# ActionScript 3 教程系列

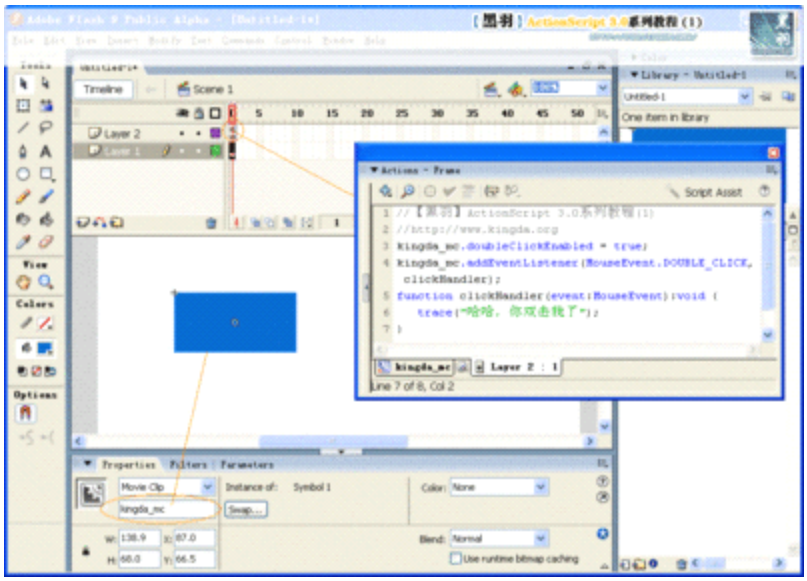
# 【黑羽】ActionScript 3.0系列教程(1)：与Flash9先来一次亲密接触！

如蒙转载，请留下我的Blog链接：www.kingda.org， thx）

Flash Professional 9 ActionScript 3.0 Preview 版本今天发布了，意味着从此我们从此不仅仅只能使用Flex 2来使用AS3.0，更可以使用我们一直很熟悉的Flash IDE来进行AS3.0开发了。

与Flex 2不同，Flash 9 alpha(即上面的Flash Professional 9 ActionScript 3.0 Preview )允许我们创建基于时间轴的ActionScript 3.0的Fla文档，而不是基于State的MXML文档。 在Flash 9 alpha 里，我们和以前一样可以在舞台上直接手绘矢量图，创建元件，添加动画，等等。

我黑羽是急性子，先跳开一些特色的介绍，单刀直入，马上来个实例。边讲解边说说Flash 9的特点。



点击看大图，清楚点。

为了照顾新手xdjm们，看图说话一把。老鸟略过勿看，省得嫌我罗嗦。呵呵。

新建一个fla,随便画一个方块什么的，双击选中按F8转换成MovieClip。在属性面板中命名为kingda_mc。和以前一模一样。
再新建一层，命名为actions，这是个好习惯，要保持。选中第一帧，按F9打开动作面板，写入如下代码。
Control+Enter，在测试窗口中，双击那个方块，就会有trace信息显示出来。

```
//【黑羽】ActionScript 3.0系列教程(1)
```

```
//http://www.kingda.org

kingda_mc.doubleClickEnabled = true;

kingda_mc.addEventListener(MouseEvent.DOUBLE_CLICK, clickHandler);

function clickHandler(event:MouseEvent):void {

    trace("哈哈，你双击我了");

}

//直接支持双击了， 兄弟们，爽不爽 ^_^
```

稍作解释，这儿有几个和AS2.0不同的地方了。

**1. AS2.0中，MovieClip是不可以加侦听器地，但AS3.0中，却可以了。**讲点深入的东东给老鸟听，所有AS3.0中能被我们看见的对象，其祖宗都是DisplayObject类。标准说法是都间接或直接的继承于DisplayObject类。而这个DisplayObject又是EventDispatcher的儿子。所以，我们就有了这个推论：

**AS3.0中所有能被我们看到的东西，都能发送事件和加侦听器。** 完全适用于Event Model.
爽吧， 我是爽歪了。AS2.0中为了解决这个麻烦我还自己编了一个代理发送事件类EventSender。省了不少事儿，而现在连这个也不用了，霍哈哈。

2.AS3.0中要让MovieClip在接受click事件，rollover事件能够像Button一样，鼠标放上去显示手型，那么一定要加上一句：

```
kingda_mc.buttonMode = true;
```

小事一桩，一笔带过。

**3.AS3.0中的事件模型和AS2.0大不一样了。**

简而言之，就是“规范”。不再直接使用字符串来定义事件名称了。又要讲深一点了，都是使用了新的const型变量来定义事件字符串名称，一旦定义，不能再更改。
如

```
public static const MOVE:String = "move";
```

极大的避免了我们因为手误，打错字符串，而花上一个下午找bug。使用了这种模式，我们一旦

打错，编译器立刻会发现并告诉我们。多好。
给出一些鼠标事件列表，大家可以替换上面源码中的事件类型，自己试着玩儿。
如，你可以换成MouseEvent.MOUSE_OVER就变成了以前的onRollOver效果。

CLICK : String = "click"

[static] Dispatched when a user presses and releases the main button of the user's pointing device over the same InteractiveObject. MouseEvent

DOUBLE_CLICK : String = "doubleClick"

[static] Dispatched when a user presses and releases the main button of a pointing device twice in rapid succession over the same InteractiveObject when that object's doubleClickEnabled flag is set to true. MouseEvent

MOUSE_DOWN : String = "mouseDown"

[static] Dispatched when a user presses the pointing device button over an InteractiveObject instance in the Flash Player window. MouseEvent

MOUSE_LEAVE : String = "mouseLeave"

[static] Dispatched by the Stage object when the mouse pointer moves out of the Flash Player window area. Event

MOUSE_MOVE : String = "mouseMove"

[static] Dispatched when a user moves the pointing device while it is over an InteractiveObject. MouseEvent

MOUSE_OUT : String = "mouseOut"

[static] Dispatched when the user moves a pointing device away from an InteractiveObject instance. MouseEvent

MOUSE_OVER : String = "mouseOver"

[static] Dispatched when the user moves a pointing device over an InteractiveObject instance in the Flash Player window. MouseEvent

MOUSE_UP : String = "mouseUp"

[static] Dispatched when a user releases the pointing device button over an InteractiveObject instance in the Flash Player window. MouseEvent

MOUSE_WHEEL : String = "mouseWheel"

//支持鼠标滚轮喽，霍霍。

指出一点，在我给出的例子中，使用了双击这个事件。这个有点特殊，在使用双击事件之前，要加上一句：

kingda_mc.doubleClickEnabled = true;

因为MovieClip对于双击事件默认是false，关闭的。

**4.侦听器的不同。**

在AS2.0中我们通常要新建一个对象做侦听器。也可以像我的例子中用function做侦听器。但是，很可惜，由于AS2.0的设计缺陷，使得function中的this指向常常给我们带来困扰。于是有了Delegate类来解决。

而如今，AS3.0中采用了优秀的Traits Object架构（唔，这个，就暂不解释了），使得它能记住this的指向。所以，兄弟们，放心大胆使用Function作为侦听器使用吧。

今天就写这么多了，主要是Flash 9出来，我老人家激动了一下，一下子写了这么多东东。希望对大家有所帮助，希望大家狂顶支持一把，不然没动力，本系列教程会变成太监贴！霍哈哈！ ^_^ 快回帖支持！

本篇主要涉及了一下AS3.0中的事件模型部分，这是很重要的。以后会有更深入的教程来详细介绍。本篇的目的就是让大家使用一下Flash 9和AS3，消除陌生感。写的浅了，还请包涵。

**下一篇介绍非常实用的东东，类和MovieClip的绑定，和Flash 9中一大特色：Document Class，用来替代在时间轴写代码的好东东。** 

# 【黑羽】AS 3.0教程(2)：AS3.0的类及绑定

hi，鸟笼山的朋友们，这次我们开始介绍AS3.0中的类如何和库中元件绑定，和特殊的Document

Class设计。

总共4步：
1.建一个标准的AS3.0类 (暂命名为KingdaMC，多么伟大的名字啊，简称“有名”)
2.新建一个元件，并设置它的Linkage和上面的类绑定。
3.在时间轴上写代码，用AS3.0代码创建n份“有名”。
4.删掉时间轴上代码，使用Flash 9新特色Document Class在舞台上创造n份“有名”。just 体验一把 。

（写起来才知道要讲的内容多啊，晕，打了两个小时，累了，本节专门只讲讲AS3.0的类吧，其余的慢慢再讲）

在创建AS3.0之前，请允许我先对AS3.0类的语法和继承设计表示敬意。与标准的完全兼容，更好更严格的封装特性，特别是命名空间(name space)的引入。从今天起从AS3.0起， 在OOP层面上，AS3.0已经和Java，C#平起平坐了，甚至在某些方面(比如name space)比java更有意思。

让俺对AS2.0老鸟说几句憋在心里的激动之语： 

如果说AS2.0只是外表接近OOP标准语言，内在还是乱糟糟的AS1.0脚本语言，那么AS3.0不论是从OOP设计级别，还是从编译器级别（如，对弱引用—— week reference—— 的支持）来评估的都是标准、正宗、强大的语言。

从AS3.0起，我们ActionScript 开发者可以挺起胸口，俺们是真正的行业标准级程序员。

在我继续教程之前，我要唱首歌给大家听：

AS3的英明，绝对不是一句两句能说清！~~~~~~ 

打完，收工。

1.建立AS3.0类文件，

类文件是干嘛用的？比如说，我们想让一个对象(Object)有很多功能，比如说这个对象是MovieClip型的，支持拖拽，支持双击等等。那么先在一个类文件里写清楚这些要求和实现方法，然后就可以用这个类创建许多实例，这些实例就全具有了这些功能。写一次，就能用很多次，多好。最重要的是它还可以通过继承来重用很多代码，为将来节省更多的时间。

废话少说，Ctrl+N打开“新建”窗口，选择建立“ActionScript file”，Ctrl+S，暂存为"KingdaMC.as"文件。（即“有名”的类文件）。

输入如下代码：

```
//【黑羽】ActionScript 3.0系列教程(2)

// http://www.kingda.org

//代码如下
```

```
//package  见讲解1

package {

    import flash.display.MovieClip;         //讲解2

    import flash.events.MouseEvent;



    //讲解3

    public class KingdaMC extends MovieClip {

            public function KingdaMC() {

              trace("Kingda created: " + this.name);



              this.buttonMode = true;

              this.addEventListener(MouseEvent.CLICK, clickHandler);

              this.addEventListener(MouseEvent.MOUSE_DOWN, mouseDownListener);

              this.addEventListener(MouseEvent.MOUSE_UP, mouseUpListener);

        }

        private function clickHandler(event:MouseEvent):void {

              trace("You clicked the ball");

        }

        function mouseDownListener(event:MouseEvent):void {
```

```
            this.startDrag();
        }
        function mouseUpListener(event:MouseEvent):void {
            this.stopDrag();
        }
    }
}
```

讲解1： AS2.0中我们使用的是全饰名称来声明类，通俗说，包括了类的路径在类名前。AS3.0则把路径提取出放在package这个关键字后面。本文例子中的类文件和Fla文件在同一个目录下，因此 package后面没有什么东西。如果类文件在 org目录下的kingda目录里，那么就要写成 // ActionScript 2.0

```
class org.kingda.KingdaMC {
}
```

```
// ActionScript 3.0
package org.kingda {
   public class KingdaMC {}
}
```

你在package中可以定义好几个类，不用再写全饰名称了。但我不推荐这样做。一个文件一个类比较好管理。

讲解2：

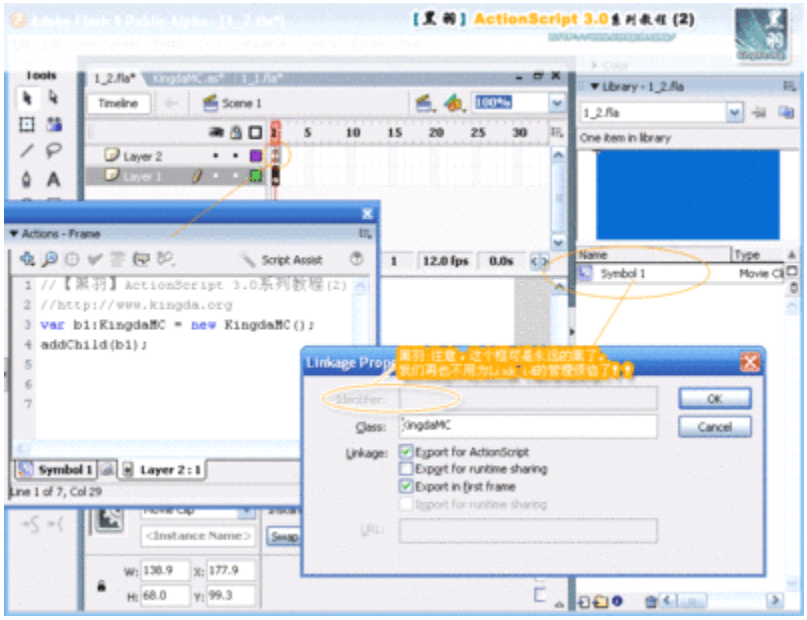
MovieClip类再也不像AS2.0那样是默认的全局类了。你要使用MovieClip类一定要写这一句导入。

讲解3：

类在AS3.0中也有public 和 internal的区分了。
public 表示这个类可以在任何地方导入使用。
internal表示这个类只能在同一个package里面使用。
不写，就默认为internal这个新的关键字。
还有一个属性是final，表示这个类不能被继承了，继承树到此为止。
说白了，这三个属性都是用来让我们更加规范的管理类之间的关系，以便将来修改时心里有谱，大大的方便了修改。

同时，对架构的设计能力要求更高，新手们和小项目还是多使用public吧。越到后来你会越喜欢internal的。我只要看到internal和private这两个关键字，心中就无比的稳定和舒服。想来不少programmer和我都会有同感吧。

2.新建一个元件，并设置它的Linkage和上面的类绑定。
和教程第一章一样，画一个方块，按F8转成MovieClip，再在库中右键点击它，选择"linkage"




点击看大图。

在Class里面写上KingdaMC。注意，id输入框已经被废止了。因为在AS3.0中，再也没有MovieClip.attachMovie(), MovieClip.createEmptyMovieClip(), 以及 MovieClip.createTextField()的存在了。

所有舞台的可见对象都由new来创建。
比如说本例中，symbol1绑定了KingdaMC，那么如果我要在舞台上创建一个KingdaMC，只要写

```
var b1:KingdaMC = new KingdaMC();

addChild(b1);
```

