a classe Socket permite que o ActionScript faça conexões de soquete e leia e grave dados binários brutos. Ela é semelhante à classe XMLSocket, mas não determina o formato dos dados recebidos ou transmitidos. A classe Socket é útil para interoperar com servidores que usam protocolos binários. Usando conexões de soquete binário, você pode gravar código que permite a interação com vários protocolos de Internet diferentes, como POP3, SMTP, IMAP e NNTP. Isso, por sua vez, permite que o Flash Player se conecte com servidores de email e de notícias.

O Flash Player pode fazer interface com um servidor usando o protocolo binário desse servidor diretamente. Alguns servidores usam a ordem de bytes big-endian e alguns usam a ordem de bytes little-endian. A maioria dos servidores na Internet usa a ordem de bytes big-endian, porque a "ordem de bytes da rede" é big-endian. A ordem de bytes little-endian é popular porque é a usada pela arquitetura Intel® x86. Você deve usar a ordem de bytes endian que corresponde à ordem de bytes do servidor que está enviando ou recebendo dados. Todas as operações executadas pelas interfaces IDataInput e IDataOutput, e as classes que implementam essas interfaces (ByteArray, Socket e URLStream), por padrão, são codificadas no formato big-endian, isto é, com o byte mais significativo primeiro. Isso é feito para corresponder a ordem de bytes Java e a da rede oficial. Para alterar se a ordem de bytes big-endian ou a little-endian é usada, você pode definir a propriedade `endian` como `Endian.BIG_ENDIAN` ou `Endian.LITTLE_ENDIAN`.

A classe Socket herda todos os métodos implementados pelas interfaces IDataInput e IDataOutput (localizadas no pacote flash.utils), e esses métodos devem ser usados para gravar e ler a partir do Socket.

classe XMLSocket

O ActionScript fornece uma classe XMLSocket embutida, que permite abrir uma conexão contínua com um servidor. Essa conexão aberta remove problemas de latência e é usada normalmente para aplicativos em tempo real, como aplicativos de bate-papo ou jogos para vários jogadores. Uma solução tradicional de bate-papo com base em HTTP freqüentemente faz o polling do servidor e baixa novas mensagens usando uma solicitação HTTP. Em contraste, uma solução de bate-papo XMLSocket mantém uma conexão aberta com o servidor, o que permite que esse servidor envie automaticamente mensagens sem uma solicitação do cliente.

Para criar uma conexão de soquete, você deve criar um aplicativo do lado do servidor para aguardar a solicitação de conexão do soquete e enviar uma resposta ao arquivo SWF. Esse tipo de aplicativo do lado do servidor pode ser gravado em uma linguagem de programação, como Java, Python ou Perl. Para usar a classe XMLSocket, o computador servidor precisa executar um daemon que compreenda o protocolo usado pela classe XMLSocket. O protocolo é descrito na lista a seguir:

- Mensagens XML são enviadas por meio de uma conexão de soquete de fluxo TCP/IP full-duplex.
- Cada mensagem XML é um documento XML completo, terminado por um byte zero (0).
- Um número ilimitado de mensagens XML pode ser enviado e recebido por meio de uma única conexão XMLSocket.

A classe XMLSocket não pode ser encapsulada automaticamente por meio de firewalls porque, ao contrário do protocolo RTMP, a XMLSocket não tem nenhum recurso de encapsulamento HTTP. Se você precisar usar o encapsulamento HTTP, considere o uso do Flash Remoting ou do Flash Media Server (que oferece suporte a RTMP).

Nota: *Configurar um servidor para se comunicar com o objeto XMLSocket pode ser um desafio. Se o seu aplicativo não exigir interatividade em tempo real, use a classe URLLoader em vez da classe XMLSocket.*

Você pode usar os métodos `XMLSocket.connect()` e `XMLSocket.send()` da classe XMLSocket para transferir XML para o servidor e vice-versa por uma conexão de soquete. O método `XMLSocket.connect()` estabelece uma conexão de soquete com a porta do servidor da Web. O método `XMLSocket.send()` transmite um objeto XML para o servidor especificado na conexão de soquete.

Ao chamar o método `XMLSocket.connect()`, o Flash Player abre uma conexão TCP/IP com o servidor e mantém essa conexão aberta até que ocorra uma das seguintes ações:

- O método `XMLSocket.close()` da classe XMLSocket é chamado.
- Não existe mais nenhuma referência ao objeto XMLSocket.
- O Flash Player é encerrado.
- A conexão é interrompida (por exemplo, o modem se desconecta).

Criação e conexão a um servidor de soquete Java XML

O código a seguir demonstra um servidor XMLSocket simples escrito em Java que aceita conexões de entrada e exibe as mensagens recebidas na janela de prompt de comando. Por padrão, um novo servidor é criado na porta 8080 da máquina local, embora seja possível especificar um número de porta diferente ao iniciar o servidor na linha de comando.

Crie um novo documento de texto e adicione o seguinte código:

```
import java.io.*;
import java.net.*;

class SimpleServer
{
    private static SimpleServer server;
    ServerSocket socket;
    Socket incoming;
    BufferedReader readerIn;
    PrintStream printOut;

    public static void main(String[] args)
    {
        int port = 8080;

        try
        {
            port = Integer.parseInt(args[0]);
        }
        catch (ArrayIndexOutOfBoundsException e)
        {
            // Catch exception and keep going.
        }

        server = new SimpleServer(port);
    }

    private SimpleServer(int port)
    {
        System.out.println(">> Starting SimpleServer");
        try
        {
            socket = new ServerSocket(port);
            incoming = socket.accept();
            readerIn = new BufferedReader(new InputStreamReader(incoming.getInputStream()));
            printOut = new PrintStream(incoming.getOutputStream());
            printOut.println("Enter EXIT to exit.\r");
            out("Enter EXIT to exit.\r");
            boolean done = false;
            while (!done)
            {
                String str = readerIn.readLine();
                if (str == null)
                {
                    done = true;
                }
                else
                {
```

```
                    out("Echo: " + str + "\r");
                    if(str.trim().equals("EXIT"))
                    {
                        done = true;
                    }
                }
                incoming.close();
            }
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }

    private void out(String str)
    {
        printOut.println(str);
        System.out.println(str);
    }
}
```

Salve o documento no disco rígido como SimpleServer.java e compile-o usando um compilador Java que cria um arquivo de classe Java denominado SimpleServer.class.

Você pode iniciar o servidor XMLSocket abrindo um prompt de comando e digitando `java SimpleServer`. O arquivo SimpleServer.class pode estar localizado em qualquer lugar no computador local ou na rede. Ele não precisa ser colocado no diretório raiz do servidor da Web.


Se você não puder iniciar o servidor porque os arquivos não estão localizados dentro do caminho de classe Java, tente iniciar o servidor com `java -classpath. SimpleServer`.

Para se conectar ao XMLSocket do aplicativo ActionScript, é necessário criar uma nova ocorrência da classe XMLSocket, e chamar o método `XMLSocket.connect()` ao transmitir um nome do host e um número de porta, da seguinte maneira:

```
var xmlsock:XMLSocket = new XMLSocket();
xmlsock.connect("127.0.0.1", 8080);
```

Sempre que você receber dados do servidor, o evento `data` (`flash.events.DataEvent.DATA`) é despachado:

```
xmlsock.addEventListener(DataEvent.DATA, onData);
private function onData(event:DataEvent):void
{
    trace("[" + event.type + "] " + event.data);
}
```

Para enviar dados ao servidor XMLSocket, use o método `XMLSocket.send()` e transmita um objeto ou string XML. O Flash Player converte o parâmetro fornecido em um objeto String e envia o conteúdo ao servidor XMLSocket seguido de um byte zero (0):

```
xmlsock.send(xmlFormattedData);
```

O método `XMLSocket.send()` não retorna um valor que indica se os dados foram transmitidos com êxito. Se ocorrer um erro ao tentar enviar dados, um erro IOError será emitido.


Cada mensagem enviada ao servidor de soquete XML deve ser terminada por uma nova linha (caractere `\n`).

Armazenamento de dados locais

Um objeto compartilhado, às vezes referenciado como um "cookie Flash". é um arquivo de dados que pode ser criado no computador pelos sites visitados. Objetos compartilhados são usados com maior freqüência para aprimorar sua experiência de navegação na Web, por exemplo, permitindo personalizar a aparência de um site visitado com freqüência. Objetos compartilhados, por si só, não podem fazer nada em relação aos dados no computador. O mais importante é que os objetos compartilhados nunca podem acessar ou lembrar seu endereço de email ou outras informações pessoais, a não ser que você forneça essas informações voluntariamente.

Novas ocorrências de objetos compartilhados podem ser criadas usando os métodos estáticos `SharedObject.getLocal()` ou `SharedObject.getRemote()`. O método `getLocal()` tenta carregar um objeto persistente compartilhado localmente que está disponível apenas para o cliente atual, enquanto que o método `getRemote()` tenta carregar um objeto compartilhado remoto que pode ser compartilhado entre vários clientes por meio de um servidor, como o Flash Media Server. Se o objeto compartilhado local ou remoto não existir, os métodos `getLocal()` e `getRemote()` criarão uma nova ocorrência de SharedObject.

O código a seguir tenta carregar um objeto compartilhado local denominado `test`. Se esse objeto compartilhado não existir, um novo objeto compartilhado com esse nome será criado.

```
var so:SharedObject = SharedObject.getLocal("test");
trace("SharedObject is " + so.size + " bytes");
```

Se um objeto compartilhado denominado teste não puder ser encontrado, um novo será criado com um tamanho de 0 bytes. Se o objeto compartilhado existia anteriormente, seu tamanho atual (em bytes) será retornado.

É possível armazenar dados em um objeto compartilhado atribuindo valores ao objeto de dados, como mostrado no exemplo a seguir:

```
var so:SharedObject = SharedObject.getLocal("test");
so.data.now = new Date().time;
trace(so.data.now);
trace("SharedObject is " + so.size + " bytes");
```

Se um objeto compartilhado já existir com o nome `test` e o parâmetro `now`, o valor existente será substituído. Você pode usar a propriedade `SharedObject.size` para determinar se um objeto compartilhado já existe, conforme mostrado no exemplo a seguir:

```
var so:SharedObject = SharedObject.getLocal("test");
if (so.size == 0)
{
    // Shared object doesn't exist.
    trace("created...");
    so.data.now = new Date().time;
}
trace(so.data.now);
trace("SharedObject is " + so.size + " bytes");
```

O código anterior usa o parâmetro `size` para determinar se a ocorrência do objeto compartilhado com o nome especificado já existe. Se você testar o código a seguir, observará que cada vez que o código é executado, o objeto compartilhado será criado novamente. Para que um objeto compartilhado seja salvo na unidade de disco rígido do usuário, você deve chamar explicitamente o método `SharedObject.flush()`, conforme mostrado no exemplo a seguir:

```
var so:SharedObject = SharedObject.getLocal("test");
if (so.size == 0)
{
    // Shared object doesn't exist.
    trace("created...");
    so.data.now = new Date().time;
}
trace(so.data.now);
trace("SharedObject is " + so.size + " bytes");
so.flush();
```

Ao usar o método `flush()` para gravar objetos compartilhados em uma unidade de disco rígido do usuário, verifique atentamente se o usuário desativou explicitamente o armazenamento local usando o Gerenciador de configurações do Flash Player (www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager07.html), conforme mostrado no exemplo a seguir:

```
var so:SharedObject = SharedObject.getLocal("test");
trace("Current SharedObject size is " + so.size + " bytes.");
so.flush();
```

Os valores podem ser recuperados de um objeto compartilhado especificando o nome da propriedade `data` do objeto compartilhado. Por exemplo, se você executar o código a seguir, o Flash Player exibirá o número de minutos em que a ocorrência do SharedObject foi criada:

```
var so:SharedObject = SharedObject.getLocal("test");
if (so.size == 0)
{
    // Shared object doesn't exist.
    trace("created...");
    so.data.now = new Date().time;
}
var ageMS:Number = new Date().time - so.data.now;
trace("SharedObject was created " + Number(ageMS / 1000 / 60).toPrecision(2) + " minutes ago");
trace("SharedObject is " + so.size + " bytes");
so.flush();
```

Na primeira vez que o código anterior for executado, uma nova ocorrência de SharedObject denominada `test` será criada e terá um tamanho inicial de 0 bytes. Como o tamanho inicial é 0 bytes, a instrução `if` é avaliada como `true` e uma nova propriedade denominada `now` é adicionada ao objeto compartilhado local. A idade do objeto compartilhado é calculada subtraindo o valor da propriedade `now` da hora atual. A cada vez subseqüente em que o código anterior for executado, o tamanho do objeto compartilhado deverá ser superior a 0, e o código rastreará há quantos minutos o objeto compartilhado foi criado.

Exibição de conteúdo de um objeto compartilhado

Os valores são armazenados em objetos compartilhados dentro da propriedade `data`. Você pode executar um loop em cada valor dentro de uma ocorrência do objeto compartilhado usando um loop `for..in`, conforme mostrado no exemplo a seguir:

```
var so:SharedObject = SharedObject.getLocal("test");
so.data.hello = "world";
so.data.foo = "bar";
so.data.timezone = new Date().timezoneOffset;
for (var i:String in so.data)
{
    trace(i + ":\t" + so.data[i]);
}
```

Criação de um SharedObject seguro

Ao criar um SharedObject remoto ou local usando `getLocal()` ou `getRemote()`, há um parâmetro opcional denominado `secure` que determina se o acesso a esse objeto compartilhado é restrito para arquivos SWF que são entregues por uma conexão HTTPS. Se esse parâmetro estiver definido como `true` e o arquivo SWF for entregue por HTTPS, o Flash Player criará um novo objeto compartilhado protegido ou obterá uma referência a um objeto compartilhado protegido existente. Esse objeto compartilhado protegido pode ser lido ou gravado apenas por arquivos SWF entregues por HTTPS que chamam `SharedObject.getLocal()` com o parâmetro secure definido como `true`. Se esse parâmetro estiver definido como `false` e o arquivo SWF for entregue por HTTPS, o Flash Player criará um novo objeto compartilhado ou obterá uma referência a um objeto compartilhado existente.

Esse objeto compartilhado pode ser lido ou gravado por arquivos SWF entregues por conexões não-HTTPS. Se o arquivo SWF for entregue por uma conexão não-HTTPS e você tentar definir esse parâmetro como `true`, haverá uma falha na criação de um novo objeto compartilhado (ou o acesso de um objeto compartilhado protegido criado anteriormente), um erro será emitido e o objeto compartilhado será definido como `null`. Se você tentar executar o seguinte snippet em uma conexão não-HTTPS, o método `SharedObject.getLocal()` emitirá um erro:

```
try
{
    var so:SharedObject = SharedObject.getLocal("contactManager", null, true);
}
catch (error:Error)
{
    trace("Unable to create SharedObject.");
}
```

Independentemente do valor desse parâmetro, os objetos compartilhados criados serão levados em consideração na contagem da quantidade total de espaço em disco permitida para um domínio.

Trabalho com arquivos de dados

Um objeto FileReference representa um arquivo de dados em um computador cliente ou servidor. Os métodos da classe FileReference permitem que o seu aplicativo carregue e salve os arquivos de dados localmente, além de transferir dados de arquivos para ou de servidores remotos.

A classe FileReference agora oferece duas abordagens diferentes para carregamento, transferência e gravação de arquivos de dados. A abordagem original usa o método browse() para permitir ao usuário selecionar um arquivo, o método upload() para transferir os dados do arquivo para um servidor remoto e o método download() para recuperar os dados do servidor e salvá-los em um arquivo local. As versões recentes de tempo de execução do Flash Player e do AIR simplificam esse processo com os novos métodos FileReference load() e save(), similares aos usados nas classes URLLoader e Loader. Essa seção discute as duas abordagens.

Nota: *O tempo de execução do AIR fornece classes adicionais (no pacote flash.filesystem) para trabalho com os arquivos e com o sistema de arquivos local. As classes flash.filesystem oferecem mais recursos do que a classe FileReference, mas são apenas suportadas no tempo de execução do AIR e não no Flash Player.*

classe FileReference

Cada objeto FileReference refere-se a um arquivo de dados único no computador local. As propriedades da classe FileReference contêm informações sobre as seguintes características do arquivo: tamanho, tipo, nome, extensão, criador, data de criação e data de modificação.

Nota: *Há suporte para a propriedade `creator` apenas no Mac OS. Todas as outras plataformas retornam `null`.*

Nota: *A propriedade* `extension` *é suportada somente no tempo de execução do AIR.*

Você pode criar uma ocorrência da classe FileReference de duas maneiras:

1 Use o operador `new`, conforme apresentado no código a seguir:

```
import flash.net.FileReference;
var fileRef:FileReference = new FileReference();
```

2 Chame o método `FileReferenceList.browse()`, que abre uma caixa de diálogo que solicita ao usuário selecionar um ou mais arquivos para carregar. Ele pode criar uma matriz de objetos FileReference se o usuário conseguir selecionar um ou mais arquivos.

Após criar um objeto FileReference, você pode proceder da seguinte maneira:

1 Chame o método `FileReference.browse()`, que abre uma caixa de diálogo que solicita ao usuário selecionar um ou mais arquivos no sistema de arquivos local. Geralmente, isso é feito antes de uma chamada subseqüente para o método FileReference.upload() para carregar o arquivo em um servidor remoto.

2 Chame o método FileReference.download(). Uma caixa de diálogo será aberta para possibilitar a seleção de um local para gravação de um novo arquivo. Em seguida, os dados do servidor serão baixados e armazenados no novo arquivo.

3 Chame o método `FileReference.load()`, que abre uma caixa de diálogo e solicita ao usuário selecionar um único arquivo do sistema de arquivos local e, em seguida, inicia o carregamento dos dados do arquivo selecionado.

4 Chame o método `FileReference.save()`, que abre uma caixa de diálogo e solicita ao usuário selecionar um único arquivo do sistema de arquivos local e, em seguida, inicia o carregamento dos dados do arquivo selecionado.

Nota: *Você pode executar apenas uma ação* `browse()` *ou* `download()` *de cada vez, porque apenas uma caixa de diálogo pode ser aberta em qualquer ponto.*

As propriedades do objeto FileReference, como `name`, `size` ou `modificationDate` não serão preenchidas até que ocorra um dos eventos a seguir:

- O método `FileReference.browse()` ou o método `FileReferenceList.browse()` foi chamado e o usuário selecionou um arquivo usando a caixa de diálogo.
- O método `FileReference.download()` foi chamado e o usuário especificou um novo local para os arquivos usando a caixa de diálogo.
- O método `FileReference.load()` foi chamado e o usuário selecionou um arquivo usando a caixa de diálogo.

Nota: *Ao fazer um download, apenas a propriedade* `FileReference.name` *é preenchida antes do download ser concluído. Após o download do arquivo, todas as propriedades estão disponíveis.*

Enquanto as chamadas dos métodos `FileReference.browse()`, `FileReferenceList.browse()`, `FileReference.download()`, `FileReference.load()` ou `FileReference.save()` são executadas, a maioria dos players continua a reproduzir o arquivo SWF.

Para operações de upload e download, um arquivo SWF pode acessar arquivos apenas em seu próprio domínio, incluindo todos os domínios especificados por um arquivo de política. Será necessário colocar um arquivo de política no servidor de arquivos, se o arquivo SWF que estiver iniciando o upload ou o download não for proveniente do mesmo domínio que o servidor de arquivos.

Carregamento de dados de arquivos

O método FileReference.load() lhe permite carregar dados na memória a partir de um arquivo local. O seu código deve chamar primeiro o método FileReference.browse() para permitir que o usuário selecione um arquivo a ser carregado.

O método FileReference.load() é retornado imediatamente após a chamada. Em seguida, o objeto FileReference despacha eventos para chamar métodos de ouvintes a cada etapa do processo de carregamento.

O objeto FileReference pode despachar os eventos a seguir durante o processo de carregamento.

- `Event.OPEN`: despachado quando uma operação de carregamento é iniciada.
- `ProgressEvent.PROGRESS`: despachado periodicamente na medida em que os dados do arquivo são lidos.
- `Event.COMPLETE`: despachado quando a operação de carregamento é concluída com sucesso.
- `IOErrorEvent.IO_ERROR`: despachado se o processo de carregamento falhar devido a um erro de entrada/saída durante a abertura ou leitura de dados do arquivo.

O exemplo a seguir mostra como solicitar ao usuário a seleção de um arquivo e como carregar os dados do arquivo na memória:

```
package
{
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.FileFilter;
    import flash.net.FileReference;
    import flash.net.URLRequest;
    import flash.utils.ByteArray;

    public class FileReferenceExample1 extends Sprite
    {
        private var fileRef:FileReference;
        public function FileReferenceExample1()
        {
            fileRef = new FileReference();
            fileRef.addEventListener(Event.SELECT, onFileSelected);
            fileRef.addEventListener(Event.CANCEL, onCancel);
            fileRef.addEventListener(IOErrorEvent.IO_ERROR, onIOError);
            fileRef.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
                        onSecurityError);
            var textTypeFilter:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)",
                        "*.txt;*.rtf");
            fileRef.browse([textTypeFilter]);
        }
        public function onFileSelected(evt:Event):void
        {
            fileRef.addEventListener(ProgressEvent.PROGRESS, onProgress);
            fileRef.addEventListener(Event.COMPLETE, onComplete);
            fileRef.load();
        }

        public function onProgress(evt:ProgressEvent):void
        {
            trace("Loaded " + evt.bytesLoaded + " of " + evt.bytesTotal + " bytes.");
        }

        public function onComplete(evt:Event):void
        {
            trace("File was successfully loaded.");
            trace(fileRef.data);
        }
```

```
        public function onCancel(evt:Event):void
        {
            trace("The browse request was canceled by the user.");
        }

        public function onSaveCancel(evt:Event):void
        {
            trace("The save request was canceled by the user.");
        }

        public function onIOError(evt:IOErrorEvent):void
        {
            trace("There was an IO Error.");
        }
        public function onSecurityError(evt:Event):void
        {
            trace("There was a security error.");
        }
    }
}
```

O código de exemplo cria o objeto FileReference primeiro e, em seguida, chama o método FileReference.browse(), que abre uma caixa de diálogo que solicita ao usuário selecionar um arquivo. Quando um arquivo é selecionado, o método onFileSelected() é chamado. Esses métodos adicionam ouvintes para os eventos Event.PROGRESS e Event.COMPLETE e, em seguida, chamam o método FileReference.load(). Os outros métodos do manipulador do exemplo rastreiam strings de saída para o relatório em execução para a operação de carregamento. Quando a carga estiver completa, o conteúdo do arquivo carregado será exibido por meio do método trace().

Gravação de dados em arquivos locais

O método FileReference.save() lhe permite salvar dados em um arquivo local. Ele é iniciado ao abrir uma caixa de diálogo para permitir ao usuário digitar um novo nome para o arquivo e definir um local onde o arquivo será salvo. Após a seleção do nome do arquivo e do local, os dados são gravados no novo arquivo. Quando o arquivo é salvo com sucesso, as propriedades do objeto FileReference são substituídas pelas do arquivo local.

Nota: *O código deverá chamar apenas o método FileReference.save() em resposta ao evento de usuário, como um manipulador de eventos para um evento de clique de mouse ou de pressionamento de tecla. Caso contrário, um erro será gerado.*

O método FileReference.save() é retornado imediatamente após a chamada. Em seguida, o objeto FileReference despacha eventos para chamar métodos de ouvintes em cada etapa do processo de gravação do arquivo.

O objeto FileReference despacha os eventos a seguir durante o processo de gravação do arquivo:

- `Event.SELECT`: despachado quando o usuário especifica o local e o nome do arquivo para o novo arquivo a ser salvo.
- `Event.CANCEL`: despachado quando o usuário clica no botão Cancelar da caixa de diálogo.
- `Event.OPEN`: despachado quando uma operação de gravação é iniciada.
- `ProgressEvent.PROGRESS`: despachado periodicamente na medida em que os dados são gravados no arquivo.
- `Event.COMPLETE`: despachado quando a operação de gravação é concluída com sucesso.
- `IOErrorEvent.IO_ERROR`: despachado se o processo de carregamento falhar devido a um erro de entrada/saída durante a abertura ou leitura de dados do arquivo.

O tipo de objeto transmitido no parâmetro de dados do método FileReference.save() determinará como os dados serão gravados no arquivo:

- Se for um valor String, eles serão salvos como um arquivo de texto usando a codificação UTF-8.
- Se for um objeto XML, eles serão gravados em um arquivo no formato XML com todas as formatações originais.
- Se for um objeto ByteArray, seu conteúdo será gravado diretamente no arquivo, sem conversão.
- Se for outro tipo de objeto, o método FileReference.save() chamará o método toString() para o objeto e salvará o valor String resultante em um arquivo de texto UTF-8. Se o método toString() do objeto não puder ser chamado, um erro será gerado.

Se o valor do parâmetro de dados for null, um erro será gerado.

O código a seguir também se aplica ao exemplo apresentado acima para o método FileReference.load(). Após a leitura dos dados do arquivo, esse exemplo solicita ao usuário um nome de arquivo e, em seguida, salva os dados em um novo arquivo:

```
package
{
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.FileFilter;
    import flash.net.FileReference;
    import flash.net.URLRequest;
    import flash.utils.ByteArray;

    public class FileReferenceExample2 extends Sprite
    {
        private var fileRef:FileReference;
        public function FileReferenceExample2()
        {
            fileRef = new FileReference();
            fileRef.addEventListener(Event.SELECT, onFileSelected);
            fileRef.addEventListener(Event.CANCEL, onCancel);
            fileRef.addEventListener(IOErrorEvent.IO_ERROR, onIOError);
            fileRef.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
                        onSecurityError);
            var textTypeFilter:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)",
                        "*.txt;*.rtf");
            fileRef.browse([textTypeFilter]);
        }
        public function onFileSelected(evt:Event):void
        {
            fileRef.addEventListener(ProgressEvent.PROGRESS, onProgress);
            fileRef.addEventListener(Event.COMPLETE, onComplete);
            fileRef.load();
        }

        public function onProgress(evt:ProgressEvent):void
        {
            trace("Loaded " + evt.bytesLoaded + " of " + evt.bytesTotal + " bytes.");
        }
        public function onCancel(evt:Event):void
        {
            trace("The browse request was canceled by the user.");
        }
        public function onComplete(evt:Event):void
```

```
        {
            trace("File was successfully loaded.");
            fileRef.removeEventListener(Event.SELECT, onFileSelected);
            fileRef.removeEventListener(ProgressEvent.PROGRESS, onProgress);
            fileRef.removeEventListener(Event.COMPLETE, onComplete);
            fileRef.removeEventListener(Event.CANCEL, onCancel);
            saveFile();
        }
        public function saveFile():void
        {
            fileRef.addEventListener(Event.SELECT, onSaveFileSelected);
            fileRef.save(fileRef.data,"NewFileName.txt");
        }

        public function onSaveFileSelected(evt:Event):void
        {
            fileRef.addEventListener(ProgressEvent.PROGRESS, onSaveProgress);
            fileRef.addEventListener(Event.COMPLETE, onSaveComplete);
            fileRef.addEventListener(Event.CANCEL, onSaveCancel);
        }

        public function onSaveProgress(evt:ProgressEvent):void
        {
            trace("Saved " + evt.bytesLoaded + " of " + evt.bytesTotal + " bytes.");
        }

        public function onSaveComplete(evt:Event):void
        {
            trace("File saved.");
            fileRef.removeEventListener(Event.SELECT, onSaveFileSelected);
            fileRef.removeEventListener(ProgressEvent.PROGRESS, onSaveProgress);
            fileRef.removeEventListener(Event.COMPLETE, onSaveComplete);
            fileRef.removeEventListener(Event.CANCEL, onSaveCancel);
        }

        public function onSaveCancel(evt:Event):void
        {
            trace("The save request was canceled by the user.");
        }

        public function onIOError(evt:IOErrorEvent):void
        {
            trace("There was an IO Error.");
        }
        public function onSecurityError(evt:Event):void
        {
            trace("There was a security error.");
        }
    }
}
```

Quando todos os dados do arquivo são carregados, o método onComplete() é chamado. O método onComplete() removeu os ouvintes para os eventos de carregamento e, em seguida, chamou o método saveFile(). O método saveFile() chama o método FileReference.save(), que abre uma nova caixa de diálogo que permite ao usuário digitar um novo nome para o arquivo, além de seu local de gravação. Os métodos restantes do ouvinte de eventos rastreiam o andamento do processo de gravação do arquivo até que seja concluído.

Upload de arquivos em um servidor

Para carregar arquivos para um servidor, chame primeiro o método `browse()` para permitir que um usuário selecione um ou mais arquivos. Em seguida, quando o método `FileReference.upload()` for chamado, o arquivo selecionado será transferido para o servidor. Se o usuário selecionou vários arquivos usando o método `FileReferenceList.browse()`, o Flash Player criará uma matriz dos arquivos selecionados chamada `FileReferenceList.fileList`. Em seguida, você pode usar o método `FileReference.upload()` para carregar cada arquivo individualmente.

Nota: *O uso do método `FileReference.browse()` permite carregar apenas arquivos únicos. Para permitir que um usuário carregue vários arquivos, você deve usar o método `FileReferenceList.browse()`.*

Por padrão, a caixa de diálogo de seletor de arquivos do sistema permite que os usuários selecionem qualquer tipo de arquivo do computador local, embora os desenvolvedores possam especificar um ou mais filtros do tipo de arquivo personalizado usando a classe FileFilter e transmitindo uma matriz de ocorrências de filtros de arquivo para o método `browse()`:

```
var imageTypes:FileFilter = new FileFilter("Images (*.jpg, *.jpeg, *.gif, *.png)", "*.jpg;
*.jpeg; *.gif; *.png");
var textTypes:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)", "*.txt; *.rtf");
var allTypes:Array = new Array(imageTypes, textTypes);
var fileRef:FileReference = new FileReference();
fileRef.browse(allTypes);
```

Quando o usuário seleciona os arquivos e clica no botão Abrir no seletor de arquivos do sistema, o evento `Event.SELECT` é despachado. Se o método `FileReference.browse()` for usado para selecionar um arquivo a ser carregado, o seguinte código será necessário para enviar o arquivo a um servidor da Web:

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
try
{
    var success:Boolean = fileRef.browse();
}
catch (error:Error)
{
    trace("Unable to browse for files.");
}
function selectHandler(event:Event):void
{
    var request:URLRequest = new URLRequest("http://www.[yourdomain].com/fileUploadScript.cfm")
    try
    {
        fileRef.upload(request);
    }
    catch (error:Error)
    {
        trace("Unable to upload file.");
    }
}
function completeHandler(event:Event):void
{
    trace("uploaded");
}
```

Você pode enviar dados ao servidor com o método `FileReference.upload()` usando as propriedades `URLRequest.method` e `URLRequest.data` para enviar variáveis que usam os métodos `POST` ou `GET`.

Ao tentar carregar um arquivo usando o método `FileReference.upload()`, qualquer um dos seguintes eventos pode ser despachado.

- `Event.OPEN`: despachado quando uma operação de upload é iniciada.
- `ProgressEvent.PROGRESS`: despachado periodicamente na medida em que os dados do arquivo são carregados.
- `Event.COMPLETE`: despachado quando a operação de upload é concluída com sucesso.
- `HTTPStatusEvent.HTTP_STATUS`: despachado quando há uma falha no processo de upload devido a um erro HTTP.
- HTTPStatusEvent.HTTP_RESPONSE_STATUS: despachado se uma chamada do método upload() ou uploadUnencoded() tentar acessar os dados via HTTP e o Adobe AIR puder detectar e retornar o código de status para a solicitação.
- `SecurityErrorEvent.SECURITY_ERROR`: despachado quando há uma falha na operação de upload devido a uma violação de segurança.
- DataEvent.UPLOAD_COMPLETE_DATA: despachado quando os dados são recebidos do servidor após um upload bem-sucedido.
- `IOErrorEvent.IO_ERROR`: despachado se houver falha no processo de upload por causa de qualquer um dos seguintes motivos:
  - Ocorreu um erro de entrada/saída enquanto o Flash Player estava lendo, gravando ou transmitindo o arquivo.
  - O SWF tentou carregar um arquivo para um servidor que requer autenticação (como um nome de usuário e senha). Durante o upload, o Flash Player não fornece um meio para os usuários inserirem senhas.
  - O parâmetro `url` contém um protocolo inválido. O método `FileReference.upload()` deve usar HTTP ou HTTPS.

O Flash Player não oferece suporte completo para servidores que requerem autenticação. Apenas arquivos SWF que estão em execução em um navegador que usa o plug-in do navegador ou o controle Microsoft ActiveX® podem fornecer uma caixa de diálogo para solicitar que o usuário digite um nome de usuário e senha para autenticação e apenas para downloads. Ocorre falha na transferência de arquivos, para uploads que usam o plug-in ou o controle ActiveX, ou upload/download que usa player autônomo ou externo.

Se você criar um script de servidor no ColdFusion para aceitar um upload de arquivo do Flash Player, poderá usar código semelhante para o seguinte:

```
<cffile action="upload" filefield="Filedata" destination="#ExpandPath('./')#"
nameconflict="OVERWRITE" />
```

Esse código ColdFusion carrega o arquivo enviado pelo Flash Player e salva-o no mesmo diretório que o modelo ColdFusion, substituindo qualquer arquivo com o mesmo nome. O código anterior mostra a quantidade mínima básica de código necessária para aceitar um upload de arquivo. Esse script não deve ser usado em um ambiente de produção. Em condições ideais, você deve adicionar validação de dados para garantir que os usuários carreguem apenas tipos de arquivo aceitos, como uma imagem, em vez de um script do lado do servidor potencialmente perigoso.

O código a seguir demonstra uploads de arquivo usando PHP e inclui validação de dados. O script limita o número de arquivos carregados no diretório de upload até 10, garante que o arquivo tenha menos do que 200 KB e permite que apenas arquivos JPEG, GIF ou PNG sejam carregados e salvos no sistema de arquivos.

```
<?php
$MAXIMUM_FILESIZE = 1024 * 200; // 200KB
$MAXIMUM_FILE_COUNT = 10; // keep maximum 10 files on server
echo exif_imagetype($_FILES['Filedata']);
if ($_FILES['Filedata']['size'] <= $MAXIMUM_FILESIZE)
{
    move_uploaded_file($_FILES['Filedata']['tmp_name'],
"./temporary/".$_FILES['Filedata']['name']);
    $type = exif_imagetype("./temporary/".$_FILES['Filedata']['name']);
    if ($type == 1 || $type == 2 || $type == 3)
    {
        rename("./temporary/".$_FILES['Filedata']['name'],
"./images/".$_FILES['Filedata']['name']);
    }
    else
    {
        unlink("./temporary/".$_FILES['Filedata']['name']);
    }
}
$directory = opendir('./images/');
$files = array();
while ($file = readdir($directory))
{
    array_push($files, array('./images/'.$file, filectime('./images/'.$file)));
}
usort($files, sorter);
if (count($files) > $MAXIMUM_FILE_COUNT)
{
    $files_to_delete = array_splice($files, 0, count($files) - $MAXIMUM_FILE_COUNT);
    for ($i = 0; $i < count($files_to_delete); $i++)
    {
        unlink($files_to_delete[$i][0]);
    }
}
print_r($files);
closedir($directory);

function sorter($a, $b)
{
    if ($a[1] == $b[1])
    {
        return 0;
    }
    else
    {
        return ($a[1] < $b[1]) ? -1 : 1;
    }
}
?>
```

Você pode transmitir variáveis adicionais para o script de upload usando o método de solicitação `POST` ou o`GET`. Para enviar variáveis `POST` adicionais ao script de upload, use o seguinte código:

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
fileRef.browse();
function selectHandler(event:Event):void
{
    var params:URLVariables = new URLVariables();
    params.date = new Date();
    params.ssid = "94103-1394-2345";
    var request:URLRequest = new
URLRequest("http://www.yourdomain.com/FileReferenceUpload/fileupload.cfm");
    request.method = URLRequestMethod.POST;
    request.data = params;
    fileRef.upload(request, "Custom1");
}
function completeHandler(event:Event):void
{
    trace("uploaded");
}
```

O exemplo anterior cria um novo objeto URLVariables que você transmite para o script do lado do servidor remoto. Em versões anteriores do ActionScript, era possível transmitir variáveis para o script de upload de servidor transmitindo valores na string da consulta. O ActionScript 3.0 permite transmitir variáveis para o script remoto usando um objeto URLRequest, o que permite que você transmita dados usando o método `POST` ou o `GET`. Isso, por sua vez, faz com que a transmissão de conjuntos de dados maiores seja mais fácil e mais limpa. Para especificar se a variáveis são transmitidas usando o método de solicitação `GET` ou `POST`, defina a propriedade `URLRequest.method` como `URLRequestMethod.GET` ou `URLRequestMethod.POST`, respectivamente.

