Optimización del rendimiento para la PLATAFORMA ADOBE® FLASH®

Última modificación 1/5/2010

Optimización del rendimiento para la plataforma Adobe® Flash®

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

Contenido

Capítulo 1: Introducción

Con la publicación de Adobe® Flash® Player 10.1, los desarrolladores pueden utilizar el mismo Flash Player en los dispositivos móviles y en el escritorio. Con la ayuda de ejemplos de código y casos de uso, en este documento se describen las mejores prácticas para desarrolladores que implementan aplicaciones en dispositivos móviles. Se analizan temas importantes generales como, por ejemplo, el uso de memoria y CPU y posteriormente se destacan las optimizaciones de ActionScript específicas para dispositivos móviles. En este documento se analizan las mejores prácticas para las aplicaciones que se ejecutan dentro de un navegador móvil, pero estas prácticas también se aplican a las aplicaciones de Adobe® AIR®. Asimismo, se aplican a las aplicaciones creadas utilizando Packager for iPhone Preview® en Adobe® Flash® Professional CS5.

Para obtener información sobre el desarrollo de aplicaciones para iPhone, consulte Creación de aplicaciones para iPhone con ActionScript 3.0.

Fundamentos de la ejecución de código del motor de ejecución

Un aspecto fundamental para comprender cómo mejorar el rendimiento de la aplicación consiste en conocer el modo en que el motor de ejecución de la plataforma Flash ejecuta el código. El motor de ejecución funciona en un bucle con determinadas acciones que se producen en cada "fotograma". En este caso, un fotograma es simplemente un bloque de tiempo determinado por la velocidad de fotogramas especificada para la aplicación. La cantidad de tiempo asignada a cada fotograma se corresponde directamente con la velocidad de fotogramas. Por ejemplo, si se especifica una velocidad de fotogramas de 30 fotogramas por segundo, el motor de ejecución intenta que cada fotograma dure una trigésima parte de un segundo.

La velocidad de fotogramas inicial de la aplicación se especifica en tiempo de edición. La velocidad de fotogramas se puede establecer utilizando la configuración de Adobe® Flash® Builder™ o Flash Professional. También se puede especificar la velocidad de fotogramas inicial en el código. Establezca la velocidad de fotogramas en una aplicación de sólo ActionScript, aplicando la etiqueta de metadatos `[SWF(frameRate="24")]` a la clase del documento raíz. En MXML, establezca el atributo `frameRate` en la etiqueta Application o WindowedApplication.

Cada bucle de fotograma consta de dos fases, divididas en tres partes: eventos, el evento `enterFrame` y la representación.

La primera fase incluye dos partes (eventos y el evento `enterFrame`) y en ambas se llamará potencialmente a su código. En la primera parte de la primera fase, los eventos del motor de ejecución llegan y se distribuyen. Estos eventos puede representar la finalización o el progreso de operaciones asíncronas como, por ejemplo, una respuesta de la carga de datos en una red. También incluyen eventos de las entradas de usuario. Conforme se distribuyen los eventos, el motor de ejecución ejecuta el código en los detectores registrados. Si no se producen eventos, el motor de ejecución espera a que se complete esta fase de ejecución sin realizar ninguna operación. El motor de ejecución nunca acelera la velocidad de fotogramas debido a la falta de actividad. Si los eventos se producen durante otras partes del ciclo de ejecución, el motor de ejecución pone en cola estos eventos y los distribuye en el siguiente fotograma.

La segunda parte de la primera fase es el evento `enterFrame`. Este evento es distinto a los demás, ya que siempre se distribuye una vez por fotograma.

Una vez distribuidos todos los eventos, comienza la fase de representación del bucle del fotograma. En ese momento, el motor de ejecución calcula el estado de todos los elementos visibles en pantalla y los dibuja en la pantalla. Posteriormente, el proceso se vuelve a repetir, de forma similar a un corredor que da vueltas a una pista.

Nota: *para los eventos que incluyen una propiedad* `updateAfterEvent`*, es posible forzar la representación para que se produzca de forma inmediata en lugar de esperar a la fase de representación. No obstante, evite el uso de* `updateAfterEvent` *si esto suele implicar problemas de rendimiento.*

Resulta más fácil imaginar que las dos fases del bucle de fotograma adoptan cantidades similares de tiempo. En ese caso, durante la mitad de cada bucle de fotograma, los controladores de eventos y el código de la aplicación se están ejecutando y durante la otra mitad se está llevando a cabo la representación. Sin embargo, la realidad suele ser diferente. En ocasiones, el código de la aplicación tarda más de la mitad del tiempo disponible en el fotograma, ampliando su asignación de tiempo y reduciendo la asignación disponible para la representación. En otros casos, especialmente con el contenido visual completo como, por ejemplo, modos de fusión y filtros, la representación requiere más de la mitad del tiempo del fotograma. El tiempo real empleado en las fases es flexible, por lo que el bucle de fotograma se suele denominar “pista elástica”.

Si las operaciones combinadas del bucle de fotograma (representación y ejecución de código) tardan demasiado, el motor de ejecución no podrá mantener la velocidad de fotogramas. El fotograma se expande, tardando más tiempo que su tiempo asignado, por lo que se activa un retraso antes del siguiente fotograma. Por ejemplo, un bucle de fotograma tarda más de una trigésima parte de un segundo, el motor de ejecución no podrá actualizar la pantalla a 30 fotogramas por segundo. Si la velocidad de fotogramas se ralentiza, la experiencia se degrada. Una mejor animación se vuelve irregular. En el peor caso, la aplicación se bloquea y la ventana se queda en blanco.

Para obtener más información sobre la ejecución del código del motor de ejecución de la plataforma Flash y el modelo de representación, consulte los siguientes recursos:

- Flash Player Mental Model - The Elastic Racetrack (Modelo mental de Flash Player - La pista elástica). (Artículo de Ted Patrick; en inglés).
- Asynchronous ActionScript Execution (Ejecución de ActionScript asíncrona). (Artículo de Trevor McCauley; en inglés).
- Optimizing Adobe AIR for code execution, memory & rendering (Optimización de Adobe AIR para la ejecución de código, memoria y representación) http://www.adobe.com/go/learn_fp_air_perf_tv_es (Ví-deo de la presentación de la conferencia MAX de Sean Christmann; en inglés.)

Rendimiento percibido frente a rendimiento real

Los que valorarán en última instancia si la aplicación funciona correctamente son los usuarios de la misma. Los desarrolladores pueden medir el rendimiento de la aplicación en términos de cuánto tiempo tardan en ejecutarse determinadas operaciones o cómo se crean varias instancias de objetos. Sin embargo, esta métrica no es importante para los usuarios finales. En ocasiones los usuarios miden el rendimiento mediante diferentes criterios. Por ejemplo, ¿funciona la aplicación rápidamente y de forma fluida y responde con rapidez? ¿Tiene un efecto negativo en el rendimiento del sistema? Hágase estas preguntas que constituyen pruebas del rendimiento percibido:

- ¿Presentan las animaciones alguna irregularidad?
- ¿Parece fluido el contenido de vídeo?
- ¿Se reproducen los clips de audio de forma continua o por el contrario se detienen y vuelve a reanudarse su reproducción?

- ¿Parpadea la ventana o se queda en blanco durante operaciones largas?
- Mientras escribo, ¿es fluida la entrada de texto o queda rezagada?
- Si hago clic, ¿ocurre algo inmediatamente o hay algún retraso?
- ¿Hace más ruido el ventilador de la CPU cuando se ejecuta la aplicación?
- En un equipo portátil o dispositivo móvil, ¿se agota la batería rápidamente mientras se ejecuta la aplicación?
- ¿Responden otras aplicaciones con dificultad durante la ejecución de la aplicación?

La distinción entre el rendimiento percibido y el real es importante, ya que la forma de obtener el mejor rendimiento percibido no es la misma que para lograr el rendimiento pleno más rápido. Asegúrese de que la aplicación nunca ejecute demasiado código de modo que el motor de ejecución no sea capaz de actualizar con frecuencia la pantalla ni captar la entrada del usuario. En algunos casos, lograr este equilibrio implica la división de una tarea del programa en partes de forma que, entre partes, el motor de ejecución actualiza la pantalla. (Para obtener instrucciones específicas, consulte "Rendimiento de la representación" en la página 47.)

Las sugerencias y técnicas descritas aquí representan mejoras de destino tanto en el rendimiento de ejecución del código real como en el modo en que los usuarios perciben el rendimiento.

Objetivo de las optimizaciones

Algunas mejoras de rendimiento no suponen mejoras perceptibles para los usuarios. Es importante concentrar las optimizaciones de rendimiento en las áreas que supongan problemas para la aplicación específica. Algunas optimizaciones del rendimiento constituyen buenas prácticas generales y siempre se puede realizar un seguimiento de las mismas. Para otras optimizaciones, el que sean útiles o no depende de las necesidades de la aplicación de su base de usuarios anticipada. Por ejemplo, las aplicaciones siempre funcionan mejor si no se utiliza ninguna animación, vídeo o efectos y filtros gráficos. No obstante, uno de los motivos para el uso de la plataforma Flash para crear aplicaciones se debe a sus capacidades gráficas y de medios que permiten aplicaciones de gran expresividad. Considere si el nivel deseado de complejidad se corresponde bien con las características de rendimiento de los equipos y dispositivos en los que se ejecuta la aplicación.

Una advertencia común es "evitar la optimización demasiado pronto". Algunas optimizaciones de rendimiento requieren la escritura de código de un modo que resulta más difícil de leer o menos flexible. Una vez optimizado, el mantenimiento de este código es más complejo. Para esos tipos de optimizaciones de rendimiento, en ocasiones es mejor esperar y determinar si una sección concreta de código funciona incorrectamente antes de optar por optimizarlo.

En ocasiones, la mejora del rendimiento implica algunas desventajas menores. Idealmente, al reducir la cantidad de memoria consumida por una aplicación también aumenta la velocidad con la que la aplicación realiza una tarea. Sin embargo, el tipo idóneo de mejora no siempre es posible. Por ejemplo, si una aplicación se bloquea durante una operación, la solución implica a veces la división del trabajo para que se ejecute en varios fotogramas. Debido a la división del trabajo, es probable que se tarde más en general en llevar a cabo el proceso. Sin embargo, es posible que el usuario no note el tiempo adicional, si la aplicación continúa respondiendo y no se bloquea.

Un aspecto fundamental para saber qué optimizar y si las optimizaciones resultan útiles es la realización de pruebas de rendimiento. En "Prueba comparativa e implementación" en la página 88 se describen varias técnicas y sugerencias para probar el rendimiento.

Para obtener más información sobre la determinación de partes de una aplicación que son buenas candidatas para la optimización, consulte los siguientes recursos:

- Performance-tuning apps for AIR, en http://www.adobe.com/go/learn_fp_goldman_tv_es (Ví-deo de la presentación de la conferencia MAX de Oliver Goldman; en inglés)
- Performance-tuning Adobe AIR applications, en http://www.adobe.com/go/learn_fp_air_perf_devnet_es (Artículo del centro de desarrollo de Adobe de Oliver Goldman, basado en la presentación)

Capítulo 2: Conservación de memoria

La conservación de memoria siempre es importante en el desarrollo de las aplicaciones, incluso para las aplicaciones de escritorio. Sin embargo, los dispositivos móviles implican una sobrecarga en el consumo de memoria y merece la pena limitar la cantidad de memoria que consume la aplicación.

Objetos de visualización


Seleccione un objeto de visualización adecuado.

ActionScript 3.0 incluye un amplio conjunto de objetos de visualización. Una de las sugerencias de optimización más sencillas para limitar el uso de memoria consiste en utilizar el tipo adecuado de objeto de visualización. Para las formas simples que no son interactivas, utilice objetos Shape. Para los objetos interactivos que no necesiten una línea de tiempo, utilice objetos Sprite. Para la animación que utilice una línea de tiempo, use objetos MovieClip. Elija siempre el tipo de objeto más eficaz para su aplicación.

El siguiente código muestra el uso de memoria para diferentes objetos de visualización:

```
trace(getSize(new Shape()));
// output: 236
trace(getSize(new Sprite()));
// output: 412
trace(getSize(new MovieClip()));
// output: 440
```

El método `getSize()` muestra la cantidad de bytes que consume en memoria un objeto. Se puede observar que el uso de varios objetos MovieClip en lugar de simples objetos Shape, puede suponer un uso innecesario de memoria si las capacidades de un objeto MovieClip no resultan necesarias.

Tipos simples o primitivos


Utilice el método `getSize()` para realizar una prueba comparativa del código y determinar el objeto más eficaz para la tarea.

Todos los tipos simples excepto String utilizan de 4 a 8 bytes de memoria. No existe ningún modo de optimizar la memoria utilizando un tipo específico para un tipo simple:

```
// 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
```

Al tipo Number, que representa un valor de 64 bits, se le asignan 8 bytes mediante la máquina virtual ActionScript (AVM), si no se le asigna ningún valor. Todos los demás tipos simples se almacenan en 4 bytes.

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

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

El comportamiento difiere para el tipo String. La cantidad de almacenamiento asignada se basa en la longitud de la cadena:

```
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
```

Utilice el método `getSize()` para realizar una prueba comparativa del código y determinar el objeto más eficaz para la tarea.

Reutilización de objetos


Si es posible, reutilice los objetos en lugar de volver a crearlos.

Otra forma sencilla de optimizar la memoria consiste en reutilizar objetos y evitar su recreación siempre que sea posible. Por ejemplo, en un bucle, no utilice el siguiente 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);
}
```

Al volver a crear el objeto Rectangle en cada iteración del bucle se utiliza más memoria y el proceso es más lento, ya que se crea un nuevo objeto en cada iteración. Utilice el siguiente enfoque:

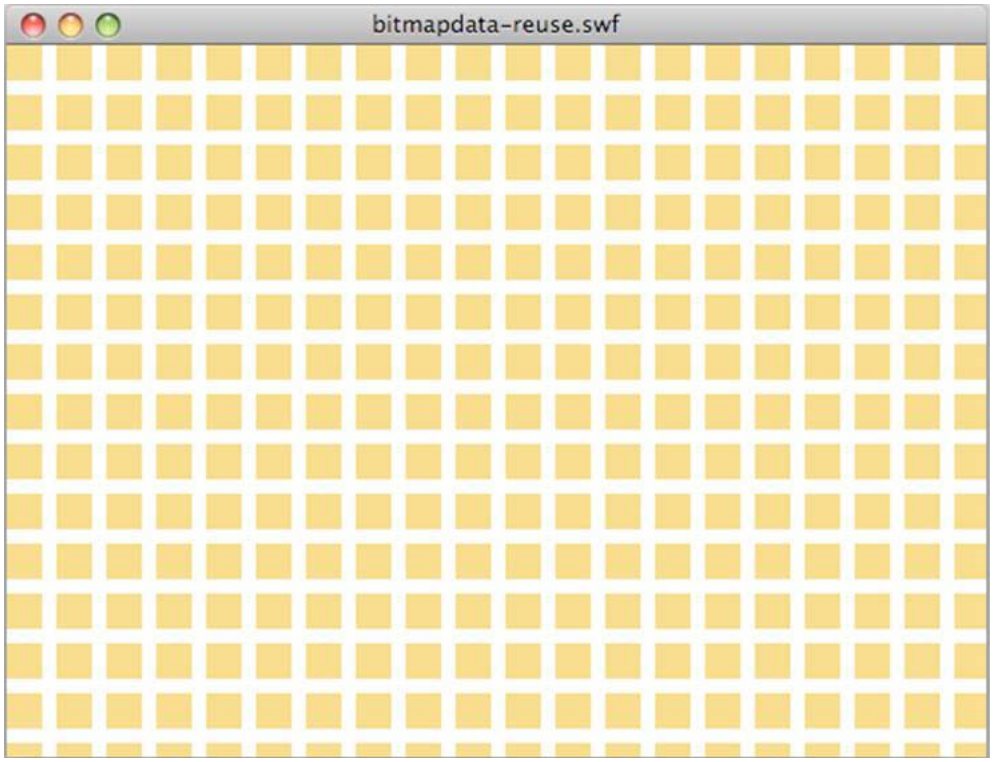
```
const MAX_NUM:int = 18;
const COLOR:uint = 0xCCCCCC;

// 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);
}
```

El ejemplo anterior utilizaba un objeto con un impacto de memoria relativamente pequeño. En el siguiente ejemplo se muestra un mayor ahorro de memoria mediante la reutilización de un objeto BitmapData. El siguiente código utilizado para crear un efecto mosaico supone un desaprovechamiento de memoria:

```
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: *cuando se utilizan valores positivos, la conversión del valor redondeado a int resulta mucho más rápido que el uso del método* `Math.floor()`.

La siguiente imagen muestra el resultado del mosaico del mapa de bits:



Resultado del mosaico de mapa de bits

Una versión optimizada crea un sola instancia de BitmapData a la que se hace referencia mediante varias instancias de Bitmap y se produce el mismo 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);
}
```

Con este enfoque se ahorran unos 700 KB de memoria, lo cual representa un ahorro significativo en un dispositivo móvil tradicional. Todos los contenedores de mapas de bits se pueden manipular sin modificar la instancia de BitmapData original y utilizando las propiedades 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();
}
```

La siguiente imagen muestra el resultado de las transformaciones del mapa de bits:

Resultado de las transformaciones del mapa de bits

Más temas de ayuda

"Almacenamiento en caché de mapas de bits" en la página 53

Agrupación de objetos

Utilice la agrupación de objetos siempre que sea posible.

Otra optimización importante se denomina agrupación de objetos e implica la reutilización de los mismos en el tiempo. Durante la inicialización de la aplicación se crea un número definido de objetos que se almacenan dentro de un grupo como, por ejemplo, un objeto Array o Vector. Una vez que se ha terminado con un objeto, se puede desactivar para que no consuma recursos de CPU y se eliminan todas las referencias a otros objetos. Sin embargo, no se establecen las referencias a `null`, lo que podría hacerlo adecuado para la recolección de elementos no utilizados. Sólo es necesario volver a situar el objeto en el grupo y recuperarlo cuando se necesite un nuevo objeto.

Con la reutilización de objetos se reduce la necesidad de crear instancias de objetos, lo cual puede resultar costoso. Asimismo, se reducen las posibilidades de ejecución del recolector de elementos no utilizados, que puede ralentizar la aplicación. El siguiente código muestra la técnica de agrupación 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;
        }
    }
}
```

La clase SpritePool crea un grupo de objetos nuevos cuando se inicia la aplicación. El método `getSprite()` devuelve instancias de estos objetos y `disposeSprite()` las libera. El código permite que el grupo crezca cuando se ha consumido por completo. También es posible crear un grupo de tamaño fijo donde los nuevos objetos no se asignen cuando el grupo esté agotado. Intente evitar la creación de nuevos objetos en bucles, si es posible. Para obtener más información, consulte "Liberación de memoria" en la página 11. En el siguiente código se utiliza la clase SpritePool para recuperar nuevas instancias:

```
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 );
    }
}
```

El siguiente código elimina todos los objetos de visualización de la lista de visualización cuando se hace clic en el ratón y, posteriormente, se reutilizan para otra tarea:

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

Nota: *el vector del grupo siempre hace referencia a objetos Sprite. Si desea eliminar el objeto de la memoria completamente, sería necesario disponer de un método `dispose()` en la clase SpritePool, lo que podría eliminar todas las referencias restantes.*

Liberación de memoria

Elimine todas las referencias a los objetos para garantizar que se active la recolección de elementos no utilizados.

No se puede iniciar el recolector de elementos no utilizados directamente en la versión oficial de Flash Player. Para garantizar que un objeto se recolecta como elemento no utilizado, elimine todas las referencias al objeto. Se debe tener en cuenta que el anterior operador `delete` utilizado en ActionScript 1.0 y 2.0 se comporta de forma distinta en ActionScript 3.0. Únicamente puede utilizarse para eliminar las propiedades dinámicas en un objeto dinámico.

Nota: *es posible llamar al recolector de objetos directamente en Adobe® AIR® y en la versión de depuración de Flash Player.*

Por ejemplo, el siguiente código establece una referencia a Sprite en `null`:

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

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

Recuerde que cuando un objeto se define como `null`, no se elimina necesariamente de la memoria. En ocasiones, el recolector de elementos no utilizados no se ejecuta, si la memoria disponible no se considera lo suficientemente baja. El recolector de elementos no utilizados no es previsible. La asignación de memoria, en lugar de la eliminación del objeto, activa la recolección de elementos no utilizados. Cuando se ejecuta el recolector de elementos no utilizados, busca gráficos de objetos que aún no se han recopilado. Detecta objetos inactivos en los gráficos buscando objetos que se hacen referencia entre sí, pero que la aplicación ya no utiliza. Los objetos inactivos detectados de este modo se eliminan.

En las aplicaciones de gran tamaño, este proceso puede implicar una carga para los recursos de CPU y puede afectar al rendimiento, así como implicar una ralentización evidente en la aplicación. Intente limitar el uso de la recolección de elementos no utilizados, reutilizando objetos siempre que sea posible. Asimismo, establezca referencias a null, si es posible, de modo que el recolector de elementos no utilizados emplee menos tiempo de procesamiento en la búsqueda de objetos. Considere el recolector de elementos no utilizados como recurso adicional e intente administrar la duración de los objetos de forma explícita.

El recolector de elementos no utilizados se puede iniciar utilizando el método `System.gc()`, disponible en Adobe AIR y en la versión de depuración de Flash Player. El generador de perfiles incluido con Adobe® Flash® Builder permite iniciar el recolector de elementos no utilizados de forma manual. La ejecución del recolector de elementos no utilizados permite ver el modo en que responde la aplicación y si los objetos se eliminan correctamente de la memoria.

Nota: *si un objeto se ha utilizado como detector de eventos, otro objeto puede hacer referencia al mismo. Si es así, elimine los detectores de eventos utilizando el método `removeEventListener()` antes de establecer las referencias a `null`.*

Afortunadamente, la cantidad de memoria utilizada por los mapas de bits puede reducirse al instante. Por ejemplo, la clase BitmapData incluye un método `dispose()`. En el ejemplo siguiente se crea una instancia de BitmapData de 1,8 MB. La memoria actual en uso crece 1,8 MB y la propiedad `System.freeMemory` devuelve un pequeño valor:

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

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

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

A continuación, BitmapData se borra manualmente (se elimina) de la memoria y el uso de memoria se vuelve a comprobar una vez más:

```
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
```

Aunque el método `dispose()` elimina los píxeles de la memoria, la referencia aún se debe establecer en `null` para liberarla por completo. Llame siempre al método `dispose()` y establezca la referencia en `null` cuando ya no necesite el objeto BitmapData; de este modo, la memoria se libera de inmediato.

Nota: *Flash Player 1.5.2 y AIR 1.5.2 introducen un nuevo método denominado `disposeXML()` en la clase System. Este método permite hacer que un objeto XML esté disponible de forma inmediata para la recolección de elementos no utilizados, transmitiendo el árbol XML como parámetro.*

Utilización de mapas de bits

El uso de mapas de bits siempre ha sido un tema importante para los desarrolladores de contenido móvil. El uso de vectores en lugar de mapas de bits constituye una buena forma de ahorrar memoria. No obstante, el uso de vectores, especialmente en grandes cantidades, aumenta en gran medida los recursos de CPU o GPU. El uso de mapas de bits constituye una buena forma de optimizar la representación, ya que Flash Player necesita menos recursos de procesamiento para dibujar píxeles en pantalla que para representar el contenido vectorial.

Más temas de ayuda

"Almacenamiento en caché manual de mapas de bits" en la página 59

Disminución de resolución de mapas de bits

Para hacer un mejor uso de la memoria, las imágenes opacas de 32 bits se reducen a imágenes de 16 bits cuando Flash Player detecta una pantalla de 16 bits. Esta disminución de resolución consume la mitad de recursos de memoria y las imágenes se representan con más rapidez. Esta función sólo está disponible en Flash Player 10.1 para Windows Móvil.

Nota: *antes de Flash Player 10.1, todos los píxeles creados en memoria se almacenaban en 32 bits (4 bytes). Un simple logotipo de 300 x 300 píxeles consumía 350 KB de memoria (300*300*4/1024). Con este nuevo comportamiento, el mismo logotipo opaco sólo consume 175 KB. Si el logotipo es transparente, su resolución no disminuye a 16 bits y mantiene el mismo tamaño en memoria. Esta función sólo se aplica a los mapas de bits incorporados o a las imágenes cargadas en tiempo de ejecución (PNG, GIF, JPG).*

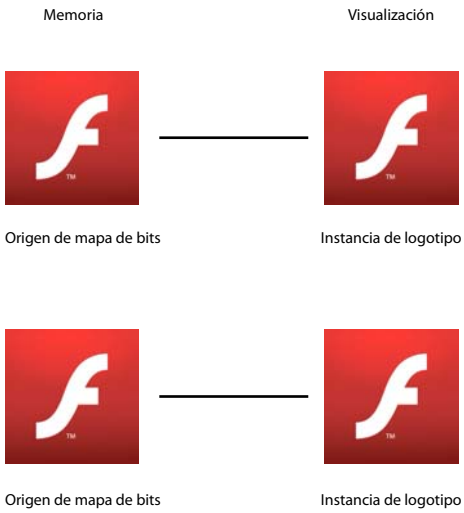
En los dispositivos móviles, es difícil notar la diferencia entre una imagen representada en 16 bits y la misma imagen procesada en 32 bits. En una imagen sencilla que sólo contiene unos cuantos colores, no existen diferencias apreciables. Incluso en una imagen más compleja, resulta difícil detectar las diferencias. Sin embargo, puede haber cierta degradación del color cuando nos acercamos a la imagen y un degradado de 16 bits puede parecer menos suave que la versión de 32 bits.

Referencia única de BitmapData

Es importante optimizar el uso de la clase BitmapData, reutilizando instancias siempre que sea posible. Flash Player 10.1 introduce una nueva función para todas las plataformas denominada referencia única de BitmapData. Al crear instancias de BitmapData a partir de una imagen incorporada, se utiliza una sola versión del mapa de bits para todas las instancias de BitmapData. Si un mapa de bits se modifica con posterioridad, se le asigna su propio mapa de bits en memoria. La imagen incorporada puede proceder de la biblioteca o ser una etiqueta [Embed].

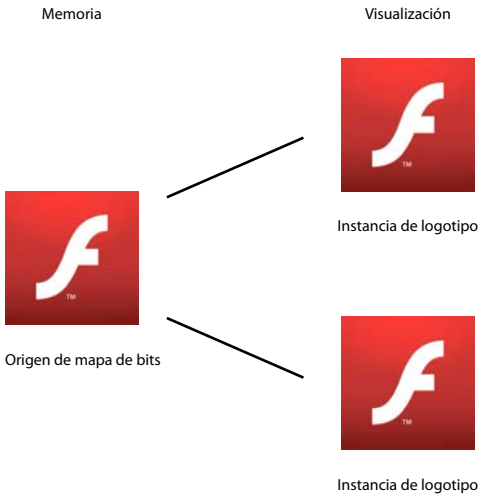
Nota: *el contenido existente también se beneficia de esta nueva función, ya que Flash Player 10.1 reutiliza automáticamente los mapas de bits.*

Cuando se crea una instancia de una imagen incorporada, un mapa de bits asociado se crea en memoria. Antes de Flash Player 10.1, a cada instancia se le asignaba un mapa de bits independiente en memoria, tal y como se muestra en el siguiente diagrama:



Mapas de bits en memoria antes de Flash Player 10.1

En Flash Player 10.1, cuando se crean varias instancias de la misma imagen, se utiliza una sola versión del mapa de bits para todas las instancias de BitmapData. El siguiente diagrama ilustra este concepto:

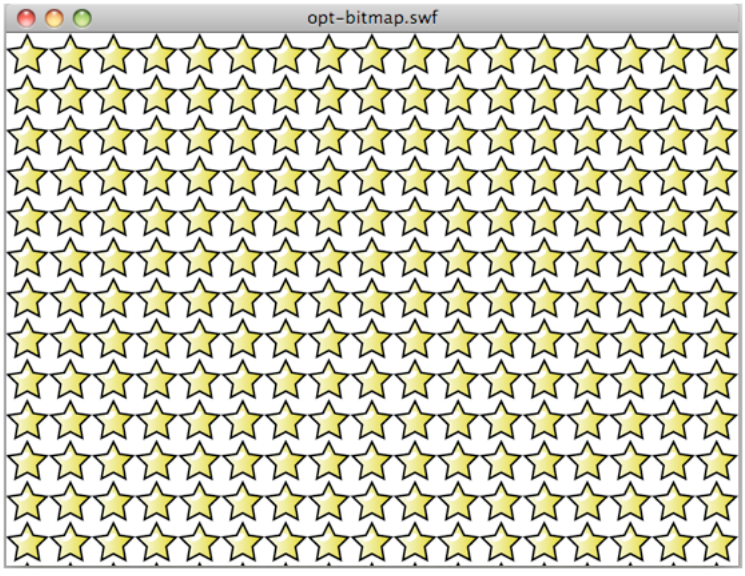


Mapas de bits en memoria en Flash Player 10.1

Con este enfoque se reduce en gran medida la cantidad de memoria que utiliza la aplicación con diversos mapas de bits. El siguiente código crea varias instancias de un 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)
    }
}
```

La siguiente imagen muestra el resultado del código:



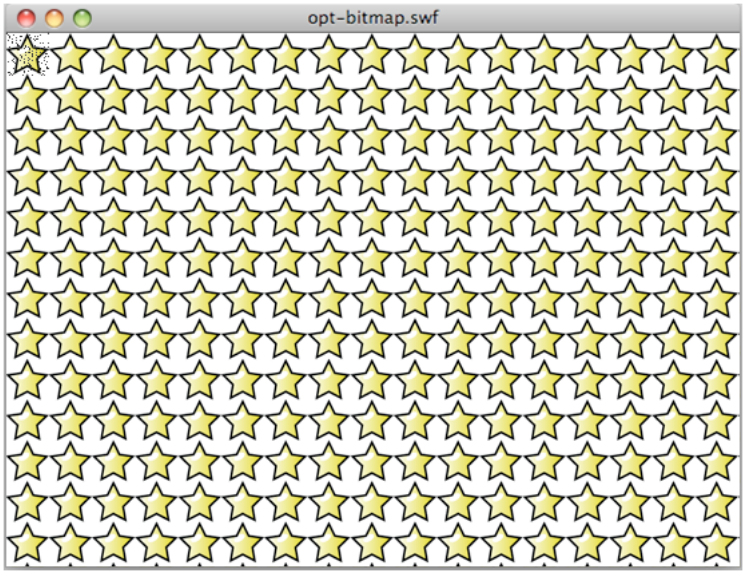
Resultado del código para crear varias instancias del símbolo

La animación anterior utiliza unos 1008 KB de memoria con Flash Player 10. En el escritorio y en un dispositivo móvil, la animación sólo usa 4 KB con Flash Player 10.1.

El siguiente código modifica una instancia de BitmapData:

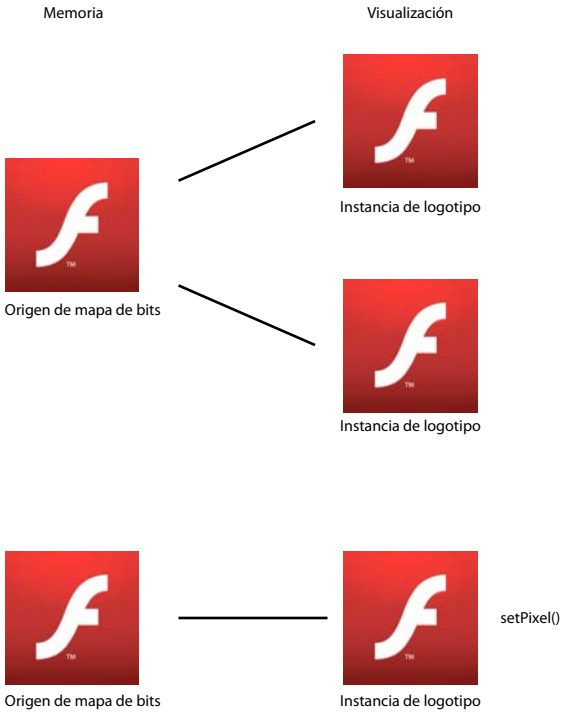
```
const MAX_NUM:int = 18;

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);
```

La imagen siguiente muestra el resultado de la modificación de una instancia de `Star`:



Resultado de la modificación de una instancia

Internamente, Flash Player 10.1 crea y asigna de forma automática un mapa de bits en memoria para administrar las modificaciones de píxeles. Cuando se llama a un método de la clase BitmapData, lo que implica modificaciones de píxel, se crea una nueva instancia en memoria y no se actualiza ninguna otra instancia. La siguiente figura ilustra el concepto:



Resultado en memoria de la modificación de un mapa de bits

Si se modifica una estrella, se crea una nueva copia en memoria; la animación resultante utiliza unos 8 KB en memoria en Flash Player 10.1.

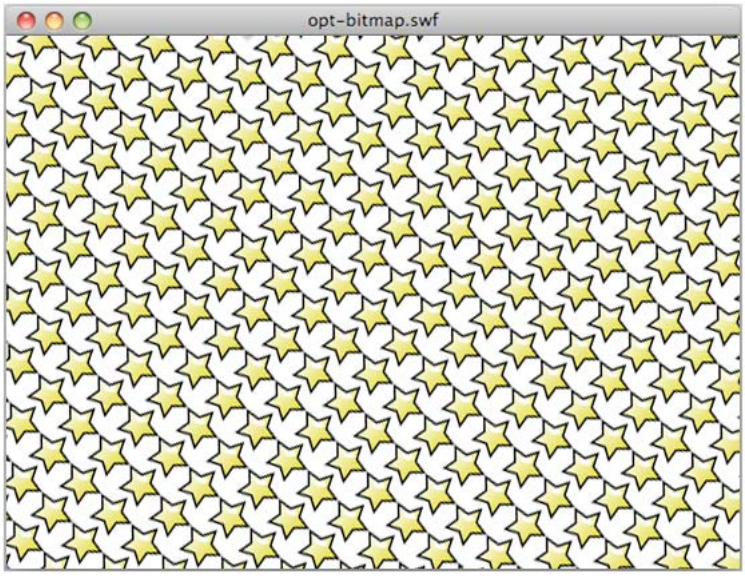
En el ejemplo anterior, cada mapa de bits está disponible individualmente para su transformación. Para crear únicamente el efecto mosaico, el método `beginBitmapFill()` es el más adecuado.

```
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);
```

Este enfoque produce el mismo resultado con sólo una única instancia creada de BitmapData. Para girar las estrellas de forma continua, en lugar de acceder a cada instancia de Star, utilice un objeto Matrix que se gire en cada fotograma. Transmita este objeto Matrix al 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);

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

Con el uso de esta técnica, no se requiere ningún bucle de ActionScript para crear el efecto. Flash Player realiza todas las operaciones internamente. La siguiente imagen muestra el resultado de la transformación de las estrellas:



Resultado del giro de las estrellas

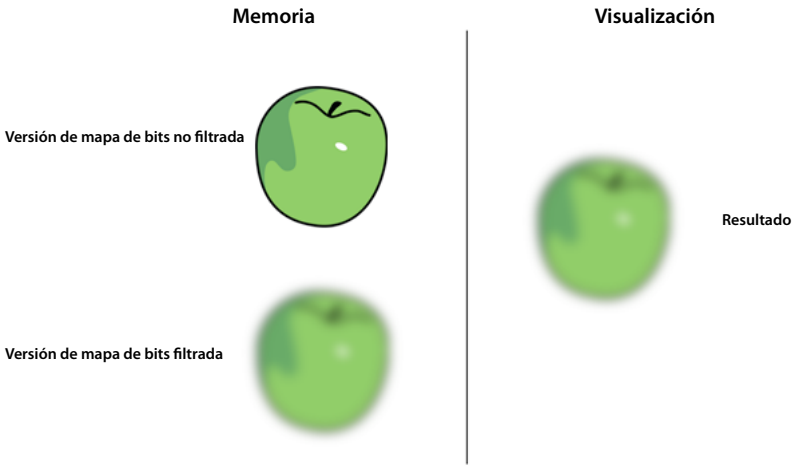
Con este enfoque, con la actualización del objeto de origen BitmapData original se actualiza automáticamente su uso en cualquier lugar del escenario, lo cual puede ser una técnica eficaz. Sin embargo, con este enfoque no se permitiría que se aplicara escala a cada estrella de forma individual, tal y como sucedía en el ejemplo anterior.

Nota: *al utilizar varias instancias de la misma imagen, el dibujo depende de si una clase se asocia al mapa de bits original en memoria. Si no existe ninguna clase asociada al mapa de bits, las imágenes se dibujan como objetos Shape con rellenos de mapa de bits.*

Filtros y descarga dinámica de mapas de bits

Evite los filtros, incluyendo los procesados mediante Pixel Bender.

Se recomienda evitar efectos como los filtros, incluyendo los procesados en dispositivos móviles mediante Pixel Bender. Cuando se aplica un filtro a un objeto de visualización, Flash Player crea dos mapas de bits en memoria. Estos mapas de bits son cada uno del tamaño del objeto de visualización. El primero se crea como una versión rasterizada del objeto de visualización que, en cambio, se utiliza para producir un segundo mapa de bits con el filtro aplicado:



Dos mapas de bits en memoria cuando se aplica el filtro

Al modificar una de las propiedades de un filtro, ambos mapas de bits se actualizan en memoria para crear el mapa de bits resultante. Este proceso implica parte de procesamiento de la CPU y los dos mapas de bits pueden utilizar una cantidad significativa de memoria.

Flash Player 10.1 introduce un nuevo comportamiento de filtro en todas las plataformas. Si el filtro no se modifica en 30 segundos, o bien, si se oculta o está fuera de pantalla, la memoria que utiliza el mapa de bits sin filtro se libera.

Con esta función se ahorra la mitad de memoria utilizada por un filtro en todas las plataformas. Por ejemplo, pongamos como ejemplo un objeto de texto con un filtro de desenfoque aplicado. En este caso, el texto se utiliza para simple decoración y no se modifica. Transcurridos 30 segundos, se libera el mapa de bits sin filtro en memoria. Se obtiene el mismo resultado si el texto se oculta durante 30 segundos o está fuera de pantalla. Si se modifica una de las propiedades del filtro, el mapa de bits sin filtro en memoria se vuelve a crear. Esta función se denomina descarga dinámica de mapas de bits. A pesar de estas optimizaciones, se debe ser prudente con los filtros, ya que aún requieren un amplio procesamiento de CPU o GPU cuando se modifican.

Como práctica más recomendada, utilice mapas de bits creados mediante una herramienta de edición como, por ejemplo, Adobe® Photoshop®, para emular los filtros cuando sea posible. Evite el uso de mapas de bits dinámicos creados en tiempo de ejecución en ActionScript. La utilización de mapas de bits creados externamente, ayuda a Flash Player a reducir la carga de CPU o GPU, especialmente cuando las propiedades del filtro no cambian con el tiempo. Si es posible, cree todos los efectos que necesite en un mapa de bits en una herramienta de edición. Posteriormente, puede mostrar el mapa de bits en Flash Player sin realizar ninguna representación en el mismo, lo que puede resultar mucho más rápido.

"Mipmapping" directo

Utilice la técnica de mipmapping para escalar imágenes grandes, si es necesario.

Otra nueva función disponible en Flash Player 10.1 en todas las plataformas se relaciona con la técnica de mipmapping. Flash Player 9 introdujo una función de mipmapping que mejoró la calidad y el rendimiento de los mapas de bits con reducción de escala.

Nota: *la función de mipmapping sólo se aplica a imágenes cargadas dinámicamente o mapas de bits incorporados. La técnica de mipmapping no se aplica a los objetos de visualización que se han filtrado o se han almacenado en caché. El mipmapping sólo se puede procesar si el mapa de bits dispone de una anchura y una altura expresadas en números pares. Si se encuentra algún valor de anchura o altura que presente un número impar, el mipmapping se detiene. Por ejemplo, a una imagen de 250 x 250 se le puede aplicar mipmapping para reducirla a 125 x 125, pero no podrá aplicarse más reducción. En este caso, al menos una de las dimensiones es un número impar. Los mapas de bits con dimensiones que son potencias de dos obtienen los mejores resultados; por ejemplo: 256 x 256, 512 x 512, 1024 x 1024, etc.*

Por ejemplo, imaginemos que se carga una imagen de 1024 x 1024 y un desarrollador desea aplicarle una escala para crear una miniatura en una galería. La función de mipmapping representa la imagen correctamente cuando se escala utilizando versiones de cambio de resolución intermedias del mapa de bits como texturas. En las versiones anteriores de Flash Player se creaban versiones con reducción de escala intermedias del mapa de bits en memoria. Si se ha cargado una imagen de 1024 x 1024 y se muestra a 64 x 64, las versiones anteriores de Flash Player podrían crear todos los mapas de bits de mitad de tamaño. Por ejemplo, en este caso se podrían crear mapas de bits de 521 x 512, 256 x 256, 128 x 128 y 64 x 64.

Flash Player 10.1 admite mipmapping directamente desde el origen al tamaño de destino necesario. En el ejemplo anterior, sólo se crearían el mapa de bits original de 4 MB (1024 x 1024) y el mapa de bits con mipmapping aplicado de 16 KB (64 x 64).

La lógica de la técnica de mipmapping también funciona con la función de descarga dinámica de mapas de bits. Si sólo se utiliza el mapa de bits de 64 x 64, el mapa de bits de 4 MB original se libera de la memoria. Si se debe volver a crear el mapa MIP, el original volverá a cargarse. Asimismo, si son necesarios otros mapas de bits de distintos tamaños con mipmapping aplicado, la cadena de mapas MIP de mapas de bits se utiliza para crear el mapa de bits. Por ejemplo, si se debe crear un mapa de bits de 1:8, los mapas de bits de 1:4, 1:2 y 1:1 se examinan para determinar cuál se carga en memoria en primer lugar. Si no se encuentran otras versiones, el mapa de bits de 1:1 original se carga desde el recurso y se utiliza.

El descompresor de JPEG puede llevar a cabo el mipmapping en su propio formato. Este mipmapping directo permite que un mapa de bits de gran tamaño se descomprima directamente en un formato de mapa MIP sin cargar toda la imagen descomprimida. La generación del mapa MIP es un proceso mucho más rápido y la memoria utilizada por los mapas de bits de gran tamaño no se asigna y se libera posteriormente. La calidad de la imagen JPEG se puede comparar con la técnica de mipmapping general.

