中国第一家专业FLASHRIA开源网站【我爱RIA网】收集整理，更多电子书：www.52ria.com

Adobe Flex编码指南v1.2

MXML 和 ActionScript 3

v1.2 – February/2007
Fabio Terracini
fabio.terracini@dclick.com.br

01/26/2008
Dofy尝试翻译
dofyyu@gmail.com

目录

一、介绍：

该文档旨在为使用 Adobe Flex 2 和 ActionScript 3 编写应用程序建立编码指南。

要创建通俗易懂的编码规范，因为在软件开发的生命周期中，大部分时间都是在维护。这样，易于理解的代码片段变得很重要，因为不总是最初的开发者去维护代码。通俗地说就是，让开发者能快速理解别人的代码。除此之外，程序或组件也可以方便地部署或销售给第三方。

制订编码规范的前提是：

- 通用性
- 易理解性

该文档中创建的实例是基于 DClick 的工作方法，Java 编码约定和 Adobe Flex 2 SDK 中的约定。

二、文件

2.1 文件扩展名

- MXML 代码：.mxml
- ActionScript 代码：.as
- CSS 代码：.css

2.2 文件名

- 不能包含空格、标点和特殊符号
- ActionScript
  - 类和接口使用开头字母大写的驼峰式命名例：ExampleClassName
  - 接口总是以一个大写字母 I 开头例：IExampleInterface
  - 包含(includes)使用开头字母小写的驼峰式命名；
  - 命名空间使用开头字母小写的驼峰式命名例：myNamespace
- MXML
  - 总是使用开头字母大写的驼峰式命名
- CSS
  - 总是使用开头字母小写的驼峰式命名

2.3 编码

所有文件都必须使用 UTF-8 编码

三、ActionScript 3.0

3.1 文件结构

ActionScript 文件必须包含以下组成部分：

#	元素	注释
1	开篇注释	
2	定义包	
3	声明命名空间 如果存在命名空间，这是最后部分	一个文件要定义命名空间,这么做就行了
4	Import 描述 1. flash 包 2. mx 包 3. com.adobe 包 4. 公司组件 5. 第三方包，按字母顺序排序 6. 属于该文件的工程包 使用完整的引用名，不要使用星号（*），除非使用了包的绝大部分。 推荐使用：importmx.core.Application 避免使用：importmx.core.*	按字母顺序排序； 如果导入命名空间，同名的包要先于类
5	use 声明（命名空间）	按字母顺序排序
6	元数据 1. Event 2. Style 3. Effect 4. 其他元数据，按字母顺序排序	
7	定义包和接口	
8	静态变量（static） 1. public a) const b) 其他 publicstatic 2. internal 3. protected 4. private 5. 自定义命名空间（按字母顺序排序）	
9	没有使用 getter 和 setter 的变量 1. public 2. internal 3. protected 4. private	

	5. 自定义命名空间（按字母顺序排序）	
10	构造函数	

11	用 getter 和 setter 处理的变量和方法本身,还有相关的变量，例：
	`private var_enabled:Boolean=true;`
	`private var enabledChanged:Boolean = false;`
	`public function getenabled():Boolean{`
	`  return_enabled;`
	`}`
	`public function setenabled(value:Boolean):void{`
	`  _enabled = value;`
	`  enabledChanged = true;`
	`}`
		相关规则参考文档中关于变量的部分

12	方法	将实现相关功能方法组织在一起，而不是按作用域

3.2 样式

3.2.1 行与换行

如果一段描述不能放在一行中，依据下列规则将其拆分成多行：

- 从逗号后换行；
- 在运算符之前换行；
- 最好在较高级别代码处换行；
- 换行后与上一行对齐；
- 如果上一条规则不适用，加入两个缩进

推荐使用：

```
//第一行：在 implements 操作符之前换行
//第二行：在逗号后面换行
//第二、三行：两个缩进
public classButton extends UIComponent
        implements IDataRenderer, IDropInListItemRenderer,
        IFocusManagerComponent
```

避免使用：

```
public classButton extends UIComponent
        implements IDataRenderer, IDropInListItemRenderer,
        IFocusManagerComponent
```

