

# Flex 3 CookBook 简体中文

可公布

Flex 3 CookBook 翻译协作组

申明：本电子书可整份免费自由复制流传，但未经译者同意，不得加以切割、剪辑和修改内容使之用于商业用途。

Flex 3 CookBook 简体中文是众多网友共同努力的成果，没有他们的辛勤劳动也就不会有此书，在此请允许我代表广大的 Flex 爱好者向所有自愿参与翻译的人员表示最衷心的感谢。由于此书采用多人协同翻译，每个人的水平又不尽相同，难免会出现或多或少的问题，在这里也请每位读者能怀着一份感激和宽容的心情阅读此书。如在阅读中发现错误和不妥的地方，请到我们指定的论坛留言，留下您自己的修改建议，我将会收集这些建议，如果足够多的话，不排除出一个修正版，谢谢大家。

-- 常青  
-- 2009.01.15

论坛讨论专区：[http://bbs.airia.cn/FLEX3\\_COOKBOOK/list-1.aspx](http://bbs.airia.cn/FLEX3_COOKBOOK/list-1.aspx)

感谢 [aria.cn](#) 和 [riabook.cn](#) 提供论坛和下载支持！



## 翻译人员大名单

网名:常青  
QQ昵称 (QQ号码) : 常青(83671336)  
电子邮件:xinye0123@gmail.com  
个人blog:  
<http://blog.csdn.net/lixinye0123>

网名: Spark.FandLR  
QQ昵称 (QQ号码) : ASer(837415201)  
电子邮件: spark.fandlr@gmail.com  
个人blog: <http://www.4nothing.net/blog/>

Fx

网名: na  
QQ昵称 (QQ号码) : na(326074772)  
电子邮件: kohoho@126.com  
个人blog:  
<http://blog.sina.com.cn/polythenepam>

Fx

网名: 飞客  
QQ昵称 (QQ号码) : 石头(121015664)  
电子邮件: xu\_nigel@hotmail.com  
个人BLOG:  
<http://xunigel.spaces.live.com>

Fx

网名 : native / eas  
QQ昵称(QQ号码):sai|黑白瞎猫(329618)  
电子邮件 saicn@qq.com  
个人blog: <http://www.eascn.net>

Fx

网名: 草衣薰  
QQ昵称(QQ号码):ω草·衣·薰(68552233)  
电子邮件: machaoii@263.net  
个人blog: <http://newx3d.cn/blog>

Fx

网名: 小河  
QQ昵称 (QQ号码) : 小河 (122783879)  
电子邮件: hewskiq@163.com  
个人blog:  
<http://hi.baidu.com/heguang>

Fx

网名:Roast  
QQ昵称(QQ号码):Roast(349985877)  
电子邮件:zhang.libing@gmail.com  
个人blog:<http://www.libing.name/>

Fx

姓名: 王平  
QQ昵称 (QQ号码) : 王平(121503317)  
电子邮件: msnwangping@hotmail.com  
个人BLOG: 无

Fx

网名:一文  
QQ昵称 (QQ号码) : 桃之夭夭(2815943)  
电子邮件:conoon@163.com  
个人blog:<http://conoon.blogbus.com>

Fx

网名: TONY IAN  
QQ昵称 (QQ号码) :TONY IAN(675559240)  
电子邮件: iankawa@gmail.com  
个人blog: 无

Fx

网名:ken  
QQ昵称 (QQ号码) : ken(839381279)  
电子邮件:839381279@qq.com  
个人blog:无

Fx

网名:屋檐下  
QQ昵称 (QQ号码) : 屋檐下(839381279)  
电子邮件:372735509@qq.com  
个人blog:无

Fx

# 第一章. Flex 与 ActionScript 基础 (常青)

一个Flex应用程序有ActionScript和MXML两种语言代码组成。从3.0开始ActionScript已经从基于原型脚本语言进化到完全面向对象的，强类型的符合ECMAScript标准的脚本语言。MXML则是一种标记语言，非常类似于大家所熟悉的超文本标记语言(HTML)，扩展标记语言(XML)。

如何把MXML和ActionScript相互关联起来呢？对于编译器来说，解析这两种语法后最终被翻译成同一个对象，比如：

```
<mx:Button id="btn" label="My Button" height="100"/>  
和  
var btn:Button = new Button();  
btn.label = "My Button";  
btn.height = 100;
```

产生的是同一个对象，两者的主要不同是，ActionScript创建的对象（上面第二个例子）除了Button就没有别的了，而MXML中创建的对象将Button添加到包含MXML代码的任何组件上。Flex框架根据MXML中的对象描述来调用构造函数，然后将其添加到父对象上或设置其为父对象的某个属性。

MXML文件中可用<mx:Script>标签包含ActionScript，不过ActionScript文件是不能包含在MXML里的。你可以这样理解：MXML是描述应用程序外观及其组件，而ActionScript则描述如何处理应用程序的事件和自定义逻辑，虽然这么说不完全确切，但可以让你明白这两者之间的大致关系。在某些方面，比如循环，函数定义，条件语句等等都必须通过ActionScript实现，除了最简单程序可能不需要ActionScript外，绝大多数都是需要MXML和ActionScript来相互协作。

这一章讨论的内容很多都是关于MXML和ActionScript交互问题：用MXML创建组件，在ActionScript中创建类，添加事件监听器，编写ActionScript和MXML代码，创建函数申明，虽然没有列举所有的内容，但这些都是ActionScript和MXML的基础内容。

## 1.1节. 用Flex Builder创建Flex项目

### 1.1.1. 问题

我想用Flex Builder创建Flex项目。

### 1.1.2. 解决方法

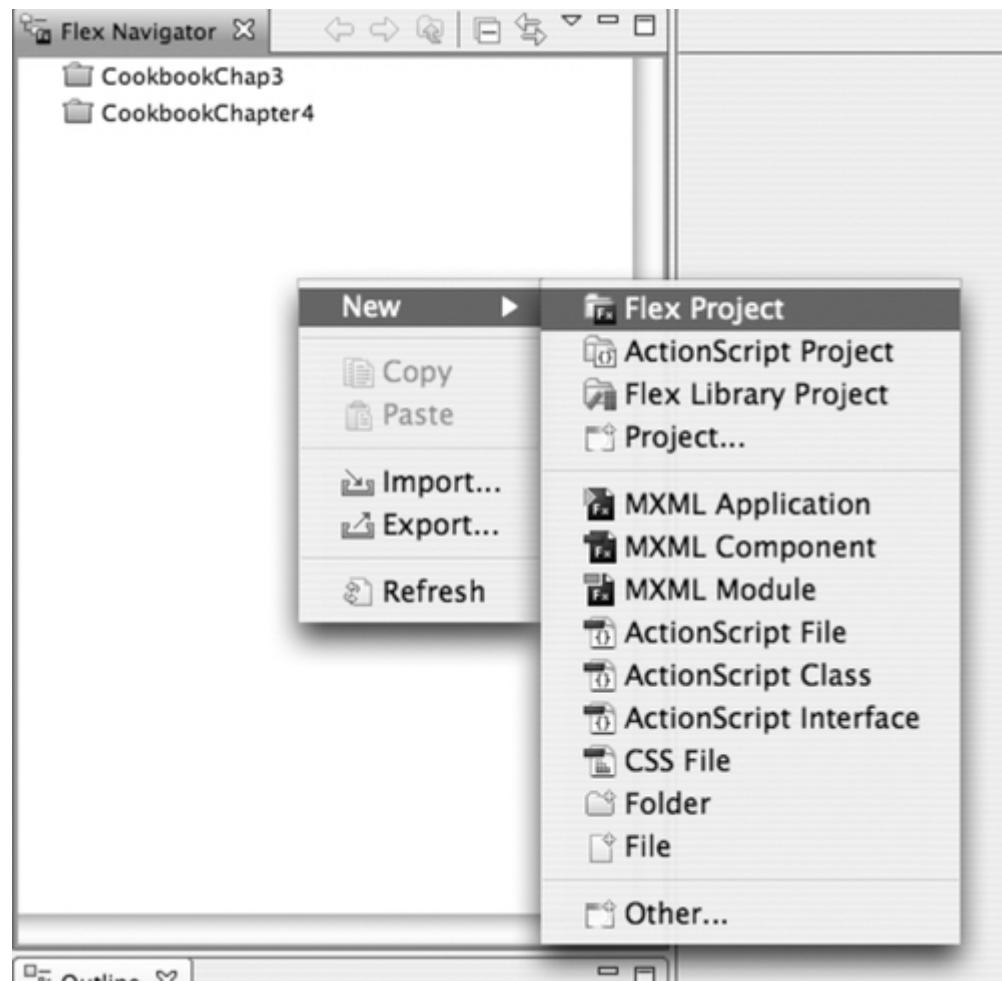
使用Create New Project 向导

### 1.1.3. 讨论

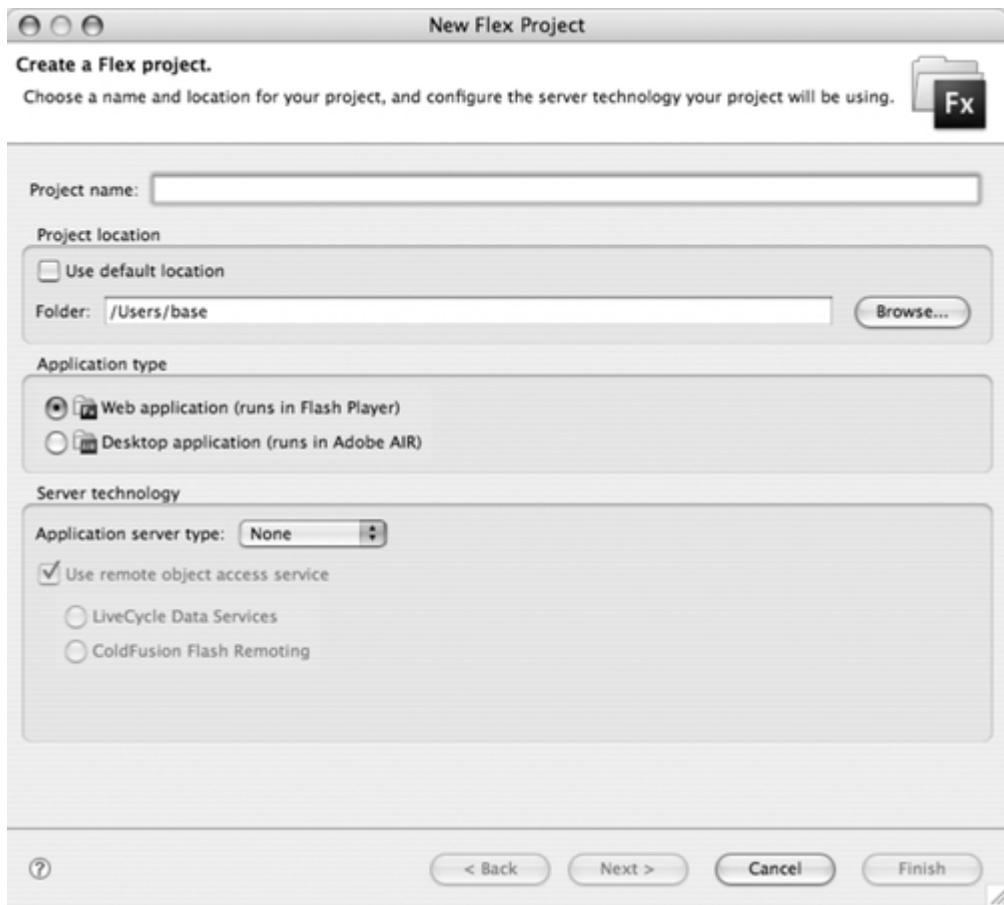
Flex Builder 构建在 Eclipse 之上，eclipse 是一个非常好的交互式开发环境 (IDE)，非常流行的 Java 开发工具。虽然开发 Flex 程序不一定需要 Flex Builder，但是 Flex Builder 提供了很多优秀特性可帮助你有效提高设计和开发效率，Flex Builder 可独立安装也可作为 eclipse 插件形式安装。

对于开发者来说第一件事就是如何创建一个Flex项目。Flex项目和其他项目有点不一样，因为它需要包含SWC (Flex库) Flex library SWC (不像ActionScript项目) 编译生成的可被Flash播放器执行的SWF文件(不像Flex Library项目)。要想创建项目，在Flex Navigator视图中点击鼠标右键，在弹出菜单中选择 New→Flex Project，然后会弹出创建项目对话框。

**Figure 1-1.** 创建一个Flex新项目



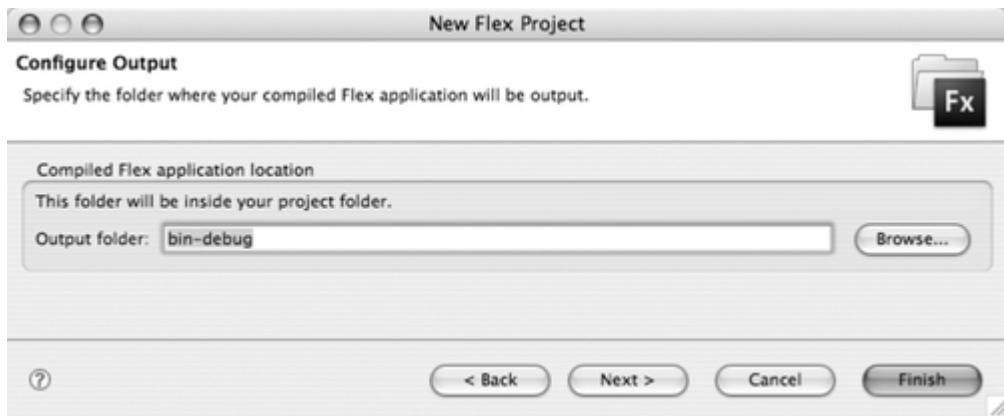
**Figure 1-2.** 用Flex Builder创建新项目



输入项目名称以及项目的存放路径，Windows系统默认情况下存放在C:/Documents and Settings/Username/Documents/workspace/Projectname，而MAC系统默认情况下存放在Users/Username/Documents/workspace/Projectname，当然你可以改变它，存放到自己喜欢的位置。项目名称必须是唯一的，不能重复。项目类型可选择Web程序或桌面程序(AIR程序)，最后选择可能需要的服务器技术用于数据交互。

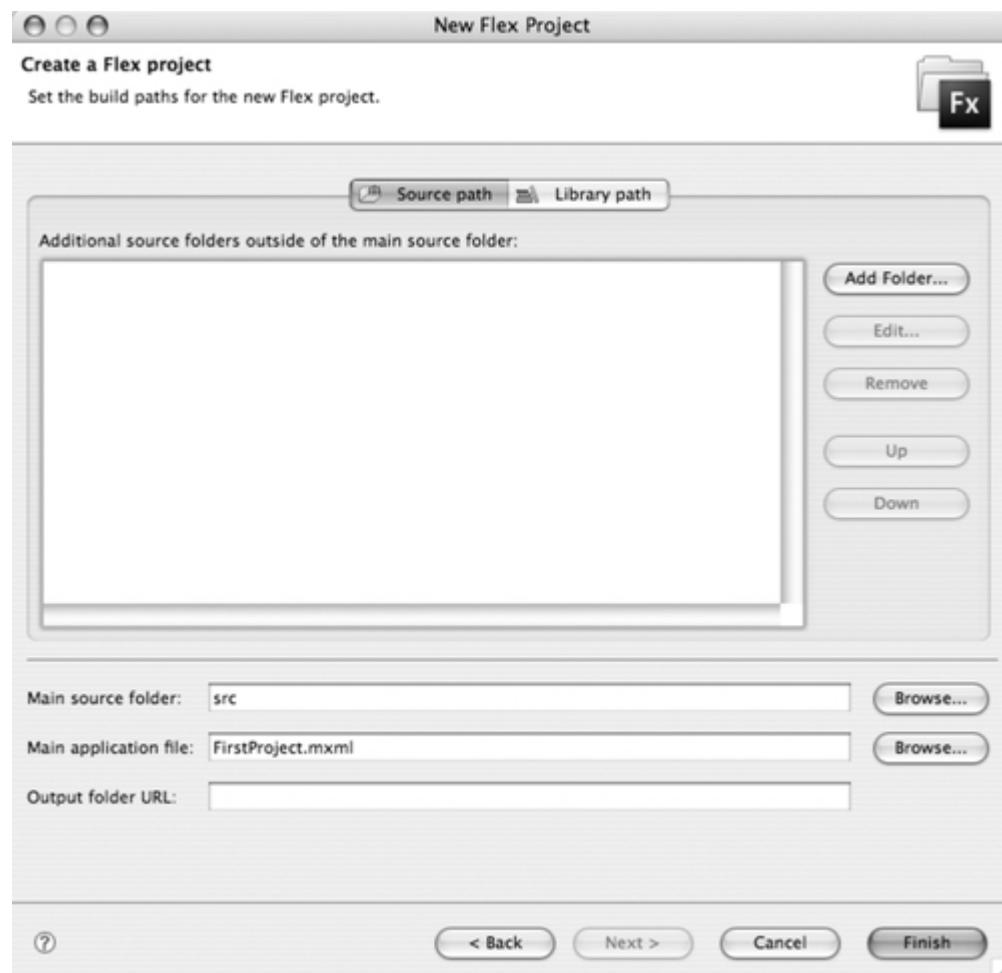
设置好了后，点击Finish，设置SWF输出目录路径，点击下一步

**Figure 1-3.** 设置输出SWF文件的区域属性



设置好SWF输出路径后，还要设置下源文件存放目录或SWC库文件路径。在Source path标签中还可添加其他目录，在Library Path标签中添加SWC文件，在这个对话框中还可更改主MXML程序文件名，默认是和项目名相同。

**Figure 1-4.** 设置源文件目录和主程序文件名



**Figure 1-5.** 设置其他需要的库



所以路径设置好后，点击Finish，项目就算创建好了，现在可以开始开发了。

## 1.2节. 用Flex Builder创建Flex库项目

### 1.2.1. 问题

我想创建Flex库项目

### 1.2.2. 解决办法

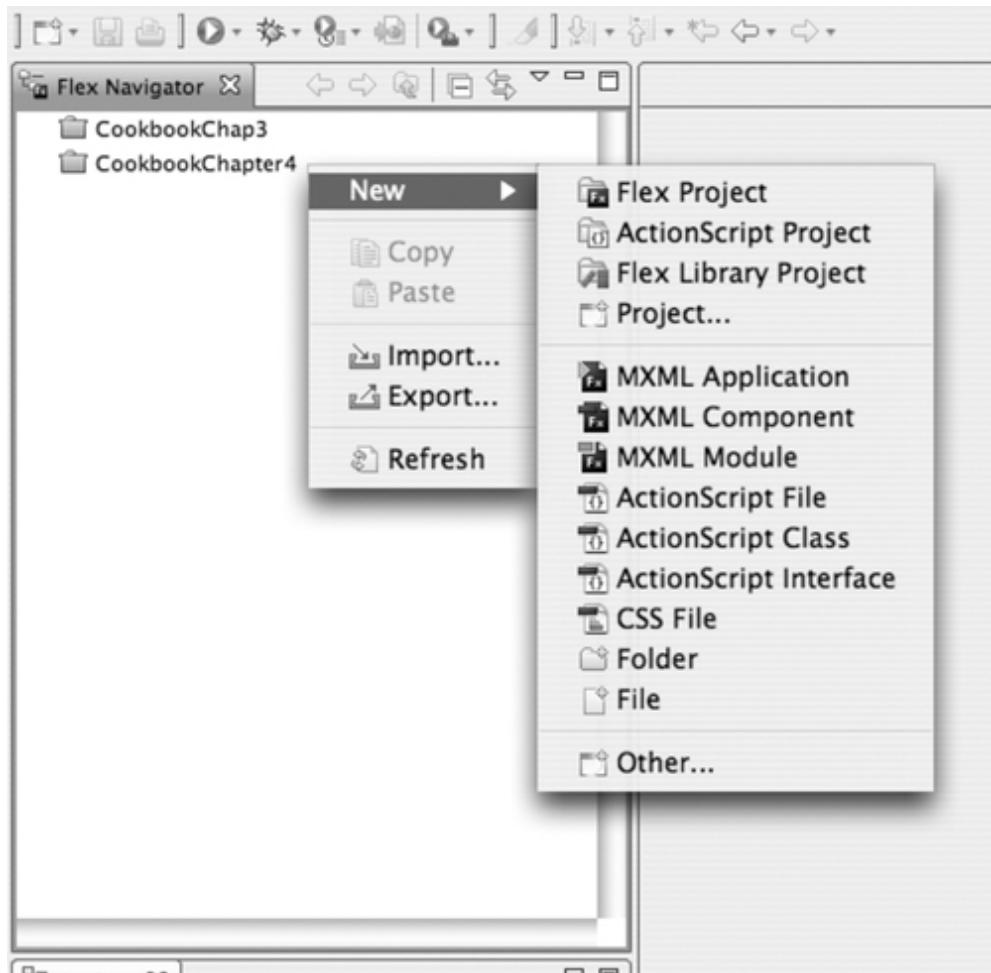
选择New→Flex Library Project 访问创建新项目向导

### 1.2.3. 讨论

Flex Library 项目没有编译成SWF的主MXML文件。相应的是编译成SWC文件供其他应用程

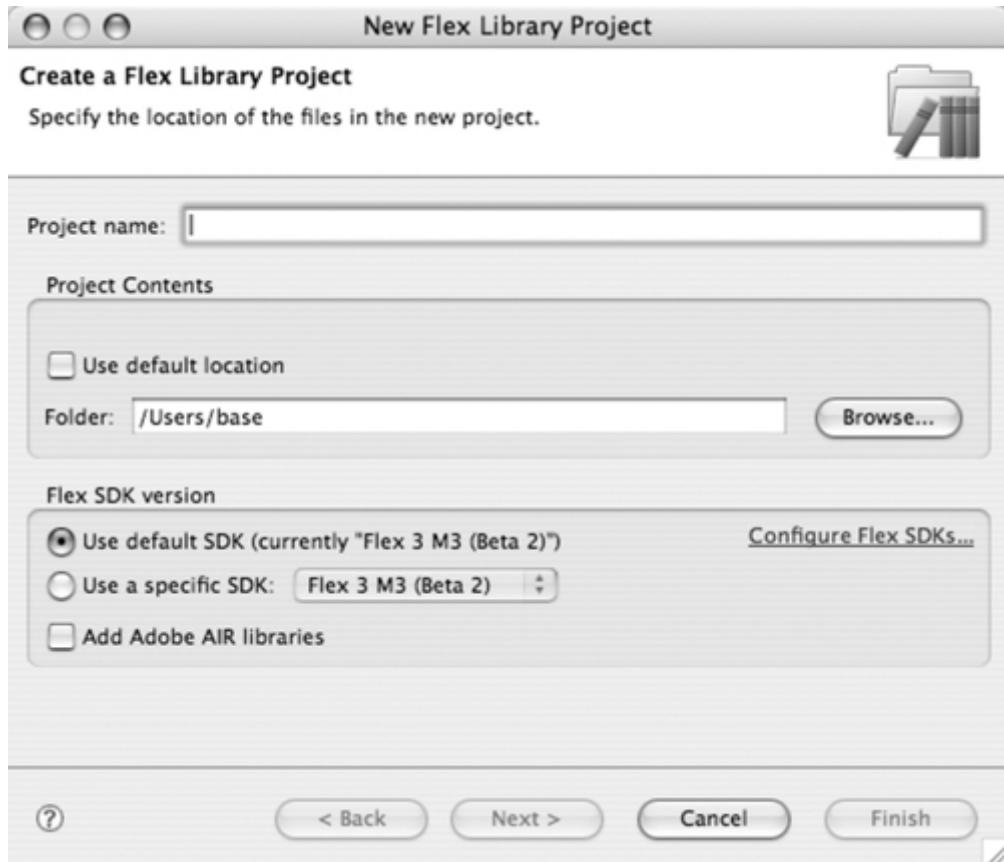
序使用或作为运行时共享库（RSL），要想创建Flex库项目，鼠标右击Flex Builder的项目navigator视图打开关联菜单(Figure 1-6)或通过File菜单，选择New→Flex Library Project.

**Figure 1-6. 创建Flex库项目**



在下面的对话框中(Figure 1-7)，指定项目名称及存放路径。

**Figure 1-7. 设置项目路径和SDK版本**



设置很好后，点Finish，如果你还有加入文件，图片资源或SWC文件，可点击Next。

## 1.3节. 创建ActionScript项目

### 1.3.1. 问题

我想创建ActionScript项目

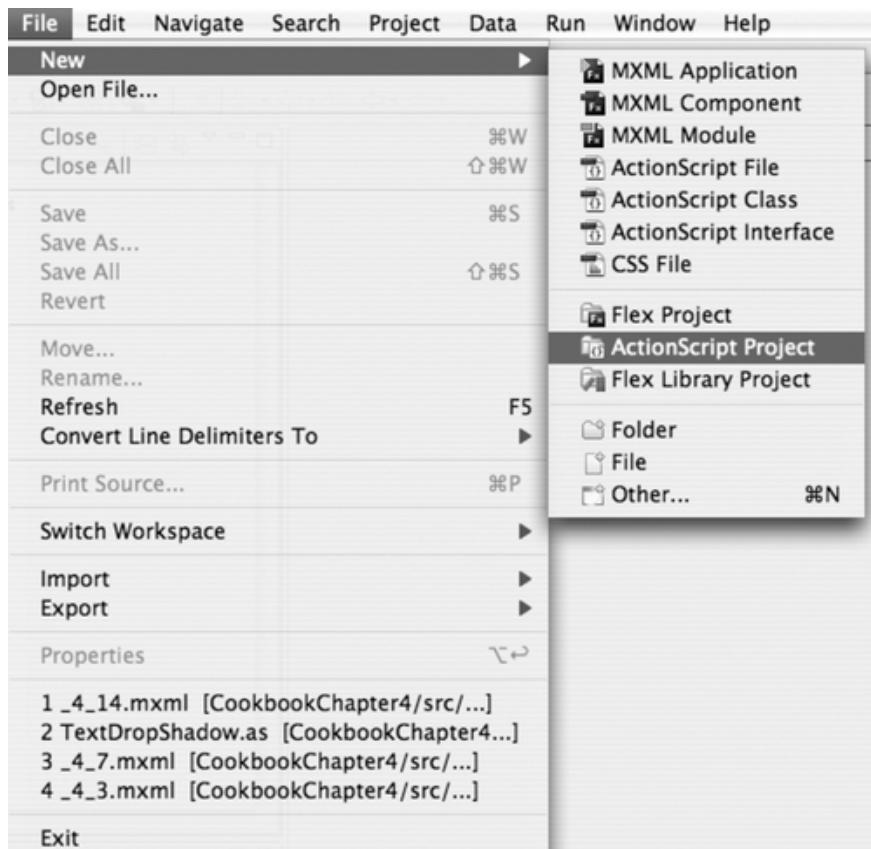
### 1.3.2. 解决办法

通过新建向导并选择ActionScript Project.

### 1.3.3. 讨论

ActionScript项目不同于Flex项目，因为它根本不包含Flex框架，ActionScript项目完全由基于Flash代码的核心ActionScript类所组成，它不需要访问Flex框架中的任何组件。要创建ActionScript项目，选择File→New→ActionScript Project (Figure 1-8).

**Figure 1-8. 创建ActionScript项目**



在打开的对话框中，指定项目名称以及项目存放路径和SWF编译输出路径，点Finish完成设置，点Next可继续添加库或源代码保存目录，或更改主MXML文件名等等。

## 1.4节. 在Flex Builder中设置MXML编译器选项

### 1.4.1. 问题

我想设置MXML编译器选项

### 1.4.2. 解决办法

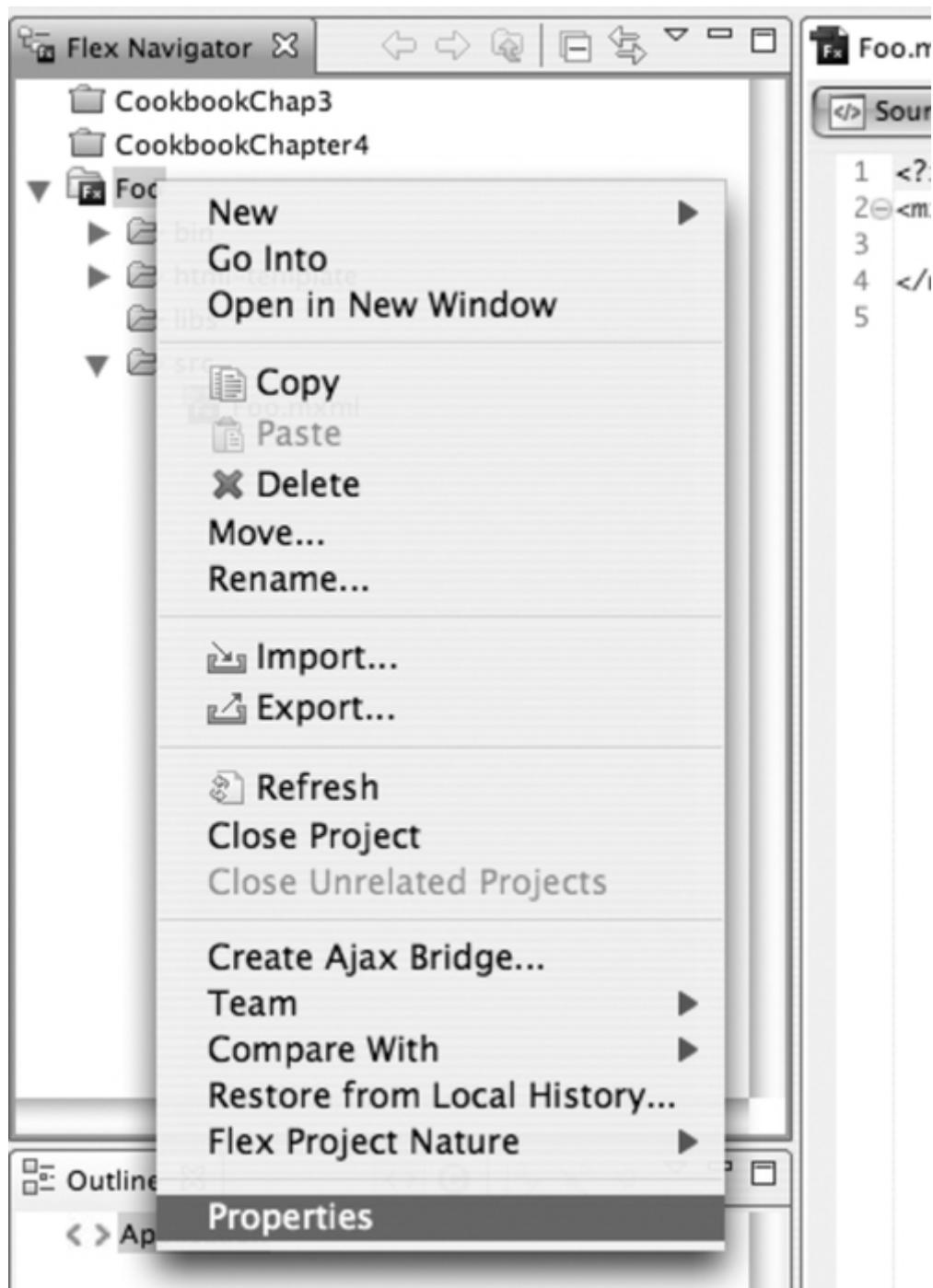
在项目属性对话框中设置编译器参数

### 1.4.3. 讨论

MXML编译器，就是mxmlc，它把ActionScript和MXML文件编译为SWF文件以供Flash Player运行。当你在Flex Builder中运行或调试Flex程序时，MXML编译器会被调用，文件作为编译器参数被传递过去，当你调式时会把调试版的SWF作为参数传递给MXML编译器。例如你可以把外部的库文件路径作为参数，或允许SWF文件访问本地文件或设置背景颜色。

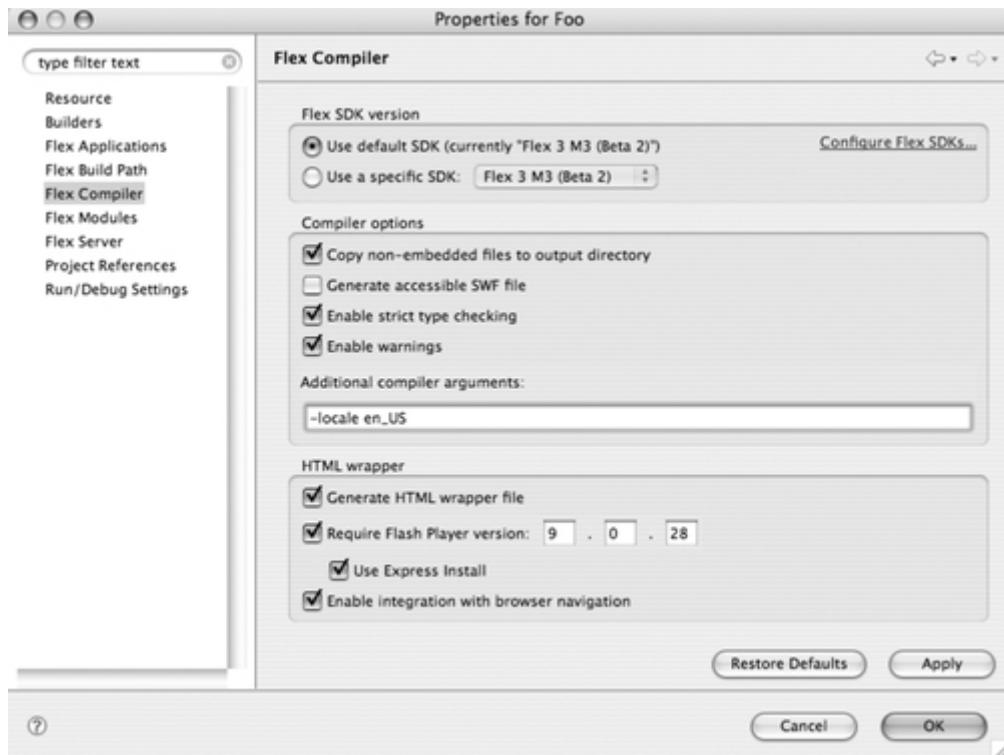
要想改变项目的编译器设置，可右击项目名称并选择Properties(Figure 1-9)，

**Figure 1-9. 更改项目属性**



在下面的对话框中(Figure 1-10), 选择Flex Compiler, 这里有一些选项控制SWF文件如何被编译, 在输入框里可添加额外的编译器参数, 可一次性添加多个参数, 每个参数前加上 (-) 符号. 参数之间用空格分开。

Figure 1-10. 设置编译器选项



下面是一些常见的编译器参数：

#### [verbose-stacktraces](#)

指定SWF在运行时异常信息中包含行号和文件名，这将使产生的SWF文件更大些，带 verbose-stacktraces 的SWF还是和调试版本的SWF有区别的。

#### [source-path path-element](#)

添加其他源代码目录或文件，可以使用通配符来添加目录中所有文件或子目录，也可使用 += 在默认路径上来追加新参数，例如

-source-path+=/Users/base/Project

#### [include-libraries](#)

指定SWF文件被编译到程序中并链接库中所有类和资源到SWF上。如果你的程序需要加载其他模块这个参数就很有用了。

#### [library-path](#)

跟include-libraries选项类似，但是只引用类和资源以供SWF使用，这样可保持SWF文件的可管理性。

#### **locale**

指定SWF文件的区域属性，例如使用-locale=es\_ES 指定SWF区域为西班牙

#### **use-network**

指示SWF是否可以访问网络服务或者应用标准的Flash Player权限策略。例如-use-network=false 指定SWF有本地文件系统访问权但不能访问任何网络服务，默认为true

#### **frames.frame**

启动应用程序资源代理流，然后通过ModuleManager类公布其接口，在特殊情况下，比如在代码中已经引入资源但是并不需要移动资源到外部SWF文件，这时此参数可使应用程序启动时间大大减少，这是一个很复杂但很有用的参数。

#### **keep-all-type-selectors**

保证所有样式信息都被编译进SWF，甚至是程序没有用到的。这点非常重要，因为有可能程序加载的其他组件需要这些样式信息。默认值为false，也就是说没有用到的样式信息不会被编译进SWF。

设置好编译器参数后，点击Apply按钮保存。

## **1.5节. 在Flex Builder外部编译Flex项目**

### **1.5.1. 问题**

我不想在Flex Builder里进行编译项目

### **1.5.2. 解决办法**

使用终端或命令行窗口调用MXML编译器

### **1.5.3. 讨论**

虽然Flex Builder是一个功能强大的Flex开发工具，但是这不是创建Flex程序所必需的，你仍然可以用Flex编译器(mxmlc)来编译Flex代码，Flex编译器是免费的，在Adobe网站上可免费下载。在Windows下的命令行或Mac OS X的终端里调用MXML编译器，以及待编译的文件作为参数，例如：

```
home:base$ ./Users/base/Flex SDK 3/bin/mxmlc ~/Documents/FlexTest/FlexTest.mxml
```

上面通过目录中的编译器编译MXML文件，在终端或命令行窗口会显示可能出现的编译警告和错误。如要添加MXML编译器选项，可在调用编译器命令时追加参数，例如：

```
home:base$ ./mxmlc ~/Documents/FlexTest/FlexTest.mxml -output=/Users/base/test/genera
```

```
ted/Index.swf -library-path+=/Users/lib/MyLib.swc
```

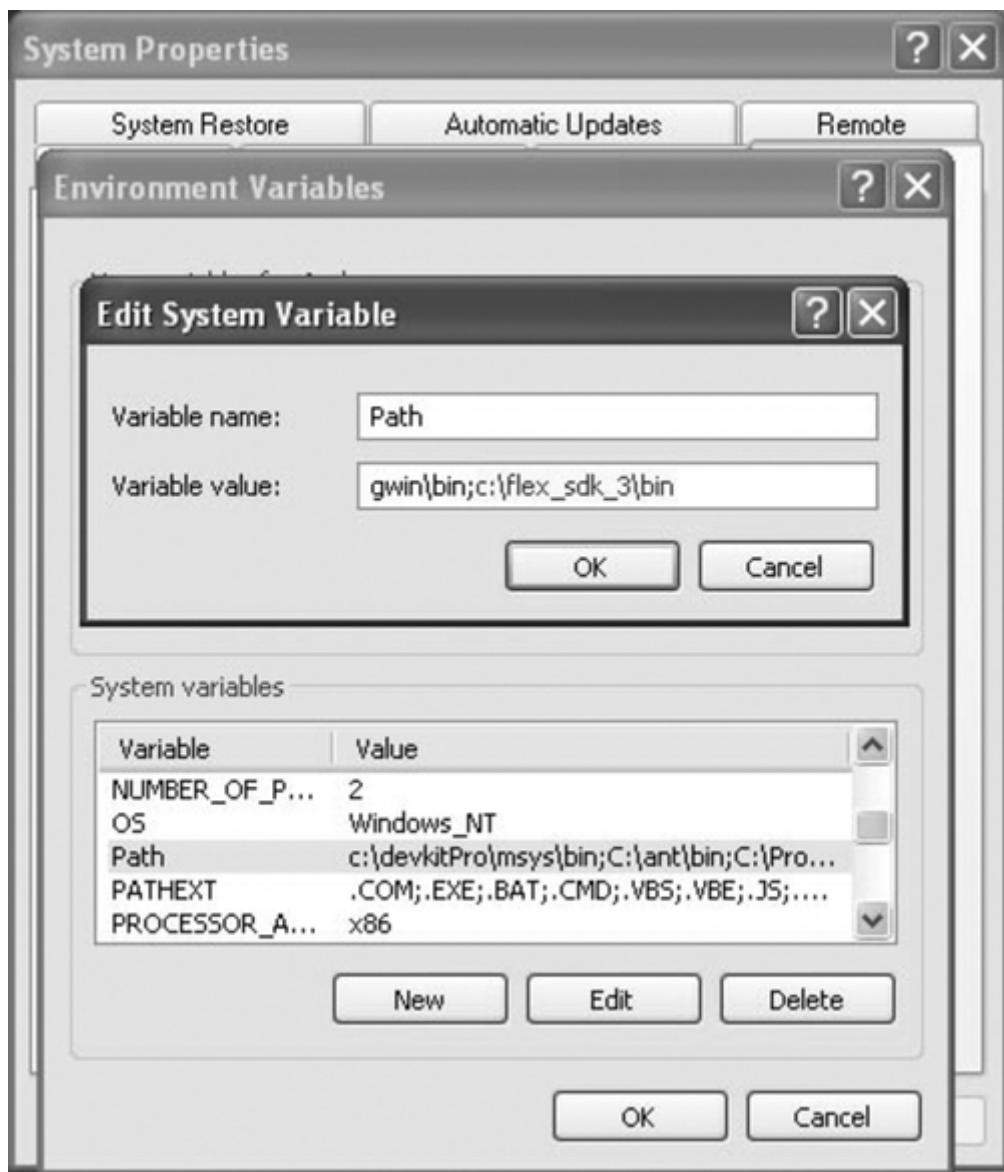
产生的SWF文件被重命名为Index.swf，被保存在/Users/base/test/generated/，编译时引入SWC库/Users/lib/MyLib.swc。

要调用MXML编译器，可直接在命令行下输入命令而不需要指定SDK全路径（例如C:\flex\_sdk\_3），当然在此之前你需要添加bin目录到系统的Path的环境变量。

### 在 Windows 上：

1. 打开控制面板的系统选项
2. 选择高级标签.
3. 点击环境变量
4. 在系统变量中，找到Path，双击它。
5. 加入SDK的bin目录路径 ([Figure 1-11](#)).

**Figure 1-11. 设置Flex SDK 3 Path 变量**



6. 设置好后，打开命令行，定位到项目目录，输入下面的命令：

```
C:\Documents\FlexTest> mxmcl FlexTest.mxml
```

这样会在C:\Documents\FlexTest目录下生成FlexTest.swf文件，因为已经在先前设置好了SDK路径，这里调用编译器时就不用输入全路径了。

7. 如果在第6步产生如下错误信息：

```
Error: could not find JVM
```

这时你需要手动指定Java Runtime Environment (JRE)安装路径，打开Flex 3 SDK的bin目录，用文本编辑器打开jvm.config文件，找到java.home变量（没有则添加之）。设置你的JRE安装路径：

```
java.home=C:/Java/jre
```

## 在 Mac OS X 或 Linux:

打开.bash\_profile文件（如果你是使用Bash）编辑path变量，如下：

```
PATH="$PATH":~/flex3SDK/bin  
export PATH
```

.bash\_profile文件保存在你的home目录下（可通过cd ~目录查看），如果你使用的是tsch，则需要编辑.profile文件：

```
PATH="$PATH":~/flex3SDK/bin  
export PATH
```

## 1.6节. 在MXML中添加事件监听器

### 1.6.1. 问题

我想在MXML中添加事件监听器来监听MXML文件中的子对象所发出的事件。

### 1.6.2. 解决办法

传递一个方法名给组件的event标签并发送一个event对象（可选）。

### 1.6.3. 讨论

当一个行为发生时，Flex组件便会发出相应事件信号，比如用户点击一个按钮，选择列表框的某一项或者数据读取。要监听这些被广播出去的事件，最简单的方法就是添加一个函数引用，该函数将处理这个事件，例如：

Code View:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"  
height="300">  
    <mx:Script>  
        <! [CDATA[  
            private function buttonClick():void  
            {  
                trace(" Button has been clicked ");  
            }  
        ]]>  
    </mx:Script>  
    <mx:Button click="buttonClick()" label="Click Me"/>  
</mx:Canvas>
```

添加click="buttonClick()", 当按钮发出click事件时将调用buttonClick函数。

你还可以传递事件对象本身给这个函数，每次组件发出该事件时，组件也会发送Event类型的对象给监听它的处理函数，例如：

Code View:

```

<mx:HBox      xmlns:mx="http://www.adobe.com/2006/mxml"      width="400"
height="300">
<mx:Script>
<! [CDATA[
    private function buttonClick(event:Event):void
    {
        trace(event.target.id);
        if(event.target.id == "buttonOne")
        {
            trace(" button one was clicked")
        }
        else
        {
            trace(" button two was clicked")
        }
    }
]]>
</mx:Script>
<mx:Button click="buttonClick(event)"
label="Click Me One" id="buttonOne"/>
<mx:Button click="buttonClick(event)" label="Click Me Two"
id="buttonTwo"/>
</mx:HBox>

```

通过事件监听器以侦听的Event类型的对象，给事件监听器发送该事件并以不同的方式反馈，具体取决于指定的条件。在此示例中，响应该事件取决于事件来源。

Flex中的事件对象以及事件发送系统是一个非常重要的内容。所有事件都包含一个正在侦听该事件时所使用的类型，如果该事件是click事件，那么子对象的click事件就会加入事件监听方法：

```
<mx:Button click="trace('I was clicked')"/>
```

用户交互的通知，应用程序发送消息或定时发送给服务器，事件对象定义了一些任何监听函数都可以访问的属性，如下所示：

### **bubbles**

指示事件是否是冒泡事件，即是否从已接收任何监听器进一步沿事件链向上重新发送该事件对象。

### **cancelable**

指示该事件是否是可取消的。

### **currentTarget**

处于活动进程的事件对象。

#### **eventPhase**

事件流的当前阶段

#### **Target**

事件目标，即发出该事件的对象

#### **Type**

事件类型

你也可以在MXML中通过绑定标签{}直接写入事件处理语句。

Code View:

```
<mx:Button click="{textComponent.text = 'You clicked the  
button!'" label="Click Me"/>  
  
<mx:Text id="textComponent"/>
```

当编译这段代码时，编译器会创建一个函数，设置textComponent.text='You clicked the button'作为函数体的内容。这种写法看起来不一样，但是其结果都是一样的：监听事件并执行代码。这种方法本身并没有错误，但是如果处理的是复杂的事情而非简单的设置一个属性，定义一个函数会使你的代码看起来清晰的多，且更易于阅读和理解。

## 1.7节. 设置子节点属性

### 1.7.1. 问题

我想通过MXML中的script标签内容的某个方法来设置子节点属性。

### 1.7.2. 解决办法

通过id属性查找子节点组件，并使用id属性调用方法。

### 1.7.3. 讨论

人们很容易把组件的脚本代码部分与mxml部分分割开来看，但实际上它们是一体的，例如下面的例子：

Code View:

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"  
height="300">  
    <mx:Script>  
        <! [CDATA[  
            private function changeAppearance():void
```

```

    {
        this.width = Number(widthInputField.text);
        this.height = Number(heightInputField.text);
    }
] ]>
</mx:Script>
<mx:Image id="imageDisplay"/>
<mx:Text text="Enter a width"/>
<mx:TextInput id="widthInputField"/>
<mx:Text text="Enter an height"/>
<mx:TextInput id="heightInputField"/>
<mx:Button click="changeAppearance()" label="Change Size"/>
</mx:HBox>

```

正如你所看到的，在changeAppearance方法中，this变量指向组件本身，即包含所有子节点组件的HBox，用于改变组件的宽度和高度。通过参照两个输入框widthInputField和heightInputField中的内容。每个输入框都是通过id属性进行引用，这跟在Document Object Model (DOM) 脚本中通过id引用是一样的。在整个程序中Id值必须是唯一的，可用于指向单级层次结构内部一个组件而不管组件之间的嵌套关系：

Code View:

```

<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="520"
height="650">
<mx:Script>
<! [CDATA[
    private var fileName:String = "";
    private function saveResume():void
    {
        //....a service call to send the data and set the
filename
        fileNameDisplay.text = "Your resume has been saved
as "+fileName;
    }
] ]>
</mx:Script>
<mx:Text id="fileNameDisplay" text="" width="500"/>
<mx:RichTextEditor id="richTextControl" width="500"
height="400"/>
<mx:Button id="labelButton" label="Submit Resume"
click="saveResume()"/>
</mx:VBox>

```

上面的例子中，通过id属性来引用子节点组件并通过id设置其属性值，MXML中的所有添加

进来的组件默认下都是可视的，父组件都可以访问到它们。

## 1.8节. 定义数组和对象

### 1.8.1. 问题

您需要定义数组对象或哈希表 — 样式对象来存储值或其他对象。

### 1.8.2. 解决办法

使用ActionScript语法之构造函数调用创建一个新的对象或数组，或在MXML中定义它们

### 1.8.3. 讨论

数组和对象是最常见的存储数据的两种数据类型，可通过ActionScript或在MXML中定义。要想在MXML中定义数据，是使用`<mx:Array>`标签包裹数组的所有数据项：

```
<mx:Array>
    <mx:String>Flex</mx:String>
    <mx:String>Flash</mx:String>
    <mx:String>Flash Media Server</mx:String>
    <mx:String>Flash Lite</mx:String>
    <mx:String>AIR</mx:String>
</mx:Array>
```

数组中的所有数据都是通过索引进行访问。在MXML中还可以创建多维数组，如：

```
<mx:Array>
    <mx:Array>
        <mx:String>Flex</mx:String>
        <mx:String>Flash</mx:String>
    <mx:Array>
</mx:Array>
```

如要在MXML中创建对象，可使用`<mx:Object>` 标签并添加所有对象属性及其值。例如：

```
<mx:Object id="person" firstName="John" lastName="Smith"
    age="50" socialSecurity="123-45-6789"/>
```

在MXML创建对象有个限制即不能创建多个嵌套的对象，而在script标签中可创建包含多个复杂对象的对象，你可以先创建一个类型为Object的变量，再调用构造函数，最好添加属性：

```
var object:Object = new Object();
var otherObject:Object = new Object();

object.other = otherObject;
```

你还可以通过大括号来创建对象，例如：

Code View:

```
var person:Object = {name:"John Smith", age:22,
    position:{department:"Accounting",
    salary:50000, title:"Junior Accountant"}, id:303};
```

注意**Person**对象的**position**属性指向另一个包含不同属性的对象，还发现这里的**position**对象并不需要先申明。

如要在ActionScript中创建数组，先创建变量然后调用**Array**构造函数：

Code View:

```
var arr:Array = new Array("red", "blue", "white", "black", "green", "yellow");
```

你也可以不调用构造函数来创建数据，而是通过中括号，如：

```
var noConstructorArray:Array = [2, 4, 6, 8, 10, 12, 14, 16];
```

这跟调用**Array**构造函数效果是一样的。

## 1.9节. 在ActionScript中设置变量的作用域

### 1.9.1. 问题

我需要有些变量可公开访问但有些防止被外部访问。

### 1.9.2. 解决办法

使用ActionScript的作用域修饰符。

### 1.9.3. 讨论

无论是在ActionScript或在MXML文件中，变量都有各种作用域。组件中的私有变量和方法只可被其自身所访问，其他组件都无法访问。这样的定义很有用，这些数据只能有一个组件可以修改。当你设计一个复杂类时，最好是把那些外部组件不需要的变量属性设置为私有。公共变量对任何对象都是可见的。应仔细考虑哪些属性是需要被外部访问以及如何限制访问这些属性以便创建一个更好的类结构，要时刻提醒程序员注意这一特性的重要性。对于继承类来说私有属性也是不可见的，只有定义它的类或组件才可见。最后**protected**变量只被定义它的类及其继承类所访问。

标记为私有的变量和函数只在定义它的类中可见，保护的在任何派生类中可见，如下面的例子使用了**protected**和**private**属性：

```
package oreilly.cookbook
{
    public class Transport
    {
        protected var info:Object;
        private var speed:Object;
    }
}
```

通过使用**extends**关键字，你可以共享类属性和方法给另一个类，在下面的例子中，**Car**类继承自**Transport**类，并继承了其非私有的所有属性和方法。**Car**不能访问**Transport**的私有属性，否则会提示错误信息“**属性没有找到**”。

Code View:

```

package oreilly.cookbook
{
    public class Car extends Transport
    {
        public function set data(value:Object):void
        {
            info = value;
            speed = value.speed; /* this will throw an error
because the speed
variable is private and access to it is not allowed to any other
class */
        }
    }
}

```

Transport 类的protected属性可用于Car类，但是不能用于把它作为一个属性的其他类实例中。两个类的任何公共属性都可以被其他类实例访问。任何指定了**static**关键字的类都可以不用实例化而直接访问其属性，换句话说，static定义的变量是所有类实例所共享的。把下面这一行加入到Car类中。

```
public static const NUMBER_OF_WHEELS:int= 4;
```

现在也直接访问NUMBER\_OF\_WHEELS属性而不用类实例化，例如：

```

import oreilly.cookbook.Car;
public class CompositionTest
{
    private var car:Car;
    public function CompositionTest ()
    {
        trace(" a Car object has "+Car.NUMBER_OF_WHEELS+
            "wheels");
    }
}
```

}到目前为止，所讲的变量作用域都是与类相关的，但是不仅仅是类属性，我们还可以在函数内创建或销毁变量，函数内的变量作用域只在函数体内有效。例如：

```

private function processSpeedData():void
{
    var speed:int;
    var measurement:String = "kilometers";
}
```

在这个函数体外面，变量speed是毫无意义的，也就是说函数内任何地方定义的变量其作用域都只能在函数体范围内。下面的代码能正确输出newCar对象：

```

private function makeCars():void
{
}
```

```

for(var i:int = 0; i<10; i++)
{
    var newCar:Car = new Car();
    carArray.push(newCar);
}
trace(newCar);
}

```

newCar的引用在函数返回后便会失效。

ActionScript也支持**final** 修饰符，指示编译器该方法不能被修改了。这个修饰符可以使你定义的方法不被修改和覆盖，例如，在Transport类中定义如下方法：

Code View:

```

public final function drive(speed:Number):void{ /* final and cannot be overriden in
any other class*/ }

```

指示任何继承Transport的类将拥有此方法，其public作用域说明可被任何对象所访问，但是Transport子类无法重新定义它。

## 1.10节. 在ActionScript中创建组件

### 1.10.1. 问题

我想用ActionScript而不是MXML去创建组件。

### 1.10.2. 解决办法

创建ActionScript文件并继承一个Flex库组件。

### 1.10.3. 讨论

除了在MXML中创建组件为，你还可以在ActionScript中创建它们而根本不需要MXML。操作有点不同，只需要几步。首先确定你的类正确定义包名，下面的例子中，组件所在目录是以应用程序级目录开始，然后是oreilly/cookbook/，这就是包名称的意义：

```

package oreilly.cookbook
{

```

另一个区别是任何被包括或引入的类都必须使用全路径导入进来，这包括任何组件即将继承的类，例如这里的mx.containers.Canvas：

```

import mx.containers.Canvas;
import mx.controls.Text;
import mx.controls.Image;
import oreilly.cookbook.Person;

```

```
public class PersonRenderer extends Canvas
{
```

通常类申明后面会列出所有常量或变量。下面的例子中，类的所有私有变量都被列出。这些属性只有组件自身可以访问，其他组件都不可以。要访问这些属性，需要提供get和set方法。Getter和Setter方法是一种常见的用于访问私有变量的方式。

```
private var _data:Object;
private var nameText:Text;
private var ageText:Text;
private var positionText:Text;
private var image:Image;
```

在ActionScript中，构造函数总是公有的，且没有返回值，名称与类名相同，例如：

```
public function PersonRenderer()
{
    super();
```

任何将被加入到组件的组件都有其构造函数，作为addChild方法的参数被加入到显示列表中，他们的属性可以被修改。

```
nameText = new Text();
addChild(nameText);
ageText = new Text();
addChild(ageText);
```

下面的例子中，ageText组件进行手动定位，这是必要的，因为PersonRenderer是一个Canvas，它并不具有布局管理器，不像VBox或HBox组件。

```
ageText.y = 20;
positionText = new Text();
addChild(positionText);
positionText.y = 40;
image = new Image();
addChild(image);
image.y = 60;
}
```

如果组件已经定义了方法处理数据，比如这里的mx.containers.Canvas，你必须重写这些方法来执行自定义行为。要重写则需使用**override**关键字指示编译器试图重写父类的方法，例如：

```
override public function set data(value:Object):void
{
    _data = value;
```

```

        nameText.text = value.name;
        ageText.text = String(value.age);
        positionText.text = value.position;
        image.source = value.image;
    }

override public function getData():Object
{
    return _data;
}

```

这是最后一个方法，很原始的方法，作用域为公开：

```

public function retrievePerson():Person
{
    /* do some special employee processing */
    return null;
}
}

```

要想把这个类添加到其他组件中，可使用下面的ActionScript代码：

```

var renderer:PersonRenderer = new PersonRenderer();
addChild(renderer);

```

或者在MXML中：

```
<renderers:PersonRenderer id="renderer"/>
```

在ActionScript中构造函数是显式调用。

## 1.11节. 使用事件冒泡机制

### 1.11.1. 问题

我想监听从子组件传递到父组件的所有事件而不必创建一连串事件监听器。

### 1.11.2. 解决办法

使用Flash Player的事件冒泡机制监听从子组件传递来的事件。

### 1.11.3. 讨论

我们需要通过几个类来了解冒泡事件，很多类型的事件都可以冒泡：mouse-down事件，click事件，keyboard事件。术语“向上冒泡”指的是事件通过其自身的处理方式从显示列表传递

到应用程序容器，这就像水里的气泡上升到水面那样。当用户点击任何组件时，其事件就会通过层级向上传递，也就意味着父组件也能监听到click事件，无论哪个子组件发出事件，父组件都能收到。要想父组件监听所有子组件发出的某一类型的事件，父组件只需添加一个事件监听器即可收到子组件传递过来的事件。

看看BubblingComponent.mxml中定义的类：

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="200">
<mx:Script>
<! [CDATA[
    private function sendClick():void
    {
        trace(" BubblingComponent:: click ");
    }
]]>
</mx:Script>
<mx:Button click="sendClick()" />
</mx:HBox>
```

该组件包含一个按钮，它会发出click事件并传递给包含它的任何父组件，要监听此事件，使用click处理函数：

```
<cookbook:BubblingComponent click="handleClick()" id="bubbler"/>
```

BubblingHolder包含BubblingComponent，如下面的代码：

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300"
xmlns:cookbook="oreilly.cookbook.*"
creationComplete="complete()">
<mx:Script>
<! [CDATA[
    private function handleClick():void
    {
        trace(" BubblingComponentHolder:: click ");
    }
]]>
</mx:Script>
<cookbook:BubblingComponent click="handleClick()"
id="bubbler"/>
</mx:Canvas>
```

该组件将发出事件给任何监听此事件的组件，当你把BubblingHolder添加到主应用程序文件，甚至会传递给应用程序层。

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
```

```

layout="vertical"
xmlns:cookbook="oreilly.cookbook.*">
<mx:Script>
<! [CDATA[
    public function createName ():void
    {
        name = "Flex Cookbook";
    }
]]>
</mx:Script>
<cookbook:BubblingComponentHolder click="handleClick()"/>
</mx:Application>

```

BubblingComponent.mxml中的click事件将被广播一直传到应用程序层。

MouseEvent类中的事件序列发出关于此事件的信息，比如click事件和click时鼠标的位置，通过显示列表到达所有的子节点，接着子节点接收此事件，然后从显示列表返回到Stage。

Stage 检测 MouseEvent 并传递它到显示列表，直到找到此事件的目标，这就是用户与之交互的目标组件，这称之为捕获阶段。接着，事件目标对象的事件处理函数被调用，这称之为目标阶段，最后，冒泡阶段触发，发送事件返回给显示列表给其他监听器,所有方式重新回到 Stage 阶段。

## 1.12节. 使用代码隐藏模式分离MXML和ActionScript

### 1.12.1. 问题

我想使用代码隐藏模式将ActionScript和MXML代码分离开。

### 1.12.2. 解决办法

在ActionScript创建继承自Flex库的类，添加属性和方法提供相应的功能，然后创建MXML文件并继承你创建的那个类。

### 1.12.3. 讨论

如果你熟悉ASP.NET开发，一定听说过“[代码隐藏](#)”，同样地，如果你熟悉脚本语言 (Ruby on Rails, JavaServer Pages (JSP) 开发, PHP,等等)中采用的应用程序视图和控制器相分离的观念。要控制这些代码便于阅读以及清晰度，最好的策略就是把实际布局元素从代码中分离出来。用此方法开发的程序项目所需的文件数会使得项目操作变得很困难，因为每个组件都要产生两个文件。此外，分离业务逻辑和视图逻辑经常也是很困难的，这样会导致组件中的代码分离部份难以理解。但是还是有很多开发人员喜欢这种方式，因为有时候它能帮助你阐明应用程序的工作原理。

首先看一下“代码隐藏”的后面部分：一个组件继承这个类(mx.containers.Canvas)，包含监听组件被添加到stage的方法以及处理任何事件的方法和专门处理鼠标单击事件的方法。

```
package oreilly.cookbook
```

```

{
    import mx.containers.Canvas;
    import flash.events.Event;
    public class CodeBehindComponent extends Canvas
    {
        public function CodeBehindComponent()
        {
            super();
            addEventListener(Event.ADDED_TO_STAGE,
                addedToStageListener);
        }
        protected function addedToStageListener(event:Event):void
        {
            trace(" Added to Stage from Code Behind ");
        }
        protected function clickHandler(event:Event):void
        {
            trace(" Click handled from component "+event.target);
        }
    }
}

```

在这个例子中，方法被标记为protected和private作用域相当，因为这是代码隐藏的一部分代码，MXML将继承CodeBehindComponent类及其方法：

```

<cookbook:CodeBehindComponent xmlns:mx="http://www.adobe.com/2006
/mxml" width="200"
height="400" xmlns:cookbook="oreilly.cookbook.*">
<mx:Button click="clickHandler(event)"/>
</cookbook:CodeBehindComponent>

```

## 1.13节. 组件属性绑定

### 1.13.1. 问题

我创建的组件中想让其属性是可绑定的，可绑定到其他组件上。

### 1.13.2. 解决办法

创建getter和setter方法，用metadata标签把这些方法标记为Bindable，元数据标签里还包含当属性被设置时其方法所发出的事件名称。

### 1.13.3. 讨论

当属性值发生改变时，在属性上添加Bindable元数据标签，发出相应事件，任何对象都可以被定义为可绑定属性。最佳方法就是通过get和set函数定义这些绑定属性。当属性被设置时，

一个名称和Bindable标签一样的事件发出，例如：

```
package oreilly.cookbook
{
    import flash.events.EventDispatcher;
    import flash.events.Event;
    public class Person extends EventDispatcher
    {
        public static var NAME_CHANGE:String = "nameChange";
        private var _name:String;
        [Bindable(event=NAME_CHANGE)]
        public function get name():String
        {
            return _name;
        }
        public function set name(value:String):void
        {
            dispatchEvent(new Event(NAME_CHANGE));
            _name = value;
        }
    }
}
```

Bindable标签需要属性被设置时所发出的事件名称，下面确认当Person对象的name属性值发生改变时任何绑定的组件都能被通知到。

现在绑定属性被设置为Person对象，你可以在绑定表达式中使用任意Person实例：

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
<mx:Script>
<! [CDATA[
[Bindable]
private var _person:Person;
public function set person(person:Person:void
{
    _person = person;
}
]]>
</mx:Script>
<mx:Label text="{_person.name}"/>
</mx:Canvas>
```

## 1.14节. 使用自定义事件以及事件数据传递

### 1.14.1. 问题

我想使用自定义事件类发出事件以及数据。

### 1.14.2. 解决办法

继承flash.events.Event创建类，创建与事件数据相关的属性。

### 1.14.3. 讨论

有时候我们希望发送事件时也附带数据对象以便监听者不用访问发出事件的对象也能访问数据。渲染器或深度嵌入对象通过发出事件通过多个组件到达监听器并发送数据而不需要监听组件去寻找对象及访问其组件。作为一种解决方案，需要创建一个事件类型的类，在事件构造函数中添加需要的数据类型，记得要调用Event的super方法以便Event对象能正确被实例化，例如：

```
package oreilly.cookbook
{
    import flash.events.Event;
    public class CustomPersonEvent extends Event
    {
        public var person:Person;
        public var timeChanged:String;
        Public function CustomPersonEvent(
            type:String,bubbles:Boolean=false,
            cancelable:Boolean=false,
            personValue:Person=null,
            timeValue:String="")
        {
            super(type, bubbles, cancelable);
            person = personValue;
            timeChanged = timeValue;
        }
        override public function clone():Event
        {
            return new CustomPersonEvent(type, bubbles,
                cancelable, personValue,timeValue);
        }
    }
}
```

在这个自定义Event类，继承的Event.clone方法被重写以便复制CustomPersonEvent自身。如果事件监听器想试图重新发出自定义事件，可以这样写：

```
Private function ustomPersonHandler(event:CustomPersonEvent):void
{
    dispatchEvent(event);
}
```

这个发出的事件并不是先前收到的那个，而是使用clone方法创建的CustomPersonEvent 一个复本，如果clone方法没有被重新则会把CustomPersonEvent的所有属性都被复制，那时clone方法返回的将是flash.events.Event而不会有CustomPersonEvent的任何属性。

## 1.15节.监听键盘事件

### 1.15.1. 问题

我想监听用户的按键，检测哪个键被按下并处理相应事件。

### 1.15.2. 解决办法

为应用程序的stage或组件的keyDown事件添加监听器，读取KeyboardEvents的keyCode属性。

### 1.15.3. 讨论

使用keyDown事件处理器监听KeyboardEvent，这些类都扩展自UIComponent。KeyboardEvent类定义了一个keyCode属性用于存储用户按下的键码，例如：

Code View:

```
<mx:HBox      xmlns:mx="http://www.adobe.com/2006/mxml"      width="400"
height="300"
keyDown="keyHandler(event)"  backgroundColor="#0000ff">
<mx:Script>
<! [CDATA[
import flash.events.KeyboardEvent;
private function keyHandler(event:KeyboardEvent):void
{
    switch(event.keyCode){
        case 13:
            trace(" Enter pressed ");
            break;
        case 32:
            trace(" Space Bar pressed ");
            break;
        case 16:
            trace(" Shift Key pressed ");
            break;
        case 112:
            trace(" F1 pressed ");
            break;
        case 8:
            trace(" Delete pressed ");
            break;
    }
}
```

```

        ]]>
</mx:Script>
<mx:Button label="One"/>
</mx:HBox>
```

请注意这个类，只有当button被激活时才会监听到它发出的事件。如果你删除了button，那么就没有东西可以激活了，keyHandler函数也永远不会被调用。要想在程序中捕获所有的KeyEvents事件而不管有没有组件被激活，请添加下面的句子：

Code View:

```
addedToStage="stage.addEventListener(KeyboardEvent.KEY_DOWN,keyHandler)"
```

## 1.16节. 定义方法参数

### 1.16.1. 问题

我想定义一个方法，其参数有默认值或null值，以便调用方法时不必每次都进行传值。

### 1.16.2. 解决办法

在方法申明时直接对方法参数进行赋值，赋予默认值或null值。

### 1.16.3. 讨论

要想为方法定义一个或多个可选参数，最简单的办法就是为参数对象设置为默认值或null。ActionScript基本类型String, Number, int, 和Boolean不能设置为null值，不过可以设置一个默认值，例如：

```

public function optionalArgumentFunction(value:Object,
    string:String, count:int = 0, otherValue:Object = null):void
{
    if(count != 0)
    {
        /*if the count is not the default value handle the value
        the call
        passes in*/
    }
    if(otherValue != null)
    {
        /* if the otherValue is not null handle the value the
        call passes in */
    }
}
```

还有一个策略就是在变量名前使用...标记定义可选参数以及不定数量的参数。该变量将包含一个参数数组，可被循环遍历处理。

```

public function unlimitedArgumentsFunction(...arguments):void
{
    for each(var arg:Object in arguments)
    {
        /* process each argument */
    }
}

```

## 1.17节. 检测对象数据类型

### 1.17.1. 问题

我想检测下传入到方法的对象是什么类型。

### 1.17.2. 解决办法

使用`is`操作符检测对象类型或者是父类对象的`type`属性。

### 1.17.3. 讨论

要检测一个对象的类型，ActionScript提供了`is`操作符，检测对象类型并返回`true`或`false`。如果对象与测试目标一致或是其子类则返回`true`，比如，因为`Canvas`对象继承自`UIComponent`，`is`操作符返回`true`。如果`UIComponent`测试它为`Canvas`类型则返回`false`。因为`UIComponent`并不是继承自`Canvas`。看下面的代码：

```

public function TypeTest()
{
    var uiComponent:UIComponent = new UIComponent();
    var canvas:Canvas = new Canvas();
    trace(" uiComponent is UIComponent "+(uiComponent is
        UIComponent));
    trace(" uiComponent is Canvas "+(uiComponent is
        Canvas));
    trace(" canvas is UIComponent "+(canvas is
        UIComponent));
}

```

输出一下内容：

```

uiComponent is UIComponent true
uiComponent is Canvas false
canvas is UIComponent true

```

类型检测最常用到的地方是当组件抛出一个事件时。在事件处理函数中检测是什么对象发出动作。

```

private function eventListener(mouseEvent:MouseEvent):void
{
    if(mouseEvent.target is Button)

```

```

{
    /* handle button specific actions */
}
else if(mouseEvent.target is ComboBox)
{
    /* handle combobox specific things */
}
else
{
    /* handle all other cases */
}
}

```

## 1.18节. 接口的定义和实现

### 1.18.1. 问题

我想创建一个接口，并创建一个组件实现这个接口。

### 1.18.2. 解决办法

创建一个ActionScript文件，申明此文件为一个接口，定义此接口需要的任意方法。要实现此接口，在定义类时使用**implements**关键字。

### 1.18.3. 讨论

接口是一个很强大的工具，它描述一个契约，所有实现它的类都必须完全按照接口所定义的方法包括作用域，名称，参数和返回值保持一致。反过来使用此对象的组件希望这组方法已存在，这样只需要创建一个轻量级的类申明而不需要创建一个新类来破坏你的继承树关系。实现接口的类也被认为是接口类型，这常被用来设置方法参数的类型或者方法的返回类型，例如：

```
public function pay(payment:IPaymentType):IReceipt
```

这个方法接受实现IPaymentType接口的任何对象，以及返回实现IReceipt接口的对象。

接口中不能定义方法体以及任何变量，在下面的代码片段中，IDataInterface 申明和定义了5个方法，任何实现此接口的对象都必须定义这些方法：

```

package oreilly.cookbook
{
    public interface IDataInterface
    {
        function set dataType(value:Object) :void;
        function get dataType() :Object;
        function update() :Boolean;
        function write() :Boolean;
        function readData():Object;
    }
}
```

}

要实现这个接口，申明类并添加`implements`标记到类申明中，所有接口中定义的方法都必须被实现。在下面的代码中，所有接口方法被包含进来并提供函数体：

```
package oreilly.cookbook
{
    import flash.events.EventDispatcher;
    import flash.events.IEventDispatcher;
    public class ClientData extends EventDispatcher implements
        IDataInterface
    {
        private var _dataType:Object;
        public function ClientData(target:IEventDispatcher=null)
        {
            super(target);
        }
        public function set dataType(value:Object):void
        {
            _dataType = value;
        }
        public function get dataType():Object
        {
            return _dataType;
        }
        public function update():Boolean
        {
            //do the actual updating
            var updateSuccessful:Boolean;
            if(updateSuccessful)
            {
                return true;
            }
            else
            {
                return false;
            }
        }
        public function write():Boolean
        {
            var writeSuccess:Boolean;
            if(writeSuccess)
            {
                return true;
            }
            else
```

```
        {
            return false;
        }
    }
public function readData():Object
{
    var data:Object;
    //get all the data we need
    return data;
}
}
```

如果在MXML中实现一个接口，在顶层的组件中使用implements，例如：

```
<mx:HBox      xmlns:mx="http://www.adobe.com/2006/mxml"      width="400"
height="300" implements= "IDataInterface">
```

## 第二章. 控件与菜单(Native|eas)

### 2.1 节. 监听按钮点击

#### 2.1.1. 问题

我想执行一段任务以便对用户的交互作出响应, 比如当用户点击一个按钮时在控制台输出名称列表。

#### 2.1.2. 解决办法

使用`<mx:Button>`标签的`click`事件属性来设定一个`click`事件处理程序。也可以在ActionScript中通过`Button`实例的`addEventListener`方法来添加`click`事件的监听器来达到同样的效果。

#### 2.1.3. 讨论

如下的代码展示了如何在MXML中是用`<mx:Button>`的`click`事件属性来监听一个按钮行为:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">
    <mx:Button id="btn" label="Show Names"
        click="showNames(event)"/>

    <mx:Script>
        <![CDATA[
            private function showNames(evt:MouseEvent):void
            {
                var temp:Array =
                    new Array("George", "Tim", "Alex", "Dean");
                trace(temp.toString());
            }
        ]]>
    </mx:Script>

</mx:Application>
```

上面的代码创建了一个包含名为 btn 的按钮控件实例的应用程序。当这个 btn 的按钮实例被点击时，应用程序会输出一列名字到控制台。可以看到 btn 这个按钮的 click 事件属性指向了一个方法 showNames：

```
<mx:Button id="btn" label="Show Name click="showNames(event)" />
```

每当用户点击这个按钮，Flex 框架会发布一个 MouseEvent.CLICK 事件。上面这行代码设定了在每次 click 事件被发布的时候，都去调用 showNames 方法，然后一个名字的数组就被创建以及输出到了控制台中。需要注意的是，MouseEvent 类型的事件实例会被自动传递到处理的函数。根据发出的事件，查询此对象可以获得更多关于该事件本身的信息。在调试模式（Eclipse 中 F11）中运行程序，你就会看到控制台窗口有如下输出。

George,Tim,Alex,Dean

事件监听器也能通过 ActionScript 来设定。下面的例子就是使用 ActionScript 设定了 showNames 和 showTitles 两个监听器到 btn 这个按钮实例。

code view:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    creationComplete="initApp(event);">

    <mx:Button id="btn" label="Show Names"/>

    <mx:Script>
        <![CDATA[
            import mx.events.FlexEvent;

            private function initApp(evt:FlexEvent):void
            {
                btn.addEventListener(MouseEvent.CLICK, showNames);
                btn.addEventListener(MouseEvent.CLICK, showTitles);
            }

            private function showNames(evt:MouseEvent):void
            {
                var temp:Array =
                    new Array("George", "Tim", "Alex", "Dean");
                trace(temp.toString());
            }

            private function showTitles(evt:MouseEvent):void
            {
        
```

```

        var temp:Array = new Array("Director", "Vice-
        President", "President", "CEO");
        trace(temp.toString());
    }

] ]>
</mx:Script>

</mx:Application>

```

注意，<mx:Application>的 creationComplete 事件用来设定了按钮点击事件的 2 个监听器，showNames 和 showTitles.

```

private function initApp(evt:FlexEvent):void
{
    btn.addEventListener(MouseEvent.CLICK,showNames);
    btn.addEventListener(MouseEvent.CLICK,showTitles);
}

```

Running this application in debug mode generates the following output in the Console window:

在调试模式中运行这个程序，控制台窗口会输出如下内容：

```

George,Tim,Alex,Dean
Director,Vice-President,President,CEO

```

监听器根据被注册时的顺序进行调用。因为 showNames 比较 showTitles 早注册监听，所以名字列表比标题列表先生成 输出。如果要改变他们（监听器）的执行顺序，可以修改他们对按钮注册事件监听的顺序。甚至，你可以在注册监听器时设定它们的优先级，例如下面的代码：

```

private function initApp(evt:FlexEvent):void
{
    btn.addEventListener(MouseEvent.CLICK,showNames, true);
    btn.addEventListener(MouseEvent.CLICK,showTitles);
}

```

使用修改后的代码，再次在调试模式中运行程序，我们可以看到：

```

Director,Vice-President,President,CEO
George,Tim,Alex,Dean

```

使用大的优先级值 (priority) 注册的监听器会比使用小优先级的优先级注册的监听器更早的运行。如果有超过一个监听器适用了相同的优先级值，则执行顺序还是依照监听器注册的顺序。

## 2.2 节. 创建一组状态按钮

### 2.2.1. 问题

我需要提供一系列按钮供用户选择。

### 2.2.2. 解决办法

使用 ToggleButtonBar 组件以及用来创建一组按钮的 ArrayCollection。

### 2.2.3. 讨论

创建一个带 ToggleButtonBar 的应用程序来放置创建的一列按钮。ToggleButtonBar 定义了一套用来维护按钮选择状态的纵向或者横向按钮组。具体方法如下：

Code View:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    initialize="initApp(event)">

    <mx:ToggleButtonBar id="toggle"
        dataProvider="{dataProvider}"
        itemClick="setMode(event)"/>

    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;
            import mx.events.FlexEvent;
            import mx.events.ItemClickEvent;

            [Bindable]
            private var dataProvider:ArrayCollection;

            private function initApp(evt:FlexEvent):void {
                var temp:Array = new Array({label:"Show Labels",

```

```

        mode:"labels"},
        {label:"Show Titles",
        mode:"titles}));
dataProvider = new ArrayCollection(temp);
}

private function setMode(evt:ItemClickEvent):void {
    switch (evt.item.mode) {
        case "labels":
            trace("George, Tim, Dean");
            break;
        case "titles":
            trace("Vice President, President, Director");
            break;
        default:
            break;
    }
}
]]>
</mx:Script>

</mx:Application>
```

当应用程序初始化的时候， initialize 事件会调用 initApp 方法。 initApp 方法会使用一个 ArrayCollection 来设定 ToggleButtonBar 实例的数据源 dataProvider. 因为 ToggleButtonBar 实例的 dataProvider 属性被限制成 ArrayCollection 类型。它会反射出新的按钮来更新显示的内容。

默认情况。 ArrayCollection 中组成元素的 label 属性，会被 ToggleButtonBar 实例中按钮的标签使用。如果你需要使用任意其他的属性（例如 mode）来用作按钮的标签，可以使用 ToggleButtonBar 的 labelField 属性，代码如下：

```
<mx:ToggleButtonBar id="toggle"
    dataProvider="{dataProvider}" labelField="mode"
    itemClick="setMode(event)"/>
```

ToggleButtonBar 实例的 itemClick 事件的处理函数被设定为 setMode。 ItemClickEvent 类型的事件实例都会被传递到 setMode 方法。 ItemClickEvent 实例的 item 属性引用了 dataProvider 里面的对应一致的元素。

通常的情况下，当应用程序启动后，ToggleButtonBar 没有选定的按钮，等待用户来选择一个按钮。无论如何，当 dataProvider 被赋值后，第一个按钮会被默认选中。幸运的是，Flex3 提供了 SDK 的源代码，所以你能够使用这些知识来扩展 ToggleButtonBar 控件来适应你的需求。ToggleButtonBar 的源代码可以在如下位置找到：

Code View:

```
<Flex 3 installation  
dir>/sdks/3.0.0/frameworks/projects/framework/src/mx/controls/ToggleBarButton.as
```

HighlightSelectedNavItem 方法提供了如何来取消按钮的选择的线索。它能获取一个当前选中按钮的引用，并且取消这个按钮的选中状态。

```
child = Button(getChildAt(selectedIndex));  
child.selected = false;
```

你可以利用从 Flex 框架代码中的信息来创建一个特殊版本的 ToggleBarButton，符合特殊的需求。例如，让所有的按钮在开始的时候都是未选中 状态。

如下代码展示了一个 ToggleBarButton 类的子类 CustomToggleButtonBar 。它是用来实现任意时间改变 dataProvider 都会取消选择所有按钮。这里它覆盖了 ToggleBarButton 的 dataProvider setter 方法以及使用了一个叫做 dataReset 的布尔值用来追踪控件中的 dataProvider 是否被重设过。控件中的 updateDisplayList 方法也会被覆盖用来实现每次 dataProvider 被重设后都会取消选择当前被选中的按钮。在显示列表被更新 后， dataReset 这个标记会被设置回默认状态。 代码如下

Code View:

```
package {  
    import mx.controls.Button;  
    import mx.controls.ToggleBarButton;  
  
    public class CustomToggleButtonBar extends ToggleBarButton  
    {  
        public function CustomToggleButtonBar() {  
            super();  
        }  
  
        private var dataReset:Boolean = false;  
        override public function set  
        dataProvider(value:Object) :void {  
            super.dataProvider = value;  
            this.dataReset = true;  
        }  
  
        override protected function  
        updateDisplayList(unscaledWidth:Number,  
        unscaledHeight:Number) :void {  
            super.updateDisplayList(unscaledWidth,unscaledHeight);
```

```

        if(this.dataReset) {
            if(selectedIndex != -1) {
                var child:Button;
                child = Button getChildAt(selectedIndex);
                if(child) {
                    child.selected = false;
                    this.dataReset = false;
                }
            }
        }
    }
}

```

简单的操作就能使用新的组件 CustomToggleButtonBar 来替换原先的 ToggleButtonBar, 实现如下:

```

<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:local="*"
    layout="vertical"
    initialize="initApp(event)">

    <local:CustomToggleButtonBar id="toggle" selectedIndex="-1"
       dataProvider="{dataProvider}" itemClick="setMode(event)"/>

```

如你所见, 扩展默认的控件来适应你的应用程序的需求实现起来非常之简单。阅读 Flex SDK 源代码并不是唯一的学习 Framework 的方法, 但是它也提供了很多扩展默认组件的想法。需要注意的是, 有一些文档没有记录的在 mx\_internal 命名空间下的特性 或者属性/方法之类不要去使用, 因为这些都是有可能在未来的 SDK 版本中有巨大改变的。

## 2.3 节. 使用 **ColorPicker** 设置 **Canvas** 颜色

### 2.3.1. 问题

我想要让用户使用色彩选取器来修改一个组件的颜色。

### 2.3.2. 解决办法

为用户提供一个用来选取颜色的调色板。并且使用 ColorPicker 控件的 change 事件来设定 Canvas 的背景颜色。

### 2.3.3. 讨论

让用户使用一个调色板。创建一个程序，使用 ColorPicker 控件来改变 Canvas 控件的 backgroundColro 属性(背景色)。 ColorPicker 控件提供给用户一种从色彩采样选取颜色的方式。为了实现我们的需求,Colorpicker控件的 change 事件的处理程序被指 向到 setColor 方法。 setColor 方法会接受一个包含了当前选中颜色的 ColorPickerEvent。具体实现代码如下：

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">

    <mx:Canvas id="cnv" width="450" height="450"
        backgroundColor="#eeeeaa">
        <mx:ColorPicker id="pckr" right="10" top="10"
            change="setColor(event)"/>
    </mx:Canvas>

    <mx:Script>
        <! [CDATA[
            import mx.events.ColorPickerEvent;

            private function setColor(evt:ColorPickerEvent):void
            {
                cnv.setStyle("backgroundColor",evt.color); }
        ]]>
    </mx:Script>

</mx:Application>
```

When the user selects a new color, the backgroundColor style of the Canvas is updated. Note that because backgroundColor is a style attribute rather than a property of the Canvas control, the setStyle method is used to update the style as shown:

当用户选择了一个新的颜色, Cancas 控件的 backgroundColor 样式就会更新。值得注意的是, backgroundColor 是一个样式而不是 Canvas 的普通属性, 所以, 更新 backgroundColor 需要使用 setStyle 方法来操作, 譬如:

```
private function setColor(evt:ColorPickerEvent):void
{
    cnv.setStyle("backgroundColor",evt.color); }
```

## 2.4 节. 使用 SWFLoader 载入 SWF

### 2.4.1. 问题

我想要运行时载入外部的可能由 Flex3 或者 FlashCS3 创建的 SWF 文件到当前的 Flex 应用程序。

### 2.4.2. 解决办法

使用 SWFLoader 组件在运行时载入外部 SWF 文件。

### 2.4.3. 讨论

使用 SWFLoader 组件在运行时载入外部 SWF 文件。下列例子载入了外部的 SWF 到 TabNavigator 的一个子元件 Canvas 容器 中。SWFLoader 的 source 属性引用了需要被运行时载入的外部 SWF 文件的路径。Sub1.swf 是一个 Flex3 应用程序; Sub2.swf 由 FlashCS3 创建。

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">

    <mx:TabNavigator resizeToContent="true"
        paddingTop="0">
        <mx:Canvas>
            <mx:SWFLoader source="assets/Sub1.swf"/>
        </mx:Canvas>
        <mx:Canvas>
            <mx:SWFLoader source="assets/Sub2.swf"/>
        </mx:Canvas>
    </mx:TabNavigator>
</mx:Application>
```

SWFLoader 组件也能够载入已经被嵌入到 Flex 应用程序中的 SWF 内容。使用 Embed 命令能实现。例子如下，Sub2.swf 会被嵌入到主应用程序。

```
<mx:SWFLoader source="@Embed('assets/Sub2.swf')"/>
```

## 2.5 节. 设置组件的标签索引

### 2.5.1. 问题

我需要改变 Flex 应用程序中默认的组件索引顺序

### 2.5.2. 解决办法

使用 Flex 组件的 `tabIndex` 属性来指定组件的特定组件的索引顺序。

### 2.5.3. 讨论

默认情况，所有可被 Tab 访问的 Flex 组件（指可以通过 Tab 键顺序访问的组件）都拥有基于屏幕布局的索引顺序。在如下例子中，`TextInput` 组件的 `tabIndex` 属性会被设定为从左到右的 Tab 索引顺序：

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal">

    <mx:VBox>
        <mx:Label text="First Name : "/>
        <mx:TextInput tabIndex="1"/>
        <mx:Label text="Home # : "/>
        <mx:TextInput tabIndex="3"/>
    </mx:VBox>
    <mx:VBox>
        <mx:Label text="Last Name : "/>
        <mx:TextInput tabIndex="2"/>
        <mx:Label text="Work # : "/>
        <mx:TextInput tabIndex="4"
            text="978-111-2345"/>
        <mx:Button label="Submit" tabIndex="5"/>
    </mx:VBox>
</mx:Application>
```

如果 Tab 索引没有被设定过，默认的顺序是屏幕上从上到下。组件的 `tabIndex` 属性也会被也能通过编程方式使用 ActionScript 来设定，如运行时动态创建子元件的自定义组件。如需要控制 Tab 索引顺序即可如此进行。

## 2.6 节. 设置控件的 labelFunction

### 2.6.1. 问题

我需要组合一个数据提供器中不同的字段来自定义 ComboBox 组件的显示文本。

### 2.6.2. 解决办法

使用 ComboBox 组件的 labelFunction 属性来指定定义显示文本的自定义函数。

### 2.6.3. 讨论

默认情况下，在 Flex 中基于 List 的控件都是使用 dataProvider 中的元素的 label 属性来做显示。在一些情况中，无论如何，dataProvider 中都没有 label 属性存在，这些情况就需要你来设定连接 dataProvider 中的多个字段来实现一个显示值。这个 labelFunction 属性允许用户定义自己的方法来呼叫 dataProvider 中每个元素，然后对于每个元素返回显示值。如下范例，ComboBox 的 labelFunction 属性包含了一个 getFullName 函数的引用，这个函数连接了 dataProvider 中单个元素的 fName 和 lName 字段来返回一个全名的字串。

Code View:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal">

    <mx:ComboBox dataProvider="{myDP}"
        labelFunction="getFullName"/>

    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            private var myDP:ArrayCollection =
                new ArrayCollection([
                    {id:1,fName:"Lucky", lName:"Luke"},
                    {id:2, fName:"Bart", lName:"Simpson"}]);
            private function getFullName(item:Object):String{
                return item.fName + " " + item.lName;
            }
        ]]>
    </mx:Script>
</mx:Application>
```

## 2.7 节. 提供菜单数据

### 2.7.1. 问题

我需要通过数据提供器来创建一个菜单栏

### 2.7.2. 解决办法

为 mxml 中定已的MenuBar 控件的 dataProvider 属性分配一个 Collection 对象（例如 ArrayCollection 或者 XMLListCollection）。

### 2.7.3. 讨论

在 MXML 中使用数据来填充MenuBar 控件的最简单办法，就是在控件内部创建一个 XMLList 的实例。

Code View:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal">

    <mx:MenuBar labelField="@label">
        <mx:XMLList>
            <menuitem label="File">
                <menuitem label="New"/>
                <menuitem label="Open"/>
                <menuitem label="Close" enabled="false"/>
            </menuitem>
            <menuitem label="Edit"/>
            <menuitem label="Source"/>
            <menuitem label="View">
                <menuitem label="50%" type="radio" groupName="one"/>
                <menuitem label="100%" type="radio" groupName="one" selected="true"/>
                <menuitem label="150%" type="radio" groupName="one"/>
            </menuitem>
        </mx:XMLList>
    </mx:MenuBar>
</mx:Application>
```

```
</mx:MenuBar>  
</mx:Application>
```

因为 `dataProvider` 属性是 `MenuBar` 控件的默认属性，所以这个 `XMLList` 对象可以作为 `<mx:MenuBar>` 的直接子级。该 `XMLList` 对象的顶级节点和 `MenuBar` 上的按钮相符合，而 `menuItem` 节点则匹配每个顶级按钮下面的菜单项的层级。这些节点可以被命名成任意名字，例如，我们可以用 `subnode` 来替代 `menuItem`。节点的属性，会有一些特殊的意思并且会影响显示效果以及菜单的用户交互动作。这些属性如下所列。

#### `enabled`

指定用户是否可以选中该菜单项。

#### `groupName`

适用于当菜单项是单选框类型的按钮时，指定单选框组的名字用来给菜单项分组。

#### `icon`

指定一个图像素材的类标识符。

#### `label`

指定菜单项的显示文本。注意当 `dataProvider` 采用 E4X 格式的时候，就如上面的代码范例，`MenuBar` 的 `labelFiele` 属性必须被明确指定。即使 `dataProvider` 中已经有了 `label` 属性。

#### `toggled`

当菜单项是复选框或者单选框类型时，指定是否被选中。

#### `type`

指定如下菜单类型，例如：check, radio, separator。

## 2.8. 动态填充菜单

### 2.8.1. 问题

动态的来填充和修改一个菜单栏。

## 2.8.2. 解决办法

使用 ActionScript 为MenuBar 控件的dataProvider 属性分配一个 Collection 对象（例如 ArrayCollection 或者 XMLListCollection）。

## 2.8.3. 讨论

Flex3 中的MenuBar 控件支持运行时菜单栏动态创建。本节会创建一个带有MenuBar 控件的程序，在程序初始化的时候使用一个 ArrayCollection 来填充这个控件。

Code View:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    creationComplete="initApp(event)">

    <mx:MenuBar id="menu" dataProvider="{menu_dp}" />

    <mx:Script>
        <! [CDATA[
            import mx.collections.ArrayCollection;
            import mx.events.FlexEvent;

            [Bindable]
            private var menu_dp:ArrayCollection;

            private function initApp(evt:FlexEvent):void {
                var temp:Array = new Array();
                var subNodes:ArrayCollection = new ArrayCollection([
                    {label:"New"}, {label:"Open"},
                    {label:"Close", enabled:false}]);
                temp.push({label:"File", children:subNodes});
                temp.push({label:"Edit"});
                temp.push({label:"Source"});

                subNodes = new ArrayCollection([
                    {label:"50%", type:"radio", groupName:"one"},
                    {label:"100%", type:"radio",
                        groupName:"one", selected:true},
                    {label:"150%", type:"radio", groupName:"one"}]);
                temp.push({label:"View", children:subNodes});
                menu_dp = new ArrayCollection(temp);
            }
        ]]>
```

```
</mx:Script>  
</mx:Application>
```

如上代码使用了绑定机制来绑定 menu\_db 这个 ArrayCollection 对象到MenuBar 组件的dataProvider 属性。在 Application 的 creationComplete 事件发出的时候，menu\_dp 会被初始化并且填充菜单栏。和 flex 中其他的数据驱动组件一样，使用 Collection 对象（例如 ArrayCollection 或者 XMLListCollection）可以确保数据的任意变化都会触发控件相关的更新显示。

Collection 类提供了从菜单中编辑，添加以及删除项目的建议方法。做个示范，下面的例子在MenuBar 下面添加了简单的 Form 控件，允许你通过 ArrayCollection 中项的索引，来编辑相应菜单的项。

```
<mx:Form>  
    <mx:FormHeading label="Menu Editor"/>  
    <mx:FormItem label="Menu Index">  
        <mx:TextInput id="menuIdx" restrict="0-9" text="0"  
            width="20"/>  
    </mx:FormItem>  
    <mx:FormItem label="Sub-Menu Index">  
        <mx:TextInput id="subMenuIdx" restrict="0-9"  
            width="20"/>  
    </mx:FormItem>  
    <mx:FormItem label="Menu Label">  
        <mx:TextInput id="label_ti"/>  
    </mx:FormItem>  
    <mx:FormItem>  
        <mx:Button label="Edit" click="editMenu()"/>  
    </mx:FormItem>  
</mx:Form>
```

这是一个基础的表单，使用了输入控件允许你来指定数组的索引来获取指定的菜单项。在 menuIdx 输入框输入 0 然后在 subMenuIdx 输入框留空就会得到顶层 File 菜单。在 menuIdx 输入框和 subMenuIdx 输入框都输入 0 会返回 New 子菜单项。

当用户点击了 Edit 按钮，editMenu 方法会被调用，指定的索引来获取菜单项的索引并且修改它的显示文字。如下所示：

Code View:

```
private function editMenu():void {  
    var itemToEdit:Object;  
    try {  
        itemToEdit = menu_dp.getItemAt(int(menuIdx.text));  
        if (subMenuIdx.text) {
```

```

        itemToEdit =
            itemToEdit.children.getItemAt(int(subMenuItemIdx.text));
    }
    itemToEdit.label = label_ti.text;
    menu_dp.itemUpdated(itemToEdit);
}
catch(ex:Error){
    trace("could not retrieve menu item");
}
}

```

editMenu 方法的代码会见识 menuIdx 和 subMenuItemIdx 的输入值，来获取他们指定的菜单项，并且使用 label\_ti 的值来更新获取到的菜单项的显示文本。需要注意的是编辑菜单显示的运作顺序，dataProvider 和MenuBar 联合以后，可以再修改数据之后，通过 ArrayCollection 的 itemUpdated 方法来做显示更新的请求。特别是和此例一样的嵌套数据使用的时候，呼叫 itemUpdated 方法来请求显示内容更新是非常重要的。不然的话，数据会修改但是显示可能还是显示旧的数据。例子里面使用了一个 try..catch 块劳作一个基础的错误处理。

## 2.9 节. 为菜单类控件创建事件处理函数

### 2.9.1. 问题

我想让菜单对用户的交互作出响应。

### 2.9.2. 解决办法

给MenuBar 控件的 itemClick 事件添加事件监听器。

### 2.9.3. 讨论

为MenuBar 控件的 itemClick 事件指定一个监听处理函数 handleMenuItemClick 来处理菜单栏交互点击。当用户选则一个菜单项的时候 itemClick 事件就会被触发。监听函数会接收到作为参数传来的 MenuEvent 对象。MenuEvent 对象包含了发出该事件对象的菜单项的信息。MenuEvent 对象的 item 属性包含了 dataProvider 中的一个对应唯一菜单项的项的引用。代码如下：

Code View:

```

<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"

```

```

layout="vertical">

<mx:MenuBar
    labelField="@label"
    itemClick="handleMenuItemClick(event)">
    <mx:XMLList>
        <menuitem label="File">
            <menuitem label="New"/>
            <menuitem label="Open"/>
            <menuitem label="Close" enabled="false"/>
        </menuitem>
        <menuitem label="Edit"/>
        <menuitem label="Source"/>
        <menuitem label="View">
            <menuitem label="50%" type="radio" groupName="one"/>
            <menuitem label="100%" type="radio" groupName="one" selected="true"/>
            <menuitem label="150%" type="radio" groupName="one"/>
        </menuitem>
    </mx:XMLList>
</mx:MenuBar>

<mx:Label id="disp0_lbl"/>

<mx:Script>
<! [CDATA[
    import mx.events.MenuEvent;

    private function handleMenuItemClick(evt:MenuEvent):void {
        this.disp0_lbl.text = evt.item.@label + " was
            selected";
    }
]]>
</mx:Script>
</mx:Application>

```

注意，因为 dataProvider 使用了 E4X 格式，所以例子使用了 E4X 的语法@label 来接受 label 属性。MenuBar 控件同事也支持其他的事件类型，例如 change,itemRollOut,itemRollOver,menuHide 以及 menuShow.

## 2.10 节. 显示一个通知窗口

### 2.10.1. 问题

我需要向用户显示一个模式消息并提供可供用户选择的操作选项

### 2.10.2. 解决办法

使用 Alert 控件来显示信息。

### 2.10.3. 讨论

Alert 控件提供了一个带按钮的模式窗口，用户可以点击来回答对话框的消息。Alert 控件不能 MXML 创建。你必须使用 ActionScript 才能实现，例如：

Code View:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">

    <mx:Button id="btn" click="showAlert(event)" label="Alert"/>
    <mx:Label id="lbl"/>

    <mx:Script>
        <! [CDATA[
            import mx.events.CloseEvent;
            import mx.controls.Alert;
            import mx.events.MenuEvent;

            private function showAlert(evt:MouseEvent):void {
                var alert:Alert = Alert.show(
                    "Button was clicked", "Alert Window Title",
                    Alert.OK|Alert.CANCEL|Alert.NO|Alert.YES,
                    this,
                    onAlertClose);
            }

            private function onAlertClose(evt:CloseEvent):void {
                switch(evt.detail) {
                    case Alert.OK:
                        lbl.text = "OK Clicked";
                        break;
                    case Alert.CANCEL:
                }
            }
        ]]>
    </mx:Script>

```

```
        lbl.text = "CANCEL Clicked";  
        break;  
    case Alert.NO:  
        lbl.text = "NO Clicked";  
        break;  
    case Alert.YES:  
        lbl.text = "YES Clicked";  
        break;  
    }  
}  
]  
>  
</mx:Script>  
</mx:Application>
```

当用户点击按钮 btn 的时候，范例代码通过 Alert 类中的静态方法创建了一个 Alert 控件。show 方法接受如下参数来配置 Alert。

#### **text**

需要显示给用户的信息

#### **title**

Alert 对话框的标题。

#### **flags**

Alert 中显示的按钮。可用的值包括 Alert.OK, Alert.CANCEL, Alert.NO 以及 Alert.Yes。显示多个按钮可以使用或位操作符|。例如 Alert.OK|Alert.CANCEL.

#### **parent**

Alert 相对剧中显示的显示元件。

#### **closeHandler**

在 Alert 控件上任意按钮被点击时呼叫的事件处理程序

#### **iconClass**

Icon 的资源类， icon 会被放置到 Alert 中显示信息的左方。

#### **defaultButtonFlag**

设定 Alert 控件的默认按钮。按回车键会触发默认按钮，可用的值包括 Alert.OK, Alert.CANCEL, Alert.NO, or Alert.Yes.

`onAlertClose` 方法被设定为 Alert 的 `closeHandler` 事件处理函数。这个方法会接受一个 `CloseEvent` 对象的参数，然后使用 `CloseEvent` 的 `detail` 属性来决定 Alert 中哪个按钮被点击。

## 2.11 节. 使用 **Calendar** 控件

### 2.11.1. 问题

你想要允许用户在一个日历样式的控件中选择日期。

### 2.11.2. 解决办法

使用 `DateField` 控件或 `DataChooser` 控件让用户选择日期。

### 2.11.3. 讨论

Flex 框架提供了两个控件用来实现日历样的功能：`DateField` 和 `DateChooser` 控件。`DateField` 控件提供了一个 `TextInput` 控件和一个点击可以打开日历的日历图标。`DateChooser`，则不同，它提供了一个可视的日历给用户操作。下列范例是一个简单的旅程计算器用来展示两种类型的控件。用户使用 `DateField` 选择开始日期，使用 `DateChooser` 选择结束日期。程序会在控件的 `change` 事件触发 `update` 事件处理函数中计算旅途持续时间。两个控件的 `selectedDate` 属性会返回用户选择的 `Date` 对象。

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">
    <mx:Form>
        <mx:FormHeading label="Trip Calculator"/>
        <mx:FormItem label="Start Date">
            <mx:DateField id="startDate" change="update(event)"/>
        </mx:FormItem>
        <mx:FormItem label="End Date">
            <mx:DateChooser id="endDate" change="update(event)"/>
        </mx:FormItem>
        <mx:FormItem label="Trip Duration (days)">
            <mx:Label id="display"/>
        </mx:FormItem>
    </mx:Form>
    <mx:Script>
        <![CDATA[
            import mx.events.CalendarLayoutChangeEvent;
        ]]>
    
```

```

private static const MILLISECONDS:int = 1000;
private static const SECONDS:int = 60;
private static const MINUTES:int = 60;
private static const HOURS:int = 24;
private function
update(evt:CalendarLayoutChangeEvent):void {
    try {
        var diff:Number = endDate.selectedDate.getTime()
            -startDate.selectedDate.getTime();
        // convert the millisecond into days
        var days:int =
            int(diff/(MILLISECONDS*SECONDS*MINUTES*HOURS));
        display.text = days.toString();
    }
    catch(ex:Error) {
    }
}
] ]>
</mx:Script>
</mx:Application>

```

在日期运算中，使用 Date 对象的 getTime 方法是非常重要的，以便正确处理 闰年。getTime 方法返回从 1970 年 1 月 1 日起流逝的毫秒数。

## 2.12 节. 弹出窗口的显示和位置

### 2.12.1. 问题

我想要在使用弹出窗口时给用户显示附加信息。

### 2.12.2. 解决办法

通过用户交互，使用 PopUpManager 来创建 TitleWindow 组件实例

### 2.12.3. 讨论

Flex 框架包含了一个 PopUpManager 类，它包含了若干静态方法来管理穿件，放置，移除 Flex 应用程序的顶级窗口。如下代码所示：

Code View:

```
<mx:Application
```

```

xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute">

<mx:Canvas horizontalCenter="0" verticalCenter="0">

    <mx:LinkButton label="Top" x="100" y="10"
        click="showDetail(event)"/>
    <mx:LinkButton label="Left" x="10" y="100"
        click="showDetail(event)"/>
    <mx:LinkButton label="Bottom" x="100" y="200"
        click="showDetail(event)"/>
    <mx:LinkButton label="Right" x="200" y="100"
        click="showDetail(event)"/>
    <mx:Canvas width="100" height="100" x="125" y="40"
        backgroundColor="#ff0000" rotation="45">
        </mx:Canvas>
    </mx:Canvas>

<mx:Script>
<![CDATA[
    import mx.managers.PopUpManager;
    private const POPUP_OFFSET:int = 10;

    private function showDetail(evt:MouseEvent):void {
        // create the popup
        var popup:CustomPopUp =
            CustomPopUp(PopUpManager.createPopUp(this,
                CustomPopUp, false));
        popup.message = "This is the detail for " +
            evt.target.label;

        // position the popup
        var pt:Point = new Point(0, 0);
        pt = evt.target.localToGlobal(pt);
        popup.x = pt.x + POPUP_OFFSET;
        popup.y = pt.y + evt.target.height + POPUP_OFFSET;
    }
]]>
</mx:Script>
</mx:Application>

```

在这个例子中，一系列的 LinkButton 控件被创建并被绝对定位放置在 Canvas 中。当用户点击了一个 LinkButton，一个弹出窗口会 显示用来为用户显示详细信息。LinkButton 控件的 click 事件被连接到 showDetail 方法上。showDetail 方法使用了 PopUpManager 的 createPopUp

方法来创建了一个自定义组件 CustomPopUp 的实例。然后弹出窗口的 message 属性被赋值成 需要被显示给用户的值。最后，弹出窗口被相对定位到发起初始请求的 LinkButton。LinkButton 左上角的（LinkButton 的坐标 内，x=0,y=0），会被 localToGlobal 方法从组件内部坐标空间装换到全局坐标空间。这个转换方法在所有的显示元件和他们的子类都可用。CustomPopUp 类代码如下所示：

Code View:

```
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    width="300" height="50"
    styleName="customPopUp"
    showCloseButton="true"
    close="handleClose(event)">

    <mx:Style>
        .customPopUp {
            header-height:2;
            padding-left:5;
            padding-right:5;
            padding-top:5;
            padding-bottom:5;
            border-color:#000000;
            border-alpha:.5;
            border-thickness-left:5;
            border-thickness-right:5;
            border-thickness-bottom:5;
            border-thickness-top:5;
            background-color:#666666;
            color:#ffffff;
        }
    </mx:Style>

    <mx:Text width="100%" height="100%" text="{message}"/>

    <mx:Script>
        <![CDATA[
            import mx.managers.PopUpManager;
            import mx.events.CloseEvent;
            [Bindable]
            public var message:String;

            private function handleClose(evt:CloseEvent):void {
                PopUpManager.removePopUp(this);
        ]]>
    </mx:Script>

```

```
        }
    ]]>
</mx:Script>
</mx:TitleWindow>
```

CustomPopUp 类继承于 TitleWindow 类，并且添加了一个 Text 控件用来显示信息。当通过 PopUpManager 的 removePopUp 方法来关闭弹出窗口时，TitleWindow 的 close 事件被指向到 handleClose 方法。上面的代码还包含了自定义 CustomPopUp 显示的 CSS 风格。

## 2.13 节. 自定义弹出式窗口边框

### 2.13.1. 问题

我想要自定义弹出窗口的边框来显示窗口。

### 2.13.2. 解决办法

创建一个 PanleSkin 类的子类，覆盖 updateDisplayList 方法来绘画调出箭头。设定这个子类为弹出窗口的 borderSkin 风格。

### 2.13.3. 讨论

这个技术是基于 Section 2.12CustomPopUp 组件修改的。定制你的窗口边框，这次设一个自定义类 CustomPanelSkin 到 borderSkin 风格。

Code View:

```
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    width="300" height="50"
    styleName="customPopUp"
    showCloseButton="true"
    close="handleClose(event)"
    borderSkin="CustomPanelSkin"
    initialize="initPopUp()">>

<mx:Style>
    .customPopUp {
        header-height:2;
        padding-left:5;
        padding-right:5;
        padding-top:5;
```

```

padding-bottom:5;
border-color:#000000;
border-alpha:.5;
border-thickness-left:5;
border-thickness-right:5;
border-thickness-bottom:5;
border-thickness-top:5;
background-color:#666666;
color:#ffffff;
}
</mx:Style>

<mx:Text width="100%" height="100%" text="{message}" />

<mx:Script>
<![CDATA[
import mx.managers.PopUpManager;
import mx.events.CloseEvent;
[Bindable]
public var message:String;

private function handleClose(evt:CloseEvent):void {
    PopUpManager.removePopUp(this);
}

private function initPopUp():void {
    this.isPopUp = false;
}
]]>
</mx:Script>
</mx:TitleWindow>
```

如下是 CustomPanelSkin 类的代码。注意 TitleWindow 的 isPop 属性需要设为 false，用来阻止用户拖动弹出窗口。

Code View:

```

package
{
    import flash.display.Graphics;
    import mx.skins.halo.PanelSkin;

    public class CustomPanelSkin extends PanelSkin
    {
        override protected function updateDisplayList(w:Number,
```

```

    h:Number) :void {
        super.updateDisplayList(w,h);

        var gfx:Graphics = this.graphics;
        gfx.beginFill(this.getStyle("borderColor"),
                      this.getStyle("borderAlpha"));
        gfx.moveTo(this.getStyle("cornerRadius"),0);
        gfx.lineTo(15,-10);
        gfx.lineTo(25,0);
    }
}

}

```

这个简单的类集成了 TitleWindow 的默认边框皮肤类 PanelSkin。updateDisplayList 方法被覆盖，增加了在 CustomPopUp 组件左上角绘制调出箭头的逻辑。

## 2.14 节. 处理 focusIn 和 focusOut 事件

### 2.14.1. 问题

我想要在用户聚焦在一个标签上时，显示一个弹出窗口，并且当用户离开聚焦的时候关闭这个弹出窗口。

### 2.14.2. 解决办法

使用 focusIn 和 focusOut 事件（在 InteractiveObject 类的所有实例都可用）来调用 PopUpManager 相关的方法。

### 2.14.3. 讨论

在用户聚焦时启动一个窗口，你可以重用前两节的代码。不过就是把弹出窗口的启动从用户点击 LInkButton 切换到通过 focusIn 事件来创建而已。组件接受到聚焦的时候就会发布 focusIn 事件，例如当用户按 tab 切换到组件上或者点击了它。focusIn 事件的处理代码只需要在之前章节的内容添加一点：

```
systemManager.removeFocusManager(IFocusManagerContainer(popup))
```

对应的上下文：

Code View:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute">

    <mx:Canvas horizontalCenter="0" verticalCenter="0">
        <mx:LinkButton id="lbl" label="Top" x="100" y="10"
            focusIn="showDetail(event)" focusOut="closePopUp()"/>
        <mx:LinkButton label="Left" x="10" y="100"
            focusIn="showDetail(event)" focusOut="closePopUp()"/>
        <mx:LinkButton label="Bottom" x="100" y="200"
            focusIn="showDetail(event)" focusOut="closePopUp()"/>
        <mx:LinkButton label="Right" x="200" y="100"
            focusIn="showDetail(event)" focusOut="closePopUp()"/>
        <mx:Canvas width="100" height="100" x="125" y="40"
            backgroundColor="#ff0000" rotation="45">
        </mx:Canvas>
    </mx:Canvas>

    <mx:Script>
        <![CDATA[
            import mx.managers.IFocusManagerContainer;
            import mx.managers.PopUpManager;

            private const POPUP_OFFSET:int = 10;

            private var popup:CustomPopUp;

            private function showDetail(evt:FocusEvent):void {
                // create the popup
                popup =
                    CustomPopUp(PopUpManager.createPopUp(this,CustomPopUp,false));
                popup.message = "This is the detail for " +
                    evt.target.label;

                // position the popup
                var pt:Point = new Point(0, 0);
                pt = evt.target.localToGlobal(pt);
                popup.x = pt.x + POPUP_OFFSET;
                popup.y = pt.y + evt.target.height + POPUP_OFFSET;

                systemManager.removeFocusManager(IFocusManagerContainer(popup))
            }
        ]]>
    </mx:Script>

```

```
}

private function closePopUp():void {
    PopUpManager.removePopUp(popup);
}

] ]>
</mx:Script>
</mx:Application>
```

当任意的弹出窗口被创建， 默认情况， SystemManager 会被弹出窗口联结的 FocusManager 激活。它允许基于刚创建的弹出窗口实现 的聚焦循环（控制 tab 顺序）。在这节，我们期望一个不同的行为。当用户聚焦离开的时候（例如用户 tab 出组件）， 弹出窗口因该关闭。这就是从 SystemManager 上移除了弹出窗口的 FocusManager 所获得的。因此重新启动应用程序的 FocusManager. FocusOut 事件的处理函数 closePopUp 包含了关闭弹出窗口的逻辑。当这个应用程序运行中， 重复按下 tab 键的时候， 聚焦会循环的在 LinkButton 上产 生，并且相关的弹出窗口会被创建或者移除。

## 第三章 容器 (Flexer::Nigel)

容器几乎涉及到了 Flex 框架 mx. containers 包内的所有类。容器继承自 UIComponent 类，添加了布局管理功能，使用一定的创建方针来控制或管理子组件的创建，和自动滚动功能。容器的不同实现具有不同的特性，但是所有的容器都具有决定子组件位置、使用约束条件或风格来布置子组件和控制滚动以及其子组件如何响应滚动事件的功能。

约束是 Flex3 的新特性。它让开发者能够创建针对容器的、可配置的、子组件的布局规则，包括位置和尺寸。约束条件仅对容器起作用，比如 Canvas 容器，拥有的绝对布局约束，在 CSS 里面有同样含义。Box 和 Tile 则对布局管理器所包含的子组件提供自动布局和控制方法。

### 3.1 节 使用布局管理器布置子组件

#### 3.1.1 问题

我们需要从横向或纵向来布置多种不同类型的子组件并且控制它们的布局。

#### 3.1.2 解决办法

使用 HBox 或者 VBox 容器，并且分别为 HBox 或者 VBox 设置 horizontalGap 或者 verticalGap 风格属性，达到设置控件之间间距的目的。

#### 3.1.3 讨论

通过扩展普通 mx. containers. Box 基类，HBox 和 VBox 组件分别从横向或者纵向布置它们的子组件，并且它们可以拥有的子组件个数没有限制。当子组件的尺寸大于 Box 控件的宽或者高的时候，默认情况下，Box 会自动添加滚动条。Vbox 容器使用 verticalGap 属性而 Hbox 则使用 horizontalGap 来分别设置各自的子组件间距。例如：

```
<mx:VBox width="400" height="300" verticalGap="20">
    <mx:Button label="Button"/>
    <mx:LinkButton label="Link Button"/>
</mx:VBox>
```

然而，HBox 和 VBox 容器却不遵守 bottom、left、right 或 top 这几个约束条件。我们可以像下面的例子这样，在 Box 内的子组件之间加上 Spacer 控件：

```
<mx:VBox width="400" height="300" verticalGap="20">
    <mx:Button label="Button"/>
    <mx:ComboBox top="60"/>
    <mx:Spacer height="20"/>
    <mx:LinkButton label="Link Button"/>
```

```
</mx:VBox>
```

可以通过添加 paddingTop、paddingLeft、paddingRight 或 paddingBottom 风格属性来改变补白风格属性或者说改变边框与子组件、子组件与子组件之间的距离。这对添加到容器中的所有子组件奏效。如果要把某一单个子组件在 VBox 内移动到左边或右边，或者在 HBox 内移动到上边或下边，则需要添加一个内部容器来使用这些属性布置该子组件：

```
<mx:HBox x="400" horizontalGap="10" top="15">
    <mx:Canvas>
        <mx:Button top="50" label="Button" y="20"/>
    </mx:Canvas>
    <mx:Panel height="40" width="40"/>
    <mx:Spacer width="25"/>
    <mx:LinkButton label="Label"/>
    <mx:ComboBox/>
</mx:HBox>
```

下面的这个例子在 Canvas 中同时使用了 HBox 和 VBox 来演示多种布局方式：

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" >
    <mx:VBox width="400" height="300" verticalGap="20">
        <mx:Button label="Button"/>
        <mx:ComboBox/>
        <mx:Spacer height="20"/>
        <mx:LinkButton label="Link Button"/>
    </mx:VBox>
    <mx:HBox x="400" horizontalGap="10" top="15">
        <mx:Canvas>
            <mx:Button top="50" label="Button" y="20"/>
        </mx:Canvas>
        <mx:Panel height="40" width="40"/>
        <mx:Spacer width="25"/>
        <mx:LinkButton label="Label"/>
        <mx:ComboBox/>
    </mx:HBox>
</mx:Canvas>
```

## 3. 2 节 通过百分比方式配置容器的布局和尺寸

### 3.2.1 问题

需要根据父亲控件的尺寸来设置子组件们的尺寸。

### 3.2.2 解决办法

使用百分比设置尺寸的时候，当控件的尺寸变化时，Flex 框架会自动调整它的子组件的尺寸。

### 3.2.3 讨论

百分比布局方式是一种强大工具，可以让你轻松地定义一个子组件的尺寸和位置，同时兼顾其父亲容器。例如，下面的 RelativePositioningChild.mxml 组件就将自己的宽度设置为其父组件宽度的 40%，高度为其父组件高度的 70%。

代码：

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="40%"  
height="70%" backgroundColor="#0033ff">  
    <mx:Image source="@Embed('../assets/image.png')"/>  
    <mx:Image source="@Embed('../assets/image.png')"/>  
</mx:VBox>
```

在下面的这个例子当中，多个 RelativePositioningChild 的实例被放进一个同样是百分比尺寸的父亲容器中。如果我们把这个父亲容器以及它所有的子组件放在一起看作一个组合起来的控件的话。那么，无论这个组合控件被添加到什么父亲里面去，这个组合控件的宽和高都由它的父亲来决定，从而也决定了组合控件内子组件的宽和高。

代码：

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="75%"  
height="50%" backgroundColor="#0099ff" alpha="0.3"  
xmlns:cookbook="oreilly.cookbook.*">  
    <cookbook:RelativePositioningChild/>  
    <cookbook:RelativePositioningChild/>  
</mx:HBox>
```

要示范百分比尺寸方式下大小的调整，将前面的代码段保存为 RelativePositioningParent.mxml 文件。然后像下面这样使用之：

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"  
layout="absolute" xmlns:cookbook="oreilly.cookbook.*">  
    <mx:Script>  
        <! [CDATA [  
  
            private function changeWidth():void  
            {  
                this.width = slider.value*150;  
            }  
  
        ]]>  
    </mx:Script>  
    <cookbook:RelativePositioningParent/>  
    <mx:HSlider id="slider" change="changeWidth()"/>
```

```
</mx:Application>
```

当滑块控件改变应用的宽度时, `RelativePositioningParent` 和 `RelativePositioningChild` 即会根据父亲应用的宽度来调整它们自己的宽度。

## 3.3节. 以不同的坐标系统跟踪鼠标位置

### 3.3.1. 问题

我想跟踪用户鼠标位置, 可以是相对于父容器或相对于容器中其他子组件。

### 3.3.2. 解决办法

使用 `Stage` 和 `MouseEvent` 类的本地位置属性以及所有容器都继承自 `UIComponent` 的 `mouseX` 和 `mouseY` 属性。

### 3.3.3. 讨论

`MouseEvent` 类有四个属性可用于确定鼠标位置。`localX` 和 `localY` 属性提供与抛出 `mouse` 事件相关的组件位置, 而 `stageX` 和 `stageY` 提供与 `Stage` 相关的位置。

下面的例子, 如果鼠标移动到 `LinkButton` 组件上, `localX` 和 `localY` 属性将反应出鼠标在 `LinkButton` 组件上的位置。如果鼠标没有在 `LinkButton` 上, 这两个属性就是指鼠标在 `VBox` 上的位置:

Code View:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml">
    mouseMove="traceMousePosition(event)">
        <mx:LinkButton label="MyButton"/>
    </mx:VBox>
```

要检测某个组件上的鼠标位置(无论鼠标是否在其之上), 使用 `DisplayObject` 的 `mouseX` 和 `mouseY` 位置, 它将返回相对于容器或组件的0, 0坐标的位置。最后, `Container` 类定义了 `contentMouseX` 和 `contentMouseY` 位置, 描述鼠标相对于整个容器内容的位置。下面的例子返回与两个被添加到 `HBox` 的 `Panel` 组件相关的鼠标位置, 而不是与 `HBox` 左上角相关:

Code View:

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="500">
```

```

mouseMove="trace(this.contentMouseX+ ' : '+this.contentMouseY);
height="300"">
<mx:Panel width="400">
    <mx:Label text="Center" horizontalCenter="200"/>
</mx:Panel>
<mx:Panel width="400">
    <mx:Label text="Center" horizontalCenter="200"/>
</mx:Panel>
</mx:HBox>

```

因为 HBox 两个子组件的和为812像素，当你滚动鼠标到 HBox 右边时，你会发现 x 值已经超出 HBox 的设置值。容器定义了异恶 contentPane 属性，它是容纳添加进来的所有子组件的私有的 DisplayObject。如果子组件的高或宽超出容器自身设置的高或宽，那超出部分将被隐藏。而 contentMouseX 和 contentMouseY 属性就是测量鼠标在保护内容 DisplayObject 上的位置。

下面是完整代码：

Code View:

```

<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="300"
height="500" paddingTop="10" paddingLeft="10" verticalGap="15"
mouseMove="traceMousePosition(event)">
<mx:Script>
    <![CDATA[

        private function
        traceMousePosition(event:MouseEvent):void
        {
            trace(" MouseEvent local position "+event.localX+
                "+event.localY);
            trace(" MousePosition stage position
                "+event.stageX+" "+event.stageY);
            trace(" MouseEvent position from w/in component
                "+this.mouseX+" "+this.mouseY);
            trace(" Content Mouse Position
                "+this.contentMouseX+" "+this.contentMouseY);
        }
    ]]>
</mx:Script>
<mx:Image source="@Embed('.../..../assets/image.png')"/>
<mx:Image source="@Embed('.../..../assets/image.png')"/>
<mx:Image source="@Embed('.../..../assets/image.png')"/>
<mx:Image source="@Embed('.../..../assets/image.png')"/>

```

```
</mx:VBox>
```

当鼠标移到 VBox 组件时函数将被调用，输出 mouseX 和 mouseY 值。

## 3.4 节 在容器中动态添加和移除子组件

### 3.4.1 问题

需要程序在运行时从容器添加和移除子组件而不使用 Repeater 或者 DataProvide 控件。

### 3.4.2 解决办法

使用 `addChild` 或者 `addChildAt` 方法来添加子组件，同理，使用 `removeChildAt` 或者 `removeAllChildren` 方法来移除子组件。

### 3.4.3 讨论

Flex 方法使得添加和移除子组件操作变得容易，但是 UIComponent 控件和容器则遵从稍显不同的两组规则。

`addChild` 方法将任何继承自 `UIComponent` 的子组件添加到调用这个方法的控件中。例如：

```
var component:UIComponent = new UIComponent();
addChild(component);
```

`addChildAt` 方法的不同在于，它在子组件被添加到容器的时候需要传入一个该子组件在目标容器内的索引。在容器没有布局管理器的情况下，比如 `Canvas` 对象，这意味着被添加的子组件会在指定的 z 索引深度上显示。而对于拥有布局管理器的容器来说，例如 `HBox` 和 `VBox` 组件，被添加的子组件则会出现在提供的索引上。例如：

```
var component:UIComponent = new UIComponent();
addChildAt(component, 3);
```

要移除任意子组件，调用 `removeChildAt`，即会移除指定索引上的子组件。

```
removeChildAt(2);
```

容器和 `UIComponent` 控件也有 `removeChild` 方法，该方法要求传入一个需要移除的子组件的引用。

Flex 提供了不同的方法来访问被添加到容器中的子组件。在获悉如何访问子组件之后，你可以使用访问子组件得到的引用来移除它们。例如，任何添加到 MXML 容器的子组件都可以

通过它们的 id 属性来访问。任何容器的所有子组件都可以使用 getChildAt 来访问它们，如下：

```
getChildAt(index:int);
```

如果赋以了 name 属性，也可以通过 name 属性来访问它们。要确定任何组件所包含的子组件个数，使用 numChildren 属性：

```
var i:int = 0;  
while(i<this.numChildren)  
{  
    trace(getChildAt(i));  
    i++;  
}
```

移除子组件的时候，可以使用子组件的 index 或者 id 作为引用，像后面这样的句式，  
removeChild(getChildAt(2));或者 removeChild(this.childId);

最后，removeAllChildren 方法则可以移除添加到某一组件内的所有子组件。

## 3.5 节 对容器使用基于约束的布局

### 3.5.1 问题

需要在父组件的内边界基础上来定义子组件的尺寸。

### 3.5.2 解决办法

使用约束性属性：left, right, top, and bottom.

### 3.5.3 讨论

UIComponent 类的约束性属性允许为组件定义制约于父组件装订线的宽度和高度。一个宽度为 200 像素的组件，如果它的子组件的 left 和 right 属性均为 20 像素，则该子组件的宽度将为 160 像素。同样，高度为 200 像素的组件的子组件如果 bottom 属性为 40 像素，则会被局限在 160 像素内。约束不会被应用到没有绝对布局的组件中去，例如 Hbox 或 Vbox 组件。子组件能够使用约束型属性的容器有 Canvas、Panel 和 Application.