即可。

还记得以前那些乱七八糟的创建影片和组件的语法吗？什么createClassObject(),DepthManager的createChildAtDepth(),createClassChildAtDepth(),等等等等。我的一个java同事刚刚学习AS时，被上面哪些乱乱的创建函数弄得一头雾水。统统没有了。只有一统天下的new ClassName()，多标准哪，多舒服啊，多好学啊。

**所以说，没学AS2.0直接学AS3.0绝对是新手的福气。**

第二句,addChild这个很重要。

光有第一句new可不行。那只是告诉Flash我建了一个名字叫b1的KingdaMC要显示，但还没告诉Flash什么时候显示呢。
你什么时候打addChild(b1)，那Flash才会把它显示在舞台上。怎么样，多简单。
其实这里省略了一个this。如果你有一个MovieClip名叫BigKingda，希望在这个BigKingda里面加上一个KingdaMC实例，那么就要写

```
BigKingda.addChild(b1);
```

非常简单。

Ctrl+Enter测试，发现创建的KingdaMC，支持拖拽。

大家试着用代码多创建几个KingdaMC，比如用for循环来建个10个8个的来玩玩。

好了，累了，要歇歇了，下次讲Document Class。

本节代码例子参考了Jen deHaan的教程，原因很简单，她选的例子很好的说明了绑定和Document Class，黑羽做了少量的中文改动。她原来的源码例子点此下载。

# AS3.0教程(3):Document Class特色为我们带来了什么？

Document Class，中文直译为“文档类”。顾名思义，就是和文档绑在一起的类。

文档是啥？就是要和这个类绑在一起的Fla文件。

什么用处？

这个玩意儿根本目的就是想把AS代码和Flash设计完全剥离。
从此，Fla里面只管设计，逻辑代码全部由外部的类来包办。
对于设计者和新手，黑羽可以打个比方，就好比三步走：
1.把Fla里面所有代码集中到第一帧，
2.再把第一帧里的代码拷贝到外部的一个as文件里。
3.再按照AS3.0的形式，把这些代码放到这个类的构造函数里去。再根据这些个代码抽出一些类的属性和方法就OK了。
唔，这个比喻有很多小毛病。但比较直观，比较好理解，一步步来。
等你明白有哪些小毛病了，你已经不需要这个破比喻了。

对于程序开发者(比如c#)

Document Class就好比一个Entry，它的构造函数就相当于一个main函数。

教程相关例子下载：点击下载

先来一个例子：
上次建立的KingdaMC.fla和KingdaMC.as还在吧？

删了？重来一遍吧，本次要用到fla里面的加好链接类的元件，以及相关的KingdaMC类文件。

我们干两件事：
1.新建一个Document Class类。
2.用上之前的那个Kingda.fla,绑定Document Class

**1.新建一个as文件，命名为KingdaMCDocumentClass,记得要和fla在同一目录下：**

输入如下代码：

```
//因为在统一目录下，所以package后面没有路径

package {

    import flash.display.MovieClip;



    public class KingdaMCDocumentClass extends MovieClip {

        private var tempMC:KingdaMC; //临时变量，持有当时创建的KingdaMC的引用。
```

```
        private var MAX_MCS:int = 10;   //最多创建的KingdaMC数目


      //构造函数，和类同名，在AS3.0中必须为public。不能用private,protected或者自定义的namespace
        public function KingdaMCDocumentClass() {
            var i:int;   //新的数据类型int，只要是整数，就请用int。效率快过Number。
            for (i = 0; i < MAX_MCS; i++) {
                tempMC = new KingdaMC ();


               //以下两行定义创建的KingdaMC形状大小随机
                tempMC.scaleX = Math.random();
                tempMC.scaleY = tempMC.scaleX;
               //以下两行定义创建的KingdaMC位置在舞台上随机。
                tempMC.x = Math.round(Math.random() * (this.stage.stageWidth - tempMC.width));
                tempMC.y = Math.round(Math.random() * (this.stage.stageHeight - tempMC.height));
                addChild(tempMC);
            }
        }
    }
}
```

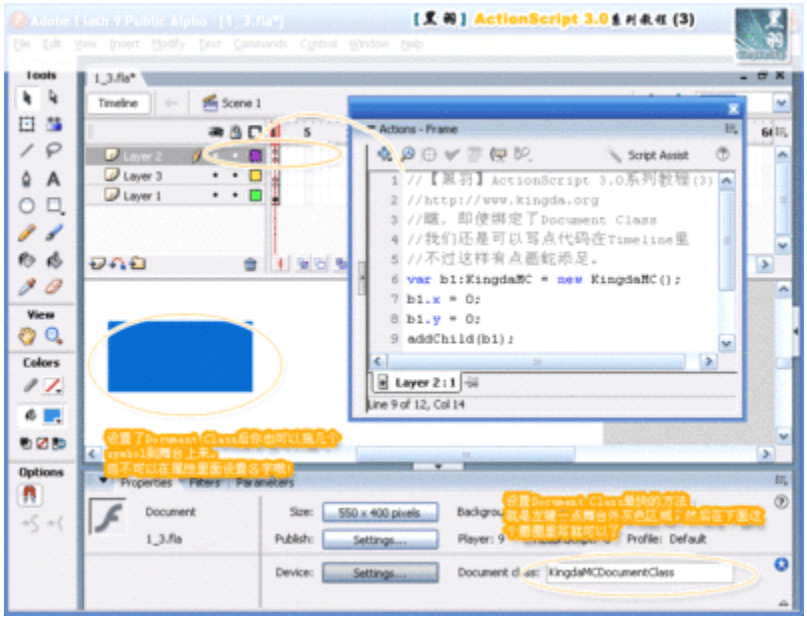
它要干的事很简单，就是用我们之前的KingdaMC类在舞台上创建十个KingdaMC实例，大小随

机，位置随机。

**2.用上之前的Kingda.fla，绑定KingdaMCDocumentClass。**

确保库中已有设好KingdaMC类链接的symbol。

设置方法如图：




先把第一帧代码全部删掉。Ctrl+Enter测试一把。是不是有十个MC出来啊。

然后再像图中那样加上第一帧代码，和舞台元件乐一乐。就是不按照Document Class规矩干干，呵呵。

**那么它到底是什么？**

简单的说，是一个用来取代原有散乱的时间轴代码的标准解决办法。

当一个Fla文件被编译时，Document Class就被主动建立了。我们可以理解为Flash IDE偷偷加上了如下代码：

```
//path表示Document Class的包，如上一节中说过的org.kingda;

//DocClass是指Document Class的名字

import path.DocClass;

var TouTouDe:DocClass = new DocClass();
```

//实例名TouTouDe是我乱写的，哈哈。可能是默认的instance1之类的名字

这样的好处是所有的时间轴逻辑都剥离到Document Class类文件中，一旦成为一个独立的类文件，那么很方便的可以和CVS软件结合起来。对团队合作(Designer + Coder)来说是一大利好。对新手和小项目来说，没什么优点和必要。

下面说点高级讯息和运用：

**有什么坏处？**

坏处大大的有。那是相当的不习惯。

所有的舞台上的元件都别直接用属性来命名（比如说随便命名为BigKingda），不然就会出现以下出错讯息：

ReferenceError: Error #1056: 无法为 KingdaMCDocumentClass 创建属性 BigKingda。

当然这个限制只在与舞台上的元件，在元件类部直接命名子元件没有问题，在Timeline写相关的代码没问题。

**重要：Document Class初始化顺序怎样？**

黑羽个人的研究结果：一旦设置了Document Class，那么初始化顺序如下：

**Document Class初始化 > 主时间轴代码执行 > 舞台元件绑定的类初始化 > 元件的时间轴代码初始化 >..**

下面的层次按这样的规律循环。

Ok了，三大特色到此详细介绍完了。

尽量讲的浅俗易懂，大家明白就好。而且黑羽也加上了一些高级讯息供老鸟参考。喜欢的顶一把！^_^

下次开始AS3.0的本质之旅：

AS 3.0的新数据类型及本质



# AS3.0教程(4):爽快使用XML

为什么放弃AS2.0选择AS3.0?如果只允许我说三个理由。那么AS3.0对XML的近乎完美的支持绝对是其中一个。

简单说说AS3.0中对于XML支持的不同吧：

**A.**AS2.0对XML的支持勉勉强强，将就着可以用。而AS3.0中对XML的支持是全方位的，极其强大和灵活的。

**B.**AS2.0对XML的支持不是内建的(build-in)，也并非基于ECMAScript for XML(E4X)标准。而AS3.0中对XML的支持符合E4X标准，它的设计有三个优点：

1. 简易。包括操作和可读性。你会发现AS3.0中对于XML的操作犹如对一个普通Object对象一样浅显易懂。语句非常浅白流畅。
2. 连续性。其各个功能的设计和AS3.0其余的部分思想一致，易于理解。
3. 熟悉。操作符和操作逻辑对我们来说都相当熟悉易用。

在AS2.0时代，为了解决这部分的问题

**C.效率。**

效率包括两方面，开发效率，和代码执行效率。开发效率的论述见上。 AS3.0对于XML的执行效率远远高过没有内建XML支持的AS2.0。

XML的输入

在AS2.0时代，在代码行中输入XML代码是一种痛苦。如果不是从文件中读取，那么我们就要忍受一长串挤在一块儿的字符串。

而在AS3.0中，太简单了。直接按照XML的内容输即可，想换行就换行，想Tab就Tab，就一个字，爽。

新建一个fla,选中第一帧，F9打开动作面板，输入如下代码：

```
//【黑羽】ActionScript 3.0系列教程(4)

//http://www.kingda.org

//例1

var kingdaXML:XML =

<tutorial>

<item id='1'>

<level>2</level>

<title> First touch of Flash 9</title>
```

```
</item>
<item id='2'>
<level>3</level>
<title> Binding Classes</title>
</item>
<item id='3'>
<level>4</level>
<title>Document Class</title>
</item>
</tutorial>
trace (kingdaXML.item[1].level);    //output:3
//例2
var kS:String = "<root><txt>this is a test</txt></root>";
var kXML:XML = new XML(kS);
trace (kXML.txt); //output:this is    a test;
```

例1中注意到没，直接写XML内容在后面，想换行就换行，想tab就tab，多爽。不想AS2.0中写string时，换个行就不行了。
写完这一句后，我们所写出的类似于string的形式立刻就被Flash理解成了XML对象了，所以我们马上就可以用"."操作符来访问相应的属性。本例中访问了第2个item节点的level值。
这么简便直观的访问方式是不是比 AS2.0中那千遍一律的childNodes要好得多？

不过要注意，最后可以加";"结束。但我为了XML的视觉美观没有加。这个没有关系，编译时不会考虑这一点。

事实上只要你喜欢，AS1.0, 2.0, 3.0中语句结束都可以不加";"号。但是这并不是一个好的编程习惯，更不符合严谨的自我语法要求。因此我建议，除了XML可以不加外，其余的都应该加，呵呵。

例2展示了如何将一个包含了XML内容的字符串转换成XML对象。用的是XML的构造函数转换的。

AS3更有趣的是，可以使用已有的变量来直接构造XML，带来方便的编程特性。如下例。

```
var rootNodeName :String = "site";

var subNodeName    :String = "orgin";

var subNodeContent :String = "Kingda's Blog";

var attributeName :String = "url"

var attributeValue :String = "http://www.kingda.org";

var extXML:XML =

<{rootNodeName} {attributeName}={attributeValue}>

<{subNodeName}>{subNodeContent}</{subNodeName}>

</{rootNodeName}>;

trace (extXML.toString());

/*output:

<site url="http://www.kingda.org">

<orgin>Kingda's Blog</orgin>

</site>

*/
```

要点就是要把变量用"{}"括起来，并且设置属性时不要再加引号了。

这个特性黑羽非常喜欢。

XML的外部读取

包括读取外部xml文件，和通过URL读取xml。AS3.0中不像2.0那样集成了一个load()。
AS3.0在架构上就设计了所有与外部打交道的都由URLrequest对象来进行，数据都由URLloader对象来接受。这个我们会在下一部分教程详细讲解。这一次只要知道这样的架构设计是深思熟虑，且简洁优美的即可。

```
    var myXML:XML = new XML();



//初始化XML地址，可以是本地的"xxx.xml"，也可以是如下的URL地址。

    var XML_URL:String = "http://www.kingda.org/blog/index.xml";            //我的Blog RSS Feed

var myXMLURL:URLRequest = new URLRequest(XML_URL);

    var myLoader:URLLoader = new URLLoader(myXMLURL);

//添加装载完成侦听器,

    //Event.COMPLETE的值是"complete",直接用此字符串也可以。

    myLoader.addEventListener(Event.COMPLETE, xmlLoaded);

function xmlLoaded(evtObj:Event) {

    myXML = XML(myLoader.data);

    trace("数据装载完成.");

    trace (myXML);

    }
```

XML的操作。

精彩的部分到了。详细看我下面的例子代码。

1.查询

```
//显示level为4的节点的title值

    trace (kingdaXML.item.(level == 4).title);

    //output：Document Class

//显示level>2的节点的title值，本处结果大于1，所以是一个XML Array。

    trace (kingdaXML.item.(level > 2).title);

    /*output:

    <title>Binding Classes</title>

    <title>Document Class</title>

    */


//使用属性用@开头即可。真方便。

    trace (kingdaXML.item.(level > 2).@id);

    //output:23

    //这儿要注意，实际上是2,3。一个Array.

//也可以用属性来做判断
```

```
trace (kingdaXML.item.(@id > 1).title);
```

2.添加或者修改属性

方便的不能再方便，直接写即可。爽翻天啊。

```
//把id == 1的节点level值改为2
kingdaXML.item.(@id==1).level = 2;
//把id==1的节点添加一个属性  page
kingdaXML.item.(@id==1).page = 100;
trace (kingdaXML.item.(@id==1));
```

3.按某条件插入节点

```
var newNode1:XML = <item id='2.5'><level>0</level><title>None</title></item>
var newNode2:XML = <item id='1.5'><level>0</level><title>None</title></item>
//把newNode1插入到id==2的节点后面
kingdaXML = kingdaXML.insertChildAfter(kingdaXML.item.(@id==2), newNode1);
//把newNode1插入到id==2的节点前面
kingdaXML = kingdaXML.insertChildBefore(kingdaXML.item.(@id==2), newNode2);
trace (kingdaXML);
```

XML的高级操作。

常用的操作上面已经介绍的很清楚了。高级操作则是留给对XML应用更深的兄弟们。
几点注意：
1.在AS3.0中， XML类的ignoreWhitespace默认为true。
2.AS3.0支持对comments的直接操作。但默认

```
XML.ignoreComments = false;

var kingdaXML:XML =

<item>

<!-- comment    1-->

<!-- comment    2-->

</item>;

trace(kingdaXML.toXMLString());   //默认为true时，不会显示comment的
```