O ActionScript 3.0 também permite substituir o nome do campo do arquivo de upload `Filedata` padrão fornecendo um segundo parâmetro para o método `upload()`, conforme demonstrado no exemplo anterior (que substituiu o valor padrão `Filedata` por `Custom1`).

Por padrão, o Flash Player não tenta enviar um upload de teste, embora seja possível substituir isso transmitindo um valor `true` como o terceiro parâmetro para o método `upload()`. A finalidade do upload de teste é verificar se o upload do arquivo real será bem-sucedido e se a autenticação do servidor, se necessária, terá êxito.

Nota: *No momento, um upload de teste ocorre apenas em Flash Players com base no Windows.*

O script de servidor que manipula o upload do arquivo deve esperar uma solicitação `POST` HTTP com os seguintes elementos:

- `Content-Type` com um valor de `multipart/form-data`.
- `Content-Disposition` com um atributo `name` definido como "`Filedata`" e um atributo `filename` definido como o nome do arquivo original. É possível especificar um atributo `name` personalizado, transmitindo um valor para o parâmetro `uploadDataFieldName` no método `FileReference.upload()`.
- O conteúdo binário do arquivo.

Esta é uma solicitação `POST` HTTP de amostra:

```
POST /handler.asp HTTP/1.1
Accept: text/*
Content-Type: multipart/form-data;
boundary=----------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
User-Agent: Shockwave Flash
Host: www.mydomain.com
Content-Length: 421
Connection: Keep-Alive
Cache-Control: no-cache

------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Filename"

sushi.jpg
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Filedata"; filename="sushi.jpg"
Content-Type: application/octet-stream

Test File
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Upload"

Submit Query
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
(actual file data,,,)
```

A seguinte solicitação `POST` HTTP de amostra envia três variáveis `POST`: `api_sig`, `api_key` e `auth_token` e usa um valor de nome de campo de dados de upload personalizado de `"photo"`:

```
POST /handler.asp HTTP/1.1
Accept: text/*
Content-Type: multipart/form-data;
boundary=----------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
User-Agent: Shockwave Flash
Host: www.mydomain.com
Content-Length: 421
Connection: Keep-Alive
Cache-Control: no-cache

------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="Filename"

sushi.jpg
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="api_sig"

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="api_key"

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="auth_token"

XXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="photo"; filename="sushi.jpg"
Content-Type: application/octet-stream

(actual file data,,,)
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="Upload"

Submit Query
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7--
```

Download de arquivos de um servidor

É possível permitir que os usuários baixem arquivos de um servidor usando o método `FileReference.download()` que utiliza dois parâmetros: `request` e `defaultFileName`. O primeiro parâmetro é o objeto URLRequest que contém a URL do arquivo a ser baixado. O segundo parâmetro é opcional, ele permite especificar um nome de arquivo padrão que aparece na caixa de diálogo do arquivo de download. Se você omitir o segundo parâmetro, `defaultFileName`, o nome de arquivo da URL especificada será usado.

O código a seguir baixa um arquivo denominado index.xml do mesmo diretório que o documento SWF:

```
var request:URLRequest = new URLRequest("index.xml");
var fileRef:FileReference = new FileReference();
fileRef.download(request);
```

Para definir o nome padrão como currentnews.xml, em vez de index.xml, especifique o parâmetro `defaultFileName`, conforme mostrado no seguinte snippet:

```
var request:URLRequest = new URLRequest("index.xml");
var fileToDownload:FileReference = new FileReference();
fileToDownload.download(request, "currentnews.xml");
```

Renomear um arquivo poderá ser muito útil, se o nome de arquivo do servidor não for intuitivo ou gerado pelo servidor. Também é bom especificar explicitamente o parâmetro `defaultFileName` ao baixar um arquivo usando um script do lado do servidor, em vez de baixar o arquivo diretamente. Por exemplo, você precisa especificar o parâmetro `defaultFileName`, se tiver um script do lado do servidor que baixe arquivos específicos com base em variáveis de URL transmitidas para ele. Caso contrário, o nome padrão do arquivo baixado será o nome do script do lado do servidor.

Os dados podem ser enviados ao servidor usando o método `download()` anexando parâmetros à URL para análise pelo script do servidor. O seguinte snippet do ActionScript 3.0 baixa um documento com base em quais parâmetros são transmitidos para um script ColdFusion:

```
package
{
    import flash.display.Sprite;
    import flash.net.FileReference;
    import flash.net.URLRequest;
    import flash.net.URLRequestMethod;
    import flash.net.URLVariables;

    public class DownloadFileExample extends Sprite
    {
        private var fileToDownload:FileReference;
        public function DownloadFileExample()
        {
            var request:URLRequest = new URLRequest();
            request.url = "http://www.[yourdomain].com/downloadfile.cfm";
            request.method = URLRequestMethod.GET;
            request.data = new URLVariables("id=2");
            fileToDownload = new FileReference();
            try
            {
                fileToDownload.download(request, "file2.txt");
            }
            catch (error:Error)
            {
                trace("Unable to download file.");
            }
        }
    }
}
```

O código a seguir demonstra o script ColdFusion, download.cfm, que baixa um de dois arquivos do servidor, dependendo do valor de uma variável de URL:

```
<cfparam name="URL.id" default="1" />
<cfswitch expression="#URL.id#">
    <cfcase value="2">
        <cfcontent type="text/plain" file="#ExpandPath('two.txt')#" deletefile="No" />
    </cfcase>
    <cfdefaultcase>
        <cfcontent type="text/plain" file="#ExpandPath('one.txt')#" deletefile="No" />
    </cfdefaultcase>
</cfswitch>
```

classe FileReferenceList

A classe FileReferenceList permite que o usuário selecione um ou mais arquivos a serem carregados para um script do lado do servidor. O upload de arquivo é manipulado pelo método `FileReference.upload()` que deve ser chamado em cada arquivo selecionado pelo usuário.

O código a seguir cria dois objetos FileFilter (`imageFilter` e `textFilter`) e transmite-os em uma matriz para o método `FileReferenceList.browse()`. Isso faz com que a caixa de diálogo de arquivo do sistema operacional exiba dois filtros possíveis para tipos de arquivos.

```
var imageFilter:FileFilter = new FileFilter("Image Files (*.jpg, *.jpeg, *.gif, *.png)",
"*.jpg; *.jpeg; *.gif; *.png");
var textFilter:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)", "*.txt; *.rtf");
var fileRefList:FileReferenceList = new FileReferenceList();
try
{
    var success:Boolean = fileRefList.browse(new Array(imageFilter, textFilter));
}
catch (error:Error)
{
    trace("Unable to browse for files.");
}
```

Permitir que o usuário selecione e carregue um ou mais arquivos usando a classe FileReferenceList é o mesmo que usar `FileReference.browse()` para selecionar arquivos, embora o FileReferenceList permita selecionar mais de um arquivo. O upload de vários arquivos requer que você carregue cada um dos arquivos selecionados usando `FileReference.upload()`, conforme mostrado no código a seguir:

```
var fileRefList:FileReferenceList = new FileReferenceList();
fileRefList.addEventListener(Event.SELECT, selectHandler);
fileRefList.browse();

function selectHandler(event:Event):void
{
    var request:URLRequest = new URLRequest("http://www.[yourdomain].com/fileUploadScript.cfm");
    var file:FileReference;
    var files:FileReferenceList = FileReferenceList(event.target);
    var selectedFileArray:Array = files.fileList;
    for (var i:uint = 0; i < selectedFileArray.length; i++)
    {
        file = FileReference(selectedFileArray[i]);
        file.addEventListener(Event.COMPLETE, completeHandler);
        try
        {
            file.upload(request);
        }
        catch (error:Error)
        {
            trace("Unable to upload files.");
        }
    }
}
function completeHandler(event:Event):void
{
    trace("uploaded");
}
```

Como o evento `Event.COMPLETE` é adicionado a cada objeto FileReference individual na matriz, o Flash Player chama o método `completeHandler()` quando o upload de cada arquivo individual é concluído.

Exemplo: Criação de um cliente Telnet

O exemplo de Telnet demonstra técnicas de conexão com um servidor remoto e transmissão de dados usando a classe Socket. O exemplo demonstra as seguintes técnicas:

- Criação de um cliente Telnet personalizado usando a classe Socket
- Envio de texto ao servidor remoto usando um objeto ByteArray
- Tratamento de dados recebidos de um servidor remoto

Para obter os arquivos do aplicativo deste exemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_br. Os arquivos do aplicativo Telnet podem ser encontrados na pasta Amostras/Telnet. O aplicativo consiste nos seguintes arquivos:

Arquivo	Descrição
TelnetSocket.mxml	O arquivo do aplicativo principal que consiste na interface do usuário MXML.
com/example/programmingas3/Telnet/Telnet.as	Fornece a funcionalidade de cliente Telnet para o aplicativo, como conectar-se a um servidor remoto e enviar, receber e exibir dados.

Visão geral do aplicativo de soquete Telnet

O arquivo TelnetSocket.mxml principal é responsável por criar a interface do usuário (UI) para o aplicativo inteiro.

Além da UI, esse arquivo também define dois métodos, `login()` e `sendCommand()`, para conectar o usuário ao servidor especificado.

O código a seguir lista o ActionScript no arquivo principal do aplicativo:

```
import com.example.programmingas3.socket.Telnet;

private var telnetClient:Telnet;
private function connect():void
{
    telnetClient = new Telnet(serverName.text, int(portNumber.text), output);
    console.title = "Connecting to " + serverName.text + ":" + portNumber.text;
    console.enabled = true;
}
private function sendCommand():void
{
    var ba:ByteArray = new ByteArray();
    ba.writeMultiByte(command.text + "\n", "UTF-8");
    telnetClient.writeBytesToSocket(ba);
    command.text = "";
}
```

A primeira linha de código importa a classe Telnet do pacote com.example.programmingas.socket personalizado. A segunda linha de código declara uma ocorrência da classe Telnet, `telnetClient`, que será inicializada posteriormente pelo método `connect()`. Em seguida, o método `connect()` é declarado e inicializa a variável `telnetClient` declarada anteriormente. Esse método transmite o nome do servidor telnet especificado pelo usuário, a porta do servidor telnet e uma referência a um componente TextArea na lista de exibição que é usada para exibir as respostas de texto do servidor de soquete. As duas linhas finais do método `connect()` definem a propriedade `title` do Painel e ativam o componente Painel, o que permite que o usuário envie dados ao servidor remoto. O método final no arquivo principal do aplicativo, `sendCommand()`, é usado para enviar os comandos do usuário ao servidor remoto como um objeto ByteArray.

Visão geral da classe Telnet

A classe Telnet é responsável por conectar-se ao servidor Telnet remoto e enviar/receber dados.

A classe Telnet declara as seguintes variáveis particulares:

```
private var serverURL:String;
private var portNumber:int;
private var socket:Socket;
private var ta:TextArea;
private var state:int = 0;
```

A primeira variável, `serverURL`, contém o endereço do servidor especificado pelo usuário ao qual conectar-se.

A segunda variável, `portNumber`, é o número da porta na qual o servidor Telnet está em execução atualmente. Por padrão, o serviço Telnet é executado na porta 23.

A terceira variável, `socket`, é uma ocorrência de Socket que tenta conectar-se ao servidor definido pelas variáveis `serverURL` e `portNumber`.

A quarta variável, `ta`, é uma referência a uma ocorrência do componente TextArea no Palco. Esse componente é usado para exibir respostas do servidor Telnet remoto ou qualquer mensagem de erro possível.

A variável final, `state`, é um valor numérico usado para determinar quais opções são suportadas pelo cliente Telnet.

Conforme visto anteriormente, a função do construtor da classe Telnet é chamada pelo método `connect()` no arquivo principal do aplicativo.

O construtor Telnet utiliza três parâmetros: `server`, `port` e `output`. Os parâmetros `server` e `port` especificam o nome do servidor e o número da porta em que o servidor Telnet está executando. O parâmetro final, `output`, é uma referência a uma ocorrência do componente TextArea no Palco onde a saída do servidor será exibida aos usuários.

```
public function Telnet(server:String, port:int, output:TextArea)
{
    serverURL = server;
    portNumber = port;
    ta = output;
    socket = new Socket();
    socket.addEventListener(Event.CONNECT, connectHandler);
    socket.addEventListener(Event.CLOSE, closeHandler);
    socket.addEventListener(ErrorEvent.ERROR, errorHandler);
    socket.addEventListener(IOErrorEvent.IO_ERROR, ioErrorHandler);
    socket.addEventListener(ProgressEvent.SOCKET_DATA, dataHandler);
    Security.loadPolicyFile("http://" + serverURL + "/crossdomain.xml");
    try
    {
        msg("Trying to connect to " + serverURL + ":" + portNumber + "\n");
        socket.connect(serverURL, portNumber);
    }
    catch (error:Error)
    {
        msg(error.message + "\n");
        socket.close();
    }
}
```

Gravação de dados em um soquete

Para gravar dados em uma conexão de soquete, chame qualquer um dos métodos de gravação na classe Socket (como `writeBoolean()`, `writeByte()`, `writeBytes()` ou `writeDouble()`) e libere os dados no buffer de saída usando o método `flush()`. No servidor Telnet, os dados são gravados na conexão de soquete usando o método `writeBytes()` que utiliza a matriz de bytes como um parâmetro e envia-a ao buffer de saída. O método `writeBytesToSocket()` é o seguinte:

```
public function writeBytesToSocket(ba:ByteArray):void
{
    socket.writeBytes(ba);
    socket.flush();
}
```

Esse método é chamado pelo método `sendCommand()` do arquivo principal do aplicativo.

Exibição de mensagens do servidor de soquete

Sempre que uma mensagem é recebida do servidor de soquete, ou ocorre um evento, o método `msg()` personalizado é chamado. Esse método anexa uma string ao TextArea no Palco e chama um método `setScroll()` personalizado, o que faz com que o componente TextArea role para a parte mais inferior. O método `msg()` é o seguinte:

```
private function msg(value:String):void
{
    ta.text += value;
    setScroll();
}
```

Se você não rolou automaticamente o conteúdo do componente TextArea, os usuários precisarão arrastar manualmente as barras de rolagem na área de texto para ver a resposta mais recente do servidor.

Rolagem de um componente TextArea

O método `setScroll()` contém uma única linha do ActionScript que rola o conteúdo do componente TextArea verticalmente de forma que o usuário possa ver a última linha do texto retornado. O seguinte snippet mostra o método `setScroll()`:

```
public function setScroll():void
{
    ta.verticalScrollPosition = ta.maxVerticalScrollPosition;
}
```

Esse método define a propriedade `verticalScrollPosition`, que é o número da linha superior de caracteres exibida atualmente, e define-a como o valor da propriedade `maxVerticalScrollPosition`.

Exemplo: Upload e download de arquivos

O exemplo de FileIO demonstra técnicas para baixar e carregar arquivos no Flash Player. Estas técnicas são:

- Download de arquivos em um computador do usuário
- Upload de arquivos de um computador do usuário para um servidor
- Cancelamento de um download em andamento
- Cancelamento de um upload em andamento

Para obter os arquivos do aplicativo desta amostra, consulte www.adobe.com/go/learn_programmingAS3samples_flash_br. Os arquivos do aplicativo FileIO estão localizados na pasta Amostras/FileIO. O aplicativo consiste nos seguintes arquivos:

Arquivo	Descrição
FileIO.fla ou FileIO.mxml	O arquivo principal do aplicativo no Flash (FLA) ou no Flex (MXML).
com/example/programmingas3/fileio/FileDownload.as	Uma classe que inclui métodos para baixar arquivos de um servidor.
com/example/programmingas3/fileio/FileUpload.as	Uma classe que inclui métodos para carregar arquivos em um servidor.

Visão geral do aplicativo FileIO

O aplicativo FileIO contém a interface do usuário que permite que um usuário carregue ou baixe arquivos usando o Flash Player. O aplicativo define primeiro um par de componentes personalizados, FileUpload e FileDownload, que podem ser encontrados no pacote com.example.programmingas3.fileio. Depois que cada componente personalizado despacha seu evento `contentComplete`, o método `init()` do componente é chamado e transmite referências a uma ocorrência do componente ProgressBar e Button, o que permite que os usuários vejam o andamento do upload ou do download do arquivo ou cancelem a transferência de arquivo em andamento.

O código relevante do arquivo FileIO.mxml é o seguinte (observe que na versão do Flash, o arquivo FLA contém componentes colocados na palco, cujos nomes correspondem ao nomes dos componentes Flex descritos nesta etapa):

```
<example:FileUpload id="fileUpload" creationComplete="fileUpload.init(uploadProgress,
cancelUpload);" />
<example:FileDownload id="fileDownload"
creationComplete="fileDownload.init(downloadProgress, cancelDownload);" />
```

O código a seguir mostra o painel Upload File que contém uma barra de progresso e dois botões. O primeiro botão, `startUpload`, chama o método `FileUpload.startUpload()`, que chama o método `FileReference.browse()`. O trecho a seguir mostra o código para o painel Upload File:

```
<mx:Panel title="Upload File" paddingTop="10" paddingBottom="10" paddingLeft="10"
paddingRight="10">
    <mx:ProgressBar id="uploadProgress" label="" mode="manual" />
    <mx:ControlBar horizontalAlign="right">
        <mx:Button id="startUpload" label="Upload..." click="fileUpload.startUpload();" />
        <mx:Button id="cancelUpload" label="Cancel" click="fileUpload.cancelUpload();"
enabled="false" />
    </mx:ControlBar>
</mx:Panel>
```

Esse código coloca uma ocorrência do componente ProgressBar e duas ocorrências do botão do componente Button no Palco. Quando o usuário clica no botão Upload (`startUpload`), é iniciada uma caixa de diálogo do sistema operacional que permite que o usuário selecione um arquivo a ser carregado em um servidor remoto. O outro botão, `cancelUpload`, por padrão, está desativado, mas quando um usuário inicia um upload de arquivo, o botão é ativado e permite que o usuário anule a transferência do arquivo a qualquer momento.

O código para o painel Download File é o seguinte:

```
<mx:Panel title="Download File" paddingTop="10" paddingBottom="10" paddingLeft="10"
paddingRight="10">
    <mx:ProgressBar id="downloadProgress" label="" mode="manual" />
    <mx:ControlBar horizontalAlign="right">
        <mx:Button id="startDownload" label="Download..."
click="fileDownload.startDownload();" />
        <mx:Button id="cancelDownload" label="Cancel" click="fileDownload.cancelDownload();"
enabled="false" />
    </mx:ControlBar>
</mx:Panel>
```

Esse código é muito semelhante ao código de upload de arquivo. Quando o usuário clica no botão Download, (`startDownload`), o método `FileDownload.startDownload()` é chamado, o que inicia o download do arquivo especificado na variável `FileDownload.DOWNLOAD_URL`. Conforme o arquivo é baixado, a barra de progresso é atualizada, mostrando a porcentagem do arquivo que foi baixada. O usuário pode cancelar o download a qualquer momento clicando no botão `cancelDownload` que pára imediatamente o download do arquivo em andamento.

Download de arquivos de um servidor remoto

O download de arquivos de um servidor remoto é manipulado pela classe flash.net.FileReference e a classe com.example.programmingas3.fileio.FileDownload personalizada. Quando o usuário clica no botão Download, o Flash Player começa a baixar o arquivo especificado na variável `DOWNLOAD_URL` da classe FileDownload.

A classe FileDownload começa definindo quatro variáveis dentro do pacote com.example.programmingas3.fileio, conforme mostrado no código a seguir:

```
/**
 * Hard-code the URL of file to download to user's computer.
 */
private const DOWNLOAD_URL:String = "http://www.yourdomain.com/file_to_download.zip";

/**
 * Create a FileReference instance to handle the file download.
 */
private var fr:FileReference;

/**
 * Define reference to the download ProgressBar component.
 */
private var pb:ProgressBar;

/**
 * Define reference to the "Cancel" button which will immediately stop
 * the current download in progress.
 */
private var btn:Button;
```

A primeira variável, `DOWNLOAD_URL`, contém o caminho para o arquivo, que é baixado no computador do usuário quando ele clica no botão Download no arquivo principal do aplicativo.

A segunda variável, `fr`, é um objeto FileReference que é inicializado dentro do método `FileDownload.init()` e manipula o download do arquivo remoto no computador do usuário.

As duas últimas variáveis, `pb` e `btn`, contêm referências às ocorrências dos componentes ProgressBar e Button no Palco, inicializadas pelo método `FileDownload.init()`.

Inicialização do componente FileDownload

O componente FileDownload é inicializado chamando o método `init()` na classe FileDownload. Esse método utiliza dois parâmetros, `pb` e `btn`, que são ocorrências dos componentes ProgressBar e Button, respectivamente.

O código para o método `init()` é o seguinte:

```
/**
 * Set references to the components, and add listeners for the OPEN,
 * PROGRESS, and COMPLETE events.
 */
public function init(pb:ProgressBar, btn:Button):void
{
    // Set up the references to the progress bar and cancel button,
    // which are passed from the calling script.
    this.pb = pb;
    this.btn = btn;

    fr = new FileReference();
    fr.addEventListener(Event.OPEN, openHandler);
    fr.addEventListener(ProgressEvent.PROGRESS, progressHandler);
    fr.addEventListener(Event.COMPLETE, completeHandler);
}
```

Início do download do arquivo

Quando o usuário clica na ocorrência do componente Download Button no Palco, o método `startDownload()` inicia o processo de download do arquivo. O seguinte trecho mostra o método `startDownload()`:

```
/**
 * Begin downloading the file specified in the DOWNLOAD_URL constant.
 */
public function startDownload():void
{
    var request:URLRequest = new URLRequest();
    request.url = DOWNLOAD_URL;
    fr.download(request);
}
```

Primeiro, o método `startDownload()` cria um novo objeto URLRequest e define a URL de destino como o valor especificado pela variável `DOWNLOAD_URL`. Em seguida, o método `FileReference.download()` é chamado e o objeto URLRequest recém-criado é transmitido como um parâmetro. Isso faz com que o sistema operacional exiba uma caixa de diálogo no computador do usuário solicitando que ele selecione um local para salvar o documento solicitado. Depois que o usuário seleciona um local, o evento `open` (`Event.OPEN`) é despachado e o método `openHandler()` é chamado.

O método `openHandler()` define o formato do texto para a propriedade `label` do componente ProgressBar e ativa o botão Cancelar, o que permite que o usuário pare imediatamente o download em andamento. O método `openHandler()` é o seguinte:

```
/**
 * When the OPEN event has dispatched, change the progress bar's label
 * and enable the "Cancel" button, which allows the user to abort the
 * download operation.
 */
private function openHandler(event:Event):void
{
    pb.label = "DOWNLOADING %3%%";
    btn.enabled = true;
}
```

Monitoramento do andamento do download de um arquivo

Conforme um arquivo é baixado de um servidor remoto para o computador do usuário, o evento `progress` (`ProgressEvent.PROGRESS`) é despachado em intervalos regulares. Sempre que o evento `progress` é despachado, o método `progressHandler()` é chamado e a ocorrência do componente ProgressBar no Palco é atualizada. O código para o método `progressHandler()` é o seguinte:

```
/**
 * While the file is downloading, update the progress bar's status.
 */
private function progressHandler(event:ProgressEvent):void
{
    pb.setProgress(event.bytesLoaded, event.bytesTotal);
}
```

O evento de progresso contém duas propriedades, `bytesLoaded` e `bytesTotal`, que são usadas para atualizar o componente ProgressBar no Palco. Isso dá ao usuário uma idéia da quantidade de download do arquivo que já foi concluída e da quantidade que resta. O usuário pode anular a transferência do arquivo a qualquer momento clicando no botão Cancelar abaixo da barra de progresso.

Se o arquivo for baixado com êxito, o evento `complete` (`Event.COMPLETE`) chamará o método `completeHandler()`, que notificará o usuário que o download do arquivo foi concluído e desativará o botão Cancelar. O código para o método `completeHandler()` é o seguinte:

```
/**
 * Once the download has completed, change the progress bar's label one
 * last time and disable the "Cancel" button since the download is
 * already completed.
 */
private function completeHandler(event:Event):void
{
    pb.label = "DOWNLOAD COMPLETE";
    btn.enabled = false;
}
```

Cancelamento do download de um arquivo

Um usuário pode anular a transferência de um arquivo e parar o download de qualquer byte adicional a qualquer momento, clicando no botão Cancelar no Palco. O trecho a seguir mostra o código para cancelar um download:

```
/**
 * Cancel the current file download.
 */
public function cancelDownload():void
{
    fr.cancel();
    pb.label = "DOWNLOAD CANCELLED";
    btn.enabled = false;
}
```

Primeiro, o código pára a transferência do arquivo imediatamente, impedindo o download de qualquer dado adicional. Em seguida, a propriedade de rótulo da barra de progresso é atualizada para notificar o usuário que o download foi cancelado com êxito. Finalmente, o botão Cancelar é desativado, o que evita que o usuário clique no botão novamente até que o download do arquivo seja tentado novamente.

Upload de arquivos em um servidor remoto

O processo de upload de arquivo é muito semelhante ao processo de download de arquivo. A classe FileUpload declara as mesmas quatro variáveis, conforme mostrado no código a seguir:

```
private const UPLOAD_URL:String = "http://www.yourdomain.com/your_upload_script.cfm";
private var fr:FileReference;
private var pb:ProgressBar;
private var btn:Button;
```

Ao contrário da variável `FileDownload.DOWNLOAD_URL`, a variável `UPLOAD_URL` contém a URL para o script do lado do servidor que carrega o arquivo do computador do usuário. As três variáveis restantes comportam-se da mesma maneira que suas contrapartes na classe FileDownload.

Inicialização do componente FileUpload

O componente FileUpload contém um método `init()` que é chamado do aplicativo principal. Esse método utiliza dois parâmetros, `pb` e `btn`, que são referências a uma ocorrência dos componentes ProgressBar e Button no Palco. Em seguida, o método `init()` inicializa o objeto FileReference definido anteriormente na classe FileUpload. Finalmente, o método atribui quatro ouvintes de eventos ao objeto FileReference. O código para o método `init()` é o seguinte:

```
public function init(pb:ProgressBar, btn:Button):void
{
    this.pb = pb;
    this.btn = btn;

    fr = new FileReference();
    fr.addEventListener(Event.SELECT, selectHandler);
    fr.addEventListener(Event.OPEN, openHandler);
    fr.addEventListener(ProgressEvent.PROGRESS, progressHandler);
    fr.addEventListener(Event.COMPLETE, completeHandler);
}
```

Início de um upload de arquivo

O upload do arquivo é iniciado quando o usuário clica no botão Upload no Palco, o que chama o método `FileUpload.startUpload()`. Esse método chama o método `browse()` da classe FileReference que faz com que o sistema operacional exiba uma caixa de diálogo do sistema solicitando que o usuário selecione um arquivo a ser carregado no servidor remoto. O trecho a seguir mostra o código para o método `startUpload()`:

```
public function startUpload():void
{
    fr.browse();
}
```

Depois que o usuário seleciona um arquivo para upload, o evento `select` (`Event.SELECT`) é despachado, fazendo com que o método `selectHandler()` seja chamado. O método `selectHandler()` cria um novo objeto URLRequest e define a propriedade `URLRequest.url` como o valor da constante `UPLOAD_URL` definida anteriormente no código. Finalmente, o objeto FileReference carrega o arquivo selecionado no script do lado do servidor especificado. O código para o método `selectHandler()` é o seguinte:

```
private function selectHandler(event:Event):void
{
    var request:URLRequest = new URLRequest();
    request.url = UPLOAD_URL;
    fr.upload(request);
}
```

O código restante na classe FileUpload é o mesmo que o código definido na classe FileDownload. Se um usuário desejar encerrar o upload a qualquer momento, poderá clicar no botão Cancelar, o que define o rótulo na barra de progresso e pára a transferência do arquivo imediatamente. A barra de progresso é atualizada sempre que o evento `progress` (`ProgressEvent.PROGRESS`) é despachado. De modo semelhante, depois que o upload foi concluído, a barra de progresso é atualizada para notificar o usuário que o arquivo foi carregado com êxito. Em seguida, o botão Cancelar é desativado até que o usuário inicie uma nova transferência de arquivo.

Capítulo 28: Ambiente do sistema cliente

Este capítulo explica como interagir com o sistema do usuário. Ele mostra como determinar quais recursos são suportados e como criar arquivos SWF multilíngües usando o IME (editor de método de entrada) instalado do usuário, se disponível. Ele também mostra usuários típicos para domínios de aplicativo.

Noções básicas do ambiente do sistema cliente

Introdução ao ambiente do sistema cliente

Conforme você cria aplicativos ActionScript mais avançados, talvez ache necessário conhecer detalhes — e funções de acesso — dos sistemas operacionais dos usuários. O ambiente do sistema cliente é uma coleção de classes no pacote flash.system que permitem acessar funcionalidades em nível do sistema, como as seguintes:

- Determinar em qual aplicativo e domínio de segurança um SWF está em execução.
- Determinar as capacidades da ocorrência do Flash® Player ou do Adobe® AIR™ do usuário, como o tamanho da tela (resolução), e se determinadas funcionalidades estão disponíveis, como áudio mp3.
- Criar sites multilíngües usando o IME.
- Interagir com o contêiner do Flash Player (que pode ser uma página HTML ou um aplicativo de contêiner) ou com um contêiner do AIR.
- Salvar informações na área de transferência do usuário.

O pacote flash.system também inclui as classes IMEConversionMode e SecurityPanel. Essas classes contêm constantes estáticas que são usadas com as classes IME e Security, respectivamente.

Tarefas comuns do ambiente do sistema cliente.

As seguintes tarefas comuns para trabalho com o sistema cliente usando o ActionScript são descritas neste capítulo:

- Determinar quanto de memória o aplicativo está usando.
- Copiar texto na área de transferência do usuário.
- Determinar capacidades do computador do usuário, como:
  - Resolução, cor, DPI e proporção de pixels da tela
  - Sistema operacional
  - Suporte para fluxo de áudio, fluxo de vídeo e reprodução de mp3
  - Se o Flash Player instalado é uma versão de depurador
- Trabalhar com domínios de aplicativo:
  - Definir um domínio de aplicativo.
  - Separar código de arquivos SWF em domínios de aplicativo.
- Trabalhar com um IME no aplicativo:
  - Determinar se o IME está instalado.
  - Determinar e definir o modo de conversão do IME.

- Desativar o IME para campos de texto.
- Detectar quando ocorre a conversão do IME.

Conceitos e termos importantes

A lista de referência a seguir contém termos importantes usados neste capítulo:

- Sistema operacional: O programa principal executado em um computador, dentro do qual todos os outros aplicativos são executados, como o Microsoft Windows, o Mac OS X ou o Linux®.
- Área de transferência: O contêiner do sistema operacional onde manter texto ou itens que são copiados ou recortados e do qual os itens são colados nos aplicativos.
- Domínio do aplicativo: Um mecanismo para separação de classes usadas em diferentes arquivos SWF, de forma que os arquivos SWF incluam diferentes classes com o mesmo nome, as classes não são substituídas umas pelas outras.
- IME (editor de método de entrada): Um programa (ou ferramenta do sistema operacional) usado para inserir caracteres ou símbolos complexos usando um teclado padrão.
- Sistema cliente: Em termos de programação, um *cliente* é a parte de um aplicativo (ou todo o aplicativo) que é executado no computador individual e é usada por um único usuário. O *sistema cliente* é o sistema operacional subjacente no computador do usuário.

Teste dos exemplos do capítulo

Talvez você queira testar algumas das listagens de código de exemplo por si próprio, durante a leitura deste capítulo. Todas as listagens de código deste capítulo incluem a chamada da função `trace()` apropriada para anotar os valores que estão sendo testados. Para testar as listagens de código deste capítulo:

1. Crie um documento do Flash vazio.
2. Selecione um quadro-chave na linha de tempo.
3. Abra o painel Ações e copie a listagem de código no painel Script.
4. Execute o programa usando Controlar > Testar filme.

   Você verá os resultados das funções `trace()` da listagem de código no painel Saída.

Algumas das últimas listagens de código são mais complexas e são escritas como uma classe. Para testar esses exemplos:

1. Crie um documento Flash vazio e salve-o no seu computador.
2. Crie e salve um novo arquivo do ActionScript no mesmo diretório do documento Flash. O nome do arquivo deve corresponder ao nome da classe na listagem de código. Por exemplo, se a listagem de código definir uma classe denominada SystemTest, use o nome SystemTest.as para salvar o arquivo ActionScript.
3. Copie a listagem de código no arquivo do ActionScript e salve o arquivo.
4. No documento Flash, clique em uma parte em branco do Palco ou do espaço de trabalho para ativar o Inspetor de propriedades do documento.
5. No Inspetor de propriedades, no campo Classe do documento, digite o nome da classe ActionScript que você copiou do texto.
6. Execute o programa usando Controlar > Testar filme.

   Você verá os resultados do exemplo no painel Saída.

Técnicas para testar listagens de código de exemplo são descritas em mais detalhes em "Teste de listagens de código de exemplo dos capítulos" na página 36.

Uso da classe System

A classe System contém métodos e propriedades que permitem interagir com o sistema operacional do usuário e recuperar o uso da memória atual para o Flash Player ou o AIR. Os métodos e propriedades da classe System também permitem ouvir eventos `imeComposition`, instruir o Flash Player ou o AIR a carregar arquivos de texto externos usando a página de código atual do usuário ou carregá-las como Unicode ou definir o conteúdo da área de transferência do usuário.

Obtenção de dados sobre o sistema do usuário em tempo de execução

Com a verificação da propriedade `System.totalMemory`, é possível determinar a quantidade de memória (em bytes) que o Flash Player ou o AIR estão usando no momento. Essa propriedade permite monitorar o uso de memória do monitor e otimizar os aplicativos com base em como o nível de memória é alterado. Por exemplo, se um efeito visual específico provocar um grande aumento no uso de memória, você poderá desejar considerar modificar ou eliminar o efeito completamente.

A propriedade `System.ime` é uma referência ao IME (Editor de método de entrada) instalado. Essa propriedade permite ouvir eventos `imeComposition` (`flash.events.IMEEvent.IME_COMPOSITION`) usando o método `addEventListener()`.