在下面的例子里，内部 Canvas 组件将距离外部 Canvas 的顶部、左边和右边 20 像素，底部延伸 50 像素处放置，这样就使得内部 Canvas 变成 360 像素宽，230 像素高。如果外部 Canvas 扩大或收缩，内部 Canvas 仍然保持在所有方向上相同的装订线距离。组件内的 Button 和 TextInput 控件将根据内部 Canvas 的尺寸，同时保持自身约束性属性制定的装订线距离。  
代码如下：

```

<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
<mx:Canvas left="20" right="20" top="20" bottom="50"
id="innerCanvas">
<mx:Button label="Button Label" left="20" right="20"
top="10"/>
<mx:TextInput left="30" right="30" bottom="10"/>
</mx:Canvas>
</mx:Canvas>

```

在下面的代码段中，由于应用到面板的约束性属性的原因，两个 Panel 组件的宽度都将是 320 像素。顶部的 Panel 将根据 top 制约置于 40 像素处，同样底部的 Panel 将根据 bottom 制约置于 260 像素处。

代码如下：

```

<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
<mx:Panel left="40" right="40" top="40" id="topPanel">
<mx:Label text="Label text"/>
</mx:Panel>
<mx:Panel left="40" right="40" bottom="40" id="bottomPanel">
<mx:Label text="Label text"/>
</mx:Panel>
</mx:Canvas>

```

## 3.6 节 在容器内为子组件设置最大、最小尺寸

### 3.6.1 问题

你需要添加多个子组件到某个组件里，并且保证如果子组件的数量扩大超过一定数量的时候，子组件将添加到容器下一行去。

### 3.6.2 解决办法

使用 maxWidth 或者 maxHeight 属性来决定组件内的子组件放置在何处。

### 3.6.3 讨论

maxWidth 和 maxHeight 样式属性定义了组件父亲允许其显示的最大高度和宽度。下面的代码段里，将检查 maxWidth 样式属性以保证添加的组件不会引起容器超过其父亲容器所允许

的最大宽度。如果即将超过最大宽度，则会生成另一个 HBox 组件，并且接下来添加的图片会被添加到新生成的 HBox 里。VBox 容器的布局管理器会负责适当的布置新添加的子组件。

代码如下：

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" height="400"
    xmlns:cookbook="oreilly.cookbook.*" backgroundColor="#0000ff">
    <cookbook:AddConstraintChildren maxHeight="400" maxWidth="800"
        horizontalAlign="center" verticalScrollPolicy="off"/>
</mx:Canvas>

<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <! [CDATA [
            import mx.controls.Image;

            [Embed(source="..../assets/image.png")]
            public var image:Class;
            private var childCount:int = 0;
            private var currentHolder:HBox;

            private function addMoreChildren():void
            {
                var image:Image = new Image();
                image.source = image;
                if ((currentHolder.width + image.width) >=
                    this.maxWidth)
                {
                    var holder:HBox = new HBox();
                    addChild(holder);
                    currentHolder = holder;
                }
                currentHolder.addChild(image);
            }
        ]]>
    </mx:Script>
    <mx:Button label="addMoreChildren()"
        click="addMoreChildren()"/>
    <mx:HBox id="topHolder" creationComplete="currentHolder =
        topHolder"/>
</mx:VBox>
```

### 3.6.4 小节 另参照 [3.5. 节](#)

## 3.7 节 为容器指定行和列的约束

### 3.7.1 问题

你想使用独特的约束性属性成行或者成列地定义独特的子组件，而不是对每个子组件进行定义。

### 3.7.2 解决办法

在可添加约束条件的地方使用 ConstraintRow 和 ConstraintColumn 属性来定义容器领域。

### 3.7.3 讨论

ConstraintRow 和 ConstraintColumn 对象让我们可以定义一族用来布置组件的约束条件。我们使用这些约束条件与使用容器边缘来定义约束条件的方式异曲同工。例如，下面例子的这个句法：left="columnName:10"就将组件布置于距离列的左边 10 像素处。

```
<mx:Canvas>
    <mx:constraintColumns>
        <mx:ConstraintColumn id="leftCol" width="200"/>
        <mx:ConstraintColumn id="rightCol" width="60%"/>
    </mx:constraintColumns>
    <mx:constraintRows>
        <mx:ConstraintRow id="topRow" height="80%"/>
        <mx:ConstraintRow id="bottomRow" height="20%"/>
    </mx:constraintRows>
<mx:Button label="Click Me" left="leftCol:0" right="leftCol:0"
    top="row2:0" bottom="row2:0"/>
</mx:Canvas>
```

另外，我们可以添加滑动控件让用扩大或缩小分配给每个列的空间大小，从而展示了布局如何随着容器尺寸的变化而变化的，例如：

Code View:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

<mx:Script>
    <![CDATA[
        [Bindable]
        public var txt:String = "Cortázar is highly regarded as a
            master of short stories of a fantastic bent, with the
            collections Bestiario (1951) +
            " and Final de Juego (1956) containing many of his best
            examples in the genre, including the remarkable
            \"Continuidad de los Parques\" +
    ]]>
```

```

    " and \"Axolotl.\" These collections received early
    praise from \"Alvaro Cepeda Samudio, and selections from
    the two volumes were published\" in 1967 in English
    translations by Paul Blackburn, under the title
End of the Game and Other Stories (in later editions, Blow-Up and
" +
    "Other Stories, in deference to the English title of
Antonioni's
celebrated film of 1966 of Cortázar's story Las babas del
diablo) .";
}

private function changeColumnProportion():void
{
    this.leftCol.percentWidth = (10*c_slider.value);
    this.rightCol.percentWidth = 100 -
        (10*c_slider.value);
}

private function changeRowProportion():void
{
    this.row1.percentHeight = (10*r_slider.value);
    this.row2.percentHeight = 100 -
        (10*r_slider.value);
}

] ]>
</mx:Script>

<mx:Canvas width="100%" height="100%"
    horizontalScrollPolicy="off" verticalScrollPolicy="off">
    <mx:constraintColumns>
        <mx:ConstraintColumn id="leftCol" width="50%" />
        <mx:ConstraintColumn id="rightCol" width="50%" />
    </mx:constraintColumns>
    <mx:constraintRows>
        <mx:ConstraintRow id="row1" height="20%" />
        <mx:ConstraintRow id="row2" height="80%" />
    </mx:constraintRows>

    <mx:HSlider id="c_slider" change="changeColumnProportion()"
        value="5"/>
    <mx:HSlider id="r_slider" x="200"
        change="changeRowProportion()" value="5"/>

```

```

<mx:Text text="{txt}" left="leftCol:0" right="leftCol:0"
    top="row2:0" bottom="row2:0"/>

<mx:Text text="{txt}" left="rightCol:0" right="rightCol:0"
    top="row2:0" bottom="row2:0"/>

</mx:Canvas>
</mx:Application>

```

## 3.8 节 使用约束条件为文本创建排版流程 (Layout Flows)

### 3.8.1 问题

你要为多段文字创建一个排版流程 (layout flow)。

### 3.8.2 解决办法

创建并添加一个 `ConstraintColumn` 和 `ConstraintRow` 对象到 `Canvas` 组件，然后使用它们为子组件设置约束条件。

### 3.8.3 讨论

所有支持约束条件的容器都具有两个数组来存储加到该 `Canvas` 的行和列的轨迹。只需为各自数组添加现成的约束条件即可保证所有的子组件可以存取约束条件到它的属性。本节的例子就是用约束条件来控制排版流程 (layout flow)。首先，创建多个 `ConstraintColumn` 和 `ConstraintRow` 对象，然后将这些对象添加到 `Canvas` 的 `constraintRows` 和 `constraintColumns` 属性。使用 `setStyle` 方法动态的将约束条件设置到现成的 `UIComponent` 对象来约束之。这个例子使用 `text.setStyle("left", constraintColumn.id+":10")`；从 `ConstraintColumn` 对象的数组中动态地存取约束条件，或者使用下面这样稍微复杂的句式以保证访问到正确的列。如：

```

child.setStyle("left", (constraintColumns[i-
    (constraintColumns.length/2)-2] as ConstraintColumn).id+":10");

```

在接下来的清单里，生成 `ConstraintColumn` 对象并且基于这些对象来设置新子组件的样式属性。当添加新的行时，相关的子组件会使用新的行重新设置样式来改变自己的位置，并且所有未使用过的 `ConstraintColumn` 对象会从容器中移除，`constraintColumns` 即是这些数组的缩写。

```

<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="1000"
    height="800">

```

```

<mx:Script>
<! [CDATA[
    import mx.core.UIComponent;
    import mx.controls.TextArea;
    import mx.containers.utilityClasses.ConstraintColumn;

    [Bindable]
    public var txt:String = "Cortázar is highly regarded
        as a master of short stories of a fantastic bent,
        with the collections Bestiario (1951) +
        " and Final de Juego (1956) containing many of his
        best examples in the genre, including the remarkable
        \"Continuidad de los Parques\" +
        " and \"Axolotl.\"";

    private function addText(event:Event):void
    {
        var text:TextArea = new TextArea();
        addChild(text);
        text.text = txt;
        var constraintColumn:ConstraintColumn =
            new ConstraintColumn();
        constraintColumn.id =
            "column"+numChildren.toString();
        constraintColumns.push(constraintColumn);
        if(constraintColumns.length > 1)
        {
            for each(var col:ConstraintColumn in
                constraintColumns){
                col.width = (width / (numChildren-2));
            }
        }
        constraintColumn.width = (width / (numChildren-2));
        text.setStyle("top", "row:30");
        text.setStyle("bottom", "row:30");
        text.setStyle("left", constraintColumn.id+":10");
        text.setStyle("right", constraintColumn.id+":10");
    }

    private function addRow(event:Event):void
    {
        var constraintRow:ConstraintRow =
            new ConstraintRow();
        constraintRows.push(constraintRow);
    }
]
```

```

constraintRow.id = "row"+constraintRows.length;
for each(var row:ConstraintRow in constraintRows) {
    row.height = (height / (constraintRows.length-1));
}
var i:int = Math.round(numChildren - (numChildren-2)/constraintRows.length);
while(i < numChildren) {
    var child:UIComponent = getChildAt(i) as UIComponent;
    child.setStyle("top",
        "row"+constraintRows.length+":30");
    child.setStyle("bottom",
        "row"+constraintRows.length+":30");
    child.setStyle("left", (constraintColumns[i-(constraintColumns.length/2)-2] as ConstraintColumn).id+":10");
    child.setStyle("right", (constraintColumns[i-(constraintColumns.length/2)-2] as ConstraintColumn).id+":10");
    i++;
}
constraintColumns.length = constraintColumns.length / constraintRows.length;
}

]]>
</mx:Script>
<mx:constraintRows>
    <mx:ConstraintRow id="row" height="100%"/>
</mx:constraintRows>
<mx:Button click="addText(event)" label="add text"/>
<mx:Button click="addRow(event)" label="add row" x="150"/>
</mx:Canvas>

```

## 3.9 节 在容器内控制滚动和溢出

### 3.9.1 问题

你需要禁用容器的垂直滚动条并且在某个组件上创建一个用户可以通过鼠标滑过来控制滚动的区域。

### 3.9.2 解决办法

使用 horizontalScrollPolicy、verticalScrollPolicy 和 verticalScrollPosition 属性。

### 3.9.3 讨论

我们可以通过 horizontalScrollPolicy 和 verticalScrollPolicy 属性控制滚动条。如果要使其中一个或全部滚动条始终显示，则将其中一个或全部设置成“on”，反之亦然。而“auto”值则导致滚动条仅在容器的实际测量尺寸大于 width 或 height 指定的值时出现。例如，将 horizontalScrollPolicy 设置为“auto”则意味着当容器的宽度超过 width 属性的值时出现滚动条。

要滚动一个组件，使用 horizontalScrollPosition 和 verticalScrollPosition 属性。通过这两个属性，我们可以设定组件内容的可视部分往右下角滚动多远的距离。例如：

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="600"
height="200" horizontalScrollPolicy="auto"
verticalScrollPolicy="off" mouseMove="autoScroll(event)">
<mx:Script>
<! [CDATA[

    private var hasAddedScroll:Boolean = false;

    private function autoScroll(event:MouseEvent):void
    {
        if(mouseX > width - 50 && !hasAddedScroll)
        {
            addEventListener(Event.ENTER_FRAME, scrollRight);
            hasAddedScroll = true;
        }
        else if(mouseX < 50 && !hasAddedScroll)
        {
            addEventListener(Event.ENTER_FRAME, scrollLeft);
            hasAddedScroll = true;
        }
        else
        {
            removeEventListener(Event.ENTER_FRAME,
                scrollRight);
            removeEventListener(Event.ENTER_FRAME,
                scrollLeft);
            hasAddedScroll = false;
        }
    }

    private function scrollRight(event:Event):void
    {
```

```

        if(horizontalScrollPosition <
            maxHorizontalScrollPosition)
        {
            horizontalScrollPosition+=4;
        }
        else
        {
            removeEventListener(Event.ENTER_FRAME,
                scrollRight);
            hasAddedScroll = false;
        }
    }

private function scrollLeft(event:Event):void
{
    if(horizontalScrollPosition > 0)
    {
        horizontalScrollPosition-=4;
    }
    else
    {
        removeEventListener(Event.ENTER_FRAME,
            scrollLeft);
        hasAddedScroll = false;
    }
}

]]>
</mx:Script>
<mx:Image source="@Embed('assets/image.png')"/>
<mx:Image source="@Embed('assets/image.png')"/>
<mx:Image source="@Embed('assets/image.png')"/>
<mx:Image source="@Embed('assets/image.png')"/>
<mx:Image source="@Embed('assets/image.png')"/>
</mx:HBox>

```

## 3. 10 节 控制 Box 组件的布局

### 3. 10. 1 问题

你既要控制 Box 组件的横向和纵向布局，也要控制组件之间的横向和纵向间距及其子组件的居中设置。

### 3.10.2 解决办法

使用 verticalAlign 和 horizontalAlign 属性同时使用 direction 属性设置 Box 的布局方向。

### 3.10.3 讨论

mx.containers.Box 类定义了几个属性，来控制 Box 内的子组件的布局。它们是：

#### direction

决定容器如何布置其子组件。值可为 vertical 或 horizontal。

#### horizontalAlign

决定子组件的横向对齐方式，可以设置为 left、right 或者 center

#### horizontalGap

在 direction 属性为 horizontal 的时候，决定了子组件之间的间距。如果 direction 为非 horizontal 的值，则 horizontalGap 属性可以忽略。

#### verticalAlign

决定子组件的纵向对齐方式，可以设置为 top、bottom 或者 center

#### verticalGap

在 direction 属性为 vertical 的时候，决定了子组件之间的间距。如果 direction 为非 vertical 的值，则 verticalGap 属性可以忽略。

下面的例子，作为示范，将所有的属性绑定到控件上实现动态变化。

```
<mx:HSlider change="{vbox.verticalGap = vSlider.value*10}"  
    id="vSlider"/>  
<mx:VBox y="100" direction="_direction" id="vbox">  
    <mx:Panel width="90" height="60">  
        <mx:Label text="Some Text" />  
    </mx:Panel>  
    <mx:Panel width="90" height="60">  
        <mx:Label text="Some Text" />  
    </mx:Panel>  
    <mx:Panel width="90" height="60">  
        <mx:Label text="Some Text" />  
    </mx:Panel>  
    <mx:Panel width="90" height="60">
```

```
<mx:Label text="Some Text" />
</mx:Panel>
<mx:Panel width="90" height="60">
    <mx:Label text="Some Text" />
</mx:Panel>
</mx:Box>
```

因为 verticalGap 是可绑定属性，我们可以在运行时调用 `setStyle()` 方法动态地设置它。

## 3.11 节 使用容器初始化

### 3.11.1 问题

为了提高应用的响应能力，我们需要保证容器的所有子组件在应用初始化的时候即被创建。

### 3.11.2 解决办法

使用容器的类的 `creationPolicy` 属性来决定什么时候创建组件。

### 3.11.3 讨论

所有容器，事实上所有 `UIComponent` 组件都使用一种三步走的过程来创建本身，即创建成员属性、创建子组件和设置自己的布局。所有容器的第一步都是由框架调用它的构造方法并且广播一个预初始化事件。第二步，假设容器与其所有的子孙的关系构成一棵树形结构，那么，预初始化的工作是由上而下的，也就是说，组件优先预初始化父亲容器，按照深度顺序首先预初始化根节点位置上的容器，直到叶子节点的子组件预初始化完毕。第三步，类似于初始化的过程，真正初始化完成的顺序却是和预初始化结束的顺序相反，如此，先出初始化完毕叶子节点位置的子组件，然后此过程往上行节点重复，当某个节点位置的子组件初始化完毕时即广播一个 `creationComplete` 事件，同理，随后上一级子组件也会广播一次这个事件，以此类推，所有的上行的父亲容器直到根节点位置上的容器都会在自身初始化完毕的时候广播该事件。下面的代码简单地展示初始化的顺序：

```
<mx:HBox>
    <mx:VBox>
        <mx:Panel/>
        <mx:Panel/>
    </mx:VBox>
</mx:HBox>
```

初始化将按照下面的顺序：

```
HBox preinitialize
    VBox preinitialize
        FirstPanel preinitialize
        SecondPanel preinitialize
        FirstPanel initialize
        SecondPanel initialize
    VBox initialize
HBox initialize
    FirstPanel creationComplete
    SecondPanel creationComplete
VBox creationComplete
HBox creationComplete
```

当组件广播 preinitialize 和 initialize 事件时，它的子组件还没有创建完成。因此，要访问某个组件的所有子组件，必须侦听 creationComplete 时间。在广播 initialize 事件后，组件本身已经完成测量、绘制和布局，但是其子组件可能仍未完全完成实例化过程。最后，creationComplete 事件表明所有容器内的子组件都完全实例化结束。

## 3.12 节 创建 TitleWindow

### 3.12.1 问题

我们需要创建一个 TitleWindow 组件来显示对话框并在满足一定标准的时候使用 PopUpManager 移除该对话框。

### 3.12.2 解决办法

TitleWindow 组件，继承自 Panel，添加了可以为窗口设置标题的功能，同时也为边框提供样式信息。

### 3.12.3 讨论

PopUpManager.removePopUp(this);

此例中，使用 TitleWindow 为应用程序创建一个登陆界面。PopUpManager 类提供 PopUpManager.removePopUp 方法使组件可以在屏幕上移除其本身。

这说明 TitleWindow 组件是由 PopUpManager 添加的。

```
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml"
    borderColor="#0000ff" backgroundAlpha="0.6"
    title="Title Window" x="168" y="86">
```

```

<mx:Script>
<! [CDATA[
    import mx.managers.PopUpManager;
    import mx.controls.Text;

    // A reference to the TextInput control in which to put
    // the result.
    public var loginName:Text;
    public var loggedIn:Boolean;

    // Event handler for the OK button.
    private function returnName():void {
        loginName.text="Name entered: " + userName.text;
        PopUpManager.removePopUp(this);
    }

    private function checkUserNameAndPass():void
    {
        /* Do some processing */
    }

    /* have this handle the event when the server has
    logged in */
    private function returnValueFromLogin(event:Event):void
    {
        if(loggedIn)
        {
            PopUpManager.removePopUp(this);
        }
        else
        {
            successText.text = "User/Pass not recognized";
        }
    }
}

] ]>
</mx:Script>
<mx:HBox>
    <mx:Label text="Username" />
    <mx:TextInput id="userName" width="100%" />
</mx:HBox>
<mx:HBox>
    <mx:Label text="Password" />
    <mx:TextInput id="password" width="100%" />

```

```

</mx:HBox>
<mx:HBox>
    <mx:Button label="Enter" click="checkUserNameAndPass();"/>
    <mx:Button label="Cancel"
        click="PopUpManager.removePopUp(this);"/>
</mx:HBox>
<mx:Text id="successText" color="#ff0000"/>
</mx:TitleWindow>

```

## 3.13 节 通过 LinkBar 控制 ViewStack

### 3.13.1 问题

我们需要使用 LinkBar 组件控制 ViewStack

### 3.13.2 解决办法

使用 LinkBar 的 selectedIndex 或 selectedItem 属性的任意之一来决定显示 ViewStack 的哪一条。

### 3.13.3 讨论

LinkBar 既可以使用数组作为数据提供者，也可以使用一个拥有多个子组件的容器（例如 ViewStack）作为数据提供者。后者对此节最有用。当你传递一个容器的时候，该容器选中显示的项目将自动绑定到 LinkBar 控件选中的项目上。这意味着可以把拥有多个子组件的容器传递给 LinkBar 并且让该容器的子组件关联到 LinkBar 上。LinkBar 会为这些子组件自动添加相同个数的按钮以正确设定 ViewStack 的 selectedChild 属性。

Code View:

```

<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="800"
height="600">
    <mx:LinkBar dataProvider="{viewStack}" direction="horizontal"
        labelField="name"/>
    <mx:ViewStack id="viewStack" y="60">
        <mx:Panel width="150" height="150" name="first"
            label="First Panel" title="First Panel">
            <mx:Label text="First label"/>
        </mx:Panel>
        <mx:Panel width="150" height="150" name="second"
            label="Second Panel" title="Second Panel">
            <mx:Label text="Second label"/>
        </mx:Panel>
    </mx:ViewStack>
</mx:Canvas>

```

```

<mx:Panel width="150" height="150" name="third"
    label="Third Panel" title="Third Panel">
    <mx:Label text="Third label"/>
</mx:Panel>
</mx:ViewStack>
</mx:Canvas>

```

### 3.13.4 小节 另参照 [3.16.](#) 节

## 3.14 节 将 ViewStack 的选中索引数绑定到一个变量上

### 3.14.1 问题

我们需要将 ViewStack 的 selectedIndex 属性绑定到一个可以在组件的其他地方改变的整数变量。

### 3.14.2 解决办法

申明一个可绑定的变量，然后将 ViewStack 控件的选中索引属性绑定到它上。

### 3.14.3 讨论

在 LinkBar 控件的情形下，ViewStack 的选中项目会自动绑定到 LinkBar 的选中项目上。当使用其它控件时，ViewStack 或其他拥有多个同时显示的子组件的控件的选中索引或者项目，需要绑定到一个可绑定的变量或者对其设置事件。要使用其他方法来控制 ViewStack，将 ViewStack 的 selectedIndex 属性绑定到变量上，然后在程序运行时改变其值。

```

<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="300" click="changeViewStack()">
    <mx:Script>
        <! [CDATA [
            [Bindable]
            private var selectedIndexInt:int = 0;

            private function changeViewStack():void
            {
                if(selectedIndexInt == 2)
                {
                    selectedIndexInt = 0;
                }
            }
        ]>
    </mx:Script>

```

```

        else
        {
            selectedIndexInt++;
        }
    }

] ]>
</mx:Script>
<mx:ViewStack selectedIndex="{selectedIndexInt}">
    <mx:HBox height="{this.height}" width="{this.width}">
        <mx:Label text="First View Item"/>
        <mx:Label text="First View Item"/>
    </mx:HBox>
    <mx:VBox height="{this.height}" width="{this.width}">
        <mx:Label text="Second View Item"/>
        <mx:Label text="Second View Item"/>
    </mx:VBox>
    <mx:Canvas height="{this.height}" width="{this.width}">
        <mx:Label text="Third View Item"/>
        <mx:Label text="Third View Item" y="40"/>
    </mx:Canvas>
</mx:ViewStack>
</mx:Canvas>

```

### 3.14.4. 小节 另参照 [3.17.](#) 节

## 3.15 节 使用延迟实例化提高启动效率

### 3.15.1 问题

我们要保证组件仅仅在需要显示在屏幕上的时候才被创建。

### 3.15.2 解决办法

为容器类设置创建方针队列并且根据需要对每个子组件使用 creationIndex。

### 3.15.3 讨论

Container 类默认情况下仅仅在要显示组件的时候创建它，因为 UIComponent 默认的 creationPolicy 值为 auto。换句话说，当视图(view)设置成可视的时候，如果还没有创建该视图，则组件会实例化此视图。creationPolicy 的其他可能值为 none，意指，所有组件

将会创建、列队，这里的创建和列队指根据其 creationIndex 值进行创建。creationIndex 是以 0 起始的顺序数组，组件的子组件会根据这个顺序数组来创建。

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" height="600"
width="600">
<mx:Script>
<! [CDATA[

    private function changeViewStackCreation():void
    {
        viewStack.creationPolicy = (comboBox.selectedItem as
            String);
        viewStack.createComponentsFromDescriptors(true);
    }

    private function changeViewStack():void
    {
        viewStack.selectedIndex =
            comboBoxChangeIndex.selectedIndex;
    }
]]>
</mx:Script>
<mx:Fade alphaFrom="0" alphaTo="1" duration="4000"
id="fadeIn"/>
<mx:ComboBoxdataProvider="{'none', 'all', 'queued',
'auto'}" change="changeViewStackCreation()"
id="comboBox"/>
<mx:ComboBoxdataProvider="[{1, 2, 3, 4}]" x="150"
change="changeViewStack()" id="comboBoxChangeIndex"/>
<mx:ViewStack id="viewStack" width="400" height="300"
creationPolicy="none" y="100">
    <mx:Canvas creationCompleteEffect="{fadeIn}"
        creationIndex="0" backgroundColor="#0000ff"
        id="canvas1">
        <mx:LinkButton label="Link Button Number One"/>
    </mx:Canvas>
    <mx:Canvas creationCompleteEffect="{fadeIn}"
        creationIndex="1" backgroundColor="#0000ff"
        id="canvas2">
        <mx:LinkButton label="Link Button Number Two"/>
    </mx:Canvas>
    <mx:Canvas creationIndex="2" id="canvas3"
        creationCompleteEffect="{fadeIn}" backgroundColor
        ="#0000ff">
```

```

<mx:LinkButton label="Link Button Number Three"/>
</mx:Canvas>
<mx:Canvas creationIndex="3" id="canvas4"
creationCompleteEffect="{fadeIn}" backgroundColor
="#0000ff">
<mx:LinkButton label="Link Button Number Four"/>
</mx:Canvas>
</mx:ViewStack>
</mx:Canvas>

```

## 3.16 节 创建并控制可调整大小的容器

### 3.16.1 问题

我们需要创建一个可以通过拖拽角落图标而调整大小的容器。

### 3.16.2 解决办法

在拖拽图标上使用 MouseEvent 类侦听 mouseDown、mouseMove 和 mouseUp 事件。当拖拽图标释放的时候重新设置容器的尺寸。

### 3.16.3 讨论

通过在 MXML 和 ActionScript 里为这些事件添加侦听，我们可以在 Icon 对象上侦听到 mouseDown 事件。当捕获 mouseDown 事件时，为鼠标在舞台上的所有动作添加了一个侦听，这个侦听可以捕获所有鼠标动作并且让用户决定什么时候调整容器的大小。通过使用 UIComponent 类的 startDrag 和 stopDrag 方法，我们可以设置什么时候是否拖拽代表拖拽图标的 UIComponent。下面的例子，经由 explicitWidth 和 explicitHeight 的 setter 方法，将图标的位置用以设置 Canvas 的新宽和高。

```

<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300" backgroundColor="#ccccff"
verticalScrollPolicy="off" horizontalScrollPolicy="off">
<mx:Script>
<! [CDATA[

```

```

private function startResize():void
{
    stage.addEventListener(MouseEvent.MOUSE_MOVE,
        resize);
    resizeIcon.addEventListener(MouseEvent.MOUSE_UP,
        stopResize);
}

```

```

        resizeIcon.addEventListener(MouseEvent.ROLL_OUT,
            stopResize);
        resizeIcon.startDrag();
    }
}

```

用户按住鼠标按钮不放时，舞台的所有 mouseMove 事件即被捕获。用户释放鼠标按钮的时候，移除所有的事件侦听同时调用 resizeIcon 的 stopDrag 方法。这防止图标移动，同时防止这个组件的调整大小的方法被调用。

```

private function stopResize(mouseEvent:MouseEvent):void
{
    resizeIcon.removeEventListener(MouseEvent.MOUSE_UP,
        stopResize);
    resizeIcon.removeEventListener(MouseEvent.ROLL_OUT,
        stopResize);
    stage.removeEventListener(MouseEvent.MOUSE_MOVE,
        resize);
    resizeIcon.stopDrag();

}

```

Flex 框架不考虑任何测量和调整大小的逻辑，直接使用组件的 explicitHeight 和 explicitWidth 属性指定组件的像素尺寸。

```

private function resize(mouseEvent:MouseEvent):void
{
    this.explicitHeight = resizeIcon.y +
        resizeIcon.height + 10;
    this.explicitWidth = resizeIcon.x + resizeIcon.width
        + 10;
}
]]>
</mx:Script>
<mx:Panel width="60%" height="60%" top="20" left="20"/>
<mx:Image id="resizeIcon"
    source="@Embed('../assets/Resize.png')" mouseDown=
    "startResize()" x="360" y="260"/>
</mx:Canvas>

```

3.16.4 小节 另参照 [3.9. 节](#)

## 3.17 节 在 TabNavigator 内创建、启用和禁用 TAB 组件(TabControls)

### 3.17.1 问题

我们需要从 TabNavigator 动态添加和移除 tab 项，或者偶尔禁用某些 tab 项。

### 3.17.2 解决办法

使用 TabNavigator 的 addChild 和 removeChild 方法添加或者移除子组件，同时设置子组件的 enabled 属性来启用或者禁用 TabNavigator 里的某个 tab 项。

### 3.17.3 讨论

所有添加到 TabNavigator 中的子组件，都会在 TabNavigator 顶部的导航条添加一个新 tab 项。同样移除子组件的时候也会自动移除对应的 tab 项。通过绑定 TabNavigator 的 dataProvider 属性也能行得通。dataProvider 的所有改变都会引起 TabNavigator 更新及其顶部导航条子组件的增加或者减少。这里，组件内创建了访问方法以提供对 TabNavigator 控件的访问：

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
<mx:Script>
<! [CDATA[
import mx.core.UIComponent;

public function
addChildToNavigator(value:UIComponent):void
{
    navigator.addChild(value);
}

public function removeNavigatorChild(value:int =
0):void
{
    if(value == 0) {
        navigator.removeChildAt(navigator.numChildren-1);
    }
    else
    {
        navigator.removeChildAt(value);
    }
}

public function disableTab(value:int = 0):void{
```

```

        if(value == 0) {
            (navigator.getChildAt(navigator.numChildren - 1)
                as UIComponent).enabled = false;
        }
    else
    {
        (navigator.getChildAt(value) as
            UIComponent).enabled = false;
    }
}

public function get tabNumChildren():Number{
    return navigator.numChildren;
}
]]>
</mx:Script>
<mx:TabNavigator id="navigator">
</mx:TabNavigator>
</mx:Canvas>

```

要使用该控件，传入任意一个 Container 对象的实例，此时，Canvas，对组件来说，通过使用该组件的 id 属性的引用来调用方法名。随着 Container 的传入，其 label 属性也传入到 TabNavigator 内，这个 label 属性将用来设置即将添加的新 tab 项的标题。其中，例子中的 Button 控件，用来添加、移除和禁用对应的控件。

```

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute" xmlns:cookbook="oreilly.cookbook.*">
<mx:Script>
<! [CDATA[
    import mx.controls.Label;
    import mx.containers.Canvas;

    private function addCanvas():void
    {
        var canvas:Canvas = new Canvas();
        canvas.title = "Tab "+this.tabNavigator.tabNumChildren.toString();
        var label:Label = new Label();
        label.text = "Hello Label "+this.tabNavigator.tabNumChildren.toString();
        canvas.addChild(label);
        canvas.minHeight = 200;
        canvas.minWidth = 200;
        this.tabNavigator.addChildToNavigator(canvas);
    }
}

```

```

        ] ]>
</mx:Script>
<cookbook:TabNavigatorDemo id="tabNavigator"/>
<mx:HBox y="300">
    <mx:Button click="addCanvas()" label="add child"/>
    <mx:Button click="tabNavigator.removeNavigatorChild()" y="300" label="remove child"/>
    <mx:Button click="tabNavigator.disableTab()" y="300" label="disable child"/>
</mx:HBox>
</mx:Application>

```

## 3.18 节 使用可关闭 Tabs 创建一个 TabNavigator

### 3.18.1 问题

我们需要创建一个拥有的 tab 项的 TabNavigator，即具有关闭按钮的 tab 项，当点击这个按钮的时候同时移除对应的 tab 项和对应的索引的 tab 项下的所有子组件。

### 3.18.2 解决办法

使用 flexlib 库内的 SuperTabNavigator 组件。

### 3.18.3 讨论

Flexlib 库是一组由多个 Flex 开发者开发的开源控件，提供可在所有 Flex 工程内使用的组件。SuperTabNavigator 由 Doug McCune 开发和演示，提供可关闭的 tab 项，tab 重排序和右上角的下拉列表。

```

<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="600" height="600" xmlns:containers="flexlib.containers.*">
    <containers:SuperTabNavigator>
        <mx:VBox width="500" label="First Label">
            <mx:Label text="Label"/>
            <mx:TextInput/>
        </mx:VBox>
        <mx:VBox height="500" label="Second Label">
            <mx:Label text="Label"/>
            <mx:TextInput/>
        </mx:VBox>
        <mx:VBox label="Third Label">

```

```

<mx:Label text="Label"/>
<mx:TextInput/>
</mx:VBox>
</containers:SuperTabNavigator>

</mx:Canvas>Flexlib 库可以从这个地址下载: http://code.google.com/p/flexlib/。下载此包并将解压后的包添加到 FlexBuilder 的工程内, 作为工程的源码包或者作为 MXML 编译器的选项。

```

## 3.19 节 创建和控制 Alert

### 3.19.1 问题

我们需要创建一个 Alert 控件并且控制它的显示。

### 3.19.2 解决办法

使用 mx.controls.Alert 类的 show 方法并且给 show 方法注册一个回调。

### 3.19.3 讨论

我们可以通过使用 mx.controls.Alert 的静态引用, 设置所有 Alert 控件的属性, 这些设定的属性值会一直保存到被再次覆盖为止。

Alert 类的 show 方法需要一个警告所要使用的消息, 一个标题, 一组值以指定显示各自的 Yes, No, 或者 Cancel 按钮, 警报框的父组件, 警报框关闭时的回调函数和警报框里显示的一个图标。例如:

```
mx.controls.Alert.show("This is an alert", "title of the alert", 1|2|8, this,
alertClosed, iconclass);
```

除了前面两个参数, 其余的参数都是可选参数。默认情况下, 警报框会显示一个标签为 OK 的按钮, 点击这个按钮即关闭该警报框。下面是完整的清单:

Code View:

```

<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
<mx:Script>
<! [CDATA[
import mx.events.CloseEvent;
import mx.controls.Alert;

[Embed(source="..../assets/4.png")]

```

```

private var iconclass:Class

private function createAlert():void
{
    mx.controls.Alert.buttonHeight = 20;
    mx.controls.Alert.buttonWidth = 150;
    mx.controls.Alert.yesLabel = "Click Yes?"
    mx.controls.Alert.noLabel = "Click No?"
    mx.controls.Alert.cancelLabel = "Click Cancel?"
    mx.controls.Alert.show("This is an alert", "title of
        the alert", 1|2|8, this, alertClosed, iconclass);
}

private function alertClosed(event:CloseEvent):void
{
    trace(" alert closed ");
    if(event.detail == Alert.YES)
    {
        trace(" user clicked yes ");
    }
    else if(event.detail == Alert.CANCEL)
    {
        trace(" user clicked cancel ");
    }
    else
    {
        trace(" user clicked no ");
    }
}

] >
</mx:Script>
<mx:Button label="Create Simple Alert" click="createAlert()"/>
</mx:Canvas>

```

## 3.20 节 根据呼出组件设置对话框的尺寸和位置

### 3.20.1 问题

我们需要生成一个对话框，该对话框具有和呼出它的组件相同尺寸和位置。

### 3.20.2 解决办法

使用 MouseEvent 的 target 属性来确定调用该方法的组件的信息，同时，使用 mx.geometry.Rectangle 类来确定呼出的组件实际宽高及其在 Stage 内的位置。

### 3.20.3 讨论

为了保证无论应用程序的 layout 设定为 absolute, horizontal, 还是 vertical, 对话框都会添加到应用程序中并且显示在正确的位置，你需要创建一个可关闭的 TabNavigator，该 TabNavigator 的所有 tab 项都是可关闭的，关闭某个 tab 项的时候即从 TabNavigator 中移除对应 tab 项下的子组件。将对话框的 includeInLayout 属性设置为 false 以保证应用程序不改变对话框的位置。

```
dialogue = new Dialogue();
mx.core.Application.application.rawChildren.addChild(dialogue);
dialogue.includeInLayout = false;
```

getBounds 方法返回一个带 x 和 y 位置的矩形，同时 DisplayObject 的宽和高作为传入到方法里面的对象的成员属性。在下面的例子里，Stage 作为传入 getBounds 方法的 DisplayObject 参数，这样，就会根据整个应用返回组件的位置信息了。

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="700"
height="500" left="50" top="50" verticalGap="50">
<mx:Script>
<! [CDATA[
import mx.core.Application;
import mx.core.UIComponent;

private var dialogue:Dialogue;

private function placeLabel(event:MouseEvent):void
{
    if(dialogue == null)
    {
        dialogue = new Dialogue();

        mx.core.Application.application.rawChildren.addChild(dialogue);
        dialogue.includeInLayout = false;
    }

    var rect:Rectangle = (event.target as
UIComponent).getBounds(this.stage);
    dialogue.x = rect.x;
    dialogue.y = rect.y + (event.target as
UIComponent).height;
}
]]>
</mx:Script>
```

```

<mx:HBox horizontalGap="50">
    <mx:Label text="First label."/>
    <mx:Button label="Place First" click="placeLabel(event)"/>
</mx:HBox>
<mx:HBox horizontalGap="50">
    <mx:Label text="Second label."/>
    <mx:Button label="Place Second" click="placeLabel(event)"/>
</mx:HBox>
<mx:HBox horizontalGap="50">
    <mx:Label text="Third label."/>
    <mx:Button label="Place Third" click="placeLabel(event)"/>
</mx:HBox>
<mx:HBox horizontalGap="50">
    <mx:Label text="Fourth label."/>
    <mx:Button label="Place Fourth" click="placeLabel(event)"/>
</mx:HBox>

</mx:VBox>

```

## 3.21 节 管理多个弹出对话框

### 3.21.1 问题

你需要访问并改变多个对话框。

### 3.21.2 解决办法

使用 PopUpManager 类的 createPopUp 方法。

### 3.21.3 讨论

```
var pop:Panel = (PopUpManager.createPopUp(this, mx.containers.Panel,
false, PopUpManagerChildList.POPUP) as Panel);
```

访问并改变多个对话框需要有对这些弹出控件的引用，但是 PopUpManager.addPopUp 方法并没有提供这个引用。因此，你需要使用 PopUpManager 类的 createPopUp 方法，这个方法返回一个被创建对象的引用，这样可以将这个引用添加到一个数组里面去。在一个大的应用程序里面，这样的数组应该申明成全局可访问的，通过 public static 修饰，同时使用 getter 和 setter 方法使得所有组件在需要时都可访问生成的弹出物。例如：

createPopUp 方法需要传入一个父容器引用的参数，即生成弹出物的类，和一个 Boolean 值来判别弹出得对话框是否为模式化的，然后返回被创建对象的引用。

```

<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="600"
height="500" creationComplete="addDialog()">
<mx:Script>
<![CDATA[
import mx.managers.PopUpManagerChildList;
import mx.controls.LinkButton;
import mx.containers.Panel;
import mx.managers.PopUpManager;

public var popUpArray:Array = new Array();

private function addDialog():void
{
    var pop:Panel = (PopUpManager.createPopUp(this,
        mx.containers.Panel, false,
        PopUpManagerChildList.POPUP) as Panel);
    pop.title = "First Pop Up";
    pop.y = 100;
    popUpArray.push(pop);
    pop = (PopUpManager.createPopUp(this,
        mx.containers.Panel, false,
        PopUpManagerChildList.POPUP) as Panel);
    pop.title = "Second Pop Up";
    pop.y = 200;
    popUpArray.push(pop);
    pop = (PopUpManager.createPopUp(this,
        mx.containers.Panel, false,
        PopUpManagerChildList.POPUP) as Panel);
    pop.title = "Third Pop Up";
    pop.y = 300;
    popUpArray.push(pop);
}

private function returnDialog():void
{
    var link:LinkButton = new LinkButton();
    link.label = "Hello";
    (popUpArray[selectDialog.selectedIndex] as
    Panel).addChild(link);
}

]]>
</mx:Script>
<mx:ComboBox id="selectDialog" change="returnDialog()">

```

```

<mx:dataProvider>
    <mx:Array>
        <mx:Number>0</mx:Number>
        <mx:Number>1</mx:Number>
        <mx:Number>2</mx:Number>
    </mx:Array>
</mx:dataProvider>
</mx:ComboBox>
<mx:Panel>
    <mx:LinkButton label="Button"/>
</mx:Panel>
<mx:Panel>
    <mx:LinkButton label="Button"/>
</mx:Panel>
</mx:HBox>

```

## 3.22 节 在容器中滚动到某个指定的子组件

### 3.22.1 问题

你需要在容器上控制默认的滚动行为，同时控制滚动通过单独的控件。

### 3.22.2 解决办法

根据需要滚动到的子组件的索引，使用 getChildAt 方法获取滚动时经过的所有子组件，并计算滚动经过的这些子组件的高度之和。然后使用计算得到的值来设置容器的 verticalScrollPosition 属性值。

### 3.22.3 讨论

在下面的例子中，包含子组件的 Vbox 将 verticalScrollPolicy 属性设置为 off，并且将一个事件侦听绑定到 ComboBox 的 change 属性上。当触发 change 事件的时候，函数循环遍历所有 VBox 的子组件，知道找出需要的那个子组件，求和。然后把这个和设置到 Vbox 的 verticalScrollPolicy 属性。

```

<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="800"
height="600">
<mx:Script>
<! [CDATA[
private function showScrollValue():void
{

```

```

trace(this.verticalScrollPosition+" "+  

    this.horizontalScrollPosition);
}

private function changeScrollToShowChild():void  

{
    vbox.verticalScrollPosition =
        (returnChildrenHeights((comboBox.selectedItem as  

        Number)+1)) - vbox.height;
}

```

Vbox 内所有子组件的高度用以计算并确定一个 y 值来设定 verticalScrollPosition 属性，从而使 Vbox 滚动到指定的子组件的位置。

```

private function  

    returnChildrenHeights(index:int):Number  

{
    var i:int = 0;  

    var sumHeight:Number = 0;  

    while(i<index) {  

        sumHeight+=vbox.getChildAt(i).height;  

        i++;
    }
    return sumHeight;
}

] ]>
</mx:Script>
<mx:ComboBox id="comboBox" change="changeScrollToShowChild()">
    <mx:dataProvider>
        <mx:Array>
            <mx:Number>1</mx:Number>
            <mx:Number>2</mx:Number>
            <mx:Number>3</mx:Number>
            <mx:Number>4</mx:Number>
            <mx:Number>5</mx:Number>
        </mx:Array>
    </mx:dataProvider>
</mx:ComboBox>
<mx:VBox width="650" height="300" id="vbox">
    backgroundColor="#00ffff" y="50" verticalScrollPolicy="off"
    scroll="showScrollValue()" paddingLeft="50">
        <mx:Panel height="150" width="550">
            <mx:LinkButton label="First"/>
        </mx:Panel>

```

```

<mx:Panel height="160" width="550">
    <mx:LinkButton label="Second"/>
</mx:Panel>
<mx:Panel height="110" width="550">
    <mx:LinkButton label="Third"/>
</mx:Panel>
<mx:Panel height="150" width="550">
    <mx:LinkButton label="Fourth"/>
</mx:Panel>
<mx:Panel height="130" width="550">
    <mx:LinkButton label="Fifth"/>
</mx:Panel>
</mx:VBox>
</mx:Canvas>

```

## 3.23 节 使用 IDeferredInstance 创建模板

### 3.23.1 问题

现在你需要创建一个模板组件以处理传入的多种类型组件，同时也需要延迟实例化组件以提高启动效率。

### 3.23.2 解决办法

使用 `IDeferredInstance` 标记来指定一个数组或者属性处理所有传入的组件类型，组件实例化的同时也创建它的子组件。

### 3.23.3 讨论

所有 `UIComponents` 都实现 `IDeferredInstance` 接口并且允许将子组件添加到某个数组中，然后在之后其父组件实例化的时候才实例化这些子组件。所有传入一个数组的类型为 `IDeferredInstance` 的组件都会保存到它们添加到显示清单为止，如下所示：

```

<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            import mx.containers.HBox;
            import mx.containers.ViewStack;
            import mx.core.UIComponent;

            // Define a deferred property for the top component.
            public var header:IDeferredInstance;
        ]]>
    </mx:Script>

```

```

// Define an Array of deferred properties
// for a row of components.

[ArrayElementType("mx.core.IDeferredInstance")]
public var leftDataRow:Array;

[ArrayElementType("mx.core.IDeferredInstance")]
public var centerDataRow:Array;

[ArrayElementType("mx.core.IDeferredInstance")]
public var rightDataRow:Array;

public var layoutHBox:HBox;

public var layoutWidth:int = 0;

public function createDeferredComponents():void {
    addChild(UIComponent(header.getInstance()));
    layoutHBox = new HBox();
    if(layoutWidth != 0) {
        layoutHBox.setStyle("horizontalGap", layoutWidth);
    }
    if(leftDataRow.length > 0) {
        var leftVBox:VBox = new VBox();
        layoutHBox.addChild(leftVBox);
        for (var i:int = 0; i < leftDataRow.length; i++) {

            leftVBox.addChild(UIComponent(leftDataRow[i].getInstance()));
        }
    }
    if(centerDataRow.length > 0) {
        var centerVBox:VBox = new VBox();
        layoutHBox.addChild(centerVBox);
        for (var i:int = 0; i < centerDataRow.length;
            i++)
        {
            centerVBox.addChild(UIComponent(centerDataRow[i].
                getInstance()));
        }
    }
    if(rightDataRow.length > 0) {
        var rightVBox:VBox = new VBox();
        layoutHBox.addChild(rightVBox);
        for (var i:int = 0; i < rightDataRow.length;
            i++)
    }
}

```

```

        {
            rightVBox.addChild(UIComponent(rightDataRow[i]
                .getInstance()));
        }
    }
    // Add the HBox container to the VBox container.
    addChild(layoutHBox);
}
]]>
</mx:Script>
</mx:VBox>

```

我们可以看到，在下的例子中，你可以传入任何继承了 IDeferredInstance 对象的数组到一个组件，然后稍后再调用一个方法来将这些子组件添加到 displayList 属性。虽然这个方法可以连接在异步或者其它任何时间后调用，但在该例中，明确的调用了该方法。

```

<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="700"
height="500" xmlns:cookbook="oreilly.cookbook.*">
<mx:Script>
<! [CDATA[
private function createDeferredInstance():void
{
    this.deferredInstance.createDeferredComponents();
}

]]>
</mx:Script>
<mx:Button click="createDeferredInstance()"
label="make components"/>
<cookbook:DeferredInstance layoutWidth="30"
id="deferredInstance">
<cookbook:header>
<mx:Label text="This will be the header of my templated
component"/>
</cookbook:header>
<cookbook:leftDataRow>
<mx:Array>
<mx:Label text="data"/>
<mx:Label text="data"/>
<mx:Label text="data"/>
</mx:Array>
</cookbook:leftDataRow>
<cookbook:centerDataRow>
<mx:Array>

```

```
<mx:Button label="press me"/>
<mx:Label text="data"/>
</mx:Array>
</cookbook:centerDataRow>
<cookbook:rightDataRow>
<mx:Array>
<mx:CheckBox label="click"/>
<mx:Button label="press me"/>
<mx:Label text="data"/>
</mx:Array>
</cookbook:rightDataRow>
</cookbook:DeferredInstance>

</mx:Canvas>
```

## 3.24 节 手动布局容器

### 3.24.1 问题

你需要根据类型以及类型的属性来布置容器的子组件。

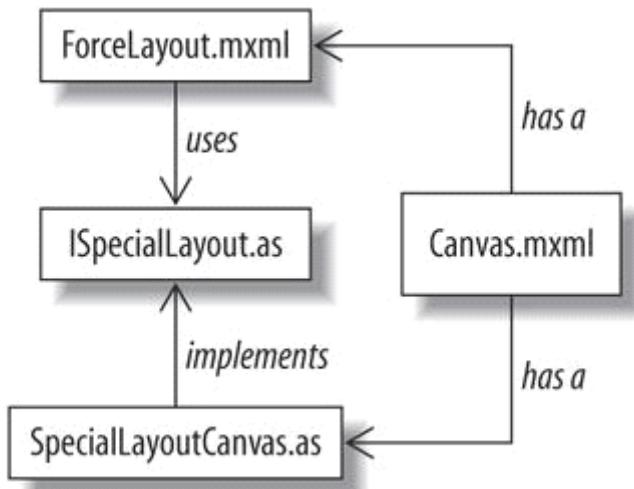
### 3.24.2 解决办法

覆盖 UIComponent 的 updateDisplayList 方法来移动子组件。

### 3.24.3 讨论

要在任何 Container 或者 UIComponent 对象内覆盖任何布局或设置大小的逻辑，只要覆盖 updateDisplayList 方法并在 super.updateDisplayList 分方法之后插入你自己的布局逻辑即可。本节专注于如何通过使用子组件的自定义的属性来决定 Hbox 子组件的布局。这个例子由 4 个文件组成，三个类和一个接口，它们各司其职并拥有自己独有的属性。大致的 UML 图(图 [Figure 3-1](#))描述了它们之间的关系。

figure 3-1. Relationships between containers and interfaces



Canvas.mxml 文件内添加了 ForceLayout 组件并提供其子组件。ForceLayout 组件可以包含任何子组件，但如果子组件实现了 ISpecialLayout 接口，ForceLayout 组件将会根据此类子组件是否被选中来进行不同的布局。SpecialLayoutCanvas 定义了简单的方法来决是否被选中。

首先，看一下 SpecialLayoutCanvas.as 文件：

```
package oreilly.cookbook
{
    import mx.containers.Canvas;
    import mx.controls.Label;
    import mx.core.UIComponent;
    import flash.events.MouseEvent;

    public class SpecialLayoutCanvas extends Canvas implements
        ISpecialLayout
    {

        private var titlelabel:Label;
        private var selectedlabel:Label;
        private var _isSelected:Boolean = false;

        public function SpecialLayoutCanvas()
        {
            super();
        }
    }
}
```

```

        titlelabel = new Label();
        addChild(titlelabel);
        titlelabel.text = "Label";
        this.addEventListener(MouseEvent.MOUSE_DOWN,
            setIsSelected);
        minHeight = 45;
        minWidth = 80;
        selectedlabel = new Label();
        //addChild(selectedlabel);
        selectedlabel.text = "Selected";
    }

private function setIsSelected(mouseEvent:MouseEvent):void
{
    _isSelected ? isSelected = false : isSelected = true;
}

public function set isSelected(value:Boolean):void
{
    _isSelected = value;
    if(isSelected)
    {
        addChild(selectedlabel);
        selectedlabel.y = 30;
    }
    else
    {
        try{
            removeChild(selectedlabel);
        }catch(err:Error){}
    }
    if(parent != null)
    {
        (parent as UIComponent).invalidateDisplayList();
    }
}

public function get isSelected():Boolean
{
    return _isSelected;
}
}

```

这个类使用 getter 和 setter 方法简单地定义了一个 selected 属性，并且当对象被选中的时候给其添加一个标签，反之删除这个标签。

下一步，我们来看 ForceLayout 组件，它读取所有的子组件来判别它们中是否有继承 IspecialLayout 接口的，如果有，是否选中了。

```
package oreilly.cookbook
{
    import mx.core.EdgeMetrics;
    import mx.core.UIComponent;
    import mx.containers.VBox;
    import mx.containers.Panel;
    import mx.containers.Canvas;
    import flash.display.DisplayObject;

    public class ForceLayout extends VBox
    {
        public var gap:Number;
        public function ForceLayout()
        {
            super();
        }
    }
}
```

无论何时，只要该组件需要重绘的时候，Flex 框架即调用 updateDisplayList 方法。因为重绘的时候组件需要做的一件事情是重新调整所有子组件的位置，所有调整位置的逻辑都在这里实现：

```
override protected function
updateDisplayList(unscaledWidth:Number,
    unscaledHeight:Number):void
{
    super.updateDisplayList(unscaledWidth, unscaledHeight);
    var yPos:Number = unscaledHeight;
    // Temp variable for a container child.
    var child:UIComponent;
    var i:int = 0;
    while(i<this.numChildren)
    {
        // Get the first container child.
        child = UIComponent getChildAt(i));
        // Determine the y coordinate of the child.
        yPos = yPos - child.height;
        // Set the x and y coordinate of the child.
    }
}
```

```

// Note that you do not change the x coordinate.
if(child is ISpecialLayout)
{
    if((child as ISpecialLayout).isSelected)
    {
        yPos -= 20;
        child.move(child.x, yPos);
        yPos -= 20;
    }
    else
    {
        child.move(child.x, yPos);
    }
}
else
{
    child.move(child.x, yPos);
}

// Save the y coordinate of the child,
// plus the vertical gap between children.
// This is used to calculate the coordinate
// of the next child.
yPos = yPos - gap;
i++;
}

i = 0;
var amountToCenter:Number = yPos / 2;
while(i<this.numChildren)
{
    getChildAt(i).y -= amountToCenter;
    i++;
}
}
}

```

最后的清单将两个组件付诸使用，将 ForceLayout 容器添加到 Canvas 并且添加 SpecialLayoutCanvas 子组件。注意，如果它们是当前存在的，仅仅改变布局即可，不需要任何特别属性，任何类型子组件都可以添加到 ForceLayoutCanvas，并且实际上所有实现了 ISpecialLayout 接口的子组件都是可用的。

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="800"  
height="600" xmlns:cookbook="oreilly.cookbook.*">  
    <cookbook:ForceLayout width="400" height="500"  
        backgroundColor="#ffffff">
```

```

<mx:HBox>
    <mx:Button label="button"/>
    <mx:LinkButton label="link"/>
</mx:HBox>
<cookbook:SpecialLayoutCanvas isSelected="false"
    backgroundColor="#c0c0cc"/>
<mx:HBox>
    <mx:Button label="button"/>
    <mx:LinkButton label="link"/>
</mx:HBox>
<cookbook:SpecialLayoutCanvas isSelected="false"
    backgroundColor="#ccc0c0"/>
<cookbook:SpecialLayoutCanvas isSelected="true"
    backgroundColor="#cc00cc"/>
<cookbook:SpecialLayoutCanvas isSelected="false"
    backgroundColor="#ccc0c0"/>
</cookbook:ForceLayout>

</mx:Canvas>

```

## 3.25 节 测量并改变容器尺寸

### 3.25.1 问题

你需要它的根据子组件改变某个容器的尺寸。

### 3.25.2 解决办法

覆盖容器的 measure 属性，当 Flex 框架调用 updateDisplayList 方法的时候调用该属性。

### 3.25.3 讨论

无论何时，只要容器需要确定子组件有多大以及根据所有的式样和约束信息自己有多大的时候，Flex 框架都会调用 measure 方法来确定容器本身的尺寸。类似 3.24 节覆盖 updateDisplayList 方法的做法，你可以覆盖 measure 方法来执行所有可能需要改变尺寸的自定义计算。

```

<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <! [CDATA[
            import mx.core.Container;
            import mx.core.UIComponent;

```

```

override protected function measure():void
{
    super.measure();
    var childrenWidth:int = 0;
    var childrenHeight:int = 0;
    //loop through all children, and determine the height and width
    //of all the children components
    for(var i:int = 0; i<this.numChildren; i++)
    {
        var obj:UIComponent = (getChildAt(i) as UIComponent);
        if(obj is Container)
        {
            //here we are using the viewMetricsAndPadding so that we get any style information affiliated
            //with the child as well as its actual width
            childrenWidth += Container(obj).viewMetricsAndPadding.left+
Container(obj).viewMetricsAndPadding.right+obj.width;
            childrenHeight += Container(obj).viewMetricsAndPadding.top+
Container(obj).viewMetricsAndPadding.bottom+obj.height;
        }
        else
        {
            childrenWidth += obj.width;
            childrenHeight += obj.height;
        }
    }
    //set this components measured height based on our calculations
    measuredHeight = childrenHeight;
    measuredWidth = childrenWidth;
}

]]>
</mx:Script>
</mx:HBox>

```

下一步，使用 ExtendableCanvas. mxml 文件，向这个 Canvas 里添加子组件来测试它是否适当的扩大了。

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="1400"
```

```

height="500" xmlns:cookbook="oreilly.cookbook.*">
<mx:Script>
<! [CDATA[
import mx.containers.Panel;
private function addChildToExtendableCanvas():void
{
    var panel:Panel = new Panel();
    panel.height = 100 + Math.random()*200;
    panel.width = 100 + Math.random()*200;
    extCanvas.addChild(panel);
}
]]>
</mx:Script>
<mx:Button click="addChildToExtendableCanvas()"
label="add child"/>
<cookbook:ExtendableCanvas id="extCanvas" y="50"
verticalScrollPolicy="off" horizontalScrollPolicy="off"/>
</mx:Canvas>

```

当更多的子组件添加到 Container 里面时，ExtendableCanvas 实例将使用所有子组件的宽和高加上 padding 样式信息指定的值来重绘并调整自己的大小。

## 3.26 节 控制子组件的可见性和布局

### 3.26.1 问题

你需要无破坏地从容器的已有布局中移动子组件。

### 3.26.2 解决办法

使用 UIComponent 类的 includeInLayout 属性同时把 visibility 设置为空。

### 3.26.3 讨论

某容器的子组件的 includeInLayout 属性表明该子组件是否包含在父亲布局子组件使用的任意布局计划中：VBox, HBox, 或一个 Canvas 的居中设置。如果仅仅简单地将子组件的 visibility 设置为 false, 它将仍然包含在父亲的布局机构当中，拥有足够容纳其宽和高所需要的空间。改为将 includeInLayout 设置成 false 以保证 Hbox 不会给该子组件布局空间：

```
(event.target as UIComponent).includeInLayout = false;
```

重新包含此子组件到布局中，设置 includeInLayout 为 true。下面的例子通过 includeInLayout 属性使得子组件可被拖拽。随着包含到 Hbox 的布局中，布局管理器会改变子组件的位置以遵守其布局。由于 includeInLayout 属性设置成 false，当 Hbox 设置它所有子组件的 x 值的时候会忽略这个子组件。

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="600"
height="400">
<mx:Script>
<! [CDATA[
import mx.core.UIComponent;

private function
removeFromLayout(event:MouseEvent):void
{
    (event.target as UIComponent).includeInLayout =
        false;
    (event.target as UIComponent).startDrag();
}

private function
reincludeInLayout(event:MouseEvent):void
{
    (event.target as UIComponent).stopDrag();
    (event.target as UIComponent).includeInLayout = true;
}

private function
hideAndRemoveFromLayout(event:Event):void
{
    (event.target as UIComponent).visible = false;
    (event.target as UIComponent).includeInLayout =
        false;
}

]]>
</mx:Script>
<mx:VBox>
<mx:LinkButton id="firstLabel" label="this is the first
label" mouseDown="removeFromLayout(event)"
mouseUp="reincludeInLayout(event)"/>
<mx:LinkButton id="secondLabel" label="this is the second
label" mouseDown="removeFromLayout(event)"
mouseUp="reincludeInLayout(event)"/>
<mx:Button id="button" label="My First Button"
```

```
        mouseDown="removeFromLayout(event)"
        mouseUp="reincludeInLayout(event)"/>
    </mx:VBox>
    <mx:VBox>
        <mx:Button label="Remove from Layout and Hide"
        click="hideAndRemoveFromLayout(event)" borderColor="#0000ff"/>
        <mx:Button label="Remove from Layout and Hide"
        click="hideAndRemoveFromLayout(event)"
        borderColor="#00ff00"/>
        <mx:Button label="Remove from Layout and Hide"
        click="hideAndRemoveFromLayout(event)"
        borderColor="#ff0000"/>
    </mx:VBox>

</mx:HBox>
```

## 3.27 节 用简单重组行为创建 Tile 容器

### 3.27.1 问题

你想要让用户可以在 Tile 容器里面拖拽其瓷砖 (tile) 并且在用户放下瓷砖 (tile) 的时候容器重组。

### 3.27.2 解决办法

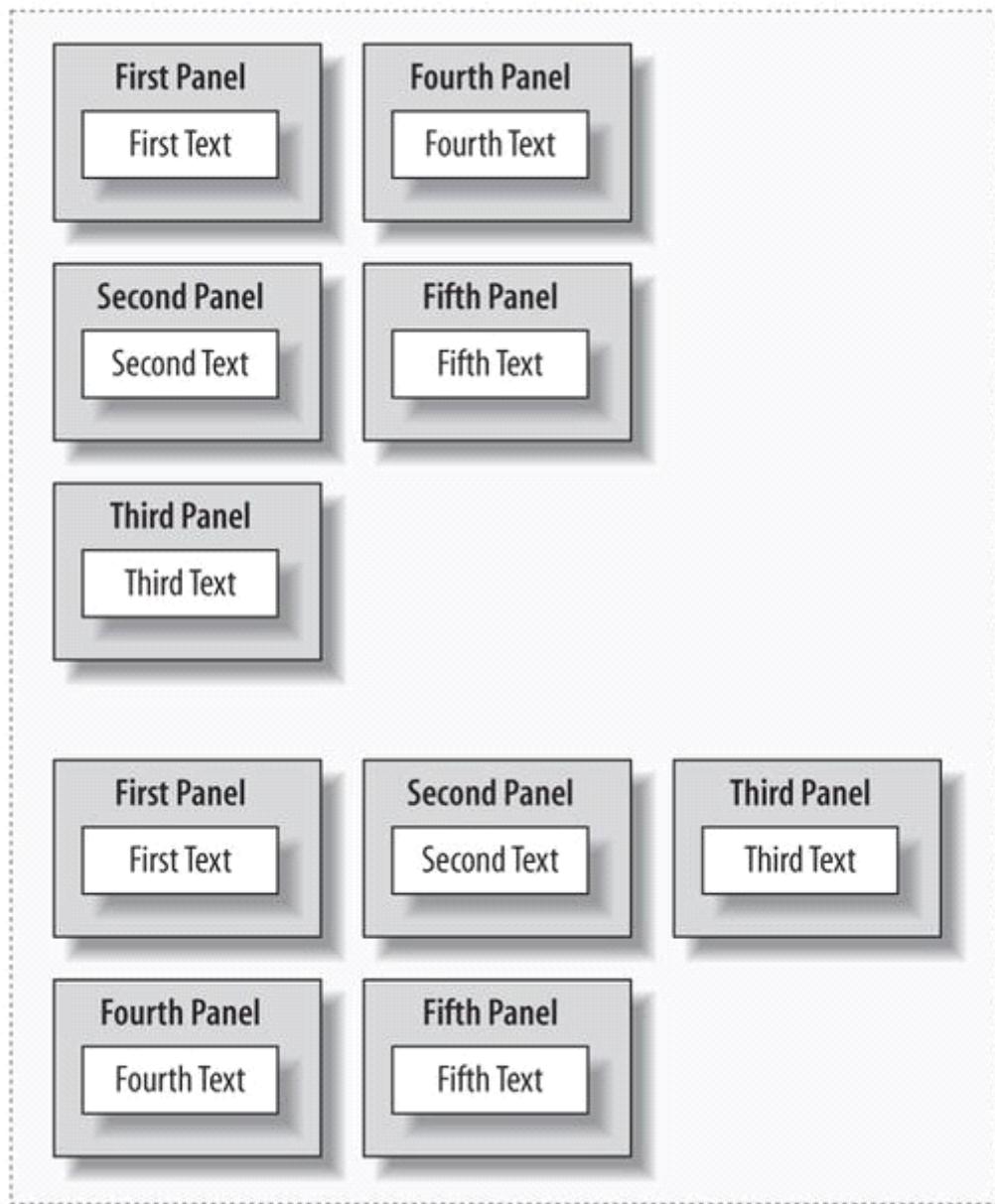
使用 Tile 继承自 DisplayObjectContainer 类的 swapChildrenmethod 方法来改变 Tile 内子组件的位置。

### 3.27.3 讨论

Tile 容器给子组件布局的方式和 Box 大体相同，除了有空间放置适合非第一方向的第二渲染器存在的情形，Tile 容器可添加多个 itemRenderer。如果 Tile 容器的 direction 属性设置为 horizontal，例如，并且 Tile 容器为 400 像素宽，而添加进来的元素宽度 150 像素，那么这个瓷砖(tile)在添加两个这样的组件之后，会根据需要提供垂直方向上的滚动条来添加后面的组件直到添加结束。Tile 组件通常创建像网格这样的排列方式（图 [Figure 3-2](#)）

**Figure 3-2. Tiles with vertical and horizontal directions**

**Figure 3-2. Tiles with vertical and horizontal directions**



在下面的例子中，Tile 组件用来排列它的子组件。当一个子组件拖拽操作放下时，Tile 被标记，并通过调用 invalidateDisplayList 方法重绘。

Code View:

```
<mx:Tile xmlns:mx="http://www.adobe.com/2006/mxml" width="300"  
height="600" direction="horizontal">  
<mx:Script>  
    <! [CDATA[  
        import mx.core.UIComponent;
```

```

private function childStartDrag(event:Event) :void
{
    (event.currentTarget as UIComponent).startDrag(false,
        this.getBounds(stage));
    (event.currentTarget as
        UIComponent).addEventListener(MouseEvent.MOUSE_UP,
            childStopDrag);
    (event.currentTarget as
        UIComponent).addEventListener(MouseEvent.ROLL_OUT,
            childStopDrag);
    swapChildren((event.currentTarget as UIComponent),
        getChildAt(numChildren-1));
}

private function childStopDrag(event:Event) :void
{
    swapChildren((event.currentTarget as UIComponent),
    hitTestChild((event.currentTarget as UIComponent)));
    (event.currentTarget as UIComponent).stopDrag();
    this.invalidateDisplayList();
    this.invalidateProperties();
}

private function
hitTestChild(obj:UIComponent):DisplayObject
{
    for(var i:int = 0; i<this.numChildren; i++)
    {
        if(this.getChildAt(i).hitTestObject(obj))
        {
            return getChildAt(i);
        }
    }
    return getChildAt(0)
}
] ]>
</mx:Script>
<mx:Panel title="First Panel"
    mouseDown="childStartDrag(event)">
    <mx:Text text="First Text"/>
</mx:Panel>
<mx:Panel title="Second Panel"
    mouseDown="childStartDrag(event)">

```

```

<mx:Text text="Second Text"/>
</mx:Panel>
<mx:Panel title="Third Panel"
    mouseDown="childStartDrag(event)">
    <mx:Text text="Third Text"/>
</mx:Panel>
<mx:Panel title="Fourth Panel"
    mouseDown="childStartDrag(event)">
    <mx:Text text="Fourth Text"/>
</mx:Panel>
</mx:Tile>

```

## 3.28 节 给 Hbox 设置背景图片和圆角

### 3.28.1 问题

你需要创建一个圆角和背景图片的 Hbox。

### 3.28.2 解决办法

加载一个图片对象并且使用 beginBitmapFill 方法创建一个位图填充。

### 3.28.3 讨论

如果背景不是一张图片的话，设置 Hbox 的 cornerRadius 会出现圆角。但是，如果按照下面这样给 HBox 设置一张背景图片：

```

<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="300" backgroundImage="../../assets/beach.jpg"
    borderStyle="solid" borderThickness="0" cornerRadius="20">

```

Hbox的角则会以图片的直角为准。要将图片的角变成圆角，需要将图片转化换成填充图片。所有的UIComponent都有graphics属性，一个flash.display.Graphic对象的实例，这个属性具有低水平的制图程序。通过使用beginBitmapFill方法，你可以为drawRoundRect复杂方法创建一个填充(fill)以使用，用你加载的图片的二进制数据填充到一个圆角矩形内。调用endFill方法完成绘画程序，如下：

```

this.graphics.beginBitmapFill(bm, m, true, true);
this.graphics.drawRoundRectComplex(0, 0, this.width, this.height,
    20, 20, 20, 20);
this.graphics.endFill();

```

一个加载器加载图片，然后加载的图片的所有数据都存入一个 BitmapData 对象。

```

var bm:BitmapData = new BitmapData(loader.width, loader.height,
    true, 0x000000);
bm.draw(this.loader);

```

现在你可以使用 `BitmapData` 对象来创建填充。完整例子代码如下：

```

<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300" creationComplete="createFill()"
cornerRadius="20">
<mx:Script>
<! [CDATA[
import flash.net.URLRequest;
private var loader:Loader;
private function createFill():void
{
    loader = new Loader();
    loader.contentLoaderInfo.addEventListener(
        Event.COMPLETE, completeLoad);
    loader.load(new
        URLRequest("../../assets/beach.jpg"));
}
private function completeLoad(event:Event):void
{
    var bm:BitmapData = new BitmapData(loader.width,
        loader.height, true, 0x000000);
    bm.draw(this.loader);
    var m:Matrix = new Matrix();
    m.createBox(this.width/loader.width,
        this.height/loader.height);
    this.graphics.beginBitmapFill(bm, m, true, true);
    this.graphics.drawRoundRectComplex(0, 0, this.width,
        this.height, 20, 20, 20, 20);
    this.graphics.endFill();
}
]]>
</mx:Script>
</mx:HBox>

```

## 3.29 节 控制子组件的位置和滚动

### 3.29.1 问题

你需要滚动一个父组件并且移动除了一个子组件之外的其它所有子组件。

### 3.29.2 解决办法

在容器定义的 scrollChildren 方法内, 根据 verticalScrollPosition 属性重新定位子组件。

### 3.29.3 讨论

容器的 scrollChildren 方法测量容器的 contentPane 这个 DisplayObject, 它包含了添加到容器中的所有子组件, 同时确定在滚动的时候测量到的子组件要显示出多少。contentPane 则根据 horizontalScrollPosition 和 verticalScrollPosition 属性移动。容器自身则像是 contentPane 的遮罩一样, 并且 contentPane 的位置由滚动条的相对位置以及容器的 ViewMetrics 属性值确定。

下面的代码段把顶部组件的 y 位置存储起来, 允许用户拖拽组件的同时保持组建距离父亲容器顶部位置的不变, 而正常设置其它组件:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="500"
height="500">
<mx:Script>
<! [CDATA[
    private var top:TopComponent;
    //store the y position
    private var topY:Number;
    private function showTop():void
    {
        top = new TopComponent();
        addChild(top);
        top.verticalScrollPolicy = "none";
        top.x = 200;
        top.y = 100;
        topY = top.y;
        top.addEventListener(MouseEvent.MOUSE_DOWN, dragTop);
    }

    private function dragTop(event:Event):void
    {
        top.startDrag(false, this.getBounds(stage));
        top.addEventListener(MouseEvent.MOUSE_UP,
        stopDragTop);
    }
]}>
```

```

        }
private function stopDragTop(event:Event):void
{
    topY = top.y;
    top.stopDrag();
    top.removeEventListener(MouseEvent.MOUSE_UP,
        stopDragTop);
}
override protected function scrollChildren():void
{
    super.scrollChildren();
    if(top) {
        top.y = verticalScrollPosition+topY;
        //
        top.verticalScrollPosition =
            this.verticalScrollPosition/height *
            top.height;
    }
}

]]>
</mx:Script>
<mx:Panel>
    <mx:Label text="LABEL BABEL"/>
    <mx:Label text="LABEL BABEL"/>
    <mx:Label text="LABEL BABEL"/>
</mx:Panel>
<mx:Panel y="500" height="200">
    <mx:Label text="LABEL BABEL"/>
    <mx:Label text="LABEL BABEL"/>
    <mx:Label text="LABEL BABEL"/>
</mx:Panel>
<mx:Button click="showTop()"/>

</mx:Canvas>

```

这有点过于简单了，而且同样的分类逻辑可以被扩展以给容器全部子组件创建自定义的滚动。如果你已经知道容器全部子组件可以滚动，你可以覆盖 scrollChildren 方法，如下：

```

private var dir:String;
private function handleScroll(event:ScrollEvent):void {
    dir = event.direction;
}

override protected function scrollChildren():void {

```

```

var i:int = 0;
do{
    var comp:DisplayObject = getChildAt(i);
    if(comp is Container){
        trace( Container(comp).maxVerticalScrollPosition );
        dir == "horizontal" ?
            Container(comp).horizontalScrollPosition =
                horizontalScrollPosition *
                (Container(comp).maxHorizontalScrollPosition/maxHorizontalScrollPosition) :
            Container(comp).verticalScrollPosition =
                verticalScrollPosition *
                (Container(comp).maxVerticalScrollPosition/maxVerticalScrollPosition);
    }
    i++;
} while (i < numChildren)
}

```

本例使用 maxVerticalScrollPosition 和 maxHorizontalScrollPosition 计算每个容器滚动的正确位置。容器通过从其所有子组件的宽和高中减去容器的实际宽和高来决定这些属性。在前面的代码段中，每个子组件的 maxVerticalScrollPosition 和 maxHorizontalScrollPosition 都由父亲的值来分隔以确定子组件滚动多少。这保证即使父亲和子组件不一样大小，传给子组件的滚动量也是正确的。

## 第四章 文本 (草衣薰)

当你需要在 Flex 程序中使用文本时, 请用以下这些组件: mx.text.Text, mx.text.RichTextEditor, mx.text.Label, mx.text.TextArea, mx.text.TextInput, 和 flash.text.TextField。这些组件的每一个都可以在 Flex 程序的内容表现中实现不同的功能。TextInput, TextArea, 和 RichTextEditor 控制器考虑到用户交互和编辑。TextArea, RichTextEditor, 和 Text 控制器考虑到多行文本的显示操作。最后 flash.text.TextField 是一个底层类, 它给你对 TextField 中的文本布局和处理提供了细节操作能力, 但是在 Flex 程序中使用时需要先用 UIComponent 实例来装载它。实际上, 任何一个 mx.text 包中的 Flex 控制器都是利用 flash.text.TextField, 对这个组件添加不同功能。

Flex 允许显示一个简单的文本或者一个 HTML 子集, 通过文本格式或者 css 样式控制文本的表现。当使用 Flash 播放器支持的 HTML 子集时, 图片和其他 swf 文件中的内容都可以被加载到 Flash 中。文本格式就是控制字体大小和颜色, 通过 css 或使用了 flash.text.TextFormat 对象的 flash.text.TextField 组件来设置 mx.text 组件上的属性来完成。文本可以被用户选择或者用 setSelection 方法编程实现。本章的处方覆盖了对全部 6 个这类组件的应用。

### 4.1 节 正确的设置一个文本对象的值

#### 4.1.1. 问题

我想正确的显示出可能被传送到文本对象的 HTML 和简单字符串

#### 4.1.2. 解决办法

使用 htmlText 和 text 属性, 依靠输入模式, 来适当地渲染文本和采用正则表达式分析被传到 Text 对象中的字符串。

#### 4.1.3. 讨论

Text 和 TextArea 不会正确的显示 HTML 除非 HTML 被传 Text or TextArea 组件的 htmlText 属性。通常把非 HTML 文本传递给 Text or TextArea 不会有问题, 除非文本里可能包含了 HTML 字符.

正则表达式是一个强力工具, 它可以让你快速而且高效解析文本或者文本模板, 避免冗长乏味的字符串操作。这个表达式寻找“<”, 接上任意数量的字母字符, 接上“>”:

```
var regexp:RegExp = /<.+>/;
```

这个例子用一个正则表达式来决定是否把这个含有 HTML 或 XML 的字符串传递给 Text 组件:

代码视图:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
```

```

height="300">
<mx:Script>
<! [CDATA[
    private var htmlStr1:String =
        '<b>Header</b><br/>Hello.<i>Hello.</i> <font
        color="#ff0000" size="15">RED</font>';
    private var htmlStr2:String =
        "<ul><li>Item 1</li><li>Item
        2</li><li>Item3</li></ul>";
    private var textStr1:String = "It is a long established
        fact that a reader will be distracted by the readable
        content of a page when looking at its layout, if say
        the amount of text > 100.";
    private var textStr2:String = " We can use <>String<>
        to indicate in Erlang that the values being passed
        are Strings";
    private function setNewText():void
    {
determineTextType(changeText.selectedItem.value.toString());
    }
    private function determineTextType(str:String):void
    {

```

这里用正则表达式 来决定是否找到任何通过模板一个“<”符号，接上任何字字母，接上另一个“> ”符号测试的 HTML 标签:

代码视图:

```

var regexp:RegExp = /<.+>/;
if(regexp.test(str))
{
    textArea.htmlText = str;
}
else
{
    textArea.text = str;
}
]
]]>
</mx:Script>
<mx:ComboBox id="changeText" dataProvider="[{label:'HTML1',
    value:htmlStr1}, {label:'HTML2', value:htmlStr2},
    {label:'Text1', value:textStr1}, {label:'Text2',
    value:textStr2}]}> change="setNewText()"/>
<mx:TextArea id="textArea" height="100%"/>
</mx:VBox>
```

## 4.2 节. 将 TextInput 绑定一个值

### 4.2.1. 问题

我想将一个用户输入到 TextInput 控制器中的值绑定到另一个控制器。

### 4.2.2. 解决办法

使用绑定标签将 TextInput 组件中的文本绑定到 Text 组件中，以便显示输入的内容。

### 4.2.3. 讨论

TextInput 控制器在这里被用来提供要显示在 TextArea 控制器中的文本。随着文本内容的增加，采用了 Flex 框架的绑定机制的 TextArea 的宽度也会随之增加。

Code View:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="300">
    <mx:TextInput id="input" width="200"/>
    <mx:TextArea text="{input.text}" width="200" id="area"
        backgroundAlpha="0"
        height="{(Math.round(input.text.length/40)+1)*20}"
        wordWrap="true"/>
</mx:VBox>
```

TextInput 和它的值都可以绑定到一个变量。当用户输入文本时并不会把文本内容传递给变量，但是他会改变绑定了 TextInput 的任何组件的文本属性。例如：

Code View:

```
<mx:Script>
    <![CDATA[
        [Bindable]
        private var bindableText:String = "Zero Text";
        private function setText():void
        {
            bindableText = String(cb.selectedItem);
        }
    ]]>
</mx:Script>
<mx:ComboBox id="cb" dataProvider="['First Text', 'Second
    Text', 'Third Text', 'Fourth Text']" change="setText()"/>
<mx:TextInput id="inputFromCB" width="200"
    text="{bindableText}"/>
<mx:Text id="textFromCB" width="200"
    text="{inputFromCB.text}"/>
```

## 4.3 节. 创建一个具有文字提示的文本输入框

### 4.3.1. 问题

我想创建一个具有预测能力的 TextInput，他会从词典中找出一些推荐的单词供用户选择。

### 4.3.2. 解决办法

使用 TextInput 组建定义的 change 事件监听用户输入，同时使用正则表达式测试词典里是否有何用户输入的内容相匹配的单词。

### 4.3.3. 讨论

TextInput 组建定义了 change 事件，TextInput 组件的值的每次变更都会由它来分发。你可以用这个事件去检测用户输入和测试短词典中所有单词的匹配行为。例如：

Code View:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
<mx:Script>
<! [CDATA[
    [Bindable]
    private var probableMatches:Array;
    private var allWords:Array = ["apple", "boy", "cat",
        "milk", "orange", "pepper", "recipe", "truck"];
    private var regexp:RegExp;
    private function checkInput():void
    {
        var i:int = 0;
        var temp:Array = allWords.filter(filter);
        input.text = temp[0];
    }
    private function filter(element:*, index:int,
        arr:Array):Boolean
    {
        regexp = new RegExp(input.text);
        return (regexp.test(element as String));
    }
]]>
</mx:Script>
<mx:TextInput id="input" change="checkInput()"/>
</mx:Canvas>
```

这里使用的这个过滤方法是 Array 类的一部分，它让你创建一个可以载入任何一个对象的方法，然后通过计算，返回一个布尔值来指明是否这一项可以被包括在过滤后的数组里。这个临时数组在检测输入的方法中创建，它包含了匹配正则表达式的所有项。

## 4.4 节. 创建一个合适的编辑器

### 4.4.1. 问题

我想创建一个合适的编辑器组件，当用户点击文本去有的时候可以进行编辑。

### 4.4.2. 解决办法

使用 Text 组件的 click 监听器改变组件的状态来显示一个 TextInput。使用 TextInput 组件的 enter 和 focusOut 事件来决定当用户完成了编辑然后返回 Text 组件的状态。

### 4.4.3. 讨论

状态是一个强力和方便的给单一组件添加多种视觉的方法。本节的例子采用两种状态：显示状态和编辑状态。显示状态保留了显示文本值的 Label，编辑状态保留了允许用户编辑值的 TextInput 组件。

把变量 currentState 的属性设置成你希望显示的状态对应的字符串值就可以切换状态了。例如：

```
currentState = "display";
```

要保证在点击 Enter 按钮后存储用户输入完的值或者在用户点击 TextInput 的某个位置后，把焦点设置到 TextInput 组件自己身上，就需要在创建组件时监听 enter 事件和 focusOut 事件来调用 changeState 方法实现。

Code View:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="250"
height="40" top="10" currentState="display">
<mx:Script>
<! [CDATA [
[Bindable]
private var value:String;
private function changeState(event:Event = null):void
{
    if(this.currentState == "display")
    {
        currentState = "edit";
    }
}
```

```

        }
    else
    {
        value = editInput.text;
        currentState = "display";
    }
}
] ]>
</mx:Script>
<mx:states>
    <mx:State id="display" name="display">
        <mx:AddChild>
            <mx:Label text="{value}" id="text" x="{editorValue.x
                + editorValue.width}" click="changeState()"
                minWidth="100" minHeight="20"/>
        </mx:AddChild>
    </mx:State>
</mx:states>

```

当设置组件的currentState为编辑状态时，TextInput的文本会被设置到显示状态中的Label的值里。当用户按了回车键，组件的状态会返回到显示模式，同时从TextInput传过来的值会被设置到显示状态中的Label里。TextInput 组件的enter事件指明了在组件成为焦点的时候用户是按了回车还是返回键。

Code View:

```

<mx:State id="edit" name="edit">
    <mx:AddChild>
        <mx:TextInput
            creationComplete="editInput.setFocus()"
            focusOut="changeState()" id="editInput"
            x="{editorValue.x + editorValue.width}"
            text="{value}" minWidth="100" minHeight="20"
            enter="changeState()"/>
    </mx:AddChild>
</mx:State>
</mx:states>
<mx:Label text="Value: " id="editorValue"/>
</mx:Canvas>

```

## 4.5 节. 确定用户电脑上安装的所有字体

### 4.5.1. 问题

我想确定用户电脑上安装的所有字体，然后为 Text 组件选择列表中的一个字体显示。

### 4.5.2. 解决办法

使用 Font 类中的 enumerateFonts 方法设置一个带有选择好字体的 fontName 属性的 Text 组件的 fontFamily 风格

### 4.5.3. 讨论

Font 类定义了一个静态方法叫做 enumerateFonts 来返回一个用户电脑上所有字体的数组。这个方法返回包括 flash.text.Font 对象的数组。flash.text.Font 定义了三种属性。

#### fontName

这是一个系统报告的字体名称。在某些情况下，比如日语，韩语或者阿拉伯语，Flash Player 可能无法正常地渲染这些字体。

#### fontStyle

这是一个字体风格: Regular, Bold, Italic, or BoldItalic.

#### fontType

这个属性有两个选择。一个叫设备字体，意思是用户电脑上已经安装了的字体。另一个叫嵌入式字体，意思是把字体嵌入到 swf 文件中。

在下面的例子中，这些字体被传递给一个 ComboBox，用户可以为 Text area 选择一个字体类型。 setStyle 方法的调用

```
text.setStyle("fontFamily", (cb.selectedItem as Font).fontName);
```

设置了 Text 组件中的当前字体为 ComboBox 中 Font 对象的 fontName 属性所对应的字体。这里是你需要的完整代码:

Code View:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"  
height="300" creationComplete="findAllFonts()">>  
<mx:Script>  
<! [CDATA [
```

```

private var style:StyleSheet;
[Bindable]
private var arr:Array;
private function findAllFonts():void
{
    arr = Font.enumerateFonts(true);
    arr.sortOn("fontFamily", Array.CASEINSENSITIVE);
}
private function setFont():void
{
    text.setStyle("fontFamily", (cb.selectedItem as
        Font).fontName);
}
]]>
</mx:Script>
<mx:ComboBox id="cb" dataProvider="{arr}" change="setFont()"
    labelField="fontFamily"/>
<mx:Text text="Sample Text" id="text" fontSize="16"/>
</mx:VBox>

```

## 4.6 节. 创建一个自定义的 TextInput

### 4.6.1. 问题

我需要创建一个自定义的 TextInput 组件，包括可以输入文本的多行区域和绑定该组件的输出到一个 Text 显示。

### 4.6.2. 解决

使用 UIComponent 同时添加一个 flash.text.TextField 到这个组件中。然后设置一个可绑定的文本属性，将 Text 组件的 htmlText 属性绑定到新组件的文本。

### 4.6.3. 讨论

mx.controls.TextInput 组件限制访问到 flash.text.TextField 组件内部。如果需要对 TextField 进行更完全的访问和控制，简单的把一个 TextField 添加到 UIComponent 中。

提供用来访问 TextField 的任何方法都应该被定义在这个组件中。如果需要对 TextField 进行完全访问，把 TextField 当作一个公共属性定义可能更容易。然而，无论何时，当 TextField 的属性被访问或者被改变，采用第一种方法通知组件会比较方便。看一下例子：

Code View:

```
package oreilly.cookbook

{
    import flash.events.Event;
    import flash.events.TextEvent;
    import flash.text.TextField;
    import flash.text.TextFieldType;

    import mx.core.UIComponent;

    public class SpecialTextInput extends UIComponent
    {
        private var textInput:TextField;
        public static var TEXT_CHANGED:String = "textChanged";

        public function SpecialTextInput()
        {
            textInput = new TextField();
            textInput.multiline = true;
            textInput.wordWrap = true;
            textInput.type = flash.text.TextFieldType.INPUT;
            textInput.addEventListener(TextEvent.TEXT_INPUT,
                checkInput);
            addChild(textInput);
            super();
        }

        private function checkInput(textEvent:TextEvent):void
        {
            text = textInput.text;
        }

        override public function set height(value:Number):void
        {
            textInput.height = this.height;
            super.height = value;
        }

        override public function set width(value:Number):void
        {
            textInput.width = this.width;
            super.width = value;
        }
    }
}
```

```

[Bindable(event="textChanged")]
public function get text():String
{
    return textInput.text;
}

public function set text(value:String):void
{
    dispatchEvent(new Event("textChanged"));
}
}

```

要使用这个新组件，把 Text 的 htmlText 属性绑定到新组件的 text 属性上去：

Code View:

```

<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300" xmlns:cookbook="oreilly.cookbook.*">
<cookbook:SpecialTextInput id="htmlInput" height="200"
width="300"/>
<mx:TextArea htmlText="{htmlInput.text}" />
</mx:VBox>

```

## 4.7 节. 为一段文本设置风格属性

### 4.7.1. 问题

我想把一段非 HTML 文本设置字体信息。

### 4.7.2. 解决部分

使用 TextRange 类设置一段字符的属性。

### 4.7.3. 讨论

TextRange 类接受一个拥有 TextField 的组件。用一个参数指明是否通过 TextRange 中的属性设置来对该组件进行修改，然后用 2 个整数决定 TextField 中的起始字符和结束字符。

TextRange 对象是这样构造的：

Code View:

```
var textRange:TextRange = new TextRange(component:UIComponent, modify:Boolean,  
startIndex:int, endIndex:int);
```

TextRange 对象的属性影响了载入的组件。在下面的例子中，当选择了 check box 时，用户已选的文字区域的颜色和字符间隔会被重新设置。

Code View:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"  
height="300">  
    <mx:Script>  
        <! [CDATA [  
            import mx.controls.textClasses.TextRange;  
            private function alterTextSnapshot():void  
            {  
                var textRange:TextRange = new TextRange(area, true,  
                    area.selectionBeginIndex, area.selectionEndIndex);  
                textRange.color = 0xff0000;  
                textRange.letterSpacing = 3;  
            }  
        ]]>  
    </mx:Script>  
    <mx:CheckBox change="alterTextSnapshot()"/>  
    <mx:TextArea id="area" text="Lorem ipsum dolor sit amet,  
        consectetur adipiscing elit." width="200" height="50"/>  
</mx:VBox>
```

## 4.8 节. 在 HTML 里显示图片和 SWF 文档

### 4.8.1. 问题

我需要在你的 Flex 组件中显示一段包含图片和外部 SWF 文件的 HTML 文本。

### 4.8.2. 解决办法

使用 Flash 播放器中的 HTML 渲染引擎支持的标签，设置 src 属性为你要加载的 SWF 或图片的 URL 地址。

### 4.8.3. 讨论

<img>标签允许你指明将要加载到 Text 组件中显示的图片或 SWF 文件的地址。<img>标签支持如下这些属性：

#### src

指定一个 GIF, JPEG, PNG 或 SWF 文件的访问地址。这是唯一的必填属性。所有其他简单控制与文本相关的图片布局都在它周围。

#### align

指定文本域中的嵌入图片的水平对齐属性。有效值是 left 和 right。默认值是 left.

#### height

指定图片高度，单位为像素。

#### hspace

指定图片周围的水平空白区域大小。默认值是 8。

#### vspace

指定图片周围的垂直空白区域大小。默认值是 8。

#### width

指定图片宽度，单位为像素。

在下面的代码片断中，通过使用<src>HTML 标签，一个 SWF 文件被加载到应用程序中，然后把它的 vspace 属性设为 10，指名图片上下各有 10 像素的空白空间：

Code View:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
<mx:TextArea width="300" height="300" backgroundAlpha="0">
<mx:htmlText>
<![CDATA[
<img      src='../../assets/fries.jpg'      width='100'
height='100' align='left'
hspace='10' vspace='10'>
<p>This is the text that is going to appear above
the swf.</p><p>
<img      src='../../assets/test_swf.swf'      width='100'      height='100'
align='left' hspace='10'
vspace='10'>
Here is text that is going to be below the
```

```
image.</p>
    ]]>
</mx:htmlText>
</mx:TextArea>
</mx:Canvas>
```

## 4.9 节. 在一个搜索域中高亮显示用户输入

### 4.9.1. 问题

我想创建这样一个 TextArea，一个用户在 TextInput 中输入的文本可以在这个 TextArea 被搜索到同时高亮显示搜索到的内容。

### 4.9.2. 解决办法

使用 flash.text.TextField 对象并设置它的 alwaysShowSelection 属性为 true。然后用 setSelection 方法来设置被选文字的起始索引和长度。

### 4.9.3. 讨论

mx.controls.TextArea 组件需要将焦点聚在它的身上来显示文本的选择。要做到这个，你可以创建一个 TextArea 组件的子类，这样你就可以访问 TextArea 包含的 flash.text.TextField:

```
public function createComp():void{
    textField.alwaysShowSelection = true;
}
```

把 alwayShowSelection 属性设置为 true 意味着不管是否在焦点上，TextField 都会显示选择了的文本。现在每当 setSelection 方法被调用时，TextArea 组件中的 TextField 都会显示，同时 TextArea 的滑动条会自动滚动到正确的位置来显示选择的文本。

Code View:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="1000"
height="800"
xmlns:cookbook="oreilly.cookbook.*">
<mx:Script>
<! [CDATA[
[Bindable]
private var text_string:String = "Aenean quis nunc id
purus pharetra hac etra. Cras a felis sit amet ipsum
```

```

ornare      luctus.      Nullam      scelerisque"      +
" placerat velit. Pellentesque ut arcu congue risus
facilisis pellentesque. Duis in enim. Mauris eget
est. Quisque tortor. ";

private function searchText():void
{
    var index:int = masterText.text.indexOf(input.text);
    masterText.verticalScrollPosition = 0;
    if(index != -1)
    {
        masterText.setSelection(index,
            index+input.text.length);
    }
}
] ]>
</mx:Script>
<mx:TextInput id="input" change="searchText()"/>
<cookbook:SpecialTextArea editable="false" id="masterText"
    text="{text_string}" fontSize="20" width="600" height="200"
    x="200"/>
</mx:Canvas>

```

## 4.10 节. 把字符当作独立图像处理

### 4.10.1. 问题

我想把独立的字符当作图像一样进行特效处理。

### 4.10.2. 解决办法

使 `getCharBoundaries` 方法来取得 `TextField` 中字符的实际宽高 `xy` 值。然后从 `TextField` 创建一个包含你想要的字符的位图。

### 4.10.3. 讨论

`getCharBoundaries` 返回一个形容 `TextField` 中字符索引的宽高 `xy` 值的矩形。你可以用这个信息创建一个字符的位图，保留它的全部可视信息，并使这些位图看上去很真实。下面这段代码就是处理的关键：

Code View:

```
char_bounds = addCharacters.getTextField().getCharBoundaries(i);
```

```

bitmapData=new BitmapData(char_bounds.width, char_bounds.height, true, 0);
matrix = new Matrix(1, 0, 0, 1, -char_bounds.x, char_bounds.y);
bitmapData.draw(addCharacters.getTextField(), matrix, null, null, null, true);
bitmap = new Bitmap(bitmapData);

```

char\_bounds 对象是一个将要储存所有这个字符相关信息的矩形。这个信息用来在创建位图时创建一个 flash.display.BitmapData 对象以便显示在位图内。BitmapData 的构造器接收 4 个参数:

Code View:

```
BitmapData(width:Number, height:Number, transparency:boolean,      fillColor:Number);
```

现在你拥有了这个位图对象，创建一个 Matrix 来储存那个你想捕捉的 TextField 中特殊区域的信息，也就是，getCharBoundaries 方法返回的绑定了 TextField 的矩形区域。这个 Matrix 被传给 BitmapData draw 方法，这个方法同时也从载入的 DisplayObject 中提取像素数据。当你画完实际的位图数据后，通过使用最新最流行的 BitmapData 对象，你就可以创建一个 Bitmap 对象，这个对象也同时是一个 DisplayObject，可以被添加到显示列表中。

The 剩下的例子就是处理 TextField 中的字符循环了，对每一个字符都执行前面的操作，然后一个又一个栩栩如生的位图对象就被创建了：

Code View:

```

<mx:Canvas  xmlns:mx="http://www.adobe.com/2006/mxml"  width="600"
height="300"  xmlns:cookbook="oreilly.cookbook.*">
<mx:Script>
<! [CDATA[
import flash.display.Sprite;
import mx.core.UIComponent;
import flash.text.TextField;
import flash.text.TextFormat;

private var characterArray:Array = new Array();
//set up our master character index we're going to use
when animating the characters
private var charIndex:int = 0;
//keep track of the final position we're going to place
all the characters in
private var finalX:Number = 0;

private function addNewCharacters():void
{
    render();
    invalidateDisplayList();
    invalidateProperties();
}

```

```

public function render():void
{
    //define all the variables that we'll need
    var bitmapData:BitmapData;
    var bitmap:Bitmap;
    var char_bounds:Rectangle;
    var matrix:Matrix;
    var component:UIComponent;
    //get the text format and set the
    //tf.defaultTextFormat = addCharacters.getTextField().defaultTextFormat;
    //tf.text = addCharacters.text;
    for each(component in characterArray)
    {
        holder.removeChild(component);
    }
    characterArray.length = 0;
    for(var i:int=0; i<addCharacters.text.length; i++)
    {
        char_bounds =
addCharacters.getTextField().getCharBoundaries(i);
        bitmapData=new
BitmapData(char_bounds.width,char_bounds.height,
        true,0);
        matrix = new Matrix(1,0,0,1,-
            char_bounds.x,char_bounds.y);
        bitmapData.draw(addCharacters.getTextField(),
            matrix, null, null,null, true);
        bitmap = new Bitmap(bitmapData);
        component = new UIComponent();
        component.width = bitmapData.width;
        component.height = bitmapData.height;
        component.addChild(bitmap);
        holder.addChild(component);
        component.x=char_bounds.x;
        characterArray[i] = component;
    }
    holder.invalidateDisplayList();
    startEffect();
}

private function startEffect():void
{

```

```

        addEventListener(Event.ENTER_FRAME, moveCharacter);
    }

private function moveCharacter(event:Event):void
{
    var comp:UIComponent = (characterArray[charIndex] as
        UIComponent);
    if(comp)
    {
        if(comp.x < 200 - finalX)
        {
            (characterArray[charIndex] as Sprite).x+=2;
        }
        else
        {
            if(charIndex == characterArray.length - 1)
            {
                removeEventListener(Event.ENTER_FRAME,
                    moveCharacter);
                return;
            }
            finalX += comp.width+2;
            charIndex++;
        }
    }
}
]]>
</mx:Script>
<mx:HBox>
    <cookbook:AccessibleTextInput id="addCharacters"
        fontSize="18"/>
    <mx:Button label="add characters"
        click="addNewCharacters()"/>
</mx:HBox>
<mx:Canvas id="holder" y="200"/>
</mx:Canvas>

```

这个例子使用一个 AccessibleTextInput 来控制，一个简单的 TextInput 控制器扩展，用来提供对 TextInput 中的 TextField 控制器的访问能力。：

```

<mx:TextInput xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <! [CDATA[
            public function getTextField():IUITextField

```

```

    {
        return this.textField;
    }
] ] >
</mx:Script>
</mx:TextInput>
```

这让你简单地把 TextFormat 对象传递给用来进行位图操作的 TextField，同时为 getCharBoundaries 方法直接从 TextField 读取。

## 4.11 节. 指定 **TextField** 中的 **HTML** 样式

### 4.11.1. 问题

我想通过外部加载的 CSS 文件或是写在程序里的 CSS 样式，来实现 TextField 中的 HTML 的 CSS 样式。

### 4.11.2. 解决办法

使用 StyleSheet parseStyle 方法来对 TextArea 进行字符串解析和样式表分配。

### 4.11.3. 讨论

StyleSheet 对象可以把任何正确的 CSS 当成字符串解析，处理它的所有信息，把它分配到一个组件或是在 HTML 文本中应用。要实现这个，你需要用一个 URLLoader 对象加载文件，把加载的数据当作字符串传递给 parseCSS 方法，也可以把字符串写出来直接传给 StyleSheet 对象。下面的例子就是把样式写成字符串，当 creationComplete 事件发生时，就将它解析。TextArea 的 htmlText 属性在样式被应用后就会进行设置，确保样式被完全应用到 HTML 里。使用如下样式属性会<span>的样式：

```
"<span class='largered'>
```

This style is specified in the string that will be passed to the StyleSheet.parseStyle method:

```
.largered { font-family:Arial, Helvetica; font-size:16; color: #ff0000; }
```

例子：

Code View:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
width="1000" height="800" creationComplete="createStyle()">
```

```

<mx:Script>
<! [CDATA[
    //note even though names are camel cased here when used,
    all lowercase
    private var styleText:String = '.largered { font-
family:Arial, Helvetica;
font-size:16; color: #ff0000; }' +
        '.smallblue { font-size: 11; color: #0000ff;
font-family:Times New
Roman, Times; }' +
            'li { color:#00ff00; font-size:14; }' +
            'ul {margin-left:50px;}';

    [Bindable]
    private var lipsum:String = "<span
class='largered'>Nulla metus.</span>
Nam ut dolor vitae risus condimentum auctor."+
        " <span class='smallblue'>Cras sem quam,</span>
malesuada eu, faucibus
aliquam, dictum ac, dui. Nullam blandit"+
        " ligula sed arcu. Fusce nec est.<ul><li>
Etiam</li><li> aliquet,</li>
<li>nunc</li></ul> eget pharetra dictum, magna"+
        " leo suscipit pede, in tempus erat arcu et velit.
Aenean condimentum.
Nunc auctor"+

        " nulla vitae velit imperdiet gravida";

```

```

    [Bindable]
    private var style:StyleSheet;

    private function createStyle():void
    {
        style = new StyleSheet();
        style.parseCSS(styleText);
        text.styleSheet = style;
        text.htmlText = lipsum;
    }
] ]>
</mx:Script>
<mx:TextArea id="text" width="200" height="300"/>
</mx:Application>

```

## 4.12 节. 使用 RichTextEditor

### 4.12.1. 问题

我想要创建一个可以让用户输入富文本的组件,然后使用富文本创建的 HTML。这个组件可以使用用户电脑上的所有字体。

### 4.12.2. 解决办法

创建一个 RichTextEditor, 然后从它的控制器里面读取 htmlText 属性。设置定义在 RichTextEditor 里的 fontFamilyCombo 的 dataProvider 来添加从 Font.enumerateFonts 返回的所有结果。

### 4.12.3. 讨论

RichTextEditor 相当的方便, 允许用户创建 HTML 文本, 通过使用 htmlText 属性, 就可以从编辑器将文本读出。RichTextEditor 包含了多个按钮来设置文本的粗体, 斜体, 或是下划线, 一个 ComboBox 来设置字体, 和一个 ColorPicker 来设置已选择的文字的颜色, 这些都可以从 RichTextEditor 中访问。下面的例子通过访问 fontFamilyCombo ComboBox 来添加所有用户安装了的字体以便用户从中选择一个使用。

要访问用户创建的文本的所有属性, 请使用 RichTextEditor 的 htmlText 属性。这个属性既可以获取也可以设置, 因此如果你想在编辑器中使用 HTML, 简单地把一段正确的 HTML 字符串赋给 htmlText 属性就可以了。

Code View:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="900"
height="500"
creationComplete="addFonts()">
<mx:Script>
<! [CDATA[
import mx.collections.ArrayCollection;
private function addFonts():void
{
    var arr:Array = Font.enumerateFonts(true);
    richText.fontFamilyCombo.labelField = 'fontName';
    richText.fontFamilyCombo.dataProvider =
        Font.enumerateFonts(true);
}
]]>
</mx:Script>
<mx:RichTextEditor id="richText" width="400" height="400"
change="trace(richText.htmlText+' '+richText.text)"/>
<mx:TextArea height="100%" width="400"
htmlText="{richText.htmlText}" x="410"/>
</mx:Canvas>
```

## 4.13 节. 在 HTML 中应用嵌入字体

### 4.13.1. 问题

我想在一个 HTML 文本中使用一个嵌入字体。

### 4.13.2. 解决办法

使用一个样式中的@font-face 标签来嵌入字体，然后使用<font> 标签来设置这个标签的 family 属性。

### 4.13.3. 讨论

在 HTML 文本中应用嵌入字体要比使用系统字体复杂的多。应用字体的标准方法是简单的设置一个样式中的 font-family 属性中的 font，然后把样式应用在一个 span 上。然而，嵌入字体要求 HTML 中的 font 标签必须应用应用嵌入字体。

```
<font size="20" family="DIN">Using the new font</font>
```

这个 font 标签通过将 fontFamily 属性插入到<mx:Style>标签中的 font-face 声明中来使用。

```
@font-face{
    src:url("../assets/DIN-BLAC.ttf");
    fontFamily:DIN;
    advancedAntiAliasing: true;
}
```

下面是完整的例子：

Code View:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute">
    <mx:Style>
        @font-face{
            src:url("../assets/DIN-BLAC.ttf");
            fontFamily:DIN;
            advancedAntiAliasing: true;
        }
        .embeddedFontStyle{
            fontFamily:DIN;
            fontSize:18px;
            color:#CCCCFF;
```

```

        }
        .white{
            color:#ffffff;
        }
    </mx:Style>
<mx:VBox backgroundColor="#000000">
    <mx:Label text="This is some test text"
        styleName="embeddedFontStyle"/>
    <mx:TextArea id="ta" backgroundAlpha="0" width="250"
        height="150" styleName="white">
        <mx:htmlText>
            <![CDATA[
                Not      Using      the      New      Font.<font      size="20"
family="DIN">Using the new
font</font>
            ]]>
        </mx:htmlText>
    </mx:TextArea>
</mx:VBox>
</mx:Application>

```

## 4.14 节. 给一个 **Text** 组件中的文本添加拖拽阴影

### 4.14.1. 问题

我想给 TextArea 组件中的当前文字添加一个拖拽阴影。

### 4.14.2. 解决办法

使用 BitmapData 对象来获取一份 TextField 的拷贝，同时添加这个位图到父组件的一个偏移点来模拟阴影

### 4.14.3. 讨论

当你尝试显示一个 TextArea 或者 Text 组件中的当前内容的阴影图像，你只不过需要取得可以再现文本域中所有信息的位图，然后添加到父组件中就行了。移动图像稍微偏离中心同时降低透明度值来降低其亮度，这样就提供了正常的视觉效果。既然这样基于 UIComponent 类来创建一个自定义组件以便减少开发进程，让你从 flash.display 包中直接读取和添加 ActionScript 底层显示组件。The 取得位图数据及添加到位图的功能已经在 4.10 节描述过了。

```
package oreilly.cookbook
```

```

{
    import flash.display.Bitmap;
    import flash.display.BitmapData;
    import flash.events.TextEvent;
    import flash.text.TextField;

    import mx.core.UIComponent;

    public class TextDropShadow extends UIComponent
    {

        private var _useShadow:Boolean = false;
        private var _shadowHolder:Bitmap;
        private var _bitmapData:BitmapData;
        private var _textField:TextField;
    }
}

```

这里，位图已经创建而且部署到了父组件的一个偏移点，来模拟一个阴影：

Code View:

```

public function TextDropShadow()
{
    super();
    _shadowHolder = new Bitmap();
    addChild(_shadowHolder);
    _shadowHolder.x = 5;
    _shadowHolder.y = 5;
    _textField = new TextField();
    _textField.type = "input";
    _textField.addEventListener(TextEvent.TEXT_INPUT,
    inputListener);
    addChild(_textField);
}

```

updateDisplayList 方法是一个覆写函数，用来绘出 TextField 及其相关所有可视信息，包括文本和位图。

Code View:

```

override protected function
updateDisplayList(unscaledWidth:Number,
    unscaledHeight:Number):void
{
    super.updateDisplayList(unscaledWidth, unscaledHeight);
    if(_useShadow)
    {
}

```

```

        _bitmapData = new BitmapData(_textField.width,
            _textField.height, true);
        _bitmapData.draw(_textField);
        _shadowHolder.bitmapData = _bitmapData;
        _shadowHolder.alpha = 0.7;
    }
}

private function inputListener(event:TextEvent):void
{
    invalidateDisplayList();
}

public function set useShadow(value:Boolean):void
{
    _useShadow = value;
}

public function get useShadow():Boolean
{
    return _useShadow;
}
}

```

## 4.15 节. 找出一个 **TextArea** 中最后显示的字符

### 4.15.1. 问题

我想找出一个 **TextArea** 中最后显示的字符。

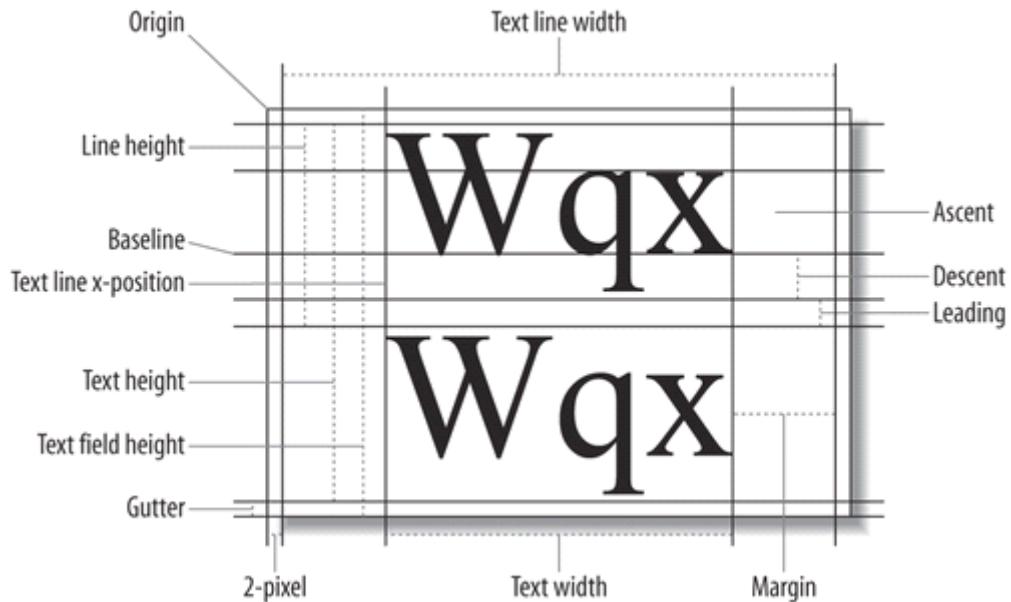
### 4.15.2. 解决办法

使用 **TextField** 的绑定方法来返回 **TextArea** 的大小，然后使用 **getLineMetrics** 方法确定实际行高。然后决定最后一个可视行，使用 **getLineOffset** 和 **getLineLength** 方法找出最后一个可视行中的最后一个字符。

### 4.15.3. 讨论

一个 TextField 中的每一行都有它自己特征的属性并传给 getLineMetrics 方法,同时使用 TextLineMetrics 对象来返回。TextLineMetrics 定义了一行文本的多个属性:高度, 宽度, 基线和行间距。图 4-1 描述了 TextLineMetrics 对象定义的属性。

图 4-1. TextLineMetrics 对象的属性



下面的例子使用了高度属性来找出 TextArea 中最后显示的行。首先, 使用 getBounds 方法取回一个关于可视 TextArea 的宽高 xy 值得矩形。然后使用 TextLineMetrics 对象取得最后一个可视行, 然后添加它。最后, 使用 getLineOffset 方法获取这一行的第一个字符索引, getLineOffset 可以取得一行的第一个字符索引, 把它与这一行的总长度相加就得到了最后一个字符:

Code View:

```
changeArea.text.charAt(changeArea.getLineOffset(i-1)+changeArea.getLineLength(i-1))
```

下面是完整的例子:

Code View:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300" xmlns:cookbook="oreilly.cookbook.*">
<mx:Script>
<! [CDATA[
    private function findLineMetrics():void
    {
        if (changeField.text.length > 1)
        {
            var rect:Rectangle = changeArea.getBounds(this);
            var visibleTextHeight:Number = 0;
            var i:int = 0;
```

```
while(visibleTextHeight < rect.height && i <
    changeArea.numLines)
{
    var metrics:TextLineMetrics =
        changeArea.getLineMetrics(i);

    visibleTextHeight+=metrics.ascent+metrics.height;
    i++;
}

trace(changeArea.text.charAt(changeArea.getLineOffset(i-1)+
changeArea.getLineLength(i-1)));
}

]

]]>
</mx:Script>
<mx:TextInput id="changeField" width="200"
    textInput="findLineMetrics()"/>
<cookbook:SpecialTextArea id="changeArea"
    text="{changeField.text}" wordWrap="true" width="150"
    height="30" y="100"/>
</mx:Canvas>
```

# 第五章 Lists, Tiles, 和 Trees (桃之夭夭)

本章介绍的这三种组件均继承自 `mx.controls.listclasses.ListBase` 类，并且为这些数据驱动的控件工作的方法的执行和使用过程中都是一致的，每一个数据驱动控件都有自己的由数据提供者生成的孩子结点用来排序，过滤，重排，并且这个可视组件的设置效果如条目渲染器（官方将 item 翻成“项目”，个人认为翻成条目可能更贴切一些）和条目编辑器操作。这些控件也允许拖曳。

本章所讲的内容主要是怎样使用条目渲染器，控制选项，设置 `ListBase` 控件的风格。并使用数据提供者为不同的控件类型工作，这并不是说我们将一直专注于怎样使用 `ListBase` 组件这些话题。也有许多内容介绍怎么样使用拖曳（见第十章：“Dragging and Dropping.”）介绍怎样使用皮肤。（见第九章：“Skinning and Styling.”）

## 5.1 节. 创建可编辑的 list 组件

### 5.1.1 问题

创建一个所有条目都可以编辑的 list 组件

### 5.1.2 解决方法

将 list 组件的 `editable` 属性设置为 `true` 并侦听 `itemEditBegin` 和 `itemEditEnd` 属性。或通过包含 `columnIndex` 和 `rowIndex` 属性的对象来设置 `editedItemPosition`。

### 5.1.3 讨论

所有的 List 组件都可以通过简单地设置 list 的 `editable` 属性为 `true` 使之可编辑。这就意味着每一个渲染都将变成一个由 `TextInput` 模块控制和用户选择的 `itemRenderer` 填充值的编辑器，List 类同样定义了一些当用户开始和结束修改正在编辑的值的时候用来通知应用程序的事件：

**itemEditBegin:** 当 `editedItemPosition` 属性已设置且项目可编辑后调度。当事件被触发，List 组件使用 `createItemEditor` 方法并且从编辑器拷贝 `data` 属性来创建一个 `itemEditor` 对象

**itemEditBeginning:** 当用户准备好编辑项目（例如，在项目上释放鼠标按键）后调度，聚焦到列表或试图编辑列表

**itemEditEnd:** 当项目编辑会话因任何原因而结束时调度。List 组件为这个拷贝自条目编辑器至 list 组件的数据提供者的数据提供了一个默认的事件处理方式，默认情况下，List 组件使用 `editorDataField` 属性来决定 `itemEditor` 的属性包括新数据和使用新数据更新条目数据。

我们将在 [Chapter 7: "Renderers and Editors."](#) 中更加地详细讨论使用这些不同的事件并且控制由 `itemEditor` 返回的数据

这个可编辑的条目可由用户点击 List 中的一个 `itemRenderer` 或设定 List 组件的带有 `columnIndex` 和 `rowIndex` 属性的 `editedItemPosition` 对象来设置，在 List 中，`columnIndex` 常常是 0，而 `rowIndex` 则是编辑器将要创建的行的位置，例如：

```
listImpl.editedItemPosition = {columnIndex:0, rowIndex:2};
```

完整例子如下：

代码：

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
<mx:Script>
<! [CDATA[
[Bindable]
private var dp:Array = [{name:"John Smith", foo:"bar"}, 
{name:"Ellen Smith", foo:"baz"}, 
{name:"James Smith", foo:"eggs"}, 
{name:"Jane Smith", foo:"spam"}]
```

selectedItem属性返回用户在渲染器中做条目编辑后的没有再修改的值，

代码：

```
private function editEnd(event:Event):void {
trace(listImpl.selectedItem.foo+
'+'+listImpl.selectedItem.name);
}
```

在由 rowIndex 和 columnIndex 属性指示的位置设置 editedItemPosition 创建编辑器

代码：

```
private function setEditor():void {
listImpl.editedItemPosition = {columnIndex:0,
rowIndex:2};
}
]]>
</mx:Script>
<mx:Button click="setEditor()"/>
<mx>List y="30" width="200" selectedIndex="6" id="listImpl"
selectionColor="#CCCCFF" labelField="name"
dataProvider="{dp}" editable="true"
itemEditBegin="trace(listImpl.editedItemPosition)"
itemEditEnd="editEnd(event)" editorXOffset="5"
editorYOffset="2"/>
</mx:Canvas>
```

## 5.2 节. 为 List 的某项设置图标

### 5.2.1 问题

用 list 提供的数据为以 itemRenderer 为基础的 List 控件设置图标

## 5.2.2 解决方法

使用 List 组件的 `iconFunction` 属性，并创建一个方法用来返回以类的型式嵌入的图片

## 5.2.3 讨论

`iconFunction` 是一个描述嵌入图片并返回类对象的方法，嵌入的图片由 `list` 组件中的 `itemRenderer` 使用，`iconFunction` 的用法是简单地定义的方法名传递，例如：`setIcon` 是一个定义来接收值的方法名；

`iconFunction="setIcon"`

`iconFunction` 常常传递 `dataProvider` 的索引中对象的值

`private function setIcon(value:*):Class`

在 `iconFunction` 方法中，`dataProvider` 的索引中数据对象的所有信息都会被传递，`iconFunction` 返回将被渲染器创建的图标的类对象，完整的示例如下：

代码：

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
<mx:Script>
<! [CDATA[
[Bindable]
private var dp:Array = [{name:"John Smith",
position:"developer"}, {name:"Ellen Smith",
position:"manager"}, {name:"James Smith",
position:"accountant"}, {name:"Jane Smith",
position:"designer"}];

[Embed(source="../assets/manager.png")]
private var managerIcon:Class;

[Embed(source="../assets/designer.png")]
private var designerIcon:Class;

[Embed(source="../assets/accountant.png")]
private var accountantIcon:Class;

[Embed(source="../assets/developer.png")]
private var developerIcon:Class;

private function setIcon(value:*):Class
{
    if(value.position != null)
    {
        switch(value.position)
        {
```

```

        case "developer":
            return developerIcon;
        break;
        case "designer":
            return designerIcon;
        break;
        case "accountant":
            return accountantIcon;
        break;
        case "manager":
            return managerIcon;
        break;
    }
}
return null;
}

]]>
</mx:Script>
<mx>List width="200" selectedIndex="6" id="listImpl"
selectionColor="#CCCCFF" labelField="name"
dataProvider="{dp}" editable="true"
iconFunction="setIcon"/>
</mx:Canvas>

```

## 5.3 节. 为 List 的内容变更添加特效

### 5.3.1 问题

当 list 的数据被改变的时候为 list 添加显示效果

### 5.3.2 解决方法

创建一个效果序列并将它们传递给 List 组件的 `itemsChangeEffect` 属性

### 5.3.3 讨论

数据改变时效果是 flex 3 新增的很强大的效果,对于之前的版本,你可以写数据改变时效果并分发和注册事件和事件侦听,但是伴随着 flex 3 新增 `itemsChangeEffect` 属性, List 组件和任何继承 `ListBase` 的类都可以在它的 `dataProvider` 改变时分发事件,然后该事件触发所有传递给 List 中新的 `itemsChangeEffect` 属性的效果或效果序列.

由于 `dataChange` 事件是由 List 的 `dataProvider` 触发的, 设置一个类似 List 组件的 `dataProvider` 的数组意味着该数组被改变时 `itemsChangeEffect` 事件不会被分发。记住,事件被改变时不分发事件,对于  `ArrayCollection` 类,当事件被改变时仍然会分发事件,类似继承自 `EventDispatcher` 并设置成分发事件一样,当一个设置方法被调用来改变潜在的数

组对象的值

简单地将 itemsChangeEffect 设置为一个 2 秒淡出的 DefaultListEffect 的实例，下面这个例子是当 List 被改变时应用一个 mx.effects.Glow 实例的效果：

代码：

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="900" top="20" left="20">
    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;
            //note that for this example to work, the dataprovider
            //must be an array collection
            [Bindable]
            private var dp:ArrayCollection =
                new ArrayCollection([{name:"John Smith",
                    position:"developer"}, {name:"Ellen Smith",
                    position:"manager"}, {name:"James Smith",
                    position:"accountant"}, {name:"Jane Smith",
                    position:"designer"}]);

            private function addItem():void
            {
                dp.addItem({name:"Jim Smith", position:"Janitor"});
            }
        ]]>
    </mx:Script>
    <mx:DefaultListEffect color="0xccccff" fadeOutDuration="2000"
        id="glow"/>
    <mx>List width="300" itemsChangeEffect="{glow}"
        dataProvider="{dp}" editable="true" labelField="name"/>
    <mx:Button click="addItem()" label="add item"/>
    <mx>List width="300" itemsChangeEffect="{glow}"
        dataProvider="{dp}" editable="true" labelField="name"/>
</mx:VBox>
```

另一个关于 itemsChangeEffect 的例子，这是一个使用 Sequence 标签来创建一个当 List 组件的数据被改变时就触发的事件序列：

代码：

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="900" top="20" left="20">
    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;
            import mx.effects.easing.*;
```

```

[Bindable]
private var dp:ArrayCollection =
    new ArrayCollection([{name:"John Smith",
        position:"developer"}, {name:"Ellen Smith",
        position:"manager"}, {name:"James Smith",
        position:"accountant"}, {name:"Jane Smith",
        position:"designer"}]);

]]>
</mx:Script>
<mx:Sequence id="itemsChangeEffect">
    <mx:WipeDown duration="500"/>
    <mx:Parallel>
        <mx:Move
            duration="750"
            easingFunction="{Elastic.easeOut}"
            perElementOffset="20"/>
        <mx:RemoveItemAction
            startDelay="400"
            filter="removeItem"/>
        <mx:AddItemAction
            startDelay="400"
            filter="addItem">
            </mx:AddItemAction>
        <mx:Blur
            startDelay="410"
            blurXFrom="18" blurYFrom="18" blurXTo="0"
            blurYTo="0" duration="300"
            filter="addItem"/>
    </mx:Parallel>
</mx:Sequence>
<mx>List width="300" itemsChangeEffect="{itemsChangeEffect}"
   dataProvider="{dp}" editable="true" labelField="name"/>
<mx:Button click="{dp.addItem({name:'Jim Smith',"
    position:'Janitor'});}" label="add item"/>
<mx:Button click="{dp.removeItemAt(3)}" label="remove item"/>
<mx:Button click="{dp.setItemAt(dp.getItemAt(1), 2)}"
    label="change item"/>
</mx:VBox>

```

## 5.4 为 TileList 创建一个基本的条目渲染器

### 5.4.1 问题

为 TileList 类设置一个定制的 itemRenderer，当数据从 TileList 传递给渲染器时它将附一个特定的依赖于代码的图象

### 5.4.2 解决方法

当数是为渲染器设置时创建一个 VBox 对象并且覆盖 `set data` 方法来从杂乱数据中读取相匹配的图象来编码传递

### 5.4.3 讨论

一个 List 的条目渲染器传递一个描述当前表中每一个条目的数据对象，它用来创建渲染器中详细的列和行，所有定制数据处理都需要在 item renderer 中完成，它将在为数据属性设置方法时发生，所以 item renderer 将会与父级列表同步

对于下面这个例子，一个图象列表以静态公共引用形式存放在单独的文件来存取并且被引用到这里的一个不规则列表来与 positionType 的值作比较以使这个参数值传递到数据属性，代码：

```
<mx:VBox      xmlns:mx="http://www.adobe.com/2006/mxml"      width="400"
height="300">
<mx:Script>
<! [CDATA[
[Bindable]
private var type:String;

[Bindable]
private var imageClass:Class;

private var typeToPositionHash:Object =
{1:"Manager",2:"Accountant",
3:"Designer", 4:"Developer"};
private var typeToImageHash:Object =
{1:Assets.managerIcon, 2:Assets.accountantIcon,
3:Assets.designerIcon, 4:Assets.developerIcon};

override public function set data(value:Object):void {
type = typeToPositionHash[value.positionType];
imageClass = typeToImageHash[value.positionType];
nameText.text = value.name;
}

]]>
</mx:Script>
<mx:Text text="{type}" />
<mx:Image source="{imageClass}" />
<mx:Text id="nameText" fontSize="16" />
</mx:VBox>
```

这里使用 itemRenderer 显示，简单地通过 List 组件的一个有效类名的引用，注意如果 dataProvider 对象没有一个 positionType 属性，itemRenderer 就不会按照期望的效果运行，在大型应用程序中，所有这些数据类型都表现为强类型数据对象以确保不会引发任何问题。代码：

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
<mx:Script>
<! [CDATA[
import oreilly.cookbook.SimpleRenderer;
import mx.collections.ArrayCollection;

[Bindable]
private var dp:ArrayCollection =
new ArrayCollection([{name:"John Smith",
positionType:1}, {name:"Ellen Smith",
positionType:2}, {name:"James Smith",
positionType:3}, {name:"Jane Smith",
positionType:4}]);

]]>
</mx:Script>
<mx>List itemRenderer="oreilly.cookbook.SimpleRenderer"
dataProvider="{dp}" />
</mx:Canvas>
```

## 5.5 节. 为 Tree 设置 XML 数据

### 5.5.1 问题

用一个 Tree 组件来表现由外部载入的 XML 数据

### 5.5.2 解决方法

以 e4x 标准为 HTTPService 对象设置类型并且加载指定 XML 文件，设置请求结果为 Tree 的数据提供者，使用像 labelField 这样的 Tree 叶子结点，通过 XML 语法来传递结点属性，确保 Tree 将会显示正确的标签

### 5.5.3 讨论

只要 Tree 组件的 labelField 属性显示正确的属性值，Tree 组件处理 XML 数据就是正常的，例如，label 属性一个 label 调用的属性值：

```
<data label="2004">
<result label="Jan-04">
```

```

<product label="apple">81156</product>
<product label="orange">58883</product>
<product label="grape">49280</product>
</result>
<result label="Feb-04">
    <product label="apple">81156</product>
    <product label="orange">58883</product>
    <product label="grape">49280</product>
</result>
</data>

```

以 E4X 语法设置 labelField 属性来显示你需要提供给标签用的指定属性值或属性:

`labelField="@label"`

这个使用属性值调用标签的显示结果存在于每一个 XML 结点, 默认情况下, 如果没有 labelField 被提供, Tree 组件显示的结点值, 这是一个完整的例子:

代码:

```

<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300" left="20" top="10">
    <mx:HTTPService id="xmlLoader" url="assets/data.xml"
        resultFormat="e4x" contentType="application/xml"
        result="xmlReturned(event)"/>
    <mx:Script>
        <![CDATA[
            import mx.collections.XMLListCollection;

            [Bindable]
            private var xmlValue:XMLListCollection;

            private function xmlReturned(event:Event):void
            {
                xmlValue = new XMLListCollection(
                    new XMLList(xmlLoader.lastResult));
            }
        ]]>
    </mx:Script>
    <mx:Button click="xmlLoader.send()" label="get XML"/>
    <mx:Tree dataProvider="{xmlValue}" y="100" labelField="@label"
        width="300" showRoot="true"/>
</mx:Canvas>

```

## 5.6 节. 为 Tree 创建项渲染器

### 5.6.1. 问题

为 Tree 组件更改功能

### 5.6.2. 解决方法

创建一个 `itemRenderer` 继承自 TreeItemRenderer 类

### 5.6.3 讨论

更改一个 Tree 组件功能使之比其它基于 list 的组件更灵活,与 DataGrid, TileList, 或 List 组件不同的是,你不能为 Tree 组件使用 dropInItemRenderer, 只能继承 TreeItemRenderer 类来创建一个 itemRenderer, TreeItemRenderer 为 Tree 组件定义了默认的条目渲染器, TreeItemRenderer 的默认行为是在 tree 中绘制关联的每一个条目的文本,一个可选择的图标, 和一个可选的可定义图标.

通过这一节使用 TreeListData 对象的父级传递 TreeItemRenderer, TreeListData 定义了以下属性:

`depth : int`

tree 中条目的层级

`disclosureIcon : Class`

为 Tree 组件的条目描绘可定义的图标的一个类

`hasChildren : Boolean`

若该结点有孩子结点时为 true

`icon : Class`

为 Tree 组件的条目描绘图标的一个类

`indent : int`

Tree 组件的当前行的默认缩排

`item : Object`

Tree 组件的当前条目的数据

`label : String`

条目数据的文本表现, 基于 List 类的 `itemToLabel` 方法

`open : Boolean`

结点开放时为 true

下面这个例子使用了这个方法来改变子文本为紫色和粗体. 它同样为每一个目录增加了一些文本以显示这个特别的分支中有多少对象。

```
package oreilly.cookbook

{
    import mx.controls.treeClasses.*;
    import mx.collections.*;

    public class CustomTreeItemRenderer extends TreeItemRenderer
    {
        public function CustomTreeItemRenderer() {
            super();
            mouseEnabled = false;
        }
    }
}
```

TreeItemRenderer 中的 listData 属性涉及到 Tree 对象父级的数据, 它是用来判断当前渲染器数据对象包含的所有孩子

```
override public function set data(value:Object):void {
    if(value != null) {
        super.data = value;
        if(TreeListData(super.listData).hasChildren) {
            setStyle("color", 0x660099);
            setStyle("fontWeight", 'bold');
        } else {
            setStyle("color", 0x000000);
            setStyle("fontWeight", 'normal');
        }
    }
}
```

updateDisplayList 方法是被覆盖来检查当前渲染器传递的结点是否占有父级 Tree 中 TreeListData 正在使用的所有孩子

代码:

```
override protected function
updateDisplayList(unscaledWidth:Number,
    unscaledHeight:Number):void {
    super.updateDisplayList(unscaledWidth, unscaledHeight);
    if(super.data)
    {
        if(TreeListData(super.listData).hasChildren)
```

```

    {
        var tmp:XMLList = new
            XMLList(TreeListData(super.listData).item);
        var myStr:int = tmp[0].children().length();
        super.label.text =
            TreeListData(super.listData).label + "(" +
            myStr + " objects)";
    }
}
}
}
}

```

## 5.7 节. 在 Tree 控件中使用复杂数据对象

### 5.7.1 问题

为 Tree 控件传递复杂数据，并使用 Tree 适当地解析它们

### 5.7.2 解决方法

在一个类中实现 `ITreeDataDescriptor` 接口，并该类的新数据描述符中设置一个示例对象给 Tree 中 `dataDescriptor` 属性，

### 5.7.3 讨论

使用一个对象和 Tree 一起，将对象传递给 Tree 来实现 `ITreeDataDescriptor` 以解析数据并返回与数据中的对像关系相关的正确信息，XML 数据很容易将自己达到这个目的，但是复杂数据对象可以和为 Tree 定义关系的对象一起被适当的使用。

下面的这个例子，`ComplexDataType`，是一个典型的复杂数据类型：

代码：

```

package oreilly.cookbook
{
    import flash.events.Event;
    import flash.events.EventDispatcher;
    import flash.events.IEventDispatcher;

    public class ComplexDataType extends EventDispatcher
    {

        public static const NAME_CHANGED:String = "nameChanged";
        public static const OFFICE_CHANGED:String = "officeChanged";
        public static const RECORD_CHANGED:String = "recordChanged";
    }
}

```

```

private var _name:Name;
private var _office:Office;
private var _label:String;

public function
    ComplexDataType(target:IEventDispatcher=null)
{
    super(target);
}

[Bindable(RECORD_CHANGED)]
public function set label(value:String):void
{
    _label = value;
    dispatchEvent(new Event(RECORD_CHANGED));
}

public function get label():String
{
    return _label;
}

[Bindable(NAME_CHANGED)]
public function set name(value:Name):void
{
    _name = value;
    dispatchEvent(new Event(NAME_CHANGED));
}

public function get name():Name
{
    return _name;
}

[Bindable(OFFICE_CHANGED)]
public function set office(value:Office):void
{
    _office = value;
    dispatchEvent(new Event(OFFICE_CHANGED));
}

public function get office():Office
{
}

```

```
    return _office;
}
}

}
```

Office 和 Name 对象都是由 Office 对象的名字和地址组成的简单值对象，和 firstName, lastName 属性的 Name 对象。没有一个 ITreeDataDescriptor 对象来描述 ComplexDataType, Office 和 Name 所有属性之间的关系, Tree 将不能完全显示这些对象的 Array 和 ArrayCollection，而将显示 Array 中不能使扩展的两个简单对象。

ITreeDataDescriptor 接口定义了以下数据:

```
addChildAt(parent:Object, newChild:Object, index:int, model:Object = null):Boolean
```

这个方法插入了一个新的对象到结构中，并决定新加对象放在数据结构中的什么位置.

```
getChildren(node:Object, model:Object = null):ICollectionView
```

对于所有的结点，这个方法决定这个结点或对象是否有孩子结点可以显示，如果有孩子结点，该方法将它们返回到 CollectionView，如果没有，则返回空.

```
getData(node:Object, model:Object = null):Object
```

这个方法返回给定结点的所有数据.

```
hasChildren(node:Object, model:Object = null):Boolean
```

这个方法决定该结点是否有孩子结点并返回一个布尔值.

```
isBranch(node:Object, model:Object = null):Boolean
```

这个方法测试该结点是否是最终结点，并判断其孩子结点是否能被显示.

```
removeChildAt(parent:Object, child:Object, index:int, model:Object = null):Boolean
```

这个方法从结构中移除一个对象并决定数据结构中结点被移除后将会发生什么

ComplexDataType 的数据描述符正确执行如下:

代码:

```
package oreilly.cookbook
{
    import mx.collections.ArrayCollection;
```

```

import mx.collections.ICollectionView;
import mx.controls.treeClasses.ITreeDataDescriptor;

public class ComplexDataCollection implements
    ITreeDataDescriptor
{
    public function getChildren(node:Object,
        model:Object=null):ICollectionView
    {
        var col:ArrayCollection;
        if(node is Office){
            col = new ArrayCollection([node.officeAddress,
                node.officeName]);
            return col;
        }
        if(node is Name){
            col = new ArrayCollection([node.firstName,
                node.lastName]);
            return col;
        }
        if(node is ComplexDataType){
            col = new ArrayCollection([node.office, node.name]);
            return col;
        }
        return null;
    }

    public function hasChildren(node:Object,
        model:Object=null):Boolean
    {
        if(node is Office){
            if(node.officeAddress != "" && node.officeName != "") {
                return true;
            } else {
                return false;
            }
        }
        if(node is Name){
            if(node.firstName != "" && node.firstName != "") {
                return true;
            } else {
                return false;
            }
        }
    }
}

```

```

        }

    if(node is ComplexDataType) {
        if(node.office != null && node.name != null) {
            return true;
        } else {
            return false;
        }
    }

    return false;
}

public function isBranch(node:Object,
model:Object=null):Boolean
{
    if(node is Office) {
        if(node.officeAddress != "" && node.officeName != "")
        {
            return true;
        } else {
            return false;
        }
    }

    if(node is Name) {
        if(node.firstName != "" && node.firstName != "") {
            return true;
        } else {
            return false;
        }
    }

    if(node is ComplexDataType) {
        if(node.office != null && node.name != null) {
            return true;
        } else {
            return false;
        }
    }

    return true;
}

public function getData(node:Object,
model:Object=null):Object {
    if(node is Office) {
        return {children:{label:node.officeName,
label:node.officeAddress}};
    }
}

```

```

        }
        if(node is Name) {
            return {children:{label:node.firstName,
label:node.lastName}};
        }
        if(node is ComplexDataType) {
            return {children:{label:node.office,
label:node.name}};
        }
        return null;
    }

public function addChildAt(parent:Object, newChild:Object,
index:int,model:Object=null):Boolean
{
    return false;
}

public function removeChildAt(parent:Object, child:Object,
index:int,model:Object=null):Boolean
{
    return false;
}

}
}

```

使用数据描述符并使 Tree 控件完全显示 ComplexDataTypes 的数组，简单地设置一个之前代码中说到的 ComplexDataCollection 类的实例给 Tree 的 dataDescriptor 属性：

```
dataDescriptor="{new ComplexDataCollection()}"
```

完整代码如下：

代码：

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300" creationComplete="initCmp ()">
<mx:Script>
<! [CDATA[
import mx.collections.ArrayCollection;
```

```

[Bindable]
private var dataProv:ArrayCollection;

private function initCmp():void
{
    var compData:ComplexDataType = new ComplexDataType();
    compData.label = "13";
    var _name:Name = new Name();
    _name.firstName = "Joe";
    _name.lastName = "Smith";
    compData.name = _name;
    var office:Office = new Office();
    office.officeAddress = "544 Happy St. Anytown NY
                                01092";
    office.officeName = "Happy Town Branch";
    compData.office = office;

    var compData2:ComplexDataType =
        new ComplexDataType();
    compData2.label = "328";
    var _name2:Name = new Name();
    _name2.firstName = "Jane";
    _name2.lastName = "Doe";
    compData2.name = _name2;
    var office2:Office = new Office();
    office2.officeAddress = "544 Happy St. Anytown NY
                                01092";
    office2.officeName = "Happy Town Branch";
    compData2.office = office2;

    dataProv =
        new ArrayCollection([compData, compData2]);
}

] >
</mx:Script>
<mx:Tree dataDescriptor="{new ComplexDataCollection()}"
         dataProvider="{dataProv}" labelField="label" width="300"/>
</mx:Canvas>

```

## 5.8 节. 只允许 List 的某一项可被选中

### 5.8.1 问题

解析列表的 dataProvider 以确保某些条目不能被用户选择

### 5.8.2 解决方法

创建一个可以被设置在 List 组件的一个子类的 `filterFunction` 属性，使用 `mouseEventToItemRenderer` 和 `finishKeySelection` 通过 filter 函数来检查用户的选择并接受或不接受用户的选择。

### 5.8.3 讨论

为了控制用户选择列表中的某些条目，你需要控制这些条目，使用户可使用鼠标和键盘来选择，鼠标选择相对来说更容易处理一些：若 itemRenderer 包含你不想被选择的数据则可以简单地覆盖 `mouseEventToItemRenderer` 方法并返回 null，键盘事件的处理更复杂是由于当用户试图使用 up 或 down 键经过不可选的条目，而你需要将列表中的下一个可选的条目发送给用户。

为你的定制列表类中包含不可选条目的每一个实例来提供定制过滤，简单地创建一个 `disabledFilterFunction` 公有属性，以便用户可以传递一个值，创建成一个定制过滤，例如：

```
public var disabledFilterFunction:Function;
```

覆盖 `mouseEventToItemRenderer` 方法的工作完成后的返回由 MouseEvent 选中的 itemRenderer，而 `finishKeySelection` 键盘选择方法返回由 keyboardEvent 选中的 itemRenderer

代码：

```
<mx>List xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
<mx:Script>
<! [CDATA[

import flash.events.MouseEvent;
import flash.ui.Keyboard;
import mx.controls.listClasses.IListItemRenderer;
import mx.controls.List;

public var disabledFilterFunction:Function;
private var selectionIsAbove:Boolean;

// we need to catch all mouse events that could change
the index

override protected function
mouseEventToItemRenderer(event:MouseEvent):
IListItemRenderer {
var listItem:IListItemRenderer =
super.mouseEventToItemRenderer(event);
```

```

    if (listItem) {
        if (listItem.data) {
            if (disabledFilterFunction(listItem.data)) {
                return null;
            }
        }
    }
    return listItem;
}

```

当键盘选择事件结束后，下面捕获的所有键盘事件可以改变索引：

代码：

```

override protected function finishKeySelection():void {
    super.finishKeySelection();
    var i:int;
    var uid:String;
    var rowCount:int = listItems.length;
    var partialRow:int = (rowInfo[rowCount-1].y +
        rowInfo[rowCount-1].height > listContent.height) ?
        1 : 0;
    var item:IListItemRenderer = listItems[caretIndex -
        verticalScrollPosition][0];
    if (item) {
        if (item.data) {
            if (disabledFilterFunction(item.data)) {

```

下面使用 disabledFilterFunction 属性来决定特定的条目是否允许被选中，若当前条目不允许被选中，代码将定位到另一个可见的被激活的条目。

代码：

```

    rowCount = rowCount - partialRow;
    var currIndex:int = caretIndex -
        verticalScrollPosition;
    if (selectionIsAbove) {
        // look up;
        i = currIndex - 1;
        while(i>0){
            item = listItems[i][0];
            if (!disabledFilterFunction(item.data)){
                selectedIndex = i -
                    verticalScrollPosition;
                return;
            }
            i--;
        }
        i = currIndex + 1
        while(i<this.rowCount){

```

```

        item = listItems[i][0];
        if (!disabledFilterFunction(item.data)){
            selectedIndex = i -
                verticalScrollPosition;
            return;
        }
        i++;
    }
} else {
    // look down;
    while(i<this.rowCount) {
        item = listItems[i][0];
        if (!disabledFilterFunction(item.data)){
            selectedIndex = i -
                verticalScrollPosition;
            return;
        }
        i++;
    }
    while(i>0) {
        item = listItems[i][0];
        if (!disabledFilterFunction(item.data)){
            selectedIndex = i -
                verticalScrollPosition;
            return;
        }
        i--;
    }
}
}
]
]
>
</mx:Script>
</mx>List>

```

## 5.9 节. 为 List 的项编辑器添加格式化和验证数据

### 5.9.1. 问题

在提交输入值到列表之前验证用户在一个条目编辑器中输入的所有数据.

## 5.9.2. 解决方法

在 itemEditEnd 事件上，使用 ListBase 类的 itemEditorInstance 属性从条目编辑器中重新获得文本内容并解析其结果。

## 5.9.3 讨论

当用户开始和结束编辑一个列表中的条目时，为了验证和格式化所有输入数据，必须侦听由 List 发出的条目编辑事件。当条目编辑编辑完以后 List 控件发出三个事件

### itemEditBegin

当 editedItemPosition 属性已设置且条目可编辑后调度。当事件被触发，List 组件使用 createItemEditor 方法创建一个 itemEditor 对象并从条目拷贝数据属性到编辑器。默认情况下，itemEditor 对象是一个 TextInput 控件的实例。使用 List 控件的 itemEditor 属性来指定一个定制 itemEditor 类并最终设置 itemEditorInstance，来停止创建编辑器，可以将 preventDefault 作为默认事件侦听器的一部分来调用。

### itemEditBeginning

当用户准备好编辑项目（例如，在项目上释放鼠标按键）后调度。跳到 List 上面或其内部，或以任何方式试图编辑列表，该事件的默认侦听器设置 List.editedItemPosition 属性到有焦点的条目来启动条目编辑会话，为该事件写一个你自己的事件侦听器来拒绝编辑特别的条目和条目组。调用你自己事件侦听器中的 preventDefault 方法来防止默认的侦听器执行。

### itemEditEnd

当项目编辑会话因任何原因而结束时调度。List 组件有一个针对该事件的默认处理过程，它从 itemEditor 拷贝数据到 List 控件的数据提供者，默认情况下，List 使用 List 控件中的 editorDataField 属性来决定 itemEditor 的属性包含新数据并使用该新数据更新数据提供者条目，由于默认的 itemEditor 是 TextInput 控件，editorDataField 属性的默认值是 text，为 TextInput 包含新的条目数据指定 text 属性然后调用 destroyItemEditor，清除 itemEditor。在事件侦听方法内，你可以通过编辑器修改返回数据到 List 组件。在你的事件侦听器中，检查 itemEditor 中输入的数据，若数据不正确，你可以调用 preventDefault 来让 flex 停止传递新的数据到 List 组件和关闭编辑器。

下面的例子中，若基于正则表达式，事件的 preventDefault 方法用来防止编辑器把关闭和保存值给dataProvider，若被传递给 itemEditor 的值是正确的，则将该字符串格式化为适当的姓和名并保存该结果。

代码：

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
<mx:Script>
```

```

<! [CDATA[
    import mx.events.ListEvent;
    import mx.controls.TextInput;
    import mx.events.ListEvent;
    import mx.collections.ArrayCollection;

    [Bindable]
private var dp:ArrayCollection =
    new ArrayCollection([{name:"John Smith",
        positionType:1}, {name:"Ellen Smith",
        positionType:2}, {name:"James Smith",
        positionType:3}, {name:"Jane Smith",
        positionType:4}]);

```

下面定义的这个 itemEditEnd 事件的事件侦听器，注意事件是一个包含 reason 属性的 ListEvent 对象，该属性将被用作判断用户是否修改了 itemEditor 的值。

Code View:

```

public function formatData(event>ListEvent):void {
    // Check the reason for the event.
    if (event.reason == "cancelled") {
        // Do not update cell.
        return;
    }

    // Get the new data value from the editor.
    var newData:String =
        TextInput(event.currentTarget.itemEditorInstance)
            .text;
    // Determine if the new value is an empty String.
    var reg:RegExp = /\d/;
    if (newData == "" || reg.test(newData)) {
        // Prevent the user from removing focus,
        // and leave the cell editor open.
        event.preventDefault();
        // Use the errorString to inform the user that
        something is wrong.