访问comment用

```
trace(kingdaXML.comments()[1].toXMLString());
```

3.XML支持克隆。
使用copy()可以得到一份现有XML的值拷贝。

```
var kingdaCopy:XML = kingdaXML.copy();
```

对kingdaCopy操作就不会影响kingdaXML对象了。
4.极有用的descendants函数返回一个XMLList对象，包括所有的子节点。

设 ignoreComments = false; 和 ignoreProcessingInstructions = false 后， 连 comments 和 process instructions也会包含在这个XMLList对象中。
运用示例如下：

```
XML.ignoreComments = false;

var xml:XML =

<body>

<!-- comment -->

text1

<a>

<b>text2</b>

</a>

</body>;

trace(xml.descendants("*").length()); // 5

trace(xml.descendants("*")[0]); // // <!-- comment -->

trace(xml.descendants("*")[1].toXMLString()); // text1

trace(xml.descendants("a").toXMLString()); // <a><b>text2</b></a>

trace(xml.descendants("b").toXMLString()); // <b>text2</b>
```

还有太多的XML有用操作功能了(如对namespace的操作)。用到时再去翻参考书吧。
以上的介绍可以满足绝大部分运用了。

打完收工，歇歇。

对了AS2.0已有的XML类，在3.0中变成了XMLDocument类，使用方法不变。便于AS2.0程序移植。其余不推荐。

下一次讲3.0的Web交互模型和运用吧

# AS3.0教程(5):强大的事件机制(上)

（如蒙转载，请留下我的Blog链接：www.kingda.org， thx）

Event机制作为重头戏，在ActionScript 3.0中加强了很多。更加统一、易用、标准、灵活。
ActionScript 2.0中有众多的事件实现机制：

**回顾和比较**

AS1.0玩家最爱用onClipEvent(), on()，又方便又直接。缺点在于逻辑分散到了各个舞台元件中，难以管理和维护。更加别说代码重用了。别跟俺说可以Copy, Paste，这不叫重用，这叫低级。
AS2.0中，增加了一些事件处理机制：
1.回调函数: onLoad, onComplete等。相信兄弟们最熟悉的应该就是XML.onload回调函数了。
2.事件侦听器型：
这个就是addListener(), addEventListener()这种类型的。发送事件有的是内置，有的通过dispatchEvent().
嗯，这种事件机制基本上就和AS3.0很像了。自从使用了ActionScript 2.0来开发项目，**黑羽**就尽可能的多用这种事件处理机制。这个习惯很好，基本上让**黑羽**对3.0的机制很快适应过来。
说道这里，**黑羽**要多扯几句2.0，抱怨一下它的不足，再让大家看看AS3.0的光明大道。

(1).2.0中除了UIComponent能自己发送事件，绝大部分类不能自己发送事件的，比如MovieClip, 或者一些自定义的类。**黑羽**还制作了一个EventSender的事件发送类来解决。

当然你可以通过扩展来解决MovieClip这些类来解决，但是在**一些轻量级或者特殊运用中**，还是用**黑羽**这个EventSender类更加方便。
比如说，你突然需要舞台上某个A_mc的运行到第20帧时发送一个"finished"事件出来，并且希望另外某个B_mc能够捕捉到这个事件，那么用一般的扩展方法不知道有多么麻烦！
（1.要重做一个带有事件发送功能的类和A_mc通过某种方式绑定。2.同时确保在B_mc中要能访问到发送事件的对象并addEventListenr）。
而用俺的EventSender类非常简单，A_mc中写 EventSender.send("finished", this)， B_mc中写EventSender.addListener("finished", listenerFunc)，并可以通过event.target属性直接定位A_mc，真是简单的不能再简单了。(请尽量以正规方法为主，不推荐频繁使用，不是好的编程习惯)


(2)2.0中，侦听器的记忆是"有毛病"的。如果是新手，会经常觉得侦听器函数的this关键字指向飘忽不定，常常弄错。而且看看高手的代码，一会儿是Object做侦听器，一会儿是function做侦听器，真是让人头痛。其实MacroMedia也很头痛，所以就出了Delegate这个官方类（补丁？）来解决这些问题。

(3)侦听器注册方式也有两种，一种是addListener(), 如Key, 一种是addEventListener(), 如UIComponent类。为什么要这样搞两种？MacroMedia无辜的望着我，喃喃道偶也不完全清楚。

超人来了，那就是ActionScript 3.0事件处理机制：

(1)统一。全部一色用addEventListener().
(2)所有的可视对象都可以接受和发送事件。
AS3.0的类继承设计是深思熟虑的，所有的可视对象所属类都是DisplayObject的子类，DisplayObject又是EventDispatcher的子类。因此它们就都可以玩Event了，所以说，有个好的老子就是好啊。
而且有了崭新的事件冒泡机制，可以使事件层层上递到最上层的Stage，绝好的功能！
有了以上两点：黑羽的EventSender类也可以歇菜了。
(3)侦听器统一使用Function,不再使用Object了。 同时this关键字的记忆力“大大增强”,Delegate类也可以下岗了。

Event涉及到的内容极多，面很广。下面黑羽将从以下几个方面讲起：

一、如何接收事件？如何做到AS3.0的标准事件编程。
二、如何发送自定义事件？如何在OOP中正确使用AS3.0强大灵活的事件架构。
三种方式及其优劣，以及在何种情况下使用。
(1)用继承EventDispatcher实现
(2)用复合EventDispatcher实例来实现。
(3)用接口IEventDispatcher实现
三、如何使用冒泡机制(即官方所称的Event Flow机制)？以及冒泡机制的原理。
四、Event的其他高级应用。


# 【黑羽】AS3.0教程(6):强大的事件机制(2)

（如蒙转载，请留下我的Blog链接：www.kingda.org， thx）

今天Google小查了一下，居然有1060个网站转载了俺的系列教程。还包括了Blueidea, flash8, 5dmedia等顶级Web设计大站。非常开心，动力十足，呵呵。
其实到目前为止，我的3.0开发学习笔记已经有3万字左右的东西了，但都是些纲要和代码，整理成文，尤其是用比较有条理和易懂的方式写出来会比较慢。况且开发中的代码是随手拈来，但写教程的代码往往要经过一些改写，使得重点更突出。所以如果更新有点慢，请耐心一些，呵呵。

**如何接收事件？如何做到AS3.0的标准事件编程？**

Event改变的部分很多，这两天有空时，黑羽就在想怎样用一个有条理的方式来讲解Event和它相关的诸多内容，让我们感到比较容易理解，记忆和接受。
我准备这样来讲解：
先给个接受事件的代码例子。在例子中，指出：
（1） Event对象发生了什么变化
（2） addEventListener语法的不同，原因，和const型必要性和用法
（3） Listener和As2.0有何不同，和this关键字的“改进了的记忆力”

黑羽一贯的风格，先来一个例子。我很想给个短一点的代码例子，但是要达到清楚，全面和标准的示范，我还是决定采用这个Document Class的示例。我会在每个代码段注一些注释，大家不明白的地方回贴说一下，我会尽量解答。

Document Class的含义和相关用法并不难，忘了的兄弟看我第3篇教程:
AS3.0教程(3):Document Class特色为我们带来了什么？

好，come on baby.

新建一个as文件，拷贝以下代码，命名为AddListener.as。
新建一个fla，命名为 "黑羽黑羽我爱你.fla"(本教程推荐使用，倘若不遵从可能导致喝凉水塞牙泡MM被踢炒股被套等严重后果，霍哈哈。Just kidding。)。设置它的文档类(Document Class)为AddListener.

```
//【黑羽】ActionScript 3.0系列教程(6)

//http://www.kingda.org

package {

    import flash.display.Sprite;

    import flash.events.MouseEvent;



  //哈哈，看到了没，Document Class不仅可以扩展MovieClip，也可以扩展Sprite

  //package里面的类名要和文件名相同

    public class AddListener extends Sprite {

        public function AddListener() {



            //用package外面定义的类KingdaSprite创建一个实例，由于同文件中，所以不用import啦

            var outsideChild:KingdaSprite = new KingdaSprite(0x00FF00, "outside_sprite");
```

```
            addChild(outsideChild);//没有了这一句，你啥都看不到。


                outsideChild.addEventListener(MouseEvent.CLICK, inclassHandler);//注册类里面的侦听器

                outsideChild.addEventListener(MouseEvent.CLICK, outsideHandler);//注册类外面的侦听器

            }


            private function inclassHandler(event:MouseEvent):void {

                trace("类里面的侦听器侦听到MouseEvent事件: " + event);

        trace("this关键字指向:"+this);

            }


        }

}


function outsideHandler(event:MouseEvent):void {

        trace("类外面的侦听器侦听到MouseEvent事件: " + event);

        trace("this关键字指向:"+this);

}


import flash.display.Sprite;
```

```
import flash.events.MouseEvent;


//这个类就是画一个矩形，然后你点击这个矩形会发出标准鼠标click的事件

class KingdaSprite extends Sprite {
    public var nickname:String;
    public var ColorNum:uint;


        //colorNumber就是#ffcc00这种类型的数，在AS3中推荐用新的uint型来标记它
        public function KingdaSprite(colorNumber:uint, nameString:String) {
                ColorNum = colorNumber;
                nickname = nameString;
                graphics.beginFill(ColorNum);
                graphics.drawRect(0,0,100,100);
                graphics.endFill();
        }
}
```

鼠标一点击创建出来的绿色矩形，会看到输出

类里面的侦听器侦听到MouseEvent事件: [MouseEvent type="click" bubbles=true cancelable=false eventPhase=2 localX=64 localY=80 stageX=64 stageY=80 relatedObject=null ctrlKey=false

altKey=false shiftKey=false delta=0]

this关键字指向:[object AddListener]

类外面的侦听器侦听到MouseEvent事件: [MouseEvent type="click" bubbles=true cancelable=false eventPhase=2 localX=64 localY=80 stageX=64 stageY=80 relatedObject=null ctrlKey=false altKey=false shiftKey=false delta=0]

this关键字指向:[object global]

演示完毕，我们来讲第一个话题

（1） Event对象发生了什么变化

AS2.0中创建event 对象是很随意的，只要这个对象有一个String属性叫做type的就可以了，甚至连target都可以省掉。不要以为这不规范不应该做，看看Flash类源码，Macromedia的程序员可不只一次的这样使用过。所谓上梁不正下梁歪，就是这样。(其实严格说也没什么不对，有时候是不需要target。但没有标准就是不好)

然后你在Event Object中爱添加什么就添加什么。极其自由，也就等于极其混乱。

**3.0不同了，所有的事件必须都要继承自Event这个类(全饰名称flash.events.Event)。不然，哼哼，事件发送是不能成功地。**

说说Event这个类，那叫一个NB。AS3.0的基石类是Object，Event是直接继承自Object的，开国大佬级别的。它里面定义了一些基本的事件名称，比如ACTIVATE（FlashPlayer得到系统焦点时事件），ADDED（对象被添加到显示时发送的事件）。AS3.0中有个好功能是cancel事件，但Event中这些基本事件统统是不能被cancel的，听起来似乎很NB。这不细说了，Event太多内容了。以后写高级内容时在提到如何运用吧。那叫一个爽字了的！要想Flash玩的转，Event类必须很精通。正所谓江湖人称："平生不识Event,就称闪客也枉然。"

**这样有什么好处？**

好处太多了，最大的好处是一，规范了事件的定义，二，大量由此衍生的类省去了我们大量的时间和重复开发的成本。

规范事件的定义放到下一部分讲。至于内置的Event子类好处，大家请看上面fla的运行输出：

MouseEvent就是内建的一个Event子类，它的好处一看它的内容就明白：

从AS3.0起，任何一次点击事件我们都可以得到：

1.事件发生时鼠标的MC相对坐标(localX=64 localY=80)和绝对坐标(stageX=64 stageY=80)。 爽不爽？

2.ctrl键，shift键，alt键有没有按下。你试试按着ctrl键再点击一下方块，那么ctrlKey就为true了。

爽不爽？

(其余参数不解释，涉及到高级运用。部分会在Event最后一节提到该怎么运用)
光这一个MouseEvent类，黑羽怎么着都觉得值人民币20块。

（2） addEventListener语法的不同，和const型必要性

这一节很重要，不过现在12点半了。精力不太好了，但又不想草草了事。明儿接着写，质量第一。数数也有小两千字了，先发出来顶顶先。

# 【黑羽】AS3.0教程(7):强大的事件机制(3)

（如蒙转载，请留下我的Blog链接：www.kingda.org， thx）

（2） addEventListener语法的不同，原因，和const型必要性和用法

本例如果是AS2.0，那么代码是这样写的：

```
outsideChild.addEventListener("click", inclassHandler)
```

本例是3.0,如果你trace一下其中的MouseEvent.CLICK,输出的也是字符串"click"。
看起来AddEventListener的方法和以前也没什么大的出入。和这世界上大多数相同的事物一下，表象的类似却掩盖着本质的巨大差异。

我们先从最小的差异讲起：

**1.使用类静态属性，用const定义事件字符串名称变量**

AS3.0中用了一个新的关键字定义了事件名称字符串变量，代码是

```
public static const CLICK:String = "click"
```

const，是英文constant的缩写，意思使不变的，常量。那就意思很明白了，一旦这种类型的常量被定义就不可再更改。好处通俗的说就是规范好项目，规范你自己，也规范任何其他项目人员，动不了这个变量。
你可能还是会不屑一顾，这么小的一个改动，对我没什么用！
可我的AS2.0开发血泪经验是，我曾经花了一个下午来找bug，最后却发现是某个类的addEventlistener()里面的事件名称"click"手误打成了"cilck"。编译器可不管这个。
如果是3.0，你打成了MouseEvent.CILCK，那么编译时立刻会报错提醒你，多好啊。一个项目一

个人做十几个类还好办，自己认真点还能顶的住；如果是一个项目几个人几十个类，那么没有const和static的帮忙，管理事件类型还真是有点麻烦。即使实现了也没有AS3.0这么简单直观。
回到代码，我们要记住，日后我们开发自己的Event类时，也要像这样，用public static const来定义我们自己的事件名称。
如何自定义自己的事件，我会在EventDispatch那一节讲述。

看看代码，我们还发现AS3.0中侦听器也发生变化了，只能用function来做侦听器，不再用Object。

**2.addEventListener高级运用**

AS3.0中对侦听器的改进远远不止以上这些，看一看addEventListener的实现接口：

```
function addEventListener(eventName:String,
                          listener:Object,
                          useCapture:Boolean=false,
                          priority:Integer=0,
useWeakReference:Boolean=false):Boolean;
```

