为保证 Flex 程序运行的流畅，高效。本人总结了一下几条技巧：

1，当创建一个数组的时候避免用 new 操作符,用 var a:Array = [];而不用 var a:Array = new Array();

2，快速的复制一个数组：

```
var copy:Array = sourceArray.concat ();
```

3，在整个程序的生命周期中都不会改变的变量用 const 定义常量

```
public const APPLICATION_PUBLISHER : String = "Company, Inc. ";
```

4，当一个类不需要有子类的时候应该将该类声明为 final 类型的

```
public final class StringUtils
```

5，尽量避免使用 setStyle（）方法，这个方法在 Flex 框架里面是众多代价昂贵的方法之一。

6，能用 ENTER_FRAME 事件就不用 Timer 事件

```
//使用
public function onEnterFrame(event:Event) : void { }
private function init():void {
    addEventListener(Event.ENTER_FRAME, onEnterFrame);
}

//而不使用 Timer
public function onTimerTick(event:Event) : void { }
private function init():void {
    var timer:Timer = new Timer();
    timer.start();
    timer.addEventListener(TimerEvent.TIMER, onTimerTick);
}
```

7，对于自定义的 ItemRenderer，如果不需要背景透明，或者使用自己绘制背景。

建议把 opaqueBackground 属性设置成一个固定的颜色，这样渲染器能够忽略透明度计算提高性能。

而且如果绘制自己的背景，那么把 autoDrawBackground 设置成 false，这一操作将会指示 Flex 不要浪费时间来绘制一个你无论如何都会覆盖的默认背景。

比如下面这个例子，ItemRenderer 里面使用自行绘制的渐变背景。我们就可以吧 autoDrawBackground 设置成 false，同时 opaqueBackground 随便设置成一个固定的值。

```
<s:ItemRenderer xmlns:fx="http://ns.adobe.com/mxml/2009"
                xmlns:s="library://ns.adobe.com/flex/spark"
                opaqueBackground="0x000000"
                autoDrawBackground="false">

```

```
    <s:Rect left="0" right="0" top="0" bottom="0">
        <s:fill>
            <s:LinearGradient rotation="90">
                <s:GradientEntry color="#FF0000" ratio="0"/>
                <s:GradientEntry color="#DD0000" ratio=".66"/>
            </s:LinearGradient>
        </s:fill>
    </s:Rect>

    <s:Label id="labelDisplay" left="5" right="5" top="15" bottom="15"/>

</s:ItemRenderer>
```

8，ItemRenderer 内部设置数据的时候，避免使用绑定，因为使用绑定的话会调用额外的 ActionScript 代码。通常，我们可以通过覆盖 set data 方法来设置数据。

比如，原来通过绑定是这样：

```
<s:ItemRenderer xmlns:fx="http://ns.adobe.com/mxml/2009"
                xmlns:s="library://ns.adobe.com/flex/spark">
    <s:Label id="姓名" value="{data.name}"  fontSize="12"/>
</s:ItemRenderer>
```

我们可以改成用 set data 的方式：

```
<s:ItemRenderer xmlns:fx="http://ns.adobe.com/mxml/2009"
                xmlns:s="library://ns.adobe.com/flex/spark">
    <fx:Script>
        <![CDATA[
            override public function set data(value:Object):void
            {
                super.data = value;

                if (data)
                {
                    nameLab.text = data.name;
                }
                else
                {
                    nameLab.text = "";
                }
            }
        ]]>
    </fx:Script>
    <s:Label id="nameLab" value="{data.name}"  fontSize="12"/>
</s:ItemRenderer>
```

9，ItemRenderer 内部如果要显示外部图像时，使用共享图像缓存 ContentCache

使用共享图像缓存 ContentCache，使得图像加载过一次以后，在外部视图滚动的时候不必重新加载。使用方法：声明一个静态的 ContentCache 实例，并将它设置给任一个 BitmapImage 组件的 contentLoader 属性，这样该类型的所有渲染器将共享这个 ContentCache 实例，自动地加载并缓存图像。

```
<s:ItemRenderer xmlns:fx="http://ns.adobe.com/mxml/2009"
                xmlns:s="library://ns.adobe.com/flex/spark"
                initialize="initializeHandler(event)">
    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;

            import spark.core.ContentCache;

            static private const iconCache:ContentCache = new ContentCache();

            private function initializeHandler(event:FlexEvent):void
            {
                icon.contentLoader = iconCache;
            }

            override public function set data(value:Object):void
            {
                super.data = value;

                if (data)
                {
                    icon.source = data.imageUrl;
                }
                else
                {
                    icon.source = null;
                }
            }


        ]]>
    </fx:Script>

    <s:BitmapImage id="icon" top="10" left="5" width="32" height="32" />

</s:ItemRenderer>
```

10，使用 Group 取代 BorderContainer

BorderContainer 是一种可以绘制边框和背景的容器，但有时我们可以使用包含一个 Rect 的 Group 来获得相同的效果，并获得更好的性能。

比如原来我们是这么使用 BorderContainer：

```
<s:BorderContainer width="200" height="200"
    backgroundColor="#CCCCCC" borderColor="#999999">
</s:BorderContainer>
```

现在可以使用 Group 来实现同样的效果：

```
<s:Group width="200" height="200">
    <s:Rect left="0" right="0" top="0" bottom="0">
        <s:fill>
            <s:SolidColor color="#CCCCCC"/>
        </s:fill>
        <s:stroke>
            <s:SolidColorStroke color="#999999"/>
            </s:stroke>
    </s:Rect>
</s:Group>
```

11，如果要隐藏页面上组件，使用 includeIn 或者 excludeFrom，而不是设置 visible 属性

使用 includeIn 和 excludeFrom 将不应该可见的组件从显示列表中删除，就不会再对它们进行渲染处理。 相反，如果将 visible 设置为 false，那么组件将仍然保留在显示列表上，这样需要执行不必要的布局操作和渲染。

12，显示嵌入图像，使用 BitmapImage 而不是 Image

BitmapImage 组件是一个轻量级版本的 Image 组件。这两个组件都能够显示嵌入式的图像组件。

两者之间最大的差异是：1，Image 能够运行时加载外部图像，而 BitmapImage 需要一些设置才能加载外部图像。2，与 BitmapImage 不同，Image 的皮肤也是可换的并且支持丢失图像指示器。 对于嵌入图像，不需要这些功能，应该使用轻量级的 BitmapImage 组件。

比如，原来使用 Image 组件来显示：

```
<s:Image source="@Embed('assets/bg.jpg')"/>
```

可以使用 BitmapImage 来代替：

```
<s:BitmapImage source="@Embed('assets/bg.jpg')"/>
```

13，使用 png 格式的图片，而不是 gif 或 jpeg 格式的图片

运行时对 PNG 文件格式进行解码时速度要快得多，如果可以的话，应该尽可能使用 PNG 来代替 GIF 和 JPEG 图片。

14，需要矩形阴影或圆角矩形阴影的话。使用 RectangularDropShadow，而不是 DropShadowFilter。

```
<s:Group width="300" height="150" horizontalCenter="0" verticalCenter="0">
```

```
    <s:RectangularDropShadow id="dropShadow" blurX="10" blurY="10" alpha="0.5"
distance="3"
        angle="90" color="#000000" left="0" top="0" right="0" bottom="0" blRa
dius="8" brRadius="8"
        tlRadius="8" trRadius="8" />
</s:Group>
```

15，如果不需要的话，把这四个属性设为 false 可以节约资源，提高效率

```
this.mouseChildren = false;
this.mouseEnabled = false;
this.tabChildren = false;
this.tabEnabled = false;
```