A terceira propriedade na classe System é `useCodePage`. Quando `useCodePage` está definido como `true`, o Flash Player e o AIR usam a página de código tradicional do sistema operacional que está executando o player para carregar arquivos de texto externos. Se definir essa propriedade como `false`, você indicará ao Flash Player ou ao AIR para interpretar o arquivo externo como Unicode.

Se você definir `System.useCodePage` como `true`, lembre-se de que a página de código tradicional do sistema operacional que executa o player deve incluir os caracteres usados no arquivo de texto externo para que o texto seja exibido. Por exemplo, se você carregar um arquivo de texto externo com caracteres chineses, esses caracteres não poderão ser exibidos em um sistema que use a página de código do Windows em inglês, pois essa página de código não inclui caracteres chineses.

Para garantir que os usuários de todas as plataformas possam exibir arquivos de texto externos usados em arquivos SWF, você deve codificar todos os arquivos de texto externos como Unicode e deixar `System.useCodePage` definido como `false` como padrão. Dessa maneira, o Flash Player e o AIR interpretarão o texto como Unicode.

Salvamento de texto na área de transferência

A classe System inclui um método chamado `setClipboard()` que permite que o Flash Player e o AIR definam o conteúdo da área de transferência do usuário com uma string especificada. Por motivos de segurança, não existe nenhum método `Security.getClipboard()`, pois esse método pode potencialmente permitir que sites mal-intencionados acessem os dados copiados recentemente na área de transferência do usuário.

O código a seguir ilustra como uma mensagem de erro pode ser copiada na área de transferência do usuário quando ocorre um erro de segurança. A mensagem de erro poderá ser útil se o usuário desejar relatar um bug potencial com um aplicativo.

```
private function securityErrorHandler(event:SecurityErrorEvent):void
{
    var errorString:String = "[" + event.type + "] " + event.text;
    trace(errorString);
    System.setClipboard(errorString);
}
```

Uso da classe Capabilities

A classe Capabilities permite que os desenvolvedores determinem o ambiente no qual um arquivo SWF está em execução. Com o uso de várias propriedades da classe Capabilities, é possível descobrir a resolução do sistema do usuário, se o sistema do usuário oferece suporte a software de acessibilidade, o idioma do sistema operacional do usuário, bem como a versão do Flash Player ou do AIR instalada atualmente.

Verificando as propriedade da classe Capabilities, é possível personalizar o aplicativo para trabalhar melhor com o ambiente específico do usuário. Por exemplo, verificando as propriedades `Capabilities.screenResolutionX` e `Capabilities.screenResolutionY`, é possível determinar a resolução de vídeo que o sistema do usuário está usando e decidir qual tamanho de vídeo pode ser mais apropriado. Ou é possível verificar a propriedade `Capabilities.hasMP3` para verificar se o sistema do usuário oferece suporte à reprodução de mp3 antes de tentar carregar um arquivo mp3 externo.

O código a seguir usa uma expressão regular para analisar a versão do Flash Player que está sendo usada pelo cliente:

```
var versionString:String = Capabilities.version;
var pattern:RegExp = /^(\w*) (\d*),(\d*),(\d*),(\d*)$/;
var result:Object = pattern.exec(versionString);
if (result != null)
{
    trace("input: " + result.input);
    trace("platform: " + result[1]);
    trace("majorVersion: " + result[2]);
    trace("minorVersion: " + result[3]);
    trace("buildNumber: " + result[4]);
    trace("internalBuildNumber: " + result[5]);
}
else
{
    trace("Unable to match RegExp.");
}
```

Para enviar as capacidades do sistema do usuário para um script do lado do servidor de forma que as informações sejam armazenadas em um banco de dados, é possível usar o seguinte código ActionScript:

```
var url:String = "log_visitor.cfm";
var request:URLRequest = new URLRequest(url);
request.method = URLRequestMethod.POST;
request.data = new URLVariables(Capabilities.serverString);
var loader:URLLoader = new URLLoader(request);
```

Uso da classe ApplicationDomain

O objetivo da classe ApplicationDomain é armazenar uma tabela de definições do ActionScript 3.0. Todo código em um arquivo SWF é definido para estar presente em um domínio de aplicativo. Domínios de aplicativo são usados para particionar classes que estão no mesmo domínio de segurança. Eles permitem que haja várias definições da mesma classe e permitem que filhos reutilizem definições de pai.

É possível usar domínios de aplicativo carregando um arquivo SWF externo escrito no ActionScript 3.0 usando a API da classe Loader. (Observe que não é possível usar domínios de aplicativo ao carregar uma imagem ou arquivo SWF no ActionScript 1.0 ou no ActionScript 2.0.) Todas as definições do ActionScript 3.0 contidas na classe carregada são armazenadas no domínio do aplicativo. Ao carregar o arquivo SWF, é possível especificar que o arquivo seja incluído no mesmo domínio do aplicativo que o objeto Loader, definindo o parâmetro `applicationDomain` do objeto LoaderContext como `ApplicationDomain.currentDomain`. Colocando o arquivo SWF carregado no mesmo domínio de aplicativo, é possível acessar suas classes diretamente. Isso pode ser útil se você estiver carregando um arquivo SWF que contém mídia incorporada que você pode acessar por meio de seus nomes de classes associadas ou para acessar os métodos do arquivo SWF carregado, conforme mostrado no exemplo a seguir:

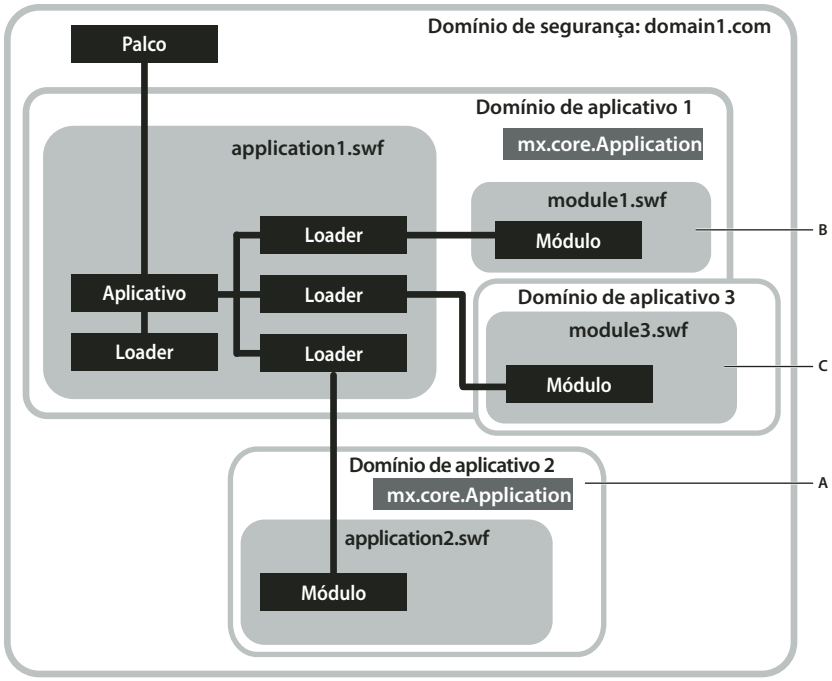
```
package
{
    import flash.display.Loader;
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.URLRequest;
    import flash.system.ApplicationDomain;
    import flash.system.LoaderContext;

    public class ApplicationDomainExample extends Sprite
    {
        private var ldr:Loader;
        public function ApplicationDomainExample()
        {
            ldr = new Loader();
            var req:URLRequest = new URLRequest("Greeter.swf");
            var ldrContext:LoaderContext = new LoaderContext(false,
ApplicationDomain.currentDomain);
            ldr.contentLoaderInfo.addEventListener(Event.COMPLETE, completeHandler);
            ldr.load(req, ldrContext);
        }
        private function completeHandler(event:Event):void
        {
            ApplicationDomain.currentDomain.getDefinition("Greeter");
            var myGreeter:Greeter = Greeter(event.target.content);
            var message:String = myGreeter.welcome("Tommy");
            trace(message); // Hello, Tommy
        }
    }
}
```

Outras coisas que devem ser lembradas ao trabalhar com domínios de aplicativos incluem:

- Todo código em um arquivo SWF é definido para estar presente em um domínio de aplicativo. O *domínio atual* é onde seu aplicativo principal é executado. O *domínio do sistema* contém todos os domínios de aplicativos, incluindo o domínio atual, o que significa que ele contém todas as classes do Flash Player.

- Todos os domínios de aplicativos, exceto o domínio do sistema, têm um domínio-pai associado. O domínio-pai do domínio do aplicativo principal é o domínio do sistema. As classes carregadas só são definidas quando o pai ainda não as definiu. Você não pode substituir uma definição de classe loaded por uma definição mais recente.

O diagrama a seguir mostra um aplicativo que carrega conteúdo de vários arquivos SWF dentro de um único domínio, domain1.com. Dependendo do conteúdo carregado, diferentes domínios de aplicativos podem ser usados. O texto a seguir descreve a lógica usada para definir o domínio apropriado do aplicativo para cada arquivo SWF no aplicativo.



***A.** Uso A **B.** Uso B **C.** Uso C*

O arquivo do aplicativo principal é application1.swf. Ele contém objetos Loader que carregam conteúdo de outros arquivos SWF. Neste cenário, o domínio atual é o Domínio do aplicativo 1. Uso A, uso B e uso C ilustram diferentes técnicas para configurar o domínio do aplicativo apropriado para cada arquivo SWF em um aplicativo.

Uso A Particionar o arquivo SWF filho criando um filho do domínio do sistema. No diagrama, o Domínio do aplicativo 2 é criado como um filho do domínio do sistema. O arquivo application2.swf é carregado no Domínio do aplicativo 2 e suas definições de classe são portanto particionadas a partir das classes definidas no application1.swf.

Um dos usos dessa técnica é fazer com que um aplicativo antigo carregue uma versão mais nova do mesmo aplicativo sem conflito. Não há conflito porque, embora os mesmos nomes de classes sejam usados, elas são particionadas em diferentes domínios de aplicativo.

O código a seguir cria um domínio de aplicativo que é filho do domínio do sistema, e começa carregando um SWF usando esse domínio de aplicativo:

```
var appDomainA:ApplicationDomain = new ApplicationDomain();

var contextA:LoaderContext = new LoaderContext(false, appDomainA);
var loaderA:Loader = new Loader();
loaderA.load(new URLRequest("application2.swf"), contextA);
```

Uso B: Adicionar novas definições de classes às definições de classes atuais. O domínio do aplicativo do module1.swf é definido como o domínio atual (Domínio do aplicativo 1). Isso permite adicionar o conjunto atual de aplicativos das

definições de classes com novas definições de classes. Isso pode ser usado para uma biblioteca de tempo de execução do aplicativo principal. O SWF carregado é tratado como uma RSL (biblioteca remota compartilhada). Use essa técnica para carregar RSLs por meio de um pré-carregador antes do início do aplicativo.

O código a seguir carrega um SWF definindo seu domínio de aplicativo para o domínio atual:

```
var appDomainB:ApplicationDomain = ApplicationDomain.currentDomain;

var contextB:LoaderContext = new LoaderContext(false, appDomainB);
var loaderB:Loader = new Loader();
loaderB.load(new URLRequest("module1.swf"), contextB);
```

Uso C: Usar as definições de classes pai criando um novo domínio filho do domínio atual. O domínio do aplicativo do module3.swf é filho do domínio atual, e o filho usa as versões de todas as classes do pai. Um uso dessa técnica pode ser um módulo de um RIA (aplicativo de Internet avançado) de várias telas, carregado como um filho do aplicativo principal que usa os tipos do aplicativo principal. Se você puder garantir que todas as classes são sempre atualizadas para serem compatíveis com versões anteriores e que o aplicativo carregado é sempre mais novo do que as coisas que ele carrega, os filhos usarão as versões do pai. Ter um novo domínio de aplicativo também permite descarregar todas as definições de classe para coleta de lixo, se você puder garantir que referências ao SWF filho não continuam a existir.

Essa técnica permite que módulos carregados compartilhem os objetos singleton do carregador e membros da classe estática.

O código a seguir cria um novo domínio filho do domínio atual e inicia o carregamento de um SWF usando aquele domínio de aplicativo:

```
var appDomainC:ApplicationDomain = new ApplicationDomain(ApplicationDomain.currentDomain);

var contextC:LoaderContext = new LoaderContext(false, appDomainC);
var loaderC:Loader = new Loader();
loaderC.load(new URLRequest("module3.swf"), contextC);
```

Uso da classe IME

A classe IME permite manipular o IME do sistema operacional de dentro do Flash Player ou do Adobe AIR.

Usando o ActionScript, é possível determinar o seguinte:

- Se um IME está instalado no computador do usuário (`Capabilities.hasIME`)
- Se o IME está ativado ou desativado no computador do usuário (`IME.enabled`)
- O modo de conversão do IME atual está usando (`IME.conversionMode`)

É possível associar um campo de texto de entrada a um contexto de IME específico. Ao alternar entre campos de texto, também é possível alternar o IME entre Hiragana (japonês), números de largura total, números de meia largura, entrada direta e assim por diante.

Um IME permite que os usuários digitem caracteres de texto não-ASCII em idiomas com vários bytes, como chinês, japonês e coreano.

Para obter mais informações sobre como trabalhar com IMEs, consulte a documentação do sistema operacional no qual você está desenvolvendo o aplicativo. Para obter recursos adicionais, consulte os seguintes sites:

- http://www.msdn.microsoft.com/goglobal/
- http://developer.apple.com/documentation/

- http://www.java.sun.com/

Nota: *Se o IME não estiver ativo no computador do usuário, chamadas para métodos ou propriedades do IME, que não sejam* `Capabilities.hasIME`*, falharão. Depois de ativar manualmente o IME, chamadas subseqüentes do ActionScript para métodos e propriedades do IME funcionarão conforme o esperado. Por exemplo, se estiver usando um IME japonês, você deverá ativá-lo para poder chamar qualquer método ou propriedade do IME.*

Verificação da instalação e ativação do IME

Antes de chamar qualquer um dos métodos ou propriedades do IME, você deve sempre verificar se o computador do usuário tem um IME instalado e ativado. O código a seguir ilustra como verificar se o usuário tem um IME instalado e ativo antes de chamar qualquer método:

```
if (Capabilities.hasIME)
{
    if (IME.enabled)
    {
        trace("IME is installed and enabled.");
    }
    else
    {
        trace("IME is installed but not enabled. Please enable your IME and try again.");
    }
}
else
{
    trace("IME is not installed. Please install an IME and try again.");
}
```

O código anterior primeiro verifica se o usuário tem um IME instalado usando a propriedade `Capabilities.hasIME`. Se essa propriedade estiver definida como `true`, o código verificará se o IME do usuário está ativado usando a propriedade `IME.enabled`.

Determinação de qual modo de conversão do IME está ativado

Ao criar aplicativos multilíngüe, talvez seja necessário determinar qual modo de conversão está ativo para o usuário. O código a seguir demonstra como verificar se o usuário tem um IME instalado e, nesse caso, qual modo de conversão do IME está ativo no momento:

```
if (Capabilities.hasIME)
{
    switch (IME.conversionMode)
    {
        case IMEConversionMode.ALPHANUMERIC_FULL:
            tf.text = "Current conversion mode is alphanumeric (full-width).";
            break;
        case IMEConversionMode.ALPHANUMERIC_HALF:
            tf.text = "Current conversion mode is alphanumeric (half-width).";
            break;
        case IMEConversionMode.CHINESE:
            tf.text = "Current conversion mode is Chinese.";
            break;
        case IMEConversionMode.JAPANESE_HIRAGANA:
            tf.text = "Current conversion mode is Japananese Hiragana.";
            break;
        case IMEConversionMode.JAPANESE_KATAKANA_FULL:
            tf.text = "Current conversion mode is Japanese Katakana (full-width).";
            break;
        case IMEConversionMode.JAPANESE_KATAKANA_HALF:
            tf.text = "Current conversion mode is Japanese Katakana (half-width).";
            break;
        case IMEConversionMode.KOREAN:
            tf.text = "Current conversion mode is Korean.";
            break;
        default:
            tf.text = "Current conversion mode is " + IME.conversionMode + ".";
            break;
    }
}
else
{
    tf.text = "Please install an IME and try again.";
}
```

O código anterior primeiro verifica se o usuário tem um IME instalado. Em seguida, ele verifica qual modo de conversão o IME atual está usando, comparando a propriedade `IME.conversionMode` com cada uma das constantes na classe IMEConversionMode.

Configuração do modo de conversão do IME

Quando você altera o modo de conversão do IME do usuário, é necessário verificar se o código está inserido em um bloco `try..catch`, porque a configuração de um modo de conversão usando a propriedade `conversionMode` poderá emitir um erro, se o IME não puder definir o modo de conversão. O código a seguir demonstra como usar um bloco `try..catch` ao definir a propriedade `IME.conversionMode`:

```
var statusText:TextField = new TextField;
statusText.autoSize = TextFieldAutoSize.LEFT;
addChild(statusText);
if (Capabilities.hasIME)
{
    try
    {
        IME.enabled = true;
        IME.conversionMode = IMEConversionMode.KOREAN;
        statusText.text = "Conversion mode is " + IME.conversionMode + ".";
    }
    catch (error:Error)
    {
        statusText.text = "Unable to set conversion mode.\n" + error.message;
    }
}
```

O código anterior primeiro cria um campo de texto que é usado para exibir uma mensagem de status para o usuário. Em seguida, se o IME estiver instalado, o código ativará o IME e definirá o modo de conversão como coreano. Se o computador do usuário não tiver um IME coreano instalado, o Flash Player ou o AIR emitirá um erro que será capturado pelo bloco `try..catch`. O bloco `try..catch` exibe a mensagem de erro no campo de texto criado anteriormente.

Desativação do IME para determinados campos de texto

Em alguns casos, convém desativar o IME do usuário enquanto ele digita caracteres. Por exemplo, se você tiver um campo de texto que aceita apenas entrada numérica, é conveniente ativar o IME e diminuir a velocidade da entrada de dados.

O exemplo a seguir demonstra como é possível ouvir os eventos `FocusEvent.FOCUS_IN` e `FocusEvent.FOCUS_OUT` e desativar o IME do usuário de maneira correspondente:

```
var phoneTxt:TextField = new TextField();
var nameTxt:TextField = new TextField();

phoneTxt.type = TextFieldType.INPUT;
phoneTxt.addEventListener(FocusEvent.FOCUS_IN, focusInHandler);
phoneTxt.addEventListener(FocusEvent.FOCUS_OUT, focusOutHandler);
phoneTxt.restrict = "0-9";
phoneTxt.width = 100;
phoneTxt.height = 18;
phoneTxt.background = true;
phoneTxt.border = true;
addChild(phoneTxt);

nameField.type = TextFieldType.INPUT;
nameField.x = 120;
nameField.width = 100;
nameField.height = 18;
nameField.background = true;
nameField.border = true;
addChild(nameField);

function focusInHandler(event:FocusEvent):void
{
    if (Capabilities.hasIME)
    {
        IME.enabled = false;
    }
}
function focusOutHandler(event:FocusEvent):void
{
    if (Capabilities.hasIME)
    {
        IME.enabled = true;
    }
}
```

Esse exemplo cria dois campos de texto de entrada, `phoneTxt` e `nameTxt` e, em seguida, adiciona dois ouvintes de eventos ao campo de texto `phoneTxt`. Quando o usuário define o foco para o campo de texto `phoneTxt`um evento `FocusEvent.FOCUS_IN` é despachado e o IME é desativado. Quando o campo de texto `phoneTxt` perde o foco, o evento `FocusEvent.FOCUS_OUT` é despachado para reativar o IME.

Ouvir eventos de composição do IME

Os eventos de composição do IME são despachados quando uma string de composição está sendo definida. Por exemplo, se o usuário tiver o IME habilitado e ativo e digitar uma string em japonês, o evento `IMEEvent.IME_COMPOSITION` despachará assim que o usuário selecionar a string da composição. Para ouvir o evento `IMEEvent.IME_COMPOSITION`, é necessário adicionar um ouvinte de eventos à propriedade estática `ime` na classe System (`flash.system.System.ime.addEventListener(...)`), conforme mostrado no exemplo a seguir:

```
var inputTxt:TextField;
var outputTxt:TextField;

inputTxt = new TextField();
inputTxt.type = TextFieldType.INPUT;
inputTxt.width = 200;
inputTxt.height = 18;
inputTxt.border = true;
inputTxt.background = true;
addChild(inputTxt);

outputTxt = new TextField();
outputTxt.autoSize = TextFieldAutoSize.LEFT;
outputTxt.y = 20;
addChild(outputTxt);

if (Capabilities.hasIME)
{
    IME.enabled = true;
    try
    {
        IME.conversionMode = IMEConversionMode.JAPANESE_HIRAGANA;
    }
    catch (error:Error)
    {
        outputTxt.text = "Unable to change IME.";
    }
    System.ime.addEventListener(IMEEvent.IME_COMPOSITION, imeCompositionHandler);
}
else
{
    outputTxt.text = "Please install IME and try again.";
}

function imeCompositionHandler(event:IMEEvent):void
{
    outputTxt.text = "you typed: " + event.text;
}
```

O código anterior cria dois campos de texto e os adiciona à lista de exibição. O primeiro campo de texto, `inputTxt`, é um campo de texto de entrada que permite que o usuário digite texto japonês. O segundo campo de texto, `outputTxt`, é um campo de texto dinâmico que exibe mensagens de erro para o usuário ou ecoa a string em japonês que o usuário digita no campo de texto `inputTxt`.

Exemplo: Detecção de capacidades do sistema

O exemplo CapabilitiesExplorer demonstra como usar a classe flash.system.Capabilities para determinar quais recursos são suportados pela versão do usuário do Flash Player ou do AIR. Este exemplo ensina as seguintes técnicas:

- Detecção das capacidades suportadas pela versão do usuário do Flash Player ou do AIR usando a classe Capabilities.
- Uso da classe ExternalInterface para detectar quais configurações de navegador são suportadas pelo navegador do usuário.

Para obter os arquivos de aplicativo deste exemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_br. Os arquivos do aplicativo CapabilitiesExplorer podem ser encontrados na pasta Amostras/CapabilitiesExplorer. O aplicativo consiste nos seguintes arquivos:

Arquivo	Descrição
CapabilitiesExplorer.fla ou CapabilitiesExplorer.mxml	O arquivo principal do aplicativo no Flash (FLA) ou no Flex (MXML).
com/example/programmingas3/capabilities/CapabilitiesGrabber.as	A classe que fornece a funcionalidade principal do aplicativo, incluindo a adição das capacidades do sistema a uma matriz, a classificação dos itens e o uso da classe ExternalInterface para recuperar as capacidades do navegador.
capabilities.html	Um contêiner HTML que contém o JavaScript necessário para se comunicar com a API externa.

Visão geral de CapabilitiesExplorer

O arquivo CapabilitiesExplorer.mxml é responsável por configurar a interface do usuário para o aplicativo CapabilitiesExplorer. As capacidades da versão do usuário do Flash Player ou do AIR serão exibidas dentro de uma ocorrência do componente DataGrid no Palco. As capacidades do navegador também serão exibidas se eles estiverem executando o aplicativo em um contêiner HTML e se a API externa estiver disponível.

Quando o evento `creationComplete` do arquivo do aplicativo principal é despachado, o método `initApp()` é chamado. O método `initApp()` chama o método `getCapabilities()` de dentro da classe com.example.programmingas3.capabilities.CapabilitiesGrabber. O código do método `initApp()` é o seguinte:

```
private function initApp():void
{
    var dp:Array = CapabilitiesGrabber.getCapabilities();
    capabilitiesGrid.dataProvider = dp;
}
```

O método `CapabilitiesGrabber.getCapabilities()` retorna uma matriz classificada do AIR ou do Flash Player e as capacidades do navegador que, em seguida, são definidas para a propriedade `dataProvider` da ocorrência do componente DataGrid `capabilitiesGrid` no Palco.

Visão geral da classe CapabilitiesGrabber

O método estático `getCapabilities()` da classe CapabilitiesGrabber adiciona cada propriedade da classe flash.system.Capabilities a uma matriz (`capDP`). Em seguida, ele chama o método estático `getBrowserObjects()` na classe CapabilitiesGrabber. O método `getBrowserObjects()` usa a API externa para loop sobre o objeto navigator do navegador que contém as capacidades do navegador. O método `getCapabilities()` é o seguinte:

```
public static function getCapabilities():Array
{
    var capDP:Array = new Array();
    capDP.push({name:"Capabilities.avHardwareDisable", value:Capabilities.avHardwareDisable});
    capDP.push({name:"Capabilities.hasAccessibility", value:Capabilities.hasAccessibility});
    capDP.push({name:"Capabilities.hasAudio", value:Capabilities.hasAudio});
    ...
    capDP.push({name:"Capabilities.version", value:Capabilities.version});
    var navArr:Array = CapabilitiesGrabber.getBrowserObjects();
    if (navArr.length > 0)
    {
        capDP = capDP.concat(navArr);
    }
    capDP.sortOn("name", Array.CASEINSENSITIVE);
    return capDP;
}
```

O método `getBrowserObjects()` retorna uma matriz de cada uma das propriedades do objeto navigator no navegador. Se essa matriz tiver um comprimento de um ou mais itens, a matriz de capacidades do navegador (`navArr`) será anexada à matriz de capacidades do Flash Player (`capDP`) e a matriz inteira será classificada alfabeticamente. Finalmente, a matriz classificada será retornada ao arquivo do aplicativo principal que, em seguida preencherá a grade de dados. O código para o método `getBrowserObjects()` é o seguinte:

```
private static function getBrowserObjects():Array
{
    var itemArr:Array = new Array();
    var itemVars:URLVariables;
    if (ExternalInterface.available)
    {
        try
        {
            var tempStr:String = ExternalInterface.call("JS_getBrowserObjects");
            itemVars = new URLVariables(tempStr);
            for (var i:String in itemVars)
            {
                itemArr.push({name:i, value:itemVars[i]});
            }
        }
        catch (error:SecurityError)
        {
            // ignore
        }
    }
    return itemArr;
}
```

Se a API externa estiver disponível no ambiente atual do usuário, o Flash Player chamará o método `JS_getBrowserObjects()` do JavaScript que executa loop sobre o objeto navigator do navegador e retorna uma string de valores codificados de URL ao ActionScript. Em seguida, essa string é convertida em um objeto URLVariables (`itemVars`) e adicionada à matriz `itemArr` que é retornada para o script de chamada.

Comunicação com o JavaScript

A parte final da criação do aplicativo CapabilitiesExplorer é a escrita do JavaScript necessário para executar loop sobre cada um dos itens no objeto navigator do navegador e anexar um par de nome e valor a uma matriz temporária. O código do método `JS_getBrowserObjects()` do JavaScript no container.html é o seguinte:

```
<script language="JavaScript">
    function JS_getBrowserObjects()
    {
        // Create an array to hold each of the browser's items.
        var tempArr = new Array();

        // Loop over each item in the browser's navigator object.
        for (var name in navigator)
        {
            var value = navigator[name];

            // If the current value is a string or Boolean object, add it to the
            // array, otherwise ignore the item.
            switch (typeof(value))
            {
                case "string":
                case "boolean":

                    // Create a temporary string which will be added to the array.
                    // Make sure that we URL-encode the values using JavaScript's
                    // escape() function.
                    var tempStr = "navigator." + name + "=" + escape(value);
                    // Push the URL-encoded name/value pair onto the array.
                    tempArr.push(tempStr);
                    break;
            }
        }
        // Loop over each item in the browser's screen object.
        for (var name in screen)
        {
            var value = screen[name];

            // If the current value is a number, add it to the array, otherwise
            // ignore the item.
            switch (typeof(value))
            {
                case "number":
                    var tempStr = "screen." + name + "=" + escape(value);
                    tempArr.push(tempStr);
                    break;
            }
        }
        // Return the array as a URL-encoded string of name-value pairs.
        return tempArr.join("&");
    }
</script>
```

O código começa com a criação de uma matriz temporária que conterá todos os pares de nome e valor no objeto navigator. Em seguida, um loop é executado no objeto navigator usando um loop`for..in` e o tipo de dados do valor atual é avaliado para filtrar valores indesejados. Neste aplicativo, estamos interessados somente nos valores de string ou booleanos e outros tipos de dados (como funções ou matrizes) são ignorados. Cada valor de string ou booleano no objeto navigator é anexado à matriz `tempArr`. Em seguida, é executado um loop no objeto da tela do navegador usando um loop `for..in` e cada valor numérico é adicionado à matriz `tempArr`. Finalmente, a matriz temporária é convertida em uma string usando o método `Array.join()`. A matriz usa um e comercial (&) como um delimitador, o que permite que o ActionScript analise os dados facilmente usando a classe URLVariables.

Capítulo 29: Copiar e colar

Use as classes na API da área de transferência para copiar informações para e da área de transferência do sistema. Os formatos de dados que podem ser transferidos para ou a partir de um aplicativo em execução no Adobe® AIR™ e no Adobe® Flash® Player incluem:

- Bitmaps (somente AIR)
- Arquivos (somente AIR)
- Texto
- Texto formatado em HTML
- Dados em RTF
- Strings de URLs (somente AIR)
- Objetos serializados
- Referências de objetos (válido somente dentro do aplicativo originador)

Noções básicas de copiar e colar

A API de copiar e colar contém as seguintes classes:

Pacote	Classes
flash.desktop	• Clipboard • ClipboardFormats • ClipboardTransferMode

A propriedade estática `Clipboard.generalClipboard` representa a área de transferência do sistema operacional. A classe Clipboard fornece métodos para a leitura e a gravação de dados em objetos da área de transferência.

As classes HTMLLoader (no AIR) e TextField implementam um comportamento padrão para os atalhos comuns do teclado para copiar e colar. Para implementar o comportamento do atalho de copiar e colar para componentes personalizados, você pode ouvir esses pressionamentos de tecla diretamente. Você também pode usar comandos de menu nativos juntamente com equivalentes de teclas para responder aos pressionamentos de tecla indiretamente.

Diferentes representações das mesmas informações podem ser disponibilizadas em um único objeto Clipboard para aumentar a capacidade de interpretação e utilização de dados por outros aplicativos. Por exemplo, uma imagem pode ser incluída como dados de imagem, um objeto Bitmap serializado ou um arquivo. A renderização dos dados em um determinado formato pode ser adiada para que o formato não seja gerado antes da leitura de seus respectivos dados.

Leitura e gravação na área de transferência do sistema

Para ler a área de transferência do sistema operacional, chame o método `getData()` do objeto `Clipboard.generalClipbooard`, transmitindo o nome do formato a ser lido:

```
import flash.desktop.Clipboard;
import flash.desktop.ClipboardFormats;

if(Clipboard.generalClipboard.hasFormat(ClipboardFormats.TEXT_FORMAT)){
    var text:String = Clipboard.generalClipboard.getData(ClipboardFormats.TEXT_FORMAT);
}
```

Nota: *O conteúdo em execução no Flash Player ou em uma caixa de proteção "não aplicativo" no AIR pode chamar apenas o método* `getData()` *em um manipulador de eventos para um evento* `paste`*. Somente códigos em execução na caixa de proteção do aplicativo AIR podem chamar o método* `getData()` *fora de um manipulador de eventos* `paste`*.*

Para gravar na área de transferência, adicione os dados ao objeto `Clipboard.generalClipboard` em um ou mais formatos. Qualquer dado existente no mesmo formato é sobregravado automaticamente. No entanto, limpar a área de transferência do sistema antes de fazer a gravação de novos dados é uma boa prática para garantir que os dados não relacionados em quaisquer outros formatos também sejam excluídos.

```
import flash.desktop.Clipboard;
import flash.desktop.ClipboardFormats;

var textToCopy:String = "Copy to clipboard.";
Clipboard.generalClipboard.clear();
Clipboard.generalClipboard.setData(ClipboardFormats.TEXT_FORMAT, textToCopy, false);
```

Nota: *O conteúdo em execução no Flash Player ou em uma caixa de proteção "não aplicação" no AIR pode chamar apenas o método* `setData()` *em um manipulador de eventos para um evento de usuário, como eventos de teclado ou do mouse, além de um evento* `copy` *ou* `cut`*. Somente códigos em execução na caixa de proteção do aplicativo AIR podem chamar o método* `setData()` *fora de um manipulador de eventos de usuários.*

Formatos de dados da área de transferência

Os formatos da área de transferência descrevem os dados inseridos em um objeto Clipboard. O Flash Player ou o AIR traduz automaticamente os formatos de dados padrão entre tipos de dados do ActionScript e formatos da área de transferência do sistema. Além disso, os objetos do aplicativo podem ser transferidos internamente em ou entre aplicativos com base no ActionScript, usando formatos definidos pelo aplicativo.

Um objeto Clipboard pode conter representações das mesmas informações em diferentes formatos. Por exemplo, um objeto Clipboard representando um Sprite poderia incluir um formato de referência para ser usado no mesmo aplicativo, um formato serializado para ser usado por outro aplicativo em execução no Flash Player ou no AIR, um formato bitmap para ser usado por um editor de imagens e um formato de lista de arquivo, talvez com renderização adiada para codificar um arquivo PNG, para copiar ou arrastar uma representação do Sprite para o sistema de arquivos.

Formatos de dados padrão

As constantes que definem os nomes de formato padrão são fornecidas na classe ClipboardFormats:

Constante	Descrição
TEXT_FORMAT	Dados em formato de texto são traduzidos para e a partir da classe String do ActionScript.
HTML_FORMAT	Texto com marcação HTML.
RICH_TEXT_FORMAT	Dados em RTF são traduzidos para e a partir da classe ByteArray do ActionScript. O markup RTF não é interpretado ou traduzido.

Constante	Descrição
BITMAP_FORMAT	(somente AIR) Dados em formato bitmap são traduzidos para e da classe BitmapData do ActionScript.
FILE_LIST_FORMAT	(somente AIR) Dados em formato de lista de arquivo são traduzidos para e de uma matriz de objetos File do ActionScript.
URL_FORMAT	(somente AIR) Dados em formato de URL são traduzidos para e a partir da classe String do ActionScript.

Formatos de dados personalizados

Você pode usar formatos personalizados definidos por aplicativo para transferir objetos como referências ou cópias serializadas. As referências são válidas apenas dentro do aplicativo em execução no AIR ou no Flash Player. Objetos serializados podem ser transferidos entre aplicativos em execução no AIR ou no Flash Player, mas podem ser usados somente com objetos que permanecem válidos quando serializados e desserializados. Objetos podem normalmente ser serializados se suas propriedades forem tipos simples ou objetos serializáveis.

Para adicionar um objeto serializado a um objeto Clipboard, defina o parâmetro serializável como `true` ao chamar o método `Clipboard.setData()`. O nome do formato pode ser um dos formatos padrão ou uma string arbitrária definida pelo seu aplicativo.

Modos de transferência

Quando um objeto é gravado na área de transferência usando um formato de dados personalizado, os dados do objeto podem ser lidos da área de transferência como referência ou uma cópia serializada do objeto original. O AIR define quatro modos de transferência que determinam se os objetos são transferidos como referências ou como cópias serializadas:

Modo de transferência	Descrição
ClipboardTransferModes.ORIGINAL_ONLY	Apenas uma referência é retornada. Se nenhuma referência estiver disponível, um valor null será retornado.
ClipboardTransferModes.ORIGINAL_PREFFERED	Uma referência é retornada, se disponível. Caso contrário, uma cópia serializada é retornada.
ClipboardTransferModes.CLONE_ONLY	Apenas uma cópia serializada é retornada. Se nenhuma cópia serializada estiver disponível, um valor null será retornado.
ClipboardTransferModes.CLONE_PREFFERED	Uma cópia serializada é retornada, se disponível. Caso contrário, uma referência é retornada.