推荐使用：

```
//在高级别代码处换行，例如右小括号
//不要在括号内换行
variable1 = variable2 + (variable3 * variable4 - variable5)
          - variable6 / variable7;
```

避免使用：

```
variable1 = variable2 + (variable3 * variable4
                - variable5) - variable6 / variable7;
```

三元操作符换行示例：

```
b = (expression) ? expression
                 : gamma;//对齐!
c = (expression)
       ? beta
       : gamma;
```

3.2.2 声明

每行只有一个声明。

正确的：

```
var a:int = 10;
var b:int = 20;
```

错误的：

```
var a:int = 10, b:int = 20;
```

尽量对变量进行初始化，如果一些变量的初始值在方法调用中付给则不需要初始化。即使是默认值也要初始化。

正确的：

```
public var isAdmin:Boolean = false;
```

错误的：

```
public var isAdmin:Boolean; //Boolean 型变量的默认值是 false
```

变量的声明放在代码块开始的位置，除非是在循环中。

```
public function getMetadata():void{
   varvalue:int = 123;//方法代码块开始
   ...
   if (condition){
      var value:int = 456;//if 开始
      ...
   }
   for (var i:int = 0; I < valor; i++){//在 for 循环中
      ...
   }
}
```

不要使用之前代码块中用过的变量名，即使是不同的作用域。

3.2.3 括号

样式规则：

- 不要在方法名和括号之间插入空格，也不要在括号和参数间插空格；
- 不要在对象名和类型间插空格；
- 左大括号放在新的一行并与方法定义行对齐；
- 右大括号独占一行并与跟成对的左大括号对齐；
- 方法间用一空行隔开；

3.2.4 声明

简单的

简单声明每行只能有一个，并以分号作为结束。

正确的：

```
i++;
setModel();
```

错误的：

```
i++; resetModel();
```

复合的

复合声明（其中要使用大括号的，例如 `if, while, switch`）必须遵守以下原则：

- 声明内的代码要缩进一级；
- 左大括号在声明下方独占一行，与声明对齐，右大括号同样占一行，与左大括号对齐；
- 在所有声明中都使用大括号，即使声明只有一行；

返回（`return`）

返回无需使用小括号，除非你想使代码更容易理解：

```
return;
return getFinalImage();
return (phase ? phase : initPhase);
```

条件语句 `if, elseif, else`

```
if (condition)
{
   simpleStatement;
}

if (condition)
{
   statements;
}
else
{
   statements;
}

if (condition)
```

```
{
   statements;
}
elseif (condition)
{
   statements;
}
else
{
   statements;
}
```

条件语句 switch, case

Switch 声明适用下面的样式：

```
switch (condition)
{
   case ABC:
   {
      statements;
      //continue,withoutbreak
   }
   case DEF:
   {
      statements;
      break;
   }
   case JKL:
   case XYZ:
   {
      statements;
      break;
   }
   default:
   {
      statements;
      break;
   }
}
```

Break 规则：

- 在 default 代码块中也要使用 break。通常这是多余的，但他能增强代码的可读性；
- 如果一段代码块中不需要 break，在 break 的位置上写上注释；
- 如果声明中包含 return 则不需要使用 break。

循环 **for**

```
for (initialization;  condition;  update)
{
      statements;
}
for (initialization;  condition;  update);
```

循环 **for..in**

```
for (var iterator:type in someObject)
{
      statements;
}
```

循环 **foreach..in**

```
foreach (var iterator:Type in someObject)
{
      statements;
}
```

循环 **while**

```
while (condition)
{
      statements;
}
```

循环 **do..while**

```
do
{
      statements;
}
while (condition);
```

错误处理 **try..catch..finally**

```
try
{
      statements;
}
catch (e:Type)
{
      statements;
}
```

也可以有 finally 声明：

```
try
```

```
{
      statements;
}
catch (e:Type)
{
      statements;
}
finally
{
      statements;
}
```