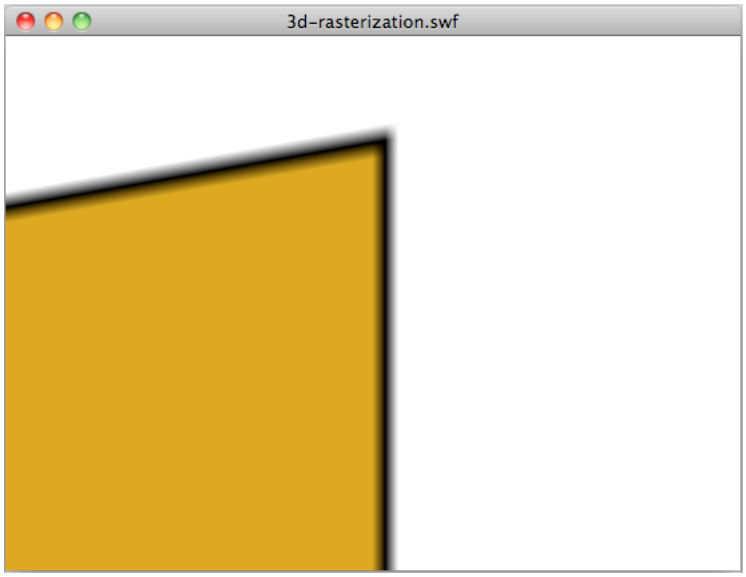
Nota: *utilice la técnica de "mipmapping" con moderación. Aunque mejora la calidad de los mapas de bits con reducción de escala, supone un impacto en el ancho de banda, la memoria y la velocidad. En algunos casos, una opción mejor puede ser el uso de una versión con escala previa del mapa de bits desde una herramienta externa y su importación en la aplicación. No comience con mapas de bits de gran tamaño si sólo se pretende la reducción de escala.*

Utilización de efectos 3D

Considere la creación manual de efectos 3D.

Flash Player 10 introdujo un motor 3D, que permite aplicar la transformación de perspectiva a los objetos de visualización. Estas transformaciones se pueden aplicar utilizando las propiedades `rotationX` y `rotationY`, o bien, con el método `drawTriangles()` de la clase Graphics. La profundidad también se puede aplicar con la propiedad `z`. Se debe tener en cuenta que cada objeto de visualización con transformación de perspectiva se rasteriza como mapa de bits y, por lo tanto, requiere más memoria.

La siguiente figura ilustra el suavizado que se produce mediante la rasterización al utilizar la transformación de perspectiva:



Suavizado resultante de la transformación de perspectiva

El suavizado es un resultado de la rasterización dinámica del contenido del vector como mapa de bits. Este suavizado se produce cuando se utilizan efectos 3D en la versión de escritorio de Flash Player. Sin embargo, el suavizado nunca se aplica en dispositivos móviles, excepto cuando se utiliza Packager for iPhone preview. Si el efecto 3D se puede crear manualmente basándose en la API nativa, se puede reducir el uso de memoria. Sin embargo, las nuevas funciones 3D introducidas en Flash Player 10 facilitan la asignación de texturas, debido a métodos como `drawTriangles()`, que administran el proceso de forma nativa.

Como desarrollador, debe decidir si el efecto 3D que se desea crear proporciona un mejor rendimiento si se procesa mediante la API nativa o manualmente. Considere la ejecución de ActionScript y rendimiento del procesamiento, así como el uso de memoria.

Nota: *la aplicación de cualquier transformación 3D a un objeto de visualización requiere dos mapas de bits en memoria: uno para el mapa de bits de origen y otro para la versión transformada de la perspectiva. De este modo, las transformaciones 3D funcionan de forma similar a los filtros. Como resultado, utilice las propiedades 3D de una aplicación móvil con moderación.*

Objetos de texto y memoria

Utilice el nuevo motor de texto para texto de sólo lectura; utilice los objetos TextField para el texto de entrada.

Flash Player 10 introdujo un nuevo y eficaz motor de texto, flash.text.engine (FTE), que conserva la memoria del sistema. Sin embargo, FTE es una API de bajo nivel que requiere una capa de ActionScript 3.0 adicional sobre la misma.

Para el texto de sólo lectura, se recomienda el uso del nuevo motor de texto, ya que ofrece un uso de poca memoria y un mejor procesamiento. Para el texto de entrada, los objetos TextField constituyen una mejor opción, ya que es necesario menos código ActionScript para crear comportamientos típicos como, por ejemplo, la administración de entrada y el ajuste de texto.

Más temas de ayuda

"Representación de objetos de texto" en la página 65

Modelo de eventos frente a funciones callback

Considere el uso de funciones callbacks sencillas, en lugar del modelo de eventos.

El modelo de eventos de ActionScript 3.0 se basa en el concepto de distribución de objetos. El modelo de eventos está orientado a objetos y se optimiza para la reutilización de código. El método `dispatchEvent()` recorre la lista de detectores y llama al método de controlador de eventos en cada objeto registrado. Sin embargo, uno de los inconvenientes del modelo de eventos radica en que es probable que se creen varios objetos durante el uso de la aplicación.

Supongamos que desea distribuir un evento desde la línea de tiempo, indicando el final de una secuencia de animación. Para llevar a cabo la notificación, se puede distribuir un evento de un fotograma específico de la línea de tiempo, tal y como se muestra en el siguiente código:

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

La clase Document puede detectar este evento con la siguiente línea de código:

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

Aunque este enfoque es correcto, el uso del modelo de eventos nativo puede ser más lento y consumir más memoria que la utilización de una función callback tradicional. Los objetos de evento se deben crear y asignar en memoria, lo que crea una ralentización del rendimiento. Por ejemplo, cuando se está detectando el evento Event.ENTER_FRAME, se crea un nuevo objeto de evento en cada fotograma para el controlador de eventos. El rendimiento puede ser especialmente lento para los objetos de visualización, debido a las fases de propagación y captura, que pueden resultar costosas si la lista de visualización es compleja.

Capítulo 3: Reducción del uso de la CPU

Otro aspecto importante en la optimización es el uso de la CPU. La optimización del procesamiento de CPU mejora el rendimiento y, como resultado, la duración de la batería en dispositivos móviles.

Mejoras de Flash Player 10.1 en el uso de la CPU

Flash Player 10.1 introduce dos nuevas funciones que ayudan a ahorrar procesamiento de CPU. Las funciones implican el modo de suspensión en los dispositivos móviles y la pausa y reanudación del contenido SWF cuando está fuera de pantalla.

Modo de suspensión

Nota: *la función de modo de suspensión no se aplica a las aplicaciones de Adobe® AIR®.*

Flash Player 10.1 introduce una nueva función en los dispositivos móviles que ayuda a ahorrar procesamiento de CPU y, como resultado, duración de la batería. Esta función implica la retroiluminación que se encuentra en diversos dispositivos móviles. Por ejemplo, si un usuario que ejecuta una aplicación móvil tiene una interrupción y deja de utilizar el teléfono, Flash Player 10.1 detecta el momento en que la retroiluminación entra en modo de suspensión. Posteriormente, reduce la velocidad de fotogramas a 4 fotogramas por segundo (fps) y detiene la representación.

El código ActionScript continúa ejecutándose en modo de suspensión, lo que es similar al establecimiento de la propiedad `Stage.frameRate` en 4 fps. Sin embargo, el paso de representación se omite, por lo que el usuario no puede ver que el reproductor se está ejecutando a 4 fps. Se ha seleccionado una velocidad de fotogramas de 4 fps, en lugar de cero, ya que permite que todas las conexiones permanezcan abiertas (NetStream, Socket y NetConnection). El cambio a cero podría interrumpir las conexiones abiertas. Se ha elegido una velocidad de actualización de 250 ms (4 fps), ya que muchos fabricantes de teléfonos utilizan este valor como frecuencia de actualización. Con este valor, la velocidad de fotogramas de Flash Player se mantiene en el mismo alcance que el propio teléfono.

Nota: *cuando Flash Player está en modo de suspensión, la propiedad* `Stage.frameRate` *devuelve la velocidad de fotogramas del archivo original SWF, en lugar de 4 fps.*

Cuando la retroiluminación vuelve al modo activo, se reanuda la representación. La velocidad de fotogramas vuelve a su valor original. Pongamos como ejemplo un reproductor con el que un usuario está escuchando música. Si la pantalla entra en modo de suspensión, Flash Player 10.1 responde en función del tipo de contenido que se está reproduciendo. Existe una lista de situaciones con el correspondiente comportamiento de Flash Player:

- La retroiluminación entra en modo de suspensión y se está reproduciendo contenido que no es A/V: la representación se detiene y el temporizador se establece en 4 fps.
- La retroiluminación entra en modo de suspensión y se está reproduciendo contenido A/V: Flash Player hace que la retroiluminación siempre esté activa y continúa la reproducción.
- La retroiluminación pasa de modo de suspensión a modo activo: Flash Player establece el temporizador en la configuración de temporizador del archivo SWF original y reanuda la representación.
- Flash Player se detiene mientras que se reproduce el contenido A/V: Flash Player restablece el estado de retroiluminación al comportamiento del sistema predeterminado debido a que A/V ya no se está reproduciendo.
- El dispositivo móvil recibe una llamada de teléfono durante la reproducción del contenido A/V: la representación se detiene y el temporizador se establece en 4 fps.

- El modo de suspensión de la retroiluminación se desactiva en un dispositivo móvil: Flash Player se comporta con normalidad.

Cuando la retroiluminación pasa a modo de suspensión, el procesamiento de las llamadas se detiene y el temporizador se ralentiza. Con esta función se ahorra procesamiento de CPU, pero no es posible basarse en la misma para crear una pausa real, tal y como sucede en una aplicación de juego.

Nota: *no se distribuye ningún evento de ActionScript cuando Flash Player entra o sale del modo de suspensión.*

Pausa y reanudación

Nota: *la función de pausa y reanudación no se aplica a las aplicaciones de Adobe® AIR®.*

Para optimizar el uso de la batería y la CPU, Flash Player 10.1 introduce una nueva función en las plataformas móviles (y modos netbook) relacionada con las instancias inactivas. El objetivo de esta función consiste en limitar el uso de la CPU deteniendo y reanudando el contenido SWF cuando éste se encuentra fuera y dentro de pantalla. Con esta función, Flash Player libera toda la memoria posible eliminando cualquier objeto que se pueda volver al crear cuando se reanude la reproducción del contenido. El contenido se considera fuera de pantalla cuando todo el contenido se encuentra de este modo.

Existen dos escenarios que hacen que el contenido SWF esté fuera de pantalla:

- El usuario desplaza la página y hace que el contenido SWF se mueva fuera de pantalla.

  En este caso, si existe alguna reproducción de audio o vídeo, el contenido se sigue reproduciendo, pero la representación se detiene. Si no hay reproducción de vídeo ni audio, para garantizar que no se detenga la reproducción o la ejecución de ActionScript, establezca el parámetro HTML `hasPriority` en true. Sin embargo, la representación del contenido SWF se detiene cuando éste se encuentra oculto o fuera de pantalla, independientemente de valor del parámetro HTML `hasPriority`.

- Una ficha está abierta en el navegador, lo que causa que el contenido SWF se desplace hacia el fondo.

  En este caso, independientemente del valor de la etiqueta HTML`hasPriority`, el contenido SWF se ralentiza a 2 fps. La reproducción de audio y vídeo se detiene y no se representa ningún contenido a no ser que el contenido SWF se haga visible de nuevo.

Administración de instancias

Utilice el parámetro HTML `hasPriority` para retrasar la carga de los archivos SWF fuera de pantalla.

Flash Player 10.1 introduce un nuevo parámetro HTML denominado `hasPriority`:

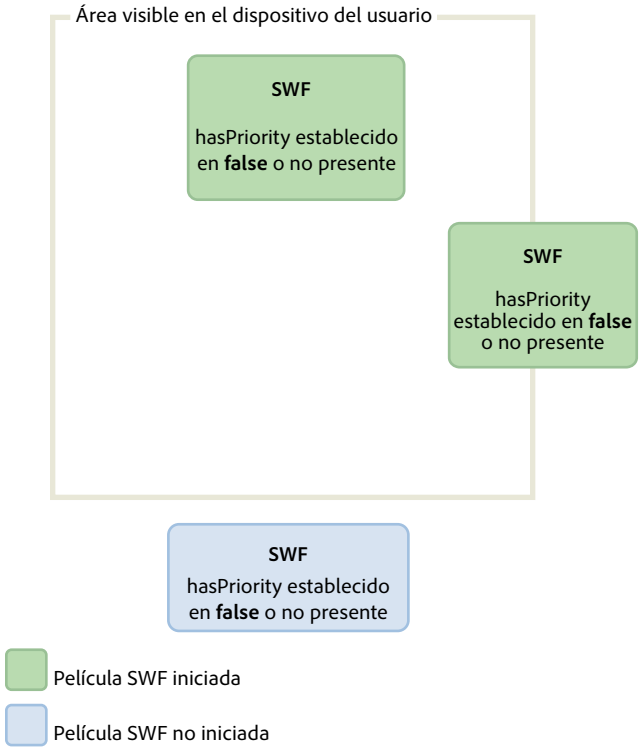
```
<param name="hasPriority" value="true" />
```

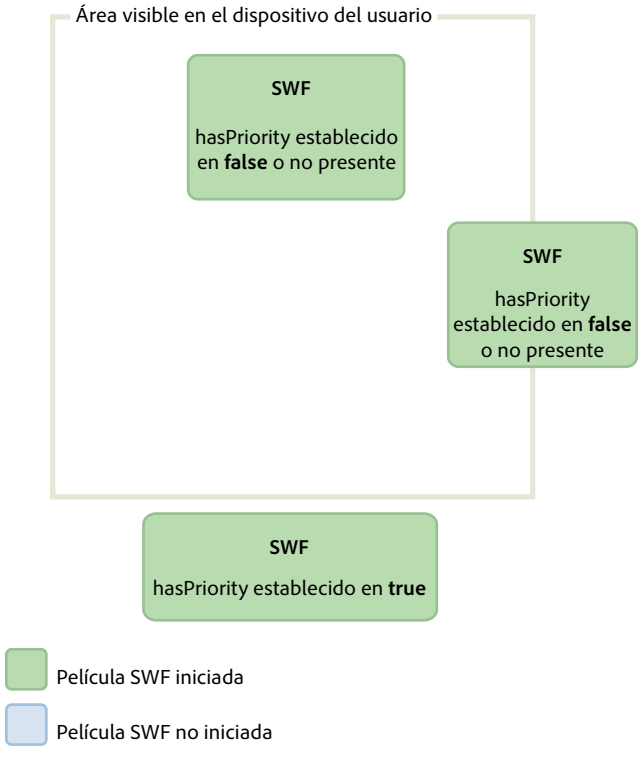
Esta función limita el número de instancias de Flash Player que se inician en una página. La limitación del número de instancias ayuda a conservar recursos de batería y CPU. La idea consiste en asignar una prioridad específica al contenido SWF, proporcionando cierta prioridad de contenido sobre los demás contenidos de una página. Pongamos un sencillo ejemplo: un usuario está navegando en un sitio web y la página de índice aloja tres archivos SWF diferentes. Uno de ellos puede verse, otro presenta visibilidad parcial en pantalla y el último está fuera de pantalla y requiere un

desplazamiento. Las primeras dos animaciones se inician con normalidad, pero la última queda aplazada hasta que sea visible. Este escenario es el comportamiento predeterminado cuando el parámetro `hasPriority no está presente o se establece en false`. Para garantizar que un archivo SWF se haya iniciado, aunque esté fuera de pantalla, establezca el parámetro `hasPriority` en `true`. No obstante, independientemente del valor del parámetro `hasPriority`, un archivo SWF que no esté visible para el usuario siempre tiene detenido su procesamiento.

Nota: *si los recursos de CPU disponibles son escasos, las instancias de Flash Player no vuelven a iniciarse automáticamente, aunque el parámetro `hasPriority` se establezca en `true`. Si las nuevas instancias se crean mediante JavaScript una vez que se haya cargado la página, estas instancias omitirán el indicador `hasPriority`. Se iniciará cualquier contenido de 1x1 ó 0x0 píxeles, evitando que los archivos SWF auxiliares se aplacen si el administrador de web no consigue incluir el indicador `hasPriority`. Sin embargo, los archivos SWF se pueden iniciar cuando se hace clic en los mismos. Este comportamiento se denomina "clic para reproducción".*

Los siguientes diagramas muestran los efectos del establecimiento del parámetro `hasPriority` en diferentes valores:





Bloqueo y desbloqueo de objetos

Bloquee y desbloquee los objetos de forma adecuada utilizando los eventos `REMOVED_FROM_STAGE` *y* `ADDED_TO_STAGE`*.*

Para optimizar el código, bloquee y desbloquee siempre los objetos. Los procesos de bloqueo y desbloqueo son importantes para todos los objetos, pero especialmente para los objetos de visualización. Aunque los objetos de visualización ya no estén en la lista de visualización y estén esperando a ser recogidos por el recolector de elementos no utilizados, aún pueden estar utilizando código con bastantes recursos de la CPU. Por ejemplo, aún pueden estar utilizando Event.ENTER_FRAME. Como resultado, es fundamental bloquear y desbloquear objetos correctamente con los eventos `Event.REMOVED_FROM_STAGE` y `Event.ADDED_TO_STAGE`. El siguiente ejemplo muestra la reproducción de un clip de película en el escenario que interactúa con el 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;
    }
}
```

Cuando se hace clic en el botón Quitar, el clip de película se elimina de la lista de visualización:

```
// 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);
}
```

Aunque se haya eliminado de la lista de visualización, el clip de película aún distribuye el evento `Event.ENTER_FRAME`. El clip de película aún se ejecuta, pero no se representa. Para administrar la situación correctamente, detecte los eventos adecuados y elimine los detectores de eventos con el fin de evitar la ejecución de código que implique una carga para los recursos de CPU:

```
// 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();
}
```

Al presionar el botón Mostrar, el clip de película se reinicia, éste vuelve a detectar los eventos `Event.ENTER_FRAME` y el teclado controla correctamente el clip de película.

Nota: *si algún objeto de visualización se elimina de la lista de visualización, el establecimiento de su referencia en* `null` *tras eliminarlo no garantiza el bloqueo del objeto. Si no se ejecuta el recolector de elementos no utilizados, el objeto continúa consumiendo memoria y procesamiento de CPU, aunque ya no se muestre. Para garantizar que el objeto consuma el mínimo procesamiento de CPU posible, asegúrese de bloquearlo por completo al eliminarlo de la lista de visualización.*

Desde Flash Player 10, si la cabeza lectora encuentra un fotograma vacío, el objeto de visualización se bloquea automáticamente aunque no se haya implementado ningún comportamiento de bloqueo.

El concepto de bloqueo también resulta importante al cargar contenido remoto con la clase Loader. Al utilizar la clase Loader con Flash Player 9, era necesario bloquear el contenido manualmente detectando el evento `Event.UNLOAD` distribuido por el objeto LoaderInfo. Todos los objetos tenían que bloquearse manualmente, lo cual implicaba una tarea compleja. Flash Player 10 introdujo un nuevo método importante en la clase Loader denominado `unloadAndStop()`. Este método permite descargar un archivo SWF, bloquear automáticamente todos los objetos en el archivo SWF cargado y provocar la ejecución del recolector de elementos no utilizados.

En el siguiente código, el archivo SWF se carga y después de descarga utilizando el método `unload()`, que requiere más procesamiento y bloqueo 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();
}
```

Una buena práctica consiste en utilizar el método `unloadAndStop()`, que administra el bloqueo de forma nativa y provoca la ejecución del proceso de recolección de elementos no utilizados:

```
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();
}
```

Cuando se llama al método `unloadAndStop()` se producen las siguientes operaciones:

- Se detienen los sonidos.
- Los detectores registrados en la línea de tiempo principal del archivo SWF se eliminan.
- Los objetos del temporizador se detienen.
- Los dispositivos periféricos de hardware (como la cámara y el micrófono) se liberan.
- Todos los clips de película se detienen.
- La distribución `Event.ENTER_FRAME`, `Event.FRAME_CONSTRUCTED`, `Event.EXIT_FRAME`, `Event.ACTIVATE` y `Event.DEACTIVATE` se detiene.

Interacciones de ratón

Considere la desactivación de interacciones de ratón, si es posible.