        TextInput(listInstance.itemEditorInstance).setStyle("borderColor",
        0xff0000);

        TextInput(listInstance.itemEditorInstance).errorString = "Enter a
valid string.";
    }
}

```

```

        return void;
    }
    //test for FirstName LastName format
    reg = /\w+\.\w+\.\w+/
    if(!reg.test(newData)) {
        event.preventDefault();
        TextInput(listInstance.itemEditorInstance)
            .setStyle("borderColor", 0xff0000);
    }
    TextInput(listInstance.itemEditorInstance).errorString =
        "Enter first name and last name";
} else {
    //make sure the name is properly formatted
    var firstName:String = newData.substring(0,
        newData.indexOf(" "));
    var lastName:String =
        newData.substring(newData.indexOf(" ") + 1);
    firstName = firstName.charAt(0).toUpperCase() +
        firstName.substr(1);
    lastName = lastName.charAt(0).toUpperCase() +
        lastName.substr(1);
    TextInput(listInstance.itemEditorInstance).text =
        firstName+" "+lastName;
    newData = newData.charAt(0).toLocaleUpperCase() +
        newData.substring(1, newData.indexOf(" ")) +
        newData.charAt(newData.indexOf(" ") + 1) +
        newData.substring(newData.indexOf(" ") + 2);
}
}
]]>
</mx:Script>
<mx>List id="listInstance" dataProvider="{dp}" editable="true"
    labelField="name" itemEditEnd="formatData(event)"
    width="300"/>
</mx:Canvas>

```

## 5.10 节. 跟踪 **TileList** 中所有被选中的子节点

### 5.10.1. 问题

为 **TileList** 的渲染器设置一个切换，并跟踪 **TileList** 中所有被选中的子节点

### 5.10.2. 解决方法

扩展 **TileList** 组件并创建一个类似 **itemRenderer** 用法的定制渲染器，对于渲染器中的切换

事件，在事件中发布唯一的标识符 uid 到 TileList 并保存所有 IDs 到一个数组。

### 5.10.3. 讨论

对于所有 itemRenderers，当它们的 listData 属性被设置了，可以访问它们的父级 ListBase 组件。当一个渲染器实现 IDropInListItemRenderer，它的实现方法 get 和 set listData 一个 BaseListData 对象。BaseListData 类通过 owner 属性设置 itemRenderer 中的数据提供对 ListBase 对象的访问。如下所示：

```
public function set listData( value:BaseListData ):void
{
    private var listParent:ListBase;
    listParent = _listData.owner as ListBase;
}
```

在引入 ListBase 创建一个渲染器以后，所有 ListBase 中的 public 属性在渲染器中都是可访问的，下面实例做了一个所有切换按钮相对 itemRenderer 都可用的 ArrayCollection，在本例中是 SelectedChildrenTileListRender 类。SelectedChildrenTileListRender 类浏览 ArrayCollection 并从依赖于其按钮开关状态的数组添加或删除它自身。

SelectedChildrenTileListRender 使用由 BaseListData 传入的 uid 属性，来判断 itemRenderer 是否被描述到渲染器的数组。这个 uid，或者说唯一标识符，即使数据有复制条目，也可以确保 itemRenderer 将会像标识符本身一样被唯一引用。

SelectedChildrenTileListRender 定义的条目渲染器将被这样使用：

代码：

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml"
    implements="mx.controls.listClasses.IDropInListItemRenderer">
<mx:Script>
    <![CDATA[
        import mx.controls.listClasses.BaseListData;
        import mx.controls.Button;
        import mx.controls.TileList;
        import mx.events.FlexEvent;
        import mx.controls.listClasses.IDropInListItemRenderer;

        private var _listData:BaseListData;
        [Bindable]
        public var key:String;
        //the list parent of this renderer
        private var tileList:SelectedChildrenTileList;

        override public function set data(value:Object):void {
            super.data = value;
            this.invalidateProperties();
            dispatchEvent(new FlexEvent(FlexEvent.DATA_CHANGE));
        }

        [Bindable("dataChange")]
        public function get listData():BaseListData {
```

```

        return _listData;
    }

public function set listData( value:BaseListData ) :void
{
    //set the list data
    _listData = value;
    //get access to the tile list
    tileList = _listData.owner as
        SelectedChildrenTileList;
    //finally, track this item's unique ID that all items
    //in a BaseListData are assigned
    key=_listData.uid;
}

private function onToggle(event:Event) :void
{
    var i:int =
        tileList.toggledButtons.getItemIndex(key);
    //if the key of this renderer appears in the list of toggled buttons, then remove it
    if(i != -1 && (event.target as
        Button).selected==false) {
        tileList.toggledButtons.removeItemAt(i);
    }
    //if the key of this renderer doesn't appear in the list of toggled buttons, then add it
    if(i == -1 && (event.target as Button).selected ==
        true)
    {
        tileList.toggledButtons.addItem(key)
    }
    tileList.invalidateList();
}

override protected function
updateDisplayList(unscaledWidth:Number,
    unscaledHeight:Number) :void
{
    super.updateDisplayList(unscaledWidth,
        unscaledHeight);
    if(key)
    {
}

```

```

        var i:int=
            tileList.toggledButtons.getItemIndex(key);
            //sometimes putting your if/else on one line
makes things neater, not always, but sometimes
        if(i != -1){ toggleBtn.selected=true;
        } else { toggleBtn.selected = false; }
    }
}
]]>
</mx:Script>
<mx:Button id="toggleBtn" width="100%" label="{data.label}"
    toggle="true" change="onToggle(event)"/>
</mx:VBox>

```

下面代码显示的 SelectedChildrenTileList 使用 SelectedChildrenTileListRender，继承自 TileList，并且像前面提到的，定义一个 toggledButtons 的 ArrayCollection 来存储所有选中的条目的 uid。你同样必须定义一个 returnAllSelected 方法来寻找所有当前选中的 itemRenderers

代码：

```

<mx:TileList xmlns:mx="http://www.adobe.com/2006/mxml"
    initialize="toggledButtons = new ArrayCollection()">
    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;
            [Bindable]
            public var toggledButtons:ArrayCollection;

            public function removeAll():void{
                toggledButtons.removeAll();
                //use the invalidateList method to redraw the grid
right away
                this.invalidateList();
            }

            public function returnAllSelected():Array{
                var arr:Array = new Array();
                for(var i:int = 0; i < (dataProvider as
                    ArrayCollection).length; i++)
                {
                    if(toggledButtons.contains((indexToItemRenderer(i)
                        as SelectedChildrenTileListRender).key))
                    {
                        arr.push((indexToItemRenderer(i) as
                            SelectedChildrenTileListRender).data);
                    }
                }
            }
        ]]>
    </mx:Script>
</mx:TileList>

```

```

        }
    }
    return arr;
}
]]>
</mx:Script>
</mx:TileList>

```

## 5.11 节. 使用和显示项渲染器的 NULL 项

### 5.11.1. 问题

显示稀疏填充数组中的空条目

### 5.11.2. 解决方法

为 List 控件设置 **nullItemRenderer**.

### 5.11.3. 讨论

只要在继承自 ListBase 的任何类中的 dataProvider 遇到空对象, 则使用 nullItemRenderer 属性:

```
<mx:TileList nullItemRenderer="oreilly.cookbook.NullItemRenderer"/>
```

NullItemRenderer.mxml 为典型的 nullItemRenderer 列出了完整的清单:

代码:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="50"
height="50">
<mx:Image source="Assets.notAvailableImage"/>
<mx:Text text="sorry, unavailable" y="30"/>
</mx:Canvas>
```

NullItemRenderer 类被传入到 TileList 的 nullItemRenderer 属性, 如下所示:

代码:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
<mx:Script>
<! [CDATA[
import mx.collections.ArrayCollection;
[Bindable]
private var dp:ArrayCollection =
new ArrayCollection([null, {name:"Ellen Smith",
positionType:2}, null, {name:"Jane Smith",
positionType:4}]);
]]>
</mx:Script>
```

```

<mx:TileList width="100%" columnWidth="150" rowHeight="150"
   dataProvider="{dp}" labelField="name"
    nullItemRenderer="oreilly.cookbook.NullItemRenderer"/>
</mx:Canvas>

```

## 5.12 节. 为 List 创建右键菜单

### 5.12.1. 问题

当用户在特定的条目上右键单击或按住 Control 键并单击（在 Macintosh 中译者注）时创建一个定制上下文菜单项来显示

### 5.12.2. 解决方法

创建 ContextMenu 和 ContextMenuItem 对象并分派它们到渲染器，它们将像 itemRenderer 一样被分派到列表。

### 5.12.3. 讨论

上下文菜单项是用户在 flex 应用程序上右键单击或按住 Control 键并单击以后出现的， 默认情况下，菜单显示像 flashplayer9 屏幕信息链接一样的 Loop, Play, Print, Quality, Rewind, Save, 和 Zoom 控制。但是你可以创建一个新的 ContextMenu 对象从而很容易地定制这个菜单。简单地调用 ContextMenu 的构造函数并设置 contextMenu 属性的对象为创建好的显示对象，如下所示：

```

var menu:ContentMenu = new ContextMenu();
this.contextMenu = menu;

```

这段代码需要在 DisplayObject 上运行，也就是说，任何显示对象。用户在 DisplayObject 或组件上右键单击或按住 Control 键并单击时被创建的定制上下文菜单项才会显示设置好的 contentMenu 属性。

使用由 ContextMenu 定义的 customItems 数组，为上下文菜单项添加新的项。实例化新的 ContextMenuItem 对象并使用 push 方法将它们添加到数组。

ContextMenuItem 的构告函数具有如下信息。

代码：

```

ContextMenuItem(caption:String, separatorBefore:Boolean = false, enabled:Boolean = true,
visible:Boolean = true)

```

Caption 属性决定了菜单项的标题如，查询职工信息，separatorBefore 属性决定了是否需要一条细线出现在菜单中的 ContextMenuItem 上以区分菜单项。最终情况下，visible 和 enabled 属性决定各个条目是否是用户可见和可选的。

当用户选择该条目时 ContextMenuItem 发出一个 SELECT 类型的 ContextMenuEvent 事件。

下面例子为 List 控件创建了一个渲染器并基于 List 传入的数据类型来创建上下文菜单项。

代码：

```

<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="150"
height="80" paddingLeft="10">
<mx:Script>
<! [CDATA[

```

```

import flash.display.*;

override public function set data(value:Object):void
{
    if(value is Name)
    {
        text1.text = value.firstName;
        text2.text = value.lastName;
        var personMenu:ContextMenu = new ContextMenu();
        var lookupRecord:MenuItem =
            new MenuItem("Look Up Record");
        var lookupPicture:MenuItem =
            new MenuItem("Look Up Picture");
        personMenu.customItems.push(lookupRecord);
        personMenu.customItems.push(lookupPicture);
        this.contextMenu = personMenu;
    }
    else if(value is Office)
    {
        text1.text = value.officeAddress;
        text2.text = value.officeName;
        var officeMenu:ContextMenu = new ContextMenu();
        var lookupMap:MenuItem =
            new MenuItem("Look Up Map");

        lookupMap.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT,
        showMap);

        var lookupEmployees:MenuItem =
            new MenuItem("Look Up Employees");

        lookupEmployees.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT,
        showEmployees);

        officeMenu.customItems.push(lookupEmployees);
        officeMenu.customItems.push(lookupMap);
        this.contextMenu = officeMenu;
    }
}

private function showMap(event:ContextMenuEvent):void
{
    //do something with the map
}

private function

```

```

        showEmployees(event:ContextMenuEvent):void
    {
        //do something to look up all the employees
    }

} ]>
</mx:Script>
<mx:Text id="text1"/>
<mx:Text id="text2"/>
</mx:VBox>

```

## 5.13 节. 自定义 List 被选中项的外观

### 5.13.1. 问题

为 List 组件中的选中项贴上一个图片

### 5.13.2. 解决方法

覆盖 ListBase 类的 drawSelectionIndicator 方法并修改由该方法使用的指示器 Sprite 对象。

### 5.13.3. 讨论

List 控件通过 drawSelectionIndicator 方法为在列表中选中的 itemRenderer 创建外观。该方法的信息如下：

代码：

**override protected function** drawSelectionIndicator(indicator:Sprite, x:Number, y:Number, width:Number, height:Number, color:uint, itemRenderer:IListItemRenderer):void

所有的偏移量，大小，色彩信息都可以由 x, y, width, height, 和 color 属性来设置。第一个参数 indicator, 是一个画在选中项上的 flash.display.Sprite 实例，最后一个参数 itemRenderer 是将被选择的条目渲染器。

实现方法的例子添加了一个图片到指示器 Sprite 对象，由于当 itemRenderer 被撤消选中时 sprite 对象被移除和销毁，所以不必担心后面的回收。

除了应用于所有高光的 itemRenderer，drawHighlightIndicator 方法和 drawSelectionIndicator 方法功能一样，itemRenderer 被用户鼠标滑过，但不选择，请看：代码：

**override protected function** drawHighlightIndicator(indicator:Sprite, x:Number, y:Number, width:Number, height:Number, color:uint, itemRenderer:IListItemRenderer):void

用一张单独的图片来表现高光，放到 itemRenderer 的边缘，并在用户鼠标滑过列表的 itemRenderer 时显示。

下面是该技术的完整清单：

代码：

```

<mx>List xmlns:mx="http://www.adobe.com/2006/mxml"
    selectionColor="#ffcccc">
<mx:Script>
<! [CDATA[

    import mx.controls.listClasses.IListItemRenderer;

    [Embed("../assets/outline_arrow.gif")]
    private var img:Class;

    [Embed("../assets/in_arrow.gif")]
    private var highlight_img:Class;

    override protected function
        drawHighlightIndicator(indicator:Sprite,      x:Number,
            y:Number,  width:Number, height:Number, color:uint,
            itemRenderer:IListItemRenderer):void
    {
        var this_img:Object = new highlight_img();
        indicator.addChild((this_img as DisplayObject));
        (this_img as DisplayObject).x = itemRenderer.width -
            (this_img as DisplayObject).width
        super.drawHighlightIndicator(indicator, x, y, width,
            height, 0xff0000,itemRenderer);
    }

    override protected function
        drawSelectionIndicator(indicator:Sprite,      x:Number,
            y:Number,  width:Number, height:Number, color:uint,
            itemRenderer:IListItemRenderer):void
    {
        var this_img:Object = new img();
        indicator.addChild((this_img as DisplayObject));
        (this_img as DisplayObject).x = itemRenderer.width -
            (this_img as DisplayObject).width
        super.drawSelectionIndicator(indicator, x, y, width,
            height, 0xffcccc,itemRenderer);
    }
]]>
</mx:Script>
</mx>List>

```

## 第六章. DataGrid 和高级 DataGrid (常青)

DataGrid 控件是基于列表的控件，以多列布局专用于显示大数据集。DataGrid 可改变列宽，自定义渲染器和排序能力等等。Flex 3 添加了两个类似于 DataGrid 的新控件：AdvancedDataGrid 和 OLAPDataGrid。AdvancedDataGrid 控件扩展自 DataGrid 控件并添加额外的数据视觉能力，比如数据聚集，多列排序等。这些功能有些类似于 Microsoft Excel 的 Pivot Tables。Flex Builder 3 专业版已支持 AdvancedDataGrid 和 OLAPDataGrid 控件的数据视觉框架。

这两个 DataGrid 控件最典型的用法就是显示数组或类似集合的数据对象。AdvancedDataGrid 控件还可显示 HierarchicalData 对象，具有层次关系的复杂数据对象，允许创建特定的数据集。

### 6.1节. 创建 DataGrid 自定义列

#### 6.1.1. 问题

我想指定 DataGrid 自定义列，控制数据显示。

#### 6.1.2. 解决办法

使用 DataColumn 标签指定自定义列属性

#### 6.1.3. 讨论

本节添加三个 DataColumn 标签到 DataGrid 的 columns 属性中。使用 homesforsale.xml 作为数据文件。DataGridColumn 标签指定列表头标题和 dataProvider 内对象属性的显示顺序。DataGridColumn 的 dataField 属性指定显示该列的对象属性。这个例子中，对象的 range 属性没有被显示在 DataGrid 控件中，因为没有 DataGridColumn 的 dataField 与 range 属性相关联：

Code View:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    creationComplete="initApp()">
    <mx:HTTPService id="srv" url="assets/homesforsale.xml"
        resultFormat="object"
        result="onResult(event)"/>
    <mx:DataGrid id="grid"
```

```

        width="100%"
        height="100%"
        dataProvider="{ homesForSale }">
<mx:columns>
    <mx:DataGridColumn headerText="Total No."
        dataField="total"/>
    <mx:DataGridColumn headerText="City"
        dataField="city"/>
    <mx:DataGridColumn headerText="State"
        dataField="state"/>
</mx:columns>
</mx:DataGrid>
<mx:Script>
    <![CDATA[
        import mx.collections.ArrayCollection;
        import mx.rpc.events.ResultEvent;
        [Bindable]
        private var homesForSale:ArrayCollection;
        private function initApp():void {
            this.srv.send();
        }
        private function onResult(evt:ResultEvent):void {
            this.homesForSale = evt.result.data.region;
        }
    ]]>
</mx:Script>

</mx:Application>

```

DataGridColumn 通过使用 itemRenderers 可支持更多的自定义显示。下面的代码例子添加一个新的 DataGridColumn，它使用自定义渲染器---RangeRenderer，以更加有意义的方式来渲染 range 属性。Range 属性包含三个值指示基于价格范围销售的房屋百分比：range1 包含 \$350,000 以下的销售比, range2 是价格在 \$350,000 和 \$600,000 之间的销售比, range3 包含价格在 \$600,000 以上的房屋销售比。

Code View:

```

<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    creationComplete="initApp()">
    <mx:HTTPService id="srv" url="assets/homesforsale.xml"
        resultFormat="object"
        result="onResult(event)"/>
    <mx:DataGrid id="grid"

```

```

width="100%"
height="100%"
dataProvider="{homesForSale}">
<mx:columns>
    <mx:DataGridColumn headerText="Total No."
        dataField="total"/>
    <mx:DataGridColumn headerText="City"
        dataField="city"/>
    <mx:DataGridColumn headerText="State"
        dataField="state"/>
    <mx:DataGridColumn headerText="Price Ranges"
        dataField="range"
        itemRenderer="RangeRenderer"/>
</mx:columns>
</mx:DataGrid>
<mx:Script>
<![CDATA[
    import mx.collections.ArrayCollection;
    import mx.rpc.events.ResultEvent;
    [Bindable]
    private var homesForSale:ArrayCollection;
    private function initApp():void {
        this.srv.send();
    }
    private function onResult(evt:ResultEvent):void {
        this.homesForSale = evt.result.data.region;
    }
]]>
</mx:Script>
</mx:Application>

```

下面的代码就是 RangeRenderer，使用百分比数值画出颜色条，表示百分比。这是通过重写 updateDisplayList 方法来画出色彩条的，更新信息请看 [第七章，“渲染器和编辑器”](#)

Code View:

```

package {
    import flash.display.Graphics;
    import mx.containers.Canvas;
    public class RangeRenderer extends Canvas {
        override public function set data(value:Object):void {
            super.data = value;
            if(value!= null && value.range != null) {

```

```

        this.invalidateDisplayList();
    }
}

override protected function
updateDisplayList(unscaledWidth:Number,
unscaledHeight:Number):void {
    var g:Graphics = this.graphics;
    if(this.data) {
        var w1:Number = (this.data.range.range1 *
            unscaledWidth)/100;
        var w2:Number = (this.data.range.range2 *
            unscaledWidth)/100;
        var w3:Number = (this.data.range.range3 *
            unscaledWidth)/100;
        var x1:Number = 0;
        var x2:Number = w1;
        var x3:Number = w1 + w2;
        g.beginFill(0x0000ff);
        g.drawRect(x1,0,w1,unscaledHeight);
        g.beginFill(0x00ff00);
        g.drawRect(x2,0,w2,unscaledHeight);
        g.beginFill(0xff0000);
        g.drawRect(x3,0,w3,unscaledHeight);
    }
}
}

}

```

注意这里如果试图对 range 列进行排序的话会抛出 runtime 异常，你可以在下一节演示的那样使用自定义排序函数解决此问题。

## 6.2节. 为 DataGrid 列设定排序函数

### 6.2.1. 问题

我想使用自定义排序逻辑排序复杂对象

### 6.2.2. 解决办法

给 DataGridColumn 标签的 sortCompareFunction 属性赋值函数引用以执行自定义排序逻辑。

### 6.2.3. 讨论

像上一节那样修改 DataGrid 并添加自定义排序函数。这个例子使用 RangeRenderer 自定义渲染器添加排序函数 sortRanges 来显示 range 属性列：

Code View:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    creationComplete="initApp() ">
    <mx:HTTPService id="srv" url="assets/homesforsale.xml"
        resultFormat="object"
        result="onResult(event)"/>
    <mx:DataGrid id="grid" width="100%" height="100%"
        dataProvider="{homesForSale}">
        <mx:columns>
            <mx:DataGridColumn headerText="Total No."
                dataField="total"/>
            <mx:DataGridColumn headerText="City"
                dataField="city"/>
            <mx:DataGridColumn headerText="State"
                dataField="state"/>
            <mx:DataGridColumn headerText="Price Ranges [< 350K]
[350K -600K]
[> 600K]"
                dataField="range"
                itemRenderer="RangeRenderer"
                sortCompareFunction="sortRanges"/>
        </mx:columns>
    </mx:DataGrid>
    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;
            import mx.rpc.events.ResultEvent;
            [Bindable]
            private var homesForSale:ArrayCollection;
            private function initApp():void {
                this.srv.send();
            }
            private function onResult(evt:ResultEvent):void {
                this.homesForSale = evt.result.data.region;
            }
        ]]>
    </mx:Script>

```

```

private function sortRanges(obj1:Object,
obj2:Object):int{
    var value1:Number = obj1.range.range1;
    var value2:Number = obj2.range.range1;
    if(value1 < value2) {
        return -1;
    }
    else if(value1 > value2){
        return 1;
    }
    else {
        return 0;
    }
}
]]>
</mx:Script>

</mx:Application>

```

第四个 DataGridColumn 的 sortCompareFunction 属性被设置为 sortRanges，即对 ranges 实现自定义排序逻辑。sortCompareFunction 属性所需函数格式为：

**sortCompareFunction(obj1:Object, obj2:Object):int**

该函数接受两个类型为 Object 的参数，代表dataProvider 中被比较的两个对象，返回值为-1, 1, 或0指示两个对象的放置顺序。当用户点击 DataGridColumn 列表头时，DataGrid 运行相应的 sortCompareFunction 对应函数。在 sortRange 函数内我们看到 range1属性被用来计算排序顺序。现在当用户点击 ProceRanges 列时排序顺序将基于 range1值。

## 6.3 节. 启动 DataGrid 多列排序

### 6.3.1. 问题

我想启动多列排序功能

### 6.3.2. 解决办法

使用 AdvancedDataGrid 控件的 AdvancedDataGridColumn 提供多列排序支持

### 6.3.3. 讨论

AdvancedDataGrid 控件内建支持多列排序。为了演示，下面的例子代码修改了上一节的例子，用 AdvancedDataGrid 和 AdvancedDataGridColumn 代替 DataGrid 和 DataColumn，

AdvancedDataGrid 支持两种多列排序方式。默认为 sortExpertMode=false，允许用户点击列表头作为主要排序方式，第二排序是通过点击多列排序区域(表头的右边)实现的。下面的例子使用专家模式 sortExpertMode=true，也就是点击列表头作为主要排序次要排序是右击列表头，

Code View:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    creationComplete="initApp () ">

    <mx:HTTPService id="srv" url="assets/homesforsale.xml"
        resultFormat="object"
        result="onResult (event) "/>

    <mx:AdvancedDataGrid id="grid"
        width="100%"
        height="100%"
        sortExpertMode="true"
        dataProvider="{ homesForSale }">
        <mx:columns>
            <mx:AdvancedGridColumn headerText="Total No."
                dataField="total"/>
            <mx:AdvancedGridColumn headerText="City"
                dataField="city"/>
            <mx:AdvancedGridColumn headerText="State"
                dataField="state"/>
            <mx:AdvancedGridColumn headerText="Price      Ranges
[<350K] [350K -600K] [>600K]"
                dataField="range"
                itemRenderer="RangeRenderer"
                sortCompareFunction="sortRanges"/>
        </mx:columns>
    </mx:AdvancedDataGrid>
    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;
            import mx.rpc.events.ResultEvent;
            [Bindable]
            private var homesForSale:ArrayCollection;
            private function initApp ():void {
                this.srv.send();
            }
            private function onResult (evt:ResultEvent):void {
                this.homesForSale = evt.result.data.region;
            }
        ]]>
    </mx:Script>

```

```

        }
    private function sortRanges(obj1:Object,
    obj2:Object):int{
        var value1:Number = obj1.range.range1;
        var value2:Number = obj2.range.range1;
        if(value1 < value2) {
            return -1;
        }
        else if(value1 > value2){
            return 1;
        }
        else {
            return 0;
        }
    }
    ]]>
</mx:Script>
</mx:Application>

```

## 6.4 节. 过滤 DataGrid 数据项

### 6.4.1. 问题

我想在客户端对显示的数据进行过滤

### 6.4.2. 解决办法

给 ArrayCollection 的 filterFunction 属性赋值自定义函数引用以执行过滤匹配。

### 6.4.3. 讨论

为了掩饰客户端过滤数据，下面的例子添加一个 city 过滤功能。UI 新增一个 TextInput 文本输入框供用户输入 city 名称并过滤掉 DataGrid 中匹配的相应数据。当用户在 cityFilter 文本框控件中输入数据时，文本框会触发 change 事件，通过 applyFilter 方法进行处理。applyFilter 方法赋值一个函数引用给 homesForSale ArrayCollection 实例的 filterFunction 属性，并调用 ArrayCollection 的 refresh 方法。filterCities 方法对dataProvider 的 city 属性值和文本框输入值进行小写字符串匹配测试：

Code View:

```

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical" creationComplete="initApp()">
    <mx:HTTPService id="srv" url="assets/homesforsale.xml"
        resultFormat="object" result="onResult(event)"/>

    <mx:Form>

```

```

<mx:FormItem label="City">
    <mx:TextInput id="cityFilter" change="applyFilter()"/>
</mx:FormItem>
</mx:Form>

<mx:AdvancedDataGrid id="grid" width="100%" height="100%" sortExpertMode="true" dataProvider="{homesForSale}">
<mx:columns>
    <mx:AdvancedGridColumn headerText="Total No." dataField="total"/>
    <mx:AdvancedGridColumn headerText="City" dataField="city"/>
    <mx:AdvancedGridColumn headerText="State" dataField="state"/>
    <mx:AdvancedGridColumn headerText="Price Ranges [&lt;350K] [350K -600K] [&gt;600K]" dataField="range" itemRenderer="RangeRenderer" sortCompareFunction="sortRanges"/>
</mx:columns>
</mx:AdvancedDataGrid>
<mx:Script>
<![CDATA[
    import mx.events.FlexEvent;
    import mx.collections.ArrayCollection;
    import mx.rpc.events.ResultEvent;

    [Bindable]
    private var homesForSale:ArrayCollection;

    private function initApp():void {
        this.srv.send();
    }

    private function onResult(evt:ResultEvent):void {
        this.homesForSale = evt.result.data.region;
    }

    private function sortRanges(obj1:Object, obj2:Object):int{
        var value1:Number = obj1.range.rang1;
        var value2:Number = obj2.range.rang1;

        if(value1 < value2) {
            return -1;
        }
    }
]]>

```

```

        }
    else if(value1 > value2){
        return 1;
    }
    else {
        return 0;
    }
}

private function applyFilter():void {
    if(this.homesForSale.filterFunction == null) {
        this.homesForSale.filterFunction =
            this.filterCities;
    }
    this.homesForSale.refresh();
}

private function filterCities(item:Object):Boolean {
    var match:Boolean = true;

    if(cityFilter.text != "") {
        var city:String = item["city"];
        var filter:String = this.cityFilter.text;
        if(!city ||
            city.toLowerCase().indexOf(filter.toLowerCase()) < 0) {
            match = false;
        }
    }
    return match;
}
] ]>
</mx:Script>
</mx:Application>

```

## 6.5节. 为 AdvancedDataGrid 创建自定义表头

### 6.5.1. 问题

我想用一个复选框作为 DataGrid 表头

## 6.5.2. 解决办法

继承 AdvancedDataGridHeaderRenderer 类，并重写 createChildren 和 updateDisplayList 方法

## 6.5.3. 讨论

这一节根据上一节例子为 city DataGridColumn 增加自定义表头渲染器。创建标题渲染器和创建项渲染器或项编辑器基本类似。一个实现 IFactory 接口的类引用被传递给 DataGridColumn 的 headerRenderer 属性，这个例子使用 CheckBoxHeaderRenderer 渲染器创建带有复选框的列表头：

Code View:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical" creationComplete="initApp()">
    <mx:HTTPService id="srv" url="assets/homesforsale.xml"
        resultFormat="object" result="onResult(event)"/>
    <mx:Form>
        <mx:FormItem label="City">
            <mx:TextInput id="cityFilter" change="applyFilter()"/>
        </mx:FormItem>
    </mx:Form>
    <mx:AdvancedDataGrid id="grid" width="100%" height="100%"
        sortExpertMode="true" dataProvider="{homesForSale}">
        <mx:columns>
            <mx:AdvancedDataGridColumn headerText="Total No."
                dataField="total"/>
            <mx:AdvancedDataGridColumn headerText="City"
                sortable="false"
                headerRenderer="CheckBoxHeaderRenderer"
                dataField="city"/>
            <mx:AdvancedDataGridColumn
                headerText="State dataField="state"/>
            <mx:AdvancedDataGridColumn headerText="Price Ranges
[<350K] [350K -600K] [>600K]" dataField="range"
itemRenderer="RangeRenderer" sortCompareFunction="sortRanges"/>
        </mx:columns>
    </mx:AdvancedDataGrid>
    <mx:Script>
        <![CDATA[
            import mx.events.FlexEvent;
            import mx.collections.ArrayCollection;
            import mx.rpc.events.ResultEvent;

            [Bindable]
        ]]>
    
```

```

private var homesForSale:ArrayCollection;

private function initApp():void {
    this.srv.send();
}

private function onResult(evt:ResultEvent):void {
    this.homesForSale = evt.result.data.region;
}

private function sortRanges(obj1:Object,
obj2:Object):int{
    var value1:Number = obj1.range.range1;
    var value2:Number = obj2.range.range1;

    if(value1 < value2) {
        return -1;
    }
    else if(value1 > value2){
        return 1;
    }
    else {
        return 0;
    }
}

private function applyFilter():void {
    if(this.homesForSale.filterFunction == null) {
        this.homesForSale.filterFunction =
            this.filterCities;
    }
    this.homesForSale.refresh();
}

private function filterCities(item:Object):Boolean {
    var match:Boolean = true;

    if(cityFilter.text != "") {
        var city:String = item["city"];
        var filter:String = this.cityFilter.text;
        if(!city ||
city.toLowerCase().indexOf(filter.toLowerCase()) < 0) {
            match = false;
        }
    }
}

```

```

        }
    }
    return match;
}

] ]>
</mx:Script>

</mx:Application>

```

下面的代码就是自定义表头渲染器类 CheckBoxHeaderRenderer。注意它重写了 AdvancedDataGridHeader 的 createChildren 方法来创建新的 CheckBox 实例并添加到显示列表。updateDisplayList 方法用来重新计算复选框大小，下一节介绍如何处罚自定义表头事件及处理细节。

Code View:

```

package {

    import flash.events.Event;

    import mx.controls.AdvancedDataGrid;
    import mx.controls.CheckBox;
    import

mx.controls.advancedDataGridClasses.AdvancedDataGridHeaderRenderere
r;
    import mx.events.AdvancedDataGridEvent;

    public class CheckBoxHeaderRenderer extends
AdvancedDataGridHeaderRenderer {

    private var selector:CheckBox;

    override protected function createChildren():void {
        super.createChildren();
        this.selector = new CheckBox();
        this.selector.x = 5;
        this.addChild(this.selector);
    }

    override protected function
updateDisplayList(unscaledWidth:Number,
    unscaledHeight:Number):void
{
    super.updateDisplayList(unscaledWidth, unscaledHeight);
}
}

```

```

        this.selector.setActualSize(
            this.selector.getExplicitOrMeasuredWidth(),
            this.selector.getExplicitOrMeasuredHeight());
    }
}

}

```

## 6.6节. 处理 DataGrid/AdvancedDataGrid 相关事件

### 6.6.1. 问题

我需要管理有 DataGrid 和它的项渲染器发出的事件

### 6.6.2. 解决办法

使用项渲染器的 owner 属性调度父组件 DataGrid 的事件

### 6.6.3. 讨论

上一节中，通过传递类引用给 column 的 headerRenderer 属性，自定义列表头渲染器被创建出来。这一节将继续上一节的表头渲染器。当表头渲染器的复选框被点击时，该类将触发一个事件到父组件 DataGrid。

Code View:

```

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical" creationComplete="initApp()">

    <mx:HTTPService id="srv" url="assets/homesforsale.xml"
        resultFormat="object"
        result="onResult(event)"/>

    <mx:Form>
        <mx:FormItem label="City">
            <mx:TextInput id="cityFilter" change="applyFilter()"/>
        </mx:FormItem>
    </mx:Form>

    <mx:AdvancedDataGrid id="grid" width="100%" height="100%"
        sortExpertMode="true" dataProvider="{homesForSale}">

```

```

creationComplete="assignListeners () ">
<mx:columns>
    <mx:AdvancedGridColumn headerText="Total No."
        dataField="total"/>
    <mx:AdvancedGridColumn headerText="City"
        sortable="false"
        headerRenderer="CheckBoxHeaderRenderer2"
        dataField="city"/>
    <mx:AdvancedGridColumn headerText="State"
        dataField="state"/>
        <mx:AdvancedGridColumn     headerText="Price      Ranges
[<350K] [350K -600K] [>600K]" dataField="range"
            itemRenderer="RangeRenderer"
            sortCompareFunction="sortRanges"/>
</mx:columns>
</mx:AdvancedDataGrid>
<mx:Script>
<! [CDATA[
    import mx.events.FlexEvent;
    import mx.collections.ArrayCollection;
    import mx.rpc.events.ResultEvent;

    [Bindable]
    private var homesForSale:ArrayCollection;

    private function initApp():void {
        this.srv.send();
    }

    private function onResult(evt:ResultEvent):void {
        this.homesForSale = evt.result.data.region;
    }

    private function sortRanges(obj1:Object,
        obj2:Object):int{
        var value1:Number = obj1.range.rangel;
        var value2:Number = obj2.range.rangel;

        if(value1 < value2) {
            return -1;
        }
        else if(value1 > value2){
            return 1;
        }
    }

```

```

        else {
            return 0;
        }
    }

private function applyFilter():void {
    if(this.homesForSale.filterFunction == null) {
        this.homesForSale.filterFunction =
            this.filterCities;
    }
    this.homesForSale.refresh();
}

private function filterCities(item:Object):Boolean {
    var match:Boolean = true;

    if(cityFilter.text != "") {
        var city:String = item["city"];
        var filter:String = this.cityFilter.text;
        if(!city ||
            city.toLowerCase().indexOf(filter.toLowerCase()) < 0) {
            match = false;
        }
    }

    return match;
}

```

根据事件冒泡机制，事件从 DataGridColumn 传递到父组件 DataGrid。onColumnSelect方法将接收自定义事件ColumnSelectedEvent，它包含哪个表头渲染器被使用的信息。

```

private function assignListeners():void {

    this.grid.addEventListener(ColumnSelectedEvent.COLUMN_SELECTED,
        onColumnSelect);
}

private function
onColumnSelect(evt:ColumnSelectedEvent):void {
    trace("column selected = " + evt.colIdx);
}

]]>
</mx:Script>

```

```
</mx:Application>
```

这个例子建立在上一节的例子之上，为 CheckBoxHeaderRenderer2，代码赋值一个监听器给 ColumnSelectedEvent，这个 AdvancedDataGrid 的自定义事件，监听器函数 onColumnSelected 只是在控制台输出被选择的列索引值，下面的代码为 CheckBoxHeaderRenderer2：

Code View:

```
package {

    import flash.events.MouseEvent;

    import mx.controls.AdvancedDataGrid;
    import mx.controls.CheckBox;
    import
    mx.controls.advancedDataGridClasses.AdvancedDataGridHeaderRenderere
r;

    public class CheckBoxHeaderRenderer2 extends
        AdvancedDataGridHeaderRenderer {

        private var selector:CheckBox;

        override protected function createChildren():void {
            super.createChildren();
            this.selector = new CheckBox();
            this.selector.x = 5;
            this.addChild(this.selector);
            this.selector.addEventListener( MouseEvent.CLICK,
                dispatchColumnSelected);
        }

        override protected function
        updateDisplayList(unscaledWidth:Number,
            unscaledHeight:Number):void
        {
            super.updateDisplayList(unscaledWidth, unscaledHeight);

            this.selector.setActualSize(this.selector.getExplicitOrMeasuredWi
dth(), this.selector.getExplicitOrMeasuredHeight());
        }

        private function
        dispatchColumnSelected(evt:MouseEvent):void {
            var event:ColumnSelectedEvent =

```

```

        new ColumnSelectedEvent(
            ColumnSelectedEvent.COLUMN_SELECTED,
            listData.columnIndex,
            selector.selected);
        AdvancedDataGrid(listData.owner).dispatchEvent(event);
    }
}

}

```

注意这里，虽然 ColumnSelectedEvent 最后是由 AdvancedDataGrid 发出的，但最初它是当复选框被点击时由表头渲染器实例发出的。 CheckBoxHeaderRenderer2 的 dispatchColumnSelected 方法使用 listData.owner 属性得到父组件 AdvancedDataGrid 的引用，接着从"owner"处触发此事件。

[AdvancedDataGrid\(listData.owner\).dispatchEvent\(event\);](#)

最后我们看一下自定义事件类 CustomSelectedEvent。它简单继承自 Event 类和两个属性： colIdx 存储列索引， isSelected 指示是否列被选择：

Code View:

```

package {
    import flash.events.Event;

    public class ColumnSelectedEvent extends Event {

        public var colIdx:int;
        public var isSelected:Boolean;

        public static const COLUMN_SELECTED:String =
            "columnSelected";

        public function
        ColumnSelectedEvent(type:String,colIdx:Int,isSelected:
        Boolean) {
            super(type);

            // Set the new property.
            this.colIdx = colIdx;
            this.isSelected = isSelected;
        }

        override public function clone():Event {
            return new ColumnSelectedEvent(type, colIdx,isSelected);
        }
    }
}

```

```
    }  
}  
  
}
```

## 6.7节. AdvancedDataGrid 数据项选择

### 6.7.1. 问题

我想编程实现选择 AdvancedDataGrid 多个单元格

### 6.7.2. 问题

设置 AdvancedDataGrid 的 selectionMode 属性为 multipleCells 和 selectedCells 属性为 object 数组，该数组包含被选择单元格的 rowIndex 和 columnIndex。

### 6.7.3. 讨论

AdvancedDataGrid 控件对于选择单元格提供了多种选项设置。selectionMode 属性值有以下几种选择：

[Multiple cells](#)

[Multiple rows](#)

[Single cell](#)

[Single row](#)

[None](#)

要允许多个单元格被选择，allowMultipleSelection 属性也需要被设置为 true。

下面的例子中，当用户选择 City 列中的复选框时，该列的所有单元格都被选中。这个例子代码建立上一节的例子之上。当 city 列表头的复选框被选择时，AdvancedDataGrid 发出 ColumnSelectedEvent 事件，由 onColumnSelect 方法处理该事件。onColumnSelect 方法构造一个 object 数组，每个对象都包含被选择单元格的 rowIndex 和 cellIndex。这个例子选中了该列的所有单元格，最后把这个 object 数组赋值给 selectedCells 属性。需要注意的是，如果表格显示发生变化，则需要重新创建一个数组并赋值给 selectedCells 属性。想通过 grid.selectedCells.push 方法直接添加单元格到 selectedCells 属性是不行的，因为表格不会去检测数组是否发生变化。

Code View:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"    creationComplete="initApp()">
    <mx:HTTPService id="srv" url="assets/homesforsale.xml"
        resultFormat="object" result="onResult(event)"/>
    <mx:Form>
        <mx:FormItem label="City">
            <mx:TextInput id="cityFilter" change="applyFilter()"/>
        </mx:FormItem>
    </mx:Form>
    <mx:AdvancedDataGrid id="grid" width="100%" height="100%" 
        sortExpertMode="true"           dataProvider="{homesForSale}"
        selectionMode="multipleCells"
        creationComplete="assignListeners()">
        <mx:columns>
            <mx:AdvancedGridColumn headerText="Total No."
                dataField="total"/>
            <mx:AdvancedGridColumn headerText="City"
                sortable="false"
                headerRenderer="CheckBoxHeaderRenderer2" dataField="city"/>
                <mx:AdvancedGridColumn headerText="State"
                    dataField="state"/>
                <mx:AdvancedGridColumn headerText="Price      Ranges
[<350K] [350K -600K] [>600K]"
                    dataField="range"
                    itemRenderer="RangeRenderer"
                    sortCompareFunction="sortRanges"/>
        </mx:columns>
    </mx:AdvancedDataGrid>
    <mx:Script>
        <![CDATA[
            import mx.events.FlexEvent;
            import mx.collections.ArrayCollection;
            import mx.rpc.events.ResultEvent;

            [Bindable]
            private var homesForSale:ArrayCollection;

            private function initApp():void {
                this.srv.send();
            }

            private function onResult(evt:ResultEvent):void {
        
```

```

        this.homesForSale = evt.result.data.region;
    }

private function sortRanges(obj1:Object,
obj2:Object):int{
    var value1:Number = obj1.range.range1;
    var value2:Number = obj2.range.range1;

    if(value1 < value2) {
        return -1;
    }
    else if(value1 > value2){
        return 1;
    }
    else {
        return 0;
    }
}

private function applyFilter():void {
    if(this.homesForSale.filterFunction == null) {
        this.homesForSale.filterFunction =
            this.filterCities;
    }
    this.homesForSale.refresh();
}

private function filterCities(item:Object):Boolean {
    var match:Boolean = true;

    if(cityFilter.text != "") {
        var city:String = item["city"];
        var filter:String = this.cityFilter.text;
        if(!city ||
city.toLowerCase().indexOf(filter.toLowerCase()) < 0) {
            match = false;
        }
    }

    return match;
}

private function assignListeners():void {

```

```

>this.grid.addEventListener(ColumnSelectedEvent.COLUMN_SELECTED,
onColumnSelect);
}

private function
onColumnSelect(evt:ColumnSelectedEvent):void {
    var selectedCells:Array = new Array();
    var colIdx:int = evt.colIdx;

    if(evt.isSelected) {
        for(var i:int=0;i<this.homesForSale.length;i++) {

selectedCells.push({rowIndex:i,columnIndex:colIdx});
    }
}
this.grid.selectedCells = selectedCells;
}

]]>
</mx:Script>
</mx:Application>

```

## 6.8节. 启动 DataGrid 拖拽功能

### 6.8.1. 问题

我想启动 DataGrid 数据项拖动功能，以便用户能拖动它们到其他表格上。

### 6.8.2. 解决办法

设置源 DataGrid 的 dragEnabled 为 true，和目标 DataGrid 的 dropEnabled 属性为 true。

### 6.8.3. 讨论

要启动基于列表的控件比如 DataGrid 的拖拽功能，最简单的方法就是设置相应的属性为 true，因为 Flex 框架将负责所有相关的底层工作。例如，下面的例子设置 AdvancedDataGrid 的 dragEnabled 属性为 true，意思就是支持该控件的数据项能被拖动到控件之外的地方。而 DataGrid 的 dropEnabled 属性设置为 true，表示该控件可接受被拖进来的数据项。另外一个属性即 dragMoveEnabled 将影响拖动效果。

对于非列表型控件或需自定义拖拽操作的，Flex 3 框架为此提供了 DragManager，它将在 [第](#)

[十一章](#), "拖拽操作" 中介绍。

Code View:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal" creationComplete="initApp()">
    <mx:HTTPService id="srv" url="assets/homesforsale.xml"
        resultFormat="object" result="onResult(event)"/>
    <mx:AdvancedDataGrid id="grid" width="100%" height="100%"
        sortableColumns="false"
        dragEnabled="true"
        dataProvider="{homesForSale}">
        <mx:columns>
            <mx:AdvancedGridColumn headerText="Total No."
                dataField="total"/>
            <mx:AdvancedGridColumn headerText="City"
                sortable="false" dataField="city"/>
            <mx:AdvancedGridColumn headerText="State"
                dataField="state"/>
        </mx:columns>
    </mx:AdvancedDataGrid>

    <mx:DataGrid width="100%" height="100%"
        dropEnabled="true">
        <mx:columns>
            <mx:DataGridColumn headerText="Total No."
                dataField="total"/>
            <mx:DataGridColumn headerText="City" sortable="false"
                dataField="city"/>
            <mx:DataGridColumn headerText="State"
                dataField="state"/>
        </mx:columns>
    </mx:DataGrid>

    <mx:Script>
        <![CDATA[
            import mx.events.FlexEvent;
            import mx.collections.ArrayCollection;
            import mx.rpc.events.ResultEvent;

            [Bindable]
            private var homesForSale:ArrayCollection;

            private function initApp():void {
                this.srv.send();
        ]]>
    </mx:Script>

```

```

        }

    private function onResult(evt:ResultEvent):void {
        this.homesForSale = evt.result.data.region;
    }
} ]>
</mx:Script>

</mx:Application>

```

## 6.9节. 编辑 DataGrid 数据项

### 6.9.1. 问题

我想让 DataGrid 的某些单元格可被编辑

### 6.9.2. 解决办法

设置 AdvancedDataGrid 或 DataGrid 的 editable 属性为 true

### 6.9.3. 讨论

这个例子中，AdvancedDataGrid 和 DataGrid 控件绑定到同一个 dataProvider。两个控件的 editable 属性都设置为 true。这样就可以编辑每个单元格了。由于两个控件都绑定同一个数据源，当编辑一个表格的单元格时将会改变另一个表格。代码如下：

Code View:

```

<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal" creationComplete="initApp ()">

    <mx:HTTPService id="srv"
        url="assets/homesforsale.xml"
        resultFormat="object"
        result="onResult(event)"/>

    <mx:AdvancedDataGrid id="grid" width="100%" height="100%"
        sortableColumns="false" editable="true"
        dataProvider="{homesForSale}">
        <mx:columns>

```

```

<mx:AdvancedDataGridColumn headerText="Total No."
    dataField="total"/>
<mx:AdvancedDataGridColumn headerText="City"
    sortable="false" dataField="city"/>
<mx:AdvancedDataGridColumn headerText="State"
    dataField="state"/>
</mx:columns>
</mx:AdvancedDataGrid>

<mx:DataGrid width="100%" height="100%" editable="true"
    dataProvider="{homesForSale}">
<mx:columns>
    <mx:DataGridColumn headerText="Total No."
        dataField="total"/>
    <mx:DataGridColumn headerText="City" sortable="false"
        dataField="city"/>
    <mx:DataGridColumn headerText="State"
        dataField="state"/>
</mx:columns>
</mx:DataGrid>

<mx:Script>
<! [CDATA[
    import mx.events.FlexEvent;
    import mx.collections.ArrayCollection;
    import mx.rpc.events.ResultEvent;

    [Bindable]
    private var homesForSale:ArrayCollection;

    private function initApp():void {
        this.srv.send();
    }

    private function onResult(evt:ResultEvent):void {
        this.homesForSale = evt.result.data.region;
    }
]]>
</mx:Script>

</mx:Application>

```

## 6.10节. 在 DataGrid 中搜索并自动滚屏到匹配项

### 6.10.1. 问题

我想搜索 DataGrid 中的数据项并滚屏到匹配项

### 6.10.2. 解决办法

在 ArrayCollection 中使用 IViewCursor 的 findFirst 方法。使用 DataGrid 的 scrollToIndex 进行滚屏。

### 6.10.3. 讨论

这项技术的关键之处在于 DataGrid 和一个简单的表达，提供用户在文本框中输入 city 名称，然后点击按钮开始搜索。当用户点击按钮(search\_btn)后，DataGrid 的 dataProvider 被搜索，相应的行被选择，如果所在行没显示的话滚屏到所在视图。

该解决方案的两个主要方面是查找匹配项和定位 DataGrid 的相应项。查找匹配项，可使用 IViewCursor，它是一个接口，规定一些属性和方法用于遍历集合。所有的 Flex 集合对象都支持 createCursor 方法返回一个 IViewCursor 类实例。这个例子中，下面的几行代码创建一个 ArrayCollection 游标实例，作为 DataGrid 的 dataProvider：

```
private function onResult(evt:ResultEvent):void {
    var sort:Sort = new Sort();
    sort.fields = [ new SortField("city", true) ];
    this.homesForSale = evt.result.data.region;
    this.homesForSale.sort = sort;
    this.homesForSale.refresh();
    this.cursor = this.homesForSale.createCursor();

}
```

注意你可以赋值 Sort 对象给定义了 ArrayCollection，dataProvider 数据项 city 属性作为可排序字段。这是因为 IViewCursor 的 findFirst 和其他查找方法只可以被可排序视图所调用。

当游标被创建后，它可以导航和查询相关联的视图，当用户点击 Search City 按钮时下面的 searchCity 方法便被调用：

Code View:

```
private function searchCity():void {
    if(search_ti.text != "") {
        if(this.cursor.findFirst({city:search_ti.text})) {
            var idx:int =
            this.homesForSale.getItemIndex(this.cursor.current);
```

```

        this.grid.scrollToIndex(idx);
        this.grid.selectedItem = this.cursor.current;
    }
}

}

```

这个方法中，用户输入的 city 名称作为 IViewCursor 的 findFirst 方法的搜索参数。当找到 ArrayCollection 中的匹配项时该方法返回 true 并更新游标对象的 current 属性，指向匹配项。当匹配项被找到后， ArrayCollection 的 getItemIndex 方法指出匹配项的具体位置，最后 DataGrid 使用 scrollToIndex 方法滚屏到匹配位置， selectedItem 属性被设置为匹配项。

完整的代码如下：

Code View:

```

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical" creationComplete="initApp()">

    <mx:HTTPService id="srv" url="assets/homesforsale.xml"
        resultFormat="object" result="onResult(event)"/>

    <mx:Form>
        <mx:FormItem label="Search">
            <mx:TextInput id="search_ti"/>
        </mx:FormItem>
        <mx:FormItem>
            <mx:Button label="Search City" click="searchCity()"/>
        </mx:FormItem>
    </mx:Form>

    <mx:DataGrid id="grid" width="300" height="150"
        editable="true" dataProvider="{homesForSale}">
        <mx:columns>
            <mx:DataGridColumn headerText="Total No."
                dataField="total"/>
            <mx:DataGridColumn headerText="City"="false"
                dataField="city"/>
            <mx:DataGridColumn headerText="State"
                dataField="state"/>
        </mx:columns>
    </mx:DataGrid>

    <mx:Script>
        <![CDATA[

```

```

import mx.collections.SortField;
import mx.collections.Sort;
import mx.collections.IViewCursor;
import mx.events.FlexEvent;
import mx.collections.ArrayCollection;
import mx.rpc.events.ResultEvent;

[Bindable]
private var homesForSale:ArrayCollection;
private var cursor:IViewCursor;

private function initApp():void {
    this.srv.send();
}

private function onResult(evt:ResultEvent):void {
    var sort:Sort = new Sort();
    sort.fields = [ new SortField("city", true) ];
    this.homesForSale = evt.result.data.region;
    this.homesForSale.sort = sort;
    this.homesForSale.refresh();
    this.cursor = this.homesForSale.createCursor();
}

private function searchCity():void {
    if(search_ti.text != "") {
        if(this.cursor.findFirst({city:search_ti.text})) {
            var idx:int =
this.homesForSale.getItemIndex(this.cursor.current);
            this.grid.scrollToIndex(idx);
            this.grid.selectedItem = this.cursor.current;
        }
    }
}

] ]>
</mx:Script>

</mx:Application>

```

## 6.11节. 使用 **GroupingCollection** 生成数据汇总

### 6.11.1. 问题

我想为表格数据生成汇总值

### 6.11.2. 解决办法

使用 GroupingCollection 生成汇总值，配置 AdvancedDataGrid 以便它看起来具有数据汇总功能。

### 6.11.3. 讨论

你可以使用 GroupingCollection 生成数据汇总，配置 AdvancedDataGrid 显示汇总数据。

当生成数据汇总时，你不想排序和分组现有的 dataField，因为你只是简单的显示数据而已，下面的示例代码生成一个虚拟群，指定 fieldNameNotPresent 作为群字段的 dataField 值。现在你可以使用 SummaryRow 和 SummaryField 对象设置汇总了。

当汇总数据准备好后，你还有第二个任务。当 ADG.dataProvider 填充数据到 GroupingCollection 后，data provider 将会试图把集合作为实现 IHierarchicalData 的 GroupingCollection 显示到树视图中。在内部，GroupingCollection 会被转换为 HierarchicalCollectionView，ADG.dataProvider 返回 HierarchicalCollectionView 实例。(这和 array 的 dataProvider 类似，它在内部将被转换为 ArrayCollection) 使用 HierarchicalCollectionView 的 showRoot 属性你可以控制根节点的显示。通过将它设置为 False，可以防止虚拟群组被显示。

AdvancedDataGrid 默认使用的是 AdvancedDataGridGroupItemRenderer 来显示层级数据。这个 itemRenderer 显示目录和父节点的图标。这里用 AdvancedDataGrid.groupItemRenderer 代替 AdvancedDataGridItemRenderer 可隐藏群组图标，下面是完整的例子：

Code View:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute" width="460" height="428" >

<mx:Script>
    <![CDATA[
        import
        mx.controls.advancedDataGridClasses.AdvancedDataGridItemRenderer;
        import mx.collections.IGroupingCollection;
        import
        mx.controls.advancedDataGridClasses.AdvancedDataGridColumn;
        import mx.collections.GroupingField;
```

```

import mx.collections.Grouping;
import mx.collections.ArrayCollection;
import mx.collections.GroupingCollection;
var flatData:ArrayCollection = new ArrayCollection(
    [
        {
            Region:"Southwest", Territory:"Arizona" ,
            Territory_Rep:"Barbara Jennings", Estimate:40000 ,
            Actual:38865 },
        {
            Region:"Southwest", Territory:"Arizona" ,
            Territory_Rep:"Dana Binn", Estimate:30000 ,
            Actual:29885 },
        {
            Region:"Southwest", Territory:"Central California" ,
            Territory_Rep:"Joe Schmoe", Estimate:30000 ,
            Actual:29134 },
        {
            Region:"Southwest", Territory:"Northern California" ,
            Territory_Rep:"Lauren Ipsum", Estimate:40000 ,
            Actual:38805 },
        {
            Region:"Southwest", Territory:"Northern California" ,
            Territory_Rep:"T.R. Smith", Estimate:40000 ,
            Actual:55498 },
        {
            Region:"Southwest", Territory:"Southern California" ,
            Territory_Rep:"Jane Grove", Estimate:45000 ,
            Actual:44913 },
        {
            Region:"Southwest", Territory:"Southern California" ,
            Territory_Rep:"Alice Treu", Estimate:45000 ,
            Actual:44985 },
        {
            Region:"Southwest", Territory:"Nevada" ,
            Territory_Rep:"Bethany Pittman", Estimate:45000 ,
            Actual:52888 }
    ]
);

```

这里AdvancedDataGrid的styleFunction属性是用来格式化显示具有summary属性的itemRenderers:

Code View:

```

private function formatSummary(data:Object,
col:AdvancedDataGridColumn):Object
{
    if (data.hasOwnProperty("summary"))
    {
        return { color:0xFF0000, fontWeight:"bold",
            fontSize:12 };
    }
}

```

```

        return {};
    }

private function flatSummaryObject():Object
{
    return { Territory_Rep:"Total", summary:true };
}

] ]>
</mx:Script>

<mx:AdvancedDataGrid id="adg"
creationComplete="groupedData.refresh();
adg.dataProvider.showRoot=false
groupItemRenderer="mx.controls.advancedDataGridClasses.
    AdvancedDataGridItemRenderer"
x="30" y="30" width="400" height="377"
styleFunction="formatSummary">
<mx:dataProvider>
<mx:GroupingCollection id="groupedData" source="{flatData}">
<
<mx:Grouping>
    <!-- use some dummy field and set showRoot=false for
the ADG dataProvider -->
    <mx:GroupingField name="fieldNameNotPresent" >
        <mx:summaries>
            <!-- use the summaryObjectFunction to return a
custom object which can then be used in the format function to
detect a summary row -->
            <mx:SummaryRow summaryPlacement="last"
                summaryObjectFunction="flatSummaryObject">
                <mx:fields>
                    <mx:SummaryField dataField="Estimate" />
                    <mx:SummaryField dataField="Actual" />
                </mx:fields>
            </mx:SummaryRow>
        </mx:summaries>
    </mx:GroupingField>
</mx:Grouping>
</mx:GroupingCollection>

</mx:dataProvider>

<mx:groupedColumns>

```

```

<mx:AdvancedDataGridColumn headerText = "Territory Rep"
    dataField="Territory_Rep" />

<mx:AdvancedDataGridColumnGroup headerText="Sales Figures"
    textAlign="center">
    <mx:AdvancedDataGridColumn headerText = "Estimate"
        textAlign="center" dataField="Estimate"
        width="100" />

    <mx:AdvancedDataGridColumn headerText = "Actual"
        textAlign="center" dataField="Actual"
        width="100" />
</mx:AdvancedDataGridColumnGroup>
</mx:groupedColumns>
</mx:AdvancedDataGrid>

</mx:Application>

```

## 6.12节. 为 GroupingCollection 创建异步刷新

### 6.12.1. 问题

我想异步刷新大数据量的 GroupingCollection 表格，只在调用时重绘。

### 6.12.2. 解决办法

使用 `GroupingCollection.refresh(async:Boolean)`，设置异步标志为 true。

### 6.12.3. 讨论

`GroupingCollection.refresh` 方法接受一个标志指示分组是否需要异步或同步执行。当数据行数量非常巨大时，设置标志为 true 可在显示之前刷新分组结果。这通常用来解决避免当 `GroupingCollection.refresh` 调用需花费很长时间而导致 Flash Player 响应超时问题。

分组的异步生成在用户需要与分组项交互时非常有用。`GroupingCollection.cancelRefresh` 可以停止正在进行的分组，并根据用户的输入开始新的分组。

下面的例子中，点击 `populateADGButton` 按钮生成随机数据显示在 `AdvancedDataGrid`。你可以修改数字分档器生成随机数据行数量，点击 `Group` 按钮开始异步刷新。`AdvancedDataGrid`

会立即显示分组结果。你可以在任何时候点击 Cancel Grouping 按钮取消分组。

Code View:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical" width="520" height="440">
<mx:Script>
    <![CDATA[
        import mx.controls.Alert;
        import mx.collections.IGroupingCollection;
        import mx.collections.GroupingField;
        import mx.collections.Grouping;
        import mx.collections.GroupingCollection;
        [Bindable]
        private var generatedData:Array = [];
        private var companyNames:Array = ["Adobe", "BEA", "Cosmos",
"Dogma", "Enigma", "Fury", "Gama", "Hima", "Indian", "Jaadu",
"Karish", "Linovo", "Micro", "Novice", "Oyster", "Puple", "Quag",
"Rendi", "Scrup", "Tempt", "Ubiquit", "Verna", "Wision", "Xeno",
"Yoga", "Zeal" ];

        private var products:Array = [ "Infuse", "MaxVis", "Fusion",
"Horizon", "Apex", "Zeeta", "Maza", "Orion", "Omega", "Zoota",
"Quata", "Morion" ];

        private var countries:Array = [ "India", "USA", "Canada",
"China", "Japan", "France", "Germany", "UK", "Brazil", "Italy",
"Chile", "Bhutan", "Sri Lanka" ];

        private var years:Array = [ "2000", "2001", "2002", "2003",
"2004", "2005", "2006", "2007", "2008", "2009", "2010", "2011",
"2012", "2013", "2014", "2015", "2016", "2017", "2018", "2019",
"2020", "2021", "2022", "2023", "2024" ];
        private var quarters:Array = [ "Q1", "Q2", "Q3", "Q4" ];
        private var months:Array = [ "Jan", "Feb", "Mar", "Apr",
"May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" ];
        private var sales:Array = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ];
        private var costs:Array = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ];
        private var dimNameMatch:Object = { Company:companyNames,
Product:products, Country:countries, Year:years, Quarter:quarters,
Month:months, Sales:sales, Cost:costs };

        private function generateData():void
        {
            generatedData = [];

```

```

var length:int = numRows.value;
var dimNameMap:Object = dimNameMatch;
for (var index:int = 0; index < length; ++index)
{
    var newObj:Object = {};
    for (var prop:String in dimNameMap)
    {
        var input:Array = dimNameMap[prop];
        var inputIndex:int = Math.random()*input.length;
        newObj[prop] = input[inputIndex];
    }
    generatedData.push(newObj);
}
}

private function populateADG():void
{
    if (generatedData.length != numRows.value)
        generateData();
    adg.dataProvider = generatedData;
}

[Bindable]
private var gc:GroupingCollection;
private function groupData():void
{
    var fields:Array = [];
    if (company.selected)
        fields.push(new GroupingField("Company"));

    if (product.selected)
        fields.push(new GroupingField("Product"));

    if (year.selected)
        fields.push(new GroupingField("Year"));

    if (fields.length == 0)
    {
        Alert.show("Select at least one of the items to
                    group on");
        return;
    }

    gc = new GroupingCollection();
}

```

```

        gc.source = generatedData;
        gc.grouping = new Grouping();
        gc.grouping.fields = fields;
        //use async refresh so that we get to see the results early.
        gc.refresh(true);
        adg.dataProvider = gc;
    }

private function handleOptionChange():void
{
    //user has not started grouping yet
    if (!gc)
        return;

    //stop any refresh that might be going on
    gc.cancelRefresh();

    var fields:Array = [];
    if (company.selected)
        fields.push(new GroupingField("Company"));

    if (product.selected)
        fields.push(new GroupingField("Product"));

    if (year.selected)
        fields.push(new GroupingField("Year"));

    //user might have checked off everything
    if (fields.length == 0)
    {
        return;
    }
    gc.grouping.fields = fields;
    gc.refresh(true);
}

] ]>
</mx:Script>

<mx:AdvancedDataGrid id="adg" width="100%" height="260" >
<mx:columns>
    <mx:AdvancedGridColumn dataField="Company" />
    <mx:AdvancedGridColumn dataField="Product" />

```

```

<mx:AdvancedDataGridColumn dataField="Year" />
<mx:AdvancedDataGridColumn dataField="Sales" />
</mx:columns>
</mx:AdvancedDataGrid>
<mx:HBox>
<mx:NumericStepper id="numRows" stepSize="1000" minimum="1000"
maximum="10000" />
<mx:Button label="Populate ADG" click="populateADG()" id="populateADGButton"/>
</mx:HBox>
<mx:VBox>
<mx:HBox>
<mx:Label text="Grouping fields:" />

<mx:CheckBox id="company" label="Company" selected="true"
click="handleOptionChange()"/>
<mx:CheckBox id="product" label="Product"
click="handleOptionChange()"/>
<mx:CheckBox id="year" label="Year"
click="handleOptionChange()"/>
</mx:HBox>
<mx:HBox>
<mx:Button label="Group" click="groupData()"/>
<mx:Button label="Cancel grouping" click="gc.cancelRefresh()"
enabled="{gc != null}"/>
</mx:HBox>
</mx:VBox>
</mx:Application>

```

三个复选框允许执行不同的分组组合。当刷新在进行时用户可以直接改变分组选择，GroupCollection 的 cancelRefresh 方法用于停止 AdvancedDataGrid 创建和显示新的分组。

## 第七章：渲染器和编辑器 (王平)

renderer (渲染器), 或 item renderer, 是 flex 框架一个功能强大的特性,它的使用频率很高,通过它您可以用自定义的组件显示数组或集合的数据。渲染器可用于 DataGrid, List, Tile, 和 ComboBox 这些数据容器。渲染器可以为这些容器的数据源中的每一个元素的数据设置渲染样式 (即显示形式),从而控制这些数据的显示与更新。在 Flex 的应用当中会经常用到表格或列表，搞清楚如何更好的、更有效的显示和编辑这些数据是非常有用的。

如何使用 item renderers 和 item editors (单项的渲染器和单项的编辑器)关键在于了解 item renderers 与包含他的组件 (即父组件) 之间的关系。所有 item renderers 都有一个”data”属性, 这个属性是由 dataProvider 中与 itemRenderer 所在行相应的记录所设定的。如何显示这些数据完全取决于开发; 例如用户可以使用“drop-in”的 item renderers 也可以用自定义的组件。item renderers 允许让用户修改 data 属性中的数据, 但这一操作会自动修改父组件中数据源的数据。(注: drop-in 是指一些已实现 IDropInListItemRenderer 接口的组件, 现有组件有: Button CheckBox Image Label Text 等。用户自己实现 IDropInListItemRenderer 的组件, 也可以算是 drop-in 。如果想深研, 大家可以看:

[http://livedocs.adobe.com/flex/3/html/help.html?content=cellrenderer\\_5.html](http://livedocs.adobe.com/flex/3/html/help.html?content=cellrenderer_5.html) 如果不想深研, 这里可以理解成 Button CheckBox 等这些 flex 自带组件很多网上的文章直译为下拉式组件, 是不对的。)

item editor (编辑器) 的功能又是与前者截然不同的, 基于 mx.controls.list 类 的一个 list 当它的 item renderer 组件被点击后, 会实例化一个 editor item renderer 组件。当这个 editor 失去焦点时, list 会试图去读取 editorDataField 属性并与之前的 data 值比较, 如果有变化就把 list 的数据源更新, 并把 itemEditor 消毁并用 itemRender 替换。这意味着 list 或 DataGrid 的行始终只能有一个 itemRenderer。(我们在实际应用时, 可能见不到 item editor, 那是因为 list 的编辑状态未打开。)

### 7.1 节. 创建自己的渲染器

#### 7.1.1.问题

你想要为 List 或 DataGrid 建 item renderers 。

#### 7.1.2.解决办法

你可以定义一个 item renderer 在 MXML 的 List 组件内部。或定义在另一个类文件中, 再把这个类名指定在 List 的 itemRenderer 属性里。

#### 7.1.3.讨论

有很多办法可以创建 item renderers 最简单的办法是: 作为 itemRenderer 属性定义在父组件的内部:

```

<mx>List dataProvider="{simpleArray}">
    <mx:itemRenderer>
        <mx:Component>
            <mx:Label color="#007700"/>
        </mx:Component>
    </mx:itemRenderer>
</mx>List>

```

Flex 会为 List 的数据源（类型可能是数组或类似数组的数据类型）中的每个元素实例化一个 itemRenderer 组件（事实是，只为显示的数据创建 itemRenderer），并且会把数据源中对应的元素赋与 itemRenderer 的 data 属性。上边的例子我们使用了 Label 组件，虽然我们没有设置 Label 的 text 属性，但依然有显示，是因为父组件会把对应的元素传给 Label 的 data 属性，Label 会默认显示 data 的值。所以 List 的数据源中的每一个元素最好是简单的 String 类型。如果一个复杂的对象或其他类型的数据传递给 Label，你在结果中会看到：[Object object]。

如果一定要使用一个复杂类型，你最好自己写一个新的 item renderer 并且重写他的 setData 方法。例如：

```

<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" height="800">
    <mx:Script>
        <![CDATA[
            [Bindable]
            private var simpleArray:Array = new Array("one", "two",
                "three", "four", "five");
        ]]>
    </mx:Script>
    <!-- this list can use an inline item renderer because the
array that the List is
using as a dataprovider consists
of simple object -->
    <mx>List dataProvider="{simpleArray}">
        <mx:itemRenderer>
            <mx:Component>
                <mx:Label color="#007700"/>
            </mx:Component>
        </mx:itemRenderer>
    </mx>List>
    <!-- the list component here requires a custom renderer
because we're passing any
object to it, which
means that the renderer will need to know how to handle
each field in the
item -->

```

```

<mx>List itemRenderer="oreilly.cookbook.SevenOneRenderer"
dataProvider= "{DataHolder.genericCollectionOne}" />

    <!-- here we can use a proper drop in item renderer because
the DataGrid handles
each field in the Object
separately -->
<mx:DataGrid dataProvider="{DataHolder.genericCollectionOne}">
    <mx:columns>
        <mx:DataGridColumn dataField="name"/>
        <mx:DataGridColumn dataField="age"/>
        <mx:DataGridColumn dataField="appearance" width="200">
            <mx:itemRenderer>
                <mx:Component>
                    <!-- note that any component placed here must
extend
IDataRenderer and have a data property that properly displays any
data passed to the data
setter method -->
<!-- 原文这里容易让人误解，其实把这里的 and 换成 or 就对了，因为这里的组件不
extend IDataRenderer也可以，只要它有data属性就可以。 -->
                    <mx:TextArea/>
                </mx:Component>
            </mx:itemRenderer>
        </mx:DataGridColumn>
    </mx:columns>
</mx:DataGrid>
</mx:VBox>

```

如果是复杂的数据，您需要创建一个自定义的 renderer，flex 没有组件可以通过内置一个 Component 标签来处理像 data 这样的复杂数据。不过这并不是个问题，因为我们通常在处理这种需求时使用 DataGrid，DataGrid 为每个列对象单独指定 renderer，每个 renderer 又用于处理 data 中的一个属性。但这种方式并不是满足所有情况，有些场合我们并不能使用 DataGrid，所以在使用 DataGrid 以外组件时为了处理好这些拥有多种类型属性的数据对象，你需要写 data 的 set 方法并且去处理传入对象的每个属性。大家请看下边的代码，在 set data 方法里，将 data 中感兴趣的属性取出来，并传到 Label 里，即使 data 中没有某个属性值，这种方法也是可以工作的。

```

package oreilly.cookbook
{
    import mx.containers.HBox;
    import mx.controls.Label;
    import mx.core.IDataRenderer;
    public class SevenOneRenderer extends HBox

```

```

{
    private var nameLabel:Label;
    private var ageLabel:Label;
    private var appearanceLabel:Label;
    private var _data:Object;
    public function SevenOneRenderer() {
        super();
    }
    override public function get data():Object {
        if(_data != null) {
            return _data;
        }
        return null;
    }
    override public function set data(value:Object):void {
        _data = value;
        if(_data.name != null) {
            nameLabel = instantiateNewLabel(_data.name);
        }
        if(_data.age != null) {
            ageLabel = instantiateNewLabel(_data.age);
        }
        if(_data.appearance != null) {
            appearanceLabel =
                instantiateNewLabel(_data.appearance);
        }
        setStyle("backgroundColor", 0xddddff);
    }
    private function instantiateNewLabel(value:*):Label {
        var label:Label = new Label();
        label.text = String(value);
        addChild(label);
        return label;
    }
}
}

```

## 7.2 节. 使用 ClassFactory 生成渲染器

### 7.2.1. 问题

你想要在运行时改变 List 或 DataGridColumn 的渲染器，或改变渲染器的属性。

## 7.2.2. 解决办法

使用一个 ClassFactory 对象做为 ItemRenderer。ClassFactory 实现了 IFactory 接口，你只要传一个类对象（即 Class 类型的对象）给它，它就可以生成这个类的对象。

## 7.2.3. 讨论

这种解决办法，使用了工厂设计模式，并且使用了 flex 的一些内部机制。在使用工厂模式时，你需要指定一个类型给工厂，这个类型的对象会被工厂类产生并做为渲染器使用，并且工厂类可以控制渲染器的生命周期。那么我们如何把类型指定给工厂类呢？我们需要传一个类对象给工厂类，正如下边的代码：

```
var factory:ClassFactory = new ClassFactory(oreilly.cookbook.SevenTwoFactory);
```

在这里原文提到了 SevenTwoFactory 继承 IFactory 等等，大家需要多了解一些 ClassFactory 的相关知识，在实际操作中，我们一般都是创建 ClassFactory 的实例赋值 itemRenderer，因为 set itemRenderer 方法需要的参数是 IFactory 类型，系统最终会用这个 factory 对象 实例我们的 renderer。也就是说，我们给系统的不是一个 renderer 而是一个能产生 renderer 的 factory，在系统需要时自己用 factory 产生 renderer 的实例，也许细心的读者会发现，当我们写 mxml 的时候，itemRenderer 的属性可以赋值一个类路径，这就是因为 flex 对 mxml 里的 itemRenderer 做了一些特殊的功能，如果你传的是一个类路径，系统帮你创建一个 ClassFactory。, 在本节的例子中其实在我们的应用中 SevenTwoFactory 只需是一个普通的类就 OK 了，我看了本节后边 SevenTwoFactory 的代码，觉得作者这样做没有必要，所以还是不扯 SevenTwoFactory 和 IFactory 的关系了，不然反倒让大家糊涂。在这里我们只需要知道，ClassFactory 的实例，赋值给 list 或 DataGrid 的 itemRenderer 属性，系统会帮我们产生 item renderer。

```
<cookbook:PurgeList id="list" itemRenderer="{factory}" width="300"/>
```

在下边的范例中，提供了一个方法用于访问 ListBase 中的一个 protected 方法 purgeItemRenderers，其实原文中并没有提到 purgeItemRenderers 的用法和出处，所以我个人认为，作者写在这里让人觉得很突兀，其实 purgeItemRenderers 是 ListBase 的一个 protected 的方法。这个方法主要是用于清除所有 ItemRenderer 和一些相当的信息，其实这个方法是应用于 ListBase 自己的内部机制，它会在 ItemRenderer 被替换时调用。因为 ListBase 是 Tree、List、DataGrid 等这些类的父类，所以这些类也继承了此方法。其实大家可以不用看下边这段代码，我在此节最后提供了一段可以运行的简单例子，比原文的例子更具代表性，更简单。还有，既然原文这里提到了 tree 我们就顺便提一下 tree 它的 item renderer 不象 list dataGrid 那样。Tree 的 item renderer 必须 extends TreeItemRenderer，这些，我们将在本章的 12 节细讲。

```
public function clearList():void{
    this.purgeItemRenderers();
    this.invalidateDisplayList();
```

```
}
```

我们看一下，ClassFactory 是如何工作的。

Code View:

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml"
    initialize="setType()"
    xmlns:cookbook="oreilly.cookbook.*">
    <mx:Script>
        <![CDATA[
            [Bindable]
            private var factory:ClassFactory;
            private function setType():void
            {
                factory = new ClassFactory(SevenTwoFactory);
                factory.properties={type:SevenTwoFactory.HORIZONTAL};
            }
        ]]>
    </mx:Script>
    <cookbook:PurgeList id="list"
        itemRenderer="{factory}" width="300"/>
    <!-- toggle between the horizontal and vertical item rendering
-- >
    <mx:Button id="toggleStyle" toggle="true"
        click="toggleStyle.selected ?
            factory.properties = {type:SevenTwoFactory.VERTICAL} :
            factory.properties={type:SevenTwoFactory.HORIZONTAL}"
        label="style"/>
    <!-- here we redraw the whole list to use our new renderers --
>
    <mx:Button id="toggleDP" toggle="true" click="list.clearList(),
        list.dataProvider = DataHolder.genericCollectionOne,
        this.invalidateDisplayList()" label="generate"/>

</mx:HBox>
```

ClassFactory 会把 properties 对象里的数据传给它生成的对象。 IFactory 这个接口只有一个方法，如下是这个方法的一个实现：

```
public function newInstance():*
{
    return new MyClass();
}
```

每当需要一个新对象时，ClassFactory 就会调用这个方法。ClassFactory 会负责把一个初始化的数据传入新建的对象。上边的例子就是这样做的：

```
factory.properties = {type:SevenTwoFactory.HORIZONTAL};
```

在这个新建对象的类中有一个 `setter` 方法捕获这个操作：

```
public function set type(value:String):void {
    _type = value;
    invalidateDisplayList();
}
```

ClassFactory 会试图把 properties 里的所有属性都赋给它新产生的对象里。这就是为什么新的数据可以在运行时置入 item renderer。你可以利用这一点特性，传一些你感兴趣的数据给 item renderer。在这里原文见意我们写 renderer 的时候尽量不要继承 mx.containers 下的那些容器，如 HBox VBox Canvas 等，作者见意我们继承 UIComponent，这样会使 renderer 效率高很多。同时还提到下边例子中用 flash.text.TextField 替代 mx.controls.Text 和 Label 也是同样的目地。作者同时还建议我们在写程序时尽量多使用一些低级别、轻量级的组件从而提高我们程序的性能。之前我在网上也看到一篇关于提高 itemRenderer 性能的文章，也提到了这些。

Code View:

```
package oreilly.cookbook
{
    import flash.text.TextField;
    import mx.controls.Label;
    import mx.controls.listClasses.IListItemRenderer;
    import mx.core.IFactory;
    import mx.core.UIComponent;
    import mx.events.FlexEvent;
    //the class implements IListItemRenderer to ensure that it will
    //function
    //properly with the List, and IFactory, to ensure that it can be
    //used in
    //a factory
    public class SevenTwoFactory extends UIComponent implements
        IFactory, IListItemRenderer
    {
        //here are our two layout types that we'll use to the
        'type'
        public static const HORIZONTAL:String = "horizontal";
        public static const VERTICAL:String = "vertical";
        //by default we'll go with horizontal
        private var _type:String = HORIZONTAL;
        private var _data:Object;
```

```

//here are our three TextFields
private var nameLabel:TextField;
private var ageLabel:TextField;
private var appearanceLabel:TextField;
//this is the property we'll set in the properties of the
//ClassFactory
public function set type(value:String):void {
    _type = value;
    invalidateDisplayList();
}
public function get data():Object {
    return _data;
}
// we need to do this to determine the correct size of the
renderer
override protected function measure():void {
    super.measure();
    if(this._type == HORIZONTAL) {
        measuredHeight = nameLabel.height;
        measuredWidth = nameLabel.width + ageLabel.width +
            appearanceLabel.width + 10;
    } else {
        measuredWidth = appearanceLabel.width;
        measuredHeight = nameLabel.height + ageLabel.height
            + appearanceLabel.height + 10;
    }
    height = measuredHeight;
    width = measuredWidth;
    trace(" w "+this.measuredWidth+ " "+
        this.measuredHeight+" "+this.width+
        "+this.height");
}
//set all the TextFields with the correct data by parsing
out
// the data value. We create the TextField only if we need
it.
public function set data(value:Object):void {
    _data = value;
    if(_data.name != null && nameLabel == null) {
        nameLabel = instantiateNewLabel(_data.name);
    } else {
        nameLabel.text = _data.name;
    }
    if(_data.age != null && ageLabel == null) {

```

```

        ageLabel = instantiateNewLabel(_data.age);
    } else {
        ageLabel.text = _data.age;
    }
if (_data.appearance != null && appearanceLabel == null)
{
    appearanceLabel =
        instantiateNewLabel(_data.appearance);
} else {
    appearanceLabel.text = _data.appearance;
}
setStyle("backgroundColor", 0xddddff);
invalidateProperties();
dispatchEvent(new FlexEvent(FlexEvent.DATA_CHANGE));
}

//Since the UIComponent doesn't possess any layout logic,
we need to
//do all of that ourselves
override protected function updateDisplayList(
unscaledWidth:Number,unscaledHeight:Number) :void
{
    super.updateDisplayList(unscaledWidth, unscaledHeight);
    var sum:Number = 0;
    if(this._type == HORIZONTAL) {
        nameLabel.x = 0;
        sum += nameLabel.width;
        ageLabel.x = sum;
        sum += ageLabel.width;
        appearanceLabel.x = sum;
    } else {
        nameLabel.y = 0;
        sum += nameLabel.height;
        ageLabel.y = sum;
        sum += ageLabel.height;
        appearanceLabel.y = sum;
    }
}
private function instantiateNewLabel(value:*) :TextField {
    var text:TextField = new TextField();
    addChild(text);
    text.text = String(value);
    text.width = text.textWidth + 5;
    text.height = text.textHeight + 5;
    return text;
}

```

```

        }
        // here, finally is the Factory method, that will return
        // a new instance for each renderer needed
    public function newInstance():* {
        return new SevenTwoFactory();
    }
}
}
}

```

下边是我为大家准备的一个简单的小例子：

DynamicRenderer.mxml:

```

<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute">
<mx:Script>
<![CDATA[
    public function changeRenderer():void{
        if (btnChange.label == "change to big"){
            list.itemRenderer = new
                ClassFactory(BigIconRenderer);
            //这里正如我在上边所说，这里只要传一个普通类就可以了，但实际开发中我们这里一般
            //都传一个自定义组件，或类似的东西。
            btnChange.label = "change to small";
        }else{
            var cf:ClassFactory = new
                ClassFactory(SmallIconRenderer);
            //在下边的代码中我实例大家如何利用ClassFactory的properties属性给新生产的对
            //象赋初值。
            //cf.properties = {xx:"heihei"};
            var o:Object = new Object();
            o.xx = "xixi";
            cf.properties = o;
            //上边的写法，和上边注掉的写法都是可以的。但我觉得更负责的写法
        是:
            //if(cf.properties == null){
            // cf.properties = new Object();
            //}
            //o.xx = "hehe";

            list.itemRenderer = cf;
            btnChange.label = "change to big";
            var c:Class = SmallIconRenderer;
            var x:Object = new c();

```

```

        }
    }
]]>
</mx:Script>
<mx>List id="list" width="300" height="300"
    itemRenderer="SmallIconRenderer" variableRowHeight="true">
<mx:dataProvider>
<mx:Object label="this is item A"/>
<mx:Object label="this is item B"/>
<mx:Object label="this is item C"/>
<mx:Object label="this is item D"/>
</mx:dataProvider>
</mx>List>
<mx:Button id="btnChange" label="change to big"
    click="changeRenderer()" x="308" y="10"/>
</mx:Application>
```

SmallIconRenderer.mxml

```

<?xml version="1.0" encoding="utf-8"?>
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:Script>
<! [CDATA[
[Bindable]
public var item:Object = new Object();
[Bindable]
public var xx:String;

]]>
</mx:Script>
<mx:Label text="{data.label}" />
<mx:Text text="小"/>
<mx:Button label="{xx}" />
</mx:HBox>
```

BigIconRenderer.mxml

```

<?xml version="1.0" encoding="utf-8"?>
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:Script>
<! [CDATA[
[Bindable]
```

```

public var item:Object = new Object();
] ]>
</mx:Script>
<mx:Label text="{data.label}" fontSize="20" fontWeight="bold"
color="0xff0000"/>
<mx:Text text="大"/>
</mx:HBox>

```

## 7.3 节. 访问设置自己渲染器的组件

### 7.3.1. 问题

item renderer 需要访问它的父组件的一些数据。

### 7.3.2. 解决办法

实现 IDropInListItemRenderer，通过 drop-in renderer 访问它的父组件.

### 7.3.3. 讨论

实现了 IDropInListItemRenderer 的 renderer 不但能访问 data 等一些传入的数据, 还能通过 BaseListData类型的一个属性访问 renderer 的父组件(如 List 或 DataGridColumn)。mx. controls. listClasses. BaseListData 有如下属性:

`columnIndex : int`

当前是第几列, 当用户用 mouse 点击 List 或 DataGrid 某个元素时这个属性被赋值。第一列是 1。（注意，这个值的含意是指选中列是当前可见表格中的第几行，如果有横向的滚动条时要小心使用此属性）（原文中指出第一列是 1，但事实是 0）

`owner : IUIComponent`

这个 drop-in itemRenderer 的所有者, 即一个 List 对象, 或 DataGridColumn 对象。(我认为原文这里是写错了, owner 不可能是 DataGridColumn 对象, 它只能是一个从 ListBase 扩展的类型, 我想作者这里是想写 DataGrid)

`rowIndex : int`

当前是第几行, 类似 columnIndex (这里要注意, 这个属性对于 Tree 有特殊含意, 这个值是根据当前所有展开结点排列所得。) 第一行是 1。 (原文中指出第一行是 1, 但事实是 0)

```
uid : String
```

用于唯一标识一个 renderer 对象。

如果 itemRenderer 实现了 IDropInListItemRenderer 接口，即 itemRenderer 会有 set listData 和 get listData 两个方法，当这个 itemRenderer 被使用时，它会被实例化并利用 set listData 传数据给 itemRenderer，这个数据就是我们在前边提到的，BaseListData 类型的对象。在调用 renderer 的 set data 方法时我们可以根据 listData 来得到拥有这个 renderer 的容器，并通过一些判断来决定对 data 的处理。在这个过程中，我们还可以调用自己的一些方法。下面我们通过例子说明 listData 的用法。

Code View:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="50" implements=
"mx.controls.listClasses.IDropInListItemRenderer">
<mx:Script>
<! [CDATA[
    import mx.controls.DataGrid;
    import mx.controls.List;
    import mx.controls.dataGridColumn;
    import mx.controls.listClasses.BaseListData;

    // store the list data item in case we want to do
    something with it
    // later on
    private var _listData:BaseListData;
    [Bindable("dataChange")]
    // a getter method.
    public function get listData():BaseListData
    {
        return _listData;
    }
    // a setter method,
```

当 BaseListData 数据被传入之后 renderer 就可以通过它自己的一些属性访问拥有它的父组件中元素的数据，甚至可以调用父组件中的方法。

Code View:

```
public function set listData(value:BaseListData):void
{
    _listData = value;
    if(value.owner is DataGridColumn) {
        trace(" DataGridColumn ");
    }
}
```

```

        } else if (value.owner is List) {
            trace(" List ");
        } else if (value.owner is CustomDataGrid) {
            trace(" CustomDataGrid ");
            (value.owner as
                CustomDataGrid).checkInMethod(this);
        }
    }

override public function set data(value:Object):void {
    nameTxt.text = value.name;
    appearanceTxt.text = value.appearance;
}

] ]>
</mx:Script>
<mx:Canvas backgroundColor="#3344ff">
    <mx:Label id="nameTxt"/>
</mx:Canvas>
<mx:Label id="appearanceTxt"/>
</mx:VBox>

```

下边是一个 IDropInListRenderer 应用于一个用户自定义的 DataGrid:

Code View:

```

<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:cookbook="oreilly.cookbook.*">
    <mx>List itemRenderer="oreilly.cookbook.IDropInListRenderer"
       dataProvider="{DataHolder.genericCollectionOne}" />
        <!-- note that creating a custom component with a new
        namespace, cookbook,
        requires that our columns
            are declared w/in the cookbook namespace -->
        <cookbook:CustomDataGrid
            dataProvider="{DataHolder.genericCollectionOne}">
            <cookbook:columns>
                <!-- since we're not declaring a dataField, the entire
                data object
                    is passed to the renderer -->
                <mx:DataGridColumn
                    itemRenderer="oreilly.cookbook.IDropInListRenderer"/>
                    <mx:DataGridColumn dataField="age"/>
                </cookbook:columns>
            </cookbook:CustomDataGrid>
        </mx:HBox>

```

下边是用户自定义的 DataGrid，它有一个方法，checkInMethod，这个方法是在 renderer 的 data 属性被赋值时调用的。

```
<mx:DataGrid xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:Script>
<! [CDATA[

    public function checkInMethod(obj:*):void {
        trace(" hello from my renderer "+obj);
    }

]]>
</mx:Script>
</mx:DataGrid>
```

### (译者注)

看完这节，大家可能还有些糊涂，我给大家总结一下：在第一节中我们说过，如果一个 renderer 实现了 IDropInListItemRenderer 接口，那么它就算是 drop-in renderer 了。IDropInListItemRenderer 接口中有两个方法：set listData 和 get listData。flex 对于 renderer 有一种优化的机制，就是：页面中显示几个 renderer 系统才实例几个 renderer，不显示的就销毁，有新的显示，如果重用也不够时，再 new。比如有一个 DataGrid，它的数据源中有 10 个对象，但页面中的 DataGrid 一次只显示 5 个（即有滚动条），也就是说系统在第一次显示这个 DataGrid 的时候一次性 new 5 个 renderer。当用户向下拖动滚动条时，下方会出现一条新数据，这时会再 new 或重用一个 renderer，上方那个移到显示区外的数据对应的 renderer 会被重用。当系统 new 或重用一个 renderer 之后如果这个 renderer 是 drop-in renderer，系统会调用 set listData 方法，传入一个 BaseListData 类型的数据，BaseListData 数据中含有一个很有用的属性 owner，它是 renderer 所在容器对象的一个引用，通过它我们可以得到容器的相关信息。接下来会调用 set data 方法，把 data 传入 renderer。这时我们可以利用之前的 BaseListData 数据中的信息控制 data 数据。

本节内容相对来说比较少，通过我填加的一些信息，希望大家能更容易理解。

## 7.4. 节. 创建一个简单的组件作为渲染器和编辑器

### 7.4.1. 问题

你需要建立一个即可以用于 item editor 又可以用于 item renderer 的组件，并且 item editor 会收到一个数组，我们希望这个数组能以 combo box 的形式显示出来。

### 7.4.2. 解决办法

建立一个已经实现了 IDropInListItemRenderer 的组件，并在它里边定义两个 states 即状态，一个状态用于显示 editor 另一个用于显示 renderer。当 data 发生变化时，即在 editor 状态用户做了修改操作，系统将会派发一个 ITEM\_EDIT\_END 事件到这个组件的父组件，父组件收到这个事件后，保存修改的数据到 dataProvider 里。

#### 7.4.3. 讨论

把一个 item renderer 改装成一个 item editor 很容易，在 List 这个类里有一个 rendererIsEditor 属性，如果这个属性为 true，当 itemRenderer 被双击时，便进入编辑状态，List 并没有建一个 item editor 而是使用 itemRenderer 监听并捕获了 edit 的事件。

原文中提到 DataGridColumn 也有 rendererIsEditor 属性，这是对的，但原文中说 DataGridColumn 是继承自 List，我看了这两个类，它们之间并没有这层关系，它们各自定义了 rendererIsEditor 属性。

当用户完成修改后即更新 dataProvider，关键在于修改完成后 renderer 要派发一个 ITEM\_EDIT\_END 事件。在下边的例子中使用 DataGridEvent 传递被修改单元格所在行数和列数，从而确定需要修改 dataProvider 中哪些数据。DataGridEvent 对象是在 setNewData 方法中建立并派发的。setNewData 方法是在 ComboBox 的值发生变化时被调用。

Code View:

```
private function setNewData():void
{
    _data.selected = selectCB.selectedItem;
    dispatchEvent(new DataGridEvent(DataGridEvent.ITEM_EDIT_END,
        true, true, _listData.columnIndex,
        'selected', _listData.rowIndex));
}
```

在接下来的例子中，这组数据将要用于 dataGrid 的显示，下面这组数据相当于在 item renderer 里得到的 data，它有几个基本属性：name、age、appearance，还有一个数组 extras，还有一个 selected 属性，用于标记 extras 数组中哪个元素被选中了，这些数据都是可以修改的。

Code View:

```
{name:"Todd Anderson", age:31, appearance:"Intimidating", extras:["bar", "foo", "baz"], selected:"bar"}
```

象这样的数据是很常见的。extras 属性中的数据是专门用做 ComboBox 数据源的，selected 是专门用于记录 ComboBox 选中项的。但是一般情况下，DataGrid 的 renderer 都只是传一个简单的数据。如果 DataGridColumn 没有明确指定 dataField 属性，整个 data 对象被传入 renderer。正如下边的样子。（原文在这里的描述很容易让人误解，其实不管你有没有指定 dataField 属性，都会把整个 data 传给 renderer。）

Code View:

```
<mx:DataGrid dataProvider="{DataHolder.genericCollectionTwo}"
```

```

width="450" itemEditEnd="checkEditedItem(event)" id="dg">
<mx:columns>
    <mx:DataGridColumn dataField="age"/>
    <mx:DataGridColumn dataField="appearance"/>
    <mx:DataGridColumn
        itemRenderer="oreilly.cookbook.ComboBoxRenderer"
        editable="true" rendererIsEditor="true"/>
</mx:columns>
</mx:DataGrid>

```

请注意 rendererIsEditor 的位置。

Code View:

```

<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="100" doubleClickEnabled="true" currentState="display"
implements="mx.controls.listClasses.IDropInListItemRenderer">
<mx:Script>
<! [CDATA[
    import mx.events.DataGridEvent;
    import mx.events.ListEvent;

    import mx.controls.listClasses.BaseListData;

    // Internal variable for the property value.
    private var _listData:BaseListData;
    private var _data:Object = {};

```

在 data 传给 renderer 之后, 这个组件开始监听鼠标的双击事件, 当它收到这个事件时, 它将进入 editing 状态:

Code View:

```

override public function set data(value:Object):void {
    _data = value;
    if (_data.selected != null) {
        addEventListener(MouseEvent.DOUBLE_CLICK,
            startEdit);
    }
}
override public function get data():Object {
    return _data;
}

```

```

}

private function startEdit(event:Event):void {
    if(currentState == "display") {
        currentState = "edit";
        addEventListener(FocusEvent.FOCUS_OUT, endEdit);
    }
}

private function endEdit(event:Event):void {
    currentState = "display";
}

// Make the listData property bindable.
[Bindable("dataChange")]
public function get listData():BaseListData {
    return _listData;
}

// Define the setter method,
public function set listData(value:BaseListData):void {
    _listData = value;
}

private function setNewData():void {
    _data.selected = selectCB.selectedItem;
    dispatchEvent(new
        DataGridEvent(DataGridEvent.ITEM_EDIT_END, true,
        true, _listData.columnIndex,
        'selected', _listData.rowIndex));
}

] ]>
</mx:Script>

```

在这个例子中，我们使用 states，使这个组件从一个只是简单显示数据的 item renderer 变成了可以修改数据 item editor，这个 item editor 含有一个 ComboBox：

Code View:

```

<mx:states>
    <mx:State name="edit">
        <mx:AddChild>
            <mx:ComboBox id="selectCB"
                addedToStage="selectCB.dataProvider =

```

```

        _data.extras;
        selectCB.selectedItem = _data.selected"
        change="setNewData()"/>
    </mx:AddChild>
</mx:State>
<mx:State name="display">
    <mx:AddChild>
        <mx:Text id="text" addedToStage="text.text =
            _data.selected"/>
    </mx:AddChild>
</mx:State>
</mx:states>
</mx:Canvas>

```

## 7.5 节. 创建一个 Item Editor, 它可以处理含有复杂数据类型的 Data

### 7.5.1. 问题

如果 item editor 收到的 data 是一个用户自定义对象，我们需要创建一个可以修改这个含有复杂类型对象的 item editor。

### 7.5.2. 解决办法

创建一个 item editor，它可以返回 data 中所有已经被修改过的属性。在 List 上创建一个监听器，用于对 itemEditEnd 事件的监听，在收到这个事件后要停止这个事件继续上父级派发（flex 冒泡的事件机制），并读取那个被修改过的 item editor 中的 data 属性。

### 7.5.3. 讨论

List 类里有一个属性叫 editorDataField, 如果你的 editor 处理的是单个属性, 使用它将是最简单最方便的选择. 但如果想用 editor 处理含有多个属性的复杂对象, 必须能够阻止 List 、 DataGridColumn 的一些原有动作和事件，并且可以通过 itemEditorInstance 读到 List 、 DataGridColumn 中的数据。

在下边的例子中， processData 这个方法是用于处理 itemEditEnd 事件的。

```

<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="700"
height="300">
    <mx:Script>
        <! [CDATA[
            import mx.events.ListEvent;
            import mx.collections.ArrayCollection;

```

```

import oreilly.cookbook.MultipleDataTypeEditor;

[Bindable]
Private var arr:ArrayCollection =
    new ArrayCollection([{age:12, name:"Joe"}, 
        {age:16, name:"Jorge"}, 
        {age:19, name:"Jojo"}, 
        {age:2, name:"James"}, 
        {age:12, name:"Joaquin"}]);

public function processData(event>ListEvent):void {
    // Disable copying data back to the control.
    event.preventDefault();
    // Get new label from editor.
    list.editedItemRenderer.data= MultipleDataTypeEditor
(list.itemEditorInstance).data;
    // Close the cell editor.
    list.destroyItemEditor();
    // Notify the list control to update its display.

list.dataProvider.notifyItemUpdate(list.editedItemRenderer);
}

]]>
</mx:Script>
<mx>List
    id="list"
    itemEditor="oreilly.cookbook.MultipleDataTypeEditor"
    dataProvider="{arr}"
    itemEditEnd="processData(event)"
    itemRenderer="oreilly.cookbook.MultipleDataTypeRenderer"
    width="350" editable="true">
</mx>List>

</mx:Canvas>

```

Event 的 preventDefault 方法，是用于终止 Event 中的默认动作，在这里我们用这个方法终止了 itemEditor 的默认动作，不再去试图从 itemEditor 里读取 text 属性。取而带之的是从 itemRendererInstance 里得到 data 赋值给 editedItemRenderers 的 data。最后调用 List dataProvider 的 notifyItemUpdate 方法，从而把用户的修改保存到 List 的数据源中 (dataProvider)。

上边代码中用到的 itemEditor，在 get data 方法中只是返回了两个 TextInput 的 text 属性，即用户的输入值。这个 itemEditor 的代码如下：

```

package oreilly.cookbook
{
    import mx.containers.Canvas;
    import mx.controls.TextInput;
    public class MultipleDataTypeEditor extends Canvas
    {
        private var nameField:TextInput;
        private var ageField:TextInput;
        public function MultipleDataTypeEditor() {
            super();
            nameField = new TextInput();
            ageField = new TextInput();
            addChild(nameField);
            nameField.focusEnabled = false;
            addChild(ageField);
            ageField.focusEnabled = false;
            ageField.x = 100;
            this.horizontalScrollPolicy = "none";
        }
        override public function set data(value:Object):void {
            super.data = value;
            nameField.text = value.name;
            ageField.text = value.age;
        }
        override public function get data():Object {
            return {name:nameField.text, age:ageField.text};
        }
    }
}

```

## (译者注)

原文译到这儿本节就已经结束了，如果你对 item editor、item renderer 理解的还算深刻，我想你应该很容易理解本节的内容，如果你对 item editor、item renderer 了解的还不够深入，您看完这一节，可能会有些糊涂，我想作者的用意就是想蜻蜓点水的把所有东西都讲给我们，读者再根据自己情况，深入学习。既然我翻译它，目地就是让大家能看明白，话不多说，我再用自己的语言给本节做个总结：

- 1 如果你只是想实现最简单的 editor（下面的例子都是以 DataGrid 为例）用户点击就进入修改状态，失去焦点就结束修改。（每个单元格只对应 data 中的一个单一属性）只需：

```

<mx:DataGrid id="myGrid"dataProvider="{initDG}" editable="true">
    <mx:columns>
        <mx:DataGridColumn dataField="Company" editable="false"/>
        <mx:DataGridColumn dataField="Contact"/>

```

关键在于 DataGrid 中的 `editable="true"` 这样表格中所有单元格都可以修改，如果你想使某一列不可被修改，则 `editable="false"`。

- 2 再复杂一些，如果你有了一个复杂的 editor 如：你想用 NumericStepper 来修改数值，这样做是很常用的，因为默认的 editor 是 label，很不友好。(当然这种情况还只是每个单元格只对应 data 中的一个单一属性。)这个时候你要用到 DataGridColumn 中的 `editorDataField` 属性，如：

```
<mx:DataGridColumn dataField="quantity" editorDataField="value">
    <mx:itemEditor>
        <mx:Component>
            <mx:NumericStepper maximum="100" stepSize="10"/>
        </mx:Component>
    </mx:itemEditor>
</mx:DataGridColumn>
```

`editorDataField` 此属性的用意是：当用户修改完毕，系统并不知道去 editor 的哪个属性里取用户修改后的值，因为默认 editor 是在 `text` 属性里取，如果你自定义了 editor，系统还会去 `text` 里取，没办法，flex 还没有那么智能化，所以您必告诉系统要去 NumericStepper 的哪个属性里去取用户修改过的值。

如果大家试运行过 editor，大家会发现，用户的每次修改都会直接修改 dataprovider 中的值，这一点大家一定要注意，因为有些时候，这并不是我们想要的。

通过以上两种情况的说明，大家要知道，看似很简单的操作，实际上系统做了很多工作：

- 1) 用户没有点击时使用 item renderer 显示数据，
- 2) 用户点击后就 item editor 修改数据
- 3) 用户修改完毕，销毁 item editor 再用 item renderer 显示数据
- 4) 用户修改完毕后要找到新的数据，并更新到 dataprovider 中去

以上两种情况不管简单还是复杂，都有以上这几步操作的体现，只是大多数工作由系统完成。

- 3 说了这么多，这种情况才是本节讲的重点。一个单元格里有多个属性对应，这种情况下边两种办法都帮不上忙，这时我们要推翻原有的系统工作机制，自己完成上边说的四个工作。其中第一项工作，系统已经帮我们完成了，剩下的三项工作全部在下边方法中完成：

```
public function processData(event:ListEvent):void {
    event.preventDefault();
    list.editedItemRenderer.data =
    MultipleDataTypeEditor(list.itemEditorInstance).data;
    list.destroyItemEditor();
    list.dataProvider.notifyItemUpdate(list.editedItemRenderer);
}
```

`event.preventDefault()` 的目地是停止原有动作，这样我们就可以自己去做剩下的三项工作了。`list.editedItemRenderer`，每个 List 或 dataGrid 都有这个属性，在 `itemEditBegin`、`itemEditEnd` 这两个事件发生时，你可以通过这个属性的 `data` 得到当前 editor 中的值。但默认情况下只会得到一个 String，所以在例子中把修改过的两个属性以对象的形式传到 `data` 中。`list.dataProvider.notifyItemUpdate(list.editedItemRenderer);` 把修改过的数据保存在了 dataprovider 中。

list.destroyItemEditor();是用于去掉 item editor 的显示。

## 7.6.节.使用项渲染器把 SWF 对象作为一个菜单项显示

由 Rico Zuniga 捐献

### 7.6.1. 问题

用一个图片或 SWF 做菜单。

### 7.6.2. 解决办法

用 itemRenderer 对象加载 SWF 文件，并自定义菜单。

### 7.6.3. 讨论

自定义菜单的第一步是把你想要用到的字体和图形存在一些 SWF 文件里，并且把这些 SWF 文件放在一文件夹下。在我们的例子中，这个文件夹命名为“swf”，你可以用任工具生成这些 SWF 文件。

接下来，创建一个 renderer 组件。在本例中我们使用 Canvas，并在 Canvas 里增加一个 SWFLoader 组件。如果我们想用 Canvas 做自定义菜单，还有一个条件，就是要让 Canvas 实现 IMenuItemRenderer 接口，因为接口的需要，我们在 Canvas 里实现了 menu 的 set get 方法，但什么也没有做。SWFLoader 的 source 属性我们赋值为 data.swf\_wp（因为这个 Canvas 最终会用做 renderer，所以用 data.xxxx，原文是 data.swf，太容易让人迷惑了，我改成了 data.swf\_wp，在原文和原文的例子中都只提到实现接口中的 set menu 和 get menu 方法其实还有三个方法需要实现，下面是我改过的代码，如果对原文兴趣，请见本节结尾部分）。

Code View: (文件名为: FontItemRenderer.mxml)

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml"
    width="100" height="25"
    verticalScrollPolicy="off" horizontalScrollPolicy="off"
    xmlns:external="flash.external.*"
    implements="mx.controls.menuClasses.IMenuItemRenderer">
    <mx:Script>
        <![CDATA[
            import mx.controls.Menu;
            public function get menu():Menu {
                return null;
            }
        ]]>
    </mx:Script>
</mx:Canvas>
```

```

    }
    public function set menu(value:Menu) :void {
    }
    public function get measuredIconWidth():Number{return 0; }

    public function get measuredTypeIconWidth():Number{return 0; }

    public function get measuredBranchIconWidth():Number{return 0; }
]]>
</mx:Script>
<mx:SWFLoader source="{data.swf_wp}" width="100" height="25"
    horizontalCenter="0" verticalCenter="0"/>
<mx:Label x="0" y="0" text="{data.label}" width="100"
    height="25"/>
</mx:Canvas>

```

接下来你要在 application 里创建自定义的 Menu。要创建一个数组用作 Menus 的数据源 (dataProvider)，在这个数组中每个元素有的两个属性，label 和 swf\_wp 分别用于菜单项上的文字显示和 SWF 显示，其中 swf\_wp 是 SWF 文件的路径。关于 ClassFactory 我们在第二章就有讲解。

Code View:

```

<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute" creationComplete="init();"
    backgroundGradientColors="#c0c0c0, #ffffff">
<mx:Script>
<! [CDATA[
import mx.events.MenuEvent;
import mx.controls.Menu;
import compent.FontItemRenderer;

private var menu:Menu;

private function init():void {
var menuData:Array = [
    {label: "SubMenuitem A", swf_wp:'swf/a.swf', children: [
        {label: "SubMenuitem A-1", swf_wp:'swf/a1.swf'},
        {label: "SubMenuitem A-2", swf_wp:'swf/a2.swf'}
    ]},
    {label: "SubMenuitem B", swf_wp:'swf/b.swf', children: [
        {label: "SubMenuitem B-1", swf_wp:'swf/b1.swf'},
        {label: "SubMenuitem B-2", swf_wp:'swf/b2.swf'}
    ]}
];
menu.dataProvider = menuData;
}

```

```

        ] }

    ] ;

menu = Menu.createMenu(this, menuData);
menu.itemRenderer = new ClassFactory(compent.FontItemRenderer);
}

]]>
</mx:Script>
<mx:Button x="10" y="10" label="Show Menu" id="btnShowMenu"
    click="menu.show(btnShowMenu.x,
    btnShowMenu.y+btnShowMenu.height);"/>
</mx:Application>

```

原文中的代码:

```

<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="100"
height="25" verticalScrollPolicy="off"
horizontalScrollPolicy="off" xmlns:external="flash.external.*"
implements="mx.controls.menuClasses.IMenuItemRenderer">
<mx:Script>
<! [CDATA[
import mx.controls.Menu;

public function get menu():Menu {
return null;
}

public function set menu(value:Menu):void {
}
]]>
</mx:Script>
<mx:SWFLoader source="{data.swf}" width="100" height="25"
horizontalCenter="0" verticalCenter="0"/>
</mx:Canvas>

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute" creationComplete="init();"
backgroundGradientColors="#c0c0c0, #ffffff">
<mx:Script>
<! [CDATA[
import mx.events.MenuEvent;

```

```

import mx.controls.Menu;

private var menu:Menu;

private function init():void {
var menuData:Array = [
{swf:'swf/coolfonts.swf', children: [
{label: "SubMenuItem A-1", swf:'swf/meridiana.swf'},
{label: "SubMenuItem A-2", swf:'swf/virinda.swf'}
]},
{swf:'swf/scriptfonts.swf', children: [
{label: "SubMenuItem A-1", swf:'swf/monotypecorsiva.swf'},
{label: "SubMenuItem A-2", swf:'swf/comicsansms.swf'}
]}
];
}

menu = Menu.createMenu(this, menuData);
menu.itemRenderer = new ClassFactory(FontItemRenderer);
}

]]>
</mx:Script>
<mx:Button x="10" y="10" label="Show Menu" id="btnShowMenu"
click="menu.show
(btnShowMenu.x, btnShowMenu.y+btnShowMenu.height);"/>
</mx:Application>
```

```

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute" creationComplete="init();"
backgroundGradientColors="#c0c0c0, #ffffff">
<mx:Script>
<! [CDATA[
import mx.events.MenuEvent;
import mx.controls.Menu;

private var menu:Menu;

private function init():void {
var menuData:Array = [
{swf:'swf/coolfonts.swf', children: [
{label: "SubMenuItem A-1", swf:'swf/meridiana.swf'},
{label: "SubMenuItem A-2", swf:'swf/virinda.swf'}
]},
]
```

```

        {swf: 'swf/scriptfonts.swf', children: [
            {label: "SubMenuItem A-1", swf: 'swf/monotypecorsiva.swf'},
            {label: "SubMenuItem A-2", swf: 'swf/comicsansms.swf'}
        ]}
    ];

menu = Menu.createMenu(this, menuData);
menu.itemRenderer = new ClassFactory(FontItemRenderer);
}

]]>
</mx:Script>
<mx:Button x="10" y="10" label="Show Menu" id="btnShowMenu"
    click="menu.show
        (btnShowMenu.x, btnShowMenu.y+btnShowMenu.height);"/>
</mx:Application>

```

## 7.7. 节. 用一个复选框渲染器选择 DataGrid 列

原文由 Ben Clinkinbeard 捐献。

### 7.7.1. 问题

我们需要创一个拥有一列 CheckBox 的 DataGrid，并且这个 DataGrid 的表头也有一个 CheckBox，当用户点击表头的 CheckBox 时，这一列 CheckBox 自动被选中或取消选中，我们这里所说的表头就是 column 的 headerRenderer。

### 7.7.2. 解决办法

创建一个类做为 headerRenderer，并且在这个类里创建一个可以向它所在的 DataGrid 派发事件的方法，并且可以设置 DataGrid 里所有 itemRenderer 的属性。

### 7.7.3. 讨论

Flex 的 header renderers 的创建与生命周期要比 item renderers 复杂一些。在 DataGrid 的生命周期里，任何与 header 相关的动作都会引起 headerRenderer 的重建。因此你需要在外部保存 DataGrid 的状态。因此我们需要用 ClassFactory。

在原文中，作者讲解了关于 ClassFactory 的使用，这些我们已经在本章的第二节做过说明，所以这里就略过。

前边说了很多理论，下面我们说一下如何实现：

我们需要在 mxml 里定义一些变量：

```
// var to hold header renderer's state
public var selectAllFlag:Boolean;

[Bindable]
public var hr:ClassFactory;
```

selectAllFlag 变量，顾名思义，它用于表示是不是所有行都被选中了。hr 变量，是 DataGridColumn 的 headerRenderer，并且被标记为 [Bindable]。接下来要做的就是在 mxml 的 creationComplete 方法里为 hr 初始化：

```
hr = new ClassFactory(CenteredCheckBoxHeaderRenderer);
hr.properties = {stateHost: this, stateProperty: "selectAllFlag"};
```

第一行创建了一个 ClassFactory 的实例，并且把我们自定义的 renderer 类赋值给 factory 的 generator 属性。在第二行赋值了两组 key-value 给 hr 的 properties 属性，renderer 被创建后会用这两组 key-value 初始化。其中 stateHost 的 value 是 “this”，它代表当前的 mxml 文件，stateProperty 的 value 是我们在前边定义好的变量 “selectAllFlag” 的变量名。下边我们看一下，如何为 DataGridColumn 指定刚刚创建的 ClassFactory 实例。

```
<mx:DataGridColumn width="30"
    sortable="false"
    dataField="addToCart"
    headerRenderer="{hr}"

    itemRenderer="CenteredCheckBoxItemRenderer" />
```

请注意这里的 sortable="false"，sortable 的默认值是 true 当用户点击一个表的表头时，当前列的数据会自动排序，当用户点击 CheckBox 时不应该自动排序，所以这里设置 sortable="false"。

下面我们看一下 header renderer 的代码：

Code View:

```
package
{
    import flash.display.DisplayObject;
    import flash.events.MouseEvent;
    import flash.text.TextField;

    import mx.controls.CheckBox;
    import mx.controls.DataGrid;

    public class CenteredCheckBoxHeaderRenderer extends
        CenteredCheckBox
```

```

{
    public var stateHost:Object;
    public var stateProperty:String;

    override public function set data(value:Object):void
    {
        selected = stateHost[stateProperty];
    }

    // this function is defined by mx.controls.CheckBox
    // it is the default handler for its click event
    override protected function
        clickHandler(event:MouseEvent):void
    {
        super.clickHandler(event);
        // this is the important line as it updates the
        external variable
        // we've designated to hold our state
        stateHost[stateProperty] = selected;
    }
}
}

```

当 headerRenderer 被创建或重建时，setter 方法会被调用，我们没有处理传入的值，而是把外部的 stateProperty 的值赋给了 selected 因为 headerRenderer 继承自 CheckBox，这里的用意是：因为 headerRenderer 会被重建，而之前是否被选中了，我们已经保存在了外部变量 stateProperty 里，所以每次重建时，要把 selected 用外部变量初始化。而在 clickHandler 里，我们做了相反的操作，目地在于将用户操作后的 CheckBox 的状态保存起来。

通过上边浅显易懂的例子，向大家讲解了 ClassFactory 的使用方法，经实践验证，这是一个最好也是最有效的方式去创建一个可重用的复杂 renderer。

现在我们剩下的工作就是：当用户点击了表头的 CheckBox 时，要让 DataGrid 里的所有 CheckBox 跟着选中或取消选中。我们只需要对 mxml 文件增加 MouseEvent.CLICK 事件的监听，因为所有的事件都会冒泡到这里，headerRenderer 的点击事件也不例外。（我觉得原文在这里提及冒泡有些多余，可能有些读者对 flex 事件的冒泡机制还不太了解，在这里我做一下简单的介绍，mxml 里的组件就像一棵树一样，applicaton 就是这棵树的根（当然，如果 mxml 是一个 module 或 component 它的根就是 canvas 或其它容器），可能它下边会有一个 canvas，canvas 里可能还包含了 N 个 canvas，这些 canvas 就是这棵树的枝干，它们可能还包含了其它原素，如 button 等，如果某个 button 发生 click 事件，事件会从 button

所在的结点开始，逐层向自己的父结点 dispatch 这个事件，直到根结点为止。在本例中，不必在 headerRenderer 上监听，而是在根结点上监听事件，就是原于这个冒泡的事件机制。当然这个冒泡的机制是可以组织或中断的。）

当我们捕捉到 MouseEvent.CLICK 事件之后，要判断一下发出这个事件的组件是不是我们自定义的 headerRenderer，如果是，就说明 header 的 CheckBox 被点击了，这时我们要根据它的选中状态更新 DataGrid 的 dataProvider。因为 dataProvider 里有一个字段 addToCart 已经绑定在了 DataGrid itemRenderer 中的 CheckBox 的 selected 属性了。我们把 dataProvider 里所有 addToCart 属性全部更新为 header CheckBox 的选中状态（即 obj.addToCart = CenteredCheckBoxHeaderRenderer(event.target).selected），这样 itemRenderer 中的 CheckBox 就自动选中或取消选中了。接下来使用了 dataProvider 的 itemUpdated 方法，此方法用于通知 DataGrid，它的 dataProvider 中的某个 item(obj) 的某个字段(addToCart)已经更新了，请重新刷新它。（我觉得在这里引入 itemUpdated 似乎没有什么必要，和上边的冒泡一样，它们大大的加重了本节的难度。如果你过看 flex2 的一些代码，会使用 notifyItemUpdate 方法，现在被替换成 itemUpdated 方法。）

Code View:

```
// click events will still bubble
private function onCheckBoxHeaderClick(event:MouseEvent):void
{
    // make sure click came from header
    if(event.target is CenteredCheckBoxHeaderRenderer)
    {
        // loop over data
        for each(var obj:Object in dg.dataProvider)
        {
            // update value based on CheckBox state
            obj.addToCart =
                CenteredCheckBoxHeaderRenderer(event.target).selected;
            // notify collection item was changed
            ListCollectionView(dg.dataProvider).itemUpdated(obj,
                "addToCart");
        }
    }
}
```

## 7.8.节.为 DataGrid 创建一个独立的复选框项渲染器

原文由 Ben Clinkinbeard 捐献。

### 7.8.1. 问题

你要为 DataGrid 创建一个含有 CheckBox 的 itemRenderer，无论 DataGridColumn 的宽度如何，都要始终保持 CheckBox 在 renderer 的中央。

### 7.8.2. 解决办法

继承 CheckBox 的类，并且重写 updateDisplayList 方法，从而设置 CheckBox 的位置，在 clickHandler 中增加部分功能：根据 CheckBox 的选中状态设置 DataGrid 里的变量值。

### 7.8.3. 讨论

下边是一个可以把自己放置在 DataGridColumn 中央的 itemRenderer。这个类已经完成了所有 renderer 的功能，并且把 MouseEvent 继续向外派发，如果你对这个事件感兴趣，可以捕获它，并做某些操作。例如你可以在 DataGrid 所在的文件里注册一个监听器用于监听 MouseEvent，在事件发生之后可以使一个 Label 显示当前已被选中 CheckBox 的个数。

Code View:

```
package
{
    import flash.display.DisplayObject;
    import flash.events.MouseEvent;
    import flash.text.TextField;
    import mx.controls.CheckBox;
    import mx.controls.dataGridClasses.DataGridListData;

    public class CenteredCheckBoxItemRenderer extends CheckBox
    {
        // this function is defined by mx.controls.CheckBox
        // it is the default handler for its click event
        override protected function
            clickHandler(event:MouseEvent):void
        {
            super.clickHandler(event);
            data[DataGridListData(listData).dataField] = selected;
        }

        // center the checkbox icon
        override protected function updateDisplayList(w:Number,
            h:Number):void
        {
            super.updateDisplayList(w, h);

            var n:int = numChildren;
            for (var i:int = 0; i < n; i++)
            {
                var c:DisplayObject = getChildAt(i);
```

```
// CheckBox component is made up of icon skin and
label TextField

// we ignore the label field and center the icon
if (!(c is TextField))
{
    c.x = Math.round((w - c.width) / 2);
    c.y = Math.round((h - c.height) / 2);
}
}

}
```

在以上的代码中有两点值得注意：1、这个类没有实现 ClassFactory 接口，这意味着，我们可以把这个类的全名作为 DataGrid 的 itemRenderer 属性。又因为这个类是一个 drop-in renderer，因为他继承了 CheckBox，CheckBox 继承了 Button，Button 实现了 IDropInListItemRenderer 接口，所以你可以根据一个传入来的外部变量设置它的选中状态。2、你不必去重写 data 的 set get 方法，因为这些方法我们已经从 CheckBox 那里继承了。

```
data[DataGridListData(listData).dataField] = selected;
```

这里我们用了 `listData`, 这是 `IDropInListItemRenderer` 里定义的属性, 也就是说, 所有 `drop-in renderer` 都有这个属性, 系统创建 `renderer` 时会首先初始化 `listData` 这个属性, 接下来才会根据 `listData` 去初始化 `data`. `dataField` 是我们在 `column` 上配置的属性, 在本例中, `dataField` 应该是 `CheckBox` 在 `dataProvider` 里对应的属性名。这样, `data[DataGridListData(listData).dataField]` 就是那个属性名对应的值了, 在这里说值, 不太恰当, 因为它在等号左边, 这里应该说是 `dataProvider` 里那个变量的引用, 就是说, 你改了它, `dataProvider` 里的值也就跟着改了。

原文是想说明，如何设置 itemRenderer 中的组件处于单元格的中央，可是整节都在讲 listData。不过，设置位，却实没有什么可说的，都在 updateDisplayList 中。

## 7.9. 节. 为渲染器设置高效图像

### 7.9.1. 问题

你需要在通过 `data` 把图片传入 `itemRenderer` 里并显示图片。

### 7.9.2. 解决办法

创建一个新的 renderer 类，并且利用 commitProperties 方法和 owner 属性可以完成这些工作。我们可以在 renderer 拥有者所在的文件里定义一个方法，用于返回我们要图示的图片。

### 7.9.3. 讨论

List 和 ListBase 的设计者 Alex Harui，在自己的 blog 里写到：“你可以把 Image 标签放到 Canvas 里，但这不是一个最好的方式。”为什么？因为 Flex 框架为了简化我的开发，通过大量很复杂的操作来计算组件的大小，并且在刷新子组件和刷新自己时，都会再做这些复杂的操作。也就是说，我们可以多做一些事情，从而减轻框架的工作，这样就可以为 FlashPlayer 减少很多不必要的负担。

本例中的 renderer 实现了 IListItemRenderer 和 IDropInListItemRenderer 接口。因为这个组件继承了 UIComponent，我们需要重写 measure 方法这样我们就可以指定组件的明确大小。你需要通过实现 measure，测量出 image 的大小，并赋值给组件的 measuredWidth 和 measuredHeight。

还有另一种方式，renderer 通过访问它的父组件，从而得到 Image class，值得注意的是这种方式中，image 并没有传到 renderer 中。在我们的例子中，renderer 通过 data 访问父组件，从而得到内嵌 image 的引用。

```
if (listData) {
    // remove the old child if we have one
    if (img) {
        removeChild(img);
    }
    if (_imgClass == null) {
        var _imgClass:Class =
            UIComponent(owner).document[listData.label];
    }
    img = new _imgClass();
    addChild(img);
}
```

} 访问父组件可以通过 owner 的 document，然后结合 listData 确定要访问父组件的，哪个方法或属性。这样，我们即可以用传入的 image 也可以直接去父组件里取 image。下面是 renderer 的代码：

Code View:

```
package oreilly.cookbook
{
    import flash.display.DisplayObject;
    import mx.events.FlexEvent;
    import mx.controls.listClasses.BaseListData;
    import mx.controls.listClasses.IDropInListItemRenderer;
    import mx.controls.listClasses.IListItemRenderer;
    import mx.core.UIComponent;

    public class SevenSixRenderer extends UIComponent implements
        IDropInListItemRenderer, IListItemRenderer
```

```

{
    private var _data:Object;
    private var img:DisplayObject;
    private var _listData:BaseListData;
    private var _imgClass:Class;

    [Bindable("dataChange")]
    public function get data():Object {
        return _data;
    }

    public function set data(value:Object):void {
        _data = value;
        if (_data.imgClass != null) {
            _imgClass = _data.imgClass;
        }
        invalidateProperties(); // invalidate properties so that
        we're certain that
        //they'll be updated.
        dispatchEvent(new FlexEvent(FlexEvent.DATA_CHANGE));
    }

    [Bindable("dataChange")]
    public function get listData():BaseListData {
        return _listData;
    }

    public function set listData(value:BaseListData):void {
        _listData = value;
    }

    override protected function commitProperties():void {
        super.commitProperties();
        // sometimes the listdata of the renderer can be null, in
        which case we
        //certainly don't
        // want to throw runtime errors
        if (listData) {
            // remove the old child if we have one
            if (img) {
                removeChild(img);
            }
            if (_imgClass == null) {
                var _imgClass:Class =

```

```

        UIComponent(owner).document[ listData.label];
    }
    img = new _imgClass();
    addChild(img);
}
}

/* create the image instance now that we know what it is */
override protected function measure():void {
    super.measure();
    if (img) {
        measuredHeight = img.height;
        measuredWidth = img.width;
    }
}

/* make sure the image is positioned correctly */
override protected function updateDisplayList(w:Number,
    h:Number):void
{
    super.updateDisplayList(w, h);
    if (img) {
        img.x = (w - img.width) / 2;
    }
}

}
}
}

```

在上边的代码中我们重写了 commitProperties，如果传入的 Image 为 null，我们就利用 owner 去调用 DataGridColumn 的 labelFunction 方法，得到一个默认的 Image，如果传入的 Image 不为 null，则直接利用传入的 Image。

Code View:

```

<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="700"
    height="300">
    <mx:Script>
        <! [CDATA[
            import mx.collections.ArrayCollection;

            [Embed(source="../assets/foo.jpeg")]
            private var img:Class;

            [Embed(source="../assets/bar.jpeg")]
        ]>
    
```

```

public var img2:Class;

//just a generic collection to store some plain old info
[Bindable]
private var genericCollectionOne:ArrayCollection = new
ArrayCollection([{name:
"josh noble", age:30, appearance:"Somewhat wild"},
{name:"Abey George", age:32, appearance:"Pretty
tight", imgClass:img},
{name:"Todd Anderson", age:31,
appearance:"Intimidating"},{name:"Ryan Taylor", age:25, appearance:"Boyishly
Handsome", imgClass:img}, {name:"Steve Weiss", age:36, appearance:"George
Clooney-ish"}]);

// for our itemRenderer we use the call into this method if
the imgClass
//property is null
private function getImage(o:Object,
c:DataGridColumn):String
{
return "img2";
}

] ]>
</mx:Script>
<mx:DataGrid dataProvider="{genericCollectionOne}">
<mx:columns>
<mx:DataGridColumn dataField="age"/>
<mx:DataGridColumn dataField="name"/>
<mx:DataGridColumn dataField="appearance"/>
<mx:DataGridColumn
itemRenderer="oreilly.cookbook.SevenSixRenderer"
labelFunction="getImage" dataField="imgClass"/>
</mx:columns>
</mx:DataGrid>
</mx:HBox>

```

## 7.10.节. 为项渲染器和项编辑器应用运行时样式

### 7.10.1. 问题

你需要在运行时修改 itemRenderer 或 itemEditor 的一些属性。

### 7.10.2. 解决办法

ListBase 和 DataGrid(原文是 DataColumn, 可能是作者搞错了。)都有 makeRowsAndColumns 方法, 继承并重写它。

### 7.10.3. 讨论

我们可以在 makeRowsAndColumns 方法中, 通过一个循环完成对 itemRenderer 样式的设置。

Code View:

```
<mx>List xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:Script>
<! [CDATA[
    import mx.core.UIComponent;
    private var _rendererStyle:String;
    //here's the most effective way to ensure that we draw
    //the item renderers only when we need them
    override protected function
        makeRowsAndColumns(left:Number, top:Number,
                           right:Number, bottom:Number, firstCol:int,
                           firstRow:int, byCount:Boolean=false,
                           rowsNeeded:uint=0.0):Point
    {
        var pt:Point;
        pt = super.makeRowsAndColumns(left, top, right,
                                      bottom, firstCol, firstRow, byCount, rowsNeeded);
        if (_rendererStyle != null) {
            rendererStyle = _rendererStyle;
        }
        return pt;
    }

    public function set
        rendererStyle(styleName:String):void
    {
        _rendererStyle = styleName;
        if( collection != null ) {
            var i:int = 0;
            do {
                try{

```

```

        var comp:UIComponent =
            (indexToItemRenderer(i) as UIComponent);
        if(comp.styleName == _rendererStyle){
            comp.styleName = _rendererStyle;
            comp.invalidateDisplayList();
        } else { continue; }
    } catch (err:Error){}
    i++;
} while ( i < rowCount )
}
]
]
>
</mx:Script>
</mx>List>

```

首先指出上边的代码中有错误，set rendererStyle 方法中 if(comp.styleName == \_rendererStyle) { 中的 “==” 应该改为 “!=” 。

原文在这里强调了，makeRowsAndColumns 方法和 indexToItemRenderer 方法。使用 indexToItemRenderer 得到 itemRenderer，得到 itemRenderer 设置样式就不难了。那么 makeRowsAndColumns 是做什么用的呢？当 List 或 DataGrid 有滚动条的时候，我们拖动滚动条时，如果有新的 item 显示时，会重建或创建新的 itemRenderer，这些新显出来的 itemRenderer 是没有样式的，所以我们要在 makeRowsAndColumns 里给这些 itemRenderer 设置样式。

原文讲的东西很少，我们再深入的谈一下上边的例子：

在上边的例子中，使用了 rowCount，这种方式只适用于 List。indexToItemRenderer 方法是从现有已显示的 itemRenderer 里取一个。rowCount 是当前 list 的高度可以显示几行。对于 list 来说，能显示多少行，就代表有多少个 itemRenderer。但这个 rowCount 用到这里又是不对的，原因我们在后边讲。如果大家看过 API，API 中对 rowCount 的说明，会发现 API 重点强调了 DataGrid，对于 DataGrid 要注意，因为 DataGrid 的表头也算一行，也就是说如果 DataGrid 里只显示了三行数据，rowCount 为四。但我在实际的测试中，DataGrid 的 rowCount 并没有把 header 算上，如果大家感兴趣可以自己考证一下。

下面说一下，前面提到的：“rowCount 用到这里不对的”。可能是作者误解了 indexToItemRenderer 方法的用法，如果大家看过 indexToItemRenderer 这个方法的源码，大家就会发现上边这个例子是有 bug 的，下面我把源码粘出来：

```

public function
indexToItemRenderer(index:int):IListItemRenderer
{
    var firstItemIndex:int = verticalScrollPosition -
offscreenExtraRowsTop;

    if (index < firstItemIndex ||
        index >= firstItemIndex + listItems.length)

```

```

{
    return null;
}

return listItems[index - firstItemIndex][0];
}

```

首先大家要理解 listItems 里保存的是当前可见的 itemRenderer，并不是所有的 itemRenderer。而 index 是 dataProvider 中某一个元素的序列。其实关于 index 在 API 中已经描述的很清楚了，这就是为什么上边的代码中要 index - firstItemIndex (index 在可见 itemRenderer 中的序列)。我们再回过头看原文中的代码，那个 do{} while() 循环，它使用 i < rowCount 控制循环的结束，我们在前边讲过，rowCount 是可见行数，可见行数必然小于等于 dataProvider 的行数。所以在这个循环里可能不会把所有可见 itemRenderer 都循环到。如果大家还不明白，我可以举几个数字给大家：假如 dataProvider 的行数为 10，List 的高度只能显示 3 行。这时我们滚动条拖到第 5 条记录处，即显示 5、6、7 三条记录。这时运行原文中的 do{}while()，i 会是 0、1、2，分别把这三个数据传入 indexToItemRenderer，在些方法中 firstItemIndex 为 5 即滚动条所在位置，index 始终小于 firstItemIndex，所以 indexToItemRenderer 始终返回 null，在这种情况下，do{}while() 没有循环到任何一个 itemRenderer。

那么上边的这个 bug 如何解决呢？有两种办法，我们先说第一种：把 while 的条件设置的足够大，比如我们可以使用 dataProvider 的长度。另一种办法我们在后面讲。

原文还有另一个隐含问题，文章的开头一直把 List 和 DataGrid 放在一起谈，可是到了例子中却绝口不提 DataGrid。那么我在这里补充一下 DataGrid：如果大家测试过，会发现：如果你的 DataGrid 里只有一列 itemRenderer，使用上边的方式是没问题的（当然要改掉 rowCount）。（还要插一句，必须在 DataGrid 里显示的写一个 itemRenderer，才可以看到效果，但在 List 中不必）。

可是如果有两列或三列 itemRenderer 呢？经过测试发现，以上的方式只能处理第一列 itemRenderer。问题出在哪里了呢？我们看一下 indexToItemRenderer 的原码，其实 listItems 是一个二维数组，但在 indexToItemRenderer 里当做一维使用了。所以我们不能再使用 indexToItemRenderer 了（通过前边所讲，相信大家也觉得，即使在 List 的情况下使用它也不方便）。我们要写一个类的方法，把 listItems 里的每一个元素全都返回。同时我们要把 do{}while() 改动。下面是涉及到改动的全部代码：

```

public function
getShowAllItemRenderers(): ArrayCollection{
    var returnArry:ArrayCollection = new
    ArrayCollection()
    for each(var i:* in listItems){
        for each(var j:* in i){
            returnArry.addItem(j);
        }
    }
    return returnArry;
}

```

```

    public function set
rendererStyle(styleName:String):void {
    _rendererStyle = styleName;
    if( collection != null ) {
        var i:int = 0;
        var temp : ArrayCollection =
getShowAllItemRenderers();
        for each(var item:* in temp){
            if(item.styleName != _rendererStyle){
                item.styleName = _rendererStyle;
                item.invalidateDisplayList();
            } else {
                continue;
            }
        }
    }
}

```

上面这两个方法，放到 List 里，就是我们在前边说的第二种解决办法。

我们使用一个按钮控制 itemRenderer 样式的改变：

Code View:

```

<mx:VBox      xmlns:mx="http://www.adobe.com/2006/mxml"      width="400"
height="700"
xmlns:cookbook="oreilly.cookbook.*">
<mx:Style>
    .firstStyle{
        color:#999999;
    }
    .secondStyle{
        color:#3344ff;
    }
</mx:Style>
<mx:Button toggle="true" click="list.rendererStyle ==
'firstStyle' ? list.renderer
Style = 'secondStyle' : list.rendererStyle = 'firstStyle'"
label="TOGGLE" />
<cookbook:StylingRendererList id="list"
dataProvider="{oreilly.cookbook.DataHolder.simpleArray}"
rendererStyle="firstStyle" width="200"/>
</mx:VBox>

```

最后补充一下，原文中的做法只能把当前的 renderer 全部上色，如果用户拖滚动条，页面上就乱了。

## 7.11. 节. 为项编辑器应用状态和变换

### 7.11.1. 问题

你需要在用户开始编辑与结束编辑时给 itemEditor 加一些 effects。

### 7.11.2. 解决办法

在 itemEditor 中创建一个 state，当用户开始编辑它的时候，利用 Transition 运行一个 effect。

### 7.11.3. 讨论

在用户完成修改后运行一个 effect 是很好实现的。使用 DefaultListEffect。DefaultListEffect 定义了一个 effect 在用户提交自己的修改之后运行：

Code View:

```
<mx>List xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="300" dataChangeEffect="{baseEffect}" editable="true">
    <mx:DefaultListEffect id="baseEffect" color="#ffff00"
        fadeInDuration="300" fadeOutDuration="200"/>
</mx>List>
```

原文中有点小错误：List 并没有 dataChangeEffect 属性，而应该是 itemsChangeEffect。

上边的代码只实现了：当用户修改完成后有 effect，但是没有实现在开始修改时有 effect。为了实现在用户开始修改时有 effect 必须有一个 itemRenderer，在需要时可以创建 itemEditor。itemRenderer 只要完成 data 的设置和定义 editor 创建时的 Transition 就好了。当用户修改完成，销毁这个 editor 的时候，这个 effect 也会运行。

Code View:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml"
    currentState="base" width="100" height="20"
    creationComplete="currentState = 'init'" focusEnabled="true"
    backgroundColor="#ffff00">
    <mx:Script>
        <! [CDATA[
            [Bindable]
            private var _data:Object;
            override public function set data(value:Object):void {

```

```

        _data = value;
    }

override public function get data():Object {
    return _data;
}
//so that the text of the input field can be set when we first start
public function set text(value:String):void {
    input.text = value;
}

//this is needed so that the item editor will return the correct value
//when the list reads that value back
public function get text():String {
    return input.text;
}

] ]>
</mx:Script>
<mx:transitions>
    <!-- this transition will play when the component is ready to be displayed -->
    <mx:Transition fromState="*" toState="init">
        <mx:Fade alphaFrom="0" alphaTo="1" duration="500"
            target="{this}"/>
    </mx:Transition>
    <mx:Transition fromState="init" toState="*"/>
        <mx:Fade alphaFrom="1" alphaTo="0" duration="500"
            effectEnd="this.dispatchEvent(new
                Event('finishTransition', true))"
            target="{this}"/>
    </mx:Transition>
</mx:transitions>
<mx:states>
    <mx:State name="base"/>
    <mx:State name="init"/>
</mx:states>
<mx:TextInput id="input"
    creationComplete="input.text = String(_data),
    input.setFocus ()"/>
</mx:Canvas>
```

## 7.12.节. 创建一个带复选框的 Tree 控件

原文由 Jeff Ploughman 捐献。

### 7.12.1. 问题

你要创建一个树。它的结点一个有三个状态的 CheckBox。

### 7.12.2. 解决办法

我们从三个方面去解决这个问题：

- 1 创建 TreeItemRenderer。这个 renderer 要包含 CheckBox；
- 2 当 CheckBox 是第三种状态时，将有一个黑色的小图片覆盖在 CheckBox 上方，也就是说，这时我们看到 CheckBox 的，不是空心的，也不是里边含有“勾”，而是里边含有一个实心的黑方块；
- 3 dataProvider 的 item 中要有一个属性用于表示 CheckBox 的选中状态；

### 7.12.3. 讨论

通常我们用 tree 来表现一个文件系统，而且我们经常需要同时选中多个文件，并对它们一起做某个操作，如删除，复制等。所以我们需要 tree 的结点能有一种状态，表示它已经表选中了。

一个父结点的选中或取消选中操作应该同时触发它的所有子结点同时被选中或同时取消选中。如果这些子结点，有些是选中状态，有些是未选中状态，这时父结点需要有第三种状态才能更好的表现这种情况，所以我们需要有至少两个状态的 CheckBox。这第三种状态表示：某个父结点的子结点有些选中，有些未选中，而它自己是选中。

我们的解决办法是自定义一个 TreeItemRenderer, (就是下边的 CheckTreeRenderer) 它继承自 TreeItemRenderer，我们使用它实现三种状态的 CheckBox。

Code View:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    creationComplete="init();" >
<mx:Script>
<! [CDATA[
    import mx.collections.*;
    [Bindable]
    public var folderList:XMLList =
        <>
        <folder state="unchecked" label="Marketing Collateral"
            isBranch="true" >
```

```

<folder state="unchecked" isBranch="true" label="Media, PR,
and Communications" >
<folder state="unchecked" isBranch="false" label="Article
Reprint Disclaimers" />
<folder state="unchecked" isBranch="false" label="Articles
Reprints" />
<folder state="unchecked" isBranch="false"
label="Interviews and Transcripts" />
<folder state="unchecked" isBranch="false" label="Press
Kits" />
<folder state="unchecked" isBranch="false" label="Press
Releases" />
<folder state="unchecked" isBranch="false" label="Quick
Hits" />
<folder state="unchecked" isBranch="false" label="Rep
Talking Points" />
<folder state="unchecked" isBranch="false" label="Special
Updates" />
<folder state="unchecked" isBranch="false" label="White
Papers" />
</folder>
<folder state="unchecked" isBranch="true" label="Forms and
Applications" >
<folder state="unchecked" isBranch="false"
label="Applications" />
<folder state="unchecked" isBranch="false" label="Forms" />
</folder>
</folder>
</>;

[Bindable]
public var folderCollection:XMLListCollection;

private function init() : void
{
    folderCollection = new XMLListCollection(folderList);
    checkTree.dataProvider = folderCollection;
}

]]>
</mx:Script>
<mx:Tree
        id="checkTree"
        itemRenderer="oreilly.cookbook.CheckTreeRenderer"

```

```

        labelField="@label"
        width="100%" height="100%" >
    </mx:Tree>
</mx:Application>
```

下面是 CheckTreeRenderer 的代码:

```

package oreilly.cookbook
{
    import mx.controls.Image;
    import mx.controls.Tree;
    import mx.controls.treeClasses.*;
    import mx.collections.*;
    import mx.controls.CheckBox;
    import mx.controls.listClasses.*;
    import flash.events.Event;
    import flash.events.MouseEvent;
    import mx.events.FlexEvent;
    import flash.display.DisplayObject;
    import flash.events.MouseEvent;
    import flash.xml.*;
    import mx.core.IDataRenderer;

    public class CheckTreeRenderer extends TreeItemRenderer
    {
```

Create a CheckBox and an Image:

Code View:

```

    protected var myImage:Image;
    protected var myCheckBox:CheckBox;
    // set image properties
    private var imageWidth:Number      = 6;
    private var imageHeight:Number     = 6;
    private var inner:String       = "assets/inner.png";
    static private var STATE_SCHRODINGER:String = "schrodinger";
    static private var STATE_CHECKED:String = "checked";
    static private var STATE_UNCHECKED:String = "unchecked";

    public function CheckTreeRenderer ()
    {
        super();
        mouseEnabled = false;
    }
    private function toggleParents (item:Object,
```

```

tree:Tree, state:String) :void
{
    if (item == null)
    {
        return;
    }
    else
    {
        item.@state = state;
        toggleParents(tree.getParentItem(item), tree,
                      getState (tree, tree.getParentItem(item)));
    }
}

private function toggleChildren (item:Object, tree:Tree,
                                state:String) :void
{
    if (item == null) {return;}
    else {
        item.@state = state;
        var treeData:ITreeDataDescriptor =
            tree.dataDescriptor;
        if (treeData.hasChildren(item)) {
            var children:ICollectionView =
                treeData.getChildren (item);
            var cursor:IViewCursor = children.createCursor();
            while (!cursor.afterLast) {
                toggleChildren(cursor.current, tree, state);
                cursor.moveToNext();
            }
        }
    }
}

```

在每个结点被点击时，要同时设置这个结点的父结点状态，和所有子结点的状态。子结点只能被设置为 CHECKED 或 UNCHECKED 状态，但父结点可以设置第三种状态----SCHRODINGER，当它的子结点有 CHECKED 和 UNCHECKED 状态时，它就是 SCHRODINGER 状态。

父结点状态的设置要根据它所有子结点的状态而定：

Code View:

```
private function getState(tree:Tree, parent:Object):String {
```

```

var noChecks:int = 0;
var noCats:int = 0;
var noUnChecks:int = 0;
if (parent != null) {
    var treeData:ITreeDataDescriptor =
        tree.dataDescriptor;
    var cursor:IViewCursor =
        treeData.getChildren(parent).createCursor();
    while (!cursor.afterLast) {
        if (cursor.current.@state == STATE_CHECKED) {
            noChecks++;
        }
        else if (cursor.current.@state == STATE_UNCHECKED)
        {
            noUnChecks++;
        }
        else {
            noCats++;
        }
        cursor.moveToNext();
    }
    if ((noChecks > 0 && noUnChecks > 0) || (noCats > 0 &&
noChecks>0)) {
        return STATE_SCHRODINGER;
    } else if (noChecks > 0) {
        return STATE_CHECKED;
    } else {
        return STATE_UNCHECKED;
    }
}
private function imageToggleHandlder(event:MouseEvent):void
{
    myCheckBox.selected = !myCheckBox.selected;
    checkBoxToggleHandler(event);

}

```

我们要为 CheckBox 和 Image 设置鼠标点击事件的监听：

```

private function checkBoxToggleHandler(event:MouseEvent):void
{
    if (data)
    {

```

```

    Var myListData:TreeListData =
TreeListData(this.listData);
    var selectedNode:Object = myListData.item;
    var tree:Tree = Tree(myListData.owner);
    var toggle:Boolean = myCheckBox.selected;
    if (toggle) {
        toggleChildren(data, tree, STATE_CHECKED);
    } else {
        toggleChildren(data, tree, STATE_UNCHECKED);
    }
    var parent:Object = tree.getParentItem (data);
    toggleParents (parent, tree,
        getState (tree, parent));
}
}

```

我们重写 createChildren,用于创建 tree 的所有子结点:

Code View:

```

override protected function createChildren():void {
    super.createChildren();
    myCheckBox = new CheckBox();
    myCheckBox.setStyle( "verticalAlign", "middle" );
    myCheckBox.addEventListener( MouseEvent.CLICK,
        checkBoxToggleHandler );
    addChild(myCheckBox);
    myImage = new Image();
    myImage.source = inner;
    myImage.addEventListener( MouseEvent.CLICK,
        imageToggleHandler );
    myImage.setStyle( "verticalAlign", "middle" );
    addChild(myImage);
}

private function setCheckState (checkBox:CheckBox,
    value:Object, state:String):void
{
    if (state == STATE_CHECKED) {
        checkBox.selected = true;
    }
    else if (state == STATE_UNCHECKED) {
        checkBox.selected = false;
    }
}

```

```

    else if (state == STATE_SCHRODINGER) {
        checkBox.selected = false;
    }
}

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

        setCheckState (myCheckBox, value, value.@state);
        if(TreeListData(super.listData).item.@type ==
            'dimension') {
            setStyle("fontStyle", 'italic');
        } else {
            if (this.parent != null) {
                var _tree:Tree = Tree(this.parent.parent);
                _tree.setStyle("defaultLeafIcon", null);
            }
            setStyle("fontStyle", 'normal');
        }
    }
}

override protected function
updateDisplayList(unscaledWidth:Number,
    unscaledHeight:Number):void
{
    super.updateDisplayList(unscaledWidth, unscaledHeight);
    if(super.data)
    {
        if (super.icon != null) {
            myCheckBox.x = super.icon.x;
            myCheckBox.y = 2;
            super.icon.x = myCheckBox.x + myCheckBox.width +
                17;
            super.label.x = super.icon.x + super.icon.width +
                3;
        } else {
            myCheckBox.x = super.label.x;
            myCheckBox.y = 2;
            super.label.x = myCheckBox.x + myCheckBox.width +
                17;
        }
        if (data.@state == STATE_SCHRODINGER) {
            myImage.x = myCheckBox.x + 4;
        }
    }
}

```

```
        myImage.y = myCheckBox.y + 4;  
        myImage.width = imageWidth;  
        myImage.height = imageHeight;  
    } else {  
        myImage.x = 0;  
        myImage.y = 0;  
        myImage.width = 0;  
        myImage.height = 0;  
    }  
}  
}  
}
```

### 7.13. 节. 改变 List 中渲染器的大小

### 7.13.1. 问题

你需要创建一个被选中时是可以改变大小的 renderer。

### 7.13.2. 解决办法

创建一个 drop-in itemRenderer，实现 IDropInListItemRenderer 接口，并且通过 listData 监听 List 的 Scroll 和 Change 事件，当这两个事件中的任意一个事件发生时，比较 itemRenderer 中的 data 与 List 中 selectedItem 中的 data，如果相等，就设置 currentState 为 selected，否则设置为 base。

### 7.13.3. 讨论

有一点很重要：itemRenderer 是被 list 重用的。当一个 renderer 被选中，在你试图设置它的 state 的时候，要在每个 renderer 里判断自己是不是那个被选中的 renderer，这个 itemRenderer 的 data 必须和 List 或 DataGridColumn 的选中项的 data 相配置，而不是靠 selectedIndex 确定。

因为你可以通过 `listData` 访问 `List` 和 `DataGridColumn`，所以你也可以通过 `listData` 对 `listData` 或 `DataGridColumn` 增加事件的监听，并且在事件发生时，可以检查 `renderer` 的 `data` 和 `state`。

Code View

```
private function resizeEventHandler(event:Event):void
{
    if((listData.owner as List).selectedIndex ==
```

```

ArrayCollection(_listData.owner as
List).dataProvider).getItemIndex(this.data) &&
currentState != "selected") {
trace(" functions "+_listData.rowIndex+
"+(_listData.owner as List).selectedIndex);
currentState = "selected";
} else if(_listData.owner as List).selectedIndex !=
ArrayCollection(_listData.owner as
List).dataProvider).getItemIndex(this.data) &&
currentState == "selected") {
currentState = "base";
}
}

```

下面的代码中假定dataProvider的数据类型为 ArrayCollection，当然我们也可以把代码写的更公用一些，但在一般情况下， ArrayCollection就足够了。

Code View:

```

<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" height="30"
currentState="base"
implements="mx.controls.listClasses.IDropInListRenderer"
verticalScrollPolicy="off">
<mx:Script>
<![CDATA[
import mx.controls.List;
import mx.events.ListEvent;
import mx.controls.listClasses.BaseListData;
import mx.collections.ArrayCollection;

private function resizeFocusInHandler(event:Event):void
{
if(_listData.owner as List).selectedIndex ==
ArrayCollection(_listData.owner as
List).dataProvider).getItemIndex(this.data)
&& currentState != "selected") {
trace(" functions "+_listData.rowIndex+
"+(_listData.owner as List).selectedIndex);
currentState = "selected";
} else if(_listData.owner as List).selectedIndex !=
ArrayCollection(_listData.owner as
List).dataProvider).getItemIndex(this.data)
&& currentState == "selected") {
currentState = "base";
}
}

```

```

        }

override public function set data(value:Object):void {
    txt.text = value as String;
}

override public function get data():Object {
    return txt.text;
}

// Internal variable for the property value.
private var _listData:BaseListData;

// Make the listData property bindable.
[Bindable("dataChange")]
// Define the getter method.
public function get listData():BaseListData {
    return _listData;
}

// set the event listeners for the Change and Scroll
events
// that the List or Column will dispatch
public function set listData(value:BaseListData):void {
    _listData = value;
    _listData.owner.addEventListener(ListEvent.CHANGE,
        resizeFocusInHandler);
    _listData.owner.addEventListener(Event.SCROLL,
        resizeFocusInHandler);
}

] ]>
</mx:Script>
<mx:transitions>
    <mx:Transition fromState="*" toState="selected">
        <mx:Resize heightTo="60" target="{this}" />
    </mx:Transition>
    <mx:Transition fromState="selected" toState="*>">
        <mx:Resize heightTo="30" target="{this}" />
    </mx:Transition>
</mx:transitions>
<mx:states>
    <mx:State name="base"/>
    <mx:State name="selected">

```

```
<mx:AddChild>
  <mx:HBox>
    <mx:Label text="some text"/>
    <mx:Label text="{ 'some text = '+txt.text }"/>
  </mx:HBox>
</mx:AddChild>
</mx:State>
</mx:states>
<mx:Text id="txt"/>
</mx:VBox>
```

## (译者注)

看了原文中的例子，没有搞明白作者要监听 SCROLL 事件的真正意义，我猜作者的用意可能是：当用户拖动滚动条时，如果被选中的 item 移出了显示区，当继续滚动滚动条，item 又显示时，item 下方的那两个 label 依然显示(不必再运行 effect 了)。但实际是那两个 label 没有再次再显示。我觉得作者可能忽略了两个问题，1、当某个选中 item 不在显示区时，这个 item 的 renderer 可能已经被其它 item 使用了。当拖动滚动条，使这个 item 又显示时，系统会把移出显示区的 item 的 renderer 拿过来重用，又会调 renderer 的 set listData，又会重新对 SCROLL 事件加监听，但这时候加监听已经晚了，因为让这个 renderer 显示的拖动事件已经发生完了。所以这个刚刚显示出来的被选中的 item 的 renderer 不会被设置 state，所以下边的两个 Label 不会再次显示。可是，如果这时候我再拖动滚动条，使这个 item 移动显示区，这时这个 renderer 的监听方法生效了，它会满足第一个 if 条件，运行第一个 effect。这里大家可能会有些不明白，大家可能会认为：虽然 if 中第一个条件能满足，但第二个条件中 currentState 一定是等于 selected，因为它就是在被选中状态，这样就使整个 if 条件不满足。但事实不是这样的，我们回过头来再看，当这个 renderer 移出显示区时，这个 renderer 就被其它 item 重用了，但是重用时并没有改变 renderer 的 currentState，即 selected。这时用户又拖动滚动条，这个重用的 renderer 会监听到 SCROLL 事件，并调用 resizeFocusInHandler 这时自然就满足第二个 if 条件，这个 renderer 的 currentState 又被设为了 base。因为当这个被选中的 item 的 renderer 在第二次移出显示区时，又一次进入 selected state 并再一次运行 effect，只是因为 item 移出显示区了，我们在页面中看不到效果。

2、结合第一个问题，如果被选中的 item 的 renderer 没有被其它 item 重用，当用户再次拖动以至这个 item 再次被显示时，它可以及时的监听到 SCROLL 事件，从而调用 resizeFocusInHandler 方法，但程序不会满足任何 if 条件。label 已经不再了，没有 state 的改变，它们是不会再出来的。

总之我认为作者监听 SCROLL 事件没能达到他的预期效果，可能根本原因是那两个 label 莫名其妙的没了。

到这里本章已经结束了，在本章中指出了原文中的一些小错误，由于我人个的水平有限，有此错误可能是我的错误判断，望大家见谅。

## 第八章：图像，位图，视频和声音(**ken**)

图像，位图，视频和声音，是一个范围很广的，可以深入探讨的话题，需要用单独的一章来介绍。因此，这里集中回答最常见的问题。当 Flash 成为最主要的互联网视频实现方法和 Flex 框架更多的用在建立图片和 MP3 应用时，了解这些元素是如何工作的就变得越来越重要了。

Flash Player 提供多层次的工具来处理图像和声音。第一步是使用 Image 类和 VideoDisplay 类，MXML 的类可以简单的对图像和视频进行很多处理，使您能够快速的把这些资源整合到您的应用程序中。下一步则是 flash.media 包，它包括了 **Video**, **Sound**, **SoundTransform**, **Camera** 和 **Microphone** 类，其他相关的 Loader, NetConnection, 和 **NetStream** 类，在 flash.net 包中。这些类提供了许多完善的控件，包括了整合声音，视频和图像成为一个应用的功能，稍微花费一些时间可以更加完善他的功能。最后，通过 BitmapData 类和 **ByteArray** 类把 Flash Player 里的所有数据转成字节型数据。这些不仅仅可让您操作加载到 Flash Player 中的图像的位图数据，而且还可以创造新的点阵图和流数据。

在这一章的许多例子把图像和视频作为位图数据来操作。这比起操作声音要简单多了，因为 Flash Player 提供许多使用 BitmapData 类的简单方法，直接操作位图数据大大提高了您的程序开发效率。您也可以使用 NetStream 类来操作视频以及用户的麦克风和摄影头。**NetStream** 是一种用来处理麦克风和摄影头的流媒体信息的有效途径，并且可以用在服务器端程序上。

### 8.1 节. 载入并显示图像

#### 8.1.1 . 问题

我需要使用 Flex 组件来显示一个图片。

#### 8.1.2 . 解决办法

使用一个嵌入声明把图像文件编译到 SWF 文件中，或者在运行时载入它。

#### 8.1.3 . 讨论

Flex 支持在运行时或在编译时导入 GIF, JPEG, PNG 和 SWF 文件，支持 SVG 文件在编译时嵌入。你选择的方法依赖于图片的文件类型和应用程序的参数。任何嵌入式图像已经成为 SWF 文件的一部分，所以不须要任何时间加载。相应的代价是增加程序的大小，这会减缓应用程序的初始化过程。使用嵌入式图像方式，当图像文件的改变时，你还需要重新编译你的应用程序。

另外，你也可以使用这两种方式在运行时装载资源，一种是把图像的 source 属性设置为一

个 URL 地址，或者使用 URLRequest 对象取得返回值来装载操作一个 BitmapAsset 对象。您可以在 SWF 文件运行从本地文件系统装载资源，或者你可以存取远程资源，通常是通过网络的一个 HTTP 请求。这些图像独立于你的程序；你可以修改他们，而不需要重新编译，只要修改后的图像名称保持不变。

任何的 SWF 文件只能通过一种方式访问外部资源，本地或通过网络；不能使用两种方式访问。当你编译应用程序时，你可以使用 use-network 标志来决定 SWF 文件允许的访问方式。当 use-network 标志设置为 false 时，你能够访问本地文件系统的资源，但不能访问网络。默认值是 true，这允许你通过网络访问资源，但不能访问本地文件系统。

要嵌入一个图像文件，使用嵌入元数据属性：

```
[Embed(source="../assets/flag.png")]
private var flag:Class;
```

现在，这个类的对象可以被设为一个图像的 source 属性：

```
var asset:BitmapAsset = new flag() as BitmapAsset;
img3rd.source = asset;
```

另外，你也可以设 source 属性为本地或外部文件系统：

```
<mx:Image source="http://server.com/beach.jpg"/>
```

完整的例子如下：

Code View:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
<mx:Script>
<! [CDATA[
import mx.core.BitmapAsset;

[Embed(source="../assets/flag.png")]
private var flag:Class;

private function imgMod():void
{
    var asset:BitmapAsset = new flag() as BitmapAsset;
    img3rd.source = asset;
}

]]>
```

```
</mx:Script>
<mx:Image source="../assets/flag.png"/>
<mx:Image source="{flag}" />
<mx:Image id="img3rd" creationComplete="imgMod()"/>
</mx:VBox>
```

## 8.2 节. 创建视频显示

### 8.2.1. 问题

我需要在应用程序中显示一个 FLV 文件。

### 8.2.2. 解决办法

在你的应用程序中使用 VideoDisplay 类，并使用 Button 对象，来实现播放和暂停。

### 8.2.3. 讨论

VideoDisplay 类包装一个 flash.media.Video 对象，并且相当简单的向对象添加视频。将 VideoDisplay 的 source 属性设置为一个 FLV 文件的 URL 地址，把 autoplay 参数设置为 true，当 NetStream 被正确的实例化并且视频信息流被接受到时，视频就开始播放。

Code View:

```
<mx:VideoDisplay source="http://localhost:3001/Trailer.flv"
    id="vid" autoplay="true"/>
```

在下面的例子中，使用 VideoDisplay 类的方法为视频定义了播放，暂停和停止按钮。

Code View:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="300">
    <mx:VideoDisplay source="http://localhost:3001/Trailer.flv"
        id="vid" autoPlay= "false" autoRewind="true"/>
    <mx:HBox>
        <mx:Button label="Play" click="vid.play();"/>
        <mx:Button label="Pause" click="vid.pause();"/>
        <mx:Button label="Stop" click="vid.stop();"/>
    </mx:HBox>
</mx:VBox>
```

## 8.3 节. Mp3 文件的播放和暂停

### 8.3.1. 问题

我希望允许用户播放一系列的 MP3 文件。

### 8.3.2. 解决办法

使用 Sound 和 SoundChannel 类，当用户选择一个新的 MP3 类时，使用渐进式下载方式下在一个新的文件。

### 8.3.3. 讨论

Sound 类的 play 方法返回一个 SoundChannel 对象，它提供存取的方法和属性控制左右声道声音音量的平衡，还有暂停和恢复一个特定声音的方法。

例子，你可以使用这样的代码来装载和播放声音文件：

```
var snd:Sound = new Sound(new URLRequest("sound.mp3"));
var channel:SoundChannel = snd.play();
```

你不能用 ActionScript 代码暂停一个声音的播放，你只能使用 SoundChannel 的 stop 方法停止播放。不过，您可以从任何一个时间点开始播放声音。也可以纪录声音停止的时间点，并且从这个时间点开始播放后面的声音。

当声音播放时，SoundChannel.position 属性指示声音文件现在播放的位置。当声音播放停止是纪录时间点的值：

```
var pausePosition:int = channel.position;
channel.stop();
```

需要恢复声音时，传递前面纪录的时间点值，声音会从前面停止时的位置重新开始播放：

```
channel = snd.play(pausePosition);
```

下面的完整代码提供了一个组合框允许用户选择不同的 MP3 文件，暂停和停止播放使用了 SoundChannel 类：

Code View:

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
<mx:Script>
<! [CDATA[
```

```

import mx.collections.ArrayCollection;

public var sound:Sound;
public var chan:SoundChannel;
public var pausePos:int = 0;
private const server:String = "http://localhost:3001/"


private var dp:ArrayCollection =
    new ArrayCollection(["Plans.mp3", "Knife.mp3",
    "Marla.mp3", "On a Neck, On a Spit.mp3",
    "Colorado.mp3"])
private function loadSound():void {
    if(chan != null) {
        //make sure we stop the sound; otherwise, they'll overlap
        chan.stop();
    }
//re-create sound object, flushing the buffer, and readd the event listener
    sound = new Sound();
    sound.addEventListener(Event.SOUND_COMPLETE,
        soundComplete);
    var req:URLRequest = new URLRequest(server +
        cb.selectedItem as String);
    sound.load(req);
    pausePos = 0;
    chan = sound.play();
}
private function soundComplete(event:Event):void {
    cb.selectedIndex++;
    sound.load(new URLRequest(server +
        cb.selectedItem as String));
    chan = sound.play();
}

private function playPauseHandler():void
{
    if(pausePlayBtn.selected) {
        pausePos = chan.position;
        chan.stop();
    } else {
        chan = sound.play(pausePos);
    }
}

```

```

    ]]>
</mx:Script>
<mx:ComboBox creationComplete="cb.dataProvider=dp" id="cb"
    change="loadSound()"/>
<mx:Button label="start" id="pausePlayBtn" toggle="true"
    click="playPauseHandler()"/>
<mx:Button label="stop" click="chan.stop()"/>
</mx:HBox>

```

## 8.4 节. 为音频文件创建进度搜索条

### 8.4.1. 问题

我需要为用户搜索 MP3 文件的不同部分创建一个搜索控制条，和创建一个音量控制条来改变的 MP3 播放的音量。

### 8.4.2. 解决办法

通过为声音的 play 方法设置 time 参数来从时间点开始播放声音。这样创建一个新的 SoundTransform 对象将作为 SoundChannel 的 soundTransform。

### 8.4.3. 讨论

声音的 play 方法接受一个开始点的参数：

Code View:

```
public function play(startTime:Number = 0, loops:int = 0, sndTransform: SoundTransform
= null):SoundChannel
```

这创建了一个新的 SoundChannel 对象来播放你停止播放和监测声音的对象返回得声音。(要控制音量，平移，平衡，访问分配到 SoundChannel 的 SoundTransform 对象。)

要控制音量的声音，传递 SoundTransform 对象到 SoundChannel。我们创建一个新的有需要的值得 SoundTransform 对象，并且传递到正在播放的 SoundChannel。

```
var trans:SoundTransform = new SoundTransform(volumeSlider.value);
chan.soundTransform = trans;
```

SoundTransform 类接受下列参数：

```
SoundTransform(vol:Number = 1, panning:Number = 0)
```

该平移值范围从-1.0，表示全部向左平移（没有声音从右侧喇叭播放）至 1.0，表示全部向右平移。全部代码在下面：

Code View:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300" creationComplete="loadSound()">
<mx:Script>
<! [CDATA[

    private var sound:Sound;
    private var chan:SoundChannel;

    private function loadSound():void {
        sound = new Sound(
            new URLRequest("http://localhost:3001/Plans.mp3"));
        chan = sound.play();
    }

    private function scanPosition():void {
        chan.stop();
        //divide by 10 because the Slider values go from 0 -
        10 and we want a value between 0 - 1.0
        chan = sound.play(positionSlider.value/10 *
            sound.length);
    }

    private function scanVolume():void
    {
        var trans:SoundTransform =
            new SoundTransform(volumeSlider.value,
                (panSlider.value - 5)/10);
        chan.soundTransform = trans;
    }
}

]]>
</mx:Script>
<mx:Label text="Position"/>
<mx:HSlider change="scanPosition()" id="positionSlider"/>
<mx:Label text="Volume"/>
<mx:HSlider change="scanVolume()" id="volumeSlider"/>
<mx:Label text="Pan"/>
<mx:HSlider change="scanVolume()" id="panSlider"/>
</mx:VBox>
```

## 8.5 节. 融合两幅图像

### 8.5.1. 问题

我需要在运行时操作和结合多张图片，并使用过滤器来改变这些图片。

### 8.5.2. 解决办法

把图像作为 `BitmapData` 对象，使用 `BitmapData` 类 `combine` 方法把两个位图并为一个新的图像。

### 8.5.3. 讨论

`BitmapData` 和 `Bitmap` 类是在运行时操作图像和建立新效果的强大的工具。这两个类经常串联使用，但又有不同。`BitmapData` 类封装实际的数据到图像，`Bitmap` 类是一个能被加入到现实列表中的显示对象。`BitmapData` 对象的建立和写入如下：

```
var bitmapAsset:BitmapAsset = new BitmapAsset(img1.width, img1.height);
bitmapAsset.draw(img1);
```

首先，设置 `BitmapAsset` 高度和宽度，确保该对象是正确的大小，并从一个图像写入所有数据。这把图像的所有数据作为一个位图来读取，并允许你操作这些数据。在下面的例子中，用 `colorTransform` 方法操作 `BitmapData` 对象的颜色数据，通过 `merge` 方法和并两个位图。`colorTransform` 方法适用于把数据从 `ColorTransform` 对象传递到 `BitmapData` 对象。`ColorTransform` 对象修改显示对象的颜色或 `BitmapData` 相应的数据被传递到结构中：

Code View:

```
ColorTransform(redMultiplier:Number = 1.0, greenMultiplier:Number = 1.0,
               blueMultiplier:Number = 1.0, alphaMultiplier:Number = 1.0, redOffset:Number = 0,
               greenOffset:Number = 0, blueOffset:Number = 0, alphaOffset:Number = 0)
```

当 `ColorTransform` 对象被应用的显示对象时，像这样每个颜色通道重新计算了新的值：

```
New red value = (old red value * redMultiplier) + redOffset
New green value = (old green value * greenMultiplier) + greenOffset
New blue value = (old blue value * blueMultiplier) + blueOffset
New alpha value = (old alpha value * alphaMultiplier) + alphaOffset
```

`BitmapData` 类的 `merge` 方法见下列标记

Code View:

```
merge(sourceBitmapData:BitmapData, sourceRect:Rectangle, destPoint:Point,
       redMultiplier:uint, greenMultiplier:uint, blueMultiplier:uint, alphaMultiplier:uint):void
```

它的参数：

**sourceBitmapData:BitmapData**

导入位图使用。源图像可以是一个不同的 BitmapData 对象或是当前的 BitmapData 对象。

**sourceRect:Rectangle**

使用导入源图像中的一个确定的矩形区域。

**destPoint:Point**

目标图像（当前 BitmapData 实例）中的点，相对于源矩形的左上角。

**redMultiplier:uint**

一个十六进制 uint 值，增加红色通道的值。

**greenMultiplier:uint**

一个十六进制 uint 值，增加绿色通道的值。

**blueMultiplier:uint**

一个十六进制 uint 值，增加蓝色通道的值。

**alphaMultiplier:uint**

一个十六进制 uint 值，增加 alpha 通道的值。

下面完整的代码，操作 ColorTransform 的值

Code View:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="500"
height="550" creationComplete="imgMod()">
<mx:Script>
<! [CDATA [
import mx.core.BitmapAsset;
import mx.controls.Image;

[Embed(source="../assets/bigshakey.png")]
private var shakey:Class;

[Embed(source="../assets/mao.jpg")]
private var mao:Class;

//superimpose the two images together
]]>
```

```

//using the vslider data
private function imgMod():void
{
    var maoData:BitmapData =
        new BitmapData(firstImg.width,firstImg.height);
    var shakeyData:BitmapData =
        new BitmapData(secondImg.width,secondImg.height);
    maoData.draw(firstImg);
    shakeyData.draw(secondImg);
    maoData.colorTransform(new Rectangle(0, 0,
        maoData.width, maoData.height),
        new ColorTransform(redSlider.value/10,
            greenSlider.value/10,
            blueSlider.value/10,alphaSlider.value/10));
    var red:uint = (uint(redSlider.value.toString(16)) /
        10) * 160;
    var green:uint =
        (uint(greenSlider.value.toString(16)) / 10) * 160;
    var blue:uint = (uint(blueSlider.value.toString(16)) /
        10) * 160;
    var alpha:uint =
        (uint(alphaSlider.value.toString(16)) / 10) * 160;
    shakeyData.merge(maoData, new Rectangle(0, 0,
        shakeyData.width,shakeyData.height),
        new Point(0, 0), red, green, blue, alpha);
    mainImg.source = new BitmapAsset(shakeyData);
}

] ]>
</mx:Script>
<mx:HBox>
    <mx:Image id="firstImg" source="{mao}" height="200"
        width="200"/>
    <mx:Image id="secondImg" source="{shakey}" height="200"
        width="200"/>
</mx:HBox>
<mx:HBox>
    <mx:Text text="Red"/>
    <mx:VSlider height="100" id="redSlider" value="5.0"
        change="imgMod()"/>
    <mx:Text text="Blue"/>
    <mx:VSlider height="100" id="blueSlider" value="5.0"
        change="imgMod()"/>
    <mx:Text text="Green"/>

```

```

<mx:VSlider height="100" id="greenSlider" value="5.0"
    change="imgMod()"/>
<mx:Text text="Alpha"/>
<mx:VSlider height="100" id="alphaSlider" value="5.0"
    change="imgMod()"/>
</mx:HBox>
<mx:Image id="mainImg"/>
</mx:VBox>

```

## 8.6 节. 将 Convolution 滤镜应用于图像

### 8.6.1. 问题

我想要让用户改变颜色，对比，或锐利图像。

### 8.6.2. 解决办法

创建一个 ConvolutionFilter 的实例，绑定 ConvolutionFilter 的矩阵属性到用户可以改变的文本输入。然后添加滤镜到图像的滤镜数组，以适用于滤镜。

### 8.6.3. 讨论

ConvolutionFilter 是 flash.filter 包中一个最功能和复杂的滤镜。它可以用来作浮雕，检测边缘，锐化，模糊，等许多其他的效果。全部参数由 Matrix 对象控制的 3 维矩阵传递他的结构到滤镜中。ConvolutionFilter 在概念上在源图像上的每个像素一个一个通过计算，并确定最后的像素和像素周边的颜色。一个矩阵，作为一个数值形数组指定，表明每个像素的周围，在多大程度上影响颜色最后的结果。构造在这里显示：

Code View:

```
ConvolutionFilter(matrixX:Number = 0, matrixY:Number = 0, matrix:Array = null,
divisor:Number = 1.0, bias:Number = 0.0, preserveAlpha:Boolean = true,
clamp:Boolean = true, color:uint = 0, alpha:Number = 0.0)
```

他的参数如下：

**matrixX:Number (default = 0)**

矩阵的 x 维（矩阵中列的数量），默认值是 0。

**matrixY:Number (default = 0)**

矩阵的 y 维（矩阵中行的数量），默认值是 0。

**matrix:Array (default = null)**

用来做矩阵变幻的数组的值。数组中的项的数量必须等于 matrixX \* matrixY.

**divisor:Number (default = 1.0)**

在矩阵变换时使用的除数。默认值是 1。除数，整体颜色强度的结果的矩阵的值的和。 0 被忽略，并使用默认值。

**bias:Number (default = 0.0)**

添加偏差值到矩阵变换的结果。默认值是 0。

**preserveAlpha:Boolean (default = true)**

值是 false 表示 alpha 值不保留，卷积适用于所有通道，包括 alpha 通道。值是 true 表示卷积只适用于颜色通道。默认值是 true。

**clamp:Boolean (default = true)**

源文件的像素，值是 true 表明，输入的图像的每个边缘？？？

**color:uint (default = 0)**

十六进制的颜色来代替源图形的像素。

**alpha:Number (default = 0.0)**

alpha 的替代颜色。

一些常见的 ConvolutionFilter 效果分别如下：

**new ConvolutionFilter(3,3,new Array(-5,0,1,1,-2,3,-1,2,1),1)**

创建一个边缘检测的图形，只有最大对比的区域保留。

**new ConvolutionFilter(3,3,new Array(0,20,0,20,-80,20,0,20,0),10)**

创建一个黑白边框线。

**new ConvolutionFilter(5,5,new Array(**

**0,1,2,1,0,1,2,4,2,1,2,4,8,4,2,1,2,4,2, 1,0,1,2,1,0),50);**

创建一个模糊效果。

**new ConvolutionFilter(3,3,new Array(-2,-1,0,-1,1,1,0,1,2),0);**

创建一个浮雕效果。

完整的代码如下：

Code View:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="450"
height="550">
```

```

<mx:Script>
<! [CDATA[
import mx.core.BitmapAsset;

[Embed(source="..../assets/mao.jpg")]
private var mao:Class;

private function convolve():void
{
    var asset:BitmapAsset = new mao() as BitmapAsset;
    var convolution:ConvolutionFilter =
new ConvolutionFilter(matrixXSlider.value, matrixYSlider.value,
    [input1.text, input2.text, input3.text, input4.text,
    input5.text, input6.text],
    divisorSlider.value, biasSlider.value, true);
    var _filters:Array = [convolution];
    asset.filters = _filters;
    img.source = asset;
}

]]>
</mx:Script>
<mx:Button click="convolve()" label="convolve away"/>
<mx:HBox>
    <mx:Text text="Matrix X"/>
    <mx:VSlider height="100" id="matrixXSlider" value="5.0"
        change="convolve()"/>
    <mx:Text text="Matrix Y"/>
    <mx:VSlider height="100" id="matrixYSlider" value="5.0"
        change="convolve()"/>
    <mx:Text text="Divisor"/>
    <mx:VSlider height="100" id="divisorSlider" value="5.0"
        change="convolve()"/>
    <mx:Text text="Bias"/>
    <mx:VSlider height="100" id="biasSlider" value="5.0"
        change="convolve()"/>
<mx:VBox>
    <mx:TextInput id="input1" change="convolve()"
        width="40"/>
    <mx:TextInput id="input2" change="convolve()"
        width="40"/>
    <mx:TextInput id="input3" change="convolve()"
        width="40"/>
    <mx:TextInput id="input4" change="convolve()"

```

```

        width="40"/>
<mx:TextInput id="input5" change="convolve()"
    width="40"/>
<mx:TextInput id="input6" change="convolve()"
    width="40"/>
</mx:VBox>
</mx:HBox>
<mx:Image id="img"/>
</mx:VBox>

```

## 8.7 节. 通过摄像头将视频发送到 FMS 实例

### 8.71. 问题

我需要从用户的摄像机发送一个流到 Flash Media Server (FMS) 实例，供一个聊天室或其他直播媒体应用程序。

### 8.7.2. 解决办法

通过使用 `flash.media.Camera.getCamera` 方法捕获用户摄像机的视频流，附加摄像机到一个 `NetStream` 并将被发送到的 Flash Media Server 实例。使用 `NetStream` 类 `publish` 方法来发送一个特定名称的流到应用程序并处理它。

### 8.7.3. 讨论

`publish` 方法表明，一个 Flash Media Server 已经通过 `NetConnection` 类连接，`NetStream` 将发送信息到服务器。服务器对信息做什么处理依赖于应用程序。但是一些标志可以在 `publish` 方法中设置，表明什么流信息将被服务器和 Flash Player 处理。`publish` 方法有以下特性。

`publish(name:String = null, type:String = null):void`

它的参数如下：

`name:String (default = null)`

一个字符串用来确定流，如果你传递 `false`，`publish` 操作停止。客户端必须在它们呼叫 `NetStream.play` 时通过这条相同的名称订阅这个流。

`type:String (default = null)`

一个字符串用来表示怎么发布流。有效的值有 `record`, `append`, `live` (默认值)。如果你传递 `record`, Flash Player 发布并记录直播的数据，保存记录的数据到一个名称和传递 `name` 参

数相配值的新的 FLV 文件。该文件存储在服务器上一个包含服务器端程序的目录内的子目录中。如果文件已经存在，它将被覆盖。如果你传送 append，Flash Player 发布和记录直播数据，附加纪录的数据到一个名称和传送 name 参数相配值的新的 FLV 文件，存储在服务器上一个包含服务器端程序的目录内的子目录中。如果没有符合 name 参数的文件，将创建一个新的文件。如果你省略参数或传送 live，Flash Player 发布直播数据但不纪录它。如果有文件存在并且名称符合传送的 name 参数值，他将被删除。

当你使用 Flash Media Server 纪录流时，服务器创建一个 FLV 文件并存储在服务器上服务器端程序所在目录内的一个子目录中。每一个流都保存在一个名称符合 NetConnection.connect 传送的应用程序实例名称的目录中。服务器自动创建目录，你不需要去为每个应用程序实例创建。如下例，下面的代码显示，你如何连接一个特定的应用程序实例，存储在你应用程序目录中的一个名称为 lectureSeries 的目录中。一个名称为 lecture.flv 的文件存储在 /yourAppsFolder/lectureSeries/streams/Monday 子目录中：

```
var myNC:NetConnection = new NetConnection();
myNC.connect("rtmp://server.domain.com/lectureSeries/Monday");
var myNS:NetStream = new NetStream(myNC);
myNS.publish("lecture", "record");
```

如果你没有传送一个符合实例名称的值，被传送的 name 属性值被储存在一个名称为 /yourAppsFolder/appName/streams/\_definst\_（默认值）子目录中。

这个方法可以触发一个带有几个不同信息对象的 netStatus 事件。如下例，如果有人已经发布了一个特定名称的流，一个带有 NetStream.Publish.BadName 的 code 属性的 netStatus 事件被触发。更多的信息，参考 netStatus 事件。

下面的例子中，服务器连接已经建立，数据流从摄像机传送到服务器。

Code View:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="500" creationComplete="setUpCam()">
<mx:Script>
<! [CDATA [
    private var cam:Camera;
    private var nc:NetConnection;
    private var ns:NetStream;

    private function setUpCam():void
    {
        trace(Camera.names.join(","));
        //I'm doing this only because it's the only way the
        //flash player will pick up the camera on my MacBook
    }
]}>
```

```

        cam = flash.media.Camera.getCamera("2");
        vid.attachCamera(cam);
        nc = new NetConnection();
        nc.addEventListener(NetStatusEvent.NET_STATUS,
            netStatus);
        nc.connect("http://localhost:3002");
    }

    private function netStatus(event:NetStatusEvent):void
    {
        switch(event.info)
        {
            case "NetConnection.Connect.Success":
                ns = new NetStream(nc);
                ns.attachCamera(cam, 20);
                ns.attachAudio(Microphone.getMicrophone());
                ns.publish("appname", "live");
                break;
        }
    }

}]>
</mx:Script>
<mx:VideoDisplay id="vid" width="360" height="320"/>
</mx:Canvas>

```

## 8.8 节. 访问用户的麦克风并创建声音显示

### 8.8.1. 问题

我需要访问用户的麦克风并使用麦克风的音量绘制一个音量。

### 8.8.2. 解决办法

使用 Microphone.getMicrophone 方法访问 microphone。访问音量方法使用 Microphone 类的 mic.activityLevel 属性监测定期区间。

### 8.8.3. 讨论

麦克风类提供了访问用户的麦克风和计算机的方法，用户必须允许你的 Flash Player 程序使

用类访问。Microphone 类显示麦克风检测的音量，在开始时和一段时间内没有声音时触发事件。

Microphone 类的 3 个属性监视和控制着监测活动。只读的 activityLevel 属性表示从麦克风监测到的声音音量，范围从 0 到 100。silenceLevel 属性表示活动的麦克风需要的音量并且触发 ActivityEvent.ACTIVITY 事件。silenceLevel 属性同样使用 0 到 100 范围，默认值是 10。silenceTimeout 属性描述活动级别低于安静水平的毫秒数，直到 ActivityEvent.ACTIVITY 事件被触发，表明该麦克风现在处于无声。默认的 silenceTimeout 值是 2000。虽然 Microphone.silenceLevel 和 Microphone.silenceTimeout 都是只读，你仍然可以使用 Microphone.setSilenceLevel 方法改变他们的值。

接下来的例子创建一个 Microphone 对象，这将提示用户接受或拒绝的 Flash Player 访问麦克风。于是，以后的麦克风的活动通过检测 Activity 事件被检测，一个 enter frame 事件监听器被添加，并将在 Canvas 中绘制一个麦克风音量。

Code View:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300" creationComplete="createMic()">>

<mx:Script>
<! [CDATA[
import flash.media.Microphone;
import flash.events.ActivityEvent;
import flash.events.Event;
import flash.events.StatusEvent;

public var mic:Microphone;

public function createMic():void
{
    mic = Microphone.getMicrophone();
    mic.setLoopBack(true);
    mic.addEventListener(ActivityEvent.ACTIVITY, activity);
    mic.addEventListener(StatusEvent.STATUS, status);
    mic.addEventListener(Event.ACTIVATE, active);
}

private function active(event:Event):void
{
    trace(' active ');
}

private function status(event:StatusEvent):void

```

```

{
    trace("status");
}

private function activity(event:ActivityEvent):void
{
    trace("active ");
    addEventListener(Event.ENTER_FRAME, showMicLevel);
}

private function showMicLevel(event:Event):void
{
    trace(mic.gain+" "+mic.activityLevel+" "+mic.silenceLevel+
        " "+mic.rate);
    level.graphics.clear();
    level.graphics.beginFill(0xccccff, 1);
    level.graphics.drawRect(0, 0, (mic.activityLevel * 30),
        100);
    level.graphics.endFill();
}

}]>
</mx:Script>
<mx:Canvas width="300" height="50" id="level"/>
</mx:VBox>

```

## 8.9 节. 在 Flex 程序中平滑播放视频

### 8.9.1. 问题

我需要在应用程序中平滑播放视频。

### 8.9.2. 解决办法

创建一个自定义组件包含 the flash.media.Video 组件，然后设置视频的平滑属性设置为 True 。

### 8.9.3. 讨论

要平滑视频，像这样，使视频看起来减少像素化，你需要访问 flash.media.Video 对象。 视频平滑，就想图像平滑，需要比非平滑播放需要更强的处理能力，可能降低大画面或高清视频的播放速度。

Flex VideoDisplay 组件不允许你为它包含的 flash.media.Video 对象设置平滑属性。所以你必须创建一个单独的组件，增加了较低级别的 FlashVideo 组件，并设置平滑属性：

Code View:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300" creationComplete="setup()">
<mx:Script>
<! [CDATA[

    private var vid:Video;

    private var nc:NetConnection;
    private var ns:NetStream;
    private var metaDataObj:Object = {};

    private function setup():void {
        vid = new Video(this.width, this.height);
        vid.smoothing = true;
        this.rawChildren.addChild(vid);
        vid.y = 50;
        this.invalidateDisplayList();
    }

    private function startVid():void {
        nc = new NetConnection();
        nc.addEventListener(NetStatusEvent.NET_STATUS,
            netStatusHandler);
        nc.connect(null);
    }

    private function
    netStatusHandler(event:NetStatusEvent):void {
        ns = new NetStream(nc);
        metaDataObj.onMetaData = this.onMetaData;
        ns.client = metaDataObj;
        vid.attachNetStream(ns);
        ns.play("http://localhost:3001/Trailer.flv");
    }

    private function onMetaData(obj:Object):void {
        var i:int = 0;
        for each(var prop:Object in obj)
        {
```

```

        trace(obj[i] + " : " + prop);
        i++;
    }
    trace(obj.duration+" "+obj framerate+
          " "+obj.bitrate);
}
]

]]>
</mx:Script>
<mx:Button click="startVid()" label="load" x="50"/>
<mx:Button click="ns.resume()" label="resume" x="120"/>
<mx:Button click="ns.pause()" label="pause" x="190"/>
</mx:Canvas>

```

## 8.10 节. 检测像素级别的碰撞

### 8.10.1. 问题

我需要检查是否有带有 alpha 透明度的图像区域与其他图像发生碰撞。

### 8.10.2. 解决办法

绘制两个图像到一个 BitmapData 对象，并使用 BitmapData.hitTest 方法。

### 8.10.3. 讨论

BitmapData 对象拥有一个 hitTest 方法，和 DisplayObject 定义的 hitTest 方法工作方法相似但有一个明显的例外：而 DisplayObject 的 hitTest 方法返回 true，如果点和对象的范围相交，BitmapData 的 hitTest 方法返回 true，如果点的像素超过了某个透明度的门槛。这里是这个方法的特点：

Code View:

```

public function hitTest(firstPoint:Point, firstAlphaThreshold:uint, secondObject:
Object, secondBitmapDataPoint:Point = null, secondAlphaThreshold:uint = 1):Boolean

```

如果一个图像是不透明的，他被这个方法认为是一个完全不透明的矩形，两个图像必须被认为是透明的才能执行像素级碰撞测试。当你测试两个透明图像时，alpha threshold 参数控制 alpha 通道的值，从 0 到 255，被认为是不透明的。这个方法的参数如下：

#### firstPoint:Point

BitmapData 图像在任意坐标空间的左上角位置。相同的坐标位置被用来定义

`secondBitmapPoint` 参数。

`firstAlphaThreshold:uint`

在这个测试中被认为是不透明的最高的 alpha 通道值。

`secondObject:Object`

一个矩形, 点, 位图, 或 `BitmapData` 对象。

`secondBitmapDataPoint:Point (default = null)`

这个点定义了一个在第二 `BitmapData` 对象中的像素位置。 只有当 `secondObject` 的值是 `BitmapData` 对象时使用这个参数。

`secondAlphaThreshold:uint (default = 1)`

在第二个 `BitmapData` 对象中被认为是不透明的最大的 alpha 通道值。只有当 `secondObject` 的值是 `BitmapData` 对象并且两个 `BitmapData` 对象都是透明的时使用这个参数。

在下面的代码例子中, 每一个矩形图形的角上, 对一个带有 alpha 透明度的 PNG 文件进行碰撞测试:

Code View:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="1500"
height="900">
<mx:Script>
<! [CDATA[
import flash.display.BlendMode;

private var mainBmp:BitmapData;
private var dragBmp:BitmapData;
private var hasDrawn:Boolean = false;

private function loaded():void{
    if(!hasDrawn) {
        mainBmp = new BitmapData(
            mainImg.width, mainImg.height, true, 0x00000000);
        dragBmp = new BitmapData(
            dragImg.width, dragImg.height, true, 0x00000000);
        hasDrawn = true;
        this.addEventListener(Event.ENTER_FRAME, showHits);
    }
}

private function showHits(event:Event):void
{
    mainBmp.draw(mainImg);
```

```

dragBmp.draw(dragImg);
if(mainBmp.hitTest(new Point(0,0), 0xff,
dragImg.getBounds(this).topLeft)) {
    trace(" true ");
    return;
}
if(mainBmp.hitTest(new Point(0,0), 0xff,
dragImg.getBounds(this).bottomRight)) {
    trace(" true ");
    return;
}
if(mainBmp.hitTest(new Point(0,0), 0xff, new
Point(dragImg.getBounds(this).left,
dragImg.getBounds(this).bottom))) {
    trace(" true ");
    return;
}
if(mainBmp.hitTest(new Point(0,0), 0xff, new
Point(dragImg.getBounds(this).right,
dragImg.getBounds(this).top))) {
    trace(" true ");
    return;
}
trace(" false ");
}

]]>
</mx:Script>
<mx:Image id="mainImg" source="../assets/alphapng.png"
cacheAsBitmap="true"/>
<mx:Image cacheAsBitmap="true" id="dragImg"
mouseDown="dragImg.startDrag(false, this.getBounds(stage)),"
loaded()" rollOut="dragImg.stopDrag()"
mouseUp="dragImg.stopDrag()"
source="../assets/bigshakey.png"/>

</mx:Canvas>

```

当第一个图像的像素在得到的点没有取得 hitTest 方法设置的 alpha 值时这些代码返回 false。在图 8-1，两个亮的蓝色方块是带有 alpha 透明度的 PNG 文件。shake 是一个独立的图形，这时候，PNG 中没有足够高 alpha 的区域发生碰撞。可是，在图 8-2，shake 在一个区域中发生了碰撞并且这个方法返回了 true。

## 8.11 节. 读取和保持用户的网络摄像头图像

### 8.11.1. 问题

我需要从用户的网络摄像机读取一个图像并保存图像在服务器上

### 8.11.2. 解决办法

创建一个 Camera 对象，并将它附加到一个 Video 对象。然后创建一个按钮，从视频对象读取位图并使用服务器端脚本保存位图数据。

### 8.11.3. 讨论

网络摄像机捕获一个图像，从 Video 对象创建一个位图显示摄像机的图像。Flash Player 不提供任何从网路摄像机访问数据流的方法，所以，你需要在你能使用它之前渲染这些数据为位图。

图像作为 BitmapData 对象捕获以后，您可以通过把数据传送到一个 JPEGEncoder 类的实例来转换图像成为 JPEG 图像数据。接下来，添加数据到 URLRequest 对象，并使用 navigateToURL 方法传送，保存 JPEG 图像到数据库。例子：

Code View:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="500" creationComplete="setUpCam()">
<mx:Script>
<! [CDATA[
import flash.net.navigateToURL;
import flash.net.sendToURL;

import mx.graphics.codec.JPEGEncoder;

private var cam:Camera;

private function setUpCam():void {
    cam = flash.media.Camera.getCamera("2");
    vid.attachCamera(cam);
}

private function saveImage():void {
    var bitmapData:BitmapData = new BitmapData(vid.width,
    vid.height);
```

```

        bitmapData.draw(vid);
        var encode:JPEGEncoder = new JPEGEncoder(100);
        var ba:ByteArray = encode.encode(bitmapData);
        var urlRequest:URLRequest =
            new URLRequest("/jpg_reader.php");
        urlRequest.method = "POST";
        var urlVars:URLVariables = new URLVariables();
        urlVars.pic = ba;
        urlRequest.data = urlVars;
        flash.net.navigateToURL(urlRequest, "_blank");
    }

    ]]>
</mx:Script>
<mx:VideoDisplay id="vid" width="360" height="320"/>
<mx:Button label="Take Picture Now" click="saveImage()"/>
</mx:Canvas>

```

## 8.12 节. 在多幅图像中使用混合模式

### 8.12.1. 问题

我需要混合多个图像。

### 8.12.2. 解决办法

设置图像的 blendMode 属性。

### 8.12.3. 讨论

每一个显示对象定义 blendMode 属性，控制显示对象如何出现。控制 alpha 和显示列表中处于显示对象在下方的对象如何透过组件显示。任何使用过 Adobe Photoshop 或 After Effects 的人都熟悉混合模式：

#### BlendMode.ADD ("add")

在两个图像之间创建一个动画的光线融合效果。

#### BlendMode.ALPHA ("alpha")

前台到后台适用的透明度。

#### BlendMode.DARKEN ("darker")

超级加强效果

`BlendMode.DIFFERENCE ("difference")`

创建加强颜色鲜艳效果。

`BlendMode.ERASE ("erase")`

利用前景 alpha 去除部分的背景。

`BlendMode.HARDLIGHT ("hardlight")`

创建遮光效果。

`BlendMode.INVERT ("invert")`

翻转背景。

`BlendMode.LAYER ("layer")`

为特定的现实对象创建预先零时缓存区

`BlendMode.LIGHTEN ("lighten")`

超级加强类型

`BlendMode.MULTIPLY ("multiply")`

创建阴影和深度的效果。

`BlendMode.NORMAL ("normal")`

为混合图像指定像素的值覆盖基础图像。

`BlendMode.OVERLAY ("overlay")`

创建遮光效果。

`BlendMode.SCREEN ("screen")`

创建高光和镜头光斑效果。

`BlendMode.SUBTRACT ("subtract")`

在两个图像之间创建一个动画的黑暗融合效果。

下面的例子适用于两个 `Image` 对象各种混合模式：

## 8.13 节. 处理 FLV 数据的提示点

### 8.13.1. 问题

我需要在播放时使用编译在 FLV 文件中的提示点。

### 8.13.2. 解决办法

使用 NetStream 类的 onCuePoint 事件，创建一个处理方法，当遇到提示点时激活。

### 8.13.3. 讨论

提示点是插入到一个 FLV 文件中的纪录一个视频特定时间的值，包括一个简单的名称或带有一个 hash 表值的数据对象。提示点经常在文件开始编译时插入一个 FLV 文件，任何值在这里确定。Flex VideoDisplay 对象使用 mx.controls.videoClasses.CuePoint manager 类来从提示点检测和读取数据。为进一步了解这些，请考虑例子如何使用 flash.media.Video 对象。

当 NetConnection 对象已经连接时，NetStream 被实例化，你需要设置一个对象传递任何变化数据和提示点事件的处理方法。

```
var obj:Object = new Object();
obj.onCuePoint = onCuePoint;
obj.onMetaData = onMetaData;
ns.client = obj;
```

这些需要在 NetStream 的 play 方法被呼叫之前实现。注意下面的代码，onMetaData 和 onCuePoint 事件作为参数被对象接受。

Code View:

```
import flash.events.NetStatusEvent;
import flash.media.Video;
import flash.net.NetConnection;
import flash.net.NetStream;
import mx.core.UIComponent;

public class CuePointExample extends UIComponent
{
    private var ns:NetStream;
    private var nc:NetConnection;
    private var obj:Object = {};
    private var vid:Video;

    public function CuePointExample ()
    {
        super();
        vid = new Video();
        addChild(vid);
        nc = new NetConnection();
        nc.addEventListener(NetStatusEvent.NET_STATUS,
```

```

        netStatusEventHandler);
    nc.connect(null);
}

private function
netStatusEventHandler(event:NetStatusEvent):void {
    ns = new NetStream(nc);
    obj.onCuePoint = onCuePoint;
    obj.onMetaData = onMetaData;
    ns.client = obj;
    ns.play("http://localhost:3001/test2.flv");
    vid.attachNetStream(ns);
}

private function onCuePoint(obj:Object):void {
    trace(obj.name+" "+obj.time+" "+obj.length+" ");
    for each(var o:String in obj.parameters) {
        trace(obj[o]+" "+o);
    }
}

private function onMetaData(obj:Object):void{
}
}

```

使用 mx.controls.VideoDisplay 相当程度简化了提示点的工作。当使用 CuePointManager 处理 CuePointEvent 时，不同于前面的情况下，收到的事件只拥有三个属性： cuePointTime, cuePointName, and cuePointType. 如果你需要从提示点获得更多或不同的信息，你可能要写一个自定义类来返回提示点数据，并设置它为 VideoDisplay 对象的 cuePointManager 属性。完整的代码如下：

Code View:

```

<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
<mx:Script>
<! [CDATA[
    import mx.events.CuePointEvent;
    private function onCuePoint(event:CuePointEvent):void {
        trace(event.cuePointName+" "+event.cuePointTime+
              " "+event.cuePointType+" ");
    }
]]>
</mx:Script>
<mx:VideoDisplay id="vid" cuePoint="onCuePoint(event)" />

```

```
</mx:VBox>
```

## 8.14 节. 创建视频播放进度条

### 8.14.1. 问题

我需要创建一个控制条，使用户能够在视频播放时使用进度条。

### 8.14.2. 解决办法

创建一个可以拖动的 Sprite 对象，并且监听任何它发出的 DragEvent 事件。在 DragEvent 时间处理中，在 Video 对象的视频流，NetStream 设置向前或向后的搜索总数，

### 8.14.3. 讨论

您可以使用任何可拖动的显示对象在播放的视频中设定新的位置。在这个例子中，NetStream 的 seek 方法从视频开头起指定秒数的点开始播放。

```
ns.seek((playhead.x/timeline.width) * length);
```

用户打算搜索视频中制定的秒数时，用拖动 Sprite 对象在时间线区域的宽度来分配位置和在视频中的长度。NetStream 会照顾寻找适当的帧，从这一点开始重现播放视频流。

Code View:

```
import flash.display.Sprite;
import flash.events.MouseEvent;
import flash.events.NetStatusEvent;
import flash.media.Video;
import flash.net.NetConnection;
import flash.net.NetStream;

import mx.core.UIComponent;

public class Scrubber extends UIComponent
{

    private var playhead:Sprite;
    private var timeline:Sprite;
    private var ns:NetStream;
    private var nc:NetConnection;
    private var obj:Object = {};
    private var length:int;
    private var vid:Video;
```

```

public function Scrubber () {
    super();
    playhead = new Sprite();
    addChild(playhead);
    playhead.graphics.beginFill(0x0000ff, 1);
    playhead.graphics.drawCircle(0, 0, 5);
    playhead.graphics.endFill();
    playhead.addEventListener(MouseEvent.MOUSE_DOWN,
        startSeek);
    timeline = new Sprite();
    timeline.graphics.beginFill(0xcccccc, 1);
    timeline.graphics.drawRect(0, 0, 200, 10);
    timeline.graphics.endFill();
    addChild(timeline);
    timeline.addChild(playhead);
    playhead.y = 4;
    vid = new Video();
    addChild(vid);
    vid.y = 100;

    nc = new NetConnection();
    nc.addEventListener(NetStatusEvent.NET_STATUS,
        netStatus);
    nc.connect(null);
}

private function netStatus(event:NetStatusEvent):void {
    obj.onMetaData = onMetaData;
    ns = new NetStream(nc);
    ns.client = obj;
    vid.attachNetStream(ns);
    ns.play("http://localhost:3001/test.flv");
}

private function onMetaData(obj:Object):void {
    length = obj.duration;
    trace(length);
}

private function startSeek(mouseEvent:MouseEvent):void {
    playhead.startDrag(false, timeline.getBounds(this));
    addEventListener(MouseEvent.MOUSE_MOVE, seek);
    playhead.addEventListener(MouseEvent.ROLL_OUT, endSeek);
}

```

```

        playhead.addEventListener(MouseEvent.MOUSE_UP, endSeek);
    }

    private function seek(mouseEvent:MouseEvent):void {
        ns.seek((playhead.x/timeline.width) * length);
    }

    private function endSeek(mouseEvent:MouseEvent):void {
        removeEventListener(MouseEvent.MOUSE_MOVE, seek);
        playhead.stopDrag();
    }
}

```

## 8.15 节. 读取 mp3 文件的 ID3 数据

### 8.15.1. 问题

我需要从一个 MP3 文件中读取 ID3 数据。

### 8.15.2. 解决办法

使用 Event.ID3 方法，当 ID3 数据被分析时 Sound 类将被迅速处理

### 8.15.3. 讨论

当把一个装载的 MP3 文件的 ID3 数据被分析时，Sound 类迅速处理一个事件。这些数据被作为一个 ID3Info 对象保存，它定义的变量访问的所有属性被写入 MP3 开头的字节中：

```

private var sound:Sound;

public function _8_16()
{
    sound = new Sound();
    sound.addEventListener(Event.ID3, onID3InfoReceived);
    sound.load(new URLRequest("../assets/1.mp3"));
}

private function onID3InfoReceived(event:Event):void
{
    var id3:ID3Info = event.target.id3;
    for (var propName:String in id3)
    {
}

```

```

        trace(propName + " = " + id3[propName]);
    }
}

```

一首我听的歌中的信息，我写的这方似乎是这样的：

```

TCON = Alternative & Punk
TIT2 = The Pink Batman
TRCK = 2/9
TPE1 = Dan Deacon
TALB = Spiderman Of The Rings
TCOM = Dan Deacon

```

MP3 文件的 ID3 信息是一个简单的按一定次序分组的字节，读取和返回字符串或整数。MP3 是 Flash Player 支持的唯一一种格式。RichApps ([www.richapps.de](http://www.richapps.de)) 的开发者 Benjamin Dobler，不管怎么样，已经为 WAV 格式作了一些优秀的工作。得到 WAV 文件在 Flash Player 中播放稍微困难一些。如果你有兴趣，可以到 Benjamin 的网站去看看。如果你需要从 WAV 文件中分析数据，看起来像这样：

Code View:

```

public var bytes:ByteArray;
public var chunkId:String;
public var chunkSize:int;
public var chunkFormat:String;
public var subchunk1Id:String;
public var subchunk1Size;
public var audioFormat;
public var channels;
public var sampleRate;
public var bytesPersecond;
public var blockAlign;
public var bitsPerSample;
public var dataChunkSignature:String;
public var dataChunkLength;

public function read(bytes:ByteArray):void{
    this.bytes = bytes;
    // Read Header
    bytes.endian = "littleEndian";
    chunkId = bytes.readMultiByte(4, "utf"); //RIFF
    chunkSize = bytes.readUnsignedInt();
    chunkFormat = bytes.readMultiByte(4, "utf"); //WAVE
    subchunk1Id = bytes.readMultiByte(4, "iso-8859-1"); // 12
}

```

```

Header Signature
    subchunk1Size = bytes.readInt() ; // 16 4 <fmt length>
    audioFormat = bytes.readShort() ; // 20 2 <format tag> sample
    channels = bytes.readShort() ; // 22      2 <channels> 1 = mono,
    2 = stereo
    sampleRate = bytes.readUnsignedInt() ; // 24      4 <sample rate>
    bytesPerSecond = bytes.readUnsignedInt() ; // 28      4
<bytes/second>   Sample-Rate * Block-Align
    blockAlign = bytes.readShort() ; // 32 2 <block align> channel
    * bits/sample / 8
    bitsPerSample = bytes.readUnsignedShort() ; // 34      2
<bits/sample> 8, 16 or 24
    dataChunkSignature = bytes.readMultiByte(4, "iso-8859-1") ;
//RIFF
    dataChunkLength = bytes.readInt() ;
}

```

如果你需要从 AU 文件读取头信息，那看起来像这样：

```

public var bytes:ByteArray;
public var magicId;
public var header;
public var datasize;
public var channels;
public var comment;
public var sampleRate;
public var encodingInfo;

public function read(bytes:ByteArray) :void{
    this.bytes = bytes;
    // Read Header
    bytes.endian = "bigEndian";
    magicId = bytes.readUnsignedInt();
    header = bytes.readInt();
    datasize = bytes.readUnsignedInt();
    encodingInfo = bytes.readUnsignedInt();
    sampleRate = bytes.readUnsignedInt();
    channels = bytes.readInt();
    comment = bytes.readMultiByte(uint(header)-24, "utf");
}

```

MP3 可能是读取数据最简单的格式，但是他们无疑不是我们能读取得唯一格式。

## 8.16 节. 在载入图像时显示自定义引导

### 8.16.1. 问题

我需要在一个图像装载时显示定制的动画。

### 8.16.2. 解决办法

创建一个定制的图形，并且从 Image 对象装载图像是监听 ProgressEvent.PROGRESS 事件。

### 8.16.3. 讨论

当使用 Image 组件时有两种方法来显示图像： 你能在 MXML 中设置 Image 类的 source 属性，或者通过一个 URL 地址来装载并使用 img.load 方法：

```
img.load("http://thefactoryfactory.com/beach.jpg");
```

在你装载图像前，然而，你需要附加一个事件监听器来确保每一个 ProgressEvent 都被控制。

```
img.addEventListener(ProgressEvent.PROGRESS, progress);
```

在 progress 方法中，这是处理 ProgressEvent.PROGRESS 事件，一个 UIComponent 被使用 Image 的 bytesLoaded 属性重绘。

Code View:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml"
    creationComplete="loadImage()"
    >
<mx:Script>
<! [CDATA [
    private var m:Matrix;

    private function loadImage():void {
        var m:Matrix = new Matrix();
        m.createGradientBox(450, 40);
        img.addEventListener(ProgressEvent.PROGRESS,
            progress);
        img.load("http://thefactoryfactory.com/beach.jpg");
    }

    private function progress(event:Event):void{

```

```

        grid.graphics.clear();
        grid.graphics.beginGradientFill("linear", [0x0000ff,
            0xffffffff], [1, 1], [0x00, 0xff], m);
        grid.graphics.drawRect(0, 0, (img.bytesLoaded /
            img.bytesTotal) * 300, 40);
        grid.graphics.endFill();
    }

] ]>
</mx:Script>
<mx:Canvas id="grid" height="40" width="300"/>
<mx:Image id="img" y="40"/>
</mx:Canvas>

```

## 8.17 节. 启动图像上传

### 8.17.1. 问题

我需要使用户能通过 Flex 上传图像，保存在服务器中。

### 8.17.2. 解决办法

创建一个 `FileReference` 对象，并附加一个适当的滤镜，使用户只能上传正确类型的文件。然后监听 `FileReference` 对象的完成事件，并上传图像文件到一个服务器端脚本。

### 8.17.3. 讨论

在 flex 以及 Flash 中上传在图片，依赖于使用 `FileReference` 类。`FileReference` 对象，完成时，使用浏览器标准的上传窗口和图形创建一个窗口，并且当用户选择了一个上传的文件是通过 Flash Player 上传图片。为 `FileReference` 对象添加一个事件监听器表示用户已经选择了一个文件。

```
fileRef.addEventListener(Event.SELECT, selectHandler);
```

这样添加了一个方法来上传用户选择的文件：

Code View:

```
private function selectHandler(event:Event):void {
    var request:URLRequest =
        new URLRequest("http://thefactoryfactory.com/upload2.php");
    fileRef.upload(request, "Filedata", true);
```

}

文件上传以后，发送它道一个 PHP 脚本来保存上传的图像：

Code View:

```
package oreilly.cookbook
{
    import mx.core.UIComponent;
    import flash.net.FileFilter;
    import flash.net.FileReference;
    import flash.net.URLRequest;
    import flash.events.Event;

    public class _8_17 extends UIComponent
    {

        private var fileRef:FileReference;

        public function _8_17()
        {
            super();
            startUpload();
        }

        private function startUpload():void {
//set all the file types we're going to allow the user to upload
            var imageTypes:FileFilter = new FileFilter("Images (*.jpg, *.jpeg, *.gif, *.png)", "*.jpg; *.jpeg; *.gif; *.png");
            var allTypes:Array = new Array(imageTypes);
            fileRef = new FileReference();
            fileRef.addEventListener(Event.SELECT, selectHandler);
            fileRef.addEventListener(Event.COMPLETE,
                completeHandler);
//tell the FileRefence object to accept only those image
//types
            fileRef.browse(allTypes);
        }

        private function selectHandler(event:Event):void {
            var request:URLRequest = new URLRequest("http://thefactoryfactory.com/upload2.php");
            fileRef.upload(request, "Filedata", true);
        }

        private function completeHandler(event:Event):void {
            trace("uploaded");
        }
    }
}
```

```
}
```

因为文件已经上传，你需要处理服务器上的数据，移动文件到（在这种情况下）images 文件夹下

```
$file_temp = $_FILES['file']['tmp_name'];
$file_name = $_FILES['file']['name'];
$file_path = $_SERVER['DOCUMENT_ROOT']."/images";
//checks for duplicate files
if(!file_exists($file_path."/". $file_name)) {
    //complete upload
    $filestatus = move_uploaded_file($file_temp,$file_path."/". $file_name);
    if(!$filestatus) {
        //error in uploading file
    }
}
```

## 8.18 节. 比较两幅位图

### 8.18.1. 问题

我需要比较两个位图图像，并显示他们之间的差别。

### 8.18.2. 解决办法

从两个图像读取位图数据，并使用 compare 方法比较两个图像。把两个图像的差别设置为第 3 个图像的源。

### 8.18.3. 讨论

BitmapData 类的 compare 方法返回一个 BitmapData 对象，包括两个列出的图像中所有不能匹配的像素。如果两个 BitmapData 对象有相同的尺寸（宽和高），这个方法返回一个新的 BitmapData 对象，包括两个源对象中不同的每一个像素：如果两个像素相同，不同的像素值是 0x00000000。如果两个像素有不同的 RGB 值（忽略 alpha 值）不同的像素值是 0xFFRRGGBB，这里的 RR/GG/BB 是红色，绿色，蓝色通道之间的个体的差别值。在这种情况下，alpha 通道的差异被忽略。如果只是 alpha 通道的值有差别，像素的值是 0xZZFFFFFF，这里 ZZ 是 alpha 值得差别值。

Code View:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="800">
```

```
<mx:Script>
<! [CDATA[
import mx.core.BitmapAsset;

private function compare():void {
    var bmpd1:BitmapData = new BitmapData(img1.width,
        img1.height);
    var bmpd2:BitmapData = new BitmapData(img2.width,
        img2.height);
    bmpd1.draw(img1)
    bmpd2.draw(img2);
    var diff:BitmapData = bmpd2.compare(bmpd1) as
        BitmapData;
    var bitmapAsset:BitmapAsset = new BitmapAsset(diff);
    img3.source = bitmapAsset;
}

]]>
</mx:Script>
<mx:Image id="img1" source="../assets/mao.jpg" height="200"
width="200"/>
<mx:Image id="img2" source="../assets/bigshakey.png"
height="200" width="200"/>
<mx:Button click="compare()" label="compare"/>
<mx:Image id="img3"/>
</mx:VBox>
```

## 第九章. 皮肤与样式 (屋檐下)

Flex 框架强大的布局管理功能以及默认的 Halo AeonThe 主题使你可以即使不用 box 组件都能创造出十分友好的界面。你能利用容器和控件来创造一个应用程序用户界面的轻松程度，与你能否轻松地用皮肤和样式来美化这些组件是相关的。

本章的题目可能稍微有些误导读者： 皮肤和样式在 Flex 中并不是独立的两个概念；其实它们协力合作为你的程序带来视觉个性。实际上，皮肤是样式的一组属性，它能使用其他已经声明的样式值来达到同种效果。 样式和皮肤组件在使用方法是不同的，本章目的在于帮助你充分使用好它们。

样式是一种性质设置，例如色彩、大小或者字体信息，它可以修改组件的外观，可以在编译和运行时有规则地客户化。 样式性质可通过多种方式定义：在组件声明的代码段设置，通过调用 `setStyle` 方法应用样式， 或者利用级联样式表(CSS)。你可以用 CSS 在 MXML 文件或其他外部文件中局部地定义样式。

皮肤作为样式的特性，用于修改组件的外观元素。这些元素可以是图形化的，例如一个图像文件或者 SWF，再或者利用 drawing API 编程实现的类。为组件应用皮肤可以修改甚至替换组件的外观元素。这样当应用皮肤时一些样式设置可能会被忽略。

为了讲述两个过程怎样合作完成组件的客户化，我们先简单地看下按钮的视觉组成。你可以为按钮的很多性质赋值，例如与字体处理、补白有关的那些。用来定义一个按钮实例各自状态下皮肤的属性，也包含在其中。

要想大体地理解前面所讲的意思，考虑下可用于按钮的部分样式：

```
cornerRadius="4"  
fillAlphas="[0.6, 0.4]"  
fillColors="[0xE6EEEE, 0xFFFFFFFF]"  
upSkin="mx.skins.halo.ButtonSkin"
```

上述代码中这些可用于按钮的样式的属性值是包含在框架中的 `defaults.css` 文件所定义的默认值。任何按钮实例的 `upSkin` 属性默认值是 `Halo skins` 包中的 `mx.skins.halo.ButtonSkin` 类。

通过 `drawingAPI`，这个规则的皮肤利用其它已声明的属性值(`cornerRadius`, `fillAlphas`, 以及 `fillColors`)来表现按钮实例的弹起状态。尽管 `upSkin` 属性的默认值是一个编程实现的类，这些值可以替换成图形文件，这样可能会有效地忽略利用皮肤可能用到的属性值。

本章旨在讲述对你的组件使用皮肤和样式的过程，以及它们如何有效地用来改善你的程序的外观感受。

注意

在线的 Flex Style Explorer 允许在运行时修改各种组件的样式并查看作为结果的 CSS 定义。如下网址访问 Flex Style Explorer [http://www.adobe.com/go/flex\\_styles\\_explorer\\_app](http://www.adobe.com/go/flex_styles_explorer_app).

## 9.1 节. 用 CSS 定义组件样式

### 9.1.1. 问题

你希望利用 CSS 对组件应用皮肤

### 9.1.2. 解决办法

利用类选择器或类型选择器声明样式属性。

### 9.1.3. 讨论

你可以用 CSS 个性化你的用户界面。如果熟悉 HTML 文档中的样式元素，你会发现 Flex 里的 CSS 语法大致相同。你可以用类选择器为各种组件指派样式，同样你可以为一个组件定义类型选择器，它将适用于显示列表上该组件的所有实例。

类选择器经过声明后，在程序范围内可对不同组件应用任意次。类选择器的语法是一个句号（或点），后面跟着与样式有关的你所想的任何名字，驼峰样式同时以小写字母开头---举个例子，.myCustomStyle。接下来的例子用 <mx:Style> 标签创造了局部样式，并在代码内利用 styleName 属性将样式指派给一个 Label 组件：

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">
    <mx:Style>
        .labelStyle {
            font-family: 'Arial';
            font-size: 20px;
            font-weight: 'bold';
            color: #FFFFFF;
        }
    </mx:Style>
    <mx:Label text="Hello Flex!" styleName="labelStyle" />
</mx:Application>
```

本例中 .labelStyle 选择器定义了用于标签的有关字体处理的样式。本样式通过逐行将其赋值给 Label 实例的 styleName 属性起到作用。你会意识到在应用样式时，声明部分前面的句号/点已经被去掉了。

类型选择器可用来将其声明的样式应用于某一组件的所有实例。要想声明类型选择器，可使用想要处理的组件的相应类名，如下：

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">

    <mx:Style>
        Label {
            font-family: 'Arial';
            font-size: 20px;
            font-weight: 'bold';
            color: #FFFFFF;
        }
    </mx:Style>
    <mx:Label text="Hello Flex!" />
</mx:Application>
```

尽管可以使用类型和类选择器声明样式，定义了的属性值可能在编译时被组件的声明逐个重新定义。下例中，你看到应用于 Label 组件两个实例的样式，其中第二个重写了 fontSize 样式属性：

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">
    <mx:Style>
        Label {
            font-family: 'Arial';
            font-size: 20px;
            font-weight: 'bold';
            color: #FFFFFF;
        }
    </mx:Style>
    <mx:Label text="Hello Flex!" />
    <mx:Label text="Hello Flex!" fontSize="12" />
</mx:Application>
```

编译时逐项指定的样式属性值 不会受到用 `<mx:Style>` 标签局部定义的 CSS 的限制，正如上述例子。逐项定义的样式属性值会在运行和编译时重写掉外部加载的样式文件定义的样式。

你会注意到本例中嵌入的 `fontSize` 属性是直接连接的，然而在 CSS 声明中的 `style` 属性则带有连字符。带有连字符的声明符合 CSS 管理，并且可用来在外部样式表单中利用 `<mx:Style>` 标签声明样式属性。MXML 中嵌入声明的样式属性利用驼峰样式符合 ActionScript 标准，该标准不支持连字符。内部机制中，CSS 中带连字符的样式声明被转换成驼峰样式并用来利用 `getStyle` 方法检索可用的属性声明。

## 9.2 节. 重写默认的应用程序样式

由 [Marco Casario](#) 供稿。

### 9.2.1. 问题

你希望修改主 Application 容器所被指派的默认样式。

### 9.2.2. 解决办法

将主程序的 styleName 属性设置为 plain.

### 9.2.3. 讨论

Application 容器是 Flex 程序的根容器，并且描述了 Flash 播放器的绘图区域。

它含有定义其样式和外观的默认属性。例如，Application 标记含有分别设置为 8 和 6 像素的 horizontalGap 值与 verticalGap 值(子控件间的水平与垂直距离)。

你有时可能希望重新设置 Application 的所有默认属性。要达到这一目的，你必须将 styleName 属性设置为 plain:

```
<mx:Application  
    xmlns:mx="http://www.adobe.com/2006/mxml"  
    styleName="plain">  
</mx:Application>
```

通过将属性设置为 plain， 默认生成以下内容：

- 所有的补白设置为 0。
- 背景色设置为 #FFFFFF。
- 背景图象被去除
- 所有子控件均靠左对齐。

Plain 样式可以在框架中的 defaults.css 文件内找到。想要了解它们的属性值最初是如何设置的，看下 Flex 框架中默认样式表单的 .plain 样式选择器：

```
.plain  
{  
    backgroundColor: #FFFFFF;  
    backgroundImage: "";  
    horizontalAlign: "left";  
    paddingBottom: 0;  
    paddingLeft: 0;
```

```
    paddingRight: 0;
    paddingTop: 0;
}
```

类选择器能允许你将同一样式应用于不同类型的组件，同样可以使你在同一类型的组见中应用不同的样式。

Flex 同样支持设置类型选择器。如下例子在 <mx:Style> 标签中为程序声明了类型选择器，并且给程序所应用的样式同将 styleName 属性置为 plain 一致：

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:Style>
    Application {
        backgroundColor: #ffffff;
        backgroundImage: '';
        paddingLeft: 0;
        paddingRight: 0;
        paddingTop: 0;
        paddingBottom: 0;
        horizontalAlign: 'left';
    }
</mx:Style>
</mx:Application>
```

## 9.3 节. 嵌入 CSS 样式

### 9.3.1. 问题

你想在你的程序中利用 CSS 为组件嵌入使用样式。

### 9.3.2. 解决办法

在本地定义中定义样式，或者利用 <mx:Style> 标签的 source 属性从外部文件中嵌入使用 CSS 规则。

### 9.3.3. 讨论

样式可以在编译时通过多种方式嵌入进你的 Flex 程序。这一诀窍讲述利用 CSS 语法定义嵌入在你程序中的样式。在 Flex 中运用 CSS，你可以做到以下内容：

- 在 MXML 文件的 <mx:Style> 标签内声明本地样式。
- 设置 <mx:Style> 标签的 source 属性为外部样式表单。

下面的例子在主程序内局部声明两个类样式，这些样式应用于程序显示的 Label 实例：

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">
    <mx:Style>
        .header {
            font-family: 'Arial';
            font-size: 15px;
            font-weight: 'bold';
            color: #FFFFFF;
        }
        .message {
            font-family: 'Arial';
            font-size: 12px;
            color: #336699;
        }
    </mx:Style>
    <mx:Label text="I have a header style!" styleName="header" />
    <mx:Text text="I have a message style!" styleName="message" />
</mx:Application>
```

本例中，类选择器一旦局部设定便逐项设置 Label 组件和 Text 组件。每个选择器中定义的属性值将会标签实例的适当元素，同时明确地修改字体。尽管本例为 Application 容器局部地定义了样式，自定义 MXML 组件同样可以利用 <mx:Style> 标签在本子组件内声明样式。

做为选择，这些样式声明可以放在外部 CSS 文件中并且赋值给<mx:Style>标签的 source 属性。下述代码可添加到以 .css 结尾的文件中，并允许嵌入使用：

```
.header {
    font-family: 'Arial';
    font-size: 15px;
    font-weight: 'bold';
    color: #FFFFFF;
}
.message {
    font-family: 'Arial';
    font-size: 12px;
    color: #336699;
}
```

若要将外部 CSS 文件嵌入到你的程序中，你须将文件路径赋值给<mx:Style>标记的 source 属性。如下嵌入了 CSS 文件的程序片段，会产生和前面例子同样的视觉形象。

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
```

```
layout="vertical">
<mx:Style source="assets/styles/label_styles.css" />
<mx:Label text="Hello!" styleName="header" />
<mx:Text text="Welcome to the example." styleName="message" />
</mx:Application>
```

尽管 Flex 程序的默认样式在框架中的 defaults.css 文件内声明，你可以嵌入很多其他主题进行替换。查看 <flex sdk installation>/frameworks/themes 目录下的相应内容。

## 9.4 节. 修改初始样式属性

### 9.4.1. 问题

你想修改一个组件实例的初始样式属性。

### 9.4.2. 解决办法

利用组件的属性或者子标记嵌入地为指定的样式属性赋值。

### 9.4.3. 讨论

你可以在组件声明的子标记内嵌入地为样式属性赋值。其它方法定义的样式属性均会重新改变程序中本地或外部已经定义的值。考虑下面的例子：

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">
    <mx:Style>
        Label {
            font-family: 'Arial';
            font-size: 20px;
            font-weight: 'bold';
            color: #FFFFFF;
        }
    </mx:Style>
    <mx:Label text="Hello Flex!" color="#336699" />
    <mx:Text text="Hello Flex!" fontStyle="italic">
        <mx:fontFamily>Verdana</mx:fontFamily>
    </mx:Text>
</mx:Application>
```

这里在主程序中本地声明了一个类型选择器，并且指派了显示项中 Label 实例的样式属性。本质上讲，`<mx:Script>` 标记中声明的类型选择器会重写 Flex 框架内 `defaults.css` 文件定义的默认样式，组件元素又会通过嵌入或者子标记重写那些相应的属性。

Label 组件设置的字体颜色不同于类型选择器为标签实例指定的颜色。Text 组件继承类型选择器中声明的所有样式因为 Text 是 Label 的子类。同样地，为标签定义的字体效果也应适用于 Text 组件。Text 实例对类型选择器中未定义的 `fontStyle` 属性进行赋值并在子标记中重写 `fontFamily` 属性；如此，它将值由 Arial 字体变换为 Verdana.

利用嵌入和子标记赋值的方法重写样式是在编译时改变各类组件外观模样的有效方法，这些组件共享类型选择器或类选择器中定义的属性所确定的公共样式。

#### 9.4.4. 另外查看 [章节 9.3](#).

## 9.5 节. 运行时定制样式

### 9.5.1. 问题

你想在运行时定制组件所赋予的样式属性值。

### 9.5.2. 解决办法

利用 `setStyle` 方法重置样式属性值。

### 9.5.3. 讨论

`setStyle` 方法继承自 `mx.core.UIComponent` 的任意子类。你可以利用 `setStyle` 在运行时对已定义的样式对属性进行赋值。`setStyle` 方法的参数分别是样式名称和期望的值。

```
myContainer.setStyle( "backgroundColor", 0xFFFFF );
```

尽管你能利用本地或外部样式表单在编译时定义组件样式，CSS 应该在实例化基础上利用 `styleName` 属性查阅代码并且指定样式，`setStyle` 方法允许你在编译之后改变这些属性值。

下面的例子用 `setStyle` 方法改变组件的颜色属性以响应 Button 实例的点击事件：

查看代码:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    layout="vertical">
        <mx:Style>
            VBox {
```

```

        backgroundColor: #CCCCCC;
        verticalAlign: 'middle';
        horizontalAlign: 'center';
    }
}

.header {
    font-family: 'Arial';
    font-size: 15px;
    font-weight: 'bold';
    color: #FFFFFF;
}

</mx:Style>
<mx:Script>
<! [CDATA[
    // reset color properties with random values.
    private function jumbleColors():void
    {
        holder.setStyle( "backgroundColor", getRandomColor() );
        labelField.setStyle( "color", getRandomColor() );
    }

    private function getRandomColor():uint
    {
        return randomValue() << 16 ^ randomValue() << 8 ^
            randomValue();
    }

    private function randomValue():Number
    {
        return (Math.random() * 512) - 255;
    }
]]>
</mx:Script>
<mx:VBox id="holder" width="200" height="200">
    <mx:Label id="labelField" text="Hello Flex!" styleName="header" />
    <mx:Button label="jumble colors" click="jumbleColors();" />
</mx:VBox>
</mx:Application>

```

在这些组件实例化的基础之上，它们分别应用各自的样式：VBox 利用类型选择器制定样式，同时 Label 通过 styleName 属性指定 .header 样式。按钮的 click 事件处理器为 VBox 和 Label 实例的颜色样式属性应用新值。尽管本例证明了程序内部定义的样式属性被重载，外部 CSS 文件声明的样同样可以利用 setStyle 方法在运行时被重载。

`setStyle` 方法尽管比运用样式表单的计算量开销大，却能方便你控制样式在运行时如何应用和改变。

#### 9.5.4. 另阅 [章节 9.3](#) 和 [章节 9.4](#).

## 9.6 节. 运行时加载 CSS

### 9.6.1. 问题

你想通过运行时加载 CSS 文件替代在编译时嵌入它们的方法，从而尽量保持你的 SWF 的大小。

### 9.6.2. 解决办法

用 Flex 3 SDK 提供的 `mxmlc` 工具将你的 CSS 文件打包，然后利用 `mx.styles.StyleManager` 在运行时加载 CSS 文件。

### 9.6.3. 讨论

编译时加载样式使你不需重新编译程序即可改变样式定义。若要在运行时加载 SWF，你需要使用 `StyleManager` 的 `loadStyleDeclarations` 方法。

与载入 CSS 文件相反，`StyleManager` 加载样式 SWF。若要创造 SWF，可利用 Flex SDK 中 `mxmlc` 命令行工具将 CSS 文件编译成 SWF 文件。首先，第一步创作以 `.css` 结尾的文件并添加如下声明：

```
VBox {  
    backgroundColor: #CCCCCC;  
    verticalAlign: 'middle';  
    horizontalAlign: 'center';  
}  
.header {  
    font-family: 'Arial';  
    font-size: 15px;  
    font-weight: 'bold';  
    color: #FFFFFF;  
}
```

将 CSS 文件保存为 `MyStyles.css`，打开一个命令提示符。在你的系统路径中 Flex 安装目录 `/bin` 文件夹下，提示符后输入以下命令后点击回车：

```
> mxmlc MyStyles.css
```

本命令将创造出与提供的 CSS 文件同名的 SWF 文件。 本例中， MyStyles.swf 会出现在你调用编译器时所指向的目录。

若要加载样式 SWF 并应用于实例组件，你需要调用 StyleManager 的 loadStyleDeclarations 同时将 update 参数设置为 true:

查看代码：

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    creationComplete="appComplete();">
    <mx:Script>
        <![CDATA[
            // load external style SWF and force update to display.
            private function appComplete():void
            {
                StyleManager.loadStyleDeclarations( "assets/styles/MyStyles.swf",
                    true );
            }
        ]]>
    </mx:Script>
    <mx:VBox id="holder" width="200" height="200">
        <mx:Label id="labelField" text="Hello Flex!" styleName="header" />
    </mx:VBox>
</mx:Application>
```

在程序创作完成和编译的样式 SWF 文件成功加载的基础上，CSS 文件中定义的声明应用于组件。作为选择， 你可以通过给 IEventDispatcher 实例定义事件监听器来记录加载样式 SWF 的过程、成功与错误，这些实例是由 loadStyleDeclarations 方法返回的：

查看：

```
private function appComplete():void
{
    // listen to the complete event of external style SWF load
    var dispatcher:IEventDispatcher =
    StyleManager.loadStyleDeclarations("assets/styles/MyStyles.swf
        ", true );
    dispatcher.addEventListener( StyleEvent.COMPLETE,
        styleCompleteHandler );
}
```

```

}

private function styleCompleteHandler( evt:StyleEvent ):void
{
    trace( "Styles Loaded!" );
}

```

如果你选择使用 loadStyleDeclarations 返回的 IEventDispatcher 实例来监视样式 SWF 的加载过程，你可将 update 参数设为 false 同时编程将样式属性应用于组件正如你在定义的事件处理器中装配好。

当你利用<mx:Style>标记的 source 属性在编译时指派外部 CSS 文件时，编译器会中断并检查那个文件是否存在。当利用 StyleManager 在运行时加载样式 SWF，那项中断不执行，允许你为你的程序在任何时候创造文件，无论你的程序发布与否。

## 9.7 节. 运行时声明样式

### 9.7.1. 问题

你想在运行时利用 ActionScript 为 Flex 组件声明和用户化样式。

### 9.7.2. 解决办法

创造 mx.styles.CSSStyleDeclaration 对象，并将和 mx.styles.StyleManager 存储的选择器名联系起来。

### 9.7.3. 讨论

CSSStyleDeclaration 对象拥有能在运行时被设置且用户化的样式属性及值。当你通过<mx:Style>标记在本地或外部文件中定义 CSS 规则，Flex 在编译时自动地为每个声明的选择器生成 CSSStyleDeclaration 对象。若要在运行时生成样式声明，你需调用 StyleManager.setStyleDeclaration 方法，这个方法以选择器标识字符串和 CSSStyleDeclaration 对象作为其构造参数。你能通过 StyleManager.getStyleDeclaration 方法访问在运行或编译时生成的 CSSStyleDeclaration 对象。

用选择器名称去关联运行时生成的 CSSStyleDeclaration 所遵守的规则同关联运行时嵌入的样式声明时遵守的规则一致。一个类的每个实例都是通过类型选择器来应用样式规则，类型选择器就是类名的字符串表示。例如：

```
var vboxStyle:CSSStyleDeclaration = new CSSStyleDeclaration();
```

```
vboxStyle.setStyle( "backgroundColor", 0xFF0000 );
StyleManager.setStyleDeclaration( "VBox", vboxStyle, false );
```

你同样也能利用类选择器去关联一个样式声明。 类选择器以句号（或点）开头，并通过嵌入地将 styleName 属性设置为相应值以应用于任意组件：

```
var redBoxStyle:CSSStyleDeclaration = new CSSStyleDeclaration();
redBoxStyle.setStyle( "backgroundColor", 0xFF0000 );
StyleManager.setStyleDeclaration( ".redBox", redBoxStyle, false );
```

尽管运行时在 StyleManager 中设置声明同编译时 CSS 转换遵守同样的规则，样式属性不能通过调用 CSSStyleDeclaration 实例的 setStyle 方法引用带连字符的参数（如 font-family）生成，这些参数只允许出现在外部文件的声明或者在<mx:Style> 标记中间。

注意

运行时利用 StyleManager 设置样式声明会替换掉先前生成且以同一选择器存储的任何声明。

接下来的例子中，程序在初始化过程中生成 CSSStyleDeclaration 对象，然后其子组件的样式发生变化。

查看代码：

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    initialize="init();">
    <mx:Script>
        <![CDATA[
            // create and store new style declarations.
            private function init():void
            {
                StyleManager.registerColorName( "cookbookBlue",
                    0x339966 );
                var headerStyleDeclaration:CSSStyleDeclaration =
                    new CSSStyleDeclaration();
                headerStyleDeclaration.setStyle( "fontWeight",
                    "bold" );
                headerStyleDeclaration.setStyle( "fontSize", 15 );
                headerStyleDeclaration.setStyle( "color",
                    StyleManager.getColorName( "cookbookBlue" ) );
                var msgStyleDeclaration:CSSStyleDeclaration =
                    new CSSStyleDeclaration();
                msgStyleDeclaration.setStyle( "fontSize", 12 );
```

```

        msgStyleDeclaration.setStyle( "color",
            StyleManager.getColorName( "haloSilver" ) );
        StyleManager.setStyleDeclaration( ".header",
            headerStyleDeclaration, false );
        StyleManager.setStyleDeclaration( ".message",
            msgStyleDeclaration, true );
    }
    // clear previously created styles.
    private function clickHandler():void
    {
        StyleManager.clearStyleDeclaration( ".header", false
        );
        StyleManager.clearStyleDeclaration( ".message", true
        );
    }
}]>
</mx:Script>
<mx:Label text="I'm a header styled through the StyleManager"
    styleName="header"/>
<mx:Text text="I'm a message styled through the StyleManager"
    styleName="message" />
<mx:Button label="clear styles" click="clickHandler();"/>
</mx:Application>

```

本例声明了两个类选择器且存放在 StyleManager 中。样式通过嵌入的 styleName 属性应用于各自的组件。StyleManager 的 setStyleDeclaration 方法中最后一个参数是个布尔标记，用以判断实例化前是否更新样式。制定一个新样式声明的计算开销非常大。同样地，向 StyleManager 添加样式时调用刷新显示通常运用最近声明的样式。

例如通过 styleName 属性将样式应用于组件时，你就改变了那个组件的默认样式。本例子中，当应用于组件的样式通过 clickHandler 方法清除时，组件回复到原来的默认样式。

你能够利用 StyleManager.clearStyleDeclaration 方法去掉样式声明。如同 StyleManager.setStyleDeclaration，本方法中含有一个参数强制程序中的样式立即更新。本例中因响应 StyleManager.clearStyleDeclaration 方法，最后才在去除类样式后刷新界面显示。

StyleManager 类同样允许你通过 registerColorName 方法利用关键字来指派颜色值。你能利用 StyleManager.getColorName 方法访问你在运行时生成的颜色值以及 Flex 框架中默认样式定义的那些颜色—本例中看来就是将 .message 类样式的颜色属性设置为 haloSilver。

## 9.8 节.自定义组件样式属性

### 9.8.1. 问题

你想嵌入地创造并展现那些组件中本身没有的自定义样式属性。

### 9.8.2. 解决办法

给你的自定义组件增加样式元数据并用 `getStyle` 方法返回属性值。

### 9.8.3. 讨论

Flex 框架为组件提供了可用的默认样式属性,但你能利用[Style] 元数据标记为自定义组件声明附加样式属性。在 `<mx:Metadata>`标记中列出样式定义使得你在 MXML 中组件声明内嵌入地指定属性值。你同样能利用 CSS 和 `setStyle` 方法为自定义组件声明样式属性值。同设定默认属性值一样，所有的这些方法强制更新显示界面。

为自定义组件增添附加样式属性使得你能规定影响组件视觉构成的属性。下例中的自定义组件是 `mx.containers.Box` 类的子类，并在`<mx:Metadata>`标记中列出了用于自定义显示的自定义样式：

查看代码:

```
<mx:Box
    xmlns:mx="http://www.adobe.com/2006/mxml"
    width="100%" height="100%"
    horizontalAlign="center" verticalAlign="middle"
    creationComplete="creationHandler();">
    <mx:Metadata>
        [Style(name="teeterAngle", type="Number")]
        [Style(name="activeColor", type="uint", format="Color")]
        [Style(name="idleColor", type="uint", format="Color")]
    </mx:Metadata>
    <mx:Script>
        <! [CDATA[
            import mx.effects.easing.Bounce;

            [Embed("assets/fonts/verdana.TTF", fontName="MyFont")]
            public var verdana_font:Class;

            private function creationHandler():void
            {
                playAnim();
            }
        ]]>
    </mx:Script>
</mx:Box>
```

```

private function effectStartHandler():void
{
    holder.setStyle( "backgroundColor",
        getStyle( "activeColor" ) );
    holder.enabled = false;
    field.text = "Wheeeeee! ";
}
private function effectEndHandler():void
{
    holder.setStyle( "backgroundColor",
        getStyle( "idleColor" ) );
    holder.enabled = true;
    field.text = "Again! Click Me ";
}
private function playAnim():void
{
    rotater.play( [this], true );
}
]]>
</mx:Script>

<mx:Rotate id="rotater"
    effectStart="effectStartHandler();"
    effectEnd="effectEndHandler();"
    originX="{width}" originY="{height}"
    angleFrom="0" angleTo="{getStyle('teeterAngle')} "
    easingFunction="{Bounce.easeIn}" duration="500"
/>
<mx:VBox id="holder"
    width="60%" height="60%"
    horizontalAlign="center"
    verticalAlign="middle"
    click="playAnim();">
    <mx:Text id="field" width="100%">
        selectable="false"
        fontFamily="MyFont"
    />
</mx:VBox>
</mx:Box>
```

本例中声明的开始 Rotate tween 的角度与 teeterAngle 样式属性相关,但 VBox 子件的背景色的变化却与动画的状态无关。本组件中可利用的自定义样式均列在<mx:Metadata>标记中,

同时可通过学习 `getStyle` 方法访问他们相应的属性值。通过在`<mx:Metadata>`标记中声明自定义样式属性，你能够在用户组件的声明阶段嵌入地定义他们的值。

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:cookbook="oreilly.cookbook.*"
    layout="vertical">
    <cookbook:ChompBox
        width="200" height="200"
        backgroundColor="#DDDDDD"
        textAlign="center"
        teeterAngle="45"
        activeColor="#FFDD33" idleColor="#FFFFFF"
    />
</mx:Application>
```

尽管你能为组件添加自定义样式属性，固有的默认样式属性也可以用来自定义，在声明用户组件设置 `textAlign` 属性时可以看到这点。`ChompBox` 实例没有对 `textAlign` 属性值做出操作，也不允许基础的 `mx.containers.Box` 类为继承而来的任何子组件规则程序。

## 9.9 节. 同一个程序中使用多个主题

由 [Andrei Cristian 提供](#)

### 9.9.1. 问题

你想在同一程序中利用多个主题颜色以区分控件。

### 9.9.2. 解决办法

运用容器的 `themeColor` 属性来规定相应的彩色值。

### 9.9.3. 讨论

你能利用`<mx:Canvas>`的 `themeColor` 属性在同一程序中为控件指派不止一个 Flex 主题。运用主题颜色可以改变在滚动、选择等相似的视图处理时被操作控件的外观。

本招的例子向`<mx:Canvas>` 窗口添加三个子组件来显示三个主题的外貌。Flex 程序的默认主题颜色在框架中 `defaults.css` 的全局样式声明中被设为 `haloBlue`，但是还有很多可用只要你喜欢就可用。

首先创造数据提供器 dp，允许其可绑定。其次声明三个包含了 themeColor 属性将被设置的组件的 Canvas 实例。再对每个 Canvas 实例的 mx.controls.ComboBox 和 mx.controls.DataGrid 子组件运用相同的 dp 数据提供器。注视：

查看代码：

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    width="636" height="241">
    <mx:Script>
        <![CDATA[
            [Bindable]
            private var dp:Array =
            [
                {label:'Carole King', Album:'Tapestry',
                 :1},
                {label:'Paul Simon', data:2},
                {label:'Original Cast', data:3},
                {label:'The Beatles', data:4}
            ];
        ]]>
    </mx:Script>
    <mx:Label x="10" y="7" text="Standard Theme" fontWeight="bold"
              color="#ffffffff"/>
    <mx:Label x="218" y="7" text="Green Theme" fontWeight="bold"
              color="#ffffffff"/>
    <mx:Label x="426" y="7" text="Silver Theme" fontWeight="bold"
              color="#ffffffff"/>
    <mx:Canvas x="10" width="200" height="200"
               verticalCenter="10.5">
        <mx:ComboBox x="10" y="10" dataProvider="{dp}"
                     width="180">
        </mx:ComboBox>
        <mx:DataGrid x="10" y="40" width="180" height="120"
                     dataProvider="{dp}">
            <mx:columns>
                <mx:DataGridColumn headerText="Id" dataField="data"
                                   width="30" />
                <mx:DataGridColumn headerText="Artist"
                                   dataField="label" />
            </mx:columns>
        </mx:DataGrid>
        <mx:Button x="10" y="168" label="Button"/>
    </mx:Canvas>
    <mx:Canvas x="426" width="200" height="200"
               verticalCenter="10.5">

```

```

        themeColor="haloSilver"           verticalCenter="10.5"
        backgroundColor="#ffffffff"      cornerRadius="8"
        borderStyle="solid">
<mx:ComboBox x="10" y="10" dataProvider="{dp}"
    width="180">
</mx:ComboBox>
<mx:DataGrid x="10" y="40" width="180" height="120"
    dataProvider="{dp}">
<mx:columns>
    <mx:DataGridColumn headerText="Id" dataField="data"
        width="30" />
    <mx:DataGridColumn headerText="Artist"
        dataField="label" />
</mx:columns>
</mx:DataGrid>
<mx:Button x="10" y="166" label="Button"/>
</mx:Canvas>
<mx:Canvas x="218" width="200" height="200"
    themeColor="haloGreen"           verticalCenter="10.5"
    backgroundColor="#ffffffff"      cornerRadius="8"
    borderStyle="solid" alpha="0.5">
<mx:ComboBox x="10" y="10" dataProvider="{dp}"
    width="180">
</mx:ComboBox>
<mx:DataGrid x="10" y="40" width="180" height="120"
    dataProvider="{dp}">
<mx:columns>
    <mx:DataGridColumn headerText="Id" dataField="data"
        width="30" />
    <mx:DataGridColumn headerText="Artist"
        dataField="label" />
</mx:columns>
</mx:DataGrid>
<mx:Button x="10" y="166" label="Button"/>
</mx:Canvas>
</mx:Application>

```

当在容器实例中应用 themeColor 属性值，那个容器的显示列表中任一子组件通过继承取得主题颜色。尽管你能用关键字符串替代在 Flex 框架中可用的颜色值（例如 haloOrange），你最好还是按照十六进制格式赋任何正确颜色值或者在运行时利用 StyleManager.registerColorName 方法来创造自定义的关键字符串值。

## 9.10 节. 编译主题 SWC

### 9.10.1. 问题

你想将你的样式文件打包放进主题 SWC，然后编译到程序中。

### 9.10.2. 解决办法

运用命令行工具生成一个主题 SWC，接着利用 mxmcl 编译器的主题选项编译程序。

### 9.10.3. 讨论

Shockwave Component (SWC)文件是按照 PKZIP 格式打包的档案文件。SWC 文件允许你在众多开发者间交换大量文件的唯一档案而不是那些文件本身。正如你会生成 MXML 和 ActionScript 文件，并将其打包成 SWC 做为用户组件，你同样能将样式的档案文件打包。若要生成 SWC 文件，你可以利用 Flex SDK 安装路径下 /bin 目录中提供的 compc 工具。

将 Flex SDK 安装路径下 /bin 目录设置在你的系统路径中，下列命令行的输入展示了怎样调用 compc 工具生成名为 MyTheme.swc 的 SWC 文件：

查看代码：

```
> compc –include-file MyStyles.css C:/mystyles/MyStyles.css –o MyTheme.swc
```

其中 include-file 选项有两个参数：引用样式资源的文件名以及资源的文件系统位置。允许使用多个 include-file 选项来打包程序主题所需的所有样式资源。

你能够利用 mxmcl 编译器的主题选项在编译程序时应用主题：

```
> mxmcl MyApplication.mxml –theme MyTheme.swc
```

为更好地理解如何生成主题 SWC 并将文件编入程序，尝试着将 Flex SDK 中提供的一个附加主题打包。

在命令提示符下浏览 SDK 安装路径的 Smoke 主题文件夹： <flex sdk installation>/frameworks/themes/Smoke。由于已经将 Flex SDK 安装路径下 /bin 目录设置在你的系统路径中， 输入下列命令：

查看代码：

```
> compc –include-file Smoke.css Smoke.css –include-file smoke_bg.jpg smoke_bg.jpg  
–o SmokeTheme.swc
```

本条命令在原文件夹下生成 SmokeTheme.swc 文件。 将 SWC 文件移至一个新项目文件夹中，输入下列标记：

查看代码：

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:cookbook="oreilly.cookbook.*"
    layout="vertical">

    <mx:Script>
        <! [CDATA[
            [Bindable]
            private var dp:Array =
                [{label:"Josh Noble", data:0},
                 {label:"Abey George", data:1},
                 {label:"Todd Anderson", data:2}];
            private function clickHandler():void
            {
                messageField.text = "You chose: " +
                    nameCB.selectedLabel;
            }
        ]]>
    </mx:Script>

    <mx:Panel title="Pick One:" width="50%" height="50%"
        paddingLeft="10" paddingTop="10">
        <mx:VBox width="100%">
            <mx:HBox width="100%">
                <mx:Text text="Choose:" styleName="windowStyles" />
                <mx:ComboBox id="nameCB" dataProvider="{dp}" />
                <mx:Button label="select" click="clickHandler();"/>
            </mx:HBox>
            <mx:Text id="messageField" styleName="windowStyles" />
        </mx:VBox>
    </mx:Panel>
</mx:Application>
```

将该文件保存为 MXML 程序。你会注意到 MXML 文件既没有内部声明样式，也没有引用任何的外部样式表单。然后两个 Text 组件的 styleName 属性值均是 windowStyles 的选择器值。.windowStyles 类选择器在 Smoke.css 样式表单中声明，并在赋值前已经打包放进 SmokeTheme.swc 。

当你在编译程序时将 theme 选项参数指定为 SmokeTheme.swc，样式不仅应用显示项中 Text 实例，而且应用于程序的所有组件。

打开命令提示行，浏览含有程序 MXML 文件及先前生成的主题 SWC 的项目路径，并输入下列命令：

```
> mxmcl MyApplication.mxml -theme SmokeTheme.swc
```

本命令利用嵌入的主题 SWC 为你的程序生成 SWF 文件。 打开你项目路径中生成的 SWF 文件，你会看到你的程序应用了 Flex SDK 提供的 Smoke 主题样式。

能被打包进主题 SWC 的文件并不局限于 CSS 和图形文件，还包含后面将要提及的皮肤类文件和字体文件。

#### 9.10.4. 另外查看 [章节 9.2](#) 和 [章节 9.6](#)

## 9.11 节. 应用嵌入字体

### 9.11.1. 问题

你想将字体嵌入程序以确保在任何机器上的外貌感观保持一致，不用去管用户系统字体如何。

### 9.11.2. 解决办法

通过在 ActionScript 中利用[Embed]元标记或者在 CSS 中使用@font-face 以嵌入字体。

### 9.11.3. 讨论

在程序中嵌入字体的原则确保文本应用的样式是完整的，不需要考虑用户机器上系统字体。你能利用 ActionScript 和 CSS 嵌入字体。

下面例子描述了怎样在 ActionScript 嵌入字体以为 Flex 组件应用字体：

查看代码：

```
<mx:Application  
    xmlns:mx="http://www.adobe.com/2006/mxml"  
    layout="vertical">  
    <mx:Script>  
        <! [CDATA [  
            [Embed(source="assets/fonts/verdana.ttf",
```

```

        fontName="MyVerdana")]
    private var _verdana:Class;
] ]>
</mx:Script>
<mx:Label text="i have no custom style." />
<mx:Label text="i have verdana style!" fontFamily="MyVerdana"
/>
</mx:Application>

```

通过对字体应用[Embed] 标记，在组件实例化并对 fontFamily 样式属性赋值时，你能明确做为参考的 fontName 属性。本例中，字体被嵌入同时连带的 Class 实例成员也被声明。然而在对 font family 赋值时并不需要引用那个变量，但其却被增设为与嵌入的指令相关联的定义。

前面的代码嵌入了有无格式或者普通、样式外观的字体。该字体能应用于任何组件并显示在屏幕上。无论如何你都会发现一些组件显示内含的特定样式，例如 mx.controls.Button 及其黑体字标签，如果没有嵌入特定的字体它们的文本将会显示失败。除非你嵌入相应的字体并同时设置 fontWeight 属性，否则不能将粗字体应用于组件。例如：

查看代码：

```

[Embed(source="assets/fonts/verdanab.TTF", fontName="MyVerdana" fontWeight="bold")]
private var _verdanaBold:Class;
...
<mx:Button label="I have style" fontFamily="MyVerdana" />

```

大多数字体有四种主要的打印机字体样式：无格式、粗体、斜体、粗斜体。你能嵌入任意的打印机字体进 Flex 程序，但他们需要用正确的属性分开声明。

利用 ActionScript 嵌入字体是可行的方法，管理和创造待嵌入于连带类成员中的大量字体将变得很麻烦。你也能通过 CSS 嵌入字体，既能在 MXML 文件内部也可以来自外部，按如下方法利用@font-face 指令：

查看代码：

```

<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">
<mx:Style>
    @font-face {
        src: url('assets/fonts/verdana.TTF');
        font-family: MyVerdana;
    }
    @font-face {
        src: url('assets/fonts/verdanai.TTF');

```

```

        font-family: MyVerdana;
        font-style: italic;
    }
@font-face {
    src: local('Verdana');
    font-family: MyVerdana;
    font-weight: bold;
}
Application {
    font-family: MyVerdana;
}
.italicStyle {
    font-style: italic;
}
</mx:Style>
<mx:Text text="i have MyVerdana fontFamily styling." />
<mx:Label text="i am in italics with MyVerdana styling."
    styleName="italicStyle" />
<mx:Button label="i am a button. i am bold" />
</mx:Application>
```

嵌入了 Verdana 字体的三种打印机字体样式，并对程序设置了默认字体。对 Application 应用类型选择器，程序中的所有文本使用嵌入的字体。因为 Button 实例中默认样式的 fontWeight 样式属性值为 bold，本例中你不需要为应用于按钮而声明样式因为粗体字已嵌入而且在 Application 选择中使用 fontFamily 默认设置。

你也不例外可以在@font-face 指令中通过使用 local 函数替代 url 函数来依靠指定字体名而不是位置的方法嵌入字体：

```

@font-face {
    src: local('Verdana');
    font-family: MyVerdana;
    font-weight: bold;
}
```

当使用 local 函数时，你不需要指明打印机字体名称（例如本例中的 verdanab），反而要指明字体名称。可以通过在声明中添加相对应属性来嵌入粗体、斜体及粗斜体打印机字体。

将字体嵌入程序中会使 SWF 文件尺寸变大。前面的例子演示了为所有可用的字符集嵌入字体的大概。你能根据嵌入的字体在@font-face 规则中通过声明字符范围可以膨胀程序的原理缩小尺寸：

```
@font-face {
```

```
src: local('Verdana');
font-family: MyVerdana;
unicodeRange: U+0041-U+005A, U+0061-U+007A, U+002E;
}
```

你能利用 `unicodeRange` 属性明确待嵌入的一系列或单一字符。子集的声明中间用逗号分开。你能利用连字符(-)同时声明单个字符来明确范围。本例中，标准拉丁字母的大小写符号 (A 至 Z 与 a 至 z) 和句点(.) 符号都被嵌入了。

## 9.12 节. 从 SWF 文件中嵌入字体

### 9.12.1. 问题

你想从 SWF 文件中嵌入字体，并将其应用在你的程序中。

### 9.12.2. 解决办法

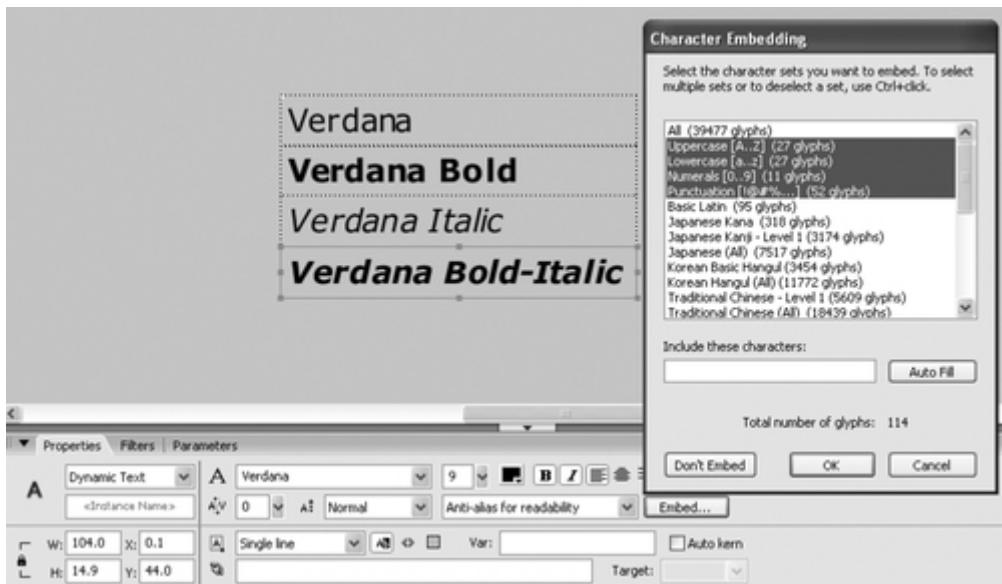
生成一个嵌有字体的 SWF，然后利用 `@font-face` 指令声明你想在程序中嵌入的打印机字体。

### 9.12.3. 讨论

你能从单个包含嵌入字体的 SWF 文件中为你的程序嵌入多种打印机字体样式。与从字体文件中嵌入字体相反，从 SWF 中嵌入的方法具有可携带性的优势。你能将你系统中可用的字体嵌入进 SWF 文件，然后传给其他开发人员，他们不再需要担心编译时是否内在地包含相应的字体。

想要在 SWF 文件中嵌入字体，首先在 Flash IDE 中生成新的 FLA 文件，然后向舞台中添加带有指定的打印机字体集的动态文本域，并选择待嵌入的字符范围 ([图 9-1](#))。

图 9-1. 添加动态文本域并嵌入字体范围。



本例中，每个动态文本域都添加了 Verdana 字体并选择了打印机字体属性。正如你能看到的，粗斜体样式是从属性面板中获取的。通过选择面板中的 Embed 按钮，你能明确字符范围以减小文件尺寸。找一个符合你的项目名称保存该 FLA 文件（本例中该文件命名为 Verdana.fla）然后发布剪辑。

声明 SWF 文件中嵌入的字体和声明字体文件中字体的方法相似。若要从 SWF 文件中嵌入字体，不用通过 url 函数定义字体文件位置或者通过 local 函数定义字体名称，你只要利用 url 指令指明 SWF 文件的位置。方法如下：

```
@font-face {
    src: url("styles/fonts/Verdana.swf");
    font-family: Verdana;
}
```

设置粗体以及斜体：

```
@font-face {
    src: url("styles/fonts/Verdana.swf");
    font-family: Verdana;
    font-weight: bold;
    font-style: italic;
}
```

嵌入的字体是 ActionScript API 中 flash.text.Font 的子类。你能通过 Font 类的 enumerateFonts 静态方法访问一系列设备和嵌入的字体。enumerateFonts 的参数是一个标记用来判断返回数

组中是否包括设备字体。下面的例子利用嵌入的字体填充数据单元，这些字体嵌入在<mx:Style>标记中声明的字体 SWF 文件中：

查看代码：

```
<mx:Application  
    xmlns:mx="http://www.adobe.com/2006/mxml"  
    layout="vertical" creationComplete="creationHandler();">  
    <mx:Style>  
        @font-face {  
            src: url("assets/fonts/Verdana.swf");  
            font-family: Verdana;  
        }  
        @font-face {  
            src: url("assets/fonts/Verdana.swf");  
            font-family: Verdana;  
            font-weight: bold;  
        }  
        @font-face {  
            src: url("assets/fonts/Verdana.swf");  
            font-family: Verdana;  
            font-style: italic;  
        }  
  
        @font-face {  
            src: url("assets/fonts/Verdana.swf");  
            font-family: Verdana;  
            font-weight: bold;  
            font-style: italic;  
        }  
        Label {  
            font-family: Verdana;  
            font-size: 15px;  
        }  
        .verdanaBoldStyle {  
            font-weight: bold;  
        }  
        .verdanaItalicStyle {  
            font-style: italic;  
        }  
        .verdanaBoldItalicStyle {  
            font-weight: bold;  
            font-style: italic;  
        }  
    </mx:Style>
```

```

<mx:Script>
<! [CDATA[
    [Bindable] private var fontArray:Array;
    [Bindable] private var selectedFontStyle:String;
    [Bindable] private var selectedFontWeight:String;

    private function creationHandler():void
    {
        var embeddedFonts:Array = Font.enumerateFonts();
        embeddedFonts.sortOn( "fontStyle",
            Array.CASEINSENSITIVE );
        fontArray = embeddedFonts;
    }
    private function changeHandler():void
    {
        var italic:RegExp = /italic/i;
        var bold:RegExp = /bold/i;
        selectedFontStyle =
            ( String( fontGrid.selectedItem.fontStyle
            ).match(italic) )?
            'italic' : 'normal';
        selectedFontWeight =
            ( String( fontGrid.selectedItem.fontStyle
            ).match(bold) )? 'bold': 'normal';
    }
}

]]>
</mx:Script>
<mx:Label text="Select a font form the grid below:" />
<mx:DataGrid id="fontGrid" dataProvider="{fontArray}"
    change="changeHandler();">
    <mx:columns>
        <mx:DataGridColumn headerText="Font Name"
            dataField="fontName" width="150" />
        <mx:DataGridColumn headerText="Font Style"
            dataField="fontStyle" />
        <mx:DataGridColumn headerText="Font Type"
            dataField="fontType" />
    </mx:columns>
</mx:DataGrid>
<mx:Label width="100%" textAlign="center"
    text="{fontGrid.selectedItem.fontName + ' : ' +
    fontGrid.selectedItem.fontStyle}"
    fontFamily="{fontGrid.selectedItem.fontName}">

```

```
        fontStyle="{selectedFontStyle}"
        fontWeight="{selectedFontWeight}"
    />
</mx:Application>
```

`creationComplete` 事件处理器访问程序中的嵌入字体数组，然后更新用作 `DataGrid` 实例的 `dataProvider` 的可绑定的 `fontArray` 成员。一旦从数据格子中选择打印机字体，标签的字体样式属性在运行时就会更新。

#### 9.12.4. 另外查看 [章节 9.10.](#)

## 9.13 节. 嵌入图像的皮肤

由 Kristopher Schultz 提供

### 9.13.1. 问题

你想用自定义的图像为组件的视觉元素应用皮肤。

### 9.13.2. 解决办法

利用组件的样式属性提供自定义的 JPEG、GIF 或者 PNG 图像。这些属性能在组件实例中嵌入地通过 MXML 或者作为 CSS 样式定义的部分而直接设定。

### 9.13.3. 讨论

内置的 Flex 主题默认地为组件应用计划好的皮肤类。你能创造自定义编程实现皮肤类或者指定图形元素为皮肤，以修改组件的视觉组成。能应用皮肤的组件通常含有一系列能在用户交互时展现出来的皮肤状态或者阶段。这样当生成自定义图形皮肤时，这将有助于牢记组件的不同交互状态。

下面例子中，自定义图像应用于一个 `Button` 组件多个背景状态。代码很明确地在 `Button` 实例上嵌入地直接设定 `upSkin`、`overSkin`、`downSkin` 和 `disabledSkin` 样式属性：

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal" backgroundColor="#FFFFFF">
    <mx:Button label="" 
        upSkin="@Embed('assets/images/text_button_up.png')"
        overSkin="@Embed('assets/images/text_button_over.png')"
        downSkin="@Embed('assets/images/text_button_down.png')"
```

```
disabledSkin="@Embed('assets/images/text_button_disabled.png')"
/>
</mx:Application>
```

注意 @Embed 编译指令强制地将图像资源绑定进最终程序的 SWF 文件中。通过这种方法嵌入图像很重要，因为这样当按钮从一种状态转换为另一种时不会产生加载延迟。

由于例子中标签属性设置为空字符串('')值，文本将不会显示在图形元素上面。本招所用的图像文件，存放在随书的代码例子中，它们本身就绘有标签。

如果你想在 Button 实例上面显示文本并调整图像大小以适应标签文本的长度变化，你能利用 Flex 框架的 scale-9 功能在你的皮肤属性中包含缩放网格信息。缩放网格值取决于图形元素的大小和设计。下面例子在 CSS 样式定义中设置了皮肤属性同时通过 styleName 属性将样式应用于多个 Button 实例。在 CSS 中使用 Embed 指令时不使用前缀@ 符合，和在脚本中嵌入地方法一致。

查看代码：

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal" backgroundColor="#FFFFFF">
    <mx:Style>
        .customButton {
            color: #FFFFFF;
            text-roll-over-color: #FFFFBB;
            text-selected-color: #9999FF;
            disabled-color: #333333;
            up-skin: Embed(
                source="assets/images/button_up.png",
                scaleGridTop="15",
                scaleGridBottom="20",
                scaleGridLeft="15",
                scaleGridRight="28");
            over-skin: Embed(
                source="assets/images/button_over.png",
                scaleGridTop="15",
                scaleGridBottom="20",
                scaleGridLeft="15",
                scaleGridRight="28");
            down-skin: Embed(
                source="assets/images/button_down.png",
                scaleGridTop="15",
                scaleGridBottom="20",
                scaleGridLeft="15",
                scaleGridRight="28");
            disabled-skin: Embed(
```

```

        source="assets/images/button_disabled.png",
        scaleGridTop="15",
        scaleGridBottom="20",
        scaleGridLeft="15",
        scaleGridRight="28");
    }
</mx:Style>
<mx:Button label="This button is enabled"
    styleName="customButton" />
<mx:Button label="This button is disabled"
    styleName="customButton"
    enabled="false"
/>
</mx:Application>
```

利用 Flex 框架的 scale-9 功能，你定义图像的九个部分以便互相独立地调整尺寸。被嵌图像的四角，由四个缩放网格属性推导而来，不能调整而且放置的位置基于其它格子元素的水平和垂直比例大小。通过为缩放网格属性赋值而使用 scale-9 的方法，对于嵌入带有明显的完整边框的图像时很有效果，正如本例中按钮图形的圆角。

## 9.14 节. 从 SWF 文件中嵌入皮肤

### 9.14.1. 问题

你想在 SWF 文件中保存一堆图像，以便用来为组件嵌入皮肤。

### 9.14.2. 解决办法

生成一个 SWF 文件，用多个记号标记表示组件多个状态的输出。

### 9.14.3. 讨论

在 SWF 文件中生成一堆图像皮肤是将设计展现给其它开发人员的方便方法。另外它实现了将矢量图形做为皮肤的可能性。

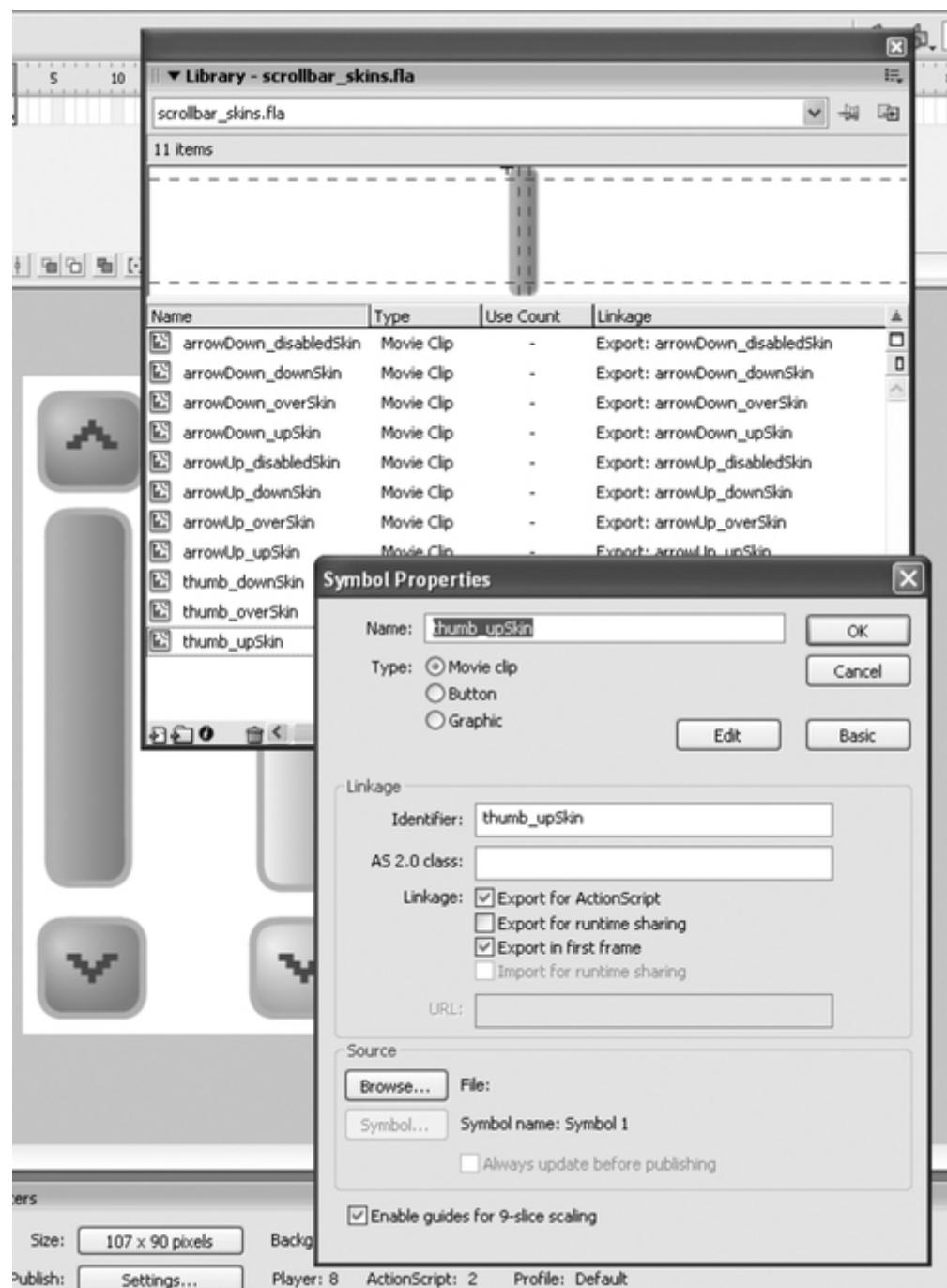
使用来自 Flash IDE 库的元件的属性面板，你能选择用于引用图形元件的类名和超类链接。由于你在 Flash IDE 中仅使用基于 ActionScript 的项目所提供的类库，所以不能选择 Flex 框架中提供的基础类。那样很好，因为在嵌入到程序中时那些元件被转换成它们的 Flex 副本。例如，代表 SWF 文件库输出的 MovieClip 实例被转换成 MovieClipAsset 对象，这些对象随后被添加到 Flex 程序的显示界面中。

创造用来扩展 MovieClip 的资料对有多个状态的图像很有用，但如果使用静态图像，将元件的基础类设置为 `flash.display.Sprite` 类就足够了。当被嵌入到 Flex 程序中时，`Sprite` 对象将被转换成 `SpriteAsset` 对象。

若要在 Flash IDE 中生成静态图像皮肤，首先打开一个新的 FLA 同时从类库中生成一个新的 MovieClip 元件。将一个图像文件导入至 MovieClip 实例的舞台上 或者利用 Flash IDE 绘图工具创造应用于程序的基于矢量的皮肤。为使元件能嵌入到 Flex 程序中，右击库中的元件，从背景菜单中选择访问元件属性面板， 然后选择输出为 ActionScript 的复选框。基础类域的默认值为 `flash.display.MovieClip`。你可以保持那个值，同样可以将它变为 `flash.display.Sprite`，因为图像被当作静态，或者没有很多依赖时间线的状态。

[图 9-2](#) 展示了设置元件属性面板的高级功能来生成应用于多个状态和滚动条元素的皮肤。从高级功能中，你可以选择嵌入的属性(也可从链接面板中访问)同时有机会利用选择 允许使用 9 切片缩放向导 在元件中显示可拖拽的 9 切片缩放网格线。如果你手动地设置元件的 `scale-9 guides`，Flex 编译器自动地设置缩放网格属性(`scaleGridLeft`, `scaleGridRight`, `scaleGridTop`, `scaleGridBottom`)的样式值。

图 9-2. 在 SWF 文件中生成并导出矢量图像



生成这些皮肤，并且给每个皮肤唯一的链接标识符同时勾选输出，以一个适合你项目的名称(本例中名称为 scrollbar\_skins.fla)保存 FLA 文件，然后发布剪辑。

从 SWF 文件中嵌入图像元件与嵌入单一的图像文件相似。你在 Embed 指令中将源属性设置为 SWF 文件，然后将元件属性值设置为你要嵌入的库元件的链接标识符的内容。例如：

查看代码：

```
thumbUpSkin: Embed(source="styles/scrollbar_skins.swf", symbol="thumb_upSkin");
```

你同样可以在源定义中利用英镑符(#)界定 SWF 位置和元件链接标识符的方法来设定元件属性值，如下：

```
thumbUpSkin: Embed(source="styles/scrollbar_skins.swf#thumb_upSkin");
```

下面的例子使用生成的样式 SWF (scrollbar\_skins.swf,可在本章代码中找到) 为 ScrollBar 类的实例应用皮肤。Flex 组件架构的 mx.controls.scrollClasses.ScrollBar 类由多个状态可变的元素组成。滚动条组件的每个元素都有表示当前状态的视觉表示。状态能传递当前用户动作和激活信息。例如，当一个用户用鼠标在滚动条组件上盘旋时，组件输入一个 "over" 状态。利用 ScrollBar 类提供的相应的样式属性，将元件皮肤应用于每个元素的每个状态阶段是可行的。

查看代码：

```
<mx:Application  
    xmlns:mx="http://www.adobe.com/2006/mxml"  
    layout="vertical">  
    <mx:Style>  
        ScrollBar  
        {  
            trackColors: #0099CC, #0099CC;  
            borderColor: #99FFFF;  
  
            thumbUpSkin:  
                Embed(source="assets/styles/scrollbar_skins.swf",  
                      symbol="thumb_upSkin");  
            thumbOverSkin:  
                Embed(source="assets/styles/scrollbar_skins.swf",  
                      symbol="thumb_overSkin");  
            thumbDownSkin:  
                Embed(source="assets/styles/scrollbar_skins.swf",  
                      symbol="thumb_downSkin");  
            upArrowUpSkin:  
                Embed(source="assets/styles/scrollbar_skins.swf",  
                      symbol="arrowUp_upSkin");  
            upArrowOverSkin:  
                Embed(source="assets/styles/scrollbar_skins.swf",  
                      symbol="arrowUp_overSkin");  
            upArrowDownSkin:  
                Embed(source="assets/styles/scrollbar_skins.swf",  
                      symbol="arrowUp_downSkin");  
            upArrowDisabledSkin:
```

```

        Embed(source="assets/styles/scrollbar_skins.swf",
               symbol="arrowUp_disabledSkin");
    downArrowUpSkin:
        Embed(source="assets/styles/scrollbar_skins.swf",
               symbol="arrowDown_upSkin");
    downArrowOverSkin:
        Embed(source="assets/styles/scrollbar_skins.swf",
               symbol="arrowDown_overSkin");
    downArrowDownSkin:
        Embed(source="assets/styles/scrollbar_skins.swf",
               symbol="arrowDown_downSkin");
    downArrowDisabledSkin:
        Embed(source="assets/styles/scrollbar_skins.swf",
               symbol="arrowDown_disabledSkin");
    }
</mx:Style>