哇塞，有三个莫名奇妙的参数。可是当你知道这三个参数背后隐藏的巨大改进之后，相信你要大叫三个哇塞。
第一个神秘参数，目前暂不解释，埋个伏笔，留到EventDispatcher那一节，讲ActionScript3崭新的Event Flow事件流机制.
第二个参数：优先级。 有趣吧，在ActionScript 3.0中我们可以控制事件的优先级，从而达到控制function侦听器的执行顺序。如果你不填这个参数，那么事件默认为同一个级别0，事件的执行按先来后到的天经地义的顺序。
如果设为1，那么事件级别降一个档次，稍后执行。数字越高优先级越高。级别可以为负数。
**(注意： Flex Builder 2 Beta3中事件级别是越低越高)**

好好利用这个功能吧，在AS2.0中要想达到同样效果可不知道要多费多少力气！

第三个参数，讲的是是否设为弱引用。
兄弟们初学As2.0时一定经常忘了在删除Listener对象时，却忘了removeEventListener吧。这会导致很多莫名奇妙的情况发生。也是最常见的诡异bug种类之一。即使老鸟偶尔也会被阴一把。
最郁闷的是这种情况往往不被发现，在后台默默的消耗大量资源。
现在可以用弱引用可以在某种程度上解决这种bug。

设为true，就是告诉垃圾回收器，这个侦听器function的引用是弱引用。一旦这个侦听器在运行时只剩下了这一个弱引用，那么垃圾回收器可以不理它，直接把它回收咯屁了，节省资源。
**从AS3.0引入这个弱引用这个概念就可以看出，AS3.0是如何的重视资源管理和有效率的运用。一个标准的重量级的程序语言必须具有这样的特征。**
**现在AS3.0有了！**

下一节：

（3） Listener和As2.0有何不同，和this关键字的“改进了的记忆力”

# 【黑羽】AS3.0教程(8):强大的事件机制(4)

本节内容：

1.弱引用的使用原则
2.listener的不同和this关键字的指向。

继续上次的话题。在讲listener 和 this关键字之前，我们先来讲讲一个高级话题：
**弱引用的使用原则。**

继续上次的话题。在讲listener 和 this关键字之前，我们先来讲讲一个高级话题：

**弱引用的使用原则。**

新手可以不看，因为暂时用不到。但这个话题很有必要。当设计大型RIA应用程序时，弱引用必须要了解。

弱引用从原则上来讲，其引入是为了防止无意识的对象保留（unintentional object retention）引起的内存泄漏。
什么是无意识的对象保留？一般情况下，对象的逻辑生命周期和实际生命周期应当相同。但是在某些情况下，比如事件侦听器机制，我们可能会创建一个引用，它在内存中生存的时间比我们预期的要长很多。比如说，下面的例子中，即使删掉了侦听器lis，但事件还是能被继续捕捉。这是由于并没有removeEventListenr，那么系统中还会保留着对侦听器的引用。
如果没有把addEventListener中对侦听器的持有改成弱引用，那么这个侦听器会一直存在、耗掉内存不被人发觉。但如果设成了弱引用，那么垃圾回收器过段时间后会将侦听器占用的内存回收，此时侦听器从物理意义上才真正的被销毁了。
在销毁之前，这个侦听器还会继续存在，继续作用。这就是为什么下例中虽然已经把useWeakReference设为了true，但是还是会有作用。
那么什么时侯垃圾回收器(Garbage Collector)才会来回收呢？
唔，这要看它高兴了，快的话10分钟，慢的话一个月。哈哈，开玩笑，垃圾回收器的工作时间咋一看确实是不可测的，但也有规律可循。但这个问题探讨在本系列教程中显得过于艰深，黑羽会单独撰文来说明垃圾回收器的部分工作机制。

那么推论来了，我们凭什么要把useWeakReference设为false呢？除非我们有意要让某个侦听器在失去了其他所有引用的时候还要工作，才有可能这样做。但谁会有这样疯狂的想法呢？道哥还

是包世宏？

所以，我建议，大家不妨养成习惯，把useWeakReference设为true。
当然最好的做法是始终记得不用侦听器了，一定要removeEventListener。在这一点上，我们要坚持“始乱终弃”!!

```
//这个是Document Class，建个fla，绑定这个class。忘了，就看我第三篇教程。

package

{

 import flash.display.MovieClip;

 import flash.events.EventDispatcher;

 import flash.events.MouseEvent;



 public class LearnEvents extends MovieClip

 {

  function LearnEvents() {

   var lis:Function = function () {

    trace ("听到了鼠标Click事件！");

   }

   var kingda_s:KingdaSprite = new KingdaSprite(0xFFCC00, "kingdaSquare");

   addChild(kingda_s);

   kingda_s.addEventListener(MouseEvent.CLICK, lis, false, 0, true); //瞧，最后一个参数，已经把弱引用设为true了
```

```
    lis = null;   //这一句，我已经从逻辑上删掉lis了，大家作证啊

    trace ("侦听器还在吗？："+ lis);   //lis也确实指向null了。但你只要继续点击方块，你会发现Click事件仍然被捕捉到。



   }

  }

}

import flash.display.Sprite;

import flash.events.MouseEvent;



//这个类就是画一个矩形，

class KingdaSprite extends Sprite {

  public var nickname:String;

  public var ColorNum:uint;



  //colorNumber就是#ffcc00这种类型的数，在AS3中推荐用新的uint型来标记它

  public function KingdaSprite(colorNumber:uint, nameString:String) {

    ColorNum = colorNumber;

    nickname = nameString;

    graphics.beginFill(ColorNum);
```

```
    graphics.drawRect(0,0,100,100);
    graphics.endFill();
  }

}
```

**Listener和As2.0有何不同，和this关键字的“改进了的记忆力”**

DOM Level 3中规定的是用Object来做listener。该Object有用来处理事件的方法(method)。
AS3虽然是按照DOM Level 3事件机制设计的，但也不完全听话，有自己独立的想法。在AS3中，侦听器就是函数。也只能用函数来侦听。
其实在ActionScript3中，Function实质也是一种特殊的Object，和AS2.0有相似之处。
见我的文章：
ActionScript高级技巧:深入了解Function
因此AS3.0为什么限定用Function，这和它的架构有关，不细说了。
但有趣的是，在AS3.0中，对IEventDispatcher的定义中，仍然按照DOM3标准用Object来做Listener。见黑羽上一篇教程和随后的评论。

那么问题来了，function中this关键字的指向会怎样？

AS2.0开发者对AS2.0 事件机制中 this关键字的水性杨花应该深有认识。如果用function做侦听器，那么谁发出事件，this就指向谁。这就等于对象之间乱搞关系，啊呀呀。所以MM派了一个Delegate代理警察类来管理。唉，糜乱的岁月不堪回首啊。

```
//ActionScript 2.0例子:拷贝以下代码到第一帧，拖一个button到舞台，命名为kingda_btn;
function lisFunc() {
  trace ("亲爱的，你会指向谁:"+ this.name);     //我们本意应当是指向_root;
}

kingda_btn.addEventListener("click", lisFunc); //点一下button，看看指向谁。
```

回到ActionScript 3.0，来看看坚定的夫妻关系，黑羽总结为：嫁鸡随鸡，嫁狗随狗，什么都不嫁，那就属于global。我靠，读一下，还挺押韵。大家也好记。

简单的解释一下，函数在哪个对象里(应该叫method)，那么this就指向谁。不在对象里，那么就指向global。

```
  //【黑羽】ActionScript 3.0系列教程
  //http://www.kingda.org
package {
    import flash.display.Sprite;
    import flash.events.MouseEvent;


    public class AddListener extends Sprite {

        public function AddListener() {


            //用package外面定义的类KingdaSprite创建一个实例，由于同文件中，所以不用import啦

            var outsideChild:KingdaSprite = new KingdaSprite(0x00FF00, "outside_sprite");

        addChild(outsideChild);//没有了这一句，你啥都看不到。


            outsideChild.addEventListener(MouseEvent.CLICK, inclassHandler);//注册类里面的侦听器

            outsideChild.addEventListener(MouseEvent.CLICK, outsideHandler);//注册类外面的侦听器

        }


        private function inclassHandler(event:MouseEvent):void {
```

```
                trace("类里面的侦听器侦听到MouseEvent事件: " + event);
        trace("this关键字指向:"+this);
            }

    }
}

function outsideHandler(event:MouseEvent):void {
        trace("类外面的侦听器侦听到MouseEvent事件: " + event);
        trace("this关键字指向:"+this);
}

import flash.display.Sprite;
import flash.events.MouseEvent;

//这个类就是画一个矩形，然后你点击这个矩形会发出标准鼠标click的事件
class KingdaSprite extends Sprite {
   public var nickname:String;
   public var ColorNum:uint;
```

```
        //colorNumber就是#ffcc00这种类型的数，在AS3中推荐用新的uint型来标记它
        public function KingdaSprite(colorNumber:uint, nameString:String) {
            ColorNum = colorNumber;
            nickname = nameString;
            graphics.beginFill(ColorNum);
            graphics.drawRect(0,0,100,100);
            graphics.endFill();
        }
}
```

# 【黑羽】AS3.0教程(9):强大的事件机制(5)

（如蒙转载，请留下我的Blog链接：www.kingda.org， thx）

有兄弟反映，似乎AS3的事件机制有些复杂。

在我看来，编程上 “复杂”这个词一般有两种定义：实现麻烦 ，或者内容众多。
AS3中的事件机制其实现并不麻烦，逻辑更加清楚简单，因此不是“实现麻烦”这一类。
那应该指的是“内容众多”这个意思。黑羽倒觉得"内容众多"往往是褒义词，意味着API丰富，控制范围和深度大。那么一旦得其要领，即思路通畅，记忆深刻，也就不会觉得“复杂”了。

我也会尽量写的简单通俗，照顾新手。但我又要同时考虑到AS2老手们的需求，不让他们打瞌睡。所以AS初学者 看到不懂的地方可以跳过，多用用Event后，有需要再回过头看看我教程的其它部分，一定会有收获。

拿今天要讲的事件发送来说，我预计写以下内容：

1. EventDispatcher和Event的简介
2. 回顾AS2.0事件发送
3. 继承EventDispatcher进行事件发送。
4. 合成EventDispatcher进行事件发送。
5. 实现IEventDispatcher接口来进行事件发送。 与设计模式中的装饰器模式相似。

那么新手看第一，第二，第三部分已经足够应付一般应用。AS2老手们要看看第四部分。 开发大型项目的AS开发者，则第五部分必看。

1.EventDispatcher和Event的简介。

AS3中Object是万物之宗母，且生养众多，共有220多个子类。Event(事件类)和EventDispatcher(事件发送者类)就是在这一代之中。 Event和EventDispatcher是事件机制的两大主角，二者缺一不可。
Event类及其子类的功能就是提供各种具体的事件供EventDispatcher使用，不能干别的。
EventDispatcher则是要让它所有的子类都能发送事件。
那么，AS3就让所有要发送事件的类都统统继承于EventDispatcher。而需要发送事件的类毫无疑问，都是些很重要的类，比如URLLoader，DisplayObject。
其中DisplayObject更是一代霸父，所有可视对象，比如MovieClip,Spirit等等等等统统都是它的子孙。
因此，父以子贵，EventDispatcher在Object的第一代子女中，位高权重，影响极大。

2. 回顾 AS2.0中发送事件

AS2.0中发送事件是怎么干的？
在类中留几个函数对象，如 dispatchEvent:Function，再搞一个EventDispatcher.initialize()来初始化一下要发送事件的类，然后，就搞定了。.....好简单。
但AS3中更简单，更加标准。

3.继承EventDispatcher进行事件发送。

继承是最方便最快的一种。
刚刚说过了，所有的可视对象都继承DisplayObject,是EventDispatcher的孙子辈，因此都可以直接使用dispatchEvent()来发送事件了。
看代码，我们让一个普通的MovieClip， KingdaMC，来发送一个莫名其妙的事件，比如说"KingdaPlaySC"。

```
KingdaMC.dispatchEvent(new Event("KingdaPlaySC"));
```

现在我们要让一个我们自己的类来发送事件。

```
  //【黑羽】ActionScript 3.0系列教程
  //http://www.kingda.org
//以下为一个名叫KingdaSampleClass的Document Class,请自行和一个Fla绑定。
//如果忘了怎么弄，看我第三篇教程

package {
```

```
import flash.display.Sprite;
import flash.events.Event;

public class KingdaSampleClass extends Sprite {

        public function KingdaSampleClass() {
                //生成一个KingdaSampleDispatcher的实例
var dispatcher:KingdaSampleDispatcher = new KingdaSampleDispatcher();

//标准的实现
dispatcher.addEventListener(KingdaSampleDispatcher.ACTION, actionHandler);
                dispatcher.doSomething();

//可以用直接字符串标识事件类型，但推荐使用静态加const的字符串
dispatcher.addEventListener("KingdaPlaySC", anotherHandler);

//直接用new Event生成一个新的事件对象，该对象的事件类型为"KingdaPlaySC"
dispatcher.dispatchEvent(new Event("KingdaPlaySC"));
//输出：
//action的侦听器: [Event type="action" bubbles=false cancelable=false eventPhase=2]
```

```
    //KingdaPlaySC 的 侦 听 器 : [Event type="KingdaPlaySC" bubbles=false cancelable=false eventPhase=2]

            }


            private function actionHandler(event:Event):void {

                trace("action的侦听器: " + event);

            }


   private function anotherHandler(event:Event):void {

                trace("KingdaPlaySC的侦听器: " + event);

            }

      }

}


import flash.events.EventDispatcher;

import flash.events.Event;


class KingdaSampleDispatcher extends EventDispatcher {

      public static var ACTION:String = "action";


      //如果你需要在自己类中某个方法中发送事件，那么示例如下
```

```
 public function doSomething():void {

   //你的代码.....



   //发送事件

                    dispatchEvent(new Event(KingdaSampleDispatcher.ACTION));

      }

}
```

# 【黑羽】AS3.0教程(10):强大的事件机制(6)

（如蒙转载，请留下我的Blog链接：www.kingda.org， thx）

(2007-10-30update: 呵呵，更正了一个手误。最后一个代码例子中的最尾部的花括号应该放在第一个class的结尾后。这样下面的kingdaClass才是包外类。否则编译不会通过。sorry^^ 谢谢daozi的细心。)
事件机制写的太多了，我自己都有点烦了。

但没办法，  ，太重要了。而且AS3做了这么多好的改进，值得我们去一一探寻，给我们日后的编程带来极大的便利。ActionScript 初学者，本节可以跳过不看。
ActionScript 2熟练工应当看看，有些价值。

今儿讲掉

4. 合成EventDispatcher进行事件发送。
5. 实现IEventDispatcher接口来进行事件发送。 与设计模式中的装饰器模式相似。
这样事件的发送和接受，就可以讲完了。

