Otimizando o desempenho para a plataforma ADOBE® FLASH®

Avisos legais

Para ver os avisos legais, consulte http://help.adobe.com/pt_BR/legalnotices/index.html.

Conteúdo

Capítulo 1: Introdução

Aplicativos Adobe® AIR® e o Adobe® Flash® Player são executados em muitas plataformas, inclusive desktops, dispositivos móveis, tablets e aparelhos de TV. Através de exemplos de código e casos de uso, este documento descreve as melhores práticas para desenvolvedores que implantam esses aplicativos. Os tópicos incluem:

- Economia de memória
- Minimização do uso da CPU
- Melhorando o desempenho do ActionScript 3.0
- Aumentando a velocidade de renderização
- Otimização da interação na rede
- Trabalhando com áudio e vídeo
- Otimizando o desempenho do banco de dados SQL
- Benchmark e aplicativos de implantação

A maioria dessas otimizações se aplicam a aplicaitos em todos os dispositivos, tanto no tempo de execução do AIR quando do Flash Player. Adições e exceções para dispositivos específicos também são abordadas.

Algumas dessas otimizações se concentram em recursos introduzidos no Flash Player 10.1 e no AIR 2.5. No entanto, muitas dessas otimizações também se aplicam a versões mais antigas do AIR e do Flash Player.

Conceitos básicos da execução do código de tempo de execução

Uma chave para compreender como melhorar o desempenho do aplicativo é compreender como o tempo de execução da plataforma Flash executa o código. O tempo de execução opera em "loop" com certas ações que ocorrem em cada "quadro". Neste caso um quadro é simplesmente um bloco de tempo determinado pela taxa de quadros especificada para o aplicativo. A quantidade de tempo alocada para cada quadro corresponde diretamente à taxa de quadros. Por exemplo, se especificar uma taxa de quadros de 30 quadros por segundo, o tempo de execução tenta fazer com que cada quadro permaneça um trigésimo de segundo.

Você especifica a taxa de quadros inicial do aplicativo na hora da autoria. Você pode definir a taxa de quadros usando as configurações do Adobe® Flash® Builder™ ou do Flash Professional. E possível especificar a taxa de quadros inicial no código Defina a taxa de quadros em um aplicativo apenas ActionScript aplicando a marca de metadados `[SWF(frameRate="24")]` à classe de documentos raiz. No MXML, defina o atributo `frameRate` no aplicativo ou na marca WindowedApplication.

Cada loop de quadros consiste em duas fases divididas em três partes: eventos, evento `enterframe` e renderização.

A primeira fase inclui duas partes (eventos e o evento `enterframe`) ambos podem levar potencialmente à chamada do código. Na primeira parte da primeira fase, os eventos em tempo de execução chegam e são despachados. Esses eventos podem representar conclusão ou progresso de operações assíncronas como, por exemplo, uma resposta dos dados da carga ao longo de uma rede. Incluem também eventos de entradas do usuário. À medida que os eventos são despachados, o tempo de execução executa o código nos ouvintes que você registrou. Se não ocorrer nenhum evento, o tempo de execução aguarda até completar essa fase de execução sem desempenhar qualquer ação. O tempo de execução nunca acelera a taxa de quadros por causa de falta de atividade. Se ocorrerem eventos durante outras partes do ciclo de execução, o tempo de execução coloca em fila esses eventos e os despacha para o próximo quadro.

A segunda parte da primeira fase é o evento `enterframe`. Esse evento é diferente dos outros porque é sempre despachado um por quadro.

Depois que todos os eventos são despachados, começa a fase de renderização do loop de quadros. Nesse momento, o tempo de execução calcula o estado de todos os elementos visíveis na tela e os desenha na tela. Em seguida, o processo repete-se sozinho, como um corredor em volta de uma pista de corrida.

Nota: *Para eventos que incluem uma propriedade* `updateAfterEvent`*, a renderização pode ser forçada a ocorrer imediatamente em vez de esperar a fase de renderização. No entanto, evite usar* `updateAfterEvent` *se esta propriedade causar frequentemente problemas de desempenho.*

É mais fácil imaginar que as duas fases no loop de quadros demoram o mesmo tempo. Nesse caso, durante a metade de cada loop de quadros, os operadores de eventos e o código do aplicativo estão em execução e, na outra metade, a renderização ocorre. No entanto, a realidade em geral é diferente. Às vezes o código do aplicativo demora mais da metade do tempo disponível no quadro, prolongando sua alocação de tempo e reduzindo a alocação disponível para a renderização. Em outros casos, principalmente, com conteúdo visual complexo como, por exemplo, filtros e modos de mesclagem, a renderização requer mais da metade do tempo dos quadros. Visto que o tempo real que as fases demoram é flexível, o loop de quadros é conhecido normalmente como "pista de corrida elástica".

Se as operações combinadas do loop de quadros (execução do código e renderização) demorarem muito, o tempo de execução não consegue manter a taxa de quadros. O quadro expande-se, demorando mais que o tempo alocado, assim há um atraso antes do próximo quadro acionar. Por exemplo, se um ciclo de quadros durar mais de um trigésimo de segundo, o tempo de execução não poderá atualizar a tela a 30 quadros por segundo. Quando a taxa de quadros fica lenta, a experiência perde qualidade. A melhor animação aparece cortada. No pior dos casos, o aplicativo congela-se e a janela fica em branco.

Para obter mais detalhes sobre a execução do código de tempo de execução da plataforma Flash e o modelo de renderização, consulte os seguintes recursos:

- Modelo mental do Flash Player - A pista de corrida elástica (artigo por Ted Patrick)
- Execução assíncrona do ActionScript (artigo por Trevor McCauley)
- Saiba mais sobre a otimização do Adobe AIR para a execução de código, memória e renderização consultando Optimizing Adobe AIR for code execution, memory & rendering, em http://www.adobe.com/go/learn_fp_air_perf_tv_br(Vídeo de apresentação da conferência MAX, por Sean Christmann)

Desempenho percebido versus desempenho real

Os últimos juízes do aplicativo ter bom desempenho são os usuários do aplicativo. Os desenvolvedores podem medir o desempenho do aplicativo em termos de quantidade de tempo que certas operações demoram para executar ou quantos ocorrências de objetos são criadas. No entanto, essas medidas não são importantes para os usuários finais. Algumas vezes os usuários avaliam o desempenho com critérios diferentes. Por exemplo, o aplicativo opera mais rápida e suavemente e responde mais rapidamente às entradas? Tem um efeito negativo no desempenho do sistema? Faça a si mesmo essas perguntas, que são testes de desempenho percebido:

- As animações são uniformes ou cortadas?
- O conteúdo do vídeo parece uniforme ou cortado?
- Os clipes de áudio reproduzem continuamente ou são interrompidos e depois recomeçam?
- A janela pisca ou fica em branco durante operações longas?
- Quando digito, a entrada de texto mantém-se no lugar ou fica para trás?
- Quando clico, acontece alguma coisa imediatamente ou há um atraso?
- O som do ventilador da CPU fica mais alto quando o aplicativo executa?
- Em laptops ou dispositivos portáteis, a bateria acaba rapidamente ao executar o aplicativo?
- Os outros aplicativos não respondem bem quando o aplicativo está em execução?

A diferença entre desempenho percebido e desempenho real é importante, porque a maneira de alcançar o melhor desempenho percebido nem sempre é a mesma maneira de obter o desempenho mais rápido absoluto. Certifique-se de que o aplicativo nunca execute excessivamente o código de forma que o tempo de execução não consiga atualizar a tela e reunir entradas do usuário com frequência. Em alguns casos, alcançar esse equilíbrio envolve dividir a tarefa do programa em partes para que, entre as partes, o tempo de execução atualize a tela. Consulte “Desempenho da renderização” na página 48 para obter orientação específica).

As dicas e as técnicas descritas neste documento objetivam melhorias no desempenho da execução do código real e na maneira como os usuários percebem o desempenho.

Objetivo das otimizações

Algumas melhorias no desempenho não criam uma melhoria perceptível para os usuários. É importante concentrar as otimizações no desempenho em áreas que são problemáticas para o aplicativo específico. Algumas otimizações de desempenho são boas práticas gerais e podem sempre ser seguidas. A utilidade de outras depende das necessidades do aplicativo e de sua base de usuários prevista. Por exemplo, os aplicativos sempre desempenham melhor quando não é usada nenhuma animação, vídeo ou filtros e efeitos gráficos. No entanto, um dos motivos para utilizar a plataforma Flash para criar aplicativos é que as capacidades de mídia e gráficas permitem aplicativos altamente expressivos. Considere se o nível desejado de riqueza corresponde bem às características de desempenho das máquinas e dos dispositivos nos quais o aplicativo é executado.

Um conselho comum é "evitar otimizar muito cedo". Algumas otimizações de desempenho requerem código de escrita de uma forma que é muito difícil de ler ou menos flexível. Esse código, uma vez otimizado, é mais difícil de manter. Nessa otimizações, frequentemente é melhor aguardar e verificar se uma determinada seção do código não tem bom desempenho antes de escolher otimizar o código.

A melhoria no desempenho às vezes envolve fazer escolhas. Em condições ideais, a redução da quantidade de memória consumida pelo aplicativo também aumenta a velocidade na qual o aplicativo desempenha uma tarefa. No entanto, esse tipo de melhoria ideal nem sempre é possível. Por exemplo, se o aplicativo congelar durante uma operação, a solução normalmente envolve dividir o trabalho para executar em múltiplos quadros. Uma vez que o trabalho é dividido, é provável que demore mais para concluir o processo. Contudo, haverá a possibilidade de o usuário não notar o tempo adicional se o aplicativo continuar respondendo às entradas e não congelar.

Uma chave para saber o quê otimizar e se a otimização é útil é realizar testes de desempenho. Diversas técnicas e dicas para testar o desempenho estão descritas em "Benchmark e implantação" na página 95.

Para obter mais informações sobre como verificar se partes de um aplicativo são boas candidatas para otimização, consulte os seguintes recursos:

- Saiba mais sobre aplicativos de ajuste de desempenho para o AIR consultando Performance-tuning apps for AIR em http://www.adobe.com/go/learn_fp_goldman_tv_br(Vídeo de apresentação da conferência MAX, por Oliver Goldman)
- Saiba mais consultando Aplicativos de Ajuste de Desempenho para o Adobe AIR, emhttp://www.adobe.com/go/learn_fp_air_perf_devnet_br(Conexão do Desenvolvedor do Adobe, um artigo de Oliver Goldman baseado na apresentação)

Capítulo 2: Economia de memória

É sempre importante economizar memória durante o desenvolvimento de aplicativos, mesmo para aplicativos de área de trabalho. Dispositivos móveis, no entanto, têm uma preocupação extrema com o consumo de memória, e vale a pena limitar a quantidade de memória consumida por seu aplicativo.

Objetos de exibição


Escolha um objeto de exibição apropriado.

O ActionScript 3.0 inclui um grande conjunto de objetos de exibição. Uma das dicas de otimização mais simples para limitar o uso da memória é usar o tipo apropriado de objeto de exibição. Para formas simples que não sejam interativas, use objetos Shape. Para objetos interativos que não precisem de uma linha de tempo, use objetos Sprite. Para animação que use uma linha de tempo, use objetos MovieClip. Escolha sempre o tipo de objeto mais eficientes para sua aplicação.

O código a seguir mostra o uso da memória para diferentes objetos de exibição:

```
trace(getSize(new Shape()));
// output: 236

trace(getSize(new Sprite()));
// output: 412

trace(getSize(new MovieClip()));
// output: 440
```

O método `getSize()` mostra quantos bytes um objeto utiliza da memória. Você pode ver que o uso de múltiplos objetos MovieClip ao invés de objetos Shape simples pode causar um desperdício de memória se as capacidades de um objeto MovieClip não forem necessárias.

Tipos primitivos


Use o método `getSize()` para criar um benchmark do código e determinar o objeto mais eficiente para a tarefa.

Todo os tipos primitivos, exceto String, usam 4 – 8 bytes de memória. Não há como otimizar a memória usando um tipo específico para um primitivo:

```
// Primitive types
var a:Number;
trace(getSize(a));
// output: 8

var b:int;
trace(getSize(b));
// output: 4

var c:uint;
trace(getSize(c));
// output: 4

var d:Boolean;
trace(getSize(d));
// output: 4

var e:String;
trace(getSize(e));
// output: 4
```

Um Número, que representa um valor de 64 bits, tem 8 bytes a ele alocados pelo ActionScript Virtual Machine (AVM), caso um valor não lhe tenha sido atribuído. Todos os outros tipos primitivos são armazenados em 4 bytes.

```
// Primitive types
var a:Number = 8;
trace(getSize(a));
// output: 4

a = Number.MAX_VALUE;
trace(getSize(a));
// output: 8
```

O comportamento difere para o tipo String. A quantidade de armazenamento alocada se baseia no comprimento da sequência de caracteres:

```
var name:String;
trace(getSize(name));
// output: 4

name = "";
trace(getSize(name));
// output: 24

name = "Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum
has been the industry's standard dummy text ever since the 1500s, when an unknown printer took
a galley of type and scrambled it to make a type specimen book. It has survived not only five
centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It
was popularized in the 1960s with the release of Letraset sheets containing Lorem Ipsum
passages, and more recently with desktop publishing software like Aldus PageMaker including
versions of Lorem Ipsum.";
trace(getSize(name));
// output: 1172
```

Use o método `getSize()` para criar um benchmark do código e determinar o objeto mais eficiente para a tarefa.

Reutilização de objetos

Reutilize objetos, quando possível, em vez de recriá-los.

Outra maneira simples de otimizar a memória é reutilizar objetos e evitar sua recriação, sempre que possível. Por exemplo, em um loop, não use o seguinte código:

```
const MAX_NUM:int = 18;
const COLOR:uint = 0xCCCCCC;

var area:Rectangle;

for (var:int = 0; i < MAX_NUM; i++)
{
    // Do not use the following code
    area = new Rectangle(i,0,1,10);
    myBitmapData.fillRect(area,COLOR);
}
```

Recriar o objeto Retangle em cada iteração de repetição utiliza mais memória e é mais lento porque um novo objeto é criado a cada iteração. Use a seguinte abordagem:

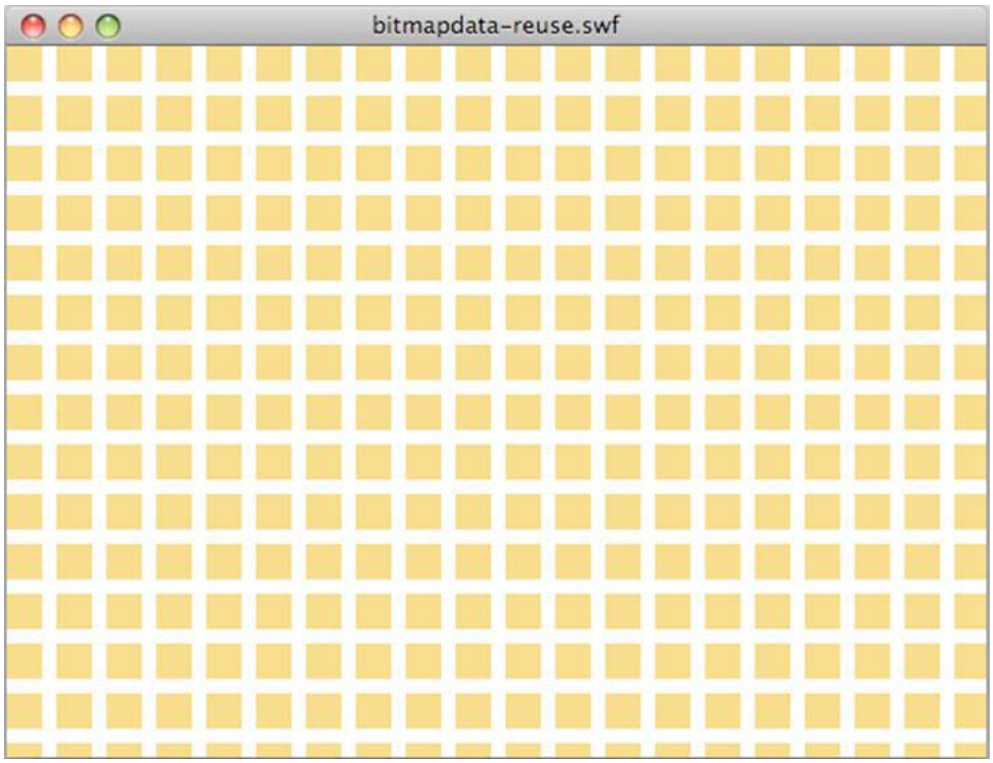
```
const MAX_NUM:int = 18;
const COLOR:uint = 0xCCCCCC;

// Create the rectangle outside the loop
var area:Rectangle = new Rectangle(0,0,1,10);

for (var:int = 0; i < MAX_NUM; i++)
{
    area.x = i;
    myBitmapData.fillRect(area,COLOR);
}
```

O exemplo anterior usou um objeto com um impacto relativamente pequeno na memória. O próximo exemplo demonstra uma economia maior de memória pela reutilização de um objeto BitmapData. O código a seguir para criar um efeito lado a lado desperdiça memória:

```
var myImage:BitmapData;
var myContainer:Bitmap;
const MAX_NUM:int = 300;

for (var i:int = 0; i< MAX_NUM; i++)
{
    // Create a 20 x 20 pixel bitmap, non-transparent
    myImage = new BitmapData(20,20,false,0xF0D062);

    // Create a container for each BitmapData instance
    myContainer = new Bitmap(myImage);

    // Add it to the display list
    addChild(myContainer);

    // Place each container
    myContainer.x = (myContainer.width + 8) * Math.round(i % 20);
    myContainer.y = (myContainer.height + 8) * int(i / 20);
}
```

Nota: *Ao usar valores positivos, converter o valor arredondado em um int é muito mais rápido do que usar o método* `Math.floor()`.

A imagem a seguir mostra o resultado do bitmap lado a lado:



Resultado do bitmap lado a lado

Uma versão otimizada cria uma única instância de BitmapData a que várias instâncias de Bitmap faz referência e produz o mesmo resultado:

```
// Create a single 20 x 20 pixel bitmap, non-transparent
var myImage:BitmapData = new BitmapData(20,20,false,0xF0D062);
var myContainer:Bitmap;
const MAX_NUM:int = 300;

for (var i:int = 0; i< MAX_NUM; i++)
{
    // Create a container referencing the BitmapData instance
    myContainer = new Bitmap(myImage);

    // Add it to the display list
    addChild(myContainer);

    // Place each container
    myContainer.x = (myContainer.width + 8) * Math.round(i % 20);
    myContainer.y = (myContainer.height + 8) * int(i / 20);
}
```

Esta abordagem economiza cerca de 700 KB de memória, o que representa uma economia significativa em um dispositivo móvel tradicional. Cada container de bitmap pode ser manipulado sem a alteração da instância de BitmapData original, usando-se as propriedades de Bitmap:

```
// Create a single 20 x 20 pixel bitmap, non-transparent
var myImage:BitmapData = new BitmapData(20,20,false,0xF0D062);
var myContainer:Bitmap;
const MAX_NUM:int = 300;

for (var i:int = 0; i< MAX_NUM; i++)
{
    // Create a container referencing the BitmapData instance
    myContainer = new Bitmap(myImage);

    // Add it to the DisplayList
    addChild(myContainer);

    // Place each container
    myContainer.x = (myContainer.width + 8) * Math.round(i % 20);
    myContainer.y = (myContainer.height + 8) * int(i / 20);

    // Set a specific rotation, alpha, and depth
    myContainer.rotation = Math.random()*360;
    myContainer.alpha = Math.random();
    myContainer.scaleX = myContainer.scaleY = Math.random();
}
```

A imagem a seguir mostra o resultado das transformações do bitmap:

Resultado das transformações do bitmap

Mais tópicos da Ajuda

"Armazenamento do bitmap em cache" na página 54

Pool de objetos

Use um pool de objetos, sempre que possível.

Outra importante otimização é chamada pool de objetos e envolve a reutilização de objetos com o tempo. Você cria um número de objetos definido durante a inicialização do aplicativo e os armazena em um pool, como um objeto Array ou Vector. Quando terminar de usar um objeto, você o desativa para que deixe de consumir recursos de CPU e remove todas as referências mutuas. No entanto, você não define as referências como `null`, o que o qualificaria para a coleta de lixo. Você simplesmente coloca o objeto de volta no pool e o recupera quando precisar de um novo objeto.

Reutilizar objetos reduz a necessidade de instanciar objetos, o que pode ser dispendioso. Também reduz a chance de execução do coletor de lixo, o que poderia tornar seu aplicativo mais lento. O código a seguir ilustra a técnica de pool de objetos:

```
package
{
    import flash.display.Sprite;

    public final class SpritePool
    {
        private static var MAX_VALUE:uint;
        private static var GROWTH_VALUE:uint;
        private static var counter:uint;
        private static var pool:Vector.<Sprite>;
        private static var currentSprite:Sprite;

        public static function initialize( maxPoolSize:uint, growthValue:uint ):void
        {
            MAX_VALUE = maxPoolSize;
            GROWTH_VALUE = growthValue;
            counter = maxPoolSize;

            var i:uint = maxPoolSize;

            pool = new Vector.<Sprite>(MAX_VALUE);
            while( --i > -1 )
                pool[i] = new Sprite();
        }

        public static function getSprite():Sprite
        {
            if ( counter > 0 )
                return currentSprite = pool[--counter];

            var i:uint = GROWTH_VALUE;
            while( --i > -1 )
                    pool.unshift ( new Sprite() );
            counter = GROWTH_VALUE;
            return getSprite();

        }

        public static function disposeSprite(disposedSprite:Sprite):void
        {
            pool[counter++] = disposedSprite;
        }
    }
}
```

A classe SpritePool cria um pool de novos objetos na inicialização do aplicativo. O método `getSprite()` retorna instâncias desses objetos e o método `disposeSprite()` as libera. O código permite que o pool cresça quando tiver sido completamente consumido. Também é possível criar um pool de tamanho fixo, onde novos objetos não seriam alocados quando o pool é exaurido. Tente evitar criar novos objetos em loops, se possível. Para obter mais informações, consulte "Liberação de memória" na página 12. O código a seguir usa a classe SpritePool para recuperar novas instâncias:

```
const MAX_SPRITES:uint = 100;
const GROWTH_VALUE:uint = MAX_SPRITES >> 1;
const MAX_NUM:uint = 10;

SpritePool.initialize ( MAX_SPRITES,  GROWTH_VALUE );

var currentSprite:Sprite;
var container:Sprite = SpritePool.getSprite();

addChild ( container );

for ( var i:int = 0; i< MAX_NUM; i++ )
{
    for ( var j:int = 0; j< MAX_NUM; j++ )
    {
        currentSprite = SpritePool.getSprite();
        currentSprite.graphics.beginFill ( 0x990000 );
        currentSprite.graphics.drawCircle ( 10, 10, 10 );
        currentSprite.x = j * (currentSprite.width + 5);
        currentSprite.y = i * (currentSprite.width + 5);
        container.addChild ( currentSprite );
    }
}
```

O código a seguir remove todos os objetos de exibição da lista de exibição quando o mouse é clicado e os reutiliza posteriormente para outra tarefa:

```
stage.addEventListener ( MouseEvent.CLICK, removeDots );

function removeDots ( e:MouseEvent ):void
{
    while (container.numChildren > 0 )
        SpritePool.disposeSprite (container.removeChildAt(0) as Sprite );
}
```

Nota: *O vetor do pool sempre faz referência a objetos Sprite. Se quiser remover o objeto completamente da memória, você precisa de um método* `dispose()` *na classe SpritePool que remova todas as referências restantes.*

Liberação de memória

Exclua todas as referências a objetos para certificar-se de que a coleta de lixo seja acionada.

Você não pode iniciar o coletor de lixo diretamente na versão de lançamento do Flash Player. Para certificar-se de que um objeto foi coletado como lixo, exclua todas as referências ao objeto. Lembre-se de que o antigo operador `delete` utilizado nas versões 1.0 e 2.0 do ActionScript se comportam de maneira diferente no ActionScript 3.0. Ele só pode ser utilizado pra deletar propriedades dinâmicas em um objeto dinâmico.

Nota: *Você pode chamar o coletor de lixo diretamente no Adobe® AIR® na versão de depuração do Flash Player.*

Por exemplo, o código a seguir defina uma referência a Sprite como `null`:

```
var mySprite:Sprite = new Sprite();

// Set the reference to null, so that the garbage collector removes
// it from memory
mySprite = null;
```

Lembre-se de que quando um objeto é definido como `null`, não é necessário removê-lo da memória. Às vezes, o coletor de lixo não é executado, se a memória disponível não for considerada baixa o suficiente. A coleta de lixo não é previsível. Alocação de memória, diferente da exclusão de objeto, aciona o coletor de lixo. Quando o coletor de lixo é executado, ele encontra os gráficos de objetos que ainda não foram coletados. Ele detecta objetos inativos nos gráficos localizando objetos que fazem referência um ao outro, mas que não são mais usados pelo aplicativo. Os objetos inativos detectados dessa forma são excluídos.

Em grandes aplicativos, esse processo pode consumir muita CPU e pode afetar o desempenho e gerar uma redução perceptível da velocidade do aplicativo. Tente limitar as passagens da coleta de lixo reutilizando objetos tanto quanto possível. Além disso, defina as referências como null, quando possível, para que o coletor de lixo gaste menos tempo de processamento localizando os objetos. Pense na coleta de lixo como um seguro e sempre gerencie o tempo de vida dos objetos explicitamente, quando possível.

Nota: *Configurar uma referência de um objeto de exibição para nulo não garante que o objeto seja congelado. O objeto continua a consumir ciclos de CPU até que o lixo seja coletado. Certifique-se de desativar corretamente seu objeto antes de definir sua referência para* `null`.

O coletor de lixo pode ser iniciado com o método `System.gc()`, disponível no Adobe AIR e na versão com depuração do Flash Player. O criador de perfil fornecido com o Adobe® Flash® Builder permite iniciar o coletor de lixo manualmente. Executar o coletor de lixo permite ver como seu aplicativo responde e se os objetos são excluídos corretamente da memória.

Nota: *Se um objeto tiver sido usado como um ouvinte de eventos, outro objeto pode fazer referência a ele. Neste caso, remova os ouvintes de evento utilizando o método* `removeEventListener()` *antes de definir as referências para* `null`.

Felizmente, a quantidade de memória utilizada por bitmaps pode ser reduzida imediatamente. Por exemplo, a classe BitmapData inclui um método `dispose()`. O próximo exemplo cria uma instância de BitmapData de 1,8 MB. A memória em uso atual chega até 1,8 MB e a propriedade `System.totalMemory` retorna um valor menor:

```
trace(System.totalMemory / 1024);
// output: 43100

// Create a BitmapData instance
var image:BitmapData = new BitmapData(800, 600);

trace(System.totalMemory / 1024);
// output: 44964
```

Em seguida, o BitmapData é removido manualmente (descartado) da memória e o uso da memória é verificado novamente:

```
trace(System.totalMemory / 1024);
// output: 43100

// Create a BitmapData instance
var image:BitmapData = new BitmapData(800, 600);

trace(System.totalMemory / 1024);
// output: 44964

image.dispose();
image = null;

trace(System.totalMemory / 1024);
// output: 43084
```

Embora o método `dispose()` remova os pixels da memória, a referência ainda precisa ser definida como `null` para que ele seja completamente liberado. Sempre chame o método `dispose()` e defina a referência como `null` quando não precisar mais de um objeto BitmapData para que a memória seja liberada imediatamente.

Nota: *O Flash Player 10.1 e o AIR 1.5.2 introduz um novo método chamado `disposeXML()` na classe System. Este método permite que você torne um objeto XML disponível imediatamente para coleta de lixo, ao passar a árvore XML como um parâmetro.*

Mais tópicos da Ajuda

"Congelando e descongelando objetos" na página 28

Uso de bitmaps

Usar vetores ao invés de bitmaps é uma boa maneira de poupar memória. Entretanto, usar vetores, especialmente em grande quantidade, aumenta drasticamente a necessidade de recursos da CPU ou da GPU. Usar bitmaps é uma boa maneira de otimizar a renderização, visto que o tempo de execução precisa de menos recursos de processamento para desenhar pixels na tela do que para renderizar o conteúdo do vetor.

