Flex性能,内存管理和对象缓存

翻译：FireYang　　来源：http://www.insideria.com

在屏幕上的显示的东西直接影响到Flex应用程序的响应能力和性能。更多的东西，更加降低了应用程序的响应速度。在这里我不做过多的实例了。这篇教程帮你在动态添加了很多UI组件的高负载情况下，仍然有很好的性能。(这句得到胡矿和火把的大力支持，感谢)

在flex组件的生存周期中，构造函数和初始化过程是最昂贵(极大的代价)的操作。你在添加和移除大量的复杂组件时，系统将变得十分繁忙并且应用程序的整体性能也会降低。组件越复杂，消耗越大。在你的系统中通过对象缓存技术将减少这种冲击(也称对象池)

减少，重用，循环

当你使用对象缓存技术的时候，创建一个应用程序能使用的对象缓存器。当你用完这些对象的时候，对象将被重新放回缓存器里这样你就可以再次重用了。这种技术减少了在运行时大量对象的创建，在程序运行时减少了整体内存的使用量，运行应用程序所需资源也趋于稳定（减少cpu和内存的冲击），并且使得正常应用的情况下应用程序的整体性能和可伸缩性也表现的更好。

用一个简单的项目来展示这种技术和技术的好处。这个项目中有两个应用程序。两者都实现了相同的功能，然而一个使用了对象缓存另一个没有。这个简单的应用程序示范在每次"ENTER_FRAME"事件中创建和移除100个“CircleRenderer”对象的时候对内存和性能的冲击。

有人认为，每个"ENTER_FRAME"事件中添加和移除100子对象的例子是不是太极端了。另一些人认为，这个例子是基础的因为CircleRenderer对象它本身就是一个很简单的对象。复杂对象即使不像这个示例那样频繁的创建和移除，创建和移除也需要很大资源消耗。CircleRenderer是拓展自UIComponent的一个简单对象。它通过颜色和半径变量绘制一个圆，并在里面创建3个TextFields(文本域)，分别显示：UID字符串-颜色-半径。应用程序里所有半径，颜色和UID值都是随机产生的。

首先，让我们看一下ObjectCache对象，然后比较它们之间的差异：

```
public class ObjectCache
{
	private static var cache : ArrayCollection;
	private static const preCache : int = 100;

	public static function getRenderer() : CircleRenderer
	{
		var renderer : CircleRenderer;

		if ( cache == null )
		{
			cache = new ArrayCollection();

			for ( var x : Number = 0; x < preCache; x ++ )
			{
				renderer = new CircleRenderer();
				cache.addItem( renderer );
			}
		}
```

```
            if ( cache.length <= 0 )
                renderer = new CircleRenderer();
            else
                renderer = cache.removeItemAt( 0 ) as CircleRenderer;

            return renderer;
        }

        public static function setRenderer( renderer : CircleRenderer ) : void
        {
            cache.addItem( renderer );
        }
}
```

你将在 ObjectCache 类中有两个 public 的静态函数：getRenderer 和 setRenderer。getRenderer 函数是从 cache 中返回一个 CircleRenderer 对象：如果 cache 还没有被初始化，cache 将预存 100 个 CircleRenderer 对象。如果 cache 已经包含了 renderer 对象，它将从 cache 中移除并返回它。如果 cache 空掉了，它将简单的 new 一个然后返回。setRenderer 函数仅仅是将一个 renderer 保存到 cache 中。

现在，让我们进入程序本身看看……

在每次的“ENTER_FRAME”事件中，这个“No Cache”应用程序把容器中所有的 circle 对象移除，并创建 100 个新的对象：

```
private function onEnterFrame( event : Event ) : void
{
        circlesContainer.removeAllChildren();
        for ( var x : Number = 0; x < children; x ++ )
        {
            var renderer : CircleRenderer = new CircleRenderer();
             RendererUtil.updateRenderer( renderer );
             circlesContainer.addChild( renderer );
        }
}
```

在每次的“ENTER_FRAME”事件中，“Cache”应用程序在容器中移除所有 circle 对象的时候直接放回到 cache 里。然后在添加 100 个对象的时候又重新从 cache 中寻回。