Leitura e gravação de formatos de dados personalizados

Você pode usar qualquer string que não comece com os prefixos reservados `air:` ou `flash:` para o parâmetro de formato ao gravar um objeto na área de transferência. Use a mesma seqüência de caracteres do formato para ler o objeto. Os exemplos a seguir ilustram como ler e gravar objetos na área de transferência:

```
public function createClipboardObject(object:Object):Clipboard{
    var transfer:Clipboard = Clipboard.generalClipboard;
    transfer.setData("object", object, true);
}
```

Para extrair um objeto serializado do objeto da área de transferência (após uma operação de soltar ou colar), use o mesmo nome de formato e os modos de transferência `cloneOnly` ou `clonePreferred`.

```
var transfer:Object = clipboard.getData("object", ClipboardTransferMode.CLONE_ONLY);
```

Uma referência é sempre adicionada ao objeto Clipboard. Para extrair a referência do objeto da área de transferência (após uma operação de soltar ou colar), em vez da cópia serializada, use os modos de transferência `originalOnly` ou `originalPreferred`:

```
var transferredObject:Object =
    clipboard.getData("object", ClipboardTransferMode.ORIGINAL_ONLY);
```

As referências são válidas apenas se o objeto Clipboard se originar do aplicativo atual em execução no AIR ou no Flash Player. Use o modo de transferência `originalPreferred` para acessar a referência quando ela estiver disponível e o clone serializado quando a referência não estiver disponível.

Renderização adiada

Se criar um formato de dados for computacionalmente caro, você poderá usar a renderização adiada fornecendo uma função que forneça os dados sob demanda. A função é chamada apenas se um receptor da operação de soltar ou colar solicitar dados no formato adiado.

A função de renderização é adicionada a um objeto Clipboard usando o método `setDataHandler()`. A função deve retornar os dados no formato apropriado. Por exemplo, se você chamou `setDataHandler(ClipboardFormat.TEXT_FORMAT, writeText)`, a função `writeText()` deverá retornar uma string.

Se um formato de dados do mesmo tipo for adicionado a um objeto Clipboard com o método `setData()`, esses dados terão precedência sobre a versão adiada (a função de renderização nunca é chamada). A função de renderização pode ou não ser chamada novamente se os mesmos dados da área de transferência forem acessados novamente.

Nota: *No Mac OS X, a renderização adiada funciona somente com formatos de dados personalizados. No caso de formatos de dados padrão, a função de renderização é chamada imediatamente.*

Colagem de texto usando uma função de renderização adiada

O exemplo a seguir ilustra como implementar uma função de renderização adiada.

Quando um usuário pressiona o botão Copiar, o aplicativo limpa a área de transferência para garantir que nenhum dado de operações anteriores seja mantido. Em seguida, o método `setDataHandler()` define a função `renderData()` como renderizador da área de transferência.

Quando um usuário seleciona o comando Colar no menu de contexto do campo de texto de destino, o aplicativo acessa a área de transferência e define o texto de destino. Como o formato de dados de texto na área de transferência foi definido com uma função em vez de uma string, a área de transferência chama a função `renderData()`. A função `renderData()` retorna o texto no texto de origem, que é então atribuído ao texto de destino.

Observe que se você editar o texto de origem antes de pressionar o botão Colar, a edição será refletida no texto colado, mesmo quando a edição ocorrer após o acionamento do botão Copiar. Isso porque a função de renderização não copia o texto de origem até que o botão Colar seja pressionado. (Ao usar a renderização adiada em um aplicativo real, talvez você queira armazenar ou proteger os dados de origem de alguma maneira para evitar esse problema.)

```
package {
   import flash.desktop.Clipboard;
   import flash.desktop.ClipboardFormats;
   import flash.desktop.ClipboardTransferMode;
   import flash.display.Sprite;
   import flash.text.TextField;
   import flash.text.TextFormat;
   import flash.text.TextFieldType;
   import flash.events.MouseEvent;
   import flash.events.Event;
   public class DeferredRenderingExample extends Sprite
   {
       private var sourceTextField:TextField;
       private var destination:TextField;
       private var copyText:TextField;
       public function DeferredRenderingExample():void
       {
           sourceTextField = createTextField(10, 10, 380, 90);
           sourceTextField.text = "Neque porro quisquam est qui dolorem "
               + "ipsum quia dolor sit amet, consectetur, adipisci velit.";

           copyText = createTextField(10, 110, 35, 20);
           copyText.htmlText = "<a href='#'>Copy</a>";
           copyText.addEventListener(MouseEvent.CLICK, onCopy);

           destination = createTextField(10, 145, 380, 90);
           destination.addEventListener(Event.PASTE, onPaste);
       }
       private function createTextField(x:Number, y:Number, width:Number,
                       height:Number):TextField
       {
           var newTxt:TextField = new TextField();
           newTxt.x = x;
           newTxt.y = y;
           newTxt.height = height;
           newTxt.width = width;
           newTxt.border = true;
           newTxt.multiline = true;
           newTxt.wordWrap = true;
           newTxt.type = TextFieldType.INPUT;
           addChild(newTxt);
           return newTxt;
       }
       public function onCopy(event:MouseEvent):void
       {
           Clipboard.generalClipboard.clear();
           Clipboard.generalClipboard.setDataHandler(ClipboardFormats.TEXT_FORMAT,
                           renderData);
       }
       public function onPaste(event:Event):void
       {
```

```
            sourceTextField.text =
            Clipboard.generalClipboard.getData(ClipboardFormats.TEXT_FORMAT).toString;
        }
        public function renderData():String
        {
            trace("Rendering data");
            var sourceStr:String = sourceTextField.text;
            if (sourceTextField.selectionEndIndex >
                     sourceTextField.selectionBeginIndex)
            {
                return sourceStr.substring(sourceTextField.selectionBeginIndex,
                                    sourceTextField.selectionEndIndex);
            }
            else
            {
                return sourceStr;
            }
        }
    }
}
```

Capítulo 30: Impressão

O Adobe® Flash® Player e o Adobe® AIR™ podem se comunicar com a interface de impressão de um sistema operacional para que você possa enviar páginas para o spooler de impressão. Cada página enviada pelo Flash Player ou pelo AIR ao spooler pode ter conteúdo visível, dinâmico ou fora da tela para o usuário, inclusive valores de banco de dados e texto dinâmico. Além disso, o Flash Player e o AIR definem as propriedades da classe flash.printing.PrintJob com base nas configurações da impressora do usuário, para que você possa formatar as páginas adequadamente.

Este capítulo detalha estratégias para uso dos métodos e propriedades da classe flash.printing.PrintJob para criar um trabalho de impressão, ler as configurações de impressão de um usuário e fazer ajustes em um trabalho de impressão com base no comentário do Flash Player ou do AIR e no sistema operacional do usuário.

Noções básicas de impressão

Introdução à impressão

No ActionScript 3.0, você usa a classe PrintJob para criar instantâneos de conteúdo de exibição a serem convertidos em uma representação de tinta e papel em uma impressão. Sob alguns aspectos, configurar o conteúdo para impressão é o mesmo que configurá-lo para exibição na tela: você posiciona e classifica os elementos por tamanho para criar o layout desejado. No entanto a impressão tem algumas particularidades que a tornam diferente do layout da tela. Por exemplo, a resolução usada por impressoras é diferente da resolução de monitores de computador. O conteúdo de uma tela do computador é dinâmico e pode ser alterado, enquanto o conteúdo impresso é basicamente estático, e no planejamento da impressão, as restrições de tamanho da página fixa e a possibilidade de impressão de várias páginas precisam ser consideradas.

Embora essas diferenças pareçam óbvias, é importante tê-las em mente ao configurar a impressão com o ActionScript. Como a impressão exata depende de uma combinação dos valores especificados por você e das características da impressora do usuário, a classe PrintJob inclui propriedades que permitem determinar as características importantes da impressora do usuário que precisam ser consideradas.

Tarefas comuns de impressão

As seguintes tarefas comuns de impressão são descritas neste capítulo:

- Iniciar um trabalho de impressão
- Incluir páginas em um trabalho de impressão
- Determinar se o usuário cancela um trabalho de impressão
- Especificar se a renderização de vetores ou de bitmaps deve ser usada
- Configurar tamanho de página, escala e orientação
- Especificar a área de conteúdo imprimível
- Converter o tamanho da tela para o tamanho da página
- Imprimir trabalhos de impressão de várias páginas

Conceitos e termos importantes

A lista de referência a seguir contém termos importantes usados neste capítulo:

- Spooler: Uma parte do sistema operacional ou do software do driver da impressora que mantém o controle das páginas que estão aguardando para serem impressas e as envia para a impressora quando ela estiver disponível.
- Orientação da página: A rotação do conteúdo impresso em relação ao papel, horizontal (paisagem) ou vertical (retrato).
- Trabalho de impressão: A página ou o conjunto de páginas que compõem uma única impressão.

Teste dos exemplos do capítulo

Talvez você queira testar as listagens de código de exemplo durante a leitura deste capítulo. Muitas das listagens de código do capítulo são pequenas partes de código, em vez de exemplos completos de trabalhos de impressão ou de código que verifica valores. O teste dos exemplos envolve a criação de elementos a serem impressos e o uso de listagens de código com esses elementos. Os dois exemplos finais do capítulo são exemplos de impressão completos. Eles incluem o código que define o conteúdo a ser impresso, bem como a execução de tarefas de impressão.

Para testar as listagens de código de exemplo:

1. Crie um novo documento Flash.
2. Selecione o quadro-chave no Quadro 1 da Linha de tempo e abra o painel Ações.
3. Copie a listagem de código no painel Script.
4. No menu principal, selecione Controle > Testar filme para criar o arquivo SWF e testar o exemplo.

Impressão de uma página

Você usa uma ocorrência da classe PrintJob para manipular a impressão. Para imprimir uma página básica por meio do Flash Player ou do AIR, use estas quatro instruções em seqüência:

- `new PrintJob()`: Cria uma nova ocorrência do trabalho de impressão com o nome especificado.
- `PrintJob.start()`: Inicia o processo de impressão do sistema operacional chamando a caixa de diálogo de impressão do usuário e preenche as propriedades somente leitura do trabalho de impressão.
- `PrintJob.addPage()`: Contém os detalhes sobre o conteúdo do trabalho de impressão, incluindo o objeto Sprite (e todos os filhos que ele contém), o tamanho da área de impressão e se a impressora deve imprimir a imagem como um vetor ou um bitmap. Você pode usar chamadas sucessivas para `addPage()` para imprimir várias entidades gráficas em várias páginas.
- `PrintJob.send()`: Envia as páginas para a impressora do sistema operacional.

Portanto, por exemplo, um script de trabalho de impressão muito simples pode ser semelhante ao seguinte (incluindo as instruções `package`, `import` e `class` de compilação):

```
package
{
    import flash.printing.PrintJob;
    import flash.display.Sprite;

    public class BasicPrintExample extends Sprite
    {
        var myPrintJob:PrintJob = new PrintJob();
        var mySprite:Sprite = new Sprite();

        public function BasicPrintExample()
        {
            myPrintJob.start();
            myPrintJob.addPage(mySprite);
            myPrintJob.send();
        }
    }
}
```

Nota: *Esse exemplo tem o objetivo de mostrar os elementos básicos de um script de trabalho de impressão e não contém nenhuma manipulação de erros. Para criar um script que responda corretamente a um usuário que cancela um trabalho de impressão, consulte "Trabalho com exceções e retornos" na página 676.*

Para limpar as propriedades de um objeto PrintJob por qualquer motivo, defina a variável PrintJob como `null` (como em `myPrintJob = null`).

Tarefas do Flash Player e do AIR e impressão do sistema

Como o Flash Player e o AIR despacham páginas para a interface de impressão do sistema operacional, você deve compreender o escopo das tarefas gerenciadas pelo Flash Player e pelo AIR e as tarefas gerenciadas pela própria interface de impressão do sistema operacional. O Flash Player e o AIR podem iniciar um trabalho de impressão, ler algumas configurações de página de uma impressora, transmitir o conteúdo de um trabalho de impressão para o sistema operacional e verificar se o usuário ou o sistema cancelou um trabalho de impressão. Outros processos, como exibir caixas de diálogo específicas à impressora, cancelar um trabalho de impressão no spool ou relatar o status da impressora, são todos manipulados pelo sistema operacional. O Flash Player e o AIR podem responder se há um problema ao iniciar ou formatar um trabalho de impressão, mas podem relatar apenas determinadas propriedades ou condições da interface de impressão do sistema operacional. Como desenvolvedor, seu código deve ter a capacidade de responder a essas propriedades ou condições.

Trabalho com exceções e retornos

Você deve verificar se o método `PrintJob.start()` retorna `true` antes de executar as chamadas `addPage()` e `send()`, no caso do usuário ter cancelado o trabalho de impressão. Uma maneira simples de verificar se esses métodos foram cancelados antes de continuar, é delimitá-los em uma instrução `if`, da seguinte maneira:

```
if (myPrintJob.start())
{
    // addPage() and send() statements here
}
```

Se `PrintJob.start()` for `true`, indicando que o usuário selecionou Print (ou o Flash Player ou o AIR iniciou um comando Print), os métodos `addPage()` e `send()` poderão ser chamados.

Além disso, para ajudar a gerenciar o processo de impressão, o Flash Player e o AIR emitem exceções para o método `PrintJob.addPage()` para que seja possível capturar os erros e fornecer informações e opções ao usuário. Se um método `PrintJob.addPage()` falhar, você também poderá chamar outra função ou parar o trabalho de impressão atual. Você captura essas exceções incorporando as chamadas de `addPage()` dentro de uma instrução `try..catch`, conforme no exemplo a seguir. No exemplo, `[params]` é um alocador de espaço para os parâmetros que especificam o conteúdo real a ser impresso:

```
if (myPrintJob.start())
{
    try
    {
        myPrintJob.addPage([params]);
    }
    catch (error:Error)
    {
        // Handle error,
    }
    myPrintJob.send();
}
```

Depois que o trabalho de impressão é iniciado, você pode adicionar o conteúdo usando `PrintJob.addPage()` e verificar se isso gera uma exceção (por exemplo, se o usuário cancelou o trabalho de impressão). Se uma exceção for gerada, você poderá adicionar lógica à instrução `catch` para fornecer informações e opções ao usuário (ou ao Flash Player ou ao AIR) ou poderá parar o trabalho de impressão atual. Se você adicionar a página com êxito, poderá continuar a enviar as páginas à impressora usando `PrintJob.send()`.

Se o Flash Player ou o AIR encontrar um problema quando enviar o trabalho de impressão para a impressora (por exemplo, se a impressora estiver offline), você também poderá capturar essa exceção e apresentar informações ou mais opções (por exemplo, exibir o texto da mensagem ou fornecer um alerta dentro de uma animação) ao usuário (ou ao Flash Player ou ao AIR). Por exemplo, você pode atribuir texto novo a um campo de texto em uma instrução `if..else`, conforme mostrado no código a seguir:

```
if (myPrintJob.start())
{
    try
    {
        myPrintJob.addPage([params]);
    }
    catch (error:Error)
    {
        // Handle error.
    }
    myPrintJob.send();
}
else
{
    myAlert.text = "Print job canceled";
}
```

Para obter um exemplo que funciona, consulte “Exemplo: Escala, corte e resposta” na página 682.

Trabalho com propriedades da página

Quando o usuário clica em OK na caixa de diálogo Imprimir e `PrintJob.start()` retorna `true`, ele pode acessar as propriedades definidas pelas configurações da impressora. Isso inclui a largura e a altura do papel (`pageHeight` e `pageWidth`), e a orientação do conteúdo no papel. Como essas são configurações da impressora, não controladas pelo Flash Player ou pelo AIR, você não pode alterá-las. Mas pode usá-las para alinhar o conteúdo enviado para a impressora para que corresponda às configurações atuais. Para obter mais informações, consulte "Configuração de tamanho, escala e orientação" na página 679.

Configuração da renderização de vetores ou de bitmaps

Você pode definir manualmente o trabalho de impressão para armazenar no spool cada página, como gráficos de vetor ou uma imagem de bitmap. Em alguns casos, a impressão de vetores produz um arquivo de spool menor e uma imagem melhor do que a impressão de bitmap. No entanto, se o conteúdo incluir uma imagem de bitmap, e você desejar preservar qualquer transparência alfa ou efeitos de cor, deverá imprimir a página como uma imagem de bitmap. Além disso, uma impressora não-PostScript converte automaticamente todos os gráficos de vetor em imagens de bitmap.

Nota: *O Adobe AIR não oferece suporte à impressão de vetores no Mac OS.*

Você especifica a impressão de bitmap no terceiro parâmetro de `PrintJob.addPage()`, passando um objeto PrintJobOptions com o parâmetro `printAsBitmap` definido como `true`, da seguinte maneira:

```
var options:PrintJobOptions = new PrintJobOptions();
options.printAsBitmap = true;
myPrintJob.addPage(mySprite, null, options);
```

Se você não especificar um valor para o terceiro parâmetro, o trabalho de impressão usará o padrão, que é a impressão de vetor.

Nota: *Se você não desejar especificar um valor para `printArea` (o segundo parâmetro), mas desejar especificar um valor para impressão de bitmap, use `null` para `printArea`.*

Controle de tempo de instruções de trabalho de impressão

O ActionScript 3.0 não restringe o objeto PrintJob a um único quadro (como faziam as versões anteriores do ActionScript). No entanto, como o sistema operacional exibe informações do status de impressão para o usuário depois que ele clicou no botão OK na caixa de diálogo Imprimir, você deve chamar `PrintJob.addPage()` e `PrintJob.send()` assim que possível para enviar as páginas para o spooler. Um atraso que atinja o quadro que contém a chamada do `PrintJob.send()` atrasará o processo de impressão.

No ActionScript 3.0, há um tempo limite de script de 15 segundos. Portanto, o tempo entre cada instrução principal em uma seqüência de trabalho de impressão não pode exceder 15 segundos. Em outras palavras, o tempo limite de 15 segundos de script se aplica aos seguintes intervalos:

- Entre `PrintJob.start()` e o primeiro `PrintJob.addPage()`
- Entre `PrintJob.addPage()` e o próximo `PrintJob.addPage()`
- Entre o último `PrintJob.addPage()` e `PrintJob.send()`

Se qualquer um desses intervalos se estender por mais de 15 segundos, a chamada seguinte para o `PrintJob.start()`, na ocorrência PrintJob, retornará `false` e o seguinte `PrintJob.addPage()` na ocorrência PrintJob fará com que o Flash Player ou o AIR emitam uma exceção de tempo de execução.

Configuração de tamanho, escala e orientação

A seção "Impressão de uma página" na página 675 fornece detalhes das etapas de um trabalho de impressão básico, em que a saída reflete diretamente o equivalente impresso do tamanho e posição da tela da entidade gráfica especificada. No entanto as impressoras usam resoluções diferentes para impressão e podem ter configurações que afetam contrariamente a aparência da entidade gráfica impressa.

O Flash Player e o AIR podem ler configurações de impressão de um sistema operacional, mas observe que essas propriedades são somente leitura: embora você possa responder a seus valores, não pode defini-los. Portanto, por exemplo, você pode localizar a configuração de tamanho da página da impressora e ajustar seu conteúdo para se adequar ao tamanho. Também é possível determinar as configurações de margem e de orientação da página da impressora. Para responder às configurações da impressora, você pode precisar especificar uma área de impressão, ajustar a diferença entre a resolução da tela e as medidas de ponto da impressora ou transformar o conteúdo para atender às configurações de tamanho ou de orientação da impressora do usuário.

Uso de retângulos para a área de impressão

O método `PrintJob.addPage()` permite especificar a região de uma entidade gráfica que você deseja que seja impressa. O segundo parâmetro, `printArea`, está na forma de um objeto Rectangle. Há três opções para fornecer um valor para esse parâmetro:

- Crie um objeto Rectangle com propriedades específicas e, em seguida, use esse retângulo na chamada de `addPage()`, conforme no exemplo a seguir:

```
private var rect1:Rectangle = new Rectangle(0, 0, 400, 200);
myPrintJob.addPage(sheet, rect1);
```

- Se você ainda não tiver especificado um objeto Rectangle, poderá fazer isso dentro da própria chamada, como no exemplo a seguir:

```
myPrintJob.addPage(sheet, new Rectangle(0, 0, 100, 100));
```

- Se você planejar fornecer valores para o terceiro parâmetro na chamada de `addPage()`, mas não desejar especificar um retângulo, poderá usar `null` para o segundo parâmetro, como no exemplo a seguir:

```
myPrintJob.addPage(sheet, null, options);
```

Nota: *Se você planejar especificar um retângulo para as dimensões de impressão, lembre-se de importar a classe* `flash.display.Rectangle`.

Comparação de pontos e pixels

A largura e a altura do retângulo são valores em pixels. Uma impressora usa pontos como unidades de medida de impressão. Os pontos têm tamanho físico fixo (1/72 polegadas), mas o tamanho de um pixel na tela depende da resolução da tela específica. A taxa de conversão entre pixels e pontos depende das configurações da impressora e se a entidade gráfica está dimensionada. Um sprite não dimensionado, com 72 pixels de largura, terá uma polegada de largura na impressão, com um ponto igual a um pixel, independentemente da resolução da tela.

Você pode usar as equivalências a seguir para converter polegadas ou centímetros em twips (1/20 de um ponto) ou pontos:

- 1 ponto = 1/72 pol = 20 twips
- 1 pol = 72 pontos = 1440 twips
- 1 centímetro = 567 twips

Se você omitir o parâmetro `printArea`, ou se ele for passado incorretamente, a área total da entidade gráfica será impressa.

Escala

Para dimensionar um objeto Sprite antes de imprimi-lo, defina as propriedades da escala (consulte "Manipulação do tamanho e dimensionamento de objetos" na página 299) antes de chamar o método `PrintJob.addPage()` e defina-as novamente como seus valores originais após a impressão. A escala de um objeto Sprite não tem nenhuma relação com a propriedade `printArea`. Em outras palavras, se você especificar uma área de impressão de 50 x 50 pixels, serão impressos 2500 pixels. Se você dimensionar o objeto Sprite, serão impressos os mesmos 2500 pixels, mas o objeto Sprite será impresso no tamanho dimensionado.

Para obter um exemplo, consulte "Exemplo: Escala, corte e resposta" na página 682.

Impressão de orientação paisagem ou retrato

Como o Flash Player e o AIR podem detectar as configurações de orientação, você pode criar lógica no ActionScript para ajustar o tamanho ou a rotação do conteúdo em resposta às configurações da impressora, conforme ilustrado no exemplo a seguir:

```
if (myPrintJob.orientation == PrintJobOrientation.LANDSCAPE)
{
    mySprite.rotation = 90;
}
```

Nota: *Se você planeja ler a configuração do sistema para obter a orientação do conteúdo no papel, lembre-se de importar a classe PrintJobOrientation. A classe PrintJobOrientation fornece valores de constante que definem a orientação do conteúdo na página. Importe a classe usando a seguinte instrução:*

```
import flash.printing.PrintJobOrientation;
```

Resposta à altura e largura da página

Usando uma estratégia semelhante à manipulação das configurações de orientação da impressora, você pode ler as configurações de altura e largura da página e responder a elas incorporando alguma lógica em uma instrução `if`. O código a seguir mostra um exemplo:

```
if (mySprite.height > myPrintJob.pageHeight)
{
    mySprite.scaleY = .75;
}
```

Além disso, as configurações da margem de uma página podem ser determinadas comparando as dimensões da página e do papel, conforme ilustrado no exemplo a seguir:

```
margin_height = (myPrintJob.paperHeight - myPrintJob.pageHeight) / 2;
margin_width = (myPrintJob.paperWidth - myPrintJob.pageWidth) / 2;
```

Exemplo: Impressão de várias páginas

Ao imprimir mais de uma página de conteúdo, você pode associar cada página de conteúdo a uma entidade gráfica diferente (neste caso, `sheet1` e `sheet2`) e usar `PrintJob.addPage()` para cada entidade gráfica. O código a seguir ilustra esta técnica:

```
package
{
    import flash.display.MovieClip;
    import flash.printing.PrintJob;
    import flash.printing.PrintJobOrientation;
    import flash.display.Stage;
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.geom.Rectangle;

    public class PrintMultiplePages extends MovieClip
    {
        private var sheet1:Sprite;
        private var sheet2:Sprite;

        public function PrintMultiplePages():void
        {
            init();
            printPages();
        }

        private function init():void
        {
            sheet1 = new Sprite();
            createSheet(sheet1, "Once upon a time...", {x:10, y:50, width:80, height:130});
            sheet2 = new Sprite();
            createSheet(sheet2, "There was a great story to tell, and it ended quickly.\n\nThe
end.", null);
        }

        private function createSheet(sheet:Sprite, str:String, imgValue:Object):void
        {
            sheet.graphics.beginFill(0xEEEEEE);
            sheet.graphics.lineStyle(1, 0x000000);
            sheet.graphics.drawRect(0, 0, 100, 200);
            sheet.graphics.endFill();

            var txt:TextField = new TextField();
            txt.height = 200;
            txt.width = 100;
            txt.wordWrap = true;
            txt.text = str;

            if (imgValue != null)
            {
                var img:Sprite = new Sprite();
                img.graphics.beginFill(0xFFFFFF);
                img.graphics.drawRect(imgValue.x, imgValue.y, imgValue.width, imgValue.height);
                img.graphics.endFill();
                sheet.addChild(img);
            }
            sheet.addChild(txt);
        }

        private function printPages():void
        {
            var pj:PrintJob = new PrintJob();
```

```
            var pagesToPrint:uint = 0;
            if (pj.start())
            {
                if (pj.orientation == PrintJobOrientation.LANDSCAPE)
                {
                    throw new Error("Page is not set to an orientation of portrait.");
                }

                sheet1.height = pj.pageHeight;
                sheet1.width = pj.pageWidth;
                sheet2.height = pj.pageHeight;
                sheet2.width = pj.pageWidth;

                try
                {
                    pj.addPage(sheet1);
                    pagesToPrint++;
                }
                catch (error:Error)
                {
                // Respond to error.
                }

                try
                {
                    pj.addPage(sheet2);
                    pagesToPrint++;
                }
                catch (error:Error)
                {
                    // Respond to error.
                }

                if (pagesToPrint > 0)
                {
                    pj.send();
                }
            }
        }
    }
}
```

Exemplo: Escala, corte e resposta

Em alguns casos, talvez você queira ajustar o tamanho (ou outras propriedades) de um objeto de exibição ao imprimi-lo para acomodar as diferenças entre a maneira como ele aparece na tela e a maneira como aparece impresso no papel. Quando você ajusta as propriedades de um objeto de exibição antes da impressão (por exemplo, usando as propriedades `scaleX` e `scaleY`), lembre-se de que, se o objeto tiver a escala maior do que o retângulo definido para a área de impressão, ele será cortado. Talvez você também queira redefinir as propriedades depois que as páginas foram impressas.

O código a seguir escala as dimensões do objeto de exibição `txt` (mas não o plano de fundo da caixa verde) e o campo de texto acaba sendo cortado pelas dimensões do retângulo especificado. Após a impressão, o campo de texto volta ao seu tamanho original para exibição na tela. Se o usuário cancelar o trabalho de impressão da caixa de diálogo Imprimir do sistema operacional, o conteúdo no Flash Player ou no AIR será alterado para alertar o usuário de que o trabalho foi cancelado.

```
package
{
    import flash.printing.PrintJob;
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.display.Stage;
    import flash.geom.Rectangle;

    public class PrintScaleExample extends Sprite
    {
        private var bg:Sprite;
        private var txt:TextField;

        public function PrintScaleExample():void
        {
            init();
            draw();
            printPage();
        }

        private function printPage():void
        {
            var pj:PrintJob = new PrintJob();
            txt.scaleX = 3;
            txt.scaleY = 2;
            if (pj.start())
            {
                trace(">> pj.orientation: " + pj.orientation);
                trace(">> pj.pageWidth: " + pj.pageWidth);
                trace(">> pj.pageHeight: " + pj.pageHeight);
                trace(">> pj.paperWidth: " + pj.paperWidth);
                trace(">> pj.paperHeight: " + pj.paperHeight);

                try
                {
                    pj.addPage(this, new Rectangle(0, 0, 100, 100));
                }
                catch (error:Error)
                {
                    // Do nothing.
                }
                pj.send();
            }
            else
            {
                txt.text = "Print job canceled";
            }
            // Reset the txt scale properties.
            txt.scaleX = 1;
            txt.scaleY = 1;
```

```
        }

        private function init():void
        {
            bg = new Sprite();
            bg.graphics.beginFill(0x00FF00);
            bg.graphics.drawRect(0, 0, 100, 200);
            bg.graphics.endFill();

            txt = new TextField();
            txt.border = true;
            txt.text = "Hello World";
        }

        private function draw():void
        {
            addChild(bg);
            addChild(txt);
            txt.x = 50;
            txt.y = 50;
        }
    }
}
```

Capítulo 31: Uso da API externa

A API externa do ActionScript 3.0 permite a comunicação direta entre o ActionScript e o aplicativo de contêiner interno no qual Adobe Flash Player está em execução. Existem diversas situações nas quais você poderá optar por usar a API externa; por exemplo, ao criar a interação entre um documento SWF e o JavaScript em uma página HTML ou ao criar um aplicativo de área de trabalho que usa o Flash Player para exibir um arquivo SWF.

Este capítulo descreve como usar a API externa para interagir com um aplicativo de contêiner, como transmitir dados entre o ActionScript e o JavaScript em uma página HTML e como estabelecer a comunicação e fazer o intercâmbio de dados entre o ActionScript e um aplicativo de área de trabalho.

Nota: *Este capítulo abrange apenas a comunicação entre o ActionScript em um SWF e o aplicativo de contêiner que inclui uma referência ao Flash Player ou à ocorrência na qual o SWF está carregado. Qualquer outro uso do Flash Player em um aplicativo está fora do escopo desta documentação. O Flash Player foi desenvolvido para ser usado como um plug-in de navegador ou como um projetor (aplicativo dedicado). Outros ambientes de uso podem ter suporte limitado.*

Noções básicas de uso da API externa

Introdução ao uso da API externa

Embora em alguns casos um arquivo SWF possa ser executado automaticamente (por exemplo, se você criar um projetor SWF), na maioria dos casos, um aplicativo SWF é executado como um elemento dentro de outro aplicativo. Normalmente, o contêiner que inclui o SWF é um arquivo HTML; um arquivo SWF é usado, com menos freqüência, em toda ou parte da interface de usuário de um aplicativo de área de trabalho.

À medida que você trabalha em aplicativos mais avançados, talvez precise configurar a comunicação entre o arquivo SWF e o aplicativo de contêiner. Por exemplo, uma página da Web normalmente exibe texto ou outras informações em HTML e inclui um arquivo SWF para exibir conteúdo visual dinâmico, como um gráfico ou vídeo. Nesse caso, talvez seja útil especificar que, quando os usuários clicam em um botão na página da Web, alguma coisa deve mudar no arquivo SWF. O ActionScript contém um mecanismo, conhecido como API externa, que facilita esse tipo de comunicação entre o ActionScript em um arquivo SWF e outro código no aplicativo de contêiner.

Tarefas comuns da API externa

As seguintes tarefas comuns da API externa são explicadas neste capítulo:

- Obtenção de informações sobre o aplicativo de contêiner
- Uso do ActionScript para chamar o código em um aplicativo de contêiner, incluindo uma página da Web ou um aplicativo de área de trabalho
- Chamada do código do ActionScript a partir do código de um aplicativo de contêiner
- Criação de um proxy para simplificar a chamada do código do ActionScript a partir de um aplicativo de contêiner

Conceitos e termos importantes

A lista de referência a seguir contém termos importantes usados neste capítulo:

- Contêiner ActiveX: um aplicativo de contêiner (não um navegador da Web) que inclui uma ocorrência do controle ActiveX do Flash Player para exibir o conteúdo do SWF no aplicativo.

- Aplicativo de contêiner: o aplicativo no qual o Flash Player está executando o arquivo SWF, como um navegador da Web e a página HTML que inclui o conteúdo do Flash Player.
- Projetor: um arquivo SWF que foi convertido em um arquivo executável dedicado que inclui o conteúdo do Flash Player e do arquivo SWF. Um projetor pode ser criado no Adobe Flash CS4 Professional ou com o Flash Player independente. Os projetores normalmente são usados para distribuir arquivos SWF por CD-ROM ou em situações similares onde o tamanho do download não é um problema e o autor do SWF quer garantir que o usuário possa executar o arquivo SWF, independentemente da instalação do Flash Player no computador do usuário.
- Proxy: um aplicativo ou código go-between que chama o código de um aplicativo (o "aplicativo externo") em favor de outro aplicativo (o "aplicativo de chamada"), e retorna valores para o aplicativo de chamada. Um proxy pode ser usado por vários motivos, como:
  - Para simplificar o processo de chamada de funções externas, convertendo as chamadas de funções nativas do aplicativo de chamada no formato reconhecido pelo aplicativo externo
  - Para solucionar problemas de segurança ou outras restrições que impedem a comunicação direta do chamador com o aplicativo externo
- Serializar: converter valores de objetos ou dados em um formato que pode ser usado para transmitir os valores em mensagens entre dois sistemas de programação, como via Internet ou entre dois aplicativos diferentes em execução em um único computador.

Teste dos exemplos do capítulo

Talvez você queira testar as listagens de código de exemplo durante a leitura deste capítulo. Muitas listagens de código do capítulo são pequenas listagens apenas para demonstração, não exemplos de trabalho completos ou código que verifica valores. Como o uso da API externa requer (por definição) a gravação do código do ActionScript, bem como do código de um aplicativo de contêiner, o teste dos exemplos envolve a criação de um contêiner (por exemplo, uma página da Web que contém o SWF) e o uso das listagens de código para interagir com o contêiner.

Para testar um exemplo de comunicação entre ActionScript e JavaScript:

1. Crie um novo documento usando a ferramenta de autoria do Flash e salve-o no computador.
2. No menu principal, escolha Arquivo > Configurações de publicação.
3. Na caixa de diálogo Configurações de publicação, na aba Formatos, verifique se as caixas de seleção Flash e HTML estão marcadas.
4. Clique no botão Publicar. Isso gera um arquivo SWF e um arquivo HTML na mesma pasta, com o mesmo nome usado para salvar o documento. Clique em OK para fechar a caixa de diálogo Configurações de publicação.
5. Desmarque a caixa de seleção HTML. Agora que a página HTML foi gerada, você irá modificá-la para adicionar o código do JavaScript apropriado. Desmarcar a caixa de seleção HTML garante que, após a modificação da página HTML, o Flash não substituirá as alterações por uma nova página HTML ao publicar o arquivo SWF.
6. Clique em OK para fechar a caixa de diálogo Configurações de publicação.
7. Com um aplicativo de editor de texto ou HTML, abra o arquivo HTML criado pelo Flash ao publicar o SWF. No código-fonte HTML, adicione as tags `script` de abertura e fechamento e copie-as no código do JavaScript na listagem de código de exemplo:

```
<script>
// add the sample JavaScript code here
</script>
```

8. Salve o arquivo HTML e volte ao Flash.
9. Selecione o quadro-chave no Quadro 1 da Linha de tempo e abra o painel Ações.