Mais tópicos da Ajuda

"Armazenamento manual de bitmaps em cache" na página 62

Redução da resolução do bitmap

Para usar melhor a memória, imagens opacas de 32 bits são reduzidas para imagens de 16 bits quando o Flash Player detecta uma tela de 16 bits. Essa redução de resolução consome metade dos recursos de memória, e as imagens são renderizadas mais rapidamente. Este recurso está disponível somente no Flash Player 10.1 para Windows Mobile.

Nota: *Antes do Flash Player 10.1, todos os pixels criados na memória eram armazenados em 32 bits (4 bytes). Um logotipo simples de 300 x 300 pixels consumia 350 KB de memória (300*300*4/1024). Com este novo comportamento, o mesmo logotipo opaco consome somente 175 KB. Se o logotipo for transparente, sua resolução não será reduzida para 16 bits e ele manterá o mesmo tamanho na memória. Este recurso se aplica apenas a bitmaps incorporados ou imagens carregadas em tempo de execução (PNG, GIF, JPG).*

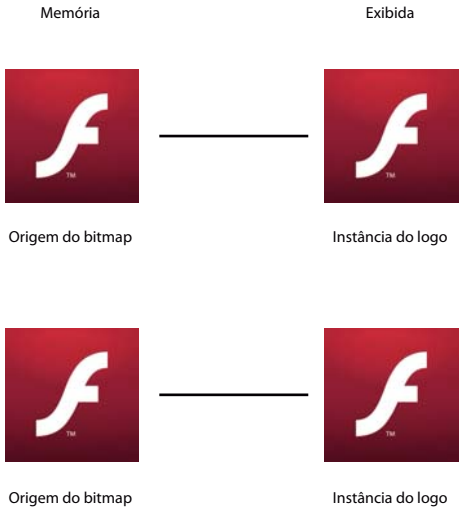
Em dispositivos móveis, não é fácil distinguir uma imagem renderizada em 16 bits da mesma imagem renderizada em 32 bits. Para uma imagem simples que contém apenas algumas cores, não há diferença perceptível. Mesmo para uma imagem mais complexa, é difícil detectar as diferenças. Entretanto, há alguma degradação da cor quando se aplica zoom à imagem, e um gradiente de 16 bits pode parecer menos suave do que a versão de 32 bits.

Referência única de BitmapData

É importante otimizar o uso da classe BitmapData pela reutilização de ocorrências sempre que possível. O Flash Player 10.1 e o AIR 2.5 introduzem um novo recurso para todas as plataformas chamado referência única de BitmapData. Quando criamos ocorrências BitmapData de uma imagem incorporada, uma versão única do bitmap é utilizada para todas as ocorrências de BitmapData. Se um bitmap for modificado posteriormente, recebe seu próprio bitmap exclusivo na memória. A imagem incorporada pode ser da biblioteca ou uma tag [Embed].

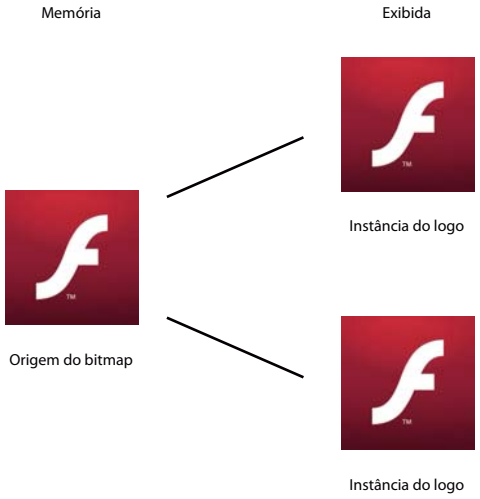
Nota: *O conteúdo existente também se beneficia deste novo recurso, visto que o Flash Player 10.1 e o AIR 2.5 reutilizam bitmaps automaticamente.*

Quando estiver criando uma instância de uma imagem incorporada, um bitmap associado é criado na memória. Antes do Flash Player 10.1 e do AIR 2.5, cada instância recebia um bitmap separado na memória, conforme mostrado no seguinte diagrama:



Bitmaps na memória antes do Flash Player 10.1 e do AIR 2.5

No Flash Player 10.1 e no AIR 2.5, quando várias instâncias da mesma imagem são criadas, um única versão do bitmap é usada para todas as instâncias de BitmapData. O diagrama a seguir ilustra este conceito:

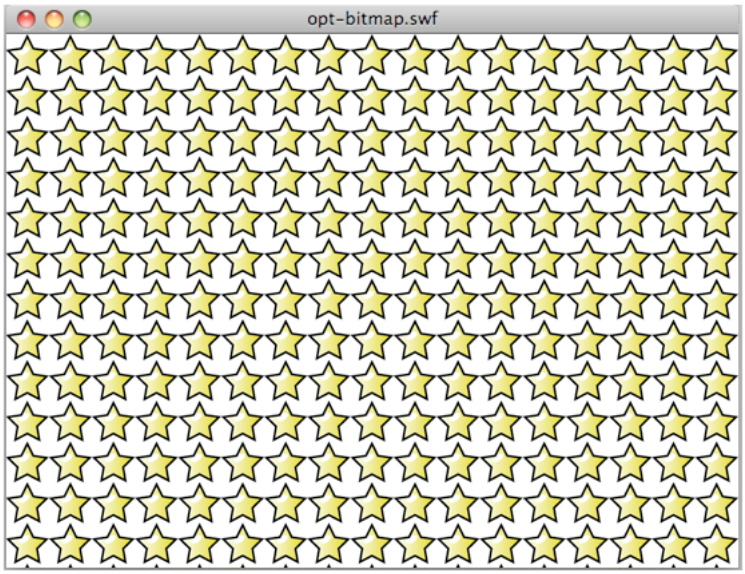


Bitmaps na memória no Flash Player 10.1 e AIR 2.5

Esta abordagem reduz drasticamente a quantidade de memória usada por um aplicativo com muitos bitmaps. O código a seguir cria várias instâncias de um símbolo `Star`:

```
const MAX_NUM:int = 18;

var star:BitmapData;
var bitmap:Bitmap;

for (var i:int = 0; i<MAX_NUM; i++)
{
    for (var j:int = 0; j<MAX_NUM; j++)
    {
        star = new Star(0,0);
        bitmap = new Bitmap(star);
        bitmap.x = j * star.width;
        bitmap.y = i * star.height;
        addChild(bitmap)
    }
}
```

A imagem a seguir mostra o resultado do código:



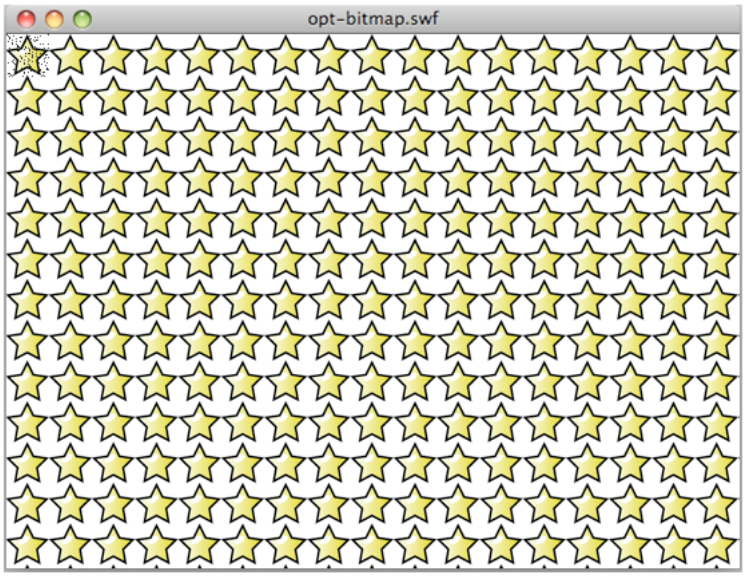
Resultado do código para criar várias instâncias do símbolo

Com o Flash Player 10, por exemplo, a animação acima usa cerca de 1008 KB de memória. Com o Flash Player 10.1, no desktop e em um dispositivo móvel, a animação usa somente 4 KB.

O código a seguir modifica uma instância de BitmapData:

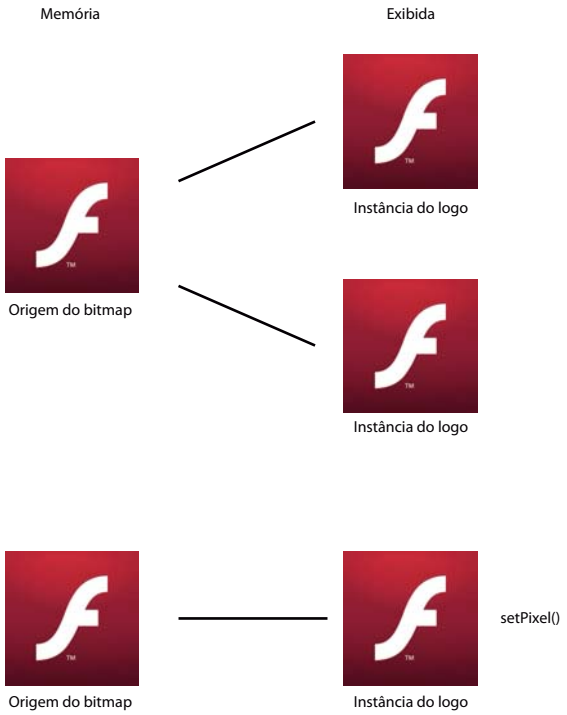
```
const MAX_NUM:int = 18;

var star:BitmapData;
var bitmap:Bitmap;

for (var i:int = 0; i<MAX_NUM; i++)
{
    for (var j:int = 0; j<MAX_NUM; j++)
    {
        star = new Star(0,0);
        bitmap = new Bitmap(star);
        bitmap.x = j * star.width;
        bitmap.y = i * star.height;
        addChild(bitmap)
    }
}

var ref:Bitmap = getChildAt(0) as Bitmap;

ref.bitmapData.pixelDissolve(ref.bitmapData, ref.bitmapData.rect, new
Point(0,0),Math.random()*200,Math.random()*200, 0x990000);
```

A imagem a seguir mostra o resultado da modificação de uma instância de `Star`:



Resultado da modificação de uma instância

Internamente, o tempo de execução automaticamente atribui e cria um bitmap na memória para lidar com as modificações dos pixels. Quando um método da classe BitmapData é chamado, causando modificações de pixel, uma nova ocorrência é criada na memória e nenhuma outra ocorrência é atualizada. A figura a seguir ilustra o conceito:



Resultado na memória da modificação de um bitmap

Se uma estrela é modificada, uma nova cópia é criada na memória. A animação resultante usa cerca de 8 KB na memória no Flash Player 10.1 e no AIR 2.5.

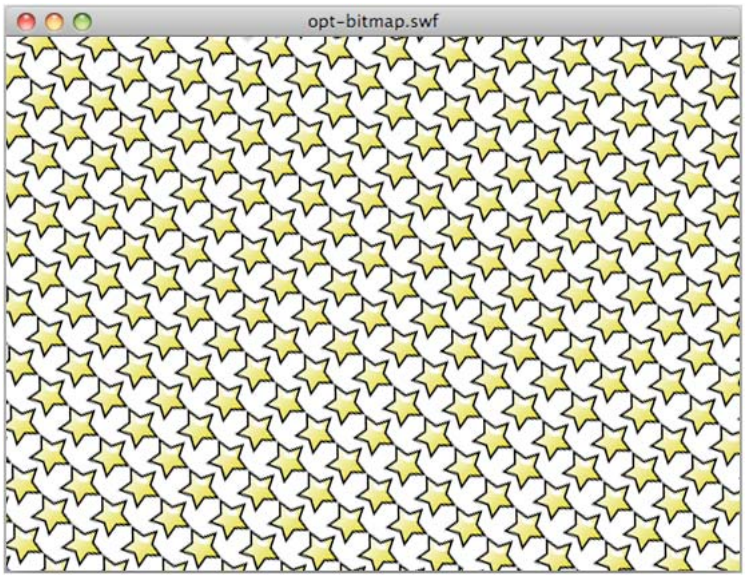
No exemplo anterior, cada bitmap está individualmente disponível para transformação. Para criar apenas o efeito lado a lado, o método `beginBitmapFill()` é o mais apropriado:

```
var container:Sprite = new Sprite();

var source:BitmapData = new Star(0,0);

// Fill the surface with the source BitmapData
container.graphics.beginBitmapFill(source);
container.graphics.drawRect(0,0,stage.stageWidth,stage.stageHeight);

addChild(container);
```

Esta abordagem produz o mesmo resultado com a criação de uma única instância de BitmapData. Para girar as estrelas continuamente, ao invés de acessar cada ocorrência Estrela, utilize um objeto Matriz que é rotacionado em cada quadro. Forneça este objeto Matriz para o método `beginBitmapFill()`:

```
var container:Sprite = new Sprite();

container.addEventListener(Event.ENTER_FRAME, rotate);

var source:BitmapData = new Star(0,0);
var matrix:Matrix = new Matrix();

addChild(container);

var angle:Number = .01;

function rotate(e:Event):void
{
    // Rotate the stars
    matrix.rotate(angle);

    // Clear the content
    container.graphics.clear();

    // Fill the surface with the source BitmapData
    container.graphics.beginBitmapFill(source,matrix,true,true);
    container.graphics.drawRect(0,0,stage.stageWidth,stage.stageHeight);
}
```

Utilizando esta técnica, nenhuma repetição ActionScript será necessária para criar o efeito. O tempo de execução faz tudo internamente. A imagem a seguir mostra o resultado da transformação das estrelas:



Resultado das estrelas girando

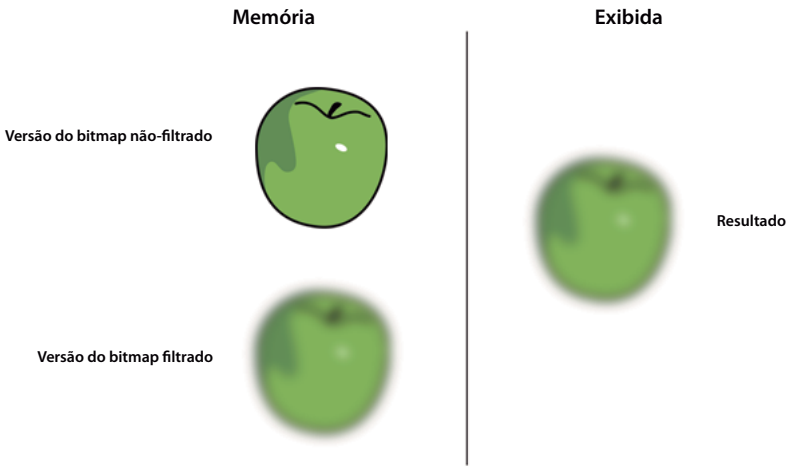
Com esta abordagem, atualizar o objeto BitmapData original atualiza automaticamente seu uso em outros locais do palco, o que pode ser uma técnica poderosa. Esta abordagem não permitiria, contudo, que cada estrela fosse dimensionada individualmente, como no exemplo anterior.

Nota: *Quando várias instâncias da mesma imagem são usadas, o desenho depende de se há uma classe associada ao bitmap original na memória. Se nenhuma classe estiver associada ao bitmap, as imagens serão desenhadas como objetos Shape com preenchimentos de bitmap.*

Filtros e descarregamento dinâmico de bitmap

Evite filtros, inclusive aqueles processados pelo Pixel Bender.

Tente reduzir a utilização de efeitos como filtros, incluindo filtros processados em dispositivos portáteis por meio do Pixel Bender. Quando um filtro é aplicado a um objeto de exibição, o tempo de execução cria dois bitmaps na memória. Esses bitmaps têm, cada um, o tamanho do objeto de exibição. O primeiro é criado como uma versão rasterizada do objeto de exibição que, por sua vez, é usado para produzir um segundo bitmap com o filtro aplicado:



Dois bitmaps na memória quando o filtro é aplicado

Quando uma das propriedades de um filtro é modificada, ambos os bitmaps são atualizados na memória para criar o bitmap resultante. Esse processo envolve algum processamento da CPU e os dois bitmaps podem usar uma quantidade considerável de memória.

O Flash Player 10.1 e o AIR 2.5 introduzem um novo comportamento de filtragem em todas as plataformas. Se o filtro não for modificado em até 30 segundos, ou se for ocultado ou estiver fora da tela, a memória usada pelo bitmap sem filtro é liberada.

Este recurso economiza metade da memória usada pelo filtro em todas as plataformas. Por exemplo, considere um objeto de texto com um filtro de desfoque aplicado. O texto nesse caso é usado para simples decoração e não é modificado. Após 30 segundos, o bitmap sem filtro é liberado da memória. O mesmo resultado ocorre se o texto for ocultado por 30 segundos ou ficar fora da tela. Quando uma das propriedades do filtro for modificada, o bitmap sem filtro é recriado na memória. Este recurso é chamado carregamento dinâmico de bitmap. Mesmo com essas otimizações, tome cuidado com os filtros; eles ainda requerem extenso processamento da CPU ou da GPU quando modificados.

Como uma boa prática, utilize bitmaps criados com uma ferramenta de autoria como o Adobe® Photoshop®, para emular filtros quando possível. Evite utilizar bitmaps dinâmicos criados durante o tempo de execução no ActionScript. Usar bitmaps criados externamente ajuda o tempo de execução a reduzir a carga da CPU ou da GPU, especialmente quando as propriedades do filtro não mudam com o tempo. Se possível, crie quaisquer efeitos que você precise em um bitmap com uma ferramenta de autoria. Você pode então exibir o bitmap no tempo de execução sem executar qualquer processamento nele, o que pode ser muito mais rápido.

Mapeamento MIP direto

Utilize mapeamento MIP para dimensionar imagens grandes, caso necessário.

Outro novo recurso disponível no Flash Player 10.1 e no AIR 2.5 em todas as plataformas está relacionado ao mipmap. O Flash Player 9 e o AIR 1.0 introduziram um recurso de mipmap que aprimorou a qualidade e o desempenho de bitmaps com resolução reduzida.

Nota: *O recurso de mapeamento MIP se aplica apenas para imagens dinamicamente carregadas ou bitmaps incorporadas. O mapeamento MIP não se aplica a objetos de exibição que tenham sido filtrados ou armazenados em cache. O mapeamento MIP pode ser processado somente se o bitmap tiver uma largura e uma altura que sejam números pares. Quando uma largura ou altura com número ímpar for encontrada, o mapeamento MIP é interrompido. Por exemplo, uma imagem de 250 x 250 pode ser mapeada até 125 x 125, mas seu mapeamento MIP é interrompido nesse ponto. Neste caso, pelo menos uma das dimensões é um número ímpar. Bitmaps com dimensões que forem potências de dois obtêm os melhores resultados como, por exemplo: 256 x 256, 512 x 512, 1024 x 1024, e assim por diante.*

Por exemplo, imagine que uma imagem de 1024 x 1024 seja carregada, e um desenvolvedor queira escalar a imagem para criar uma miniatura em uma galeria. O recurso de mapeamento MIP renderiza a imagem corretamente quando escalada pelo uso das versões intermediárias com resolução reduzida do bitmap como texturas. Versões anteriores do tempo de execução criavam versões intermediárias com resolução reduzida do bitmap na memória. Se uma imagem de 1024 x 1024 fosse carregada e exibida como 64 x 64, as versões mais antigas do tempo de execução criariam cada bitmap de metade do tamanho. Por exemplo, neste caso seriam criados bitmaps de 512 x 512, 256 x 256, 128 x 128 e 64 x 64.

O Flash Player 10.1 e o AIR 2.5 agora oferecem suporte ao mipmap diretamente da fonte original para o tamanho de destino necessário. No exemplo anterior, somente o bitmap original de 4 MB (1024 x 1024) e o bitmap com mapeamento MIP de 16 KB (64 x 64) seriam criados.

A lógica do mapeamento MIP também funciona com o recurso de carregamento dinâmico de bitmap. Se somente o bitmap de 64 x 64 for usado, o bitmap original de 4 MB será liberado da memória. Se for preciso recriar o mapa MIP, o original será recarregado. Além disso, se outros bitmaps com mapeamento MIP de vários tamanhos forem necessários, a cadeia de bitmaps de mapas MIP será usada para criar o bitmap. Por exemplo, se for preciso criar um bitmap de 1:8, os bitmaps de 1:4, 1:2 e 1:1 serão examinados para determinar qual deve ser carregado na memória primeiro. Se nenhuma outra versão for encontrada, o bitmap original de 1:1 será carregado a partir do recurso e usado.

O descompactador de JPEG pode executar o mapeamento MIP em seu próprio formato. Esse mapeamento MIP direto permite que um bitmap grande seja descompactado diretamente para um formato de mapa MIP, sem a necessidade de carregar toda a imagem descompactada. A geração do mapa MIP é consideravelmente mais rápida, e a memória usada por bitmaps grandes não é alocada e então liberada. A qualidade da imagem JPEG é comparável à da técnica geral de mapeamento MIP.

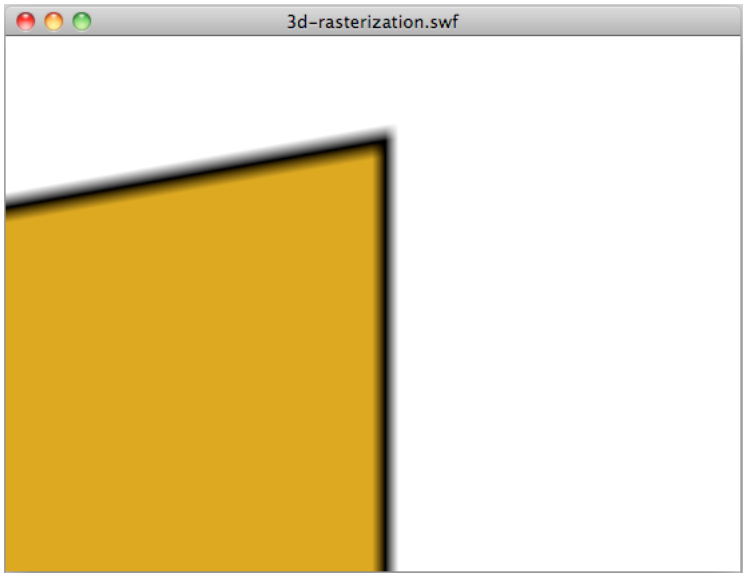
Nota: *Usar mapeamento MIP com moderação. Embora ele melhore a qualidade de bitmaps reduzidos, tem um impacto sobre a largura de banda, memória e velocidade. Em alguns casos, uma opção melhor pode ser utilizar um versão pré dimensionada do bitmap de uma ferramenta externa e importá-la em sua aplicação. Não inicie com bitmaps grandes se você apenas pretende dimensioná-los para baixo.*

Uso de efeitos 3D

Considere criar efeitos 3D manualmente.

O Flash Player 10 e o AIR 1.5 introduzam um mecanismo 3D, o que permite aplicar transformação de perspectiva a objetos de exibição. Você pode aplicar essas transformações usando as propriedades `rotationX` e `rotationY` ou com o método `drawTriangles()` da classe Graphics. Também é possível aplicar profundidade com a propriedade `z`. Lembre-se de que cada objeto de exibição com a perspectiva transformada é rasterizado como um bitmap e, portanto, requer mais memória.

A figura a seguir ilustra a suavização de borda produzida pela rasterização quando se usa a transformação de perspectiva:



Suavização de borda resultante da transformação de perspectiva

A suavização é um resultado do conteúdo de vetores sendo dinamicamente rasterizados como um bitmap. Essa suavização de borda acontece quando você usa efeitos 3D na versão para desktop do AIR e do Flash Player, e no AIR 2.0.1 e AIR 2.5 para celular. No entanto, a suavização de borda não é aplicada no Flash Player para dispositivos móveis.

Se você puder criar seu efeito 3D manualmente sem depender da API nativa, isso poderá reduzir o uso da memória. No entanto, os novos recursos 3D introduzidos no Flash Player 10 e no AIR 1.5 facilitam o mapeamento de texturas, graças a métodos como `drawTriangles()`, que lidam de maneira nativa com o mapeamento de texturas.

Como desenvolvedor, decida se o efeito 3D que deseja criar oferece melhor desempenho se processado por meio da API nativa ou manualmente. Considere a execução do ActionScript e o desempenho da renderização, bem como o uso de memória.

Em aplicativos móveis do AIR 2.0.1 e do AIR 2.5 em que você define a propriedade `renderMode` do aplicativo como `GPU`, a GPU faz as transformações 3D. No entanto, se o `renderMode` for `CPU`, a CPU, e não a GPU, executará as transformações 3D. Em aplicativos Flash Player 10.1, a CPU executa as transformações 3D.

Quando a CPU fizer as transformações 3D, considere que aplicar qualquer transformação 3D a um objeto de exibição exige dois bitmaps na memória. Um bitmap é para o bitmap de origem e o segundo pra a versão com a perspectiva transformada. Desta forma, transformações 3D funcionam de maneira similar aos filtros. Como resultado, use as propriedades 3D com moderação quando a CPU fizer as transformações 3D.

Objetos de texto e memória

Use o Adobe® Flash® Text Engine para texto somente leitura; use objetos TextField para texto de entrada.

O Flash Player 10 e o AIR 1.5 introduziram um novo e poderoso mecanismo de texto, o Adobe Flash Text Engine (FTE), que economiza a memória do sistema. Entretanto, o FTE é uma API de baixo nível que requer uma camada adicional do ActionScript 3.0 sobre ele, fornecida no pacote flash.text.engine.

Para texto somente leitura, é melhor usar o Flash Text Engine, que oferece baixo uso de memória e melhor renderização. Para texto de entrada, objetos TextField são a melhor opção, visto que menos código ActionScript é necessário para criar comportamentos típicos, como manipulação de entradas e quebra automática de linha.

Mais tópicos da Ajuda

"Renderização de objetos de texto" na página 68

Modelo de evento versus retorno

Considere utilizar retornos simples, ao invés de modelo de evento.

O modelo de evento do ActionScript 3.0 é baseado no conceito de envio de objeto. O modelo de evento é orientado a objeto e otimizado para reutilização de código. O método `dispatchEvent()` se repete através da lista de ouvintes e chama o método manipulador de evento em cada objeto registrado. No entanto, uma das desvantagens ode modelo de evento é que você provavelmente criará muitos objetos durante o tempo de vida de sua aplicação.

Imagine que você precisa enviar um evento da linha do tempo, indicando o fim de uma sequência de animação. Para concluir a notificação, você pode enviar um evento de um quadro especifico na linha do tempo, como ilustra o código a seguir:

```
dispatchEvent( new Event ( Event.COMPLETE ) );
```

A classe Documento pode ouvir este evento com a seguinte linha de código:

```
addEventListener( Event.COMPLETE, onAnimationComplete );
```

Embora esta aproximação esteja correta, utilizar o modelo de evento nativo pode ser mais lento e consumir mais memória que uma função de retorno tradicional. Objetos de evento devem ser criados e alocados na memória, o que cria uma redução no desempenho. Por exemplo, quando estiver ouvindo o evento Event.ENTER_FRAME, um novo objeto de evento é criado em cada quadro para o manipulador de evento. O desempenho pode ser especialmente lento para exibir objetos, por causa das fases de captura e bolha o que pode ser dispendioso se a lista de exibição for complexa.

Capítulo 3: Minimização do uso da CPU

Outra importante área de foco para otimização é o uso da CPU. A otimização do processamento de CPU aumenta o desempenho, e como resultado, o ciclo de vida da bateria em dispositivos remotos.

Aprimoramentos do Flash Player 10.1 para uso da CPU

O Flash Player 10.1 introduz dois novos recursos que ajudam a economizar processamento da CPU. Os recursos envolvem pausar e retomar conteúdo SWF quando este estiver fora da tela e limitar o número de instâncias do Flash Player em uma página.

Pausa, limitação e retomada

Nota: *O recurso de pausa, limitação e retomada não se aplica aos aplicativos do Adobe® AIR®.*

Para otimizar o uso da CPU e da bateria, o Flash Player 10.1 introduz um novo recurso relacionado a instâncias inativas. Este recurso permite que você limite o uso da CPU pausando e reiniciando o arquivo SWF quando o conteúdo chegar ao fim na tela. Com este recurso, o Flash Player libera tanta memória quanto possível removendo quaisquer objetos que possam ser recriados quando a reprodução do conteúdo é retomada. O conteúdo é considerado fora da tela quando todo o conteúdo estiver fora da tela.