<mx:Label text="Keep typing to see the scroll!"
           color="0xFFFFFFFF" />
<mx:TextArea width="200" height="150"
              text="Lorem ipsum dolor sit amet..." 
              borderColor="0x0099CC" />
    <mx:HScrollBar width="200" enabled="false" />
</mx:Application>

```

ScrollBar 实例的类型选择器在<mx:Style>标记中为程序进行了本地地声明。在声明中定义了三个子元素的状态皮肤：压下，向上箭头和向下箭头。

mx.controls.TextArea 组件显示基于文本长度的滚动条。想要看到利用样式 SWF 应用了皮肤的垂直滚动条，在文本域中键入更多内容。mx.controls.HScrollBar 的实例也被加入到显示界面中，不能说明皮肤对两种滚动条实例(VscrollBar 和 HScrollBar)应用结果相同而且样式属性不依赖于方向。

#### **9.14.4. 另外查看 [章节 9.12.](#)**

## **9.15 节. 编程实现组件应用皮肤**

### **9.15.1. 问题**

你想更好地控制视觉元素如何在不指定图像皮肤时显示在组件中。

## 9.15.2. 解决办法

生成一个继承 mx.skins.ProgrammaticSkin 类的自定义皮肤类并重载保护类型的 updateDisplayList 方法。

## 9.15.3. 讨论

与图像皮肤相反，编程实现皮肤需要对 ActionScript 更高级的理解，但提供了组件视觉表示的更深入的控制。编程实现的皮肤类是利用一些显示对象，这些对象利用绘图 API 显示皮肤元素并使你能运用在指定图像皮肤时可能会打消的一些其它样式属性值。

通常来讲 Flex 有两类组件：容器和控件。窗口有边框皮肤来表示显示背景，然而控件通常有一系列描述状态的皮肤（弹起、按下、经过和失效）。当然也有一些例外，如 TextInput 控件使用边框皮肤。总之，在编程生成自定义皮肤之前注意组件的视觉构成是很重要的。

当为控件编程生成皮肤时你需要继承 mx.skins.ProgrammaticSkin 类。当为容器生成自定义皮肤时，你需生成 ProgrammaticSkin 的子类，通常也生成 Border（有 borderMetrics 属性）或者 RectangleBorder（Border 的子类且支持背景样式）的子类。[表 9-1](#) 列出 Flex API 中用于编程生成自定义皮肤的基础类。

**表 9-1.** Flex API 中用于编程生成自定义皮肤的基础类

类	用途
mx.skins.ProgrammaticSkin	所有编程实现自身的皮肤元素的基础类。
mx.skins.Border	绘制矩形和非矩形边框的基础类。包含 borderMetrics 属性。ProgrammaticSkin 的子类。
mx.skins.RectangularBorder	绘制矩形边框的基础类。 Border 的子类。支持 backgroundImage、backgroundSize 和 backgroundAttachment 样式。

当你为自定义皮肤生成这些基类的子类时，你重载了 ProgrammaticSkin 中保护类型的 updateDisplayList 方法以增加基于样式属性值的绘图方法。例如：

查看代码：

```
package
{
    import mx.skins.ProgrammaticSkin;
    public class MyCustomSkin extends ProgrammaticSkin
    {
        public function MyCustomSkin() {}

        override protected function
            updateDisplayList(unscaledWidth:Number,
            unscaledHeight:Number ):void
```

```

{
    // grab the value for the backgroundColor style
    property
    var backgroundColor:Number =
        getStyle( "backgroundColor" );
    // implement drawing methods.
}
}
}

```

当每次由于更新属性而需绘制或重新绘制皮肤元素时，都要在类实例中内部调用 updateDisplayList 方法。同样地在重载该方法过程中，你能够通过 getStyle 方法访问样式属性并且利用绘图 API 自定义显示。

你也可以通过重载 measuredWidth 和 measuredHeight 的只读属性，从而有选择地设置皮肤元素的默认尺寸。方法如下：

查看代码：

```

package oreilly.cookbook
{
    import mx.skins.ProgrammaticSkin;
    public class MyCustomSkin extends ProgrammaticSkin
    {
        private var _measuredWidth:Number;
        private var _measuredHeight:Number;

        public function MyCustomSkin()
        {
            _measuredWidth = 120;
            _measuredHeight = 120;
        }
        // return defaulted constant for width
        override public function get measuredWidth():Number
        {
            return _measuredWidth;
        }
        // return defaulted constant for height
        override public function get measuredHeight():Number
        {
            return _measuredHeight;
        }

        override protected function
            updateDisplayList( unscaledWidth:Number,

```

```

        unscaledHeight:Number ) :void
    {
        // grab the value for the backgroundColor style
        property
        var backgroundColor:Number =
            getStyle( "backgroundColor" );
        // implement drawing methods.
    }
}
}

```

如果你不重载 measuredWidth 和 measuredHeight 的只读属性，它们的默认值将设为 0。前面的例子中，应用编程实现皮肤类的组件将默认值设置为 120 宽和 120 高。尺寸可以在组件声明阶段利用 width 和 height 属性进行修改。

下面代码片段生成自定义边框皮肤，应用于窗口的 borderSkin 样式属性：

查看代码：

```

package oreilly.cookbook
{

    import mx.graphics.RectangularDropShadow;
    import mx.skins.Border;

    public class CustomBorder extends Border
    {

        private var _dropShadow:RectangularDropShadow;
        private static const CNR_RAD:Number = 5;
        private static const POINT_POS:String = 'bl';
        private static const BG_COL:uint = 0x336699;
        private static const BG_ALPHA:Number = 1.0;

        public function CustomBorder()
        {
            super();
            _dropShadow = new RectangularDropShadow();
        }

        private function getCornerRadiusObj( rad:Number,
            pointPos:String ):Object
        {
            var pt:String = pointPos ? pointPos : POINT_POS;
            return {tl:rad, bl:0, tr:rad, br:rad};
        }
    }
}

```

```

override protected function
    updateDisplayList( unscaledWidth:Number,
                      unscaledHeight:Number ):void
{
    super.updateDisplayList(unscaledWidth, unscaledHeight );
    var cornerRadius:Number = getStyle( "cornerRadius" ) ?
        getStyle( "cornerRadius" ) :CustomBorder.CNR_RAD;
    var backgroundColor:Number =
        getStyle( "backgroundColor" ) ?
            getStyle( "backgroundColor" ):CustomBorder.BG_COL;
    var backgroundAlpha:Number =
        getStyle( "backgroundAlpha" ) ?
            getStyle( "backgroundAlpha" ):CustomBorder.BG_ALPHA;
    var cnrRadius:Object = getCornerRadiusObj( cornerRadius,
        getStyle( "pointPosition" ) );
    graphics.clear();
    drawRoundRect( 0, 0, unscaledWidth, unscaledHeight,
                  cnrRadius , backgroundColor, backgroundAlpha );
    _dropShadow.tlRadius = cnrRadius.tl;
    _dropShadow.blRadius = cnrRadius.bl;
    _dropShadow.trRadius = cnrRadius.tr;
    _dropShadow.brRadius = cnrRadius.br;
    _dropShadow.drawShadow( graphics, 0, 0,
                           unscaledWidth, unscaledHeight );
}
}
}

```

无论何时调用 `updateDisplayList` 方法，图像层将被清空并利用继承而得的 `drawRoundRect` 方法重新绘制。另外将应用一个阴影滤镜；这是编程实现皮肤相对图像皮肤的一个优势，因为这个技术允许你访问如滤镜等更多底层功能。

你能利用指定其它样式属性值的任一方法来应用编程实现的皮肤：嵌入地，利用 `setStyle` 方法或者 CSS。下面的例子描述了这几项技术：

```

<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    initialize="initHandler();">
<mx:Script>

```

```

<! [CDATA[
    import oreilly.cookbook.CustomButton;

    private function initHandler():void
    {
        myVBox.setStyle( "borderSkin", CustomBorder );
    }
]]>
</mx:Script>
<mx:VBox width="100" height="50"
    borderSkin="oreilly.cookbook.CustomButton" />
<mx:VBox width="50" height="20"
    borderSkin="{CustomBorder}" />
<mx:VBox id="myVBox" width="80" height="20" />
</mx:Application>

```

当嵌入地或利用 `setStyle` 方法应用皮肤时，你可以导入类并用它的简短名称或者利用完整的合格的类名明确皮肤。当运用简短名称来嵌入地应用皮肤类时，需要用花括号({})来求导入类的值。

利用 CSS，你需要在 `ClassReference` 指令中明确输入完整的分类的类名。这里通过使用类型选择器将 `CustomBorder` 皮肤应用于一个 `mx.containers.VBox` 实例的 `borderSkin` 属性：

```

<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">

    <mx:Style>
        VBox {
            borderSkin:
ClassReference("oreilly.cookbook.CustomButton");
            cornerRadius: 30;
            pointPosition: 'bl';
            backgroundColor: #999933;
            backgroundAlpha: 1;
            paddingLeft: 5;
            paddingRight: 5;
        }
    </mx:Style>

    <mx:VBox id="myBox"
        width="120" height="100"
        verticalAlign="middle" horizontalAlign="center">
        <mx:Text text="i'm a styled VBox!" textAlign="center" />
    </mx:VBox>

```

```
</mx:Application>
```

在重载 CustomBorder 中的 updateDisplayList 方法时，VBox 选择中声明的其它的样式属性 (cornerRadius、pointPosition 等等) 用于自定义显示 VBox 实例的背景内容。

#### 9.15.4. 另外查看 [章节 9.3](#) 和 [章节 9.5](#).

## 9.16 节. 编程实现状态控件的皮肤

### 9.16.1. 问题

你想编程实现皮肤，该皮肤能够解决控件显示不同状态的问题。

### 9.16.2. 解决办法

生成 mx.skins.ProgrammaticSkin 的子类，然后利用 updateDisplayList 方法依据 name 属性值更新其显示内容。可以嵌入地利用 skin 样式属性，或者利用 setStyle 方法，再或者利用 CSS 三种方法应用编程实现的自定义皮肤。

### 9.16.3. 讨论

通常控件都有状态或包含有状态的子元素。将编程实现的皮肤做为样式来显示控件状态与用其来显示容器背景是一样的原理：样式属性值用编程实现的皮肤类的名称定义。控件皮肤视觉元素的更新与容器的更新不同，但是大多数控件基于一个状态显示其内容。

为状态组件编程生成皮肤类的过程中，你生成 ProgrammaticSkin 的一个子类同时重载保护类型的 updateDisplayList 方法。在重载 updateDisplayList 过程内，你能利用 switch case 语句响应状态名称值。每个皮肤片段的 name 属性值与控件的样式属性是同一字符串值，而且嵌入地应用于你待处理的控件。

下面的例子编程生成能依据以下状态更新显示的皮肤，upSkin、overSkin、downSkin 和 disabledSkin：

查看代码：

```
package oreilly.cookbook

{
    import flash.filters.BevelFilter;
    import mx.skins.ProgrammaticSkin;

    public class CustomButtonSkin extends ProgrammaticSkin
    {
        private var _measuredWidth:Number = 100;
```

```

private var _measuredHeight:Number = 100;
// override measuredWidth and measuredHeight sets default size.

override public function get measuredWidth():Number
{
    return _measuredWidth;
}

override public function get measuredHeight():Number
{
    return _measuredHeight;
}

// update display based on state phase name.

override protected function
updateDisplayList( unscaledWidth:Number,
    unscaledHeight:Number ):void
{
    var backgroundAlpha:Number = 1.0;
    var bevelAngle:Number = 45;
    var backgroundColor:uint;

// assign property values based on state name
switch( name )
{
    case "upSkin":
        backgroundColor = 0xEEEEEE;
        break;
    case "overSkin":
        backgroundColor = 0xDDDDDD;
        break;
    case "downSkin":
        backgroundColor = 0xDDDDDD;
        bevelAngle = 245;
        break;
    case "disabledSkin":
        backgroundColor = 0xFFFFFFF;
        backgroundAlpha = 0.5;
        break;
}
// clear the display and redraw with values assigned in switch

graphics.clear();
drawRoundRect( 0, 0, unscaledWidth, unscaledHeight,
    null, backgroundColor, backgroundAlpha );
// apply bevel filter
var bevel:BevelFilter = new BevelFilter(2, bevelAngle );

```

```

        this.filters = [bevel];
    }
}
}

```

若要应用编程实现的响应组件状态阶段的皮肤，你需将类名做为 skin 样式属性值。下例利用 CSS 在一个 ClassReference 指令中提供 CustomButtonSkin 类的完整合法的类名：

```

<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">
<mx:Style>
    .customButton {
        skin:
            ClassReference( "oreilly.cookbook.CustomButtonSkin" );
    }
</mx:Style>
<mx:Button id="skinnedBtn"
    styleName="customButton"
    label="click me" />
<mx:Button label="toggle enabled"
    click="{skinnedBtn.enabled = !skinnedBtn.enabled;}"/>
</mx:Application>

```

skin 样式属性是 Flex 3 SDK 中的新增功能。在 Flex 2 中即使每个值都是同样的 ClassReference 值，每个皮肤状态也需要重新定义。在 Flex 中，你能指定单一的 skin 属性应用于每个状态。定义完 skin 属性后，你仍然可以重载特定的状态皮肤属性，例如 overSkin，以对状态组件的自定义拥有更多控制。

#### 9.16.4. 另外查看 [章节 9.14.](#)

## 9.17 节. 从 SWF 文件中生成动态皮肤

### 9.17.1. 问题

你想在 Flash IDE 中创造一个按钮，它的每个状态有不同的动作。

### 9.17.2. 解决办法

创建一个 FLA 并在此 FLA 中创建一个意味着有多个帧和动作的输出的 MovieClip 元件。利用生成的 SWF 和 MovieClip 名称，在源代码和元件引用中通过利用 [Embed] 元数据标记将 MovieClip 为你动作的拓展。把一个 mx.core.UIComponent 子类当作 mx.controls.Button 实例的皮肤，然后利用继承的 currentState 属性控件当前状态并在 Button 实例内修改 MovieClip 的动作帧。

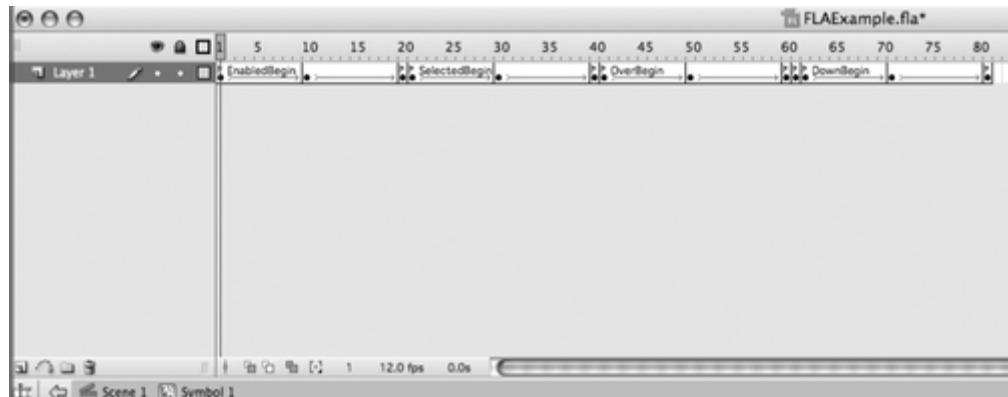
### 9.17.3. 讨论

应用 SWF 文件中的动画皮肤类似于在组件声明阶段应用编程实现的皮肤。两者之间皮肤的生成不同于，前者的动作由元件中帧提供，相反后者则是编程实现组件的动作元素。

若要生成待嵌入的动态皮肤，从一个 FLA 文件的库中使用 MovieClip 元件，该 FLA 文件中帧标签的定义与其状态相关。当你为 FLA 库中的元件指定链接属性时，元件随后在类声明中被嵌入进去。你然后通过引用类将动态皮肤添加到显示列表中，动作效果由基于组件状态阶段的前向帧控制。

若要生成动态皮肤，首先在 Flash IDE 中添加一个新的 MovieClip 元件，然后添加许多命名了的帧以及帧与帧之间沿时间线的动作，如下图 9-3.

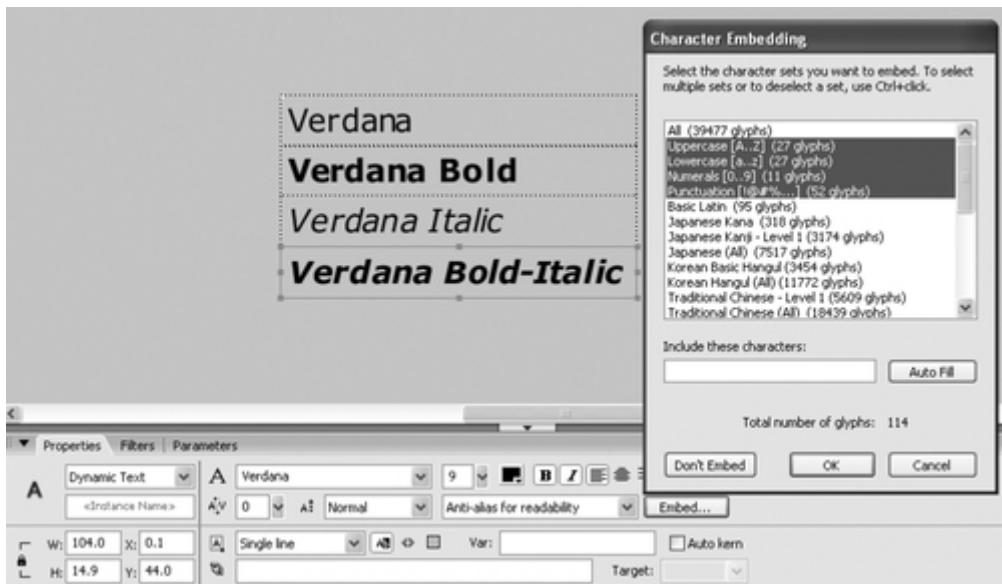
图 9-3. 创建一些命名的帧及其间的动作。



只要你喜欢，你的动作当然可以比本例中所含的动作复杂得多。重要的是记住帧标签名是逻辑的，与五个主要的按钮状态相关：弹起、按下、选中、经过和失效。在图 9-3 的 FLA 中，每个状态都添加两个帧标签：一个表示当前状态动作的开始，一个则表示结束（例如 OverBegin 和 OverEnd）。响应状态阶段的自定义类会利用那些标签预定相应帧。下一步就是从 Flash 中导出元件，以便 Flex 编译器能够使用它。

为允许访问 SWF 文件中元件的帧，基类属性定义为 flash.display.MovieClip 类的一个实例且指定了类名。观察下图 9-4，先前创建的多帧的 MovieClip 元件被赋予类名 ButtonClip 并以此导出。

图 9-4. 利用唯一的类名导出剪辑。



要从生成的 SWF 文件中导入元件，需在类声明阶段使用 [Embed] 元数据标记。source 属性定义为 SWF 文件的位置，然后用导出的元件所赋的类名去定义 symbol 属性，如下：

```
package oreilly.cookbook
{
    import flash.display.MovieClip;
    [Embed(source="assets/styles/FLAExample.swf",
        symbol="ButtonClip")]
    public class ButtonClass extends MovieClip {}
}
```

该类会作为自定义元素添加到显示列表中，从列表中可利用帧标签浏览先前定义的两者。ButtonClass 实例不能直接添加到 Flex 程序的显示列表中，因为它是 Flash 组件的一个子类并没用实现 mx.core.IFlexDisplayObject。若要将此类的实例添加到显示列表中，应将其变成 mx.core.UIComponent 的子类。

将 ButtonClass 实例添加到显示中的自定义 UIComponent 类同样会依据对 currentState 属性的更新而预先获取动作帧。下面例子是一个应用于 Button 实例的自定义皮肤：

查看代码：

```
package oreilly.cookbook
{
    import flash.display.MovieClip;
    import flash.events.Event;
    import mx.core.UIComponent;

    public class ButtonAnimatedSkin extends UIComponent
    {
```

```

private var _state:String;
private var _movieClipAsset:MovieClip;
public function ButtonAnimatedSkin() {
    super();
}
//当我们创建子类时，实际上实例化了我们的ButtonClass
override protected function createChildren():void {
    super.createChildren();
    _movieClipAsset = new ButtonClass();
    _movieClipAsset.addEventListener(Event.ENTER_FRAME,
        enterFrameHandler);
    addChild(_movieClipAsset);
}
//我们需要确认有一个固态背景 to hit detection against.
// 即使我们看不见，那也是必要的。
override protected function
updateDisplayList( unscaledWidth:Number,
    unscaledHeight:Number ):void
{
    graphics.clear();
    graphics.beginFill( 0x000000, 0 );
    graphics.drawRect( 0, 0, _movieClipAsset.width,
        _movieClipAsset.height );
    graphics.endFill();
    //确保剪辑总是放置在0,0像素
    _movieClipAsset.x = _movieClipAsset.width/2;
    _movieClipAsset.y = _movieClipAsset.height/2;
}
//按钮状态发生变化时会调用该方法。
override public function set currentState(value:String
) :void
{
    _state = value;
    //有一些值我们现在还没有用到；但是如果我们希望，
    //我们当然应该创建不能的动画和剪辑。
    if(_state == "selectedUp")      _state = "selected";
    if(_state == "selectedDown")     _state = "down";
    if(_state == "selectedOver")    _state = "over";
    switch( _state )
    {
        case "over" :
            //对于每个状态我们仅简单地 gotoAndPlay
            //相应状态的正确帧
            _movieClipAsset.gotoAndPlay("OverBegin");
    }
}

```

```

        break;
    case "down":
        _movieClipAsset.gotoAndPlay("DownBegin");
        break;
    case "selected":
        _movieClipAsset.gotoAndPlay("SelectedBegin");
        break;
    case "up" :
        _movieClipAsset.gotoAndPlay("EnabledBegin");
        break;
    case "disabled" :
        _movieClipAsset.gotoAndPlay("DisabledBegin");
        break;
    }
}

//我们每次输入一个帧时， 我们希望判断是否处在每个状态时
//我们某个动作的结尾，如果是在结尾，我们要将剪辑调回 //生
命周期的开始。
private function enterFrameHandler(event:Event):void {
    switch(_state){
        case "over" :
            if(_movieClipAsset.currentLabel == "OverEnd") {
                _movieClipAsset.gotoAndPlay("OverBegin");
            }
            break;
        case "down":
            if(_movieClipAsset.currentLabel == "DownEnd") {
                _movieClipAsset.gotoAndPlay("DownBegin");
            }
            break;
        case "up" :
            if(_movieClipAsset.currentLabel == "EnabledEnd")
{
                _movieClipAsset.gotoAndPlay("EnabledBegin");
}
            break;
        case "selected":
            if(_movieClipAsset.currentLabel == "SelectedEnd")
{
                _movieClipAsset.gotoAndPlay("SelectedBegin");
}
            break;
        case "disabled" :
            // 由于disabled 状态只有一帧， 不需要做任何事情
    }
}

```

```
break;  
    }  
}  
}  
}  
}
```

在重载 `createChildren` 方法过程中，实例化了一个 `ButtonClass` 新实例并添加到显示列表中。每次在 `Button` 中更新当前状态时，`ButtonAnimatedSkin` 类用一个新的状态通知 `ButtonClass` 获取指定的帧。在 `enterFrameHandler` 方法中生成一个循环来判断按钮是否在动画的结尾。如果动画已经到了结束帧，影片将回到状态动画的开始。

正如通过在皮肤样式属性中定义类以应用编程实现的皮肤，同样可以这样应用 `ButtonAnimatedSkin`：

```
<mx:Button toggle="true" skin="oreilly.cookbook.ButtonAnimatedSkin"  
y="300" x="300"/>
```

#### 9.17.4. 另外查看 [章节 9.13](#), [章节 9.14](#), 和 [章节 9.15](#).

## 9.18 节. 自定义引导界面

### 9.18.1. 问题

你想自定义在下载和初始化 Flex 程序时显示的引导界面。

### 9.18.2. 解决办法

通过生成 `mx.preloaders.DownloadProgressBar` 类（默认的程序引导界面）的子类或者生成实现了 `mx.preloaders.IPreloaderDisplay` 接口的 `flash.display.Sprite` 的子类型两种方法创建自定义的引导界面。

### 9.18.3. 讨论

一个 Flex 程序默认地由两个画面组成。第一个生成引导界面，该界面发出与程序的加载和初始化有关的一系列事件。默认的进度条依据这些事件更新其显示。下载接近完成时，系统管理器向第二个画面发出请求，同时继续生成和初始化程序。当程序快要完成初始化时，系统管理器将得到通知并去除引导界面。

这一过程由系统管理器进行内在处理，管理器初始化 `mx.preloaders.Preloader` 类的一个实例监控你的程序下载和初始化状态。`Preloader` 实例实例化了指定的下载进度条，将其添加

到显示列表并将 IPreloaderDisplay 实现的 preloader 属性设为本身。定义的 Preloader 实例能监听预加载器发出的许多事件，详情见 [表 9-2](#)。

表 9-2. Preloader 类发出的事件	
事件	描述
ProgressEvent.PROGRESS	在 SWF 文件正在下载时监听。
Event.COMPLETE	当 SWF 文件下载完成时监听。
FlexEvent.INIT_PROGRESS	程序正在初始化时监听。
FlexEvent.INIT_COMPLETE	程序初始化完成时监听。IPreloaderDisplay 实例应当监听本事件以相应地发出一个 COMPLETE 事件。COMPLETE 事件由已生成的 Preloader 实例监听以通知系统管理器程序已经准备好显示。
RslEvent.RSL_ERROR	运行时共享库(RSL)加载失败时监听。
RslEvent.RSL_PROGRESS	正在加载 RSL 时监听。
RslEvent.RSL_COMPLETE	RSL 加载成功完成时监听。

要想创建一个自定义进度条以处理这些事件，你可以生成 mx.preloaders.DownloadProgressBar 的子类或者生成实现了 mx.preloaders.IPreloaderDisplay 接口的 mx.display.Sprite 的子类，Sprite 本身就是 DownloadProgressBar 组成部分。

DownloadProgressBar 类为[表 9-2](#) 中列出的事件定义了保护型的事件处理器。你可以通过生成 DownloadProgressBar 子类重载这些事件处理器以相应地修改和更新你的自定义显示：

查看代码：

```
package oreilly.cookbook

{
    import flash.events.Event;
    import flash.events.ProgressEvent;
    import flash.geom.Rectangle;
    import mx.preloaders.DownloadProgressBar;

    public class DPBSubclass extends DownloadProgressBar
    {
        public function DPBSubclass()
        {
            super();
            // 下载完成后最短的显示时间
            MINIMUM_DISPLAY_TIME = 3000;
            // 设置初始化进程中的默认文本。
            initializingLabel =
                "Download complete!\nInitializing...";
        }
    }
}
```

```

    }

    // 重载标签域以显示自定义文本
    override protected function get labelRect():Rectangle
    {
        return new Rectangle(14, 5, 150, 30);
    }

    // 重载下载进度处理器以显示自定义文本。
    override protected function progressHandler(
        event:ProgressEvent ):void
    {
        super.progressHandler(event);
        label = Math.round(event.bytesLoaded / 1000 ).toString()
            + "k of  " + Math.round( event.bytesTotal / 1000
            ).toString() + "k";
    }

    // 重载以确保在初始化和下载时显示进度条。
    override protected function
        showDisplayForInit(elapsedTime:int,
            count:int):Boolean
    {
        return true;
    }

    override protected function
        showDisplayForDownloading( elapsedTime:int,
            event:ProgressEvent):Boolean
    { return true; }
}
}

```

本例重载 DownloadProgressBar 定义的 progressHandler 事件处理器显示自定义文本以响应引导器所发出 PROGRESS 事件。在重载只读的 labelRect 同时修改标签的显示域。显示文本布告的标签由父类内在地生成。若想将 DPBSubclass 类设置为自定义引导界面，利用完整合法的类名设置<mx:Application>标记的 preloader 属性，代码如下：

```

<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    preloader="oreilly.cookbook.DPBSubclass">
    <mx:Script>
        <! [CDATA[
            // embed large audio file in order to see

```

```

        // the preloader display.
        [Embed(source="assets/audio/audio.mp3")]
        private var _audio:Class;
    ]]>
</mx:Script>
</mx:Application>

```

当修改继承来的显示元素时，如果你想对事件处理有更细微的控制，你可以在 DownloadProgressBar 的子类中修改公共的 preloader 设置方法以定义你自己的的事件处理器。但如果你想对自定义引导界面的视觉组成有更多控制，你可以生成实现了 IPreloaderDisplay 接口的 Sprite 子类。

当生成 IPreloaderDisplay 的实例时，你应当定义实现各种属性和方法。例如 backgroundColor 和舞台尺寸等与视觉外貌有关的属性值由 Preloader 实例在自定义引导界面实例化前赋值。Preloader 实例随后调用 initialize 方法。 IPreloaderDisplay 实例的 preloader 属性是实现时应当赋予事件处理器的 Preloader 实例。

下列代码片段实现 IPreloaderDisplay 以生成自定义下载进度条：

查看代码：

```

package oreilly.cookbook
{
    import flash.display.Shape;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.events.ProgressEvent;
    import flash.events.TimerEvent;
    import flash.text.TextField;
    import flash.text.TextFormat;
    import flash.utils.Timer;

    import mx.events.FlexEvent;
    import mx.preloaders.IPreloaderDisplay;
    import mx.preloaders.Preloader;

    public class CustomProgress extends Sprite implements
IPreloaderDisplay
    {
        private var _bgAlpha:Number;
        private var _bgColor:uint;
        private var _bgImage:Object;
        private var _bgSize:String;
        private var _stageHeight:Number;
        private var _stageWidth:Number;
        private var _preloader:Preloader;
    }
}

```

```

private var _downloadBar:Shape;
private var _initBar:Shape;
private var _initField:TextField;

public function CustomProgress()
{
    _initField = new TextField();
    _initField.defaultTextFormat =
        new TextFormat( 'Arial', 12, 0xFFFFFF, true );
    _downloadBar = new Shape();
    addChild( _downloadBar );
    _initBar = new Shape();
    addChild( _initBar );
}
// 初始化待显示的任何属性。
public function initialize():void
{
    _downloadBar.x = ( _stageWidth / 2 ) - 20;
    _initBar.x = _downloadBar.x - 2;
    _downloadBar.y = ( _stageHeight / 2 ) - 50;
    _initBar.y = _downloadBar.y;
    _initField.x = _initBar.x + 2;
    _initField.y = _initBar.y + 100 - 15;
}
// 为Preloader实例定义事件处理器。
public function set preloader( obj:Sprite ):void
{
    _preloader = obj as Preloader;
    _preloader.addEventListener( ProgressEvent.PROGRESS,
                                downloadProgressHandler );
    _preloader.addEventListener( FlexEvent.INIT_PROGRESS,
                                initProgressHandler );
    _preloader.addEventListener( FlexEvent.INIT_COMPLETE,
                                initCompleteHandler );
}
public function get backgroundAlpha():Number
{
    return _bgAlpha;
}
public function set backgroundAlpha(value:Number):void
{
    _bgAlpha = value;
}
public function get backgroundColor():uint

```

```
{
    return _bgColor;
}
public function set backgroundColor(value:uint):void
{
    _bgColor = value;
}
public function get backgroundImage():Object
{
    return _bgImage;
}
public function set backgroundImage(value:Object):void
{
    _bgImage = value;
}
public function get backgroundSize():String
{
    return _bgSize;
}
public function set backgroundSize(value:String):void
{
    _bgSize = value;
}
public function get stageHeight():Number
{
    return _stageHeight;
}
public function set stageHeight(value:Number):void
{
    _stageHeight = value;
}
public function get stageWidth():Number
{
    return _stageWidth;
}
public function set stageWidth(value:Number):void
{
    _stageWidth = value;
}
// 处理 SWF 文件下载过程。
private function downloadProgressHandler( evt:ProgressEvent
):void
{
    var perc:Number = ( ( evt.bytesLoaded / evt.bytesTotal )
```

```

        * 100 );
    var top:Number = 100 - perc;
    _downloadBar.graphics.clear();
    _downloadBar.graphics.beginFill( 0xFF0000, 1 );
    _downloadBar.graphics.moveTo( 0, 0 );
    _downloadBar.graphics.lineTo( 10, 0 );
    _downloadBar.graphics.lineTo( 10, perc * 0.9 );
    _downloadBar.graphics.lineTo( 0, perc * 0.9 );
    _downloadBar.graphics.lineTo( 0, 0 );
    _downloadBar.graphics.endFill();

    _initBar.graphics.clear();
    _initBar.graphics.beginFill( 0xFFFFFFF, 1 );
    _initBar.graphics.moveTo( 0, 100 );
    _initBar.graphics.lineTo( 2, 100 );
    _initBar.graphics.lineTo( 2, top );
    _initBar.graphics.lineTo( 0, top );
    _initBar.graphics.lineTo( 0, 100 );
    _initBar.graphics.endFill();
}

// 处理程序初始化过程。
private function initProgressHandler( evt:FlexEvent ):void
{
    _initField.text = "initializing...";
    addChild( _initField );
}

// 处理下载和初始化的完成。
private function initCompleteHandler( evt:FlexEvent ):void
{
    var timer:Timer = new Timer( 3000, 1 );
    timer.addEventListener( TimerEvent.TIMER_COMPLETE,
        notifyOfComplete );
    timer.start();
}

// 通报下载和初始化的完成。
private function notifyOfComplete( evt:TimerEvent ):void
{
    dispatchEvent( new Event( Event.COMPLETE ) );
}
}
}

```

为通知系统管理器所有操作已经完成并且可从显示列表中去除引导界面，自定义引导界面需要在从 Preloader 实例接收一个 INIT\_COMPLETE 事件后发出一个 COMPLETE 事件。同样地，如果你在 DownloadProgressBar 子类中重载公共 preloader 设置方法或者生成 IPreloaderDisplay 实例时，确保为 INIT\_COMPLETE 事件定义了处理器并相应地发出 COMPLETE 事件。前面的例子中， COMPLETE 事件由计时器完成后发出的目的在于使用户能稍微多观察下初始化界面。

正如你在本节前面所做得一样，你在<mx:Application> 的声明中将 CustomProgress 类赋值给 preloader 属性值。例如：

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    preloader="oreilly.cookbook.CustomProgress">
    <mx:Script>
        <![CDATA[
            // embed large audio file in order to see
            // the preloader display.
            [Embed(source="assets/audio/audio.mp3")]
            private var _audio:Class;

        ]]>
    </mx:Script>
</mx:Application>
```

你并不被限定于只使用绘图 API 来自定义下载进度条。你可以嵌入图像元素，例如一幅图像或者一个 SWF，同样用来修改程序引导界面的外观样貌。

# 第十章. 拖拽操作 (小河)

Flex 框架的拖拽能力允许用户可视化地从一个地方移动数据到另一个地方，这大大增强了富互联网应用程序的体验性。任何扩展了 `mx.core.UIComponent` 类的组件都支持拖拽。在一个拖拽操作中，有一个初始方 (initiator) 和一个接收方 (receiver)。任何一个 `UIComponent` 的实例都能接受由拖拽动作初始的释放操作。一些列表类的 Flex 组件，如 `List`、`Tree` 和 `DataGrid`，具有管理拖拽操作的内置支持。这实现了从一个地方移动数据到另一个地方和组件本身的过程的自动化。

一个拖拽操作始于一个鼠标动作。你通过点击鼠标选择一个组件或条目 (item)，然后保持鼠标按钮的按下状态，拖拽这个条目。在拖拽的同时，一幅被称为拖拽代理的图像显示出来，并随着鼠标移动，表明这个条目正在被拖拽。连同拖拽代理一起，一些内置的图标显示出来表明指针经过一个接受释放操作的组件。要使一个组件接受释放操作，你应该在组件上设置拖拽事件处理器。一个可释放组件被认为是一个释放目标，它可以检查拖拽源数据，以确定这些数据的格式是否被这个组件所接受。拖拽源数据能够从一个组件被复制或移动到另一个组件，甚至是同一个组件——这个组件既是拖拽初始器也是释放目标。

本章着重讲解 Flex 框架的拖拽能力，以及它们怎样丰富用户的体验。

## 10.1. 节 使用 DragManager 类

### 10.1.1. 问题

你想在程序中从一个地方移动数据到另一个地方。

### 10.1.2. 解决方法

使用 `mx.manager.DragManager` 类管理拖拽操作以及在释放目标上监听拖拽事件。

### 10.1.3. 讨论

`DragManager` 类用于管理在你的程序里执行的拖拽操作。当一个拖拽操作被初始化，拖拽源 (drag source) 通过静态方法 `doDrag` 增加到 `DragManager`。被称为释放目标 (drop target) 的组件注册事件监听器，从而监听由 `DragManager` 发出的事件。它们接受 `DragManager` 上可用的数据源。

数据源通过一个初始组件赋予 `DragManager`，它们能够被移动或复制。一个拖拽操作的缺省处理过程是从一个地方移动数据到另一个地方。但如有需要实现自己的复制过程，你可以使用 `DragManager` 手动添加拖拽支持。

下面的例子让你在一个 `Canvas` 容器里移动一个 `Box` 组件。

Code View:

```
<mx:Application>
```

```

xmlns:mx="http://www.adobe.com/2006/mxml"
layout="horizontal">

<mx:Script>
<! [CDATA[
    import mx.core.DragSource;
    import mx.core.IUIComponent;
    import mx.events.DragEvent;
    import mx.managers.DragManager;

    private static const FORMAT:String = "box";

    private function mouseDownHandler(evt:MouseEvent ):void
    {
        var initiator:IUIComponent = evt.currentTarget as
        IUIComponent;
        var dragSource:DragSource = new DragSource();
        dragSource.addData( initiator, FORMAT );

        DragManager.doDrag( initiator, dragSource, evt );
    }

    private function dragEnterHandler( evt:DragEvent ):void
    {
        if( evt.dragSource.hasFormat( FORMAT ) )
        {

            DragManager.acceptDragDrop( Canvas( evt.currentTarget ) );
        }
    }

    private function dropHandler( evt:DragEvent ):void
    {
        var box:Box = Box( evt.dragInitiator );
        box.x = evt.localX;
        box.y = evt.localY;
    }

] ]>
</mx:Script>

<mx:Canvas id="canvas"
    backgroundColor="0xEEEEEE"
    width="300" height="300"
    dragEnter="dragEnterHandler(event);"
    dragDrop="dropHandler(event);">

```

```

<mx:Box id="dragItem"
    width="20" height="20"
    backgroundColor="#FFCC00"
    mouseDown="mouseDownHandler(event);"
/>
</mx:Canvas>

</mx:Application>

```

当<mx:Box>实例发出 mouseDown 事件时，mouseDownHandler 方法被调用并且拖拽源数据被增加到 DragManager。DragManager. doDrag 方法初始一个拖拽操作，需要至少三个参数：拖拽初始条目的引用，一个 mx. core. DragSource 对象和调用事件处理器的 flash. events. MouseEvent 对象。该方法还包括与释放操作相关的鼠标信息。拖拽期间默认的被渲染图像是一个具有透明度的矩形。你可以通过 doDrag 方法改变这幅图像（也被称为拖拽代理），但在这个例子里使用了默认的参数值。

Canvas 组件是由 Box 组件初始的拖拽动作的一个释放目标。它为 DragManager 发出的 dragEnter 和 dragDrop 事件指派事件处理器。DragManager 对象的静态方法 acceptDragDrop 检查 dragEventHandler 方法，拖拽源数据格式，doDrag 方法里的原始集合，以激活拖拽动作。acceptDragDrop 方法所需的参数是将要响应拖拽事件的释放目标 (drag target)。程序的 dropHandler 方法响应释放动作 (drop action) 并且处理这个移动的初始器 (在这个例子里是 Box 组件) 的位置，其位置根据鼠标释放时指针的位置确定。

前面的例子从一个地方移动数据到另一个地方使用了缺省设置，但你也可以很容易地建立处理过程来复制那些数据。接下来的例子复制了一个可拖拽的 Box 组件的信息给一个拖拽源 (DragSource) 对象，这个对象用来创建一个新的 Box 实例，然后将这个 Box 实例增加到释放目标 (drop target) 的显示清单中。

Code View:

```

<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal">

<mx:Script>
<! [CDATA[
    import mx.core.DragSource;
    import mx.events.DragEvent;
    import mx.managers.DragManager;

    private static const FORMAT:String = "box";

    private function mouseDownHandler(evt:MouseEvent):void
    {
        var initiator:Box = evt.currentTarget as Box;
        var boxData:Object = new Object();

```

```

        boxData.width = initiator.width;
        boxData.height= initiator.height;
        boxData.backgroundColor =
            initiator.getStyle( "backgroundColor" );
        var dragSource:DragSource = new DragSource();
        dragSource.addData( boxData, FORMAT );

        DragManager.doDrag( initiator, dragSource, evt );
    }

    private function dragEnterHandler( evt:DragEvent ):void
    {
        if( evt.dragSource.hasFormat( FORMAT ) )
        {

            DragManager.acceptDragDrop(Canvas(evt.currentTarget));
        }
    }

    private function dropHandler( evt:DragEvent ):void
    {
        var boxData:Object =
            evt.dragSource.dataForFormat( FORMAT );
        var box:Box = new Box();
        box.width = boxData.width;
        box.height = boxData.height;
        box.setStyle( "backgroundColor",
            boxData.backgroundColor );
        box.x = evt.localX;
        box.y = evt.localY;
        canvas.addChild( box );
    }

}

] ]>
</mx:Script>

<mx:Canvas id="canvas"
    backgroundColor="0xEEEEEE"
    width="300" height="300"
    dragEnter="dragEnterHandler(event);"
    dragDrop="dropHandler(event);">
    <mx:Box id="dragItem"
        width="20" height="20"
        backgroundColor="0x00FFCC"
        mouseDown="mouseDownHandler(event);"
    />

```

```
</mx:Canvas>  
  
</mx:Application>
```

在 mouseDownHandler 方法里，创建一个自定义的通用 object，它的属性与初始的 Box 组件相关。这个 object 被增加到 DragSource 对象并且在程序的拖拽事件处理器内通过 DragSource. dataForFormat 方法访问。在 dropHandler 方法内，一个新的 Box 组件被初始，它的属性由拖拽操作确定并且被添加到 Canvas 容器的显示清单中。

## 10.2 节 指定一个拖拽代理

### 10.2.1 问题

你希望在拖拽开始时自定义一幅图像来表现这个被拖拽的对象。

### 10.2.2 解决办法

为 DragManager. doDrag 方法的可选参数 dragImage 指定一幅自定义图像。

### 10.2.3. 讨论

默认情况下，在拖拽操作中使用的图像是一个包含透明度的矩形。这个在拖操作开始时被渲染的显示对象被称为拖拽代理。通过给 dragImage 参数传递一个 IFlexDisplayObject 实例，你可以改变这幅图像。Flex 框架提供的绝大部分组件都能用作拖拽代理，因为它们都扩展了 mx. core. UIComponent 类，而该类实现了 IFlexDisplayObject 接口。虽然添加一个拖拽代理是表现物体正被移动到何处的一个简单方法，但这样做更多是为了防止不必要的误拖拽。BitmapAsset 类同样实现了 IFlexDisplayObject 接口，并且它能方便地抓取应用程序中正在移动的可视对象的位图数据。

在本示例中，我们为一个拖拽操作指定一个 BitmapAsset 实例作为拖拽代理：

Code View:

```
<mx:Application  
    xmlns:mx="http://www.adobe.com/2006/mxml"  
    layout="horizontal">  
  
<mx:Script>  
    <! [CDATA[  
        import mx.core.BitmapAsset;  
        import mx.core.DragSource;  
        import mx.events.DragEvent;  
        import mx.managers.DragManager;
```

```

private var xoffset:Number;
private var yoffset:Number;
private static const FORMAT:String = "box";

private function mouseDownHandler(evt:MouseEvent ):void
{
    xoffset = evt.localX;
    yoffset = evt.localY;
    var initiator:Box = evt.currentTarget as Box;
    var proxyBox:BitmapAsset = new BitmapAsset();
    proxyBox.bitmapData =
        New BitmapData(initiator.width,
                        initiator.height );
    proxyBox.bitmapData.draw( initiator );
    var dragSource:DragSource = new DragSource();
    dragSource.addData( initiator, FORMAT );

    DragManager.doDrag( initiator, dragSource, evt,
                        proxyBox, 0, 0, 0.5 );
}

private function dragEnterHandler( evt:DragEvent ):void
{
    if( evt.dragSource.hasFormat( FORMAT ) )
    {

        DragManager.acceptDragDrop(Canvas(evt.currentTarget));
    }
}

private function dropHandler( evt:DragEvent ):void
{
    var box:Box = Box( evt.dragInitiator );
    box.x = evt.localX - xoffset;
    box.y = evt.localY - yoffset;
}

] ]>
</mx:Script>

<mx:Canvas id="canvas"
            backgroundColor="#FFFFFF"
            width="300" height="300"
            dragEnter="dragEnterHandler(event);"
            dragDrop="dropHandler(event);">

```

```
<mx:Box id="dragItem"
    width="20" height="20"
    backgroundColor="0x00FFCC"
    mouseDown="mouseDownHandler(event);"
/>
</mx:Canvas>

</mx:Application>
```

当拖操作开始 mouseDownHandler 事件处理器被触发时，一幅代表拖动发起者的位图数据被复制到一个 BitmapAsset 实例中。该 BitmapAsset 实例被用作拖拽代理，在拖动过程中被渲染成一个 50%透明的可视对象。通过在拖操作开始时记录鼠标按下的本地座标， Box 实例被拖放到具有相同偏移量的本地座标上。

你不但只可以设定一幅位图作为拖拽代理，还能指定一幅嵌入的自定义图像来表现拖拽过程中的正在被拖动的对象。

#### 10.2.4. 参考

### 10.3 节 在 List 内部进行拖拽操作

#### 10.3.1 问题

你希望在同一个列表类的组件实例内移动和复制数据。

#### 10.3.2 解决办法

使用列表类组件内建的拖拽管理器。

#### 10.3.3 讨论

通过为来自于 DragManager 的事件指派事件处理器，你可以激活任何组件的拖拽能力。尽管你可以手动添加这种支持到列表控件 (list controls)，但 Flex 框架的列表类组件，例如 list, Tree 和 DataGrid，具有管理拖拽操作的内建支持。列表类的控件（即扩展了 mx.controls.listClasses.ListBase 类的控件）具有公共方法和属性去激活这些操作和管理数据，以及在内部处理来自 DragManager 的事件。

下面的例子展示了一个 list 组件，通过设置 dragEnabled 和 dropEnabled 属性，它的条目可以在其集合中移动。

Code View:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal"
    creationComplete="creationHandler();">

    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;
            private function creationHandler():void
            {
                var collection:ArrayCollection =
                    new ArrayCollection( ['Josh', 'Todd', 'Abey'] );
                contactList.dataProvider = collection;
            }
        ]]>
    </mx:Script>

    <mx:Panel title="Contact List:" width="200" height="200">
        <mx>List id="contactList" width="100%" height="100%" dragEnabled="true" dropEnabled="true" dragMoveEnabled="true" />
    </mx:Panel>
</mx:Application>
```

例子里的 List 控件的内置 dragEnabled, dropEnabled 和 dragMoveEnabled 属性都被设置为 true。本质上，dragEnabled 和 dropEnabled 属性允许该组件响应来自 DragManager 类的事件。dragMoveEnabled 属性是条目是否能够在 list 内移动或复制的标志。缺省地，该值为 false，允许增加拖拽源数据到 list 里。通过设置 dragMoveEnabled 属性为 true，当拖拽事件完成时，list 内的条目将会从它先前的位置被移动到鼠标指针所指向的位置。

设置或保留 dragMoveEnabled 属性的缺省值为 false，将允许复制数据。本例中，在 list 中执行拖拽操作选择条目时，你会看到一些奇妙的事情。这是因为这些类来自于操作 API 的集合，它们操作数据，使用唯一标识符（UID）来标识集合中的条目。

当一个条目被增加到列表类控件的 dataProvider，私有的方法 ListBase.copyItemWithUID 被调用并给该拖拽的条目指派一个唯一的标识符。然而，当数据源是一个简单的字符串对象列表时，如本例使用的源于 Array 的 ArrayCollection，这些条目不会在内部被赋予一个新的 ID。所以，在这个例子中当你在 list 控件中执行选择操作，你会看到此操作通过 UID 在

最大索引处选择条目。换言之，如果你首次从一个列表的第五个索引处拖拽复制条目，那么无论何时你再次选择这个条目，列表都会将第五个索引当做正确的选择来渲染这个条目。

为了确保复制的数据是唯一的并且增加到集合中有一个 UID，你可以为 drag-and-drop 操作增加事件处理，以及在需要的时候复制数据。接下来的例子修改了先前的例子，为 lists 控件发出的 dragComplete 事件指派了一个事件处理器。

Code View:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal"
    creationComplete="creationHandler();">

    <mx:Script>
        <![CDATA[
            import mx.utils.ObjectUtil;
            import mx.events.DragEvent;
            import mx.collections.ArrayCollection;

            private function creationHandler():void
            {
                var collection:ArrayCollection =
                    new ArrayCollection( ['Josh', 'Todd', 'Abey'] );
                contactList.dataProvider = collection;
            }

            private function dropHandler( evt:DragEvent ):void
            {
                var listItem:Object =
                    evt.dragSource.dataForFormat( "items" );
                var index:int = contactList.calculateDropIndex(evt);
                ArrayCollection( contactList.dataProvider )
                    .setItemAt( ObjectUtil.copy( listItem ), index );
            }
        ]]>
    </mx:Script>

    <mx:Panel title="Contact List:" width="200" height="200">
        <mx>List id="contactList"
            width="100%" height="100%"
            dragEnabled="true"List>
    </mx:Panel>

```

```
        dropEnabled="true"
        dragMoveEnabled="false"
        dragComplete="dropHandler(event);"
    />
</mx:Panel>

</mx:Application>
```

设置 `dragMoveEnabled` 属性为 `false` 来激活复制操作，这个 `List` 实例可以接受一个 `drag-and-drop` 操作产生的新条目。内置的属性 `dragComplete` 注册了 `dropHandler` 方法，作为一个拖拽事件完成的事件处理器。在这个事件处理器中，当前被拖拽的条目通过使用条目格式从源数据获取，这些格式在拖拽操作初始时确定。`List` 的基本索引由 `List.calculateDropIndex` 方法获得，并且用于更新增加到 `collection` 内部的条目。一个条目的深复制被创建并且通过调用 `ObjectUtil` 的 `copy` 方法指派它一个新的唯一标识符，更新 `collection` 里的任何元素都将会触发 `List` 实例里的数据绑定，这些改变会立刻反映在显示里。

## 10.4 节 在 `List` 之间进行拖拽操作

### 10.4.1 问题

你希望从一个 `List` 中拖拽数据到另一个 `List`。

### 10.4.2 解决办法

使用 `List` 内建的拖拽管理器并把每个组件设置成允许接受拖拽操作。

### 10.4.3 讨论

Flex 列表类控件内建的拖拽管理器，去掉了手动操作 `DragManager` 对象的必要，从而使数据从一个 `List` 移动到另一个 `List` 变得相对简单。通过设置 `dragEnabled` 和 `dropEnabled` 属性为 `true`，你就能打开移动和复制拖拽源数据的内置操作。

当需要在两个 `List` 之间进行双向拖拽的时候，这两个 `List` 都必须设置成允许接受拖拽操作。当拖拽发起者和拖放目标具有相同基类的时候，例如两个 `List`，拖拽源数据的结构对成功完成拖拽操作来说并不重要，因为这两个控件使用相同的条目渲染器来渲染数据。当需要在不同类型的列表类控件之间实现单向或双向拖拽操作的时候，例如一个 `List` 和一个 `DataGrid`，你需要考虑拖拽源数据的结构，因为这两个控件使用不同的条目渲染器，呈现的方式不一样。

在本示例中，我们允许用户从一个 `List` 移动条目到一个 `DataGrid`，在 `DataGrid` 中显示条目的完整信息。

Code View:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal"
    creationComplete="creationHandler();">

<mx:Script>
<! [CDATA[
    import mx.collections.ArrayCollection;

    private function creationHandler():void
    {
        contactList.dataProvider = new ArrayCollection([
            {label:'Josh Noble', phone:'555.111.2222'},
            {label:'Todd Anderson', phone:'555.333.4444'},
            {label:'Abey George', phone:'555.777.8888'}
        ]);
    }
]]>
</mx:Script>

<mx:Panel title="Contact List:"
    width="200" height="200">
    <mx>List id="contactList"
        width="100%" height="100%"
        dragEnabled="true"
        dropEnabled="true"
        dragMoveEnabled="false"
    />
</mx:Panel>
<mx:Panel title="Contact Info:"
    width="300" height="200">
    <mx>DataGrid id="contactGrid"
        width="100%" height="100%"
        dragEnabled="true"
        dropEnabled="true"
        dragMoveEnabled="true">
        <mx:columns>
            <mx:DataGridColumn dataField="label"
                headerText="Name"/>
            <mx:DataGridColumn dataField="phone"
                headerText="Phone"/>
        </mx:columns>
    </mx>DataGrid>

```

```
</mx:Panel>  
  
</mx:Application>
```

一些包含 label 属性和 phone 属性的对象被添加到 ArrayCollection 对象中。当把该 ArrayCollection 赋给 List 实例的 dataProvider 属性后，List 上会显示每个 label 属性。从 List 中拖动条目到 DataGrid 中的时候，数据被复制了，拖拽源对象的 label 属性和 phone 属性则会显示在 DataGrid 中。

由于这两个列表类控件的 dropEnabled 属性都被设置为 true，所以该例子示范了一个双向拖拽的系统。

#### 10.4.4. 节 参考

### 10.5 节. 启动和禁止拖操作

#### 10.5.1. 问题

你想在运行时启动和禁止列表类控件的拖拽操作。

#### 10.5.2. 解决办法

使用列表类控件的拖拽事件属性来管理属性值。

#### 10.5.3 讨论

Flex 框架的列表类控件具有内置的管理器来与 DragManager 互动，通过使用 dragEnabled 和 dropEnabled 属性，提供了一个便捷的途径来启动控件响应拖拽动作。通过使用内建的事件属性 dragStart, dragEnter, dragOver, dragExit, dragDrop 和 dragComplete，你可以像在任何其它 UIComponent 驱动的程序中一样设置事件处理器。

为了启动一个列表类控件接收拖拽动作，你可以设置该控件实例的 dragEnabled 属性的布尔值。列表类控件具有单向或者双向的拖拽能力。在一个单向的体系中，该控件实例能够接受释放动作或者允许拽动。在双向体系中，该控件允许拽动和释放动作都执行。

通过指派 DragManager 发出的拖拽事件的处理器，从而对程序中怎样处理事件和拖拽源数据进行管理。下面的例子为 dragEnter 事件创建一个事件处理器，从而决定在两个 List 控件中是否允许或禁止拖拽动作。

Code View:

```

<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal"
    creationComplete="creationHandler();">

<mx:Script>
<! [CDATA[
    import mx.events.DragEvent;
    import mx.collections.ArrayCollection;

    [Bindable]
    public var isEnabled:Boolean = true;
    private static const DIS_LABEL:String =
        "disable drag and drop";
    private static const EN_LABEL:String =
        "enable drag and drop";

    private function creationHandler():void
    {
        list1.dataProvider = new ArrayCollection([
            'Spider Monkey', 'Orangutan', 'Gorilla'
        ]);
        list2.dataProvider = new ArrayCollection([
            'Lion', 'Cheetah', 'Puma'
        ])
    }

    private function clickHandler():void
    {
        enableBtn.label = ( enableBtn.label == DIS_LABEL )
            ? EN_LABEL
            : DIS_LABEL;
        isEnabled = !isEnabled;
    }

    private function dragEnterHandler( evt:DragEvent ):void
    {
        evt.target.dropEnabled = ( evt.target !=
            evt.dragInitiator );
    }
]]>
</mx:Script>

```

```

<mx:VBox width="100%" height="100%">
    <mx:Button id="enableBtn"
        label="disable drag and drop"
        click="clickHandler();"
    />
    <mx:HBox width="100%" height="100%">
        <mx>List id="list1"
            width="200" height="200"
            dragEnabled="{isEnabled}"
            dragMoveEnabled="true"
            dragEnter="dragEnterHandler(event);"
        />
        <mx>List id="list2"
            width="200" height="200"
            dragEnabled="{isEnabled}"
            dragMoveEnabled="true"
            dragEnter="dragEnterHandler(event);"
        />
    </mx:HBox>
</mx:VBox>

</mx:Application>

```

每一个 list 是否允许拖拽动作取决于本地类绑定的 isEnabled 属性， isEnabled 由按钮的点击事件处理器更新。如果这些列表控件允许拖拽动作，当鼠标指针进入该组件范围并且选择其中一个条目时， dragEnter 事件被发出。

通过在一个 list 里设置 dragMoveEnabled 为 true，拖拽完成时你执行的是一个移动的动作。默认地， dragMoveProperty 属性为 false，这意味着当一个拖拽操作完成，拖拽源数据对象将会从拖拽初始器被复制到释放目标。

为了确保数据不会在一个 List 实例中移动，每一个组件的 dropEnabled 属性值紧跟着 dragEnter 事件后更新。dragEnterHandler 事件处理器检查释放目标是否跟拽动初始器相同并且更新 dropEnabled 属性值。要禁止一个初始化了拽动的组件执行释放操作， dropEnabled 属性应该设置为 false。

## 10.6 节 自定义列表类控件的拖动图像

### 10.6.1 问题

你希望在一个自定义列表类控件中进行拖拽操作时，自定义它的拖动图像。

## 10.6.2 解决办法

创建一个需要在拖拽操作时显示的 UIComponent，并覆盖自定义列表类控件的 dragImage getter 方法。

## 10.6.3 讨论

Flex 框架中的列表类控件天生就会处理拖拽。这意味着你不需要为 DragManager 发出的事件设置事件监听器。尽管如此，由于你不直接跟 DragManager 打交道，所以当一个列表类控件中进行拖拽操作的时候，拖动图像默认是一个半透明的 item renderer。你不能把拖动图像当作列表类控件的一个属性来设置，像在 DragManager 的 doDrag 方法中那样手动设置它。

要为一个列表类控件自定义拖动图像，你需要创建一个继承自该列表类控件的类，然后覆盖其受保护的 dragImage getter 方法。覆盖该方法后，你可以返回你的自定义拖动图像。

下面的代码演示了如何创建一个继承自 UIComponent 的拖动图像类，覆盖其受保护的 createChildren 方法，在该方法中添加图像作为拖动时要显示的图像：

Code View:

```
package oreilly.cookbook

{
    import flash.display.Bitmap;
    import flash.display.Loader;
    import flash.display.LoaderInfo;
    import flash.events.Event;
    import flash.net.URLRequest;

    import mx.controls.List;
    import mx.controls.listClasses.IListItemRenderer;
    import mx.core.UIComponent;

    public class CustomDragProxy extends UIComponent
    {
        public function CustomDragProxy()
        {
            super();
        }

        override protected function createChildren():void
        {
            super.createChildren();

            var list:List = List( owner );
            var items:Array = list.selectedIndices;
            items.sort();
        }
    }
}
```

```

for( var i:int = 0; i < items.length; i++ )
{
    var item:Object = list.dataProvider[items[i]];
    var loader:Loader = new Loader();

    loader.contentLoaderInfo.addEventListener(
        Event.COMPLETE,completeHandler );
    addChild( loader );

    loader.load( new URLRequest( item.image ) );

    var source:IListItemRenderer =
        list.indexToItemRenderer(items[i]);
    loader.x = source.x;
    loader.y = source.y - 20 + ( i * 45 );
}
}

private function completeHandler( evt:Event ):void
{
    var info:LoaderInfo = LoaderInfo( evt.target );
    var image:Bitmap = Bitmap( info.content );
    image.width = image.height = 40;
}
}
}

```

在对象内部，当其实例化的时候会调用受保护的 `createChildren` 方法。在本例所覆盖的 `createChildren` 方法中，你可以添加一些自定义对象到控件的显示列表中。我们假定把该控件用作拖动图像的控件为 `List` 控件，所以 `owner` 属性被转化为一个 `List` 的实例。通过转换 `owner` 属性，父 `List` 实例里面的属性将可被本控件访问，用于修改它的显示列表。

`Loader` 的实例数量依赖与列表类控件中被选择的条目数量，它们被添加到显示列表中，根据与其相关联的条目中的图像 URL 地址载入图像。图像载入后，一个 `item renderer` 的实例通过该 `List` 被访问了。使用 `indexToItemRenderer` 方法定位父 `Loader` 对象。

要为一个列表类控件自定义拖动图像，你需扩展这个控件并覆盖它的 `dragImage` `getter` 方法。下面的代码演示了如何在 `dragImage` `getter` 方法被调用时返回一个自定义的拖动图像：

```

package oreilly.cookbook
{
    import mx.controls.List;
}

```

```

import mx.core.IUIComponent;

public class CustomList extends List
{
    public var dragProxy:Class;

    public function CustomList()
    {
        super();
    }

    override protected function get dragImage():IUIComponent
    {
        if( dragProxy == null )
            return super.dragImage;

        var proxy:IUIComponent = new dragProxy();
        proxy.owner = this;
        return proxy
    }
}

```

上述代码中，当 dragImage getter 方法被调用，一个新的拖动图像被初始化并返回给调用者。如果你对之前的章节还有印象，你应该明白 CustomDragProxy 可以通过 owner 属性访问父列表的实例。在 CustomList 中，dragImage getter 方法被覆盖了，它返回 dragProxy 属性所声明的对象的一个实例。为 CustomList 的 dragProxy 属性设置一个拖拽代理的完整类名，可以让它在每次在 dragImage 属性被访问的时候实例化。

下面的代码在应用程序中添加这个 CustomList，并把它的 dragProxy 属性设为 CustomDragProxy：

Code View:

```

<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:flexcookbook="oreilly.cookbook.*"
    layout="horizontal"
    creationComplete="creationHandler();">

    <mx:Script>
        <! [CDATA[

            import mx.collections.ArrayCollection;

            private function creationHandler():void

```

```

    {
        contactList.dataProvider = new ArrayCollection([
            {label:'Josh', image:'assets/bigshakey.png'},
            {label:'Todd', image:'assets/smiley.png'}
        ]);
    }

] ]>
</mx:Script>

<mx:Panel title="Contact List:" width="200" height="200">
<flexcookbook:CustomList id="contactList" width="100%" height="100%" allowMultipleSelection="true" dragEnabled="true" dropEnabled="true" dragMoveEnabled="true" dragProxy="com.oreilly.flexcookbook.CustomDragProxy" />
</mx:Panel>

</mx:Application>

```

dataProvider 为自定义列表提供了一个数组，数组中每个元素包含一个 label 属性和一个 image 属性。列表的 dragProxy 属性值在行内声明为 CustomDragProxy 的完整类名。当一个拖动动作开始，一个 CustomDragProxy 实例将被创建，然后载入所选列表条目中 image 属性所指向的图片，并在拖拽操作时显示出来。

#### 10.6.4 参考

## 10.7 节. 自定义列表类控件的拽动指示器

### 10.7.1 问题

你想自定义拖拽指示器的图形，并在 list 控件的拖拽操作期间显示。

### 10.7.2 解决办法

创建一个自定义的可编程皮肤(programmatic skin)，并且为一个List控件设置dropIndicatorSkin样式属性。

### 10.7.3. 讨论

Flex 框架的列表类组件具有默认的可编程(programmatic)皮肤，在你执行拖拽操作时用来渲染指示器。当 ListBase.showDropFeedback 方法在一个 list 的内部被调用，一个指示器(indicator)类的实例被创建，在这个条目的上边或左边一个像素位置处被渲染(具体取决于条目在 list 控件里是如何组织的)。通过扩展 mx.skins.ProgrammaticSkin 类和设置一个组件的 dropIndicatorSkin 样式属性，你可以自定义拖拽指示器。

接下来的例子自定义一个拖拽指示器，它重写了 ProgrammaticSkin 类的 updateDisplayList 方法，使用 drawing API 来绘制一个基于当前方向属性的箭头图形。

Code View:

```
package oreilly.cookbook

{
    import mx.skins.ProgrammaticSkin;

    public class CustomDropIndicator extends ProgrammaticSkin
    {
        public var direction:String = "horizontal";

        public function CustomDropIndicator()
        {
            super();
        }

        override protected function updateDisplayList
            (unscaledWidth:Number, unscaledHeight:Number ):void
        {
            super.updateDisplayList(unscaledWidth, unscaledHeight );

            graphics.clear();
            graphics.beginFill( 0x000000 );

            if( direction == "horizontal" )
            {
                graphics.moveTo( 4, -10 );
                graphics.lineTo( 6, -10 );
                graphics.lineTo( 6, -4 );
                graphics.lineTo( 10, -4 );
                graphics.lineTo( 5, 0 );
                graphics.lineTo( 0, -4 );
                graphics.lineTo( 4, -4 );
            }
        }
    }
}
```

```
        graphics.lineTo( 4, -10 );
    }
else
{
    graphics.moveTo( 10, 4 );
    graphics.lineTo( 10, 6 );
    graphics.lineTo( 5, 6 );
    graphics.lineTo( 5, 10 );
    graphics.lineTo( 0, 5 );
    graphics.lineTo( 5, 0 );
    graphics.lineTo( 5, 4 );
    graphics.lineTo( 10, 4 );
}
graphics.endFill();
}
```

方向属性取决于父 List 控件怎样组织显示列表新增的 collection 条目。当方向设置为水平，一个向下的箭头在条目的上方显示。如果不为水平，则被当作垂直，显示一个向右的箭头。

下面的程序增加一个 List 控件和一个 TileList 控件到显示列表，并且设置每个组件的 dropIndicatorSkin 属性，从而自定义先前例子中创建的指示器。

### Code View:

```
<mx:Application

    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal"
    creationComplete="creationHandler();">

<mx:Script>
<! [CDATA[
    import mx.collections.ArrayCollection;

    private function creationHandler():void
    {
        contactList.dataProvider = new ArrayCollection([
            'Josh', 'Abey', 'Todd'
        ]);
    }
]]>
</mx:Script>
```

```
<mx>List id="contactList"
    width="200" height="200"
    allowMultipleSelection="true"
    dragEnabled="true"
    dropEnabled="true"

    dropIndicatorSkin="com.oreilly.flexcookbook.CustomDropIndicator"
    />
<mx:TileList id="tileList"
    width="180" height="200"
    dropEnabled="true

    dropIndicatorSkin="com.oreilly.flexcookbook.CustomDropIndicator"
    />

</mx:Application>
```

当你拖动一个来自于 List 组件的条目经过 TileList 时，可以看到一个箭头指向 item renderer 实例，它位于拖拽期间鼠标指针经过的索引处。当执行一个经过这个 List 组件的拖拽动作时，拖拽指示器是一个指向可见的拖拽源数据对象索引位置处的箭头。

dragIndicatorSkin 是一个列表类组件的样式属性，它可以像任何其它样式属性一样被设置，接受一个完全匹配的名为 programmatic skin 的类来渲染自定义的显示图像。

#### 10.7.4. 请看 [10.3](#) 和 [10.4](#).

# 第十一章. States (常青)

States 是一套用于创建状态组件的强大工具，也就是说组件可以有多个视图。需要此操作的组件可以是一个编辑器和显示器，一个有多个示屏的对话框，或一个有菜单视图和细节视图的组件。这些多个视图都被包含在一个组件中，被归类为一个 states。Flex Framework 定义了一个类叫 State，包含在 mx.state 包中，可让你在单个组件中定义特定视图的属性。所有 UIComponents 都允许添加一个或多个 mx.state.State 对象到他们的 states 数组中，这样你可以轻松添加和删除任何子组件，控制样式和进入或退出 State 时使用特效(Effects)和转换(Transitions)。使用 x.states.State 是最清晰最简单的方式在单个组件中实现多个状态或视图。

States 中可以添加子节点到组件中，当离开 state 时添加的子节点将被删除。你也可以定义 transitions，在组件的 currentState 发生改变时进行播放，或者在 state 属性发生改变时应用特效。组件的任何属性都意味着是临时的或只关联与特定的组件 state。

## 11.1节.设置 State 的样式和属性

### 11.1.1. 问题

我想为某个当前视图设置样式或属性以及在退出时移除该样式或属性。

### 11.1.2. 解决办法

当进入 state 时用 SetStyle 标记来改变任一样式并在其退出时自动回到先前样式。

### 11.1.3. 讨论

当进入一个 state 时都可通过 SetProperty 和 SetStyle 标签设置任意组件的样式和属性，设置 state 中的样式和属性就像是添加子节点一样，当回到初始 state 时这些都将被恢复。

SetStyle 和 SetProperty 标签需要一个目标属性确定在目标上设置样式或属性，根据名称和值进行设置：

```
<mx:SetProperty target="{this}" name="height" value="500"/>  
  
<mx:SetStyle target="{this}" name="backgroundColor"  
value="#ccccff"/>
```

下面的组件通过 currentState 属性设置为初始 state:

Code View:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300" currentState="initialState">
```

State 中的任何改变都是不能撤销的，包括初始 state。下面的例子代码演示如何改变组件当前 state，移除上一个 state 添加的内容，在新的 state 中定义新组件并设置为 currentState：

Code View:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300" currentState="initialState">
<mx:Script>
<![CDATA[

    private var i:int = 1;

    private function cycleStates():void
    {
        switch(i)
        {
            case 0:
                currentState = "initialState";
                break;
            case 1:
                currentState = "addImg";
                break;
            case 2:
                currentState = "changeHolderBG";
                break;
        }
        if(i == 2){i=0;}else{i++;}
    }
}

]]>
</mx:Script>
<mx:states>
<mx:State name="initialState"/>
<mx:State name="addImg">
<mx:SetProperty target="{this}" name="height"
value="500"/>
<mx:SetStyle target="{this}" name="backgroundColor"
value="#ccccff"/>
<mx:AddChild relativeTo="{mainHolder}">
<mx:Image source="../assets/image.jpg"/>
</mx:AddChild>

```

```

</mx:State>
<mx:State name="changeHolderBG">
    <mx:SetProperty target="{mainHolder}" name="height"
        value="500"/>
    <mx:SetStyle target="{this}" name="backgroundColor"
        value="#ffcccc"/>
    <mx:SetProperty target="{mainHolder}" name="alpha"
        value="0.5"/>
</mx:State>
</mx:states>
<mx:Button click="cycleStates()" label="change"/>
<mx:HBox id="mainHolder"/>

</mx:VBox>

```

## 11.2节. 为进入和离开 States 创建 Transitions

### 11.2.1. 问题

我想创建一个特效，当进入或退出 state 时进行播放

### 11.2.2. 解决办法

使用 Transitions 对象，设置其 fromState 和 toState 属性。

### 11.2.3. 讨论

一个 transition 就是一个特效或一系列特效。Transition 对象给出了 fromState 和 toState 属性定义何时播放。fromState 和 toState 属性及可以是特定的 states 或通配符(\*)。

有几个方法创建 Transition 对象，添加一个特效到组件，绑定 Transition 的 effect 属性到特效。下面的代码定义了一个 Transition，其 effect 属性绑定到 id 为 glow 的特效上。

Code View:

```

<mx:Transition id="thirdTrans" fromState="edit" toState="show"
    effect="{glow}"/>

```

就像 State 那样，一个 Transition 对象定义被执行的行为。该行为需要一个目标，但是这个目标并不是 Tansition 本身的一部分。你可以在 Transition 中添加 SetPropertyStyle 或

SetPropertyAction 来改变样式或当前组件的子组件属性如 height 或 width 等：

Code View:

```
<mx:Transition id="firstTrans" fromState="show" toState="edit">
    <mx:SetPropertyAction target="{holder}" name="alpha"
        value="0"/>

</mx:Transition>
```

使用 SetStyleAction 时 Transition 将设置目标组件的 style 属性：

Code View:

```
<mx:Transition id="secondTrans" fromState="*" toState="upload">
    <mx:SetStyleAction target="{holder}" name="backgroundColor"
        value="#ff0000"/>

</mx:Transition>
```

下面的代码创建三个 States，定义 Transition 对象在各个 states 之间切换：

Code View:

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="300" currentState="show">
    <mx:Script>
        <! [CDATA [
            [Bindable]
            private var _imgURL:String;

            [Bindable]
            private var _title:String;
        ]]>
    </mx:Script>
    <mx:Glow blurXTo="20" blurYTo="20" duration="1000"
        color="0xfffff00" id="glow" target="{holder}"/>
    <mx:Glow blurXTo="25" blurYTo="25" duration="1000"
        color="0xffffffff" id="fade" target="{holder}"/>
```

这里定义了多个 Transition 对象，设置了 fromState 和 toState 属性，指示何时播放：

Code View:

```

<mx:transitions>
    <mx:Transition id="firstTrans" fromState="show"
        toState="edit">
        <mx:SetPropertyAction target="{holder}" name="alpha"
            value="0"/>
    </mx:Transition>
    <mx:Transition id="secondTrans" fromState="*"
        toState="upload">
        <mx:SetStyleAction target="{holder}"
            name="backgroundColor" value="#ff0000"/>
    </mx:Transition>
    <mx:Transition id="thirdTrans" fromState="edit"
        toState="show" effect="{glow}"/>
    <mx:Transition id="fifthTrans" fromState="upload"
        toState="*" effect="{fade}"/>
</mx:transitions>
<mx:states>
    <mx:State/>
    <mx:State name="show">
        <mx:AddChild relativeTo="{holder}">
            <mx:HBox>
                <mx:Label text="{_title}"/>
                <mx:Image source="{_imgURL}"/>
            </mx:HBox>
        </mx:AddChild>
    </mx:State>
    <mx:State name="edit" exitState="_title = input.text;">
        <mx:AddChild relativeTo="{holder}">
            <mx:HBox>
                <mx:TextInput id="input" text="{_title}"/>
            </mx:HBox>
        </mx:AddChild>
    </mx:State>
    <mx:State name="upload">
        <mx:AddChild relativeTo="{holder}">
            <mx:HBox>
                <mx:Button label="start upload"/>
            </mx:HBox>
        </mx:AddChild>
    </mx:State>
</mx:states>
<mx:ComboBox dataProvider="[{ 'show',      'edit',      'upload' }]"
    change="{this.currentState      =      cb.selectedItem      as      String}"
    id="cb"/>

```

```
<mx:Canvas id="holder"/>  
</mx:HBox>
```

## 11.3节. 使用 AddChildAction 和 RemoveChildAction

### 11.3.1. 问题

我想在播放 transition 时能控制何时添加或删除子组件。

### 11.3.2. 解决办法

使用 AddChildAction 和 RemoveChildAction 标签分别控制子组件的添加和删除。

### 11.3.3. 讨论

AddChildAction 和 RemoveChildAction 对象执行方式和 SetPropertyAction 和 SetPropertyStyle 对象类似，它们包装了 State 对象的功能，可在 Transition 中使用，为了与 Transition 中的并行(Parallel)或序列(Sequence)对象进行交互。

默认情况下 State 中是通过 AddChild 对象添加子组件的。为了控制何时子组件被添加或在添加和删除前播放某个特效，我们可以在执行序列中使用 AddChildAction 代替 State 中的 AddChild 标签，例如下面的代码提供一个 Transition 对象的执行顺序：

Code View:

```
<mx:Transition fromState="view" toState="edit">  
    <mx:Sequence>  
        <mx:Fade alphaFrom="1" alphaTo="0" duration="1000"  
            target="{viewCanvas}" />  
        <mx:RemoveChildAction target="{viewCanvas}" />  
        <mx:AddChildAction relativeTo="{this}">  
            <mx:target>  
                <mx:Canvas id="editCanvas"  
                    addedToStage="editCanvas.includeInLayout = true"  
                    removedFromStage="editCanvas.includeInLayout = false">  
                    <mx:TextInput text="SAMPLE"/>  
                </mx:Canvas>  
            </mx:target>  
        </mx:AddChildAction>
```

```
</mx:Sequence>

</mx:Transition>
```

这个例子首先执行 Fade 特效，然后移除子组件并添加新子组件。如果你在 State 中使用 AddChild，则子组件将在 Fade 特效完成之前就被添加了。这个例子中是不会发生这样的事，通过使用 AddChildAction，你可以控制子组件何时被添加进来。

还有 RemoveChild 标签是用来控制何时移除组件，但不是删除组件，AddChildAction 的机理不是去自动调用 remove 方法。要想移除子组件，你必须在 State 中添加 RemoveChild 对象：

```
<mx:State name="edit">
    <mx:RemoveChild target="{viewCanvas}" />

</mx:State>
```

完整代码如下：

Code View:

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="300" currentState="view">
    <mx:transitions>
        <mx:Transition fromState="view" toState="edit">
            <mx:Sequence>
                <mx:Fade alphaFrom="1" alphaTo="0" duration="1000"
                    target="{viewCanvas}" />
                <mx:RemoveChildAction target="{viewCanvas}"
                    effectStart="trace('removing')"/>
                <mx:AddChildAction relativeTo="{this}">
                    <mx:target>
                        <mx:Canvas id="editCanvas"
                            addedToStage="editCanvas."
                            includeInLayout = "true"
                            removedFromStage="editCanvas.includeInLayout = false">
                            <mx:TextInput text="SAMPLE"/>
                        </mx:Canvas>
                    </mx:target>
                </mx:AddChildAction>
                <mx:SetPropertyAction target="{editCanvas}"
                    name="includeInLayout" value="true"/>
                <mx:SetPropertyAction target="{editCanvas}"
                    name="alpha" value="1"/>
                <mx:Glow color="0xffff00" blurXTo="30" blurYTo="30"
                    blurXFrom="0" blurYFrom="0" duration="1000" target="{this}" />
            </mx:Sequence>
        </mx:Transition>
    </mx:transitions>
</mx:HBox>
```

```

        <mx:Glow color="0xfffff00" blurXTo="30" blurYTo="30"
            blurXFrom="0" blurYFrom="0" duration="1000"
            target="{editCanvas}"/>
    </mx:Sequence>
</mx:Transition>
<mx:Transition fromState="edit" toState="view">
    <mx:Sequence>
        <mx:Fade alphaFrom="1" alphaTo="0" duration="1000"
            target="{editCanvas}"/>
        <mx:RemoveChildAction target="{editCanvas}"/>
        <mx:AddChildAction relativeTo="{this}"
            effectStart="trace('removing')">
            <mx:target>
                <mx:Canvas id="viewCanvas"
                    addedToStage="viewCanvas.includeInLayout =
true" removedFromStage="viewCanvas.includeInLayout = false">
                    <mx:Text text="DIFFERENT TEXT"/>
                </mx:Canvas>
            </mx:target>
        </mx:AddChildAction>
        <mx:SetPropertyAction target="{viewCanvas}"
            name="includeInLayout" value="true"/>
        <mx:SetPropertyAction target="{viewCanvas}"
            name="alpha" value="1"/>
        <mx:Glow color="0xfffff00" blurXTo="30" blurYTo="30"
            blurXFrom="0" blurYFrom="0" duration="1000"
            target="{this}"/>
        <mx:Glow color="0xfffff00" blurXTo="30" blurYTo="30"
            blurXFrom="0" blurYFrom="0" duration="1000"
            target="{viewCanvas}"/>
    </mx:Sequence>
</mx:Transition>
</mx:transitions>
<mx:states>
    <mx:State name="view">
        <mx:RemoveChild target="{editCanvas}"/>
    </mx:State>
    <mx:State name="edit">
        <mx:RemoveChild target="{viewCanvas}"/>
    </mx:State>
</mx:states>
<mx:ComboBoxdataProvider="{'view', 'edit'}"
    change="currentState = cb.selectedItem as String" id="cb"/>

```

```
</mx:HBox>
```

## 11.4节. 为特定类型的子节点过滤 Transitions

### 11.4.1. 问题

我想让 transition 只影响到某些子组件

### 11.4.2. 解决办法

使用 EffectTargetFilter 对象定义过滤函数来检测哪些目标将被应用 Transition。

### 11.4.3. 讨论

EffectTargetFilter 对象可定义过滤器检测哪些目标被应用到 transition。EffectTargetFilter 对象需要一个过滤函数，类似于一个数组，为每个传进 Transition 的对象返回 true 或 false，传递过滤函数给 EffectTargetFilter 的 filterFunction 属性：

```
filter.filterFunction = func;
private function func(propChanges:Array,
    instanceTarget:Object):Boolean
{
    if(instanceTarget is HBox)
    {
        return true;
    }
    return false;
}
```

你还要添加额外的语句决定是否可以让 Sequence 或 Parallel 应用到对象上。注意不能只应用 Transition 中个别的特效(详见[第11.5节](#).)

像下面这样传递 EffectTargetFilters 对象给 Parallel 或 Sequence:

Code View:

```
<mx:Sequence filter="resize" targets="[one, two, three]"
    customFilter="{filter}">
```

当 Sequence 被调用时，customFilter 对象返回过滤器中被允许的所有对象，这意味着如果必要的话在运行期可多次改变过滤函数产生不同的结果。完整的代码如下：

Code View:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="300" creationComplete="comp()"
    xmlns:cookbook="oreilly.cookbook.*">
<mx:Script>
<! [CDATA[
    import mx.containers.HBox;
    import mx.effects.EffectTargetFilter;

    [Bindable]
    private var filter:EffectTargetFilter;

    private function comp():void {
        filter = new EffectTargetFilter();
        filter.filterFunction = func;
    }

    private function func(propChanges:Array,
        instanceTarget:Object):Boolean {
        if(instanceTarget is HBox) {
            return true;
        }
        return false;
    }
]]>
</mx:Script>
<mx:transitions>
    <mx:Transition toState="closeState" fromState="openState">
        <mx:Sequence filter="resize" targets="[one, two,
            three]" customFilter="{filter}">
            <mx:Move xTo="200" xFrom="0"/>
            <mx:Glow color="0xffff00" blurXTo="20"
                blurYTo="20"/>
        </mx:Sequence>
    </mx:Transition>
</mx:transitions>
<mx:states>
    <mx:State name="closeState"/>
    <mx:State name="openState"/>
</mx:states>
```

```

<mx:HBox id="one">
    <mx:Text text="one"/>
    <mx:Text text="two"/>
</mx:HBox>
<mx:HBox id="two">
    <mx:Text text="one"/>
    <mx:Text text="two"/>
</mx:HBox>
<mx:Canvas id="three">
    <mx:Text text="one"/>
    <mx:Text text="two" y="10"/>
</mx:Canvas>
<mx:Button click="(currentState == 'openState') ? currentState
= 'closeState' : currentState = 'openState'" />

</mx:VBox>

```

## 11.5节. 对指定组件应用局部 Transition

### 11.5.1. 问题

我想应用一部分 Transition, Sequence, 或 Parallel 对象到某个子组件上。

### 11.5.2. 解决办法

在过滤函数基础上为每个特效过滤目标直到返回期望的所有子组件数据。

### 11.5.3. 讨论

正如[第11.4节](#)提及的那样 EffectTargetFilter 对象过滤只能应用整个 sequence 或整个 Parallel 组合特效。要想为每个特效过滤目标，你必须编写自定义函数返回每个特效的 targets 属性数组。因为特效都有自己的 targets 而不依赖与 Transition 的 targets，过滤函数必须遍历组件内的所有子组件，这需要付出昂贵的代码，有时候好的办法是把被应用的子组件添加到独立的数组中。

这个例子中的过滤函数循环遍历所有子组件，根据传入方法的参数返回组件中包含的所有 HBox 或 Canvas 对象数组：

```

private function returnArray(state:*):Array
{

```

```

var arr:Array = new Array();
var i:int;
if(state == "foo") {
    for(i = 0; i<this.numChildren; i++) {
        if(getChildAt(i) is HBox) {
            arr.pushgetChildAt(i));
        }
    }
} else {
    for(i = 0; i<this.numChildren; i++) {
        if(getChildAt(i) is Canvas) {
            arr.pushgetChildAt(i));
        }
    }
}
return arr;
}

```

下面的代码将调用这个方法:

Code View:

```

<mx:states>
    <mx:State name="closeState"/>
    <mx:State name="openState"/>

</mx:states>

```

每次特效被调用，数组都会被重建，这意味着基于组件的 currentState 状态单个特效可应用到不同类型的子组件，这让你创建单个特效却可以以不同的方式使用它，根据过滤函数返回特效所需的所有目标，完整代码如下：

Code View:

```

<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
    <mx:Script>
        <! [CDATA[
            import mx.core.UIComponent;

            [Bindable]
            private function returnArray(state:*):Array
            {
                var arr:Array = new Array();

```

```

        var i:int;
        if(state == "foo") {
            for(i = 0; i<this.numChildren; i++) {
                if(getChildAt(i) is HBox) {
                    arr.push(getChildAt(i));
                }
            }
        } else {
            for(i = 0; i<this.numChildren; i++) {
                if(getChildAt(i) is Canvas) {
                    arr.push(getChildAt(i));
                }
            }
        }
        return arr;
    }

    ]]>
</mx:Script>
<mx:transitions>
    <mx:Transition toState="closeState"
        fromState="openState">
        <mx:Sequence>
            <mx:Move xTo="200" xFrom="0"
                targets="{returnArray('foo')}"/>
            <mx:Glow color="0xffff00" blurXTo="20"
                blurYTo="20" targets="{returnArray('bar')}"/>
        </mx:Sequence>
    </mx:Transition>
</mx:transitions>
<mx:states>
    <mx:State name="closeState"/>
    <mx:State name="openState"/>
</mx:states>
<mx:HBox id="one">
    <mx:Text text="one"/>
    <mx:Text text="two"/>
</mx:HBox>
<mx:HBox id="two">
    <mx:Text text="one"/>
    <mx:Text text="two"/>
</mx:HBox>
<mx:Canvas id="three">
    <mx:Text text="one"/>

```

```

<mx:Text text="two" y="10"/>
</mx:Canvas>
<mx:Canvas id="four">
    <mx:Text text="three"/>
    <mx:Text text="four" x="10" y="10"/>
</mx:Canvas>
<mx:Button click="(currentState == 'openState') ? currentState = 'closeState' : currentState = 'openState'" />

</mx:VBox>

```

## 11.6节. 建立在基础 State 之上的 State

### 11.6.1. 问题

我想创建一个 state，继承其他 state 的所有属性并重新设置某些属性。

### 11.6.2. 解决办法

在新 state 中设置 basedOn 属性。

### 11.6.3. 讨论

在其他 states 基础上创建 states 是一种方便的方式以便创建出一组层级关系的 states。当一个 state 基于另一个 states 时，它继承了第一个 state 的所有属性，新 state 中任何重写的定义都被添加进来。这意味着如果一个 state 定义了一个 AddChild 方法，另一个基于此 state 的 state 也有自己的 AddChild 方法，原始 state 的 AddChild 方法仍存在于新的 State 中。

基于其他 State 的 State 很简单：

```
<mx:State name="secondaryState2" basedOn="primaryState">
```

下面的例子中，secondaryState2对象和 primaryState 对象一样添加同样的子组件，同样调用 SetProperty 方法。secondaryState2的 SetProperty 最后被调用，所以 title 将显示为 Third Title，而不是 Super New Title。

Code View:

```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="500" title="Initial Title">
```

```

<mx:states>
    <mx:State name="primaryState">
        <mx:AddChild>
            <mx:VBox>
                <mx:Text fontSize="18" text="NEW TEXT 1"/>
                <mx:Text fontSize="18" text="NEW TEXT 2"/>
            </mx:VBox>
        </mx:AddChild>
        <mx:SetProperty target="{this}" name="title"
            value="'Super New Title'"/>
    </mx:State>
    <mx:State name="secondaryState1">
        <mx:AddChild>
            <mx:RichTextEditor height="300" width="250"/>
        </mx:AddChild>
        <mx:SetProperty target="{this}" name="title"
            value="'Lame Old Title'"/>
    </mx:State>
    <mx:State name="secondaryState2" basedOn="primaryState">
        <mx:SetProperty target="{this}" name="title"
            value="'Third Title'"/>
    </mx:State>
</mx:states>
<mx:ComboBox dataProvider="{'primaryState', 'secondaryState1',
    'secondaryState2'}"
    change="currentState=cb.selectedItem as String" id="cb"/>

</mx:Panel>

```

## 11.7节. 用 HistoryManagement 整合 States 视图

### 11.7.1. 问题

我想用 Flex 框架的 HistoryManagement 机制整合 states。

### 11.7.2. 解决办法

创建扩展自 IHistoryManagerClient 接口的应用程序或组件。使用 HistoryManagement 注册应用程序，当 state 改变时使用 HistoryManager.save 方法保存当前 state。

### 11.7.3. 讨论

IHistoryManager client 定义了以下方法:

**loadState(state:Object):void**

加载此对象的状态

**saveState():Object**

保存此对象的状态

**toString():String**

将此对象转换为唯一的字符串

这些方法允许组件能正确的保存 State 的任何信息，在需要时进行还原。loadState 方法从存储的 URL 载入 State 信息:

```
public function saveState():Object {
    trace(" save state ");
    var state:Object = {};
    state.lastSearch = lastSearch;
    state.currentState = currentState;
    return state;
}
```

loadState 方法接受和读取从 HistoryManager 中传过来的 object 中的 State，并设置组件的 currentState:

```
public function loadState(state:Object):void {
    if (state) {
        trace(" last search "+state.lastSearch);
        lastSearch = searchInput.text = state.lastSearch;
        currentState = state.currentState;
    }
}
```

完整代码如下:

Code View:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    implements="mx.managers.IHistoryManagerClient"
    creationComplete=" HistoryManager.register(this) "
```

```

currentState="search">

<mx:Script>
<! [CDATA[
import mx.managers.HistoryManager;

public function loadState(state:Object):void {
    if (state!=null) {
        trace(" last search "+state.lastSearch);
        lastSearch = searchInput.text = state.lastSearch;
        currentState = state.currentState;
    }
}

// Save the current state and the searchString value.
public function saveState():Object {
    trace(" save state ");
    var state:Object = {};
    state.lastSearch = lastSearch;
    state.currentState = currentState;
    return state;
}

// The search string value.
[Bindable]
public var lastSearch:String;

public function search():void {
    lastSearch = searchInput.text;
    currentState = "display";
    HistoryManager.save();
}

public function reset():void {
    trace(" reset ");
    currentState = 'search';
    searchInput.text = "";
    lastSearch = "";
    HistoryManager.save();
}
]]>
</mx:Script>

<mx:states>

```

```

<mx:State name="display">
    <mx:SetProperty target="{panel}" name="title"
        value="Results"/>
    <mx:AddChild relativeTo="{panel}">
        <mx:VBox id="results">
            <mx:Text text="Getting Results"/>
            <mx:Button label="Reset" click="reset()"/>
        </mx:VBox>
    </mx:AddChild>
</mx:State>
<mx:State name="search">
    <mx:SetProperty target="{panel}" name="title"
        value="Search"/>
    <mx:AddChild relativeTo="{panel}">
        <mx:HBox id="searchFields" defaultButton="{btn}">
            <mx:TextInput id="searchInput" />
            <mx:Button id="btn" label="Find"
                click="search();"/>
        </mx:HBox>
    </mx:AddChild>
</mx:State>
</mx:states>
<mx:Panel id="panel" title="Results" resizeEffect="Resize">
</mx:Panel>
</mx:Application>

```

## 11.8节. 使用 States 的延时实例工厂

### 11.8.1. 问题

我需要一个对象，它能为 AddChild 对象实例化不同类型的对象。

### 11.8.2. 解决办法

创建工厂类，并赋值给 AddChild 对象的 targetFactory 属性。

### 11.8.3. 讨论

AddChild 对象的 targetFactory 属性需要一个实现 IDelayedInstance 接口的对象。

IDeferredInstance 接口只需要一个方法: `getInstance():Object`. 当 AddChild 对象需要一个新的可视化对象被添加到组件时该方法返回所需的实例对象。

这里提供的类很简单, 但它可根据 type 属性值返回不同类型的 UIComponent:

Code View:

```
package oreilly.cookbook
{
    import mx.containers.HBox;
    import mx.containers.VBox;
    import mx.controls.Button;
    import mx.controls.Text;
    import mx.controls.TextInput;
    import mx.core.IDeferredInstance;
    import mx.core.UIComponent;

    public class SpecialDeferredInstance implements
        IDeferredInstance
    {

        private var comp:UIComponent;
        private var _type:String;

        public function set type(str:String):void {
            _type = str;
        }

        public function get type():String{
            return _type;
        }

        public function getInstance():Object
        {
            var text:Object;
            if(_type == "TextVBox") {
                comp = new VBox();
                text = new Text();
                text.text = "TEXT";
                comp.addChild(text as Text);
                var btn:Button = new Button();
                btn.label = "LABEL";
                comp.addChild(btn);
                comp.height = 160;
                comp.width = 320;
            }
        }
    }
}
```

```

        }
    else
    {
        comp = new HBox();
        text = new TextInput();
        text.text = "TEXT";
        comp.addChild(text as TextInput);
        var btn:Button = new Button();
        btn.label = "LABEL";
        comp.addChild(btn);
        comp.height = 160;
        comp.width = 320;
    }
    return comp;
}
}
}

```

设置好 targetFactory 后，AddChild 方法根据 SpecialDeferredInstance 实例的 type 参数可有不同类型的对象，例如：

Code View:

```

<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300" currentState="empty">
<mx:Script>
<! [CDATA[
import oreilly.cookbook.SpecialDeferredInstance;
[Bindable]
private var defInst:SpecialDeferredInstance =
new SpecialDeferredInstance();
]]>
</mx:Script>
<mx:states>
<mx:State name="defInst">
<mx:AddChild relativeTo="{mainHolder}"
targetFactory="{defInst}" />
</mx:State>
<mx:State name="empty"/>
</mx:states>
<mx:Button click="currentState == 'defInst' ? currentState =
'empty' : currentState = 'defInst'" label="change"/>
<mx:HBox id="mainHolder"/>

```

```
</mx:VBox>
```

## 11.9节. 对添加到 **State** 中的 **Object** 进行数据绑定

### 11.9.1. 问题

我想绑定一个对象到属性上，该对象是进入某个 state 时才会被创建。

### 11.9.2. 解决办法

使用 `mx.binding.utils.BindingUtils` 类的 `bindProperty` 方法动态创建绑定。

### 11.9.3. 讨论

你可以在编译器在 MXML 文件内使用 {} 或者在运行期使用 `bindProperty` 方法创建绑定。  
`bindProperty` 方法格式如下：

```
public static function bindProperty(site:Object, prop:String, host:Object,  
chain:Object, commitOnly:Boolean = false):ChangeWatcher
```

方法参数如下：

#### site

定义绑定到 `chain` 的属性的 Object。如果你想使用绑定来改变 `TextField` 的值，例如，这个 `site` 就是 `TextField`。

#### prop

在要绑定的 `site Object` 中定义的公用属性的名称。当 `chain` 值更改时，该属性将接收 `chain` 的当前值。如果你使用绑定来改变 `TextField` 的值，那 `prop` 就是 `TextField` 的 `text`。

#### host

用于承载要监视的属性或属性链的对象。如果你要绑定到 `TextInput` 的文本值，那 `host` 就是 `TextInput`。

#### chain

用于指定要监视的属性或属性链的值。合法值可以是包含宿主对象公用可绑定属性名称的字

字符串。如果你要绑定到 TextInput 的文本值，那 chain 就是这个文本值。

#### commitOnly

如果仅在提交 change 事件时需要调用处理函数，则设置为 true。

关于数据绑定的细节内容我们将在[第十四章, "数据绑定"](#)详细讲解，这一节重点讲解使用 bindProperty 创建绑定，将新创建的 RichTextEditor 绑定到 TextArea 上：

Code View:

```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml" width="450"
height="650" title="Initial Title" layout="vertical">
<mx:Script>
<! [CDATA[
import mx.binding.utils.*;
]]>
</mx:Script>
<mx:states>
<mx:State name="primaryState">
<mx:AddChild>
<mx:VBox id="vbox">
<mx:Text fontSize="18" text="NEW TEXT 1"/>
<mx:Text fontSize="18" text="NEW TEXT 2"/>
</mx:VBox>
</mx:AddChild>
<mx:SetProperty target="{this}" name="title"
value="'Super New Title'"/>
</mx:State>
<mx:State name="secondaryState1">
<mx:AddChild>
```

这里通过 bindProperty 方法将 TextArea 的 htmlText 属性被绑定到 RichTextEditor 的 htmlText 值。  
如果你试图把 TextArea 的 htmlText 属性绑定到一个还没创建的控件上将会抛出异常：

Code View:

```
<mx:RichTextEditor id="richText" height="200"
width="250"
creationComplete="BindingUtils.bindProperty(area, 'htmlText',
richText, 'htmlText')"/>
</mx:AddChild>
<mx:SetProperty target="{this}" name="title"
value="'Lame Old Title'"/>
</mx:State>
<mx:State name="secondaryState2" basedOn="primaryState">
<mx:SetProperty target="{this}" name="title"
value="'Third Title'"/>
```

```

</mx:State>
</mx:states>
<mx:ComboBox
   dataProvider=" [ 'primaryState', 'secondaryState1',
    'secondaryState2' ] "
    change="currentState=cb.selectedItem as String" id="cb"/>
<mx:TextArea height="100" width="450" id="area"
    htmlText="foo bar baz"/>

</mx:Panel>

```

## 11.10节. 在 State Changes 事件中添加和删除事件监听器

### 11.10.1. 问题

我想给 State 中创建添加的组件注册事件监听器，当 state 改变时移除监听器。

### 11.10.2. 解决办法

将事件监听器关联到组件的 addedToStage 事件，在 removedFromStage 事件中移除事件监听器。或者使用 SetEventHandler 对象创建事件监听器。

### 11.10.3. 讨论

在 ActionScript3中确保事件处理器能被正确的移除是保证应用程序不会造成消耗 Flash Player 大量内存的最好方法。也就是说，添加事件处理器也就伴随着要移除这些事件处理器。例如在组件被添加或移除时也要添加和移除事件处理器：

Code View:

```

<mx:AddChild relativeTo="{holder}">
    <mx:TextInput text="TEXT" id="textInput1" width="200"
        addedToStage="{textInput1.addEventListener(TextEvent.TEXT_INPUT,
            checkNewTextInput) }"
        removedFromStage="{textInput2.addEventListener( TextEvent.TEXT_INPUT,
            checkNewTextInput) }" />
</mx:AddChild>

```

使用 SetEventHandler 对象也能完成同样的功能，它接受一个事件处理器相关联的目标。

handlerFunction 指定事件处理器名称：

Code View:

```
<mx:SetEventHandler handlerFunction="checkNewTextInput"  
name="{TextEvent.TEXT_INPUT}" target="{textInput2}"/>
```

当用户改变 state 时事件处理器就会被添加和删除。这个可以通过把事件监听器改为 Event.ENTER\_FRAME 来的已确认何时监听器被添加和移除。完整代码如下：

Code View:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"  
height="300">  
<mx:Script>  
<! [CDATA [  
  
    private function checkNewTextInput(event:Event):void  
    {  
        trace(" event "+event.target);  
    }  
  
    ]]>  
</mx:Script>  
<mx:states>  
    <mx:State id="openState" name="openState">  
        <mx:AddChild relativeTo="{holder}">  
            <mx:TextInput text="TEXT" id="textInput1" width="200"  
                addedToStage="{textInput1.addEventListener(TextEv  
                    ent.TEXT_INPUT, checkNewTextInput)}"  
                removedFromStage="{textInput2.addEventListener  
(TextEvent.TEXT_INPUT, checkNewTextInput)}"/>  
        </mx:AddChild>  
    </mx:State>  
    <mx:State id="closedState" name="closedState">  
        <mx:SetEventHandler handlerFunction="checkNewTextInput"  
name="{TextEvent.TEXT_INPUT}" target="{textInput2}"/>  
        <mx:AddChild relativeTo="{holder}">  
            <mx:TextInput id="textInput2" width="200"  
                text="MORE TEXT"/>  
        </mx:AddChild>  
    </mx:State>  
</mx:states>  
<mx:VBox id="holder">
```

```

<mx:Button click="(this.currentState == 'openState') ?
    currentState = 'closedState' : currentState =
    'openState'" label="change"/>
</mx:VBox>

</mx:Canvas>

```

## 11.11节。添加视图 States 到 Flash 组件

### 11.11.1. 问题

我想使用 Flash 组件作为 states。

### 11.11.2. 解决办法

在 UIMovieClip 或 ContainerMovieClip 实例内提供帧标签作为 states。

### 11.11.3. 讨论

首先创建将在 Flex 应用程序中实例化的类。在 Flex 程序中如果没有添加子对象则这个类必须扩展自 UIMovieClip 类，否则需扩展自 ContainerMovieClip 类，例如：

Code View:

```

package{
    import flash.text.TextField;
    import mx.flash.ContainerMovieClip;
    public class FlashAssetClass extends ContainerMovieClip{
        private var txt:TextField;
        public function FlashAssetClass() {
            txt = new TextField();
            addChild(txt);
            txt.text = "INIT";
            super();
        }

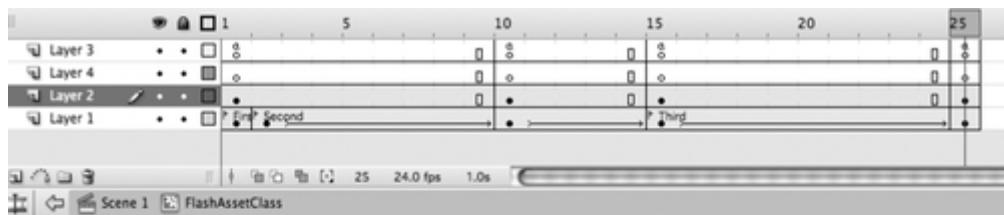
        override public function set
        currentState(value:String):void {
            trace(" set current state ");
            super.currentState = value;
        }
    }
}

```

```
txt.text = value;  
}  
  
override public function gotoAndStop(frame:Object,  
scene:String=null) :void {  
    trace(" go to and stop ");  
    txt.text = String(frame);  
    super.gotoAndStop(frame, scene);  
}  
}  
}
```

`currentState` 和 `gotoAndStop` 方法被重写，这样当他们改变和 `TextField` 被添加时两个方法都能输出信息。[Figure 11-1](#) 清楚显示多个 Flash 帧，标记为 First, Second, 和 Third, 这些在 Flex 程序里将被作为 states。

**Figure 11-1. Creating distinct frames within the Flash IDE**



## Code View:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="400" creationComplete="createComp()">
<mx:Script>
    <![CDATA[
        import mx.controls.Label;
        import FlashAssetClass;
```

```

private var classInst:FlashAssetClass;

private function createComp():void {
    classInst = new FlashAssetClass();
    rawChildren.addChild(classInst);
    invalidateDisplayList();
}

] ]>
</mx:Script>
<mx:Button click="classInst.currentState = 'First';"
    label="First" y="300"/>
<mx:Button click="classInst.currentState = 'Second';"
    label="First" y="330"/>
<mx:Button click="classInst.currentState = 'Third';"
    label="First" y="360"/>

</mx:Canvas>

```

注意要改变 FlashAssetClass 对象的 state，你只要设置下 currentState 属性即可，和一般的组件差不多。

当 state 改变时，ContainerMovieClip 类默认下调用 gotoAndStop 方法，停止帧回放。如果你要设置 currentState 来播放指定的动画，你可以重写 currentState setter 去调用 gotoAndPlay 方法，这样将不能停止帧回放，例如：

```

override public function set currentState(value:String):void {
    trace(" set current state ");
    //super.currentState = value;
    gotoAndPlay(value);
    txt.text = value;
}

```

## 11.12节. 处理 State Change 事件

### 11.12.1. 问题

我想了解和 states 改变时所关联的事件。

## 11.12.2. 解决办法

添加 trace 语句到 ENTER\_STATE 事件和子组件的 CREATION\_COMPLETE 事件中，来显示当进入 state 子组件何时被创建，离开 state 组件何时被移除，但不是删除。

## 11.12.3. 讨论

当 state 改变时有一系列事件会发生，看起来有点复杂。State 自身广播的事件有从子组件发出的 creation, addedToStage, and removedFromStage 事件。当 State 对象被创建，进入，离开时有下列不同类型的事件触发：

`mx.events.StateChangeEvent.CURRENT_STATE_CHANGE`

定义当视图状态更改时调度的事件的 type 属性的值

`mx.events.StateChangeEvent.CURRENT_STATE_CHANGING`

定义当视图状态将要更改时调度的事件的 type 属性值

State 事件有：

`mx.events.FlexEvent.ENTER_STATE`

当进入视图状态时调度。这个事件是在基础视图状态改变后触发

`mx.events.FlexEvent.EXIT_STATE`

在离开视图状态前调度。这个事件是在基础视图状态被移除前触发

子组件事件有：

`mx.events.FlexEvent.ADD`

当组件通过 addChild 或 addChildAt 方法添加到容器作为子控件时触发。

`mx.events.FlexEvent.REMOVE`

D 当组件通过 removeChild 或 removeChildAt 方法从容器中移除时触发。

`mx.events.FlexEvent.PREINITIALIZE`

当组件按顺序初始化前触发

`mx.events.FlexEvent.INITIALIZE`

当组件完成构造和设置所有属性后触发。

`mx.events.FlexEvent.CREATION_COMPLETE`

当组件完成构造，属性处理，计算尺寸，布局和绘制后触发。

当一个组件从基础 state(组件被创建时的状态)转换到第二个 state 时，事件的触发顺序是这

样的：

```
[child] constructor()  
  
[component] CURRENT_STATE_CHANGING;  
  
[child] ADD;  
  
[child] PREINITIALIZE;  
  
[child] createChildren();  
  
[child] INITIALIZE;  
  
[state] ENTER_STATE; (second state)  
  
[component] CURRENT_STATE_CHANGE;  
  
[child] commitProperties();  
  
[child] updateDisplayList();  
  
[child] CREATION_COMPLETE;
```

当组件从第二个 state 转换到基础 state 时，事件的顺序和方法调用顺序是这样的：

```
[component] CURRENT_STATE_CHANGING;  
  
[state] EXIT_STATE; (second state)  
  
[child] REMOVE;  
  
[component] CURRENT_STATE_CHANGE;
```

当组件再次回到第二个 state 时：

```
[component] CURRENT_STATE_CHANGING;  
  
[child] ADD;  
  
[state] ENTER_STATE; (second state)  
  
[component] CURRENT_STATE_CHANGE;
```

```
[child] updateDisplayList();
```

我们注意到组件第一次进入第二个 state 和第二次进入第二个 state 时是不一样的，子组件没有被重建，只是简单的被重新添加进来。

## 11.13节. 动态生成 States 和 Transitions

### 11.13.1. 问题

我需要动态生成新的 states 和 transitions。

### 11.13.2. 解决办法

创建新的 State 和 Transition 对象，添加它们的属性，把它们添加到每个 UIComponent 对象定义的 states 和 transition 数组中。

### 11.13.3. 讨论

一般情况下是不能频繁地创建新的 states 和 transition 的。但是在某些情况下是很有必要的。比如模板组件。因为每个 UIComponent 对象有一个 states 数组和 transition 数组，包含所有的 State 和 Transition 对象，创建新的 states 和 transitions 只需要简单的定义 State 或 Transition 并添加到对应的数组中即可：

```
var state:State = new State();
var button:Button = new Button();
button.label = "LABEL";
var addChild:AddChild = new AddChild(vbox, button);
state.overrides = [addChild];
state.name = "buttonState";
states.push(state);
```

这里被重写的数组是将被使用的 State 的重写列表，就是，哪些功能将被定义，哪些特定的 State 属性将被定义，比如任何 SetProperty 或 SetStyle 操作，所有为 State 定义的重写都是实现了 IOverride 接口，一般包含下列：

[AddChild](#)

[RemoveChild](#)

[SetEventHandler](#)

[SetProperty](#)

[SetStyle](#)

(For more information on the `IOVERRIDE` interface, see Recipe 11.15.)

下面是完整的代码，注意创建 `Transition` 对象，添加属性，指定 `toState` 和 `fromState` 属性的这些方法：

Code View:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
<mx:Script>
<! [CDATA[
    import mx.controls.Button;
    import mx.states.AddChild;
    import mx.states.State;
    import mx.states.Transition;
    import mx.effects.Move;

    public function addTransitions():void {
        var transition:Transition = new Transition();
        var move:Move = new Move();
        move.duration=400;
        move.target = vbox;
        transition.fromState = "buttonState";
        transition.toState = "*";
        transition.effect = move;
        transitions.push(transition);
    }

    public function addState():void {
        var state:State = new State();
        var button:Button = new Button();
        button.label = "LABEL";
        var addChild:AddChild = new AddChild(vbox, button);
        state.overrides = [addChild];
        state.name = "buttonState";
        states.push(state);
    }
}]>
```

```
    ]]>
</mx:Script>
<mx:VBox id="vbox"/>
<mx:Button click="addTransitions()" label="new transition"/>
<mx:Button click="addState()" label="new state"/>
<mx:Button click="currentState = 'buttonState'" label="change
state"/>

</mx:VBox>
```

## 11.14节. 创建 State 的自定义动作(action)

### 11.14.1. 问题

我想在进入 state 时创建 State 对象的自定义动作。

### 11.14.2. 解决办法

创建一个 IOOverride 接口的实现类，重写所有需要传递自定义行为的所有方法。

### 11.14.3. 讨论

要实现额外的 state 动作，你需要创建自定义 IOOverride 对象，当进入 state 时去执行你自己的条件逻辑。任何实现此接口的对象都能被添加到 state 的重写数组中--任何组件都有下列方法：

#### apply(parent:UIComponent):void

该方法应用重写，执行重写(override)的自定义行为。将被改变的 Parent 参数值会被存储在此方法中以便当离开 state 执行撤销操作。

#### initialize():void

初始化重写

#### remove(parent:UIComponent):void

删除重写。在 apply() 方法中记住的值将被恢复。当进入状态时自动调用此方法。不应直接对其进行调用

下面的例子中，实现 IOverride 接口的类叫 CustomOverride ，通过 apply 语句应用自定义条件逻辑：

Code View:

```
package oreilly.cookbook
{
    import flash.display.DisplayObject;
    import mx.core.UIComponent;
    import mx.states.IOverride;

    public class CustomOverride implements IOverride {
        private var widthValue:Number;
        private var _target:DisplayObject;

        public function CustomOverride(target:DisplayObject = null)
        {
            _target = target;
        }

        public function get target():DisplayObject {
            return _target;
        }

        public function set target(value:DisplayObject):void {
            _target = value;
        }
        //empty
        public function initialize():void {}

        //here we make sure to store the value of the parent before
        we change it
        //so that we can use this value in the remove method
        public function apply(parent:UIComponent):void {
            widthValue = _target.width;
            if(_target.width > 500) {
                _target.width = 500;
            }
        }

        //here we use the stored value
        public function remove(parent:UIComponent):void {
            _target.width = widthValue;
        }
    }
}
```

```
}
```

这里新的 CustomOverride 类被用来改变 vbox 的大小，这个类可以被扩展去使用多个目标和做更可能多的事情而不仅仅局限于改变目标宽度，这里只是为了做个简单的演示：

```
<mx:states>
  <mx:State name="openState">
    <cookbook:CustomOverride target="{box1}" />
  </mx:State>
</mx:states>
```

## 第十二章. 特效(Flexer:Nigel)

效果是 Flex 应用程序中一个重要的部分同时也是 Rich Internet Application 称呼中 Rich 的重要元素之一。理解 Flex 的效果框架和效果不仅仅对设计和实现，即用户能看得见的元素效果重要，同时也对用户往往忽略的部分很重要，即用户往往不关心效果是否正确地实现，而只关心应用程序缓慢或没有进行适当的垃圾回收的现象。

```
var timer:Timer = new Timer(100, 0);
timer.addEventListener(TimerEvent.TIMER, performEffect);
timer.start();

private function performEffect(event:Event):void {
    //effect implementation
```

当然，与简单地允许开发者创建一个效果的实例并调用它的 play 方法相比，实际上特效框架非常复杂。事实上，EffectManager 类管理所有特效的所有实例以避免不必要的定时器和方法调用造成处理器负担过重。一个效果应该看作两个特殊的元素：效果的 EffectInstance(它包括效果的相关信息，效果应该做什么，会影响什么元素)，和 Effect 类(Effect“扮演”工厂的角色，生成效果，开始执行效果，并且在结束后删除它)。

一个效果的播放由四个动作组成。第一，它为每个目标组件创建一个 EffectInstance 类的实例。这表明如果一个效果要作用于四个目标对象的话，就会导致创建四个 EffectInstance 对象的结果。第二，框架会从工厂对象里复制所有的配置信息到每一个实例：持续时间，重复次数，延迟时间，等等，都作为新实例的属性配置。第三，效果在对象上使用前面创建的实例对象播放。最后，框架，特别是 EffectManager 类，效果播放结束的时后删除实例对象。

通常在使用效果的时候，作为开发者的你仅仅是在使用工厂类处理生成的效果。但是，当你开始创建自定义的效果的时候，你将同时创建作为效果类型的工厂 Effect 对象和实际在目标上播放的 EffectInstance 对象。使用效果，无论注意与否，都要创建生成实例对象的工厂。你要做的所有配置都在工厂对象里面设置，然后由工厂将这些设置传入生成的实例对象。查看一下框架源码，你会注意到 Glow 类和 GlowInstance 类。要创建自己的效果，就要成对的创建与之类似的类。

### 12.1 节. 在 MXML 和 ActionScript 里调用 Effect

#### 12.1.1. 问题

我想在应用程序里创建并调用一个效果实例。

#### 12.1.2. 解决办法

如果要在 MXML 里面定义一个效果，将 Effect 标签加到你的组件的顶层标签内。若要在 ActionScript 里面定义一个效果，导入正确的效果类，实例化之，分配一个 UIComponent 作为其目标，并调用 play 方法播放效果。

### 12.1.3. 讨论

Effect 类要求设置一个目标 UIComponent。当在 ActionScript 内实例化一个 Effect，目标可以通过构造器传入 Effect 内：

```
var blur:Blur = new Blur(component);
```

The target can also be set once the Effect has been instantiated by using the target property of the Effect class. The target is the UIComponent that the Effect will affect when the play method of the Effect is called. When an Effect is defined in MXML, the target UIComponent must be assigned within a binding tag:

目标的设置也可以在 Effect 实例化完成后通过设置 Effect 类的 target 属性来完成。所谓目标就是一个当调用 Effect 的 play 方法的时候会对其产生作用的 UIComponent。当一个 Effect 定义在 MXML 后，目标 UIComponent 必须以绑定标签的形式进行分配：

Code View:

```
<mx:Glow id="glowEffect" duration="1000" color="#ff0f0f"
target="{glowingTI}" />
```

在下面的例子里，MXML 里面的 Glow 效果将在点击按钮的时候进行实例化：

```
<mx:Button click="glowEffect.play()" />
```

下一个例子里面，在 applyBlur 方法里通过构造器将 glowingTI 作为目标分配给 Blur 效果。在 Effect 的相关属性成功设置后，调用 play 方法。

Code View:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="600">
<mx:Script>
<! [CDATA[
import mx.effects.Blur;
private var blur:Blur;
private function applyBlur():void {
    blur = new Blur(glowingTI);
    blur.blurXFrom = 0;
    blur.blurXTo = 20; //the amount of blur in pixels
    blur.blurYFrom = 0;
    blur.blurYTo = 20; //the amount of blur in pixels
    blur.duration = 1000;
}
]]>
```

```

        blur.play();
    }
] ]>
</mx:Script>
<!-- the properties of the Glow effect set here are the color
of the Glow and the
length of time that the Glow will be displayed --&gt;
&lt;mx:Glow id="glowEffect" duration="1000" color="#ff0f0f"
target="{glowingTI}" /&gt;
&lt;mx:TextInput id="glowingTI"/&gt;
&lt;mx:Button click="applyBlur()" toggle="true" id="glowToggle"
label="Play the BlurEffect"/&gt;
&lt;mx:Button click="glowEffect.play()" label="Play the Glow
Effect"/&gt;
&lt;/mx:VBox &gt;</pre>

```

## 12.2 节. 建立一个自定义效果

### 12.2.1 问题

我想创建一个既可以在 MXML 也可以在 ActionScript 内使用的自定义效果。

### 12.2.2 解决办法

创建一个继承 Effect 的类以及任何你想要再实例化的时候传入实例的 getter 、 setter 属性。然后再创建一个继承 EffectInstance 的实例类，该类即是实际播放变化效果的类。

### 12.2.3 讨论

在 Flex 框架中，每个效果有两个元素组成：一个 Effect 和一个 EffectInstance。Effect 创建 EffectInstance 并传递属性给它们。这种责任分配允许你轻易地创建在多个对象上播放的效果。

要这样做的话，首先定义一个 TestEffect 类(随后创建 EffectInstance 的工厂)，设置它们的属性，并且调用每个实例的 play 方法：

Code View:

```

package oreilly.cookbook
{
    import mx.effects.Effect;
    import mx.effects.IEffectInstance;
    import mx.events.EffectEvent;
```

```

import oreilly.cookbook.TestInstance;

public class TestEffect extends Effect
{
    public var color:uint;
    public var alpha:Number;

    public function TestEffect(target:Object=null) {
        // call the base constructor of course
        super(target);
        // set our instance class to the desired instance type
        instanceClass = TestInstance;
    }

    override protected function
    initInstance(instance:IEffectInstance):void {
        trace(" instance initialized ");
        super.initInstance(instance);
        // now that we've instantiated our instance, we can set
        its properties
        TestInstance(instance).color = color;
        TestInstance(instance).alpha = alpha;
    }

    override public function getAffectedProperties():Array {
        trace(" return all the target properties ");
        return [];
    }

    override protected function
    effectEndHandler(event:EffectEvent):void {
        trace(" effect ended ");
    }

    override protected function
    effectStartHandler(event:EffectEvent):void {
        trace(" effect started ");
    }
}

```

注意前面的代码，在 `initInstance` 方法里，创建 `EffectInstance` 类的实例。`TestInstance` 类申明为 `TestInstance` 类型，并且 `TestInstance` 的属性都设置为 `TestEffect` 工厂的属性值。这让你可以通过 `TestEffect` 工厂一次性设置每个 `TestInstance` 实例的属性。

由 TestEffect 工厂生成的 TestInstance 类代码如下：

Code View:

```
package oreilly.cookbook

{
    import flash.display.DisplayObject;
    import mx.core.Container;
    import mx.core.FlexShape;
    import mx.core.UIComponent;
    import mx.effects.EffectInstance;

    public class TestInstance extends EffectInstance
    {
        public var alpha:Number;
        public var color:uint;

        public function TestInstance(target:Object)
        {
            super(target);
        }

        override public function play():void
        {
            super.play();
            (target as DisplayObject).alpha = alpha;
            var shape:FlexShape = new FlexShape();
            shape.graphics.beginFill(color, 1.0);
            shape.graphics.drawRect(0, 0, (target as
                DisplayObject).width, (target as
                DisplayObject).height);
            shape.graphics.endFill();
            var uiComp:UIComponent = new UIComponent();
            uiComp.addChild(shape);
            UIComponent(target).addChild(uiComp);
        }
    }
}
```

创建 TestInstance 的时候，每个 TestInstance 的 target 属性都由 TestEffect 工厂类设置。这保证如果传入多个目标对象给 Effect 类的 targets 属性，TestEffect，一个 TestInstance 的实例，将会创建并且在每个对象上播放。TestInstance 的 color 和 alpha 属性将会在实例创建的时候由 TestEffect 的 initInstance 方法设置。

TestInstance 类里覆盖的 play 方法包含了显示逻辑以改变分配给 TestInstance 的目标 UIComponent。

## 12.3 节. 创建 Effects 的 Parallel 系列或 Sequence 系列

### 12.3.1 问题

我想创建多个效果并行或者顺序播放。

### 12.3.2 解决办法

使用 Parallel 标签包含多个需要同时播放的效果或者使用 Sequence 标签包含多个需要顺序播放的效果。

### 12.3.3 讨论

Sequence 标签会在上一个 Effect 对象广播它的 effectComplete 事件时继续播放下一个效果。 Sequence 过程序列由多个 Parallel 效果标签组成，因为 Parallel 标签作为 Effect 同等对待并且具有 Sequence 在 Effect 或 Parallel 播放结束的时候需要调用的 play 方法。

Code View:

```
<mx:Sequence id=" sequencee" target="{ this }">
    <mx:Blur duration="3000" blurXTo="10" blurYTo="10"
        blurXFrom="0" blurYFrom="0"/>
    <mx:Glow duration="3000" color="#ffff00"/>
</mx:Sequence>
```

Parallel 标签通过传递所有的 target 对象，或者每个 Effect 或 Sequence 标签包含的对象，并且调用 Parallel 标签包含的所有 Effect 对象的 play 方法。

Code View:

```
<mx:Parallel id=" parallel" targets="[ bar, foo ]">
    <mx:Blur duration="3000" blurXTo="10" blurYTo="10"
        blurXFrom="0" blurYFrom="0"/>
    <mx:Glow duration="3000" color="#ffff00"/>
</mx:Parallel>
<mx:ComboBox id="bar" dataProvider="{'one', 'two', 'three'}"/>
<mx:ComboBox id="foo" dataProvider="{'one', 'two', 'three'}"/>
```

## 12.4 节 暂停、倒放和重新播放一个 Effect

### 12.4.1 问题

我需要在效果运行的时候能够暂停，并且之后在当前或开始位置重新播放效果。

## 12.4.2 解决办法

使用 pause 方法停止效果，这样效果可以重新启动同样也可以使用 resume 方法让效果从停止的地方继续播放。

## 12.4.3 讨论

Effect 类的 stop 方法和 pause 方法产生相同的结果：它们都使正在播放的效果停下来。但是 stop 方法重置了效果的计时器使得效果不能继续播放。而 pause 方法则只是简单的暂停计时器，因此，效果可以从暂停的时间点重新继续播放。暂停的效果也可以回放；但是停止的效果却不能。

你也可以暂停并继续效果的 Parallel 或 Sequence。代码如下：

Code View:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
<mx:Parallel id="parallel" target="{this}">
<mx:Blur duration="3000" blurXTo="10" blurYTo="10"
blurXFrom="0" blurYFrom="0"/>
<mx:Glow duration="3000" color="#ffff00"/>
</mx:Parallel>
<mx:Button click="parallel.play();" label="play()"/>
<mx:Button click="parallel.pause();" label="pause()"/>
<mx:Button click="parallel.reverse()" label="reverse()"/>
<mx:Button click="parallel.resume()" label="resume ()"/>
</mx:VBox >
```

If the reverse method is called on the Sequence, Parallel, or effect after the pause method has been called, the resume method will need to be called before the effect will begin playing in reverse.