那么，事件部分就这样完了？没有！你晕，我也同晕。因为还有一个很重要的特性，Event flow机制还没讲。这就是我所说的事件冒泡机制。给我们编程带来了莫大的方便。

好，下面先讲：

**4. 合成EventDispatcher进行事件发送。**

什么情况下用合成EventDispatcher来发送Event呢？
一般发生在某个较复杂的类里面。

这个类可能是因为本身已经继承了其它类，无法再继承EventDispatcher。
如果仅仅是因为这个原因，那么我更加建议**使用 实现IEventDispatcher接口来进行事件发送。**
但如果原因不止如此，比如，我们不愿意这个类不是一个单纯事件发送类，而是在执行某个方法(method)，比如doSomething()时，附带的发送一些事件。
这些事件发送者往往是这个类的组成部分，一些更小的类，通常是Sprite等。
那么用这种做法就比较合理。
看代码例子

```
   //【黑羽】ActionScript 3.0系列教程

   //http://www.kingda.org

//以下为一个名叫KingdaSampleClass的Document Class,请自行和一个Fla绑定。

//如果忘了怎么弄，看我第三篇教程

package {

      import flash.display.Sprite;

      import flash.events.Event;

      import flash.events.EventDispatcher;



      public class LearnCompositeEvents extends Sprite {

   public function LearnCompositeEvents() {

     var kingdaObj:KingdaClass = new KingdaClass();



                                  //一定要用kingdaObj.getSender()来返回事件发送对象，才能
addEventListener

                                     kingdaObj.getSender().addEventListener(KingdaClass.ACTION,
lisFunc);
```

```
    kingdaObj.doSomething();
                                        //输出:
                                        //doSomething
                                        //listened:yeahyeah




  }
  //侦听器
  private function lisFunc(evtObj:Event):void {
    trace ("listened:"+evtObj.type);
  }
}
import flash.events.EventDispatcher;
import flash.events.Event;

class KingdaClass extends EventDispatcher {
  public var _dispatcher:EventDispatcher;
  public static const ACTION:String = "yeahyeah";
```

```
  public function KingdaClass() {
    initSender();
  }

  private function initSender():void {
      _dispatcher = new EventDispatcher();
  }

                    //调用一个专门的方法(method)来返回发送事件的EventDispatcher。
  public function getSender():EventDispatcher {
    return _dispatcher;
  }
  public function doSomething():void {
    trace("doSomething");
                                //除了以下两行发送事件，还可以写入其它你要干的事儿。灵活。
    var evtObj:Event = new Event(KingdaClass.ACTION);
    _dispatcher.dispatchEvent(evtObj);
  }
}
```

}

**5.实现IEventDispatcher接口来进行事件发送。**

在哪种情况下使用？
类可能是因为本身已经继承了其它类，无法再继承EventDispatcher。
而我们恰恰希望它能实现EventDispatcher类所有功能，比如说addEventListener, hasListener等等，看起来简直和继承EventDispatcher没什么分别。
那么OK,我建议使用 实现IEventDispatcher接口来进行事件发送。
**其实质是一个装饰器模式(Decorator)，以对客户端透明的方式扩展对象功能，是继承关系的一个替代方案。其关键在于扩展是完全透明的，使用起来和继承父类几乎没什么区别。**

具体方法
由于IEventDispatcher需要实现5个接口，addEventListener，hasListener, willTrigger,removeEventListener,hasEventListener,那么我们的装饰类也必须实现这五个接口。
其余看代码

优点：

1.类的用户完全感觉不到差别
2.在被包装的方法中还可以加入其它自己希望加进去的动作，比如，在addEventListenr方法中可以再插入一个计数，看看到底被add了多少次，超过某些次后，调用某个方法等等。
总而言之，给我们带来了极大的灵活性。这就是装饰器模式的好处。

```
package {

    import flash.display.Sprite;

    import flash.events.Event;

    import flash.events.MouseEvent;

    import flash.events.EventDispatcher;



    public class LearnDecoratorEvents extends Sprite {

  public function LearnDecoratorEvents() {
```

```
    var kingdaObj:KingdaClass = new KingdaClass();

    kingdaObj.addEventListener(KingdaClass.ACTION, lisFunc); //用起来和EventDispatcher对象一样哦，呵呵。



    var evtObj:Event = new Event(KingdaClass.ACTION);

    kingdaObj.dispatchEvent(evtObj);//确实一样吧  :)

    //输出:listened:yeahyeah

   }



   private function lisFunc(evtObj:Event):void {

    trace ("listened:"+evtObj.type);

   }

  }

}

  import flash.events.IEventDispatcher;

  import flash.events.EventDispatcher;

  import flash.events.Event;



  class KingdaClass implements IEventDispatcher{

   public var _dispatcher:EventDispatcher;
```

```
public static const ACTION:String = "yeahyeah";


public function KingdaClass() {
  // other ....
  initSender();
}


private function initSender():void {
  _dispatcher = new EventDispatcher(this);
}
        //哈哈，在实现接口时还可以乘机干点别的，比如我喜欢吧useWeakReference设为true
    public function addEventListener(type:String, listener:Function, useCapture:Boolean = false, priority:int = 0, useWeakReference:Boolean = true):void{
                    // do other things;
          _dispatcher.addEventListener(type, listener, useCapture, priority, useWeakReference);
    }


    public function dispatchEvent(evt:Event):Boolean{
          // do other things;
  return _dispatcher.dispatchEvent(evt);
    }
```

```
        public function hasEventListener(type:String):Boolean{
                        // do other things;
                        return _dispatcher.hasEventListener(type);
        }

        public function removeEventListener(type:String, listener:Function, useCapture:Boolean = false):void{
                // do other things;
                _dispatcher.removeEventListener(type, listener, useCapture);
        }

        public function willTrigger(type:String):Boolean {
                // do other things;
                return _dispatcher.willTrigger(type);
        }
 }
```

# 【黑羽】AS3.0教程(11): 视觉元件精要(1)

.

题外话：

不少兄弟问我，AS3教程咋不出了哩？其实看我博客的兄弟们应该都知道，黑羽目前的Flash RIA整体网站，接近竣工，11月即将推出。结尾工作实在太多，从十一到上周末，基本上放假都在加班。一些博客或者论坛上兄弟的问题都没有精力回复，抱歉了。

上次还在5d上看到版主鼓动大家给我打气，让我继续，真是非常不好意思。

这个周末偶得空闲，遂集中精力，将自己开发笔记整理了一篇放上来。项目发布后，更新会正常。



# 彻底了解AS3视觉元件架构

DisplayObject，直译为视觉对象，意为可以被看到的对象。

视觉是Flash 成功的主要基石。当我赏析ActionScript 3 的所有视觉元件类型和其整体构架时，感到非常满意。可以看出，这次整体的架构设计是深思熟虑的结果。与其他语言，比如C#，相比有自己强烈的特色，是对Flash视觉行为贴身定做的结果。

ActionScript 3的架构乍一看很复杂，不包括UIComponent的子类，就有7到8个层，20多个莫名奇妙的类。头疼！但实际上，它的设计是非常的简洁优雅，远远比ActionScript 2一个MovieClip打天下强太多了。实际上，只要真正弄清了它的设计思路，就可以高屋建瓴，一览无余，会发现这个架构逻辑清晰，非常的易懂易记。所以，先抛开ActionScript 3 的帮助文件，我们一起来看看为什么要设计这样一个架构，搞出20多个怪胎出来。

（如蒙转载，请留下我的Blog链接：www.kingda.org， thx）

先来追忆一把ActionScript 2中无所不能的先贤: MovieClip（影片剪辑）。 这位兄台无所不能：可在其中画矢量图，可在其中贴位图；可在其中做影片，也可嵌套子影片；偶尔用来加载，闲来客串按钮；三教九流皆可放，肚皮天下第一广。它的父亲何人，原来是元始天尊Object。

这样的玩意儿，新手用起来很爽，大大节约了脑细胞。但任何一个有过大型OOP项目经验的老手，都会毫不犹豫的指出，这样的架构设计是失败和混乱的。代价是巨大的。MovieClip类公开的属性和方法共有一百多个（自己数数）！居然直接继承根类！居然拥有这样多公开(public)属性和功能的类！居然应用范围如此广泛！

首当其冲，其第一弱点就是系统资源的浪费。

举个例子， 我新建一个空MovieClip A，只是想让它做个容器，好在里面放几个有内容子MC。这样我操作A的位置和渐变时，子MC会统一变化。这样的经验大家都有吧。可就是这么一个简单的纯容器A，ActionScript 2&1都会毫不犹豫的把MovieClip所有的属性和方法都赋给A。谁让A是Movie C lip类的实例呢？可在这个应用上，我们要A的其他90多个功能干什么呢？而且还不算最耗资源的内建的对Timeline的支持！大家想想，我们每天都在创建MC。但事实上我们做开发时，创建的MC有多少用到了大部分的功能和Timeline？只有一部分的通过Flash创建的MovieClip才需要时间轴的支持，其余大部分根本不需要时间轴支持。这样的设计是不是有问题呢？

痛批了一顿ActionScript 3之前的MC后，我们不得不说几句公道话：这样的错误是有其历史局限

性的，我们不能苛求古人。且看现在的视觉元件架构，那叫一个爽。爽，就爽在系统设计师对整个Flash视觉系统的抽象上。抽象和解构的功力很深！不得不佩服！系统各个超类和子类的设计划分，职责清晰，稳健高效，堪称优雅！我看.Net FrameWork 的System.Drawing架构设计时都没有这个感觉。毕竟Flash是靠视觉起家，与视觉动画交互打交道最深阿。

下面来欣赏ActionScript 3 的元件架构。

ActionScript 3 中所有可以被看到的视觉元件都统一于DisplayObject，即其子类的实例。DisplayObject是一个抽象类，不能生成实例。从系统架构设计上来说，这样的超类设计是常识。DisplayObject，我在AS3.0教程(5):强大的事件机制(1)中讲过，继承于EventDispatcher类，也就意味着所有的DisplayObject子类都可以发送事件了。

啊哈，DisplayObject下面一层的抽象就精彩了，架构设计师的原意是将所有视觉元件分为两大类：可以接受人机交互事件的，和不可以接受人机交互事件的。所以就有了InteractiveObject类和非InteractiveObject类之分。由于非InteractiveObject的几个类之间差别太大，也抽象不出什么共同点，所以，干脆就分成了InteractiveObject的六个同级兄弟 AVM1Movie, Bitmap, MorphShape, Shape, StaticText, Video。但黑羽认为从系统的优雅性出发，不妨就设一个UnInteractiveObject的超类，将这六个孩子放在这个超类的下面。还便于日后的功能扩展。

在讲最重要的InteractiveObject之前，我们先把这几个不伦不类的兄弟先扫掉。这几个子类中，我们把它分为可以代码创建的，和不可以代码接触的。所谓不可以代码创建的，是指只能通过Flash创作工具来创建的。和以前一样，StaticText还是不可以用代码实现。另外一个是MorphShape，这个东东是指在Flash中创建Shape形变时，由Flash Player自动生成的，同样的，代码无法实现。

剩下的几个都是可以用代码创建的：Bitmap，位图对象，可以通过BitmapData对象来创建，也可以从外部载入，比如通过loader。Shape，形状，专门用来绘制矢量图的，通过Graphic对象创建的。Video,视频对象，专门用来播放视频的，可以来自文件也可以来自网络流媒体。

下面要说到的是AVM1Movie，所谓AVM1Movie，意思就是说Actionscript Virtual Machine 1所支持的SWF影片，也就是ActionScript 1和2的影片。由于ActionScript 3 采用的是AVM2，所以和AVM1影片无法跨脚本交流，必须要把它同AVM2 swf影片区分开来，所以有个这个类。关于这个，感兴趣的兄弟看我的这篇文：小谈ActionScript 3.0与AS 2.0，1.0的swf兼容性。这个一般我们不用关心，也不用直接接触，一旦我们加载一个AVM1老影片到AVM2 swf中时，Flash Player 会自动创建一个AVM1Movie的实例包装这个swf。我们打交道的往往是装载AVM1 swf的容器元件。

到了InteractiveObject下一层了，这一层共有三个类。

这一层的抽象理念又要赞一下。架构设计师又用了一个容器和非容器的概念来区分视觉元件。所谓容器，就是可以在其中加载其他的DisplayObject子类对象。当然也包括它的叔叔们，即非InteractiveObject的那几个类。所谓非容器，那就是说不能在它的视图里面加入其他的DisplayObject了。“容器”，这个公共性质实在太重要了，在这一层才这样抽象出来实在很高明，很到位。

那么老套路，先讲讲非容器的两个类，TextField和SimpleButton。 TextField，就基本上是我们熟悉的动态文本框，这里暂不细说。来说说SimpleButton，这个名字虽然和我们在ActionScript 2中碰到的SimpleButton一样，但实际上二者有很大的差别了。实质上， ActionScript 3 中

SimpleButton这个类是将Button这个重要常用的UI控件单独提出作为一类，而不是像以前和MovieClip混淆不清。谁都知道，在ActionScript 2和1中，只要改改MovieClip前几个帧的标签，这个MovieClip就变得和Button一样了。关于SimpleButton的使用，可以说非常简洁实用，这个放在后面细说。我在这里所要强调的是Button和MovieClip不是一个性质。虽然ActionScript 3中Sprite和其子类也可以通过buttonMode来做出和Butoon相似的行为，但原理是不同的。

剩下的就是DisplayObjectContainer类了。DisplayObjectContainer的所有子类对象，都可以在其中添加其他DisplayObject子类对象。但要注意一点，DisplayObjectContainer本身也是一个抽象类，不可以生成实例。这是出于架构设计稳健性扩展性的考虑，和将DisplayObject设计成抽象类的原因一样，不多说了。

重要的是看看它的几个子类，Sprite, Loader, Stage。 Stage，就是舞台，所有的视觉元件都是在它之中，当之无愧是最终容器，放在这一层也是合情合理。Loader就有趣了，它把以前MovieClip装载的部分全部抽象分离出来了。所有和外部资源的加载，都是通过Loader来进行的。而Loader也不能直接和网络资源打交道，要通过专门的URLRequest对象来进行，各司其职，非常好。Loader能干什么？装载swf, 和各种图片。注意，视频还是要通过Video类来进行，直接addChild到各种Container中，不一定要放在Loader中。