Dois cenários fazem com que o conteúdo SWF fique fora da tela:

- O usuário rola a página e faz com que o conteúdo SWF se mova para fora da tela.

  Neste caso, se houver alguma reprodução de áudio ou vídeo, o conteúdo continuará sendo reproduzido, mas a renderização irá parar. Se não houver nenhuma reprodução de áudio ou vídeo, para assegurar que a reprodução ou a execução do ActionScript não sejam pausada, defina o parâmetro de HTML `hasPriority` como verdadeiro. Entretanto, lembre-se de que a renderização do conteúdo SWF será pausada quando o conteúdo ficar fora da tela ou oculto, independentemente do valor do parâmetro de HTML `hasPriority`.

- Uma guia é aberta no navegador, o que faz com que o conteúdo SWF se mova para o segundo plano.

  Neste caso, independentemente do valor da marca de HTML `hasPriority`, o conteúdo de SWF terá sua velocidade diminuída, ou *limitada*, para 2 a 8 fps. A reprodução de áudio e vídeo é interrompida e nenhum conteúdo que está sendo renderizado é processado, a menos que o conteúdo de SWF se torne visível novamente.

Para o Flash Player 11.2 e posterior executado em navegadores de desktops Windows e Mac, você pode usar o ThrottleEvent no seu aplicativo. O Flash Player despacha um ThrottleEvent quando é pausado, limitado ou retoma a reprodução.

O ThrottleEvent é um evento de transmissão, ou seja, ele é despachado por todos os objetos EventDispatcher com um ouvinte registrado para esse evento. Para obter mais informações sobre eventos de transmissão, consulte a classe DisplayObject.

Gerenciamento de ocorrência

Nota: *O recurso de gerenciamento de instâncias não se aplica aos aplicativos Adobe® AIR®.*

Use o parâmetro HTML `hasPriority` para atrasar o carregamento de arquivos SWF que estejam fora da tela.

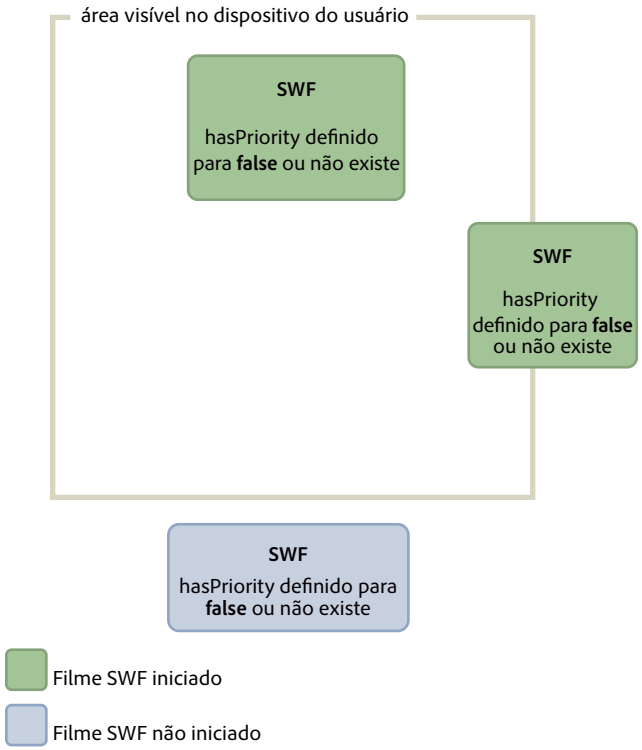
O Flash Player 10.1 introduz um novo parâmetro HTML chamado `hasPriority`:

```
<param name="hasPriority" value="true" />
```

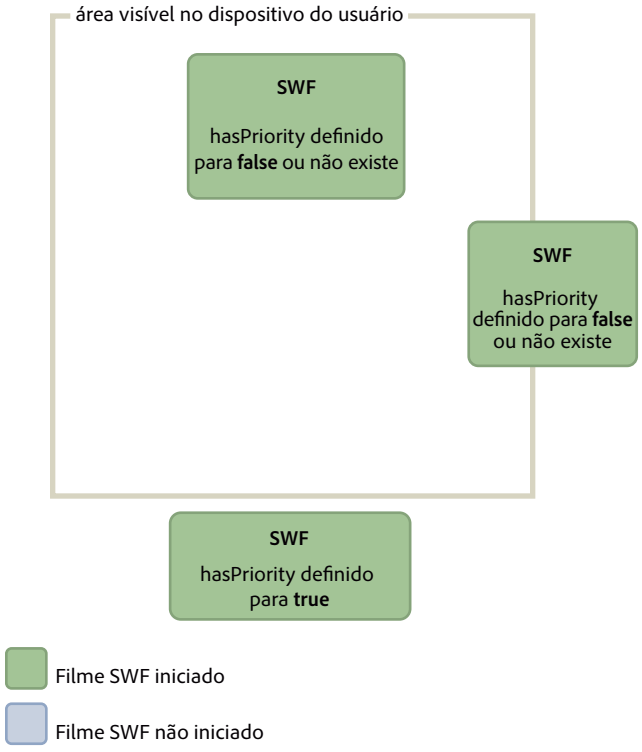
Este recurso limita o número de ocorrências do Flash Player iniciadas em uma página. Limitar o número de ocorrências ajuda a conservar recursos de CPU e bateria. A ideia é atribuir uma prioridade específica ao conteúdo SWF, dando prioridade a algum conteúdo com relação a outros conteúdos na página. Considere um simples exemplo: um usuário está navegando por um site e a página de índice hospeda três arquivos SWF diferentes. Um deles é visível, outro é parcialmente visível na tela e o último fica fora da tela, sendo necessária rolagem. As duas primeiras animações têm início normalmente, mas a última é atrasada até que se torne visível. Este cenário é o comportamento padrão quando o parâmetro `hasPriority`não está presente ou está definida como `false`. Para garantir que um arquivo SWF seja iniciado, mesmo que esteja fora da tela, defina o parâmetro `hasPriority` como `true`. No entanto, independentemente do valor do parâmetro `hasPriority`, um arquivo SWF que não é visível ao usuário terá sua renderização sempre pausada.

Nota: *Se os recursos de CPU disponíveis ficarem baixos, as instâncias do Flash Player não são mais iniciadas automaticamente, mesmo que o parâmetro `hasPriority` esteja definido como `true`. Se novas instâncias forem criadas por meio do JavaScript depois que a página for carregada, essas instâncias ignorarão o sinalizador `hasPriority`. Qualquer conteúdo de 1x1 pixel ou 0x0 pixel é iniciado, o que impede arquivos SWF de ajuda de serem atrasados caso o webmaster deixe de incluir o sinalizador `hasPriority`. No entanto, arquivos SWF podem ainda ser iniciados quando clicados. Esse comportamento é chamado de "clique para reproduzir".*

Os seguintes diagramas mostram os efeitos de se configurar o parâmetro `hasPriority` para valores diferentes:



Efeitos de valores diferentes do parâmetro hasPriority



Efeitos de valores diferentes do parâmetro hasPriority

Modo de suspensão

O Flash Player 10.1 e o AIR 2.5 introduzem um novo recurso em dispositivos móveis que ajuda a economizar processamento da CPU e, como resultado, a vida útil da bateria. Esse recurso envolve a luz de fundo encontrada em muitos dispositivos móveis. Por exemplo, se um usuário que executa um aplicativo móvel for interrompido e parar de usar o dispositivo, o tempo de execução detecta quando a luz e fundo entra em modo de suspensão. Em seguida, ele reduz a taxa de quadros para 4 quadros por segundo (fps) e pausa a renderização. Para aplicativos AIR, o modo de suspensão também começa quando o aplicativo passa para o segundo plano.

O código ActionScript continua a ser executado no modo de suspensão, de modo semelhante a definir a propriedade `Stage.frameRate` como 4 fps. Mas a etapa de renderização é ignorada, de forma que o usuário não possa ver que o player está sendo executado a 4 fps. Uma taxa de quadros de 4 fps foi escolhida em vez de zero, por permitir que todas as conexões permaneçam abertas (NetStream, Socket e NetConnection). Alternar para zero quebraria as conexões abertas. Uma taxa de atualização de 250 ms foi escolhida (4 fps) porque muitos fabricantes de dispositivos usam essa taxa de quadros como taxa de atualização. Usar esse valor mantém a taxa de quadros do tempo de execução com o mesmo valor aproximado do próprio dispositivo.

Nota: *Quando o tempo de execução estiver no modo de suspensão, a propriedade* `Stage.frameRate` *retornará a taxa de quadros do arquivo SWF original, em vez de 4 fps.*

Quando a luz de fundo volta para o modo ligado, a renderização é retomada. A taxa de quadros retorna para seu valor original. Considere um aplicativo de media player em que um usuário está reproduzindo uma música. Se a tela passar para o modo de suspensão, o tempo de execução responderá com base no tipo de conteúdo que está sendo reproduzido. Eis uma lista de situações com o comportamento correspondente do tempo de execução:

- A luz de fundo entra no modo de suspensão e conteúdo que não é de áudio nem vídeo está sendo reproduzido: a renderização é pausada e a taxa de quadros é definida como 4 fps.
- A luz de fundo entra no modo de suspensão e conteúdo de áudio/vídeo está sendo reproduzido: o tempo de execução força a luz de fundo a ficar sempre acesa, dando continuidade à experiência do usuário.
- A luz de fundo passa do modo de suspensão para o modo ligado: o tempo de execução define a taxa de quadros com a configuração de taxa de quadros original do arquivo SWF e retoma a renderização.
- O Flash Player é pausado enquanto conteúdo de áudio/vídeo é reproduzido: o Flash Player redefine o estado da luz de fundo para o comportamento padrão do sistema, já que o áudio/vídeo não está mais em reprodução.
- O dispositivo móvel recebe uma chamada telefônica enquanto conteúdo de áudio/vídeo é reproduzido: a renderização é pausada e a taxa de quadros é definido como 4 fps.
- O modo de suspensão da luz de fundo é desativado em um dispositivo móvel: o tempo de execução se comporta normalmente.

Quando a luz de fundo entra no modo de suspensão, a renderização é pausada e a taxa de quadros é desacelerado. Este recurso economiza o processamento da CPU, mas não se pode depender dele para criar uma pausa real, como em um aplicativo de jogo.

Nota: *Nenhum evento ActionScript é enviado quando o tempo de execução entra ou sai do modo de suspensão.*

Congelando e descongelando objetos

Congelar e descongelar objetos da maneira correta utilizando os eventos `REMOVED_FROM_STAGE` *e* `ADDED_TO_STAGE`*.*

Para otimizar seu código, sempre congele e descongele seus objetos. Congelar e descongelar são importantes para todos os objetos, mas são especialmente importantes para objetos de exibição. Mesmo que os objetos de exibição não estejam mais na lista de exibição e estiverem aguardando serem coletados como lixo, ainda podem estar usando código que consome muita CPU. Por exemplo, ainda podem estar usando Event.ENTER_FRAME. Como resultado, é importante congelar e descongelar os objetos utilizando os eventos `Event.REMOVED_FROM_STAGE` e `Event.ADDED_TO_STAGE`. O exemplo a seguir mostra um clipe de filme reproduzindo em um palco que interage como o teclado:

```
// Listen to keyboard events
stage.addEventListener(KeyboardEvent.KEY_DOWN, keyIsDown);
stage.addEventListener(KeyboardEvent.KEY_UP, keyIsUp);

// Create object to store key states
var keys:Dictionary = new Dictionary(true);

function keyIsDown(e:KeyboardEvent):void
{
    // Remember that the key was pressed
    keys[e.keyCode] = true;

    if (e.keyCode==Keyboard.LEFT || e.keyCode==Keyboard.RIGHT)
    {
        runningBoy.play();
    }
}

function keyIsUp(e:KeyboardEvent):void
{
    // Remember that the key was released
    keys[e.keyCode] = false;

    for each (var value:Boolean in keys)
        if ( value ) return;
    runningBoy.stop();
}

runningBoy.addEventListener(Event.ENTER_FRAME, handleMovement);
runningBoy.stop();

var currentState:Number = runningBoy.scaleX;
var speed:Number = 15;

function handleMovement(e:Event):void
{
    if (keys[Keyboard.RIGHT])
    {
        e.currentTarget.x += speed;
        e.currentTarget.scaleX = currentState;
    } else if (keys[Keyboard.LEFT])
    {
        e.currentTarget.x -= speed;
        e.currentTarget.scaleX = -currentState;
    }
}
```

Clipe que interaje com o teclado

Quando o botão Remover é clicado, o clipe de filme é removido da lista de exibição:

```
// Show or remove running boy
showBtn.addEventListener (MouseEvent.CLICK,showIt);
removeBtn.addEventListener (MouseEvent.CLICK,removeIt);

function showIt (e:MouseEvent):void
{
    addChild (runningBoy);
}

function removeIt(e:MouseEvent):void
{
    if (contains(runningBoy)) removeChild(runningBoy);
}
```

Mesmo quando removido da lista de exibição, o clipe de filme ainda aciona o evento `Event.ENTER_FRAME`. O clipe de filme ainda é executado, mas não renderizado. Para lidar corretamente com esta situação, ouça os eventos apropriados e remova os ouvintes de eventos para evitar que códigos intensos de CPU sejam executados.

```
// Listen to Event.ADDED_TO_STAGE and Event.REMOVED_FROM_STAGE
runningBoy.addEventListener(Event.ADDED_TO_STAGE,activate);
runningBoy.addEventListener(Event.REMOVED_FROM_STAGE,deactivate);

function activate(e:Event):void
{
    // Restart everything
    e.currentTarget.addEventListener(Event.ENTER_FRAME,handleMovement);
}

function deactivate(e:Event):void
{
    // Freeze the running boy - consumes fewer CPU resources when not shown
    e.currentTarget.removeEventListener(Event.ENTER_FRAME,handleMovement);
    e.currentTarget.stop();
}
```

Quando o botão Exibir é pressionado, o clipe de filme é reiniciado ele volta a ouvir eventos `Event.ENTER_FRAME` novamente e o teclado controla o corretamente o clipe de filme.

Nota: *Se um objeto de exibição for removido da lista de exibição, definir sua referência como* `null` *após a remoção não garante que o objeto seja congelado. Se o coletor de lixo não for executado, o objeto continua a consumir memória e processamento da CPU, mesmo que o objeto não seja mais exibido. Para garantir que o objeto consuma o mínimo de processamento de CPU possível, certifique-se de congelá-lo completamente ao removê-lo da lista de exibição.*

A partir do Flash Player 10 e do AIR 1.5, o comportamento a seguir também ocorre. Se a cabeça de reprodução encontrar um quadro vazio, o objeto de exibição é congelado automaticamente, mesmo que você não tenha implementado nenhum comportamento de congelamento.

O conceito de congelamento também é importante ao carregar conteúdo remoto com a classe Loader. Ao usar a classe Loader com o Flash Player 9 e o AIR 1.0, era necessário congelar manualmente o conteúdo ouvindo o evento `Event.UNLOAD` emitido pelo objeto LoaderInfo LoaderInfo. Cada objeto precisava ser congelado manualmente, o que não era uma tarefa trivial. O Flash Player 10 e o AIR 1.5 introduzem um importante novo método na classe Loader chamado `unloadAndStop()`. Esse método permite descarregar um arquivo SWF, congelar automaticamente cada objeto no arquivo SWF carregado e forçar a execução do coletor de lixo.

No código a seguir, o arquivo SWF é carregado e então descarregado pelo método `unload()`, que requer mais processamento e congelamento manual:

```
var loader:Loader = new Loader();

loader.load ( new URLRequest ( "content.swf" ) );

addChild ( loader );

stage.addEventListener ( MouseEvent.CLICK, unloadSWF );

function unloadSWF ( e:MouseEvent ):void
{
    // Unload the SWF file with no automatic object deactivation
    // All deactivation must be processed manually
    loader.unload();
}
```

Uma melhor prática é usar o método `unloadAndStop()`, que lida com o congelamento nativamente e força a execução do processo de coleta de lixo:

```
var loader:Loader = new Loader();

loader.load ( new URLRequest ( "content.swf" ) );

addChild ( loader );

stage.addEventListener ( MouseEvent.CLICK, unloadSWF );

function unloadSWF ( e:MouseEvent ):void
{
    // Unload the SWF file with automatic object deactivation
    // All deactivation is handled automatically
    loader.unloadAndStop();
}
```

As ações a seguir ocorrem quando o método `unloadAndStop()` é chamado:

- Os sons são parados.
- Os ouvintes registrados na linha de tempo principal do arquivo SWF são removidos.
- Objetos Timer são parados.
- Dispositivos periféricos de hardware (como câmera e microfone) são liberados.
- Cada clipe de filme é parado.
- O acionamento de `Event.ENTER_FRAME`, `Event.FRAME_CONSTRUCTED`, `Event.EXIT_FRAME`, `Event.ACTIVATE` e `Event.DEACTIVATE` é parado.

Ativar e desativar eventos

Use os eventos `Event.ACTIVATE` e `Event.DEACTIVATE` para detectar inatividade em segundo plano e otimizar seu aplicativo devidamente.

Dois eventos (`Event.ACTIVATE` e `Event.DEACTIVATE`) podem ajudar você a fazer ajustes finos em seu aplicativo, de modo a usar o menor número de ciclos da CPU possível. Esses eventos permitem detectar quando o tempo de execução ganha ou perde o foco. Como resultado, o código pode ser otimizado de modo a reagir a alterações de contexto. O código a seguir ouve ambos os eventos e altera dinamicamente a taxa de quadros para zero quando o aplicativo perde o foco. Por exemplo, a animação pode perder o foco quando o usuário alternar para outra guia ou colocar o aplicativo em segundo plano:

```
var originalFrameRate:uint = stage.frameRate;
var standbyFrameRate:uint = 0;

stage.addEventListener ( Event.ACTIVATE, onActivate );
stage.addEventListener ( Event.DEACTIVATE, onDeactivate );

function onActivate ( e:Event ):void
{
    // restore original frame rate
    stage.frameRate = originalFrameRate;
}

function onDeactivate ( e:Event ):void
{
    // set frame rate to 0
    stage.frameRate = standbyFrameRate;
}
```

Quando o aplicativo volta a receber o foco, a taxa de quadros é redefinida para seu valor original. Em vez de mudar a taxa de quadros dinamicamente, você também pode considerar fazer outras otimizações, como o congelamento e o descongelamento de objetos.

Os eventos activate e deactivate permitem implementar um mecanismo semelhante no recurso "Pausar e retomar" algumas vezes encontrado em dispositivos móveis e netbooks.

Mais tópicos da Ajuda

"Taxa de quadros do aplicativo" na página 53

"Congelando e descongelando objetos" na página 28

Interação com mouse

Considere desabilitar interações como mouse, quando possível.

Quando estiver usando um objeto interativo, como um objeto MovieClip ou Sprite, o tempo de execução executa código nativo para detectar e lidar com interações de mouse. A detecção de interações de mouse pode ser intensivo para a CPU quando muitos objetos interativos são exibidos na tela, especialmente se eles se sobrepõe. Uma maneira fácil de evitar este processamento é desabilitar interações de mouse em objetos que não precisam de interações do mouse. O código a seguir ilustra o uso das propriedades `mouseEnabled` e `mouseChildren`:

```
// Disable any mouse interaction with this InteractiveObject
myInteractiveObject.mouseEnabled = false;
const MAX_NUM:int = 10;

// Create a container for the InteractiveObjects
var container:Sprite = new Sprite();

for ( var i:int = 0; i< MAX_NUM; i++ )
{
    // Add InteractiveObject to the container
    container.addChild( new Sprite() );
}

// Disable any mouse interaction on all the children
container.mouseChildren = false;
```

Quando possível, considere desabilitar as interações de mouse, o que ajuda seu aplicativo a usar menos processamento de CPU e, como resultado, reduz o uso da bateria.

Timers X eventos ENTER_FRAME

Escolha entre temporizadores ou eventos `ENTER_FRAME`, dependendo de qual conteúdo é animado.

Temporizadores tem preferência sobre eventos `Event.ENTER_FRAME` para conteúdo não animado executados por longo tempo.

No ActionScript 3.0, existem duas maneiras de chamar uma função em intervalos específicos. A primeira abordagem é utilizar o evento `Event.ENTER_FRAME` despachado por objetos de exibição (DisplayObject). A segunda aproximação é usar um temporizador. Desenvolvedores de ActionScript usam com frequência a aproximação de evento `ENTER_FRAME`. O evento `ENTER_FRAME` é enviado em cada quadro. Como resultado, o intervalo no qual a função é chamada esta relacionada com o quadro a taxa de quadros atual. A taxa de quadros pode ser acessada através da propriedade `Stage.frameRate`. No entanto, em alguns casos, a utilização de um temporizador pode ser uma escolha melhor do sobre o evento `ENTER_FRAME`. Por exemplo, se você não utilizar animação, mas quiser que seu código seja chamado em intervalos específicos, a utilização de um timer pode ser uma escolha melhor.

Um temporizador pode se comportar de uma maneira similar a um evento `ENTER_FRAME`, mas um evento pode ser enviado sem estar atrelado a taxa de quadros. Este comportamento pode oferecer alguma otimização significativa. Considere uma aplicação de reprodutor de vídeo como um exemplo. Neste caso, você não precisa utilizar uma taxa de quadros alta, porque apenas os controles do aplicativo estão se movendo.

Nota: *A taxa de quadros não afeta o vídeo, porque o vídeo não está incorporado à linha do tempo. Em vez disso, o vídeo é carregado dinamicamente através de download progressivo ou streaming.*

Neste exemplo, a taxa de quadros é definido para um valor baixo de 10 fps. O temporizador atualiza os controles a uma taxa de uma atualização por segundo. A taxa de atualização mais alta é possível por conta do método `updateAfterEvent()`, que está disponível no objeto TimerEvent. Este método força a tela a ser atualizada a cada vez que o temporizador envia um evento, caso seja necessário. O código a seguir ilustra a ideia:

```
// Use a low frame rate for the application
stage.frameRate = 10;

// Choose one update per second
var updateInterval:int = 1000;
var myTimer:Timer = new Timer(updateInterval,0);

myTimer.start();
myTimer.addEventListener( TimerEvent.TIMER, updateControls );

function updateControls( e:TimerEvent ):void
{
    // Update controls here
    // Force the controls to be updated on screen
    e.updateAfterEvent();
}
```

Chamar o método `updateAfterEvent()` não modifica a taxa de quadros. Isto apenas força o tempo de execução a atualizar o conteúdo que mudou na tela. A linha do tempo continua em 10 fps. Lembre-se que temporizadores e eventos `ENTER_FRAME` não são muito precisos em dispositivos de baixo desempenho, ou se as funções do manipulador de eventos contiverem código que necessita processamento pesado. Assim como a taxa de quadros do arquivo SWF, a taxa de atualização de quadros do temporizador pode variar em algumas situações.

Minimize o número de objetos do cronômetro e operadores `enterFrame` registrados no aplicativo.

Em cada quadro, o tempo de execução despacha o evento `enterframe` para cada objeto de exibição na lista de exibição. Embora você possa registrar os ouvintes para o evento `enterFrame` com múltiplos objetos de exibição, isso indica que mais códigos são executados em cada quadro. Em vez disso, considere a utilização de um operador simples centralizado `enterframe` que executa todos os códigos para executar cada quadro. Centralizando esse código, é mais fácil gerenciar todos os códigos que executam com frequência.

De forma semelhante, se estiver utilizando objetos do cronômetro, há sobrecarga associada à criação e ao despacho de eventos de múltiplos objetos do cronômetro. Se você ativar diferentes operações em intervalos diferentes, aqui estão algumas sugestões de alternativas:

- Utilize o número mínimo de objetos do cronômetro e operações de grupo de acordo com a frequência em que ocorrem .

  Por exemplo, utilize um cronômetro em operações frequentes; defina para acionar a cada 100 milissegundos. Utilize outro cronômetro para operações menos frequentes ou em segundo plano, defina para acionar a cada 2.000 milissegundos.

- Utilize um objeto cronômetro simples e faça as operações acionarem em múltiplos do intervalo da propriedade `delay` do objeto cronômetro.

  Por exemplo, suponha que você tem algumas operações que espera que ocorram a cada 100 milissegundos e outras que deseja que ocorram a cada 200 milissegundos. Nesse caso, utilize um objeto cronômetro simples com valor `delay` de 100 milissegundos. No operador de eventos `timer`, adicione a instrução condicional que somente executa operações de 200 milissegundos a cada hora. O exemplo abaixo demonstra esta técnica:

```
var timer:Timer = new Timer(100);
timer.addEventListener(TimerEvent.Timer, timerHandler);
timer.start();

var offCycle:Boolean = true;

function timerHandler(event:TimerEvent):void
{
    // Do things that happen every 100 ms

    if (!offCycle)
    {
        // Do things that happen every 200 ms
    }

    offCycle = !offCycle;
}
```

Pare os objetos do cronômetro quando não estão em uso.

Se um operador de eventos `timer` do objeto Timer desempenhar somente operações em certas condições, invoque o método `stop()` do objeto Timer se nenhuma condição for verdadeira.

No evento `enterFrame` ou nos operadores do cronômetro, diminua o número de alterações na aparência dos objetos de exibição que fazem com que a tela seja redesenhada.

Em cada quadro, a fase de renderização redesenha a parte da fase que foi alterada durante esse quadro. Se a área redesenhada for grande, ou se for pequena mas tiver uma grande quantidade de objetos ou objetos de exibição complexos, o tempo de execução precisa de mais tempo para a renderização. Para testar a quantidade de redesenho necessário, utilize o recurso "mostrar regiões de redesenho" na depuração no Flash Player ou AIR.

Para obter mais informações sobre como melhorar o desempenho para ações repetidas, consulte o seguinte artigo:

- Escrevendo aplicativos do AIR bem comportados e eficientes (artigo e aplicativo de exemplo por Arno Gourdol)

Mais tópicos da Ajuda

"Comportamentos de isolamento" na página 65

Síndrome de interpolação

Para poupar potência de CPU, limite o uso de interpolação, o que poupa processamento de CPU, memória e duração de bateria.

Designers e desenvolvedores produzindo conteúdo para Flash no desktop tendem a utilizar muita interpolação de movimento em seus aplicativos. Quando você estiver produzindo conteúdo para dispositivos móveis de baixo desempenho, tente minimizar o uso de interpolação de movimentos. Limitar o seu uso ajuda o conteúdo a ser executado mais rápido em dispositivos menos sofisticados.

Capítulo 4: Desempenho do ActionScript 3.0

Classe Vector x classe Array

Use a classe Vector em vez da classe Array, sempre que possível.

A classe Vector permite acesso mais rápido para leitura e gravação do que a classe Array.

Um simples benchmark mostra os benefícios da classe Vector em relação à classe Array. O código a seguir mostra um benchmark da classe Array:

```
var coordinates:Array = new Array();
var started:Number = getTimer();

for (var i:int = 0; i< 300000; i++)
{
    coordinates[i] = Math.random()*1024;
}

trace(getTimer() - started);
// output: 107
```

O código a seguir mostra um benchmark da classe Vector:

```
var coordinates:Vector.<Number> = new Vector.<Number>();
var started:Number = getTimer();

for (var i:int = 0; i< 300000; i++)
{
    coordinates[i] = Math.random()*1024;
}

trace(getTimer() - started);
// output: 72
```

O exemplo pode ser otimizado ainda mais por meio da atribuição de um comprimento específico ao vetor e da configuração de seu comprimento como fixo:

```
// Specify a fixed length and initialize its length
var coordinates:Vector.<Number> = new Vector.<Number>(300000, true);

var started:Number = getTimer();

for (var i:int = 0; i< 300000; i++)
{
    coordinates[i] = Math.random()*1024;
}

trace(getTimer() - started);
// output: 48
```

Caso o tamanho do vetor não seja definido antes, o tamanho aumenta quando o vetor estiver sem espaço. Cada vez que o tamanho do vetor aumenta, um novo bloco de memória é alocado. O conteúdo atual do vetor é copiado para o novo bloco de memória. Esta alocação e cópia extra de dados diminuem o desempenho. O código acima é otimizado ao especificar o tamanho inicial do vetor. No entanto, o código não está otimizado para manutenção. Para otimizar a mantenabilidade também, guarde o valor reutilizado em uma constante:

```
// Store the reused value to maintain code easily
const MAX_NUM:int = 300000;

var coordinates:Vector.<Number> = new Vector.<Number>(MAX_NUM, true);
var started:Number = getTimer();

for (var i:int = 0; i< MAX_NUM; i++)
{
    coordinates[i] = Math.random()*1024;
}

trace(getTimer() - started);
// output: 47
```

Utilize as APIs de objeto Vector, sempre que possível, já que elas costumam ser executadas mais rapidamente.