如果在暂停后对 Sequence、Parallel 或者效果调用 reverse 方法，实际上在回放开始之前，需要先调用了 resume 方法让效果可以开始回放。

## 12.5 节 创建自定义 Effect 触发器

### 12.5.1 问题

我想为组件创建自定义的效果触发器。

## 12.5.2 解决办法

在组件里使用 Effect 元数据标签定义触发器的名称和绑定到触发器的事件。

## 12.5.3 讨论

一个触发器定义了一个播放效果的事件，触发器通常在 Flex 框架中使用—例如，要定义一个 ComboBox 组件的 mouseDownEffect 事件。

```
<mx:ComboBox mouseDownEffect="{glowEffect}" />
```

当在 ComboBox 内广播 mouseDown 事件的时候，绑定到该 ComboBox 实例的 mouseDownEffect 属性的效果即被播放。这里，即播放名为 glowEffect 的效果。要定义一个自定义的触发器名字，定义一个具有名称和事件类型的 Event，并且定义一个具有触发器名称和绑定到该触发器的事件名称的 Effect。

```
[Event(name="darken", type="flash.events.Event")]
[Effect(name="darkenEffect", event="darken")]
```

在前面的代码中，定义一个名为 darkenEffect 的触发器，可以给这个触发器绑定一个 Effect。绑定到这个触发器的 Effect 将会在此组件广播一个名为 darken 的 Event 的时候开始播放。下面是完整的代码：

Code View:

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
<mx:Metadata>
    [Event(name="darken", type="flash.events.Event")]
    [Effect(name="darkenEffect", event="darken")]

    [Event(name="lighten", type="flash.events.Event")]
    [Effect(name="lightenEffect", event="lighten")]
</mx:Metadata>
<mx:Script>
    <![CDATA[
        import flash.events.Event;

        private function dispatchDarken():void {
            //this will cause whatever Effect is bound
            //to the darkenEffect trigger to play
            dispatchEvent(new Event("darken"));
        }

        private function dispatchLighten():void {
    
```

```

        //this will cause whatever Effect is bound
        //to the lightenEffect trigger to play
        dispatchEvent(new Event("lighten"));
    }

] ]>
</mx:Script>
<mx:Button click="dispatchDarken()" label="darken"/>
<mx:Button click="dispatchLighten()" label="lighten"/>
</mx:HBox>

```

上面的代码保存为 CustomTriggerExample 文件并用<cookbook:CustomTriggerExample>标签实现下面的例子。要实现这个组件并使用两个新触发器，即两个 Glow 效果绑定到两个触发器上：

Code View:

```

<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="900" xmlns:cookbook="oreilly.cookbook.*">
<mx:Glow color="#000000" duration="3000" id="darkenFilter"/>
<mx:Glow color="#ffffff" duration="3000" id="lightenFilter"/>
<cookbook:CustomTriggerExample      darkenEffect="{darkenFilter}"
lightenEffect="{lightenFilter}"/>
</mx:HBox>

```

## 12.6 节 创建渐变特效

### 12.6.1 问题

我想要创建一个自定义的渐变效果，此类效果播放的时候在指定的持续时间内慢慢的改变它的属性。

### 12.6.2 解决办法

继承 TweenEffect 和 TweenEffectInstance 类创建一个工厂对象和一个传给每个目标的由工厂生成的类。

### 12.6.3 讨论

Effect 和 TweenEffect 显著的区别在于 TweenEffect 需要一段时间播放。TweenEffect 的开始属性和结束属性传入到 EffectInstance 内，它使用这些属性在一段时间内对目标对象创建、添加新的过滤器实例或者改变目标对象的属性。通过在 TweenInstance 类里使用一个 mx.effects.Tween 对象来在持续的一段时间内生成属性值的变化效果。

这一小节将展示如何建立一个简单的渐变效果，根据赋值到 TweenEffect 类的 duration 属性使它目标对象的透明度淡出。Tween 对象，像效果一样，也是由两个类组成，比如：一个为每个传入 TweenEffect 的 target 生成 TweenInstance 实例的 TweenEffect 工厂类，和一个创建 Tween 对象并使用 Tween 对象的值生成持续时间内效果的 TweenInstance 对象。

先看看 TweenEffect:

Code View:

```
package oreilly.cookbook

{
    import mx.effects.TweenEffect;

    public class CustomTweenEffect extends TweenEffect
    {

        public var finalAlpha:Number = 1.0;

        public function CustomTweenEffect (target:Object=null) {
            super(target);
        }

        public function
        CustomDisplacementEffect(target:Object=null) {
            super(target);
            this.instanceClass = CustomTweenInstance;
        }

        //create our new instance
        override protected function
        initInstance(instance:IEffectInstance):void {
            super.initInstance(instance);
            // now that we've instantiated our instance, we can set
            its properties
            CustomTweenInstance(instance).finalAlpha =
                this.finalAlpha;
        }

        override public function getAffectedProperties():Array {
            trace(" return all the target properties ");
            return [];
        }
    }
}
```

```
    }

}

}
```

当实例化 TweenInstance 对象的时候，设置用以传入 initInstance 方法的每个 CustomTweenInstance 对象的 finalAlpha 属性。

CustomTweenInstance 类继承 TweenEffectInstance 类并覆盖该类的 play 和 onTweenUpdate 方法。覆盖的 play 方法包含例示 Tween 对象的逻辑，该对象即为生成 TweenEffect 持续属性变化的 Tween:

```
override public function play():void {
    super.play();
    this.tween = new Tween(this, 0, finalAlpha, duration);
    (target as DisplayObject).alpha = 0;
}
```

从 CustomTweenEffect 里传入 finalAlpha 和 duration 属性，同时 mx.effects.Tween 会为 SWF 的每帧生成一个值，从初始值(这里是 0)平滑地移动，直至最终值(这里是 finalAlpha 变量)。如果需要可以给 Tween 对象传入多值数组，只要初始值数组的元素个数和终止值数组的元素个数相同即可。TweenEffectInstance 的 play 方法，在这里通过 super.play 调用，为 onTweenUpdate 方法添加侦听到 Tween 上。通过覆盖此方法，你可以给 TweenEffectInstance 添加任何自定义逻辑。

```
override public function onTweenUpdate(value:Object):void {
    (target as DisplayObject).alpha = value as Number;
}
```

这里 target 的 alpha 属性设置成 Tween 变量返回的值，慢慢的将 target 的 alpha 属性值变化为 finalValue 变量值：

```
package oreilly.cookbook
{
    import flash.display.DisplayObject;
    import mx.effects.effectClasses.TweenEffectInstance;

    public class CustomTweenInstance extends TweenEffectInstance
    {

        public var finalAlpha:Number;

        public function NewTweenInstance(target:Object) {

```

```

        super(target);
    }

    override public function play():void {
        super.play();
        this.tween = new Tween(this, 0, finalAlpha, duration);
        (target as DisplayObject).alpha = 0;
    }

    override public function onTweenUpdate(value:Object):void {
        (target as DisplayObject).alpha = value as Number;
    }
}
}

```

每次调用 onTweenUpdate 的时候，都要重新计算 alpha 值并更新 target。

## 12.7 节 在 Flex Effect 里使用 DisplacementMapFilter 过滤器

### 12.7.1 问题

你需要创建一个在图片之间置换的渐变效果。

### 12.7.2 解决办法

继承 TweenEffect 和 TweenEffectInstance 类，创建一个具有最终置换值的 TweenEffect 实例，然后将这些最终置换值传给它所创建的 TweenEffectInstance 类实例。在自定义的 TweenEffectInstance 类里，创建一个 DisplacementMapFilter 对象并且使用 Flex 框架的渐变引擎通过在每个 onTweenUpdate 事件上生成新的过滤器来达到预期的置换值。

### 12.7.3 讨论

DisplacementMapFilter 通过使用另一张图片的像素点决定变形的位置和量来置换或者变形当前图片的像素点。

置换的位置和量到设定的像素是通过置换目标图片的像素点色值决定的。

DisplacementMapFilter 的构造方法如下：

Code View:

```

public function DisplacementMapFilter(mapBitmap:BitmapData = null,
mapPoint:Point = null, componentX:uint = 0, componentY:uint = 0,

```

```
scaleX:Number = 0.0, scaleY:Number = 0.0, mode:String = "wrap",
color:uint = 0, alpha:Number = 0.0)
```

如此长的一行代码拆开理解较为简单：

#### [BitmapData \(default = null\)](#)

这是过滤器应用到的置换目标图片或组件所用到的 BitmapData 对象。

#### [mapPoint](#)

这个是被过滤图片的位置，对应于过滤器要应用到的置换图片的左上角位置。如果仅仅过滤图片的一部分的话，可以使用这个参数。

#### [componentX](#)

该参数指定作用 x 位置上的图片象素颜色通道。BitmapDataChannel 定义了所有有效的常量选项值：BitmapDataChannel.BLUE 或 4, BitmapDataChannel.RED 或 1, BitmapDataChannel.GREEN 或 2, or BitmapDataChannel.ALPHA 或 8。

#### [componentY](#)

指定作用 y 位置上的图片象素颜色通道。取值范围和 componentX 的相同。

#### [scaleX](#)

这个参数值指定 X 轴方向上的置换强度。

#### [scaleY](#)

这个参数值指定 Y 轴方向上的置换强度。

#### [mode](#)

这是一个字符串，它决定怎样处理置换像素后形成的空白空间。可选值申明为 DisplacementMapFilterMode 类的常量，用以显示原始的像素(mode = IGNORE)、从图片另一边封装边缘像素点(mode = WRAP,默认值)、使用最近的置换像素(mode = CLAMP)或者使用某个颜色填充这些空间(mode = COLOR)。

CustomDisplacementEffect 例示 CustomDisplacementInstance。如下：

Code View:

```
package oreilly.cookbook
{
    import mx.effects.IEffectInstance;
```

```

import mx.effects.TweenEffect;
import mx.events.EffectEvent;

public class CustomDisplacementEffect extends TweenEffect
{

    public var image:Class;
    public var yToDisplace:Number;
    public var xToDisplace:Number;

    public function CustomDisplacementEffect(target:Object=null)
    {
        super(target);
        this.instanceClass = CustomDisplacementInstance;
    }

    override protected function
    initInstance(instance:IEffectInstance) :void {
        trace(" instance initialized ");
        super.initInstance(instance);
        // now that we've instantiated our instance, we can set its properties
        CustomDisplacementInstance(instance).image = image;
        CustomDisplacementInstance(instance).xToDisplace =
            this.xToDisplace;
        CustomDisplacementInstance(instance).yToDisplace =
            this.yToDisplace;
    }

    override public function getAffectedProperties():Array {
        trace(" return all the target properties ");
        return [];
    }
}
}

```

实际上 CustomDisplacementInstance 负责进行创建应用到目标的 DisplacementEffect 对象。而位图对象，过滤器在 DisplacementEffect 使用的，以及 CustomDisplacementTween 的 x 与 y 置换量都应用到该实例并传给 DisplacementEffect。

CustomTweenEffect 创建 CustomDisplacementInstance 的实例，如下：

Code View:

```

package oreilly.cookbook
{
    import flash.display.BitmapData;
    import flash.display.BitmapDataChannel;
    import flash.display.DisplayObject;
    import flash.filters.DisplacementMapFilter;
    import flash.filters.DisplacementMapFilterMode;
    import flash.geom.Point;

    import mx.effects.Tween;
    import mx.effects.effectClasses.TweenEffectInstance;

    public class CustomDisplacementInstance extends
        TweenEffectInstance
    {

        public var image:Class;
        public var xToDisplace:Number;
        public var yToDisplace:Number;
        public var filterMode:String =
            DisplacementMapFilterMode.WRAP;

        private var filter:DisplacementMapFilter;
        private var img:DisplayObject;
        private var bmd:BitmapData;

        public function CustomDisplacementInstance(target:Object)
        {
            super(target);
        }

        override public function play():void {
            super.play();
            //make our embedded image accessible to use
            img = new image();
            bmd = new BitmapData(img.width, img.height, true);
            //draw the actual byte data into the image
            bmd.draw(img);
        }
    }
}

```

这个新过滤器被创建，将被设置初始状态的所有值：

Code View:

```

filter = new DisplacementMapFilter(bmd, new
    Point(DisplayObject(target).wi
        dth/2 - (img.width/2), DisplayObject(target).height/2 -
        (img.height/2))),
    BitmapDataChannel.RED, BitmapDataChannel.RED, 0, 0,
filterMode, 0.0, 1.0);
    //copy any filters already existing on the target so
    //that we don't
    destroy them when we add our new filter
    var targetFilters:Array = (target as
        DisplayObject).filters;
    targetFilters.push(filter);
    //set the actual filter onto the target
    (target as DisplayObject).filters = targetFilters;
    //create a tween that will begin to generate the next
    values of each frame of our effect
    this.tween = new Tween(this, [0, 0], [xToDisplace,
yToDisplace],
duration);

}

```

该类的很多繁重工作都在 setDisplacementFilter 中完成。因为过滤器是累积的(它们是叠加应用的), 前面的 DisplacementMapFilter 必须移除。这需要通过循环遍历目标对象的过滤器数组来完成, 移除所有 DisplacementMapFilter 的实例。然后使用 Tween 传过来的值创建一个新的过滤器并且将此过滤器应用到目标对象上。注意要让过滤器适当的显示, 过滤器数组必须要重置。使用 Array.push 方法添加过滤器到数组中不会引起目标 DisplayObject 使用过滤器重绘。

Code View:

```

private function setDisplacementFilter(displacement:Object):void
{
    var filters:Array = target.filters;
    // Remove any existing Displacement filter to ensure that
    // ours is the only one
    var n:int = filters.length;
    for (var i:int = 0; i < n; i++) {
        if (filters[i] is DisplacementMapFilter)
            filters.splice(i, 1);
    }
    //create the new filter with the values passed in from
    //the tween

```

```

        filter = new DisplacementMapFilter(bmd, new Point(0, 0),
            BitmapDataChannel.RED, BitmapDataChannel.RED,
            displacement[0] as Number, displacement[1] as
            Number, filterMode, 0.0, 0);
        //add the filter to the filters on the target
        filters.push(filter);
        target.filters = filters;
    }

    //each time we're ready to update, re-create the
displacement map filter
override public function onTweenUpdate(value:Object):void
{
    setDisplacementFilter(value);
}

//set the filter one last time and then dispatch the tween
end event
override public function onTweenEnd(value:Object):void
{
    setDisplacementFilter(value);

    super.onTweenEnd(value);
}
}
}

```

当渐变结束时，DisplacementMapFilter 的最终值用来设置目标 DisplayObject 的最终外观，同时调用 TweenEffectInstance 类的 onTweenEnd 方法。

## 12.8 节 创建 AnimateColor 特效

Contributed by Darron Schall

Darron Schall 创作

### 12.8.1 问题

我想平滑地过渡颜色。

## 12.8.2 解决办法

使用自定义效果，AnimateColor，获得一个在两个色值间平滑过渡。

## 12.8.3 讨论

使用 AnimateProperty 效果在颜色之间转换会因为色值本身的性质造成闪烁的问题。

AnimateProperty 效果平滑地在 fromValue 和 toValue 之间转换，但是仅限于值为数字的情况下。例如，AnimateProperty 可以出色地将组件从 x 等于 10 位置平滑地移动到 x 等于 100 位置。

另一方面，颜色并不是真正的数字。它们由数值表达，例如十六进制 0xFFCC33 或十进制 16763955，但每个数字都由三个独立的数字(颜色通道)组成：红，绿，蓝。每个通道取值范围为从 0 到 255 (0xFF)。0xFF0000 表示纯红，0x00FF00 表示纯绿，0x0000FF 表示纯蓝。于是，色值 0x990099 即为由部分红加部分蓝混合而成。

要平滑地转换一个颜色，必须单独地看颜色的三个部分。例如，如果颜色从 0x000000 过渡到 0xFF0000，转换应该仅应用在 red 通道上。但是，AnimateProperty 则把这个转换看作从 0 到 16711680。数字 255 因恰好在开始和结束值之间而可能在过渡过程中显示出来，但是 255 代表的是 0x0000FF 纯蓝。因此，纯蓝会出现在过渡过程中，但是在此强调一下，在这里仅需要从 red 通道过渡即可。

基于这种理解，从一个颜色到另一个颜色的过渡需要单独在每个通道的基础上完成，于是创建一个新的名为 AnimateColor 的效果。

像下面这样使用 AnimateColor 效果：

```
<ds:AnimateColor xmlns:ds="com.darronschall.effects.*"  
    id="fadeColor"  
    target="{someTarget}"  
    property="backgroundColor" isStyle="true"  
    fromValue="0xFF0000"  
    toValue="0x00FF00"  
    duration="4000" />
```

一个 AnimateColor 效果计算从 fromValue 到 toValue 间不同值的每个颜色通道。每一步过渡，都要先考虑每个独立通道需要变化多少，然后生成一个新的的颜色值。整体效果会形成一个平滑的颜色过渡。在前面代码例子中，someTarget 的 backgroundColor 在 4 秒内从纯红过渡到纯蓝，在过渡过程中会显示一点点紫色。

## 12.9 节 使用 Convolution Filter 创建渐变效果

### 12.9.1 问题

我想要使用 ConvolutionFilter 在 MXML 组件上创建一个 TweenEffect。

### 12.9.2 解决办法

创建一个 TweenEffectInstance 类，在 onTweenUpdate 事件的回调方法中实例化新的 ConvolutionFilter 实例并将这些 ConvolutionFilter 实例分配到目标 DisplayObject 的过滤器数组。

### 12.9.3 讨论

ConvolutionFilter 用一种柔和的方式变换它的目标 DisplayObject 或 BitmapImage，允许效果的创建进行例如模糊，边缘检测，打磨，浮雕和斜角。源图片的每个象素都会根据周围的像素值进行转化。每个象素的转化都由传入 ConvolutionFilter 构造器的 Matrix 数组参数来决定。ConvolutionFilter 的构造器申明如下：

Code View:

```
public function ConvolutionFilter(matrixX:Number = 0, matrixY:Number = 0, matrix:Array  
= null, divisor:Number = 1.0, bias:Number = 0.0, preserveAlpha:Boolean = true,  
clamp:Boolean = true, color:uint = 0, alpha:Number = 0.0)
```

一点一点地仔细看看：

**matrixX:Number (default = 0)**

这个数字代表矩阵的列数。

**matrixY:Number (default = 0)**

这个指定了矩阵的行数。

**matrix:Array (default = null)**

这个数组的值用以决定如何转化每个象素。数组内元素的数字个数应与 matrixX 乘以 matrixY 的结果相同。

**divisor:Number (default = 1.0)**

该参数指定矩阵转化和 ConvolutionFilter 如何应用矩阵计算的约数。如果对矩阵值求和，总和将会是一个平均分配色彩强度的约数。

**bias:Number (default = 0.0)**

该参数是对矩阵变换结果添加的偏好设置参数。

**preserveAlpha:Boolean (default = true)**

值为 false 表明不保存 alpha 值并且 convolution 会应用到所有通道，包括 alpha 通道。True 值则表示 convolution 仅仅应用到颜色通道。

**clamp:Boolean (default = true)**

针对源图片之外的像素，值为 true 表示会在输入图片边缘处通过复制边缘处的颜色来扩展图片的边框。值为 false 表示会在指定 color 和 alpha 属性时使用该色值。默认值是 true。

**color:uint (default = 0)**

十六进制颜色数值，填充源图片周围的边框像素颜色。

**alpha:Number (default = 0.0)**

填充颜色的透明度。

此处的 TweenEffect 类像 12.8 节当中的 TweenEffect 类一样，在它所使用的 TweenEffectInstances 类里面使用 ConvolutionFilter。

```
package oreilly.cookbook
{
    import mx.effects.IEffectInstance;
    import mx.effects.TweenEffect;

    public class ConvolutionTween extends TweenEffect
    {
```

创建新EffectInstance时可能会传入的值如下：

```
public var toAlpha:Number;
public var fromAlpha:Number;

public var toColor:uint;
public var fromColor:uint;

public var fromMatrix:Array;
public var toMatrix:Array;
```

```

public var toDivisor:Number;
public var fromDivisor:Number;

public var toBias:Number;
public var fromBias:Number;

public function ConvolutionTween(target:Object=null)
{
    super(target);
    this.instanceClass = ConvolutionTweenInstance;
}

```

下面是每个新建 ConvolutionTweenInstance 类实例属性设置:

Code View:

```

override protected function
initInstance(instance:IEffectInstance):void {
    trace(" instance initialized ");
    super.initInstance(instance);
    // now that we've instantiated our instance, we can set its properties
    ConvolutionTweenInstance(instance).toAlpha = toAlpha;
    ConvolutionTweenInstance(instance).fromAlpha =
        fromAlpha;

    ConvolutionTweenInstance(instance).toColor = toColor;
    ConvolutionTweenInstance(instance).fromColor =
        fromColor;

    ConvolutionTweenInstance(instance).fromMatrix =
        fromMatrix;
    ConvolutionTweenInstance(instance).toMatrix = toMatrix;

    ConvolutionTweenInstance(instance).toDivisor =
        toDivisor;
    ConvolutionTweenInstance(instance).fromDivisor =
        fromDivisor;

    ConvolutionTweenInstance(instance).toBias = toBias;
    ConvolutionTweenInstance(instance).fromBias = fromBias;

}

override public function getAffectedProperties():Array {
    trace(" return all the target properties ");
}

```

```

        return [];
    }
}

}

```

ConvolutionTweenInstance 从 ConvolutionEffect 工厂类那里获取它的目标对象和属性值。

```

package oreilly.cookbook
{
    import flash.filters.ConvolutionFilter;

    import mx.effects.Tween;
    import mx.effects.effectClasses.TweenEffectInstance;

    public class ConvolutionTweenInstance extends
        TweenEffectInstance
    {

        private var convolutionFilter:ConvolutionFilter;

        public var toAlpha:Number;
        public var fromAlpha:Number;

        public var toColor:uint;
        public var fromColor:uint;

        public var fromMatrix:Array;
        public var toMatrix:Array;

        public var toDivisor:Number;
        public var fromDivisor:Number;

        public var toBias:Number;
        public var fromBias:Number;

        public function ConvolutionTweenInstance(target:Object) {
            super(target);
        }
    }
}

```

在覆盖的 play 方法中，每一个起始值（代表着 ConvolutionFilter 的初始化值）和终止值（代表着应用到目标的 ConvolutionFilter 的终点值），传入 Tween 实例。

Code View:

```
override public function play():void {

    this.tween = new Tween(this,
        [fromMatrix[0], fromMatrix[1], fromMatrix[2],
        fromMatrix[3], fromMatrix[4], fromMatrix[5],
        fromMatrix[6], fromMatrix[7], fromDivisor,
        fromBias, fromAlpha, fromColor],
        [toMatrix[0], toMatrix[1], toMatrix[2], toMatrix[3],
        toMatrix[4], toMatrix[5], toMatrix[6], toMatrix[7], ,
        toDivisor, toBias, toAlpha, toColor], duration);
    convolutionFilter = new ConvolutionFilter(fromMatrixX,
        fromMatrixY, fromMatrix, 1.0, 0, true, true,
        fromAlpha, fromColor);
}
```

每个来自 Tween 的新值都作为对象传入 onTweenUpdate。在该对象内，0 起始的数组里，存储了从初始化状态到结束状态之间任一给定时间点的转换状态值。因为 ConvolutionFilter 需要一个数组作为参数，数组里面的每个值都是中间状态值，然后作为 ConvolutionFilter 的矩阵参数传入一个新数组。

Code View:

```
override public function onTweenUpdate(value:Object):void {
    //get the filters from the target
    var filters:Array = target.filters;
    // Remove any existing Displacement filter to ensure
    // that ours is the only one
    var n:int = filters.length;
    for (var i:int = 0; i < n; i++) {
        if (filters[i] is ConvolutionFilter)
            filters.splice(i, 1);
    }
    //create the new filter
    convolutionFilter = new ConvolutionFilter(3, 3,
        [value[0], value[1], value[2], value[3], value[4], value[5],
        value[6], value[7]], value[8], value[9] true, true, value[10],
        value[11]);
    //add the filter to the target
    filters.push(convolutionFilter);
    target.filters = filters;

}
```

}

注：所有应用到目标对象上的 ConvolutionFilters 都要移除。如果不这样做，多个 ConvolutionFilters 的效果会叠加地应用到目标对象上去，造成与预期有很大差别的效果。

# 第十三章. 集合 (常青)

集合是 ActionScript 中功能强大的基于索引的数组组件，添加了如对内容进行排序等功能，操作数组的读取位置，创建经过排序的数组视图。集合也能通知其任意事件监听器监听其数据是否改变，以及任何数据项被添加到源数组时可执行自定义逻辑。当数据改变时可知其监听器，这是集合的新功能，叫数据绑定，还有就是允许 DataGrid 和 List 组件对其内容进行排序和过滤。集合是使用数据驱动控件以及从数据库返回的服务器端服务的重要内容。

经常被使用的两个集合类型是 ArrayCollection 类 XMLListCollection 类。 ArrayCollection 是 Array 的包装类，提供更方便的如添加和移除数据项以及能够创建游标启用轻松地存储数组中的最后读取的位置等方法。而 XMLListCollection 是 XML 对象的包装类，提供的功能有：根据索引访问数据，添加新对象以及游标等方法。 XMLListCollection 对于处理 XML 对象以及经常需要解析 XML 为数组时特别有用。

## 13.1节. 为 **ArrayCollection** 添加，排序和获取数据

### 13.1.1. 问题

我需要添加新数据到 ArrayCollection 以及从同一个 ArrayCollection 中获取某个数据。

### 13.1.2. 解决办法

创建 ArrayCollection，使用 addItemAt 或 addItem 方法插入对象到 ArrayCollection， getItemIndex 或 contains 方法用于检测数据项是否已存在于数组中，而 ArrayCollection 的 sort 属性是对 ArrayCollection 排序以及通过某个字段决定接收第一个或最后一个数据。

### 13.1.3. 讨论

为了了解 ArrayCollection 中添加，测试，排序数据项的各种方法是如何工作的，首先创建一个 ArrayCollection，类似如下的代码：

Code View:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300" creationComplete="init()">
<mx:Script>
<! [CDATA[
import mx.collections.SortField;
import mx.collections.Sort;
import mx.collections.ArrayCollection;
```

```

private var coll:ArrayCollection;

private function init():void {
    coll = new ArrayCollection(
        [{name:"Martin Foo", age:25},
         {name:"Joe Bar", age:15},
         {name:"John Baz", age:23}]);
}

```

要插入元素到指定位置，可使用 addItemAt 方法：

```

private function addItem():void {
    coll.addItemAt({name:"James Fez", age:40}, 0);

}

```

要检查 ArrayCollection 中是否已存在复杂对象，你需要比较两个对象的值：

Code View:

```

private function checkExistence():void {
    trace(coll.contains({name:nameTI.text,
                           age:Number(ageTI.text)}));
    trace(coll.getItemIndex({name:nameTI.text,     age:ageTI.text}));
// traces -1 if not present

}

```

但是上面的做法是不对的，因为 contains 和 getItemIndex 方法都是比较对象的指针，而不是他们的值。因为两个对象是截然不同的对象，因为存在于内存的不同位置，Flash Player 认为他们是不相等的。即便 ArrayCollection 包含该元素，但 getItemIndex 方法也不会返回其索引值。要检测元素是否具有相同的值，你需要和所有的数据对象进行值比较：

```

private function checkExistence():int {
    var i:int;
    var arr:Array = coll.souce;
    while(i < arr.length) {
        if(arr[i].name == nameTI.text && arr[i].age ==
            ageTI.text) {
            return i;
        }
        i++;
    }
    return -1;
}

```

```
}
```

Sort 对象提供 `findItem` 方法用于搜索这个 ArrayCollection 中的所有元素。方法原型如下：

```
public function findItem(items:Array, values:Object, mode:String,  
returnInsertionIndex:Boolean = false, compareFunction:Function = null):int
```

`Value` 参数可以是包含属性和所需值的任何对象。`Mode` 字符串可以是 `Sort.ANY_INDEX_MODE`, 表示返回任何匹配项索引, `Sort.FIRST_INDEX_MODE` 表示返回第一个匹配项索引, `Sort.LAST_INDEX_MODE` 表示返回最后一个匹配项索引。`returnInsertionIndex` 参数表示如果该方法找不到由 `values` 参数标识的项目, 并且此参数为 `true`, 则 `findItem()` 方法将返回这些值的插入点, 也就是排序顺序中应插入此项目的。`compareFunction` 设置用于查找该项目的比较运算符函数.

使用 Sort 对象的 `findItem` 方法代替上面的方法:

Code View:

```
private function checkExistence():int {  
    var sort:Sort = new Sort();  
    return sort.findItem(coll.source,  
        {name:nameTI.text, age:Number(ageTI.text)},  
        Sort.ANY_INDEX_MODE);  
  
}
```

首先要创建一个 Sort, 传递一个 SortField 对象数组给 `fields` 属性。这些 SortField 对象包含的字符串正是每个 ArrayCollection 元素将要用来排序的属性。如要对每个对象的 `age` 属性进行排序, 创建 Sort 对象, 传递 SortField, 设置排序字段为 `age`:

```
private function getOldest():void {  
    var sort:Sort = new Sort();  
    sort.fields = [new SortField("age", false)];  
    coll.sort = sort;  
    coll.refresh();  
    trace(coll.getItemAt(0).age+" "+coll.getItemAt(0).name);  
  
}
```

这个排序函数对 `age` 值从低向高排序。

## 13.2节. 过滤 ArrayCollection

### 13.2.1. 问题

我需要对 ArrayCollection 数据进行过滤，设置过滤器移除些不匹配的结果。

### 13.2.2. 解决办法

J 将原型为 function(item:Object):Boolean 的函数传递给 ArrayCollection 的 filter 属性。如果返回 true 表示值继续留在 ArrayCollection，返回 false 表示其值被移除。

### 13.2.3. 讨论

filterFunction 属性是由 ListCollectionView 类定义，它是 ArrayCollection 的父类。当过滤器函数被传递给继承自 ListCollectionView 的任何子类后，这里为 ArrayCollection 对象，应用过滤器后必须调用 refresh 方法：

Code View:

```
import mx.collections.ArrayCollection;

private var coll:ArrayCollection;

private function init():void {
    coll = new ArrayCollection([
        {name:"Martin Foo", age:25},
        {name:"Joe Bar", age:15},
        {name:"John Baz", age:23},
        {name:"Matt Baz", age:21}]);
    coll.filterFunction = filterFunc;
    coll.refresh();
    for(var i:int = 0; i<coll.length; i++) {
        trace(coll.getItemAt(i).name);
    }
}

private function filterFunc(value:Object):Object {
    if(Number(value.age) > 21) {
        return true;
    }
    return false;
}
```

这里需要注意的是 ArrayCollection 元素并没有被过滤器函数所修改，在这个例子中，源数组有4个数据，过滤后仍保持4个数据。

## 13.3节. 确定 ArrayCollection 数据项是否被修改

### 13.3.1. 问题

我想检测 ArrayCollection 中的数据项是否被修改。

### 13.3.2. 解决办法

监听 ArrayCollection 类发出的扩展自 EventDispatcher 的事件类型 collectionChange 或 CollectionEvent.COLLECTION\_CHANGE。

### 13.3.3. 讨论

任何时候对集合数据的添加和删除都会引发 CollectionEvent 类型事件 collectionChange。当控件绑定集合后，集合的改变通过此事件来通知绑定器。添加一个事件监听器给集合来监听 COLLECTION\_CHANGE 事件，你可以对集合的数据变化作出处理：

Code View:

```
private var coll:ArrayCollection = new ArrayCollection();  
  
coll.addEventListener(CollectionEvent.COLLECTION_CHANGE, collChangeHandler);
```

CollectionEvent 类定义了些额外属性：

`items : Array`

当数据项被添加时发出此事件， items 属性代表被添加的数据数组，如果是删除时触发的，则代表被删除的数据数组。

`kind`

这是一个字符串代表事件类型，其值可以是 add, remove, replace, or move.

## location

该属性为 items 属性中指定的项目集合中基于零的索引

## oldLocation

如果 kind 的值为 CollectionEventKind.MOVE，则此属性为 items 属性中指定的项目在目标集合中原来位置的从零开始的索引。

默认值为 -1.

使用 CollectionEvent，就可以推断 ArrayCollection 或 XMLListCollection 前后所处状态，在确定服务器上的 Flex 应用程序是否被更新时非常有用。

## 13.4节. 创建 GroupingCollection

### 13.4.1. 问题

我想基于集合中数据项的某个属性创建唯一的分组数据

### 13.4.2. 解决办法

传递一个 Array 参数给 GroupingCollection 构造器或设置 GroupingCollection 对象的 source 属性。

### 13.4.3. 讨论

任何 GroupingCollection 都有一个 Grouping 对象实例，其拥有对应的 GroupingField，定义了将用来生成分组数据的数据对象属性。你可以根据对象属性使用 GroupingCollection 进行数据分组。要对数据对象的 state 和 region 属性进行分组，具体如下：

```
var groupingColl:GroupingCollection = new GroupingCollection();

groupingColl.source = [{city:"Columbus", state:"Ohio", region:"East"},

{city:"Cleveland", state:"Ohio", region:"East"}, {city:"Sacramento",

state:"California", region:"West"}, {city:"Atlanta", state:"Georgia",
```

```
region:"South"}];
```

对 state 属性进行分组，首先创建具有相同 state 的所有对象分组，创建 Grouping 实例，传递 GroupingField 对象数组给 Grouping 实例的 fields：

```
var groupingInst:Grouping = new Grouping();

groupingInst.fields = [new GroupingField("state")];

groupingColl.grouping = groupingInst;

groupingColl.refresh(false);
```

Grouping 实例初始化后，设置 GroupingCollection 的 grouping 属性为 groupingInst，然后刷新，集合将根据 state 属性值对集合所有数据进行分组：

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml"
width="400" height="300" creationComplete="init()">
<mx:Script>
<! [CDATA[
import mx.collections.Grouping;
import mx.collections.GroupingField;
import mx.collections.GroupingCollection;
[Bindable]
private var groupingColl:GroupingCollection;

private function init():void {
    groupingColl = new GroupingCollection();

    groupingColl.source = [
        {city:"Columbus",state:"Ohio",region:"East"},
        {city:"Cleveland", state:"Ohio", region:"East"},
        {city:"Sacramento",state:"California",region:"West"},
        {city:"Atlanta",state:"Georgia", region:"South"}];
    var groupingInst:Grouping = new Grouping();
    groupingInst.fields = [new GroupingField("state")];
    groupingColl.grouping = groupingInst;
    groupingColl.refresh(false);
}
```

设置好GroupingCollection的grouping属性后，设置另外的分组将会覆盖当前分组：

```
private function createRegionGrouping():void {
    var groupingInst:Grouping = new Grouping();
    groupingInst.fields =
        [new GroupingField("region")];
    groupingColl.grouping = groupingInst;
```

```

        groupingColl.refresh(false);
    }
] ]>
</mx:Script>
<mx:AdvancedDataGrid dataProvider="{groupingColl}">
<mx:columns>
    <mx:AdvancedGridColumn dataField="city"/>
</mx:columns>
</mx:AdvancedDataGrid>
<mx:Button click="createRegionGrouping()"/>

</mx:VBox>

```

要想传递多个分组，只要传递多个 GroupingField 对象即可。

```
groupingInst.fields = [new GroupingField("region"), new GroupingField("state")];
```

这将会先对 region 分组字段分组，在对 state 字段分组。

## 13.5节. 为控件创建层级数据供应器

### 13.5.1. 问题

我想用平面对象(对象没有父子关系)表示层级数据，作为 DataGrid 的 dataProvider。

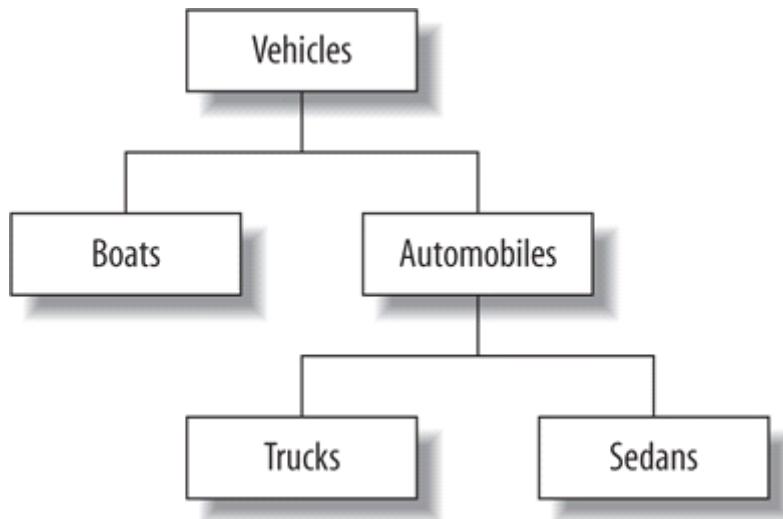
### 13.5.2. 解决办法

创建实现 IHierarchicalData 接口的自定义数据类，创建方法用于检测是否是节点或对象有父节点以及是否有子节点。

### 13.5.3. 讨论

IHierarchicalData 接口定义了 DataGrid 和 AdvancedDataGrid 组件显示层级数据所需的所有方法。层级数据表示的是数据的父子关系。例如各种类型的交通工具—汽车，卡车，轮船—每种里面包含更多类型的交通工具。其中轿车的层次应该是这样：

**Figure 13-1. An object hierarchy**



下面的就是数据对象表示方式：

Code View:

```

private var data:Object = [{name:"Vehicles", id:1, parentId:0, type:"parent"},

{name:"Automobiles", id:2, parentId:1, type:"parent"},

{name:"Boats", id:3, parentId:0, type:"parent"},

{name:"Trucks", id:4, parentId:1, type:"parent"},

{name:"Sedans", id:5, parentId:2, type:"parent"}];
  
```

这里给每个节点分配一个 id 和表示父节点的 parentId。这一类型的数据结构一般很难处理。这里的方法是使用 IHierarchicalData 接口；有了它，AdvancedDataGrid 以分组数据显示，Tree 控件以数据树显示。IHierarchicalData 接口定义了下列方法：

**canHaveChildren(node:Object):Boolean**

此方法确定指定的节点是否具有子节点

**dispatchEvent(event:Event):Boolean**

此方法发出事件

**getChildren(node:Object):Object**

此方法返回指定节点的所有子节点

**getData(node:Object):Object**

此方法返回指定节点的搜所有数据，包括子节点

**getParent(node:Object):\***

此方法返回任意节点的父节点

**getRoot():Object**

此方法返回层级数据的根节点

**hasChildren(node:Object):Boolean**

此方法确认指定节点是否具有子节点

下面的 ObjectHierarchicalData 类实现了实现上面这些方法:

Code View:

```
package oreilly.cookbook
{
    import flash.events.EventDispatcher;
    import mx.collections.IHierarchicalData;

    public class ObjectHierarchicalData extends EventDispatcher
    implements IHierarchicalData
    {
        private var source:Object;
        public function ObjectHierarchicalData(value:Object)
        {
            super();
            source = value;
        }
        /* in our simple system, only parents with their type set
        to 'parent' can have children; otherwise, they can't have
        children */
        public function canHaveChildren(node:Object):Boolean
        {
            if (node.type == "parent") {
                return true;
            }
            return false;
        }

        /* for any given node, determine whether a node has any
        children by looking through all the other nodes for that node's
        ID as a parentTask */
        public function hasChildren(node:Object):Boolean
        {
            trace(node.name);
            var children:Array = new Array(); // = source.parentTask
```

```

== parentId);
    for each(var obj in source) {
        if(obj.parentTask == node.id) {
            children.push(obj);
        }
    }
    if (children.length > 0)
        return true;

    return false;
}
/* for any given node, return all the nodes that are
children of that node in an array */
public function getChildren(node:Object):Object
{
    var parentId:String = node.id;
    var children:Array = new Array();
    for each(var obj in source) {
        if(obj.parentTask == parentId) {
            children.push(obj);
        }
    }
    return children;
}

public function getData(node:Object):Object
{
    for each(var obj in source) {
        for(var prop in node) {
            if(obj[prop] == node[prop]) {
                return obj;
            } else {
                break;
            }
        }
    }
    return null;
}
/* we want to return every obj that is a root object
which, in this case, is going to be all nodes that have a
parent node of '0' */
public function getRoot():Object
{
    var rootsArr:Array = new Array();

```

```

        for each(var obj in source) {
            if(obj.parentTask == "0") {
                rootsArr.push(obj);
            }
        }
        return rootsArr;
    }

    public function getParent(node:Object):*
    {
        for each(var obj in source) {
            if(obj.parentTask == node.parentTask) {
                return obj;
            }
        }
        return null;
    }
}
}

```

现在这些方法都能确定数据对象中所有节点之间的关系。你可以建立一个新的层级数据类给 AdvancedDataGrid 的 dataProvider。使得控件能正确显示层级数据。

Code View:

```

<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300" creationComplete="init()">
<mx:Script>
<! [CDATA[
import mx.collections.*;

[Bindable]
private var ohd:ObjectHierarchicalData;
/* here's the huge object that we're going to use to
populate our ObjectHierarchicalData object */
private var largeObject:Object =
[
{
    "id":"1", "name":"Misc", "type":"parent",
    "parentTask":"0"}, {
    "id":"2", "name":"Clean the kitchen", "type":"parent",
    "parentTask":"0"}, {
    "id":"3", "name":"Pay the bills", "type":"parent",
    "parentTask":"0"}, {
    "id":"4", "name":"Paint the shed", "type":"parent",
    "parentTask":"0"}];

```

```

        "parentTask": "1"} ,
    {"id": "5", "name": "Get ready for party",
     "type": "parent",
     "parentTask": "1"} ,
    {"id": "6", "name": "Do the dishes", "type": "child",
     "parentTask": "2"} ,
    {"id": "7", "name": "Take out trash", "type": "child",
     "parentTask": "2"} ,
    {"id": "8", "name": "Gas Bill", "type": "child",
     "parentTask": "3"} ,
    {"id": "9", "name": "Registration", "type": "child",
     "parentTask": "3"} ,
    {"id": "10", "name": "Fix the car", "type": "parent",
     "parentTask": "0"} ,
    {"id": "11", "name": "New tires", "type": "child",
     "parentTask": "10"} ,
    {"id": "12", "name": "Emissions test", "type": "child",
     "parentTask": "10"} ,
    {"id": "13", "name": "Get new paint", "type": "child",
     "parentTask": "4"} ,
    {"id": "14", "name": "Buy brushes", "type": "child",
     "parentTask": "4"} ,
    {"id": "15", "name": "Buy Drinks", "type": "child",
     "parentTask": "5"} ,
    {"id": "16", "name": "clean living room", "type": "child",
     "parentTask": "5"} ,
    {"id": "16", "name": "finish invitations", "type": "child",
     "parentTask": "5"} ];
```

**private function** init():**void** {  
 ohd = **new** ObjectHierarchicalData(largeObject);  
}

]

</mx:Script>  
<mx:AdvancedDataGrid dataProvider="**{ohd}**" width="300"  
height="200">  
<mx:columns>  
<!-- all we want to display of the object is the name,<br/>
the ADG will take care of displaying the parent child  
relationship -->  
<mx:AdvancedDataGridColumn dataField="name"/>  
</mx:columns>  
</mx:AdvancedDataGrid>

```
</mx:Canvas>
```

## 13.6节. 遍历集合对象并记录位置

### 13.6.1. 问题

我想双向遍历集合，并保持当前所在位置

### 13.6.2. 解决办法

使用 ListViewCollection 类的 createCursor 方法创建可前后移动的游标。

### 13.6.3. 讨论

可使用视图游标浏览集合数据视图中所有数据项，访问和修改集合数据。游标是一个位置指示器，它执行特定位置的数据项。你可以使用集合的 createCursor 方法返回一个视图游标。游标的各种方法和属性都由 IViewCursor 接口定义。

通过 IViewCursor 方法，你可以前后移动游标，用特定条件搜索数据项，获取指定位置的数据项，保存游标所在位置，以及添加，删除或修改数据值。

当你使用标准的 Flex 集合类如 ArrayCollection 和 XMLListCollection 时，可直接使用 IViewCursor 接口，不需要引用对象实例，例如：

Code View:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300" creationComplete="init()">
<mx:Script>
<! [CDATA[
import mx.collections.SortField;
import mx.collections.Sort;
import mx.collections.IViewCursor;
import mx.collections.CursorBookmark;
import mx.collections.ArrayCollection;
[Bindable]
private var coll:ArrayCollection;
[Bindable]
private var cursor:IViewCursor;

private function init():void {
    coll = new ArrayCollection([
        {city:"Columbus", state:"Ohio", region:"East"},
```

```

        {city:"Cleveland", state:"Ohio", region:"East"},
        {city:"Sacramento", state:"California", region:"West"},
            {city:"Atlanta", state:"Georgia", region:"South"}]);
cursor = coll.createCursor();
}

```

这个例子中, IViewCursor对象的findFirst方法根据文本框中的数据定位第一个匹配的集合数据:

Code View:

```

private function findRegion():void {
    var sort:Sort = new Sort();
    sort.fields = [new SortField("region")];
    coll.sort = sort;
    coll.refresh();
    cursor.findFirst({region:regionInput.text});
}

private function findState():void {
    var sort:Sort = new Sort();
    sort.fields = [new SortField("state")];
    coll.sort = sort;
    coll.refresh();
    cursor.findFirst({region:stateInput.text});
}

]]>
</mx:Script>
<mx:Label text="{cursor.current.city}" />
<mx:Button click="cursor.moveToNext()" label="Next"/>
<mx:Button click="cursor.moveToPrevious()" label="Previous"/>
<mx:HBox>
    <mx:TextInput id="regionInput"/>
    <mx:Button click="findRegion()" label="find region"/>
</mx:HBox>
<mx:HBox>
    <mx:TextInput id="stateInput"/>
    <mx:Button click="findRegion()" label="find state"/>
</mx:HBox>

</mx:VBox>

```

IViewCursor 接口定义了三个方法用于搜索集合:

**findFirst(values:Object):Boolean**

此方法查找集合中具有指定属性的第一个项目，并将光标定位到该项目

### `findLast(values:Object):Boolean`

此方法查找集合中具有指定属性的最后一个项目，并将光标定位到该项目

### `findAny(values:Object):Boolean`

此方法查找集合中具有指定属性的项目并将光标定位到该项目，这个是速度最快的方法。

注意这三个方法可工作在未排序的 ArrayCollection 或 XMLListCollection。

## 13.7节. 创建 **HierarchicalViewCollection** 对象

### 13.7.1. 问题

我想创建一个集合，它能让我把 IHierarchicalData 对象当作集合处理。

### 13.7.2. 解决办法

创建一个实现 IHierarchicalData 接口的类用于检测每个节点的父节点和子节点。创建一个 HierarchicalCollectionView 对象，传递 IHierarchicalData 对象作为 HierarchicalCollectionView 类构造器的参数。

### 13.7.3. 讨论

默认情况下，要使用 HierarchicalData，AdvancedDataGrid 需创建一个 HierarchicalCollectionView。HierarchicalCollectionView 允许 AdvancedDataGrid 接收一个 ArrayCollection，应用所有方法到 HierarchicalData。这不仅仅对于 AdvancedDataGrid 很有用，而且对于使用自定义组件显示层级数据时也很有用。第13.5节 实现 IHierarchicalData 接口的 ObjectHierarchicalData 类提供检测不同节点之间的父节点和子节点关系。HierarchicalCollectionView 类使用这些方法更直观的打开和关闭节点，以及检测数据对象是否包含特定的数值。这一节使用 ObjectHierarchicalData 创建 HierarchicalCollectionView 实例对象。

HierarchicalCollectionView 的方法有：

### `addChild(parent:Object, newChild:Object):Boolean`

为数据的节点添加子节点。

### `addChildAt(parent:Object, newChild:Object, index:int):Boolean`

将子节点添加到指定索引处的节点

`closeNode(node:Object):void`

关闭要隐藏其子项的节点

`contains(item:Object):Boolean`

C 使用标准相等测试检查数据项目的集合，这意味着不同内存中的数据即便有相同值测试结果也不会返回 true。

`createCursor():IViewCursor`

返回此视图中有关项目的视图迭代器的新实例。

`getParentItem(node:Object):*`

返回节点的父节点

`openNode(node:Object):void`

打开要显示其子项的节点

`removeChild(parent:Object, child:Object):Boolean`

从父节点删除子节点

`removeChildAt(parent:Object, index:int):Boolean`

从指定索引处的节点删除子节点

确定哪个节点被操作取决于 IHierarchicalData 接口的 getData 方法是否有良好的实现方法。把键值对的对象传入 getData 方法，返回包含相同键值对的节点。HierarchicalCollectionView 可检测源数据对象中哪个对象将被操作。这里定义了层级数据对象，传递一个 HierarchicalData 对象，创建 HierarchicalCollectionView：

Code View:

```
var largeObject:Object =  
    [{ id:"1", name:"Misc", type:"parent", parentTask:"0"},  
     {id:"2",      name:"Clean      the      kitchen",      type:"parent",  
      parentTask:"0"},  
      {id:"3", name:"Pay the bills", type:"parent", parentTask:"0"},  
      {id:"4", name:"Paint the shed", type:"parent", parentTask:"1"},  
      {id:"5", name:"Get      ready      for      party", type:"parent",  
      parentTask:"1"},  
      {id:"6", name:"Do the dishes", type:"child", parentTask:"2"},  
      {id:"7", name:"Take out trash", type:"child", parentTask:"2"},  
      {id:"8", name:"Registration", type:"child", parentTask:"3"},  
      {id:"9", name:"Fix the car", type:"parent", parentTask:"0"},  
      {id:"10", name:"New tires", type:"child", parentTask:"9"},  
      {id:"11", name:"Get new paint", type:"child", parentTask:"4"},  
      {id:"12", name:"Buy Drinks", type:"child", parentTask:"5"},
```

```

{id:"13",      name:"finish      invitations",      type:"child",
parentTask:"5"}];

/* create a new class that implements the IHierarchicalData
interface */
var dataObj:ObjectHierarchicalData =
    new ObjectHierarchicalData(largeObject);

/* pass that class to the HierarchicalCollectionView class*/
var hCollView:HierarchicalCollectionView = new
    HierarchicalCollectionView(dataObj);
hCollView.openNode(largeObject[2]);
var ac:ArrayCollection =
    hCollView.getChildren( hCollView.source.getData({id:"3"}));

hCollView.closeNode(hCollView.source.getData({name:"Pay      the
bills"}));

```

HierarchicalViewCollection 包装了 IHierarchicalData 视图对象，通过 getChildren 方法基于集合对象创建视图。

## 13.8节. 过滤和排序 XMLListCollection

### 13.8.1. 问题

我想过滤和排序 XMLListCollection.

### 13.8.2. 解决办法

使用 XMLListCollection 继承的 ListViewCollection 类的 filterFunction 和 sortFunction 属性或直接传递 Sort 类型对象给 XMLListCollection 的 sort 属性。

### 13.8.3. 讨论

XMLListCollection 用于描述根节点下有多个节点的 XML 数据。例如 nutrition 节点下有 food 节点被翻译为 XMLListCollection 后允许 food 节点可被当作集合处理：

<nutrition>

```

<food>

    <name>Avocado Dip</name>

    <calories>110</calories>

    <total-fat>11</total-fat>

    <saturated-fat>3</saturated-fat>

    <cholesterol>5</cholesterol>

    <sodium>210</sodium>

    <carb>2</carb>

    <fiber>0</fiber>

    <protein>1</protein>

</food>

```

</nutrition>

过滤 XMLListCollection 的方法和 ArrayCollection 一样：传递一个函数引用，该函数接受一个 object，返回 boolean 值，指示对象是否应该留在过滤视图中。例如：

```

coll.filterFunction = lowCalFilter;
private function lowCalFilter(value:Object):Boolean {
    if(Number(value.calories) < 200) {
        return true;
    }
    return false;
}

```

排序 XMLListCollection 需要一个 Sort 对象，其 fields 属性需要一个 SortField 对象数组：

Code View:

```

var sort:Sort = new Sort();
sort.fields = [new SortField("calories", false,
    false, true)];
coll.sort = sort;

```

coll.refresh();完整代码如下：

Code View:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300" creationComplete="xmlService.send()">
<mx:HTTPService url="assets/data.xml" resultFormat="xml"
id="xmlService" result="createXMLCollection(event)"/>
<mx:Script>
<! [CDATA[
    import mx.collections.SortField;
    import mx.collections.Sort;
    import mx.rpc.events.ResultEvent;
    import mx.collections.XMLListCollection;

    [Bindable]
    private var coll:XMLListCollection;

    private function
createXMLCollection(event:ResultEvent):void {
    var list:XMLList = new XMLList(event.result);
    coll = new XMLListCollection(list.food);
    var sort:Sort = new Sort();
    sort.fields = [new SortField("calories", false,
        false, true)];
    coll.sort = sort;
    coll.refresh();
}
private function applyFilter():void {
    coll.filterFunction = lowCalFilter;
    coll.refresh();
}

private function lowCalFilter(value:Object):Boolean {
    if(Number(value.calories) < 200) {
        return true;
    }
    return false;
}
]]>
</mx:Script>
<mx:DataGrid dataProvider="{coll}">
<mx:columns>
    <mx:DataGridColumn dataField="calories"/>
    <mx:DataGridColumn dataField="name"/>
```

```

        </mx:columns>
    </mx:DataGrid>
    <mx:Button click="applyFilter()" label="filter"/>

</mx:VBox>

```

你还可以使用 E4X 语句执行复杂的过滤操作，例如使用@语法访问属性：

```

private function lowFatFilter(value:Object):Boolean {
    if(value.calories(@fat) < Number(value.calories)/5) {
        return true;
    }
    return false;
}

```

## 13.9节. 对集合的多个字段进行排序

### 13.9.1. 问题

我需要根据多个字段对集合进行排序。

### 13.9.2. 解决办法

传递多个 SortField 对象给 Sort 对象，并赋值给集合的 sort 属性。

### 13.9.3. 讨论

因为数组可被多个 SortFields 排序，Sort 对象的 fields 属性就是一个数组。这些 sort 创建一个层级的结构，所有对象进行分组排序，先根据 field 属性里的第一个 SortField 对象，再第一个，以此类推。这个例子代码先根据 regions 再 states 对集合进行排序：

Code View:

```

coll = new ArrayCollection([
    {city:"Cleveland", state:"Ohio", region:"East"},
    {city:"Sacramento", state:"California", region:"West"},
    {city:"Atlanta", state:"Georgia", region:"South"},
    {city:"Columbus", state:"Ohio", region:"East"}
]);
var sort:Sort = new Sort();

```

```
sort.fields = [new SortField("region"), new SortField("state")];
coll.sort = sort;
coll.refresh();
```

Array 集合数据项显示如下 [Figure 13-2](#).

**Figure 13-2.** 多字段数据排序

city	region	state
Cleveland	East	Ohio
Columbus	East	Ohio
Atlanta	South	Georgia
Sacramento	West	California

## 13.10节. 对集合的日期类型数据进行排序

### 13.10.1. 问题

我需要数据对象中存为字符串的日期值属性进行排序。

### 13.10.2. 解决办法

为每个日期属性创建 Date 对象，使用 mx.utils.ObjectUtil 类的 dateCompare 方法比较日期。

### 13.10.3. 讨论

ObjectUtil 类提供一个 dateCompare 方法用于检测两个 Date 对象哪个更早。你可以用 ObjectUtil.dateCompare 方法来比较两个日期的大小。dateCompare 方法返回 1, 0, 或 -1：如果值为 null 或相等返回 0，如果第一个值为 null 或小于第二个值返回 1，如果第二个值

为 null 或小于第一个值返回-1:

Code View:

```
import mx.collections.Sort;
import mx.collections.ArrayCollection;

import mx.utils.ObjectUtil;

//the signature of a sort function must be
//function [name](a:Object, b:Object, fields:Array = null):int
private function sortFunction(a:Object, b:Object, fields:Array =
null):int {
    var tempDateA:Date = new Date(Date.parse(a.dob));
    var tempDateB:Date = new Date(Date.parse(b.dob));
    return ObjectUtil.dateCompare(tempDateA, tempDateB);
}

private var arrColl:ArrayCollection;

private function init():void {
    arrColl = new ArrayCollection([
        {name:"Josh", dob:"08/17/1983"}, 
        {name:"John", dob:"07/30/1946"}, 
        {name:"John", dob:"07/30/1990"}, 
        {name:"John", dob:"07/30/1986"}]);
    var sort:Sort = new Sort();
    sort.compareFunction = sortFunction;
    arrColl.sort = sort;
    arrColl.refresh();
    trace(arrColl);
}
```

## 13.11节. 创建 ArrayCollection 的深度拷贝

### 13.11.1. 问题

我需要拷贝索引数组的所有数据项或对象到新对象上。

### 13.11.2. 解决办法

使用 mx.utils.ObjectUtil.copy 方法。

### 13.11.3. 讨论

为了快速演示，我们简单的拷贝对象引用到新对象上，这意味着对第一个对象的值改变都会反映到第二个对象上：

```
var objOne:Object = {name:"foo", data:{first:"1", second:"2"}};
var objTwo = objOne;
objOne.data.first = "4";

trace(objTwo.data.first); //traces 4
```

要想拷贝对象到另一个对象上，使用 mx.utils.ObjectUtil 类。该方法接受一个对象参数，返回内存新地址上的该对象的深度拷贝。这意味着改变对象的所有属性都不会影响到原始对象，这样使用这个方法：

```
var objTwo = mx.utils.ObjectUtil.copy(objOne);
```

该拷贝方法的工作原理是把原对象转换为 ByteArray，然后再把 ByteArray 写到新对象上，具体如下：

```
var ba:ByteArray = new ByteArray();
ba.writeObject(objToCopy);
ba.position = 0;
var objToCopyInto:Object = ba.readObject();

return objToCopyInto;
```

现在应该能达到我们的目的了：

```
var objOne:Object = {name:"foo", data:{first:"1", second:"2"}};
var objTwo = objOne;
var objThree = mx.utils.ObjectUtil.copy(objOne);
objOne.data.first = "4";

trace(objTwo.data.first); //traces 4

trace(objThree.data.first); //traces 1, which is the original value
```

用这个方法拷贝指定类型的对象却有些困难，如下面的代码将会抛出异常：

```
var newFoo:Foo = ObjectUtil.copy(oldFoo) as Foo;
```

因为 Flash Player 将不知道如何把 ByteArray 转换到指定的类型，通过使用 ByteArray，对象被序列化为 ActionScript Message Format (AMF) 二进制数据，和 Flash Remoting 序列化对象的方式一样。要想反序列化数据对象，改类型必须使用 flash.net.registerClassAlias 方法事先在 Flash Player 中进行注册，该方法注册这个类，以便所有该类型的对象都能被反序列化为该类型的对象。registerClassAlias 方法需要两个参数：

Code View:

```
public function registerClassAlias(  
    aliasName:String, classObject:Class):void
```

第一个参数为类的完全限定类名，第二个参数为对象类型。完全限定类名类似于 mx.containers.Canvas 或 com.oreilly.cookbook.Foo。这个例子中，当拷贝对象时需要先知道类名或类引用。还好 flash.utils.getQualifiedClassName 返回对象的完全限定类名，而 flash.utils.getDefinitionByName 返回类的类对象引用。通过使用这两个方法，你就可以注册任何对象的类了：

Code View:

```
private function copyOverObject(  
    objToCopy:Object, registerAlias:Boolean = false):Object  
{  
    if(registerAlias) {  
        var className:String =  
            flash.utils.getQualifiedClassName(objToCopy);  
        flash.net.registerClassAlias(className,  
            (flash.utils.getDefinitionByName(className) as Class));  
    }  
    return mx.utils.ObjectUtil.copy(objToCopy);  
}
```

现在强类型的 ArrayCollection 对象中每个对象都可正确的拷贝到新对象上：

```
private function copyOverArray(arr:Array):Array {  
  
    var newArray:Array = new Array();  
    for(var i:int; i<arr.length; i++) {  
        newArray.push(copyOverObject(arr[i], true));  
    }  
    return newArray;  
}  
  
var ac:ArrayCollection = new ArrayCollection([{name:'Joseph',  
id:21}, foo, {name:'Josef', id:81}, {name:'Jose', id:214}]);
```

```
var newAC:ArrayCollection = new
    ArrayCollection(copyOverArray(ac.source));
```

注意如果只是简单的使用 mx.utils.ObjectUtil.copy 方法则源 ArrayCollection 中所有对象的数据将会丢失。但是每个对象的类信息还是在的，任何试图转换集合对象为原来类型时结果会是错误的或是 null 值。

## 13.12节. 用唯一的 IDs 标识数据对象

### 13.12.1. 问题

我的应用程序在多个位置有多个数据对象，但需要确保所有对象的都分配有唯一的 ID 属性，以便用于测试对象之间的平等并确定他们是否表示相同的数据片段。

### 13.12.2. 解决办法

你的数据可实现 IUID 接口，使用 mx.core.UIDUtil.createUID 方法给对象生成新的唯一的 id。

### 13.12.3. 讨论

在有些情况下是非常有用的，比如使用 Adobe LiveCycle messaging 或其他服务，因为这些对象是通过简单相等(== 操作符)或复杂相等( === 操作符)。通过频繁比较对象的所有属性是否相同来确定两个对象是否表示相同的数据，对于数量巨大而复杂的对象的计算将非常消耗资源。实现 IUID 接口后，类将被 id 属性所标记来作为对象间比较的依据。即便对象是深度拷贝，uid 属性也会保留下。

通过 UIDUtil 类的 createUID 生成的 id 是一个32位的十六进制数值格式：

E4509FFA-3E61-A17B-E08A-705DA2C25D1C

下面的例子使用 createUID 方法创建实现 IUID 接口的新的 Message 类实例。IUID 接口的 get uid 和 set uid 方法用于访问对象生成的 id 属性：

```
package {
    import mx.core.IUID;
    import mx.utils.UIDUtil;

    [Bindable]
    public class Message implements IUID {
```

```
public var messageStr:String;
public var fromID:String;
private var _uid:String;

public function Message() {
    _uid = UIDUtil.createUID();
}

public function get uid():String {
    return _uid;
}

public function set uid(value:String):void {
    // Since we've already created the id, there's
    //nothing to be done here, but the method is
    //required by the IUID interface
}
}
```

## 第十四章. 数据绑定(Roast)

Flex 为基于组件的应用程序提供了一种健全的架构模式，在这个强大的框架里，是一个基于事件的系统，在这个事件系统中通过数据绑定，可以通过其它的对象来修改另外一个对象的内部的属性值。

数据绑定使得在应用程序中不同的层间传递数据的变得简单和方便，通过将源属性与目标属性进行关联来实现。当源属性的值有更新时，会产生一个事件来通知目的属性来进行更新。当一个变量标记为可绑定后，其它对象就可以修改该变量的其它目的属性的值。将一个变量上进行数据绑定，你必须使用下面三种方式中的一种来定义[Bindable]标记。

- 在一个类的定义之前

```
package com.oreilly.flexcookbook
{
    import flash.events.EventDispatcher;
    [Bindable]
    public class DataObject extends EventDispatcher{}  
}
```

在一个类创建之前添加[Bindable]标记创建一个绑定的表达式从而使得类的所有共有属性变得可以绑定。所有可以绑定的类必须由 IEventDispatcher 类实现，因为数据绑定是基于事件驱动，来复制源数据到目的数据。

- 在变量的前面进行声明

```
[Bindable] private var _lastName:String;
[Bindable] protected var _age:Number;
[Bindable] public var firstName:String;
```

声明为私有的变量标记为可绑定时，则只能在类中进行绑定。保护的变量则只能在继承或者类本身可见。而共有变量都可见。

- 在属性的前面通过隐含的 getter/setter 方法来变向的进行绑定：

```
private var _lastName:String;
...
[Bindable]
public function get lastName():String
{
    return _lastName;
}
public function set lastName( str:String ):void
{
    _lastName = str;
}
```

当你通过添加[Bindable]标记在 getter 的声明的上方，来定义隐含的 getter/setter 为可绑定的方法，则该变量可以通过点语法进行存取。这样可以你通过同样的语法来存取非绑定的变量，自有变量等来设置数据源绑定。

在框架内部，当绑定的变量值更新时，框架会发送 propertyChange 事件来更新数据。  
[Bindable]标记接受一个事件属性，通过定义一个自定义的事件类型：

[Bindable(event="myValueChanged")]

默认情况下事件属性被设置为 propertyChange。如果不进事件类型进行修改，则目的变量会被内部使用该类型进行提示。如果你自定义了该属性，则必须在类的内部进行声明。

绑定是通过事件通知来实现的，当应用程序中的源变量修改或者在初始化时，就会被触发。你可以通过执行绑定的方法，来强制要求目标为 mx.core.UIComponent 子类的对象的数据绑定执行。

数据绑定提供一层而达到在不同的对象间进行数据同步，从而帮助创建富应用程序。这一节的多种数据绑定的技巧到你的应用程序的架构中去。

## 14.1 节. 绑定一个属性

### 14.1.1 问题

我需要绑定一个对象的属性到另外一个对象中去。

### 14.1.2 解决办法

在 MXML 组件中使用{}标记或者<mx:Binding>标记。

### 14.1.3 讨论

当你声明一个对象的属性到绑定到另外一个对象的属性，一个事件通知从源对象到目标对象则进行了更新的分发了。内部，则将该属性复制到目标属性的变量。为了绑定一个在 MXML 中定义的变量，你可以使用{}和<mx:Binding>。在一个组件中声明一个可绑定的属性，花括号用于包含源属性和来执行更新值。如下面的实例：

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">

    <mx:Panel
        paddingLeft="5" paddingRight="5"
        paddingTop="5" paddingBottom="5">
        <mx:Label text="Enter name:" />
        <mx:TextInput id="nameInput" maxChars="20" />
        <mx:HRule width="100%" />
        <mx:Label text="You've typed:" />
    
```

```

<mx:Text text="{nameInput.text}" />
</mx:Panel>
</mx:Application>

```

在这个例子中，在文本控件中的 text 属性绑定到了 TextInput 控件中。当 TextInput 中的值更新时，Text 控件中的 txt 属性也进行了同步更新。在这个花括号中，点语法用于文本控件中中的 TextInput，该控件的 ID 被声明为 nameInput。

你也可以在 MXML 中使用<mx:Binding>来声明数据绑定，声明的结果和在组件中使用花括号声明一样。那种方法你喜欢使用呢？结果则是看你使用的控制器了。在使用 MVC 架构的应用中，当你定义为<mx:Binding>，那就是在为你的视图定义一个控制器。当你使用花括号，则你不需要特别的对视力和控制器进行分享，因为你的视图就扮演了控制器的作用。

考虑到花括号易于使用，快速开发，并且最终的结果是一样的，你开发过程中，选择使用<mx:Binding>则会从中受益，因为该语法易于阅读，并且允许能为同一个目标源定义不止一个数据源。

使用<mx:Binding>标记，需要定义源属性和目标属性：

```

<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">

    <mx:Binding source="nameInput.text"
        destination="nameOutput.text" />

    <mx:Panel
        paddingLeft="5" paddingRight="5"
        paddingTop="5" paddingBottom="5">
        <mx:Label text="Enter name:" />
        <mx:TextInput id="nameInput" maxChars="20" />
        <mx:HRule width="100%" />
        <mx:Label text="You've typed:" />
        <mx:Text id="nameOutput" />
    </mx:Panel>
</mx:Application>

```

结果和上一个是一样的，但是这个例子中声明的 ID 属性不仅是 TextInput 控件，同时还有 Text 控件做为数据源，分别声明于<mx:Binding>中。

你可能意识到花括号在源和目的属性中没有必要，做为为什么它要存在在内部的声明中。它存在的原因是因为源和目的属性在 ActionScript 中都是可以求值的。这样可以允许你添加任意附加的表达式。举例说明，假如你需要这个例子中一个文本控件去显示你文本输入控件中附加字符串'letters'，在源属性中定义的。

```

<mx:Binding source="nameInput.text.length + ' letters.'"
    destination="nameOutput.text" />

```

## 14.2 节. 绑定到一个函数

### 14.2.1. 问题

我想使用一个函数做为数据源绑定到一个属性的值上。

### 14.2.2. 解决办法

在一个组件的定义时使用花括号进行绑定，传递一个绑定的属性或者一个基于绑定事件从而能被调用的函数做为函数的参数值来实现。

### 14.2.3. 讨论

通过源数据来对目标数据进行更新和同步，是一种快速且简便的方法。当使用属性值做为绑定的数据源时，那么只能绑定在同一种数据类型中。但是当你想使用另外一种类型的数据绑定到另外一种类型中时，使用函数进行绑定则是最适合的了。

你可以使用两种方式来使用函数进行绑定：传递一个已经绑定的属性做为参数进行传递或者定义一个可以绑定的函数做为属性值。

接下来的例子通过传递了一个绑定的源属性值到函数中来更新目标对象的值：

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">

    <mx:CurrencyFormatter id="formatter" precision="2" />

    <mx:Form>
        <mx:FormItem label="Enter the withdrawl amount:>
            <mx:TextInput id="amtInput" />
        </mx:FormItem>
        <mx:FormItem label="Formatted amount:>
            <mx:TextInput editable="false"
                restrict="1234567890"
                text="{formatter.format( amtInput.text )}" />
        </mx:FormItem>
    </mx:Form>
</mx:Application>
```

第一个 TextInput 的实例是用于格式化第二个 TextInput 的值，通过<mx:CurrencyFormatter>来进行格式化。通过绑定，当 amtInput 的值更新的同时，格式化的函数也同时得到调用。传递一个绑定的属性做为参数给一个函数是一种简洁的方法来保证一对一的数据同步是可靠的。

除了传递一个可以绑定的值做为参数来传递外，还可以使用[Binding]标签去定义一个函数，使其变得可以绑定。当定义的事件类型被捕获时，函数也被调用同时也强制去更新任意绑定的属性。细想下面这个例子：

代码视图：

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    creationComplete="initHandler();">

    <mx:Script>
        <![CDATA[
            private var _fruit:String;
            private var _fruits:Array = ["Apple", "Banana",
                "Orange"];

            private function initHandler():void
            {
                fruitCB.dataProvider = _fruits;
            }

            [Bindable(event="fruitChanged")]
            private function isOrangeChosen():Boolean
            {
                return _fruit == "Orange";
            }

            public function get fruit():String
            {
                return _fruit;
            }

            public function set fruit( str:String ):void
            {
                _fruit = str;
                dispatchEvent( new Event( "fruitChanged" ) );
            }
        ]]>
    </mx:Script>

    <mx:Label text="select a fruit:" />
    <mx:HBox>
        <mx:ComboBox id="fruitCB"
            change="{fruit = fruitCB.selectedLabel}" />
        <mx:Button label="Eat the orange."
            enabled="{isOrangeChosen()}" />
    </mx:HBox>
</mx:Application>
```

在这个例子中，Button 实例中的 enabled 属性被绑定到一个布尔型返回值的 isOrangeChosen 方法。该函数的返回值基于 \_fruit 的值，当 ComboBox 被选择时，该值也同时进行了更新。

所以当 fruit 值更新时都会分发 fruitChanged 事件同时也调用了 isOrangeChosen 访求，同时也强制的更新了 Button 实例中的 enabled 属性，从而使 enabled 变得是否可以使用。

本质上，button 实例的是否可用依赖于 ComboBox 控制所选择的值。在不同的数据源类型中进行数据绑定，使用函数来进行绑定是最快捷的方法。

## 14.3 节. 创建一个双向绑定

### 14.3.1. 问题

我想在两个源和目标间做相互间的数据绑定。

### 14.3.2. 解决办法

在两个控件间同时使用数据绑定。

### 14.3.3. 讨论

在一组的数据绑定的控件间，两个组件彼此同时扮演了源和目标的角色。Flex 框架支持相互间的数据绑定，且不会导致一个死循环。请看例子：

```
<mx:VBox  
    xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical">  
    <mx:Label text="From Input 2:" />  
    <mx:TextInput id="input1" text="{input2.text}" />  
    <mx:HRule />  
    <mx:Label text="From Input 1:" />  
    <mx:TextInput id="input2" text="{input1.text}" />  
</mx:VBox>
```

两个 TextInput 实例同时扮演了数据源和目标，更新时则会影响到彼此。当 TextInput 中有文本输入时，这个值同时也复制到另外一个 TextInput 中。

## 14.4 节. 使用 ActionScript 来进行数据绑定

### 14.4.1. 问题

我想去通过 ActionScript 而不是 MXML 来创建一个数据绑定。

### 14.4.2. 解决办法

使用类 mx.utils.binding.BindingUtils 来创建 mx.utils.binding.ChangeWatcher 对象。

### 14.4.3. 讨论

使用 ActionScript 来创建数据绑定，当目标更新的时候，可以给你提供更多的可控性。为了使用 ActionScript 创建一个数据绑定，使用类 BindingUtils 来创建一个 ChangeWatcher 对象。BindingUtils 提供了两个静态方法，可用于创建数据绑定：bindProperty 和 bindSetter。

使用方法 bindProperty 和 BindingUtils 和在 MXML 中使用<mx:Binding>标记的效果是一致的。不像使用<mx:Binding>标记一样，有可使用的属性，需要使用 ActionScript 来进行声明，BindingUtils.bindProperty 的参数用于定义源和目标对象以及属性。如下：

代码视图:

```
var watcher:ChangeWatcher = BindingUtils.bindProperty( destination, "property", source,
"property" );
```

通过方法 BindingUtils.bindSetter，你可以声明函数来处理数据绑定的源数据的更新事件。

```
var watcher:ChangeWatcher =  BindingUtils.bindSetter( invalidateProperty, source, "property" );
...
private function invalidateProperty( arg:* ):void
{
    // perform any necessary operations.
}
```

当使用了静态方法 bindProperty 和 bindSetter 后，则没有必要定义变量 ChangeWatcher 了。然而，有的时候你可能需要利用返回的 ChangeWatcher 对象，因为通过该对象，则有可以更新数据源、目标属性以及停止数据绑定。

接下来的例子中，使用方法 BindingUtils.bindProperty 在控件 TextInput 和控件 Text 间的 text 属性间进行了数据绑定：

代码视图:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    creationComplete="initHandler();">

    <mx:Script>
```

```

<! [CDATA[

    import mx.binding.utils.ChangeWatcher;
    import mx.binding.utils.BindingUtils;

    private var _nameWatcher:ChangeWatcher;

    private function initHandler():void
    {
        _nameWatcher = BindingUtils.bindProperty( nameField,
            "text", nameInput, "text" );
    }

    private function clickHandler():void
    {
        if( _nameWatcher.isWatching() )
        {
            _nameWatcher.unwatch();
            btn.label = "watch";
        }
        else
        {
            _nameWatcher.reset( nameInput );
            btn.label = "unwatch";
        }
    }
}

]]>
</mx:Script>

<mx:Panel title="User Entry."
paddingLeft="5" paddingRight="5"
paddingTop="5" paddingBottom="5">
<mx:Form>
    <mx:FormItem label="Name:>
        <mx:TextInput id="nameInput" />
    </mx:FormItem>
</mx:Form>
<mx:HRule width="100%" />
<mx:Label text="You Entered:" fontWeight="bold" />
<mx:HBox>
    <mx:Label text="First Name:" />
    <mx:Text id="nameField" />
</mx:HBox>
<mx:Button id="btn" label="unwatch"
    click="clickHandler();"/>
</mx:Panel>
</mx:Application>
```

使用方法 BindingUtils.bindProperty 时，数据绑定是在一个源和目标间进行一对一的绑定。在这个例子中，使用 TextInput 制件实例中的 text 属性进行更新时，同步也更新了 Text 控件实例。数据绑定的生命周期停止和重围，可以通过 Button 实例中的操作，来设置 ChangeWatcher 实现。

更新一个或者多个目标绑定的数据时，可以如下面展示的一样使用 BindingUtils.bindSetter 来声明一个函数来做为数据绑定的引导：

代码视图：

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    creationComplete="initHandler();">

    <mx:Script>
        <! [CDATA[
            import mx.binding.utils.ChangeWatcher;
            import mx.binding.utils.BindingUtils;

            private var _nameWatcher:ChangeWatcher;

            private function initHandler():void
            {
                _nameWatcher =
                    BindingUtils.bindSetter( invalidateName, nameInput, "text" );
            }

            private function invalidateName( arg:* ):void
            {
                if( btn.label == "unwatch" )
                    nameField.text = nameInput.text;
            }

            private function clickHandler():void
            {

                if( _nameWatcher.isWatching() )
                {
                    _nameWatcher.unwatch();
                    btn.label = "watch";
                }
                else
                {

                    _nameWatcher.reset( nameInput );
                    btn.label = "unwatch";
                }
            }
        ]>
    </mx:Script>

```

```

]]>

</mx:Script>

<mx:Panel title="User Entry."
paddingLeft="5" paddingRight="5"
paddingTop="5" paddingBottom="5">
<mx:Form>
<mx:FormItem label="Name:>
<mx:TextInput id="nameInput" />
</mx:FormItem>
</mx:Form>
<mx:HRule width="100%" />
<mx:Label text="You Entered:" fontWeight="bold" />
<mx:HBox>
<mx:Label text="First Name:" />
<mx:Text id="nameField" />
</mx:HBox>
<mx:Button id="btn" label="unwatch"
click="clickHandler();"/>
</mx:Panel>

</mx:Application>

```

任意数据的更新通过 BindingUtils.bindSetter 做为参数传递从而来实现。Setter 方法做为事件的处理器，当目标值更改时通知它来进行处理。在这个例子中，text 的值是根据 Button 实例的 label 值来进行更新的。

nameInput 控件的 text 属性更新时，invalidateName 方法也会被调用，对于目标数据的更新则是当前活动 ChangeWatcher 来处理。

注释：

为了适应 Flash Player 的垃圾收集，任何在一个实例中创建的 ChangeWatcher 对象是设置为 unwatch 的数据绑定语句，这一点是很重要的。当创建一个 ChangeWatcher 对象时，就是上一个例子中使用 BindingUtils 类一样，在内存中会对进行绑定的源和目标都会保存一个引用。为了从内存中释放这些引用和标记这些对象，以便于垃圾收集器能正确的进行释放，它们需要使用 unwatch 方法来进行移除。

## 14.5 节. 链式的属性绑定

### 14.5.1 问题

我想定义一个源属性做为一个链式属性中的一部分。

### 14.5.2. 解决办法

使用<mx:Binding>标签或者花括号，通过使用点连接方式存取一个属性链中的属性，或者在静态方法 BindingUtils.bindProperty 和 BindingUtils.bindSetter 使用链式的字符串数组做为参数来进行处理。

### 14.5.3. 讨论

当一个源属性被在数据绑定表达式中定义，则所有的属性的更改都会监视。如果你指定了 TextInput 控件的 text 属性做为绑定的数据源，则 TextInput 控件实例则是可绑定的属性链中的一部分。

```
<mx:TextInput id="myInput" />
<mx:Label text="{myInput.text}" />
```

技术上，myInput 控件所在的类也是属性绑定链中的一部分，但是数据绑定语句是有可见性，所以该指向是没有必要。本质上，myInput 的值首先被设置为非空，同时，绑定也移动到了链中的源：TextInput 实例中的 text 属性。只有当源数据被绑定时，才会触发将数据复制到目的对象的事件。

在上一章节的例子中，我们是存取同一个模块中的一个数据链，就像源属性来自于同一个控制器。在 MXML 中，你可以通过使用点标记的语法定义一个可以绑定的数据链。

代码视图：

```
<!-- property chain binding using <mx:Binding> -->
<mx:Binding source="usermodel.name.firstName"
    destination="fNameField.text" />
<mx:Label id="fNameField" />

<!-- property chain binding using curly braces ({}). -->
<mx:Label text="{usermodel.name.firstName}" />
```

在 ActionScript 3 为了定义一个可以绑定的数据链，你必须指定这个链为一个字符串值的数组，不管是你调用 BindingUtils.bindProperty 或者 BindingUtils.bindSetter 方法：

```
BindingUtils.bindProperty( nameField, "text", usermodel, ["name", "firstName"] );
BindingUtils.bindSetter( invalidateProperties, this, ["usermodel", "name", "firstName"] );
```

这两个方法的参数是一个字符串的数组，其中定义了绑定的数据链：

接下来的例子中，使用了花括号、<mx:Binding>标记，以及方法 BindingUtils.bindProperty 来使用数据链来进行数据绑定。

代码视图：

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    creationComplete="initHandler();">

    <mx:Script>
```

```

<! [CDATA[

    import mx.binding.utils.BindingUtils;
    // create data binding using BindingUtils.
    private function initHandler():void
    {
        BindingUtils.bindProperty( lastNameField, "text",
            usermodel, ["name", "lastName"] );
    }

    private function clickHandler():void
    {
        usermodel.name.firstName = fNameInput.text;
        usermodel.name.lastName = fNameInput.text;
        usermodel.birth.date = dateInput.text;
    }
]]>

</mx:Script>

<!-- defined model -->
<mx:Model id="usermodel">
    <user>
        <name>
            <firstName>Ted</firstName>
            <lastName>Henderson</lastName>
        </name>
        <birth>
            <date>February 29th, 1967</date>
        </birth>
    </user>
</mx:Model>

<!-- create data binding using <mx:Binding> -->
<mx:Binding source="usermodel.birth.date"
    destination="dateField.text" />
<mx:Form>
    <mx:FormItem label="First Name:>
        <!-- create data binding using curly braces -->
        <mx:Text text="{usermodel.name.firstName}" />
    </mx:FormItem>
    <mx:FormItem label="Last Name:>
        <mx:Text id="lastNameField" />
    </mx:FormItem>
    <mx:FormItem label="Birthday:>
        <mx:Text id="dateField" />
    </mx:FormItem>
</mx:Form>
<mx:HRule />
```

```

<mx:Form>
    <mx:FormItem label="First Name:>
        <mx:TextInput id="fNameInput" />
    </mx:FormItem>
    <mx:FormItem label="Last Name:>
        <mx:TextInput id="lNameInput" />
    </mx:FormItem>
    <mx:FormItem label="Birthday:>
        <mx:TextInput id="dateInput" />
    </mx:FormItem>
    <mx:FormItem label="Submit Changes">
        <mx:Button label="ok" click="clickHandler();"/>
    </mx:FormItem>
</mx:Form>
</mx:Application>

```

## 14.6 节. 使用 E4X 进行绑定 XML 的数据

### 14.6.1 问题

我想使用一个 XML 的数据做为一个绑定的对象的数据来源。

### 14.6.2. 解决办法

在使用花括号或者`<mx:Bindable>`标签进行数据绑定时使用 E4X 来进行数据绑定。

### 14.6.3. 讨论

ActionScript 3 中的 E4X 语言是用于在语句中来过滤 XML 中的数据，且的语法也与 ActionScript 语法相近。在这一章节中没有太多的时间来讨论使用 E4X 语句的好处的细节，但是它很有必有提醒你可以使用这种语言方便在一个控件和一个 XML 间进行数据绑定。

E4X 语句在组件的声明都可以使用`<mx:Binding>`和花括号来。不能在类 BindingUtils 中使用 E4X。可能通过下面这个基于 XML 的例子来更好的理解 E4X 是如何工作的：

```

<item>
    <name>Moe</name>
    <type>The brains.</type>
    <description>Has bowl cut.</description>
</item>

```

你可以在一个控件的属性中使用花括号来包含一段 E4X 语句。

```

<mx:Label text="{_data..item.(name == 'Moe').description}" />

```

或者你可以使用<mx:Binding>标签来创建一个绑定:

```
<mx:Binding source="_data..item.(name == 'Moe').description"
    destination="desc.text" />
<mx:Label id="desc" />
```

两种方法得到的结果是一样的。花括号在<mx:Binding>标签中的属性中是不需要的，因为这里边的值会当做 ActionScript 语句一样来进行处理。在接下来的例子使用 E4X 来为 dataProvider 属性在一个 List 和 DataGrid 间进行了数据绑定。

代码视图:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">

    <mx:Script>
        <![CDATA[
            [Bindable] private var _data:XML =
                <items>
                    <item id='1'>
                        <name>Larry</name>
                        <type>The foil.</type>
                        <description>Has curly hair.</description>
                    </item>
                    <item id='2'>
                        <name>Moe</name>
                        <type>The brains.</type>
                        <description>Has bowl cut.</description>
                    </item>
                    <item id='3'>
                        <name>Curly</name>
                        <type>The braun.</type>
                        <description>Has bowl cut.</description>
                    </item>
                </items>;
        ]]>
    </mx:Script>

    <mx:Binding source="{_data..item.(@id == '1').name}
        {_data..item.(@id =='1').description.toLowerCase() }"
        destination="lab.text" />
    <mx:Label id="lab" />
    <mx>List width="200" dataProvider="{_data..item.name}" />
    <mx>DataGrid width="200" dataProvider="{_data..item}">
        <mx:columns>
            <mx:DataGridColumn dataField="name" />
        </mx:columns>
    </mx>DataGrid>

```

```

<mx:DataGridColumn dataField="type" />
</mx:columns>
</mx:DataGrid>
</mx:Application>

```

在组件展现初始化的过程中，绑定也得到了执行，基于 E4X 语句的相关属性的值也得到了更新。

## 14.7 节. 创建个性化可绑定的属性

### 14.7.1 问题

我想创建一个基于某种特殊的而非依赖于 `propertyChange` 事件的数据绑定。

### 14.7.2. 解决办法

设置`[Bindable]`标签的 `event` 属性，使用一个字符串做为类型的参数来进行事件的分发。

### 14.7.3. 讨论

Flex 框架中的数据绑定的基础其实是一个基于事件的系统。默认的数据绑定的事件类型是分发到 `propertyChange` 事件。在框架内部，对于目标属性数据的更新并不一定是直接由数据绑定的源直接分发过去的。你可以在一个数据绑定语句中自定义一个事件类型，通过使用 `[Bindable]` 标签的 `event` 属性。例如：

```
[Bindable(event="myValueChanged")]
```

当你覆盖了默认的`[Bindable]`标签的 `event` 属性的定义，你必须分布指定的事件来使数据绑定生效。

接下来的例子使用了自定义的绑定事件来更新目标属性的值。

代码视图:

```

<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">

    <mx:Script>
        <! [CDATA[

            private var _firstName:String;
            private var _lastName:String;
            public static const FIRST_NAME_CHANGED:String =

```

```

        "firstNameChanged";
    public static const LAST_NAME_CHANGED:String =
        "lastNameChanged";

    private function clickHandler():void
    {
        firstName = fnInput.text;
        lastName = lnInput.text;
    }

    [Bindable(event="firstNameChanged")]
    public function get firstName():String
    {
        return _firstName;
    }
    public function set firstName( str:String ):void
    {
        _firstName = str;
        dispatchEvent( new Event( FIRST_NAME_CHANGED ) );
    }

    [Bindable(event="lastNameChanged")]
    public function get lastName():String
    {
        return _lastName;
    }
    public function set lastName( str:String ):void
    {
        _lastName = str;
        dispatchEvent( new Event( LAST_NAME_CHANGED ) );
    }
}

]]>

</mx:Script>

<mx:Panel title="User Entry."
paddingLeft="5" paddingRight="5"
paddingTop="5" paddingBottom="5">
<mx:HBox>
    <mx:Label text="First Name:" />
    <mx:TextInput id="fnInput" />
</mx:HBox>
<mx:HBox>
    <mx:Label text="Last Name:" />
    <mx:TextInput id="lnInput" />
</mx:HBox>

```

```

<mx:Button label="submit" click="clickHandler()" />
<mx:HRule width="100%" />
<mx:Label text="You Entered:" fontWeight="bold" />
<mx:HBox>
    <mx:Label text="First Name:" />
    <mx:Text text="{firstName}" />
</mx:HBox>
<mx:HBox>
    <mx:Label text="Last Name:" />
    <mx:Text text="{lastName}" />
</mx:HBox>
</mx:Panel>
</mx:Application>

```

当用户提交了输入的数据，属性 firstName 和 lastName 得到了更新。在各自的 setter 方法，对应的在[Bindable]中定义的事件被触发，从而更新了目标的值。

自定义的绑定属性的一个好处是可以自由支配何时来对目标数据进行更新。因为数据绑定是基于一个事件模型，使用自定义的绑定为你提供一种更好的手段来判断何时或者是否该事件是否需要触发。

接下来你的例子添加了一个记时器来推迟分发一个数据绑定的事件：

代码视图:

```

<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    creationComplete="initHandler()">

    <mx:Script>
        <![CDATA[
            private var _timer:Timer;
            private var _firstName:String;
            private var _lastName:String;
            public static const FIRST_NAME_CHANGED:String =
                "firstNameChanged";
            public static const LAST_NAME_CHANGED:String =
                "lastNameChanged";

            private function initHandler():void
            {
                _timer = new Timer( 2000, 1 );
                _timer.addEventListener( TimerEvent.TIMER_COMPLETE,
                    timerHandler );
            }
        ]]>
    </mx:Script>

```

```

private function clickHandler():void
{
    firstName = fnInput.text;
    lastName = lnInput.text;
}

private function timerHandler( evt:TimerEvent ):void
{
    dispatchEvent( new Event( FIRST_NAME_CHANGED ) );
}

[Bindable(event="firstNameChanged") ]
public function get firstName():String
{
    return _firstName;
}

public function set firstName( str:String ):void
{
    _firstName = str;
    _timer.reset();
    _timer.start();
}

[Bindable(event="lastNameChanged") ]
public function get lastName():String
{
    return _lastName;
}

public function set lastName( str:String ):void
{
    _lastName = str;
    dispatchEvent( new Event( LAST_NAME_CHANGED ) );
}

]]>
</mx:Script>

<mx:Panel title="User Entry."
paddingLeft="5" paddingRight="5"
paddingTop="5" paddingBottom="5">
<mx:HBox>
    <mx:Label text="First Name:" />
    <mx:TextInput id="fnInput" />
</mx:HBox>
<mx:HBox>
    <mx:Label text="Last Name:" />
    <mx:TextInput id="lnInput" />

```

```

</mx:HBox>
<mx:Button label="submit" click="clickHandler();"/>
<mx:HRule width="100%"/>
<mx:Label text="You Entered:" fontWeight="bold"/>
<mx:HBox>
    <mx:Label text="First Name:"/>
    <mx:Text text="{firstName}" />
</mx:HBox>
<mx:HBox>
    <mx:Label text="Last Name:"/>
    <mx:Text text="{lastName}" />
</mx:HBox>
</mx:Panel>
</mx:Application>

```

事件类型还是定义在 `firstName` 属性隐含的 `getter` 方法，但是事件分发则是一个计时器实例来实现了。当你运行这段程序时，数据绑定到 `lastName` 属性会立即生效，因为自定义的事件分发是 `setter` 方法的属性。然而对于 `firstName` 属性的目标的数据更新，则会 2 秒后才生效，因为计时器实例被设置用于分发自定义的 `firstNameChanged` 事件。

## 14.8 节. 绑定到一个一般的对象

### 14.8.1. 问题

我想通过使用一个顶层的对象实例做为源来绑定到一个属性上。

### 14.8.2. 解决办法

使用类 `mx.utils.ObjectProxy` 来在对象与分发绑定的事件间进行交换。

### 14.8.3. 讨论

直接创建一个绑定到一个一般的对象上只会在目标对象初始化的时候引入一个更新操作。为了源对象的属性值得到更新的时候，能同时更新目标对象的值，需要使用 `ObjectProxy`。创建一个 `ObjectProxy` 对象，需要传递该对象到构造函数中。例如：

```

var obj:Object = {name:'Tom Waits', album:'Rain Dogs',
    genre:'Rock'};
var proxy:ObjectProxy = new ObjectProxy( obj );

```

对象的属性值的修改会被 `ObjectProxy` 进行处理，当更新操作发生时，`ObjectProxy` 对象会分发一个 `propertyChange` 事件。该 `propertyChange` 事件是默认绑定分发的事件。当默认的事件

被分发时，源属性的值就会被复制到目标对象的属性。在接下来的例子中传递了一个一般的对象到一个类 ObjectProxy 的实例的构造函数：

代码视图：

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">

    <mx:Script>
        <![CDATA[
            import mx.utils.ObjectProxy;

            private var obj:Object = {name:'Tom Waits',
                album:'Rain Dogs',
                genre:'Rock'};

            [Bindable]
            private var proxy:ObjectProxy = new ObjectProxy( obj );

            private function clickHandler():void
            {

                proxy.name = nameField.text;
                proxy.album = albumField.text;
                proxy.genre = genreField.text;
            }
        ]]>
    </mx:Script>

    <mx:Form>
        <mx:FormItem label="Name:">
            <mx:TextInput id="nameField" />
        </mx:FormItem>
        <mx:FormItem label="Album:">
            <mx:TextInput id="albumField" />
        </mx:FormItem>
        <mx:FormItem label="Genre:">
            <mx:TextInput id="genreField" />
        </mx:FormItem>
        <mx:FormItem label="Submit Changes">
            <mx:Button label="ok" click="clickHandler();"/>
        </mx:FormItem>
    </mx:Form>
    <mx:HRule width="100%" />
    <mx:Form>
        <mx:FormItem label="Name:">
            <mx:Text text="{proxy.name}" />
        </mx:FormItem>
    </mx:Form>

```

```

</mx:FormItem>
<mx:FormItem label="Album:>
    <mx:Text text="{proxy.album}" />
</mx:FormItem>
<mx:FormItem label="Genre:>
    <mx:Text text="{proxy.genre}" />
</mx:FormItem>
</mx:Form>

</mx:Application>

```

在这个例子中，当更新被提交后，对象 ObjectProxy 的属性被修改，同时更改也被映射到绑定到该对象的组件中。只有当属性预定义在一个代理的对象中，没有任何限制的进行更新，同样也可以为这些语句定义绑定的语句。

你应当创建一个自定义的类，同时使用让绑定的属性为共有，来取代使用一般的对象。然而，当在你的应用程序架构中这是不可能的时候，使用 ObjectProxy 来处理是受益不少的。

## 14.9 节. 绑定到一个动态类的属性

### 14.9.1. 问题

你需要绑定一个目标的属性到一个目标并不明确的动态类对象的一个属性上。

### 14.9.2. 解决办法

创建一个 mx.utils.Proxy 的子类，实现 mx.events.IEventDispatcher 接口，覆盖 flash\_proxy 名字空间的 setProperty 方法，来分发 propertyChange 事件。

### 14.9.3. 讨论

Proxy 类允许你使用点语法来存取属性。为了能有效的与动态属性的引用进行工作，在你的子类的实现中重写 flash\_prox 名字空间中的方法 getProperty 和 setProperty。如果类中的这些方法被定义为共有的类，则你可以自定义来存取这些属性。然后，动态的属性引用不足以创建绑定，因为数据绑定是基本事件系统的。

因为绑定是通过事件来触发的，创建一个 Proxy 类是适合来进行数据绑定的，你必须同时实现 IEventDispatcher 以及它们的接口。为了使动态属性的引用能被进行绑定，类需要用关键字 dynamic 来进行声明，同时使用[Bindable]标记来进行定义，且设置标签的 event 属性值为 propertyChange：

代码视图：

[**Bindable** (event="propertyChange") ]

```
dynamic public class Properties extends Proxy implements  
IEventDispatcher {}
```

一个不错的例子，当你需要创建一个自定义的 Proxy 类用来存取一个从内部源加载的数据，通过在重载的 setProperty 和 getProperty 方法创建一定的规则，而不是去编写一个分析器，会填充属性在一个自定义的对象从加载的数据中。

例如，一个程序加载下面的 XML 数据，且这些 XML 数据的属性是能进行存取和修改的：

```
<properties>  
  <property id="name"><![CDATA[Tom Waits]]></property>  
  <property id="album"><![CDATA[Rain Dogs]]></property>  
  <property id="genre"><![CDATA[Rock]]></property>  
</properties>
```

你可以创建一个 mx.utils.Proxy 的子类，同时在重载的 setProperty 和 getProperty 方法中使用 E4X，允许一个客户端来存取和修改 XML 中的属性的值。

代码视图：

```
override flash_proxy function getProperty( name:* ):*  
{  
  return xml..property.(@id == String( name ) );  
}  
  
override flash_proxy function setProperty( name:*, value:*, value:void  
{  
  var index:Number = xml..property.(@id == String( name )  
    ).childIndex();  
  xml.replace( index, '<property id="' + name + '">' + value +  
    '</property>' );  
}
```

数据绑定的事件当一个属性的值被修改时会被触发。在这个例子中的重载的 setProperty 方法，虽然它更新了一个属性的值，但是并没有分发一个更新的通知。为了能绑定到一个动态的属性引用上，你必须通过 Proxy 子类分发一个 PropertyChange 的事件。

代码视图：

```
override flash_proxy function setProperty( name:*, value:*, value:void  
{  
  var oldVal:String = xml..property.(@id == String( name ) );  
  var index:Number = xml..property.(@id == String( name )  
    ).childIndex();  
  xml.replace( index, '<property id="' + name + '">' + value +  
    '</property>' );  
  var evt:Event = PropertyChangeEvent.createUpdateEvent( this,  
    name,oldVal, value );  
  dispatchEvent( evt );  
}
```

类 PropertyChangeEvent 中的静态 createUpdateEvent 方法返回一个属性被设置为 propertyChange 的 PropertyChangeEvent 的实例， propertyChange 是默认用来进行绑定的事件类型，被设置为这个类中[Bindable]的属性。

在接下来的例子中，是一个完整的 Proxy 子类的实现，适合于进行数据绑定：

代码视图：

```
package com.oreilly.flexcookbook

{
    import flash.events.Event;
    import flash.events.EventDispatcher;
    import flash.events.IEventDispatcher;
    import flash.net.URLLoader;
    import flash.net.URLRequest;
    import flash.utils.Proxy;
    import flash.utils.flash_proxy;

    import mx.events.PropertyChangeEvent;

    [Event(name="complete", type="flash.events.Event")]
    [Bindable(event="propertyChange")]
    dynamic public class Properties extends Proxy
        implements IEventDispatcher
    {
        private var _evtDispatcher:EventDispatcher;
        private var _data:XML;
        private var _loader:URLLoader;

        public static const COMPLETE:String = "complete";

        public function Properties()
        {
            _evtDispatcher = new EventDispatcher();
        }
        // load external xml.
        public function loadProperties( fnm:String ):void
        {
            _loader = new URLLoader();
            _loader.addEventListener( Event.COMPLETE, loadHandler );
            _loader.load( new URLRequest( fnm ) );
        }
        // set data property and dispatch 'complete' notification.
        private function loadHandler( evt:Event ):void
        {
            data = XML( _loader.data );
            dispatchEvent( new Event( Properties.COMPLETE ) );
        }
    }
}
```



```

) :void
{
    _evtDispatcher.removeEventListener( type, listener,
        useCapture );
}
// IEventDispatcher implementation.
public function dispatchEvent( evt:Event ):Boolean
{
    return _evtDispatcher.dispatchEvent( evt );
}
// IEventDispatcher implementation.
public function hasEventListener( type:String ):Boolean
{
    return _evtDispatcher.hasEventListener( type );
}
// IEventDispatcher implementation.
public function willTrigger( type:String ):Boolean
{
    return _evtDispatcher.willTrigger( type );
}
}
}

```

你可以存取和修改这个加载的 XML 的元素，被放置在 Properties proxy 中，通过点连接语法：

```

var myProxy:Properties = new Properties();
myProxy.load('properties.xml' );
..
var name:String = myProxy.name;
myProxy.album = "Blue Valentine";

```

虽然你可以使用点连接语法与动态的属性引用进行工作，你确不能在花括号中使用点连接语法，或者在 MXML 中使用<mx:Binding>进行的数据绑定。如果你想使用点连接语法的话，在进行编译的时候会得到一个警告提示：

因为这些 XML 数据是在运行的时候进行加载的，这样可以理解为当你是在数据加载后再创建绑定的。为了达到这点，使用 mx.utils.BindingUtils 类来强制更新和确认数据绑定到 proxy.

下面的代码片断创建一个程序，使用上面的 Properties 代理的一个实例来创建一个数据绑定到一个组件的属性：

代码片段：

```

<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    creationComplete="initHandler();">

    <mx:Script>

```

```

<! [CDATA[

    import mx.binding.utils.BindingUtils;
    import com.oreilly.flexcookbook.Properties;

    private var _properties:Properties;
    // create proxy and load xml.
    private function initHandler():void
    {
        _properties = new Properties();
        _properties.addEventListener( Event.COMPLETE,
                                     propertiesHandler );
        _properties.loadProperties( "data/properties.xml" );
    }
    // xml data loaded. establish data binding.
    private function propertiesHandler( evt:Event ):void
    {
        BindingUtils.bindProperty( nameOutput, "text",
                                   _properties, "name" );
        BindingUtils.bindProperty( albumOutput, "text",
                                   _properties, "album" );
        BindingUtils.bindProperty( genreOutput, "text",
                                   _properties, "genre" );
    }
    // change properties of proxied data.
    private function changeHandler():void
    {
        _properties.name = nameField.text;
        _properties.album = albumField.text;
        _properties.genre = genreField.text;
    }
]]>
</mx:Script>
<mx:Label text="Data Loaded." />
<mx:Form>
    <mx:FormItem label="Name:>
        <mx:Text id="nameOutput" />
    </mx:FormItem>
    <mx:FormItem label="Album:>
        <mx:Text id="albumOutput" />
    </mx:FormItem>
    <mx:FormItem label="Genre:>
        <mx:Text id="genreOutput" />

    </mx:FormItem>
</mx:Form>
<mx:HRule width="100%" />
<mx:Form>

```

```
<mx:FormItem label="Name:>
    <mx:TextInput id="nameField" />
</mx:FormItem>
<mx:FormItem label="Album:>
    <mx:TextInput id="albumField" />
</mx:FormItem>
<mx:FormItem label="Genre:>
    <mx:TextInput id="genreField" />
</mx:FormItem>

<mx:FormItem label="Submit Changes">
    <mx:Button label="ok" click="changeHandler();" />
</mx:FormItem>
</mx:Form>
</mx:Application>
```

在 propertiesHandler 事件处理器中，当 XML 数据通过 Properties 实例加载后，通过使用 BindingUtils.bindProperty 方法能进行熟练的数据绑定。第一个表单的各个文本控件的 text 属性被绑定到了对应的 XML 元素上。在类 Properties 中的重载的 getProperty 使用 E4X，一个绑定的更新被创建，该值也被复制。

属性的值的更新是通过 changeHandler 事件处理器中使用点连接的语法来进行操作的，它在 Properties 实例中引入了 setProperty 方法，同时分发一个事件通知，通过使用 PropertyChangeEvent 对象来引入数据绑定。

## 第十五章. 验证, 格式化及正则表達式 (TONY\_IAN)

验证, 格式化及正则表達式这几个单词给读者的第一印象可能会比较陌生。其实, 开发者在日常的工作中已经不知不觉地接触过不少类似的事物, 比如说: 透过剖析字符串的格式去检测某种模式; 当特定的字符串模式被检测或没有检测到时修改字符串至一定模式; 当一些必要的属性没有被检测到的时候, 返回错误讯息等。因为一些日常事物如电话号码, 姓氏字符的大写, 货币格式, 邮编及国际标准图书编号(ISBN)等数据, 一般来说都会由第三方程序或用户本身来提供的, 所以难以保证这些数据的格式是否合乎我们程序的要求。Flex Framework 本身就内建了两个强大的类分别为 Validator 和 Formatter, 它们分别为各个 UI 组件提供这一类型剖析及格式化的功能。接下来就是正则表達式(Regular Expressions), 它是一个新引入的 ActionScript 语言及 Flash Player 的编程工具。它强大的功能为不少用户留下深刻的印象, 但其过于复杂的语法也使人对它又爱又恨。

Validator, 即 **验证器** 作为一个事件发送器, 可以用来检测 Flex 控件的属性, 以确保它可以满足一些预定的参数。这些参数可以被指定为某种的格式, 例如一个属性是否必要的或者该属性的长度限制等。由于验证器的高度整合性, 通过以下两个设置, 验证的结果可以轻易地显示出来:

1. 设置验证器类的事件源, 即是使用者输入的接口。
2. 设置那些属性会被会检查。

验证器会发送一个事件去控件, 根据所传递的事件, 这个控件会在验证器内显示一个自定义的错误信息。Flex Framework 本身已经提供了不少内建的验证器, 例如: 信用卡号码, 电话号码, 电邮地址及社会安全号码等。在这个章节里, 我们集中讨论如何建设自定义的验证器及如何将验证器及相关事件跟控件整合。

Formatter, 以下称为**格式器**, 负责一个简单而非常重要的工作, 那就是把任何传入的数值转换作所指定的格式。比如将一个包含 9 位连续数目的字符串转换成一个格式化的电话号码, 那就是由 555555555 转成(555) 555-5555, 还有日期格式转换, 或不同国家邮政编号的格式转换等。格式器本身就为用户定义了一个重要的方法, 那就是 format(格式)。它是用来接收输入数据以及返回一个恰当的字符串。

以上所提及的两个类, 原则上是可以通过正则表达式(Regular Expression)来进行字符串的处理, 虽然这种方法在基础类中是不常用的。正则表达式是一种被公认为强大, 高雅且难于应用的现代化编程开发工具。程序员可以使用这套工具去为一些特定的字符串创建各种复杂的处理规则。几乎所有主流的编程语言都内建了正则表达式处理引擎, 虽然它们可能功能上各有分别, 但语法却是大致相同的。所以, 正则表达式为你的作品来说绝对是一套非常实用的工具。在 ActionScript 里头, 正则表达式是以一个名为 RegExp 的类来表达的, 它包含着两个基础的方法:

“test”, 是用于检测一个字符串是否包含任何与 **RegExp** 相配的文字, 并返回一个布尔值(True / False)。“exec”, 用于返回一个包含所有相配的项目的数组及对象, 以及包含这个事件第一次发生时所在字符串的相对位置的字符串。

正则表达式也可以通过 match, search 及 replace 等 String 类的方法来进行检测。当中, 我发

现 String 类的方法是十分有用的，因为在正则表达式中它们允许来对个别的字符进行处理。正则表达式是一个非常庞大的主题，如果要对其进深入探讨的话，恐怕要用掉整本书的篇幅。所以在本章节中，我们只会对特定的观点进行探讨以及为一些一般性的问题提供解决方案，而不会尝试去展示一些比较普遍的使用个案。

## 15.1 节. 在 **TextInput** 及 **TextArea** 控件上使用 **Validator** 和 **Formatter**

### 15.1.1 问题

你需要为多个 **TextInput** 和 **TextArea** 控件进行验证和格式化。

### 15.1.2 解决办法

对于所有类型的输入数据，无论是日期，电话号码或者货币，都要做以下的步骤：

使用 **Validator** 去确保所有输入数据都是有效的。

使用 **Formatter** 去规范所有输入数据的格式。

### 15.1.3 讨论

如果要让多个 **validator** 和 **formatter** 跟一个组件一起使用的话，我们就需要为不同类型的验证创建独立的 **validator**。举例说，当某个 **TextInput** 组件返回一个 **focusOut** 事件的时候，我们就需要呼叫相对应的 **validator** 内的 **validate** 方法。假如你想把一个 **validator** 绑定到指定的 **TextInput** 组件，你就需要把那个 **TextInput** 设定为事件源及将 **TextInput** 的属性修改为 **text**。

代码如下：

```
<mx:NumberValidator id="numValidator" source="{inputCurrency}"  
property="text"/>
```

当所有数据都通过验证以后，**formatter** 就会被呼叫。基础的 **Formatter** 类可以接受一些包含”#”号格式化字符串，用以代入一些格式化的数字或字符。比如说一个电话号码，它的格式化字符串将会是这样子：

(###) ###-####

**Formatter** 的申明方法如下：

```
<mx:PhoneFormatter id="phoneFormatter" formatString="(###) ###-  
###" validPatternChars="#-() "/>
```

如果要使用刚才宣告的 **formatter**，我们需要呼叫一个名为 **format** 的方法及将所需要的 **TextInput** 的 **text** 属性传递给它。

```
inputPhone.text = phoneFormatter.format(inputPhone.text);
```

需要注要的是在下列的例子里，如果输入的数据不能通过验证，程序将会清空用户所输入的内容以及返回一个错误信息。在实际应用上来说，这样很可能会为使用者带来不便。

代码如下：

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="600"  
height="400">  
<mx:Script>
```

```

<![CDATA[

    import mx.events.ValidationResultEvent;
    private var vResult:ValidationResultEvent;

//负责验证及格式化的事件处理器。
    private function dateFormat():void
    {
        vResult = dateVal.validate();

        if (vResult.type==ValidationResultEvent.VALID) {
            inputDate.text =
                dateFormatter.format(inputDate.text);
        } else {
            inputDate.text= "";
        }
    }

    private function phoneFormat():void {
        vResult = phoneValidator.validate();
        if (vResult.type==ValidationResultEvent.VALID) {
            inputPhone.text =
                phoneFormatter.format(inputPhone.text);
        } else {
            inputPhone.text= "";
        }
    }

    private function currencyFormat():void {
        vResult = numValidator.validate();
        if (vResult.type==ValidationResultEvent.VALID) {
            inputCurrency.text =
                currencyFormatter.format(inputCurrency.text);
        } else {
            inputCurrency.text= "";
        }
    }

    ]]>
</mx:Script>
<mx:DateFormatter id="dateFormatter" formatString="day: DD,
    month: MM, year: YYYY"/>
<mx:DateValidator id="dateVal" source="{inputDate}"
    property="text" inputFormat="mm/dd/yyyy"/>

```

```

<mx:PhoneNumberValidator id="phoneValidator" property="text"
    source="{inputPhone}" />
<mx:PhoneFormatter id="phoneFormatter" formatString="####-###-
    ###" validPatternChars="#-() " />
<mx:CurrencyFormatter id="currencyFormatter" currencySymbol="€"
    thousandsSeparatorFrom="." decimalSeparatorFrom="," />
<mx:NumberValidator id="numValidator" source="{inputCurrency}"
    property="text" />
<mx:Form>
    <mx:FormItem label="Currency Input">
        <mx:TextInput id="inputCurrency"
            focusOut="currencyFormat()" width="300"/>
    </mx:FormItem>
    <mx:FormItem label="Phone Number Input">
        <mx:TextInput id="inputPhone" focusOut="phoneFormat()"
            width="300"/>
    </mx:FormItem>
    <mx:FormItem label="Date Input">
        <mx:TextInput id="inputDate" focusOut="dateFormat();"
            width="300"/>
    </mx:FormItem>
</mx:Form>
</mx:VBox>

```

## 15.2 节. 如何创建一个自定义的 **Formatter**

### 15.2.1 问题

假如你想创建一个自定义的 formatter, 用来接收任何正确的字符串并以一个正确的格式来返回结果, 应怎么办呢?

### 15.2.2 解决办法

可以把 Formatter 类扩展并把 format 方法覆盖重写。

### 15.2.3 讨论

在 format 方法里头, 可以创建一个名为 SwitchSymbolFormatter 的成员并将一个特定格式的字符串传递给 SwitchSymbolFormatter 的 formatValue 方法, 这些包含着="#"号的字符串用来代入所需的内容的。 举例说, 如果把"###-###"这个格式及"123456"这个字符串的传递给 formatValue 方法, 它就会返回"123-456"。 这个结果正是由我们所自定义的 formatter 类中的 format 方法返回的。

这些包含”#”号的字符串，都会被所有传递给 Formatter 的字符所代入。其工作原理十分简单，就是以循环的方式，把这个字符串里每个字符逐个代入，最后得出一个已经格式化的字符串。代码如下：

```
package oreilly.cookbook

{
    import mx.formatters.Formatter;
    import mx.formatters.SwitchSymbolFormatter;

    public class ISBNFormatter extends Formatter
    {
        public var formatString : String = "####-##-###";

        public function ISBNFormatter()
        {
            super();
        }

        override public function format(value:Object):String
        {

            //我们需要去检查字符串的长度
            //ISBN的长度可以分别为10或13位数
            if( ! (value.toString().length == 10 ||
                    value.toString().length == 13) )
            {
                error="Invalid String Length";
                return "";
            }

            //根据我们的format string的="#"号总数来计算
            var numCharCnt:int = 0;
            for( var i:int = 0; i<formatString.length; i++ ) {
                if( formatString.charAt(i) == "#" ) {
                    numCharCnt++;
                }
            }

            //如果我们不能给格式化的字符串提供长度符合的字符串的话
            //这样子就会返回一个错误！
            if( ! (numCharCnt == 10 || numCharCnt == 13) ) {
                error="Invalid Format String";
                return "";
            }
        }
    }
}
```

```

    }

    //如果formatString跟所提供的值都有效，就把这个数目格式化。
    var dataFormatter:SwitchSymbolFormatter =
        new SwitchSymbolFormatter();
    return dataFormatter.formatValue( formatString, value );
}

}

```

## 15.3 节. 使用正则表达式创建国际化邮政编码 Validator

### 15.3.1 问题

你如何去验证一些南美洲国家的邮政编号呢？

### 15.3.2 解决办法

对于不同的国家，我们可以创建一系列不同的正则表达式使用群组。比如说，我们可以创建一个自定义的 Validator 类用以传递一个国家值。然后，根据这个值，呼叫相对应的 RegExp 里的 doValidation 方法。如果这个值跟 RegExp 相配的话，或者该国家本来就没有邮政编号的话，就会返回 true。否则，就会返回 false。

### 15.3.3 讨论

在自定义的 validator 里使用正则表达式，可以让你很方便地创建各种多功能的验证方法。如果没有它的帮助，validator 就只能为单一的字符串进行验证。同时，通过使用多个的正则表达式，你可很轻易地创建一个能验证复数字符串的类。

在以下的代码里，有着一个包含着不同国家的邮政编号哈希表(hash table)。当使用者选择了所需的国家并把这个值传递给 validator 后，它就会把相对应的正则表达式就会从这个哈希表(hash table)中选出。

代码如下：

```

private var countryHash:Object = {"Argentina":/[a-zA-Z]\d{4}[a-zA-Z]{3}/, "Brazil":/\d{5}-\d{3}/, "Mexico":/\d{5}/,
    "Bolivia":/\d{4}/, "Chile":/\d{7}/, "Paraguay":/\d{4}/,
    "Uruguay":/\d{5}/};

```

在以下的例子中，validator 的国家属性会被引用到 Validator 类的 doValidation 方法中：

代码如下：

```

//用来确认 country 这个值是否为空
if(countryHash[_country] != null) {
    //由哈希表(hash table)中读取并取得正确的 RegExp

```

```

var regEx:RegExp = countryHash[_country];
if(regEx.test(value as String)) {
    return results;
} else {
//如果邮编无效，则返回一个错误
    var err:ValidationResult = new ValidationResult(true, "", "", "Please Enter A Correct Postal Code");
    results.push(err);
}
} else {
    return results;
}

```

完整的 validator 代码如下：

```

package oreilly.cookbook
{
    import mx.validators.ValidationResult;
    import mx.validators.Validator;

    public class SouthAmericanValidator extends Validator
    {
        //把所有需要的国家名跟它们的邮政编号储存在哈希表 (hash table) 中
        private var countryHash:Object = {"Argentina":/[a-zA-Z]\d{4}[a-zA-Z]\d{3}/, "Brazil":/\d{5}-\d{3}/,
        "Mexico":/\d{5}/, "Bolivia":/\d{4}/, "Chile":/\d{7}/,
        "Paraguay":/\d{4}/, "Uruguay":/\d{5}/};

        private var results:Array;
        private var _country:String;

        public function SouthAmericanValidator() {
            super();
        }

        public function set country(str:String):void {
            _country = str;
            trace(_country);
        }

        //定义doValidation()方法
    }
}

```

```

override protected function doValidation(value:Object):Array
{
    // Clear results Array.
    results = [];

    //如果country这个值为空, 返回一个错误
    if (_country == "") {
        var err:ValidationResult = new ValidationResult(true,
            "", "", "Please Select a Country");
        results.push(err);
        return results;
    } else {

        //如果这个国家没有政编, 就不会返回任何错误
        if(countryHash[_country] != null) {
            //read from our hash table and get the correct
            RegExp

            var regEx:RegExp = countryHash[_country];
            if(regEx.test(value as String)) {
                return results;
            } else {

                // 如果该邮编号码不正确, 则返回一个错误
                var err:ValidationResult =
                    new ValidationResult(true, "", "", "Please
                    Enter A Correct Postal Code");
                results.push(err);
            }
        } else {
            return results;
        }
    }
    return results;
}
}

```

如果要应用这个自定义的 validator, 我们首先要确保其中包含国家的值是否就绪, 再去呼叫 validator 里的 doValidation 方法。在以下的例子里, 一个 ComboBox 组件就被用来设置 SouthAmericanValidator 里的国家的值。

代码如下:

```

<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300" xmlns:cookbook="oreilly.cookbook.*">
```

```

<cookbook:SouthAmericanValidator property="text" source="{zip}"
    required="true" id="validator"
    invalid="showInvalid(event)"/>
<mx:Script>
<! [CDATA[
    import mx.events.ValidationResultEvent;
    private function
        showInvalid(event:ValidationResultEvent):void
    {
        trace( " event " + event.message );
        zip.errorString = event.message;
    }
]]>
</mx:Script>
<mx:Form>
    <mx:FormItem label="Select country">
        <mx:ComboBox dataProvider="[{ 'Argentina', 'Brazil',
            'Mexico', 'Bolivia', 'Ecuador', 'Colombia',
            'Chile', 'Paraguay', 'Uruguay' }]"
            id="cb" change="validator.country = cb.selectedItem as
            String"/>
    </mx:FormItem>
    <mx:FormItem label="Enter zip ">
        <mx:TextInput id="zip"/>
    </mx:FormItem>
</mx:Form>
</mx:HBox>

```

## 15.4 节. 如何创建一个 Validator 去验证通用商品代码(UPC)

### 15.4.1 问题

怎样在一个窗体内验证复数的 UPC 码呢?

### 15.4.2 解决办法

首先创建一个 validator, 然后用它来检查 UPC 码的检验和是否存在及正确, 否则即返回一个错误。

### 15.4.3 讨论

一般商业中使用的 UPC 码，都是由 12 个数字所组成的。 它包含着一个隐藏的检验和，这个数是由每个相隔 3 位数之积及它们之和相加所得出的。通过以下代码来解释会比较容易去理解：

代码如下：

```
var sum:Number = 0;
for ( var i:Number=0; i < UPC.length; i += 2) {
    sum += Number(UPC.charAt(i)) * 3;
}
for ( i = 1; i < UPC.length-1; i += 2) {
    sum += Number(UPC.charAt(i));
}
var checkSum:Number = ( 10 - (sum % 10) ) % 10;
//如果检验和不对，就返回一个验证错误
if ( Number(UPC.charAt(11)) != checkSum ) {
    results.push(new ValidationResult(true, null, "invalidUPC",
        "Invalid UPC Number."));
}
return results;
}
```

以上的代码都是用来确保 UPC 码的有效性。接下的代码就更为直接：

代码如下：

```
package com.passalong.utils
{
    import mx.validators.Validator;
    import mx.validators.ValidationResult;
    import mx.controls.Alert;

    public class UPCValidator extends Validator
    {
        private var results:Array;
        public function UPCValidator()
        {
            super();
        }

        override protected function doValidation(value:Object):Array
        {
            //把数值转换成字符串，以方便对个别数字进行分析
            var UPC:String = String(value);
            // strip off decimal point from beginning -- added to
```

```

force recognition of leading zeros

    UPC = UPC.substring(1);

    var UPCnum:Number = Number(UPC);

    // 把results这个数组清空
    results = [];

    //呼叫doValidation()这个基础类
    results = super.doValidation(value);
    // Return if there are errors.

    if (results.length > 0)
        return results;

    // 如果输入数据不是数目字或不是一个数值，返回一个验证错误
    if (isNaN(UPCnum) || !value )
    {
        results.push(new ValidationResult(true, null, "NaN",
            "UPC required."));
        return results;
    }

    if ( UPC.length != 12 )
    {
        results.push(new ValidationResult(true, null,
            "invalidUPCLength", "Please enter a full 12-digit
            UPC."));
        return results;
    }
    else
    {
        var sum:Number = 0;

        for ( var i:Number=0; i < UPC.length; i += 2)
            sum += Number(UPC.charAt(i)) * 3;

        for ( i = 1; i < UPC.length-1; i += 2) {
            sum += Number(UPC.charAt(i));
        }
        var checkSum:Number = ( 10 - (sum % 10) ) % 10;

        //如果检验和不正确，返回一个验证错误
        if ( Number(UPC.charAt(11)) != checkSum )

```

```
    }

    results.push(new ValidationResult(true, null,
        "invalidUPC", "Invalid UPC Number."));

    return results;
}

return results;

}

}

}
```

特别鸣谢 Mike Orth 提供他的个人意见跟代码的指导。

## 15.5 节. 如何去验证多个 Combo Box 及 Radio Button 组件

### 15.5.1 问题

如何去为多个 radio button 及 combo box 组件加入验证功能，以确保最少其中一个项目会被选取？

### 15.5.2 解决办法

可以使用 NumberValidator 去检查 radio button 组件，再使用一个自定义的 Validator 去验证 combo box 组件。

### 15.5.3 讨论

如果要从 radio button 组件群中返回一个 ValidationResultEvent 事件，我们可以使用 NumberValidator 去检查 RadioButtonGroup 的 selectedIndex 属性是否为”-1”。如果为”-1”的话，即代表 radio button 中没有任何一项被选取了。而对于 combo box 组件，则可以创建一个自定义的 validator，用来检查 ComboBox 的 selectedItem 属性是否也空，而被选取的项目又是否有效。

以下有关 ComboBox 中自定义的 validator 的代码会比较容易理解，并且已加入注释：

代码如下：

```
package oreilly.cookbook

{
    import mx.validators.ValidationResult;
    import mx.validators.Validator;

    public class ComboValidator extends Validator
    {
        // 如果ComboBox中没有项目被选中，则返回这个错误信息
    }
}
```

```
public var error:String;  
//如果开发者把一个自定义的项目推进ComboBox的数组中(这种情况我见过不少)  
//我们就会把这个项跟已选取的项进行对比。  
  
public var prompt:String;  
public function ComboValidator() {  
    super();  
}  
  
//在这里我们进行两个检查：  
//1. comboBox中有没有项目被选中  
//2. 开发者有没有为comboBox加入自定义的项目  
//任何一个条件为ture的话，则返回一个错误  
  
override protected function doValidation(value:Object):Array  
{  
    var results:Array = [];  
    if(value as String == prompt || value == null) {  
        var res:ValidationResult = new ValidationResult(true, "",  
            "", error);  
        results.push(res);  
    }  
    return results;  
}  
}
```

还有一种更方便地进行复数验证的方法，那就是使用数组。首先，我们要把所有需要组件中的 validator 都加入到一个数组中，然后使用 `public static Validator.validateAll` 去存取数组中的所有 validator。这种方法对于需要进行复数属性的验证时是十分有用的。假如任何一个 validator 返回错误，这些错误会被收集并在 Alert 控件中显示出来：

代码如下

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="600"
           height="400" xmlns:cookbook="oreilly.cookbook.*"
           creationComplete="init()">>
<mx:Script>
<! [CDATA[
    import mx.events.ValidationResultEvent;
    import mx.validators.Validator;
    import mx.controls.Alert;

    [Bindable]
    private var validatorArr:Array;
//建立一个包含所有validator的数组，我们将会以一个方法对其进行检查

```

```

private function init():void {
    validatorArr = new Array();
    //push all the validators into the same array
    validatorArr.push(rbgValidator);
    validatorArr.push(toggleValidator);
    validatorArr.push(comboValidator);
}

//对所有validator数组中的项目进行验证。如果有任何错误的话，则会发出一个警告。
private function validateForm():void {
    // the validateAll method will validate all the Validators in an array
    //passed to the validateAll method
    var validatorErrorArray:Array =
        Validator.validateAll(validatorArr);
    var isValidForm:Boolean = validatorErrorArray.length
        == 0;
    if (!isValidForm) {
        var err:ValidationResultEvent;
        var errorMessageArray:Array = [];
        for each (err in validatorErrorArray) {
            errorMessageArray.push(err.message);
        }
        Alert.show(errorMessageArray.join("\n"),
            "Invalid form...", Alert.OK);
    }
}
]

]]>
</mx:Script>
<mx:StringValidator id="rbgValidator" source="{rbg}"
    property="selectedValue"/>
<mx:NumberValidator id="toggleValidator" source="{toggleButton}"
    property="selectedIndex" allowNegative="false"
    negativeError="Please select a Radio Button"/>
<cookbook:ComboValidator id="comboValidator" error="Please
    Select A State" prompt="{stateCB.prompt}"
    source="{stateCB}" property="selectedItem"/>
<mx:Form id="form">
    <mx:FormItem>
        <mx:ComboBox id="stateCB" dataProvider="{someDataProvider}"
            prompt="Select AState"/>
    </mx:FormItem>

```

```

<mx:FormItem>
    <mx:RadioButtonGroup id="rbg"/>
    <mx:RadioButton group="{rbg}" label="first" id="first"/>
    <mx:RadioButton group="{rbg}" id="second" label="second"/>
    <mx:RadioButton id="third" label="third" group="{rbg}"/>
</mx:FormItem>
<mx:FormItem>
    <mx:ToggleButtonBar id="toggleButton">
        <mx:dataProvider>
            <mx:Array>
                <mx:String> First Option </mx:String>
                <mx:String> Second Option </mx:String>
                <mx:String> Third Option </mx:String>
                <mx:String> Fourth Option </mx:String>
                <mx:String> Fifth Option </mx:String>
            </mx:Array>
        </mx:dataProvider>
    </mx:ToggleButtonBar>
</mx:FormItem>
</mx:Form>
<mx:Button label="validate" click="validateForm()"/>
</mx:VBox>

```

## 15.6 节. 如何在一个表单内通过 ToolTips 来反映一个错误

### 15.6.1 问题

如何在多个不同的控件焦点下，创建及显示复数的验证错误呢？

### 15.6.2 解决办法

首先使用 ToolTipManager 去创建一个新的 ToolTip 类，并把它放置在控件之上。然后创建一个 Style 对象，把它指派到刚刚创建的 ToolTip 类之中，以做出指定的字型及背景配色。

### 15.6.3 讨论

这种错误提示的方式其实是基于 ToolTip 组件，它会在 validator 报错的情况下被显示。你可以通过设定不同的风格来得出各种各样的视觉效果，例如：ackgroundColor, fontColor, fontType 等等。而使用 ToolTip 中的 setStyle 方法，则可以把预先设计好的风格套用到每个新的错误提示中。例如：

```
errorTip.setStyle("styleName", "errorToolTip");
```

要去显示复数的 tooltip 的话，首先把相关的控件定位到舞台上。例如：

```
var pt:Point = this.stage.getBounds(err.currentTarget.source);
var yPos:Number = pt.y * -1;
var xPos:Number = pt.x * -1;

//现在创建一个错误提示
var errorTip:ToolTip = ToolTipManager.createToolTip(err.message,
    xPos + err.currentTarget.source.width, yPos) as ToolTip;
```

当所有窗体都通过验证以后，这些提示都会通过 ToolTipManager 的 destroyToolTip 方法来进行删除。所需的代码如下：

代码如下：

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="500"
    height="400" xmlns:cookbook="oreilly.cookbook.*"
    creationComplete="init();">

<mx:Style>

/*在这里，css这个类是用来给予我们的tooltip一个像错误信息的外观*/
.errorToolTip {
    color: #FFFFFF;
    fontSize: 9;
    fontWeight: "bold";
    shadowColor: #000000;
    borderColor: #CE2929;
    borderStyle: "errorTipRight";
    paddingBottom: 4;
    paddingLeft: 4;
    paddingRight: 4;
    paddingTop: 4;
}

</mx:Style>
<mx:Script>
<![CDATA[
    import mx.controls.ToolTip;
    import mx.managers.ToolTipManager;
    import mx.events.ValidationResultEvent;
    import mx.validators.Validator;
    import mx.controls.Alert;

    [Bindable]
    private var validatorArr:Array;

    private var allErrorTips:Array;
]]>
```

```

private function init():void {
    validatorArr = new Array();
    validatorArr.push(comboValidator1);
    validatorArr.push(comboValidator2);
}

```

以下是验证过程的代码:

代码如下:

```

private function validateForm():void {
    //如果表格上已有错误提示，我们就去把它移除
    if (!allErrorTips) {
        allErrorTips = new Array();
    } else {
        for (var i:int = 0; i<allErrorTips.length; i++)
        {
            //移除tooltip
            ToolTipManager.destroyToolTip(allErrorTips[i]);
        }
    }
    //把数组清空
    allErrorTips.length = 0;
}

var validatorErrorArray:Array =
    Validator.validateAll(validatorArr);

```

如果 validatorErrorArray 这个数组为空，即告诉你这里没有任何错误可供抛出。否则，你就要创建相及按置对应的错误提示。

代码如下:

```

var isValidForm:Boolean = validatorErrorArray.length == 0;
if (!isValidForm) {
    var err:ValidationResultEvent;
    for each (err in validatorErrorArray) {
//通过设置目标的 x,y 坐标可以设定它的位置
//如果程序中已经存在一些版面管理的设定，我们会使用 getBounds 方法去取得它
//在舞台上的实际位置

```

由于 ErrorEvent 的目标属性就是负责抛出事件的控件或组件，所以这个属性会用来放置错误提示。

代码如下：

```
var pt:Rectangle =
    this.stage.getBounds(err.currentTarget.source);
var yPos:Number = pt.y * -1;
var xPos:Number = pt.x * -1;
//现在创建错误提示
var errorTip:ToolTip =
    ToolTipManager.createToolTip(err.message,
        xPos + err.currentTarget.source.width, yPos)
    as ToolTip;
//套用errorTip类中的selector
errorTip.setStyle("styleName", "errorToolTip");
//储存错误提示以方便在日后验证通过的时候把它移除
allErrorTips.push(errorTip);
}
}
]
]]>
</mx:Script>
<!-- our two validators -->
<cookbook:ComboValidator id="comboValidator1" error="Please
Select A State" prompt="{stateCB1.prompt}"
source="{stateCB1}" property="selectedItem"/>
<cookbook:ComboValidator id="comboValidator2" error="Please
Select A State" prompt="{stateCB2.prompt}"
source="{stateCB2}" property="selectedItem"/>
<mx:Form id="form">
<mx:FormItem>
<mx:ComboBox id="stateCB1"
dataProvider="{someDataProvider}" prompt="Select A
State"/>
</mx:FormItem>
<mx:FormItem>
<mx:ComboBox id="stateCB2"
dataProvider="{someDataProvider}" prompt="Select A
State"/>
</mx:FormItem>
</mx:Form>
<mx:Button label="validate" click="validateForm()"/>
</mx:VBox>
```

## 15.7 节. 如何使用正则表达式去定位电邮地址

### 15.7.1 问题

怎样辨识任何在文本中被输入或遇到的电邮地址呢?

### 15.7.2 解决办法

首先创建一个正则表达式去找出符合 name@host.com 格式的电邮地址。 接着，使用 global 标志来表示该表达式可以用来进行复数匹配。

### 15.7.3 讨论

我们所需的表达式就像以下这个:

```
var reg:RegExp = /\w+\@\w+\.\w{3}/g;
```

如果要在一大段的文字中找出相配的电邮地址，可以使用 String match 方法。 它会返回一个包含所有相配项目的数组，而这个方法也可以用来进行字符串搜寻或正则表达式搜寻。

## 15.8 节. 如何使用正则表达式去验证信用卡号码

### 15.8.1 问题

如何建立一个正则表达式，用来证验主流的信用卡包括 Visa, MasterCard, American Express, Discover 等等呢?

### 15.8.2 解决办法

根据以下的原则去创建一个正则表达式:

找出起首的几个数字跟哪个信用卡公司的卡相符

然后根据相对应的卡的种类，检查输入的数字是否正确。

### 15.8.3 讨论

实际上，所有主流信用卡号码都会存在着一些用以辨识的数字，我们可以通过这原理去创建所需要的正则表达式。 例如: MasterCard 会以 5 为起首, Visa 卡则会以 4 作为起首，还有 American Express 的卡都会以 30 作为起首，而 Discover 卡就会以 6011 为起首。 代码如下:  
(5[1-5]\d{14})|(4\d{12}(\d{3})?)(3[47]\d{13})|(6011\d{14})

就以这段代码(5[1-5]\d{14})为例，它可以用来验证任何不包含空格的 MasterCard 号码。 所以，在进行任何进一步处理之前，我们都应该把信用卡号里的空格都清除掉。 接下来的代码以”|”号这个标记隔开分别为 Visa, American Express 和 Discover 卡的表达式，而”|”号则代表着你可以跟其中任何一种卡进行匹配。

## 15.9 节. 如何使用正则表达式来验证 ISBN 号

### 15.9.1 问题

如何创建一个正则表达式用以验证国际标准图书号码(ISBN)呢?

### 15.9.2 解决办法

根据 ISBN 的特性, 它一般都是一个 10 位至 13 数字组成, 有时候会以”X”作为结尾, 并会以”-“号来分隔数组。 我们需要根据这个特性去创建所需的正则表达式。

### 15.9.3 讨论

这条表达式中的”^”号跟”\$”限制了该模式只可以存在于单行中。 如果一个文本中有多个不同的 ISBN 存在, 这些符号是可以除去的。

代码如下:

```
private var isbnReg:RegExp = /^(?=.{13}$)\d{1,5}([-]
) \d{1,7} \1 \d{1,6} \1 (\d|X)$/;
private function testISBN():void {
    var s:String = "ISBN 1-56389-016-X";
    trace(s.match(isbnReg));
}
```

“**^**”号表示该行一定要以这个模式作为起首, 而“**\$**”号则表示该行的结尾的模式, 而“**-**”号就是表示 ISBN 的数字组之间可以用“-”来分隔。

## 15.10 节. 如何通过指定字符类(**Explicit Character Class**)来创建正则表达式

### 15.10.1 问题

如果你想找出一文本中包含元音的词语, 怎样通过正则表达式的指定字符来进行模式匹配呢?

### 15.10.2 解决办法

可以使用”[“和”]”来包含所需要进行匹配的字符, 例如: [aeiou]来包括所有元音字母。

### 15.10.3 讨论

如果你需要在一个文本中为不同的模式进行匹配的话, 可以在表达式中加入不同的字符标记。 然后通过它们来呼叫你想进行匹配的字符类。 以下是几种常用的标记:

[ ] (square brackets)

用来定义一个字符类，它会包含所有可能用来匹配的字符。例如: /[aeiou]/

#### - (hyphen)

在一个字符类内，可以使用”-“号来指定一个字符的范围值。例如: /[A-Z0-9]/ 就表示要跟 A 至 Z 或者 0 至 9 内所有字符进行匹配。

#### / (backslash) 有错，原文用了”\“

在字符类里，可以插入一个”\“号来分隔”]“或”-“号等字符。比如说: /[-]d+/ 这个表达式，就表示在一个或以上的数字前对”+“或”-“号进行匹配。在一些字符类里面的 元字符 (metacharacter)，会默认为一个普通字符来处理，所以无须使用”\“来分隔。例如: /[\$£]/ 就表示跟”\$“或者”£“号进行匹配。详细情况请参考 Flex 使用文档中有关字符类的部份。

#### | (pipe)

表示可供选择的意思，用以选择及匹配”|“号两旁的内容。例如: /abc|xyz/ 则表示对”abc“或”xyz“进行匹配，只要任何一个条件乎合，即通过匹配，相当 OR 的意思。

如果想对奇数进行匹配，可以这样做:

```
var reg:RegExp = / [13579] /;
```

想对元音字进行匹配，可以这样做:

```
var vowels:RegExp = / [aeiou] /;
```

对非元音字进行匹配的话，可以这样做:

```
var notVowels:RegExp = /[^aeiou] /;
```

值得注意的是，“^”号的使用是不只限制于括号内的。在括号外使用的话，则表示这个字符串一定要发生在每行的开端。

## 15.11 节. 如何在正则表达式中使用字符类型

### 15.11.1 问题

怎样在一个正则表达式中对一些字符类型如(整数，字符，空格以及它们的反值)进行匹配？

### 15.11.2 解决办法

可以通过字符类型标记来实行。

### 15.11.3 讨论

通过使用字符类来进行字符匹配一种容易且非常有效的方法。首先，我们需要建一个包含”\“号的字符类型标记，它是用来告诉正则表达式处理引擎接下来的字符是一个字符类型而非普通字符。然后，在”\“号后加上所需的字符类。要注意的是，很多字符类型都包含着反值。例

如:

\d 表示对数目字进行匹配, 等同于[0-9]

\D 表示对任何非数目字进行匹配, 等同于[^0-9]

\b 表示对一个单词字符及非单词字符之间的内容进行匹配, 即一个字符串中的开头或结尾的字符为为单词字符的话, 它就会对这个字符串的开头或结尾进行匹配。

\B 表示对两个单词字符或两个非单词字符之间的内容进行匹配。

\f 表示对格式馈送类的字符进行匹配。

\n 表示对所有换行的字符进行匹配。

\r 对返回的字符进行匹配。

\s 对任何空白的字符进行匹配, 包括(space, tab, 换行等)。

\S 对任何非空白的字符进行匹配。

\t 对 tab 键的字符进行匹配。

\unnnn 用来对 Unicode 编码的字符进行匹配, 当中 nnnn 是代表一个十六进制的字符码。例如: \u263a 就代表一个笑脸的符号。

\v 对垂直馈入的字符进行匹配。

\w 对所有单词字符, 包括(A-Z, a-z, 0-9, or \_) 进行匹配。注意, 不包括任何非英语的字符, 例如: é, ñ, 或 ç 等。

\W 对所有非单词字符进行匹配。

\xnn 对特定的 ASCII 编码的字符进行匹配, 其中 nn 是一个十六进制的字符码。

\ 用以分隔含有特别意义的元字符。

. (点号) 对单个字符进行匹配。如果要以"."号来为换行字符"\n"进行匹配的话, "s"(意即 dotall)这个标记一个要预先设定好。详细情况请参考 Flex 使用文档中有关"s"(dotall)的部份。以下的例子展示了有关元字符的用法:

对"1"及两个随后的单词字符进行匹配的话, 可以这样做:

/1\w\w/;

对"1"及两个随后的非单词字符进行匹配的话, 可以这样做:

/1\W\W/;

对 5 个连续的数目字进行匹配的话, 可以这样做:

\d\d\d\d\d/;

虽然这样做会比较方便一点:

\d{5}/;

如果两个数目字之间存在空格的话, 可以这样做:

\d\b\d/;

如果三个数目字之间被某一个字符分隔开的话, 可以这样做:

\d.\d.\d/;

通过使用元字符, 你可以创建各类的表达式来为任何类型的字符, 包括: 整数, 英文字母, 空

格字符，或者它们的反值等。这样子你就可以建立各种强大而精炼的正则表达式。

## 15.12 节. 如何通过子表达式来验证 IP 地址是否有效呢？

### 15.12.1 问题

如何在一个文本中，寻找一些有效的 IP 地址呢？

### 15.12.2 解决办法

可以使用子表达式去验证 IP 地址中的每个 3 位数组是否有效。

### 15.12.3 讨论

根本上一章(章节 15.11)所学习过的内容，一个包含 1 位和 3 位数目字的 IP 地址可以用”d”标记来匹配的。

\d{1,3}

如果你想为 3 组包含 1 和 4 位数的数组进行匹配，可以这样做：

(\d{1,4}){3}

就如”/d{3}”可以用来匹配”333”一样，你可以通过创建子表达式来进行匹配。 子表达式本身就是一个独立的元素，跟表达中其他的模式有着相同的地位。 比如说一个 IP 地址，当中包含着 4 组 3 位数的数组。 你可以把它分成 3 组 3 位数组跟 1 组 3 位数组来处理，这样做会比较方便：

(\d{1,3}\.){3}\d{1,3}

可是，这个方法是有问题的，它给返回一个像这样子的字符串：838.381.28.999，而非一个有效的 IP 地址。 要解决这个问题，你需要把每个 3 位数组的最大值限制为 255。 通过使用子表达式，可以这样做：

代码如下：

```
(((\d{1,2})(1\d{2}))(2[0-4]\d)(25[0-5])\.){3}((\d{1,2})(1\d{2}))(2[0-4]\d)(25[0-5])  
))
```

首先，先对代码的前面部分中作个深入的了解：

((\d{1,2})(1\d{2}))(2[0-4]\d)(25[0-5])\.){3}

在这段代码中，你会看到 4 个主要部份：

(\d{1,2}) 表示包含着”1”或者”2”的数字 或者

(1\d{2}) 表示“1”随后跟着两个数目 或者

(2[0-4]\d) 表示“2”随后跟着两个“0 至 4”范围内的数字或者  
(25[0-5]) 表示“25”随后跟着 1 个“0 至 5”范围内的数位  
最后“\.”及{3}代表包含着 3 组根据以上规则的数组并以“.”号分隔。  
最后还有一组子表达式代表着第 4 组数组:  
((\d{1,2})|(1\d{2}))|(2[0-4]\d)|(25[0-5]))

它跟之前的表达式很相似，只是去除了尾端的“.”号。这是由于一个正确的 IP 地址(例如：192.168.0.1)，尾端的是没有“.”号的。

以下是一些有关子表达式的语法功能：

{n}代表最少执行 n 次。

{n,m}代表最少执行 n 次但不多于 m 次。

## 15.13 节. 如何使用正则表达式来为不同类型进行匹配?

### 15.13.1 问题

在一些情况下，如果想对一些以正则表达式的模式进行匹配的话，应怎么做呢？

### 15.13.2 解决办法

可以使用群组语法，例如“.”或“+”来为不同的群组进行不同次数的匹配。

### 15.13.3 讨论

正如在章节 15.12 中所见，“{}”号语法可以用来表示一个子表达式需要匹配的次数及该结果是否需要返回。举例说，如果你想对包含 0 至 4 范围内的字符串进行比较：

```
var firstString:String = "12430";
var secondString:String = "603323";
```

就以上的两个字符串来说，你可以使用以下的修饰器来对它们进行匹配：

?只进行 0 到 1 次的匹配

\*?进行 0 次或以上的匹配

+?进行 1 次或以上的匹配

需要记住的是，匹配跟返回匹配是两个不同的观念。例如：你想找出两个字符串之间是否只包含 0 至 4 范围内的数字，可以使用 RegExp 里的 test 方法，它会返回一个布尔值(true/false)。如果你想对一个字符串内的所有字符进行匹配，直到发现非匹配的字符为止，可以使用 String 里的 match 方法。如果你想对所有字符进行检查，无论它们是否匹配，则可以在正则表达式中(例如: /[0-4]+g/)使用 global 标记及 match 方法。

例如: /[abc]+/可以用来对 abbbca 或 abba 进行匹配，也可以从 abcss 中返回 abc.