Cuando se utiliza un objeto interactivo como, por ejemplo, MovieClip o Sprite, Flash Player ejecuta código nativo para detectar y administrar las interacciones de ratón. La detección de la interacción de ratón puede implicar una carga para los recursos de CPU cuando varios objetos interactivos se muestran en pantalla, especialmente si se superponen. Una forma sencilla de evitar este procesamiento consiste en deshabilitar las interacciones de ratón en objetos que no requieran ninguna interacción de este tipo. El siguiente código ilustra el uso de las propiedades `mouseEnabled` y `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;
```

Si es posible, considere la desactivación de la interacción de ratón, lo que ayuda a que la aplicación utilice menos procesamiento de CPU y, como resultado, reduce el uso de la batería.

Temporizadores frente a eventos ENTER_FRAME

Seleccione temporizadores o eventos ENTER_FRAME, dependiendo de si el contenido es animado.

Los temporizadores son más recomendables que los eventos `Event.ENTER_FRAME` para el contenido no animado que se ejecuta durante mucho tiempo.

En ActionScript 3.0, existen dos maneras de llamar a una función en intervalos específicos. El primer enfoque consiste en utilizar el evento `Event.ENTER_FRAME` distribuido mediante objetos interactivos (InteractiveObject). El segundo enfoque radica en utilizar un temporizador. Los desarrolladores de ActionScript suelen utilizar el enfoque del evento `ENTER_FRAME`. El evento `ENTER_FRAME` se distribuye en todos los fotogramas. Como resultado, el intervalo en el que se llama a la función se relaciona con la velocidad de fotogramas real actual. A la velocidad de fotogramas se accede mediante la propiedad `Stage.frameRate`. Sin embargo, en algunos casos, el uso de un termporizador puede ser una mejor opción que la utilización del evento `ENTER_FRAME`. Por ejemplo, si no se utiliza la animación pero se desea que el código se llame en intervalos específicos, el uso de un temporizador puede constituir una opción más adecuada.

Un temporizador se puede comportar del mismo modo que un evento `ENTER_FRAME`, pero un evento se puede distribuir sin tener que asociarse a la velocidad de fotogramas. Este comportamiento puede ofrecer cierta optimización importante. Considere una aplicación de reproductor de vídeo como ejemplo. En este caso, no es necesario utilizar una velocidad de fotogramas elevada, ya que únicamente se están moviendo los controles de la aplicación.

Nota: *la velocidad de fotogramas no afecta al vídeo, ya que éste no está incorporado en la línea de tiempo. En cambio, el vídeo se carga dinámicamente mediante la transmisión o la descarga progresiva.*

En este ejemplo, la velocidad de fotogramas se establece en un valor bajo de 10 fps. El temporizador actualiza los controles a una velocidad de 25 actualizaciones por segundo. La velocidad de actualización más elevada es posible a través del método `updateAfterEvent()`, que está disponible en el objeto TimerEvent. Este método obliga a la pantalla a actualizarse cada vez que el temporizador distribuye un evento, si es necesario. De este modo es posible obtener lo mejor de ambos enfoques: una representación suave sin asociación a la velocidad de fotogramas. El siguiente código ilustra la idea:

```
// Use  a low frame rate for the application
stage.frameRate = 10;
// Update rate for the timer
const FRAME_RATE:int = 30;
// Choose 30 updates a second
var updateInterval:int = 1000/FRAME_RATE;
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();
}
```

La llamada al método `updateAfterEvent()` no modifica la velocidad de fotogramas. Simplemente obliga a Flash Player a actualizar el contenido en pantalla que se ha modificado. La línea de tiempo aún se ejecuta a 10 fps. Recuerde que los temporizadores y eventos `ENTER_FRAME` no son totalmente precisos en dispositivos de bajo rendimiento o si las funciones del controlador de eventos contienen código que requiere un gran procesamiento. Al igual que sucede con la velocidad de fotogramas de los archivos SWF, la velocidad de fotogramas de actualización del temporizador puede variar en algunas situaciones.

Reduzca el número de objetos Timer y controladores `enterFrame` registrados en la aplicación.

Cada fotograma, el motor de ejecución distribuye un evento `enterFrame` en cada objeto de visualización en su lista de visualización. Aunque se pueden registrar detectores para el evento `enterFrame` con varios objetos de visualización, esto significa que se ejecuta más código cada fotograma. Considere el uso de un solo controlador `enterFrame` centralizado que ejecute todo el código que se vaya a ejecutar en cada fotograma. Al centralizar este código, resulta más fácil administrar todo el código que se ejecuta frecuentemente.

Asimismo, si está utilizando objetos Timer, existe una sobrecarga asociada a la creación y distribución de eventos desde varios objetos Timer. Se deben activar diferentes operaciones en intervalos distintos; a continuación se incluyen algunas alternativas sugeridas:

- Utilice un número mínimo de objetos Timer y operaciones de grupo según la frecuencia en que se produzcan.

  Por ejemplo, utilice un objeto Timer para operaciones frecuentes y defínalo para que se active cada 100 milisegundos. Utilice otro objeto Timer para operaciones en segundo plano menos frecuentes y defínalo para se active cada 2000 milisegundos.

- Use un solo objeto Timer y establezca las operaciones para que se activen en múltiplos del intervalo de la propiedad `delay` del objeto Timer.

  Por ejemplo, supongamos que se tienen algunas operaciones que se espera que se produzcan cada 100 milisegundos y otras que se desea que sucedan cada 200 milisegundos. En ese caso, utilice un solo objeto Timer con un valor `delay` de 100 milisegundos. En el controlador de eventos `timer`, añada una sentencia condicional que sólo ejecute las operaciones de 200 milisegundos cada vez. En el siguiente ejemplo se demuestra 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;
}
```

Detenga los objetos Timer cuando no se utilicen.

Si un controlador de eventos `timer` del objeto Timer sólo realiza operaciones en determinadas condiciones, llame al método `stop()` del objeto Timer cuando ninguna de las condiciones tenga el valor de true.

En el evento `enterFrame` o los controladores Timer, reduzca el número de cambios para mostrar el aspecto de los objetos de visualización que hacen que la pantalla deba redibujarse.

Con cada fotograma, la fase de representación vuelve a dibujar la parte del escenario que ha cambiado durante ese fotograma. Si la región de redibujo es extensa, o bien, si es pequeña pero contiene una gran cantidad de objetos de visualización complejos, el motor de ejecución necesita más tiempo para la representación. Para comprobar la cantidad de redibujo necesario, utilice la función “Mostrar regiones de redibujo” en la versión de depuración de Flash Player o AIR.

Para obtener más información sobre la mejora del rendimiento para acciones repetidas, consulte el siguiente artículo:

- Writing well-behaved, efficient, AIR applications (Escritura de aplicaciones de AIR eficaces y de buen funcionamiento; en inglés). (Artículo y aplicación de ejemplo de Arno Gourdol).

Más temas de ayuda

"Aislamiento de comportamientos" en la página 62

Síndrome de interpolación

Para ahorrar CPU, limite el uso de la interpolación, lo que ahorra procesamiento de CPU, memoria y duración de la batería.

Los diseñadores y desarrolladores que producen contenido para Flash en el escritorio, tienden a utilizar diversas interpolaciones de movimiento en sus aplicaciones. Al producir contenido para dispositivos móviles con bajo rendimiento, intente reducir el uso de las interpolaciones de movimiento. La limitación de su uso ayuda a que el contenido se ejecute más rápido en dispositivos de capa baja.

Capítulo 4: Rendimiento de ActionScript 3.0

Clase Vector frente a la clase Array


Si es posible, utilice la clase Vector en lugar de la clase Array.

Flash Player 10 introdujo la clase Vector, que permite un acceso más rápido de lectura y escritura que la clase Array.

Una sencilla prueba comparativa muestra las ventajas de clase Vector respecto de la clase Array. El siguiente código muestra una prueba comparativa para la clase 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
```

El siguiente código muestra una prueba comparativa para la clase 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
```

El ejemplo se puede optimizar con un mayor nivel asignando una longitud específica al vector y estableciendo su longitud como fija:

```
// 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
```

Si el tamaño del vector no se especifica con antelación, el tamaño aumenta cuando el vector se ejecuta sin espacio. Cada vez que aumenta el tamaño del vector, se asigna un nuevo bloque de memoria. El contenido actual del vector se copia en el nuevo bloque de memoria. Esa asignación adicional y copia de datos daña el rendimiento. El código anterior se optimiza para el rendimiento especificando el tamaño inicial del vector. Sin embargo, el código no se optimiza para el mantenimiento. Para mejorar también el mantenimiento, almacene el valor reutilizado en una 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
```

Intente utilizar las API del objeto Vector, si es posible, ya que es probable que se ejecuten con más rapidez.

API de dibujo

Utilice la API de dibujo para una ejecución más rápida del código.

Flash Player 10 incluía una nueva API de dibujo, que permite obtener un mejor rendimiento en la ejecución del código. Esta nueva API no supone una mejora del rendimiento de representación, pero puede reducir en gran medida el número de líneas de código que se debe escribir. Al contar con menos líneas de código, se puede obtener un mejor rendimiento de ejecución de ActionScript.

La nueva API de dibujo incluye los siguientes métodos:

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

***Nota:** este análisis no se centra en el método `drawTriangles()`, que está relacionado con 3D. No obstante, este método puede mejorar el rendimiento de ActionScript, ya que administra los mapas de textura nativos.*

El siguiente código llama de forma explícita al método adecuado para cada línea que se dibuja:

```
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 );
```

El siguiente código se ejecuta más rápido que el ejemplo anterior, ya que se ejecutan menos líneas de código. Cuanto más complejo sea el trazado, más aumentará el rendimiento al utilizar el 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 );
```

El método `drawGraphicsData()` proporciona mejoras de rendimiento similares.

Propagación y captura de eventos

Utilice la propagación y captura de eventos para reducir los controladores de eventos.

El modelo de eventos de ActionScript 3.0 introdujo los conceptos de propagación y captura de eventos. El aprovechamiento de la propagación de un evento puede ayudar a optimizar el tiempo de ejecución de código ActionScript. Con el fin de mejorar el rendimiento, es posible registrar un controlador de eventos en un objeto, en lugar de en varios objetos.

Como ejemplo, imagine la creación un juego en el que el usuario debe hacer clic en manzanas lo más rápido posible para destruirlas. El juego elimina cada manzana de pantalla cuando se hace clic en la misma y añade puntos a la puntuación del usuario. Para detectar el evento `MouseEvent.CLICK` distribuido por cada manzana, se puede tender a escribir el siguiente 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 );
}
```

El código llama al método `addEventListener()` en cada instancia de Apple. También se elimina cada detector de evento cuando se hace clic en una manzana, utilizando el método `removeEventListener()`. Sin embargo, el modelo de eventos de ActionScript 3.0 proporciona una fase de propagación y captura para algunos eventos, lo que permite detectarlos desde un objeto InteractiveObject principal. Como resultado, es posible optimizar el código anterior y reducir el número de llamadas a los métodos `addEventListener()` y `removeEventListener()`. El siguiente código utiliza la fase de captura para detectar los eventos del objeto principal:

```
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 );
}
```

El código se simplifica y se optimiza mucho más, con sólo una llamada al método `addEventListener()` en el contenedor principal. Los detectores no vuelven a registrarse en las instancias de Apple, por lo que no es necesario eliminarlos cuando se hace clic en una manzana. El controlador `onAppleClick()` puede optimizarse más, deteniendo la propagación del evento, lo que evita que éste vaya más allá:

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

La fase de propagación también se puede utilizar para capturar el evento, transmitiendo `false` como tercer parámetro al 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 );
```

El valor predeterminado para el parámetro de fase de captura es `false`, por lo que puede omitirlo:

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

Trabajo con píxeles

Pinte píxeles utilizando el método `setVector()`.

Al pintar píxeles, se pueden llevar a cabo algunas optimizaciones sencillas simplemente utilizando los métodos apropiados de la clase BitmapData. Un modo rápido de pintar píxeles consiste en usar el método `setVector()` introducido en Flash Player 10:

```
// 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 );
```

Al emplear métodos lentos como, por ejemplo, `setPixel()` o `setPixel32()`, utilice los métodos `lock()` y `unlock()` para agilizar el proceso de ejecución. En el siguiente código, los métodos `lock()` y `unlock()` se utilizan para mejorar el rendimiento:

```
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
```

El método `lock()` de la clase BitmapData bloquea una imagen y evita que los objetos que le hacen referencia se actualicen cuando cambia el objeto BitmapData. Por ejemplo, si un objeto Bitmap hace referencia a BitmapData, es posible bloquear el objeto BitmapData, cambiarlo y posteriormente bloquearlo. El objeto Bitmap no se cambia hasta que se desbloquee el objeto BitmapData. Para mejorar el rendimiento, utilice este método junto con el método `unlock()` antes y después de realizar numerosas llamadas al método `setPixel()` o `setPixel32()`. Al llamar a `lock()` y `unlock()`, se evita que la pantalla se actualice de forma innecesaria.

Nota: *cuando se procesan píxeles en un mapa de bits fuera de la lista de visualización (doble almacenamiento en búfer), a veces esta técnica no mejora el rendimiento. Si un objeto de mapa de bits no hace referencia al búfer de mapa de bits, el uso de `lock()` y `unlock()` no supone una mejora de rendimiento. Flash Player detecta que el búfer no cuenta con referencias y el mapa de bits no se representa en pantalla.*

Es probable que los métodos que se repiten en los píxeles como, por ejemplo, `getPixel()`, `getPixel32()`, `setPixel()` y `setPixel32()`, sean lentos, especialmente en dispositivos móviles. Si es posible, utilice métodos que recuperen todos los píxeles en una llamada. Para la lectura de píxeles, utilice el método `getVector()`, que resulta más rápido que `getPixels()`. Asimismo, recuerde el uso de las API basadas en los objetos Vector, ya que es probable que su ejecución sea más rápida.

Expresiones regulares

Utilice métodos de la clase String como, por ejemplo, `indexOf()`, `substr()` o `substring()` en lugar de una expresión regular para la extracción y búsqueda básica de cadenas.

Determinadas operaciones que se pueden llevar a cabo utilizando una expresión regular, también se pueden realizar utilizando métodos de la clase String. Por ejemplo, para comprobar si una cadena contiene otra cadena, se puede utilizar el método `String.indexOf()` o emplear una expresión regular. Sin embargo, cuando un método de la clase String está disponible, se ejecuta más rápido que la expresión regular equivalente y no requiere la creación de otro objeto.

Utilice un grupo sin captura ("`(?:xxxx)`") en lugar de un grupo ("`(xxxx)`") en una expresión regular si necesita agrupar elementos, pero no es necesario aislar el contenido del grupo en el resultado.

Con frecuencia, en las expresiones regulares de complejidad moderada, se agrupan partes de la expresión. Por ejemplo, en el siguiente patrón de expresión regular, los paréntesis crean un grupo alrededor del texto "ab." Por lo tanto, el cuantificador "+" se aplica al grupo en lugar de a un solo carácter:

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

De forma predeterminada, el contenido de cada grupo se "captura". Es posible obtener el contenido de cada grupo en el patrón como parte del resultado de la ejecución de la expresión regular. La captura de estos resultados de grupo tarda más tiempo y requiere más memoria, ya que los objetos se crean para contener los resultado de grupo. Como alternativa, puede utilizar la sintaxis de grupo sin captura incluyendo un signo de interrogación y dos puntos tras el paréntesis de apertura. Esta sintaxis especifica que los caracteres se comportan como grupo pero que no se capturan para el resultado:

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

El uso de la sintaxis de grupo sin captura es más rápido y utiliza menos memoria que la utilización de la sintaxis de grupo estándar.


Considere el uso de un patrón de expresión regular alternativo si una expresión regular funciona de forma inadecuada.

En ocasiones, es posible utilizar varios patrones de expresión regular para probar o identificar el mismo patrón de texto. Por diversos motivos, determinados patrones se ejecutan más rápido que otras alternativas. Si se determina que una expresión regular está causando que el código se ejecute de forma más lenta de lo necesario, considere el uso de patrones de expresión regulares alternativos con los que se obtenga el mismo resultado. Pruebe estos patrones alternativos para determinar cuál es el más rápido.

Otras optimizaciones


Para un objeto TextField, use el método `appendText()` en lugar del operador `+=`.

Al trabajar con la propiedad `text` de la clase TextField, utilice el método `appendText()` en lugar del operador `+=`. Al usar el método `appendText()`, se obtienen mejoras de rendimiento.

Por ejemplo, el siguiente código utiliza el operador `+=` y el bucle tarda 1120 ms en completarse:

```
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
```

En el siguiente ejemplo, el operador `+=` se reemplaza con el 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
```

El código tarda ahora 847 ms en completarse.


Actualice los campos de texto fuera de los bucles, si es posible.

Este código se puede optimizar aún más utilizando una sencilla técnica. En la actualización del campo de texto en cada bucle se emplea mucho procesamiento interno. Simplemente al concatenar una cadena y asignarle el valor del campo de texto fuera del bucle, el tiempo de ejecución del código disminuye sustancialmente. El código tarda ahora 2 ms en completarse:

```
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
```

Al trabajar con texto HTML, el enfoque anterior resulta tan lento que se genera una excepción `Timeout` en Flash Player, en algunos casos. Por ejemplo, se puede emitir una excepción si el hardware subyacente es demasiado lento.

Nota: *Adobe® AIR® no emite esta excepción.*

```
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 );
```

Al asignar el valor a una cadena fuera del bucle, el código necesita únicamente 29 ms para completarse:

```
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: *en Flash Player 10.1, la clase String se ha mejorado, por lo que las cadenas utilizan menos memoria.*

Si es posible, se debe evitar el uso del operador de corchetes.

La utilización del operador de corchetes puede ralentizar el rendimiento. Para evitar su uso, se puede almacenar la referencia en una variable local. En el siguiente código se muestra el uso ineficaz del operador de corchetes:

```
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
```

En la siguiente versión optimizada se reduce el uso del operador de corchetes:

```
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
```


Cuando sea posible, sitúe código entre líneas para reducir el número de llamadas a funciones.

Las llamadas a funciones pueden resultar caras. Intente reducir el número de llamadas a funciones moviendo el código en línea. El movimiento del código en línea constituye un buen modo de optimizar para un rendimiento total. Sin embargo, se debe tener en cuenta que el código en línea puede hacer que el código sea más difícil de reutilizar y puede aumentar el tamaño del archivo SWF. Algunas llamadas a funciones como, por ejemplo, los métodos de la clase Math, son más fáciles de mover en línea. En el siguiente código se utiliza el 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
```

El cálculo realizado por `Math.abs()` se puede llevar a cabo de forma manual y se puede mover en línea:

```
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
```

El movimiento de las llamadas a funciones en línea implica que la velocidad del código se multiplique por un valor superior a cuatro. Este enfoque resulta útil en diversas situaciones, pero se debe tener en cuenta el efecto que tiene en la reutilización y el mantenimiento.

Nota: *el tamaño del código tiene un gran impacto en la ejecución general del reproductor. Si la aplicación incluye grandes cantidades de código ActionScript, el equipo virtual emplea importantes cantidades de tiempo en la verificación del código y la compilación JIT. Las búsquedas de propiedades pueden resultar más lentas, debido a las jerarquías de herencia más profundas y a que las cachés internas tienden a una mayor hiperpaginación. Para reducir el tamaño del código, evite la utilización de la arquitectura Adobe® Flex®, la biblioteca de la arquitectura TLF o cualquier otra gran biblioteca de ActionScript de terceros.*


Evite la evaluación de sentencias en bucles.

Se puede lograr otra optimización al no evaluar una sentencia dentro de un bucle. El siguiente código establece una iteración en un conjunto, pero no se optimiza porque la longitud del conjunto se evalúa en cada repetición:

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

Es mejor almacenar el valor y volver a utilizarlo:

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

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


Utilice el orden inverso para los bucles while.

Un bucle while en orden inverso es más rápido que un bucle forward:

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

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

Estas sugerencias proporcionan algunas maneras de optimizar ActionScript, mostrando el modo en que una sola línea de código puede afectar al rendimiento y la memoria. Son posibles otras muchas optimizaciones de ActionScript. Para obtener más información, consulte: http://www.rozengain.com/blog/2007/05/01/some-actionscript-30-optimizations/ (en inglés).

Capítulo 5: Rendimiento de la representación

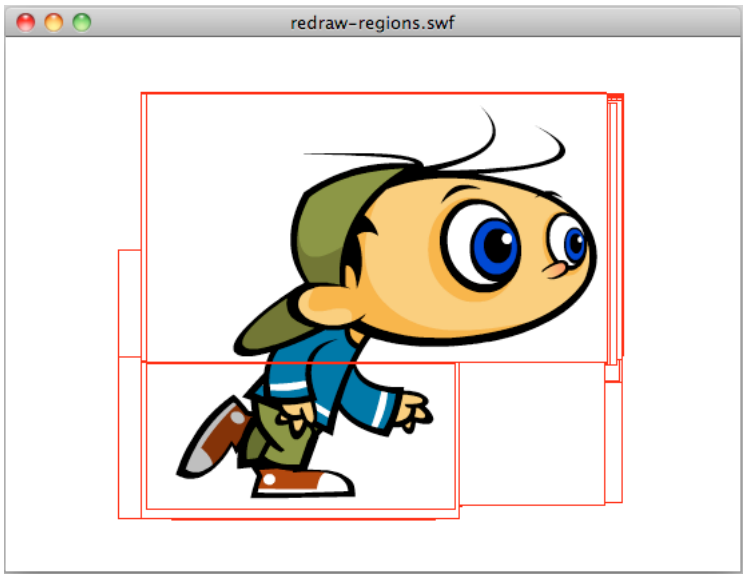
Regiones de redibujo

Utilice siempre la opción de regiones de redibujo al crear un proyecto.

Para mejorar la representación, es importante utilizar la opción de regiones de redibujo al crear un proyecto. Esta opción permite ver las regiones que Flash Player está representando y procesando. Esta opción se puede activar seleccionando Mostrar regiones de redibujo en el menú contextual del reproductor de depuración.

Nota: *la opción Mostrar regiones de redibujo no está disponible en la versión oficial del reproductor.*

La siguiente imagen ilustra la opción activada con un sencillo objeto MovieClip animado en la línea de tiempo:



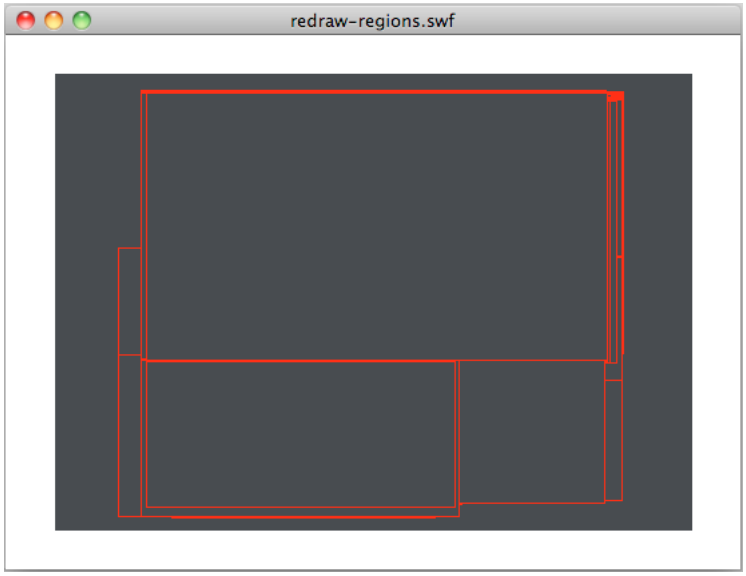
Opción de regiones de redibujo activadas.

También se puede habilitar esta opción mediante programación, con el uso del método `flash.profiler.showRedrawRegions()`:

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

Esta línea de código resulta útil al utilizar Adobe AIR, donde el menú contextual no está disponible. Cuando se utiliza en un sitio web completo, proporciona una indicación del grado de optimización general ya realizado. Recuerde que aunque algunos objetos de visualización no se muestren, todavía pueden consumir ciclos de CPU, ya que aún se están procesando.

La siguiente imagen ilustra esta idea. Una forma vectorial negra cubre el personaje animado en ejecución. La imagen muestra que el objeto no se ha eliminado de la lista de visualización y que aún se está representando. Con esto se desaprovechan ciclos de CPU:



Regiones redibujadas.

Para mejorar el rendimiento, elimine el personaje en ejecución de la lista de visualización cuando no se esté utilizando y detenga su línea de tiempo. Con estos pasos se garantiza que el objeto de visualización se bloquee y utilice los recursos mínimos de CPU.

Recuerde utilizar la opción de regiones de redibujo durante todo el ciclo de desarrollo. Con esta opción evitará sorpresas al final del proyecto con la aparición de regiones de redibujo innecesarias y áreas de optimización que pueden haberse perdido.

Más temas de ayuda

"Bloqueo y desbloqueo de objetos" en la página 27

Calidad de la película

Utilice la configuración de calidad del escenario adecuada para mejorar la representación.

Al desarrollar contenido para dispositivos móviles con pequeñas pantallas como, por ejemplo, teléfonos, la calidad de la imagen es menos importante que al desarrollar aplicaciones de escritorio. El establecimiento de la calidad del escenario en una configuración adecuada puede mejorar el rendimiento de la representación.

La siguiente configuración está disponible para la calidad del escenario:

- `StageQuality.LOW`: favorece la velocidad de reproducción sobre el aspecto y no utiliza la visualización suavizada.
- `StageQuality.MEDIUM`: aplica un cierto grado de suavizado, pero no suaviza los mapas de bits.

- `StageQuality.HIGH`: (opción predeterminada en el escritorio) favorece al aspecto frente a la velocidad de reproducción y siempre utiliza la visualización suavizada. Si el archivo SWF no contiene animación, los mapas de bits se suavizan; de lo contrario, no se suavizan.
- `StageQuality.BEST`: proporciona la mejor calidad de visualización y no considera la velocidad de reproducción. Las imágenes resultantes y los mapas de bits siempre se suavizan.

  Con el uso de `StageQuality.MEDIUM` se suele obtener una calidad suficiente para los dispositivos móviles y, en algunos casos, la utilización de `StageQuality.LOW` puede proporcionar la calidad necesaria. Desde Flash Player 8, el texto suavizado se puede representar de forma adecuada, incluso con la calidad del escenario (Stage) establecida en `LOW`.

  Nota: *en algunos dispositivos móviles, aunque la calidad se establece en `HIGH` (alta) (4x4 AA), el valor `MEDIUM` (media) (2x2 AA) se emplea para un mejor rendimiento. Sin embargo, la definición de la calidad como HIGH no suele implicar una diferencia notable, ya que las pantallas móviles suelen tener un valor más alto de ppp. Los escritorios presentan alrededor de 100 ppp, mientras que las pantallas móviles se acercan a 200 ppp, lo que significa que el "tamaño de subpíxel" es el mismo aproximadamente. El valor de ppp puede variare dependiendo del dispositivo.*

  En la siguiente figura, la calidad de la película se establece en `MEDIUM` (media) y la representación del texto se define como Suavizado para animación:

Calidad media del escenario y representación de texto establecidos como Suavizado para animación

La configuración de la calidad del escenario afecta a la calidad del texto, ya que no se está utilizando la configuración adecuada de representación de texto.

Una función de Flash Player 8 permite establecer la representación de texto en Suavizado para legibilidad. Esta configuración mantiene la calidad perfecta del texto (suavizado) independientemente de la configuración de calidad del escenario que se utilice:

Calidad baja del escenario y representación de texto establecidos en Suavizado para legibilidad

Se puede obtener la misma calidad de representación estableciendo la representación de texto en Texto de mapa de bits (sin suavizado):

Calidad baja del escenario y representación de texto establecidos en Texto de mapa de bits (sin suavizado)

Los últimos dos ejemplos muestran que es posible obtener texto de alta calidad, independientemente de la configuración de calidad del escenario que se utilice. Esta función ha estado disponible desde Flash Player 8 y se puede utilizar en dispositivos móviles. Se debe tener en cuenta que Flash Player 10.1 cambia automáticamente a `StageQuality.MEDIUM` en algunos dispositivos con el fin de aumentar el rendimiento.

Mezcla alfa

Evite el uso de la propiedad alpha, si es posible.

Evite la utilización de efectos que requieran mezcla alfa al usar la propiedad `alpha` como, por ejemplo, efectos de aparición o desaparición progresiva. Se debe tener en cuenta que la mezcla alfa requiere un uso intensivo del procesador para Flash Player y puede dañar el rendimiento en dispositivos lentos. Evite el uso de la propiedad `alpha`, si es posible.

Más temas de ayuda

"Almacenamiento en caché de mapas de bits" en la página 53

"Representación de objetos de texto" en la página 65

Velocidad de fotogramas de la aplicación

En general, utilice la menor velocidad de fotogramas posible para obtener un mejor rendimiento..

La velocidad de fotogramas de la aplicación determina cuánto tiempo hay disponible para cada ciclo de "representación y código de la aplicación", tal y como se describe en "Fundamentos de la ejecución de código del motor de ejecución" en la página 1. Con una velocidad de fotogramas más elevada se crea una animación más fluida. Sin embargo, cuando no se está produciendo la animación ni otros cambios visuales, no suele haber motivos para utilizar una velocidad de fotogramas elevada.

A continuación se incluyen algunas instrucciones generales para una velocidad de fotogramas adecuada para la aplicación:

- Si está utilizando la arquitectura de Flex, deje la velocidad de fotogramas inicial con el valor predeterminado.
- Si la aplicación incluye animación, una velocidad de fotogramas adecuada debe ser como mínimo de 20 fotogramas por segundo. Un valor superior a 30 fotogramas por segundo suele ser innecesario.
- Si la aplicación no incluye animación, probablemente una velocidad de fotogramas de 12 fotogramas por segundo sea suficiente.

Se debe tener en cuenta que la "menor velocidad de fotogramas posible" puede variar dependiendo de la actividad actual de la aplicación. Para obtener más información, consulte la siguiente sugerencia "Cambio dinámico de la velocidad de fotogramas de la aplicación".

Utilice una velocidad de fotogramas baja si el vídeo es el único contenido dinámico en la aplicación..

El motor de ejecución reproduce el contenido de vídeo cargado a su velocidad de fotogramas nativa, independientemente de la velocidad de fotogramas de la aplicación. Si la aplicación no tiene animación ni cambia rápidamente el contenido visual, el uso de una velocidad de fotogramas baja no degrada la experiencia del usuario.

Cambio dinámico de la velocidad de fotogramas de la aplicación.

La velocidad de fotogramas inicial de la aplicación se define en un proyecto o configuración del compilador, pero la velocidad no se fija en ese valor. La velocidad de fotogramas se puede modificar estableciendo la propiedad `Stage.frameRate` (o la propiedad `WindowedApplication.frameRate` en Flex).

La velocidad de fotogramas se debe cambiar en función de las necesidades actuales de la aplicación. Por ejemplo, durante un tiempo en el que la aplicación no esté realizando ninguna animación, reduzca la velocidad de fotogramas. Si una transición animada está a punto de empezar, aumente la velocidad de fotogramas. Del mismo modo, si la aplicación se está ejecutando en segundo plano (tras dejar de ser la primera selección), generalmente se puede reducir aún más la velocidad de fotogramas. El probable que el usuario se centre en otra aplicación o tarea.

A continuación se incluyen algunas instrucciones generales para utilizar como punto de partida a la hora de determinar la velocidad de fotogramas adecuada para diferentes tipos de actividades:

- Si está utilizando la arquitectura de Flex, deje la velocidad de fotogramas inicial con el valor predeterminado.

- Cuando se está reproduciendo una animación, la velocidad de fotogramas debe ser de al menos 20 fotogramas por segundo. Un valor superior a 30 fotogramas por segundo suele ser innecesario.
- Cuando no se está reproduciendo ninguna animación, probablemente una velocidad de 12 fotogramas por segundo sea suficiente.
- El vídeo cargado se reproduce en su velocidad de fotogramas nativa independientemente de la velocidad de fotogramas de la aplicación. Si el vídeo es el único contenido en movimiento en la aplicación, probablemente una velocidad de 12 fotogramas por segundo sea suficiente.
- Cuando la aplicación no tiene ninguna selección de entrada, probablemente una velocidad de 5 fotogramas por segundo sea suficiente.
- Cuando una aplicación de AIR no está visible, una velocidad de 2 fotogramas por segundo o menos resulta adecuada probablemente. Por ejemplo, esta pauta se aplica si una aplicación se minimiza o si la propiedad `visible` de la ventana nativa es `false`.

Para las aplicaciones creadas en Flex, la clase spark.components. WindowedApplication tiene compatibilidad incorporada para cambiar dinámicamente la velocidad de fotogramas de la aplicación. La propiedad `backgroundFrameRate` define la velocidad de fotogramas de la aplicación cuando ésta no se encuentra activa. El valor predeterminado es 1, lo que cambia la velocidad de fotogramas de una aplicación creada con la arquitectura de Spark a 1 fotograma por segundo. Se puede cambiar la velocidad de fotogramas en segundo plano, estableciendo la propiedad `backgroundFrameRate`. La propiedad se puede establecer en otro valor, o bien, defínalo en -1 para desactivar la limitación automática de velocidad de fotogramas.

Para obtener más información sobre el cambio dinámico de la velocidad de fotogramas de una aplicación, consulte los siguientes artículos:

- Reducing CPU usage in Adobe AIR (Reducción del uso de la CPU en Adobe AIR). (Artículo del Centro de desarrollo de Adobe y código de ejemplo de Jonnie Hallman; en inglés.)
- Writing well-behaved, efficient, AIR applications (Escritura de aplicaciones de AIR eficaces y de buen funcionamiento; en inglés). (Artículo y aplicación de ejemplo de Arno Gourdol).

Grant Skinner ha creado una clase limitadora de la velocidad de fotogramas. Esta clase se puede utilizar en las aplicaciones para reducir automáticamente la velocidad de fotogramas cuando la aplicación está en segundo plano. Para obtener más información y descargar el código fuente de la clase FramerateThrottler, consulte el artí-culo de Grant "Idle CPU Usage in Adobe AIR and Flash Player" (Uso de la CPU inactiva en Adobe AIR y Flash Player; en inglés), en http://www.adobe.com/go/learn_fp_framerate_throttling_es.

Velocidad del reproductor adaptable

Al compilar un archivo SWF, se establece una velocidad de fotogramas específica para la película. En un entorno restringido con una velocidad de CPU baja, la velocidad de fotogramas se suele reducir durante la reproducción. Para conservar una velocidad aceptable para el usuario, Flash Player omite la representación de algunos fotogramas. Con esta operación, se evita que la velocidad de fotogramas disminuya bajo un valor aceptable.

Nota: *en este caso, Flash Player no omite fotogramas; únicamente omite la representación del contenido de los fotogramas. El código aún se ejecuta y la lista de visualización se actualiza, pero las actualizaciones no se muestran en pantalla. No existe ningún modo de especificar un valor límite de fps que indique cuántos fotogramas omitir si Flash Player no puede mantener la velocidad de fotogramas estable.*

Almacenamiento en caché de mapas de bits

Utilice la función de almacenamiento en caché de mapas de bits para el contenido vectorial complejo, si resulta necesario.

Se puede realizar una buena optimización utilizando la función de almacenamiento en caché de mapas de bits introducida en Flash Player 8. Esta función almacena en caché un objeto vectorial, lo representa como un mapa de bits internamente y utiliza este mapa de bits para la representación. El resultado puede ser un enorme aumento de rendimiento para la representación, pero se requiere una cantidad considerable de memoria. Utilice el almacenamiento en caché de mapas de bits para el contenido vectorial complejo, como los degradados complejos o el texto.

La activación del almacenamiento en caché de mapas de bits para un objeto animado que contiene gráficos vectoriales complejos (por ejemplo, texto o degradados) mejora el rendimiento. Sin embargo, si el almacenamiento en caché de mapas de bits está activado en un objeto de visualización como un clip de película cuya línea de tiempo se está reproduciendo, se obtiene el resultado contrario. En cada fotograma, Flash Player debe actualizar el mapa de bits en caché y posteriormente redibujarlo en pantalla, lo que requiere diversos ciclos de CPU. La función de almacenamiento en caché de mapas de bits sólo supone una ventaja cuando el mapa de bits en caché se puede generar una vez y posteriormente utilizarse sin necesidad de actualizarlo.

Si se activa el almacenamiento en caché de mapas de bits para un objeto Sprite, el objeto se puede mover sin hacer que Flash Player vuelva a generar el mapa de bits en caché. El cambio de las propiedades `x` e `y` del objeto no implica una regeneración. No obstante, cualquier intento de rotarlo, escalarlo o cambiar su valor alfa hace que Flash Player vuelva a generar el mapa de bits en caché y, como resultado, se daña el rendimiento.

Nota: *la propiedad* `DisplayObject.cacheAsBitmapMatrix` *disponible en AIR o Packager for iPhone Preview no presenta esta limitación. Con el uso de la propiedad* `cacheAsBitmapMatrix`*, un objeto se puede girar o escalar sin ninguna regeneración de mapa de bits. Esta propiedad garantiza que la GPU se utilice para representar el mapa de bits en pantalla, lo que proporciona mejoras de rendimiento.*

Un mapa de bits en caché utiliza más memoria que una instancia de clip de película normal. Por ejemplo, si el clip de película del escenario es 250 x 250 píxeles, cuando se almacena en caché utiliza alrededor de 250 KB, en lugar de 1 KB sin almacenarse en caché.

El siguiente ejemplo incluye un objeto Sprite que contiene una imagen de una manzana. La siguiente clase se añade al símbolo de la manzana:

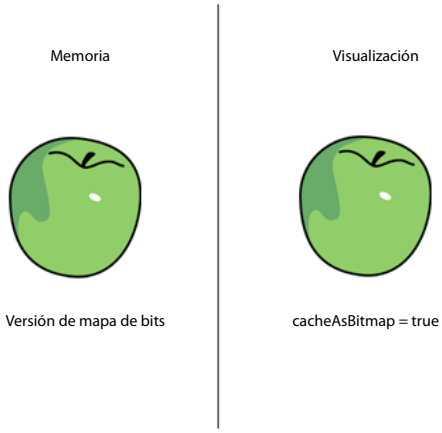
```
package org.bytearray.bitmap
{
    import flash.display.Sprite;
    import flash.events.Event;

    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();
        }
    }
}
```

El código utiliza la clase Sprite en lugar de MovieClip porque la línea de tiempo no es necesaria para cada manzana. Para obtener un mejor rendimiento, utilice el objeto más ligero posible. A continuación, se crea una instancia de la clase con el siguiente 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);
        }
    }
}
```

Cuando el usuario hace clic con el ratón, las manzanas se crean sin almacenamiento en caché. Cuando el usuario presiona la tecla C (código de tecla 67), los vectores de la manzana se almacenan en caché como mapas de bits y se muestran en pantalla. Esta técnica mejora en gran medida el rendimiento de la representación, tanto en el escritorio como en los dispositivos móviles, cuando la CPU es lenta.

No obstante, aunque el uso de la función de almacenamiento en caché de mapas de bits mejora el rendimiento de representación, puede consumir con rapidez grandes cantidades de memoria. Cuando un objeto se almacena en caché, su superficie se captura como un mapa de bits transparente () y se almacena en memoria, tal y como se muestra en el diagrama siguiente:

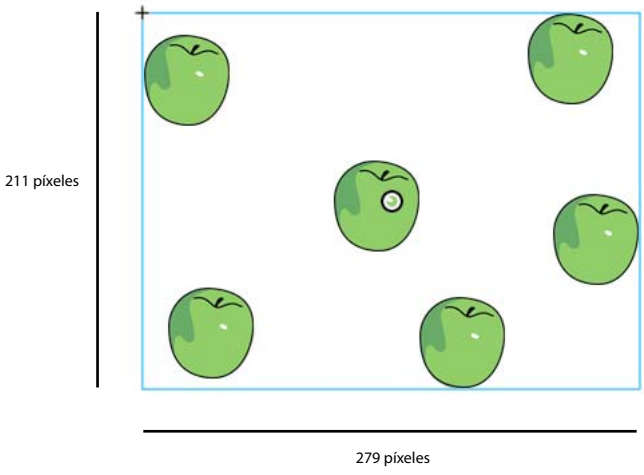


Objeto y su mapa de bits de superficie almacenado en memoria

Flash Player 10.1 optimiza el uso de la memoria adoptando el mismo enfoque, tal y como se describe en "Filtros y descarga dinámica de mapas de bits" en la página 19. Si un objeto de visualización en caché está oculto o fuera de pantalla, su mapa de bits en memoria se libera cuando no se utiliza durante un tiempo.