API de desenho

Use a API de desenho para executar o código mais rapidamente.

O Flash Player 10 e o AIR 1.5 fornecem uma uma nova API de desenho, que permite obter melhor desempenho da execução de código. Essa nova API não proporciona nenhum aprimoramento ao desempenho da renderização, mas pode reduzir drasticamente o número de linhas de código a ser programado. Um número menor de linhas de código pode proporcionar melhor desempenho da execução do ActionScript.

A nova API de desenho inclui os seguintes métodos:

- `drawPath()`
- `drawGraphicsData()`
- `drawTriangles()`

Nota: *Esta discussão não se concentra no método `drawTriangles()`, relacionado a 3D. No entanto, esse método pode aprimorar o desempenho do ActionScript, visto que lida com o mapeamento de textura nativo.*

O código a seguir chama explicitamente o método apropriado para cada linha que está sendo desenhada:

```
var container:Shape = new Shape();
container.graphics.beginFill(0x442299);

var coords:Vector.<Number> = Vector.<Number>([132, 20, 46, 254, 244, 100, 20, 98, 218, 254]);

container.graphics.moveTo ( coords[0], coords[1] );
container.graphics.lineTo ( coords[2], coords[3] );
container.graphics.lineTo ( coords[4], coords[5] );
container.graphics.lineTo ( coords[6], coords[7] );
container.graphics.lineTo ( coords[8], coords[9] );

addChild( container );
```

O código a seguir é executado mais rapidamente do que o exemplo anterior, visto que executa menos linhas de código. Quanto mais complexo o caminho, maior o ganho em desempenho obtido com o uso do método `drawPath()`:

```
var container:Shape = new Shape();
container.graphics.beginFill(0x442299);

var commands:Vector.<int> = Vector.<int>([1,2,2,2,2]);
var coords:Vector.<Number> = Vector.<Number>([132, 20, 46, 254, 244, 100, 20, 98, 218, 254]);

container.graphics.drawPath(commands, coords);

addChild( container );
```

O método `drawGraphicsData()` oferece aprimoramentos semelhantes ao desempenho.

Captura de eventos e animações

Utilize captura de eventos e bolhas para minimizar manipuladores de eventos.

O modelo de evento no ActionScript 3.0 introduziu os conceitos de captura de evento e de bolhas de evento. Você pode aproveitar a bolha de um evento para ajudá-lo na otimização do tempo de execução do código ActionScript. Você pode registrar um manipulador de evento em um objeto, ao invés de múltiplos objetos, para melhorar o desempenho.

Como exemplo, imagine criar um jogo no qual o usuário tem que clicar em maçãs o mais rápido possível para destruí-las. O jogo remove cada maçã da tela quando elas são clicadas e adiciona pontos a pontuação do usuário. Para ouvir o evento `MouseEvent.CLICK` acionado por cada maçã, você pode ficar tentado a escrever o seguinte código:

```
const MAX_NUM:int = 10;
var sceneWidth:int = stage.stageWidth;
var sceneHeight:int = stage.stageHeight;
var currentApple:InteractiveObject;
var currentAppleClicked:InteractiveObject;

for ( var i:int = 0; i< MAX_NUM; i++ )
{
    currentApple = new Apple();
    currentApple.x = Math.random()*sceneWidth;
    currentApple.y = Math.random()*sceneHeight;
    addChild ( currentApple );

    // Listen to the MouseEvent.CLICK event
    currentApple.addEventListener ( MouseEvent.CLICK, onAppleClick );
}

function onAppleClick ( e:MouseEvent ):void
{
    currentAppleClicked = e.currentTarget as InteractiveObject;
    currentAppleClicked.removeEventListener(MouseEvent.CLICK, onAppleClick );
    removeChild ( currentAppleClicked );
}
```

O código chama o método `addEventListener()` em cada ocorrência de Maçã. Isto também remove cada ouvinte quando uma maçã é clicada, utilizando o método`removeEventListener()`. No entanto, o modelo de evento no ActionScript 3.0 fornece uma fase de captura e de bolha para alguns eventos, permitindo que você os escute de um InteractiveObject pai. Como resultado, é possível otimizar o código acima e minimizar o número de chamadas para os métodos `addEventListener()` e `removeEventListener()`. O código a seguir utiliza a fase de captura para escutar os eventos do objeto pai:

```
const MAX_NUM:int = 10;
var sceneWidth:int = stage.stageWidth;
var sceneHeight:int = stage.stageHeight;
var currentApple:InteractiveObject;
var currentAppleClicked:InteractiveObject;
var container:Sprite = new Sprite();

addChild ( container );

// Listen to the MouseEvent.CLICK on the apple's parent
// Passing true as third parameter catches the event during its capture phase
container.addEventListener ( MouseEvent.CLICK, onAppleClick, true );

for ( var i:int = 0; i< MAX_NUM; i++ )
{
    currentApple = new Apple();
    currentApple.x = Math.random()*sceneWidth;
    currentApple.y = Math.random()*sceneHeight;
    container.addChild ( currentApple );
}

function onAppleClick ( e:MouseEvent ):void
{
    currentAppleClicked = e.target as InteractiveObject;
    container.removeChild ( currentAppleClicked );
}
```

O código está simplificado e muito mais otimizado, com apenas uma chamada para o método `addEventListener()` no contêiner pai. Os ouvintes não são mais registrados nas ocorrências Maçã, portanto não há necessidade de removê-los quando uma maçã é clicada. O manipulador `onAppleClick()` pode ser melhor otimizado, se você interromper a propagação do evento fazendo com que ele não avance ainda mais:

```
function onAppleClick ( e:MouseEvent ):void
{
    e.stopPropagation();
    currentAppleClicked = e.target as InteractiveObject;
    container.removeChild ( currentAppleClicked );
}
```

A fase de bolha também pode ser utilizada para capturar o evento, transmitindo `false` como o terceiro parâmetro para o método `addEventListener()`:

```
// Listen to the MouseEvent.CLICK on apple's parent
// Passing false as third parameter catches the event during its bubbling phase
container.addEventListener ( MouseEvent.CLICK, onAppleClick, false );
```

O valor padrão para o parâmetro da fase de captura é `false`, portanto você pode omiti-lo:

```
container.addEventListener ( MouseEvent.CLICK, onAppleClick );
```

Trabalhando com pixels

Pintar pixels usando o método `setVector()` .

Ao pintar pixels, algumas otimizações simples podem ser feitas pelo simples uso dos métodos apropriados da classe BitmapData. Uma maneira rápida de pintar pixels é usar o método `setVector()`:

```
// Image dimensions
var wdth:int = 200;
var hght:int = 200;
var total:int = wdth*hght;

// Pixel colors Vector
var pixels:Vector.<uint> = new Vector.<uint>(total, true);

for ( var i:int = 0; i< total; i++ )
{
    // Store the color of each pixel
    pixels[i] = Math.random()*0xFFFFFF;
}

// Create a non-transparent BitmapData object
var myImage:BitmapData = new BitmapData ( wdth, hght, false );
var imageContainer:Bitmap = new Bitmap ( myImage );

// Paint the pixels
myImage.setVector ( myImage.rect, pixels );
addChild ( imageContainer );
```

Ao usar métodos lentos, como `setPixel()` ou `setPixel32()`, use os métodos `lock()` e `unlock()` para acelerar a execução. No código a seguir, os métodos `lock()` e `unlock()` são usados para aumentar o desempenho:

```
var buffer:BitmapData = new BitmapData(200,200,true,0xFFFFFFFF);
var bitmapContainer:Bitmap = new Bitmap(buffer);
var positionX:int;
var positionY:int;

// Lock update
buffer.lock();
var starting:Number=getTimer();

for (var i:int = 0; i<2000000; i++)
{
    // Random positions
    positionX = Math.random()*200;
    positionY = Math.random()*200;
    // 40% transparent pixels
    buffer.setPixel32( positionX, positionY, 0x66990000 );
}

// Unlock update
buffer.unlock();
addChild( bitmapContainer );

trace( getTimer () - starting );
// output : 670
```

O método `lock()` da classe BitmapData bloqueia uma imagem e impede que objetos que façam referência a ela sejam atualizados quando o objeto BitmapData é alterado. Por exemplo, se um objeto Bitmap fizer referência a um objeto BitmapData, você pode bloquear o objeto BitmapData, alterá-lo e, em seguida, desbloqueá-lo. O objeto Bitmap não é alterado até que o objeto BitmapData seja desbloqueado. Para melhorar o desempenho, use esse método junto com o método `unlock()` antes e depois de várias chamadas aos métodos `setPixel()` ou `setPixel32()`. Chamar `lock()` e `unlock()` impede que a tela seja atualizada desnecessariamente.

Nota: *Ao processar pixels em um bitmap que não consta na lista de exibição (double-buffering), as vezes esta técnica não aumenta o desempenho. Caso um objeto de bitmap não faça referência ao buffer do bitmap, o uso de `lock()` e `unlock()` não aumenta o desempenho. O Flash Player detecta que não há referência ao buffer, e o bitmap não é renderizado na tela.*

Métodos que iteram sobre pixels, como `getPixel()`, `getPixel32()`, `setPixel()`, e `setPixel32()`, costumam ser lentos, especialmente em dispositivos móveis. Se possível, utilize métodos que obtenham todos os pixels em uma chamada. Para ler pixels, use o método `getVector()`, que é mais rápido do que o método `getPixels()`. Além disso, lembre-se de usar as APIs que dependem de objetos Vector, quando possível, visto que elas provavelmente serão executadas mais rapidamente.

Expressões regulares

Use métodos da classe String como, por exemplo, `indexOf()`, `substr()` ou `substring()` em vez de uma expressão regular para localizar e extrair sequências de caracteres básicas.

É possível desempenhar certas operações utilizando uma expressão regular, como também utilizando métodos da classe String. Por exemplo, para descobrir se uma sequência de caracteres contém outra sequência de caracteres, é possível usar o método `String.indexOf()` ou usar uma expressão regular. No entanto, quando está disponível o método da classe String, ela executa mais rápido que a expressão regular equivalente e não requer a criação de outro objeto.

Use um grupo de não captura ("`(?:xxxx)`") em vez de um grupo ("`(xxxx)`") em uma expressão regular se precisar agrupar elementos, mas não precisar isolar o conteúdo do grupo no resultado.

Com frequência, em expressões regulares de complexidade moderada, você agrupa partes da expressão. Por exemplo, na seguinte expressão regular, o padrão parênteses cria um grupo ao redor do texto "ab". Como resultado, o quantificador "+" aplica-se ao grupo em vez de a um simples caractere:

```
/(ab)+/
```

Como padrão, o conteúdo de cada grupo é "capturado". É possível obter o conteúdo de cada grupo no seu padrão como parte do resultado da execução da expressão regular. Capturar o resultado desse grupo demora mais e requer mais memória porque os objetos são criados para conter resultados de grupos. Como alternativa, é possível usar sintaxe de grupo de não captura, incluindo ponto de interrogação e dois pontos depois de abrir parênteses. Essa sintaxe especifica que os caracteres comportam-se como grupo mas não são capturados para o resultado:

```
/(?:ab)+/
```

Utilizar sintaxe de grupo de não captura é mais rápido e utiliza menos memória que sintaxe de grupo padrão.

Considere a utilização de um padrão de expressão regular alternativo, se a expressão regular não tiver bom desempenho.

Às vezes, é possível usar mais de um padrão de expressão regular para testar ou identificar o mesmo padrão de texto. Por vários motivos, certos padrões executam mais rápido que outros alternativos. Se você determinar que uma expressão regular está fazendo o seu código executar mais lentamente do que o necessário, considere padrões de expressões regulares alternativos que obtenham o mesmo resultado. Teste os padrões alternativos para verificar qual é o mais rápido.

Otimizações diversas

Para um objeto TextField, use o método `appendText()` ao invés do operador `+=`.

Ao trabalhar com a propriedade `text` da classe TextField, use o método `appendText()` em vez de usar o operador `+=`. O uso do método `appendText()` aumenta o desempenho.

Como exemplo, o código a seguir usa o operador `+=`, e o loop demora 1120 ms para ser concluído:

```
addChild ( myTextField );

myTextField.autoSize = TextFieldAutoSize.LEFT;
var started:Number = getTimer();

for (var i:int = 0; i< 1500; i++ )
{
    myTextField.text += "ActionScript 3";
}

trace( getTimer() - started );
// output : 1120
```

No exemplo a seguir, o operador `+=` é substituído pelo método `appendText()`:

```
var myTextField:TextField = new TextField();
addChild ( myTextField );
myTextField.autoSize = TextFieldAutoSize.LEFT;

var started:Number = getTimer();

for (var i:int = 0; i< 1500; i++ )
{
    myTextField.appendText ( "ActionScript 3" );
}

trace( getTimer() - started );
// output : 847
```

Agora o código leva 847 ms para ser concluído.

Atualize campos de texto fora dos loops, sempre que possível.

Esse código pode ser otimizado ainda mais por meio do uso de uma técnica simples. Atualizar o campo de texto em cada loop usa muito processamento interno. Por meio da simples concatenação de uma sequência de caracteres que é então atribuída ao campo de texto fora do loop, o tempo de execução do código é consideravelmente reduzido. Agora o código leva 2 ms para ser concluído:

```
var myTextField:TextField = new TextField();
addChild ( myTextField );
myTextField.autoSize = TextFieldAutoSize.LEFT;

var started:Number = getTimer();
var content:String = myTextField.text;

for (var i:int = 0; i< 1500; i++ )
{
    content += "ActionScript 3";
}

myTextField.text = content;

trace( getTimer() - started );
// output : 2
```

Ao trabalhar com texto HTML, a abordagem anterior é tão lenta que pode gerar uma exceção `Timeout` no Flash Player, em alguns casos. Por exemplo, uma exceção pode ser gerada se o hardware subjacente for muito lento.

Nota: *Adobe® AIR® não gera essa exceção.*

```
var myTextField:TextField = new TextField();
addChild ( myTextField );
myTextField.autoSize = TextFieldAutoSize.LEFT;

var started:Number = getTimer();

for (var i:int = 0; i< 1500; i++ )
{
    myTextField.htmlText += "ActionScript <b>2</b>";
}

trace( getTimer() - started );
```

Com a atribuição do valor a uma sequência de caracteres fora do loop, o código requer apenas 29 ms para ser concluído:

```
var myTextField:TextField = new TextField();
addChild ( myTextField );
myTextField.autoSize = TextFieldAutoSize.LEFT;

var started:Number = getTimer();
var content:String = myTextField.htmlText;

for (var i:int = 0; i< 1500; i++ )
{
    content += "<b>ActionScript<b> 3";
}

myTextField.htmlText = content;

trace ( getTimer() - started );
// output : 29
```

Nota: *No Flash Player 10.1 e no AIR 2.5, a classe String foi aprimorada de modo que as strings usem menos memória.*

Evite usar o operador colchete, sempre que possível.

Usar o operador colchete pode reduzir o desempenho. Você pode evitar seu uso armazenando sua referência em uma variável local. O exemplo de código a seguir demonstra o uso ineficiente do operador colchete:

```
var lng:int = 5000;
var arraySprite:Vector.<Sprite> = new Vector.<Sprite>(lng, true);
var i:int;

for ( i = 0; i< lng; i++ )
{
    arraySprite[i] = new Sprite();
}

var started:Number = getTimer();

for ( i = 0; i< lng; i++ )
{
    arraySprite[i].x = Math.random()*stage.stageWidth;
    arraySprite[i].y = Math.random()*stage.stageHeight;
    arraySprite[i].alpha = Math.random();
    arraySprite[i].rotation = Math.random()*360;
}

trace( getTimer() - started );
// output : 16
```

A seguinte versão otimizada reduz o uso do operador colchete:

```
var lng:int = 5000;
var arraySprite:Vector.<Sprite> = new Vector.<Sprite>(lng, true);
var i:int;

for ( i = 0; i< lng; i++ )
{
    arraySprite[i] = new Sprite();
}

var started:Number = getTimer();
var currentSprite:Sprite;

for ( i = 0; i< lng; i++ )
{
    currentSprite = arraySprite[i];
    currentSprite.x = Math.random()*stage.stageWidth;
    currentSprite.y = Math.random()*stage.stageHeight;
    currentSprite.alpha = Math.random();
    currentSprite.rotation = Math.random()*360;
}

trace( getTimer() - started );
// output : 9
```

Crie código embutido, sempre que possível, para reduzir o número de chamadas a funções no código.

Chamar funções pode ser dispendioso. Tente reduzir o número de chamadas a funções embutindo o código. Embutir o código é uma boa maneira de otimizar o desempenho. No entanto, lembre-se de que o código embutido dificulta a reutilização do código e pode aumentar o tamanho do arquivo SWF. Algumas chamadas a funções, como métodos da classe Math, podem ser facilmente embutidas. O código a seguir usa o método `Math.abs()` para calcular valores absolutos:

```
const MAX_NUM:int = 500000;
var arrayValues:Vector.<Number>=new Vector.<Number>(MAX_NUM,true);
var i:int;

for (i = 0; i< MAX_NUM; i++)
{
    arrayValues[i] = Math.random()-Math.random();
}

var started:Number = getTimer();
var currentValue:Number;

for (i = 0; i< MAX_NUM; i++)
{
    currentValue = arrayValues[i];
    arrayValues[i] = Math.abs ( currentValue );
}

trace( getTimer() - started );
// output : 70
```

O cálculo realizado por `Math.abs()` pode ser feito manualmente e embutido:

```
const MAX_NUM:int = 500000;
var arrayValues:Vector.<Number>=new Vector.<Number>(MAX_NUM,true);
var i:int;

for (i = 0; i< MAX_NUM; i++)
{
    arrayValues[i] = Math.random()-Math.random();
}

var started:Number = getTimer();
var currentValue:Number;

for (i = 0; i< MAX_NUM; i++)
{
    currentValue = arrayValues[i];
    arrayValues[i] = currentValue > 0 ? currentValue : -currentValue;
}

trace( getTimer() - started );
// output : 15
```

Embutir a chamada de função resulta em código mais de quatro vezes mais rápido. Essa abordagem é útil em muitas situações, mas esteja ciente do efeito que isso pode ter na capacidade de reutilização e manutenção.

Nota: *O tamanho do código tem grande impacto na execução geral do player. Caso a aplicação inclua grandes quantidades de código ActionScript, então a máquina virtual gastará quantidades significativas de tempo verificando o código e a compilação JIT. Pesquisas de propriedades podem ser lentas, por causa da profundidade das hierarquias de herança e porque caches internos costumam acumular mais informação desnecessária. Para reduzir o tamanho do código, evite o uso da estrutura Adobe® Flex®, da biblioteca da estrutura TLF, ou quaisquer grandes bibliotecas ActionScript de terceiros.*


Evite avaliações de instruções nas repetições.

Outra otimização pode ser obtida deixando-se de avaliar uma instrução dentro de uma repetição. O código a seguir é iterado por uma matriz, mas não é otimizado, porque o comprimento da matriz é avaliado a cada iteração:

```
for (var i:int = 0; i< myArray.length; i++)
{
}
```

É melhor guardar o valor e reutiliza-lo.

```
var lng:int = myArray.length;

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


Utilize ordem reversa para repetições enquanto.

Uma repetição enquanto na ordem reversa é mais rápida que uma repetição avançar:

```
var i:int = myArray.length;

while (--i > -1)
{
}
```

Estas dicas fornecem algumas maneiras de otimizar o ActionScript, mostrando como uma linha de código pode afetar desempenho e memória. Muitas outras otimizações ActionScript são possíveis. Para obter mais informações, consulte: http://www.rozengain.com/blog/2007/05/01/some-actionscript-30-optimizations/.

Capítulo 5: Desempenho da renderização

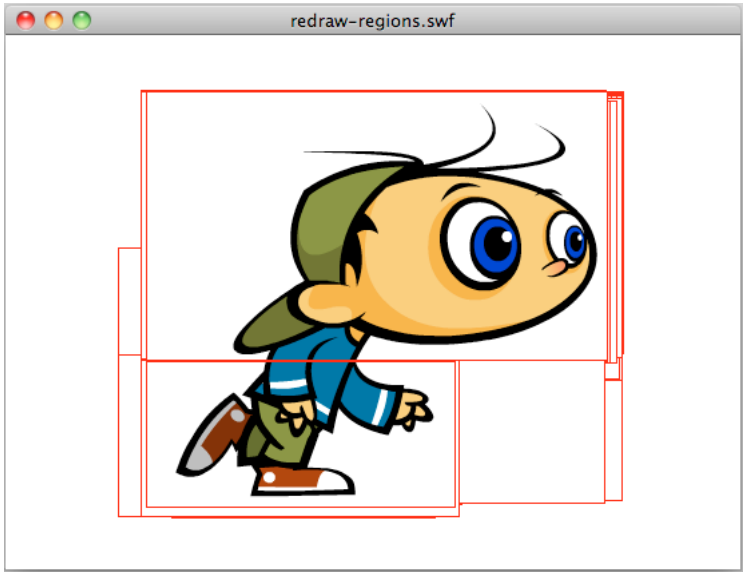
Redesenhar regiões

Use sempre a opção de redesenho de regiões ao criar seu projeto.

Para melhorar a renderização, é importante usar a opção de redesenho de regiões ao criar seu projeto. Esta opção permite ver as regiões que o Flash Player está renderizando e processando. Você pode ativar esta opção selecionando Mostrar regiões de redesenho no menu de contexto da versão de depuração do Flash Player.

Nota: *A opção Mostrar regiões de redesenho não está disponível no Adobe AIR nem na versão de lançamento do Flash Player. (No Adobe AIR, o menu de contexto fica disponível somente em aplicativos para desktop, mas não tem itens incorporados ou padrão, como Mostrar regiões de redesenho.)*

A imagem abaixo ilustra a opção ativada com um simples clipe de vídeo animado na linha de tempo:



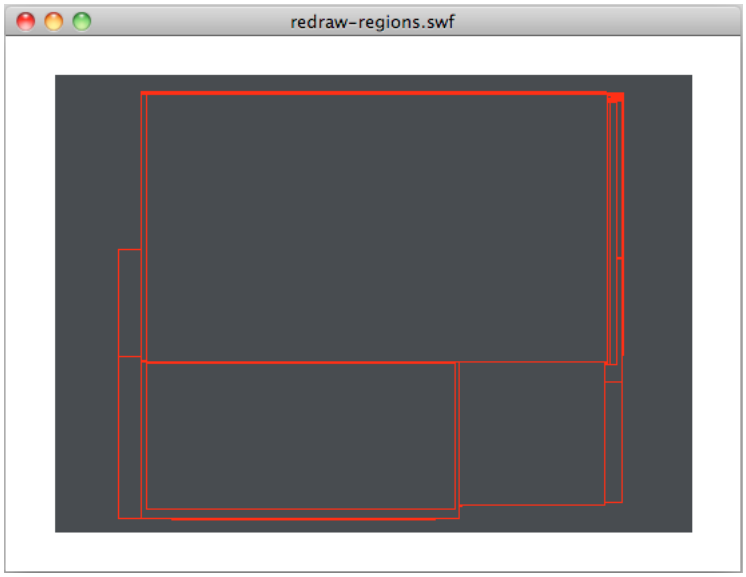
Opção "Redesenhar regiões" ativada

Também é possível ativar esta opção de forma programada, usando o método `flash.profiler.showRedrawRegions()`:

```
// Enable Show Redraw Regions
// Blue color is used to show redrawn regions
flash.profiler.showRedrawRegions ( true, 0x0000FF );
```

Em aplicativos Adobe AIR, esse método é a única maneira de ativar a opção de regiões de redesenho.

Use as regiões de redesenho para identificar oportunidades de otimização. Lembre-se de que embora alguns objetos de exibição não sejam exibidos, eles continuam consumindo ciclos de CPU porque ainda estão sendo renderizados. A imagem a seguir ilustra este conceito. Um formato de vetor preto cobre o caractere de execução animada. A imagem mostra que este objeto de exibição não foi removido da lista de exibição e ainda está sendo renderizado. Isso consome ciclos de CPU:



Redesenhar regiões

Para aumentar o desempenho, defina a propriedade `visible` do caractere oculto em execução como `false` ou remova-o da lista de exibição. Você também deve parar a linha de tempo. Esta providência garante que o objeto de exibição seja congelado e use o mínimo de potência da CPU.

Lembre-se de usar a opção de redesenho de regiões durante todo o ciclo de desenvolvimento. Usar esta opção impede que você seja surpreendido no fim do projeto por regiões redesenhadas e áreas de otimização desnecessárias, que poderiam ter sido ignoradas.

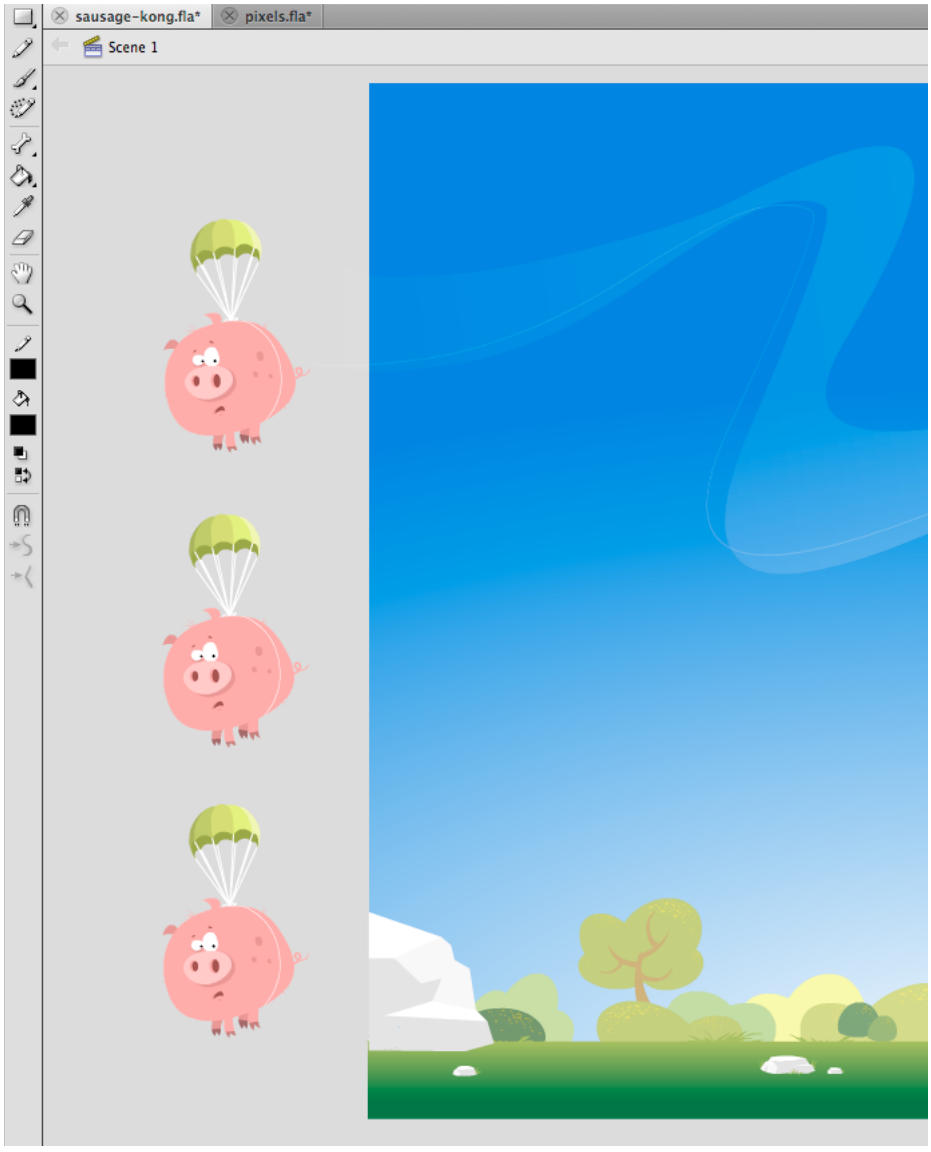
Mais tópicos da Ajuda

"Congelando e descongelando objetos" na página 28

Conteúdo fora do palco

Evite colocar conteúdo fora do palco. Em vez disso, coloque os objetos na lista de exibição quando necessário.

Se possível, tente não colocar conteúdo gráfico fora do palco. Designers e desenvolvedores costumam colocar elementos fora do palco para reutilizar ativos durante a vida útil do aplicativo. A figura a seguir ilustra essa técnica comum:



Conteúdo fora do palco

Mesmo que os elementos fora do palco não sejam mostrados na tela e não sejam renderizados, eles ainda constam da lista de exibição. O tempo de execução continua executando testes internos nesses elementos para assegurar-se de que ainda estão fora do palco e que o usuário não está interagindo com eles. Consequentemente, tanto quando possível, evite colocar objetos fora do palco e, em vez disso, remova-os da lista de exibição.

Qualidade dos filmes

Use a configuração de qualidade apropriada do Palco para aprimorar a renderização.

Quando se desenvolve conteúdo para dispositivos móveis com telas pequenas, como telefones, a qualidade da imagem é menos importante do que quando se desenvolvem aplicativos para um computador desktop. Definir a qualidade do Palco com a configuração apropriada pode aumentar o desempenho da renderização.

As configurações a seguir estão disponíveis para a qualidade do Palco:

- `StageQuality.LOW`: favorece a velocidade de reprodução em relação à aparência e não usa suavização de borda. Esta configuração não conta com o suporte do Adobe AIR para desktop ou TV.
- `StageQuality.MEDIUM`: aplica alguma suavização de borda, mas não suaviza bitmaps dimensionados. Essa configuração é o valor padrão do AIR nos dispositivos móveis, mas não é suportada no AIR para computador pessoal ou para TV.
- `StageQuality.HIGH`: (padrão para desktop) favorece a aparência com relação à velocidade de reprodução e sempre usa a suavização de borda. Se o arquivo SWF não contiver animação, os bitmaps redimensionados são suavizados; se o arquivo SWF contiver animação, os bitmaps não são suavizados.
- `StageQuality.BEST`: oferece a melhor qualidade de exibição e não leva em consideração a velocidade de reprodução. Toda a saída tem suavização de borda e os bitmaps dimensionados são sempre suavizados.

O uso de `StageQuality.MEDIUM` frequentemente fornece qualidade suficiente para aplicativos em dispositivos móveis e, em alguns casos, usar `StageQuality.LOW` pode proporcionar qualidade suficiente. Desde o Flash Player 8, o texto com suavização de borda pode ser renderizado com precisão, mesmo com a qualidade do Palco definida como `LOW`.

Nota: *Em alguns dispositivos móveis, mesmo que a qualidade esteja definida como `HIGH`, `MEDIUM` é usado para proporcionar melhor desempenho em aplicativos Flash Player. No entanto, configurar a qualidade como HIGH não costuma fazer uma diferença perceptível, porque as telas móveis geralmente tem uma taxa de dpi mais alta. (A taxa de dpi pode variar, dependendo do dispositivo.)*

Na figura a seguir, a qualidade do filme é definida como `MEDIUM` e a renderização do texto é definida como Suavização de borda para animação:

Qualidade média do Palco e renderização definida como Suavização de borda para animação

A configuração de qualidade do Palco afeta a qualidade do texto, porque a configuração de renderização de texto apropriada não está sendo usada.

O tempo de execução permite definir a renderização de texto como Suavização de borda para legibilidade. Essa configuração mantém a perfeita qualidade do texto (suavizado), independentemente da configuração de qualidade de Palco usada:

Qualidade baixa do Palco e renderização definida como Suavização de borda para legibilidade

A mesma qualidade de renderização pode ser obtida pela configuração da renderização de texto como Texto de bitmap (Sem suavização de borda):

Qualidade baixa do Palco e renderização de texto definida como Texto de bitmap (Sem suavização de borda)

Os últimos dois exemplos mostram você pode obter texto de alta qualidade, independentemente da configuração de qualidade de Palco usada. Este recurso está disponível desde o Flash Player 8 e pode ser utilizado em dispositivos móveis. Tenha em mente que o Flash Player 10.1 muda para `StageQuality.MEDIUM` automaticamente em alguns dispositivos para aumentar o desempenho.

Mesclagem Alfa

Evite utilizar a propriedade alfa, quando possível.

Evite utilizar efeitos que exigem mesclagem alfa quando utilizar a propriedade `alpha`, como nos efeitos de desaparecimento. Quando um objeto de exibição usa mistura alfa, o tempo de execução precisa combinar os valores de cor de cada objeto de exibição empilhado e a cor de fundo para determinar a cor final. Assim, a mistura alfa pode consumir mais processador do que desenhar uma cor opaca. Essa computação extra pode prejudicar o desempenho em dispositivos mais lentos. Evite utilizar a propriedade `alfa` , quando possível.

Mais tópicos da Ajuda

"Armazenamento do bitmap em cache" na página 54

"Renderização de objetos de texto" na página 68

Taxa de quadros do aplicativo

Em geral, utilize a menor taxa possível de quadros para obter melhor desempenho.

A taxa de quadros do aplicativo verifica a quantidade de tempo disponível para cada ciclo "código do aplicativo e renderização", conforme descrito em "Conceitos básicos da execução do código de tempo de execução" na página 1. Uma taxa de quadros maior produz uma animação mais suave. Contudo, quando não estão ocorrendo animações ou outras alterações visuais, em geral não há motivo para uma taxa de quadros alta. Uma taxa de quadros mais alta gasta mais ciclos da CPU e energia da bateria do que uma taxa mais baixa.

Abaixo estão algumas orientações gerais para uma taxa de quadros padrão apropriada ao aplicativo:

- Se utilizar a estrutura Flex, deixe a taxa de quadros inicial no valor padrão.
- Se o aplicativo incluir animações, a taxa de quadros adequada deverá ser de pelo menos 20 quadros por segundo. Qualquer taxa acima de 30 quadros por segundo geralmente é desnecessária.
- Se o aplicativo não incluir animações, provavelmente a taxa de 12 quadros por segundo será suficiente.

A "menor taxa de quadros possível" pode variar dependendo da atividade atual do aplicativo. Consulte a próxima dica "Alterando dinamicamente a taxa de quadros do aplicativo" para obter mais informações.

Utilize uma taxa de quadros baixa quando o vídeo tem apenas conteúdo dinâmico no aplicativo.

O tempo de execução reproduz o conteúdo do vídeo na taxa de quadros nativa, independentemente da taxa de quadros do aplicativo. Se o aplicativo não tiver animações ou outro conteúdo visual com alterações rápidas (exceto vídeo), utilizar uma taxa de quadros baixa não diminui a qualidade da experiência da interface do usuário.

Altere dinamicamente a taxa de quadros do aplicativo.

Você define a taxa de quadros inicial do aplicativo nas configurações do projeto ou compilador; porém, a taxa de quadros não é estabelecida nesse valor. É possível alterar a taxa de quadros definindo a propriedade `Stage.frameRate` (ou a propriedade`WindowedApplication.frameRate` no Flex).

Altere a taxa de quadros de acordo com as necessidades atuais do aplicativo. Por exemplo, quando o aplicativo não estiver desempenhando quaisquer animações, diminua a taxa de quadros. Quando uma transição animada estiver prestes a começar, aumente a taxa de quadros. Da mesma forma, se o aplicativo estiver executando em segundo plano (depois de perder o foco), geralmente você poderá diminuir a taxa de quadros ainda mais. Provavelmente o usuário estará concentrado em outro aplicativo ou em outra tarefa.

A seguir estão algumas orientações gerais para utilizar como ponto de partida para especificar a taxa de quadros apropriada a diferentes tipos de atividades:

- Se utilizar a estrutura Flex, deixe a taxa de quadros inicial no valor padrão.
- Quando animações são reproduzidas, a taxa de quadros deve ser pelo menos 20 quadros por segundo. Qualquer taxa acima de 30 quadros por segundo geralmente é desnecessária.
- Quando não são reproduzidas animações, provavelmente, a taxa de 12 quadros por segundo é suficiente.

- Vídeo carregado é reproduzido na taxa de quadros nativa independentemente da taxa de quadros do aplicativo. Se o vídeo for o único conteúdo em movimento no aplicativo, provavelmente, a taxa de 12 quadros por segundo é suficiente.
- Quando o aplicativo não tem foco de entrada, provavelmente, a taxa de 5 quadros por segundo é suficiente.
- Quando um aplicativo AIR não está visível, uma taxa de quadros de 2 quadros por segundo ou menos geralmente será apropriada. Por exemplo, esta orientação se aplica se uma aplicativo estiver minimizado. Também se aplica a dispositivos desktop se a propriedade `visible` da janela nativa estiver definida como `false`.

Para aplicativos construídos no Flex, a classe spark.components.WindowedApplication tem suporte incorporado para alterar dinamicamente a taxa de quadros do aplicativo. A propriedade `backgroundFrameRate` define a taxa de quadros do aplicativo quando o aplicativo não está ativo. O valor padrão é 1, que altera a taxa de quadros de um aplicativo criado com a estrutura Spark a 1 quadro por segundo. Você pode alterar a taxa de quadros em segundo plano configurando a propriedade `backgroundFrameRate`. Você pode definir a propriedade com outro valor ou defini-la como -1 para desativar o estrangulamento automático de taxa de quadros.

Para obter mais informações sobre como alterar dinamicamente a taxa de quadros do aplicativo, consulte os seguintes artigos:

- Reduzindo a utilização da CPU no Adobe AIR (Artigo e código de exemplo do Centro de Desenvolvedores da Adobe por Jonnie Hallman)
- Escrevendo aplicativos do AIR bem comportados e eficientes (artigo e aplicativo de exemplo por Arno Gourdol)

Grant Skinner criou uma classe de estrangulador de taxa de quadros. Você pode usar esta classe em seus aplicativos para diminuir automaticamente a taxa de quadros quando o seu aplicativo estiver em segundo plano. Para obter mais informações e para baixar o código-fonte da classe FramerateThrottler, consulte o artigo de Grant Uso ocioso da CPU no Adobe AIR e no Flash Player (em inglês) em http://gskinner.com/blog/archives/2009/05/idle_cpu_usage.html.

Taxa de quadros adaptável

Ao compilar um arquivo SWF, você define uma taxa de quadros específica para o filme. Em um ambiente restrito com uma baixa taxa de CPU, a taxa de quadros às vezes cai durante a reprodução. Para preservar uma taxa de quadros aceitável para o usuário, o tempo de execução ignora a renderização de alguns quadros. Ignorar a renderização de alguns quadros impede que a taxa de quadros fique abaixo de um valor aceitável.

Nota: *Neste caso, o tempo de execução não ignora quadros, ignorando apenas a renderização do conteúdo nos quadros. O código ainda é executado e a lista de exibição é atualizada, mas as atualizações não são exibidas na tela. Não há como especificar um valor de fps limite que indique quantos quadros devem ser ignorados se o tempo de execução não puder manter a estabilidade da taxa de quadros.*

Armazenamento do bitmap em cache

Use o recurso de armazenamento de bitmap em cache para conteúdo complexo de vetores, quando apropriado.

Uma boa otimização pode ser feita por meio do uso do recurso de armazenamento de bitmaps em cache. Este recurso armazena um objeto de vetor no cache, o renderiza internamente como um bitmap e usa o bitmap para renderização. O resultado pode ser um enorme aumento do desempenho da renderização, mas isso pode requerer uma quantidade significativa de memória. Use o recurso de armazenamento de bitmap em cache para conteúdo de vetores complexos, como gradientes ou textos complexos.

Ligar o caching de bitmap para um objeto animado que contenha gráficos vetoriais complexos (como um texto ou gradientes) melhora o desempenho. Contudo, se o armazenamento em cache de bitmap estiver ativado em um objeto de exibição tal como um clipe de vídeo cuja linha de tempo esteja em execução, você poderá obter o resultado oposto. Em cada quadro, o caching feature precisa atualizar o bitmap armazenado em cache e, em seguida, redesenhá-lo na tela, o que requer muito ciclos de CPU. O recurso de armazenamento de bitmap em cache é uma vantagem apenas quando o bitmap em cache pode ser gerado uma vez e então usado sem ser atualizado.

Se você ativar o armazenamento de bitmap em cache para um objeto Sprite, o objeto pode ser movido sem obrigar o tempo de execução a gerar novamente o bitmap armazenado em cache. Alterar as propriedades `x` e `y` do objeto não causa uma nova geração. No entanto, qualquer tentativa de girá-lo, dimensioná-lo ou mudar seu valor alfa faz com que o tempo de execução gere novamente o bitmap armazenado em cache, o que prejudica o desempenho.

Nota: *A propriedade `DisplayObject.cacheAsBitmapMatrix` disponível no AIR e o Packager para iPhone não têm essa limitação. Usando a propriedade `cacheAsBitmapMatrix`, você pode girar, dimensionar, inclinar e alterar o valor alfa de um objeto de exibição sem acionar uma nova geração do bitmap.*

Um bitmap em cache pode usar mais memória do que uma instância comum de clipe de filme. Por exemplo, se o clipe de filme no Palco tiver 250 x 250 pixels, quando armazenado em cache ele usará cerca e 250 KB, em vez de 1 KB, usado quando não armazenado em cache.

O exemplo a seguir envolve um objeto Sprite que contém a imagem de uma maçã. A classe a seguir é anexada ao símbolo da maçã:

```
package org.bytearray.bitmap
{
    import flash.display.Sprite;
    import flash.events.Event;

    public class Apple extends Sprite
    {
        private var destinationX:Number;
        private var destinationY:Number;

        public function Apple ()
        {
            addEventListener(Event.ADDED_TO_STAGE,activation);
            addEventListener(Event.REMOVED_FROM_STAGE,deactivation);
        }

        private function activation(e:Event):void
        {
            initPos();
            addEventListener (Event.ENTER_FRAME,handleMovement);
        }

        private function deactivation(e:Event):void
```

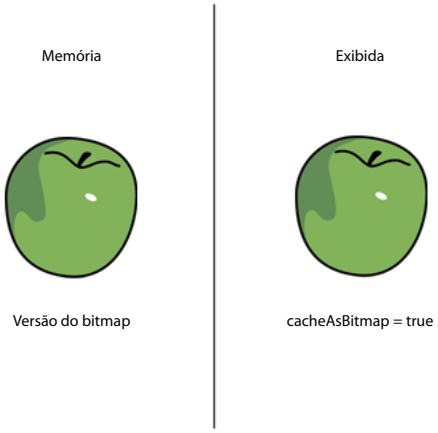
```
        {
            removeEventListener(Event.ENTER_FRAME,handleMovement);
        }

        private function initPos():void
        {
            destinationX = Math.random()*(stage.stageWidth - (width>>1));
            destinationY = Math.random()*(stage.stageHeight - (height>>1));
        }

        private function handleMovement(e:Event):void
        {
            x -= (x - destinationX)*.5;
            y -= (y - destinationY)*.5;

            if (Math.abs(x - destinationX) < 1 && Math.abs(y - destinationY) < 1)
                initPos();
        }
    }
}
```

O código usa a classe Sprite em vez da classe MovieClip, porque uma linha de tempo não é necessária para cada maçã. Para aumentar ao máximo o desempenho, use o objeto mais leve possível. Em seguida, a classe é instanciada com o seguinte código:

```
import org.bytearray.bitmap.Apple;

stage.addEventListener(MouseEvent.CLICK,createApples);
stage.addEventListener(KeyboardEvent.KEY_DOWN,cacheApples);

const MAX_NUM:int = 100;
var apple:Apple;
var holder:Sprite = new Sprite();

addChild(holder);

function createApples(e:MouseEvent):void
{
    for (var i:int = 0; i< MAX_NUM; i++)
    {
        apple = new Apple();

        holder.addChild(apple);
    }
}

function cacheApples(e:KeyboardEvent):void
{
    if (e.keyCode == 67)
    {
        var lng:int = holder.numChildren;

        for (var i:int = 0; i < lng; i++)
        {
            apple = holder.getChildAt (i) as Apple;

            apple.cacheAsBitmap = Boolean(!apple.cacheAsBitmap);
        }
    }
}
```

Quando o usuário clica no mouse, as maçãs são criadas sem armazenamento em cache. Quando o usuário pressiona a tecla C (código de tecla 67), os vetores da maçã são armazenados em cache como bitmaps e mostrados na tela. Essa técnica aumenta muito o desempenho da renderização, tanto em computadores desktop quanto em dispositivos móveis, quando a CPU é lenta.

No entanto, embora o uso do recurso de armazenamento de bitmap em cache aumente o desempenho da renderização, ele pode consumir rapidamente grandes quantidades de memória. Assim que um objeto é armazenado em cache, sua superfície é capturada como um bitmap transparente e armazenada na memória, conforme mostrado no seguinte diagrama:

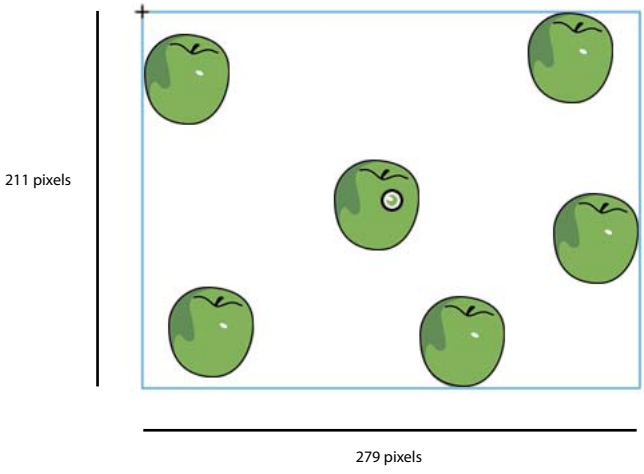


Bitmap do objeto e de sua superfície armazenado na memória

O Flash Player 10.1 e o AIR 2.5 otimizam o uso de memória usando a mesma abordagem descrita em "Filtros e descarregamento dinâmico de bitmap" na página 20. Se um objeto de exibição armazenado em cache estiver oculto ou fora da dela, seu bitmap na memória é liberado quando ficar sem uso por um tempo.

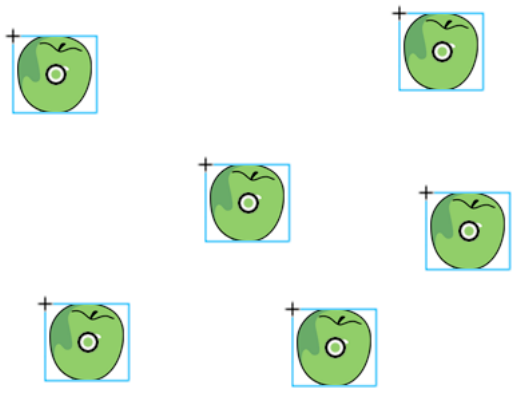
Nota: *Se a propriedade* `opaqueBackground` *do objeto de exibição for definida com uma cor específica, o tempo de execução considerará o objeto como sendo opaco. Quando usado com a propriedade* `cacheAsBitmap`*, o tempo de execução cria um bitmap não transparente de 32 bits na memória. O canal alfa é definido como 0xFF, o que aumenta o desempenho, porque nenhuma transparência é necessária para desenhar o bitmap na tela. Evitar a mescla alfa torna a renderização ainda mais rápida. Se a profundidade atual da tela estiver limitada a 16 bits, o bitmap na memória será armazenado como uma imagem de 16 bits. Utilizar a propriedade* `opaqueBackground` *não ativa o armazenamento de bitmap implicitamente.*

Para economizar memória, use a propriedade `cacheAsBitmap` e ative-a em cada objeto de exibição em vez de fazê-lo no container. Ativar o armazenamento de bitmap em cache no container torna o bitmap final muito maior na memória, criando um bitmap transparente com dimensões de 211 x 279 pixels. A imagem usa cerca de 229 KB de memória:



Ativação do armazenamento de bitmap em cache no container

Além disso, ao armazenar o contêiner, você corre o risco atualizar todo o bitmap na memória, caso qualquer maçã comece a se mover em um quadro. A ativação do armazenamento de bitmap em cache em cada instância individual resulta no armazenamento em cache de seis superfícies de 7 KB na memória, o que usa somente 42 KB de memória:



Ativação do armazenamento de bitmap em cache nas instâncias

Acessar cada instância da maçã por meio da lista de exibição e chamar o método `getChildAt()` armazena referências em um objeto Vector para tornar o acesso mais fácil:

```
import org.bytearray.bitmap.Apple;

stage.addEventListener(KeyboardEvent.KEY_DOWN, cacheApples);

const MAX_NUM:int = 200;
var apple:Apple;
var holder:Sprite = new Sprite();

addChild(holder);

var holderVector:Vector.<Apple> = new Vector.<Apple>(MAX_NUM, true);

for (var i:int = 0; i< MAX_NUM; i++)
{
    apple = new Apple();

    holder.addChild(apple);

    holderVector[i] = apple;
}

function cacheApples(e:KeyboardEvent):void
{
    if (e.keyCode == 67)
    {
        var lng:int = holderVector.length

        for (var i:int = 0; i < lng; i++)
        {
            apple = holderVector[i];

            apple.cacheAsBitmap = Boolean(!apple.cacheAsBitmap);
        }
    }
}
```

Tenha em mente que o armazenamento de bitmap aumenta a renderização caso o conteúdo armazenado não seja rotacionado, dimensionado ou alterado a cada quadro. Entretanto, para qualquer transformação que não seja translação nos eixos x e y, a renderização não é aprimorada. Nesses casos, o Flash Player atualiza a cópia do bitmap no cache para cada transformação que ocorra no objeto exibido. Atualizar a cópia em cache pode resultar em um alto uso da CPU, em baixo desempenho e em alto consumo de bateria. Da mesma forma, a propriedade `cacheAsBitmapMatrix` disponível no AIR ou no Packager para iPhone não tem essa limitação.

O código a seguir altera o valor alfa no método de movimento, que altera a opacidade da maçã em cada quadro:

```
private function handleMovement(e:Event):void
{
        alpha = Math.random();
        x -= (x - destinationX)*.5;
        y -= (y - destinationY)*.5;

        if (Math.abs(x - destinationX) < 1 && Math.abs(y - destinationY) < 1)
            initPos();
}
```

O uso do armazenamento de bitmap em cache causa lentidão do desempenho. Alterar o valor alfa força o tempo de execução a atualizar o bitmap em cache na memória sempre que o valor alfa for modificado.

Filtros dependem de bitmaps que são atualizados sempre que a cabeça de reprodução de um clipe de filme armazenado em cache se move. Assim, usar um filtro define automaticamente a propriedade `cacheAsBitmap` como `true`. A figura a seguir ilustra um clipe de filme animado:

Clipe de filme animado

Evite usar filtros em conteúdo animado, visto que isso pode causar problemas de desempenho. Na figura a seguir, o designer adiciona um filtro de sombra:

Clipe de filme animado com filtro de sombra projetada

Como resultado, se a linha de tempo dentro do clipe de filme está sendo reproduzida, o bitmap precisa ser gerado novamente. O bitmap também precisa ser gerado novamente se o conteúdo for modificado de qualquer forma diferente de uma simples transformação x ou y. Cada quadro força o tempo de execução a redesenhar o bitmap, o que requer mais recursos da CPU, reduz o desempenho e consome mais vida útil da bateria.

Paul Trani fornece exemplos de utilização do Flash Professional e do ActionScript para otimizar gráficos usando bitmaps no seguinte vídeo de tutorial:

- Otimização de Gráficos
- Otimização de Gráficos com o ActionScript

Bitmaps armazenados em cache transformam matrizes no AIR

Define a propriedade `cacheAsBitmapMatrix` ao usar bitmaps armazenados em cache em aplicaitovos móveis do AIR.

No perfil móvel do AIR, você pode atribuir um objeto Matrix à propriedade `cacheAsBitmapMatrix` de um objeto de exibição. Quando você define esta propriedade, pode aplicar qualquer transformação bidimensional do objeto sem gerar novamente o bitmap armazenado em cache. Você também pode alterar a propriedade alpha sem gerar novamente o bitmap armazenado em cache. A propriedade `cacheAsBitmap` também precisa ser definida como `true` e o objeto precisa não ter nenhuma propriedade 3D definida.

Definir a propriedade `cacheAsBitmapMatrix` gera o bitmap armazenado em cache, mesmo se o objeto de exibição estiver fora da tela, oculto da visão ou tiver a propriedade `visible` definida como `false`. Redefinir a propriedade `cacheAsBitmapMatrix` usando um objeto matrix que contenha uma transformação diferente também gera novamente o bitmap armazenado em cache.

A transformação da matriz que você aplicar à propriedade `cacheAsBitmapMatrix` é aplicada ao objeto de exibição à medida que este é renderizado no cache de bitmap. Assim, se a transformação contiver uma escala 2x, a renderização do bitmap tem o dobro do tamanho da renderização do vetor. O renderizador se aplica à transformação inversa do bitmap armazenado em cache, de forma que a exibição final tenha a mesma aparência. Você pode reduzir o tamanho do bitmap armazenado em cache para reduzir a utilização da memóra, possivelmente abrindo mão da fidelidade da renderização. Você também pode dimensionar o bitmap com um tamanho maior para aumentar a qualidade da renderização em alguns casos, o que resulta em maior utilização da memória. Em geral, use uma matriz de identidade, que é uma matriz que não aplica nenhuma transformação, a fim de evitar alterações na aparência, conforme mostrado no seguinte exemplo:

```
displayObject.cacheAsBitMap = true;
displayObject.cacheAsBitmapMatrix = new Matrix();
```

Assim que a propriedade `cacheAsBitmapMatrix` for definida, você pode dimensionar, inclinar, girar e transladar o objeto sem acionar a nova geração do bitmap.

Você também pode alterar o valor alfa no intervalo entre 0 e 1. Se você alterar o valor alfa por meio da propriedade `transform.colorTransform` com uma transformação de cor, o alfa usado no objeto transform precisará estar entre 0 e 255. Alterar a transformação de cor de qualquer maneira implicará em nova geração do bitmap armazenado em cache.

Defina a propriedade `cacheAsBitmapMatrix` sempre que você definir `cacheAsBitmap` como `true` no conteúdo criado para dispositivos móveis. No entanto, considere as seguintes desvantagens potenciais. Depois que um objeto é girado, dimensionado ou inclinado, a renderização final pode exibir dimensionamento de bitmap ou artefatos de suavização de borda, quando comparada à renderização normal de vetor.

Armazenamento manual de bitmaps em cache

Use a classe BitmapData para criar o comportamento de armazenamento de bitmap personalizado em cache.

O exemplo a seguir reutiliza uma simples versão do bitmap rasterizado de um objeto de exibição e faz referência ao mesmo objeto BitmapData. Ao escalar cada objeto de exibição, o objeto BitmapData original na memória não é atualizado nem redesenhado. Esta abordagem economiza recursos da CPU e faz com que os aplicativos sejam executados mais rapidamente. Quando o objeto de exibição é dimensionado, o bitmap contido é alongado.

Esta é a classe `BitmapApple` atualizada:

```
package org.bytearray.bitmap
{
    import flash.display.Bitmap;
    import flash.display.BitmapData;
    import flash.events.Event;

    public class BitmapApple extends Bitmap
    {
        private var destinationX:Number;
        private var destinationY:Number;

        public function BitmapApple(buffer:BitmapData)
        {
            super(buffer);

            addEventListener(Event.ADDED_TO_STAGE,activation);
            addEventListener(Event.REMOVED_FROM_STAGE,deactivation);
        }

        private function activation(e:Event):void
        {
            initPos();
            addEventListener(Event.ENTER_FRAME,handleMovement);
        }

        private function deactivation(e:Event):void
        {
            removeEventListener(Event.ENTER_FRAME,handleMovement);
        }

        private function initPos():void
        {
            destinationX = Math.random()*(stage.stageWidth - (width>>1));
            destinationY = Math.random()*(stage.stageHeight - (height>>1));
        }

        private function handleMovement(e:Event):void
        {
            alpha = Math.random();

            x -= (x - destinationX)*.5;
            y -= (y - destinationY)*.5;

            if ( Math.abs(x - destinationX) < 1 && Math.abs(y - destinationY) < 1)
                initPos();
        }
    }
}
```

O valor alfa continua sendo modificado a cada quadro. O código a seguir envia o buffer de fonte original para cada ocorrência BitmapApple.