With

```
with (this)
{
      alpha = 0.5;
}
```

3.2.5 换行与空格

换行（这里指一空行）

换行能使代码看上去更清晰，更有逻辑。

下列情况需要换行：

- 函数之间；
- 方法的局部变量和声明之间；
- 代码块前；
- 单行注释前面或一段特殊功能代码的多行注释前面；
- 将一段代码的逻辑之间分开，使代码更清晰。

空格

在关键字和小括号之间插入空格，但不要在方法和他的小括号中插入空格。

```
while (true)
{
      getSomething();
}
```

在方法的形参列表中，逗号后面要有空格：

```
addSomthing(data1, data2, data3)
```

所有操作符（两个操作数之间的，例如+，-，=，==）与其操作数之间要用空格分隔，但不要分割一元操作符（例如++，--）。

```
a += (5 + b) / c;
while (basint < f)
{
```

```
            i++;
        }
```

三元操作符要用空格分开，有必要的话可以拆分成多行：

```
a = (expression) ? expression : expression;
```

for 表达式中要用空格分开：

```
for (expr1; expr2; expr3)
```

3.3 注释

3.3.1 文档注释

文档注释是指在每个类、接口、变量、方法以及元标签等定义前面放置的一段注释，文档注释的作用是，让那些将要使用这些定义的人能更好的了解定义的作用，而无需去研究代码。

文档注释的语法和格式参考 ASDoc（在 Flex SDK 中也可以找到），地址是：http://labs.adobe.com/wiki/index.php/ASDoc:Creating_ASDoc_Comments

例如:

```
/**
 *The Button control is a commonly used rectangular button.
 *Button controls look like they can be pressed.
 *
 *@mxml
 *
 *...
 *
 *@includeExample examples/ButtonExample.mxml
 */
public class Button extends UIComponent
```

3.3.2 执行注释

执行注释用来对一些不易理解的特殊代码进行说明，使用//进行注释，不管是多行还是单行。

如果注释独占一行，要将注释放在相关代码之前：

```
// 确定没有可见列
if (!visibleColumns || visibleColumns.length == 0)
```

如果不是太长，注释可以和代码放在同一行：

```
colNum = 0; // 补偿可见初始列的偏移量
```

不要翻译代码：

```
colNum = 0; // 把列数设为 0
```

四、MXML

4.1 文件结构

MXML 文件必须包括以下组成部分：

#	元素	注释
1	XML 文件头 <?xml version="1.0" encoding="UTF-8" ?>	总是在文件头中定义编码，并且总是使用 UTF-8 编码
2	根标签	必须包含文件中使用到的所有命名空间
3	元标签 1. Event 2. Style 3. Effect 4. 其他元标签，按字母顺序排列	
4	定义样式	尽量使用外部样式文件
5	定义脚本	只能存在一个脚本块
6	非可视组件	
7	可视组件	

4.2 样式

4.2.1 行与换行

在一组可视组件之间插入空白行可使代码更清晰

在同一个父组件的子组件之间（也包括他们的子组件）插入空白行，如果这个子组件拥有至少一个子组件：

```
<mx:series>
   <mx:ColumnSeries yField="prev" displayName="Forecast">
      <mx:stroke>
         <mx:Stroke color="0xB35A00" />
      </mx:stroke>
      <mx:fill>
         <mx:LinearGradient angle="0">
            <mx:entries>
               <mx:GradientEntry... />
               <mx:GradientEntry... />
            </mx:entries>
         </mx:LinearGradient>
      </mx:fill>
   </mx:ColumnSeries>
```

```
      <comp:ColumnSeriesComponent />
   </mx:series>
```

也就是说，如果一个组件之有一个子组件则不需要插入空白行。下面的 LinearGradient 就只有一个子组件 entries.

```
   <mx:LinearGradient angle="0">
      <mx:entries>
         <mx:GradientEntry... />
         <mx:GradientEntry... />
      </mx:entries>
   </mx:LinearGradient>
```

同样，像 entries 这样子组件都在一行的也不用插入空白行

```
   <mx:entries>
      <mx:GradientEntry... />
      <mx:GradientEntry... />
   </mx:entries>
```