下面来了Sprite，这个3.0中我们打交道最多的容器了。一句话，它是去掉了时间轴的MovieClip（即MovieClip被阉掉了）。如我开头例子所说，倘若我们只是为了创建一个容器，那么Sprite是首选。甚至可以说，我们这些写代码的开发人员，90%以上的情况都只需要和Sprite打交道。含有时间轴的MovieClip一般是Flash工具创建出来的，往往只需要加载就可以了。准确的说，Sprite比ActionScript 2中的MovieClip不止少一个TimeLine，如装载。Sprite中也含有Graphic对象，这意味着，它也可以直接在其中代码绘图。但我们始终要记住，Sprite不同于Shape,区别就在于Sprite是容器，而Shape不是。从代码角度说，就是，Sprite可以addChild(),但Shape不可以。

都说到这儿了，我们亲爱的MovieClip还不见踪影，到底在哪儿了呢？其实就在Sprite的下一层。Sprite下一层中，共有四个子类，MovieClip是其中一个。大家可以想到，现在的MovieClip重要性大不如前了，主要就是代表用Flash创建的含有时间轴的影片。常用的gotoAndStop, currentFrame等等这些属性现在才能碰到了。

其余三个子类，也是来头不小，最牛的就是其中的FlexSprite。虽然它的改动只是变了一下toString()函数，但它却是所有Flex组件的共同基石。著名的UIComponent的老爸现在就是它了。UIComponent何许人也？天下组件，皆由它出。黑羽大胆预测，我们在Flash 9中所要用到的控件，也就是现在的Flex里面的组件，即mx.controls里面的所有组件。此类不可小看。

另两个子类Preloader和DownloadProgressBar，已经不属于flash.display包了，而是属于mx.preloaders包，主要管理的是下载进度和共享库的下载等功能，不予细说了。

到此，ActionScript 3 主要的视觉元件架构已经理出了一个清晰的脉络，设计的原因，理由，思路都做了一一剖析，希望大家喜欢。其实InteractiveObject下的另外两根枝条TextField和SimpleButton也有很重要的地位，在后续教程中会一一细说。

下面到了老鸟时间了，和老鸟们分享一下我对ActionScript 3 视觉架构的分析心得。可以看出的是，抽象的界限非常明确，应用的模式也比较统一。应用的模式就是OOP编程中最常用的模式之一：Composite模式。所有的DisplayContainer属于Composite模式中的树枝，非DisplayContainer部分的都是树叶。树枝中都有相应的聚集对象。

另外要指出的是，MovieClip和Sprite再也不像以前2.0的MovieClip一样，对内部的不同子对象给出包装接口。比如Shape下的Graphic对象， Sprite下面的SoundTransform对象， Loader下的LoaderInfo对象，等等等等。我本来迷惑，为何不用装饰器模式，让这些元件的行为和ActionScript 2更加相似一些，过渡的痛苦少一些，但转念一想明白了架构设计师的苦心。就是要用这样清清楚楚，明明白白的Composite来加强所有开发者OOD时分工协作，更加快的熟悉和运用新的强大的架构。与这个优点相比，这么一点过渡的痛苦又算得了什么呢？

关于ActionScript 3视觉编程后续部分和具体的编程例子，会陆续放出。

（11月份更新：才发现在拷贝Word文档到博客，并编辑内容时，不小心在最后一段遗漏了几行字，现在加上。请转载的网站也加以更新，不然恐有歧义，读者难以理解。）

黑羽 ActionScript 3 系列教程 之 12 http://www.kingda.org/ **黑羽ActionScript 3 系列教程之12**
（欢迎转载！但请注明来自www.kingda.org，thx ^_^）
本章教程首发于www.j2eemx.com和www.kingda.org。

# 12. 异常和错误的捕捉和处理

（上）

在ECMAScript Edition 4 draft中规定异常（Error）类要作为内置类。ActionScript 3中的顶级包(Top Level)中，Error及其子类共占据了11个位置，超过了核心类总数三分之一。ActionScript 3对异常处理的改进是非常显著的。在AVM2宣传中，对异常处理机制的良好支持一直是主要宣传点。

掌握异常处理机制，不论是对于程序易维护性、代码健壮性、用户友好性都至关重要。这也是为什么ActionScript 3在异常处理机制上下如此大的功夫之原因。

而且异常处理机制并不像初学者想象的那么抽象和复杂，实际上只要理解了它的设计用意和思想，那么我们会发现它是非常实用而且简洁的。

下面我们就慢慢的来了解这个日后经常要打交道的好帮手。

## 12.1. 什么是异常和错误？

**异常和错误**是指程序执行时遇到的任何错误情况或意外行为。

似乎有些抽象。没有关系，最快了解异常的方法就是先看一个出现异常的代码例子。然后，我们再来看一下异常的详细概念、ActionScript 3 中的实现以及与其他语言的不同之处。

### 12.1.1. 一个简单的异常例子

**示例 12-1 简单的异常例子**

```
package org.kingda.codesample.error
{
import flash.display.Sprite;
public class ErrorSample extends Sprite {
public function ErrorSample() {
var dataSource:OtherData = new OtherData();
try {
dataSource.getArray().push("kingda");
} catch (error:Error) {
trace (error);
//输出：
```

```
//TypeError: Error #1009: 无法访问空对象引用的属性或方法。
}
}
}
}
internal class OtherData {
private var foo:Array;
public function OtherData() {
//...
}
public function getArray():Array {
return foo; //注意：foo并没有被初始化
}
}
```

在上面的ErrorSample中，使用了某个第三方的类的实例（dataSource）所返回的Array对象。在本例中，这个第三方的类就是同一个包下的OtherData实例。现实开发中，这个第三方的类可能处在不同的包中，甚至是别人已经编译好的组件，甚至不能被我们查看或者修改代码的。

在try{}中的代码试图操作dataSource返回的Array对象，本例中的操作是希望在中多加入一个字符串“kingda”。结果怎料到，出于某种原因，dataSource其实返回的是一个null。（本例中的原因是未经初始化。）如果没有try-catch，那么会导致程序没有办法正常执行下去，一旦执行，Flash Debug Player就会抛出一个Error(异常)：TypeError: Error #1009: 无法访问空对象引用的属性或方法。

这是一个好事儿，至少通知我们出了错。而不是像在ActionScript 2 中那样“蒙混过关”，静悄悄的失败让我们一头雾水，调试时也不知从何查起。

读者可能已经从try-catch的字面意思已经猜出，try{}就是试试try下面花括号括住的代码块，执行一下，如果发生异常就会有Error对象抛出，那么就被catch语句抓住，使用catch语句中的代码来处理这个错误。本例中的处理方法是将Error对象的内容trace出来。

这就是一个很简单的代码发生异常的例子。

异常的具体定义是什么？还有哪些时候会引发异常？这就是我们下一小节要讲述的内容。

### 12.1.2. 异常的概念和在ActionScript 3中的实现

可将异常理解成是一类**事件**（Event），表示程序出错了的消息。

它传送一些AVM问题、故障及未按预想的逻辑执行的相关信息。在ActionScript 3中，使用Error类及其子类的实例来统一表示异常和错误。这些实例对象中，一般包括异

常信息、异常ID、堆栈跟踪数据等。通过这些Error实例将信息从应用程序的一部分发送到另一部分。

由于ActionScript 3 没有对线程的支持，所以异常分为两种：同步异常和异步异常。Error和其子类一般用来处理同步时的程序异常。在异步处理时，则交给所产生的异常用相关的**异常事件(ErrorEvent)**。

异常事件并不是`Error`的子类，而是`Event`的子类，这一部分我们将放在以后教程的事件处理机制中介绍。

本章教程关注于`Error`，开始，后文中如无特别说明，将以**Error(异常)**[1]来指代异常和错误。

在Java中，Throwable 类是 Java 语言中所有错误或异常的超类。两个子类Error 和 Exception，通常用于指示发生了异常情况。Error用于指示合理的应用程序不应该试图捕获的严重问题。Exception指示合理的应用程序想要捕获的条件。

但ActionScript 3 并没有这样的划分，所有错误和异常都是Error类或其子类的实例。处于顶级包(Top Level)中的Error子类们近似于Java中的Exception。flash.errors包中的Error子类代表发生的严重问题，类似于Java中的Error。而且ActionScript 3中的Error不支持异常链，不包含cause(原因)。

在 .NET Framework 中，异常是从 Exception 类类继承的对象。从这个方面说，ActionScript 3中的Error与.NET Framework中的Exception地位对等。但是.NET中Exception分了两个子类: 一个是 SystemException，从其 派生的预定义公共语言运行库异常类；一个是 ApplicationException，从其派生的用户定义的应用程序异常类。在ActionScript 3 中不是这样.，所有AVM系统内置异常类和用户的自定义类都是直接从Error派生的。

ActionScript 3 不支持线程的概念，但有同步和异步之分。异步应用产生的异常将发出异常事件（ErrorEvent），而不是异常(Error)。要加以注意。

以下这些情况都可以引发异常：

您的代码或调用的代码中有错误，操作系统资源不可用，AVM遇到意外情况，等等。对于这些情况，ActionScript 3 应用程序可以从其中一些恢复，而对于另一些，则不能恢复。尽管可以从大多数应用程序异常中恢复，但一般不能从大多数AVM异常中恢复。

### 12.1.3. 使用异常处理机制的好处

使用异常好处一个最主要的原因是让程序更加健壮，更加易于维护。比如，从一些异常状态中修复程序的运行。比如当异常发生时(比如所读取资源出错，运行时出现空对象时)，提示用户或者改变程序的运行路线，确保程序正常运行。

异常处理机制，对设计一个“用户友好”的RIA程序，至关重要。

但是初初一想，这些情况下“似乎可能好像”可以不用异常也可以处理。直接在异常发生处解决不好吗？

1 Error按字面翻译，应当译为错误。但由于ActionScript 3 中`Error`类实际代表了通常所说的异常和错误。本教程大部分程序语言的习惯，将`Error`称为异常。

那么，为什么要引入异常？为什么不在异常出现位置直接写入相关逻辑来处理呢？
作为大部分语言所采用的异常处理机制，当然有很多原因和优点。
比如，有时，根本不能修改代码来处理异常。原因可能有异常来自于第三方组件或者源码的缺陷。如果是编译好的组件，则不能访问其代码。如果是源码，可能由于其架构庞大，又没有详细文档，查错困难，不如在自己的代码中集中处理这些异常。
有时需要在程序整个逻辑系统中交流错误信息，以便按统一方式来处理问题。在大型的RIA软件设计中，经常要使用该策略。
有时候，出现了异常但可能有若干种处理它的方式，但您不知道使用哪一种。这时，可以通过try-catch-finally将异常处理托付给异常所在方法的调用者。方法调用者，一般更加了解异常的原因，以及异常发生的环境，可以更加准确的选择处理方法。
这样的总结有点抽象，如果您不太明白没有关系，后文中有详细的异常处理实例和处理原则。相信您可以在具体的实例应用中体会到异常的好处。
不过在这之前，要先简单介绍一下try-catch-finally的使用方法。

## 12.2. 使用try-catch-finally处理异常

在本章教程开头的例子中已经讲述了一个具体的异常处理的代码例子。一旦我们认为某段代码可能会有潜在的异常发生时，那么就应该在这段代码包括在try花括号的代码块中。

常见的潜在的异常发生情景有：某对象为undefined，却对它进行操作；运行时访问了本不属于类定义的方法；运行时调用方法，类型不匹配；使用网络Socket连接时，跨越安全沙箱限制或者传入的端口参数不正确等等。这些在下一节：“ActionScript 3 中异常的层次和结构”有更加详细介绍。
先看一个简单的代码例子，处理Socket连接异常。

在ActionScript 3 中Socket的连接只允许连接65535以下的端口号。本例中，端口号居然是66666，虽然顺口却是非法的，因此会抛出一个SecurityError异常。

示例 12-2 Socket连接异常处理的简单示例

```
package org.kingda.codesample.error
{
import flash.display.Sprite;
import flash.net.Socket;
public class SecurityErrorExample extends Sprite
{
public function SecurityErrorExample() {
var targetServer:String = "www.kingda.org";
var port:uint = 66666;
try {
var socket:Socket = new Socket();
```

```
socket.connect(targetServer, port);
//注意一旦异常发生后，下面这句trace是不会执行到的。
trace("try end");
}
catch(e:SecurityError) {
trace(e);
//输出：SecurityError: Error #2003: 指定的 socket 端口号无效。
}
finally {
trace ("finally ended!");
}
}
}
}
```

下面，依次讲述try,catch, finally的含义和用法。

### 12.2.1. try

将可能会抛出异常的语句，放置在try里面。这是告诉编译器，我的这些语句可能会出现异常哦，你要小心点，使用特殊的机制来评估。一旦发生异常，将异常提交给catch语句，而不要“蒙混过关”，沉默失败(Silent Failure)。

要注意的是，不要一股脑儿将所有语句都放入try块。笔者认为尽可能只放可能出错的语句，这样try中的内容简洁，一眼就知道调试重点在哪里。

要特别注意的是，异常一旦发生，那么try块中的语句就会立即停止执行。换句话说，产生异常的语句后面所有其他语句都会停止执行。在示例 12-2中，由于异常发生了，try块中最后一句trace不会被执行。如果将端口号改到65535以下，异常不会发生，trace语句也就会执行了。但try块外面的语句不受影响，会继续执行。

那么，犯罪嫌疑语句已经放入try后，该找相关负责人来处理它们了。这些相关负责人就是catch块中的内容。

### 12.2.2. catch

catch就是抓住的意思。

一个try可以跟随几个catch，以便应付可能抛出的不同异常。

异常一旦发生，那么如果try{}后面跟着catch，那么实质上有以下几件事发生：

1.生成异常对象（Error类实例或其子类实例），try块中语句执行立刻中止。

2.AVM会按catch语句块出现的先后顺序，查找和异常对象对应的catch块。

3.一旦发现异常和catch()中定义的异常类型相符。就会被catch“抓住” 。抓住之后，就会交给这个catch块中的语句处理。

本例中，只是简单的将异常对象trace出来，在实际运用中，可能是在其中更换端口号，

再重新连接一遍；或者提示用户程序异常，需要退出。如果当前没法解决，比如无法决定有效的端口号，那么继续抛出一个异常，希望更高层的代码能捕捉到这个错误。

### 12.2.3. finally

不论抛出的异常是否被try{}后面跟随的catch语句捕捉到，finally中的语句一定都会执行。这就是finally的好处，无论try块中的语句是否碰到异常而中止，finally块中的语句执行不会受到影响。