10 Copie a listagem de código do ActionScript no painel Script.

11 No menu principal, escolha Arquivo > Publicar para atualizar o arquivo SWF com as alterações feitas.

12 Usando um navegador da Web, abra a página HTML editada para visualizá-la e testar a comunicação entre o ActionScript e a página HTML.

Para testar um exemplo de comunicação de contêiner entre ActionScript e ActiveX:

1 Crie um novo documento usando a ferramenta de autoria do Flash e salve-o no computador. Salve-o na pasta onde o aplicativo de contêiner deverá localizar o arquivo SWF.

2 No menu principal, escolha Arquivo > Configurações de publicação.

3 Na caixa de diálogo Configurações de publicação, na aba Formatos, verifique se apenas a caixa de seleção Flash está marcada.

4 No campo Arquivo próximo à caixa de seleção Flash, clique no ícone de pasta para selecionar a pasta na qual o arquivo SWF será publicado. Uma vez definido o local do arquivo SWF, você pode, por exemplo, manter o documento de autoria em uma determinada pasta e colocar o arquivo SWF publicado em outra pasta, como a que contém o código-fonte do aplicativo de contêiner.

5 Selecione o quadro-chave no Quadro 1 da Linha de tempo e abra o painel Ações.

6 Copie o código do ActionScript do exemplo no painel Script.

7 No menu principal, escolha Arquivo > Publicar para republicar o arquivo SWF.

8 Crie e execute o aplicativo de contêiner para testar a comunicação entre o ActionScript e o aplicativo de contêiner.

Os dois exemplos do final deste capítulo são exemplos completos de uso da API externa para comunicação com uma página HTML e um aplicativo de área de trabalho C#, respectivamente. Esses exemplos incluem o código completo, com o código de verificação de erro do ActionScript e do contêiner, que deve ser usado ao gravar o código com a API externa. Para obter outro exemplo completo de uso da API externa, consulte o exemplo da classe ExternalInterface na Referência dos componentes e da linguagem do ActionScript 3.0.

Requisitos e vantagens da API externa

A API externa é a parte do ActionScript que fornece um mecanismo para comunicação entre o ActionScript e o código em execução em um "aplicativo externo" que está agindo como contêiner para o Flash Player (normalmente um navegador da Web ou aplicativo de projetor dedicado). No ActionScript 3.0, a funcionalidade da API externa é fornecida pela classe ExternalInterface. Nas versões do Flash Player anteriores ao Flash Player 8, a ação `fscommand()` era usada para estabelecer a comunicação com o aplicativo de contêiner. A classe ExternalInterface é uma substituição de `fscommand()`, e seu uso é recomendado para todas as comunicações entre JavaScript e ActionScript.

Nota: *Se precisar usar a função `fscommand()` antiga (por exemplo, para manter a compatibilidade com aplicativos mais antigos ou para interagir com um aplicativo de contêiner SWF de terceiros ou o Flash Player), ela ainda está disponível como uma função no pacote flash.system.*

A classe ExternalInterface é um subsistema que permite a comunicação fácil entre ActionScript e Flash Player com JavaScript em uma página HTML ou com qualquer aplicativo de área de trabalho que inclui uma ocorrência do Flash Player.

A classe ExternalInterface só está disponível nas seguintes condições:

- Em todas as versões compatíveis do Internet Explorer para Windows (5.0 e posterior)

- Em um aplicativo de contêiner como um aplicativo de área de trabalho que usa uma ocorrência do controle ActiveX do Flash Player
- Em qualquer navegador compatível com a interface NPRuntime, que no momento inclui o Firefox 1.0 e posterior, o Mozilla 1.7.5 e posterior, o Netscape 8.0 e posterior e o Safari 1.3 e posterior.

Em todas as outras situações (como a execução em um player dedicado), a propriedade `ExternalInterface.available` retorna `false`.

No ActionScript, é possível chamar uma função JavaScript na página HTML. A API externa tem uma funcionalidade aprimorada em comparação com `fscommand()`:

- É possível usar qualquer função JavaScript, não apenas as funções que podem ser usadas com a função `fscommand()`.
- É possível transmitir qualquer número de argumentos, com qualquer nome; você não fica limitado a transmitir um comando e um único argumento de string. Desse modo, a API externa tem muito mais flexibilidade do que `fscommand()`.
- É possível transmitir vários tipos de dados (como booleano, número e string); você não está mais limitado aos parâmetros String.
- Você pode receber o valor de uma chamada e esse valor retorna imediatamente para o ActionScript (como o valor de retorno da chamada feita).

Importante: *Se o nome dado à ocorrência do Flash Player em uma página HTML (atributo `id` da tag `object`) incluir um hífen (`-`) ou outros caracteres que são definidos como operadores no JavaScript (como `+`, `*`, `/`, `\`, `.` e assim por diante), as chamadas de ExternalInterface a partir do ActionScript não funcionarão quando a página da Web do contêiner for visualizada no Internet Explorer.Além disso, se as tags HTML que definem a ocorrência do Flash Player (as tags `object` e `embed`) estiverem aninhadas em uma tag HTML `form`, as chamadas de ExternalInterface a partir do ActionScript não funcionarão.*

Uso da classe ExternalInterface

A comunicação entre o ActionScript e o aplicativo de contêiner pode ser realizada de duas formas: o ActionScript pode chamar o código (como uma função do JavaScript) definido no contêiner ou o código do contêiner pode chamar uma função do ActionScript que foi designada como chamável. De qualquer modo, as informações podem ser enviadas para o código que está sendo chamado e os resultados podem ser retornados para o código que está fazendo a chamada.

Para facilitar essa comunicação, a classe ExternalInterface inclui duas propriedades estáticas e dois métodos estáticos. Esses métodos e propriedades são usados para obter informações sobre a conexão da interface externa, para executar o código no contêiner do ActionScript e para disponibilizar as funções do ActionScript para serem chamadas pelo contêiner.

Obtenção de informações sobre o contêiner externo

A propriedade `ExternalInterface.available` indica se o Flash Player atual está em um contêiner que oferece uma interface externa. Se a interface externa estiver disponível, esta propriedade será `true`; caso contrário, ela será `false`. Antes de usar qualquer outra funcionalidade na classe ExternalInterface, sempre verifique se o contêiner atual oferece suporte à comunicação da interface externa do seguinte modo:

```
if (ExternalInterface.available)
{
    // Perform ExternalInterface method calls here.
}
```

Nota: *A propriedade* `ExternalInterface.available` *informa se o contêiner atual é um tipo compatível com a conectividade de ExternalInterface. Ela não informa se o JavaScript está ativado no navegador atual.*

A propriedade `ExternalInterface.objectID` permite determinar o identificador exclusivo da ocorrência do Flash Player (especificamente, o atributo `id` da tag `object` no Internet Explorer ou o atributo `name` da tag `embed` nos navegadores que usam a interface NPRuntime). Essa ID exclusiva representa o documenta SWF atual no navegador e pode ser usada para fazer referência ao documento SWF (por exemplo, ao chamar uma função do JavaScript em uma página HTML de contêiner). Quando o contêiner do Flash Player não é um navegador da Web, essa propriedade é `null`.

Chamada do código externo a partir do ActionScript

O método `ExternalInterface.call()` executa o código no aplicativo de contêiner. Ele requer pelo menos um parâmetro, uma string que contém o nome da função a ser chamada no aplicativo de contêiner. Quaisquer parâmetros adicionais transmitidos para o método `ExternalInterface.call()` são transmitidos ao longo do contêiner como parâmetros da chamada de função.

```
// calls the external function "addNumbers"
// passing two parameters, and assigning that function's result
// to the variable "result"
var param1:uint = 3;
var param2:uint = 7;
var result:uint = ExternalInterface.call("addNumbers", param1, param2);
```

Se o contêiner for uma página HTML, esse método invocará a função do JavaScript com o nome especificado, que deve ser definido em um elemento `script` na página HTML de contêiner. O valor de retorno da função do JavaScript é transmitido novamente para o ActionScript.

```
<script language="JavaScript">
    // adds two numbers, and sends the result back to ActionScript
    function addNumbers(num1, num2)
    {
        return (num1 + num2);
    }
</script>
```

Se o contêiner for algum outro contêiner ActiveX, esse método fará com que o controle ActiveX do Flash Player envie o evento `FlashCall`. O nome da função e os parâmetros especificados são serializados em uma string XML pelo Flash Player. O contêiner pode acessar essas informações na propriedade `request` do objeto de evento e usá-las para determinar como seu próprio código deve ser executado. Para retornar um valor para o ActionScript, o código do contêiner chama o método `SetReturnValue()` do objeto ActiveX, transmitindo o resultado (serializado em uma string XML) como um parâmetro desse método. Para obter mais informações sobre o formato XML usado para essa comunicação, consulte "O formato XML da API externa" na página 691.

Independentemente de o contêiner ser um navegador da Web ou outro contêiner ActiveX, se a chamada falhar ou o método do contêiner não especificar um valor de retorno, `null` será retornado. O método `ExternalInterface.call()` lançará uma exceção SecurityError se o ambiente incluído pertencer a uma caixa de proteção de segurança para a qual o código de chamada não tem acesso. Para solucionar isso temporariamente, defina um valor apropriado para `allowScriptAccess` no ambiente incluído. Por exemplo, para alterar o valor de `allowScriptAccess` em uma página HTML, edite o atributo adequado nas tags `object` e `embed`.

Chamada do código do ActionScript a partir do contêiner

Um contêiner só pode chamar o código do ActionScript que esteja em uma função - nenhum outro código do ActionScript pode ser chamado por um contêiner. Para chamar um função do ActionScript a partir do aplicativo de contêiner, você deve fazer duas coisas: registrar a função com a classe ExternalInterface e chamá-la a partir do código do contêiner.

Primeiro, você deve registrar a função do ActionScript para indicar se ela deve ser disponibilizada para o contêiner. Use o método `ExternalInterface.addCallback()` do seguinte modo:

```
function callMe(name:String):String
{
    return "busy signal";
}
ExternalInterface.addCallback("myFunction", callMe);
```

O método `addCallback()` tem dois parâmetros. O primeiro, um nome de função como String, é o nome pelo qual a função será reconhecida pelo contêiner. O segundo parâmetro é a função real do ActionScript que será executada quando o contêiner chamar o nome de função definido. Como esses nomes são diferentes, você pode especificar um nome de função que será usado pelo contêiner, mesmo se a função real do ActionScript tiver um nome diferente. Isso é especialmente útil se o nome da função não for conhecido - por exemplo, se uma função anônima for especificada ou se a função a ser chamada for determinada em tempo de execução.

Depois que uma função do ActionScript é registrada com a classe ExternalInterface, o contêiner pode realmente chamar a função. O modo como isso é feito varia de acordo com o tipo de contêiner. Por exemplo, no código do JavaScript em um navegador da Web, a função do ActionScript é chamada com o nome de função registrado, visto que este é um método do objeto de navegador do Flash Player (isto é, um método do objeto JavaScript que representa a tag `object` ou `embed`). Em outras palavras, os parâmetros são transmitidos e um resultado é retornado como uma função local que está sendo chamada.

```
<script language="JavaScript">
    // callResult gets the value "busy signal"
    var callResult = flashObject.myFunction("my name");
</script>
...
<object id="flashObject"...>
    ...
    <embed name="flashObject".../>
</object>
```

Como alternativa, ao chamar uma função do ActionScript em um arquivo SWF em execução em um aplicativo de área de trabalho, o nome de função registrado e os parâmetros devem ser serializados em uma string XML. Em seguida, a chamada é realmente feita para o método `CallFunction()` do controle ActiveX com a string XML como parâmetro. Para obter mais informações sobre o formato XML usado para essa comunicação, consulte "O formato XML da API externa" na página 691.

De qualquer modo, o valor de retorno da função do ActionScript é transmitido novamente para o código do contêiner, diretamente como um valor quando o chamador é o código JavaScript em um navegador ou serializado com uma string XML quando o chamador é um contêiner ActiveX.

O formato XML da API externa

A comunicação entre o ActionScript e um aplicativo que hospeda o controle ActiveX do Shockwave Flash usa um formato XML específico para codificar chamadas e valores de função. Duas partes do formato XML são usadas pela API externa. Um formato é usado para representar chamadas de função. Outro formato é usado para representar valores individuais; esse formato é usado para parâmetros em funções e em valores de retorno de função. O formato XML das chamadas de função é usado para chamadas feitas e recebidas pelo ActionScript. Para uma chamada de função a partir do ActionScript, o Flash Player transmite o XML para o contêiner; para uma chamada a partir do contêiner, o Flash Player espera que o aplicativo de contêiner o transmita como uma string XML nesse formato. O fragmento XML a seguir mostra o exemplo de uma chamada de função em formato XML:

```
<invoke name="functionName" returntype="xml">
    <arguments>
        ... (individual argument values)
    </arguments>
</invoke>
```

O nó raiz é o nó `invoke`. Ele tem dois atributos: `name` indica o nome da função a ser chamada e `returntype` é sempre `xml`. Se a chamada de função incluir parâmetros, o nó `invoke` terá um nó filho `arguments`, cujos nós filho serão os valores de parâmetro formatados que usam o formato de valor individual explicado a seguir.

Os valores individuais, incluindo parâmetros de função e valores de retorno de função, usam um esquema de formatação que inclui informações sobre o tipo de dados além dos valores reais. A tabela a seguir lista as classes do ActionScript e o formato XML usado para codificar valores desse tipo de dados:

Classe/valor do ActionScript	Classe/valor do C#	Formato	Comentários
nulo	nulo	<nulo/>	
Booleano `true`	bool true	<true/>	
Booleano `false`	bool false	<false/>	
String	string	<string>valor da string</string>	
Número, int, uint	único, duplo, int, uint	`<number>27.5</number>` `<number>-12</number>`	

Classe/valor do ActionScript	Classe/valor do C#	Formato	Comentários

Array (os elementos podem ser de tipos diferentes)	Uma coleção que permite elementos de tipos diferentes, como ArrayList ou object[]	`<array>`
		`<property id="0">`
		`<number>27.5</number>`
		`</property>`
		`<property id="1">`
		`<string>Hello there!</string>`
		`</property>`
		`...`
		`</array>`
			O nó `property` define elementos individuais e o atributo `id` é o índice numérico baseado em zero.

Object	Um dicionário com chaves de string e valores de objeto, como HashTable com chaves de string	`<object>`
		`<property id="name">`
		`<string>John Doe</string>`
		`</property>`
		`<property id="age">`
		`<string>33</string>`
		`</property>`
		`...`
		`</object>`
			O nó `property` define propriedades individuais e o atributo `id` é o nome da propriedade (uma string).

Outras classes internas ou personalizadas		`<null/> or` `<object></object>`	O ActionScript codifica outros objetos como nulos ou como um objeto vazio. De qualquer modo, os valores de propriedade são perdidos.

Nota: *Por exemplo, esta tabela mostra as classes C# equivalentes além das classes do ActionScript; no entanto, a API externa pode ser usada para comunicação com qualquer linguagem de programação ou tempo de execução compatível com os controles ActiveX, não se limitando aos aplicativos C#.*

Quando está criando seus próprios aplicativos usando a API externa com um aplicativo de contêiner ActiveX, você provavelmente achará útil gravar um proxy que irá converter as chamadas de função nativas no formato XML serializado. Para obter um exemplo de uma classe de proxy gravada em C#, consulte "Dentro da classe ExternalInterfaceProxy" na página 702.

Exemplo: uso da API externa em um contêiner de página da Web

Este aplicativo de amostra demonstra técnicas adequadas de comunicação entre o ActionScript e o JavaScript em um navegador da Web, no âmbito de um aplicativo de mensagem instantânea que permite que uma pessoa converse consigo mesma (daí o nome do aplicativo: Introvert IM). As mensagens são enviadas entre um formulário HTML na página da Web e uma interface SWF usando a API externa. As técnicas demonstradas neste exemplo incluem as seguintes:

- Início adequado da comunicação, verificando se o navegador está preparado para se comunicar antes de estabelecer uma comunicação
- Verificação da compatibilidade entre o contêiner e a API externa
- Chamada das funções do JavaScript a partir do ActionScript, transmitindo parâmetros e recebendo valores em resposta
- Disponibilização dos métodos do ActionScript para serem chamados pelo JavaScript, e realização dessas chamadas

Para obter os arquivos de aplicativo desse exemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_br. Os arquivos do aplicativo Introvert IM estão localizados na pasta Amostras/IntrovertIM_HTML. O aplicativo consiste nos seguintes arquivos:

Arquivo	Descrição
IntrovertIMApp.fla ou IntrovertIMApp.mxml	O arquivo de aplicativo principal no Flash (FLA) ou Flex (MXML).
com/example/programmingas3/introvertIM/IMManager.as	A classe que estabelece e gerencia a comunicação entre o ActionScript e o contêiner.
com/example/programmingas3/introvertIM/IMMessageEvent.as	Tipo de evento personalizado, enviado pela classe IMManager quando uma mensagem é recebida do contêiner.
com/example/programmingas3/introvertIM/IMStatus.as	Enumeração cujos valores representam diferentes status de "disponibilidade" que podem ser selecionados no aplicativo.
html-flash/IntrovertIMApp.html ou html-template/index.template.html	A página HTML do aplicativo para Flash (html-flash/IntrovertIMApp.html) ou o modelo usado para criar a página HTML do contêiner do aplicativo para Adobe Flex (html-template/index.template.html). Esse arquivo contém todas as funções do JavaScript que fazem parte do contêiner do aplicativo.

Preparação da comunicação entre o ActionScript e o navegador

A API externa normalmente é usada para permitir a comunicação dos aplicativos do ActionScript com um navegador da Web. Usando a API externa, os métodos do ActionScript podem chamar o código gravado em JavaScript e vice-versa. Devido à complexidade dos navegadores e ao modo de renderização interna das páginas, não é possível garantir que um documento SWF registrará seus retornos de chamada antes da execução do primeiro JavaScript na página HTML. Por esse motivo, antes de chamar funções no documento SWF a partir do JavaScript, o documento SWF sempre deve chamar a página HTML para notificar se o documento SWF está pronto para aceitar conexões.

Por exemplo, por meio de uma série de etapas realizadas pela classe IMManager, o Introvert IM determina se o navegador está pronto para comunicação e prepara o arquivo SWF para comunicação. A primeira etapa, determinar quando o navegador está pronto para comunicação, acontece no construtor IMManager do seguinte modo:

```
public function IMManager(initialStatus:IMStatus)
{
    _status = initialStatus;

    // Check if the container is able to use the external API.
    if (ExternalInterface.available)
    {
        try
        {
            // This calls the isContainerReady() method, which in turn calls
            // the container to see if Flash Player has loaded and the container
            // is ready to receive calls from the SWF.
            var containerReady:Boolean = isContainerReady();
            if (containerReady)
            {
                // If the container is ready, register the SWF's functions.
                setupCallbacks();
            }
            else
            {
                // If the container is not ready, set up a Timer to call the
                // container at 100ms intervals. Once the container responds that
                // it's ready, the timer will be stopped.
                var readyTimer:Timer = new Timer(100);
                readyTimer.addEventListener(TimerEvent.TIMER, timerHandler);
                readyTimer.start();
            }
        }
        ...
    }
    else
    {
        trace("External interface is not available for this container.");
    }
}
```

Primeiro, o código verifica se a API externa está realmente disponível no contêiner atual usando a propriedade `ExternalInterface.available`. Se estiver, o processo de configuração da comunicação é iniciado. Como exceções de segurança e outros erros podem ocorrer ao tentar se comunicar com um aplicativo externo, o código fica entre um bloco `try` (os blocos `catch` correspondentes foram omitidos da listagem para facilitar).

O código a seguir chama o método `isContainerReady()`, listado aqui:

```
private function isContainerReady():Boolean
{
    var result:Boolean = ExternalInterface.call("isReady");
    return result;
}
```

O método `isContainerReady()`, por sua vez, usa o método `ExternalInterface.call()` para chamar a função `isReady()` do JavaScript do seguinte modo:

```
<script language="JavaScript">
<!--
// ------- Private vars -------
var jsReady = false;
...
// ------- functions called by ActionScript -------
// called to check if the page has initialized and JavaScript is available
function isReady()
{
    return jsReady;
}
...
// called by the onload event of the <body> tag
function pageInit()
{
    // Record that JavaScript is ready to go.
    jsReady = true;
}
...
//-->
</script>
```

A função `isReady()` simplesmente retorna o valor da variável `jsReady`. Essa variável é inicialmente `false`; assim que o evento `onload` da página da Web é acionado, o valor da variável muda para `true`. Em outras palavras, se o ActionScript chamar a função `isReady()` antes que a página seja carregada, o JavaScript retornará `false` para `ExternalInterface.call("isReady")` e, conseqüentemente, o método `isContainerReady()` do ActionScript retornará `false`. Assim que a página for carregada, a função `isReady()` do JavaScript retornará `true`, de modo que o método `isContainerReady()` do ActionScript também retornará `true`.

De volta ao construtor IMManager, uma de duas coisas acontecem dependendo da disponibilidade do contêiner. Se `isContainerReady()` retornar `true`, o código simplesmente chamará o método `setupCallbacks()`, que concluirá o processo de configuração da comunicação com o JavaScript. Por outro lado, se `isContainerReady()` retornar `false`, o processo será basicamente suspenso. Um objeto Timer é criado e suspenso para chamar o método `timerHandler()` a cada 100 milissegundos do seguinte modo:

```
private function timerHandler(event:TimerEvent):void
{
    // Check if the container is now ready.
    var isReady:Boolean = isContainerReady();
    if (isReady)
    {
        // If the container has become ready, we don't need to check anymore,
        // so stop the timer.
        Timer(event.target).stop();
        // Set up the ActionScript methods that will be available to be
        // called by the container.
        setupCallbacks();
    }
}
```

Sempre que é chamado, o método `timerHandler()` verifica mais uma vez o resultado do método `isContainerReady()`. Assim que o contêiner é inicializado, esse método retorna `true`. Em seguida, o código pára o objeto Timer e chama o método `setupCallbacks()` para finalizar o processo de configuração da comunicação com o navegador.

Exposição dos métodos do ActionScript ao JavaScript

Conforme mostrou o exemplo anterior, assim que o código determina que o navegador está pronto, o método `setupCallbacks()` é chamado. Esse método prepara o ActionScript para receber chamadas do JavaScript, conforme mostrado a seguir:

```
private function setupCallbacks():void
{
    // Register the SWF client functions with the container
    ExternalInterface.addCallback("newMessage", newMessage);
    ExternalInterface.addCallback("getStatus", getStatus);
    // Notify the container that the SWF is ready to be called.
    ExternalInterface.call("setSWFIsReady");
}
```

O método `setCallBacks()` termina a tarefa de preparação de comunicação com o contêiner chamando `ExternalInterface.addCallback()` para registrar os dois métodos que estarão disponíveis para serem chamados a partir do JavaScript. Neste código, o primeiro parâmetro - o nome pelo qual o método é conhecido pelo JavaScript (`"newMessage"` e `"getStatus"`) - é igual ao nome do método no ActionScript. Neste caso, não seria útil usar nomes diferentes, de modo que o mesmo nome foi reutilizado para simplificar. Finalmente, o método `ExternalInterface.call()` é usado para chamar a função `setSWFIsReady()` do JavaScript, que avisa ao contêiner que as funções do ActionScript foram registradas.

Comunicação do ActionScript com o navegador

O aplicativo Introvert IM demonstra diversos exemplos de chamada das funções do JavaScript na página do contêiner. No caso mais simples (um exemplo do método `setupCallbacks()`), a função `setSWFIsReady()` do JavaScript é chamada sem transmitir nenhum parâmetro nem receber um valor de retorno:

```
ExternalInterface.call("setSWFIsReady");
```

Em outro exemplo do método `isContainerReady()`, o ActionScript chama a função `isReady()` e recebe um valor booleano em resposta:

```
var result:Boolean = ExternalInterface.call("isReady");
```

Também é possível transmitir parâmetros para as funções do JavaScript usando a API externa. Por exemplo, considere o método `sendMessage()` da classe IMManager, que é chamado quando o usuário está enviando uma nova mensagem para seu "parceiro de conversa":

```
public function sendMessage(message:String):void
{
    ExternalInterface.call("newMessage", message);
}
```

Novamente, `ExternalInterface.call()` é usado para chamar a função designada do JavaScript, notificando o navegador sobre a nova mensagem. Além disso, a mensagem propriamente dita é transmitida como um parâmetro adicional para `ExternalInterface.call()` e, conseqüentemente, transmitida como parâmetro para a função `newMessage()` do JavaScript.

Chamada do código do ActionScript a partir do JavaScript

A comunicação é uma rua de mão dupla e o aplicativo Introvert IM não foge à regra. O cliente IM do Flash Player não só chama o JavaScript para enviar mensagens, mas o formulário HTML chama o código do JavaScript para enviar mensagens e solicitar informações do arquivo SWF também. Por exemplo, quando o arquivo SWF avisa o contêiner que terminou de estabelecer o contato e está pronto para se comunicar, a primeira coisa que o navegador faz é chamar o método `getStatus()` da classe IMManager para recuperar o status de disponibilidade inicial do usuário a partir do cliente IM do SWF. Isso é feito na página da Web, na função `updateStatus()`, do seguinte modo:

```
<script language="JavaScript">
...
function updateStatus()
{
    if (swfReady)
    {
        var currentStatus = getSWF("IntrovertIMApp").getStatus();
        document.forms["imForm"].status.value = currentStatus;
    }
}
...
</script>
```

O código verifica o valor da variável `swfReady`, que controla se o arquivo SWF notificou o navegador sobre o registro dos métodos com a classe ExternalInterface. Se o arquivo SWF estiver pronto para receber comunicação, a próxima linha (`var currentStatus =` ...) realmente chamará o método `getStatus()` na classe IMManager. Três coisas acontecem nesta linha de código:

- A função `getSWF()` do JavaScript é chamada, retornando uma referência ao objeto do JavaScript que representa o arquivo SWF. O parâmetro transmitido para `getSWF()` determina qual objeto de navegador é retornado caso haja mais de um arquivo SWF em uma página HTML. O valor transmitido para esse parâmetro deve corresponder ao atributo `id` da tag `object` e ao atributo `name` da tag `embed` usados para incluir o arquivo SWF.
- Usando a referência ao arquivo SWF, o método `getStatus()` é chamado embora seja um método do objeto SWF. Nesse caso, o nome da função "`getStatus`" é usado porque esse é o nome com o qual a função do ActionScript foi registrada com `ExternalInterface.addCallback()`.
- O método `getStatus()` do ActionScript retorna um valor e esse valor é atribuído à variável `currentStatus`, que é atribuída como conteúdo (a propriedade `value`) do campo de texto `status`.

Nota: *Se estiver seguindo o código, você provavelmente percebeu que, no código-fonte da função `updateStatus()`, a linha de código que chama a função `getSWF()` é realmente escrita do seguinte modo: var currentStatus = getSWF("${application}").getStatus(). O texto `${application}` é um alocador de espaço contido no modelo da página HTML. Quando o Adobe Flex Builder 3 gera a página HTML real para o aplicativo, esse alocador é substituído pelo mesmo texto usado como o atributo `id` da tag `object` e o atributo `name` da tag `embed` (que corresponde a `IntrovertIMApp` no exemplo). Este é o valor esperado pela função `getSWF()`.*

A função `sendMessage()` do JavaScript demonstra a transmissão de um parâmetro como uma função do ActionScript. (`sendMessage()` é afunção que foi chamada quando o usuário pressionou o botão Enviar na página HTML.

```
<script language="JavaScript">
...
function sendMessage(message)
{
    if (swfReady)
    {
        ...
        getSWF("IntrovertIMApp").newMessage(message);
    }
}
...
</script>
```

O método `newMessage()` do ActionScript espera um parâmetro, de modo que a variável `message` do JavaScript é transmitida para o ActionScript como um parâmetro na chamada do método `newMessage()` no código do JavaScript.

Detecção do tipo de navegador

Devido às diferenças de acesso ao conteúdo dos navegadores, é importante sempre usar o JavaScript para detectar qual navegados está sendo executado pelo usuário e para acessar o filme de acordo com a sintaxe específica do navegador, usando o objeto de janela ou documento, como mostra a função `getSWF()` do JavaScript neste exemplo:

```
<script language="JavaScript">
...
function getSWF(movieName)
{
    if (navigator.appName.indexOf("Microsoft") != -1)
    {
        return window[movieName];
    }
    else
    {
        return document[movieName];
    }
}
...
</script>
```

Se o seu script não detectar o tipo de navegador do usuário, o usuário poderá observar um comportamento inesperado ao reproduzir arquivos SWF em um contêiner HTML.

Exemplo: uso da API externa em um contêiner ActiveX

Este exemplo demonstra o uso da API externa para comunicação entre o ActionScript e um aplicativo de área de trabalho que usa o controle ActiveX. O exemplo reutiliza o aplicativo Introvert IM, incluindo o código do ActionScript e o mesmo arquivo SWF, e, assim, não descreve o uso da API externa no ActionScript. Para entender esse exemplo, é útil estar familiarizado com o exemplo anterior.

O aplicativo de área de trabalho desse exemplo é gravado em C# com o Microsoft Visual Studio .NET. O foco da discussão são as técnicas específicas para trabalhar com a API externa usando o controle ActiveX. Este exemplo demonstra o seguinte:

- Chamada das funções do ActionScript a partir de um aplicativo de área de trabalho que hospeda o controle ActiveX do Flash Player

- Recebimento das chamadas de função do ActionScript e processamento dessas chamadas em um contêiner ActiveX
- Uso de uma classe de proxy para ocultar os detalhes do formato XML usado pelo Flash Player para as mensagens enviadas para um contêiner ActiveX

Para obter os arquivos de aplicativo desse exemplo, consulte www.adobe.com/go/learn_programmingAS3samples_flash_br. Os arquivos do aplicativo Introvert IM em C# estão localizados na pasta Amostras/IntrovertIM_CSharp. O aplicativo consiste nos seguintes arquivos:

Arquivo	Descrição
AppForm.cs	O arquivo de aplicativo principal com a interface do Windows Forms em C#.
bin/Debug/IntrovertIMApp.swf	O arquivo SWF carregado pelo aplicativo.
ExternalInterfaceProxy/ExternalInterfaceProxy.cs	A classe que serve como envoltório no controle ActiveX para comunicação da interface externa. Ela fornece mecanismos para chamar e receber chamadas a partir do ActionScript.
ExternalInterfaceProxy/ExternalInterfaceSerializer.cs	A classe que converte as mensagens em formato XML do Flash Player em objetos do .NET.
ExternalInterfaceProxy/ExternalInterfaceEventArgs.cs	Esse arquivo define dois tipos de C# (classes): um delegado personalizado e uma classe de argumentos de evento, que são usados pela classe ExternalInterfaceProxy para notificar o ouvinte sobre uma chamada de função do ActionScript.
ExternalInterfaceProxy/ExternalInterfaceCall.cs	Essa classe é um objeto de valor que representa uma chamada de função do ActionScript para o contêiner ActiveX, com propriedades para o nome da função e os parâmetros.
bin/Debug/IntrovertIMApp.swf	O arquivo SWF carregado pelo aplicativo.
obj/AxInterop.ShockwaveFlashObjects.dll, obj/Interop.ShockwaveFlashObjects.dll	Assemblies envoltórios criados pelo Visual Studio .NET que são necessários para acessar o controle ActiveX do Flash Player (Adobe Shockwave® Flash) a partir do código gerenciado.

Visão geral do aplicativo Introvert IM em C#

Este aplicativo de exemplo representa dois programas cliente de mensagem instantânea (um no arquivo SWF e outro incorporado no Windows Forms) que se comunicam. A interface de usuário inclui uma ocorrência do controle ActiveX do Shockwave Flash, na qual é carregado o arquivo SWF que contém o cliente IM do ActionScript. A interface também inclui diversos campos de texto que fazem parte do cliente IM do Windows Forms: um campo para inserir mensagens (`MessageText`), outro que exibe a transcrição das mensagens enviadas entre os clientes (`Transcript`) e um terceiro (`Status`) que exibe o status de disponibilidade definido no cliente IM do SWF.

Inclusão do controle ActiveX do Shockwave Flash

Para incluir o controle ActiveX do Shockwave Flash no seu próprio aplicativo Windows Forms, adicione-o primeiro à caixa de ferramentas do Microsoft Visual Studio.

Para adicionar o controle à caixa de ferramentas:

1. Abra a caixa de ferramentas do Visual Studio.
2. Clique com o botão direito do mouse na seção Windows Forms no Visual Studio 2003 ou em qualquer seção no Visual Studio 2005. No menu de contexto, selecione Adicionar/Remover Itens no Visual Studio 2003 (Escolher Itens... no Visual Studio 2005).

   A caixa de diálogo Personalizar caixa de ferramentas (2003)/Escolher itens da caixa de ferramentas (2005) é exibida.

3 Selecione a aba Componentes COM, que lista todos os componentes COM disponíveis no seu computador, incluindo o controle ActiveX do Flash Player.

4 Navegue até Objeto do Shockwave Flash e selecione-o.

Se esse item não estiver listado, verifique se o controle ActiveX do Flash Player está instalado no seu sistema.

Noções básicas sobre a comunicação do ActionScript com o contêiner ActiveX

A comunicação feita com a API externa e um aplicativo de contêiner ActiveX funciona como a comunicação com um navegador da Web, com uma diferença importante. Conforme descrito anteriormente, quando o ActionScript se comunica com um navegador da Web, contanto que o navegador queira, as funções são chamadas diretamente; os detalhes de como as chamadas e respostas de função são formatadas para serem transmitidas entre o player e o navegador são ocultos. No entanto, quando a API externa é usada para se comunicar com um aplicativo de contêiner ActiveX, o Flash Player envia mensagens (chamadas de função e valores de retorno) ao aplicativo em um formato XML específico e espera chamadas de função e valores de retorno do aplicativo de contêiner para usar o mesmo formato XML. O desenvolvedor do aplicativo de contêiner ActiveX deve gravar o código entendido e pode criar chamadas de função e respostas no formato apropriado.

O exemplo do Introvert IM em C# inclui um conjunto de classes que permitem evitar mensagens de formatação; em vez disso, você pode trabalhar com tipos de dados padrão ao chamar funções do ActionScript e receber chamadas de função do ActionScript. A classe ExternalInterfaceProxy, junto com outras classes auxiliares, fornece essa funcionalidade e pode ser reutilizada em qualquer projeto .NET para facilitar a comunicação da API externa.

As seções de código a seguir, extraídas do formulário do aplicativo principal (AppForm.cs), demonstram a interação simplificada que é atingida com a classe ExternalInterfaceProxy:

```
public class AppForm : System.Windows.Forms.Form
{
    ...
    private ExternalInterfaceProxy proxy;
    ...
    public AppForm()
    {
        ...
        // Register this app to receive notification when the proxy receives
        // a call from ActionScript.
        proxy = new ExternalInterfaceProxy(IntrovertIMApp);
        proxy.ExternalInterfaceCall += new
ExternalInterfaceCallEventHandler(proxy_ExternalInterfaceCall);
        ...
    }
    ...
```

O aplicativo declara e cria uma ocorrência de ExternalInterfaceProxy chamada `proxy`, transmitindo uma referência ao controle ActiveX do Shockwave Flash que está na interface de usuário (`IntrovertIMApp`). Em seguida, o código registra o método `proxy_ExternalInterfaceCall()` para receber o evento `ExternalInterfaceCall` do proxy. Esse evento é enviado pela classe ExternalInterfaceProxy quando uma chamada de função vem do Flash Player. A inscrição nesse evento é o modo como o código C# recebe e responde às chamadas de função do ActionScript.