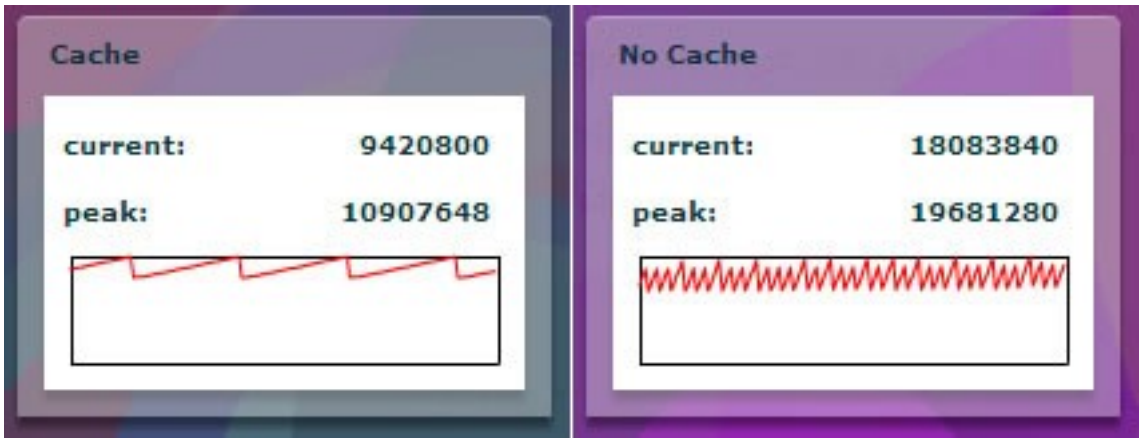
```
private function onEnterFrame( event : Event ) : void
{
        while ( circlesContainer.numChildren > 0 )
        {
            ObjectCache.setRenderer( circlesContainer.removeChildAt( 0 ) as CircleRenderer );
        }

        for ( var x : Number = 0; x < children; x ++ )
        {
            var renderer : CircleRenderer = ObjectCache.getRenderer();
            RendererUtil.updateRenderer( renderer );
            circlesContainer.addChild( renderer );
```

```
        }
}
```

“RendererUtil.updateRenderer”方法简单设置所有随机产生值到组件上。

现在，看看结果……

在这个应用程序中，有一个性能监测类，来源这里，这个性能监测器跟踪当前内存使用情况，内存使用情况的峰值和历史内存使用情况。**注意**：性能监测器使用了 System.totalMemory 来计算内存的使用量。这些就是 flashplayer 占用的全部内存，因此不要同时运行这两个程序，否则你的结果将会是不正确的！



我的机器是[Inte(R) Core(TM)2 Duo 2 GHz, 2 GB RAM]｛这是原作者的，我的可没这么好的配置^_^｝，我运行这些示例用了 2 分钟。chche 的版本的内存使用峰值是 10907648 bytes，而 non-cached 版本的峰值是 19681280 bytes。这里相差了 8773632 bytes，或 8.36 MB。不仅仅是 cache 版本使用了更少的内存，而且在内存和 cpu 使用也更趋于稳定了）。（我希望我能在 flash/Flex 中跟踪 cpu 的使用情况，但是目前我还没有找到方法）。

两个图形有相同的横像时间轴（而纵像的轴是根据你的峰值决定的）。从图形中可以看到，non-cached 版本频率更高些，内存的波动也更激烈，它也需要更加频繁的垃圾回收。

你可以在你自己的电脑上运行此应用程序（警告－这可是很耗 cpu 的代码）：

Cache:
http://www.cynergysystems.com/blogs/blogs/andrew.trice/objectcache/Cache.html
No Cache:
http://www.cynergysystems.com/blogs/blogs/andrew.trice/objectcache/NoCache.html

你可以在这里查看源代码:
http://www.cynergysystems.com/blogs/blogs/andrew.trice/objectcache/srcview/

你可以在这里下载源代码:
http://www.cynergysystems.com/blogs/blogs/andrew.trice/objectcache/srcview/ObjectCache.zip