一般情况下，是在finally中做一些释放资源，清除打扫的工作，比如关闭Socket的连接等。

finally是可选的，不是必须的。finally只能有一个。

### 12.2.4. try-catch-finally的语法规则

try-catch-finally异常处理机制必须符合本小节介绍语法规则，不然编译器会报错。

try后面至少要跟有一个catch语句块或者finally语句块。否则会报错“1073: 语法错误: 需要 catch 或 finally 子句。”

try后面可以跟有多个catch语句块。一个catch块处理一种异常类型。catch块的顺序应根据所处理的具体异常类型来排列。一般来说，从最具体的Error子类到最不具体的Error类顺序来先后排列。更加详细的原则请参见“处理异常的原则和方式”。

catch语句块里面可以再嵌套try-catch-finally的结构。

try,catch,finally这三个块中可以再次抛出异常。如何抛出异常？请看下一节。

## 12.3. 使用throw抛出异常

除了系统API可以抛出异常外，我们还可以在自己的代码中指定抛出异常。我们不仅可以抛出系统预定义好的异常类实例，还可以扩展Error或其子类生成自定义的异常类。

在ActionScript 3 中使用throw关键字来抛出异常。

Java用户注意，在ActionScript 3中，类方法抛出异常无需声明。throw关键字只用于在语句中抛出异常。

在继续讲述用throw抛出异常之前，先必须唠叨两句：抛出异常要谨慎使用。何时该处理异常，何时该抛出异常，本文在“处理异常的原则和方式”中有详细讲述。先给大家提个醒。

### 12.3.1. 抛出Error类或其子类的实例

抛出Error类或其子类的异常很简单。Error类的构造函数接受两个参数：第一个是字符串变量，作为异常的文字信息，默认为空；第二个是异常的数字ID，int型的，默认为0。两个参数都是可选的。从使用上来说一般都会定义一下异常的信息，即第一个变量。

第二个参数较少用到，如果需要自定义，那么建议避开1000到2152这段数字。因为这段数字已经被

ActionScript 3 内置的Runtime Error占用了。为了避免冲突，和考虑到ActionScript 3 日后可能添加更多的内置Runtime Error，笔者建议保险起见，避开1000-10000这一段。

具体有如下几种常用形式：

生成一个Error类或其子类的实例，用throw抛出。因为Error是动态类，所以某些情况下，我们可能希望再让它携带一些信息，再将它抛出。

生成一个匿名的Error或其子类的实例，用throw抛出。这种情况往往是只需要抛出异常即可。最多定义一下异常的信息。

将一些相关信息传给一个特定的方法（或者函数），由该方法（或者函数）根据这些信息来判断返回什么样的异常。这样的应用，比较灵活，代码集中，常见于比较复杂的环境中。值得推荐。

下面这个代码例子按顺序给出了这几种常用形式的代码。读者可以更换这几段代码顺序，来查看异常处理的结果。

**示例 12-3 抛出异常的几种常用形式**

```
package org.kingda.codesample.error
{
import flash.display.Sprite;
import flash.utils.*;
public class ThrowErrorSample extends Sprite
{
public function ThrowErrorSample() {
try {
//1.生成一个Error对象，并抛出
var CustomError_1:TypeError = new TypeError("Try the first error:", 1200);
throw CustomError_1;
//2.抛出一个匿名Error对象
throw new Error("Try the second error")
//3.抛出一个函数返回的Error对象
var errorCause:Number = 5000;
throw reportErrorFunc(errorCause);
} catch(e:Error) {
trace (e);
}
}
//特定的根据相关信息判断，并返回不同类型的Error对象
private function reportErrorFunc(eC:Number):Error {
if (!(eC is Number))
```

```
return new TypeError("CustemFuncError:Not a Number");
if (eC > 1000) {
return new Error("CustemFuncError:A big number error.");
} else {
return new Error("CustemFuncError:A small number error.");
}
}
}
}
```

### 12.3.2. 自定义异常

所谓自定义异常，就是通过扩展Error类，或者Error的子类来创建一个新的异常类。
为什么要自定义异常？
我们开发具体应用程序时，会碰到如下几种类似情况：
当某种不符合我们程序规定的对象（或者行为）出现时，我们希望抛出一种自定义异常，从而利用异常处理机制来管它。
这种对象（或者行为），本身符合语法，而且也不会引起任何系统内置异常。我们抛出异常，只是因为它不符合我们应用程序的要求。
比如说，我们从第三方数据源接收到一个格式良好的XML。我们RIA程序要求它包含三个名为action的子元素，但是这个XML并不符合这个要求。而且这很可能会引起我们后续程序出错。于是我们新建了一个CustomXMLError这个类，扩展自Error类。一旦发现验证失败，我们就抛出这个Error子类的实例，由专门catch这个CustomXMLError类实例的代码块捕捉到，并执行相关的处理。
某种系统内置异常出现了，但是在我们的程序中这种异常可能有多种含义。而且不同的含义中，需要在这个异常对象中加入不同的信息。而对不同信息又有不同的异常处理方式。那么这时候，一个好的做法就是扩展这个系统内置异常类，生成几个子类，各自代表不同含义下的异常。
比如，在示例 12-2中，我们得到了一个无效的端口号导致了连接失败。如果这个类中有三个方法调用了socket.connect()，虽然都会抛出SecurityError异常。但如果是调用第一个方法引起的，那么可能是由于A原因；调用了B方法引起的，可能是由于B原因引起的；如果是C方法，那么可能是C原因。而且这三种原因各不相同，我们需要有不同的信息加入到Error对象中。那么虽然抛出的都是SecurityError异常实例，但已经不能满足我们的需要了。这样我们就可以扩展SecurityError类，生成三个子类各自表示原因。

**示例 12-4 使用自定义异常的例子**

```
package org.kingda.codesample.error
{
```

```
import flash.display.Sprite;
import flash.net.Socket;
import mx.validators.ValidationResult;
public class SecurityErrorExample extends Sprite
{
var socket:Socket;
public function SecurityErrorExample() {
socket = new Socket();
var targetServer:String = "www.kingda.org";
var port:uint = 66666;
try {
connectMethodA(targetServer, port);
}
catch(e:CustomBError) {
trace (e);
}
catch(e:CustomAError) {
trace ("抓到A了:"+ e +"\t出错端口号:" + e.getPort());
//输出:抓到A了:SecurityError 出错端口号:66666
}
catch(e:SecurityError) {
trace(e);
}
}
private function connectMethodA(host:String, portNum:int):void {
//...A方法的其他代码....
try {
socket.connect(host, portNum);
} catch (e:SecurityError) {
var customAErr:CustomAError = new CustomAError();
customAErr.record(host, portNum);
throw customAErr;
}
}
private function connectMethodB(host:String, portNum:int):void {
//...B方法的其他代码....
try {
socket.connect(host, portNum);
```

```
} catch (e:SecurityError) {
throw new CustomBError();
}
}
}
}
//下面就是自定义的异常类
class CustomAError extends SecurityError {
//...CustomAError的一些代码....
private var infoForA:Object = new Object();
public function record (host:String, portNum:int):void {
infoForA.host = host;
infoForA.port = portNum;
//...其他代码....
}
public function getPort():int {
return infoForA.port;
}
}
class CustomBError extends SecurityError {
//...CustomBError的代码...
}
```

我们在connectMethodA方法中抛出了特有的CustomErrorA类的异常实例。可以看到CustomErrorA有自己的属性和方法。由于实际执行的是connectMethodA，所以抛出的异常不会当成CustomErrorB接受，而被专门管CustomErrorA的catch块捕捉到了，并使用了A异常的特有的方法getPort()所返回的信息。

在了解了try-catch-finally和throw异常的应用之后，我们需要深入的了解一下ActionScript 3 中的异常架构，以及常用的异常类。这样我们才可以最好的利用ActionScript 3 强大的异常处理系统。

(未完待续，下节正在整理中，一周内发布)

## 12.4. ActionScript 3 中异常的层次和结构

### 12.4.1. ActionScript 3中对异常的良好支援

### 12.4.2. Error相关类别的层次和架构

### 12.4.3. 最常碰到的几种异常

### 12.4.4. 自定义异常的资讯

## 12.5. 处理异常的原则和方式

### 12.5.1. 处理异常的几条原则

### 12.5.2. 处理异常的常用的几种方式

### 12.5.3. 4大提倡与4大忌讳

# 【黑羽】AS3教程（13）:AS3中的数据存取方式效率比较

今天又有一位朋友和我在MSN上聊起了AS3中存取数据效率的话题，突然想起以前曾在blueidea论坛上讨论过。这个话题还有蛮多人感兴趣，比较实用，让它在论坛上沉下去有些可惜。因此特地整理成帖，希望对大家有用。废话少说，看蕉：

使用如下几种存取数据方式进行存取100万次运算，所花时间依次为（单位为毫秒,ms）：

如下几种存取数据方式100万次存取运算效率依次为：

* 类定义中的属性（如public、private类属性） 135 毫秒

Point类对象 140 毫秒
Rectangle类对象(x等属性) 140ms
* Array类对象 270ms
* Object类对象 500 ms
动态类动态属性 550
* Rectangle类对象(left等属性) 700

* 自定义类getter/setter 1000
自定义类Function 1000
（以上测试结果是由eidiot测试的，感谢eidiot。稍微做了一点文字修改。见讨论原帖）

## 问题1：为什么会有速度差别？

最快的是第一类，速度是Array的两倍，Object的四倍，getter/setter的约八倍。存取坐标要用Point和Rectangle，比Array快的多。 Object和动态类效率较低，一般情况下请尽量使用密封类 。

测试用的自定义类是密封类，那么，public等类定义中定义好的属性比Array和Object快是正常的 。因为查询密封类的类成员（包括属性和方法）只需要查询TraitObj,加一个 prototype obj。而Array, Object都是动态类，不论是内存分配和name lookup都需要额外的开销。

同理，Point、Rectangle也是密封类，当然比Array和Object要快。
这是AS3新的内部实现机制的优点。如果在AS2.0中，则不会有差别。注，AS2的密封类和动态类的支持只存在于编译阶段，在运行时没有区别，无法区分。
有兴趣的朋友可以试试将AS3自定义类设成动态类，再使用动态属性和非动态属性存取10万次比较看看。

## 问题2：为什么有些类的属性效率会存在差异？比如Rectangle的x属性和left属性为何差别这么大？

Rectangle类除了x,y,width,height，其余属性全部是getter和setter，不是真正的属性。使用getter和setter时，函数调用需要一些开销，自然比真正的属性要慢一些。

## 问题3：为何动态类和密封类的类定义中的成员效率没有差别？

这个是正常的。并不费解。之前想问有没有区分动态成员和非动态成员，就是要弄清这个问题。

只要是我们在类定义中定义的成员（包括属性和方法），都在这个类对应的Trait obj中。动态类和密封类都一样。因此，密封类和动态类的属性，方法效率都会相同。具体的实现是通过 Slot 方式来实现，通过slot index来访问，效率高。

## 问题4：动态成员的效率和非动态成员的效率为何有差距？AS2中为何没有这样的区别？

因为动态类的动态属性不存在与trait obj中，而是在prototype obj中一个类似于哈希表的数据结构中。哈希表代价是name lookup效率明显低于slot方式，而且生成时系统的开销较大。因此AS3中，动态类的动态成员效率低于类定义中定义的成员。在AS2中，动态属性和非动态属性的区别只存在于编译阶段，运行时并没有区别。因为运行时，实质上所有对象成员都是以name为索引，name都存储在一张hash table（哈希表）中。所以AS2中没有AS3效率高，也没有这些区别。

## 问题5：private成员和public成员效率有没有差别？

访问private成员和访问public成员不会更快。实际上public, private, protected, internal等只是AS3中内嵌的四个命名空间而已，并不会对访问速度产生差异。他们在Trait obj中都是同样使用slot来存储的，访问也都是通过slot index。理论上说，我们使用自定义的命名空间来修饰类成员，那么这个类成员的访问速度也应该和内嵌的命名空间一样。有兄弟感兴趣的话，可以继续试试使用自定义命名空间来测试一下。

### 有趣的发现：看测试结果，似乎Rectangle实现的getter和setter速度，要比一般的getter和sette和自定义function要快300。这个真是很有趣。

解释：AS3中不少核心类的方法都是native（关键字）实现，即最终的实现是由AVM语言自然代码(native code)实现，而不是ActionScript代码实现。比如Object类，Class类等等，他们的方法都是native的，由相应的C++代码实现。因此，会比我们自定义类的效率要高不少。

黑羽可以肯定的是，核心类中的实现都是native code。但是，Rectangle类的功能并不复杂，而且处于flash.geom包，非核心类。我在想，难道连这个类的方法也是native实现吗？

如果测试多次也是这样，并且测试方法无偏差，那么基本上可以肯定Rectangle类中也是使用了native code。
那么做一个推论，是否只要是Flash API的类基本上都是由native code实现呢？
建议使用其他包中的类，也测试一下他们的getter和setter。如果Flash API中类的getter和setter速度一致，且比自定义类的快，那么大概可以下结论，所有Flash Player API（包括核心类和非核心类）真的都是用native code实现的

# ActionScript 3教程（外篇)

## 小谈ActionScript 3.0与AS 2.0，1.0的swf兼容性

众所周知，Flash Player 9中为了解决兼容性问题，内置了两个虚拟机。AVM1来对付AS1.0&2.0, AVM2用来专门处理AS3.0。(注: 本文中ActionScript 均简写成AS)

用AS3.0中的SWFLoader来装载swf, img非常方面爽快。于是黑羽就想到一个问题:
可不可以用SWFLoader来装载使用AS2.0的组件(或1.0)编译过的swf呢？
可不可以通过AS2.0来装载AS3.0的swf呢？如果可以，那么海量的Flex2组件会让大家爽死！
如果可以的话，那岂不是可以轻松重用以前编的2.0组件了吗？

如果真的可以，那么虚拟机到底怎么工作呢？AVM2与AVM1通信？？ 

经过黑羽的探索、试验、查询资料后，答案是折衷的，不是很爽。 听俺慢慢道来：

1.使用AS3.0确实可以加载AS2.0 或者 1.0的swf。但是AS3.0不可以访问加载swf中变量和函数。为了方便理解，我们可以想成两个虚拟机并行工作，但是不能通信。事实上，我猜实现机制可能就和这个差不多。LocalConnection。

2.使用AS2.0或1.0编写的swf是不可以加载AS3.0的。换句话说Flash 8＆Flex 1.5及之前所有工具生成的swf都不可以加载(load)AS 3.0 swf的。

3.如果想让AS2.0或1.0的swf与AS3.0 swf协同工作，那么AS2.0＆1.0的文件必须进行移植。就是说转成3.0。黑羽在yy:不知道Blaze出来后会不会有另存为自动转换成AS3.0的功能，霍霍。

4.单个的swf文件中是不能混合使用AS3.0＆AS2.0(或者1.0)的。 不会像AS2.0＆1.0那样混用了，毕竟是AS2.0&AS3.0是两个不相同的虚拟机。

一句话总结，就是Flash Player 9（AS3.0）可以加载以前的所有版本的swf,但是只是简单加载，不能访问AS2.0(或1.0)的swf内部变量＆函数，无法交互。

如果一定要和AS2.0或1.0的swf通讯的话，只能使用ExternalInterface类，具体教程看 http://www.flex2.org/node/97

# AS3.0教程(外1)：Flash 9 Alpha 调试须知

1.写KingdaMC.as的时候，使用f9的autoformat功能会报错。怎么回事？

2.引用这个as以后的fla也出现这样的警告。怎么回事？

问题由leo提出的，我解答如下：

1.写KingdaMC.as的时候，使用f9的autoformat功能会报错。为什么？ **Error** E:\LEO-CHOU\MyWorks\实用fla\f9测试\KingdaMC.as: Line 6: Syntax error.

package {

**Error** E:\LEO-CHOU\MyWorks\实用fla\f9测试\KingdaMC.as: Line 11: Attribute used outside class.

public class KingdaMC extends MovieClip {

**Error** E:\LEO-CHOU\MyWorks\实用fla\f9测试\KingdaMC.as: Line 20: The class or interface 'flash.events.MouseEvent' could not be loaded.

private function clickHandler(event:MouseEvent):void {

.....