Quando uma chamada de função vem do ActionScript, a ocorrência de ExternalInterfaceProxy (`proxy`) recebe a chamada, a converte em formato XML e notifica os objetos que são ouvintes do evento `ExternalInterfaceCall` do proxy. No caso da classe AppForm, o método `proxy_ExternalInterfaceCall()` manipula esse evento da seguinte forma:

```
/// <summary>
/// Called by the proxy when an ActionScript ExternalInterface call
/// is made by the SWF
/// </summary>
private object proxy_ExternalInterfaceCall(object sender, ExternalInterfaceCallEventArgs e)
{
    switch (e.FunctionCall.FunctionName)
    {
        case "isReady":
            return isReady();
        case "setSWFIsReady":
            setSWFIsReady();
            return null;
        case "newMessage":
            newMessage((string)e.FunctionCall.Arguments[0]);
            return null;
        case "statusChange":
            statusChange();
            return null;
        default:
            return null;
    }
}
...
```

O método é transmitido como uma ocorrência de ExternalInterfaceCallEventArgs, chamada `e` neste exemplo. Esse objeto tem uma propriedade `FunctionCall` que é uma ocorrência da classe ExternalInterfaceCall.

Uma ocorrência de ExternalInterfaceCall é um simples objeto de valor com duas propriedades. A propriedade `FunctionName` contém o nome de função especificado na instrução `ExternalInterface.Call()` do ActionScript. Se algum parâmetro for adicionado ao ActionScript, esses parâmetros serão incluídos na propriedade `Arguments` do objeto ExternalInterfaceCall. Nesse caso, o método que manipula o evento é simplesmente uma instrução `switch` que age como um gerenciador de tráfego. O valor da propriedade `FunctionName` (`e.FunctionCall.FunctionName`) determina qual método da classe AppForm é chamado.

As ramificações da instrução `switch` na listagem de código anterior demonstram cenários comuns de chamada de método. Por exemplo, qualquer método deve retornar um valor para o ActionScript (como a chamada do método `isReady()`) ou deve retornar `null` (conforme visto nas outras chamadas de método). O acesso aos parâmetros transmitidos do ActionScript é demonstrado na chamada do método `newMessage()` (que é transmitido como o parâmetro `e.FunctionCall.Arguments[0]`, o primeiro elemento da matriz `Arguments`).

Chamar uma função do ActionScript em C# usando a classe ExternalInterfaceProxy é ainda mais simples do que receber uma chamada de função a partir do ActionScript. Para chamar uma função do ActionScript, use o método `Call()` da ocorrência de ExternalInterfaceProxy do seguinte modo:

```
    /// <summary>
    /// Called when the "Send" button is pressed; the value in the
    /// MessageText text field is passed in as a parameter.
    /// </summary>
    /// <param name="message">The message to send.</param>
    private void sendMessage(string message)
    {
        if (swfReady)
        {
            ...
            // Call the newMessage function in ActionScript.
            proxy.Call("newMessage", message);
        }
    }
    ...
    /// <summary>
    /// Call the ActionScript function to get the current "availability"
    /// status and write it into the text field.
    /// </summary>
    private void updateStatus()
    {
        Status.Text = (string)proxy.Call("getStatus");
    }
    ...
}
```

Como mostra este exemplo, o método `Call()` da classe ExternalInterfaceProxy é muito parecido com seu correspondente no ActionScript, `ExternalInterface.Call()`. O primeiro parâmetro é uma string, o nome da função a ser chamada. Todos os parâmetros adicionais (não mostrados aqui) são transmitidos junto com a função do ActionScript. Se a função do ActionScript retornar um valor, esse valor será retornado pelo método `Call()` (conforme visto no exemplo anterior).

Dentro da classe ExternalInterfaceProxy

Usar um envoltório de proxy em torno do controle ActiveX talvez nem sempre seja prático ou você talvez queira gravar sua própria classe de proxy (por exemplo, em uma linguagem de programação diferente ou visando uma plataforma diferente). Embora nem todos os detalhes da criação do proxy sejam explicados aqui, é útil entender o funcionamento da classe de proxy desse exemplo.

Use o método `CallFunction()` do controle ActiveX do Shockwave Flash para chamar uma função do ActionScript a partir do contêiner ActiveX usando a API externa. Isso é mostrado no trecho do método `Call()` da classe ExternalInterfaceProxy:

```
// Call an ActionScript function on the SWF in "_flashControl",
// which is a Shockwave Flash ActiveX control.
string response = _flashControl.CallFunction(request);
```

Neste trecho de código, `_flashControl` é o controle ActiveX do Shockwave Flash. As chamadas de função do ActionScript são feitas com o método `CallFunction()`. Esse método assume um parâmetro (`request` no exemplo), que é uma string que contém as instruções em formato XML, incluindo o nome da função do ActionScript a ser chamada e os parâmetros. Todos os valores retornados do ActionScript são codificados como uma string XML e enviados de volta como o valor de retorno da chamada `CallFunction()`. Neste exemplo, essa string XML é armazenada na variável `response`.

O recebimento de uma chamada de função do ActionScript é um processo de várias etapas. As chamadas de função do ActionScript fazem com que o controle ActiveX do Shockwave Flash envie o evento FlashCall, de modo que a classe (como a classe ExternalInterfaceProxy) que deve receber as chamadas de um arquivo SWF precisa definir um manipulador para esse evento. Na classe ExternalInterfaceProxy, a função do manipulador de eventos é chamada de `_flashControl_FlashCall()`, e é registrada para ouvir o evento no construtor da classe do seguinte modo:

```
private AxShockwaveFlash _flashControl;

public ExternalInterfaceProxy(AxShockwaveFlash flashControl)
{
    _flashControl = flashControl;
    _flashControl.FlashCall += new
_IShockwaveFlashEvents_FlashCallEventHandler(_flashControl_FlashCall);
}
...
private void _flashControl_FlashCall(object sender, _IShockwaveFlashEvents_FlashCallEvent e)
{
    // Use the event object's request property ("e.request")
    // to execute some action.
    ...
    // Return a value to ActionScript;
    // the returned value must first be encoded as an XML-formatted string.
    _flashControl.SetReturnValue(encodedResponse);
}
```

O objeto de evento (`e`) tem uma propriedade `request` (`e.request`) que é uma string que contém informações sobre a chamada de função, como o nome da função e os parâmetros, em formato XML. Essas informações podem ser usadas pelo contêiner para determinar o código a ser executado. Na classe ExternalInterfaceProxy, a solicitação é convertida do formato XML em um objeto ExternalInterfaceCall, que fornece as mesmas informações de forma mais acessível. O método `SetReturnValue()` do controle ActiveX é usado para retornar um resultado de função para o chamador do ActionScript; novamente, o parâmetro resultante deve ser codificado no formato XML usado pela API externa.

A comunicação entre o ActionScript e um aplicativo que hospeda o controle ActiveX do Shockwave Flash usa um formato XML específico para codificar chamadas e valores de função. No exemplo do Introvert IM em C#, a classe ExternalInterfaceProxy possibilita isso para o código no formulário do aplicativo a ser aplicado diretamente nos valores enviados ou recebidos do ActionScript, e ignora os detalhes do formato XML usado pelo Flash Player. Para fazer isso, a classe ExternalInterfaceProxy usa os métodos da classe ExternalInterfaceSerializer para realmente converter as mensagens XML em objetos .NET. A classe ExternalInterfaceSerializer tem quatro métodos públicos:

- `EncodeInvoke()`: codifica um nome de função e um C# ArrayList de argumentos no formato XML apropriado.
- `EncodeResult()`: codifica um valor de resultado no formato XML apropriado.
- `DecodeInvoke()`: decodifica uma chamada de função do ActionScript. A propriedade `request` do objeto de evento FlashCall é transmitida para o método `DecodeInvoke()` e converte a chamada em um objeto ExternalInterfaceCall.
- `DecodeResult()`: decodifica o XML recebido em resultado da chamada de uma função do ActionScript.

Esses métodos codificam valores C# no formato XML da API externa e decodificam o XML em objetos C#. Para obter informações sobre o formato XML usado pelo Flash Player, consulte "O formato XML da API externa" na página 691.

Capítulo 32: Segurança do Flash Player

Segurança é uma preocupação principal da Adobe, dos usuários, dos proprietários de site e dos desenvolvedores de conteúdo. Por esse motivo, o Adobe® Flash® Player inclui um conjunto de regras de segurança e controles para proteger o usuário, o proprietário de site e o desenvolvedor de conteúdo. Este capítulo discute como trabalhar com o modelo de segurança do Flash Player ao desenvolver aplicativos. Neste capítulo, presume-se que todos os arquivos SWF discutidos são publicados com o ActionScript 3.0 e executados no Flash Player 9.0.124.0 ou posterior, a menos que indicado de outra forma.

Este capítulo é apenas uma visão geral de segurança; ele não tenta explicar de maneira abrangente todos os detalhes de implementação, cenários de uso ou ramificações do uso de determinadas APIs. Para obter uma discussão mais detalhada sobre conceitos de segurança do Flash Player, consulte o tópico "Segurança" do Centro de desenvolvedores do Flash Player, em www.adobe.com/go/devnet_security_br.

Para obter informações sobre problemas de segurança do Adobe® AIR™, consulte o capítulo "Segurança do AIR", em http://www.adobe.com/go/learn_air_flash_br.

Visão geral da segurança do Flash Player

Grande parte da segurança do Flash Player é baseada no domínio de origem dos arquivos SWF, mídia e outros ativos carregados. Um arquivo SWF de um domínio específico da Internet, como www.example.com, sempre pode acessar todos os dados daquele domínio. Esses ativos são colocados no mesmo agrupamento de segurança, conhecido como uma *caixa de proteção de segurança*. (Para obter mais informações, consulte "Caixas de proteção de segurança" na página 706.)

Por exemplo, um arquivo SWF pode carregar arquivos SWF, bitmaps, áudio, arquivos de texto e qualquer outro ativo de seu próprio domínio. Além disso, cross-scripting entre dois arquivos SWF do mesmo domínio sempre é permitido, desde que os dois arquivos sejam escritos usando o ActionScript 3.0. *Cross-scripting* é a capacidade de um arquivo SWF de usar o ActionScript para acessar propriedades, métodos e objetos em outro arquivo SWF.

O cross-scripting não é suportado entre arquivos SWF escritos usando o ActionScript 3.0 e versões anteriores do ActionScript. No entanto esses arquivos podem se comunicar usando a classe LocalConnection. Além disso, a capacidade de um arquivo SWF de executar cross-script de arquivos SWF do ActionScript 3.0 de outros domínios e de carregar dados de outros domínios é proibida por padrão. No entanto esse acesso pode ser concedido com uma chamada para o método `Security.allowDomain()` no arquivo SWF carregado. Para obter mais informações, consulte "Cross-scripting" na página 721.

Por padrão, as seguintes regras básicas de segurança sempre se aplicam:

- Recursos na mesma caixa de proteção de segurança sempre podem acessar um ao outro.
- Arquivos SWF em uma caixa de proteção remota nunca acessam arquivos e dados locais.

O Flash Player considera os seguintes domínios como sendo domínios individuais e configura caixas de proteção de segurança individuais para cada um:

- `http://example.com`
- `http://www.example.com`
- `http://store.example.com`
- `https://www.example.com`

- `http://192.0.34.166`

Mesmo que um domínio nomeado, como http://example.com, mapeie para um endereço IP específico, como http://192.0.34.166, o Flash Player configurará caixas de proteção de segurança separadas para cada um.

Existem dois métodos básicos que podem ser usados por um desenvolvedor para conceder acesso a um arquivo SWF a ativos das caixas de proteção diferente daquela do arquivo SWF:

- O método `Security.allowDomain()` (consulte "Controles de autor (desenvolvedor)" na página 714)
- O arquivo de política da URL (consulte "Controles de site (arquivos de política)" na página 711)

No modelo de segurança do Flash Player, há uma distinção entre carregamento de conteúdo e extração ou acesso a dados. *Conteúdo* é definido como mídia, incluindo mídia visual que o Flash Player pode exibir, áudio, vídeo ou um arquivo SWF que inclui mídia exibida. *Dados* são definidos como algo que é acessível apenas ao código ActionScript. Conteúdo e dados são carregados de maneiras diferentes.

- Carregamento de conteúdo — é possível carregar conteúdo usando classes, como as classes Loader, Sound e NetStream.
- Extração de dados — é possível extrair dados de conteúdo de mídia carregada usando objetos Bitmap, o método `BitmapData.draw()`, a propriedade `Sound.id3` ou o método `SoundMixer.computeSpectrum()`.
- Acesso a dados — é possível acessar dados diretamente carregando-os de um arquivo externo (como um arquivo XML) usando classes, como as classes URLStream, URLLoader, Socket e XMLSocket.

O modelo de segurança do Flash Player define diferentes regras para carregamento de conteúdo e acesso a dados. Em geral, há menos restrições no carregamento de conteúdo do que no acesso a dados.

Em geral, o conteúdo (arquivos SWF, bitmaps, arquivos mp3 e vídeos) pode ser carregado de qualquer lugar, mas se o conteúdo for de um domínio diferente daquele do arquivo SWF carregado, ele será particionado em uma caixa de proteção de segurança separada.

Há algumas barreiras no que diz respeito ao carregamento de conteúdo:

- Por padrão, arquivos SWF locais (carregados de um endereço que não seja da rede, como do disco rígido de um usuário) são classificados na caixa de proteção local com sistema de arquivos. Esses arquivos não podem carregar conteúdo da rede. Para obter mais informações, consulte "Caixas de proteção locais" na página 706.
- Servidores de protocolo RTMP podem limitar o acesso ao conteúdo. Para obter mais informações, consulte"Conteúdo entregue usando servidores RMTP" na página 721.

Se a mídia carregada for uma imagem, áudio ou vídeo, seus dados, como dados de pixel e dados de som, poderão ser acessados por um arquivo SWF fora de sua caixa de proteção de segurança apenas se o domínio daquele arquivo SWF tiver sido incluído em um arquivo de política de URL no domínio de origem da mídia. Para obter detalhes, consulte "Acesso à mídia carregada como dados" na página 724.

Outras formas de dados carregados incluem arquivos de texto ou XML que são carregados com um objeto URLLoader. Novamente nesse caso, para acessar quaisquer dados de outra caixa de proteção de segurança, permissão deve ser concedida por meio de um arquivo de política de URL no domínio de origem. Para obter detalhes, consulte "Uso de URLLoader e URLStream" na página 727.

Caixas de proteção de segurança

Computadores clientes podem obter arquivos SWF individuais de várias origens, como de sites externos ou de um sistema de arquivos local. O Flash Player atribui arquivos SWF e outros recursos individualmente, como objetos compartilhados, bitmaps, sons, vídeos e arquivos de dados, a caixas de proteção de segurança com base em sua origem quando eles são carregadas no Flash Player. As seções a seguir descrevem as regras reforçadas pelo Flash Player que controlam o que um arquivo SWF dentro de uma determinada caixa de proteção pode acessar.

Para obter mais informações sobre a segurança do Flash Player, consulte o tópico "Segurança" do Centro de desenvolvedores do Flash Player, em www.adobe.com/go/devnet_security_br.

Caixas de proteção remotas

O Flash Player classifica ativos (incluindo arquivos SWF) da Internet em caixas de proteção separadas que correspondem a seus domínios de origem do site. Por padrão, esses arquivos têm permissão para acessar quaisquer recursos de seu próprio servidor. Arquivos SWF remotos podem receber permissão para acessar dados adicionais de outros domínios por permissões explícitas de autor e de site, como arquivos de política de URL e o método `Security.allowDomain()`. Para obter detalhes, consulte "Controles de site (arquivos de política)" na página 711 e"Controles de autor (desenvolvedor)" na página 714.

Arquivos SWF remotos não podem carregar nenhum recurso ou arquivo local.

Para obter mais informações sobre a segurança do Flash Player, consulte o tópico "Segurança" do Centro de desenvolvedores do Flash Player, em www.adobe.com/go/devnet_security_br.

Caixas de proteção locais

Arquivo local descreve qualquer arquivo que é referenciado usando o protocolo `file:` ou um caminho UNC (Convenção de nomenclatura universal). Arquivos SWF locais são colocados em uma de quatro caixas de proteção locais:

- A caixa de proteção local com sistema de arquivos — por motivos de segurança, o Flash Player coloca todos os arquivos SWF e ativos locais na caixa de proteção local com sistema de arquivos, por padrão. Nessa caixa de proteção, arquivos SWF podem ler arquivos locais (usando a classe URLLoader, por exemplo), mas não podem se comunicar com a rede de nenhuma maneira. Isso garante ao usuário que os dados locais não podem ser vazados para a rede ou, de outra forma, compartilhados de maneira inadequada.
- A caixa de proteção local com a rede — ao compilar um arquivo SWF, é possível especificar que ele tem acesso à rede quando executado como um arquivo local (consulte "Configuração de tipo de caixa de proteção de arquivos SWF locais" na página 707). Esses arquivos são colocados na caixa de proteção local com a rede. Arquivos SWF que são atribuídos à caixa de proteção local com a rede perdem o acesso a seus arquivos locais. Em troca, os arquivos SWF têm permissão para acessar dados da rede. No entanto um arquivo SWF local com a rede ainda não tem permissão para ler nenhum dado derivado da rede a menos que permissões estejam presentes para aquela ação, por meio de um arquivo de política de URL ou de uma chamada ao método `Security.allowDomain()`. Para conceder essa permissão, um arquivo de política de URL deve conceder permissão a *todos* os domínios usando `<allow-access-from domain="*"/>` ou usando `Security.allowDomain("*")`. Para obter informações, consulte"Controles de site (arquivos de política)" na página 711 e"Controles de autor (desenvolvedor)" na página 714.

- A caixa de proteção local confiável — arquivos SWF locais que estão registrados como confiáveis (pelos usuários ou pelos programas instaladores) são colocados na caixa de proteção local confiável. Administradores do sistema e usuários também têm a capacidade de reatribuir (mover) um arquivo SWF local para ou de um caixa de proteção local confiável com base em considerações de segurança (consulte"Controles de administrador" na página 708 e "Controles de usuário" na página 710). Arquivos SWF atribuídos à caixa de proteção local confiável podem interagir com quaisquer outros arquivos SWF e carregar dados de qualquer lugar (remoto ou local).
- A caixa de proteção do aplicativo AIR — esta caixa de proteção contém conteúdo instalado com o aplicativo AIR em execução. Por padrão, os arquivos na caixa de proteção do aplicativo AIR podem executar cross-script de qualquer arquivo em qualquer domínio. No entanto, arquivos fora da caixa de proteção do aplicativo AIR não têm permissão para executar cross-script no arquivo AIR. Por padrão, os arquivos na caixa de proteção do aplicativo AIR podem carregar conteúdo e dados de qualquer domínio.

A comunicação entre as caixas de proteção local com rede e local com sistema de arquivos, bem como a comunicação entre as caixas de proteção remota e local com sistema de arquivos, é estritamente proibida. Permissão para essa comunicação não pode ser concedida por um aplicativo executado no Flash Player ou por um usuário ou administrador.

Script em qualquer direção entre arquivos HTML locais e arquivos SWF locais, por exemplo, usando a classe ExternalInterface, exige que tanto os arquivos HTML quanto os arquivos SWF envolvidos estejam na caixa de proteção confiável local. Isso ocorre porque os modelos de segurança locais de navegadores diferem do modelo de segurança local do Flash Player.

Arquivos SWF na caixa de proteção local com rede não podem carregar arquivos SWF na caixa de proteção local com sistema de arquivos. Arquivos SWF na caixa de proteção local com sistema de arquivos não podem carregar arquivos SWF na caixa de proteção local com rede.

Configuração de tipo de caixa de proteção de arquivos SWF locais

É possível configurar um arquivo SWF para a caixa de proteção local com sistema de arquivos ou a caixa de proteção local com rede definindo-se as configurações de publicação do documento na Ferramenta de autoria.

Um usuário final ou o administrador de um computador pode especificar que um arquivo SWF local é confiável, permitindo que ele carregue dados de todos os domínios, tanto locais quanto de rede. Isso é especificado nos diretórios Global Flash Player Trust e User Flash Player Trust. Para obter mais informações, consulte "Controles de administrador" na página 708 e "Controles de usuário" na página 710.

Para obter mais informações sobre caixas de proteção locais, consulte "Caixas de proteção locais" na página 706.

A propriedade Security.sandboxType

Um autor de um arquivo SWF pode usar a propriedade estática somente leitura `Security.sandboxType` para determinar o tipo de caixa de proteção à qual o Flash Player atribuiu o arquivo SWF. A classe Security inclui constantes que representam valores possíveis da propriedade `Security.sandboxType`, da seguinte maneira:

- `Security.REMOTE` — o arquivo SWF é proveniente de uma URL da Internet e opera de acordo com as regras da caixa de proteção com base em domínio.
- `Security.LOCAL_WITH_FILE` — o arquivo SWF é um arquivo local, mas não foi considerado como confiável pelo usuário e não foi publicado com uma designação de rede. O arquivo SWF pode ler de fontes de dados locais, mas não pode se comunicar com a Internet.
- `Security.LOCAL_WITH_NETWORK` — o arquivo SWF é um arquivo local e não foi considerado como confiável pelo usuário, mas foi publicado com uma designação de rede. O arquivo SWF pode se comunicar com a Internet mas não pode ler de fontes de dados locais.

- `Security.LOCAL_TRUSTED` — o arquivo SWF é um arquivo local e foi considerado como confiável pelo usuário, seja usando o Gerenciador de configurações ou um arquivo de configuração de confiança do Flash Player. O arquivo SWF pode ler de fontes de dados locais e se comunicar com a Internet.
- `Security.APPLICATION` — o arquivo SWF está em execução em um aplicativo AIR e foi instalado com o pacote (arquivo AIR) daquele aplicativo. Por padrão, os arquivos na caixa de proteção do aplicativo AIR podem executar cross-script em qualquer arquivo em qualquer domínio. No entanto, arquivos fora da caixa de proteção do aplicativo AIR não têm permissão para executar cross-script no arquivo AIR. Por padrão, os arquivos na caixa de proteção do aplicativo AIR podem carregar conteúdo e dados de qualquer domínio.

Controles de permissão

O modelo de segurança de tempo de execução do cliente Flash Player foi designado em torno de recursos que são objetos, como arquivos SWF, dados locais e URLs da Internet. *Participantes* são as partes que são as proprietárias ou que usam esses recursos. Os participantes podem exercer controles (configurações de segurança) sobre seus próprios recursos e cada recurso tem quatro participantes. O Flash Player reforça estritamente uma hierarquia de autoridade para esses controles, conforme mostrado na ilustração a seguir:

Hierarquia de controles de segurança

Isso significa, por exemplo, que se um administrador restringir o acesso a um recurso, nenhum outro participante poderá substituir essa restrição.

Controles de administrador

Um usuário administrativo de um computador (um usuário que fez logon com direitos administrativos) pode aplicar configurações de segurança do Flash Player que afetam todos os usuários do computador. Em um ambiente não empresarial, como em um computador doméstico, normalmente há um usuário que também tem acesso administrativo. Mesmo em um ambiente empresarial, usuários individuais podem ter direitos administrativos no computador.

Há dois tipos de controles de usuário administrativo:

- O arquivo mms.cfg
- O diretório Global Flash Player Trust

O arquivo mms.cfg

O arquivo mms.cfg é um arquivo de texto que permite que os administradores ativem ou restrinjam o acesso a vários recursos. Quando o Flash Player é iniciado, lê as configurações de segurança desse arquivo e as usa para limitar a funcionalidade. O arquivo mms.cfg inclui configurações que o administrador usa para gerenciar recursos, como controles de privacidade, segurança de arquivo local, conexões de soquete, entre outros.

Um arquivo SWF pode acessar algumas informações sobre capacidades que foram desativadas chamando as propriedades `Capabilities.avHardwareDisable` e `Capabilities.localFileReadDisable`. No entanto a maioria das configurações no arquivo mms.cfg não podem ser consultadas no ActionScript.

Para reforçar as políticas de segurança e de privacidade independentemente do aplicativo para um computador, o arquivo mms.cfg deve ser modificado apenas por administradores do sistema. O arquivo mms.cfg não deve ser usado por instaladores de aplicativos. Embora um instalador que executa com privilégios administrativos possa modificar o conteúdo do arquivo mms.cfg, a Adobe considera esse uso como uma violação da confiança do usuário e recomenda que os criadores de instaladores nunca modifiquem o arquivo mms.cfg.

O arquivo mms.cfg é armazenado no seguinte local:

- Windows: *system*\Macromed\Flash\mms.cfg

  (por exemplo, C:\WINDOWS\system32\Macromed\Flash\mms.cfg)

- Mac: *app support*/Macromedia/mms.cfg

  (por exemplo, /Library/Application Support/Macromedia/mms.cfg)

Para obter mais informações sobre o arquivo mms.cfg, consulte o Guia de administração do Flash Player, em www.adobe.com/go/flash_player_admin_br.

O diretório Global Flash Player Trust

Usuários administrativos e aplicativos de instalador podem registrar arquivos SWF locais especificados como confiáveis para todos os usuários. Esses arquivos SWF são atribuídos à caixa de proteção confiável local. Eles podem interagir com quaisquer outros arquivos SWF e podem carregar dados de qualquer lugar remoto ou local. Os arquivos são designados como confiáveis no diretório Global Flash Player Trust, no seguinte local:

- Windows: *system*\Macromed\Flash\FlashPlayerTrust

  (por exemplo, C:\WINDOWS\system32\Macromed\Flash\FlashPlayerTrust)

- Mac: *app support*/Macromedia/FlashPlayerTrust

  (por exemplo, /Library/Application Support/Macromedia/FlashPlayerTrust)

O diretório Flash Player Trust pode conter qualquer número de arquivos de texto, cada um dos quais lista caminhos confiáveis, com um caminho por linha. Cada caminho pode ser um arquivo SWF individual, um arquivo HTML ou um diretório. Linhas de comentários começam com o símbolo `#`. Por exemplo, um arquivo de configuração de confiança do Flash Player que contém o texto a seguir concede status confiável a todos os arquivos no diretório especificado e em todos os subdiretórios:

```
# Trust files in the following directories:
C:\Documents and Settings\All Users\Documents\SampleApp
```

Os caminhos listados em um arquivo de configuração de confiança devem sempre ser caminhos locais ou caminhos de rede SMB. Qualquer caminho HTTP em um arquivo de configuração confiável é ignorado. Apenas arquivos locais podem ser confiáveis.

Para evitar conflitos, dê a cada arquivo de configuração confiável um nome de arquivo correspondente ao aplicativo de instalação e use uma extensão de arquivo .cfg.

Como um desenvolvedor que distribui um arquivo SWF executado localmente por meio de um aplicativo instalador, é possível fazer com que o aplicativo instalador adicione um arquivo de configuração ao diretório Global Flash Player Trust, concedendo privilégios totais ao arquivo que você está distribuindo. O aplicativo instalador deve ser executado por um usuário com direitos administrativos. Ao contrário do arquivo ms.cfg, o diretório Global Flash Player Trust é incluído com o objetivo de aplicativos instaladores concederem permissões de confiança. Os usuários administrativos e os aplicativos instaladores podem designar aplicativos locais confiáveis usando o diretório Global Flash Player Trust.

Também há diretórios Flash Player Trust para usuários individuais (consulte "Controles de usuário" na página 710).

Controles de usuário

O Flash Player fornece três mecanismos diferentes em nível de usuário para definir permissões: a UI de Configurações, o Gerenciador de configurações e o diretório User Flash Player Trust.

A UI de Configurações e o Gerenciador de configurações

A UI de Configurações é um mecanismo rápido e interativo para definir as configurações de um domínio específico. O Gerenciador de configurações tem uma interface mais detalhada e permite fazer alterações globais que afetam as permissões de muitos ou de todos os domínios. Além disso, quando um arquivo SWG solicita uma nova permissão, exigindo a tomada de decisões relativas a segurança ou privacidade durante a execução, são exibidas caixas de diálogo nas quais os usuários podem ajustar algumas configurações do Flash Player.

O Gerenciador de configurações e a UI de Configurações oferecem opções relacionadas a segurança, como configurações de câmera e microfone, configurações de armazenamento de objetos compartilhados, configurações relacionadas a conteúdo pré-existente etc.

Nota: *Qualquer configuração feita no arquivo mms.cfg (consulte "Controles de administrador" na página 708) não é refletida no Gerenciador de configurações.*

Para obter detalhes sobre o Gerenciador de configurações, consulte www.adobe.com/go/settingsmanager_br.

O diretório User Flash Player Trust

Usuários e aplicativos instaladores podem registrar arquivos SWF locais como confiáveis. Esses arquivos SWF são atribuídos à caixa de proteção confiável local. Eles podem interagir com quaisquer outros arquivos SWF e podem carregar dados de qualquer lugar remoto ou local. Um usuário designa um arquivo como confiável no diretório User Flash Player Trust, que está no mesmo diretório que a área de armazenamento de objetos compartilhados, nos seguintes locais (os locais são específicos do usuário atual):

- Windows: app data\Macromedia\Flash Player\#Security\FlashPlayerTrust

  (por exemplo, C:\Documents and Settings\JohnD\Application Data\Macromedia\Flash Player\#Security\FlashPlayerTrust no Windows XP ou C:\Users\JohnD\AppData\Roaming\Macromedia\Flash Player\#Security\FlashPlayerTrust no Windows Vista)

  No Windows, a pasta Application Data fica ocultada por padrão. Para mostrar pastas e arquivos ocultos, selecione Meu computador para abrir o Windows Explorer, selecione Ferramentas > Opções de pasta e selecione a aba Exibir. Na aba Exibir, selecione o botão de opção Mostrar pastas e arquivos ocultos.

- Mac: app data/Macromedia/Flash Player/#Security/FlashPlayerTrust

  (por exemplo, /Users/JohnD/Library/Preferences/Macromedia/Flash Player/#Security/FlashPlayerTrust)

  Essas configurações afetam apenas o usuário atual, não outros usuários que fazem logon no computador. Se um usuário sem direitos administrativos instalar um aplicativo em sua própria parte do sistema, o diretório User Flash Player Trust permitirá que o instalador registre o aplicativo como confiável para aquele usuário.

Como um desenvolvedor que distribui um arquivo SWF executado localmente por meio de um aplicativo instalador, você pode fazer com que o aplicativo instalador adicione um arquivo de configuração ao diretório User Flash Player Trust, concedendo privilégios totais ao arquivo que está distribuindo. Mesmo nessa situação, o arquivo do diretório User Flash Player Trust é considerado um controle de usuário, porque uma ação do usuário (instalação) o inicia.

Também existe um diretório Global Flash Player Trust, usado pelo usuário administrativo ou por instaladores para registrar um aplicativo para outros usuários de um computador (consulte "Controles de administrador" na página 708).

Controles de site (arquivos de política)

Para disponibilizar dados do servidor Web para arquivos SWF de outros domínios, crie um arquivo de política do servidor. Um *arquivo de política* é um arquivo XML colocado em um local específico no servidor.

arquivos de política afetam o acesso a vários ativos, incluindo os seguintes:

- Dados em bitmaps, sons e vídeos
- Carregamento de arquivos de texto e XML
- Importação de arquivos SWF de outros domínios de segurança no domínio de segurança do arquivo SWF que está sendo carregado
- Acesso a soquete e conexões de soquete XML

Objetos ActionScript instanciam dois tipos diferentes de conexões de servidor: conexões de servidor com base em documento e conexões de soquete. Os objetos do ActionScript, como Loader, Sound, URLLoader e URLStream instanciam conexões de servidor com base em documento e esses objetos carregam um arquivo de uma URL. Os objetos Socket e XMLSocket do ActionScript fazem conexões de soquete que funcionam com fluxos de dados, não com documentos carregados.

Como o Flash Player oferece suporte a dois tipos de conexões de servidor, há dois tipos de arquivos de política: arquivos de política de URL e arquivos de política de soquete.

- Conexões com base em documento exigem *arquivos de política de URL*. Esses arquivos permitem que o servidor indique que seus dados e documentos estão disponíveis para arquivos SWF servidos em determinados domínios ou em todos os domínios.
- Conexões de soquete exigem *arquivos de política de soquetes* que ativam a rede diretamente no nível inferior do soquete TCP usando as classes Socket e XMLSocket.

O Flash Player exige que arquivos de política sejam transmitidos usando o mesmo protocolo que a tentativa de conexão quer usar. Por exemplo, quando você coloca um arquivo de política no servidor HTTP, os arquivos SWF de outros domínios recebem permissão para carregar dados dele como um servidor HTTP. No entanto, se não fornecer um arquivo de política de soquete no mesmo servidor, você estará proibindo que arquivos SWF de outros domínios se conectem ao servidor no nível de soquete. Em outras palavras, o meio pelo qual um arquivo de política é recuperado deve corresponder ao meio de conexão.

O uso e a sintaxe do arquivo de política são discutidos brevemente no restante desta seção, pois eles se aplicam aos arquivos SWF publicados para o Flash Player 10. A implementação do arquivo de política é um pouco diferente em versões anteriores do Flash Player, uma vez que versões sucessivas reforçaram a segurança do Flash Player. Para obter informações mais detalhadas sobre arquivos de política, consulte o tópico "Alterações em arquivos de política do Flash Player 9" do Centro de desenvolvedores do Flash Player, em www.adobe.com/go/devnet_security_br.

arquivos de política mestre

Por padrão, primeiro o Flash Player (e o conteúdo AIR que não está na caixa de proteção do aplicativo AIR) procura um arquivo de política de URL denominado `crossdomain.xml` no diretório raiz do servidor e procura um arquivo de política de soquete na porta 843. Um arquivo em um desses locais é chamado de *arquivo de política mestre*. No caso de conexões de soquete, o Flash Player também procura um arquivo de política de soquete na mesma porta que a conexão principal. No entanto, um arquivo de política encontrado naquela porta não é considerado um arquivo de política mestre.

Além de especificar permissões de acesso, o arquivo de política mestre também pode conter uma instrução *meta-policy*. Uma metapolítica especifica quais locais podem conter arquivos de política. A metapolítica padrão dos arquivos de política de URL é "somente mestre", o que significa que /crossdomain.xml é o único arquivo de política permitido no servidor. A metapolítica padrão para arquivos de política de soquete é "all", o que significa que qualquer soquete no host pode servir um arquivo de política de soquete.

Nota: *No Flash Player 9 e anterior, a metapolítica padrão de arquivos de política de URL era "all", o que significa que qualquer diretório pode conter um arquivo de política. Se você implantou aplicativos que carregam arquivos de política de locais diferentes do arquivo padrão /crossdomain.xml e se esses aplicativos estiverem executando agora no Flash Player 10, verifique se você (ou outro administrador do servidor) modificou o arquivo de política mestre para permitir arquivos de política adicionais. Para obter informações sobre como especificar uma metapolítica diferente, consulte o tópico "Alterações em arquivos de política do Flash Player 9" do Centro de desenvolvedores do Flash Player, em www.adobe.com/go/devnet_security_br.*

Um arquivo SWF pode verificar se existe um nome de arquivo de política diferente ou um local de diretório diferente chamando o método `Security.loadPolicyFile()`. No entanto, se o arquivo de política mestre não especificar que o local de destino pode servir arquivos de política, a chamada para `loadPolicyFile()` não terá nenhum efeito, mesmo que haja um arquivo de política naquele local. Chame `loadPolicyFile()` antes de tentar qualquer operação de rede que exija o arquivo de política. O Flash Player enfileira automaticamente solicitações de rede por trás de tentativas do arquivo de política correspondente. Portanto, é aceitável, por exemplo, chamar `Security.loadPolicyFile()` imediatamente antes de iniciar uma operação de rede.