```
import org.bytearray.bitmap.BitmapApple;

const MAX_NUM:int = 100;
var holder:Sprite = new Sprite();

addChild(holder);

var holderVector:Vector.<BitmapApple> = new Vector.<BitmapApple>(MAX_NUM, true);
var source:AppleSource = new AppleSource();
var bounds:Object = source.getBounds(source);

var mat:Matrix = new Matrix();
mat.translate(-bounds.x,-bounds.y);

var buffer:BitmapData = new BitmapData(source.width+1, source.height+1, true, 0);
buffer.draw(source,mat);

var bitmapApple:BitmapApple;

for (var i:int = 0; i< MAX_NUM; i++)
{
    bitmapApple = new BitmapApple(buffer);

    holderVector[i] = bitmapApple;

    holder.addChild(bitmapApple);
}
```

Esta técnica utiliza apenas uma pequena quantidade de memória, porque apenas um bitmap armazenado individualmente é utilizado na memória e compartilhado por todas as ocorrências `BitmapApple`. Além disso, apesar de quaisquer modificações feitas às instâncias de `BitmapApple`, como alfa, rotação ou dimensionamento, o bitmap de fonte original não é atualizado nunca. O uso desta técnica impede a desaceleração do desempenho.

Para obter um bitmap final suave, defina a propriedade `smoothing` como `true`:

```
public function BitmapApple(buffer:BitmapData)
{
    super (buffer);

    smoothing = true;

    addEventListener(Event.ADDED_TO_STAGE, activation);
    addEventListener(Event.REMOVED_FROM_STAGE, deactivation);
}
```

O ajuste da qualidade do Palco pode aumentar o desempenho também. Defina a qualidade do Palco como `HIGH` antes da rasterização e, em seguida, mude-a para `LOW`:

```
import org.bytearray.bitmap.BitmapApple;

const MAX_NUM:int = 100;
var holder:Sprite = new Sprite();

addChild ( holder );

var holderVector:Vector.<BitmapApple> = new Vector.<BitmapApple>(MAX_NUM, true);
var source:AppleSource = new AppleSource();
var bounds:Object = source.getBounds ( source );

var mat:Matrix = new Matrix();
mat.translate ( -bounds.x, -bounds.y );

var buffer:BitmapData = new BitmapData ( source.width+1, source.height+1, true, 0 );

stage.quality = StageQuality.HIGH;

buffer.draw ( source, mat );

stage.quality = StageQuality.LOW;

var bitmapApple:BitmapApple;

for (var i:int = 0; i< MAX_NUM; i++ )
{
    bitmapApple = new BitmapApple( buffer );

    holderVector[i] = bitmapApple;

    holder.addChild ( bitmapApple );
}
```

Alterar a qualidade de Palco antes e depois de desenhar o vetor em um bitmap pode ser uma poderosa técnica de conseguir conteúdo suavizado na tela. Esta técnica pode ser eficaz independente da qualidade final do palco. Por exemplo, você pode conseguir um bitmap suavizado com texto suavizado, mesmo com a qualidade de Palco ajustada para `LOW`. Essa técnica não é possível com a propriedade `cacheAsBitmap`. Nesse caso, definir a qualidade do Palco como `LOW` atualiza a qualidade do vetor, o que atualiza as superfícies do bitmap na memória e atualiza a qualidade final.

Comportamentos de isolamento

Isole eventos como o `Event.ENTER_FRAME` em um manipulador único, quando possível.

O código pode ser otimizado ainda mais pelo isolamento do evento `Event.ENTER_FRAME` na classe `Apple` em um único manipulador. Esta técnica economiza recursos de CPU. O exemplo a seguir mostra esta aproximação diferente, onde a classe BitmapApple não manipula mais o comportamento do movimento:

```
package org.bytearray.bitmap
{
    import flash.display.Bitmap;
    import flash.display.BitmapData;

    public class BitmapApple extends Bitmap
    {
        private var destinationX:Number;
        private var destinationY:Number;

        public function BitmapApple(buffer:BitmapData)
        {
            super (buffer);

            smoothing = true;
        }
}
```

O código a seguir instancia as maçãs e manipula seu movimento em um único manipulador:

```
import org.bytearray.bitmap.BitmapApple;

const MAX_NUM:int = 100;
var holder:Sprite = new Sprite();

addChild(holder);

var holderVector:Vector.<BitmapApple> = new Vector.<BitmapApple>(MAX_NUM, true);
var source:AppleSource = new AppleSource();
var bounds:Object = source.getBounds(source);

var mat:Matrix = new Matrix();
mat.translate(-bounds.x,-bounds.y);

stage.quality = StageQuality.BEST;

var buffer:BitmapData = new BitmapData(source.width+1,source.height+1, true,0);
buffer.draw(source,mat);

stage.quality = StageQuality.LOW;

var bitmapApple:BitmapApple;

for (var i:int = 0; i< MAX_NUM; i++)
{
    bitmapApple = new BitmapApple(buffer);

    bitmapApple.destinationX = Math.random()*stage.stageWidth;
    bitmapApple.destinationY = Math.random()*stage.stageHeight;

    holderVector[i] = bitmapApple;

    holder.addChild(bitmapApple);
}

stage.addEventListener(Event.ENTER_FRAME,onFrame);
```

```
var lng:int = holderVector.length

function onFrame(e:Event):void
{
        for (var i:int = 0; i < lng; i++)
        {
            bitmapApple = holderVector[i];
            bitmapApple.alpha = Math.random();

            bitmapApple.x -= (bitmapApple.x - bitmapApple.destinationX) *.5;
            bitmapApple.y -= (bitmapApple.y - bitmapApple.destinationY) *.5;

            if (Math.abs(bitmapApple.x - bitmapApple.destinationX ) < 1 &&
                Math.abs(bitmapApple.y - bitmapApple.destinationY ) < 1)
            {
                    bitmapApple.destinationX = Math.random()*stage.stageWidth;
                    bitmapApple.destinationY = Math.random()*stage.stageHeight;
            }
        }
}
```

O resultado é um único evento `Event.ENTER_FRAME` manipulando o movimento, em vez de 200 manipuladores movendo cada maçã. Toda a animação pode ser facilmente pausada, o que pode ser útil em um jogo.

Por exemplo, um jogo simples pode usar o seguinte manipulador:

```
stage.addEventListener(Event.ENTER_FRAME, updateGame);
function updateGame (e:Event):void
{
    gameEngine.update();
}
```

A próxima etapa é fazer com que as maçãs interajam com o mouse ou o teclado, o que requer modificações na classe `BitmapApple`.

```
package org.bytearray.bitmap
{
    import flash.display.Bitmap;
    import flash.display.BitmapData;
    import flash.display.Sprite;

    public class BitmapApple extends Sprite
    {
        public var destinationX:Number;
        public var destinationY:Number;
        private var container:Sprite;
        private var containerBitmap:Bitmap;

        public function BitmapApple(buffer:BitmapData)
        {
            container = new Sprite();
            containerBitmap = new Bitmap(buffer);
            containerBitmap.smoothing = true;
            container.addChild(containerBitmap);
            addChild(container);
        }
}
```

O resultado são instâncias interativas de `BitmapApple`, como objetos Sprite tradicionais. No entanto, as instâncias são vinculadas a um único bitmap, que não é amostrado novamente quando os objetos de exibição são transformados.

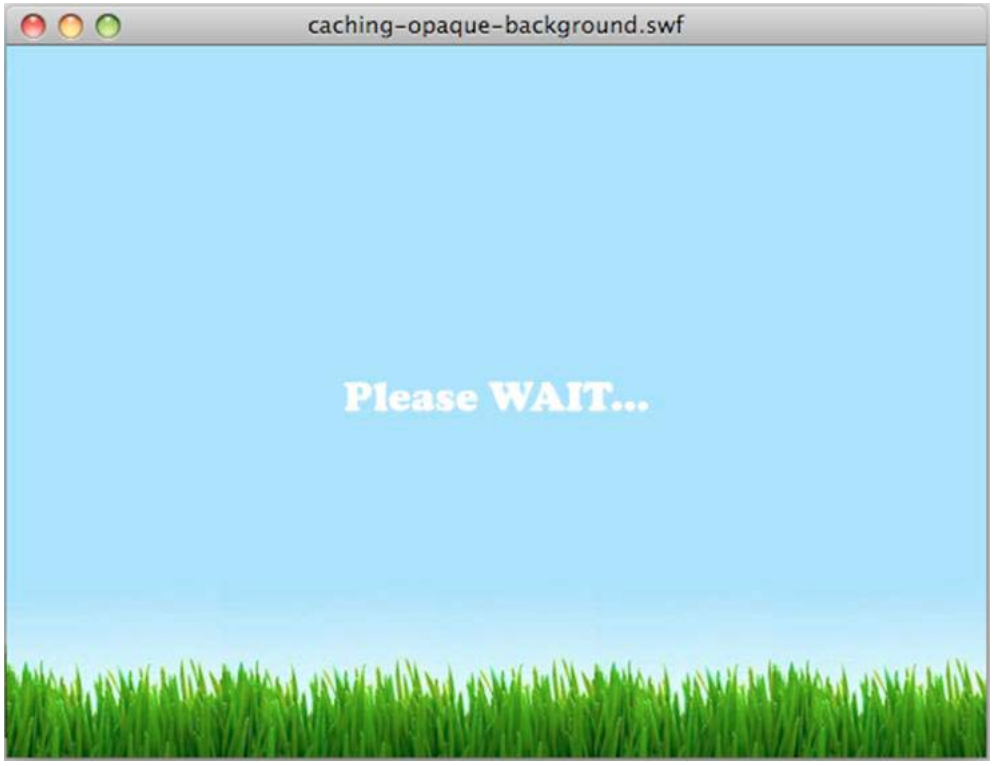
Renderização de objetos de texto

Use o recurso de armazenamento de bitmap no cache e a propriedade `opaqueBackground` para aprimorar o desempenho da renderização de texto.

O Flash Text Engine oferece algumas excelentes otimizações. No entanto, várias classes são necessárias para mostrar uma única linha de texto. Por isso, criar um campo de texto editável com a classe TextLine requer uma grande quantidade de memória e muitas linhas de código ActionScript. A classe TextLine é melhor aproveitada para texto estático não editável, que ela renderiza mais rapidamente, com menor exigência de memória.

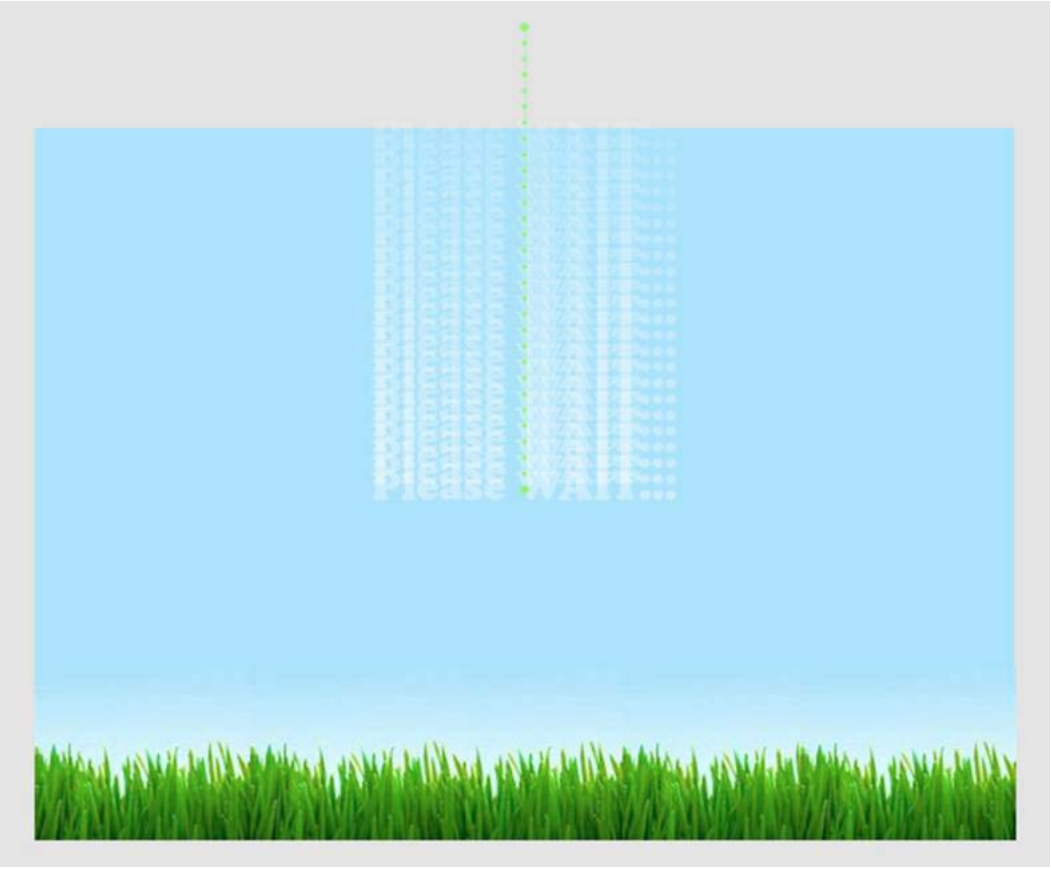
O recurso de armazenamento de bitmap no cache permite armazenar conteúdo de vetor em cache como bitmaps para aumentar o desempenho da renderização. Esse recurso é útil para conteúdo de vetor complexo e, também, quando usado com conteúdo de texto que requer processamento para ser renderizado.

O exemplo a seguir mostra como o recurso de armazenamento de bitmap em cache e a propriedade `opaqueBackground` podem ser usados para aumentar o desempenho da renderização. A figura a seguir ilustra uma tela típica de boas-vindas que pode ser exibida enquanto um usuário aguarda o carregamento de alguma coisa:



Tela Bem-vindo

A figura a seguir ilustra o easing aplicado programaticamente ao objeto TextField. O texto se aproxima/afasta lentamente desde a parte superior da cena até seu centro:



Easing de texto

O código a seguir cria o easing. A variável `preloader` armazena o objeto alvo atual para minimizar consultas de propriedade, o que pode afetar o desempenho:

```
wait_mc.addEventListener( Event.ENTER_FRAME, movePosition );

var destX:Number=stage.stageWidth/2;
var destY:Number=stage.stageHeight/2;
var preloader:DisplayObject;

function movePosition( e:Event ):void
{
    preloader = e.currentTarget as DisplayObject;

    preloader.x -= ( preloader.x - destX ) * .1;
    preloader.y -= ( preloader.y - destY ) * .1;

    if (Math.abs(preloader.y-destY)<1)
        preloader.removeEventListener( Event.ENTER_FRAME, movePosition );
}
```

A função `Math.abs()` pode ser movida em linha aqui para reduzir o número de chamados de função e conseguir mais aumento de desempenho. Uma melhor prática é usar o tipo int para as propriedades `destX` e `destY` para obter valores de ponto fixo. O uso do tipo int permite obter o encaixe perfeito dos pixels sem a necessidade de arredondar os valores manualmente por meio de métodos lentos como `Math.ceil()` ou `Math.round()`. Este código não arredonda as coordenadas para int, porque ao arredondar os valores constantemente, o objeto não se move de forma suave. O objeto pode ter movimentos instáveis, porque as coordenadas são encaixadas para o inteiro arredondado mais próximo em cada quadro. No entanto, esta técnica pode ser útil quando for definir uma posição final para um objeto de exibição. Não utilize o seguinte código:

```
// Do not use this code
var destX:Number = Math.round ( stage.stageWidth / 2 );
var destY:Number = Math.round ( stage.stageHeight / 2);
```

O seguinte código é muito mais rápido:

```
var destX:int = stage.stageWidth / 2;
var destY:int = stage.stageHeight / 2;
```

O código anterior pode ser otimizado ainda mais por meio do uso de operadores de deslocamento bit a bit para dividir os valores:

```
var destX:int = stage.stageWidth >> 1;
var destY:int = stage.stageHeight >> 1;
```

O recurso de armazenamento de bitmap em cache torna mais fácil para o tempo de execução renderizar objetos usando bitmaps dinâmicos. No exemplo atual, o clipe de filme que contém o objeto TextField é armazenado em cache:

```
wait_mc.cacheAsBitmap = true;
```

Outra maneira de aumentar o desempenho é remover a transparência alfa. A transparência alfa sobrecarrega o tempo de execução por desenhar imagens de bitmap transparentes, como no código anterior. Você pode usar a propriedade `opaqueBackground` para se desviar disso, especificando uma cor como fundo.

Quando a propriedade `opaqueBackground` é usada, a superfície do bitmap criado na memória ainda usa 32 bits. No entanto, o deslocamento alfa é definido como 255 e não é utilizada nenhuma transparência. Como resultado, a propriedade `opaqueBackground` não reduz o uso da memória, mas aumenta o desempenho da renderização quando o recurso de armazenamento de bitmap em cache é usado. O código a seguir contém todas as otimizações:

```
wait_mc.addEventListener( Event.ENTER_FRAME, movePosition );
wait_mc.cacheAsBitmap = true;

// Set the background to the color of the scene background
wait_mc.opaqueBackground = 0x8AD6FD;
var destX:int = stage.stageWidth >> 1;
var destY:int = stage.stageHeight >> 1;
var preloader:DisplayObject;

function movePosition ( e:Event ):void
{
    preloader = e.currentTarget as DisplayObject;

    preloader.x -= ( preloader.x - destX ) * .1;
    preloader.y -= ( preloader.y - destY ) * .1;

    if ( Math.abs ( preloader.y - destY ) < 1 )
        e.currentTarget.removeEventListener ( Event.ENTER_FRAME, movePosition );
}
```

Agora a animação está otimizada e o armazenamento do bitmap em cache foi otimizado por meio da remoção da transparência. Em dispositivos móveis, considere mudar a qualidade do Palco para `LOW` e `HIGH` durante os diferente estados da animação enquanto usa o recurso de armazenamento de bitmap em cache:

```
wait_mc.addEventListener( Event.ENTER_FRAME, movePosition );
wait_mc.cacheAsBitmap = true;
wait_mc.opaqueBackground = 0x8AD6FD;

// Switch to low quality
stage.quality = StageQuality.LOW;
var destX:int = stage.stageWidth>>1;
var destY:int = stage.stageHeight>>1;
var preloader:DisplayObject;

function movePosition( e:Event ):void
{
    preloader = e.currentTarget as DisplayObject;

    preloader.x -= ( preloader.x - destX ) * .1;
    preloader.y -= ( preloader.y - destY ) * .1;

    if (Math.abs(e.currentTarget.y-destY)<1)
    {
        // Switch back to high quality
        stage.quality = StageQuality.HIGH;
        preloader.removeEventListener( Event.ENTER_FRAME, movePosition );
    }
}
```

No entanto, neste caso, alterar a qualidade do Palco força o tempo de execução a gerar a superfície do objeto TextField novamente de modo a corresponder à qualidade atual do Palco. Por isso, é melhor não alterar a qualidade do Palco ao usar o recurso de armazenamento de bitmap em cache.

Uma abordagem de armazenamento manual de bitmap em cache poderia ter sido usada aqui. Para simular a propriedade `opaqueBackground`, o clipe de filme pode ser desenhado em um objeto BitmapData que não seja transparente, o que não força o tempo de execução a gerar novamente a superfície do bitmap.

Esta técnica funciona bem para conteúdo que não muda com o tempo. Entretanto, se o conteúdo do campo de texto puder ser alterado, considere usar uma estratégia diferente. Por exemplo, imagine um campo de texto que seja continuamente atualizado com uma porcentagem que representa quanto do aplicativo foi carregado. Se o campo de texto ou o objeto de exibição que o contém tiver sido armazenado em cache, sua superfície precisa ser gerada toda vez que o conteúdo for alterado. Não é possível usar o armazenamento manual de bitmap em cache aqui, porque o conteúdo do objeto de exibição é alterado constantemente. Esta alteração constante pode forçar você a chamar manualmente o método `BitmapData.draw()` para atualizar o bitmap armazenado.

Lembre-se de que, desde o Flash Player 8 (e o AIR 1.0), independentemente do valor da qualidade do Palco, um campo de texto com renderização definida como Suavização de borda para legibilidade permanece com a suavização perfeita da borda. Esta abordagem consome menos memória, mas requer mais processamento da CPU e sua renderização é um pouco mais lenta do que a do recurso de armazenamento de bitmap em cache.

O código a seguir usa essa abordagem:

```
wait_mc.addEventListener( Event.ENTER_FRAME, movePosition );

// Switch to low quality
stage.quality = StageQuality.LOW;
var destX:int = stage.stageWidth >> 1;
var destY:int = stage.stageHeight >> 1;
var preloader:DisplayObject;
function movePosition ( e:Event ):void
{
    preloader = e.currentTarget as DisplayObject;

    preloader.x -= ( preloader.x - destX ) * .1;
    preloader.y -= ( preloader.y - destY ) * .1;