Total ActionScript Errors: 6 Reported Errors: 6

答：AutoFormat必须在检查语法通过后才会执行。你使用“检查语法”的那个对号，也会出现同样错误。

但如果使用fla来编译，就不会出现相关错误讯息。
这可能是Flash 9 alpha的不足，还没做完呢，可以理解。^ ^

2.引用这个as以后的fla也出现这样的警告。WARNING: ActionScript 3.0 does not support actions on button or MovieClip instances. All scripts on object instances will be ignored.

Kingda created: instance1。。。
July 3, 2006 11:10 AM

答：这是因为你可能把代码错写到MovieClip上了，应该写在第一帧的脚本面板中。

补充一句，在AS3.0中，不支持onClip event类型的事件了。所以会有"All scripts on object instances will be ignored"这样的提示。 ^ ^

# AS3.0教程(外2)：用flash9试用版编译as3的问题

问题及表现：

我在d盘安装了flash9试用版，在论坛上复制了第一个as3,发现不能编译，请高手解答：

```
package learnAs3
{
import flash.display.Sprite;
public class HelloAs3 extends Sprite
{
public function HelloAs3(){
trace("Hello World!");
}
}
}
```

提示：

**Error** G:\flash9\1\learnAs3.as: Line 1: Syntax error.
package learnAS3

**Error** G:\flash9\1\learnAs3.as: Line 2: ActionScript 2.0 class scripts may only define class or interface constructs.

{

**Error** G:\flash9\1\learnAs3.as: Line 4: Attribute used outside class.

public class HelloAs3 extends Sprite

**Error** G:\flash9\1\learnAs3.as: Line 5: The class or interface 'flash.display.Sprite' could not be loaded.

{

**Error** G:\flash9\1\learnAs3.as: Line 10: ActionScript 2.0 class scripts may only define class or interface constructs.

}

**Error** G:\flash9\1\learnAs3.as: Line 1: Syntax error.

package learnAs3

**Error** G:\flash9\1\learnAs3.as: Line 4: Attribute used outside class.

public class HelloAs3 extends Sprite

**Error** G:\flash9\1\learnAs3.as: Line 5: The class or interface 'flash.display.Sprite' could not be loaded.

{

**Error** G:\flash9\1\learnAs3.as: Line 10: ActionScript 2.0 class scripts may only define class or interface constructs.

}

Total ActionScript Errors: 9 Reported Errors: 9

//
改为as2格式，可以编译：
class HelloAs3 {
public function HelloAs3() {
trace("Hello World!");
}
}

//

这是什么原因呢？

解答：

既然写了package learnAs3 { class HelloAs3

那么你的文件名应当改成 HelloAs3，且应当放在 fla所在目录下的 learnAs3目录中
然后，你的fla Document Class应该设为： learnAs3.HelloAs3
Ctrl+Enter,就Ok 了。

# AS3.0教程(外3)：如何移除容器下所有子显示对象？

容器中的子显示对象分为两类：

处于显示列表中的子显示对象。被numChildren所记录的。
由容器graphics对象绘制出来的矢量图。这个矢量图不属于Shape类型，不在容器的显示列表中，不被numChildren所记录。而是做为容器的背景矢量图存在，始终处于最后面。
因此，准确的说，如果要移除容器下所有子显示对象，就要分别移除这两类对象。

比如有一个容器，其下有8个sprite子对象，和一个graphics绘制出的图像：

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

addChild(con);



for(var n:int =0; n<8; n++) {

  con.addChild(new Sprite());

}

con.graphics.beginFill(0xFF0000);

con.graphics.drawCircle(100,100,100);

con.graphics.endFill();

trace ("con's children:" + con.numChildren);

//输出：con's children:8      graphics对象绘制的图形果然不在其中。
```

先说，如何移除显示列表中的显示对象。

在AS3中，容器对子对象的深度管理是自动的，深度是连续的。
每移除指定深度的显示对象，那么其他子显示对象会移动并填充这个空位。因此，代码编写需要注意这一点。
容易误写的移除代码：

```
//以下是错误代码：

for(var i:int=0; i
  con.removeChildAt(i);

}

trace (con.numChildren); //输出：4
```

错误原因：每次删除1个子对象，后面的就填补上深度索引空缺，numChildren又会动态改变，因此出错。这样，实际只移除了四个显示对象，分别是0，2，4，6这四个索引的显示对象。因此，是错误的代码。

正确方式：

```
for(var i:int=con.numChildren-1; i>=0; i--) {

  con.removeChildAt(0);

}
```

或：

```
var len:uint = con.numChildren;

while(len>0) {

  con.removeChildAt(0);
```

len--;

}

由于显示列表会自动移动显示对象填充空缺，所以只移除0索引的位置即可。这样效率最高。
到这儿，大家可能会发现，虽然numChildren已经为0，但是graphics绘制的圆形仍然存在。所以最后，应当调用一句下面的代码把它也清除掉，才算真正的移除了容器下所有的显示内容。

con.graphics.clear();

# 仔细说说Flash Player Update 3 Beta 3（含截图和下载）【1】

（题外话：这两天系统崩溃，重装系统居然失败n次，极度郁闷。终于在昨天重装成功，并将不计其数的常用软件装上。这才感觉生活恢复了正常... 电脑简直成了肢体的一部分..）

Adobe这次跳过Beta 2,直接推出Flash Player Update 3 Beta 3无疑于吹出一记胜利的号角，连续布下多手好棋:

**MovieStar核心**

**Flash Player 缓存**

下面来仔细分解：

1.首先重拳推出的MovieStar核心，率先支持了H.264和HEAAC，使得Flash Player在后面的3G和HDTV时代占领了制高点。
先看一下MovieStar带给我们的全新视觉冲击：高清晰的Flash视频播放。下面是两张Adobe演示网站中播放的视频截图。注意，截图分辨率是850×480，超过一般DVD的清晰度。


演示观看地址:http://labs.adobe.com/technologies/flashplayer9/fullscreendemo/
注意啦，要先卸载机子上原有的Flash Player，再安装Flash Player Update 3 Beta3才行。
1 卸 载 方 法 ： 下 载 uninstall_flash_player.exe （ 官 方 地 址 ）， 在 cmd 模 式 下 ， 运 行 "uninstall_flash_player.exe /clean"。
2.安装方法：下载和安装ActiveX控件，以及Plugin插件

那么其MovieStar实际的意义是什么？

有三点：
1.从技术上说：之前的Flash Player只能播放flv格式的影片，其编码器是 Sorenson Spark video codec和 On2 VP6。其中Sorenson Spark video codec就是基于H.263开发的，H263是H264的前身，主要为了远程会议开发的，以64k速率传输。H.264则能以更低的速率传输更高画质的图像，比On2 VP6也更好一些。H264是被广泛支持的
2.从硬件支持上说：
同时，Flash Player支持全屏硬件加速播放H.264视频了。黑羽在看Demo网站视频时，真是有点激动的说，Flash居然可以如此流畅的播放高清晰视频。在全屏模式下，也很流畅。不过仔细注意，像素缩放还是有些不尽如人意，不够平滑。不过这毕竟是网页插件播放啊，比微软的mediaplayer毫不逊色。
3.从License和市场上说：
毕竟Sorenson Spark video codec技术是别的公司版权所有的，且不说运用范围不太广，别的东西用在自家核心产品上总是有些束手束脚。采用了H.264后，一下子跳到了一个广阔的空间。Flash

Player一下子支持了MP4, M4A, MOV, MP4V, 3GP, 3G2 这么多的格式。使用这些视频的主流公司从Apple到主要的手机厂商，无所不包，广大的现成的媒体资源都成了是Flash 视频市场的新鲜血液。

说了以上几点后，还有一个消息就是，Adobe官方肯定了AIR 1.0正式版也将完全支持H.264，这意味着我们可以编写出基于Flash的桌面高清视频播放器，令人开心啊。 AIR 1.0下一个beta版将于今秋晚些时候放出。

下一篇文章来聊一聊我们开发者更关心的Flash Player Cache功能和机制，这是Flex开发者的福音

# 详解Flash Player Cache——仔细说说FP U3B3【2】

接上一篇：仔细说说Flash Player Update 3 Beta 3（含截图和下载）【1】

Flash Player Cache的概念不是flash player 9 update 3 beta 3才提出的，事实上在两三个月前就有过这方面的讨论了。这次Update 3将它和MovieStar一同宣传，可见它的重要性也不会低于MovieStar的冲击力。

如果说MovieStar是FlashPlayer在媒体播放方面的一个重大突破，那么Flash Player Cache可以算是Flex 应用程序部署的重大突破。

有什么好处？最直接的好处是**Flash Player Cache将极大的减小Flex SWF的体积！**

想一想，现在编译一个Flex应用程序，至少也要100多kb。如果在应用程序中使用了大部分或全套Flex组件，SWF文件将超过500k大关。

我们每天可能要浏览数个Flex网站，每看一个Flex 程序，都要隐性的重复下载SWF所包含的Flex框架和组件。不仅浪费我们下载的时间和带宽，一次次重复初始化这些东西也让人有些不爽。

现在好了，Flash Player Cache功能会自动将这些框架和组件缓存起来，一旦客户访问我们的Flex程序（要经过一些设置），Flash Player Cache功能会自动查看本机Flash Player有没有缓存Flex程序中要求的框架和组件，如果缓存了，直接拿来使用。

注意，不仅不需要下载，连初始化的时间也省了！效率极大提高！

一个原本120多Kb的Hello world Flex程序会瘦身到53kb！一个使用全部框架和组件的程序甚至会减少500kb大小！不仅减去多余的体重，也免掉了重复初始化的时间，没有毒副作用，不呕吐，不拉肚。还犹豫什么，赶快拿起xx订购吧！——flash减肥必备！

原理是什么？RSL认证机制和跨域缓存。RSL，就是runtime-shared-libraries，运行时共享库。这个大家不陌生，AS2时也有类似概念。就是把共用的脚本和资源，放在一个swf文件中，供多个SWF程序使用。

其实Flash Player Cache核心思想就是这个。所不同的是，有认证RSL和非认证RSL之分。只有Adobe认可的RSL才会被Flash Player Cache所共享，包括Flex框架，组件以及一些工具类，也不

排除未来包括其他框架。我们自己的RSL是不能被Flash Player Cache所缓存的，只能存在浏览器的缓存中。

如何识别认证RSL和非认证RSL呢？如何识别不同版本的Flex框架呢？这些其实都是靠认证机制完成的。每个认证的RSL都有独一无二的加密哈希表示自己，根据这个就可以判断是否是同一版本Flex框架和组件，或者是否是官方认证过得RSL。这样，也就避免了潜在的篡改代码可能性。

跨域共享是什么？缓存的认证RSL是共享的，是跨域的（cross-domain的），任何网站的认证RSL都可以共用。

如果本机没有缓存Flex组件，Flash Player Cache不仅仅是访问目标SWF所在的网站来下载Flex框架和组件，而是根据目标SWF提供的加密哈希（SHA256 hash）到所有已下载的网站中去查找。意思是，如果用户第一次访问kingda.org的一个flex swf，发现需要flex 3 moxie的框架和组件；那么它会根据加密哈希查看本机，发现访问www.adobe.com时已经缓存了一个同样框架和组件，那么就会直接使用这个。

最后一个问题，**如何使用Flash Player Cache功能呢？**直接编译mxml当然是不行的，要通过一些设置，告诉Flex，这是一个将认证RSL分离出来的swf文件才行。

以Flex 3 SDK编译器编译example.mxml为例，很简单，如下:

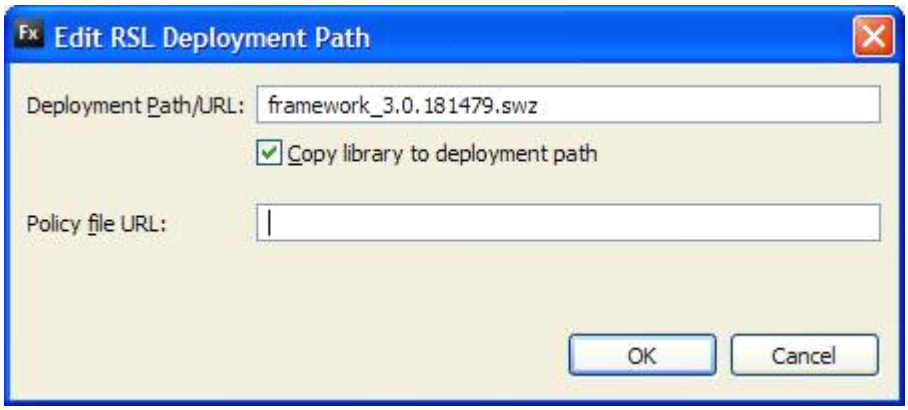
class=code

mxmlc example.xmml

-runtime-shared-library-path=(Flex SDK 路径)\frameworks\libs\framework.swc,framework_3.0.165352.swf

其中3.0.165352是Flex 3 beta1 moxie组件框架版本号。注意，这功能最低要Flash Player 9.0.60以上才支持。

如果是Flex 3用户，做法如下，在Build path中，选中library path,将下拉框中Default link type改成RSL。选中Flex moxie M2（往往是第一个），点击左边三角，展开。选中framework.swf，展开。选中RSL URL(n/a),点击右边的Edit,勾选Use default link type, 点击Digest，在选择Add..，然后会弹出下面的对话框。




选择确定即可。然后编译，就会发现你的swf立刻瘦身了。

要想精益求精，还可以对分离出的RSL再进行瘦身，将其中的调试信息和元数据去掉后，RSL本身还会减少50%左右。优化方法下次再说，打字多了，累了，睡觉了。