Ao verificar um arquivo de política mestre, o Flash Player aguarda três segundos por uma resposta do servidor. Se uma resposta não for recebida, o Flash Player assumirá que não existe nenhum arquivo de política. No entanto não há nenhum valor de tempo limite padrão para chamadas para `loadPolicyFile()`. O Flash Player assume que o arquivo que está chamando existe, e aguarda quanto tempo for necessário para carregá-lo. Portanto, para verificar se um arquivo de política mestre está carregado, use `loadPolicyFile()` para chamá-lo explicitamente.

Mesmo que o método seja denominado `Security.loadPolicyFile()`, um arquivo de política não será carregado até que uma chamada de rede que exija um arquivo de política seja emitida. Chamadas para `loadPolicyFile()` simplesmente informam ao Flash Player onde procurar por arquivos de política quando eles são necessários.

Não é possível receber notificação de quando uma solicitação de arquivo de política é iniciada ou concluída e não há motivo para isso. O Flash Player executa verificações de políticas assincronamente e aguarda automaticamente para iniciar conexões até após o êxito das verificações do arquivo de política.

As seções a seguir contêm informações que se aplicam apenas a arquivos de política de URL. Para obter mais informações sobre arquivos de política de soquete, consulte "Conexão a soquetes" na página 727.

Escopo do arquivo de política de URL

O arquivo de política de URL se aplica apenas ao diretório do qual ele é carregado e a seus diretórios filho. Um arquivo de política no diretório raiz se aplica ao todo o servidor. Um arquivo de política carregado de um subdiretório arbitrário aplica-se apenas àquele diretório e a seus subdiretórios.

Um arquivo de política afeta o acesso somente ao servidor específico onde ele reside. Por exemplo, um arquivo de política localizado em https://www.adobe.com:8080/crossdomain.xml aplica-se apenas a chamadas de carregamento de dados feitas para www.adobe.com por HTTPS na porta 8080.

Especificação de permissões de acesso em um arquivo de política de URL

Um arquivo de política contém uma única tag `<cross-domain-policy>` que, por sua vez, contém zero ou mais tags `<allow-access-from>`. Cada tag `<allow-access-from>` contém um atributo, `domain`, que especifica um endereço IP exato, um domínio exato ou um domínio curinga (qualquer domínio). Os domínios curinga são indicados de duas maneiras:

- Por um único asterisco (*), que corresponde a todos os domínios e a todos os endereços IP.
- Por um asterisco seguido por um sufixo, que corresponde apenas aos domínios que terminam com o sufixo especificado.

Sufixos devem começar com um ponto. No entanto domínios curinga com sufixos podem corresponder a domínios que consistem apenas no sufixo sem o ponto inicial. Por exemplo, xyz.com é considerado como parte de *.xyz.com. Curingas não são permitidos em especificações de domínios IP.

O exemplo a seguir mostra um arquivo de política de URL que permite acesso a arquivos SWF originados de *.example.com, www.friendOfExample.com e 192.0.34.166:

```
<?xml version="1.0"?>
<cross-domain-policy>
    <allow-access-from domain="*.example.com" />
    <allow-access-from domain="www.friendOfExample.com" />
    <allow-access-from domain="192.0.34.166" />
</cross-domain-policy>
```

Se você especificar um endereço IP, o acesso será concedido somente a arquivos SWF carregados daquele endereço IP usando a sintaxe IP (por exemplo, http://65.57.83.12/flashmovie.swf). O acesso não é concedido a arquivos SWF que usam a sintaxe de nome de domínio. O Flash Player não executa resolução de DNS.

É possível permitir acesso a documentos originários de qualquer domínio, conforme mostrado no exemplo a seguir:

```
<?xml version="1.0"?>
<!-- http://www.foo.com/crossdomain.xml -->
<cross-domain-policy>
<allow-access-from domain="*" />
</cross-domain-policy>
```

Cada tag `<allow-access-from>` também tem o atributo opcional `secure` que é padronizado como `true`. Se o arquivo de política estiver em um servidor HTTPS e você quiser permitir que arquivos SWF em um servidor não-HTTPS carreguem dados do servidor HTTPS, poderá definir o atributo como `false`.

A configuração do atributo `secure` como `false` pode comprometer a segurança oferecida pelo HTTPS. Especificamente, a configuração desse atributo como `false` abre conteúdo de segurança a ataques por falsificação. A Adobe recomenda enfaticamente que você não defina o atributo `secure` como `false`.

Se os dados a serem carregados estiverem em um servidor HTTPS, mas o carregamento do arquivo SWF estiver em um servidor HTTP, a Adobe recomenda mover o arquivo SWF que está sendo carregado para um servidor HTTPS. Fazer isso permite manter todas as cópias de seus dados seguras sob a proteção de HTTPS. No entanto, se você decidir que deve manter o arquivo SWF que está sendo carregado em um servidor HTTP, adicione o atributo `secure="false"` à tag `<allow-access-from>`, conforme mostrado no código a seguir:

```
<allow-access-from domain="www.example.com" secure="false" />
```

Outro elemento que pode ser usado para permitir acesso é a tag `allow-http-request-headers-from`. Esse elemento concede a um cliente, que hospeda conteúdo de outro domínio, permissão para enviar cabeçalhos definidos pelo usuário ao seu domínio. Enquanto a tag `<allow-access-from>` concede a outros domínios permissão para enviar dados ao seu domínio, a tag `allow-http-request-headers-from` concede a outros domínios permissão para enviar dados a seu domínio na forma de cabeçalhos. No exemplo a seguir, qualquer domínio tem permissão para enviar o cabeçalho SOAPAction ao domínio atual:

```
<cross-domain-policy>
    <allow-http-request-headers-from domain="*" headers="SOAPAction"/>
</cross-domain-policy>
```

Se a instrução `allow-http-request-headers-from` estiver no arquivo de política mestre, ela será aplicada a todos os diretórios no host. Caso contrário, ela será aplicada apenas ao diretório e subdiretórios do arquivo de política que contêm a instrução.

Pré-carregamento de arquivos de política

O carregamento de dados de um servidor ou a conexão a um soquete é uma operação assíncrona. O Flash Player simplesmente aguarda que o download do arquivo de política seja concluído antes de iniciar a operação principal. No entanto, a extração de dados de pixels de imagens ou a extração de dados de amostra de sons é uma operação síncrona. O arquivo de política deve ser carregado para que os dados possam ser extraídos. Ao carregar a mídia, especifique que ela verifique a existência de um arquivo de política:

- Ao usar o método `Loader.load()`, defina a propriedade `checkPolicyFile` do parâmetro `context`, que é um objeto LoaderContext.
- Ao incorporar uma imagem em um arquivo de texto usando a tag `<img>`, defina o atributo `checkPolicyFile` da tag `<img>` como `"true"`, da seguinte maneira:

  ```
  <img checkPolicyFile = "true" src = "example.jpg">
  ```

- Ao usar o método `Sound.load()`, defina a propriedade `checkPolicyFile` do parâmetro `context`, que é um objeto SoundLoaderContext.
- Ao usar a classe NetStream, defina a propriedade `checkPolicyFile` do objeto NetStream.

Ao definir um desses parâmetros, o Flash Player primeiro verifica se há qualquer arquivo de política que já foi baixado naquele domínio. Em seguida, ele procura o arquivo de política no local padrão do servidor, verificando as instruções `<allow-access-from>` e a presença de uma metapolítica. Finalmente, ele considera quaisquer chamadas pendentes para o método `Security.loadPolicyFile()` para verificar se elas estão em escopo.

Controles de autor (desenvolvedor)

A API ActionScript principal usada para conceder privilégios de segurança é o método `Security.allowDomain()` que concede privilégios a arquivos SWF nos domínios especificados por você. No exemplo a seguir, um arquivo SWF concede acesso a arquivos SWF servidos no domínio www.example.com:

```
Security.allowDomain("www.example.com")
```

Esse método concede permissões para o seguinte:

- Cross-scripting entre arquivos SWF (consulte “Cross-scripting” na página 721)
- Acesso à lista de exibição (consulte “Como percorrer a lista de exibição” na página 723)
- Detecção de eventos (consulte “Segurança de eventos” na página 724)
- Acesso total à propriedades e métodos do objeto Stage (consulte“segurança de Palco” na página 722)

O objetivo principal da chamada do método `Security.allowDomain()` é conceder permissão para arquivos SWF em um domínio externo para executar script do arquivo SWF chamando o método `Security.allowDomain()`. Para obter mais informações, consulte "Cross-scripting" na página 721.

Especificar um endereço IP como parâmetro do método `Security.allowDomain()` não permite acesso de todas as partes que se originam no endereço IP especificado. Ao contrário, isso permite o acesso apenas de uma parte que contém o endereço IP especificado como sua URL, em vez de um nome de domínio mapeia para o endereço IP. Por exemplo, se o nome do domínio www.example.com for mapeado para o endereço IP 192.0.34.166, uma chamada para `Security.allowDomain("192.0.34.166")` não concederá acesso a www.example.com.

É possível passar o curinga `"*"` para o método `Security.allowDomain()` para permitir acesso a todos os domínios. Como ele concede permissão para arquivos SWF de *todos* os domínios para executarem script na chamada do arquivo, use o curinga `"*"` com cuidado.

O ActionScript inclui uma segunda API de permissão, chamada `Security.allowInsecureDomain()`. Esse método faz a mesma coisa que o método `Security.allowDomain()`, exceto que, quando chamado de um arquivo SWF servido por uma conexão HTTPS segura, ele permite adicionalmente acesso ao arquivo SWF de chamada por outros arquivos SWF que são servidos em um protocolo inseguro, como HTTP. No entanto, não é uma boa prática de segurança permitir script entre arquivos de um protocolo seguro (HTTPS) e arquivos de protocolos inseguros (como HTTP). Isso pode abrir conteúdo seguro a ataques de falsificação. Esses ataques podem ocorrer da seguinte maneira: como o método `Security.allowInsecureDomain()` permite acesso aos dados de HTTPS seguro por arquivos SWF servidos em conexões HTTP, um invasor situado entre o servidor HTTP e seus usuários pode substituir seu arquivo SWF de HTTP por um arquivo próprio, que pode então acessar seus dados HTTPS.

Outro método importante relacionado à segurança é o método `Security.loadPolicyFile()` que faz com que o Flash Player verifique se há um arquivo de política em um local não padrão. Para obter mais informações, consulte "Controles de site (arquivos de política)" na página 711.

Restrição de APIs de rede

APIs de rede podem ser restringidas de duas maneiras. Para impedir atividades mal-intencionadas, o acesso a portas comumente reservadas é bloqueado. Você não pode substituir esses bloqueios em seu código. Para controlar o acesso de um arquivo SWF à funcionalidade da rede com relação a outras portas, é possível usar a configuração `allowNetworking`.

Portas bloqueadas

O Flash Player e o Adobe AIR têm restrições sobre o acesso HTTP a determinadas portas, assim como navegadores. Solicitações HTTP não são permitidas para determinadas portas padrão que convencionalmente são usadas para tipos de servidores não-HTTP.

Qualquer API que acesse uma URL da rede está sujeita a essas restrições de bloqueio de porta. A única exceção são APIs que chamam soquetes diretamente, como `Socket.connect()` e `XMLSocket.connect()`, ou chamadas para `Security.loadPolicyFile()` nas quais um arquivo de política de soquete está sendo carregado. Conexões de soquete são permitidas ou negadas por meio do uso de arquivos de política de soquete no servidor de destino.

A lista a seguir mostra as APIs do ActionScript 3.0 às quais se aplica o bloqueio de portas:

`FileReference.download()`, `FileReference.upload()`, `Loader.load()`, `Loader.loadBytes()`, `navigateToURL()`, `NetConnection.call()`, `NetConnection.connect()`, `NetStream.play()`, `Security.loadPolicyFile()`, `sendToURL()`, `Sound.load()`, `URLLoader.load()`, `URLStream.load()`

O bloqueio de portas também se aplica à importação da Biblioteca compartilhada, ao uso da tag `<img>` em campos de texto e ao carregamento de arquivos SWF em uma página HTML usando as tags `<object>` e`<embed>`.

As listas a seguir mostram quais portas são bloqueadas:

HTTP: 20 (dados ftp), 21 (controle ftp)

HTTP e FTP: 1 (tcpmux), 7 (echo), 9 (discard), 11 (systat), 13 (daytime), 15 (netstat), 17 (qotd), 19 (chargen), 22 (ssh), 23 (telnet), 25 (smtp), 37 (time), 42 (name), 43 (nicname), 53 (domain), 77 (priv-rjs), 79 (finger), 87 (ttylink), 95 (supdup), 101 (hostriame), 102 (iso-tsap), 103 (gppitnp), 104 (acr-nema), 109 (pop2), 110 (pop3), 111 (sunrpc), 113 (auth), 115 (sftp), 117 (uucp-path), 119 (nntp), 123 (ntp), 135 (loc-srv / epmap), 139 (netbios), 143 (imap2), 179 (bgp), 389 (ldap), 465 (smtp+ssl), 512 (print / exec), 513 (login), 514 (shell), 515 (printer), 526 (tempo), 530 (courier), 531 (chat), 532 (netnews), 540 (uucp), 556 (remotefs), 563 (nntp+ssl), 587 (smtp), 601 (syslog), 636 (ldap+ssl), 993 (ldap+ssl), 995 (pop3+ssl), 2049 (nfs), 4045 (lockd), 6000 (x11)

Uso do parâmetro allowNetworking

É possível controlar o acesso de um arquivo SWF à funcionalidade da rede definindo o parâmetro `allowNetworking` nas tags `<object>` e`<embed>` na página HTML que contém o conteúdo do SWF.

Os valores possíveis de `allowNetworking` são:

- `"all"` (o padrão) — todas as APIs de rede têm permissão no arquivo SWF.
- `"internal"` — o arquivo SWF não pode chamar APIs de interação de navegador ou navegação de navegador, listadas posteriormente nesta seção, mas ele pode chamar quaisquer outras APIs de rede.
- `"none"` — o arquivo SWF não pode chamar APIs de interação de navegador ou de navegação de navegador, listadas posteriormente nesta seção, e não pode usar quaisquer APIs de comunicação SWF com SWF, também listadas posteriormente.

O parâmetro `allowNetworking` foi projetado para uso principalmente quando o arquivo SWF e a página HTML que o delimita estão em domínios diferentes. O uso do valor de `"internal"` ou de `"none"` não é recomendado quando o arquivo SWF que está sendo carregado é do mesmo domínio que as páginas HTML que o delimitam, porque não é possível garantir que um arquivo SWF sempre seja carregado com a página HTML pretendida. Partes não confiáveis podem carregar um arquivo SWF do seu domínio sem HTML delimitado, em cujo caso a restrição `allowNetworking` não funcionará como pretendido.

A chamada de uma API impedida lança uma exceção SecurityError.

Adicione o parâmetro `allowNetworking` e defina seu valor nas tags `<object>` e `<embed>` da página HTML que contém uma referência ao arquivo SWF, conforme mostrado neste exemplo:

```
<object classic="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
    Code
base="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,124,
0"
    width="600" height="400" ID="test" align="middle">
<param name="allowNetworking" value="none" />
<param name="movie" value="test.swf" />
<param name="bgcolor" value="#333333" />
<embed src="test.swf" allowNetworking="none" bgcolor="#333333"
    width="600" height="400"
    name="test" align="middle" type="application/x-shockwave-flash"
    pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

Uma página HTML também pode usar um script para gerar tags de incorporação SWF. É necessário alterar o script para que ele insira as configurações corretas de `allowNetworking`. As páginas HTML geradas pelo Flash e pelo Adobe Flex Builder usam a função `AC_FL_RunContent()` para incorporar referências aos arquivos SWF. Adicione as configurações do parâmetro `allowNetworking` ao script, como no seguinte:

```
AC_FL_RunContent( ... "allowNetworking", "none", ...)
```

As seguintes APIs são impedidas quando `allowNetworking` está definido como `"internal"`:

`navigateToURL()`, `fscommand()`, `ExternalInterface.call()`

Além das APIs da lista anterior, as seguintes APIs também são impedidas quando o parâmetro `allowNetworking` está definido como `"none"`:

`sendToURL()`, `FileReference.download()`, `FileReference.upload()`, `Loader.load()`, `LocalConnection.connect()`, `LocalConnection.send()`, `NetConnection.connect()`, `NetStream.play()`, `Security.loadPolicyFile()`, `SharedObject.getLocal()`, `SharedObject.getRemote()`, `Socket.connect()`, `Sound.load()`, `URLLoader.load()`, `URLStream.load()`, `XMLSocket.connect()`

Mesmo que a configuração `allowNetworking` selecionada permita que um arquivo SWF use uma API de rede, pode haver outras restrições com base nas limitações da caixa de proteção de segurança (consulte "Caixas de proteção de segurança" na página 706).

Quando `allowNetworking` está definido como `"none"`, não é possível fazer referência a mídia externa em uma tag `<img>` na propriedade `htmlText` de um objeto TextField (uma exceção SecurityError é lançada).

Quando `allowNetworking` está definido como `"none"`, um símbolo de uma biblioteca compartilhada importada adicionado na ferramenta de autoria do Flash (não no ActionScript) é bloqueado em tempo de execução.

Segurança de modo de tela cheia

O Flash Player 9.0.27.0 e versões posteriores oferecem suporte ao modo de tela cheia, no qual o conteúdo executado no Flash Player pode ocupar toda a tela. Para entrar em modo de tela cheia, a propriedade `displayState` do Palco é definida como a constante `StageDisplayState.FULL_SCREEN`. Para obter mais informações, consulte "Configuração de propriedades do palco" na página 287.

Para arquivos SWF executados em um navegador, há mais considerações sobre segurança.

Para ativar o modo de tela cheia, nas tags `<object>` e `<embed>` na página HTML que contém uma referência ao arquivo SWF, adicione o parâmetro `allowFullScreen` com o valor definido como `"true"` (o valor padrão é `"false"`), conforme mostrado no exemplo a seguir:

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"

codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,
18,0"
    width="600" height="400" id="test" align="middle">
<param name="allowFullScreen" value="true" />
<param name="movie" value="test.swf" />
<param name="bgcolor" value="#333333" />
<embed src="test.swf" allowFullScreen="true" bgcolor="#333333"
    width="600" height="400"
    name="test" align="middle" type="application/x-shockwave-flash"
    pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

Uma página HTML também pode usar um script para gerar tags de incorporação SWF. É necessário alterar o script para que ele insira as configurações corretas de `allowFullScreen`. As páginas HTML geradas pelo Flash e pelo Flex Builder usam a função `AC_FL_RunContent()` para incorporar referências a arquivos SWF e você precisa adicionar as configurações do parâmetro `allowFullScreen`, conforme a seguir:

```
AC_FL_RunContent( ... "allowFullScreen", "true", ...)
```

O ActionScript que inicia o modo de tela cheia pode ser chamado apenas em resposta a um evento do mouse ou do teclado. Se ele for chamado em outras situações, o Flash Player lançará uma exceção.

Um mensagem é exibida quando o conteúdo entra em modo de tela cheia, fornecendo ao usuário instruções sobre como sair e retornar ao modo normal. A mensagem é exibida por alguns segundos e, em seguida, desaparece.

No caso de conteúdo executado em um navegador, o uso do teclado fica restrito no modo de tela cheia. No Flash Player 9, só é dado suporte para atalhos de teclado que retornam o aplicativo ao modo normal, como pressionar a tecla Escape. Os usuários não podem inserir texto em campos de texto ou navegar pela tela. No Flash Player 10 e versões posteriores, há suporte para teclas que não são impressas (como as teclas de seta, espaço, Shift e Tab). No entanto, a entrada de texto ainda não é permitida.

O modo de tela cheia sempre é permitido no player independente ou em um arquivo de projetor. Além disso, o uso do teclado (inclusive para entrada de texto) é totalmente suportado nesses ambientes.

Chamar a propriedade `displayState` de um objeto Stage lança uma exceção para qualquer chamador que não esteja na mesma caixa de proteção de segurança que o proprietário do Palco (o arquivo SWF principal). Para obter mais informações, consulte "segurança de Palco" na página 722.

Os administradores podem desativar o modo de tela cheia de arquivos SWF em execução em navegadores configurando `FullScreenDisable = 1` no arquivo mms.cfg. Para obter detalhes, consulte "Controles de administrador" na página 708.

Em um navegador, um arquivo SWF deve ser contido em uma página HTML para permitir o modo de tela cheia.

Carregamento de conteúdo

Um arquivo SWF pode carregar os seguintes tipos de conteúdo:

- Arquivos SWF
- Imagens
- Som
- Vídeo

Carregamento de arquivos SWF e de imagens

Use a classe Loader para carregar arquivos SWF e imagens (arquivos JPG, GIF ou PNG). Qualquer arquivo SWF, a não ser um que esteja na caixa de proteção local com arquivos do sistema, pode carregar arquivos SWF e imagens de qualquer domínio de rede. Apenas arquivos SWF em caixas de proteção locais podem carregar arquivos SWF e imagens do sistema de arquivos local. No entanto arquivos na caixa de proteção local com rede podem carregar apenas arquivos SWF locais que estejam na mesma caixa de proteção confiável local ou local com rede. Os arquivos SWF na caixa de proteção local com rede carregam conteúdo local que não sejam arquivos SWF (como imagens), no entanto eles não podem acessar dados no conteúdo carregado.

Ao carregar um arquivo SWF a partir de uma origem não confiável (como um domínio diferente daquele do arquivo SWF da raiz do objeto Loader), convém definir uma máscara para o objeto Loader para impedir que o conteúdo carregado (que é filho do objeto Loader) seja desenhado em partes do Palco fora daquela máscara, conforme mostrado no código a seguir:

```
import flash.display.*;
import flash.net.URLRequest;
var rect:Shape = new Shape();
rect.graphics.beginFill(0xFFFFFF);
rect.graphics.drawRect(0, 0, 100, 100);
addChild(rect);
var ldr:Loader = new Loader();
ldr.mask = rect;
var url:String = "http://www.unknown.example.com/content.swf";
var urlReq:URLRequest = new URLRequest(url);
ldr.load(urlReq);
addChild(ldr);
```

Ao chamar o método `load()` do objeto Loader, é possível especificar um parâmetro `context` que é um objeto LoaderContext. A classe LoaderContext inclui três propriedades que permitem definir o contexto de como o conteúdo carregado pode ser usado:

- `checkPolicyFile`: Use essa propriedade apenas ao carregar um arquivo de imagem (não um arquivo SWF). Especifique-a para um arquivo de imagem de um domínio diferente daquele do arquivo que contém o objeto Loader. Se você definir a propriedade como `true`, o Loader verificará o servidor de origem para obter um arquivo de política de URL (consulte"Controles de site (arquivos de política)" na página 711). Se o servidor conceder permissão ao domínio Loader, o ActionScript de arquivos SWF no domínio Loader poderá acessar dados na imagem carregada. Em outras palavras, você pode usar a propriedade `Loader.content` para obter uma referência ao objeto Bitmap que representa a imagem carregada ou o método `BitmapData.draw()` para acessar pixels da imagem carregada.
- `securityDomain`: Só use essa propriedade ao carregar um arquivo SWF (não uma imagem). Especifique-a para um arquivo SWF de um domínio diferente daquele do arquivo que contém o objeto Loader. Apenas dois valores são suportados atualmente para a propriedade `securityDomain`: `null` (o padrão) e `SecurityDomain.currentDomain`. Se você especificar `SecurityDomain.currentDomain`, ele solicitará que o arquivo SWF carregado seja *importado* para a caixa de proteção do arquivo SWF que está sendo carregado, indicando que ele funcionará como se fosse carregado do próprio servidor do arquivo SWF que está sendo carregado. Isso será permitido apenas se um arquivo de política de URL for encontrado no servidor do arquivo SWF carregado, permitindo acesso pelo domínio do arquivo SWF que está sendo carregado. Se o arquivo de política for encontrado, o carregador e o carregado poderão executar script livremente um no outro assim que o carregamento for iniciado, uma vez que estão na mesma caixa de proteção. Observe que a importação da caixa de proteção pode ser substituída principalmente executando um carregamento comum e, em seguida, fazendo com que o arquivo SWF chame o método `Security.allowDomain()`. Esse último método pode ser mais fácil de usar, pois o arquivo SWF carregado estará em sua própria caixa de proteção natural e, portanto, capaz de acessar recursos de seu próprio servidor real.
- `applicationDomain`: Use essa propriedade apenas ao carregar um arquivo SWF gravado no ActionScript 3.0 (não em uma imagem ou arquivo SWF gravado no ActionScript 1.0 ou no ActionScript 2.0). Ao carregar o arquivo, é possível especificar se ele será colocado em um domínio de aplicativo específico, em vez de ser colocado como padrão em um novo domínio de aplicativo que é filho do domínio do aplicativo do arquivo SWF que está sendo carregado. Observe que os domínios de aplicativos são subunidades de domínios de segurança e portanto você poderá especificar um domínio de aplicativo de destino apenas se o arquivo SWF que está sendo carregado for de

seu próprio domínio de segurança, seja porque ele é de seu próprio servidor ou porque você o importou com êxito em seu domínio de segurança usando a propriedade `securityDomain`. Se você especificar um domínio de aplicativo, mas o arquivo SWF carregado fizer parte de um domínio de segurança diferente, o domínio especificado em `applicationDomain` será ignorado. Para obter mais informações, consulte "Uso da classe ApplicationDomain" na página 657.

Para obter detalhes, consulte "Especificação do contexto do carregamento" na página 315.

Uma propriedade importante de um objeto Loader é a propriedade `contentLoaderInfo` que é um objeto LoaderInfo. Ao contrário da maioria dos outros objetos, um objeto LoaderInfo é compartilhado entre o arquivo SWF que está sendo carregado e o conteúdo carregado e sempre está acessível para as duas partes. Quando o conteúdo carregado é um arquivo SWF, ele pode acessar o objeto LoaderInfo por meio da propriedade `DisplayObject.loaderInfo`. Objetos LoaderInfo incluem informações, como o progresso do carregamento, as URLs do carregador e do carregado, o relacionamento de confiança entre o carregador e o carregado e outras informações. Para obter mais informações, consulte "Monitoramento do progresso do carregamento" na página 314.

Carregamento de som e vídeo

Todos os arquivos SWF, além dos que estão na caixa de proteção local com sistema de arquivos, têm permissão para carregar som e vídeo de origens da rede, usando os métodos `Sound.load()`, `NetConnection.connect()` e `NetStream.play()` .

Apenas arquivos SWF locais podem carregar mídia do sistema de arquivos local. Apenas arquivos SWF na caixa de proteção local com sistema de arquivos ou na caixa de proteção local confiável podem acessar dados nesses arquivos carregados.

Há outras restrições ao acessar dados de mídia carregada. Para obter detalhes, consulte "Acesso à mídia carregada como dados" na página 724.

Carregamento de arquivos SWF e de imagens usando a tag <img> em um campo de texto

É possível carregar arquivos SWF e bitmaps em um campo de texto usando a tag `<img>`, conforme mostrado no código a seguir:

```
<img src = 'filename.jpg' id = 'instanceName' >
```

É possível carregar conteúdo carregado dessa maneira usando o método `getImageReference()` da ocorrência TextField, conforme mostrado no código a seguir:

```
var loadedObject:DisplayObject = myTextField.getImageReference('instanceName');
```

No entanto observe que arquivos SWF e imagens carregados dessa maneira são colocados na caixa de proteção que corresponde a sua origem.

Ao carregar um arquivo de imagem usando uma tag `<img>` em um campo de texto, o acesso aos dados da imagem pode ser permitido por um arquivo de política de URL. É possível verificar se há um arquivo de política adicionando um atributo `checkPolicyFile` à tag `<img>`, como no código a seguir:

```
<img src = 'filename.jpg' checkPolicyFile = 'true' id = 'instanceName' >
```

Ao carregar um SWF usando uma tag `<img>` em um campo de texto, é possível permitir acesso aos dados daquele arquivo SWF por meio de uma chamada ao método `Security.allowDomain()`.

Ao usar uma tag `<img>` em um campo de texto para carregar um arquivo externo (em vez de usar uma classe Bitmap incorporada em seu SWF), um objeto Loader é criado automaticamente como um filho do objeto TextField, e o arquivo externo é carregado naquele Loader exatamente como se você tivesse usado um objeto Loader no ActionScript para carregar o arquivo. Nesse caso, o método `getImageReference()` retorna o Loader que foi criado automaticamente. Nenhuma verificação de segurança é necessária para acessar esse objeto Loader porque ele está na mesma caixa de proteção de segurança que o objeto de chamada.

No entanto quando você faz referência à propriedade `content` do objeto Loader para acessar a mídia carregada, regras de segurança são aplicadas. Se o conteúdo for uma imagem, será necessário implementar um arquivo de política de URL e, se o conteúdo for um arquivo SWF, você precisará fazer com que o código no arquivo SWF chame o método `allowDomain()`.

Conteúdo entregue usando servidores RMTP

O Flash Media Server usa o protocolo RTMP para servir dados, áudio e vídeo. Um arquivo SWF carrega essa mídia usando o método `connect()` da classe NetConnection, passando uma URL de RTMP como o parâmetro. O Flash Media Server pode restringir conexões e impedir que conteúdo seja baixado, com base no domínio do arquivo solicitante. Para obter detalhes, consulte a documentação do Flash Media Server.

Para mídia baixada de origens RTMP, não é possível usar os métodos `BitmapData.draw()` e `SoundMixer.computeSpectrum()` para extrair gráficos e dados de som em tempo de execução.

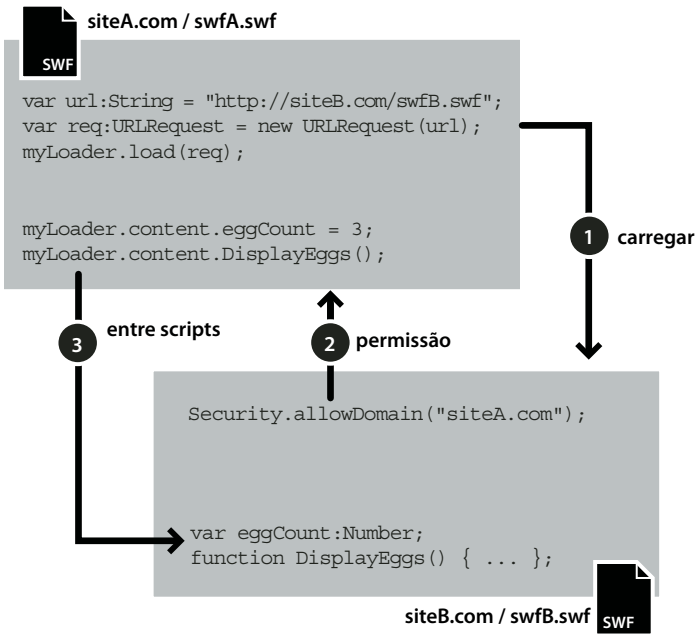
Cross-scripting

Se dois arquivos SWF escritos com o ActionScript 3.0 forem servidos no mesmo domínio, por exemplo, se a URL de um arquivo SWF for http://www.example.com/swfA.swf e a URL do outro for http://www.example.com/swfB.swf, um arquivo SWF poderá examinar e modificar variáveis, objetos, propriedades, métodos e assim por diante na ordem e vice-versa. Isso é chamado de *cross-scripting*.

O cross-scripting não é suportado entre arquivos SWF AVM1 e SWF AVM2. Um arquivo SWF AVM1 é criado usando o ActionScript 1.0 ou o ActionScript 2.0. (AVM1 e AVM2 se referem a ActionScript Virtual Machine.) No entanto é possível usar a classe LocalConnection para enviar dados entre o AVM1 e o AVM2.

Se dois arquivos SWF forem servidos em domínios diferentes, por exemplo, http://siteA.com/swfA.swf e http://siteB.com/siteB.swf, por padrão, o Flash Player não permitirá que o swfA.swf execute script do swfB.swf, nem que o swfB.swf execute script do swfA.swf. O arquivo SWF dá permissão a arquivos SWF de outros domínios chamando `Security.allowDomain()`. Com a chamada de `Security.allowDomain("siteA.com")`, o swfB.swf concede aos arquivos SWF do siteA.swf permissão para executar script nele.

Em qualquer situação entre domínios, é importante que as duas partes envolvidas sejam bem-definidas. Para fins desta discussão, o lado que executa o cross-scripting é chamado de *parte de acesso* (normalmente o SWF que faz o acesso) e o outro lado é chamado de *a parte que está sendo acessada* (normalmente o SWF que está sendo acessado). Quando o siteA.swf executa script no siteB.swf, o siteA.swf é a parte que acessa e o siteB.swf é a parte que está sendo acessada, conforme mostrado na ilustração a seguir:



Permissões entre domínios estabelecidas com o método`Security.allowDomain()` são assimétricas. No exemplo anterior, o siteA.swf pode executar script no siteB.swf, mas o siteB.swf não pode executar script no siteA.swf, pois o siteA.swf não chamou o método `Security.allowDomain()` para conceder permissão aos arquivos SWF no siteB.com para executar script nele. É possível configurar permissões simétricas fazendo com que os dois arquivos SWF chamem o método `Security.allowDomain()`.

Além de proteger arquivos SWF contra a execução de script entre domínios originadas por outros arquivos SWF, o Flash Player protege arquivos SWF conta a execução de script entre domínios originadas por arquivos HTML. A execução de script de HTML para SWF pode ocorrer com retornos de chamada estabelecidos por meio do método `ExternalInterface.addCallback()`. Quando a execução de script de HTML para SWF cruza domínios, o arquivo SWF que está sendo acessado deve chamar o método `Security.allowDomain()` exatamente como quando a parte que faz o acesso é um arquivo SWF ou haverá falha na operação. Para obter mais informações, consulte "Controles de autor (desenvolvedor)" na página 714.

Além disso, o Flash Player fornece controles de segurança para execução de script de SWF para HTML. Para obter mais informações, consulte "Controle do acesso à URL de saída" na página 731.

segurança de Palco

Algumas propriedades e métodos do objeto Stage estão disponíveis para qualquer entidade gráfica ou clipe de filme na lista de exibição.

No entanto o objeto Stage é considerado como tendo um proprietário: o primeiro arquivo SWF carregado. Por padrão, as seguintes propriedades e métodos do objeto Stage estão disponíveis apenas para arquivos SWF na mesma caixa de proteção de segurança que o proprietário do Palco:

Propriedades	Métodos
`alinhar`	`addChild()`
displayState	`addChildAt()`
frameRate	`addEventListener()`
`height`	`dispatchEvent()`
`mouseChildren`	`hasEventListener()`
`numChildren`	`setChildIndex()`
`quality`	`willTrigger()`
`scaleMode`	
`showDefaultContextMenu`	
`stageFocusRect`	
`stageHeight`	
`stageWidth`	
`tabChildren`	
`textSnapshot`	
`width`	

Para que um arquivo SWF em uma caixa de proteção diferente daquela do proprietário do Palco acesse essas propriedades e métodos, o arquivo SWF do proprietário do Palco deve chamar o método `Security.allowDomain()` para permitir o domínio da caixa de proteção externa. Para obter mais informações, consulte "Controles de autor (desenvolvedor)" na página 714.

A propriedade `frameRate` é um caso especial, qualquer arquivo SWF pode ler a propriedade `frameRate`. No entanto, apenas aqueles que estão na caixa de proteção de segurança do proprietário do Palco (ou aqueles que receberam permissão por uma chamada para o método `Security.allowDomain()`) podem alterar a propriedade.

Também há restrições nos métodos `removeChildAt()` e `swapChildrenAt()` do objeto Stage, mas essas são diferentes das outras restrições. E vez de ser necessário estar no mesmo domínio que o proprietário do Palco para chamar esses métodos, o código deve estar no mesmo domínio que o proprietário dos objetos filho afetados ou os objetos filho podem chamar o método `Security.allowDomain()`.

Como percorrer a lista de exibição

A capacidade de um arquivo SWF acessar objetos de exibição carregados de outras caixas de proteção é restrita. Para que um arquivo SWF acesse um objeto de exibição criado por outro arquivo SWF em uma caixa de proteção diferente, o arquivo SWF que está sendo acessado deve chamar o método `Security.allowDomain()` para permitir acesso pelo domínio do arquivo SWF que está acessando. Para obter mais informações, consulte "Controles de autor (desenvolvedor)" na página 714.

Para acessar um objeto Bitmap que foi carregado por um objeto Loader, um arquivo de política de URL deve existir no servidor de origem do arquivo de imagem, e esse arquivo de política deve conceder permissão ao domínio do arquivo SWF que está tentando acessar o objeto Bitmap (consulte "Controles de site (arquivos de política)" na página 711).

O objeto LoaderInfo que corresponde a um arquivo carregado (e ao objeto Loader) inclui as três seguintes propriedades que definem o relacionamento entre o objeto carregado e o objeto Loader: `childAllowsParent`, `parentAllowsChild` e `sameDomain`.

Segurança de eventos

Eventos relacionados à lista de exibição têm limitações de acesso de segurança com base na caixa de proteção do objeto de exibição está despachando o evento. Um evento na lista de exibição tem fases de bubbling e de captura (descritas em "Manipulação de eventos" na página 251). Durante as fases de bubbling e de captura, um evento migra do objeto de exibição de origem por meio de objetos de exibição pai na lista de exibição. Se um objeto pai estiver em uma caixa de proteção de segurança diferente do objeto de exibição de origem, a fase de captura e de bubble parará abaixo daquele objeto pai, a menos que haja confiança mútua entre o proprietário do objeto pai e o proprietário do objeto de origem. Essa confiança mútua pode ser obtida pelo seguinte:

1. O arquivo SWF que possui o objeto pai deve chamar o método `Security.allowDomain()` para confiar no domínio do arquivo SWF que possui o objeto de origem.
2. O arquivo SWF que possui o objeto de origem deve chamar o método `Security.allowDomain()` para confiar no domínio do arquivo SWF que possui o objeto pai.

O objeto LoaderInfo que corresponde a um arquivo carregado (e ao objeto Loader) inclui as duas seguintes propriedades que definem o relacionamento entre o objeto carregado e o objeto Loader: `childAllowsParent` e `parentAllowsChild`.

Para eventos despachados de objetos que não são objetos de exibição, não há verificações de segurança ou implicações relacionadas à segurança.

Acesso à mídia carregada como dados

Os dados carregados são acessados usando métodos, como `BitmapData.draw()` e `SoundMixer.computeSpectrum()`. Por padrão, um arquivo SWF de uma caixa de proteção de segurança não pode obter dados de pixels ou dados de áudio do gráfico ou de objetos de áudio renderizados ou reproduzidos pela mídia carregada em outra caixa de proteção. No entanto é possível usar os métodos a seguir para conceder essa permissão:

- Em um arquivo SWF carregado, chame o método `Security.allowDomain()` para conceder acesso a dados a arquivos SWF em outros domínios.
- Para uma imagem, som ou vídeo carregado, adicione um arquivo de política de URL no servidor do arquivo carregado. Esse arquivo de política deve conceder acesso ao domínio do arquivo SWF que está tentando chamar os métodos `BitmapData.draw()` ou `SoundMixer.computeSpectrum()` para extrair dados do arquivo.

As seções a seguir fornecem detalhes sobre o acesso a dados de bitmap, som e vídeo.

Acesso a dados de bitmap

O método `draw()` de um objeto BitmapData permite desenhar os pixels exibidos atualmente de qualquer objeto de exibição para o objeto BitmapData. Isso pode incluir os pixels de um objeto MovieClip, de um objeto Bitmap ou de qualquer objeto de exibição. As seguintes condições devem ser atendidas para que o método `draw()` desenhe pixels para o objeto BitmapData:

- No caso de um objeto de origem diferente de um bitmap carregado, o objeto de origem e (no caso de um objeto Sprite ou MovieClip) todos os seus objetos filho devem vir do mesmo domínio que o objeto que está chamando o método `draw()`, ou devem estar em um arquivo SWF que possa ser acessado pelo chamador fazendo com que o método `Security.allowDomain()` seja chamado.
- No caso de um objeto de origem de bitmap Loaded, o objeto de origem deve vir do mesmo domínio que o objeto que está chamando o método `draw()`, ou seu servidor de origem deve incluir um arquivo de política de URL que conceda permissão ao domínio que está fazendo a chamada.

Se essas condições não forem atendidas, uma exceção SecurityError será lançada.

Ao carregar a imagem usando o método `load()` da classe Loader, você pode especificar um parâmetro `context` que é um objeto SoundLoaderContext. Se você definir a propriedade `checkPolicyFile` do objeto LoaderContext como `true`, o Flash Player verificará se há um arquivo de política de URL no servidor do qual a imagem é carregada. Se houver um arquivo de política e o arquivo permitir que o domínio do arquivo SWF que está sendo carregado, o arquivo terá permissão para acessar os dados no objeto Bitmap. Caso contrário, o acesso será negado.

Também é possível especificar uma propriedade `checkPolicyFile` em uma imagem carregada por meio de uma tag `<img>` em um campo de texto. Para obter detalhes, consulte "Carregamento de arquivos SWF e de imagens usando a tag <img> em um campo de texto" na página 720

Acesso a dados de som

As seguintes APIs do ActionScript 3.0 relacionadas a som têm restrições de segurança:

- O método `SoundMixer.computeSpectrum()` — sempre permitido para arquivos SWF que estão na mesma caixa de proteção de segurança que o arquivo de som. Para arquivos em outras caixas de proteção, há verificações de segurança.
- O método `SoundMixer.stopAll()` — sempre permitido para arquivos SWF que estão na mesma caixa de proteção de segurança que o arquivo de som. Para arquivos em outras caixas de proteção, há verificações de segurança.
- A propriedade `id3` da classe Sound — sempre permitida para arquivos SWF que estão na mesma caixa de proteção de segurança que o arquivo de som. Para arquivos em outras caixas de proteção, há verificações de segurança.

Todo evento tem dois tipos de caixas de proteção associadas a ele, uma caixa de proteção de conteúdo e uma caixa de proteção do proprietário:

- O domínio de origem do som determina a caixa de proteção de conteúdo e isso determina se os dados podem ser extraídos do som por meio da propriedade `id3` do som e do método `SoundMixer.computeSpectrum()`.
- O objeto que iniciou a reprodução do som determina a caixa de proteção do proprietário e isso determina se o som pode ser parado usando o método `SoundMixer.stopAll()`.

Ao carregar o som usando o método `load()` da classe Sound, você pode especificar um parâmetro `context` que é um objeto SoundLoaderContext. Se você definir a propriedade `checkPolicyFile` do objeto SoundLoaderContext como `true`, o Flash Player verificará se há um arquivo de política de URL no servidor do qual o som é carregado. Se houver um arquivo de política e o arquivo permitir o domínio do arquivo SWF que está sendo carregado, o arquivo receberá permissão para acessar a propriedade `id` do objeto Sound. Caso contrário, não terá essa permissão. Além disso, a configuração da propriedade `checkPolicyFile` pode ativar o método `SoundMixer.computeSpectrum()` para sons carregados.

É possível usar o método `SoundMixer.areSoundsInaccessible()` para descobrir se uma chamada para o método `SoundMixer.stopAll()` não parará todos os sons porque a caixa de proteção de um ou mais proprietários de som está inacessível para o chamador.

A chamada do método `SoundMixer.stopAll()` pára esses sons cuja caixa de proteção de proprietário é a mesma que a do chamador de `stopAll()`. Ela também pára aqueles sons cuja reprodução foi iniciada por arquivos SWF que chamaram o método `Security.allowDomain()` para permitir acesso pelo domínio do arquivo SWF que está chamando o método `stopAll()`. Qualquer outro som não é parado e a presença desses sons pode ser revelada chamando o método `SoundMixer.areSoundsInaccessible()`.

A chamada do método `computeSpectrum()` requer que todo som que esteja sendo reproduzido seja da mesma caixa de proteção que o objeto que está chamando o método ou de uma origem que recebeu permissão para a caixa de proteção do chamador. Caso contrário, uma exceção SecurityError é lançada. Para sons que foram carregados de sons incorporados em uma biblioteca em um arquivo SWF, a permissão será concedida com uma chamada para o método `Security.allowDomain()` no arquivo SWF carregado. Para sons carregados de origens que não sejam arquivos SWF (originários de arquivos mp3 carregados ou de arquivos de vídeo), um arquivo de política de URL no servidor de origem concede acesso a dados contidos na mídia carregada. Você não poderá usar o método `computeSpectrum()` se um som for carregado de fluxos RTMP.

Para obter mais informações, consulte“Controles de autor (desenvolvedor)” na página 714 e“Controles de site (arquivos de política)” na página 711.

Acesso a dados de vídeo

É possível usar o método `BitmapData.draw()` para capturar os dados de pixels do quadro atual de um vídeo.

Há dois tipos diferentes de vídeo:

- Vídeo RTMP
- Vídeo progressivo que é carregado de um arquivo FLV sem um servidor RTMP

Não é possível usar o método `BitmapData.draw()` para acessar vídeo RTMP.

Ao chamar o método `BitmapData.draw()` com vídeo progressivo como o parâmetro `source`, o chamador de `BitmapData.draw()` deve estar na mesma caixa de proteção que o arquivo FLV, ou o servidor do arquivo FLV deve ter um arquivo de política que conceda permissão para o domínio do arquivo SWF que está fazendo a chamada. É possível solicitar que o arquivo de política seja baixado configurando a propriedade `checkPolicyFile` do objeto NetStream como `true`.

Carregamento de dados

Os arquivos SWF podem carregar dados de servidores no ActionScript e enviar dados do ActionScript para servidores. O carregamento de dados é um tipo de operação diferente do carregamento de mídia porque as informações carregadas aparecem diretamente no ActionScript em vez de serem exibidas como mídia. Geralmente, os arquivos SWF podem carregar dados de seus próprios domínios. No entanto, eles costumam exigir arquivos de política para carregar dados de outros domínios (consulte "Controles de site (arquivos de política)" na página 711).

Uso de URLLoader e URLStream

É possível carregar dados, como um arquivo XML ou um arquivo de texto. Os métodos `load()` das classes URLLoader e URLStream são governados pelas permissões do arquivo de política de URL.

Se você usar o método `load()` para carregar conteúdo de um domínio diferente daquele do arquivo SWF que está chamando o método, o Flash Player verificará se existe um arquivo de política de URL no servidor dos ativos carregados. Se houver um arquivo de política e ele conceder acesso ao domínio do arquivo SWF que está sendo carregado, você poderá carregar os dados.

Conexão a soquetes

Por padrão, o Flash Player procura um arquivo de política de soquete servido na porta 843. Assim como os arquivos de política de URL, esse arquivo é chamado de *arquivo de política mestre*.

Quando arquivos de política foram introduzidos no Flash Player 6 pela primeira vez, não havia suporte para arquivos de política de soquete. Conexões a servidores de soquete eram autorizadas por um arquivo de política no local padrão em um servidor HTTP na porta 80 do mesmo host que o servidor de soquete. O Flash Player 9 ainda oferece suporte a essa capacidade, mas o Flash Player 10 não oferece. No Flash Player 10, apenas arquivos de política de soquete podem autorizar conexões de soquete.

Como os arquivos de política de URL, os arquivos de política de soquete oferecem suporte a uma instrução de metapolítica que especifica quais portas podem servir arquivos de política. No entanto, em vez de "master-only", a metapolítica padrão para arquivos de política de soquete é "all". Isto é, a menos que o arquivo de política mestre especifique uma configuração mais restritiva, o Flash Player assumirá que qualquer soquete no host pode servir um arquivo de política de soquete.

O acesso a conexões de soquete XML e a soquete está desativado por padrão, mesmo que o soquete que você está conectando esteja no mesmo domínio que o arquivo SWF. É possível permitir acesso em nível de soquete servindo um arquivo de política de segurança de qualquer um dos seguintes locais:

- Porta 843 (o local do arquivo de política mestre)
- A mesma porta que a conexão de soquete principal
- Uma porta diferente da conexão de soquete principal

Por padrão, o Flash Player procura um arquivo de política de soquete na porta 843 e na mesma porta que a conexão de soquete principal. Se desejar servir um arquivo de política de soquete em uma porta diferente, o arquivo SWF deverá chamar `Security.loadPolicyFile()`.

Um arquivo de política de soquete tem a mesma sintaxe que um arquivo de política de URL, exceto que ele também deve especificar as portas às quais ele concede acesso. Quando um arquivo de política de soquete é servido em uma porta com número inferior a 1024, ele pode conceder acesso a qualquer porta. Quando um arquivo de política é proveniente da porta 1024 ou superior ele só pode conceder acesso a portas 1024 e superiores. As portas permitidas são especificadas em um atributo `to-ports` na tag `<allow-access-from>`. Os valores de números de portas únicos, intervalos de portas e curingas são aceitos.

Este é um exemplo de arquivo de política de soquete:

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM "http://www.adobe.com/xml/dtds/cross-domain-policy.dtd">
<!-- Policy file for xmlsocket://socks.mysite.com -->
<cross-domain-policy>
    <allow-access-from domain="*" to-ports="507" />
    <allow-access-from domain="*.example.com" to-ports="507,516" />
    <allow-access-from domain="*.example.org" to-ports="516-523" />
    <allow-access-from domain="adobe.com" to-ports="507,516-523" />
    <allow-access-from domain="192.0.34.166" to-ports="*" />
</cross-domain-policy>
```

Para recuperar um arquivo de política de soquete da porta 843 ou da mesma porta como uma conexão de soquete principal, chame o método `Socket.connect()` ou `XMLSocket.connect()`. O Flash Player primeiro verifica se há um arquivo de política mestre na porta 843. Se um arquivo de política mestre for localizado, ele verificará se o arquivo contém uma instrução de metapolítica que proíbe arquivos de política de soquete na porta de destino. Se o acesso não for proibido, o Flash Player primeiro procurará a instrução `allow-access-from` apropriada no arquivo de política mestre. Se uma instrução não for localizada, ele procurará um arquivo de política de soquete na mesma porta que a conexão de soquete principal.

Para recuperar um arquivo de política de soquete de um local diferente, primeiro chame o método `Security.loadPolicyFile()` com a sintaxe especial `"xmlsocket"`, como no seguinte:

```
Security.loadPolicyFile("xmlsocket://server.com:2525");
```

Chame o método `Security.loadPolicyFile()` antes de chamar o método `Socket.connect()` ou `XMLSocket.connect()`. O Flash Player então aguarda até que tenha atendido sua solicitação de arquivo de política antes de decidir se permite sua conexão principal. No entanto, se o arquivo de política mestre especificar que o local de destino não pode servir arquivos de política, a chamada para `loadPolicyFile()` não terá nenhum efeito, mesmo que haja um arquivo de política naquele local.

Se estiver implementando um servidor de soquetes e precisar fornecer um arquivo de política de soquete, decida se o arquivo de política deve ser fornecido usando a mesma porta que aceita conexões principais ou usando uma porta diferente. Em qualquer dos casos, o servidor deve aguardar a primeira transmissão de seu cliente antes de enviar uma resposta.

Quando o Flash Player solicita um arquivo de política, ele sempre transmite a string a seguir assim que a conexão é estabelecida:

```
<policy-file-request/>
```

Quando o servidor recebe essa string, ele pode transmitir o arquivo de política. A solicitação do Flash Player sempre é terminada por um byte nulo e a resposta do servidor também deve ser terminada por um byte nulo.

Não espere reutilizar a mesma conexão para uma solicitação de arquivo de política e para uma conexão principal. Feche a conexão depois de transmitir o arquivo de política. Se isso não for feito, o Flash Player fechará a conexão do arquivo de política antes de reconectar-se para configurar a conexão principal.

Envio de dados

O envio de dados ocorre quando código ActionScript de um arquivo SWF envia dados a um servidor ou a um recurso. O envio de dados é sempre permitido para arquivos SWF de domínio da rede. Um arquivo SWF local pode enviar dados a endereços de rede apenas se estiver na caixa de proteção local confiável ou local com rede. Para obter mais informações, consulte “Caixas de proteção locais” na página 706.

É possível usar a função `flash.net.sendToURL()` para enviar dados a uma URL. Outros métodos também enviam solicitações a URLs. Esses incluem métodos de carregamento, como `Loader.load()` e `Sound.load()`, métodos de carregamento de dados, como `URLLoader.load()` e `URLStream.load()`.

Upload e download de arquivos

O método `FileReference.upload()` inicia o upload de um arquivo selecionado por um usuário a um servidor remoto. É necessário chamar o método `FileReference.browse()` ou `FileReferenceList.browse()` antes de chamar o método `FileReference.upload()`.

O ActionScript que inicia o método `FileReference.browse()` ou `FileReferenceList.browse()` só pode ser chamado em resposta a um evento de mouse ou de teclado. Se ele for chamado em outras situações, o Flash Player 10 e versões posteriores irão gerar uma exceção.

A chamada do método `FileReference.download()` abre a caixa de diálogo na qual o usuário pode baixar um arquivo de um servidor remoto.

Nota: *Se o servidor exigir autenticação de usuário, apenas os aplicativos Flash em execução em um navegador (ou seja, que utilizam plug-in para navegador ou controle ActiveX), poderão fornecer uma caixa de diálogo para solicitar que o usuário insira um nome e uma senha para autenticação, e somente para downloads. O Flash Player não permite uploads para um servidor que exige autenticação de usuário.*

Uploads e downloads não serão permitidos se o arquivo SWF que está chamando estiver na caixa de proteção local com sistema de arquivos.

Por padrão, um arquivo SWF não pode iniciar um upload ou um download de um servidor que não seja seu próprio servidor. Um arquivo SWF pode carregar ou baixar de um servidor diferente se esse servidor fornecer um arquivo de política que conceda permissão ao domínio do arquivo SWF que está chamando.

Carregamento de conteúdo incorporado de arquivos SWF importados em um domínio de segurança

Ao carregar um arquivo SWF, você pode definir o parâmetro `context` do método `load()` do objeto Loader usado para carregar o arquivo. Esse parâmetro usa um objeto LoaderContext. Ao definir a propriedade `securityDomain` do objeto LoaderContext como `Security.currentDomain`, o Flash Player verificará se há um arquivo de política de URL no servidor do arquivo SWF carregado. Se houver um arquivo de política e ele conceder acesso ao domínio do arquivo SWF que está sendo carregado, você poderá carregar o arquivo SWF como mídia importada. Dessa maneira, o arquivo carregado pode obter acesso a objetos na biblioteca do arquivo SWF.

Uma maneira alternativa de um arquivo SWF acessar classes no arquivo SWF carregado de uma caixa de proteção de segurança diferente é fazer com que o arquivo SWF carregado chame o método `Security.allowDomain()` para conceder acesso ao domínio do arquivo SWF que está chamando. É possível adicionar a chamada ao método `Security.allowDomain()` ao método construtor da classe principal do arquivo SWF carregado e fazer com que o arquivo SWF que está sendo carregado adicione um ouvinte de eventos para responder ao evento `init` despachado pela propriedade `contentLoaderInfo` do objeto Loader. Quando esse evento for despachado, o arquivo SWF carregado terá chamado o método `Security.allowDomain()` no método construtor e as classes no arquivo SWF carregado estarão disponíveis para o arquivo SWF que está sendo carregado. O arquivo SWF que está sendo carregado pode recuperar classes do arquivo SWF carregado chamando `Loader.contentLoaderInfo.applicationDomain.getDefinition()`.

Trabalho com conteúdo legado

No Flash Player 6, o domínio usado para determinadas configurações do Flash Player se baseia na parte posterior do domínio do arquivo SWF. Essas configurações incluem configurações para permissões de câmera e microfone, quotas de armazenamento e armazenamento de objetos compartilhados persistentes.

Se o domínio de um arquivo SWF incluir mais de dois segmentos, como www.example.com, o primeiro segmento do domínio (www) será removido e a parte restante do domínio será usada. Assim, no Flash Player 6, www.example.com e store.example.com usam example.com como domínio dessas configurações. De modo semelhante, www.example.co.uk e store.example.co.uk usam example.co.uk como o domínio dessas configurações. Isso pode resultar em problemas nos quais arquivos SWF de domínios não relacionados, como example1.co.uk e example2.co.uk, têm acesso aos mesmos objetos compartilhados.

No Flash Player 7 e posterior, as configurações do player são escolhidas por padrão de acordo com o domínio exato de um arquivo SWF. Por exemplo, um arquivo SWF do www.example.com usa as configurações do player para www.example.com. Um arquivo SWF do store.example.com usa as configurações separadas do player para store.example.com.

Em um arquivo SWF gravado com o ActionScript 3.0, quando `Security.exactSettings` é definida como `true` (o padrão), o Flash Player usa domínios exatos para configurações do player. Quando definido como `false`, o Flash Player usa as configurações de domínio usadas no Flash Player 6. Se alterar o valor padrão de `exactSettings`, você deverá fazê-lo antes de ocorrer algum evento que exija que o Flash Player escolha configurações de player, por exemplo: usando uma câmera ou um microfone ou recuperando um objeto compartilhado persistente.

Se você publicou um arquivo SWF da versão 6 e criou objetos compartilhados persistentes a partir dele, para recuperar esses objetos compartilhados persistentes de um SWF que usa o ActionScript 3.0, você deve definir `Security.exactSettings` como `false` antes de chamar `SharedObject.getLocal()`.

Configuração de permissões de LocalConnection

A classe LocalConnection permite desenvolver arquivos SWF que podem enviar instruções um para o outro. Objetos LocalConnection podem se comunicar apenas entre arquivos SWF que estão em execução no mesmo computador cliente, mas podem estar em execução em diferentes aplicativos; por exemplo, um arquivo SWF em execução em um navegador e um arquivo SWF em execução em um projetor.

Para cada comunicação LocalConnection, há um arquivo SWF e um arquivo SWF ouvinte. Por padrão, o Flash Player permite comunicação LocalConnection entre arquivos SWF no mesmo domínio. Para arquivos SWF em diferentes caixas de proteção, o ouvinte deve permitir permissão de remetente usando o método `LocalConnection.allowDomain()`. A string passada como um argumento para o método `LocalConnection.allowDomain()` pode conter qualquer um dos seguintes: nomes exatos do domínio, endereços IP e `*` asterisco.

A forma do método `allowDomain()` foi alterada desde o ActionScript 1.0 e 2.0. Nessas versões anteriores, `allowDomain()` era um método de retorno de chamada que você implementava. No ActionScript 3.0, `allowDomain()` é um método embutido da classe LocalConnection que você chama. Com essa alteração, `allowDomain()` funciona da mesma maneira que `Security.allowDomain()`.

Um arquivo SWF usa a propriedade `domain` da classe LocalConnection para determinar seu domínio.

Controle do acesso à URL de saída

Script e acesso à URL de saída (por meio do uso de URLs HTTP, mailto: etc.) são executados com o uso das seguintes APIs do ActionScript 3.0:

- A função `flash.system.fscommand()`
- O método `ExternalInterface.call()`
- A função `flash.net.navigateToURL()`

Para arquivos SWF em execução localmente, as chamadas para esses métodos serão bem-sucedidas apenas se o arquivo SWF e a página que o contém (se houver) estiverem na caixa de proteção de segurança local confiável. Haverá falha em chamadas para esses métodos se o conteúdo estiver na caixa de proteção local com rede ou local com sistema de arquivos.

Para arquivos SWF que não estão em execução localmente, todas essas APIs podem se comunicar com a página da Web na qual estão incorporados, dependendo do valor do parâmetro AllowScriptAccess descrito a seguir. A função `flash.net.navigateToURL()` tem a capacidade adicional de comunicação com qualquer janela ou quadro de navegador aberto, não apenas com a página na qual o arquivo SWF está incorporado. Para obter mais informações sobre essa funcionalidade, consulte "Uso da função navigateToURL()" na página 732.

O parâmetro `AllowScriptAccess` no código HTML que carrega um arquivo SWF controla a capacidade de executar acesso à URL a partir do arquivo SWF. Defina esse parâmetro dentro da tag PARAM ou EMBED. Se nenhum valor estiver definido para `AllowScriptAccess`, o arquivo SWF e a página HTML poderão se comunicar apenas se os dois forem do mesmo domínio.

O valor do parâmetro `AllowScriptAccess` pode ser um destes três valores possíveis: `"always"`, `"sameDomain"` ou `"never"`.

- Quando `AllowScriptAccess` é `"always"`, o arquivo SWF pode se comunicar com a página HTML na qual ele está incorporado, mesmo quando o arquivo é de um domínio diferente da página HTML.
- Quando `AllowScriptAccess` é `"sameDomain"`, o arquivo SWF pode se comunicar com a página HTML na qual ele está incorporado apenas quando o arquivo SWF está no mesmo domínio que a página HTML. Esse valor é o padrão de `AllowScriptAccess`. Use essa configuração, ou não defina um valor para `AllowScriptAccess`, para evitar que um arquivo SWF hospedado em um domínio acesse um script em uma página HTML proveniente de outro domínio.
- Quando `AllowScriptAccess` está definido como `"never"`, um arquivo SWF não pode se comunicar com nenhuma página HTML. O uso deste valor tornou-se obsoleto no Adobe Flash CS4 Professional. Ele não é recomendado e não deveria ser necessário se você não disponibiliza arquivos SWF não confiáveis de seu próprio domínio. Se for necessário servir arquivos SWF não confiáveis, a Adobe recomenda que seja criado um subdomínio diferente e que todo o conteúdo não confiável seja colocado nele.

Este é um exemplo de configuração da tag `AllowScriptAccess` em uma página HTML para permitir acesso à URL de saída para um domínio diferente:

```
<object id='MyMovie.swf' classid='clsid:D27CDB6E-AE6D-11cf-96B8-444553540000'
codebase='http://download.adobe.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,0,0'
height='100%' width='100%'>
<param name='AllowScriptAccess' value='always'/>
<param name='src' value=''MyMovie.swf'/>
<embed name='MyMovie.swf' pluginspage='http://www.adobe.com/go/getflashplayer'
src='MyMovie.swf' height='100%' width='100%' AllowScriptAccess='never'/>
</object>
```

Uso da função navigateToURL()

Além da configuração de segurança especificada pelo parâmetro `allowScriptAccess` discutido acima, a função `navigateToURL()` tem um segundo parâmetro - `target`. O parâmetro `target` pode ser usado para especificar o nome de uma janela ou quadro HTML para o qual enviar a solicitação de URL. Restrições de segurança adicionais se aplicam a essas solicitações, e as restrições variarão se `navigateToURL()` estiver sendo usado como uma instrução de script ou de não-script.

Para instruções de script, como `navigateToURL("javascript: alert('Hello from Flash Player.')")`, as seguintes regras são aplicadas.

- Se o arquivo SWF for um arquivo confiável localmente, a solicitação terá êxito.
- Se o destino for a página HTML na qual o arquivo SWF está incorporado, as regras de `allowScriptAccess` descritas acima serão aplicadas.
- Se o destino contiver conteúdo carregado do mesmo domínio que o arquivo SWF, a solicitação terá êxito.
- Se o destino contiver conteúdo carregado de um domínio diferente do arquivo SWF, e nenhuma das duas condições anteriores for atendida, haverá falha na solicitação.

Para instruções de não-script (como HTTP, HTTPS e `mailto:`, haverá falha na solicitação se todas as condições a seguir se aplicarem:

- O destino é uma das palavras-chave `"_top"` ou `"_parent"` e
- o arquivo SWF é uma página da Web hospedada em um domínio diferente e
- o arquivo SWF está incorporado com um valor de `allowScriptAccess` diferente de `"always"`.

Para obter mais informações

Para obter mais informações sobre acesso à URL de saída, consulte as seguintes entradas no *Referência dos componentes e da linguagem do ActionScript 3.0:*

- A função `flash.system.fscommand()`
- O método `call()` da classe ExternalInterface
- A função `flash.net.navigateToURL()`

Objetos compartilhados

O Flash Player fornece a capacidade de usar *objetos compartilhados* que são objetos ActionScript que persistem fora de um arquivo SWF, seja localmente no sistema de arquivos de um usuário ou remotamente em um servidor RTP. Objetos compartilhados, como outras mídias no Flash Player, são particionados em caixas de proteção de segurança. No entanto o modelo de caixa de proteção para objetos compartilhados é um pouco diferente porque objetos compartilhados não são recursos que podem ser acessados além dos limites do domínio. Ao contrário, os objetos compartilhados sempre são recuperados de um armazenamento de objetos compartilhados que é específico ao domínio de cada arquivo SWF que chama métodos da classe SharedObject. Normalmente, um objeto compartilhado é mais particular do que o domínio de um arquivo SWF: por padrão, cada arquivo SWF usa um armazenamento de objetos compartilhados específico a toda a sua URL de origem.

Um arquivo SWF pode usar o parâmetro `localPath` dos métodos `SharedObject.getLocal()` e `SharedObject.getRemote()` para usar um armazenamento de objetos compartilhados com apenas uma parte de sua URL. Dessa maneira, o arquivo SWF pode permitir o compartilhamento com outros arquivos SWF de outras URLs. Mesmo que você passe `'/'` como o parâmetro `localPath`, ele ainda especifica um objeto compartilhado específico a seu próprio domínio.

Os usuários podem restringir o acesso a objetos compartilhados usando a caixa de diálogo Configurações do Flash Player ou o Gerenciador de configurações. Por padrão, os objetos compartilhados podem ser criados até um máximo de 100 KB de dados por domínio. Usuários administrativos e usuários também podem aplicar restrições quanto à capacidade de gravar no sistema de arquivos. Para obter mais informações, consulte "Controles de administrador" na página 708 e "Controles de usuário" na página 710.

É possível especificar que um objeto compartilhado está protegido especificando `true` para o parâmetro `secure` do método `SharedObject.getLocal()` ou do método `SharedObject.getRemote()`. Observe o seguinte sobre o parâmetro `secure`:

- Se esse parâmetro for definido como `true`, o Flash Player criará um novo objeto compartilhado ou obterá uma referência a um objeto compartilhado protegido existente. Esse objeto compartilhado protegido apenas poderá ser lido ou gravado por arquivos SWF entregues via HTTPS que chamarem `SharedObject.getLocal()` com o parâmetro `protegido` definido como `true`.
- Se esse parâmetro for definido como `false`, o Flash Player criará um novo objeto compartilhado ou obterá uma referência a um objeto compartilhado existente que pode ser lido ou gravado por arquivos SWF entregues via conexões não-HTTPS.

Se o arquivo SWF que faz a chamada não for de uma URL HTTPS, especificar `true` para o parâmetro `secure` do método `SharedObject.getLocal()` ou do método `SharedObject.getRemote()` resultará em uma exceção SecurityError.

A opção de armazenamento de um objeto compartilhado tem como base a URL de origem do arquivo SWF. Isso é verdadeiro mesmo nas duas situações em que um arquivo SWF não é originário de uma URL simples: carregamento de importação e carregamento dinâmico. O carregamento de importação se refere à situação em que você carrega um arquivo SWF com a propriedade `LoaderContext.securityDomain` definida como `SecurityDomain.currentDomain`. Nessa situação, o arquivo SWF carregado terá uma pseudo-URL que começa com o domínio dou arquivo SWF carregado e especifica sua URL de origem real. O carregamento dinâmico refere-se ao carregamento de um arquivo SWF usando o método `Loader.loadBytes()`. Nessa situação, o arquivo SWF carregado terá uma pseudo-URL que começa com a URL completa do arquivo SWF carregado, seguida por uma ID de número inteiro. Em ambos os casos de carregamento de importação e de carregamento dinâmico, a pseudo-URL de um arquivo SWF pode ser examinada usando a propriedade `LoaderInfo.url`. A pseudo-URL é tratada exatamente como uma URL real para a escolha de um armazenamento de objetos compartilhados. É possível especificar o parâmetro `localPath` de um objeto compartilhado que use parte ou toda a pseudo-URL.

Usuários e administradores podem optar por desativar o uso de *objetos compartilhados de terceiros*. Esse é o uso de objetos compartilhados por qualquer arquivo SWF que esteja em execução em um navegador da Web, quando a URL de origem desse arquivo SWF for de um domínio diferente da URL mostrada na barra de endereço do navegador. Usuários e administradores podem optar por desativar o uso de objetos compartilhados de terceiros por motivos de privacidade, com o objetivo de impedir rastreamento entre domínios. Para evitar essa restrição, convém garantir que qualquer arquivo SWF que usa objetos compartilhados seja carregado apenas dentro de estruturas da página HTML que garantem que o arquivo SWF se origina do mesmo domínio mostrado na barra de endereço do navegador. Quando você tenta usar objetos compartilhados de um arquivo SWF de terceiros, e o objeto compartilhado de terceiros está desativado, os métodos `SharedObject.getLocal()` e `SharedObject.getRemote()` retornam `null`. Para obter mais informações, consulte www.adobe.com/products/flashplayer/articles/thirdpartylso.

Acesso a câmera, microfone, área de transferência, mouse e teclado

Quando um arquivo SWF tenta acessar a câmera ou o microfone de um usuário usando os métodos `Camera.get()` ou `Microphone.get()`, o Flash Player exibe a caixa de diálogo Privacidade na qual o usuário pode permitir ou negar acesso à sua câmera e microfone. O usuário e o usuário administrativo também podem desativar o acesso à câmera em uma base por site ou global, por meio de controles no arquivo mms.cfg, da UI de Configurações e do Gerenciador de configurações (consulte "Controles de administrador" na página 708 e "Controles de usuário" na página 710). Com restrições de usuário, cada método `Camera.get()` e `Microphone.get()` retorna um valor `null`. É possível usar a propriedade `Capabilities.avHardwareDisable` para determinar se a câmera e o microfone foram proibidos (`true`) ou permitidos (`false`) administrativamente.

O método `System.setClipboard()` permite que um arquivo SWF substitua o conteúdo da área de transferência por uma string de caracteres de texto simples. Isso não impõe nenhum risco de segurança. Para proteção contra o risco imposto por recorte e cópia de senhas e outros dados confidenciais nas áreas de transferência, não há nenhum método `getClipboard()` correspondente.

Um aplicativo executado no Flash Player só pode monitorar eventos de teclado e de mouse que ocorrem dentro de seu foco. O conteúdo executado no Flash Player não consegue detectar eventos de teclado ou de mouse em outro aplicativo.

Índice

F

N

O

T

U

V

W

X