4.2.2 Nstling components（雏鸟组件？）

子组件必须依据其父组件进行缩进排版。

```
   <mx:TabNavigator>
      <mx:Container>
         <mx:Button />
      </mx:Container>
   </mx:TabNavigator>
```

4.2.3 属性

按下列顺序排序：

- 属性
  - 如果有 id 的话，永远放在第一位；
  - 记住，width，height 和 styleName 是属性而不是样式；
- 事件（Events）
- 效果（Effects）
- 样式（Style）

如果有 id，永远放在第一位。

```
   <mx:ViewTack id=”mainModules” width=”75%” height=”75%” />
```

标签属性如果放在多行要进行缩进。

```
<mx:Label
   width="100%" height="100%" truncateToFit="true"
   text="Herecomesalongenoughextthat..." />
```

如果标签的声明有多行，处在第一行的永远只有 id 一个属性，其他属性按上面的顺序放在其他行中。

```
   <mx:ViewStack id="mainModules"
      height="75%" width="75%"
      paddingTop="10" paddingLeft="10" paddingRight="10">

   <mx:ViewStack
      height="75%" width="75%"
      paddingTop="10" paddingLeft="10" paddingRight="10">
```

同一类的属性放在一行，下面的例子中第二行定义了属性，第三行定义事件，第四行定义样式，最后一行是效果。

```
<mx:Panel
   title="VBoxContainerExample" status="Somestatus"
   hide="doSomething()" creationComplete="doSomething()"
   paddingTop="10" paddingLeft="10" paddingRight="10"
   resizeEffect="Resize"/>
```

在同类属性需要放入多行的情况下，视乎情况将更为相近的属性放入同一行，下例中第四、五行均为样式，第五行定义了一些 padding：

```
<mx:Panel id="pnLoginInfo"
   title="VBoxContainerExample" height="75%" width="75%"
   resize="resizeHandler(event)"
   titleStyleName="titleLogin" headerHeight="25"
   paddingTop="10" paddingLeft="10" paddingRight="10" />
```

4.2.4 脚本

脚本代码块的样式：

```
<mx:Script>
   <![CDATA[
      code;
   ]]>
</mx:Script>
```

4.3 注释

4.3.1 文档注释

ASDoc 工具不支持 MXML 文件中的文档注释，但还是鼓励这样去做，如果 MXML 文件是一个可

重用的自定义组件（不只是简单的视图）。这样，文件中就包含了一个与 ActionScript 注释方法相同的脚本代码块。

```
<mx:Script>
   <![CDATA[
      /**
       *Document ation comment inside a MXML component
       *Uses the same format as the AS comment
       */
   ]]>
</mx:Script>
```

4.3.2 执行注释

使用执行注释描述那些目的不太清晰的界面元素

```
<!—这里是注释-->
```

或使用多行

```
<!--
多行注释
...
-->
```

五、样式

5.1 一般规则

- 缩进使用制表符，占 4 空格位，需要在 IDE 中进行设置；
- 单行代码不超过 100 个字符；

注：使用 Eclipse 配以 1280 分辨率，如果编码窗口占 70%（其他 30%用于导航），每行大约可以显示 103 个字符，A4 大小纸张一行大约 80 个字符。

六、命名

6.1 一般规则

- 缩写:应尽量避免使用缩写,除非是那些缩写形式更常用于它的全称的缩写(例如 URL, HTML 等)。工程名可以是用缩写，如果他就是叫那个名字。
- 只使用 AscII 字符，不包括重音符号（`)、空格、标点和特殊字符;
- 不要使用 FlexSDK 中预留的关键字命名（像 mx 包中的 Application、DataGrid 等)，也不要使用 FlashPlayer 中预留的关键字（像 Flash 包中的 IOError、Bitmap 等）
- 既然在 MXML 中编写代码与在 ActionScript 中一样容易,在 MXML 中的命名规则也和 ActionScript 中是一样的(举例来说,一个 MXML 就像一个 ActionScript 类，其内部的组件和变量等同于属性）
- 主程序文件命名为 Main.mxml
  - 不要在组件名称中使用索引，以免在用 ASDoc 工具生成文档时产生冲突。