\w+@\w+\.\w+ 是用针对电邮地址进行的验证。需要注要的是，在这里，“.”号是一个字符，用以分隔字符串而并非正则表达式中的语法部份。而"\w+@\w+"中的“+”号则表示这里可以有任何数目的字符，但其后一个要包括一个“@”号。

以下的代码展示了不同标记的用法以及对它们的结果作出说明

代码如下:

```
var atLeastOne:RegExp = /[0-4]+/g;
var zeroOrOne:RegExp = /[0-4]*/g;
var atLeastOne2:RegExp = /[0-4]+?/g;
var zeroOrOne2:RegExp = /[0-4]*?/g;
var firstString:String = "12430";
var secondString:String = "663323";

firstString.match(atLeastOne)); //returns "1243"
secondString.match(atLeastOne)); //returns "3323" because we want
as many characters as will match
firstString.match(zeroOrOne)); //returns      "1243"    the first few
characters match
secondString.match(zeroOrOne)); //returns "" because the first few
characters don't match, we stop looking
firstString.match(atLeastOne2)); //returns "1,2,4,3" because all we
need is one match
secondString.match(atLeastOne2)); //returns "3,3,2,3"
firstString.match(zeroOrOne2)); //returns ""
secondString.match(zeroOrOne2)); //returns ""
zeroOrOne2.test(firstString)); //returns true
zeroOrOne2.test(secondString)); //returns false
```

## 15.14 节. 如何用正则表达式来为行的开端或结尾进行匹配?

### 15.14.1 问题

如果某些模式只存在于每行的开端或结尾，或者这个模式占据了一整行，应该怎样进行匹配呢？

### 15.14.2 解决办法

可以在正则表达式中加入”^”和”\$”记号。

### 15.14.3 讨论

当需要为一个存在于单行，行首或行尾的模式进行匹配的时候，可以在表达式的开头加入”^”号，用以表示该模式一定要在行的起首。而在表达式的结尾加入”&”号的话，则表示这个模式一定要存在于该行的结尾。

举例说，如果要为一个任何长度的jpg或jpeg文件名进行匹配，而这个文件名是跟该行的其他字符分隔开的话，可以这样做：

```
/^.+?\.\jpe?g$/i
```

如果一些字符串只存在于行的尾端, 要对其进行匹配话可以这样做:

```
\w+?$/;
```

相反地, 如果它发生在行的开端, 则可以这样做:

```
/^\\w+?/;
```

## 15.15 节. 如何使用逆向引用?

### 15.15.1 问题

有些情况下, 如果你想对一个模式进行匹配及根据这个结果来找出下一个可能匹配之处。举例说, 对 HTML 标签进行匹配的话, 应怎么办呢?

### 15.15.2 解答

可以在正则表达式中加入逆向引用来检查最近的匹配结果。

### 15.15.3 讨论

在 Flash Player 的正则表达式处理引擎中, 逆向引用的结果会以一个列表的形式储存, 最多可以储存达 99 个的相配结果。在其中, “\1”表示最近的匹配结果, 而”\2”则表示前一个结果。同样地, 在 String 类中有个叫 replace 的方法, 是用来对其他的正则表达式进行匹配的。在这里, 最近的匹配结果会以”\$1”来表示。

为了确保每对的 HTML 标签都会被匹配(例如: <h2>和</h2>), 以下的例子使了”\1”这个逆向引用标记来返回最近的结果:

代码如下:

```
private var headerBackreference:RegExp = /<H([1-6])>.*?<\\H\\1>/g;
private function init():void {
    var s:String = "<BODY> <H2>Valid Chocolate</H2> <H2>Valid
Vanilla</H2> <H2>This is not valid HTML</H3></BODY>";
    var a:Array = s.match(headerBackreference);
    if(a != null) {
        for(var i:int = 0; i<a.length; i++) {
            trace(a[i]);
        }
    }
}
```

此外, 你也可以使用逆向引用来为有效的 URL 地址加入”<a>”卷标以创建一个超链接。这种

逆向引用的表示方法跟之前的会有一点点分别。我们会通过以下的代码及分析来理解它们的区别：

代码如下：

```
private var domainName:RegExp =
  /(ftp|http|https|file):\/\/[\s]+(\b|$)/gim;
private function matchDomain():void {
  var s:String = "Hello my domain is http://www.bar.com, but I also like http://foo.net as well as www.baz.org";
  var replacedString = (s.replace(domainName,
    '<a href="$&"$&</a>').replace(/(^\/)(www[\s]+(\b|$))/gim,
    '$1<a href="http://$2">$2</a>');
}
```

首先对有效的 URL 地址进行匹配：

```
/(ftp|http|https|file):\/\/[\s]+(\b|$)/gim;
```

接受，把这些有效的 URL 地址嵌入”“标记中间：

```
s.replace(domainName, '<a href="$&"$&</a>')
```

最后，你会得出以下结果：

代码如下：

```
Hello my domain is <a href="http://www.bar.com">http://www.bar.com</a>, but I also like
<a href="http://foo.net">http://foo.net</a> as well as www.baz.org
```

显然地，这个结果并不合乎理想。在原先的 RegExp 中，只有那些以 ftp, http, https 或 file 开头的字符串才会被匹配，而 [www.baz.org](#) 这个字符串则不合乎条件。以下的 replace 语句会在所有匹配结果中，寻找那些包含”www”但没有”/”号在它跟前的字符串。

```
replace(/(^\/)(www[\s]+(\b|$))/gim,$1<a href="http://$2">$2</a>))
```

“\$1”及”\$2”表示第一及第二个匹配结果，而在这里第二个结果正是我们所需要的 URL 名字。

## 15.16 节. 如何使用向前查找组或向后查找组语法？

### 15.16.1 问题

如果一个模式中规定，在它的之前或之后不能存在某些字符，应怎样匹配呢？

### 15.16.2 解决办法

可以使用负向前查找组”?”或负向后查找组”?!”来表示那些字符是不应该存在于某个模式之前或之后。而使用正向前查找组”?=?”或正向后查找组”?<=?”则表示这些字符是应该存在于某个模式之前或之后。

### 15.16.3 讨论

有些时候，你想某个模式以后的表达式进行匹配，但又不想把这个模式包括在你的结果里面。例如：你想把所有”\$”号之后的数目字找出，但又不想包括”\$”号本身。

400 boxes at \$100 per unit and 300 boxes at \$50 per unit.

可以在正则表达式加入以下的正向后查找组语法：

```
/(?<=\$)\d+/  
/
```

除此之外，你也可以使用负向后查找组语法找出所有不包括”\$”号在其跟前的字符：

```
\b(?<!\$)\d+\b/  
/
```

值得注意的是，负向后查找组语法其实就是把正向后查找组语法里的”=”号取代为”!”，以表示”\$”号不能存在于匹配的对象中。如果只是想把价钱里的数字抽出，可以使用正向前查找组语法：

代码如下：

```
private var lookBehindPrice:RegExp = /(?<=[\$|\€])[0-9.]+/g;  
  
private function matchPrice():void {  
    var s:String = "dfsdf24ds: €23.45 ds2e4d: $5.31 CFMX1: $899.00  
    d3923: €69";  
    trace(s.match(this.lookBehindPrice));  
}
```

如果要为一个字符串中的变量宣告进行匹配的话，可以使用正向前查找组语法：

代码如下：

```
private var lookBehindVariables:RegExp = /(?=<var>)[0-9_a-zA-Z]+/g;  
private function matchVars():void {  
    var s:String = " private var lookAheadVariables:RegExp = /blah/  
    private var str:String = 'foo';  
    trace(s.match(lookBehindVariables));  
}
```

如果你想对所有非”.jpg”的图片进行匹配的话，你可以使用负向前查找组语法：

```
var reg:RegExp = /pic(?!\\.jpg)/;
```

# 第十六章. 图表 (NA)

Flex 图表框架是一个功能强大的数据控件系列，它提供了丰富的数据支持使你可以为多种数据类型创建深入的和可交互的图表。

Flex 定义了一些最常用的图表，例如柱状图、饼图和列状图，并且可以使用 Flex 很大程度上改变这些图表的外观。

每个图表都包含一个 `ChartSeries` 对象，用于在图表上显示由数据提供器提供的数据。

`ChartSeries` 对象可以作为一个数据项目的渲染器，它可以显示数据提供器的独有字段，还可以为图表添加效果。除此之外，使用 `ChartSeries` 对象独有的属性还能处理用户交互。

Flex 图表框架是一个宽泛的主题，所以本章只能涵盖很少一部分内容。

需要注意的是 Flex 图表组件是包含在 Flex data visualization 包中，这个包只有在 flex 专业版中才能找到，在 Flex 标准版以及免费的 flex3 SDK 中是没有的。

## 16.1.节 创建一个图表

### 16.1.1. 问题

我想在程序中添加一个图表。

### 16.1.2. 解决办法

首先要创建目标类型的图表，然后为该类型的图表添加一个合适的 `CharSeries` 对象，最后为该图表绑定数据提供器(`dataProvider`)。

### 16.1.3. 讨论

图表有多种类型，在使用之前必须清楚工具箱中有哪些图表可供调用。

每一种图表都有一个 `ChartSeries` 对象与之对应。要将数据显式地表示为某个特定的图表，就要先添加对应的数列类型然后绑定到一个数据提供器上。`ChartSeries` 对象定义了在图表的 x 轴和 y 轴上显示何种数据以及数据列的名称。数据列名称可以添加滤镜来显示，包括阴影、模糊或者发光的效果。

根据数据的不同格式，你可能需要自定义一个横向或纵向的坐标。如果数据是一个集合，如日期、国家、人，你就需要使用类坐标(`CategoryAxis`)。如果数据是单纯的数字，就要使用线性坐标(`LinearAxis`)。

图表的数据提供器可以是一个数组或多个类的集合，也可以是 XMLList 对象。如果你要在图表标签上设置一个数据提供器，那么数据列对象就会继承这个数据提供器，或者你可以选择为每个数据列对象单独地指定一个数据提供器。不同的数据列可以使用不同的数据提供器。一个图表不需要使用数据提供器里面的所有数据，可以只使用指定的部分数据。

可以用与以下代码创建条状图和饼状图：

Code View:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal" backgroundColor="#FFFFFF">
    <mx:Script>
        <![CDATA[
            // a basic data set

            [Bindable] public var chartDP:Array = [
                {day: 'Monday', rainfall:10,elevation:100,temperature:78},
                {day: 'Tuesday', rainfall:7,elevation:220,temperature:66},
                {day: 'Wednesday', rainfall:5,elevation:540,temperature:55},
                {day: 'Thursday', rainfall:8,elevation:60,temperature:84},
                {day: 'Friday', rainfall:11,elevation:390,temperature:52},
                {day: 'Saturday', rainfall:12,elevation:790,temperature:45},
                {day: 'Sunday', rainfall:14,elevation:1220,temperature:24}
            ];
        ]]>
    </mx:Script>
    <mx:ToggleButtonBar dataProvider="{simpleCharts}"
        direction="vertical" />
    <mx:ViewStack id="simpleCharts" >
        <mx:Canvas label="Bar">
            <mx:BarChart dataProvider="{chartDP}" >
                <mx:verticalAxis>
                    <mx:CategoryAxis
                        dataProvider="{chartDP}"
                        categoryField="day" />
                </mx:verticalAxis>
                <mx:series>
```

```

        <!-- bar chart uses a BarSeries -->
        <mx:BarSeries
            yField="day" xField="rainfall"
            displayName="day" />
        </mx:series>
    </mx:BarChart>
</mx:Canvas>

<mx:Canvas label="Pie">
    <mx:PieChart dataProvider="{chartDP}" >
        <!-- no axes need to be defined in a pie chart -->
        <mx:series>
            <!-- pie chart uses a pie series -->
            <mx:PieSeries
                field="rainfall"
                nameField="day"
                labelPosition="callout"
                displayName="rainfall" />
        </mx:series>
    </mx:PieChart>
</mx:Canvas>

</mx:ViewStack>

</mx:Application>

```

柱状图和 HighLowOpenClose 图表则需要一些不同类型的数据集合：

Code View:

```

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal" backgroundColor="0xFFFFFFFF">
    <mx:Script>
        <! [CDATA[
            // the field names don't need to be 'high', 'open', 'low',
            and 'close', but you need four different fields to get this kind
            of chart to work
            [Bindable] public var highLowChartDP:Array = [
                {date:"1-Aug-05", open:42.57, high:43.08, low:42.08, close:42.75},
                {date:"2-Aug-05", open:42.89, high:43.5, low:42.61, close:43.19},
                {date:"3-Aug-05", open:43.19, high:43.31, low:42.77, close:43.22},
                {date:"4-Aug-05", open:42.89, high:43, low:42.29, close:42.71},

```

```

        {date: "5-Aug-
05", open:42.49,high:43.36,low:42.02,close:42.99},
        {date: "8-Aug-
05", open:43,high:43.25,low:42.61,close:42.65},
        {date: "9-Aug-
05", open:42.93,high:43.89,low:42.91,close:43.82},
        {date: "10-Aug-
05", open:44,high:44.39,low:43.31,close:43.38},
        {date: "11-Aug-
05", open:43.39,high:44.12,low:43.25,close:44},
        {date: "12-Aug-
05", open:43.46,high:46.22,low:43.36,close:46.1}
    ];
} ]>
</mx:Script>
<mx:CandlestickChart
   dataProvider="{highLowChartDP}"
    showDataTips="true">
    <mx:verticalAxis>
        <mx:LinearAxis minimum="40" maximum="50" />
    </mx:verticalAxis>
    <mx:horizontalAxis>
        <mx:CategoryAxis categoryField="date" />
    </mx:horizontalAxis>
    <mx:series>
        <mx:CandlestickSeries
           dataProvider="{highLowChartDP}"
            openField="open"
            highField="high"
            lowField="low"
            closeField="close"
            displayName="Rainfall"/>
    </mx:series>
</mx:CandlestickChart>
</mx:Application>

```

## 16.2.节. 为图表添加效果

### 16.2.1. 问题

我想要为图表添加一些效果。

### 16.2.2. 解决办法

想为图表的坐标轴或数列添加效果，要使用坐标轴定义的<mx:rollOverEffect> 或者 <mx:rollOutEffect>标签。

### 16.2.3. 讨论

来自 mx.effects 包的任何效果都能在图表上的数列或坐标轴添加的效果。一个简单的翻转效果就可以使图表的显示效果和使用效果大大提高。以下是一个简单的效果，当鼠标离开图表，图表就会变得透明，当鼠标放在图表上，图表就变得不透明。

Code View:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal" backgroundColor="#FFFFFF">
    <mx:Script>
        <![CDATA[
            [Bindable] public var chartDP:Array = [
                {day: 'Monday', rainfall:10,elevation:100,temperature:78},
                {day: 'Tuesday', rainfall:7,elevation:220,temperature:66},
                {day: 'Wednesday', rainfall:5,elevation:540,temperature:55},
                {day: 'Thursday', rainfall:8,elevation:60,temperature:84},
                {day: 'Friday', rainfall:11,elevation:390,temperature:52},
                {day: 'Saturday', rainfall:12,elevation:790,temperature:45},
                {day: 'Sunday', rainfall:14,elevation:1220,temperature:24}
            ];
        ]]>
    </mx:Script>
    <mx:AreaChart dataProvider="{chartDP}" >
        <mx:horizontalAxis>
            <mx:CategoryAxis
                dataProvider="{chartDP}"
                categoryField="day" />
        </mx:horizontalAxis>
        <mx:series>
            <mx:AreaSeries alpha=".5"
                yField="rainfall">
```

```
displayName="rainfall">
```

以下的部分是为区域图表添加的鼠标翻转的逐渐透明效果。这些效果只会在数列的部分出现，不会影响整个图表。

```
<mx:rollOverEffect>
    <mx:Fade alphaFrom=".5" alphaTo="1"
              duration="500" />
</mx:rollOverEffect>
<mx:rollOutEffect>
    <mx:Fade alphaFrom="1" alphaTo=".5"
              duration="500" />
</mx:rollOutEffect>
</mx:AreaSeries>
</mx:series>
</mx:AreaChart>
</mx:Application>
```

想要为图表增添更多的效果，可以使用 SeriesInterpolate、SeriesZoom 和 SeriesSlide 这三个效果属性值来增添动画效果。 SeriesInterpolate 可以在新旧数据变换时使数据点移动。 SeriesZoom 使旧数据缩小到不可见后再将新数据从不可见放大。 SeriesSlide 使旧数据滑出图表后让新数据滑入图表。这些事件通常添加到数列对象的 showDataEffect 和 hideDataEffect 属性中。 SeriesInterpolate 只能添加到 showDataEffect 中，当将其添加到 hideDataEffect 中是没有效果的。

以下例子显示了一个图表的滑动效果，当转换两个数据集合的时候就会显示出图表数列滑动的效果。若要显示 SeriesZoom 效果，就要把其他效果的代码加上注释，然后去掉 SeriesZoom 效果的注释。 要显示 SeriesInterpolate 效果，同样要把其他效果的代码加上注释，然后去掉 SeriesInterpolate 效果的注释，并且要把 ColumnSeries 标签中的 hideDataEffect 属性去掉。

Code View:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal" backgroundColor="0xFFFFFFFF">
    <mx:Script>
        <![CDATA[
            [Bindable] public var chartDP1:Array = [
                {day: 'Monday', rainfall:10,elevation:100,temperature:78},
                {day: 'Tuesday', rainfall:7,elevation:220,temperature:66},
                {day: 'Wednesday', rainfall:5,elevation:540,temperature:55},
                {day: 'Thursday', rainfall:8,elevation:60,temperature:84},
                {day: 'Friday', rainfall:12,elevation:150,temperature:90}
            ];
        ]]>
    </mx:Script>
</mx:Application>
```

```

{day: 'Friday', rainfall:11,elevation:390,temperature:52} ,  

{day: 'Saturday', rainfall:12,elevation:790,temperature:45} ,  

{day: 'Sunday', rainfall:14,elevation:1220,temperature:24}  

];  
  

[Bindable] public var chartDP2:Array = [  
  

{day: 'Sunday', rainfall:10,elevation:100,temperature:78} ,  

{day: 'Saturday', rainfall:7,elevation:220,temperature:66} ,  

{day: 'Friday', rainfall:5,elevation:540,temperature:55} ,  

{day: 'Thursday', rainfall:8,elevation:60,temperature:84} ,  

{day: 'Wednesday', rainfall:11,elevation:390,temperature:52} ,  

{day: 'Tuesday', rainfall:12,elevation:790,temperature:45} ,  

{day: 'Monday', rainfall:14,elevation:1220,temperature:24}  

];  

]]>  

</mx:Script>  
  

<mx:SeriesSlide id="dataIn" duration="500" direction="up"/>  

<mx:SeriesSlide id="dataOut" duration="500" direction="up"/>  
  

<mx:SeriesZoom id="dataOut" duration="500"/>  
  

<mx:SeriesInterpolate id="dataIn" duration="1000"/> -->  

<mx:BarChart id="rainfallChart" dataProvider="{chartDP1}" >  

<mx:horizontalAxis>  

<mx:CategoryAxis  

dataProvider="{chartDP1}"  

categoryField="day" />  

</mx:horizontalAxis>  

<mx:series>  

<mx:ColumnSeries  

yField="rainfall" xField="day"  

displayName="rainfall"  

showDataEffect="{dataIn}" hideDataEffect="{dataOut}" />

```

```
</mx:series>
</mx:BarChart>

<mx:HBox>
    <mx:RadioButton groupName="dataProvider"
        label="Data Provider 1" selected="true"
        click="rainfallChart.dataProvider=chartDP1;" />
    <mx:RadioButton groupName="dataProvider"
        label="Data Provider 2"
        click="rainfallChart.dataProvider=chartDP2;" />
</mx:HBox>
</mx:Application>
```

## 16.3.节 在图表中选择一个区域

### 16.3.1.问题

我需要在图表上选择某个区域或者某些元素。

### 16.3.2. 解决方法

首先使用图表的 `selectionMode` 属性设置需要选择的区域类型，然后使用鼠标键盘或者程序来选择图表中的元素。

### 16.3.3. 讨论

与其他列表组件相似，图表的元素是可选的。这有利于用数据网格或二级图表显示数据点上更多的细节。要使图表可选，就要设定 `selectionMode` 属性为 `single` 或者 `multiple` (`selectionMode` 属性的默认设置是 `none`)。`selectionMode` 属性设置为 `none` 就代表图表不可选；`single` 属性允许一次选择一个元素；`multiple` 属性允许多个元素同时被选择。

选择图表元素可以通过鼠标键盘，或者通过拖拽一个矩形框来选择多个点，或者使用 ActionScript 程序来实现。当选择图表上多个元素时，第一个被选择的元素会被系统认为是定位点，而最后一个被选择的元素会被认为是插入符号。用鼠标选择的方式是非常直观的，点击图表的元素就可以将其选中。要选择多个元素，只要按住 Shift 键可以选择连续的元素，按住 Ctrl 键(在 Mac 系统上是 Command 键)可以逐一选择不连续的元素。使用键盘上的左右方向键可以遍历一个图表上的所有数据列。当 `selectionMode` 属性设置为 `multiple`，在图表中用鼠标脱出一个矩形区域可以选择该矩形区覆盖的所以元素。

用程序创建选区稍微负责一点。图表选区的 API 提供选择元素和操作所选区域的功能。可以使用以下 `ChartSeries` 对象的属性来获得和设置选区状态：

- selectedItem
- selectedItems
- selectedIndex
- selectedIndices

除了上述的解决办法，还可以使用 ChartBase 类的方法来实现：

- getNextItem()
- getPreviousItem()
- getFirstItem()
- getLastItem()

使用图表 Change 事件可以监听用户是否使用鼠标或者键盘改变选区，但是对程序改变选区的情况无效。

下面的例子展示的是在图表中选中了某个数据条后，在 DataGrid 中就会相应地显示该数据条的数据。在程序中还有上一个和下一个按钮，它们可以实现用程序更换选区。

Code View:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical" backgroundColor="0xFFFFFFFF">
    <mx:Script>
        <![CDATA[
            [Bindable] public var chartDP:Array = [
                {day: 'Monday', rainfall:10,elevation:100,temperature:78},
                {day: 'Tuesday', rainfall:7,elevation:220,temperature:66},
                {day: 'Wednesday', rainfall:5,elevation:540,temperature:55},
                {day: 'Thursday', rainfall:8,elevation:60,temperature:84},
                {day: 'Friday', rainfall:11,elevation:390,temperature:52},
                {day: 'Saturday', rainfall:12,elevation:790,temperature:45},
                {day: 'Sunday', rainfall:14,elevation:1220,temperature:24}
            ];
        ]]>
```

```

private function changeSelectedIndex(offset:int):void
{
    barSeries.selectedIndex+=offset;
    onSelectionChange();
}

private function onSelectionChange():void
{
    // programmatic changes to chart selection don't
    fire a Change event,
so we need to manually reset
    // the dataProvider of our detail grid when we
programmatically changethe selection
    detailGrid.dataProvider =
        barChart.selectedChartItems;
}
]]>
</mx:Script>
<!-- use the change event to set the dataProvider of our
detail grid to our chart'
s selected items --&gt;
&lt;mx:BarChart id="barChart" dataProvider="{chartDP}"
selectionMode="multiple" change="onSelectionChange()"&gt;
&lt;mx:verticalAxis&gt;
&lt;mx:CategoryAxis
    dataProvider="{chartDP}"
    categoryField="day" /&gt;
&lt;/mx:verticalAxis&gt;
&lt;mx:series&gt;
    &lt;mx:BarSeries id="barSeries" selectedIndex="0"
        yField="day" xField="rainfall"
        displayName="day" /&gt;
&lt;/mx:series&gt;
&lt;/mx:BarChart&gt;
&lt;mx:HBox&gt;
    &lt;mx:Button click="changeSelectedIndex(1)" label="Previous" /&gt;
    &lt;mx:Button click="changeSelectedIndex(-1)" label="Next" /&gt;
&lt;/mx:HBox&gt;
&lt;mx:DataGrid id="detailGrid" &gt;
&lt;mx:columns&gt;
    &lt;mx:DataGridColumn dataField="xValue"
        headerText="rainfall" /&gt;
</pre>

```

```

<mx:DataGridColumn dataField="yValue" headerText="day"
/>
</mx:columns>
</mx:DataGrid>
</mx:Application>

```

## 16.4.节 设计图表的刻度线

### 16.4.1. 问题

我需要自定义图表的刻度线

### 16.4.2. 解决办法

使用 AxisRenderer 中的样式来设置图表刻度线的外观

### 16.4.3.讨论

通过样式，Flex 提供了大量控制刻度线外观的方法。在 Flex 图表中有两种类型的刻度线，分别是大刻度线和小刻度线。大刻度线与坐标轴的标签相对应，小刻度线通常用在大刻度线之间。

在 AxisRenderer 中可以定义图表刻度线的外观样式。大刻度线可用 tickPlacement、tickAlignment 和 tickLength 来定义样式；小刻度线可用 minorTickPlacement、minorTickAlignment 和 minorTickLength 来定义样式。

tickMarkPlacement 和 minorTickPlacement 定义刻度线在坐标线的那个位置出现。表 16-1 列出了相关参数值。

Table 16-1. 刻度线的值和位置	
Value	Placement
cross	Across the axis
inside	Inside the axis
outside	Outside the axis
none	No tick mark

以下这个例子将大刻度线定在坐标线的内侧，小刻度线定在坐标线的外侧。大刻度线是 5 像素长，小刻度线是 10 像素长。

Code View:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
```

```

layout="horizontal" backgroundColor="#FFFFFF">
<mx:Script>
<! [CDATA[
[Bindable] public var chartDP:Array = [
{day: 'Monday', rainfall:10,elevation:100,temperature:78},
{day: 'Tuesday', rainfall:7,elevation:220,temperature:66},
{day: 'Wednesday', rainfall:5,elevation:540,temperature:55},
{day: 'Thursday', rainfall:8,elevation:60,temperature:84},
{day: 'Friday', rainfall:11,elevation:390,temperature:52},
{day: 'Saturday', rainfall:12,elevation:790,temperature:45},
{day: 'Sunday', rainfall:14,elevation:1220,temperature:24}
];
]]>
</mx:Script>

<mx:Style>
.customTicks {
    tickPlacement:inside;
    minorTickPlacement:outside;
    tickLength:5;
    minorTickLength:10;
}
</mx:Style>

<mx:Canvas label="Area">
<mx:AreaChart dataProvider="{chartDP}" >
<mx:horizontalAxis>
<mx:CategoryAxis
    dataProvider="{chartDP}"
    categoryField="day" />
</mx:horizontalAxis>
<mx:verticalAxis>
<mx:LinearAxis id="vertAxis" />
</mx:verticalAxis>
<mx:verticalAxisRenderers>

```

```
<mx:AxisRenderer axis="{vertAxis}"  
styleName="customTicks" />  
</mx:verticalAxisRenderers>  
<mx:series>  
    <mx:AreaSeries  
        yField="rainfall"  
        displayName="rainfall" />  
    </mx:series>  
</mx:AreaChart>  
</mx:Canvas>  
</mx:Application>
```

## 16.5.节 为图表创建自定义标签

### 16.5.1. 问题

我想自定义图表的标签。

### 16.5.2. 解决办法

使用样式和标签函数。

### 16.5.3. 讨论

图表中包含两种类型的标签：坐标轴标签和数据标签。坐标轴标签用于显示坐标轴上的点的数值，使用标签函数能自定义坐标轴标签。数据标签用于在数据点的位置和图表元素的位置显示数据值。

使用坐标轴标签可以很大程度上控制坐标轴标签的形式。例如你需要特殊的数据格式或者通用的格式，就可以使用标签函数。标签函数通用适用与数据标签。

标签函数对于数字坐标轴、分类坐标轴和数列的用法略有不同，如下所示。

数字坐标轴的标签函数有如下写法：

Code View:

```
function _name(labelValue:Object, previousLabelValue:Object, axis:IAxis):String
```

其参数如下：

**labelValue**

当前标签的数值；

### previousLabelValue

该标签之前的一个标签的值。如果当前标签是第一个标签，那么 previousLabelValue 的值就是 null。

### axis

坐标轴对象，和 CategoryAxis 或者 NumericAxis 对象相似。

分类坐标轴的标签函数有如下写法：

Code View:

```
function_name(labelValue:Object, previousLabelValue:Object, axis:axis_type,  
labelItem:Object):String
```

### categoryValue

要描绘的类别的值。

### previousCategoryValue

在坐标轴上前一个类别的值

### axis

正在渲染的类别坐标轴的值。

### categoryItem

当前描绘的数据提供器提供的项目

以下例子使用了一个 CurrencyFormatter 来为纵坐标标签和数据标签定义格式

Code View:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"  
layout="horizontal" backgroundColor="#FFFFFF"  
initialize="onInit()>  
<mx:Script>  
<! [CDATA [  
    import mx.charts.chartClasses.Series;  
    import mx.charts.ChartItem;  
    import mx.charts.chartClasses.IAxis;  
    import mx.formatters.CurrencyFormatter;
```

```

[Bindable] public var chartDP:Array = [
    {month: 'Jan', costs:10000,sales:100000},
    {month: 'Feb', costs:7000,sales:220000},
    {month: 'Mar', costs:5000,sales:540000},
    {month: 'April', costs:8000,sales:60000},
    {month: 'May', costs:11000,sales:390000},
    {month: 'June', costs:12000,sales:790000},
    {month: 'July', costs:14000,sales:1220000}
];

private var formatter:CurrencyFormatter;

private function onInit():void
{
    formatter = new CurrencyFormatter();
    formatter.currencySymbol = '$';
    formatter.precision = 0;
    formatter.useThousandsSeparator = true;
}

private function currencyAxisLabel(value:Object,
    previousValue:Object, axis:IAxis):String
{
    return formatter.format(value);
}

] ]>
</mx:Script>
<mx:LineChart dataProvider="{chartDP}" >
    <mx:horizontalAxis>
        <mx:CategoryAxis
            dataProvider="{chartDP}"
            categoryField="month"
        />
    </mx:horizontalAxis>
    <mx:verticalAxis>
        <mx:LinearAxis labelFunction="{currencyAxisLabel}" />
    </mx:verticalAxis>
    <mx:series>
        <mx:LineSeries
            yField="costs" xField="month"
            displayName="Costs" />

```

```
</mx:series>
</mx:LineChart>
</mx:Application>
```

## 16.6.节 创建列状图的明细项目效果

### 16.6.1.问题

我想在查看图表细目表的时候添加一个效果。

### 16.6.2. 解决办法

为所选的图表项目创建一个新的数组，然后把列状图的数据提供器绑定到这个数组上。使用 SeriesZoom 在总体数据集和详细数据集中转换。

### 16.6.3.讨论

查看图表的细目表是一个用户界面的概念，它允许用户在一个较大的数据选择其中一个特定的数据项目来查看。

图表的明细项目效果使你能够选择图表的某一项目然后查看这个项目更详细的数据信息。

这个效果可以通过设置图表的数据提供器来实现。

当数据提供器更换的时候，使用 showDataEffect 和 hideDataEffect 属性就可实现这样的效果。

在 mx.charts.effects 包内定义了三个效果，分别是 SeriesInterpolate、SeriesSlide 和 SeriesZoom。以下例子是使用 SeriesZoom 效果。

SeriesZoom 类通过 horizontalFocus 和 verticalFocus 属性调用焦点集合来实现图表数据的缩放。

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script><![CDATA[
        import mx.charts.series.items.ColumnSeriesItem;
        import mx.graphics.SolidColor;
        import mx.charts.ChartItem;
        import mx.graphics.IFill;
        import mx.collections.ArrayCollection;
        import mx.charts.HitData;
        import mx.charts.events.ChartItemEvent;
```

这个内层数据集合不仅可以使所有项目一起显示，也可以使每个项目独立显示。

Code View:

```
[Bindable]
public var overview:ArrayCollection = new ArrayCollection
([
    { date:"01/02/2006", total:3000, food:1300, drinks:1700,
        other:0, expenses
    :2700, profit:300},
    { date:"01/08/2006", total:3500, food:1800, drinks:1500,
        other:200, expenses
    :2900, profit:600},
    { date:"01/15/2006", total:2600, food:1000, drinks:1600,
        other:0, expenses
    :2700, profit:-100},
    { date:"01/22/2006", total:3200, food:1300, drinks:1900,
        other:0, expenses
    :2900, profit:200 },
    { date:"02/1/2006", total:2200, food:1200, drinks:1000,
        other:0, expenses:
    2100, profit:100 },
    { date:"02/8/2006", total:2600, food:1300, drinks:1600,
        other:100, expense
    s:2700, profit:400 },
    { date:"02/16/2006", total:4100, food:2300, drinks:1700,
        other:100, expenses
    :2700, profit:200 },
    { date:"02/22/2006", total:4300, food:2300, drinks:1700,
        other:300, expenses
    :3300, profit:1000 }]);
}

[Bindable]
public var drillDownDataSet:ArrayCollection;

[Bindable]
public var mainDataProvider:ArrayCollection = overview;

private function zoomIntoSeries(e:ChartItemEvent):void {
    if (mainDataProvider == overview) {
        drillDownDataSet =
            new ArrayCollection(createDataForDate(e));
        columnSeries.displayName = "Daily Breakdown";
        columnSeries.yField = "amount";
        columnSeries.xField = "type";
    }
}
```

```

        cal.categoryField = "type";

        mainPanel.title = "Profits for " +
            e.hitData.item.date;
        mainDataProvider = drillDownDataSet;

    } else {
        mainDataProvider = overview;

        columnSeries.displayName = "Profit by date";
        columnSeries.yField = "profit";
        columnSeries.xField = "date";

        cal.categoryField = "date";

        mainPanel.title = "Profit Overview";
    }
}

private function profitFunction(element:ChartItem,
index:Number):IFill {
    // black for profit
    var dateColor:SolidColor = new SolidColor(0x000000);
    var item:ColumnSeriesItem = ColumnSeriesItem(element);
    var profit:Number = Number(item.yValue);

    if (profit < 0) {
        // red for not profitable
        dateColor.color = 0xFF0000;
    }
    return dateColor;
}

```

若列状图上某个特定的数列被点击，可使用 ChartItemEvent 的 hitData 属性来获取这个被点击的数据。以下是明细项目的数据集合：

Code View:

```

private function createDataForDate(e:ChartItemEvent):Array {
    var result:Array = [];

    var food:Object = { type:"food",
amount:e.hitData.item.food };
    var drinks:Object = { type:"drinks",
amount:e.hitData.item.drinks };

```

```

        var other:Object = { type:"other",
amount:e.hitData.item.other };
        var expenses:Object = { type:"expenses",
amount:e.hitData.item.expense
s };

        result.push(food);
        result.push(drinks);
        result.push(other);
        result.push(expenses);
        return result;
    }

]]></mx:Script>

<mx:SeriesZoom id="slideZoomIn" duration="1000"
verticalFocus="bottom"/>
<mx:SeriesZoom id="slideZoomOut" duration="1000"
verticalFocus="bottom"/>

<mx:Panel id="mainPanel" title="Profitability">
    <mx:ColumnChart id="chart" showDataTips="true"
itemClick="zoomIntoSeries(event)"
dataProvider="{mainDataProvider}">

        <mx:series>

```

showDataEffect 和 hideDataEffect 属性指出当图表的数据提供器变化时哪种效果将被显示。

Code View:

```

<mx:ColumnSeries id="columnSeries"
displayName="Total profit"
fillFunction="profitFunction"
yField="profit" xField="date"
hideDataEffect="slideZoomOut"
showDataEffect="slideZoomIn"/>
</mx:series>
<mx:horizontalAxis>
    <mx:CategoryAxis id="ca1" categoryField="date"/>
</mx:horizontalAxis>
</mx:ColumnChart>
</mx:Panel>
</mx:Application>

```

## 16.7 节 为图表的元素更换皮肤

### 16.7.1. 问题

我需要改变图表项目的外观。

### 16.7.2. 解决办法

创建一个 skin 类，该类需要继承 ProgrammaticSkin 类和实现 IDataRenderer 类的接口。将此类设置为图表的 ChartSeries 类的渲染器。

### 16.7.3. 讨论

mx.charts.ChartItem 代表图表数列中一个数据点，每个数列的数据提供器都有一个 ChartItem。ChartItem 定义以下属性：

**currentState** : String

Defines the appearance of the ChartItem.

定义 ChartItem 的外观。

**element** : IChartElement

The series or element that owns the ChartItem.

使用 ChartItem 的数列和元素

**index** : int

The index of the data from the series' dataProvider that the ChartItem represents.

ChartItem 所代表的数据提供器中数据的索引

**item** : Object

The item from the series' dataProvider that the ChartItem represents.

ChartItem 代表的数列的数据提供器中的项目

**itemRenderer** : IFlexDisplayObject

The instance of the chart's item renderer that represents this ChartItem.

当前 ChartItem 的图表项目渲染器实例。

The ChartItem is owned by a ChartSeries, which uses the items to display the data points.

ChartItem 属于 ChartSeries，它使用项目条来显示数据点。

ChartSeries 属于一个图表，它拥有一个或多个 ChartSeries 条目，这些条目具有单独的数据属性。

每个数列有一个默认的数据渲染器，flex 使用这个渲染器来描绘数列的 ChartItem 对象，如表 16-2 所列。使用数列的 itemRenderer 样式属性可以定义一个新的渲染器。这个属性指向一个定义 ChartItem 对象外观的类。

Table 16-2. 图表中可供使用的 itemRenderers	
Chart Type 图表类型	Renderer Classes 渲染器的类
AreaChart	AreaRenderer
BarChart	BoxItemRenderer
BubbleChart	CircleItemRenderer
ColumnChart	CrossItemRenderer
PlotChart	DiamondItemRenderer
	ShadowBoxRenderer
	TriangleItemRenderer
CandlestickChart	CandlesitckItemRenderer
HLOCChart	HLOCItemRenderer
LineChart	LineRenderer
	ShadowLineRenderer
PieChart	WedgeItemRenderer

将一张图片设定为图表数列的 itemRenderer，只需简单地把已嵌入的图片设置为 itemRenderer 的属性即可。

Code View:

```
<mx:PlotSeries xField="goals" yField="games" displayName="Goals per game" itemRenderer="@Embed(source='..../assets/soccerball.png') radius="20" legendMarkerRenderer="@Embed(source='..../assets/soccerball.png')"/>
```

为图表的条目创建皮肤，需要创建一个 ProgrammaticSkin 类，并且覆盖 updateDisplayList() 方法：

Code View:

```
package oreilly.cookbook
{
    import flash.display.BitmapData;
```

```

import flash.display.DisplayObject;
import flash.display.IBitmapDrawable;

import mx.charts.series.items.ColumnSeriesItem;
import mx.core.IDataRenderer;
import mx.skins.ProgrammaticSkin;

public class CustomRenderer extends ProgrammaticSkin implements
IDataRenderer {

private var _chartItem:ColumnSeriesItem;

[Embed(source="../../assets/Shakey.png")]
private var img:Class;

public function get data():Object {
    return _chartItem;
}

public function set data(value:Object):void {
    _chartItem = value as ColumnSeriesItem;
    invalidateDisplayList();
}

override protected function
updateDisplayList(unscaledWidth:Number,
unscaledHeight:Number):void
{
    super.updateDisplayList(unscaledWidth, unscaledHeight);
    var img_inst = new img();
    var bmd:BitmapData = new BitmapData((img_inst as
DisplayObject).height,
(foo_inst as DisplayObject).width, true);
    bmd.draw(img_inst as IBitmapDrawable);
    graphics.clear();
    graphics.beginBitmapFill(bmd);
    graphics.drawRect(0, 0, unscaledWidth, unscaledHeight);
    graphics.endFill();
}
}
}
}

```

现在 ProgrammaticSkin 类就可以设置为列状图的 itemRenderer。

```
<mx:ColumnChart id="column" dataProvider="{expenses}">
    <mx:horizontalAxis>
        <mx:CategoryAxis
            dataProvider="{expenses}"
            categoryField="Month"
        />
    </mx:horizontalAxis>
    <mx:series>
        <mx:Array>
            <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
                displayName="Expenses"
                itemRenderer="oreilly.cookbook.CustomRenderer"
            />
        </mx:Array>
    </mx:series>
</mx:ColumnChart>
```

## 16.8.节 使用 ActionScript 动态添加和去除图表中的列

### 16.8.1. 问题

我想在程序运行期间添加或删除列状图中的数据列。

### 16.8.2. 解决办法

用 ActionScript 创建数列集合，就可以在任何时间动态地添加或删除数据数列。

### 16.8.3. 讨论

在一个图表中数列可以组合成一个集合对象。每种图表类型都有不同的集合类型，如表 16-3 所列。例如，列状图用 ColumnSeries 来组合一个列集合。

**Table 16-3. Types of Groupings for Series Items**

数列元素的集合类型

属性	描述
Clustered 串集合	Series items are grouped side by side in each category. This is the default for BarCharts and ColumnCharts.

**Table 16-3. Types of Groupings for Series Items**

数列元素的集合类型	
属性	描述
	数列元素在每个类别中挨个组合，这是条状图和列状图的默认属性。
Stacked 堆叠集合	Series items are stacked on top of each other. Each item represents the cumulative value of the items beneath it. 数列元素按堆叠的方式组合，每个元素代表其下的元素的和。
Overlaid 覆盖集合	Series items are overlaid on top of each other, with the last series on top. 数列元素由下往上覆盖，最后一个数列在最上面。
100 percent 百分比集合	Series items are stacked on top of each other, adding up to 100 percent. Each item represents the percentage value of the whole set. 数列元素按堆叠的方式组合，用百分比表示。每个元素表示整个集合的百分比。

图表集合有一个数列属性可以接收数列对象的数组。使用这个集合，可以用多种方法来捆绑数据。例如，图表可以有两个堆叠集合，每个堆叠集合可以有一个串集合。

使用 ActionScript 设定数列和集合后就可以在任何时间在图表中添加和移除数据。

以下的例子是使用了的数据集合是追踪一周的雨量毫米记录。一天中的每个时刻都被表示为一个数列，并且由 ActionScript 来初始化。每个数列都有一个复选框，当选择复选框时，列状图都会重新初始化，然后将所选的复选框代表的数列添加到列状图中。列集合会被添加到一个新的数组中，以及设置列状图的数列属性。

Code View:

```

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
layout="vertical" backgroundColor="#FFFFFF" creationComplete="onComplete()">
<mx:Script>
<! [CDATA[
import mx.charts.series.ColumnSet;
import mx.charts.series.ColumnSeries;

[Bindable] private var chartDP:Array =
[ {day: 'Monday', dawnRainfall:10, morningRainfall:12, midDayRainfall:6, afternoonRainfall:4, duskRainfall:5},
{day: 'Tuesday', dawnRainfall:7, morningRainfall:10, midDayRainfall:5, afternoonRainfall:5, duskRainfall:6},
{day: 'Wednesday', dawnRainfall:5, morningRainfall:9, midDayRainfall:3, afternoonRainfall:2, duskRainfall:3},
{day: 'Thursday', dawnRainfall:8, morningRainfall:9, midDayRainfall:6, duskRainfall:4} ];
]]>

```

```

afternoonRainfall:6,
duskRainfall:6},
{day: 'Friday', dawnRainfall:11,morningRainfall:13,midDayRainfall:4,
afternoonRainfall:5,
duskRainfall:7},
{day: 'Saturday', dawnRainfall:12,morningRainfall:13,midDayRainfall:
9,afternoonRainfall:
3,duskRainfall:4},
{day: 'Sunday', dawnRainfall:14,morningRainfall:12,midDayRainfall:5,
afternoonRainfall:1,
duskRainfall:3}
];

```

```

private var dawnSeries:ColumnSeries;
private var morningSeries:ColumnSeries;
private var midDaySeries:ColumnSeries;
private var afternoonSeries:ColumnSeries;
private var duskSeries:ColumnSeries;

private var columnSet:ColumnSet;

private function onComplete():void
{
    //initialize our clustered ColumnSet
    columnSet = new ColumnSet();
    columnSet.type = "clustered";

    //initialize all of our series
    dawnSeries = new ColumnSeries();
    dawnSeries.yField = "dawnRainfall";
    dawnSeries.xField = "day";
    dawnSeries.displayName = "Dawn Rainfall";

    morningSeries = new ColumnSeries();
    morningSeries.yField = "morningRainfall";
    morningSeries.xField = "day";
    morningSeries.displayName = "Morning Rainfall";

    midDaySeries = new ColumnSeries();
    midDaySeries.yField = "midDayRainfall";
    midDaySeries.xField = "day";
    midDaySeries.displayName = "Mid-day Rainfall";

    afternoonSeries = new ColumnSeries();

```

```

afternoonSeries.yField = "afternoonRainfall";
afternoonSeries.xField = "day";
afternoonSeries.displayName = "Afternoon Rainfall";

duskSeries = new ColumnSeries();
duskSeries.yField = "duskRainfall";
duskSeries.xField = "day";
duskSeries.displayName = "Dusk Rainfall";

updateSeries();

}

//以下是添加到列集合中的数列，添加后更新图表。

```

Code View:

```

private function updateSeries():void
{
    //reinit columnSet
    columnSet.series = new Array();
    //for each check box, add the corresponding series if it's
checked
    if(showDawnCheckBox.selected) {
        columnSet.series.push(dawnSeries);
    }
    if(showMorningCheckBox.selected) {
        columnSet.series.push(morningSeries);
    }
    if(showMidDayCheckBox.selected) {
        columnSet.series.push(midDaySeries);
    }
    if(showAfternoonCheckBox.selected) {
        columnSet.series.push(afternoonSeries);
    }
    if(showDuskCheckBox.selected) {
        columnSet.series.push(duskSeries);
    }
    // put columnSet in an array and set to
    // the chart's "series" attribute
    rainfallChart.series = [columnSet];
}

]]>
</mx:Script>

```

```

<mx:ColumnChart id="rainfallChart" dataProvider="{chartDP}" >
    <mx:horizontalAxis>
        <mx:CategoryAxis categoryField="day" />
    </mx:horizontalAxis>
    <mx:verticalAxis>
        <mx:LinearAxis minimum="0" maximum="14" />
    </mx:verticalAxis>
    <!-- notice there is no series attribute defined, we do
that in AS above -->
</mx:ColumnChart>
<mx:HBox>
    <mx:CheckBox id="showDawnCheckBox"
        label="Dawn Rainfall" selected="true"
        change="updateSeries()" />
    <mx:CheckBox id="showMorningCheckBox"
        label="Morning Rainfall" change="updateSeries()" />
    <mx:CheckBox id="showMidDayCheckBox"
        label="Mid-day Rainfall" change="updateSeries()" />
    <mx:CheckBox id="showAfternoonCheckBox"
        label="Afternoon Rainfall" change="updateSeries()" />
    <mx:CheckBox id="showDuskCheckBox"
        label="Dusk Rainfall" change="updateSeries()" />
</mx:HBox>
</mx:Application>

```

## 16.9.节 重叠多个图表

### 16.9.1. 问题

我想要使用不同的类型的图表来表示重叠数据集合。

### 16.9.2. 解决办法

使用列状图来装载复合图表，然后使用<mx:Series>标签来定义复合图表和他们的属性

### 16.9.3. 讨论

任何图表都可以在其数列数组中包含复合图表数列，每个数组可以代表不同的字段，这些字段可以由一个数据提供器控制，也可以由多个不同的数据提供器控制。在以下的例子中，是用列状图承载复合数列数组

Code View:

```

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  creationComplete="genData()">
  <mx:Script>
    <![CDATA[

      private var DJIA:Number = Math.random()*50 - 20;
      private var NASDAC:Number = DJIA - Math.random() * 20;
      private var SP500:Number = Math.random()*40;

      public function genData():void
      {
        // assigning the data that the chart is bound to
        // is best done via a local variable that is then
        // set to the chart data, rather than adding values
to the
        // dataprovider of the chart
        var newArr:Array = [];
        for(var i:int = 0; i<10; i++)
        {
          DJIA = DJIA + Math.random()*10 - 5;
          NASDAC = NASDAC - Math.random() * 5;
          SP500 = Math.random()*40;
          newArr.push({ "DJIA": DJIA, "NASDAC": NASDAC,
            "SP500": SP500 });
        }
        chartData = newArr;
      }
      [Bindable] public var chartData:Array = [];

    ]]>
  </mx:Script>
  <mx:SeriesInterpolate id="eff" elementOffset="1"
    minimumElementDuration="40" duration="2000"/>
  <mx:Button click="genData()" label="Generate data"/>
  <mx:ColumnChart y="100" width="100%" height="100%"
    dataProvider="{chartData}">
    <mx:series>
      <mx:ColumnSeries showDataEffect="{eff}"
        yField="DJIA"/>
      <mx:ColumnSeries showDataEffect="{eff}"
        yField="NASDAC"/>
      <mx:ColumnSeries showDataEffect="{eff}"
        yField="SP500"/>
    
```

```
</mx:series>
</mx:ColumnChart>
</mx:Application>
```

在列状图中，复合列状图对象将被一个叠着一个来渲染。

## 16.10.节 拖曳图表中的项目

### 16.10.1. 问题

我想从一个数据源里拖曳项目到图表中。

### 16.10.2. 解决办法

覆盖图表组件的 dragEnterHandler()和 dragDropHandler()方法则可创建一个可拖曳的图表。

### 16.10.3. 讨论

在 flex 框架中图表的拖曳功能和其他组件的拖曳功能是相同。父类组件为 mouseMove 事件定义一个处理器，为 dragDrop 事件定义一个处理器来接收被拖曳的数据。在下面的例子中，两个饼图的 dragEnabled 和 dropEnabled 属性都设置为 true，分别有单独的 ArrayCollection 作为各自的数据提供器。当数据从一个组件中被拖出，它就会从组件中脱离然后被添加到其他的数据提供器中。

Code View:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
      xmlns:cookbook="oreilly.cookbook.*">

    <mx:Script>
        <! [CDATA[
            import mx.events.DragEvent;
            import mx.charts.PieChart;
            import mx.core.IUIComponent;
            import mx.core.DragSource;
            import mx.containers.Panel;
            import mx.managers.DragManager;
            import mx.collections.ArrayCollection;

            [Bindable]
            private var masterArrColl:ArrayCollection =
                new ArrayCollection([
                    { "name": "C Ronaldo", "sog":128, "goals":20, "games":33 },
                    { "name": "A Adebayor", "sog":128, "goals":20, "games":35 },
                ]);
        ]]>
    </mx:Script>

```

```

    { "name": "F Torres", "sog": 98, "goals": 18, "games": 32 },
    { "name": "W Rooney", "sog": 89, "goals": 17, "games": 34 },
    { "name": "D Drogba", "sog": 114, "goals": 16, "games": 31 } ] );

    [Bindable]
    private var subColl:ArrayCollection =
        new ArrayCollection([
        { "name": "C Ronaldo", "sog": 128, "goals": 20, "games": 33 },
        { "name": "A Adebayor", "sog": 128, "goals": 20, "games": 35
    ] );

    // Initializes the drag-and-drop operation.
    private function
        mouseMoveHandler(event:MouseEvent):void {
            event.preventDefault();

            // Get the drag initiator component from the event
            // object.
            //从事件对象中取得拖曳事件初始化组件
            var dragInitiator:PieChart =
                PieChart(event.currentTarget);

            // Create a DragSource object.
            //创建一个DragSource对象
            var ds:DragSource = new DragSource();
            //make sure that the chart has a selected item
            //确定图表的某个元素是处于选择状态中
            if(dragInitiator.selectedChartItem == null) return;
            // Call the DragManager doDrag() method to start the
            //drag.
            //调用DragManager doDrag()方法开始拖曳事件
            DragManager.doDrag(dragInitiator, ds, event);
        }
    
```

mouseMoveHandler()方法用于传递 dragInitiator,该组件向 DragManager.doDrag 管理器发出拖动事件和 DataSource 对象（该例子没有用到），鼠标事件也被激活。

mouseMoveHandler()方法

Code View:

```

    // Called if the target accepts the dragged object and
    // the user
    // releases the mouse button while over the Canvas

```

*container.*

//目标接受dragged对象以及用户在Canvas容器中释放鼠标就调用以下方法:

```
private function dragDropHandler(event:DragEvent):void {  
  
    // Get the selected data from the chart  
    //取得图表中被选的元素  
    var index:Number = (event.dragInitiator as  
        PieChart).selectedChartItem.index;  
    (event.currentTarget as  
        PieChart).dataProvider.addItem((event.dragInitiator as  
            PieChart).dataProvider.getItemAt(index));  
    (event.dragInitiator as  
        PieChart).dataProvider.removeItemAt(index);  
}  
}
```

对象是首次添加到 dragEvent 的 currentTarget 的数据提供器中，就是添加到了饼图上。从 DragEvent.dragInitiator 中移出的数据(以及相应的对象)，也从饼图中移除了。

Code View:

```
]]>  
</mx:Script>  
<mx:PieChart dataProvider="{subColl}" selectionMode="single"  
    dragEnabled="true" dropEnabled="true"  
    mouseMove="mouseMoveHandler(event)"  
    dragDrop="dragDropHandler(event)">  
    <mx:series>  
        <mx:PieSeries field="goals" nameField="name"  
            labelField="name" labelPosition="callout"  
            selectable="true"/>  
    </mx:series>  
</mx:PieChart>  
<mx:PieChart dataProvider="{masterArrColl}" dragEnabled="true"  
    dropEnabled="true" selectionMode="single"  
    mouseMove="mouseMoveHandler(event)"  
    dragDrop="dragDropHandler(event)">  
    <mx:series>  
        <mx:PieSeries field="goals" nameField="name"  
            labelField="name" labelPosition="callout"  
            selectable="true"/>  
    </mx:series>  
</mx:PieChart>  
</mx:Application>
```

饼图的 Selectable 属性、dragEnabled 属性和 dropEnabled 属性必须设置为 true。当一个图表元素从图表中拖出，dragProxy 就会以一个位图的方式渲染，这个位图就是被拖曳的图表元

素的复印件。

## 16.11.节 创建一个可以编辑线状图

### 16.11.1. 节 问题

在其他属性的变化时，需要更新图表数据提供器中某属性的值。

### 16.11.2. 节 解决办法

创建一个图表，该图表具有复合的 ChartSeries 对象，并且设置每个可变的 ChartSeries 对象的 selectable 属性为 true。然后创建拖放事件处理器，当一个值改变的时候，该处理器就执行计算。

### 16.11.3. 节 讨论

在以下的例子中，系列图表代表开支和销售之间的盈利关系。代表开支和销售的线状图的 selectable 属性都为 true：

Code View:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            import mx.charts.chartClasses.AxisLabelSet;
            import mx.core.DragSource;
            import mx.charts.series.LineSeries;
            import mx.events.DragEvent;
            import mx.managers.DragManager;

            import mx.collections.ArrayCollection;

            [Bindable]
            private var expensesAC:ArrayCollection = new
            ArrayCollection( [
                { Month: "Jan", Profit: 2000, Expenses: 1500, Sales:
3550 },
                { Month: "Feb", Profit: 1000, Expenses: 200, Sales:
1200 },
                { Month: "Mar", Profit: 1500, Expenses: 500, Sales:
2000 },
                { Month: "Apr", Profit: 1800, Expenses: 1200, Sales:
3000 },
            ] );
        ]]>
    </mx:Script>
</mx:Application>
```

```
{ Month: "May", Profit: 2400, Expenses: 575, Sales: 2975}]);
```

```
// Initializes the drag-and-drop operation. 初始化拖曳操作
private function
mouseMoveHandler(event:MouseEvent):void {
    event.preventDefault();
    // Get the drag initiator component from the event object.
```

从event对象中取得拖曳初始化组件

```
var dragInitiator:LineSeries =
    LineSeries(event.currentTarget);
// if a selectedItem isn't set, ignore the mouse event
```

如果没有选择任何元素，则忽略鼠标事件

```
if(dragInitiator.selectedItem == null) return;
```

```
// Create a DragSource object.
```

创建一个DragSource对象

```
var ds:DragSource = new DragSource();
```

```
// Call the DragManager doDrag() method to start the drag.
```

调用DragManager的doDrag()方法来启动拖动事件

```
DragManager.doDrag(dragInitiator, ds, event);
```

```
}
```

mouseMoveHandler()为每个可选的线状图对象处理鼠标运动。拖动鼠标激活

DragManager.doDrag()方法，该方法包含dragInitiator（在本例中就是线状图对象），拖动鼠标时将其传递给doDrag()方法。

Code View:

```
private function setDragDropData(event:DragEvent):void {
    var index:Number = (event.dragInitiator as LineChart).selectedChartItem.index;
    var newYVal:Number = (event.dragInitiator as LineChart).mouseY;
    var selectedSeries:LineSeries = (event.dragInitiator as LineChart).selectedChartItem.element as LineSeries;
    var editedObj:Object = (event.dragInitiator as LineChart).dataProvider.getItemAt(index);
```

```

var als:AxisLabelSet =
    linechart.verticalAxis.getLabelEstimate();
var maxValue:Number = als.labels[als.labels.length -
1].value;

```

横坐标和纵坐标都实现了 IAxis 接口，它的 getLabelEstimate 方法返回一个 AxisLabelSet 对象。AxisLabelSet 对象定义一个类型数组的标签属性，该属性包含数轴上所有标签。在这种情况下，最后一个数值用于定义图表中最大的数值。因为用户在每一次改变数值的时候最后一个数值都可以改变，每次读取用户拖曳出来的新数值是非常重要的，因为这样才能确定图表的正确的最大值，以便计算用户要得到的数值。

Code View:

```

if(selectedSeries.yField == "Expenses")
{
    var yPos:Number = ((linechart.height - newYVal) /
        linechart.height);
    var newVal:Number = maxValue * yPos;
    editedObj.Expenses = newVal;
}
else
{
    var yPos:Number = ((linechart.height - newYVal) /
        linechart.height);
    var newVal:Number = maxValue * yPos;
    editedObj.Sales = newVal;
}
editedObj.Profit = editedObj.Sales - editedObj.Expenses;
(event.dragInitiator as LineChart).clearSelection();

```

为父图表调用 clearSelection()方法是很重要的，这样可以确定图表的选区没有被鼠标事件干预。线状图的数据提供器被更新，使得图表被重绘。

Code View:

```

// force the chart to redraw - note if we weren't using a simple
array collection
// the data object in the array could dispatch an event,
forcing the binding to update
(event.dragInitiator as LineChart).dataProvider =

```

```

        expensesAC;

    }

    ]]>
</mx:Script>

<mx:Panel title="LineChart and AreaChart Controls Example"
height="100%" width="100%" layout="horizontal">

<mx:LineChart id="linechart" height="100%" width="100%"
paddingLeft="5" paddingRight="5"
dragDrop="setDragDropData(event)"
showDataTips="true" dataProvider="{expensesAC}"
selectionMode="single" dragEnabled="true"
dropEnabled="true">

<mx:horizontalAxis>
    <mx:CategoryAxis categoryField="Month"/>
</mx:horizontalAxis>

```

每个线状图的 CircleItemRenderer 用于使图表在 selected 属性为 true 的情况下可拖曳。在本例中，因为线状图代表盈利，是由代表开支和销售的组件来决定的，所以盈利的 selected 属性就要设置为 false。

Code View:

```

<mx:series>
    <mx:LineSeries selectable="false" id="profitSeries"
        yField="Profit" form="curve" displayName="Profit"
        itemRenderer = "mx.charts.renderers.CircleItem
        Renderer"/>
    <mx:LineSeries mouseMove="mouseMoveHandler(event)"
        yField="Expenses" form="curve"
        displayName="Expenses" selectable="true"
        itemRenderer =
            "mx.charts.renderers.CircleItemRenderer"/>
    <mx:LineSeries mouseMove="mouseMoveHandler(event)"
        yField="Sales" form="curve" displayName="Sales"
        selectable="true" itemRenderer = "mx.charts.
        renderers.CircleItemRenderer"/>

```

```
        </mx:series>
    </mx:LineChart>
    <mx:Legend dataProvider="{linechart}" />
</mx:Panel>
</mx:Application>
```

# 第十七章. 共享对象(native|eas)

持久化数据以及数据的通信构成了用户体验应用程序的完整部分。所谓持久化数据，就是当用户来使用你的应用程序时，由你的本地或者网络应用程序来存储、访问数据，以及在多个会话中来提供容量来存贮状态信息。当你的应用程序和其他应用程序进行数据通讯时，数据会被传输来影响当先会话中的任意应用程序。尽管在线服务可以提供实时数据与持久化数据之间的通讯，但是你的基于 web 的应用程序可能需要更多本地的操作。

通过使用 ShareObject 数据可以被存贮在用户的本地硬盘上。ShareObject 的功能非常像浏览器的 cookie 并能允许你的 Flex 应用程序去存储简单数据类型（例如 String 和 Array）并能注册自定义的数据类型到文件系统中一个应用程序制定的位置，这个为遏制可以被多个会话访问。和浏览器的 cookies 不同的是，ShareObject 由一个较大的文件大小限制（100k），并且可以通过动作脚本消息格式（AMF）来序列化。AMF 是被用来序列化 AS 对象以及在服务端和客户端之间通过 Flash Remoting 技术来传输 remoting 对象的一种二进制文件格式。使用手上这些工具，你可以开始创建与数据交互的应用程序，而不用依赖于在线服务去存贮数据和反映实时会话数据。

## 17.1 节. 创建一个共享对象

### 17.1.1. 问题

我想要存贮数据到本地硬盘，并想要在多个会话中获取这些数据。

### 17.1.2. 解决办法

使用 ShareObject 类来创建会被存储到用户系统并能被相同会话或以后的会话访问的数据对象。

### 17.1.3. 讨论

实际上 ShareObject 的行为有些像浏览器的 cookies, ShareObjects 常常被成为 Flash Cookies, 它可以：

- 维护本地持久化
- 在 Flash Media Server 服务器上存储共享数据。
- 实时共享数据

你可以使用 ShareObject 来存贮数据到用户的硬盘上并且在应用程序的当前会话或者以后的会话来获取这些数据。无论如何，一个 ShareObject 只能被创建它的应用程序所访问。另外，应用程序对 ShareObject 的访问被限制，只能运行于相同的域。一个应用程序不能访问定位到不同域的 ShareObject。

ActionScript3 提供了两种类型的 ShareObject:

#### 本地共享对象 (Local SharedObjects)

类似于浏览器 cookie, 本地存贮对象适于用力维护和持久化多个会话中的数据。例如, 你可以在用户访问网站之后, 使用他们来存贮数据。你可以追踪人口统计数据、档案资料以及文章的潜在阅读用户来影响和改变用户下次访问应用程序的用户体验。

#### 远端共享对象 (Remote ShareObjects)

远端共享对象需要使用 FMS, 它更像是实时的数据传输设备。当你想要在一个多人参与的聊天室中实时查看聊天记录时, 你可以在这个聊天应用程序中使用远端共享对象。当一个远端共享对象在客户端的机器上被更新, 它会回馈更新到服务器上的这个共享对象, 并会影响和更新其他正在查看相同共享对象的机器上指定的共享对象。

创建或者打开一个已经存在的本地共享对象, 使用如下代码:

Code View:

```
private var myLocalSharedObject : SharedObject = SharedObject.getLocal( "myLso" );
```

静态方法 getLocal 返回一个客户端上的共享对象的引用。对于远端共享对象, 使用如下代码:

Code View:

```
private var myRemoteObject : SharedObject = SharedObject.getRemote( "myRso" );
```

静态方法 getRemote 返回一个 FMS 上能被多个客户端访问到的共享对象引用。在呼叫了 getLocal 或者 getRemote 之后, 例子会在客户端机器上使用一个\*.sol 连接来打开或者创建一个文件。至此, 上面的例子会以两位个文件名作为结束 myLso.sol 和 myRso.sol .SOL 文件的默认位置依赖于正在运行的操作系统。这些位置对应不同的操作系统, 如下所示:

Windows 95/98/ME/2000/XP

C:/Documents and Settings/{ 用户 域 }/Application Data/Macromedia/Flash Player/#SharedObjects/{web 域}/{应用程序路径}/{应用程序名}/object.sol

Windows Vista

C:/Users/username/{ 用户 域 }/AppData/Roaming/Macromedia/Flash Player/#SharedObjects/web\_domain/path\_to\_application/application\_name/object\_name.sol

Mac OS X

/Users/{用户名}/Library/Preferences/Macromedia/Flash Player/#SharedObjects/{web 域}/{应用程序路径}/{应用程序名称}/object\_name.sol

Linux/Unix

/home/{用户名}/.macromedia/Flash Player/#SharedObjects/{web 域}/{应用程序路径}/{应用程序程

序名称}/object\_name.sol

默认情况，共享对象的最大文件大小是 100kb。你也可以通过 Flash Player 的设置管理器来改变本地共享对象（LSO）的默认文件大小

## 17.2 节. 写入数据到共享对象

### 17.2.1. 问题

我想要写数据到一个共享对象。

### 17.2.2. 解决办法

使用共享对象的 data 属性添加数据到共享对象

### 17.2.3. 讨论

你可以使用 SharedObject 类的 data 属性来添加数据到一个共享对象的 SOL 文件。应为 SharedObject 的 data 属性是个动态原件，你可以直接存贮你想要的属性的实例。这个是特殊的属性允许你持久化标准类型和非标准类型的数据。如下的例子展示了如何来存储简单类型到一个共享对象：

```
// 获取一个存在的共享对象; 或者创建这个对象
public var soInstance : SharedObject = SharedObject.getLocal( "myLso" );
// 字串存贮
public var oString : String = "Adobe Flex 3 Rocks";
soInstance.data.oPhrase = oString;
//数组存贮
public var oArrayItems : Array = [ 10, 11, 12 ];
soInstance.data.oArrayNumbers = oArrayItems;
// 布尔值存贮
public var oBoolean : Boolean = true;
soInstance.data.oDecision = oBoolean;
```

为了共享对象的数据能被写到磁盘上，你不需存贮数据的实例到 data 属性。试图直接指定共享对象的实例会引起一个编译类型错误：

```
// 引发编译错误
soInstance.variable = "compile error";
```

另外，当指定实例数据到 data 属性时，必须确保没有直接指定值，而是使用了一个变量

```
// 也会引发编译错误  
soInstance.data = "compile error";
```

为了避免这些类型的错误，确保实现一个用于共享对象的变量，然后指定这个变量，如下所示

```
// 这样能避免错误。  
private var myData : Array = [ 12, 13, 14 ];  
soInstance.data.myData = myData;
```

## 17.3 节. 保存本地共享对象

### 17.3.1. 问题

You want to write data to the file system via a SharedObject.

你想要通过共享对象写数据到文件系统

### 17.3.2. 解决办法

使用 ShareObject 的 flush 方法。

### 17.3.3. 讨论

默认情况下，一个应用程序创建一个共享对象，在应用程序退出时会自动写 SOL 文件到文件系统，而不管共享对象指定的默认文件大小。在会话打开时，SharedObject.flush 方法被用来将数据写入到 SOL 文件。SOL 文件大小默认最大为 100kb。你也可以使用 flush 方法的 minDiskSpace 参数来改变已经分配的文件大小。最终用户可以使用 Adobe Flash 播放器浏览器插件的设置管理器来允许你的应用写文件到硬盘，以及分配 SOL 文件大小和访问。明确的指定 flash 播放器来写一个共享对象到硬盘上，使用增加的分配文件大小，如下所示：

```
lso.flush( 5000 );
```

当 flush 方法被调用，它会尝试写数据到客户端电脑。flush 调用的结果是如下三个可能之一：

- 如果用户禁止对应域的本地共享对象（LSO）存贮，或者如果 Flash 播放器存贮数据因为相同原因失败，数据就不会被存贮并且 flush 方法会抛出一个错误
- 如果保存本地共享对象所需要的磁盘空间小于对于对应域本地存贮设置的大小，数据会被写到磁盘上并且方法会返回 SharedObjectFlushStatus.FLUSHED，来表示方法成功运行。如果给 flush 方法赋予了可选参数最小磁盘空间，则已经被分配的磁盘空间必须大

于或者等于这个数值，才能成功运行 flush。

- 如果共享对象的数据需要大于已分配空间的，则用户会被提示来允许足够的空间大小访问，用来存贮数据。当这个（对用户左提示）发生时，会返回 SharedObjectFlushStatus.PENDING.当用户允许了访问之后，超出的空间会自动被分配并且数据会被保存

下面的例子说明了 flush 的使用方法以及如何来检测用户对共享对象的预定义设定。

Code View:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
    initialize="onInit()>
<mx:Script>
    <![CDATA[
        private var lso : SharedObject;
        private function onInit() : void
        {
            lso = SharedObject.getLocal( "mylso" );
        }
        private function saveValue( event : Event ) : void
        {
            var flushStatus : String = null;
            lso.data.myLsoData = textinput.text;
            try{
                flushStatus = lso.flush( 5000 );
            }
            catch( error : Error ) {
                status.text = "fault : cannot write shared object
                    to disk";
            }
            if ( flushStatus != null )
            {
                switch( flushStatus )
                {
                    case SharedObjectFlushStatus.PENDING:
                        status.text = "requesting permission to
                            save lso";
                        lso.addEventListener(NetStatusEvent.NET_STATUS,
                            onFlushStatus);
                    break;
                    case SharedObjectFlushStatus.FLUSHED:
                        status.text = "value flushed to disk";
                    break;
                }
            }
        }
    ]>
```

```

        }
    }

private function
onFlushStatus(event:NetStatusEvent):void
{
    switch (event.info.code)
    {
        case "SharedObject.Flush.Success":
            status.text = "permission granted -- value saved.";
            break;
        case "SharedObject.Flush.Failed":
            status.text = "permission denied -- value not saved.";
            break;
    }
    lso.removeEventListener(NetStatusEvent.NET_STATUS,
        onFlushStatus);
}

private function clearValue(event:MouseEvent):void
{
    status.text = "cleared lso data value";
    delete lso.data.myLsoData;
}

]]>
</mx:Script>
<mx:Form>
    <mx:FormItem label="Lso Data Input" width="100%">
        <mx:TextInput id="textinput" width="100%" />
    </mx:FormItem>
    <mx:FormItem direction="horizontal" width="100%">
        <mx:Button label="save value" width="100%" 
            click="{ saveValue( event ) }"/>
        <mx:Button label="clear value" width="100%" 
            click="{ clearValue( event ) }"/>
    </mx:FormItem>
    <mx:FormItem label="status message" width="100%">
        <mx:Label id="status" width="100%" />
    </mx:FormItem>
</mx:Form>
</mx:Application>
```

## 17.4 节. 从共享对象中读取数据

### 17.4.1. 问题

我想要从磁盘上读取一个共享对象或者检测一个共享对象是否已经被创建。

### 17.4.2. 解决办法

使用一个 SharedObject 实例的 data 属性来检查数据或者其他以变量属性是否可用。

### 17.4.3. 讨论

SharedObject.getLocal() 方法用来访问你的应用程序存储在用户硬盘上的本地共享对象。如果调用了 getLocal()方法的时候共享对象还不存在，默认会创建一个新的 SOL 文件。为了读取会话中新创建本地共享对象中存储的数据，你必须首先呼叫 flush()方法来写入数据到文件。很长时间内（用户允许 Flash player 浏览器插件的设定时间长）都可以通过 data 属性来访问共享对象中预先存储的数据。

如下的范例，在某个值在 data 对象不可用或者共享对象不存在的时候，写入数据到一个动态的 someData1 属性。在呼叫 flush()方法之后，数据会被核实。如果 someData1 属性的某个值已经可用，那这个操作就是简单读操作。

Code View:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute">
    <mx:Script>
        <![CDATA[
            private var lso : SharedObject;
            private function readSharedObject() : void
            {
                var sampleData1 : String = "data1";
                lso = SharedObject.getLocal( "mylso" );
                if( lso.data.someData1 == undefined ) {
                    lso.data.someData1 = sampleData1;
                    lso.flush();
                    status.text += "lso data after assignment: " +
                    lso.data.someData1 + "\n\n";
                } else {
                    status.text += "lso already written to: " +
                    lso.data.someData1 + "\n";
                }
            }
        ]]>
    </mx:Script>
</mx:Application>
```

```

        ]]>
    </mx:Script>

    <mx:VBox width="100%" height="100%">
        <mx:Button label="read a shared object"
            click="{ readSharedObject() }" />
        <mx:TextArea id="status" width="100%" height="100%" />
    </mx:VBox>
</mx:Application>

```

## 17.5 节. 删 除共享对象中的数据

### 17.5.1. 问题

我想要从磁盘上清除一个共享对象或者从一个共享对象中删除数据的一个指定位置。

### 17.5.2. 解决办法

使用 clear()方法来清除一个共享对象。使用 delete 关键字来移出一个共享对象的 data 属性中的制定数据。

### 17.5.3. 讨论

你需要简单的呼叫一个 SharedObject 的 clear()方法来从你的文件系统中清除这个共享对象。

```

// create an lso
private var lso : SharedObject = getLocal( "myLso" );
// create some sample data
private var sampleData : String = "data";
//assign some data to the lso
lso.data.sampleData = sampleData;
//flush to the file system
lso.flush();
//check the lso value
trace( lso.data.sampleData );
// delete the lso
lso.clear();
// should be "undefined"

```

```
trace( lso.data.sampleData );
```

使用 `delete` 关键字来删除本地共享对象 (`lso`) 中的一个指定数据元素，但是注意，这个不能把共享对象从文件系统中清除。

```
// delete sampleData  
delete lso.data.sampleData;  
// trace( lso.data.sampleData )
```

下面的例子移除共享对象数据并完全删除共享对象：

Code View:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">  
    <mx:Script>  
        <![CDATA[  
            private var lso : SharedObject;  
  
            private function removeSharedObject() : void  
            {  
                var sampleData1 : String = "data1";  
                var sampleData2 : String = "data2";  
  
                //create an lso object and assign some data to it  
                lso = SharedObject.getLocal( "mylso" );  
                lso.data.someData1 = sampleData1;  
                lso.data.someData2 = sampleData2;  
                lso.flush();  
                status.text = "lso data after assignment" + "\ndata1: " +  
                    lso.data.someData1 + "\ndata2: " + lso.data.someData2;  
  
                //delete a piece of data from the lso  
                delete lso.data.someData1;  
                status.text += "\n\nlso after deleting lso.data.someData1" + "\ndata1: " +  
                    lso.data.someData1 + "\ndata2: " + lso.data.someData2;  
                //remove the lso completely from the filesystem  
                lso.clear();  
                status.text += "\n\nlso data after calling clear()" + "\ndata1: " +  
                    lso.data.someData1 + "\ndata2: " + lso.data.someData2;  
            }  
        ]]>  
    </mx:Script>  
  
<mx:VBox width="100%" height="100%">  
    <mx:Button label="remove a shared object"  
        click="{ removeSharedObject() }" />
```

```
<mx:TextArea id="status" width="100%" height="100%" />
</mx:VBox>
</mx:Application>
```

## 17.6 节. 序列化类型对象

### 17.6.1. 问题

我想要能去保存自定数据类型的对象到一个共享对象中。

### 17.6.2. 解决办法

使用 `registerClassAlias()`方法来注册类型化的对象到 Flash 运行时，然后存贮这个对象实例到一个共享对象。

### 17.6.3. 讨论

所有的共享对象都包含了一个叫作 `objectEncoding` 的属性，用来标示在这个共享对象中使用的 AMF 版本。默认情况，`objectEncoding` 被设为 AMF3—标准的 as3 格式。你也可以设定一个共享对象的编码来使用 as1 和 as2 格式，通过设定 `objectEncoding` 为 AMF0。

当存贮类型化数据到一个共享对象是，确保已经在运行时注册了类型化对象，这样在特殊的情况发生时候你的应用程序才能确切地指导如何序列化和反序列化这个对象。呼叫 `registerClassAlias` 方法来注册类。`registerClassAlias` 方法需要 2 个参数。第一个必须是目标对象的完整限定类名以字符串的形式，通常就是类的别名。第二个参是你想要注册到第一个参所提供的类名的对象。

```
registerClassAlias( "package.ClassType", ClassType );
```

下面的例子是一个自定数据类型通过 `registerClassAlias()`方法来注册，确保属性值被序列化之后存到共享对象中，以及被同一个会话或其他会话读取出来并且反序列化。

Code View:

```
// the strongly typed object to use for this example
package oreilly.cookbook
{
    [Bindable]
    public class Automobile
    {
        private var _make : String;
```

```

public function get make() : String { return _make; }
public function set make( value : String ) : void {
    _make = value;
}

private var _model : String;
public function get model() : String { return _model; }
public function set model( value : String ) : void {
    _model = value;
}

private var _year : Number;
public function get year() : Number { return _year; }
public function set year( value : Number ) : void {
    _year = value;
}

public function Automobile()
{
    super();
}

public function toString() : String
{
    return "make: " + _make + "\nmodel: " + _model +
        "\nyear: " + _year;
}
}

```

Automobile 类拥有与制造一个汽车相关的基础属性。属性值可以被分别地存贮存储到共享对象实例中，但是你也需要注册这些自定的数据类型，确保当数据被再次从共享对象中读取到应用程序时，数据反序列化操作上可以序列化以及保留属性值。下面的例子注册了一个 Automobile 类的类名并且读写自定义对象到本地共享对象：

Code View:

```

// the main application
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
    initialize="onInit()">
    <mx:Script>
        <![CDATA[
            import mx.utils.ObjectUtil;
            import oreilly.cookbook.Automobile;
        
```

```

import flash.net.registerClassAlias;

private var lso : SharedObject;

private function onInit() : void
{
    // register the class alias to be able to later correctly
    // deserialize and serialize this class type
    registerClassAlias( "oreilly.cookbook.Automobile", Automobile );
    // initialize the shared object
    lso = SharedObject.getLocal( "automobile" );
}

private function save() : void
{
    var automobile : Automobile = new Automobile();
    automobile.make = make.text;
    automobile.model = model.text;
    automobile.year = parseFloat( year.text );

    lso.data.automobile = automobile;
    lso.flush();
    status.text = "your automobile has been saved";
}

private function retrieve() : void
{
    if( lso.data.automobile != undefined )
    {
        var objectInfo : String = ObjectUtil.toString(lso.data.automobile);
        status.text = "retrieving type information about the currently +
                     \" stored object\n\n";
        status.text += lso.data.automobile.toString() + "\n\n";
        status.text += objectInfo;
    }
    else
    {
        status.text = "nothing has been stored into the shared object";
    }
}
]]>
</mx:Script>

<mx:VBox width="100%" height="100%">
    <mx:Form>

```

```

<mx:FormItem label="make" width="100%>
    <mx:TextInput id="make" width="100%"/>
</mx:FormItem>
<mx:FormItem label="model" width="100%>
    <mx:TextInput id="model" width="100%"/>
</mx:FormItem>
<mx:FormItem label="year" width="100%>
    <mx:TextInput id="year" width="100%"/>
</mx:FormItem>
<mx:FormItem direction="horizontal" width="100%>
    <mx:Button label="save lso data" width="100%" 
        click="{ save() }"/>
    <mx:Button label="retrieve lso data" width="100%" 
        click="{ retrieve() }"/>
</mx:FormItem>
</mx:Form>
<mx:TextArea id="status" editable="false" width="100%" height="100%" />
</mx:VBox>
</mx:Application>

```

当你需要存贮指定的数据到用户的本地机器上时，上面的方法非常有效，特别是大的强类型值对象。通过注册类和初始化一个对象，你可以避免处理 XML 或者文本数据的运算开销，还能很好的维护对象的属性和类型。

## 17.7 节. 在多个应用程序中访问同一个共享对象

### 17.7.1. 问题

我想要在同一台机器上不同的 Flex 应用程序中共享使用同一个贡献对象。

### 17.7.2. 解决办法

在调用 SharedObject.getLocal() 方法的时候，指定一个外部路径。

### 17.7.3. 讨论

SharedObject.getLocal()方法的第二个参数指定了本地硬盘存贮这个共享对象的位置。存贮路径在 17.1 节中有讨论过，这些路径是依赖于应用程序所运行的操作系统平台。getLocal()方法特征如下：

Code View:

```
SharedObject.getLocal("objectName", "pathname" [ optional parameter ] ): SharedObject
```

如果你指定了 pathname, flash 播放器会存贮共享对象到相关的本地主机目录。通过这个, 你可以在访问共享对象的时候告诉应用程序一个已知位置来查找共享对象。例如:

```
lso = SharedObject.getLocal("myLso", "/");
```

你可以使用共享对象的已知位置来作为条件, 让多个应用程序访问指定共享对象中存贮的信息。

## 17.8 节. 记住用户输入文本框的内容

Contributed by Kristopher Schultz

### 17.8.1. 问题

我想要再用户离开应用程序的时候, 记住用户输入的 TextInput 的字段。

### 17.8.2. 解决办法

创建一个 TextInput 组建的子类, 当用户输入的时候来使用本地共享对象存贮用户输入的文本值。

### 17.8.3. 讨论

为了方便, 现代浏览器都提供了记住用户上次在公共表单区域输入值和登陆提示, 这些值在用户再次访问的时候不用再次输入。默认情况, Flex 应用程序没有继承这个非常有用的行为, 但是你可以很简单的通过使用本章的例子中的自定义组件来实现。

这个组件继承于 TextInput 类, 增加了一个静态方法到 TextInput 的 API, 被设计了用来记住最后一次的值。 persistenceId 属性是存贮 ID 用来指定这个组件的实例。静态方法 clearStoredValues()让你可以全局晴空所有之前存贮的值。代码如下:

Code View:

```
package custom
{
    import flash.events.Event;
    import flash.net.SharedObject;
```

```

import mx.controls.TextInput;

public class PersistentTextInput extends TextInput
{
    /**
     * The ID this component will use to save and later look up its
     * associated value.
     */
    public var persistenceId:String = null;

    /**
     * The SharedObject name to use for storing values.
     */
    private static const LOCAL_STORAGE_NAME:String =
        "persistentTextInputStorage";

    /**
     * Clears previously stored values for all PersistentTextInput instances.
     */
    public static function clearStoredValues() :void
    {
        var so:SharedObject = SharedObject.getLocal(LOCAL_STORAGE_NAME);
        so.clear();
    }

    /**
     * Handles initialization of this component.
     */
    override public function initialize() :void
    {
        super.initialize();
        addEventListener(Event.CHANGE, handleChange);
        restoreSavedValue();
    }

    /**
     * Event handler function for CHANGE events from this instance.
     */
    protected function handleChange(event:Event) :void
    {
        saveCurrentValue();
    }

    /**

```

```

* Restores the previously saved value associated with the
* persistenceID of this instance.
*/
protected function restoreSavedValue() :void
{
    if (persistenceId != null)
    {
        var so:SharedObject =
            SharedObject.getLocal(LOCAL_STORAGE_NAME);
        var value:String = so.data[persistenceId];
        if (value != null)
        {
            text = value;
        }
    }
}

/**
 * Saves the text value of this instance. Associates the value with
 * the persistenceID of this instance.
*/
protected function saveCurrentValue() :void
{
    if (persistenceId != null)
    {
        var so:SharedObject =
            SharedObject.getLocal(LOCAL_STORAGE_NAME);
        so.data[persistenceId] = text;
        so.flush();
    }
}
}

```

如下应用程序示范了如何使用这个新组件。测试这些功能，载入应用程序，输入一些文字，然后关闭应用程序。当你再次打开应用程序的时候，你会看到之前输入的一些字段会有之前你输入的值。

Code View:

```

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:custom="custom.*" layout="vertical">
```

```

<mx:Script>
<![CDATA[
```

```
import custom.PersistentTextInput;

public function clearValues():void
{
    PersistentTextInput.clearStoredValues();
    message.text = "A page refresh will reveal that the
                    values have not persisted.";
}
]]>
</mx:Script>

<custom:PersistentTextInput id="firstNameInput"
    persistenceId="firstName" />
<custom:PersistentTextInput id="lastNameInput"
    persistenceId="lastName" />
<mx:Button label="Clear Persistent Values"
    click="clearValues()" />
<mx:Label id="message" />

</mx:Application>
```

# 第十八章. 数据服务和服务端通信(Native|eas)

使用 Flex 最重要的部分之一 就是和服务器以及数据库的通讯。本章的内容主要关注与配置一个 Flex 应用程序来与服务器通讯以及 处理从服务器发到应用程序的数据，这些数据从三种主要的服务器应用之间的通讯方式传送。

Flex 提供了三个类来与服务器通讯：HTTPService，RemoteObject 以及 WebService。HTTPService 类提供了使用超文本传输协议（HTTP）与服务器通讯的方式。一个 Flex 应用程序可以使用 GET 或者 POST 请求来发送数据到一个服务器并且处理这个请求返回的 XML 或者字符串。使用 HTTPService 类，你可以与 PHP 页面,ColdFusion 页面，JavaServer 页面(jsp)，Java servlet，Ruby onRails，以及 ASP 动态网页通讯。你可以使用 RemoteObject 类通过 AMF 格式对象来和服务器通讯。RemoteObject 也可以与 Java 或者 ColdFusion remoting 网关或者使用了开源项目的.NET 以及 PHP 程序来通讯，开源项目包括 AMFPHP，SabreAMF 以及 WebORB。WebService 类可以和使用 web 服务描述语言（WSDL）所预先定义接口的 Web 服务通过 XML 以及基于 SOAP 的 XML 来通讯。

开始创建一个服务组件，你需要配置服务的属性，设定相对于服务器的所需要用来发送请求和接收数据的 URL 地址，以及相关的预期的数据类型的信息。对于 HTTPService 对象，你需要设定传输参数到服务器的方法，GET 或者 POST，以及 resultFormat – 返回值格式。对于 WebService 组件，你必须要设定服务的 WSDL 文档的 URL，并且在<mx:Operation>标签里面描述 WebService 中的每个操作，以及设定每个操作的得到结果（result）和错误（fault）处理函数。对于 RemoteObject 类，服务的 URL 需要在 services-config.xml 文件里面描述，这个 xml 文件会被编译到 SWF 文件中。服务的每个方法都需要被定义，他们得到的结果和错误的处理函数也需要被定义。

在做了一个呼叫到一个 HTTPService 之后，数据会从服务返回，并被放置到服务组件所包含的 lastResult 对象。服务组件的 resultFormat 属性默认是一个 ActionScript 对象。服务返回的所有数据作为对象的一个属性表现。从 WebService 或者 HTTPService 返回的任意 XML 数据都会被 Flex 转化成各自的基础类型，数字，字符串，布尔值以及日期。如果需要一个强类型对象，自定一个数据类型，然后从 lastResult 中存贮的对象创建一个实例。WebService 和 RemoteObject 类使用一个 result 事件处理函数来处理返回的数据，一个 fault 事件处理函数来处理返回的错误。所有的返回数据处理都会在服务指定的 result 处理函数内部完成。

## 18.1 节. 配置 HTTPService

### 18.1.1. 问题

我想要创建和配置一个 HTTPService 组件来允许你的应用程序来与基于 HTTP 的服务通讯。

### 18.1.2. 解决办法

为你的应用程序添加一个 `HTTPService` 组件，设定它的 `url` 属性为应用程序用来接受数据的 URL。如果服务的反馈是 XML，就需要定制的处理，指定一个方法到组件的 `xmlDecode` 属性来处理 XML 对象。

### 18.1.3. 讨论

`HTTPService` 对象提供了所有 HTTP 上跑的通讯。它包含了通过 GET 或者 POST 方法发送的信息，以及从 URL 请求上获取的信息，甚至静态的文件。`HTTPService` 对象可以设定 `result` 和 `fault` 方法处理函数分别来接受 `mx.event.ResultEvent` 对象以及 `mx.event.FaultEvent` 对象：

```
<mx:HTTPService url="http://192.168.1.101/service.php"
    id="service"
    result="serviceResult(event)" fault="serviceFault(event)">
```

这样可以让你处理 HTTP 请求的结果内容。`HTTPService` 对象的 `result` 属性可以使用 `HTTPService` 对象的 `result` 属性。

```
<mx:Image source="{service.lastResult as String}" />
```

注意这个 `HTTPService` 的 `lastResult` 是一个对象必须被作为一个字符广播。

`HTTPService` 对象也可以被用来通过 GET 或者 POST 变量来发送信息到一个脚本使用配置好的 `HTTPService` 对象的 `request` 属性

```
<mx:HTTPService>
    <mx:request xmlns="">
        <id>{requestedId}</id>
    </mx:request>
</mx:HTTPService>
```

这次发送 中 `requestedId` 属性会被包装在 `id` 标签，然后发送到 `HTTPService` 对象设定的 URL 上。

在下面的范例，一个 `HTTPService` 对象载入了一个来自与 PHP 脚本的 XML。

Code View:

```
<mx: Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="400" height="300">
    <mx:HTTPService url="http://localhost/service.php"
        id="service"
        result="serviceResult(event)" fault="serviceFault(event)"
        method="GET" contentType="application/xml"
        useProxy="false">
```

```

<mx:request xmlns="">
    <id>{requestedId}</id>
</mx:request>
</mx:HTTPService>
<mx:Script>
    <! [CDATA[
        import mx.rpc.events.FaultEvent;
        import mx.rpc.events.ResultEvent;

        [Bindable]
        private var requestedId:Number;

        //trace the result of the service out
        private function serviceResult(event:Event):void {
            trace(service.lastResult.name);
        }

        // in the event that the service faults or times out
        private function serviceFault(event:Event):void {
            trace('broken service');
        }

        private function callService():void {
            requestedId = input.text as Number;
            service.send();
        }
    ]]>
</mx:Script>
<mx:TextInput id="input"/>
<mx:Button label="get user name" click="callService()"/>
<mx:Text text="{service.lastResult.name}"/>
<mx:Text text="{service.lastResult.age}"/>
</mx:Application>

```

这个是 PHP 脚本，它会读取 Flex 应用发送来的 GET 变量然后返回到格式化的 XML 数据

```

<?php
$id_number = $_GET["id"];
echo('<id>'.$id_number.'</id><name>Todd Anderson</name><age>30</age>');
?>

```

## 18.2 节. 在 Flex 程序之间使用 RESTful 通信

### 18.2.1. 问题

我想要整合一个 Flex 应用程序和一台使用 RESTful 或者表达性状态转移风格通讯的服务器，例如 Rails 或者其他服务器。

### 18.2.2. 解决办法

创建一个 `HTTPService` 对象来使用合适的路径通过 POST 和 GET 方法和你的服务器通讯，呼叫远端服务器上的方法。

### 18.2.3. 讨论

所谓 RESTful 的服务一般被用来描述一个服务使用所有 4 个可能的 HTTP 头：PUT, POST, DELETE 以及 GET。这四个头通常对应四种基础的数据访问操作：创建，读取，更新以及删除，他们一般和常说的 CRUD 一起使用。在实践中，一个单独的重载的服务端方法以来 `http` 头来执行四种基础数据访问操作。在 REST 类应用中，方法时常会被映射到资源，这样四种数据访问方法，CRUD 允许资源的创建，删除，更新以及获取。这个资源可以是简单资源，数据库的一个表，或者一个复杂的模型对象。

Flash 播放器限制了只能使用 GET 和 POST 方法，意思就是任意介于 FLEX 应用与服务之间的通讯都需要指明 DELETE 或者 PUT 方法使用不同与标准 REST 的方法，例如附加他们到一个 GET 或 POST 信息上。

发送一个 PUT 命令到一个 Rails 应用，你可以这样做：

```
var request:URLRequest = new URLRequest();
var loader:URLLoader = new URLLoader();
loader.addEventListener(Event.COMPLETE, resultHandler);
loader.addEventListener(IOErrorEvent.IO_ERROR, errorHandler);
loader.addEventListener(HTTPStatusEvent.HTTP_STATUS, httpStatusHandler);
request.url = "http://rails/view/resource";

// Set the request type as POST and send the DELETE command as
// a variable in the data of the request
request.method = URLRequestMethod.POST;
request.data._method = "DELETE";

loader.load(request);
```

Ror(Ruby on Rails, 一种 web 敏捷开发框架)在正确的 HTTP 方法不能被使用的时候, 允许 \_methdo 变量来申明描述的方法。对于其他类型的 REST 类服务, 类似的操作都会被使用到。

对于 HTTPService 对象, 你可以使用 BlazeDS 或者 Adobe LiveCycle 来配合它。HTTPService 定义了一个 useProxy 属性, 当它设为 true 会指定 flash 播放器只与 services-config.xml 文件中的服务器定义来通讯。一个请求会建立并且发送一个被代理的 PUT/DELETE/OPTIONS(以及其他请求)被发送到 Adobe LiveCycle 或者 BlazeDS 服务器, 然后服务器会建立并且发送实际的 HTTP 请求并且返回反馈内容到 flash 播放器。代理同时也控制来自于 HTTP 500 代码服务器错误的失败反馈, 它可以返回给 flash 播放器让 HTTPService 可以处理。

当你配置好 HTTPService 对象使用 BlazeDS 或者 lifecycle 代理之后, 你可以在 HTTPService 上通过设定 method 属性来使用服务器的全部范围的 Http 头。

```
<mx:HTTPService id="proxyService"
    destination="http://localhost/app/url"/>
<mx:Script>
    <! [CDATA[
        private function sendPut():void {
            proxyService.method = "DELETE";
            proxyService.send("id=2");
        }
    ]]>
</mx:Script>
```

最后, 有个叫 Garbriel Hanford 的人开发了一个叫做 as3httpclient 的库使用了二进制的 Flash 套接字来读取 HTTP 流并且解码 HTTP 反馈。这个库允许你来发送和阅读 GET, POST, PUT 以及 DELETE 这四种 HTTP 反馈但是, 这个需要一个 crossdomain.xml 文件来允许 Flash 播放器连接到服务器 80 端口。默认情况, 与服务器通过套接字来通讯的时候, 你需要和服务端使用更为 REST 的反馈以及严格的 HTTP, 这个类库提供了足够强大可以用来替代标准的 HTTPService。

更多的信息和代码下载可以在这里看到:

<http://code.google.com/p/as3httpclientlib/>.

## 18.3. 配置和连接 RemoteObject

### 18.3.1. 问题

我想要为一个 Flex 应用配置一个 RemoteObject 用来连接到 ColdFusion, AMFPHP, 或者 Java 对象来提供 Flex 应用与服务的通讯。

### 18.3.2. 解决办法

在你的应用中创建一个 `RemoteObject` 实例并且为你的服务设定 id，让服务不仅仅可以通过 URL 访问。

### 18.3.3. 讨论

`RemoteObject` 允许你定义介于你的应用和服务器上实际的类对象之间的通讯。这是和 `WebService` 组件或者 `HTTPService` 组件都截然不同的。`RemoteObject` 组件可以被用来呼叫一个已经被定义用来通讯的 ColdFusion CFC 组件或者 Java 类。`RemoteObject` 也可以被用来和开源项目例例如 AMFPHP, SabreAMF 以及 WebORB 定义的对象以及资源来进行通讯。`RemoteObject` 可以定义如下属性。

#### `channelSet : ChannelSet`

提供访问 serveric 所使用的 ChanelSet。

#### `concurrency : String`

指明如何来控制同一服务多个呼叫的值

#### `constructor : Object`

类对象的引用或者一个给出的类实例的构造函数。

#### `destination : String`

服务的目的地

#### `endpoint : String`

让你快速指定 `RemoteObject` 目的地的一个端点

#### `makeObjectsBindable : Boolean`

如果为 true，则强制指定返回的匿名对象为可绑定对象。

#### `operations : Object`

指定服务定义的方法；使用在 `RemoteObject` 定义在 MXML 中的方法。

#### `requestTimeout : int`

提供对请求发送信息的超时限制访问，该属性单位为秒

#### `showBusyCursor : Boolean`

如果设为 true，则会在服务运行时现实一个繁忙状态的鼠标光标。

#### `source : String`

你可以在客户端指定一个源地址值；该属性并不支持在 swf 文件和 java 对象之间使用 Java 适配器来坐序列化通讯的操作。

因为 RemoteObject 方法可以返回一个不需要处理或者反序列化 xml 的对象。RemoteObject 呼叫的结果可以广播到一个 ArrayCollection 对象或者 ResultEvent 中的一个强类型值类型。在如下代码片段中，一个 RemoteObject 对象被配置了用来使用一个 <http://localhost:8400> 上的一个可用的 Java 服务。

Code View:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="400" height="300">
    <mx:RemoteObject id="local_service" concurrency="single"
        destination="http://localhost:8400/app"
        showBusyCursor="true"
        source="LocalService.Namespace.ServiceName">
        <mx:method name="getNames" fault="getNamesFault(event)"
            result="getNamesResult(event)"/>
        <mx:method name="getAges" fault="getAgesFault(event)"
            result="getAgesResult(event)"/>
    </mx:RemoteObject>
    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;
            import mx.rpc.events.ResultEvent;
            import mx.controls.Alert;
            import mx.rpc.events.FaultEvent;

            private function getNamesFault(event:FaultEvent):void {
                mx.controls.Alert.show(event.message as String,
                    "Service Error");
            }

            private function getNamesResult(event:ResultEvent):void
            {
                var namesColl:ArrayCollection = event.result as
                    ArrayCollection;
            }

            private function getAgesFault(event:FaultEvent):void {
                mx.controls.Alert.show(event.message as String,
                    "Service Error");
            }

            private function getAgesResult(event:ResultEvent):void
            {
                var agesColl:ArrayCollection = event.result as
                    ArrayCollection;
            }
        ]]>
    </mx:Script>

```

```

        }

    ]]>
</mx:Script>
</mx:Application>
```

这个 result 事件的每个方法可以被绑定到一个不同的事件处理方法上。在 actionscript 中可以通过为 RemoteObject 添加一个事件监听方法来实现。

Code View:

```

import mx.collections.ArrayCollection;
import mx.rpc.events.ResultEvent;
import mx.controls.Alert;
import mx.rpc.events.FaultEvent;
import mx.rpc.AbstractService;
import mx.rpc.AsyncToken;
import mx.rpc.Responder;

private function init():void {
    var responder:Responder = new Responder( getNamesResult,
        getNamesFault );
    var call:AsyncToken = ( local_service as
        AbstractService ).getNames();
    call.addResponder(responder);

}

private function getNamesFault(event:FaultEvent):void {
    Alert.show(event.message as String, "Service Error");
}

private function getNamesResult(event:ResultEvent):void {
    var namesColl:ArrayCollection = event.result as
        ArrayCollection;
}
```

在上面的例子中，`mx.rpc.Responder` 类被用来存储用来处理服务端返回的 result 和 fault 方法的处理函数。Responder 被添加到一个 AsyncToken 类中，当服务返回成功或者失败时，将会调用到相关的方法。

## 18.4节. 使用 AMFPHP 1.9中的 Flex Remoting

Contributed by Sankar Paneerselvam

此节来自: <http://hi.baidu.com/gdutpxz/blog/item/f2108e4556762323cefca3dd.html>

### 18.4.1. 问题

我想使用 Flex remoting 与安装了 AMFPHP 的服务器进行通信。

### 18.4.2. 解决办法

安装 AMFPHP 并进行连接数据源的配置，使用 RemoteObject 访问 AMFPHP 服务，调用上面的方法。

### 18.4.3. 讨论

为了演示如何使用 AMFPHP 和 Oracle Database Express Edition (XE)，这个例子使用 remoting 组件显示来自 Oracle 数据库的 employee 表数据。以下是 EMPLOYEES 表结构：

EMPLOYEE_ID	PLS_INTEGER
FIRST_NAME	VARCHAR2
LAST_NAME	VARCHAR2
EMAIL	VARCHAR2
PHONE_NUMBER	VARCHAR2
HIRE_DATE	DATE
JOB_ID	PLS_INTEGER
SALARY	NUMBER
COMMISSION_PCT	NUMBER
MANAGER_ID	PLS_INTEGER
DEPARTMENT_ID	PLS_INTEGER

下面是 PHP 中对应的 Employee 类：

```
<?php  
  
class Employee  
  
{  
  
    var $EMPLOYEE_ID;
```

```

var $FIRST_NAME;

var $LAST_NAME;

var $EMAIL;

var $PHONE_NUMBER;

var $HIRE_DATE;

var $JOB_ID;

var $SALARY;

var $COMMISSION_PCT;

var $MANAGER_ID;

var $DEPARTMENT_ID;

var $_explicitType = "project.Employee";

}

?>

```

这里的\$\_explicitType 变量是假设你有一个与 PHP 类 Employee 对应的 ActionScript 类。

本例使用开源的 AMFPHP 框架进行序列化，调用 AMFPHP 提供的服务可返回 AMF 格式的数据。本例中我们使用了个名为 EmployeeService 的服务来返回 Employee 的信息。service 文件要放在 AMFPHP 安装目录的 services 目录下

Code View:

```

<?php

require_once('Employee.php');

class EmployeeService {

    var $myconnection=null;

    var $statement=null;

    function getEmployees(){

```

```

$myconnection = oci_connect('hr','hr', "//localhost/xe");

# Check Oracle connection"

if (!myconnection) {

    # Dont use die (Fatal Error), return useful info to the client

    trigger_error("AMFPHP Remoting 'EmployeeService'

class could not connect: " . oci_error());

}

$query="SELECT * FROM EMPLOYEES";

# Return a list of all the employees

$stmt=oci_parse($myconnection,$query);

if (!$stmt) {

    oci_close($myconnection);

    trigger_error("AMFPHP Remoting 'EmployeeService'

class database SELECT query error: " . oci_error());

}

oci_execute($stmt);

while ($row = oci_fetch_array($stmt,OCI_RETURN_NULLS)) {

    $data_array[] = $row;

}

return($data_array);

}

?>

```

你可以通过 AMFPHP 提供的 Service Browser 检查你的服务（译注：以这个网址

http://localhost/amfphp 安装目录/browser/)

在 Flex 里使用 Remoting 需要 service-config.xml 文件（译注：这个同样是可选的，完全可以动态配置。）以下是 service-config.xml 文件的内容，除了 url 地址和端口外，其它地方不要改动。

Code View:

```
<services-config>

    <services>

        <service id="amfphp-flashremoting-service"
            class="flex.messaging.services.RemotingService"
            messageTypes="flex.messaging.messages.RemotingMessage">

            <destination id="amfphp">

                <channels>
                    <channel ref="my-amfphp"/>
                </channels>

                <properties>
                    <source>*</source>
                </properties>

            </destination>

        </service>

    </services>

    <channels>

        <channel-definition id="my-amfphp" class="mx.messaging.channels.AMFChannel">
            <endpoint uri="http://localhost:9999/amfphp2/amfphp/gateway.php"
                class="flex.messaging.endpoints.AMFEEndpoint"/>
        </channel-definition>
    </channels>

```

```
</channel-definition>
```

```
</channels>
```

```
</services-config>
```

加载 service-config.xml 文件需要改变 flex 的编译参数:

1,菜单 Project->Properties

2,选择 Flex Complier 块在" locale en\_US "后面添加 -services service-config.xml 。

3,确定。

现在开始创建 actionscript 类保存 remote object。

```
package project
{
    [RemoteClass(alias="project.Employee")]
    public class Employee
    {
        public var EMPLOYEE_ID:Number;
        public var FIRST_NAME:String;
        public var LAST_NAME:String;
        public var EMAIL:String;
        public var PHONE_NUMBER:Number;
        public var HIRE_DATE:Date;
        public var JOB_ID:Number;
        public var SALARY:Number;
        public var COMMISSION_PCT:Number;
        public var MANAGER_ID:Number;
        public var DEPARTMENT_ID:Number;
    }
}
```

现在创建调用 remoting 服务并显示其它返回内容的 MXML 文件:

Code View:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute">
```

Here the RemoteObject that will call the Employee service has its destination, source, and event handler methods set:

```
<mx:RemoteObject id="myservice" source="Project.EmployeeService"
    destination="amfphp" fault="faultHandler(event)"
```

```

showBusyCursor="true">
<mx:method name="getEmployees" result="resultHandler(event)"
    fault="faultHandler(event)">
</mx:method>
</mx:RemoteObject>
<mx:Script>
<! [CDATA[
    import Project.Employee;
    import mx.utils.ArrayUtil;
    import mx.collections.ArrayCollection;
    import mx.rpc.events.FaultEvent;
    import mx.rpc.events.ResultEvent;
    import mx.controls.Alert;

```

当数据返回里dp:ArrayCollection被设为服务端返回的ResultEvent的result属性的内容：

Code View:

```

[Bindable]
private var dp:ArrayCollection;

private function faultHandler(event:FaultEvent):void {
    Alert.show(event.fault.faultString,
    event.fault.faultCode.toString());
}

private function resultHandler(event:ResultEvent):void {
    dp=new ArrayCollection(ArrayUtil.toArray(event.result));
}

]]>
</mx:Script>
<mx:Canvas x="0" y="0" width="100%" height="100%">
<mx:Button x="10" y="10" label="Get data"
click="myservice.getOperation('getEmployees').send()"/>

```

每一列都与服务端返回相关联

Code View:

```

<mx:DataGrid x="10" y="40" width="100%" height="100%"
    dataProvider="{dp}">
<mx:columns>
    <mx:DataGridColumn headerText="EMPLOYEE_ID"
        dataField="EMPLOYEE_ID"/>

```

```

<mx:DataGridColumn headerText="FIRST_NAME"
    dataField="FIRST_NAME"/>
<mx:DataGridColumn headerText="LAST_NAME"
    dataField="LAST_NAME"/>
<mx:DataGridColumn headerText="EMAIL"
    dataField="EMAIL"/>
<mx:DataGridColumn headerText="PHONE_NUMBER"
    dataField="PHONE_NUMBER"/>
<mx:DataGridColumn headerText="HIRE_DATE"
    dataField="HIRE_DATE"/>
<mx:DataGridColumn headerText="JOB_ID"
    dataField="JOB_ID"/>
<mx:DataGridColumn headerText="SALARY"
    dataField="SALARY"/>
<mx:DataGridColumn headerText="COMMISSION_PCT"
    dataField="COMMISSION_PCT"/>
<mx:DataGridColumn headerText="MANAGER_ID"
    dataField="MANAGER_ID"/>
<mx:DataGridColumn headerText="DEPARTMENT_ID"
    dataField="DEPARTMENT_ID"/>
</mx:columns>
</mx:DataGrid>

</mx:Canvas>

</mx:Application>

```

当用户点击 Get Data 按钮时，远程请求就会被触发，然后获取结果并显示到 datagrid 里。

## 18.5节. 使用 **IExternalizable** 接口自定义序列化

Contributed by Peter Farland

### 18.5.1. 问题

当通过 RemoteObject 或 DataService 发送强类型数据时我想进行自定义决定哪些属性将被发送过去。

### 18.5.2. 解决办法

使用 ActionScript 3 API `flash.utils.IExternalizable`, 它兼容 `java.io.IExternalizable` API.

### 18.5.3. 讨论

通常使用可序列化类是在序列化中包含只读的属性。虽然在服务端可有多种方式完成此目的，但是在客户端就没有什么方法了。因此最好的办法就是在客户端和服务端都有效，你可以使你的类进行自定义双向序列化。

这种方法相对简单，在客户端 ActionScript 类只要实现 `flash.utils.IExternalizable`。这个 API 需要两个方法，`readExternal` 和 `writeExternal`，分别取得 `flash.utils.IDataInput` 和 `flash.utils.IDataOutput` 数据流，在服务端是由实现 `java.io.Externalizable` 接口的 Java 类实现这两个方法，它也有两个 `readExternal` 和 `writeExternal`，分别取得 `java.io.ObjectInput` 和 `java.io.ObjectOutput` 数据流。

虽然 `IDataInput` 和 `IDataOutput` 类让你设计自己的协议，但对于基本的数据类型如 `byte`, `int`, 和 UTF-8-编码 `Strings`，可以充分利用已实现的 `readObject` 和 `writeObject` 方法。作为这些使用 AMF 3有效地进行序列化和反序列化 ActionScript 对象。(记住 AMF 3 有三大优点：你可以只发送对象引用以避免多余的序列化实例，保持对象关系和处理周期性引用。你可以只发送对象特性，这样当实例重复时类型描述只发送一次。你可以只发送重复的字符串引用以避免产生冗余信息。)你甚至可以在自定义序列化代码中完全省略属性名称，通过固定的顺序发送属性值。

注意

这个例子只关注于只读属性的序列化，不过自定义序列化还有很多其他方面的用途，比如省略属性，避免多余的序列化信息，或包含来自自定义名称空间的属性。

注意 java 中的 `writeExternal` 方法如何写：

```
public void writeExternal(ObjectOutput out) throws IOException
{
    out.writeObject(id);
    out.writeObject(name);
    out.writeObject(description);
    out.writeInt(price);

}
```

对应的客户端 ActionScript 的 `readExternal` 方法：

```

public function readExternal(input:IDataInput) :void
{
    _id = input.readObject() as String;
    name = input.readObject() as String;
    description = input.readObject() as String;
    price = input.readInt();

}

```

## 18.6节。跟踪多个同时发生的服务调用的结果

Contributed by Andrew Alderson

### 18.6.1. 问题

我想确定返回的数据是多个同时发生的服务中哪个调用返回的结果。

### 18.6.2. 解决办法

在每个服务中添加 ASyncToken 变量标记。

### 18.6.3. 讨论

因为 mx.rpc.AsyncToken 是一个动态类，可以在运行期添加属性和方法。Flex 文档里描述的是“一个为异步 RPC 操作设置额外的或标记级别的数据的地方”。

例如，假定有个应用程序使用 DateChooser 控件，每次用户选择一个新的月份时，你需要接收服务器上关于此月份的一个 xml 文件。因为没有方法规定这些返回来的文件的顺序，所以你需要一个办法来识别它们。使用 ASyncToken，你可以添加一个标记属性到服务返回的 result 事件上，如：

Code View:

```

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
layout="horizontal"> <mx:Script>
<![CDATA[
import mx.rpc.events.FaultEvent;
import mx.rpc.events.ResultEvent;
import mx.rpc.AsyncToken;

```

```

import mx.events.DateChooserEvent;
private function scrollHandler(event:DateChooserEvent):void {
    var month:int = event.currentTarget.displayedMonth;
    var monthName:String =
        event.currentTarget.monthNames[month];

    service.url = "xml/" + monthName + ".xml";
    var token:AsyncToken = service.send();
    token.resultHandler = onResult;
    token.faultHandler = onFault;
    token.month = monthName;
}

private function onResult(event:ResultEvent):void {
    resultText.text = "MonthName: " + event.token.month + "\n\n";

    resultText.text += "Result: " + event.result.data.month;
}

private function onFault(event:FaultEvent):void {
    resultText.text = event.fault.faultString;
}

]]> </mx:Script>
<mx:HTTPService id="service"
    result="event.token.resultHandler(event)"
    fault="event.token.faultHandler(event)"/>
<mx:DateChooser id="dateChooser" scroll="scrollHandler(event)"/>
<mx:TextArea id="resultText" width="300" height="200"/>
</mx:Application>

```

上面的代码调用 scrollHandler 事件接收来自服务器的 XML 文件。如果用户点击的很快，你可能在同一时间就会有多个请求发送出去。在 HTPPSERVICE 里，send 方法返回一个 ASyncToken，这样你可以访问它并添加属性标记这个月份到底是和哪个数据一起返回的。你可以使用 ResultEvent 的 token 属性访问你设置的 month 属性。

这个方法也可以用在 WebService 和 RemoteObject 调用上。这些调用中，操作或方法被调用后返回 ASyncToken：

```
var token : AsyncToken = service.login( loginVO );
```

## 18.7节. 使用发布/订阅消息

### 18.7.1. 问题

我想在服务端数据改变能通知客户端 Flex 应用程序或广播消息给所有的监听器。

### 18.7.2. 解决办法

使用 mx.messaging.Producer 和 mx.messaging.Consumer 标签配置用于通信的目标通道和消息事件设置事件处理器。配置这些需要使用 Adobe LiveCycle 或 BlazeDS 服务器。

### 18.7.3. 讨论

发布/订阅模式使用两个组件: mx.messaging.Producer 和 mx.messaging.Consumer。Producer 发送消息给目标, 服务器上处理信息的地址。Consumer 在目标上订阅这些信息, 当获取数据时处理这些消息。

Flex 的消息可以是 ActionScript 消息和 Java Message Service (JMS) 消息。ActionScript 消息只支持客户端"讲"AMF 以及所需的类。JMS 消息允许 LiveCycle 或 BlazeDS 处理 Java Message 服务, 与 JMS 客户端交互。任何可以"讲"JMS 的应用程序都可被 Flex 客户端调用, 任何 Java 应用程序都可发布事件给 Flex。

Consumer 通过 mx.messaging.events.MessageEvent 接收消息:

```
private function receiveChatMessage(msgEvent:MessageEvent):void
{
    var msg:AsyncMessage = AsyncMessage(msgEvent.message);
    trace("msg.body "+msg.body);

}
```

Producer 使用 send 方法发送消息, 该方法接受 mx.messaging.AsyncMessage 作为参数。AsyncMessage 的主体作为值发送给通道的所有订阅者:

```
var msg:AsyncMessage = new AsyncMessage();
msg.body = "test message";

producer.send(msg);
```

完整代码如下:

Code View:

```

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns="*" pageTitle="Simple Flex Chat"
    creationComplete="chatSubscriber.subscribe() ">
    <!-- Messaging Declarations -->
    <mx:Producer id="producer"
        destination="http://localhost:8400/chatDestination"/>
    <mx:Consumer id="subscriber"
        destination="http://localhost:8400/chatDestination"
        message="receiveChatMessage(event)" />
    <mx:Script>
        <![CDATA[

            import mx.messaging.events.MessageEvent;
            import mx.messaging.messages.AsyncMessage;

            private function sendChatMessage():void
            {
                var msg:AsyncMessage = new AsyncMessage();
                msg.body = "test message";
                producer.send(msg);
            }

            private function
            receiveChatMessage(msgEvent:MessageEvent):void
            {
                var msg:AsyncMessage = AsyncMessage(msgEvent.message);
                trace("msg.body "+msg.body);
            }
        ]]>
    </mx:Script>

</mx:Application>

```

## 18.8节. 注册服务端数据类型

### 18.8.1. 问题

我需要在自己的应用程序中注册服务端数据类型，以便从 RemoteObject 返回的对象能正确

转换为远程类的实例。

### 18.8.2. 解决办法

使用 flash.net.RegisterClass 方法或在类申明中标记类为 RemoteClass。

### 18.8.3. 讨论

在反序列化 AMF 数据中的对象为类对象时，该类必须事先在 Flash Player 中注册，这样反序列化才能得到正确的数据类型，如下面 C# 定义的类型：

```
using System;
using System.Collections;

namespace oreilly.cookbook.vo
{
    public class RecipeVO {
        public string title;
        public ArrayList ingredients;
        public ArrayList instructions

        public RecipeVO() {}
    }
}
```

ActionScript 对应的类型为：

```
package oreilly.cookbook.vo
{
    public class RecipeVO

        public var ingredients:Array;
        public var instructions:Array;
        public var title:String;

        public function RecipeVO() {}
    }
}
```

服务将会在 C# 中创建 RecipeVO 对象并返回：

Code View:

```
using System;
using System.Web;
```

```

using oreilly.cookbook.vo;
namespace oreilly.cookbook.service
{
    public class RecipeService
    {
        public RecipeService() { }

        public RecipeVO getRecipe() {
            RecipeVO rec = new RecipeVO();
            rec.title = "Apple Pie";
            string[] ingredients = {"flour", "sugar", "apples",
                "eggs", "water"};
            rec.ingredients = new ArrayList(ingredients);
            string[] instructions = {"instructions are long",
                "baking is hard", "maybe I'll just buy it at the store"};
            rec.instruction = new ArrayList(instructions);
            return rec;
        }
    }
}

```

当服务返回时，可以这样访问 RecipeVO:

Code View:

```

<mx:RemoteObject id="recipeService" destination="fluorine"
    source="oreilly.cookbook.FlexService" showBusyCursor="true"
    result="roResult(event)" fault="roFault(event)" />

<mx:Script>
    <! [CDATA[

        private function initApp():void {
            // we have to register the object for the result to be
            // able to properly cast
            // as the RecipeVO

            flash.net.registerClassAlias("oreilly.cookbook.vo.RecipeVO",
            RecipeVO);
        }

        public function serviceResult(e:ResultEvent):void {
            var rec:RecipeVO = (e.result as RecipeVO)
        }
    ]>

```

```

public function serviceFault(e:FaultEvent) :void {
    trace(" Error :: "+(e.message as String));
}

] ]>

</mx:Script>

```

当使用 registerClassAlias 方法注册类后，匹配的对象可被转换为 RecipeVO 类。

## 18.9节. 与 WebService 通信

### 18.9.1. 问题

我的应用程序需要与服务端的 web 服务进行通信，Web 服务提供 WSDL 信息表述可使用的方法和调用这些方法的具体信息。

### 18.9.2. 解决办法

创建 mx.rpc.WebService 对象，设置 wsdl 属性为 WebService 的 WSDL 地址。

### 18.9.3. 讨论

WebService 组件使用 WSDL 文件与 web 服务建立通信。Flash Player 能识别下列 WSDL 文件的属性：

#### <binding>

指定客户端协议，例如这里的 Flex 应用程序使用 web 服务进行通信，绑定的协议有 SOAP, HTTP GET, HTTP POST, 和多功能因特网邮件扩展(MIME)。Flex 只支持 SOAP 绑定。

#### <fault>

指定错误显示信息

#### <input>

指定客户端发送给 web 服务的信息。

### `<message>`

定义 web 服务操作传输的数据

### `<operation>`

定义`<input>`, `<output>`, 和 `<fault>` 的标签集合

### `<output>`

指定 web 服务发送给客户端的信息

### `<port>`

S 指定 web 服务的终端，它关联绑定和网络地址。

### `<portType>`

定义 web 服务提供的操作

### `<service>`

定义一组`<port>`标签集合，每个服务映射到一个`<portType>`标签，以指定不同的访问方式。

### `<types>`

定义 web 服务信息使用的数据类型

Flex 应用程序检查 WSDL 文件以确定所有服务所提供的方法和每个服务返回的数据类型。一个典型的 WSDL 文件定义服务的名称，使用的类型和返回的数据类型。

创建 WebService 对象，设置 id 和 WSDL 文件的位置：

Code View:

```
<mx:WebService id="userRequest" wsdl="http://localhost:8400/service/service?wsdl">  
    <mx:operation name="getRecipes" result="getRecipeHandler()"  
        fault="mx.controls.Alert.show(event.fault.faultString)" />  
</mx:WebService>
```

WebService 发出 LoadEvent 类型事件或 LoadEvent.LOAD，它指示 WebService 已载入和解析 wsdl 属性所指定的 WSDL 文件，并准备好所有方法已可调用。在这之前 WebService 对象是不能被调用，所以强烈建议使用这事件指示服务已可调用。WebService 组件也定义了一个 ready 布尔属性可检测 WSDL 文件已被载入，WebService 已准备好。下面的例子，定义了一个方法和事件处理函数处理服务的 result 和 fault 事件：

Code View:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
```

```

width="400" height="300">
<mx:WebService id="userRequest"
    wsdl="http://localhost:8500/service/service?wsdl"
    load="callService()">
    <mx:operation name="getRecipes" resultFormat="object"
        fault="createRecipeFault(event)"
        result="createRecipeHandler(event)"/>
</mx:WebService>
<mx:Script>
<! [CDATA[
    import mx.rpc.events.FaultEvent;
    import mx.collections.ArrayCollection;
    import mx.rpc.events.ResultEvent;
    import mx.controls.Alert;

    private function callService():void {
        userRequest.getRecipes();
    }

    private function
    createRecipeHandler(event:ResultEvent):void {
        var arrayCol:ArrayCollection = event.result
            as ArrayCollection;
    }

    private function
    createRecipeFault(event:FaultEvent):void {
        Alert.show(" error :: "+event.message);
    }
]]>
</mx:Script>
</mx:Application>

```

## 18.10节. 添加 SOAP 头到 WebService 请求

### 18.10.1. 问题

我想发送 SOAP 头到 WebService 组件的请求中。

### 18.10.2. 解决办法

创建一个 SOAPHeader 对象，参数为所使用的名称空间和添加到 header 的内容。然后调用 WebService.addHeader 方法发送带有请求的 header。

### 18.10.3. 讨论

Web 服务经常用 SOAP 头接收登录，用户信息或其他数据。创建 SOAPHeader 需要一个包含数据的限定名称空间 QName 和添加到 header 的对象：

`SOAPHeader(qname:QName, content:Object)`

这里是创建两个 SOAPHeader 对象的例子：

Code View:

```
// Create a QName that can be used with your header
```

```
var qname:QName=
    new QName("http://soapinterop.org/xsd", "CookbookHeaders");
var headerone:SOAPHeader =
    new SOAPHeader(qname, {string:"header_one",int:"1"});
var headertwo:SOAPHeader =
    new SOAPHeader(qname, {string:"header_two",int:"2"});
```

要把这个 header 添加到 web 服务的所有请求中，需调用 WebService 对象本身的 addHeader 方法：

```
// calling addHeader on the WebService
```

```
service.addHeader(headerone);
```

添加 SOAPHeader 到指定的方法上：

```
// Add the headertwo SOAP Header to the getRecipe operation.
```

```
service.getRecipes.addHeader(headertwo);
```

如果 SOAP 头不再需要了，调用 WebService 或方法本身的 clearHeaders 方法：

```
service.clearHeaders();
```

```
service.getRecipes.clearHeaders();
```

## 18.11节. 解析 Webservice 的返回的 SOAP 响应

### 18.11.1. 问题

我需要解析返回的 SOAP 响应。

### 18.11.2. 解决办法

使用 Flash Player 将 WebService 返回的 SOAP 编码 XML 的 SOAP 类型转换为 ActionScript 类型。

### 18.11.3. 讨论

返回的 SOAP 响应可用 E3X 表达式进行解析。常用的类型如表 [Table 18-1](#), 以及对应的 SOAP 和 ActionScript 表示。

**Table 18-1. SOAP 类型和对应的 ActionScript 类型**

Generic Type	SOAP	ActionScript 3
String	xsd:String	String
Integer	xsd:int	Int
Float	xsd:float	Number
Boolean	xsd:Boolean	Boolean
Date	xsd:date	Date
Array	xsd:string[], xsd:int[], and so forth	ArrayCollection
Object	Element	Object
Binary	xsd:Base64Binary	flash.utils.ByteArray
Null	xsl:Nil	Null

一个 WSDL 文件定义如下返回类型:

```
<wsdl:types>

  <schema elementFormDefault="qualified"
    targetNamespace = "http://cookbook.webservices.com"
```

```

xmlns = "http://www.w3.org/2001/XMLSchema">

<complexType name="Recipe">

<sequence>

<element name="title" nillable="true" type="xsd:string"/>

<element name="ingredients" nillable="true" type="xsd:string[]"/>

<element name="instructions" nillable="true" type="xsd:string[]"/>

</sequence>

</complexType>

</schema>

</wsdl:types>

```

WebService 的返回如下：

```

<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"

soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:ns="http://cookbook.oreilly.com/service">

<ns:GetRecipes>

<ns:Recipe>

<ns:title>"Blueberry Pie"</ns:title>

<SOAP-ENC:Array SOAP-ENC:arrayType="xsd:string[3]">

<ns:ingredient>"Blueberry"</ns:ingredient>

<ns:ingredient>"Sugar"</ns:ingredient>

<ns:ingredient>"Crust"</ns:ingredient>

</SOAP-ENC:Array>

<SOAP-ENC:Array SOAP-ENC:arrayType="xsd:string[3]">

```

```
<ns:instruction>"Blueberry"</ns:instruction>

<ns:instruction>"Sugar"</ns:instruction>

<ns:instruction>"Crust"</ns:instruction>

</SOAP-ENC:Array>

</ns:Recipe>

</ns:GetRecipes>

</soap:Body>

</soap:Envelope>
```

这个响应可用(.)符号解析出所需对象，和处理任何 XML 对象一样。更多有关 SOAP 类型和 ActionScript 类型的区别，请看 <http://www.adobe.com/go/kb402005>

## 18.12节. 使用 **SecureAMFChannel** 进行 AMF 的安全通信

### 18.12.1. 问题

我需要使用 AMF 数据和安全 Sockets 层(SSL)通过 Flash Remoting 进行通信。

### 18.12.2. 解决办法

在编译程序所使用的 services-config.xml 文件中定义你的 channel 为 SecureAMFChannel。

### 18.12.3. 讨论

SecureAMFChannel 可让你使用基于 SSL 的 AMFChannel 通信，以确保通过 AMFChannel 发送的数据都是安全的。要创建一新的 channel，使用安全版本的 AMF 类，只要在 services-config.xml 文件的 channel 中使用 mx.messaging.channels.SecureAMFChannel 作为其类，终端 (endpoint)设置为 flex.messaging.endpoints.SecureAMFEndpoint 类：

Code View:

```

<channels>
    <channel ref="secure-amf"/>
</channels>

<channel-definition id="secure-amf" class="mx.messaging.channels.SecureAMFChannel">
    <endpoint uri="https://{{server.name}}:{{server.port}}/gateway/" 
        class="flex.messaging.endpoints.SecureAMFEndpoint"/>
    <properties>
        <add-no-cache-headers>false</add-no-cache-headers>
        <polling-enabled>false</polling-enabled>
        <serialization>
            <instantiate-types>false</instantiate-types>
        </serialization>
    </properties>
</channel-definition>

```

上面的代码可用于 ColdFusion, Adobe LiveCycle, 和 BlazeDS:

Code View:

```

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="400" height="300" creationComplete="init()">
    <mx:RemoteObject id="channelRO"/>
    <mx:Script>
        <! [CDATA[
            import mx.messaging.ChannelSet;
            import mx.messaging.channels.SecureAMFChannel;

            private var cs:ChannelSet

```

```

private function init():void {
    cs = new ChannelSet();
    // note that the name of the channel is the same as
in the services-config.xml file
    var chan: SecureAMFChannel =
        new SecureAMFChannel("secure-amf", "gateway")
    chan.pollingEnabled = true;
    chan.pollingInterval = 3000;
    cs.addChannel(chan);
    channelRO.channelSet = cs;
}

}]>
</mx:Script>

</mx:Application>

```

现在你可以使用此 channel 调用 RemoteObject 和建立安全的 AMF 轮询调用。

## 18.13节. 通过二进制 **Socket** 发送和接收二进制数据

### 18.13.1. 问题

我想接收二进制数据，处理完后以同样的二进制格式发送数据。

### 18.13.2. 解决办法

使用 flash.net.Socket 打开 socket 连接。

### 18.13.3. 讨论

flash.net.Socket 是 Flex 框架或 ActionScript 3中最低级别的通信工具，使得你能建立 socket 连接并读取和写入原始的二进制数据。Socket 可接收和发送 POP3, SMTP, IMAP 信息，甚至是自定义二进制格式。Flash Player 可以使用这样的二进制协议直接与服务器通信。

要创建一个 Socket，先要使用构造器创建 Socket 实例，调用 connect 方法，传递 IP 地址或域名和端口号作为方法参数：

```

var socket:Socket;
//create the new socket and connect to 127.0.0.1 on port 8080
private function init():void {
    socket = new Socket();
    socket.addEventListener(ProgressEvent.SOCKET_DATA,
        readSocketData);
    socket.connect("127.0.0.1", 8080);
}
// send data to the socket
private function sendSocketData(string:String):void {
    // send the string data and specify the encoding for the
    string
    // in this case iso-08859-1, standard western european
    encoding
    socket.writeMultiByte(string, "iso-8859-1");
}

// when data is passed to socket, read it into a new ByteArray
private function readSocketData(progressEvent:ProgressEvent):void
{
    trace(progressEvent.bytesLoaded);
    var ba:ByteArray = new ByteArray();
    trace(socket.readBytes(bs));
}

```

在上面的 sendSocketData 方法中， writeMultiByte 方法通过 Socket 连接发送数据。该方法接受一字符串值作为发送的二进制数据，第二个参数采用的数据编码。 readSocketData 方法读取任何从 Socket 发送来的数据，并把读取的数据字节存到 ByteArray 对象。如要读取 ByteArray 的数据，可使用各种版本的 read 方法，包括 integers, strings, 和 arrays 的 read 方法。把 Object 作为二进制数据发送后，如果该类型已通过 flash.net.RegisterClass 方法注册过，即可使用 ByteArray 的 readObject 方法读取。

要连接端口号低于1024的 Socket，你需要在站点根目录有一个 cross-domain.xml 文件，其中明且定义允许的端口号。例如要允许 Flash Player 与 Web 服务器的80端口通信，可这样写：

```

<?xml version="1.0"?>

<cross-domain-policy>

    <allow-access-from domain="*" to-ports="80" />

</cross-domain-policy>

```

正确放置好 cross-domain.xml 文件后， Socket 就可以和正确的端口连接了。

## 18.14节. XMLSocket 通信

### 18.14.1. 问题

我想创建一个服务器的连接，不需要请求就可接收 XML 数据。

### 18.14.2. 解决办法

使用 XMLSocket 类打开服务器连接，它允许服务器发送信息给客户端，当数据到达客户端时已经有被接收和处理的信息了。

### 18.14.3. 讨论

XMLSocket 类实现了客户端 socket，让 Flash Player 和 AIR 应用程序可以指定 IP 地址和域名即可连接到服务器。要使用 XMLSocket 类，服务器端必须运行一个能了解 XMLSocket 所使用协议的伺服器。协议为::

XML 信息是通过全双工传输控制协议/互联网协议(TCP/IP) 流发送。

每个 XML 信息都是一个完整的 XML 文档，以0字节结束。

通过单个 XMLSocket 连接可发送和接收无线数量的 XML 信息。要连接到 XMLSocket 对象，先创建一个 XMLSocket 对象，然后使用 IP 地址或域名和端口数字作为参数调用 connect 方法：

```
var xmlsock:XMLSocket = new XMLSocket();  
  
xmlsock.connect("127.0.0.1", 8080);
```

端口号是必须的，因为 XMLSocket 连接的端口号不能小于1024。给 DataEvent.DATA 事件添加监听器，用于接收数据：

```
xmlsock.addEventListener(DataEvent.DATA, onData);  
private function onData(event:DataEvent):void  
{  
    trace("[" + event.type + "] " + XML(event.data));  
}
```

返回的字符串可转换为 XML，使用 E4X 进行解析。

## 第十九章. XML(tonyian)

Flex 3 和 ActionScript 3.0 支持 ECMAScript 或 E4X 标准的 XML 语法，使你可以通过点(.)标记来存取一个 XML 分层架构的各个节点。通过 E4X 所提供的简易标记，你可以根据其名称或索引，轻易地存取特定的节点或节点组，而无需使用一些复杂的自定义回调函数。同时，它也定义了各种方法及属性，用以存取 XML 对象的各个部分，包括注释、命名空间和处理指令等。Flex Framework 及 ActionScript 3.0 定义了两个级别的对象来处理 XML 语言：XML 和 XMLList 对象。XML 对象代表单一的 XML 元素，一个 XML 文档或该文档中的一个单值元素。XMLList 则代表一组跟其他组同级的 XML 元素。XMLList 对象不需要设置顶级节点，例如：

```
<item id="2" name="Chewing Gum"/>
<item id="3" name="Cotton Candy"/>
<item id="4" name="Candy Bar"/>
```

XML 对象需要设定顶级节点：

```
<order>
  <item id="2" name="Chewing Gum"/>
  <item id="3" name="Cotton Candy"/>
  <item id="4" name="Candy Bar"/>
</order>
```

在一个 XML 文档中，XML 及 XMLList 对象分别定义了一些方法，用来追加、重新命名和重新定义父级节点，藉以进行各种 XML 转换或建立 XML 文档。要对文档进行查询的话，可以通过 E4X 查询语法来存取特定属性或对该 XML 文档进行筛选。如果要处理 XML 的命名空间，则可以通过 XMLUtil 类所定义的方法来对命名空间进行比较和存取。

### 19.1 节. 加载一个 XML 文件

#### 19.1.1. 问题

如何加载及处理一个外部 XML 文件？

#### 19.1.2. 解决办法

使用 `HTTPService` 组件来加载 XML 文件，并把 `resultFormat` 设置为”xml”。 或者，使用 `flash.net.URLLoader` 类来创建一个 `URLLoader` 实例，并调用 `load` 方法来加载 XML 文件。

### 19.1.3. 讨论

在默认的情况下，`HTTPService` 组件会把任何加载的 XML 转换成一个 ActionScript 对象。要避免这种情况，可以对 `HTTPService` 对象的 `resultFormat` 属性进行以下设置：

代码如下：

```
<mx:HTTPService url="http://server/xmlDoc.xml" id="xmlService"
resultFormat="e4x" result="xmlObj=XML(xmlService.lastResult)"/>
```

`HTTPService` 的 `lastResult` 是用来设置 `xmlObj` 的变量值。

如果该 XML 文件为静态或者它只需要被加载一次，使用 `flash.net.URLLoader` 类来进行加载会比较方便快捷：

```
private var loader:URLLoader = new URLLoader();

private function init():void {
    loader.addEventListener(Event.COMPLETE, setResult);
    var req:URLRequest = new URLRequest();
    req.url = "http://server/xmlDoc.xml";
    loader.load(req);
}

private function setResult(event:Event):void {
    trace(XML(loader.data).toString());
}
```

`Event.COMPLETE` 事件监听器必须添加到 `URLLoader` 里，这样当文件加载完成时，应用程序就会得到通知。所加载的数据将会存放到 `URLLoader.data` 属性里，而且不能被转换或修改。如果该 XML 数据需要被动态加载或多次加载，使用 `HTTPService` 来加载会比较容易。以下的完整代码展示这两种方法：

代码如下：

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300" creationComplete="init()">
<!-- if the resultFormat is not set, then the result will be
parsed as an Object -->
<!-- 如果resultFormat没有设置，该结果会解析为一个对象 -->
```

```

<mx:HTTPService url="http://server/xmlDoc.xml" id="xmlService"
resultFormat="e4x" result="xmlObj = XML(xmlService.lastResult)" />

<mx:Script>
<! [CDATA[

private var loader:URLLoader = new URLRequest();
[Bindable]
private var xmlObj:XML;

private function init():void {
    loader.addEventListener(Event.COMPLETE, setResult);
}

private function urlLoaderSend():void {
    var req:URLRequest = new URLRequest();
    req.url = "http://server/xmlDoc.xml";
    loader.load(req);

}

private function setResult(event:Event):void {
    xmlObj = (loader.data as XML)

}

]]>
</mx:Script>
<mx:Button click="xmlService.send()" />
<mx:Button click="urlLoaderSend()" label="load via URLLoader"/>
<mx:TextArea text="{xmlObj.toString()}" />
</mx:VBox>

```

## 19.2 节. 通过 E4X 语法遍历 XML 文档

### 19.2.1 问题

对于一个基于多个字段(attribute)的属性值的 XML 文件, 应如何选取其中的节点?