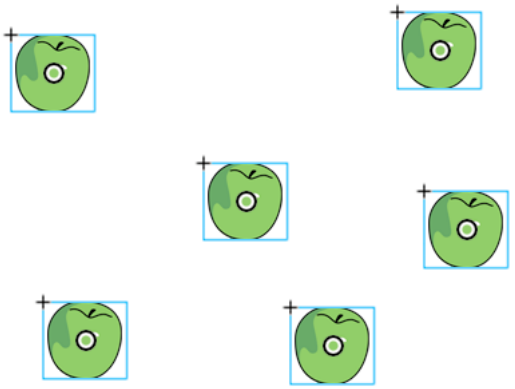
Nota: *si la propiedad* `opaqueBackground` *del objeto de visualización se establece en un color específico, Flash Player considera que el objeto de visualización es opaco. Cuando se utiliza con la propiedad* `cacheAsBitmap`*, Flash Player crea un mapa de bits de 32 bits no transparente en memoria. El canal alfa se establece en 0xFF, lo cual mejora el rendimiento, ya que no se requiere ninguna transparencia para dibujar el mapa de bits en pantalla. Al evitar la mezcla alfa se agiliza aún más la representación. Si la profundidad de pantalla actual se limita a 16 bits, el mapa de bits en memoria se almacena como una imagen de 16 bits. La utilización de la propiedad* `opaqueBackground` *no activa implícitamente el almacenamiento en caché de mapas de bits.*

Para ahorrar memoria, utilice la propiedad `cacheAsBitmap` y actívela en cada objeto de visualización en lugar de en el contenedor. La activación del almacenamiento en caché de mapas de bits en el contenedor hace que el mapa de bits final presente un tamaño mucho mayor en memoria, creando un mapa de bits transparente con dimensiones de 211 x 279 píxeles. La imagen utiliza unos 229 KB de memoria:



Activación del almacenamiento en caché de mapas de bits en el contenedor

Asimismo, al almacenar en caché el contenedor, se arriesga a tener todo el mapa de bits actualizado en memoria, si alguna manzana comienza a moverse en un fotograma. La activación del almacenamiento en caché de mapas de bits en instancias independientes implica el almacenamiento en caché de seis superficies de 7-KB en memoria, que únicamente utiliza 42 KB de memoria:



Activación del almacenamiento en caché de mapas de bits en instancias

Al acceder a cada instancia de la manzana mediante la lista de visualización y llamar al método `getChildAt()`, las referencias se almacenan en un objeto Vector para un acceso más 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);
        }
    }
}
```

Se debe tener en cuenta que el almacenamiento en caché de mapas de bits mejora la representación si el contenido en caché no se gira, se escala o se modifica en cada fotograma. Sin embargo, para cualquier transformación distinta de la traslación de los ejes x e y, la representación no mejora. En estos casos, Flash Player actualiza la copia del mapa de bits almacenada en caché para cada transformación que se produzca en el objeto de visualización. La actualización de la copia en caché puede implicar una mayor utilización de la CPU, bajo rendimiento y mayor uso de la batería. Una vez más, la propiedad `cacheAsBitmapMatrix` en AIR o Packager for iPhone Preview no presenta esta limitación.

El siguiente código cambia el valor alfa en el método de movimiento, lo que modifica la opacidad de la manzana en cada fotograma:

```
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();
}
```

El uso del almacenamiento en caché de mapas de bits implica una disminución del rendimiento. El cambio del valor alfa provoca que Flash Player actualice el mapa de bits almacenado en caché en memoria siempre que se modifique el valor alfa.

Los filtros se basan en los mapas de bits que se actualizan siempre que se mueva la cabeza lectora de un clip de película almacenado en caché. Por lo tanto, la utilización de un filtro automáticamente establece la propiedad `cacheAsBitmap` en `true`. La siguiente figura ilustra un clip de película animado:

Evite la utilización de los filtros en contenido animado, ya que se pueden producir problemas de rendimiento. En la siguiente figura, el diseñador añade un filtro de sombra:

Como resultado, si la línea de tiempo dentro del clip de película se está reproduciendo, el mapa de bits se debe regenerar. El mapa de bits también debe regenerarse si el contenido se modifica de cualquier forma distinta a una simple transformación x o y. Cada fotograma provoca que Flash Player vuelva a dibujar el mapa de bits, lo que requiere más recursos de CPU, causa problemas de bajo rendimiento y consume más duración de la batería.

Almacenamiento en caché manual de mapas de bits

Use la clase BitmapData para crear un comportamiento personalizado de almacenamiento en caché de mapas de bits.

El siguiente ejemplo reutiliza una sola versión del mapa de bits rasterizado de un objeto de visualización y hace referencia al mismo objeto BitmapData. Al escalar cada objeto de visualización, el objeto BitmapData original en memoria no se actualiza y no se vuelve a dibujar. Con este enfoque se ahorran recursos de CPU y la ejecución de las aplicaciones es más rápida. Al escalar un objeto de visualización, el mapa de bits incluido se expande.

A continuación se incluye la clase `BitmapApple` actualizada:

```
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();
        }
    }
}
```

El valor alfa aún se modifica en cada fotograma. El siguiente código transmite el búfer de origen original a cada instancia de 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 solamente una pequeña cantidad de memoria, ya que únicamente se utiliza un solo mapa de bits almacenado en caché en memoria y se comparte por todas las instancias de `BitmapApple`. Asimismo, a pesar de las modificaciones realizadas en las instancias de `BitmapApple`, como alfa, la rotación y la escala, el mapa de bits de origen nunca se actualiza. Con la utilización de esta técnica se evita la ralentización del rendimiento.

Para un mapa de bits final suavizado, establezca la propiedad `smoothing` en `true`:

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

    smoothing = true;

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

El ajuste de la calidad del escenario también puede mejorar el rendimiento. Establezca la calidad del escenario en `HIGH` antes de la rasterización; posteriormente, cambie a `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 );
}
```

Alternar la calidad del escenario antes y después del dibujo del vector a un mapa de bits puede resultar una técnica eficaz para obtener contenido suavizado en pantalla. Esta técnica puede ser eficaz independientemente de la calidad final del escenario. Por ejemplo, se puede obtener un mapa de bits suavizado con texto suavizado, aunque la calidad del escenario se establezca en `LOW`. Esa técnica no es posible con la propiedad `cacheAsBitmap`. En ese caso, el establecimiento de la calidad del escenario en `LOW` actualiza la calidad del vector, con lo cual se actualizan las superficies del mapa de bits en memoria, así como la calidad final.

Aislamiento de comportamientos

Aísle eventos como, por ejemplo, `Event.ENTER_FRAME` en un solo controlador, si es posible.

El código se puede optimizar aún más aislando el evento `Event.ENTER_FRAME` de la clase `Apple` en un solo controlador. Con esta técnica se ahorran recursos de CPU. El siguiente ejemplo muestra este enfoque diferente, donde la clase BitmapApple no vuelve a administrar el comportamiento del movimiento:

```
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;
        }
}
```

El siguiente código crea instancias de las manzanas y controla su movimiento en un solo controlador:

```
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.alpha = Math.random();
            bitmapApple = holderVector[i];

            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;
            }
        }
}
```

El resultado es un solo evento `Event.ENTER_FRAME` que controla el movimiento, en lugar de 200 controladores moviendo cada manzana. Toda la animación se puede detener fácilmente, lo cual puede ser útil en un juego.

Por ejemplo, un sencillo juego puede utilizar el siguiente controlador:

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

El siguiente paso consiste en hacer que las manzanas interactúen con el ratón o el teclado, lo que requiere una modificación en la clase `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);
        }
}
```

El resultado son instancias de `BitmapApple` interactivas, como objetos Sprite tradicionales. Sin embargo, las instancias se vinculan a un solo mapa de bits, que no se vuelve a muestrear cuando se transforman los objetos de visualización.

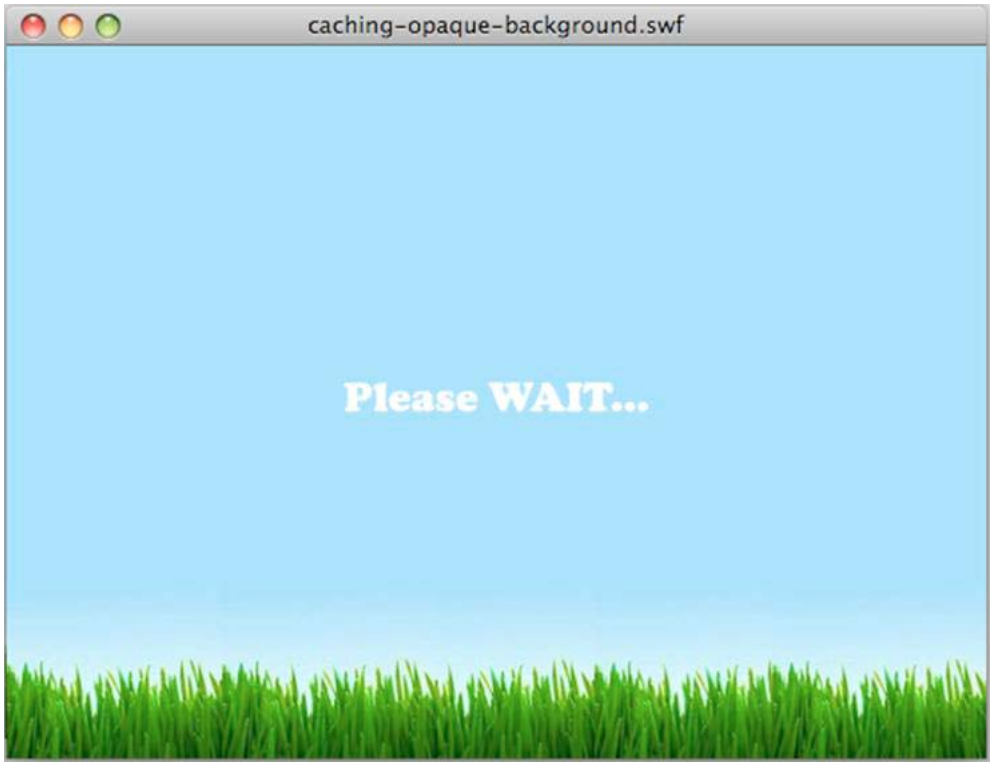
Representación de objetos de texto

Utilice la función de almacenamiento en caché de mapas de bits y la propiedad `opaqueBackground` para mejorar el rendimiento de la representación de texto.

El nuevo motor de texto de Flash Player10 proporciona algunas optimizaciones excelentes. No obstante, se requieren numerosas clases para mostrar una sola línea de texto. Por este motivo, la creación de un campo de texto editable con la clase TextLine requiere una gran cantidad de memoria y muchas líneas de código ActionScript. La clase TextLine es más adecuada para el texto estático y no editable, con el que la representación es más rápida y se requiere menos memoria.

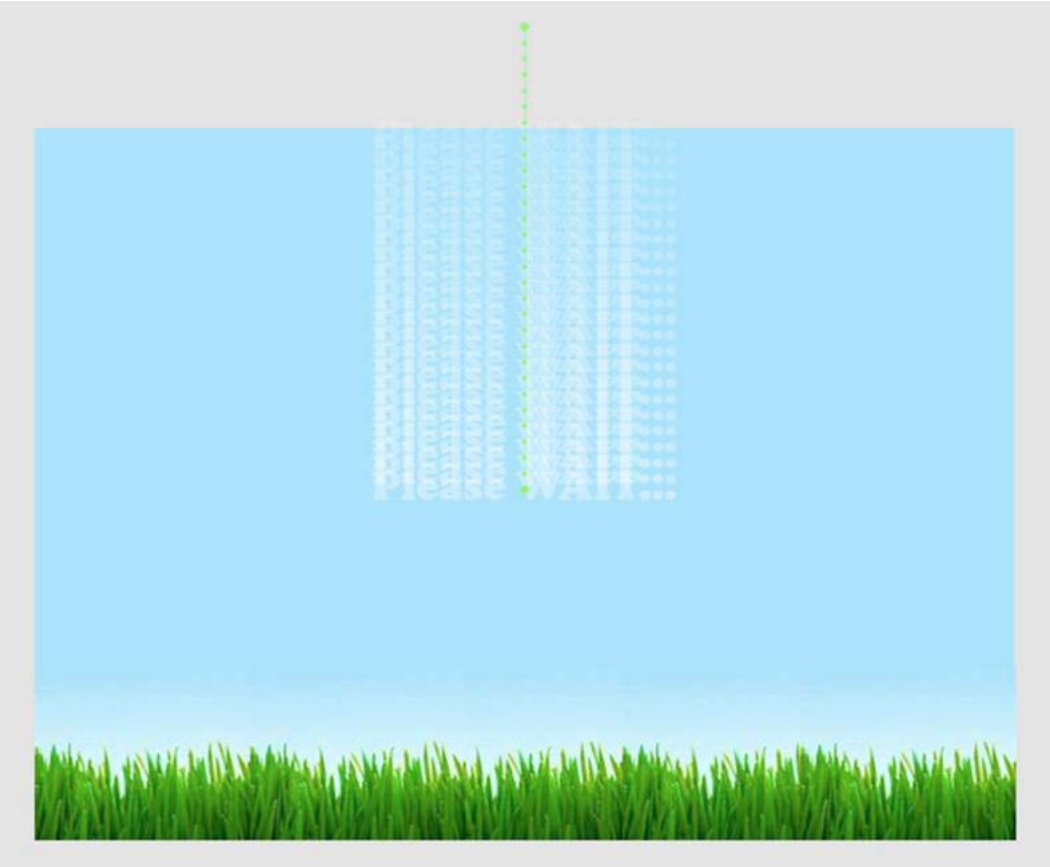
La función de almacenamiento en caché de mapas de bits permite almacenar en caché el contenido vectorial como mapas de bits para mejorar el rendimiento de la representación. Esta función resulta útil para el contenido vectorial complejo y también cuando se utiliza con contenido de texto que requiere que el procesamiento se represente.

El siguiente ejemplo muestra el modo en que la función de almacenamiento en caché de mapas de bits y la propiedad `opaqueBackground` se pueden utilizar para mejorar el rendimiento de la representación. La siguiente figura ilustra una típica pantalla de bienvenida que se puede mostrar cuando un usuario está esperando que se cargue algún elemento:



pantalla de bienvenida

La siguiente figura ilustra el suavizado que se aplica al objeto TextField mediante programación. El texto se suaviza lentamente desde la parte superior de la escena hasta el centro:



Suavizado del texto

El siguiente código crea el suavizado. La variable `preloader` almacena el objeto de destino actual para reducir las búsquedas de propiedades, lo cual puede dañar el rendimiento:

```
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 );
}
```

La función `Math.abs()` se puede mover en línea aquí para reducir el número de llamadas a funciones y lograr más mejoras de rendimiento. Se recomienda el uso del tipo int para las propiedades `destX` y `destY`, de modo que se disponga de valores de punto fijo. El uso del tipo int permite obtener un ajuste de píxeles perfecto sin tener que redondear valores manualmente a través de métodos lentos, como `Math.ceil()` o `Math.round()`. Este código no redondea las coordenadas a int, ya que al redondear los valores constantemente, el objeto no se mueve con suavidad. El objeto puede tener movimientos de temblor, ya que las coordenadas se ajustan a los enteros redondeados más cercanos en cada fotograma. Sin embargo, esta técnica puede resultar útil al establecer una posición final para un objeto de visualización. No utilice el siguiente código:

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

El siguiente código es mucho más rápido:

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

El código anterior puede optimizarse aún más, utilizando operadores de desplazamiento en modo bit para dividir los valores:

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

La función de almacenamiento en caché de mapas de bits facilita a Flash Player la representación de objetos mediante el uso de mapas de bits dinámicos. En el ejemplo actual, el clip de película que contiene el objeto TextField se almacena en caché:

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

Una forma adicional de mejorar el rendimiento consiste en eliminar la transparencia alfa. La transparencia alfa supone una carga adicional para Flash Player al dibujar imágenes de mapas de bits transparentes, tal y como se muestra en el código anterior. La propiedad `opaqueBackground` se puede utilizar para omitir este punto, especificando un color como fondo.

Al utilizar la propiedad `opaqueBackground`, la superficie del mapa de bits creada en memoria aún utiliza 32 bits. Sin embargo, el desplazamiento alfa se establece en 255 y no se utiliza la transparencia. Como resultado, la propiedad `opaqueBackground` no reduce el uso de memoria, pero mejora el rendimiento de representación al utilizar la función de almacenamiento en caché de mapas de bits. El siguiente código contiene todas las optimizaciones:

```
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 );
}
```

La animación se optimiza y el almacenamiento en caché de mapas de bits se ha optimizado eliminando la transparencia. Considere el cambio de la calidad del escenario a `LOW` y `HIGH` durante los diferentes estados de la animación mientras se utiliza la función de almacenamiento en caché de mapas de bits:

```
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
        stage.quality = StageQuality.HIGH;
        preloader.removeEventListener( Event.ENTER_FRAME, movePosition );
    }
}
```

No obstante, en este caso, el cambio de la calidad del escenario provoca que Flash Player vuelva a generar la superficie del mapa de bits del objeto TextField para coincidir con la calidad del escenario actual. Por este motivo, es mejor no modificar la calidad del escenario al utilizar la función de almacenamiento en caché de mapas de bits.

Aquí podría haberse empleado un enfoque de almacenamiento en caché de mapas de bits manual. Para simular la propiedad `opaqueBackground`, el clip de película se puede dibujar en un objeto BitmapData transparente, que no provoque que Flash Player vuelva a generar la superficie del mapa de bits.

Esta técnica funciona bien para el contenido que no cambia con el tiempo. Sin embargo, si el campo de texto puede cambiar, considere el uso de una estrategia distinta. Por ejemplo, imagine un campo de texto que se actualiza continuamente con un porcentaje que representa la parte de la aplicación que se ha cargado. Si el campo de texto, o su objeto de visualización, se ha almacenado en caché como mapa de bits, su superficie se debe generar cada vez que cambia el contenido. Aquí no se puede utilizar el almacenamiento en caché de mapas de bits manual, ya que el contenido del objeto de visualización cambia constantemente. Este cambio constante puede forzar la llamada manual al método `BitmapData.draw()` para actualizar el mapa de bits en caché.

Recuerde que desde Flash Player 8, independientemente del valor de la calidad del escenario, un campo de texto con la representación establecida en Suavizado para legibilidad permanece perfectamente suavizado. Con este enfoque se utiliza menos memoria, pero se requiere más procesamiento de CPU y la representación es algo más lenta que la función de almacenamiento en caché de mapas de bits.

El siguiente código utiliza este enfoque:

```
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 );
    }
}
```

No se recomienda el uso de esta opción (Suavizado para legibilidad) para texto en movimiento. Al escalar el texto, esta opción hace que el texto intente permanecer alineado, lo que produce un efecto de desplazamiento. Sin embargo, si el contenido del objeto de visualización cambia constantemente y se necesita texto escalado, el rendimiento puede mejorar estableciendo la calidad en `LOW`. Cuando finalice el movimiento, vuelva a cambiar la calidad a `HIGH`.

GPU

Una nueva e importante función de Flash Player 10.1 es que es posible utilizar la GPU para representar el contenido gráfico en dispositivos móviles. Anteriormente, los gráficos se representaban únicamente mediante la CPU. Con el uso de la GPU se optimiza la representación de filtros, mapas de bits, vídeo y texto. Se debe tener en cuenta que la representación mediante GPU no siempre es tan precisa como la representación con software. El contenido puede parecer algo irregular cuando se utiliza el representador de hardware. Asimismo, Flash Player 10.1 tiene una limitación que puede evitar la representación de los efectos de Pixel Bender en pantalla. Estos efectos se pueden representar como un cuadrado negro al utilizar la aceleración de hardware.

Flash Player 10 contaba con la función de aceleración de GPU, pero la GPU no se utilizaba para calcular los gráficos. Sólo se empleaba para enviar todos los gráficos a la pantalla. En Flash Player 10.1, la GPU se utilizaba para calcular los gráficos, lo cual puede mejorar la velocidad de representación en gran medida. También reduce la carga de CPU, lo que resulta útil en dispositivos con recursos limitados, tales como dispositivos móviles.

El reproductor de escritorio aún utiliza la representación con software para esta versión. La representación con software se utiliza porque los controladores pueden variar mucho en el escritorio y los mismos pueden acentuar las diferencias de representación. También puede haber diferencias entre el escritorio y algunos dispositivos móviles. El modo GPU se establece automáticamente cuando el contenido se ejecuta en dispositivos móviles con el fin de obtener el mejor rendimiento posible.

***Nota:** aunque `wmode` no se debe volver a establecer en `gpu` para obtener la representación con GPU, el establecimiento de `wmode` en `opaque` o `transparent` desactiva la aceleración por GPU.*

La representación con GPU funciona de forma diferente al utilizar Packager for iPhone Preview. La GPU sólo se utiliza para mapas de bits, formas sólidas y objetos de visualización que tienen definida la propiedad `cacheAsBitmap`. Asimismo, para los objetos que tienen establecido `cacheAsBitmap` y `cacheAsBitmapMatrix`, la GPU puede representar objetos que giran o se escalan.

Operaciones asíncronas

Utilice las versiones asíncronas de las operaciones en lugar de las sincrónicas, siempre que sea posible.

Las operaciones sincrónicas se ejecutan cuando el código se lo indica y el código espera a que se completen antes de continuar. Por lo tanto, se ejecutan en la fase del código de la aplicación del bucle de fotograma. Si una operación sincrónica tarda demasiado, amplía el tamaño del bucle de fotograma, por lo que es probable que la visualización presente bloqueos o irregularidades.

Cuando el código ejecuta una operación asíncrona, no necesariamente se ejecuta de forma inmediata. Su código y el código de la aplicación del subproceso de ejecución actual continúa ejecutándose. El motor de ejecución realiza la operación lo antes posible mientras intenta evitar problemas de representación. En algunos casos, la ejecución se produce en segundo plano y no se ejecuta como parte del bucle de fotograma del motor de ejecución. Finalmente, una vez completada la operación, el motor de ejecución distribuye un evento y su código puede detectar ese evento para seguir ejecutando más tareas.

Las operaciones asíncronas se programan y se dividen para evitar problemas de representación. Por lo tanto, resulta mucho más sencillo disponer de una aplicación con un buen nivel de respuesta utilizando versiones asíncronas de las operaciones. Consulte "Rendimiento percibido frente a rendimiento real" en la página 2 para obtener más información.

Sin embargo, existe cierta sobrecarga asociada a la ejecución de operaciones de forma asíncrona. El tiempo real de ejecución puede ser mayor en las operaciones asíncronas, especialmente en aquellas operaciones que tardan poco tiempo en completarse.

En el motor de ejecución de la plataforma Flash, diversas operaciones son sincrónicas o asíncronas de forma inherente y no se puede seleccionar cómo ejecutarlas. Sin embargo, en Adobe AIR, existen tres tipos de operaciones para las que se puede optar por una ejecución sincrónica o asíncrona:

- Operaciones de la clase File y FileStream

  Muchas operaciones de la clase File se pueden llevar a cabo de forma sincrónica o asíncrona. Por ejemplo, los métodos para copiar o eliminar un archivo o directorio y el listado del contenido de un directorio tienen todos versiones asíncronas. Estos métodos se identifican mediante el sufijo "Async" añadido al nombre de la versión asíncrona. Por ejemplo, para eliminar un archivo de forma asíncrona, llame al método `File.deleteFileAsync()` en lugar del método `File.deleteFile()`.

  Cuando se utiliza un objeto FileStream para leer o escribir en un archivo, el modo en que se abre el objeto determina si las operaciones de lectura y escritura se ejecutan de forma asíncrona. Utilice el método `FileStream.openAsync()` para las operaciones asíncronas. La escritura de datos se realiza de forma asíncrona. La lectura de datos se realiza en segmentos, por lo que los datos están disponibles una parte cada vez. Por el contrario, en el modo sincrónico, el objeto FileStream lee todo el archivo antes de continuar con la ejecución de código.

- Operaciones de base de datos SQL local

Al trabajar con una base de datos SQL local, todas las operaciones ejecutadas mediante un objeto SQLConnection se ejecutan en modo sincrónico o asíncrono. Para especificar que las operaciones se ejecuten de forma asíncrona, abra la conexión a la base de datos utilizando el método `SQLConnection.openAsync()` en lugar del método `SQLConnection.open()`. Si las operaciones de base de datos se ejecutan de forma asíncrona, lo harán en segundo plano. El motor de la base de datos no se ejecuta en el bucle de fotograma del motor de ejecución, por lo que es mucho menos probable que las operaciones de base de datos causen problemas de representación.

Para obtener estrategias adicionales para mejorar el rendimiento con la base de datos SQL local, consulte "Rendimiento de la base de datos SQL" en la página 82.

- Sombreados independientes de Pixel Bender

  La clase ShaderJob permite ejecutar una imagen o un conjunto de datos mediante un sombreado de Pixel Bender y acceder a los datos de resultado sin formato. De forma predeterminada, cuando se llama al método `ShaderJob.start()`, el sombreado se ejecuta de forma asíncrona. La ejecución se produce en segundo plano y no se utiliza el bucle de fotograma del motor de ejecución. Para provocar que el objeto ShaderJob se ejecute sincrónicamente (lo cual no se recomienda), transmita el valor `true` al primer parámetro del método `start()`.

  Para obtener más información y ejemplos del uso de un sombreado de Pixel Bender para el procesamiento en segundo plano, consulte el artículo de Elad Elrom Using Pixel Bender to do heavy lifting calculations, makes Flash Player multi-thread (El uso de Pixel Bender para realizar cálculos complejos implica subprocesos múltiples en Flash Player; en inglés).

Además de estos mecanismos incorporados para la ejecución asíncrona de código, es posible estructurar el propio código para realizar la ejecución de forma asíncrona en lugar de sincrónicamente. Si se está escribiendo código para realizar tareas de una posible ejecución larga, el código puede estructurarse para que se ejecute en partes. La división del código en partes permite al motor de ejecución realizar sus operaciones de representación entre los bloques de ejecución de código, lo que reduce la posibilidad de que aparezcan problemas de representación.

A continuación se indican varias técnicas para la división del código. La idea principal de estas técnicas radica en que el código se escribe para realizar únicamente parte de su trabajo en cualquier momento. Se realiza un seguimiento de lo que hace el código y del momento en que deja de trabajar. Por ejemplo, se utiliza un mecanismo como el siguiente: un objeto Timer comprueba reiteradamente si el trabajo continúa y realiza tareas adicionales en segmentos hasta que el trabajo finalice.

Existen unos cuantos patrones establecidos para estructurar el código y dividir el trabajo de este modo. Los siguientes artículos y bibliotecas de código describen estos patrones y proporcionan código para ayudarle a implementarlos en sus aplicaciones:

- Asynchronous ActionScript Execution (Ejecución asíncrona de ActionScript; en inglés.) (Artículo de Trevor McCauley con más información general, así como varios ejemplos de implementación.)
- Parsing & Rendering Lots of Data in Flash Player (Análisis y representación de muchos datos en Flash Player; en inglés.) (Artículo de Jesse Warden con información general y ejemplos, "builder pattern" y "green threads").
- Green Threads (Artículo de Drew Cummins donde se describe la técnica "green threads" con código fuente de ejemplo; en inglés).
- greenthreads (Biblioteca de código fuente abierto de Charlie Hubbard, para la implementación de "green threads" en ActionScript; en inglés. Para obtener más información, consulte la guía de inicio rápido sobre greenthreads greenthreads Quick Start .)
- Subprocesos en ActionScript 3 http://www.adobe.com/go/learn_fp_as3_threads_es (Artículo (en inglés) de Alex Harui, donde se incluye una implementación de ejemplo de la técnica "pseudo threading").

Ventanas transparentes

En las aplicaciones de AIR, considere el uso de una ventana de aplicación rectangular opaca en lugar de una ventana transparente.

Para utilizar una ventana opaca, para la ventana inicial de la aplicación, establezca el siguiente valor en el archivo XML descriptor de la aplicación:

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

Para las ventanas creadas mediante código de la aplicación, cree un objeto NativeWindowInitOptions con la propiedad `transparent` establecida en `false` (valor predeterminado). Transmítalo al constructor NativeWindow mientras que se crea el objeto NativeWindow:

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

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

Para un componente Window de Flex, asegúrese de que la propiedad transparent del componente se establezca en false (valor predeterminado), antes de llamar al método `open()` del objeto Window.

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

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

Una ventana transparente mostrará parte del escritorio del usuario u otras ventanas de la aplicación detrás de la ventana de la aplicación. Por lo tanto, el motor de ejecución utiliza muchos menos recursos para representar una ventana transparente. Una ventana no transparente rectangular, ya utilice un fondo cromático personalizado o del sistema operativo, no tiene la misma carga de representación.

Utilice una ventana transparente sólo cuando sea importante disponer de una visualización no rectangular o si el contenido de fondo se debe representar a través de la ventana de la aplicación.

Capítulo 6: Optimización de la interacción de red

Mejoras de Flash Player 10.1 para la interacción de red

Flash Player 10.1 incorpora un conjunto de nuevas funciones para la optimización de red en todas las plataformas, incluyendo el almacenamiento en búfer circular y la búsqueda inteligente.

Almacenamiento en búfer circular

Al cargar contenido multimedia en los dispositivos móviles, se pueden encontrar problemas que nunca se esperarían en un equipo de escritorio. Por ejemplo, es más probable quedarse sin espacio en disco o memoria. Al cargar vídeo, la versión de escritorio de Flash Player 10.1 descarga y almacena en caché todo el archivo FLV (o archivo MP4) en el disco duro. A continuación, reproduce el vídeo desde ese archivo de caché. No es frecuente quedarse sin espacio en disco. Si esto sucede, el reproductor de escritorio detiene la reproducción del vídeo.

Un dispositivo móvil puede quedarse sin espacio en disco con más fácilmente. Si el dispositivo no dispone de espacio en disco, Flash Player no detiene la reproducción, tal y como sucede en el reproductor de escritorio. Flash Player comienza a reutilizar el archivo de caché escribiendo en el mismo de nuevo desde el principio del archivo. El usuario puede continuar viendo el vídeo. El usuario no puede buscar en el área del vídeo que se ha reescrito, excepto en el principio del archivo. El almacenamiento en búfer circular no se inicia de forma predeterminada. Se puede iniciar durante la reproducción y también al principio de la reproducción, si la película presenta un tamaño mayor que el espacio en disco o RAM. Flash Player requiere 4 MB de RAM como mínimo o 20 MB de espacio en disco para poder utilizar el almacenamiento en búfer circular.

Nota: *si el dispositivo dispone de suficiente espacio en disco, la versión móvil de Flash Player se comporta del mismo modo en el escritorio. Se debe tener en cuenta que un búfer en RAM se utiliza como reserva si el dispositivo no tiene un disco o éste está lleno. En tiempo de compilación, se puede establecer un límite para el tamaño del archivo de caché y el búfer de RAM. Algunos archivos MP4 tienen una estructura que requiere que se descargue todo el archivo antes de que pueda comenzar la reproducción. Flash Player detecta estos archivos y evita la descarga, si no hay suficiente espacio en disco, y el archivo MP4 no puede reproducirse. Puede ser más adecuado no solicitar la descarga de esos archivos.*

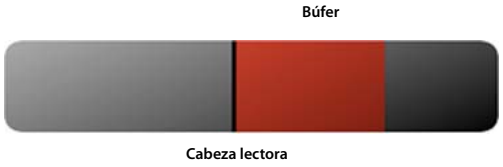
Como desarrollador, recuerde que la búsqueda sólo funciona en el límite del flujo almacenado en caché. En ocasiones, `NetStream.seek()` genera un error si está fuera de rango y, en este caso, se distribuye un evento `NetStream.Seek.InvalidTime`.

Búsqueda inteligente

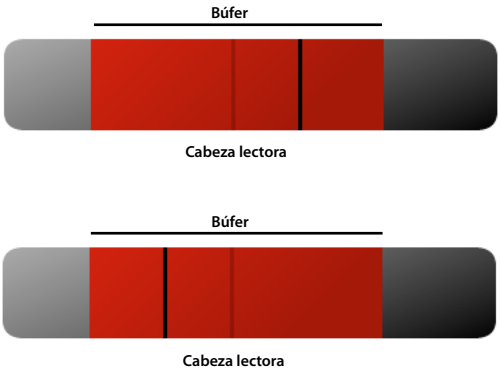
Nota: *la función de búsqueda inteligente requiere Adobe® Flash® Media Server 3.5.3.*

Flash Player 10.1 introduce un nuevo comportamiento, denominado búsqueda inteligente, que mejora la experiencia del usuario al reproducir secuencias de vídeo. Si el usuario busca un destino dentro de los límites del búfer, Flash Player vuelve a utilizar el búfer para ofrecer una búsqueda instantánea. En versiones anteriores de Flash Player, el búfer no se reutilizaba. Por ejemplo, si un usuario estaba reproduciendo un vídeo desde un servidor de secuencias y el tiempo de búfer era de 20 segundos (`NetStream.bufferTime`) y el usuario intentaba buscar 10 segundos más adelante, Flash Player podría malgastar todos los datos del búfer en lugar de reutilizar los 10 segundos ya cargados. Este comportamiento provocaba que Flash Player solicitara nuevos datos del servidor con mucha más frecuencia, así como un menor rendimiento de la reproducción en conexiones lentas.

La siguiente figura ilustra el modo en que se comportaba el búfer en la versión anterior de Flash Player. La propiedad `bufferTime` especifica el número de segundos de carga previa hacia delante de modo que si la conexión falla, el búfer se puede utilizar sin detener el vídeo:



Flash Player 10.1 introduce un nuevo comportamiento de búsqueda. Flash Player utiliza ahora el búfer para proporcionar una búsqueda instantánea hacia delante o hacia atrás cuando el usuario borre el vídeo. La siguiente figura ilustra el nuevo comportamiento:



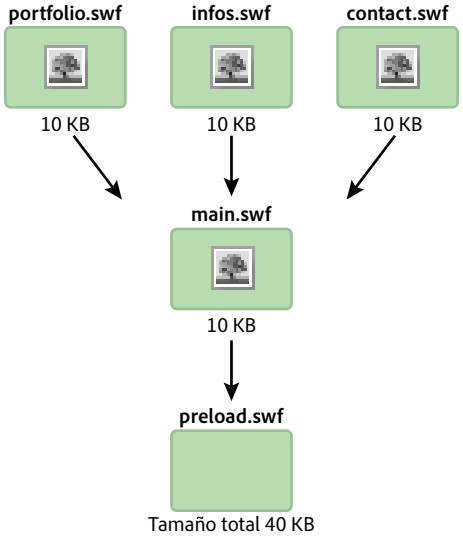
Con la búsqueda inteligente se reutiliza el búfer cuando el usuario busca hacia atrás o hacia delante, de modo que la experiencia de reproducción resulte más fácil y rápida. Una de las ventajas de este nuevo comportamiento es el ahorro de ancho de banda para los editores de vídeo. Sin embargo, si la búsqueda se realiza fuera de los límites del búfer, se aplicará el comportamiento estándar y Flash Player solicitará nuevos datos al servidor.

Nota: *este comportamiento no se aplica a la descarga progresiva de vídeo.*

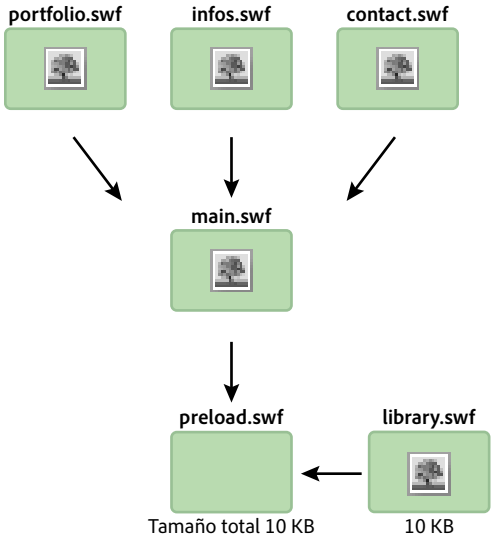
Contenido externo

Divida la aplicación en varios archivos SWF.

Los dispositivos móviles pueden tener acceso limitado a la red. Para cargar el contenido rápidamente, divida la aplicación en varios archivos SWF. Intente reutilizar la lógica y los recursos en toda la aplicación. Por ejemplo, considere una aplicación que se haya dividido en varios archivos SWF, tal y como se muestra en el siguiente diagrama:



En este ejemplo, cada archivo SWF contiene su propia copia del mismo mapa de bits. Esta duplicación se puede evitar utilizando una biblioteca compartida en tiempo de ejecución, tal y como se muestra en el siguiente diagrama:



Con el uso de esta técnica, una biblioteca compartida en tiempo de ejecución se carga para que el mapa de bits esté disponible para los demás archivos SWF. La clase ApplicationDomain almacena las definiciones de clase que se hayan cargado y las hace disponibles en tiempo de ejecución a través del método `getDefinition()`.

Una biblioteca compartida en tiempo de ejecución también puede incluir toda la lógica del código. Toda la aplicación puede actualizarse en tiempo de ejecución sin tener que volver a realizar la compilación. El siguiente código carga una biblioteca compartida en tiempo de ejecución y extrae la definición incluida en el archivo SWF en tiempo de ejecución. Esta técnica se puede utilizar con fuentes, mapas de bits, sonidos o cualquier clase de 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.");
}
```

La obtención de la definición se puede facilitar cargando las definiciones de clase en el dominio de la aplicación del archivo SWF de carga:

```
// 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.");
}
```

Las clases disponibles en este momento en el archivo SWF cargado se pueden utilizar llamando al método `getDefinition()` en el dominio de la aplicación actual. También se puede acceder a las clases llamando al método `getDefinitionByName()`. Con esta técnica se ahorra ancho de banda cargando fuentes y recursos de gran tamaño sólo una vez. Los recursos no se exportan nunca en otros archivos SWF. La única limitación es que la aplicación se debe probar y ejecutar mediante el archivo loader.swf. Este archivo carga los recursos en primer lugar y, a continuación, carga los archivos SWF diferentes que conforman la aplicación.

Errores de entrada y salida


Proporcione controladores de eventos y mensajes de error para los errores de E/S.

En un dispositivo móvil, la red puede ser menos fiable que un equipo de escritorio conectado a Internet de alta velocidad. El acceso a contenido externo en los dispositivos móviles cuenta con dos limitaciones: disponibilidad y velocidad. Por lo tanto, asegúrese de que los recursos sean ligeros y añada controladores para todos los eventos IO_ERROR para proporcionar información al usuario.

Por ejemplo, imagine que un usuario está navegando en su sitio web en un teléfono móvil y de repente se pierde la conexión de la red entre dos estaciones de metro. Se estaba cargando un dispositivo dinámico cuando la conexión se perdió. En el escritorio, se puede utilizar un detector de eventos vacío para evitar que aparezca un error en tiempo de ejecución, ya que es muy poco probable que suceda este escenario. Sin embargo, en un dispositivo móvil se necesita algo más que un simple detector de eventos.

El siguiente código no responde a un error de E/S. No lo utilice cuando se muestre:

```
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;
}
```

Es mejor administrar un error de este tipo y proporcionar un mensaje de error para el usuario. El siguiente código lleva a cabo una gestión adecuada:

```
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 práctica recomendada, recuerde ofrecer al usuario una forma de cargar de nuevo el contenido. Este comportamiento se puede implementar en el controlador `onIOError()`.

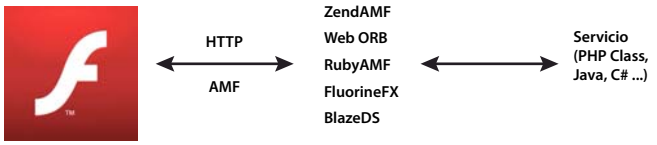
Flash Remoting

Utilice Flash Remoting y AMF para la comunicación optimizada de datos servidor-cliente.

XML se puede emplear para cargar contenido remoto en archivos SWF. No obstante, XML es texto sin formato que Flash Player carga y analiza. XML funciona mejor para las aplicaciones que cargan una cantidad limitada de contenido. Si está desarrollando una aplicación que carga una gran cantidad de contenido, considere el uso de la tecnología Flash Remoting y del formato de mensaje de acción (Action Message Format, AMF).

AMF es un formato binario utilizado para compartir datos entre un servidor y Flash Player. El uso de AMF reduce el tamaño de los datos y mejora la velocidad de transmisión. Debido a que AMF es un formato nativo para Flash Player, el envío de datos AMF a Flash Player evita la deserialización y la serialización de uso intensivo de memoria en el lado del cliente. La puerta de enlace remota administra estas tareas. Al enviar un tipo de datos de ActionScript a un servidor, la puerta de enlace remota administra la serialización en el lado del cliente. La puerta de enlace también envía el tipo de datos correspondiente. Este tipo de datos es una clase creada en el servidor que expone un conjunto de métodos que se pueden llamar desde Flash Player. Entre las puertas de enlace de Flash Remoting se incluyen ZendAMF, FluorineFX, WebORB y BlazeDS, una puerta de enlace Flash Remoting de Java de código fuente abierto oficial de Adobe.

La siguiente figura ilustra el concepto de Flash Remoting:



En el siguiente ejemplo se utiliza la clase NetConnection para conectarse a una puerta de enlace de 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 ?");
```

La conexión con una puerta de enlace remota es un proceso sencillo. No obstante, el uso de Flash Remoting se puede facilitar aún más utilizando la clase RemoteObject incluida en el SDK de Adobe® Flex®.

Nota: *los archivos SWC externos, como los abiertos desde la arquitectura de Flex, se pueden utilizar dentro de un proyecto de Adobe® Flash® Professional. El uso de los archivos SWC permite la utilización de la clase RemoteObject y sus dependencias sin emplear el resto del SDK de Flex. Los desarrolladores avanzados pueden incluso comunicarse con una puerta de enlace remota directamente mediante la clase Socket sin procesar, si es necesario.*

Operaciones de red innecesarias

Almacene en caché localmente los recursos tras cargarlos, en lugar de cargarlos desde la red cada vez que sea necesario.

Si la aplicación carga recursos como medios o datos, almacénelos en caché, guardándolos en el equipo local local. Para los recursos que cambien rara vez, considere la actualización de la caché en intervalos. Por ejemplo, la aplicación puede buscar una nueva versión de una imagen una vez al día, o bien, buscar datos actualizados una vez cada dos horas.

Los recursos se pueden almacenar en caché de varias formas, dependiendo del tipo y la naturaleza de los mismos:

- Recursos de medios como, por ejemplo, imágenes y vídeo: guarde los archivos en el sistema de archivos utilizando las clases File y FileStream.
- Valores de datos individuales o pequeños conjuntos de datos: guarde los valores como objetos compartidos locales utilizando la clase SharedObject.
- Grandes conjuntos de datos: guarde los datos en una base de datos local o serialícelos y guárdelos en un archivo.

Para el almacenamiento en caché de valores de datos, el proyecto AS3CoreLib de código fuente abierto incluye una clase ResourceCache que realiza las tareas de carga y almacenamiento en caché.

Capítulo 7: Trabajo con medios

Audio

Desde la versión 9.0.115.0, Flash Player puede reproducir archivos AAC (AAC Main, AAC LC y SBR). Se puede realizar una sencilla optimización utilizando archivos AAC en lugar de archivos MP3. El formato AAC ofrece mejor calidad y un tamaño de archivo más pequeño que el formato MP3 con una velocidad de bits equivalente. Al reducir el tamaño del archivo se ahorra ancho de banda, lo que representa un factor importante en los dispositivos móviles que no ofrecen conexiones a Internet de alta velocidad.

Descodificación de audio de hardware

Al igual que sucede con la descodificación de vídeo, el proceso de descodificación de audio requiere ciclos elevados de CPU y la optimización se puede llevar a cabo aprovechando el hardware disponible en el dispositivo. Flash Player 10.1 puede detectar y utilizar controladores de audio de hardware para mejorar el rendimiento al descodificar archivos AAC (perfiles LC, HE/SBR) o MP3 (PCM no se admite). El uso de la CPU se ve reducido en gran medida, lo que implica un uso mejor de la batería y hace que la CPU se encuentre disponible para otras operaciones.

Nota: *al utilizar el formato AAC, el perfil AAC Main no se admite en los dispositivos debido a la falta de compatibilidad de hardware en la mayoría de los dispositivos.*

La descodificación de audio de hardware es un proceso transparente para el usuario y el desarrollador. Cuando Flash Player comienza a reproducir flujos de audio, en primer lugar comprueba el hardware, al igual que lo hace con el vídeo. Si un controlador de hardware está disponible y se admite el formato de audio, se lleva a cabo la descodificación de audio de hardware. Sin embargo, aunque la descodificación del flujo AAC o MP3 entrante se puede administrar mediante el hardware, en algunas ocasiones el contenido de hardware no puede procesar todos los efectos. Por ejemplo, a veces el hardware no procesa el muestro y la mezcla de audio, dependiendo de las limitaciones de hardware.