6.2 语言

编码本身必须使用英语，除了涉及到商业领域的动词和名词（特殊专业领域的软件是为了解决问题，也就是说，现实世界和系统是相关联的）

因此，推荐使用下面的命名：

```
DEFAULT_CATEGORIA
addNotaFiscal()
getProdutosByCategoria()
changeState()
UsuarioVO
screens/Reports.mxml
```

不推荐下列命名：

```
popularCombo()
mudarEstado()
UsuarioObjetoDeTransferencia
```

6.3 包

包的命名必须使用 lowerCamelCase（驼峰式命名)，第一个单词的开头字母小写，其他单词的开头字母大写

包名的第一部分使用顶级域（com, org, mil, edu, net 或 gov）或者使用两个单词，国家标识加顶级域，参考文档 ISO3166（br.com, ar.edu, cn.org 等）

下一部分是所有者（公司或客户）的名称，接下来是工程名和模块：

```
cn.com.company.project.module
```

6.4 类

类名推荐使用名词，也可以使用形容词。总是使用 UpperCamelCase（开头字母大写的驼峰式命名）形式

6.5 接口

接口的命名规则和类的是一样的，只是开头加上一个大写字母 I

6.6 方法

方法必须使用动词开头，并使用 lowerCamelCase（开头字母小写的驼峰式命名）形式，如果方法被一个事件调用，结尾加上 Handler

6.7 变量

变量使用 lowerCamelCase 形式命名，并且使用有意义的名称，如果变量使用 getter 和 setter 方法读/写值，开头应加上下划线（_）

变量不必使用前缀，在 ActionScript 中给对象一个通俗易懂的名字比对象类型更重要。尽管如此，Boolean 型变量应以 can、is 和 has 开头。

临时变量（例如在循环中）通常使用一个字母，最常用的字母是 i, j, k, m, n, c, d。不要使用 l。

catch 中的变量必须使用 e，不管错误类型为何（如果他是个类）

6.8 常量

常量全部使用大写字母，单词间用下划线分割（_）

6.9 命名空间

命名空间全部使用小写字母，单词间用下划线分割（_）
文件名必须与命名空间名称一致。

七、一般习惯

- 在注释中使用 FIXME 关键字来标识某些需要修正的代码，使用 TODO 标识某些在重构时需要改良的代码，这可以使用 Flex Builder 2 Task 插件；
- 将需要重复调用的值在使用指派给一个变量，这对提高性能相当有用（在简单的数组中不需要）

  正确的:

```
var maxPhase:int = reallySlowMethod();
for (var i:Number = 0; I < maxPhase; i++)
{
   statements;
}
```

  正确的:

```
var months:Array = ['Jan', 'Fev', 'Mar'];
// 获取数组长度是相当快的
// 这样做同时还增强了代码的可读性
for (var i:Number = 0; I < months.length; i++)
{
   trace (months[i]);
}
```

  错误的:

```
for (var i:Number = 0; I < reallySlowMethod(); i++)
{
   statements;
}
```

- 尽量使用松耦合的组件，组件之间的关连越少，其重复利用的可能越大；
- 在 Boolean 判断中，将速度快的放前面。

  正确的:

```
if (isAdmin && slowMethod(item))
```

  错误的:

```
if (slowMethod(item) && isAdmin)
```

- 如果可用，尽量使用常量
- 如果可用，尽量使用更精确的类型
- 建议在简单的事件处理中使用匿名函数

八、附录：预留关键字

下表中列出了 ActionScript 3 中的预留关键字：

as	break	case	catch
class	const	continue	default
delete	do	else	extends
false	finally	for	function
if	implements	import	in
instanceof	interface	internal	is
native	new	null	package
private	protected	public	return
super	switch	this	throw
to	true	try	typeof
use	var	void	while
with			

还有些语法关键字，在某些语境下有特殊含义，因此也尽量避免使用：

each	get	set	namespace
include	dynamic	final	native
override	static		

还有些为未来版本预留的关键字也应尽量避免使用：

abstract	boolean	byte	cast
char	debugger	double	enum
export	float	goto	intrinsic
long	prototype	short	synchronized
throws	to	transient	type
virtual	volatile		