    if ( Math.abs ( preloader.y - destY ) < 1 )
    {
        // Switch back to high quality
        stage.quality = StageQuality.HIGH;
        preloader.removeEventListener ( Event.ENTER_FRAME, movePosition );
    }
}
```

Não recomendamos usar essa opção (Suavização de borda para legibilidade) para texto em movimento. Ao escalar o texto, esta opção faz com que o texto tente ficar alinhado, o que produz um efeito de deslocamento. Entretanto, se o conteúdo do objeto de exibição for alterado constantemente e você precisar de texto com escala, poderá aumentar o desempenho em aplicativos móveis definindo a qualidade como `LOW`. Quando o movimento for concluído, mude a qualidade de volta para `HIGH`.

GPU

Renderização pela GPU em aplicativos Flash Player

Um novo recurso importante do Flash Player 10.1 é que ele pode utilizar a GPU para renderizar conteúdo gráfico em dispositivos móveis. No passado, gráficos eram renderizados através da CPU apenas. Utilizar a GPU otimiza a renderização de filtros, bitmaps, vídeo e texto. Tenha em mente que a renderização em GPU não é sempre preciso quanto a renderização em software. O conteúdo pode parecer um pouco espesso quando utilizar o renderizador em hardware. Além disso, o Flash Player 10.1 tem uma limitação que pode impedir que efeitos Pixel Bender na tela sejam renderizados. Estes efeitos podem surgir como um quadrado branco quando utilizamos a aceleração de hardware.

O Flash Player 10 tinha um recurso de aceleração de GPU, mas a GPU não era usada para calcular os gráficos. Ela era usada apenas para enviar todos os gráficos para a tela. No Flash Player 10.1, a GPU também é usada para calcular os gráficos, o que pode aumentar a velocidade de renderização significativamente. Isto também reduz a carga da CPU, o que é útil em dispositivos com recursos limitados, como dispositivos móveis.

O modo GPU é definido automaticamente durante a execução de conteúdo em dispositivos móveis, o que assegura o melhor desempenho possível. Embora `wmode` não precise mais ser definido como `gpu` para usar renderização pela GPU, ajustar `wmode` como `opaque` ou `transparent` desabilita a aceleração da GPU.

Nota: *O Flash Player no desktop ainda usa a CPU para fazer a renderização por software. A renderização por software é usada porque os drivers variam muito no desktop, e os drivers podem acentuar as diferenças de renderização. Também pode haver diferenças de renderização entre o desktop e alguns dispositivos móveis.*

Renderização pela GPU em aplicativos AIR móveis

Ative a aceleração gráfica por hardware em um aplicativo AIR incluindo `<renderMode>gpu</renderMode>` no descritor do aplicativo. É possível alterar os modos de renderização no tempo de execução. Em computadores desktop, a configuração `renderMode` é ignorada; no momento, não há suporte para a aceleração gráfica pela GPU.

Limitações do modo de renderização pela GPU

As seguintes limitações existem ao usar o modo de renderização pela GPU no AIR 2.5:

- Se a GPU não puder renderizar um objeto, ele não será exibido. Não há uma alternativa em caso de falha da renderização pela CPU.
- Não há suporte para os seguintes modos de mistura: camada, alfa, apagar, sobrepor, realçar, clarear e escurecer.
- Não existe suporte para filtros.
- Não há suporte para PixelBender.
- Muitas unidades de GPU têm tamanho de textura máximo de 1024x1024. No ActionScript, isso se converte no tamanho máximo final renderizado de um objeto de exibição após todas as transformações.
- A Adobe não recomenda usar o modo de renderização pela GPU em aplicativos AIR que reproduzam vídeo.
- No modo de renderização pela GPU, os campos de texto nem sempre são movidos para um local visível quando o teclado virtual é aberto. Para garantir que seu campo de texto fique visível enquanto o usuário insere texto, execute um dos procedimentos a seguir. Posicione o campo de texto na metade superior da tela ou mova-o para lá quando ele receber o foco.
- O modo de renderização pela GPU é desativado para alguns dispositivos nos quais o modo não funciona de maneira confiável. Consulte as notas da versão do AIR para o desenvolvedor para obter as informações mais recentes.

Melhores práticas do modo de renderização pela GPU

As orientações a seguir podem acelerar a renderização pela GPU:

- Limite o número de itens visíveis no palco. Cada item demora algum tempo para ser renderizado e composto com outros itens ao seu redor. Quando não quiser mais exibir um objeto de exibição, defina sua propriedade `visible` como `false`. Não basta movê-lo para fora do palco, ocultá-lo atrás de outro objeto ou definir sua propriedade `alpha` como 0. Se o objeto de exibição não for mais necessário, remova-o do palco com `removeChild()`.
- Reutilize objetos em vez de criá-los e destruí-los.
- Crie bitmaps com tamanhos próximos, porém, menores que 2^n por 2^m bits. As dimensões não precisam ser potências exatas de 2, mas devem ser próximas a uma potência de 2, sem serem maiores. Por exemplo, uma imagem com 31 por 15 pixels é renderizada mais rápido do que uma imagem de 33 por 17 pixels. (31 e 15 são justamente potências menores de 2: 32 e 16).
- Se possível, defina o parâmetro repeat como false ao chamar o método `Graphic.beginBitmapFill()`.
- Não exagere. Use a cor do fundo como fundo. Não estenda em camadas formas grandes umas em cima das outras. Há um custo para cada pixel a ser desenhado.
- Evite formas com pontas longas e finas, bordas com autointerseção ou muitos detalhes ao longo das bordas. Essas formas demoram mais para serem renderizadas do que os objetos de exibição com bordas suaves.
- Limite o tamanho dos objetos de exibição.
- Ative cacheAsBitMap e cacheAsBitmapMatrix para objetos de exibição cujos gráficos não sejam atualizados com frequência.

- Evite usar o API de desenho do ActionScript (classe Graphics) para criar gráficos. Em vez disso, quando possível, crie esses objetos estaticamente no tempo de autoria.
- Dimensione ativos de bitmap com o tamanho final antes de importá-los.

Modo de renderização pela GPU no AIR 2.0.3 móvel

A renderização pela GPU é more restritiva em aplicativos AIR móveis criados com o Packager para iPhone. A GPU é eficaz apenas para bitmaps, formas sólidas e objetos de exibição que têm a propriedade `cacheAsBitmap` definida. Também, para objetos que têm `cacheAsBitmap` e `cacheAsBitmapMatrix` definidas, a GPU pode renderizar com eficácia objetos que giram ou são dimensionados. A GPU é usada em tandem para outros objetos de exibição e isso normalmente resulta em um desempenho insatisfatório da renderização.

Dicas para otimizar o desempenho de renderização da GPU

Embora a renderização da GPU possa aumentar muito o desempenho do conteúdo de SWF, o projeto do conteúdo tem um papel importante. Lembre-sd de que as configuração que historicamente funcionaram bem na renderização de software às vezes não funcionam bem na renderização da GPU. As seguintes dicas podem ajudá-lo a obter um bom desempenho na renderização da GPU sem causar nenhum prejuízo à renderização don software.

Nota: *Os dispositivos móveis que suportam renderização de hardware frequentemente acessam o conteúdo de SWF dos sites da web. Portanto, uma melhor prática é considerar essas dicas ao criar qualquer conteúdo de SWF, garantindo a melhor experiência em todas as telas.*

- Evite usar `wmode=transparent` ou `wmode=opaque` em parâmetros HTML incorporados. Esses modos podem causar diminuição de desempenho. Eles também podem causar uma pequena perda de sincronização de áudio e vídeo nas renderizações de software e hardware. Além disso, muitas plataformas não suportam a renderização da GPU quando esses modos estão ativos, prejudicando significativamente o desempenho.
- Use somente os modos de mesclagem normal e alfa. Evite usa outros modos de mesclagem, especialmente o modo de mesclagem de camadas. Nem todos os modos de mesclagem podem ser reproduzidos fielmente quando renderizados com a GPU.
- Ao renderizar imagens vetoriais, a GPU as divide em malhas feitas de pequenos triângulos antes de desenhá-las. Esse processo é denominado tecelagem. A tecelagem tem um baixo custo de desempenho, que aumenta proporcionalmente à complexidade dos formatos. Para minimizar o impacto no desempenho, evite formatos mórficos, cuja tecelagem seja renderizada pela GPU em todos os quadros.
- Evite curvas autointersecionáveis, regiões curvas muito estreitas (tais como uma estreita lua crescente) e detalhes intrincados ao longo das bordas de um formato. Esses formatos são complexos para que a GPU os teça em malhas triangulares. Para entender o motivo, considere dois vetores: um quadrado de 500 × 500 e uma lua crescente de 100 × 10. Uma GPU pode renderizar facilmente o quadrado maior porque ele é simplesmente dois triângulos. No entanto, são necessários muitos triângulos para descrever a curva da lua crescente. Portanto, a renderização desse formato é mais complicada, embora envolva menos pixels.
- Evite grandes alterações na escala, porque essas alterações também podem fazer a GPU tecer as imagens novamente.

- Evite desenhos sobrepostos sempre que possível. Desenho sobreposto é a disposição de vários elementos gráficos em camadas para que ocultem um ao outro. Usando o renderizador de software, cada pixel é desenhado somente uma vez. Portanto, para a renderização do software o aplicativo não incorre em prejuízo ao desempenho, independentemente de quantos elementos gráficos estejam se cobrindo reciprocamente em uma localização de pixel. Por contraste, o renderizador de hardware desenha cada pixel para cada elemento, estejam outros elementos ocultando aquela região ou não. Se dois retângulos se sobrepuserem reciprocamente, o renderizador de hardware desenhará a região renderizada duas vezes, enquanto que o renderizador de software desenhará a região somente uma vez.

  Portanto, no computador pessoal, que usa o renderizador de software, geralmente você não observa um impacto do desenho sobreposto no desempenho. Contudo, muitas formas sobrepostas podem afetar negativamente o desempenho em dispositivos que usam renderização de GPU. Uma boa prática é remover os objetos da lista de exibição, em vez de ocultá-los.
- Evite usar um retângulo grande preenchido como plano de fundo. Em vez disso, defina a cor de fundo do Palco.
- Evite a repetição do modo de preenchimento bitmap padrão sempre que possível. Em vez disso, use o modo de fixação de bitmap para obter um desempenho melhor.

Operações assíncronas

Use versões de operações assíncronas em vez de síncronas quando disponíveis

As operações síncronas executam assim que o código dá a instrução e o código aguarda até completar as operações antes de continuar. Como resultado, as operações executam na fase do código do aplicativo do loop de quadros. Se uma operação síncrona demorar muito, ela estica o tamanho do loop de quadros, provavelmente fazendo com a exibição apareça congelada ou cortada.

Quando o código executa uma operação assíncrona, ele não necessariamente executa de imediato. Seu código e outros códigos do aplicativo no encadeamento da execução atual continuam executando. Desta forma, o tempo de execução desempenha a operação o mais rápido possível enquanto tenta evitar problemas de renderização. Em alguns casos, a execução acontece em segundo plano e não executa como parte do loop de quadros do tempo de execução. Finalmente, quando a operação completa, o tempo de execução despacha o evento e seu código pode ouvir esse evento para desempenhar mais trabalhos.

As operações assíncronas são programadas e divididas para evitar problemas de renderização. Como resultado, é muito mais fácil ter um aplicativo responsivo usando versões de operações assíncronas. Consulte “Desempenho percebido versus desempenho real” na página 3 para obter mais informações).

Contudo, existe alguma folga na execução assíncronas de operações. O tempo de execução real não pode ser mais longo para as operações assíncronas, especialmente as operações que demoram pouco tempo para ser concluídas.

No tempo de execução, muitas operações são inerentemente síncronas ou assíncronas e você não pode escolher como executá-las. No entanto, no Adobe Air, há três tipos de operações que você pode escolher desempenhar síncrona ou assincronamente.

- Operações da classe File e FileStream

  É possível desempenhar muitas operações da classe File síncrona ou assincronamente. Por exemplo, os métodos para copiar ou excluir um arquivo ou diretório e listar o conteúdo de um diretório têm todos versões assíncronas. Esses métodos são identificados pelo sufixo "Async" adicionado ao nome da versão assíncrona. Por exemplo, para excluir um arquivo assincronamente, chame o método `File.deleteFileAsync()` em vez do método `File.deleteFile()`.

  Ao utilizar um objeto FileStream para ler um arquivo ou gravar em um arquivo, a maneira como você abre o objeto FileStream determina se as operações de leitura e gravação são executadas assincronamente. Utilize o método `FileStream.openAsync()` para operações assíncronas. A gravação de dados é desempenhada assincronamente. A leitura de dados é feita em blocos de forma que os dados estejam disponíveis uma parte de cada vez. Por contraste, no modo síncrono, o objeto FileStream lê o arquivo inteiro antes de continuar a execução do código.

- Operações de banco de dados SQL locais

  Ao trabalhar com um banco de dados SQL local, todas as operações executadas por meio do objeto SQLConnection executam no modo síncrono ou assíncrono. Para especificar se essas operações executam assincronamente, abra a conexão com o banco de dados utilizando o método `SQLConnection.openAsync()` em vez do método `SQLConnection.open()`. Quando as operações do banco de dados executam assincronamente, são executadas em segundo plano. O mecanismo do banco de dados não executa no loop de quadros do tempo de execução; dessa forma, é menos provável que as operações do banco de dados causem problemas de renderização.

  Para ver estratégias adicionais para melhorar o desempenho com o banco de dados SQL local, consulte "Desempenho do banco de dados SQL" na página 89.

- Sombreadores independentes Pixel Bender

  A classe ShaderJob permite a você executar uma imagem ou conjunto de dados por meio do sombreador Pixel Bender e acessar os dados brutos do resultado. Como padrão, quando o método `ShaderJob.start()` é chamado, o sombreador executa assincronamente. A execução ocorre em segundo plano, não utilizando o loop de quadros do tempo de execução. Para fazer com que o objeto ShaderJob execute sincronamente (o que não é recomendado), passe o valor `true` para o primeiro parâmetro do método `start()`.

Além desses mecanismos incorporados para executar código assincronamente, você pode também estruturar seu código para executar assincronamente em vez de sincronamente. Se estiver escrevendo código para desempenhar uma tarefa potencialmente de execução longa, você pode estruturar seu código para que execute em partes. Dividir seu código em partes permite que o tempo de execução desempenhe suas operações de renderização no meio de blocos de execução do código, tornando menos provável problemas de renderização.

A seguir são listadas diversas técnicas para dividir o código: A principal ideia pertinente a essas técnicas é que seu código é escrito para desempenhar somente parte de seu trabalho ao mesmo tempo. É possível rastrear o quê o código faz e onde interrompe o trabalho. Utilize um mecanismo como, por exemplo, o objeto do cronômetro para verificar repetidamente se o trabalho continua e desempenha trabalho adicional em blocos até o trabalho completar.

Há poucos padrões estabelecidos para estruturar o código para dividir o trabalho dessa forma. Os seguintes artigos e bibliotecas de códigos descrevem esses padrões e fornecem códigos para ajudar você a implantá-los nos aplicativos:

- Execução assíncrona do ActionScript (artigo por Trevor McCauley com mais detalhes sobre segundo plano e também diversos exemplos de implantação)
- Análise e renderização de vários dados no Flash Player (artigo por Jesse Warden com detalhes de segundo plano e exemplos de duas abordagens "padrão do construtor" e "encadeamentos verdes")"

- Encadeamentos verdes (artigo por Drew Cummins descrevendo a técnica “encadeamentos verdes” com exemplo de código-fonte)
- greenthreads (biblioteca de código-fonte aberta por Charlie Hubbard, para implantar “encadeamentos verdes” no ActionScript. Consulte o Guia rápido de encadeamentos verdes para obter mais informações).
- Saiba mais consultando Threads in ActionScript 3, em http://www.adobe.com/go/learn_fp_as3_threads_br(artigo de Alex Harui, incluindo um exemplo de implementação da técnica “pseudoencadeamento”)

Janelas transparentes

Nos aplicativos AIR para desktop, considere o uso de uma janela de aplicativo retangular opaca em vez de uma janela transparente.

Para usar uma janela opaca como janela inicial de um aplicativo AIR para desktop, defina o seguinte valor no arquivo XML descritor do aplicativo:

```
<initialWindow>
    <transparent>false</transparent>
</initialWindow>
```

Em janelas criadas pelo código do aplicativo, crie o objeto NativeWindowInitOptions com a propriedade `transparent` definida em `false` (o padrão). Passe-o para o construtor NativeWindow ao criar o objeto NativeWindow:

```
// NativeWindow: flash.display.NativeWindow class

var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
initOptions.transparent = false;
var win:NativeWindow = new NativeWindow(initOptions);
```

Para o componente Flex Window, verifique se a propriedade transparent do componente está definida em false, o padrão, antes de chamar o método `open()` do objeto Window.

```
// Flex window component: spark.components.Window class

var win:Window = new Window();
win.transparent = false;
win.open();
```

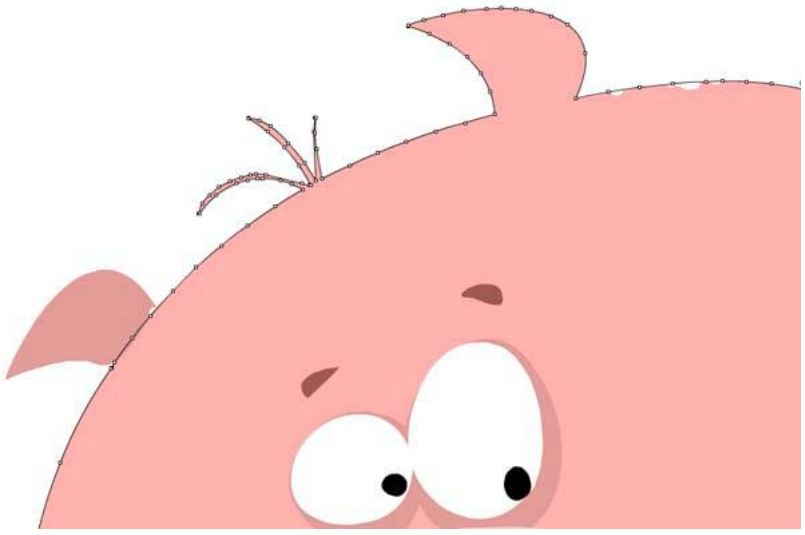
A janela transparente mostra potencialmente parte da área de trabalho do usuário ou outras janelas do aplicativo atrás da janela do aplicativo. Como resultado, o tempo de execução usa mais recursos para renderizar uma janela transparente. A janela retangular não transparente, quer use o cromo do sistema operacional ou o cromo personalizado, não tem a mesma carga de renderização.

Somente utilize a janela transparente quando é importante ter uma exibição não retangular ou ter exibição de conteúdo em segundo plano por meio da janela do aplicativo.

Suavização de forma vetor

Suavize as formas para melhorar o desempenho da renderização.

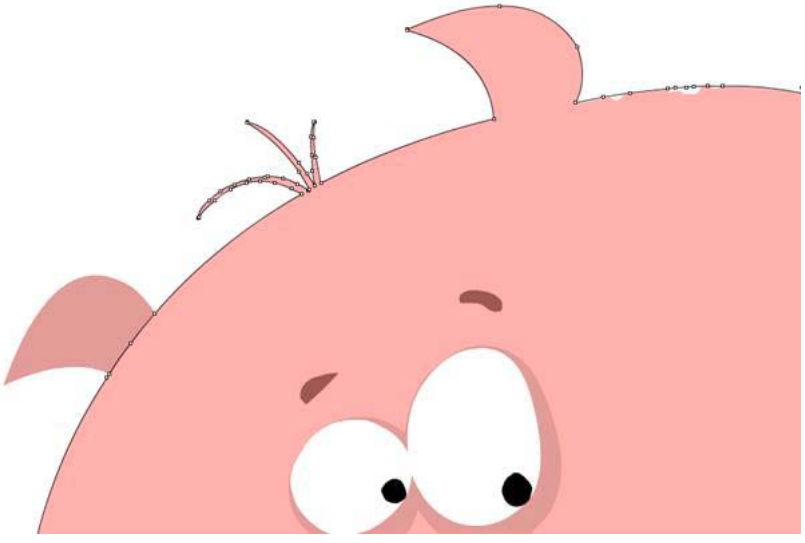
Ao contrário de bitmaps, a renderização de conteúdo de vetor requer uma grande quantidade de cálculos, especialmente para gradientes e caminhos complexos que contém muitos pontos de controle. Como designer ou desenvolvedor, certifique-se de que as formas estão otimizadas o suficiente. A imagem a seguir ilustra caminhos não simplificados com muitos pontos de controle:



Caminhos não otimizados

É possível remover pontos de controle adicionais utilizando a ferramenta Suavizar no Flash Professional. Uma ferramenta equivalente está disponível no Adobe® Illustrator® e o número total de pontos e caminhos pode ser visualizado no painel Informações do documento.

A suavização remove pontos de controle extra, reduzindo o tamanho final do arquivo SWF e melhorando o desempenho da renderização. A imagem a seguir ilustra os mesmos caminhos após a suavização:



Caminhos otimizados

A otimização não altera nada visualmente, desde que você não simplifique demais os caminhos. No entanto, a taxa de quadros média do aplicativo final pode ser melhorada de forma significativa pela simplificação de caminhos.

Capítulo 6: Otimização da interação na rede

Aprimoramentos para interação na rede

O Flash Player 10.1 e o AIR 2.5 introduzem um conjunto de novos recursos para otimização da rede em todas as plataformas, inclusive buffer circular e busca inteligente.

Buffer circular

Ao carregar conteúdo de mídia em dispositivos móveis, você pode enfrentar problemas que quase nunca esperaria em um computador desktop. Por exemplo, será mais provável que você esgote o espaço em disco ou a memória. Ao carregar vídeo, a versão para deskto do Flash Player 10.1 e do AIR 2.5 baixa e armazena em cache todo o arquivo FLV (ou arquivo MP4) no disco rígido. Em seguida, o tempo de execução reproduz o vídeo a partir daquele arquivo de cache. O esgotamento do espaço em disco é incomum. Se essa situação ocorrer, o tempo de execução do desktop para a reprodução do vídeo.

Um dispositivo móvel pode mais facilmente ficar sem espaço em disco. Se o espaço em disco do dispositivo se esgotar, o tempo de execução não para a reprodução, como acontece com o tempo de execução no desktop. Em vez disso, o tempo de execução começa a reutilizar o arquivo de cache gravando nele novamente a partir do começo do arquivo. O usuário pode continuar assistindo ao vídeo. O usuário não pode fazer uma busca na área do vídeo que foi regravada, exceto no início do arquivo. O buffer circular não é iniciado, por padrão. Ele pode ser iniciado durante a reprodução, e também no começo da reprodução, se o filme for maior do que o espaço em disco ou a RAM. O tempo de execução exige pelo menos 4 MB de RAM ou 20 MB de espaço em disco para usar o buffer circular.

Nota: *Se houver espaço em disco suficiente no dispositivo, a versão móvel do tempo de execução se comportará da mesma maneira que no desktop. Lembre-se de que um buffer na RAM é usado como reserva se o dispositivo não tiver disco ou se o disco estiver cheio. É possível definir um limite de tamanho para o arquivo de cache e para o buffer na RAM no momento da compilação. Alguns arquivos MP4 têm uma estrutura que requer que todo o arquivo seja baixado antes do início da reprodução. O tempo de execução detecta esses arquivos e impede o download se não houver espaço suficiente em disco, o que faz com que o arquivo MP4 não possa ser reproduzido. Pode ser melhor nem solicitar o download desses arquivos.*

Como desenvolvedor, lembre-se de que a busca funciona somente nos limites do fluxo armazenado em cache. Algumas vezes, pode ocorrer a falha de `NetStream.seek()` se o deslocamento estiver fora do intervalo e, nesse caso, um evento `NetStream.Seek.InvalidTime` é acionado.

Busca inteligente

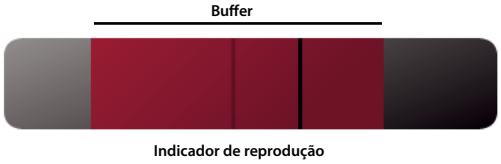
Nota: *O recurso de busca inteligente requer o Adobe® Flash® Media Server 3.5.3.*

O Flash Player 10.1 e o AIR 2.5 introduzem um novo comportamento, chamado busca inteligente, que aprimora a experiência do usuário ao reproduzir fluxo de vídeo. Se o usuário busca um destino dentro dos limites do buffer, o tempo de execução reutiliza o buffer para oferecer a busca instantânea. Em versões anteriores do tempo de execução, o buffer não era reutilizado. Por exemplo, se um usuário estivesse reproduzindo um vídeo a partir de um servidor de fluxo e o tempo do buffer estivesse definido como 20 segundos (`NetStream.bufferTime`), e o usuário tentasse buscar 10 segundos para a frente, o tempo de execução descartaria todos os dados do buffer em vez de reutilizar os 10 segundos que já foram carregados. Esse comportamento forçava o tempo de execução a solicitar novos dados do servidor com frequência muito maior e gerar um desempenho insuficiente da reprodução em conexões lentas.

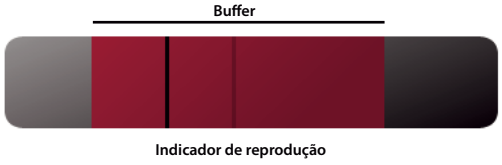
A figura abaixo ilustra como o buffer se comportava nas versões anteriores do tempo de execução. A propriedade `bufferTime` especifica o número de segundos para pré-carregar de forma que, se a conexão for perdida, o buffer pode ser usado sem parar o vídeo:

Comportamento do buffer antes do recurso de busca inteligente

Com o recurso de busca inteligente, o tempo de execução agora usa o buffer para fornecer busca instantânea para frente ou para trás quando o usuário arrasta o vídeo. A figura a seguir ilustra o novo comportamento:



Busca adiante com o recurso de busca inteligente



Busca para trás com o recurso de busca inteligente

A busca inteligente reutiliza o buffer quando o usuário busca mais adiante ou mais atrás, para que a experiência de reprodução seja mais rápida e suave. Um dos benefícios desse novo comportamento é a economia de largura de banda para publicadores de vídeo. No entanto, se a busca estiver fora dos limites do buffer, o comportamento padrão ocorre e o tempo de execução solicita novos dados do servidor.

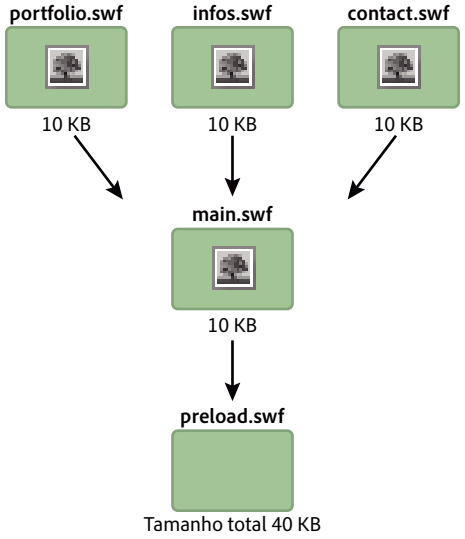
Nota: *Esse comportamento não se aplica ao download progressivo do vídeo.*

Para usar a busca inteligente, defina `NetStream.inBufferSeek` como `true`.

Conteúdo externo

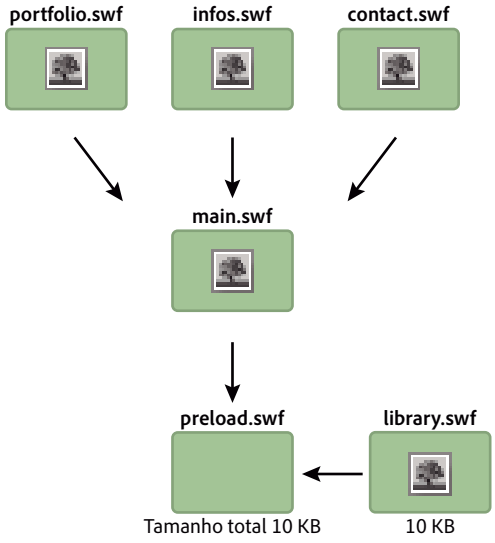
Divida seu aplicativo em vários arquivos SWF.

Dispositivos móveis podem ter acesso limitado à rede. Para carregar seu conteúdo rapidamente, divida seu aplicativo em vários arquivos SWF. Tente reutilizar a lógica e os ativos de código em todo o aplicativo. Por exemplo, considere um aplicativo que tenha sido dividido em vários arquivos SWF, conforme mostrado no seguinte diagrama:



Aplicativo dividido em diversos arquivos SWF

Neste exemplo, cada arquivo SWF contém sua própria cópia do mesmo bitmap. Essa duplicação pode ser evitada pelo uso de uma biblioteca compartilhada no tempo de execução, como ilustrado no diagrama a seguir:



Utilizando uma biblioteca compartilhada de tempo de execução

Com o uso dessa tecnologia, uma biblioteca compartilhada em tempo real é carregada para disponibilizar o bitmap para os outros arquivos SWF. A classe ApplicationDomain armazena todas as definições de classe que tiverem sido carregadas e as disponibiliza no tempo de execução por meio do método `getDefinition()`.

Uma biblioteca compartilhada no tempo de execução também pode conter toda a lógica do código. Todo o aplicativo pode ser atualizado no tempo de execução, sem a necessidade de recompilação. O código a seguir carrega uma biblioteca compartilhada em tempo de execução e extrai a definição contida no arquivo SWF no tempo de execução. Essa técnica pode ser usada com fontes, bitmaps, sons ou qualquer classe ActionScript:

```
// Create a Loader object
var loader:Loader = new Loader();

// Listen to the Event.COMPLETE event
loader.contentLoaderInfo.addEventListener(Event.COMPLETE, loadingComplete );

// Load the SWF file
loader.load(new URLRequest("library.swf") );
var classDefinition:String = "Logo";

function loadingComplete(e:Event ):void
{
    var objectLoaderInfo:LoaderInfo = LoaderInfo ( e.target );

    // Get a reference to the loaded SWF file application domain
    var appDomain:ApplicationDomain = objectLoaderInfo.applicationDomain;

    // Check whether the definition is available
    if ( appDomain.hasDefinition(classDefinition) )
    {
        // Extract definition
        var importLogo:Class = Class ( appDomain.getDefinition(classDefinition) );

        // Instantiate logo
        var instanceLogo:BitmapData = new importLogo(0,0);

        // Add it to the display list
        addChild ( new Bitmap ( instanceLogo ) );
    } else trace ("The class definition " +  classDefinition + " is not available.");
}
```

A obtenção da definição pode ser facilitada pelo carregamento das definições de classe no domínio do aplicativo do arquivo SWF:

```
// Create a Loader object
var loader:Loader = new Loader();

// Listen to the Event.COMPLETE event
loader.contentLoaderInfo.addEventListener ( Event.COMPLETE, loadingComplete );

// Load the SWF file
loader.load ( new URLRequest ("rsl.swf"), new LoaderContext ( false,
ApplicationDomain.currentDomain) );
var classDefinition:String = "Logo";

function loadingComplete ( e:Event ):void
{
    var objectLoaderInfo:LoaderInfo = LoaderInfo ( e.target );

    // Get a reference to the current SWF file application domain
    var appDomain:ApplicationDomain = ApplicationDomain.currentDomain;

    // Check whether the definition is available
    if (appDomain.hasDefinition( classDefinition ) )
    {
        // Extract definition
        var importLogo:Class = Class ( appDomain.getDefinition(classDefinition) );

        // Instantiate it
        var instanceLogo:BitmapData = new importLogo(0,0);

        // Add it to the display list
        addChild ( new Bitmap ( instanceLogo ) );
    } else trace ("The class definition " +  classDefinition + " is not available.");
}
```

Agora as classes disponíveis no arquivo SWF carregado podem ser usadas chamando-se o método `getDefinition()` no domínio do aplicativo atual. Também é possível acessar as classes chamando o método `getDefinitionByName()`. Essa técnica economiza largura de banda por carregar fontes e grandes ativos somente uma vez. Os ativos nunca são exportados em nenhum outro arquivo SWF. A única limitação é que o aplicativo precisa ser testado e executado por meio do arquivo loader.swf. Esse arquivo carrega os ativos primeiro e, em seguida, carrega os diferentes arquivos SWF que compõem o aplicativo.

Erros de entrada e saída

Fornece manipuladores de eventos e mensagens de erro para erros de entrada e saída.

Em um dispositivo móvel, a rede pode ser menos confiável que em um computador desktop conectado com alta velocidade à Internet. O acesso a conteúdos externos em dispositivos móveis tem duas limitações: disponibilidade e velocidade. Portanto, certifique-se de que os ativos sejam leves e adicione manipuladores para cada evento IO_ERROR para fornecer feedback ao usuário.

Por exemplo, imagine que um usuário está navegando por seu site com um dispositivo móvel e subitamente perde a conexão de rede entre duas estações de metrô. Um ativo dinâmico estava sendo carregado quando a conexão foi perdida. No computador desktop, você pode usar um ouvinte de eventos vazio para evitar a exibição de um erro no tempo de execução, porque esse cenário quase nunca ocorreria. Entretanto, em um dispositivo móvel, você precisa lidar com a situação com mais do que um simples ouvinte vazio.

O código a seguir não responde a um erro de entrada e saída. Não o use como mostrado:

```
var loader:Loader = new Loader();
loader.contentLoaderInfo.addEventListener( Event.COMPLETE, onComplete );
addChild( loader );
loader.load( new URLRequest ("asset.swf" ) );

function onComplete( e:Event ):void
{
    var loader:Loader = e.currentTarget.loader;
    loader.x = ( stage.stageWidth - e.currentTarget.width ) >> 1;
    loader.y = ( stage.stageHeight - e.currentTarget.height ) >> 1;
}
```

Uma melhor prática é lidar com a falha e fornecer uma mensagem de erro ao usuário. O código a seguir, lida adequadamente com o erro:

```
var loader:Loader = new Loader();
loader.contentLoaderInfo.addEventListener ( Event.COMPLETE, onComplete );
loader.contentLoaderInfo.addEventListener ( IOErrorEvent.IO_ERROR, onIOError );
addChild ( loader );
loader.load ( new URLRequest ("asset.swf" ) );

function onComplete ( e:Event ):void
{
      var loader:Loader = e.currentTarget.loader;
      loader.x = ( stage.stageWidth - e.currentTarget.width ) >> 1;
      loader.y = ( stage.stageHeight - e.currentTarget.height ) >> 1;
}

function onIOError ( e:IOErrorEvent ):void
{
      // Show a message explaining the situation and try to reload the asset.
      // If it fails again, ask the user to retry when the connection will be restored
}
```

Como melhor prática, lembre-se de oferecer uma maneira de o usuário carregar o conteúdo novamente. Esse comportamento pode ser implementado no manipulador `onIOError()`.

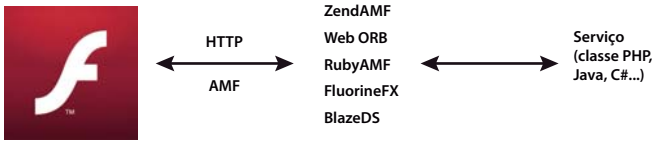
Flash Remoting

Utilize o Flash Remoting e o AMF para comunicação de dados cliente servidor otimizada.

Você pode usar XML para carregar conteúdo remoto em arquivos SWF. Entretanto, XML é texto simples que o tempo de execução carrega e analisa. O XML funciona melhor para aplicativos que carregam uma quantidade limitada de conteúdo. Se você estiver desenvolvendo um aplicativo que carrega uma grande quantidade de conteúdo, considere usar a tecnologia Flash Remoting e o formato AMF (Action Message Format).

AMF é um formato binário usado para compartilhar dados entre um servidor e o tempo de execução. O uso de AMF reduz o tamanho dos dados e melhora a velocidade de transmissão. Como o AMF é um formato nativo para o tempo de execução, enviar dados AMF para o tempo de execução evita serialização e deserialização com alto consumo de memória no lado do cliente. O gateway remoto lida com essas tarefas. Ao enviar um tipo de dado ActionScript para um servidor, o gateway remoto lida com a serialização no servidor. O gateway também envia o tipo de dado correspondente para você. Esse tipo de dado é uma classe criada no servidor que expõe um conjunto de métodos que podem ser chamados a partir do tempo de execução. Os gateways do Flash Remoting incluem ZendAMF, FluorineFX, WebORB e BlazeDS, um gateway oficial Java Flash Remoting de código aberto da Adobe.

A figura a seguir ilustra o conceito de Flash Remoting:



Flash Remoting

O exemplo a seguir usa a classe NetConnection para conectar-se a um gateway Flash Remoting:

```
// Create the NetConnection object
var connection:NetConnection = new NetConnection ();

// Connect to a Flash Remoting gateway
connection.connect ("http://www.yourserver.com/remotingservice/gateway.php");

// Asynchronous handlers for incoming data and errors
function success ( incomingData:* ):void
{
    trace( incomingData );
}

function error ( error:* ):void
{
    trace( "Error occured" );
}

// Create an object that handles the mapping to success and error handlers
var serverResult:Responder = new Responder (success, error);

// Call the remote method
connection.call ("org.yourserver.HelloWorld.sayHello", serverResult, "Hello there ?");
```

A conexão a um gateway remoto é simples. No entanto, o uso do Flash Remoting pode ser facilitado ainda mais pelo uso da classe RemoteObject incluída no SDK do Adobe® Flex®.

Nota: *Arquivos SWC externos, como os da estrutura Flex, podem ser usados dentro de um projeto do Adobe® Flash® Professional. O uso de arquivos SWC permite usar a classe RemoteObject e suas dependências sem usar o resto do SDK do Flex. Desenvolvedores avançados podem até mesmo se comunicar com um gateway remoto diretamente, por meio da classe Socket, se necessário.*

ap

Armazene os ativos localmente depois de carregá-los, em vez de carregá-los da rede cada vez que for necessário.

Se o aplicativo carregar ativos como, por exemplo, mídia ou dados, armazene os ativos em cache salvando-os no dispositivo local. Para ativos que não alteram com frequência, considere atualizar o armazenamento em intervalos. Por exemplo, seu aplicativo poderia verificar uma nova versão de um arquivo de imagem uma vez por dia ou verificar dados renovados a cada duas horas.

É possível armazenar ativos de várias maneiras, dependendo do tipo e natureza do ativo:

- Ativos de mídia como imagens e vídeo: salve os arquivos no sistema de arquivos, usando as classes File e FileStream
- Os valores de dados individuais ou pequenos conjuntos de dados: salve os valores como objetos locais compartilhados utilizando a classe SharedObject
- Conjunto de dados maiores: salve os dados em um banco de dados local ou coloque-os em série e os salve para um arquivo

Para armazenar valores de dados em cache, o projeto AS3CoreLib de código-fonte aberto inclui a classe ResourceCache que faz o carregamento e o armazenamento em cache do trabalho.

Capítulo 7: Trabalhando com mídia

Vídeo

Para obter informações sobre como otimizar o desempenhjo de vídeo em dispositivos móveis, consulte Otimização do conteúdo da web para entrega em dispositivos móveis no site Adobe Developer Connection.

Em particular, consulte as seções chamadas:

- *Reprodução de vídeo em dispositivos móveis*
- *Exemplos de códigos*

Essas seções contêm informações para o desenvolvimento de reprodutores de vídeo para dispositivos móveis, tais como:

- Diretrizes de codificação de vídeo
- Melhores práticas
- Como definir o perfil de desempenho do reprodutor de vídeo
- Uma implementação de referência do reprodutor de vídeo

StageVideo

Use a classe StageVideo para aproveitar a aceleração por hardware para apresentar o vídeo.

Para obter informações sobre como usar o objeto StageVideo object, consulte Utilização da classe StageVideo para renderização acelerada por hardware no Guia do desenvolvedor do ActionScript 3.0.

Áudio

A partir do Flash Player 9.0.115.0 e do AIR 1.0, o tempo de execução pode reproduzir arquivos AAC (AAC principal, AAC LC, e SBR). Uma otimização simples pode ser feita usando arquivos AAC em vez de arquivos MP3. O formato AAC oferece melhor qualidade e arquivos menores do que os do formato MP3 a uma taxa de bits equivalente. Reduzir o tamanho do arquivo economiza largura de banda, que pode ser um fator importante em dispositivos móveis que não oferecem conexões de internet de alta velocidade.

Decodificação de áudio no hardware

Semelhante a decodificação de vídeo, decodificação de áudio também necessita de grandes ciclos de CPU e pode ser otimizada utilizando o hardware do dispositivo. O Flash Player 10.1 e o AIR 2.5 detectam e usam drivers de hardware de áudio para melhorar o desempenho ao decodificar arquivos AAC (perfis LC, HE/SBR) ou arquivos MP 3 (não há suporte para PCM). O uso da CPU é drasticamente reduzido, o que resulta em menor uso de bateria e deixa a CPU disponível para outras operações.

Nota: *Ao utilizar o formato AAC, o perfil AAC principal não é suportado em dispositivos por conta da falta de suporte a hardware na maioria dos dispositivos.*

A decodificação de áudio em hardware é transparente para o usuário e para o desenvolvedor. Quando o tempo de execução começa a reproduzir fluxos de áudio, ele verifica o hardware primeiro, assim como faz com vídeo. Se um driver de hardware estiver disponível e houver suporte para o formato de áudio, a decodificação de áudio por hardware terá início. No entanto, mesmo se a decodificação do fluxo de entrada AAC ou MP3 possa ser feita através do hardware, às vezes o hardware não pode processar todos os efeitos. Por exemplo, às vezes o hardware não processa mixagem e repetições de amostragem de áudio, dependendo das limitações de hardware.

Capítulo 8: Desempenho do banco de dados SQL

Design de aplicativo para desempenho de banco de dados

Não altere a propriedade `text` do objeto SQLStatement depois de executá-lo. Em vez disso, use uma instância de SQLStatement em cada instrução SQL e use os parâmetros da instrução para fornecer valores diferentes.

Antes de qualquer instrução SQL ser executada, o tempo de execução a prepara (compila) para determinar as etapas realizadas internamente para executar a instrução. Quando você chama `SQLStatement.execute()` em uma ocorrência de SQLStatement que não foi executada anteriormente, a instrução é preparada de forma automática antes de ser executada. Nas chamadas subsequentes do método `execute()`, desde que a propriedade `SQLStatement.text` não tenha sido alterada, a instrução ainda será preparada. Como resultado, ela será executada mais rapidamente.

Para aproveitar ao máximo a capacidade de reutilizar instruções, se os valores alterarem entre as execuções de uma instrução, utilize parâmetros de instrução para personalizar a sua instrução. (Os parâmetros de instrução são especificados através da propriedade de matriz associativa `SQLStatement.parameters`). Diferentemente de quando se altera a propriedade `text` da ocorrência de SQLStatement, se você alterar os valores de parâmetros de instrução, o tempo de execução não terá de preparar a instrução novamente.

Quando você reutiliza uma ocorrência de SQLStatement, o aplicativo deve armazenar uma referência à ocorrência de SQLStatement depois que ela foi preparada. Para isso, declare a variável como uma variável com escopo de classe e não como uma variável com escopo de função. Uma boa maneira de fazer isso é estruturar o aplicativo para que a instrução SQL seja colocada em uma única classe. Um grupo de instruções que são executadas em combinação também pode ser colocado em uma única classe. (Essa técnica é conhecida por utilizar o padrão de design Command). Ao definir as instâncias de SQLStatement como variáveis da classe, elas persistirão enquanto a instância da classe wrapper existir no aplicativo. No mínimo, basta definir uma variável que contenha a ocorrência de SQLStatement fora de uma função para que a ocorrência persista na memória. Por exemplo, declare a ocorrência de SQLStatement como variável de membro em uma classe do ActionScript ou como variável não função em um arquivo JavaScript. Em seguida, você pode definir os valores de parâmetro da instrução e chamar o método `execute()` correspondente quando quiser executar a consulta.

Utilize os índices do banco de dados para melhorar a velocidade da execução para comparar e ordenar dados.

Quando você cria um índice para uma coluna, o banco de dados armazena uma cópia dos dados dessa coluna. A cópia é mantida ordenada em ordem numérica ou alfabética. A ordenação permite que o banco de dados faça rapidamente a correspondência dos valores (como ao utilizar o operador de igualdade) e ordene os dados dos resultados usando a cláusula `ORDER BY`.

Os índices de banco de dados são mantidos continuamente atualizados, o que faz com que as operações de atualização de dados (INSERT ou UPDATE) naquela tabela tornem-se um pouco mais lentas. No entanto, o aumento na velocidade da recuperação de dados pode ser significativo. Por causa dessa alteração de desempenho, você não deve simplesmente indexar cada coluna de cada tabela. Em vez disso, use uma estratégia para definir seus índices. Utilize as seguintes orientações para planejar a estratégia de indexação:

- Colunas de índices utilizadas em tabelas conjuntas, em cláusulas WHERE ou cláusulas ORDER BY

- Se as colunas são utilizadas em conjunto com frequência, indexe-as em conjunto em um índice simples
- Em uma coluna que contém dados de texto que você recuperou ordenada por ordem alfabética, especifique o agrupamento COLLATE NOCASE para o índice


Considere as instruções SQL de pré-compilação durante os tempos inativos do aplicativo .

Na primeira vez que uma instrução SQL é executada, ela é mais lenta porque o texto SQL é preparado (compilado) pelo mecanismo do banco de dados. Visto que preparar e executar uma instrução pode ser uma operação trabalhosa, uma boa estratégia é pré-carregar os dados iniciais e então executar outras instruções em segundo plano:

1. Carregue os dados de que o aplicativo precisa primeiro.
2. Quando as primeiras operações de inicialização do aplicativo forem concluídas, ou durante algum período de "inatividade" no aplicativo, execute outras instruções.

Por exemplo, suponha que o aplicativo não acessa o banco de dados para exibir a tela inicial. Nesse caso, aguarde até que a tela exiba antes de abrir a conexão do banco de dados. Finalmente, crie as ocorrências SQL e execute as ocorrências que conseguir.

Como alternativa, suponha que, quando o seu aplicativo é inicializado, ele imediatamente exibe alguns dados, como o resultado de uma determinada consulta. Nesse caso, vá em frente e execute a ocorrência de SQLStatement referente a essa consulta. Depois que os dados iniciais forem carregados e exibidos, crie ocorrências de SQLStatement para outras operações de banco de dados e, se possível, execute outras instruções que serão necessárias posteriormente.

Na prática, se você estiver reutilizando ocorrências de SQLStatement, o tempo adicional para preparar a instrução corresponde apenas a custos iniciais. Provavelmente não tem grande impacto no desempenho global.


Operações de alterações de dados SQL em múltiplos grupos em uma transação.

Suponha que você está executando muitas instruções SQL que envolvem a inclusão ou a alteração de dados (instruções`INSERT` ou `UPDATE`). Você pode conseguir um aumento de desempenho considerável se executar todas as instruções em uma transação explícita. Se você não iniciar uma transação de forma explícita, cada uma das instruções será executada em sua própria transação criada automaticamente. Depois que a execução de cada transação (cada instrução) for concluída, o tempo de execução gravará os dados resultantes no arquivo de banco de dados do disco.

Por outro lado, pense no que acontece se você cria uma transação explicitamente e executa as instruções no contexto dessa transação. O tempo de execução faz todas as alterações na memória e, depois, grava todas as alterações no arquivo de banco de dados de uma só vez quando a transação é confirmada. Gravar dados em disco normalmente é a parte mais demorada da operação. Como consequência, gravar no disco de uma só vez e não uma vez por instrução SQL pode melhorar o desempenho significativamente.


O processamento de consultas `SELECT` grandes resulta em partes utilizando o método `execute()` da classe SQLStatement (com parâmetro `prefetch`) e o método `next()`.

Suponha que você executa uma instrução SQL que recupera um conjunto de resultados grande. O aplicativo então processa cada fila de dados em um loop. Por exemplo, formata os dados ou cria objetos a partir dele. O processamento desses dados pode demorar muito tempo, o que faz com ocorram problemas de renderização como, por exemplo, tela congelada ou não responsiva. Conforme descrito em "Operações assíncronas" na página 75, uma solução é dividir o trabalhos em blocos. A API do banco de dados SQL facilita a divisão do processamento de dados.

O método `execute()` da classe SQLStatement tem um parâmetro `prefetch` opcional (o primeiro parâmetro) Se fornecer o valor, o parâmetro especifica o número máximo de filas de resultado que o banco de dados retorna quando a execução completa:

```
dbStatement.addEventListener(SQLEvent.RESULT, resultHandler);
dbStatement.execute(100); // 100 rows maximum returned in the first set
```

Depois que o primeiro conjunto de dados de resultados retorna, você pode chamar o método `next` para continuar a executar a instrução e recuperar outro conjunto de filas de resultados. De forma semelhante ao método `execute()`, o método `next` aceita o parâmetro `prefetch` para especificar o número máximo de filas a retornar:

```
// This method is called when the execute() or next() method completes
function resultHandler(event:SQLEvent):void
{
    var result:SQLResult = dbStatement.getResult();
    if (result != null)
    {
        var numRows:int = result.data.length;
        for (var i:int = 0; i < numRows; i++)
        {
            // Process the result data
        }

        if (!result.complete)
        {
            dbStatement.next(100);
        }
    }
}
```

Você pode continuar chamando o método `next()` até que os dados carreguem. Conforma mostra a listagem anterior, é possível determinar quando os dados foram todos carregados. Verifique a propriedade `complete` do objeto SQLResult criada cada vez que o método `execute()` ou `next()` termina.

Nota: *Use o parâmetro `prefetch` e o método `next()` para dividir o processamentos dos dados de resultado. Não use este parâmetro e este método para limitar os resultados de uma pesquisa a uma parte do seu conjunto de resultados. Se quiser recuperar um subconjunto de linhas em uma definição de resultado de uma instrução, use a cláusula `LIMIT` da instrução `SELECT`. Se o conjunto de resultados for grande, ainda é possível utilizar o parâmetro `prefetch` e o método `next()` para dividir o processamento dos resultados.*

Considere a utilização de múltiplos objetos SQLConnection assíncronos com um banco de dados simples para executar múltiplas instruções simultaneamente..

Quando o objeto SQLConnection é conectado a um banco de dados utilizando o método `openAsync()`, é executado em segundo plano em vez de no encadeamento da execução do tempo de execução principal. Além disso, cada conexão SQL executa seu próprio encadeamento em segundo plano. Ao usar múltiplos objetos SQLConnection, você pode efetivamente executar múltiplas instruções SQL simultaneamente.

Existem também potenciais desvantagens nessa abordagem. Mais significativamente, cada objeto SQLStatement adicional requer memória adicional. Além disso, execuções simultâneas também levam o processador a trabalhar mais, principalmente, em máquinas que têm somente uma CPU ou núcleo da CPU. Devido a essas questões, não se recomenda essa abordagem para utilização em dispositivos portáteis.

Uma questão adicional é que o potencial benefício de reutilizar objetos SQLStatement pode se perder porque um objeto SQLStatement está ligado a um objeto SQLConnection simples. Consequentemente, não é possível reutilizar o objeto SQLStatement, se seu objeto SQLConnection associado já está em uso.

Se escolher utilizar múltiplos objetos SQLConnection conectados a um banco de dados simples, tenha em mente que cada um executa suas instruções em sua própria transação. Certifique-se de considerar estas transações separadas em qualquer código que altere dados, tal como adicionar, modificar ou excluir dados.

Paul Robertson criou uma biblioteca de código-fonte aberta que ajuda você a incorporar os benefícios de utilizar múltiplos SQLConnection enquanto diminui potenciais desvantagens. A biblioteca utiliza um pool de objetos SQLConnection e gerencia os objetos SQLStatement associados. Dessa maneira, assegura que os objetos SQLStatement sejam reutilizados e múltiplos objetos SQLConnection estejam disponíveis para executar múltiplas declarações simultaneamente. Para mais informações e baixar a biblioteca, visite http://probertson.com/projects/air-sqlite/.

Otimização do arquivo de banco de dados

Evite alterações em esquemas em banco de dados.

Se possível, evite alterar o esquema (estrutura de tabelas) de um banco de dados depois de adicionar dados às tabelas. Normalmente, um arquivo de banco de dados é estruturado com as definições de tabela no início do arquivo. Quando você abre uma conexão com um banco de dados, o tempo de execução carrega essas definições. Quando você adiciona dados a tabelas de um banco de dados, eles são adicionados ao arquivo após os dados de definição de tabela. Contudo, se você fizer alterações nos esquemas, os dados de definição da nova tabela serão mesclados com os dados da tabela contida no arquivo do banco de dados. Por exemplo, adicionar uma coluna a uma tabela ou adicionar uma nova tabela pode resultar na mesclagem dos tipos de dados. Se os dados de definição da tabela não estiverem todos localizados no início do arquivo do banco de dados, levará mais tempo para abrir uma conexão com o banco de dados. A conexão é mais lenta para a abertura porque o tempo de execução demora mais para ler os dados de definição da tabela em diferentes partes do arquivo.

Utilize o método `SQLConnection.compact()` para otimizar um banco de dados após as alterações nos esquemas.

Caso você não precise fazer alterações no esquema, poderá chamar o método `SQLConnection.compact()` depois de concluir as alterações. Esta operação reestrutura o arquivo de banco de dados para que os dados de definição de tabela fiquem localizados juntos no início do arquivo. No entanto, a operação `compact()` pode ser demorada, principalmente quando um arquivo de banco de dados aumenta de tamanho.

Processamento desnecessário de tempo de execução de banco de dados

Utilize um nome de tabela totalmente qualificado (incluindo nome do banco de dados) na instrução SQL.

Sempre especifique explicitamente o nome do banco de dados junto com cada nome de tabela em uma instrução. (Use "main" se for o banco de dados principal). Por exemplo, o seguinte código inclui um nome de banco de dados explícito `main`:

```
SELECT employeeId
FROM main.employees
```

Quando você especifica o nome do banco de dados de maneira explícita, evita que o tempo de execução verifique cada banco de dados conectado para encontrar a tabela correspondente. Isso também evita a possibilidade de o tempo de execução escolher o banco de dados errado. Siga esta regra mesmo que um SQLConnection esteja conectado a apenas um banco de dados. Em segundo plano o SQLConnection também está conectado a um banco de dados temporário que pode ser acessado através de instruções SQL.

Utilize nomes de colunas explícitos nas instruções `INSERT` e `SELECT`.

Os exemplos a seguir mostram a utilização de nomes de colunas explícitos:

```
INSERT INTO main.employees (firstName, lastName, salary)
VALUES ("Bob", "Jones", 2000)

SELECT employeeId, lastName, firstName, salary
FROM main.employees
```

Compare os exemplos anteriores com os exemplos a seguir. Evite este estilo de código:

```
-- bad because column names aren't specified
INSERT INTO main.employees
VALUES ("Bob", "Jones", 2000)

-- bad because it uses a wildcard
SELECT *
FROM main.employees
```

Sem nomes de colunas explícitos, o tempo de execução deve fazer trabalho extra para descobrir os nomes das colunas. Se uma instrução `SELECT` usar um caractere curinga em vez de colunas explícitas, isto fará com que o tempo de execução recupere os dados extras. Esses dados extras requerem processamento extra e cria ocorrências de objetos extras que não são necessárias.

Evite juntar a mesma tabela várias vezes em uma instrução, a menos que compare a própria tabela.

Conforme as instruções SQL aumentam de tamanho, você pode vincular despercebidamente uma tabela a uma consulta várias vezes. Frequentemente, o mesmo resultado poderia ser obtido usando a tabela somente uma vez. Vincular a mesma tabela várias vezes é provável quando você usa uma ou mais exibições em uma consulta. Por exemplo, você poderia vincular uma tabela à consulta e, também, uma exibição que incluísse os dados daquela tabela. As duas operações resultariam em mais de um vínculo.

Sintaxe SQL eficiente

Utilize `JOIN` (na cláusula `FROM`) para incluir uma tabela em uma consulta em vez de uma subconsulta na cláusula `WHERE`. Essa dica aplica-se mesmo se você precisar apenas de dados da tabela para filtrar; não para o conjunto de resultados.

Colocar múltiplas tabelas em uma cláusula `FROM` desempenha melhor que utilizar uma subconsulta em uma cláusula `WHERE`.

Evite instruções SQL que não possam tirar proveito de índices. Essas instruções incluem o uso de funções agregadas em uma subconsulta, uma instrução `UNION`em uma subconsulta ou uma cláusula `ORDER BY` com uma instrução `UNION`.

Um índice pode aumentar muito a velocidade do processamento da consulta `SELECT`. No entanto, a sintaxe SQL certa impede que o banco de dados utilize índices, forçando-o a utilizar os dados reais para buscar ou ordenar operações.

Considere evitar o operador `LIKE`, especialmente com um caractere curinga inicial como em `LIKE('%XXXX%')`.

Uma vez que a operação `LIKE` suporta o uso de busca com caractere curinga, seu desempenho é inferior ao de comparações que usam correspondências exatas. Em particular, se você iniciar a sequência de caracteres de busca com um caractere curinga, o banco de dados não poderá usar os índices, de nenhuma maneira, na busca. Em vez disso, o banco de dados deverá buscar o texto completo de cada linha na tabela.

Considere evitar o operador `IN`. Se os valores possíveis forem conhecidos de antemão, a operação `IN` poderá ser escrita usando `AND` ou `OR` para uma execução mais rápida.

A segunda das duas instruções seguintes é executada mais rapidamente. Isto é mais rápido porque usa expressões de igualdade simples combinadas com `OR`, em vez de instruções `IN()` ou `NOT IN()`:

```
-- Slower
SELECT lastName, firstName, salary
FROM main.employees
WHERE salary IN (2000, 2500)

-- Faster
SELECT lastName, firstName, salary
FROM main.employees
WHERE salary = 2000
    OR salary = 2500
```

Considere formas alternativas de uma instrução SQL para melhorar o desempenho.

Como foi demonstrado nos exemplos anteriores, a maneira em que uma instrução SQL é escrita também pode afetar o desempenho do banco de dados. Com frequência, existem diversas maneiras de criar uma instrução SQL `SELECT` para recuperar um conjunto de resultados específico. Em alguns casos, uma abordagem funciona com rapidez notoriamente maior do que a outra. Além das sugestões anteriores, você pode saber mais sobre diferentes instruções SQL e seu desempenho consultando os recursos dedicados na linguagem SQL.

Desempenho de instrução SQL

Compara diretamente instruções SQL alternadas para determinar qual é a mais rápida.

A melhor maneira de comparar o desempenho de várias versões de uma instrução SQL é testá-las diretamente com seus dados e banco de dados.

As seguintes ferramentas de desenvolvimento fornecem tempos de execução enquanto executam instruções SQL. Use-as para comparar a velocidade das diferentes versões de instruções:

- Run! (AIR SQL query authoring and testing tool, por Paul Robertson)
- Lita (SQLite Administration Tool, por David Deraedt)

Capítulo 9: Benchmark e implantação

Benchmark

Existem várias ferramentas para avaliar aplicativos. Você pode utilizar a classe Stats e a classe PerformanceTest, desenvolvidas por membros da comunidade Flash. Você pode também utilizar o gerador de perfil no Adobe® Flash® Builder™, e a ferramenta FlexPMD.

A classe Stats

Para gerar o perfil do seu código no tempo de execução usando a versão de lançamento do tempo de execução, sem uma ferramenta externa, você pode usar a classe Stats desenvolvida por mr. doob da comunidade Flash. Você popde baixar a classe Stats do seguinte endereço: https://github.com/mrdoob/Hi-ReS-Stats.

A classe Stats permite que você registre os seguintes itens:

- Quadros renderizados por segundo (quanto maior o número, melhor).
- Milisegundos utilizados para renderizar um quadro (quanto menos o número, melhor).
- A quantidade de memória utilizada pelo código. Caso aumente a cada quadro, é possível que sua aplicação tenha um vazamento de memória. É importante investigar o possível vazamento de memória.
- A quantidade máxima de memória utilizada pelo aplicativo.

Uma vez baixada, a classe Stats pode ser utilizada com o seguinte código compacto:

```
import net.hires.debug.*;
addChild( new Stats() );
```

Ao usar compilação condicional no Adobe® Flash® Professional ou Flash Builder, você pode habilitar o objeto Stats:

```
CONFIG::DEBUG
{
    import net.hires.debug.*;
    addChild( new Stats() );
}
```

Ao alterar o valor da constante `DEBUG`, você pode habilitar ou desabilitar a compilação do objeto Stats. A mesma aproximação pode ser utilizada para substituir qualquer lógica de código que você não queira que seja compilada em seu aplicativo.

A classe PerformanceTeste

Para avaliar a execução de código ActionScript, Grant Skinner desenvolveu uma ferramenta que pode ser integrada em um fluxo de trabalho de unidade de teste. Você envia uma classe personalizada para a classe PerformanceTest, a qual fará uma série de testes em seu código. A classe PerformanceTest permite que você avalie aproximações diferentes facilmente. O download da classe PerformanceTest pode ser feito em:
http://www.gskinner.com/blog/archives/2009/04/as3_performance.html.

Gerador de perfil do Flash Builder

O Flash Builder possui um gerador de perfil que você avalie seu código com um grande nível de detalhe.

Nota: *Utilize a versão de depuração do Flash Player para acessar o gerador de perfil. ou você receberá uma mensagem de erro.*

O gerador de perfil também pode ser usado com conteúdo produzido no Adobe Flash Professional. Para fazer isso, carregue o arquivo SWF compilado de um projeto ActionScript ou Flex no Flash Builder e você poderá executar o gerador de perfil nele. Para obter mais informações sobre o gerador de perfil, consulte "Criando perfis de aplicativos Flex" em Uso do Flash Builder 4.

FlexPMD

Os serviços técnicos da Adobe lançaram uma ferramenta chamada FlexPMD. que permite que você audite a qualidade do código do ActionScript 3.0. FlexPMD é uma ferramenta de ActionScript similar ao JavaPMD. O FlexPMD melhora a qualidade do código auditando um diretório fonte de ActionScript 3.0 ou Flex. Ele detecta práticas de codificação ruins, como código não utilizado, código excessivamente complexo, excessivamente longo e o uso incorreto do ciclo de vida do componente Flex.

O FlexPMD é um projeto de código aberto da Adobe disponível no endereço: http://opensource.adobe.com/wiki/display/flexpmd/FlexPMD. Um plug-in Eclipse também está disponível no seguinte endereço: http://opensource.adobe.com/wiki/display/flexpmd/FlexPMD+Eclipse+plugin.

O FlexPMD torna fácil auditar código e garantir que seu código está limpo e otimizado. O verdadeiro poder do FlexPMD está em sua extensibilidade. Como desenvolvedor, você pode criar seus próprios conjuntos de regras para auditar qualquer código. Por exemplo, você pode criar um conjunto de regras que detecte o uso pesado de filtros, ou qualquer outra prática ruim de codificação que você quiser pegar.

Implantação

Quando você estiver exportando a versão final de seu aplicativo no Flash Builder. é importante ter certeza d que você está exportando a versão de lançamento. Ao exportar uma versão de lançamento as informações de depuração contidas no arquivo SWF são removidas. Remover as informações de depuração faz com que o tamanho do arquivo SWF fique menor e ajuda o aplicativo a executar mais rápido.

Para exportar a versão de lançamento de seu projeto, utilize o painel de Projeto no Flash Builder e a opção de Exportar build de lançamento.

Nota: *Quando estiver compilando seu projeto no Flash Professional, você não terá a opção de escolher entre versão de lançamento ou de depuração. O arquivo SWF compilado é a versão de lançamento por padrão.*