Capítulo 8: Rendimiento de la base de datos SQL

Rendimiento de la base de datos SQL

Diseño de la aplicación para el rendimiento de la base de datos

No modifique la propiedad `text` *de un objeto SQLStatement tras su ejecución. En su lugar, utilice una instancia de SQLStatement para cada declaración SQL y use los parámetros de declaración para proporcionar diferentes valores.*

Antes de que se ejecute una declaración SQL, el motor de ejecución la prepara (compila) para determinar los pasos que se llevan a cabo internamente para realizar la declaración. Cuando llama al método `SQLStatement.execute()` en una instancia SQLStatement que no se ha ejecutado anteriormente, la declaración se prepara automáticamente antes de que se ejecute. En llamadas posteriores al método `execute()`, siempre que la propiedad `SQLStatement.text` no haya cambiado, aún se prepara la declaración. En consecuencia, se ejecuta de forma más rápida.

Para obtener el mayor beneficio de reutilizar las declaraciones, si se deben cambiar los valores entre cada ejecución de declaraciones, utilice los parámetros de declaración para personalizar la declaración. (Los parámetros de declaración se definen usando la propiedad de conjunto asociativa `SQLStatement.parameters`. A diferencia de cambiar la propiedad `text` de la instancia SQLStatement, si cambia los valores de los parámetros de declaración el motor de ejecución no necesita preparar la declaración nuevamente.

Cuando vuelve a utilizar una instancia SQLStatement, la aplicación necesita almacenar una referencia a la instancia SQLStatement una vez que se ha preparado. Para mantener una referencia a la instancia, declare la variable como una variable del ámbito de la clase en lugar de una variable en el ámbito de la función. Una buena manera para que SQLStatement sea una variable de ámbito de clase consiste en estructurar la aplicación para que la declaración SQL se agrupe en una sola clase. Un grupo de declaraciones que se ejecutan en combinación también se pueden agrupan en una sola clase. (Esta técnica se conoce como el uso del patrón de diseño Command.) Al definir las instancias como variables de miembros de la clase, éstas permanecen siempre que la instancia de la clase agrupada existe en la aplicación. Como mínimo, simplemente puede definir una variable que contiene la instancia SQLStatement fuera de una función para que la instancia permanezca en la memoria. Por ejemplo, declare la instancia SQLStatement como una variable miembro en una clase ActionScript o como una variable sin función en un archivo JavaScript. A continuación puede definir los valores de la declaración y llamar al método `execute()` cuando quiere realmente ejecutar la consulta.

Utilice los índices de base de datos para mejorar la velocidad de ejecución para la organización y comparación de información.

Cuando se crea un índice para una columna, la base de datos almacena una copia de los datos de esa columna. Dicha copia se ordena en orden numérico o alfabético. Esto permite que la base de datos relacione valores rápidamente (p. el. cuando se utiliza el operador de igualdad) y ordene los datos de resultados utilizando la cláusula `ORDER BY`.

Los í-ndices de la base de datos se mantienen actualizados continuamente, por lo que las operaciones de cambio de datos (INSERT o UPDATE) de esa tabla son algo más lentas. No obstante, el aumento de velocidad de recuperación de datos suele ser notable. Debido a este inconveniente menor de rendimiento, no se deben indexar simplemente todas las columnas de cada tabla. Como alternativa, utilice una estrategia para definir los índices. Haga uso de las siguientes indicaciones para planificar la estrategia de indización:

- Indexe las columnas que se utilicen en tablas de unión, en cláusulas WHERE o cláusulas ORDER BY.

- Si las columnas se utilizan con frecuencia al mismo tiempo, indéxelas de forma conjunta en un solo índice.
- Para una columna que contenga datos de texto que se recuperen de forma ordenada alfabéticamente, especifique la intercalación COLLATE NOCASE para el índice.


Considere la compilación previa de declaraciones SQL durante los tiempos de inactividad de la aplicación..

La primer vez que ejecute una declaración SQL, el proceso es más lento, ya que el texto SQL está preparado (compilado) por el motor de la base de datos. Dado que la preparación y la ejecución de una declaración puede ser una operación exigente, una estrategia es cargar previamente los datos iniciales y luego ejecutar las demás declaraciones en segundo plano:

1. Cargue primero los datos que necesita la aplicación
2. Cuando se completan las operaciones de inicio de la aplicación, o en otro momento de "inactividad" de la aplicación, ejecute otras declaraciones.

Por ejemplo, supongamos que la aplicación no accede a la base de datos para mostrar su ventana inicial. En este caso, espere hasta que la ventana se muestre antes de abrir la conexión de base de datos. Finalmente, cree instancias de SQLStatement y ejecute todas las que pueda.

Como alternativa, supongamos que cuando inicia la aplicación inmediatamente muestra algunos datos como el resultado de una determinada consulta. En ese caso, proceda a ejecutar la instancia SQLStatement para esa consulta. Después de que se cargan y se muestran los datos iniciales, cree las instancias SQLStatement para las otras operaciones de la base de datos y, si es posible, ejecute las otras declaraciones que se necesitan más adelante.

En la práctica, si está reutilizando instancias de SQLStatement, el tiempo adicional necesario para preparar la declaración supone solamente un esfuerzo único. Probablemente no tenga un gran impacto en el rendimiento general.


Agrupe varias operaciones de cambio de datos SQL en una transacción.

Supongamos que está ejecutando un gran número de declaraciones SQL que implican añadir o cambiar datos (declaraciones `INSERT` o `UPDATE`). Puede aumentar el rendimiento considerablemente ejecutando todas las declaraciones en una transacción explícita. Si comienza una transacción explícitamente, cada una de las declaraciones se ejecuta en su propia transacción creada automáticamente. Después de que cada transacción (cada declaración) termina de ejecutarse, el motor de ejecución escribe los datos resultantes en el archivo de la base de datos en el disco.

Por otra parte, considere qué sucede si crea explícitamente una transacción y ejecuta las declaraciones en el contexto de dicha transacción. El motor de ejecución hace todos los cambios en la memoria, luego escribe todos los cambios en el archivo de la base de datos de una vez cuando se confirma la transacción. Generalmente, escribir los datos en el disco es la parte que lleva más tiempo en la operación. En consecuencia, si se escribe en el disco una sola vez en lugar de una vez por cada declaración SQL puede mejorar significativamente el rendimiento.


Procese grandes resultados de consultas `SELECT` en partes, utilizando el método `execute()` de la clase SQLStatement (con el parámetro`prefetch`) y el método `next()`.

Supongamos que se ejecuta una declaración SQL que recupera un gran conjunto de resultados. Posteriormente la aplicación procesa cada fila de datos en un bucle. Por ejemplo, aplica formato a los datos o crea objetos a partir de los mismos. El procesamiento de los datos puede tardar bastante tiempo, lo que puede suponer problemas de representación como, por ejemplo, pantalla bloqueada o sin respuesta. Tal y como se describe en "Operaciones asíncronas" en la página 71, una solución consiste en dividir el trabajo en segmentos. La API de la base de datos SQL facilita la división del procesamiento de datos.

El método `execute()` de la clase SQLStatement cuenta con un parámetro `prefetch` opcional (el primer parámetro). Si se proporciona un valor, éste especifica el número máximo de filas de resultados que devuelve la base de datos una vez finalizada la ejecución:

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

Una vez devuelto el primer conjunto de datos de resultado, se puede llamar al método `next()` para continuar ejecutando la sentencia y recuperar otro conjunto de filas de resultados. Al igual que sucede con el método `execute()`, el método `next()` acepta un parámetro `prefetch` para especificar un número máximo de filas para devolver:

```
// 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);
        }
    }
}
```

Se puede continuar llamando al método `next()` hasta que se carguen todos los datos. Tal y como se muestra en el anterior listado, es posible determinar el momento en que se han cargado todos los datos. Compruebe la propiedad `complete` del objeto SQLResult que se crea cada vez que finaliza el método `execute()` o `next()`.

***Nota:** utilice el parámetro `prefetch` y el método `next()` para dividir el procesamiento de los datos de resultado. No utilice este parámetro y método para limitar los resultados de una consulta a una parte de su conjunto de resultados. Si sólo desea recuperar un subconjunto de filas en un conjunto de resultados de una sentencia, utilice la cláusula `LIMIT` de la sentencia `SELECT`. Si el conjunto de resultados es amplio, aún puede utilizar el parámetro `prefetch` y el método `next()` para dividir el procesamiento de los resultados.*


Considere el uso de varios objetos SQLConnection asíncronos con una sola base de datos para ejecutar varias sentencias simultáneamente.

Si un objeto SQLConnection está conectado a una base de datos utilizando el método `openAsync()`, se ejecuta en segundo plano en lugar de en el subproceso de ejecución del motor de ejecución principal. Asimismo, cada conexión SQL se ejecuta en su propio subproceso en segundo plano. Con el uso de varios objetos SQLConnection, se pueden ejecutar eficazmente varias declaraciones SQL de forma simultánea.

Este enfoque también implica posibles desventajas. Y lo que es más importante, cada nuevo objeto SQLStatement requiere memoria adicional. Asimismo, las ejecuciones simultáneas suponen más trabajo para el procesador, especialmente en equipos que sólo cuentan con una CPU. Por estos aspectos, este enfoque no se recomienda para su uso en dispositivos móviles.

Un aspecto adicional para tener en cuenta es que puede perderse el potencial beneficio de la reutilización de los objetos SQLStatement, ya que un objeto SQLStatement está vinculado a un solo objeto SQLConnection. Por lo tanto, el objeto SQLStatement no puede reutilizarse si su objeto SQLConnection asociado ya está en uso.

Si opta por utilizar varios objetos SQLConnection conectados a una sola base de datos, se debe tener en cuenta que cada uno de ellos ejecuta sus sentencias en su propia transacción. Asegúrese de tener en cuenta estas transacciones independientes en cualquier código que modifique datos como, por ejemplo, adición, modificación o eliminación de datos.

Paul Robertson ha creado una biblioteca de código fuente abierto que ayuda a incorporar las ventajas del uso de varios objetos SQLConnection mientras se reducen los posibles inconvenientes. La biblioteca utiliza un grupo de objetos SQLConnection y administra los objetos SQLStatement asociados. De este modo, garantiza la reutilización de los objetos SQLStatement y varios objetos SQLConnection están disponibles para ejecutar varias sentencias simultáneamente. Para obtener más información y descargar la biblioteca, visite http://probertson.com/projects/air-sqlite/.

Optimización del archivo de base de datos

Evite cambios de esquemas de la base de datos.

Si es posible, se debe evitar cambiar el esquema (estructura de la tabla) de una base de datos una vez que se han añadido los datos en las tablas de la base de datos. Normalmente, un archivo de base de datos está estructurado con las definiciones de tabla al inicio del archivo. Cuando se abre una conexión a una base de datos, el motor de ejecución carga dichas definiciones. Cuando se añaden datos a las tablas de la base de datos, dichos datos se añaden al archivo después de los datos de definición de la tabla. Sin embargo, si se realizan cambios de esquema, los nuevos datos de definición de tabla se mezclan con los datos de la tabla en el archivo de base de datos. Por ejemplo, la adición de una columna a una tabla o la adición de una nueva tabla implicar la mezcla de tipo de datos. Si los datos de definición de tabla no se asignan al principio del archivo de base de datos, se tarda más en abrir una conexión a la base de datos. La apertura de la conexión es más lenta, ya que el motor de motor de ejecución tarda más en leer los datos de definición de tabla de las diferentes partes del archivo.

Utilice el método `SQLConnection.compact()` para optimizar una base de datos tras los cambios de esquema.

Si se deben realizar cambios en el esquema, se puede llamar al método `SQLConnection.compact()` después de completar los cambios. Esta operación reestructura el archivo de la base de datos para que los datos de definición de tabla se ubiquen al inicio del archivo. Sin embargo, la operación `compact()` puede demorar, especialmente a medida que se amplía el archivo de base de datos.

Evite el procesamiento en tiempo de ejecución innecesario

Utilice un nombre de tabla completo (incluyendo el nombre de base de datos) en la declaración SQL.

Especifique siempre de forma explícita el nombre de la base de datos junto con cada nombre de tabla en una sentencia. (Utilice "main" si es la base de datos principal). Por ejemplo, el código siguiente incluye un nombre de base de datos explícito `main`:

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

Si se especifica explícitamente el nombre de la base de datos se evita que el motor de ejecución tenga que verificar cada base de datos conectada para encontrar la tabla correspondiente. Asimismo, evita que el motor de ejecución elija la base de datos incorrecta. Siga esta regla aun si SQLConnection está conectada a una sola base de datos, ya que en segundo plano SQLConnection también está conectada a una base de datos temporal que se accede a través de las declaraciones SQL.

Utilice nombres de columna explícitos en las declaraciones SQL `INSERT` y `SELECT`.

Los siguientes ejemplos muestran el uso de nombres de columna explícitos:

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

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

Compare los ejemplos anteriores con los siguientes. **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
```

Sin nombres de columna explícitos, el motor de ejecución debe realizar trabajo adicional para determinar los nombres de columna. Si una sentencia `SELECT` utiliza un carácter comodín en lugar de columnas explícitas, hace que el motor de ejecución recupere datos adicionales. Esta información adicional requiere más procesamiento y crea instancias de objetos adicionales que no son necesarios.

Evite la unión de la misma tabla varias veces en una sentencia, a no ser que la tabla se esté comparando consigo misma.

Conforme crecen las declaraciones SQL, es posible unir de forma accidental una tabla de base de datos a la consulta varias veces. En ocasiones, se puede obtener el mismo resultado utilizando la tabla sólo una vez. Es probable se produzca la unión de la misma tabla varias veces si se está utilizando una o varias vistas en una consulta. Por ejemplo, se puede unir una tabla a la consulta y también una vista que incluya los datos de esa tabla. Las dos operaciones pueden causar varias uniones.

Uso eficaz de la sintaxis SQL

Utilice `JOIN` (en la cláusula `FROM`) para incluir una tabla en una consulta en lugar de una subconsulta en la cláusula `WHERE`. Esta sugerencia se aplica aunque sólo sean necesarios unos datos de la tabla para el filtro, no para el conjunto de resultados.

La unión de varias tablas en la cláusula `FROM` funciona mejor que el uso de una subconsulta en una cláusula `WHERE`.

Evite las declaraciones SQL que no puedan aprovechar los índices. Estas sentencias incluyen el uso de funciones de agregado en una subconsulta, una sentencia `UNION` en una subconsulta o una cláusula `ORDER BY` con una sentencia `UNION`.

Un índice puede aumentar considerablemente la velocidad de procesamiento de una consulta `SELECT`. Sin embargo, determinada sintaxis SQL evita que la base de datos utilice índices, haciendo que utilice los datos reales para buscar u ordenar operaciones.

Evite el operador `LIKE`, especialmente con un carácter comodín inicial como en `LIKE('%XXXX%')`.

Debido a que la operación `LIKE` admite el uso de búsquedas con caracteres comodín, funciona de forma más lenta que el uso de comparaciones de coincidencia exacta. En concreto, si se inicia la cadena de búsqueda con un carácter comodín, la base de datos no puede utilizar índices en la búsqueda. Como alternativa, la base de datos debe buscar el texto completo en cada fila de la tabla.


Intente evitar el operador `IN`. Si los posibles valores se conocen de antemano, la operación `IN` se puede escribir utilizando `AND` u `OR` para una ejecución más rápida.

La segunda de las dos siguientes sentencias se ejecuta más rápido. Resulta más rápida porque utiliza expresiones de igualdad sencillas combinadas con `OR` en lugar de utilizar las sentencias `IN()` o `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 una declaración SQL para mejorar el rendimiento.

Tal y como se ha mostrado en anteriores ejemplos, el modo en que se escribe una declaración SQL también puede afectar al rendimiento de la base de datos. Suelen existir varias maneras de escribir una declaración SQL `SELECT` para recuperar un determinado conjunto de resultados. En algunos casos, un enfoque se ejecuta notablemente más rápido que el otro. Además de las sugerencias anteriores, se puede aprender más sobre las distintas declaraciones SQL y su rendimiento en recursos dedicados sobre el lenguaje SQL.

Comprobación del rendimiento de las declaraciones SQL


Compare directamente las declaraciones SQL alternativas para determinar cuál es más rápida.

La mejor forma de comparar el rendimiento de varias versiones de una declaración SQL consiste en probarlas directamente en la base de datos y la información.

Las siguientes herramientas de desarrollo proporcionan tiempos de ejecución al ejecutar declaraciones SQL. Utilícelas para comparar la velocidad de versiones alternativas de declaraciones:

- Run! (Herramienta de prueba y edición de consultas SQL de AIR, de Paul Robertson)
- Lita (Herramienta de administración SQLite, de David Deraedt)

Capítulo 9: Prueba comparativa e implementación

Prueba comparativa

Existe una serie de herramientas disponibles para las aplicaciones de prueba comparativa. Se pueden utilizar las clases Stats y PerformanceTest, desarrolladas por los miembros de la comunidad de Flash. También se puede utilizar el visor en Adobe® Flash® Builder™ y la herramienta FlexPMD.

Clase Stats

Para analizar el código en tiempo de ejecución utilizando la versión oficial de Flash Player, sin ninguna herramienta externa, se puede utilizar la clase Stats desarrollada por mr. doob de la comunidad de Flash. La clase Stats se puede descargar en: http://code.google.com/p/mrdoob/wiki/stats.

Esta clase permite realizar un seguimiento de los siguientes elementos:

- Fotogramas representados por segundo (cuantos más, mejor).
- Milisegundos empleados para representar un fotograma (cuantos menos, mejor).
- Cantidad de memoria que utiliza el código. Si aumenta en cada fotograma, es posible que la aplicación experimente una pérdida de memoria. Es importante investigar esta posible pérdida de memoria.
- Cantidad máxima de memoria utilizada por la aplicación.

Una vez descargada, la clase Stats se puede usar con el siguiente código de compacto:

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

Al utilizar la compilación condicional en Adobe® Flash® CS4 Professional o Flash Builder, se puede habilitar el objeto Stats:

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

Al cambiar el valor de la constante `DEBUG`, se puede habilitar o deshabilitar la compilación del objeto Stats. Se puede utilizar el mismo enfoque para sustituir cualquier lógica de código que no se desee compilar en la aplicación.

Rendimiento de la clase PerformanceTest

Para perfilar la ejecución de código ActionScript, Grant Skinner ha desarrollado una herramienta que se puede integrar en un flujo de trabajo de comprobación de unidades. Es posible transmitir una clase personalizada a la clase PerformanceTest, que realiza una serie de comprobaciones en el código. La clase PerformanceTest permite realizar una prueba comparativa de los distintos enfoques con facilidad. La clase PerformanceTest se puede descargar en: http://www.gskinner.com/blog/archives/2009/04/as3_performance.html.

Visor de Flash Builder

Flash Builder se distribuye con un visor que permite realizar pruebas comparativas del código con un alto nivel de detalle.

Nota: *utilice la versión de depuración de Flash Player para acceder al visor o, de lo contrario, aparecerá un mensaje de error.*

El visor también se puede utilizar con el contenido producido en Flash CS4 Professional. Para ello, cargue el archivo SWF compilado desde un proyecto de ActionScript o Flex en Flash Builder y podrá ejecutar el visor en el mismo. Para obtener más información sobre el visor, consulte el tema "Profiling Flex applications" (Perfiles de aplicaciones de Flex) en Using Flash Builder 4 (Uso de Flash Builder 4).

FlexPMD

Los servicios técnicos de Adobe han publicado una herramienta denominada FlexPMD, que permite auditar la calidad del código ActionScript 3.0. FlexPMD es una herramienta de ActionScript similar a JavaPMD. FlexPMD mejora la calidad del código auditando un directorio de origen de ActionScript 3.0 o Flex. Detecta prácticas no adecuadas de codificación como, por ejemplo, código no utilizado, demasiado código complejo, mucho código largo, uso incorrecto del ciclo de vida del componente de Flex.

FlexPMD es un proyecto de código abierto de Adobe disponible en la siguiente dirección: http://opensource.adobe.com/wiki/display/flexpmd/FlexPMD. Un plugin Eclipse también está disponible en la siguiente dirección: http://opensource.adobe.com/wiki/display/flexpmd/FlexPMD+Eclipse+plugin.

FlexPMD facilita la auditoría de código para garantizar que el código esté limpio y optimizado. La eficacia real de FlexPMD radica en su extensibilidad. Como desarrollador, es posible crear conjuntos de reglas propias para auditar cualquier código. Por ejemplo, se puede crear un conjunto de reglas que detecten un uso excesivo de filtros o cualquier otra práctica de codificación inadecuada que se desee detectar.

Implementación

Al exportar la versión final de la aplicación en Flash Builder, es importante asegurarse de que se exporta la versión oficial. Con la exportación de una versión oficial se elimina la información de depuración incluida en el archivo SWF. La eliminación de la información de depuración reduce el tamaño del archivo SWF y ayuda a que la aplicación se ejecute con más rapidez.

Para exportar la versión oficial del proyecto, utilice el panel Project (Proyecto) de Flash Builder y la opción Export Release Build (Exportar versión oficial).

Nota: *al compilar el proyecto en Flash Professional, no se tiene la opción de seleccionar entre la versión de depuración y la oficial. El archivo SWF compilado constituye la versión oficial de forma predeterminada.*