### 19.2.2. 解决办法

使用 E4X 语法的”@”运算符来存取属性, “([])运算符(数组索引)”是用来指示多个子节点之间的关系, 而”.”运算符则用来表示已命名子节点之间的关系。

### 19.2.3. 讨论

通过 E4X, 你可以在子节点的名称后附加一个”.”运算符, 这样你就可以存取 XML 文件中特定的子节点。

例如, 从以下的文件中

```
var xml:XML = <foo>
<bar>Hello World</bar>
</foo>
```

你可以通过以下的方法来存取<bar>的数值:

```
xml.bar
```

由于<foo>为该 XML 对象的根节点, 所以无需对其进行引用。

要存取一个文件中节点的属性, 例如:

```
var xml:XML = <foo>
<bar type="salutation">Hello World</bar>
</foo>
```

可以使用”@”运算符来指定所需的属性值为一个字段:

```
xml.bar.@type
```

要存取多个名称相同的子节点, 可以使用”[]”运算符。 好像以下的例子:

```
var xml:XML = <foo>
<bar type="salutation">Hello World</bar>
<bar type="salutation">Hola</bar>
<bar type="salutation">Guten Tag</bar>
</foo>
```

你可以使用以下的方法来存取`<bar>`系列的第三个对象:

```
xml.bar[2].@type
```

对于一个用以定义在菜单上项目的简单 XML 结构, 可以使用如下的代码段:

```
private var xmlItems:XML = <order>
    <item id='1'>
        <menuName>burger</menuName>
        <price>3.95</price>
    </item>
    <item id='2'>
        <menuName>fries</menuName>
        <price>1.45</price>
    </item>
</order>

private var arr:Array;

private function init():void {
    arr = new Array();
    for each(var xml:XML in xmlItems.item) {
        arr.push(Number(xml.@id));
    }
}
```

要对属性值或节点进行测试的话, 可以使用相等运算符(`==`):

```
trace(xmlItems.item.(@id == "2").menuName);
```

任何符合条件的节点都会被返回, 否则就会被忽略。 以下的例子会把 Label 组件的 text 属性设置为一个 id 等于 2 的 item 的 menuName:

```

private var xmlItems:XML = <order>
    <item id="1">
        <menuName>burger</menuName>
        <price>3.95</price>
    </item>
    <item id="2">
        <menuName>fries</menuName>
        <price>1.45</price>
    </item>
</order>

private function init():void {
    xmlLabel.text = xmlItems.item.(@id == "2").menuName;
}

] ]>
</mx:Script>
<mx:Label id="xmlLabel"/>

```

相等(==)跟不等(!=)运算符两者都可以用来测试一个属性或节点的值，它们可以为字符串或数字，所返回的结果为一个布尔值。

## 19.3 节. 使用正则表达式在 E4X 中进行查询

### 19.3.1. 问题

如何通过使用正则表达式作为查询的一部分，来创建复杂的 E4X 查询？

### 19.3.2. 解决办法

以文本形式把正则表达式添加到 E4X 语句中，并调用正则表达式的 test 方法。

### 19.3.3 讨论

通过结合正则表达式及 E4X，可以对 XML 节点进行准确的筛选。正则表达式的文本语法允许你在不调用构造函数的情况下，添加一个正则表达式。同时，也可以对 XML 节点的值或属性使用使正则表达式的 test 方法。以下的代码行，对 item 节点的 id 属性进行了测试：

```
xmlItems.item.(/\d\d\d/.test(@id)).price
```

任何含有 3 位数字 id 属性的项目，将会返回一个该属性的价格值。 任何不包括这些字段或从正则表达式的 test 方法返回 true 的项目，将不会返回任何值。 下列代码展示了一个 XML 的循环回圈及通过 E4X 表达式来对每个节点进行测试：

代码如下：

```
private var xmlItems:XML = <order>
    <item id="1">
        <menuName>burger</menuName>
        <price>3.95</price>
    </item>
    <item id="100">
        <menuName>burger</menuName>
        <price>3.95</price>
    </item>
    <item id="2000">
        <menuName>fries</menuName>
        <price>1.45</price>
    </item>
</order>

private var arr:Array;

private function init():void {
    arr = new Array();
    for each ( var xml:XML in xmlItems) {
        arr.push(xmlItems.item.(/\d\d\d/.test(@id)).price);
    }
    trace(arr);
}
```

值得注意的是，你可以使用 E4X 查询作为控件中数据绑定的一个属性。

## 19.4 节. 添加一个 XMLList 到 XML 对象

### 19.4.1. 问题

如何追加一个 XMLList 对象到 XML 对象的节点里?

### 19.4.2. 解决方法

使用 E4X 表达式, 找出需要追加的 XMLList 对象所在的节点, 然后在该节点上调用 appendChild 方法。

### 19.4.3. 讨论

通过使用 XML 类的 appendChild 方法, 你可以直接添加一个 XMLList 到一个 XML 对象或另一个 XMLList 对象里。 例如说下列的 XML 对象:

```
var xmlNode:XML = <data>
    <item id="1"/>
    <item id="2"/>
    <item id="3"/>
</data>
```

然后, 对新的节点调用 appendChild 方法

```
var newXML:XML = <item id="4"/>
xmlNode.appendChild(newXML);
```

会得出以下结果:

```
var xmlNode:XML = <data>
    <item id="1"/>
    <item id="2"/>
    <item id="3"/>
    <item id="4"/>
</data>
```

新节点会被添加到 XML 对象的根节点。 你可以在 XML 对象内对任何节点调用 appendChild 方法。

代码如下:

```
var list:XMLList = new XMLList('
```

```

<characteristic name="cuts through metal"/>
<characteristic name="never dulls"/>
<characteristic name="dishwasher safe"/>
<characteristic name="composite handle"/>');
var node:XMLList = xmlNode.item(@id == 3);
node.appendChild(list);

}

```

要把一个 XMLList 的项目添加到另一个里面，你可以对原列表使用循环回圈，把原来的索引值分配到另一个 XMLList 里。

```

var newXML:XMLList = new XMLList();
for(var i:int = 0; i<list.length(); i++) {
    newXML[i] = list[i];
}

```

如果 newXML 的类型为 XML，这个方法是不可行的。通过使用 appendChild 方法，对列表进行循环，可以从列表里添加某个或所有项目。

```

var newXML:XML = <data></data>;
for(var i:int = 0; i<list.length(); i++) {
    newXML.appendChild(list[i]);
}

```

## 19.5 节. 对一个 XMLList 或 E4X 查询进行绑定

### 19.5.1. 问题

如何将一个控件跟一个 E4X 查询返回值进行绑定并存放到一个 XML 对象里？

### 19.5.2. 解决方法

使用绑定标记 “{}”去包装 E4X 表达式，并设置控件的属性。

### 19.5.3. 讨论

举例说以下的 XML 文件：

代码如下：

```
[Bindable]  
private var xmlItems:XML =  
<CATALOG>  
    <PLANT id="2">  
        <COMMON>Bloodroot</COMMON>  
        <BOTANICAL>Sanguinaria canadensis</BOTANICAL>  
        <ZONE>4</ZONE>  
        <LIGHT>Mostly Shady</LIGHT>  
        <PRICE>$2.44</PRICE>  
        <AVAILABILITY>031599</AVAILABILITY>  
    </PLANT>  
    <PLANT id="3">  
        <COMMON>Columbine</COMMON>  
        <BOTANICAL>Aquilegia canadensis</BOTANICAL>  
        <ZONE>3</ZONE>  
        <LIGHT>Mostly Shady</LIGHT>  
        <PRICE>$9.37</PRICE>  
        <AVAILABILITY>030699</AVAILABILITY>  
    </PLANT>  
    <PLANT id="5">  
        <COMMON>Marsh Marigold</COMMON>  
        <BOTANICAL>Caltha palustris</BOTANICAL>  
        <ZONE>4</ZONE>  
        <LIGHT>Mostly Sunny</LIGHT>  
        <PRICE>$6.81</PRICE>  
        <AVAILABILITY>051799</AVAILABILITY>  
    </PLANT>  
</CATALOG>
```

Label 控件的 text 值可以设置为 E4X 查询的结果，用以返回 id 为 5 时其 PLANT 的 PRICE 值。

```
<mx:Label text="{xmlItems.PLANT.(@id == 5).PRICE}" />
```

由于 E4X 表达式的结果所返回的多个节点为一个 XMLList，该数值可以设置为 ComboBox 的dataProvider。

```
<mx:ComboBox dataProvider="{xmlItems.PLANT.(ZONE == 4).PRICE}" />
```

同样地，当一个不带有子节点的 E4X 表达式返回多个节点的时候，整个节点会被返回，可以用来设置 DataGrid 控件的dataProvider，就像如下：

```
<mx:DataGrid dataProvider="{xmlItems.PLANT.(ZONE == 4)}">
<mx:columns>
    <mx:DataGridColumn dataField="COMMON"/>
    <mx:DataGridColumn dataField="PRICE"/>
    <mx:DataGridColumn dataField="AVAILABILITY"/>
    <mx:DataGridColumn dataField="LIGHT"/>
</mx:columns>
</mx:DataGrid>
```

## 19.6 节. 从数组中生成 XML 对象

### 19.6.1. 问题

如何从一个数组中生成一个 XML 对象？

### 19.6.2. 解决方法

使用数组索引去存取数组内的数据，并由对应的数据对象创建 XML 对象。然后，调用 XML 的 appendChild 方法把新建的节点添加到主 XML 里。

### 19.6.3. 讨论

通过 XML 类的 appendChild 方法，可以添加节点到 XML 文档里。可是，要将属性添加到 XML 对象，你需要使用绑定标记运算符”{}”去填充一个节点值或属性：

```
var arr:Array = [1, 2, 3, 4, 5];

var xml:XML = new XML(<data></data>);

for(var i:int = 0; i<arr.length; i++) {
    xml.appendChild(<id>{arr[i]}</id>);
}
```

将 `<id>` 标记的值设置为数组索引处的值并将该值追加到 XML 对象：

```

var ids:Array = [121, 122, 123];
var names:Array =
  ["Murphy", "Pat",
  ["Thibaut", "Jean"],
  ["Smith", "Vijay"]]

var x:XML = new XML("<employeeList></employeeList>");

for (var i:int = 0; i < ids.length; i++)
{
  var newnode:XML = new XML();
  newnode =
    <employee id={ids[i]}>
      <last>{names[i][0]}</last>
      <first>{names[i][1]}</first>
    </employee>

  x = x.appendChild(newnode)
}

```

## 19.7 节. 如何处理 XML 服务里所返回的命名空间

### 19.7.1. 问题

如何从一个自定义的命名空间及扩展的 Web 服务里，分析其返回的 XML？

### 19.7.2. 解决办法

申明一个命名空间变量，将它设置为返回的 XML 命名空间的位置，然后在进行任 XML 处理之前，先调用该命名空间的”use”方法。

### 19.7.3. 讨论

对包含自定义命名空间的 XML 进行分析是比较困难的，它要求该命名空间一定要在任何 XML 返回前被申明，同时要在该命名空间内进行分析。不少 Web 服务都会返回一些包含命名空间的声明，好像以下的例子：

HTTP/1.1 200 OK

Content-Type: application/soap+xml; charset=utf-8

Content-Length: nnn

```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

    <soap:Body xmlns:m="http://www.example.org/stock">
        <m:PriceResult>
            <m:Price>34.5</m:Price>
        </m:PriceResult>
    </soap:Body>

</soap:Envelope>
```

xmlns:soap="http://www.w3.org/2001/12/soap-envelope"

以上所申明的代码行一定要申明在 XML 被处理之前。这是通过如下的命名空间声明来达成的：

代码如下：

```
private namespace w3c = "http://www.w3.org/2001/12/soap-envelope";
use namespace w3c;
To access the Price node of the above SOAP response, use the namespace title in the
E4X statement as shown here:var prices:XMLList = xml.m::PriceResult.m::Price;
```

通过”.”运算符可以存取任何使用限定命名空间的 XML 的子节点，同时该命名空间申明时应跟随着”::”运算符及节点名义。例如以下的 XML 对象：

```
<m:PriceResult>
    <m:Price>34.5</m:Price>
</m:PriceResult>
```

“price”节点会通过以下的方法来存取：

m::PriceResult.m::Price

## 19.8 节. 将 ActionScript 数据对象编码成 XML

### 19.8.1. 问题

如何将一个 ActionScript 对象转换成 XML.。

### 19.8.2. 解决办法

可以使用 SimpleXMLEncoder.encodeValue 方法把一个对象及其属性写入到一个 XMLDocument 对象里。

### 19.8.3. 讨论

当创建 XML 以传送 Web 服务或服务端方法的 URL 为 XML 时, SimpleXMLEncoder 对象是非常有用的。在这个对象中定义了一个名叫 encodeValue 的方法, 其签名格式如下:

```
encodeValue(obj:Object, qname:QName, parentNode:XMLNode):XMLNode
```

所生成的 XML 会由该方法返回, 同时会附加到 parentNode 所在的 XMLDocument 对象内的 XMLNode 中, 该方法会要求所有旧式 XMLDocument 所生成的 XML 附加到其中。当 XMLDocument 生成以后, 可以通过 XML 对象的构造函数, 并将该文档当作参数传递到构造函数中, 由此转换成 XML 对象:

```
var doc:XMLDocument = new XMLDocument('<data></data>');
var xml:XML = new XML(doc);
```

将一个对象编码到一个 XML 文档的完整代码列表如下所示:

代码如下:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300" creationC
omplete="init()">
<mx:Script>
<! [CDATA [
import mx.rpc.xml.SimpleXMLEncoder;

private var o:Object = {
name:"Josh",
description_items:{height:'183cm', weight:'77k'}};

private var doc:XMLDocument;
```

```

private function init():void {
    doc = new XMLDocument('<data></data>');
    var simpleEncode:SimpleXMLEncoder =
        new SimpleXMLEncoder(doc);
    var node:XMLNode = simpleEncode.encodeValue(o,
        new QName('http://localhost/ns/ws', 'ls'),
        doc.firstChild);

}
]]>
</mx:Script>
</mx:Canvas>

```

当调用了 SimpleXMLEncoder.encodeValue 方法之后, XMLDocument 对象将会变成以下的构造:

```

<data>
  <obj>
    <description_items>
      <height>183cm</height>
      <weight>77k</weight>
    </description_items>
    <name>Josh</name>
  </obj>
</data>

```

## 19.9 节. 使用复杂 XML 数据来填充组件

### 19.9.1. 问题

当所显示的 XML 包含多个内嵌子节点时, 应如何确保该层次架构会被正确地展示?

### 19.9.2. 解决办法

可以使用 mx.controls.Tree 或 AdvancedDataGrid 控件去显示数据。创建一个 HierarchicalData 对象, 并将该 XML 传递给它, 以确保 Tree 或 AdvancedDataGrid 可以在 XML 架构中正确地定位。

### 19.9.3. 讨论

要去显示下列基于 Tree 或 AdvancedDataGrid 控件的 XML 菜单，你可以使用 HierarchicalData 对象去确保数据会被正确地显示。

代码如下：

```
<mx:XMLList id="foodXML">
    <menu label="Menu">
        <breakfast_menu label="Breakfast">
            <food label="Eggs and Homefries" price="$6.95"
                description="Two eggs, homefries, toast, coffee"/>
            <food label="Item" name="Strawberry Belgian Waffles"
                price="$7.95" descrip
tion="light Belgian waffles with strawberries and whipped cream"/>
        </breakfast_menu>
        <lunch_menu label="Lunch">
            <food label="Soup and Sandwich" price="$8.95"
                description="Sandwich served with hot soup"/>
        </lunch_menu>
    </menu>
</mx:XMLList>
```

HierarchicalData 会把 XML 或内嵌数据对象包装以提供下列方法，用来让显示控件可以正确地显示 XML 中各个节点的关系：

`canHaveChildren(node:Object):Boolean`

如果该节点可以包含子项，则返回 true。

`getChildren(node:Object):Object`

返回表示节点子项的 Object。

`getData(node:Object):Object`

返回节点中的数据。

`getRoot():Object`

返回根数据项。

```
hasChildren(node:Object):Boolean
```

如果该节点包含子项，则返回 true。

要把 XML 包装到一个 HierarchicalData 对象里，并传递到其构造函数中，可以这样做：

```
var hData:HierarchicalData = new HierarchicalData(myXMLObj);
```

在以下的例子，menuList XMLList 项目会被传递到一个 HierarchicalData 对象中，并在 AdvancedDataGrid 组件中显示：

代码如下：

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300" creationComplete="init()">
<mx:Script>
<![CDATA[

    import
mx.controls.advancedDataGridClasses.AdvancedDataGridColumn;
    import mx.collections.HierarchicalData;

    private function init():void {
}

    private function labelFunc(item:Object,
column:AdvancedDataGridColumn):String
{
    if(String(item.@label) != "") {
        return String(item.@label);
    }
    return "";
}

]]>
</mx:Script>
<mx:AdvancedDataGrid
dataProvider="{new HierarchicalData(foodXML)}" width="100%"
height="100%">
<mx:columns>
<mx:AdvancedDataGridColumn dataField="name"
labelFunction="labelFunc"/>
```

```
<mx:AdvancedGridColumn dataField="@description"/>
<mx:AdvancedGridColumn dataField="@price"/>
</mx:columns>
</mx:AdvancedDataGrid>
</mx:VBox>
```

## 19.10 节. 从 Web 服务中把 XML 译码成为强类型对象

### 19.10.1. 问题

如何将一个 XML 或 XMLList 对象转换成一个或多个强类型的对象。

### 19.10.2. 解决办法

通过使用限定命名空间及 SimpleXMLDecoder 类把 XML 译码成对象，然后使用 SchemaTypeRegistry.registerClass 方法对类进行注册。

### 19.10.3. 讨论

SchemaTypeRegistry.registerClass 可以让你注册一个类型为由 Web 服务所返回的类。这个类一定要在 WSDL 文件被描述，而有关的方法及类型亦会在该文件中被描述。例如，一个名为”Plant”的对象的所有属性定义为：

代码如下：

```
<types>
  <xsd:schema
    targetNamespace=
      "http://localhost/ns/ws"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:complexType name="Plant">
      <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" name="common"
          nillable="true" type="xsd:string"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="botanical"
          nillable="true" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:schema>
</types>
```

```

<xsd:element maxOccurs="1" minOccurs="1" name="zone"
    nillable="true" type="xsd:string"/>
<xsd:element maxOccurs="1" minOccurs="1" name="light"
    nillable="true" type="xsd:string"/>
<xsd:element maxOccurs="1" minOccurs="1" name="price"
    nillable="true" type="xsd:string"/>
<xsd:element maxOccurs="1" minOccurs="1" name="availability"
    nillable="true" type="xsd:int"/>
</xsd:sequence>
</xsd:complexType>
</xsd:schema>
</types>

```

该方法会返回一个像以下的 Plant 对象：

```

<binding name="PlantService" type="tns:Plant">
    <soap:binding style="rpc"
        transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getPlants">
        <soap:operation soapAction="getPlants"/>
        <input>
            <soap:body use="encoded"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="http://localhost/ns/ws"/>
            <soap:header message="tns:getPlants" part="header" use="literal"/>
        </input>
        <output>
            <soap:body use="encoded"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="http://localhost/ns/ws"/>
        </output>
    </operation>
</binding>

```

值得注意是，服务类型在这里被定义为”tns:Plant”，即表示该服务会返回一些 Plant 对象，它们跟之前代码中所定义的是一样的。 SchemaTypeRegistry 通过这个声明去把 Web 服务中的 Plant 对象映像到对应数据的 ActionScript 表示形式中。这个方法需要一个限定的 Namespace 和一个包含这个类的 ActionScript 表示形式的 Class 对象。

```
var qname: QName = new QName ("http://localhost/ns/ws", "Plant");
```

```
mx.rpc.xml.SchemaTypeRegistry.getInstance().registerClass(qname,
Plant);
```

一个简单值的对象会由 Plant 类生成，它会包含 Plant 对象所需数据的公共属性。

```
package oreilly.cookbook

{
    public class Plant
    {

        public var common:String;
        public var botanical:String;
        public var zone:String;
        public var light:String;
        public var price:String;
        public var availability:String;

    }
}
```

当这个类完成注册后，Web 服务的结果对象会被转换成 Plant 类型，这样子你就可以直接处理强类型对象，而无须使用 AMF 服务。

代码如下：

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300" creationComplete="init()">
<mx:WebService id="ws" wsdl="http://localhost/service.php?wsdl"
result="trace(event.result)"/>
<mx:Script>
<! [CDATA[
import mx.rpc.events.ResultEvent;
import mx.rpc.xml.SchemaTypeRegistry;
import mx.rpc.xml.QualifiedResourceManager;
import mx.rpc.xml.SimpleXMLDecoder;

private function init():void {
    var qname:QName = new QName("http://localhost/ns/ws",
"Plant");
    mx.rpc.xml.SchemaTypeRegistry.getInstance().registerClass(qname,
Plant);
}
]]>
```

```
 }

] ]>
</mx:Script>
</mx:Canvas>
```

## 第二十章 与浏览器通信 (Nigel)

很多时候，我们可能发现应用程序需要和加载它的浏览器进行通信。与浏览器的通信能够让你建立一个可以超越 Flex 应用本身的应用程序。你可以连接到已有的地址，通过 JavaScript 和其他应用程序通信，并且可以和浏览器的历史记录交互，作为开始。ExternalInterface 类让你能够调用加载 Flash 应用的浏览器，获取页面信息，并且调用 JavaScript 方法，同时也让 JavaScript 方法可以调用 Flash 应用程序。虽然已经存在一些其他集成浏览器和 Flash Player 的工具——Adobe Flex Ajax Bridge (FABridge) 和 Joe Berkovitz 的 UrlKit 之类的，但是本章旨在说明 Flex 核心框架里的功能。

### 20.1 节 连接到外部 URL

#### 20.1.1 问题

我想切换到一个独立的 URL。

#### 20.1.2 解决办法

使用 navigateToURL 方法将浏览器切换到新 URL。

#### 20.1.3 讨论

navigateToURL 方法让你可以在原窗口、新建窗口或者指定的窗口框架里切换到一个新 URL。这是 Flex 应用和浏览器通信中最普遍的一种方式。要在你的 Flex3 应用里调用 navigateToURL 函数，使用下面的方法：

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute">
    <mx:Script>
        <! [CDATA[
            import flash.net.navigateToURL;

            private function goToURL() : void
            {
                navigateToURL( new URLRequest( newUrl.text ),
                    target.selectedItem as String );
            }
        ]]>
    </mx:Script>

    <mx:TextInput
        id="newUrl"
        top="10" left="10" right="10"
```

```

    text="http://www.oreilly.com/" />

<mx:ComboBox
    id="target"
    top="40" left="10"
    dataProvider=" [ '_blank', '_self' ] " />

<mx:Button
    label="Go"
    left="10" top="70"
    click="goToURL()" />

</mx:Application>

```

该例中， 用户可以输入任何 URL 然后点击‘Go’按钮跳转过去。navigateToURL 方法的第一个参数是一个目标 URL 的 URLRequest 对象。第二个参数代表显示目标 URL 页面的窗口。它可以是浏览器中任何命名的窗口：\_blank 表示使用新建窗口，\_self 表示使用当前页面窗口，\_top 表示使用最顶层的框架容器，或者\_parent 表示使用当前框架容器的父亲容器。

## 20.2 节 使用 FlashVars

### 20.2.1 问题

我想从容器 HTML 页面传递参数给 Flex3 应用程序。

### 20.2.2 解决办法

使用 FlashVars 直接在包含你的 Flex3 SWF 的 HTML 的<embed>标签内添加参数。

### 20.2.3 讨论

你可以在包含你的 Flex 3 应用程序的 HTML 内直接添加数据，并且使用 FlashVars 变量在运行时轻易的读取这些数据。Flex 应用程序有两种办法可以获取这样的值。

你可以仿照下面例子里面的做法， 来修改 HTML 页面里用来嵌入你的 Flex 应用程序的 JavaScript 代码：

```

AC_FL_RunContent(
    "src", "${swf}",
    "width", "${width}",
    "height", "${height}",
    "align", "middle",
    "id", "${application}",

```

```

    "quality", "high",
    "bgcolor", "${bgcolor}",
    "name", "${application}",
    "allowScriptAccess", "sameDomain",
    "type", "application/x-shockwave-flash",
    "pluginspage", "http://www.adobe.com/go/getflashplayer",
    "FlashVars", "param1=one&param2=2&param3=3&param4=four"
);

```

如果不用 JavaScript 嵌入你的 Flex 3 编译的 SWF 文件，你也可以直接在 HTML 里面修改 <object> 和<embed>标签来实现：

```

<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
       id="${application}" width="${width}" height="${height}"

       codebase="http://fpdownload.macromedia.com/get/flashplayer/current/swflash.cab">
<param name="movie" value="${swf}.swf" />
<param name="quality" value="high" />
<param name="bgcolor" value="${bgcolor}" />
<param name="allowScriptAccess" value="sameDomain" />
<param name="FlashVars" value="param1=one&param2=2&param3=3&param4=four" />
<embed src="${swf}.swf" quality="high" bgcolor="${bgcolor}"
       width="${width}" height="${height}" name="${application}" align="middle"
       play="true"
       loop="false"
       quality="high"
       allowScriptAccess="sameDomain"
       type="application/x-shockwave-flash"
       pluginspage="http://www.adobe.com/go/getflashplayer"
       FlashVars="param1=one&param2=2&param3=3&param4=four"
     </embed>
</object>

```

在 Flex 应用程序里，你可以随时使用 Application.application.parameters 对象来访问 FlashVars 数据。下面这个 ActionScript 代码例子说明如何将四个 FlashVars 参数一一作为字符串访问，然后在 TextArea 的文本域里显示它们。

```

private function onCreationComplete() : void
{
    var parameters : Object = Application.application.parameters;
    var param1 : String = parameters.param1;
    var param2 : int = parseInt( parameters.param2 );
    var param3 : int = parseInt( parameters.param3 );
    var param4 : String = parameters.param4;
}

```

```
        output.text = "param1: " + param1 + "\n" +
                      "param2: " + param2 + "\n" +
                      "param3: " + param3 + "\n" +
                      "param4: " + param4;

    }
```

## 20.3 节 在 Flex 里面调用 JavaScript 函数方法

### 20.3.1 问题

我需要在 Flex 里调用 JavaScript 函数。

### 20.3.2 解决办法

在 AS 中使用 ExternalInterface 类调用 JavaScript 函数。

### 20.3.3 讨论

ExternalInterface 类封装了所有你在运行时可能使用到的与 JavaScript 通信的功能。你只需要简单的使用 ExternalInterface.call 方法来执行包含 Flex 应用程序的 HTML 页面里的 JavaScript 函数方法。

要在 ActionScript 里面调用简单 JavaScript 方法函数，使用下面的代码：

```
ExternalInterface.call( "simpleJSFunction" );
```

下面则是被调用的基础 JavaScript 函数。该 JavaScript 方法的名字作以字符串值形式传入到调用它的方法内，并且 JavaScript 的 Alert 窗口会出现在 Flex 应用程序上：

```
function simpleJSFunction()
{
    alert("myJavaScriptFunction invoked");
}
```

你也同样可以通过该技术使用函数参数来从 ActionScript 代码传递数据给 JavaScript。仿照下面代码，可以在调用 JavaScript 函数的同时传入参数：

```
ExternalInterface.call( "simpleJSFunctionWithParameters", "myParameter" );
```

使用该方法，可以从 ActionScript 代码传递多个参数，复杂值对象，或者简单参数给 JavaScript 代码。

在 JavaScript 里，也可以随意使用该方法，所有调用该方法的其他方法也需要传递这个参数。当被调用的时候，这个方法会在你的 Flex 应用之上弹出显示参数值为 myParameter 的 JavaScript 警报框。

```
function simpleJSFunctionWithParameters( parameter )
{
    alert( parameter);
}
```

通常，你可能发现需要在 Flex 应用程序里调用 JavaScript 方法来返回一个 JavaScript 代码程序里面的值。要返回这样的值，使用这个方法：

```
var result:String = ExternalInterface.call( "simpleJSFunctionWithReturn" );
```

你可以看到对应的 JavaScript 方法返回一个字符串值，这个值将存储在 ActionScript 类里面的结果字符串实例中。

```
function simpleJSFunctionWithReturn()
{
    return "this is a sample return value: " + Math.random();
}
```

## 20.4 节 在 JavaScript 中调用 ActionScript 方法函数

### 20.4.1 问题

我想在 HTML 里调用该 HTML 包含的 Flex 应用程序的 ActionScript 方法。

### 20.4.2 解决办法

使用 ExternalInterface 在 JavaScript 里设置对 Flex 的回调方法并且在 JavaScript 里调用 ActionScript 方法。

### 20.4.3 讨论

ExternalInterface 类不仅仅封装了运行时与 JavaScript 通信所需的功能，同时也包含了从 JavaScript 内调用 ActionScript 方法所有功能。

在 JavaScript 调用 ActionScript 方法之前，你需要为开放给 JavaScript 调用的 ActionScript 方法注册一个回调函数。回调函数通过 ActionScript 的 ExternalInterface 类来注册。回调函数为 JavaScript 方法提供一个对 ActionScript 方法的映射。

该例示范如何为这三个 ActionScript 方法函数注册回调方法。

```
private function registerCallbacks() : void
{
    ExternalInterface.addCallback( "function1", callback1 );
    ExternalInterface.addCallback( "function2", callback2 );
    ExternalInterface.addCallback( "function3", callback3 );
}
```

这些方法的对应 ActionScript 方法为：

```
private function callback1() : void
{
    Alert.show( "callback1 executed" );
}

private function callback2( parameter : * ) : void
{
    Alert.show( "callback2 executed: " + parameter.toString() );
}

private function callback3() : Number
{
    return Math.random()
}
```

注意到 callback1 是一个可调用的简单 ActionScript 方法。它没有参数也没有返回值。callback2 需要一个单参数，而 callback3 则返回一个随机生成的数值。

当你想要从 JavaScript 里面调用这些方法的时候，必须使用回调别名调用 JavaScript 方法。下面的 JavaScript 代码将展示如何调用已注册开放的 ActionScript 函数。

```
function invokeFlexFunctions()
{
    var swf = "mySwf";
    var container;
    //alert(navigator.appName); //不加这个测试不通过，不知为何,Firefox 均不通过
    if (navigator.appName.indexOf("Microsoft") >= 0)
    {
        container = document;
    }
```

```
else
{
    container = window;
}
//alert(container[swf]); //在 FireFox, 这个是 undefined
container[swf].function1();
container[swf].function2( "myParameter" );
var result = container[swf].function3();
alert( result );
}
```

当被嵌入到 HTML 页面时， swf 包含该 Flex 应用程序的名称(name 属性，在本节例子中为 mySwf)。该脚本做的第一件事是基于浏览器的类型获取到该 JavaScript DOM 的引用。在脚本有了正确的浏览器 DOM 后，它就可以根据注册回调的时候声明的公共开放映射关系来调用 Flex 方法。

通过调用 JavaScript 内 Flex 应用程序实例的 function1 回调函数来简单调用 ActionScript 的 callback1 方法。

```
container[swf].function1();
```

该方法被调用后， Flex 应用程序里会显示一个 alert 信息。

通过调用 function2 回调函数并传入一个值，即可简单调用 ActionScript 方法 callback2：

```
container[swf].function2( "myParameter" );
```

调用的时候，该例会在 Flex 应用程序里弹出一个 Alert 窗口来显示 JavaScript 调用的时候指定的参数值。

下面的例子展示如何从 Flex 返回值到 JavaScript 中。function3 回调函数调用 callback3 ActionScript 方法。该 ActionScript 方法返回一个随机生成的数值给 JavaScript。

当调用 callback3 时，Flex 生成一个随机数，并返回给 JavaScript。然后该值会显示在一个 JavaScript 的 alert 窗口中。例如：

```
var result = container[swf].function3();
alert( result );
```

## 20.5 节 经由 **BrowserManager** 改变 HTML 页面标题

### 20.5.1 问题

我想让 Flex 3 应用程序改变 HTML 页面标题

### 20.5.2 解决办法

使用 [BrowserManager](#) 类实例的 `setTitle` 方法来改变 HTML 页面标题。

### 20.5.3 讨论

Flex 3 的 `BrowserManager` 类能够轻易地用于和包含它的 HTML 页面的 HTML DOM 交互。其特性之一就在于能够改变包含它的 HTML 页面的标题。下面的 ActionScript 代码段即可设置标题：

```
private function changePageTitle( newTitle : String ) : void
{
    //get an instance of the browser manager
    var bm : IBrowserManager = BrowserManager.getInstance();

    //initialize the browser manager
    bm.init();

    //set the page title
    bm.setTitle( newTitle );
}
```

## 20.6 节 **BrowserManager** 解析 URL

### 20.6.1 问题

我想从浏览器的当前 URL 中读取并解析数据。

### 20.6.2 解决办法

使用 `BrowserManager` 和 `URLUtil` 类读取并解析当前页面的 URL。

### 20.6.3 讨论

下面的例子展示了如何通过使用 BrowserManager 和 URLUtil 类读取并解析当前页 URL，同时将解析结果写入一个 mx:TextArea 实例中。

URLUtil 类拥有可以帮助你解析当前 URL 内不同片断的方法。在 Flex 3 中使用深度链接的时候，URL 会分为两个部分：基部(base)和片段(fragment)。URL 基部(base)包含了#号左边的所有内容。片段(fragment)则包含了#号右边所有的内容。片段(fragment)用以传递参数给 Flex 应用程序或者用于历史管理器。适当创建的片段(fragment)可以由 URLUtil.stringToObject 方法将其包含的所有参数值解析到 ActionScript 对象中去，再打散成字符串值。URL 片段(fragment)的每个键-值对都应该用分号(;)分隔开。

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    creationComplete="parseURL()">

    <mx:Script>
        <![CDATA[
            import mx.utils.ObjectUtil;
            import mx.managers.IBrowserManager;
            import mx.managers.BrowserManager;
            import mx.utils.URLUtil;

            private function parseURL() : void
            {
                //get an instance of the browser manager
                var bm:IBrowserManager =
                    BrowserManager.getInstance();

                //initialize the browser manager
                bm.init();

                //output the url parameter values
                output.text += "Full URL:\n" + bm.url + "\n\n";
                output.text += "Base URL:\n" + bm.base + "\n\n";
                output.text += "URL Fragment:\n" + bm.fragment + "\n\n";

                //convert url parameters to an actionscript object
                //using URLUtil
                var o:Object = URLUtil.stringToObject(bm.fragment);
                output.text+="Object:\n"+ObjectUtil.toString(o)+ "\n\n";
                output.text += "name:\n" + o.name + "\n\n";
                output.text += "index:\n" + o.index + "\n\n";
                output.text += "productId:\n" + o.productId + "\n\n";
            }
        ]]>
    </mx:Script>

```

```

//parse URL using URLUtil
output.text += "URL Port:\n" +
    URLUtil.getPort( bm.url ) + "\n\n";
output.text += "URL Protocol:\n" +
    URLUtil.getProtocol( bm.url ) + "\n \n";
output.text += "URL Server:\n" +
    URLUtil.getServerName( bm.url ) + "\n\n";
output.text += "URL Server with Port:\n" +
    URLUtil.getServerNameWithPort( bm.url );
}

]]>
</mx:Script>

<mx:TextArea id="output" left="10" top="10" bottom="10" right="10"/>

</mx:Application>

```

如果前面的例子有如下的

URL<http://localhost:8501/flex3cookbook/main.html#name=Andrew;index=12345;productId=987>，结果将会是这样：

Full URL:

<http://localhost:8501/flex3cookbook/main.html#name=Andrew;index=12345;productId=987>

Base URL:

<http://localhost:8501/flex3cookbook/main.html>

URL Fragment:

[name=Andrew%20Trice;index=12345;productId=987654](#)

Object:

(Object)#0

index = 12345

name = "Andrew"

productId = 987

name:

Andrew

index:

12345

productId:  
987

URL Port:  
8501

URL Protocol:  
http

URL Server:  
localhost

URL Server with Port:  
localhost:8501

## 20.7 节 经由 **BrowserManager** 深度-链接到数据

### 20.7.1 问题

我需要从浏览器的 URL 传递数据给 Flex 控件，并且根据 Flex 应用程序里的数据更新浏览器 URL 的值，同时执行浏览器的前进或后退导航按钮。

### 20.7.2 解决办法

使用 `BrowserManager` 类和 `BrowserChangeEvent` 读取并写入数据到浏览器 URL。

### 20.7.3 讨论

无论是通过地址栏的输入框还是通过使用导航控件(前进和后退按钮)来改变浏览器 URL 的值，`BrowserManager` 实例都会广播一个 `BrowserChangeEvent.BROWSER_URL_CHANGE` 事件。无论何时，只要该类型的事件发生，你都可以简单地调用 `updateValues` 方法更新 Flex 控件的属性值。这让你可以轻易地链接、循环到你的输入值。

下面的例子展示如何从浏览器 URL 里读取数据并将读取的值放入 Flex `mx:TextInput` 的属性域里面去。当示例程序加载的时候，它会从当前 URL 里读取数据并且将 `firstName` 和 `lastName` 参数值写入文本框中去。无论是 `firstName` 还是 `lastName` 值的 `mx:TextInput` 属性域改变的时候，应用程序将在浏览器管理器调用 `setFragment` 方法，使用新的 `firstName` 和 `lastName` 参

数值更新浏览器的 URL。这让你能够复制粘贴 URL，从而轻易地直接连接到当前视图中去，同时也将每一个变化添加到浏览器历史记录中。

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    creationComplete="onCreationComplete() " >

    <mx:Script>
        <![CDATA[
            import mx.events.BrowserChangeEvent;
            import mx.managers.IBrowserManager;
            import mx.managers.BrowserManager;
            import mx.utils.URLUtil;

            private var bm:IBrowserManager

            private function onCreationComplete():void
            {
                //get an instance of the browser manager
                bm = BrowserManager.getInstance();

                //initialize the browser manager
                bm.init();

                //set initial values based on url parameters
                updateValues();

                //add event listeners to handle back/forward browser
                //buttons
                bm.addEventListener( BrowserChangeEvent.BROWSER_URL_CHANGE,
                    onURLChange );
            }

            private function updateValues():void
            {
                //update text box values based on url fragment
                var o:Object = URLUtil.stringToObject(bm.fragment);
                firstName.text = o.firstName;
                lastName.text = o.lastName;
            }

            private function updateURL():void
```

```

{
    //update URL fragment
    bm.setFragment( "firstName=" + firstName.text +
    ";lastName=" + lastName.text );
}

private function onURLChange( event :
    BrowserChangeEvent ):void
{
    //call update values based on change url
    updateValues();
}
]]>
</mx:Script>

<mx:TextInput x="10" y="10" id="firstName"
    change="updateURL() "/>
<mx:TextInput x="10" y="40" id="lastName"
    change="updateURL() "/>

</mx:Application>

```

## 20.8 节 经由 **BrowserManager** 深度-链接容器

### 20.8.1 问题

我需要根据 URL 参数控制 Flex 3 容器内的科室内容

### 20.8.2 解决办法

使用 **BrowserManager** 类和 **BrowserChangeEvent**s 控制可视性和跟踪可视 Flex 组件的历史轨迹。

### 20.8.3 讨论

在此情形下，你是用 URL 的 **fragment** 片断来控制并跟踪 Flex 应用里那些可视的容器和组件。当加载应用程序的时候，你就初始化 **BrowserManager** 类实例，该实例可以帮助你解析并处理浏览器 URL。**updateContainers** 方法(下面代码段中的方法)决定 **mx:TabNavigator** 实例内那些 tab 项为可视的。每当该 tab 导航器的可视 tab 项改变的时候，你可以通过使用下面的代码段来设置 URL fragment 片段中的 **selectedIndex** 属性。

```
bm.setFragment( "selectedIndex=" + tabNav.selectedIndex );
```

这就更新了浏览器的 URL 并且将变化添加到浏览器历史记录中。如果有人复制粘贴当前浏览器的 URL，则该用户可以直接连接到当前选中的 tab 导航器。

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    creationComplete="onCreationComplete()">

    <mx:Script>
        <![CDATA[
            import mx.events.BrowserChangeEvent;
            import mx.managers.IBrowserManager;
            import mx.managers.BrowserManager;
            import mx.utils.URLUtil;

            private var bm:IBrowserManager;

            private function onCreationComplete() : void
            {
                //get an instance of the browser manager
                bm = BrowserManager.getInstance();

                //initialize the browser manager
                bm.init();

                //set visible containers based on url parameters
                updateContainers();

                //add event listeners to handle back/forward browser
                //buttons

                bm.addEventListener( BrowserChangeEvent.BROWSER_URL_CHANGE,
                    onURLChange );
            }

            updateURL():

        }

        private function updateContainers():void
        {
            //convert url parameters to an actionscript object
            var o:Object = URLUtil.stringToObject(bm.fragment);
```

```

        //set the selected index
        if ( !isNaN(o.selectedIndex) )
        {
            var newIndex : Number = o.selectedIndex;
            if ( newIndex >= 0 && newIndex <
                tabNav.numChildren )
                tabNav.selectedIndex = newIndex;
        }
    }

private function onURLChange( event:BrowserChangeEvent
):void
{
    //call updateContainers when url value changes
    updateContainers();
}

private function updateURL():void
{
    bm.setFragment( "selectedIndex=" +
        tabNav.selectedIndex );
}

]]>
</mx:Script>

<mx:TabNavigator
    bottom="10" top="10" right="10" left="10"
    id="tabNav"
    historyManagementEnabled="false">

    <mx:Canvas label="Tab 0" show="updateURL()" >
        <mx:Label text="Tab 0 Contents" />
    </mx:Canvas>

    <mx:Canvas label="Tab 1" show="updateURL()" >
        <mx:Label text="Tab 1 Contents" />
    </mx:Canvas>

    <mx:Canvas label="Tab 2" show="updateURL()" >
        <mx:Label text="Tab 2 Contents" />
    </mx:Canvas>

</mx:TabNavigator>
```

```
</mx:Application>
```

你也可能已经注意到 TabNavigator 上的 historyManagementEnabled 参数已经设置为 false 了。这是因为你使用 BrowserManager 类的事件来判断浏览器 URL 是否已经改变，并据此更新 tab 项内容。每次可视 tab 项的变化都会导致浏览器历史记录的变化；用户可以通过使用浏览器上的前进后退按钮在该 tab 导航器的可视 tab 项之间来回切换。

## 20.9 节 实现自定义历史记录管理器

### 20.9.1 问题

我想把自定义组件上的动作或变化注册到浏览器的历史记录中，并让它们可以对浏览器的前进后退按钮进行导航。

### 20.9.2 解决办法

在 Flex 中通过实现 mx.managers.IHistoryManagerClient 接口来实现自定义的历史记录管理器。

### 20.9.3 讨论

为了实现此解决办法，历史记录管理器必须对你的 Flex 项目/工程是激活的。为了验证，我们进入 Flex Project Properties 对话框，选择 Flex Compiler 项，并验证 Enable Integration with Browser 复选框处于选中状态，这即表示你的历史记录管理器对你的 Flex 项目/工程是激活的。

下面的代码展示了如何为一个自定义的文本框实现 IHistoryManagerClient 接口。此控件的任何一个变化，都将注册到浏览器历史记录中。用户可以通过使用浏览器的前进后退按钮在这个 TextInput 控件的所有输入之间来回切换。

```
<mx:TextInput  
    xmlns:mx="http://www.adobe.com/2006/mxml"  
    text="Change Me!"  
    implements="mx.managers.IHistoryManagerClient"  
    creationComplete="mx.managers.HistoryManager.register(this);"  
    change="textChanged(event)">  
  
<mx:Script>  
    <! [CDATA[
```

```

import mx.managers.HistoryManager;

public function saveState():Object
{
    return {text:text};
}

public function loadState(state:Object):void
{
    var newState:String = state ? state.text : "";

    if (newState != text)
    {
        text = unescape( newState );
    }
}

private function textChanged(e:Event):void
{
    HistoryManager.save();
}
]]>
</mx:Script>

</mx:TextInput>

```

创建了此组件之后，你必须将该类的实例注册到历史记录管理器中。这步操作在 creationComplete 事件的处理方法中可以看到。

```
creationComplete="mx.managers.HistoryManager.register(this);"
```

实现 IHistoryManagerClient 接口的组件需要实现 saveState 和 loadState 方法。

每当自定义 TextInput 的值改变的时候，textChanged 方法即被调用，该方法即会调用历史记录管理器的保存记录的方法。当历史记录管理器保存完状态时，saveState 方法即被调用。

saveState 方法需要返回一个将会持久保存在浏览器历史记录中的对象。在这里，该方法返回一个含有 text 属性的对象，该 text 属性即为设置到 TextInput 组件的文本域的值。

当浏览器历史记录经由前进后退按钮改变的时候，loadState 方法即被调用。loadState 方法读取传进来的 State 对象的 text 属性值，然后根据这个 State 对象传入的值来设置 TextInput 控件的 text 属性域值。

你可以使用类似下面例子中的代码将该组件添加到你的 Flex 应用程序中去：

```
<mx:Application  
    xmlns:mx="http://www.adobe.com/2006/mxml"  
    layout="absolute"  
    xmlns:local="*">>  
  
    <local:MyTextInput />  
  
</mx:Application>
```

## 第二十一章. 开发策略 (**Spark, FandLR**)

对于关心工程项目快速实现的开发者来说，理解一个框架或者一个库，甚至一种编程语言，并不表示就知道如何用最优的方式使用它们创建应用程序。本章将针对使用 Flex 3, Flex Builder IDE, Flash CS3 IDE 创建组件和架构应用程序给出一些常用的技巧。主要包括——从帮助你开发一个组件来架构应用程序，到使用 Caringorm 框架。尽管有些是关于 Flex SDK 的(比如 Flex 组件工具包:Flex Component Kit)，但是绝大部分还是集中在如何整合 Flash IDE 和其他工具，帮助你开发组件，或者是如何使用其他框架(比如 PureMVC 和 Caringorm 框架)开发应用程序。

### 21.1节. 使用Flex组件工具包创建组件

#### 21.1.1. 问题

我想把Flash IDE中的创作的内容用到Flex程序中。

#### 21.1.2. 解决方案

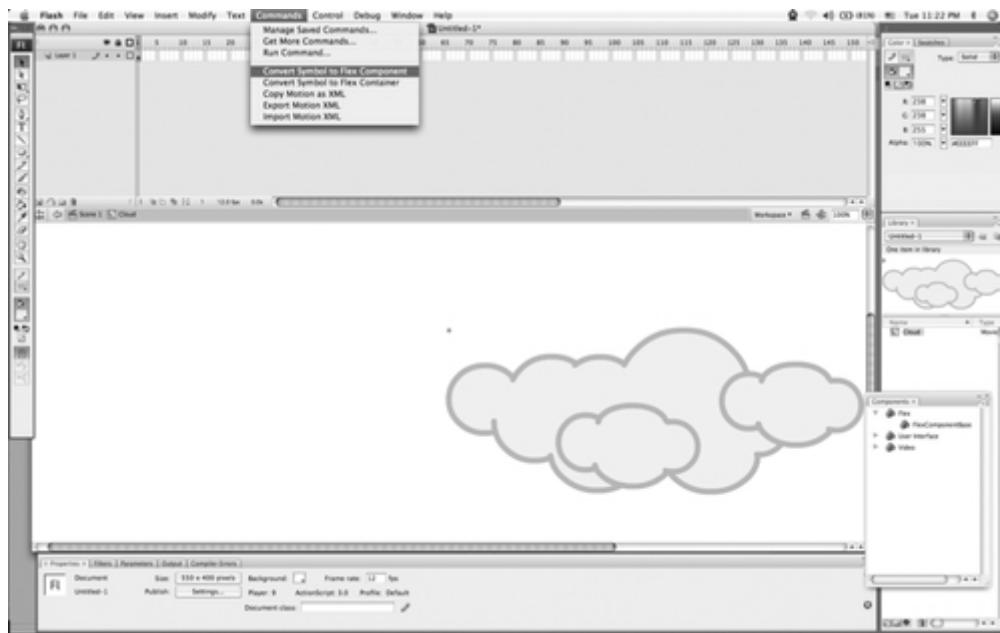
在Flash CS3 IDE中安装Flex组件工具包(Flex Component Kit)。然后创建一个元件，并使用[转换成Flex组件]命令把它转换成组件。最后把影片发布成SWC，添加到Flex项目中就可以了。

#### 21.1.3. 讨论

你可以使用SWFLoader类或者flash.loader类把Flash的内容加载到Flex应用程序中。把它们加载到应用程序后就可以控制它们了，但是使用Flex组件工具包会更容易些。Flex组件工具包简化了把Flash IDE中创建的内容加载到Flex应用程序中的流程。

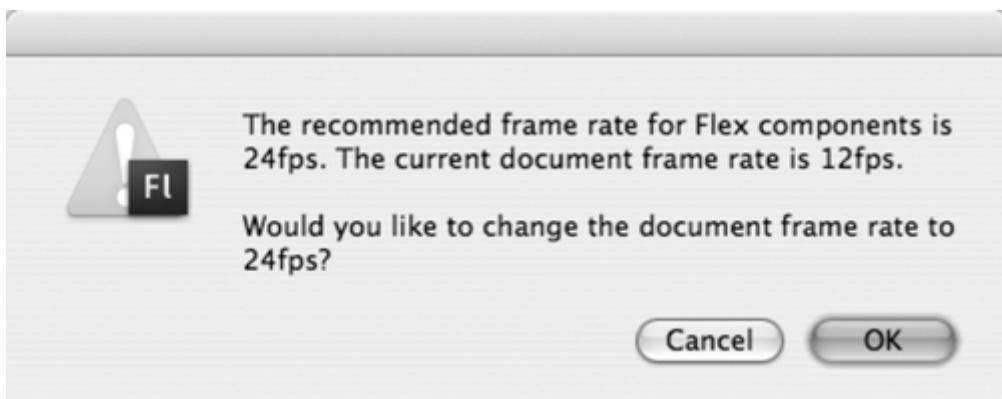
首先，从Adobe网站下载Flex组件工具包。在本书编写时，可以从<http://www.adobe.com/cfusion/entitlement/index.cfm?e=flex%5Fskins>处下载组件工具包。点击运行FlexComponentKit.mxp，就可以把Flex组件工具包安装到Flash IDE中了。要测试这个工具包，创建一个MovieClip元件，然后选择[元件转换成Flex组件]命令把元件转换成Flex中可以使用的UIMovieClip。比如在图21-1中，创建了一个名为Cloud的元件，并把它转换成Flex组件。

图21-1. 把元件转换成Flex组件



由于所有的Flex程序默认以每秒24帧的速度运行，所以Flash IDE会显示一个对话框询问是否要把元件转换成24fps。（图21-2）

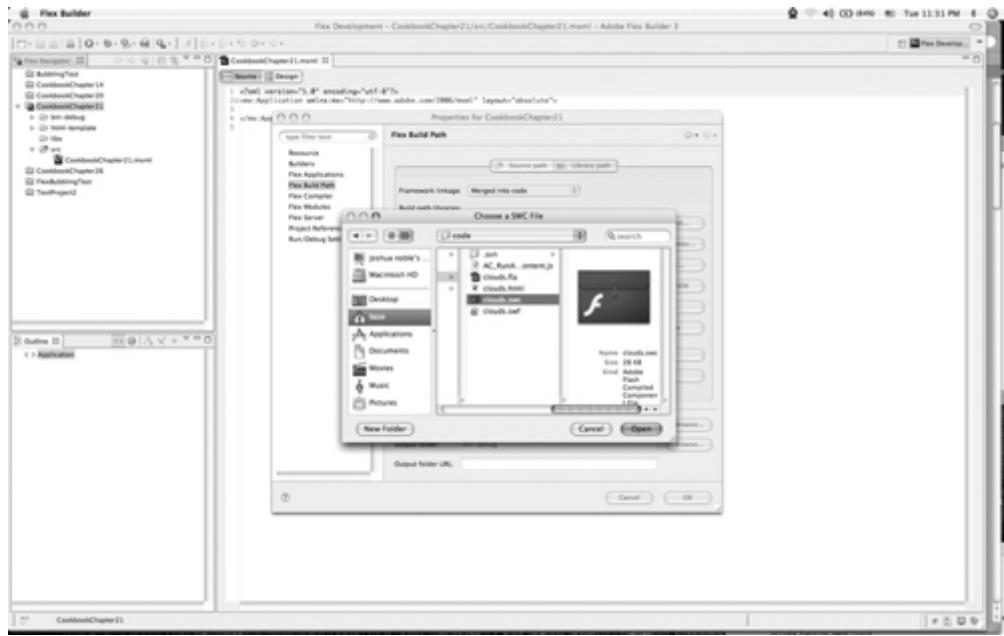
图21-2. 把文档的帧频调成跟应用程序一样



把组件文档的帧频设置成跟你的Flex应用程序的帧频一样。

当发布工程的时候，会生成一个SWC文件（默认是在跟FLA文件同一目录下——参照图21-3），你可以在Flex Build Path菜单中把它添加到Flex应用程序库路径中。

图21-3. 把SWC添加到构建路径中



现在，Flash中创建的组件就可以像下面这样用到Flex中：

代码：

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute" xmlns:local ="*" doubleClickEnabled="true">
    <local:Cloud height="200" width="400"/>
</mx:Application>
```

当这个组件第一次被导出时，Flash自动为他创建一个类。以后想要扩展这个组件时，你可能想在Cloud类上添加代码。但是自动生成的类是不能编辑的，所以，需要扩展UIMovieClip创建一个新的Cloud类，并保存为FLA文件。然后打开库中这个影片剪辑的链接属性，点击类输入框后面的验证标志（就是那个对号）。如果你把类和FLA文件放在一起，它会提示你类找到了。如果在基类框中有`mx.flash.UIMovieClip`，记得要把它清除，因为基类需要是对真实类的引用。把它替换为默认的`flash.display.MovieClip`。

```
package {
    import flash.display.MovieClip;
    import mx.flash.UIMovieClip;

    public class Cloud extends UIMovieClip {

        public function growCloud(value:int):void{
            this.scaleX *= value;
            this.scaleY *= value;
        }

        public function darkenCloud(value:int):void{
            trace(' darken ');
            //create custom filters and darken the cloud image
        }
    }
}
```

```
public function lightenCloud(value:int):void{
    trace(' lighten ');
    //create custom filters and lighten the cloud image
}
}
```

在Flash中重新编译这个SWC后，你可以在Flex中调用这些方法：

代码：

```
<local:Cloud height="200" width="400" id="cloud"
    click="{cloud.lightenCloud(1)}"
    doubleClick="{cloud.darkenCloud(1)}"/>
```

这个技术让你认识到使用Flex框架之后Flash IDE在设计方面可以发挥更多的用处，并使得设计者和开发者在一个项目中更加紧密的结合在一起。

## 21.2节.在Flash中使用ContainerMovieClip创建Flex容器

### 21.2.1. 问题

我想在Flash IDE中创建用于Flex的容器组件。

### 21.2.2. 解决方案

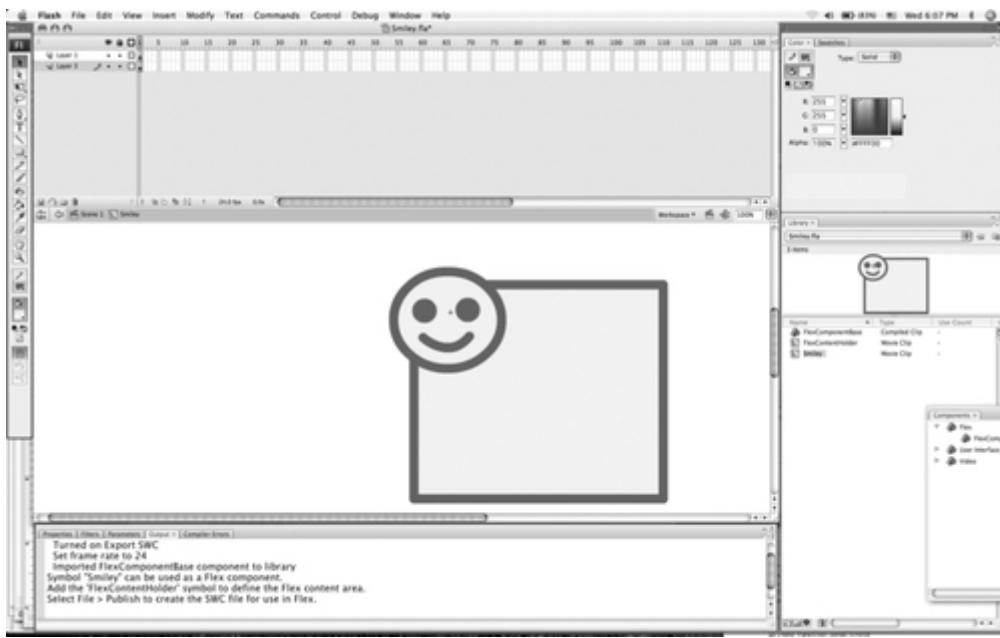
使用Flex组件工具包的[元件转换成Flex容器]的命令，或者更简单的，可以在Flash IDE中创建一个扩展自[ContainerMovieClip](#)的类，然后在你的应用程序中导入一个包含那个类的SWC。

### 21.2.3. 讨论

[ContainerMovieClip](#)的实例可以拥有子项；响应点击，鼠标移动，及其他事件；定义视图状态和过渡效果；像其他Flex组件一样使用各种效果。尽管在编译期或运行期只能往这样的容器中添加一个子项，但是你可以添加一个Flex容器，这样就可以添加多个项了。

要在Flash IDE中创建一个[ContainerMovieClip](#)，首先在Flash IDE中创建一个影片剪辑元件，并给它起个名字。[图21-4](#)中的元件叫做Smiley。然后使用Flex组件工具包的[元件转换成Flex容器]的命令把元件打包到[ContainerMovieClip](#)的实例中。

图21-4. 创建一个MovieClipContainer的实例



把元件转换成MovieClipContainer实例后，接下来就可以放置这个FlexContentHolder了。它决定了Flex内容可以被添加到这个容器的哪个区域，它是mx.flash.FlexContentHolder类的实例。当向它里面添加内容时，内容显示在它指定的区域中。注意，FlexContentHolder中只能添加一个子项，因此任何添加进来的子项都会自动填充它的整个区域。使用Flash IDE，你可以调整FlexContentHolder的尺寸和位置，就像图21-5那样：

图21-5. 调整FlexContentHolder的尺寸和位置



调整完之后，你可以把它发布成一个SWC文件，然后导入到Flex项目中。要想往MovieClipContainer中添加多个子项，需要往它里面添加一个容器，比如像下面的VBox：

```
<local:SmileyContainer id="smiley" y="100" x="100">
<mx:VBox>
<mx:Button label="button" width="100%"/>
```

```
<mx:Button label="button" width="100%"/>
<mx:Button label="button" width="100%"/>
<mx:Button label="button" width="100%"/>
</mx:VBox>
</local:SmileyContainer>
```

Button对象出现在VBox里， VBox的位置和尺寸由FlexContentHolder类指定(图21-6)。

图21-6. ContainerMovieClip的内容的区域由FlexContentHolder的位置和尺寸决定



## 21.3节. 导入Flash CS3的组件

### 21.3.1. 问题

我想在Flex中导入并使用Flash CS3的组件。

### 21.3.2. 解决方案

把所有你想使用的Flash CS3的组件放到一个SWC文件中，并导入这个SWC文件。

### 21.3.3. 讨论

有时你可能需要使用Flex Builder为设计者或更喜欢Flash CS3 IDE的人编写FLA使用的代码。有时需要创作一个既能在Flash CS3中又能在Flex Builder中编译的纯ActionScript应用程序。这时，在Flex Builder中使用Flash CS3的组件是个很好的选择。

要在Flash CS3中生成SWC文件，首先你必须创建一个新的Flash CS3 ActionScript3的FLA文件。把想要使用的组件拖到库里，选择文件→导出→导出影片，为SWF文件设置一个名称，并选择要保存的文件夹。在导出FlashPlayer对话框里，勾选[导出SWC]，然后点确定以确保设置生效。

当编译这个FLA文件时，会生成2个文件：一个SWF文件和一个SWC文件。这个SWC文件很有用，你可以把它添加到Flex Builder的应用程序的库路径中或者使用mxmlc时添加到编译选项library-path中。

## 21.4节. 认识Cairngorm小型结构

### 21.4.1. 问题

我想在我的应用程序中使用Cairngorm小型结构。

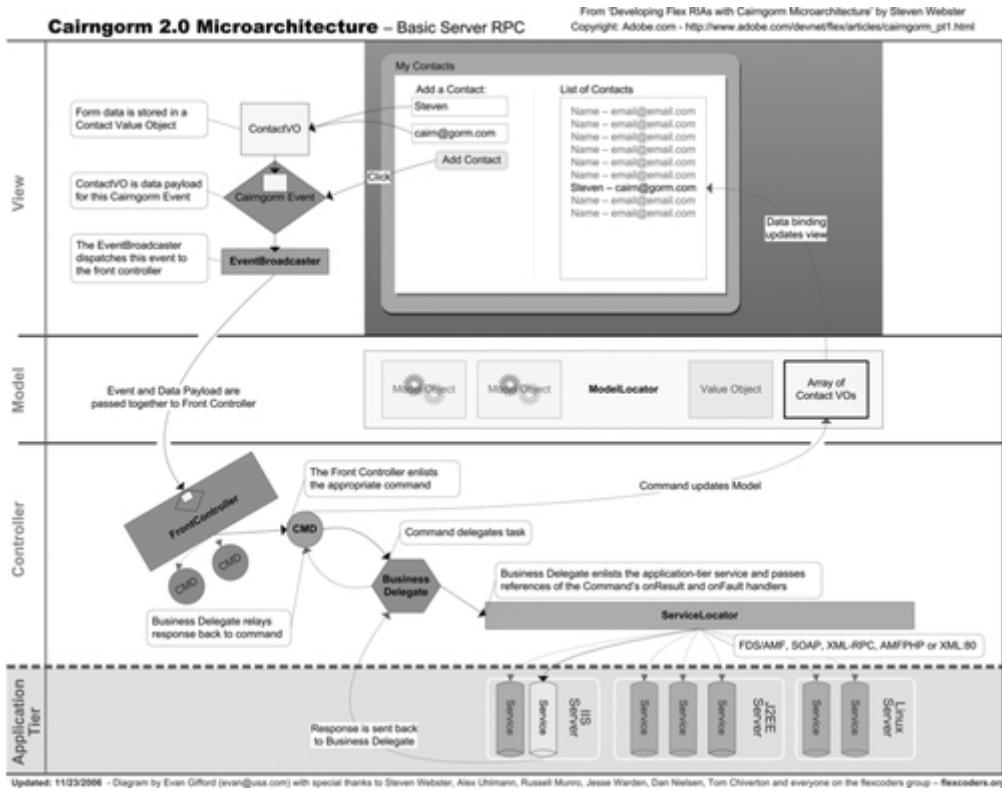
### 21.4.2. 解决方案

从<http://labs.adobe.com>处下载Cairngorm小型结构，然后把SWC文件安装到你的应用程序库中或者创建到源文件的链接。

### 21.4.3. 描述

Cairngorm小型结构是一个著名的、应用广泛的、构建Flex应用程序及把应用程序和服务器整合起来的模型-视图-控制器结构。它第一次为人所知是在Flex 1.5，并不断的用在开发多视图、一个复杂的模型、以及多服务调用的大型应用上。正是因为它如此的出名，所以对于多项目的开发者或者需要从其他开发者手中接管项目的开发者来说，熟悉这个结构是非常重要的。图21-7的UML图表简单的描述了Cairngorm的结构

图21-7. Cairngorm小型结构



正确使用Cairngorm小型结构能够使你创建非常封闭、相互之间不十分依赖的程序。对开发者来说就是即使经常更换视图也并不需要改变模型，更改模型时只需要对视图做极小的改动，这是因为各部分之间依赖性很小。Cairngorm应用程序的视图绑定到模型上。任何需要调用服务的视图上的用户动作或变化，视图都会派发一个CairngormEvent。因为视图绑定到模型类的属性上，因此同样不需要视图做任何事情。当服务层的数据更新后，模型会更新，绑定到属性上的视图也会更新。

EventBroadcaster通知Cairngorm程序的FrontController。FrontController中有许多命令。这些命令既向服务层或服务器发送数据，又处理返回的结果。FrontController使用CairngormEvent的名字作为键值来保存对这些命令的引用。每个CairngormEvent需要一个独立的事件类型，这样Cairngorm小型结构才能正常工作。

命令类知道调用哪些服务器上的方法、如何响应来自服务器的各种数据、以及数据保存在模型的哪个位置。其实，应用程序需要的大部分服务器端的通信工作都由命令类处理。命令类会调用事务代理类，然后事务代理类再调用ServiceLocator类中的方法。这些方法会被调用来接收处理来自服务器端的数据，像HTTPService对象、远程对象或者WebService对象。收到数据后，事务代理会告诉命令类：服务器返回数据了。在把它放到Cairngorm应用程序的模型中之前，Command会先对这些数据进行一些必要的处理。

由于视图绑定到模型上，当模型中的数据更新时，Flex的绑定机制会更新相应的视图。

接下来的几节会详细的介绍Cairngorm的几个部分。

## 21.5节. 创建Cairngorm视图、事件和模型

### 21.5.1. 问题

我需要创建一个Cairngorm视图，并把它绑定到模型的属性上。

### 21.5.2. 解决方案

创建一个实现了Cairngorm [ModelLocator](#)接口的模型类，往这个类里添加需要的数据属性，并把这些属性标记为可绑定的。然后把视图中所有数据驱动的控件绑定到这个模型的对应属性上。

### 21.5.3. 讨论

Cairngorm控制器的视图不需要扩展任何类，它可以是任意类型。视图最重要的特征是它们绑定到模型类的属性上，并且当需要从服务器加载数据或者需要向服务器发送数据时，它们会触发[CairngormEvent](#)。

接下来的几部分将创建一个Cairngorm应用程序，我们将按照书写的顺序而不是开发的顺序一步步显示它们。这个程序的视图包括：一个[mx:Button](#)，点击它会生成一个加载[recipes](#)的事件，该事件会调用服务并加载[recipes](#)；一个[DataGrid](#)，用来显示从服务器收到的结果：

代码：

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
<mx:Script>
<! [CDATA[
    import oreilly.cookbook.cairngorm.events.RecipeEvent;
    import oreilly.cookbook.cairngorm.model.Model;
```

在下面的方法中，会发送[RecipeEvent](#)，该事件使得[FrontController](#)接收来自服务器的数据，我们不需要知道服务器究竟是从数据库中读取数据还是从其他地方读取。更多关于[FrontController](#)的信息，请查看[21.8小节](#)。而现在，我们关心的是视图和视图发送的通知应用获取数据的事件。另外需要注意的是，[CairngormEvent](#)类有一个[dispatch](#)方法，用来把它们发送到Cairngorm [EventDispatcher](#)上。这跟通常情况下处理事件的方式不同，需要确保事件被[FrontController](#)处理而不是被其他的截获。

代码：

```
private function loadRecipes():void{
    var evt:RecipeEvent =
        new RecipeEvent(RecipeEvent.GET_RECIPE);
    evt.recipeKeywords= recipeKeywords.text.split(",");
    //Note that the CairngormEvent class dispatches itself
    //This notifies the Cairngorm EventBroadcaster that the
```

```

event
    //is being dispatched. This is used to ensure that the
    Cairngorm architecture
        //handles the event rather than the
    flash.events.EventDispatcher
    evt.dispatch();

}

]]>
</mx:Script>
<mx:VBox>
    <mx:TextInput id="recipeKeywords"/>
    <mx:Button label="Load Recipes" click="loadRecipes()"/>

```

Cairngorm 程序视图的另一个重要特征是它绑定到模型上。下面的例子中 `recipeArray` 是服务器返回的数据数组，它保存在模型中，视图就绑定到它上面。这样不需要把视图暴露给服务相关的逻辑，就能在数据加载到应用程序时更新视图。

```

<mx:DataGrid dataProvider="{Model.getInstance().recipeArray}">
    <mx:columns>
        <mx:DataGridColumn dataField="title"/>
        <mx:DataGridColumn dataField="difficulty"/>
        <mx:DataGridColumn dataField="pictureURL">
            <mx:itemRenderer>
                <mx:Component>
                    <mx:Image source="{data}"/>
                </mx:Component>
            </mx:itemRenderer>
        </mx:DataGridColumn>
        <mx:DataGridColumn dataField="preparationTime"/>
        <mx:DataGridColumn dataField="ingredients"/>
        <mx:DataGridColumn dataField="instructions"/>
    </mx:columns>
</mx:DataGrid>
</mx:VBox>
</mx:Canvas>

```

`RecipeEvent` 类有两个功能。一个是向服务器发送关键字以在数据库中搜索相关的内容。另一个功能是通知 `FrontController` 创建一个命令去接收来自服务器的数据，并把数据保存到模型中。Cairngorm 中自定义的动作必须拥有它自己的事件。这样才能保证所有需要服务器端通信的事件都用不同的事件类型，`FrontController` 也好利用它们决定应该调用哪个命令。Cairngorm 事件定义了需要发送到服务器的所有数据，另外还有用来区分不同事件类型的一个或多个字符串常量。尽管在 Flex 1.5 Cairngorm 0.99 版本中推荐每个需要 service call 的事件都应该有一个独立的事件类型，但是最近，开发者会在一个 `CairngormEvent` 类中使用多个事件类型。我个人比较喜欢在单个类中有多个事件类型，当然你怎么做由你决定。

代码：

```

package oreilly.cookbook.cairngorm.events
{
    import com.adobe.cairngorm.control.CairngormEvent;

    public class RecipeEvent extends CairngormEvent
    {

        public static const GET_RECIPE:String= "getRecipe";

        //this will be all the keywords for our recipe and is stored here so it
        //can be sent to the service for processing
        public var recipeKeywords:Array;

        public function RecipeEvent(type:String,
            bubbles:Boolean=false, cancelable:Boolean=false)
        {
            super(type, bubbles, cancelable);
        }

    }
}

```

最后，下面的部分显示的是实现了 ModelLocator 接口的模型。ModelLocator 接口并没有对实现它的类定义任何方法，但是要确保 Cairngorm 应用程序的模型类应该有一个可以用来识别它的类型：

```

package oreilly.cookbook.cairngorm.model
{
    import com.adobe.cairngorm.model.ModelLocator;
    import mx.collections.ArrayCollection;

    public class Model implements ModelLocator
    {
        // A way to get typed arrays
        [Bindable]
        [ArrayElementType ("oreilly.cookbook.cairngorm.vo.Recipe") ]
        public var recipeArray:ArrayCollection;

        public static function getInstance():Model
        {
            if(inst == null)
            {
                inst = new Model();
            }
        }
    }
}

```

```
    }
    return inst;
}
protected static var inst:Model;
```

## 21.6节. 创建Cairngorm命令和事务代理类

### 21.6.1. 问题

我们需要创建一个从FrontController中调用的命令类和一个跟这个命令通信的事务代理类。

### 21.6.2. 解决方案

创建一个实现了 `ICommand` 和 `mx.rpc.IResponder` 接口的类。你的事务代理类不需要扩展任何类，也不需要实现任何接口。它需要在构造函数中接收一个 `IResponder` 接口的实例，这样它就可以处理传递给它的 `IResponder` 实例的结果，或调用错误处理函数。这个 `IResponder` 实例就是调用事务逻辑的命令。

### 21.6.3. 讨论

在最新版本的Cairngorm小型结构之前， command类实现

`com.adobe.cairngom.business.Responder`类。在Cairngorm最新的版本(在写本书时是2.2.1)中，已经不推荐使用Responder类，而是像下面代码那样使用`mx.rpc.IResponder`接口。

Cairngorm的所有command类都实现了 [ICommand](#)接口，这个接口中定义了一个这样形式的 execute方法：

```
function execute(event:CairngormEvent) :void
```

下面是RecipeCommand类：

```
package oreilly.cookbook.cairngorm.commands
{
    import com.adobe.cairngorm.commands.ICommand;
    import com.adobe.cairngorm.control.CairngormEvent;

    import mx.controls.Alert;
    import mx.rpc.IResponder;
    import mx.rpc.events.FaultEvent;

    import oreilly.cookbook.cairngorm.business.BusinessDelegate;
```

```

import oreilly.cookbook.cairngorm.events.RecipeEvent;
import oreilly.cookbook.cairngorm.model.Model;

public class RecipeCommand implements ICommand, IResponder
{

```

当发送正常的CairngormEvent时，Cairngorm FrontController会调用command的这个execute方法。事件类型和command之间是一对一或多对一的关系，也就说一个command可以被多个事件或单个事件调用，但是一个CairngormEvent只触发一个command。Command必须处理特定操作需要的所有逻辑。下面的示例中的command的execute方法创建一个事务代理对象，并把它自己注册为处理服务调用结果或失败的IResponder:

```

public function execute(event:CairngormEvent) :void{
    if (event.type== RecipeEvent.GET_RECIPE) {

        var delegate:BusinessDelegate = new
            BusinessDelegate(this);
        delegate.getRecipes(event.recipeKeywords);
    }
}

```

如果服务器返回了正常的数据，事务代理会调用result方法。Command处理从服务器返回的结果，进行一些必要的操作，通常会设置模型中的值。在本文的例子中，模型的recipeArray被设置成返回的结果的值。因为这个程序的视图绑定到recipeArray上，recipeArray又标记为可绑定的，所以当recipeArray的值发生改变时，视图会同时更新。这样这个应用程序中的关系就很清晰；当更新或修改时，模型不需要做任何事情，视图也不需要知道服务器什么时候会返回结果以便更新显示。

```

public function result(arr:ArrayCollection) :void{
    Model.getInstance().recipeArray= arr;
}
public function fault(event:FaultEvent) :void{
    Alert.show(event.message);
}
}
}

```

基于Cairngorm的应用程序中的事务代理类中有mx.rpc.IResponder类的引用，就是既能处理返回结果也能处理失败的command。这个代理使用ServiceLocator调用特定的服务，当服务返回时，代理又把结果传递给IResponder处理。

```

package oreilly.cookbook.cairngorm.business
{
    import mx.rpc.IResponder;
}

```

```

import mx.rpc.events.FaultEvent;
import mx.rpc.events.ResultEvent;
import mx.rpc.remoting.RemoteObject;

public class BusinessDelegate
{
    private var responder:IResponder;

    public function BusinessDelegate(responder:IResponder) {
        this.responder= responder;
    }
}

```

这里这个由 `RecipeCommand` 调用的方法收到了要传给服务器的一个数组。通过使用 `ServiceLocator`，这个方法能调用正确的服务。服务器的响应或失败都会传递给创建了这个事务代理的 `command`。同样，这个分离使得代码更有弹性，更易于维护。

代码：

```

public function getRecipes(params:Array):void{
    // we access the RemoteObject stored in the ServiceLocator by using the Cairngorm ServiceLocator singleton
    // and passing a reference to the id of the RemoteObject that we wish to call
    var service:RemoteObject = com.adobe.cairngorm.business.ServiceLocator.getInstance().getRemoteObject("recipeService");
    var token:AsyncToken = service.send(params);
    var responder:mx.rpc.Responder= new mx.rpc.Responder(getRecipeResult, getRecipeFault);
    token.addResponder(responder);

}

```

当服务器返回结果时，`responder`，也就是 `command`，会处理数据，并设置模型中的相应属性。这个事务代理类的实例已经完成了它的工作，接下来它会被垃圾回收。

```

private function getRecipeResult(event:ResultEvent):void{
    responder.result(event.resultas ArrayCollection);
}

private function getRecipeFault(event:FaultEvent):void{
    responder.fault(event);
}
}

```

## 21.7节. 创建Cairngorm FrontController和ServiceLocator

### 21.7.1. 问题

我需要扩展Cairngorm [FrontController](#)类并创建自定义的事件类型。

### 21.7.2. 解决方案

创建一个扩展自Cairngorm [FrontController](#)的类。在构造函数中，使用方法[addCommand](#)把 [CairngormEvent](#)类型和[command](#)类关联起来。[ServiceLocator](#)应该扩展Cairngorm [ServiceLocator](#)类并要包含你的程序需要的所有服务。

### 21.7.3. 讨论

[FrontController](#)类在Cairngorm结构中扮演着重要角色：把[CairngormEvent](#)和[command](#)联系起来。它是通过方法[addCommand](#)完成这一工作的，看看[FrontController](#)基类中的这个方法：

代码：

```
public function addCommand(commandName : String, commandRef : Class, useWeakReference : Boolean = true) : void
{
    if(commands[commandName] != null)
        throw new CairngormError(
            CairngormMessageCodes.COMMAND_ALREADY_REGISTERED, commandName);

    commands[commandName] = commandRef;

    CairngormEventDispatcher.getInstance().addEventListener(commandName,
        executeCommand, false, 0, useWeakReference);
}
```

[Command](#)类是以事件名称作为键值存储起来的，当发送这个事件时，就会调用这个[command](#)。[CairngormEventDispatcher](#)监听这个事件，当广播这个事件时，[FrontController](#)使用两个[protected](#)方法新建一个[command](#)并运行它。方法[getCommand](#)只是在命令字典中查找[command](#)并创建这个[command](#)的实例：

代码：

```
protected function executeCommand(event : CairngormEvent) : void
{
    var commandToInitialise : Class = getCommand(event.type);
    var commandToExecute : ICommand =
        new commandToInitialise();
    commandToExecute.execute(event);
}
```

```

protected function getCommand(commandName : String) : Class
{
    var command : Class = commands[commandName];

    if(command == null)
        throw new CairngormError(
            CairngormMessageCodes.COMMAND_NOT_FOUND, commandName);

    return command;
}

```

然后会调用新command的execute方法，完成它所有的事务逻辑。下面是FrontController的一个实现：

```

package oreilly.cookbook.cairngorm.controller
{
    import com.adobe.cairngorm.control.FrontController;

    import oreilly.cookbook.cairngorm.commands.RecipeCommand;
    import oreilly.cookbook.cairngorm.events.RecipeEvent;

    public class Controller extends FrontController
    {
        public function Controller()
        {
            super();
        }
    }
}

```

Command是用CairngormEvent的名字注册的。在本例子中，RecipeEvent.GET\_RECIPE会创建并运行一个新的RecipeCommand：

代码：

```

// In this simple example there is only one Command that needs
to be added

// to the controller, but usually there are far more
Commands to be added

// when the controller is initialized.

addCommand(RecipeEvent.GET_RECIPE, RecipeCommand);
}

}

```

FrontController的其他功能隐藏在你将要扩展的类

`com.adobe.cairngorm.control.FrontController` 中。 `ServiceLocator` 类中保存了你的应用程序所需的所有要访问的服务的引用。它可能是 `HTTPService`, `WebService`, 或者 `RemoteObject` 组件。`ServiceLocator` 使用每个服务的 id 访问它们并调用它们的方法。下面例子展示了 `ServiceLocator` 是如何访问的:

```
ServiceLocator.getInstance().getRemoteObject("recipeService");
```

`ServiceLocator` 定义了 3 种方法用来访问服务:

`getRemoteObject()`

传递 `RemoteObject` 的 id，并返回一个 `RemoteObject`。

`getHTTPService()`

传递 `HTTPService` 的 id，并返回一个 `HTTPService` 对象。

`getWebService()`

传递 `WebService` 的 id，并返回一个 `WebService` 对象。

`ServiceLocator` 扩展自 Cairngorm `ServiceLocator` 类，并包含应用程序的所有服务对象:

代码:

```
<cairngorm:ServiceLocator xmlns="*"
    xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:cairngorm="com.adobe.cairngorm.business.*">
    <mx:RemoteObject id="recipeService"
        destination="http://localhost/service_location/"
        showBusyCursor="true"
        source="oreilly.cookbook.service.RecipeService"
        makeObjectsBindable="true">

        <mx:method name="getRecipe"/>

    </mx:RemoteObject>
</cairngorm:ServiceLocator>
```

在前面四节中，列出了 Cairngorm 小型结构的主要组件，并介绍了每个组件在程序中的作用。更多信息请访问：<http://www.adobe.com/go/cairngorm>.

## 21.8 节. 使用 Cairngorm 框架生成器生成应用程序骨架

### 21.8.1. 问题

我希望为我的Cairngorm程序生成一些文件。

### 21.8.2. 解决方案

使用Cairngorm框架生成器生成应用程序骨架。

### 21.8.3. 讨论

Cairngen代码生成器是由Eric Feminella开发的基于Apache Ant的项目，其作用是简化创建基于Cairngorm的应用程序初始文件的工作。可以从 <http://www.ericfeminella.com/blog/cairngen/> 处下载。需要安装Ant 1.5，Ant文件才能运行生成代码的build.xml。另外，你需要把ant-contrib-1.0b3.jar放到{ANT\_HOME}/lib目录下，或者使用ant-lib选项指定你下载的源文件中ant-contrib-1.0b3.jar的位置：

```
ant -lib {Cairngen}/build/libs/ant-contrib-1.0b3.jar
```

你应该在下载了文件后就在Cairngen文件的根目录下的project.properties文件设置这个参数。Project.properties包含如下设置：

*project.name*

生成的Cairngorm类属于的项目的名称。

*root.dir*

Flex项目目录（比如，C:/workbench/efeminella/CairngenExample）。在Windows平台，需要把反斜线（\）替换成（/）。

*com.dir*

项目域后缀（比如com,org,net或者edu）。

*domain.dir*

项目域目录（比如，ericfeminella）。

*project.dir*

要保存Cairngorm结构的项目路径名。

*cairngorm.version*

指定要使用的模板的版本，有效值是2.0,2.1和2.2.1。

*sequence.name*

指定要生成的事件命令和事务代理（可选）的名字。

*vo.name*

指定生成的值对象的名字。

*vo.remoteClass*

如果设置成true, create-value-object会生成一个远程类值对象。

*overwrite.files*

如果设为true, 那么指定的已存在文件会被覆盖; 默认值是false。

*prompt.on.delete*

如果把它设为true, 需要删除文件夹时会提示用户; 默认值是true。

*reset.properties*

如果设置为true, `sequence.name`和`vo.name`属性的值会在它们生成后设置为空字符串; 默认值是true。

*log.output*

如果是true, 控制台会把输出写入到日志文件中; 默认值是true。

*namespace*

项目类的名字空间, (比如, com.domain.project)。

*project-uri*

脚本要创建的项目目录。

项目的所有属性都设置完成后, 就可以从命令行运行Ant了:

`ant -lib ant-contrib-1.0b3.jar`

如果你把`ant-contrib-1.0b3.jar`放在了`{ANT_HOME}/lib`目录下, 那么可以这样:

`ant`

相应的文件会生成在`project.properties`文件中`root.dir`属性指定的路径中。

## 21.9节. 学习常用的提高性能的技巧

### 21.9.1. 问题

我要确保我的程序运行的既快又好。

### 21.9.2. 解决方案

使用下面的几个改善性能的技巧，如果你使用Flex Builder，那么可以使用调试窗口查看你的程序中关键的几步创建对象和内存回收的情况。

### 21.9.3. 讨论

有时细小的调整会造成很大的不同。下面是一小部分能够改善你的程序性能的做法：

- 如果不知道一个对象的类型，使用`as`操作符而不是使用`try...catch`，后者更慢一些：

```
var iface:IMyInterface = (obj as IMyInterface);
```

- 稀疏数组访问起来比较慢，所以把空的项填上`null`会提高速度。从数组中找不到一个值的操作非常慢，比找到一个值要慢20倍。
- 当加整数时，Flash Player会把整数转化成数字，加完后再转换回来。因此当执行数学计算时，应当尽量使用数字，只是在最后再转换成整数。
- 局部变量的访问速度比较快，因此频繁访问的变量尽量设置成局部变量。它们会被存储在栈上，访问起来非常快。
- 如果可能，在创建显示组件时尽量使用延迟的实例化以避免让用户等待。
- 数据绑定会耗费内存并减慢程序启动时间。如果不需要绑定，比如一个值只会更新一两次，那么最好直接设置它的值。
- 不用使用容器类作为`List`或`DataGrid`的`itemRenderer`；而应该使用`UIComponent`。
- 如果你使用组件或对象监听了某个事件，并希望以后被垃圾回收，记得要使用`removeEventListener`移除事件监听。
- 对于色彩丰富的`UIComponent`或者包含位图数据的对象使用`cacheAsBitmap`。这样Flash Player就能重复使用创建的位图来显示，直到需要重绘。注意，想要改变缓存的位图非常困难，并且在缩放时可能会失真。
- 运行期调用`setStyle`代价很大，所以要减少类似的操作。但是在`DisplayObject`对象添加到显示列表之前调用`setStyle`的开支会小一些。

## 21.10节. 在组件中创建元数据

### 21.10.1. 问题

我希望为组件创建元数据属性，这样就能用在为特定程序开发的组件上，或者保存类的外部属性。

### 21.10.2. 解决方案

设置`-keep-metadata`项，这样编译器会保存Flash编译器生成的元数据，并在运行时查看那个数据。

### 21.10.3. 讨论

运行期搜集元数据的功能使得你能够创建自己的标签，其他开发者可以使用这些标签标记类的特定属性，这很像Flex框架中的[Bindable]。它也能使你查看一个类中的任意元属性，比如，像下面这样声明一个元属性：

```
[CustomMetaProp(metaPropOne="foo", metaPropTwo="bar")]
```

就可以在运行时查看它。首先在要跟那个属性一起编译的自定义类中声明一个自定义属性：

代码：

```
package oreilly.cookbook
{
    import flash.utils.*;
    public class CustomMetaData
    {
        public function CustomMetaData()
        {
        }

        public function getdescription():Object{
            return describeType(this);
        }

        [CustomMetaProp(metaPropOne="foo", metaPropTwo="bar")]
        public function get propertyThatContainsMetaData():Object{
            return null;
        }
        public function set
            propertyThatContainsMetaData(value:Object):void{
        }
    }
}
```

真正的逻辑发生在`describeType`方法中，该方法会把存储在播放器字节码中一个类的所有信息以XML的格式输出。如果元数据和元数据的信息都设置正确了，那么不仅会返回类信息，对象的所有元数据信息也会返回。要确保元数据保存在SWF中，创建一个包含如下内容的metaconf.xml文件：

```
<?xml version="1.0"?>
<flex-config xmlns="http://www.adobe.com/2006/flex-config">
    <compiler>
```

```

<keep-as3-metadata>
  <name>CustomMetaProp</name>
</keep-as3-metadata>
<keep-generated-actionscript>
  true
</keep-generated-actionscript>
<source-path>
  <path-element>. </path-element>
</source-path>
</compiler>
</flex-config>

```

这个XML文件描述了应该保存在字节码中的元数据属性；在本例中，所有的CustomMetaProp元数据都会被保存。在编译时，像下面这样把metaconf.xml传递给编译器：

`-load-config+="meta-config.xml"`

编译完成后，就可以像下面这样访问CustomMetaData类的元数据了：

代码：

```

<mx:Application  xmlns:mx="http://www.adobe.com/2006/mxml"
  layout="vertical">
<mx:Script>
<! [CDATA[
  import oreilly.cookbook.CustomMetaData;
  private var metaData:CustomMetaData =
    new CustomMetaData();

  private function displayDescription():void
  {
    var xmlData:XML= metaData.description;
    trace(String(xmlData.accessor.(@name==
"propertyThatContainsMetaData")));
  }

]]>
</mx:Script>
<mx:Button label="Describe" click="displayDescription()"/>
</mx:Application>

```

因为`describeType`返回的X是ML，可以使用E4X的语法访问名为`propertyThatConatinsMetaData`的包含了元数据的属性，将会输出如下结果：

代码：

```
<accessor name="propertyThatContainsMetaData" access="readwrite"
  type="Object" declaredBy="oreilly.cookbook::CustomMetaData">
  <metadata name="CustomMetaProp">
    <arg key="metaPropOne" value="foo"/>
    <arg key="metaPropTwo" value="bar"/>
  </metadata>
</accessor>
```

## 第二十二章. 模块(**Modules**)和运行时共享库(**RSLs**)(常青)

当开发富互联网应用程序时，最终文件的大小和下载时间是必须要考虑的。Flex Framework 提供了多种选择把应用程序代码分散到多个SWF文件中，以提高用户体验。

Runtime shared libraries (RSLs) 是可被下载并缓存到客户端的一些文件。当RSL被下载并留在客户端后，其他应用程序就可以访问缓存的RSL资源了。应用程序载入两种RSLs：未签名和签名的。未签名的RSLs，比如标准的和跨域的SWF文件，存储在浏览器缓存里。签名的RSLs，这是经过Adobe签名过的，扩展名为.swz，存储在Flash Player 缓存中。

正如其名，RSL被称作动态链接库，在运行时被载入。静态链接库是SWC文件，通过编译器的library-path和include-libraries编译进应用程序。采用静态链接的应用程序SWF会产生比较大的文件以及更长的下载时间。 使用RSLs的应用程序载入时间短且文件比较小，而且提高了内存使用效率，只是在开始需要花点时间下载RSLs。RSLs的强大体现在多个应用程序共享公共代码时，因为RSLs只需要被下载一次，多个应用程序动态链接到相同的RSL，访问其中已经缓存在客户端的资源。虽然RSLs的巨大优点是缓存到客户端，但是它并不考虑库中的哪些类被真正用到，就把整个RSL库都下载来。

Modules和RSL类似，只不过提供了另外一个方法分离应用程序代码到不同的swf文件上以便减少下载时间和文件大小。使用Modules的好处是，它不像RSLs，主应用程序开始时不需马上下载modules。应用程序会根据需求载入和卸载modules。使用modules的开发进程中还有个好处：由于每个modules独立于其他应用程序modules，当需要改变一个modules时，你只需要重编译这个modules而不是整个应用程序。

你可以使用ActionScript和MXML创建模块化应用程序。基于Flex的模块使用<mx:Module>根标签，而基于ActionScript的模块需扩展自mx.modules.Module或mx.modules.ModuleBase。Module类类似于Applications。使用MXML编译器工具(mxmlc)编译模块，生成可被动态载入和卸载的SWF文件，你可以通过<mx:ModuleLoader> 和 mx.modules.ModuleLoader 和 mx.modules.ModuleManager类管理载入和卸载的模块。

创建模块化应用程序是提高Flex框架性能的重要方面，提供更多能力控制下载时间和文件大小，使用RSLs和modules，你可以根据哪些可被独立载入，哪些可被其他程序使用而分离应用程序代码，两项技术各有优点，这一章将介绍如何在开发进程中使用它们。

### 22.1节. 创建一个运行时共享库

#### 22.1.1. 问题

我想创建一个可被下载和缓存的运行时共享库(RSL)。

#### 22.1.2. 解决办法

使用自定义类，组件和其他资源创建一个库，编译为SWC文件，然后解压出SWF文件中的library.swf 文件，引入到你的应用程序部署目录，作为一个RSL使用。

#### 22.1.3. 讨论

SWC文件是一种压缩文件格式，里面包含一个library.swf 文件一个catalog.xml文件。这个库是一组编译进swf文件的资源，而catalog.xml是描述依赖关系。要使用这个库作为RSL，你需要从生成的SWF文件中解压缩出library.swf，引入到你的应用程序目录中。

虽然在应用程序同一个域中需要有个库才能在运行时访问它，但是在编译应用程序时这个SWF库文件是不需要的，而SWC文件在编译时是必须的，因为它使用了动态链接。

下面的例子是一个MXML组件，被打包进SWC文件，这个类被引入到生成的SWF库文件中，而类实例将显示在应用程序的显示列表中：

Code View:

```
<mx:Canvas
    xmlns:mx="http://www.adobe.com/2006/mxml"
    width="300" height="200">

    <mx:Metadata>
        [Event(name="submit", type="flash.events.Event")]
    </mx:Metadata>

    <mx:Script>
        <! [CDATA[

            public static const SUBMIT:String = "submit";
            private function clickHandler():void
            {
                dispatchEvent( new Event( CustomEntryForm.SUBMIT ) );
            }
            public function get firstName():String
            {
                return firstNameField.text;
            }
            public function get lastName():String
            {
                return lastNameField.text;
            }

        ]]>
    </mx:Script>

    <mx:Form>
        <mx:FormItem label="First Name:>
            <mx:TextInput id="firstNameField" />
        </mx:FormItem>
        <mx:FormItem label="Last Name:>
            <mx:TextInput id="lastNameField" />
    </mx:Form>

```

```

</mx:FormItem>
<mx:Button label="submit" click="clickHandler();"/>
</mx:Form>

</mx:Canvas>

```

这是个简单的组件，允许用户输入信息和发出一个submit事件。要打包这个类为SWC文件。你需要调用compc工具，设置source-path和include-classes参数，如下面的命令生成一个CustomLibrary.swc：

Code View:

```
compc -source-path . -include-classes com.oreilly.flexcookbook.CustomButtonForm -output CustomLibrary.swc
```

这个MXML组件CustomEntryForm.mxml存放在当前开发目录下的com/oreilly/flexcookbook子目录下，source path输入值为当前目录(一个点号)。你可以引入多个类文件，之间用空格分开。

打包进生成的SWF文件的library.swf文件被用来作为RSL，要解压出这个文件，你可以使用标准的unzip工具。当编译应用程序时SWC文件作为动态链接，你可以根据需要重命名解压出来的library文件。在这个例子中，被命名为CustomLibrary.swf。

下面的例子使用载入RSL中的CustomEntryForm组件：

Code View:

```

<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:flexcookbook="com.oreilly.flexcookbook.*"
    layout="vertical">

    <mx:Script>
        <! [CDATA[

            private function onFormSubmit():void
            {
                greetingField.text = "Hello " + entryForm.firstName
                    + " " + entryForm.lastName;
            }
        ]]>
    </mx:Script>
    <mx:Panel title="Enter Name:" width="400" height="400">
        <flexcookbook:CustomEntryForm id="entryForm"
            submit="onFormSubmit()"/>
        <mx:HRule width="100%"/>
    </mx:Panel>

```

```
<mx:Label id="greetingField" />  
</mx:Panel>  
  
</mx:Application>
```

使用来自RSL中的类引用和MXML组件资源就像引用静态链接库和本地开发目录中的类文件一样。在这个例子中在<mx:Application>标签中申明了flexcookbook名称空间，被用来添加CustomEntryForm组件到显示列表中。

要编译使用了RSL的应用程序，调用mxmlc工具时要使用external-library和runtime-shared-libraries参数：

Code View:

```
> mxmlc –external-library=CustomLibrary.swc –runtime-shared-libraries=CustomLibrary.swf  
RSLEExample.mxml
```

这个命令中，CustomLibrary.swc 被用来做编译时期链接检测，而RSL库被生成的SWF应用程序所调用。在开发期间，你可以把RSLEExample.swf和CustomLibrary.swf 放到服务器的同一个目录下，在启动时，应用程序载入CustomLibrary RSL，当代码可用时会出现一个表单供用户输入信息。

## 22.2节. 使用跨域的RSL

### 22.2.1. 问题

我想把RSLs存放在服务器的不同地方以便在不同域的应用程序都能访问到。

### 22.2.2. 解决办法

Compc工具创建RSL时加上compute-digest参数后，RSL摘要信息会在编译时期链接到RSL时存储到应用程序中。然后创建跨域的授权文件引入mxmlc工具的runtime-shared-library-paths选项指定的RSLs位置。

### 22.2.3. 讨论

一个RSL摘要是一个散列码，被用来确认RSL来自于信任方，已被Flash Player载入。当用compute-digest选项设置为true时创建RSL后，这个摘要会被写进SWC存档的catalog.xml文件中。当你编译时链接一个跨越的RSL到应用程序时，这个摘要会被存储到应用程序SWF文件中用来验证所需RSL的合法性。

下面的命令生成一个SWC文件命名为CustomLibrary.swc：

Code View:

```
> compc –source-path . –include-classes com.oreilly.flexcookbook.CustomEntryForm –output
```

```
CustomLibrary.swc --compute-digest=true
```

compute-digest选项的默认值为true，当编译库时你不必引入它来创建摘要。当通过MXML编译器的runtime-shared-library-paths选项链接跨域RSLs时才需要摘要。

注意：

在上一节中，你看到一个标准的RSL和应用程序放在同一个域中，标准RSLs也可以使用摘要，但是这个摘要不是必须的。

使用上面的命令生成的SWC文件是一个压缩文件，包含一个library.swf文件和一个catalog.xml文件。使用标准的unzip工具解压出这两个文件。这个library是一组编译进SWF将作为RSL被使用的资源。而catalog则使用compute-digest选项生成对library的描述信息。下面显示一个catalog文件的RSL摘要记录：

Code View:

```
<digests>
  <digest type="SHA-256" signed="false"
    value="2630d7061c913b4cea8ef65240fb295b2797bf73a0db96ceec5c319e2c00f8a5"  />
</digests>
```

Value值是编译器使用SHA-256算法生成的散列码。当你编译一个连接到RSL的应用程序时，这个摘要的value会被存储到应用程序中，被用来确认来自服务器的RSL的合法性。

除了摘要确认RSL是来自于信任的资源外，跨域授权文件也是库所在服务器所必需的。跨域授权文件是一个XML文件，列举出可访问数据的已授权远程服务器。要让其他应用程序能找到其他域中的RSL，在crossdomain.xml文件中用<allow-access-from>元素列举出域：

```
<?xml version="1.0"?>
<cross-domain-policy>
  <allow-access-from domain="*.mydomain.com" />
  <allow-access-from domain="*.myotherdomain.com" />
</cross-domain-policy>
```

上面的跨域授权文件允许任何来自<http://mydomain.com> 和 <http://myotherdomain.com> 的SWF访问服务器数据，包括跨域RSLs。

要允许上面列举出的域中任何SWF访问目标服务器上的数据，你要把跨域授权文件放到服务器的根目录。虽然这是一个可行的方法，但是你可能希望应用程序访问目标服务器上的RSL时有更多控制。通过MXML编译器的runtime-shared-library-path选项，你可以指定跨域授权文件的具体位置。

要编译动态链接到上面生成的CustomLibrary.swf 跨域RSL的应用程序，调用mxmlc工具，使用runtime-shared-library-path参数和目标服务器上的RSL完整URL路径和跨域授权文件：

```
> mxmcl RSLExample.mxml -runtime-shared-library-path=
    CustomLibrary.swc,
    http://www.mytargetdomain.com/libraries/CustomLibrary.swf,
    http://www.mytargetdomain.com/libraries/crossdomain.xml
```

runtime-shared-library-path选项使用了逗号分割符，分割每个参数值，分别是SWC文件位置，远程服务器RSL URL路径和跨域授权文件路径，提供权限给其他域程序载入RSL。作为RSL的SWF文件在编译时不会被检测，但是其URL会被存储在应用程序中，在运行期间会被检测。

使用跨域RSLs的好处除了减小文件大小和下载时间外，还可以使其他域的程序也能访问到数据，因为很多应用程序都是部署在不同的服务器上，且要使用RSL，你可以更新跨域授权文件中的许可服务器列表。

## 22.3节. 使用Flex Framework作为RSL

### 22.3.1. 问题

我想减少链接到Flex框架RSL的应用程序文件大小和下载时间。

### 22.3.2. 解决办法

使用framework.swc文件编译应用程序，在Flex 3 SDK安装目录下可找到这个文件。

### 22.3.3. 讨论

如果你熟悉用Flex Framework开发过应用程序的话，你可能注意到生成的文件很大，接着就是很长的下载时间。甚至是像下面的例子那样简单的程序，当编译时，结果生成的SWF文件和这简单的代码相比却有着出乎意料的文件大小。：

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">

    <mx:Label text="Hello World" />
</mx:Application>
```

这个程序没有多少代码，只是添加一个标签组件显示文本 Hello World到显示列表。当用mxmcl工具编译程序后，生成的SWF文件大小却接近149KB。这是因为它把Flex组件和库都编译进了应用程序中。这是合理的，毕竟对于运行于Flash Player的独立SWF需要所有的资源提高运行效率，但是，这样一个简单的程序如果在浏览器上载入运行的话，这个文件大小就是一个非常高的代价了。因此有了运行时共享库（RSL）的好处不仅仅解决了文件大小和下载时间，而且还允许多个应用程序访问同一个库资源。因此把整个Flex Framework作为RSL存储在客户端，编译应用程序都动态链接到它是一个非常好的解决方案。

这一特性是Flex 3 SDK和Flash Player release 3 (Flash Player 9.0.60之后)才提供的，可以把Flex

Framework从应用程序代码中分离出来。Flex Framework放在Flex 3 SDK的/frameworks/libs目录下：就是framework.swc文件。而/frameworks/rsl目录下的是经过签名的Flex Framework RSL。在写这本书时，这个文件已被命名为framework\_3.0.189825.swz。

为签名的RSLs，比如上一节中你自己创建的，会保存在浏览器缓存中，可被同一个域中的任何程序或用跨域授权文件指定的任何域程序所访问。签名的RSLs（扩展名.swz）会存储在Flash Player缓存中。Flash Player缓存中签名的RSLs可在配置管理器中删除。只有Adobe才可对RSLs进行签名存储到Flash Player，以此提高安全防止第三方工具注入攻击及执行代码。要编译动态链接到前面的Flex Framework RSL的应用程序，你需要使用mxmlc工具的runtime-shared-library-path参数：

```
> mxmlc HelloWorldApplication.mxml -target-player=9.0.60  
    -runtime-shared-library-path=  
        /<sdk-installation>/frameworks/framework.swc,  
        framework_3.0.1.189825.swz
```

runtime-shared-library-path选项第一个参数值表明framework.swc文件的具体位置，第二个参数（直接用逗号分开）是签名的Flex Framework RSL名称。在编写此书时framework\_3.0.1.189825.swz文件在rsls目录里。

把SWZ文件和编译生成的应用程序SWF放在服务器的同一个域里，当应用程序首次被下载时签名的RSL会被载入到客户端的Flash Player缓存中。任何使用runtime-shared-library-path编译参数编译的应用程序都可以调用这个缓存文件。也就说其他开发者使用同一个编译参数也可以使用这个缓存的RSL，这是很有好处的。

不使用RSL编译生成的SWF文件大小接近149KB，而动态链接到Flex Framework RSL的应用程序仅仅49KB。.

## 22.4节. 优化RSL

### 22.4.1. 问题

我想减小RSL文件的大小。

### 22.4.2. 解决办法

使用optimizer命令行工具删除SWC文件中的调试代码和不必要的元数据。

### 22.4.3. 讨论

默认情况下，生成的SWC文件中包含library.swf文件包含调试代码和元数据代码。这对于在远程服务器作为RSL是没有什么用处了。要想创建一个优化过的RSL，首先通过compc工具创建SWC压缩文件，然后解压出library文件。library.swf文件将作为RSL。这时你就可以使用optimizer命令行工具重新编译这个RSL。

SWC文件中的library的大小依赖于编译时引入的类库。为了演示optimizer工具的强大效果，

我们创建一个MXML组件，保存为 MyCustomComponent.mxml：

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml">  
    <mx:TextArea text="Lorem ipsum dolor sit amet" />  
</mx:Canvas>
```

这个简单组件将显示一个<mx:TextArea>。

使用下面的命令生成SWC文件并命名为library.swc：

Code View:

```
> compc -source-path . -include-classes MyCustomComponent -output library.swc
```

解压出SWC中的library.swf 文件，这时的大小接近320KB。

使用optimizer工具通过删除没用的调试和元数据代码减小RSL的大小：

```
> optimizer -keep-as3-metadata=  
    "Bindable,Managed,ChangeEvent,NonCommittingChangeEvent,Transient"  
    -input library.swf -output optimized.swf
```

优化后的RSL只有原来一半还少，接近135KB.

强烈建议你保留 Bindable, Managed, ChangeEvent, NonCommittingChangeEvent, 和 Transient 元数据名称标签，其他元数据如RemoteClass，可以添加到RSL库的类依赖项所基于的逗号分隔参数列表。

## 22.5节. 创建基于MXML的模块

### 22.5.1. 问题

我想创建基于MXML的模块，以便在运行期间载入。

### 22.5.2. 解决办法

创建一个继承自mx.modules.Module的MXML类，根标签为<mx:Module>，使用mxmlc命令行工具编译模块。

### 22.5.3. 讨论

一个模块类似于一个应用程序也是用mxmlc工具编译，生成的SWF文件可被应用程序载入或被其他模块在运行期间载入。要创建一个基于MXML的模块，需继承mx.modules.Module类，使用<mx:Module>作为MXML文件的根标签。

下面的例子是一个模块，显示一个联系人数据列表：

Code View:

```

<mx:Module
    xmlns:mx="http://www.adobe.com/2006/mxml"
    width="100%" height="100%"

    <mx:XMLList id="contacts">
        <contact>
            <name>Josh Noble</name>
            <phone>555.111.2222</phone>
            <address>227 Jackee Lane</address>
        </contact>
        <contact>
            <name>Todd Anderson</name>
            <phone>555.333.4444</phone>
            <address>1642 Ocean Blvd</address>
        </contact>
        <contact>
            <name>Abey George</name>
            <phone>555.777.8888</phone>
            <address>1984 Winston Road</address>
        </contact>
    </mx:XMLList>

    <mx:DataGrid id="contactGrid" width="100%" height="100%" rowCount="4" dataProvider="{contacts}">
        <mx:columns>
            <mx:DataGridColumn dataField="name" headerText="Name"/>
            <mx:DataGridColumn dataField="phone" headerText="Phone"/>
            <mx:DataGridColumn dataField="address" headerText="Address"/>
        </mx:columns>
    </mx:DataGrid>

</mx:Module>

```

模块的结构类似于应用程序或自定义组件。当应用程序需要显示联系人列表时，就可以载入这个模块，把它添加到显示列表了。

使用mxmlc工具编译这个例子：

> mxmlc ContactList.mxml

这个命令生成一个SWF文件，命名为ContactList.swf，你也可以使用output选项自定义名称。生成的SWF大小接近245KB—相当大，如果您认为内容都引入了编译的应用程序中，你可以

把这些代码分离到模块中，以便减少主应用程序的下载时间和文件大小。

## 22.6节. 创建基于ActionScript的模块

### 22.6.1. 问题

我想创建基于ActionScript的模块

### 22.6.2. 解决办法

创建一个继承自mx.modules.Module或mx.modules.Modulebase的 ActionScript类，使用mxmlc 编译模块。

### 22.6.3. 讨论

通过继承Module和ModuleBase类创建基于ActionScript的模块。根据模块在应用程序中所扮演的角色，继承Module或ModuleBase依据你是否需要显示列表。Module类是一个显示容器，继承自FlexSprite，引入了一些框架代码，ModuleBase类继承自EventDispatcher，可被用来分离应用程序逻辑代码使之不依赖于可视化元素。

基于MXML的模块是继承自mx.modules.Module，使用<mx:Module>作为根标签。如果你创建的模块包含可视化元素，你需要继承Module类，重写一些需要的protected方法，比如继承自UIComponent的createChildren方法。下面的例子演示一个带有可供用户输入信息的输入框组件模块：

Code View:

```
package
{
    import mx.containers.Form;
    import mx.containersFormItem;
    import mx.controls.TextInput;
    import mx.modules.Module;

    public class ASContactList extends Module
    {
        private var _form:Form;
        private var _firstNameItem:FormItem;
        private var _lastNameItem:FormItem;
        public function ASContactList()
        {
            super();
            this.percentWidth = 100;
            this.percentHeight = 100;
        }
    }
}
```

```

override protected function createChildren():void
{
    super.createChildren();
    _form = new Form();
    _firstNameItem = createInputItem( "First Name:" );
    _lastNameItem = createInputItem( "Last Name:" );
    _form.addChild( _firstNameItem );
    _form.addChild( _lastNameItem );
    addChild( _form );
}

private function createInputItem( label:String ):FormItem
{
    var item:FormItem = new FormItem();
    item.label = label;
    item.addChild( new TextInput() );
    return item;
}
}
}

```

编译方式和基于MXML的模块一样，同样是使用mxmcl工具：

> mxmcl ASContactList.as

这个命令生成一个名为ASContactList的SWF文件。Module类继承自mx.core.Container，因此所有子节点组件都会被作为mx.core.IUIComponent类型被加入到显示列表。你可以添加Flex框架的组件到ActionScript模块的显示列表中。如果是添加ActionScript API中的组件，比如flash.text.TextField和mx.media.Video，需要经过包装，成为UIComponent类实例。

Module类包含框架代码是为了与用户界面对象交互。如果你的模块没有一点框架代码，你可以继承ModuleBase。mx.modules.ModuleBase类继承自EventDispatcher提供一个方便的方法分离应用程序中的逻辑代码。

下面的例子模块是继承自ModuleBase类：

```

package
{
    import mx.modules.ModuleBase;

    public class EntryStateModule extends ModuleBase
    {
        public function EntryStateModule() {}

        public function greet( first:String, last:String ):String
        {
            return "Hello, " + first + " " + last + ".";
        }
    }
}

```

```

        }
    public function welcomBack(first:String, last:String
        ):String
    {
        return "Nice to see you again, " + first + ".";
    }
}

```

当被应用程序载入时，这个简单的模块提供一个简单的方式，用greet和welcomeBack方法提供打招呼用语。这个模块没有包含任何框架代码，因此编译后文件大小比使用Module类的模块小的多。

继承自ModuleBase的模块类的编译方式一样：

```
> mxmle EntryStateModule.as
```

这个命令生成名为EntryStateModule的SWF。要访问模块中的公开方法，父应用程序或模块需引用ModuleLoader实例的child属性或IModuleInfo实现类的factory属性，具体请看[第22.8节](#)。

## 22.7节. 使用ModuleLoader载入模块

### 22.7.1. 问题

我要载入模块

### 22.7.2. 解决办法

使用<mx:ModuleLoader>容器载入模块

### 22.7.3. 讨论

mx.modules.ModuleLoader类是一个容器类，功能类似于mx.controls.SWFLoader组件。它载入SWF并把模块添加到应用程序的显示列表。ModuleLoader和SWFLoader不同之处在于它有一个约定，这个被载入的SWF须实现IFlexModuleFactory。被编译的模块包含IFlexModuleFactory类工厂，它允许应用程序在运行期间动态载入模块化SWF而不需要在主应用程序中实现此接口。

虽然ModuleLoader对象是一个可视的容器，可载入继承自Module和ModuleBase的模块，不依赖于这个模块是否包含有框架代码或可视对象。ModuleLoader的url属性指向一个模块的具体位置。设置url属性后，组件内部会调用loadModule方法，开始下载模块。

下面的例子在应用程序同一个域中载入模块：

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">
```

```

<mx:Panel title="Contacts:" width="350" height="180"
    horizontalAlign="center" verticalAlign="middle">
    <mx:ModuleLoader url="ContactList.swf" />
</mx:Panel>

</mx:Application>

```

当应用程序启动时，ModuleLoader去载入ContactList.swf 模块，载入完成后，它被添加到应用程序显示列表中。

ModuleLoader组件也允许你动态卸载和加载模块。对ModuleLoader的url属性的设置在内部会调用loadModule方法，添加这个模块作为子节点。调用unloadModule方法可删除显示列表中的模块。调用unloadModule是设置module引用为null，但是并不会改变url属性值。

下面的例子中演示模块的加载和卸载：

Code View:

```

<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">

    <mx:Script>
        <![CDATA[

            private function displayModule( moduleUrl:String ):void
            {
                var url:String = moduleLoader.url;
                if( url == moduleUrl ) return;
                if( url != null ) moduleLoader.unloadModule();
                moduleLoader.url = moduleUrl;
            }

            private function showHandler():void
            {
                displayModule( "ContactList.swf" );
            }
            private function enterHandler():void
            {
                displayModule( "ContactEntry.swf" );
            }
        ]]>
    </mx:Script>

    <mx:Panel title="Contacts:" width="350" height="210"
        horizontalAlign="center" verticalAlign="middle">
        <mx:ModuleLoader id="moduleLoader" height="110" />

```

```

<mx:HRule width="100%" />
<mx:HBox width="100%">
    <mx:Button label="show list" click="showHandler();"/>
    <mx:Button label="enter contact"
        click="enterHandler();"/>
</mx:HBox>
</mx:Panel>

</mx:Application>

```

Button控件的Click事件处理器更新相应模块。这个应用程序通过加载ContactList.swf模块和ContactEntry.swf模块交替显示联系人信息列表和用户信息输入表单。

当模块加载到应用程序中后，它会被添加到mx.modules.ModuleManager对象的模块列表中。当删除时，引用会被设置为null以释放内存和资源。对于加载和卸载基于Flex的应用程序模块来说使用ModuleLoader是一种很方便的方法。

## 22.8节. 使用**ModuleManager**载入模块

### 22.8.1. 问题

我想具体控制模块的加载和卸载。

### 22.8.2. 解决办法

直接访问**ModuleManager**类的方法来监听加载模块的各种状态事件。

### 22.8.3. 讨论

ModuleManager类管理着加载的模块。当调用ModuleLoader.loadModule和ModuleLoader.unloadModule方法时<mx:ModuleLoader>组件内部就是和这个管理器进行着通信，你可以直接访问ModuleManager管理的模块。当模块的URL传递给ModuleManager.getModule方法后，这个模块位置被添加到模块的管理列表中，返回一个mx.modules.IModuleInfo类实例。

模块实际上是ModuleMananger的私有ModuleInfo类实例。ModuleInfo对象加载SWF文件，包装为实现了IModuleInfo的代理类，并通过ModuleManager.getModule方法返回实例对象。你可以监听这个代理的各种状态事件以便具体控制模块的加载。

下面的例子演示如何使用ModuleManager控制模块的显示：

Code View:

```

<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
    creationComplete="creationHandler();">

```

```

<mx:Script>
<! [CDATA[
    import mx.events.ModuleEvent;
    import mx.modules.ModuleManager;
    import mx.modules.IModuleInfo;
    private var _moduleInfo:IModuleInfo;
    private function creationHandler():void
    {
        _moduleInfo =
            ModuleManager.getModule( 'ContactList.swf' );
        _moduleInfo.addEventListener( ModuleEvent.READY,
                                     moduleLoadHandler );
        _moduleInfo.load();
    }
    private function moduleLoadHandler(evt:ModuleEvent
    ):void
    {
        canvas.addChild( _moduleInfo.factory.create() as
                        DisplayObject );
    }
]]>
</mx:Script>
<mx:Canvas id="canvas" width="500" height="500" />
</mx:Application>

```

当应用程序完成布局初始化操作后，通过ModuleManager.getModule方法加载ContactList模块并返回的IModuleInfo对象。当下载模块时IModuleInfo的实现扮演着Loader实例的代理。

当模块成功下载后，使用IFlexModuleFactory.create方法添加到显示列表。这个方法返回模块的实例对象，类型被映射为DisplayObject，添加到Canvas容器的显示列表。

Y通过getModule方法返回的IModuleInfo对象，你可以监听模块的各种下载状态事件。这个例子中，应用程序直到模块完全下载后才添加到显示列表，触发的事件类型为ModuleEvent。SWF模块的下载状态有progress到error等几个状态([Table 22-1](#))。

**Table 22-1. mx.events.ModuleEvent类**

常量	字符串值	描述
PROGRESS	"progress"	当模块正在加载时触发，你可以访问模块的bytesLoaded和bytesTotal属性。
SETUP	"setup"	当有足够的模块信息可用时触发。
READY	"ready"	当模块加载完成时触发。
UNLOAD	"unload"	当模块被卸载时触发。
ERROR	"error"	当下载模块途中出现错误时

		触发。
--	--	-----

IModuleInfo 实现类的unload方法用于删除ModuleManager中的模块引用，但是不会删除显示列表中的SWF。要删除显示列表中的模块，你必须显示调用父对象的removeChild方法。

相比较之下 ModuleLoader类根据url属性的更新去加载模块，而getModule方法返回的IModuleInfo实现类可以更好控制模块的加载和显示。这可以使你能预先加载模块，在与用户交互中能立即显示，缩短请求和渲染模块时间。

## 22.9节. 载入来自不同服务器的模块

### 22.9.1. 问题

我想加载不同服务器上的模块。

### 22.9.2. 解决办法

使用[flash.system.Security](#)类在主应用程序SWF文件和模块文件之间建立信任机制。

### 22.9.3. 讨论

Flash Player的安全机制是基于域的，SWF文件访问同一个域的数据是不受限制的。当SWF文件载入到Flash Player后，安全沙箱被建立，允许此域的所有资源可被访问。在此模型下请确保SWF是访问外部资源，与来自受信任源的SWF通信。

为了允许指定域的SWF等访问其他域的资源，模块，需要远程服务器有一个跨域授权文件以及在你的应用程序中使用Security.allowDomain方法。要让载入的模块能和父SWF交互---一种跨脚本通信---这个模块也需要调用allowDomain方法。

假设下面的模块放在远程服务器上：

```
<mx:Module
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    initialize="initHandler();">

    <mx:Script>
        <! [CDATA[
            private function initHandler():void
            {
                Security.allowDomain( "appserver" );
            }
        ]]>
    </mx:Script>

    <mx:Text width="100%" text="{loaderInfo.url}" />

```

```
</mx:Module>
```

当父 SWF 加载此模块并初始化事件被触发时，模块将授予加载的SWF文件访问通信权限，并显示已加载模块的 URL。.

当模块被编译放到远程服务器(这里的域名为moduleserver)，一个跨域授权文件被放到域名的根目录，允许appserver的父SWF文件能加载模块：

```
<?xml version="1.0"?>
<cross-domain-policy>
    <allow-access-from domain="appserver" to-ports="*" />
</cross-domain-policy>
```

为了让加载模块的SWF能与被加载模块进行跨脚本通信，你需要调用Security.allowDomain方法，参数为远程服务器域名，以载入crossdomain.xml文件，比如：

Code View:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    preinitialize="initHandler();">

    <mx:Script>
        <! [CDATA[
            private function initHandler():void
            {
                Security.allowDomain( "moduleserver" );
                Security.loadPolicyFile( "http://moduleserver/crossdomain.xml" );
                var loader:URLLoader = new URLRequest();
                loader.addEventListener(Event.COMPLETE,loadHandler);
                loader.load(new URLRequest("http://moduleserver/crossdomain.xml"));
            }
            private function loadHandler( evt:Event ):void
            {
                moduleLoader.url =
                    "http://moduleserver/modules/MyModule.swf";
            }
        ]]>
    </mx:Script>

    <mx:ModuleLoader id="moduleLoader" />

</mx:Application>
```

主应用程序的初始化事件处理函数调用SecurityDomain.allowDomain方法建立与任何来自moduleserver服务器的资源的通信。应用程序也调用Security.loadPolicyFile方法，Flash Player接收授权文件，确定appserver上的SWF是安全的。使用URLLoader加载跨域授权文件之前必须先调用loadPolicyFile方法，否则会抛出security异常。

当授权文件加载后，主应用程序的<mx:ModuleLoader>实例的url属性被赋值为来自远程服务器的请求模块。应用程序和模块相互授权可互访。

## 22.10节. 与模块通信

### 22.10.1. 问题

我想访问加载的模块和传递数据。

### 22.10.2. 解决办法

使用mx.modules.ModuleLoader的child属性和mx.modules.IModuleInfo实例的factory属性，监听事件，调用公开方法，访问公开属性。

### 22.10.3. 讨论

应用程序shell通过ModuleLoader和ModuleManager的属性与被加载模块通信。对于<mx:Application>实例来说通信没有限制，因为模块也可以加载另一模块，使得被加载模块访问父模块和应用程序是一样的。

要访问加载模块的数据，你需要把指定载入实例的返回属性的类型重新映射为原来被载入的模块类。当使用<mx:ModuleLoader>对象时，可通过child属性获得模块实例：

```
<mx:Script>
    <! [CDATA[
        private var myModule:MyModule;

        private function moduleReadyHandler():void
        {
            myModule = moduleLoader.child as MyModule;
            myModule.doSomething();
        }
    ]]>
</mx:Script>

<mx:ModuleLoader id="moduleLoader"
    url="MyModule.swf"
    ready="moduleReadyHandler();"/>
```

当主应用程序可以访问被加载模块的数据时，moduleReadyHandler时间处理函数被调用。  
<mx:ModuleLoader>的child属性类型重新映射为原模块类类型，现在你可以调用模块的公开方法访问其数据了。

而当使用ModuleManager类为父应用程序时，模块实例是通过IModuleInfo的IFlexModuleFactory实例的create方法返回的：

```
private var _moduleInfo:IModuleInfo;

private function creationHandler():void
{
    _moduleInfo = ModuleManager.getModule( 'MyModule.swf' );
    _moduleInfo.addEventListener( ModuleEvent.READY,
        moduleLoadHandler );
    _moduleInfo.load();
}

private function moduleLoadHandler( evt:ModuleEvent ):void
{
    var myModule:MyModule = _moduleInfo.factory.create() as
        MyModule;
    myModule.doSomething();
}
```

当ModuleLoader的child属性或IFlexModuleFactory.create方法返回值进行类型映射后，模块和主应用程序的联系分成紧密了，要想减少模块和它的类实例的紧密性，一般的做法是使用接口。使用接口，可使你的代码更具灵活性，使你的主应用程序能连接更多的类实例。

为了例证当开发模块化应用程序时使用接口所产生的灵活性，我们假定创建一个供用户输入信息的表单模块。根据程序的步骤和变化，你可能发现你需要更多类型的表单。虽然他们可能显示不同的外观，或者对用户数据执行不同的操作，但是他们访问模块数据的方法却是一样的。采用实现接口的不同模块可增加你的应用程序的灵活性。

下面的例子是一个接口，列出了和用户信息相关的属性方法：

```
package
{
    import flash.events.IEventDispatcher;

    public interface IUserEntry extends IEventDispatcher
    {
        function getFullName():String;
        function get firstName():String;
        function set firstName( str:String ):void;
        function get lastName():String;
        function set lastName( str:String ):void;
    }
}
```

创建实现此接口的模块，申明<mx:Module>的implements属性为IUserEntry接口：

Code View:

```
<mx:Module
    xmlns:mx="http://www.adobe.com/2006/mxml"
    implements="IUserEntry"
    layout="vertical"
    width="100%" height="100%>

    <mx:Metadata>
        [Event(name="submit", type="flash.events.Event")]
    </mx:Metadata>

    <mx:Script>
        <![CDATA[
            private var _firstName:String;
            private var _lastName:String;
            public static const SUBMIT:String = "submit";

            private function submitHandler():void
            {
                firstName = firstNameInput.text;
                lastName = lastNameInput.text;
                dispatchEvent( new Event( SUBMIT ) );
            }

            public function getFullName():String
            {
                return _firstName + " " + _lastName;
            }

            [Bindable]
            public function get firstName():String
            {
                return _firstName;
            }
            public function set firstName( str:String ):void
            {
                _firstName = str;
            }

            [Bindable]
            public function get lastName():String
            {
                return _lastName;
            }
            public function set lastName( str:String ):void
        ]]>
    </mx:Script>

```

```

    {
        _lastName = str;
    }
] ] >
</mx:Script>

<mx:Form>
<mx:FormItem label="First Name:>
    <mx:TextInput id="firstNameInput" width="100%" />
</mx:FormItem>
<mx:FormItem label="Last Name:>
    <mx:TextInput id="lastNameInput" width="100%" />
</mx:FormItem>
<mx:Button label="submit" click="submitHandler();"/>
</mx:Form>

</mx:Module>

```

这个模块提供一些控件供用户输入和提交用户信息。getter/setter 属性和getFullName方法将在模块的<mx:Script>标签里实现。由实现 IUserEntry 接口的 mx.modules.Module 和 mx.modules.ModuleBase类建立firstName和lastName的数据绑定。

要访问这里或其他实现IUserEntry接口的模块数据，父已运行必须对具体模块加载类的相应属性进行类型映射。

下面的例子使用<mx:ModuleLoader>的child属性实例访问实现IUserEntry模块的数据：

Code View:

```

<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">

<mx:Script>
<! [CDATA[
    private var myModule:IUserEntry;

    private function moduleReadyHandler():void
    {
        myModule = moduleLoader.child as IUserEntry;
        myModule.addEventListener( "submit", submitHandler );
    }

    private function submitHandler( evt:Event ):void
    {
        welcomeField.text = 'Hello, ' +
            myModule.getFullName();
    }
]]>
</mx:Script>

```

```

        trace( myModule.firstName + " " + myModule.lastName );
    }
] ]>
</mx:Script>

<mx:ModuleLoader id="moduleLoader"
    url="ContactEntry.swf"
    ready="moduleReadyHandler();"/>
<mx:Label id="welcomeField" />

</mx:Application>

```

<mx:ModuleLoader>实例的ready事件处理函数建立对用户信息提交的事件处理器。当submitHandler方法被调用时，通过模块的getFullName实现打印一些字符串。把ModuleLoader实例的child属性类型映射为IUserEntry接口，是确保应用程序和模块之间的松耦合设计，这使得你可以动态的与实现统一接口的不同模块类交互。

父SWF访问模块数据没有限制，通过parentApplication属性模块也可以访问父应用程序数据：

```

<mx:Module
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    creationComplete="creationHandler();">

    <mx:Script>
        <! [CDATA[
            private function creationHandler():void
            {
                //Flex SDK 3.1.0里已找不到getInformation方法
                infoField.text = parentApplication.getInformation();
            }
        ]]>
    </mx:Script>

    <mx:Text id="infoField" />

```

</mx:Module>

当模块完成初始化后，creationHandler方法被调用，调用父应用程序getInformation方法返回的子节点组件信息。

模块的parentApplication属性继承自UIComponent超类，是一个Object类型。动态类Object是所有类的父类。因此你可以通过parentApplication实例访问数据而不用关心父类实现。也就是说模块也直接调用父应用程序的属性，不管其属性是否可用而引发的运行时异常。

一般来讲，模块是不应该访问父应用程序数据的，这是根据模块和父应用程序之间的松耦合设计原则决定的。为了减小这种联系，你可以把载入模块的应用程序映射为接口，就像上面的例子那样做。要确保不同的应用程序都能和同一个模块通信，强烈建议直接提供父应用程

序数据给模块而不是通过动态parentApplication属性.做到了这些你就能轻松地开发模块化应用程序了。

## 22.11节. 使用查询字符串传递数据给模块

### 22.11.1. 问题

我想在模块载入期间传送数据给模块。

### 22.11.2. 解决办法

在模块SWF的URL里加上查询字符串，当模块加载完成后，使用模块的loaderInfo属性解析URL字符串。

### 22.11.3. 讨论

你可以追加查询字符串参数给模块载入类的URL。当模块载入后，通过 mx.modules.Module类的loaderInfo属性访问这个URL。使用ActionScript，你可以解析出有用URL中参数信息。查询字符串紧跟在?号子后用&符号分割多个参数。

下面的例子程序在模块后追加了查询字符串：

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    creationComplete="creationHandler();">

    <mx:Script>
        <! [CDATA[
            private static const F_NAME:String = "Ted";
            private static const L_NAME:String = "Henderson";

            private function creationHandler():void
            {
                var params:String = "firstName=" + F_NAME +
                    "&lastName=" + L_NAME;
                moduleLoader.url = "NameModule.swf?" + params;
            }
        ]]>
    </mx:Script>

    <mx:ModuleLoader id="moduleLoader" />

</mx:Application>
```

主应用程序初始化和<mx:ModuleLoader>实例完成后，url属性中添加了参数对。firstName和lastName属性值被传入到载入的模块中。

载入的模块通过loaderInfo属性解析URL，获得传入的参数数据：

Code View:

```
<mx:Module
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    width="100%" height="100%"
    creationComplete="creationHandler();">

    <mx:Script>
        <! [CDATA[
            import mx.utils.ObjectProxy;

            [Bindable] private var _proxy:ObjectProxy;

            private function creationHandler():void
            {
                _proxy = new ObjectProxy();

                var pattern:RegExp = /.*\?/;
                var query:String = loaderInfo.url.toString();
                query = query.replace( pattern, "" );

                var params:Array = query.split( "&" );

                for( var i:int = 0; i < params.length; i++ )
                {
                    var keyVal:Array = ( params[i] )
                        .toString().split("=");
                    _proxy[keyVal[0]] = keyVal[1];
                }
            }

        ]]>
    </mx:Script>

    <mx:Text text="{ 'Hello, ' + _proxy.firstName + ' ' +
        _proxy.lastName }" />

</mx:Module>
```

被载入模块解析URL中的字符串参数，把它们作为属性添加到ObjectProxy类对象中，通过ObjectProxy类的数据绑定特性，当属性值被更新时，Text控件值也会被更新。

使用查询字符串传递数据是被载入模块获取数据，处理数据的简便方式。

## 22.12节. 使用连接报告优化模块

### 22.12.1. 问题

我想减小文件大小和模块的后续下载时间。

### 22.12.2. 解决办法

当编译应用程序时使用mxmlc工具的link-report命令行参数生成一个连接报告文件，然后在编译模块时把报告文件作为load-externs命令行参数值，确保只有模块需要的类被编译进来。

### 22.12.3. 讨论

当你编译模块时，所有模块依赖的自定义或框架代码都被编译进生成的SWF文件中。在这些代码中，特别是框架代码很多都是和主程序和模块所共有的。你可以根据一个连接报告文件删除模块中多余的代码以减小文件大小。

连接报告文件列出了主程序依赖的类，当编译主程序时使用link-report命令行参数生成连接报告文件。下面的命令生成连接报告文件report.xml：

```
>mxmlc -link-report=report.xml MyApplication.mxml
```

生成的连接报告文件供编译模块时用，要删除多余的代码，减小模块文件大小，编译模块时使用link-externs命令行参数值设置为刚才生成的连接报告文件：

```
>mxmlc -link-externs=report.xml MyModule.mxml
```

生成的模块SWF文件不包含任何主程序和模块依赖代码。当你的主程序和模块都有框架代码时这是一个很好的优化工具。根据连接报告文件排除的代码和主程序和模块有一定关系的，也就是说如果主程序代码发生改变，你就需要重新生成连接报告文件并重新编译模块。

如果主程序使用多个模块，这项技术也可被用来编译那些在主程序中多余，而多个模块都需要的代码。实际上，这是通用的规则去编译任何管理类，比如mx.managers.DragManager 和 mx.managers.PopUpManager，模块可能依赖于主程序。这是因为模块不能访问另一模块的资源，否则就会抛出运行时异常，例如一个模块试图引用其他模块的DragManager。

要确保模块都是引用主程序中的同一个管理器，你需要在主程序中导入和申明一个类本地变量：

```
import mx.managers.PopUpManager;  
var popUpManager:PopUpManager;
```

用生成的连接报告文件去编译模块，确保模块都是使用同一个管理器引用，以减少代码冗余和模块大小。

## 第二十三章. Adobe Integrated Runtime API(常青)

Flex SDK中的类都可以用于开发基于Adobe Integrated Runtime (AIR)的桌面应用程序。. Adobe AIR是一个跨平台的运行时环境，允许开发人员借助已有的web技术把富联网应用体验带到桌面上。AIR运行时提供统一的跨操作系统环境，使得开发人员集中精力面向Adobe AIR 平台开发程序而不用关心在不同的操作系统中构建和部署应用程序。AIR 框架也提供创建基于HTML和Ajax的桌面程序功能，在这一章的例子中我们主要讨论借助Flex 框架创建AIR程序。

运行AIR程序，首先要安装运行时，可以在官方网站上<http://labs.adobe.com/technologies/air/>找到，通过Adobe AIR运行时运行的程序和本地程序是一样的。

借助Flex Framework创建AIR程序的过程和创建基于web浏览器的Flex程序非常类似。Flex 3 SDK所包含的类可与文件系统，操作系统剪贴板，和本地数据库交互。要把AIR程序打包为安装文件的话，你需要一个应用程序SWF文件和一个应用程序描述文件，以及对应用程序进行加密签名的keystore证书。要想对你的应用程序进行签名---请确认最终用户会安装纯正的版本---你可以生成自签名证书或者使用可靠的证书供应商如VeriSign或Thawte。AIR 安装文件的扩展名为 .air，可以在Adobe AIR RunTime下运行。.

涉及AIR API中所有特性可能需要一本书来讲解，这一章重点讲解一些开发桌面程序最主要的特性。

### 23.1节. 借助Flex Framework创建一个AIR程序

#### 23.1.1. 问题

我想使用Flex和AIR APIs创建桌面应用程序。

#### 23.1.2. 解决办法

创建主应用程序文件，在MXML中用<mx:WindowedApplication>作为根标签，还有一个应用程序描述文件用于提供安装，访问和运行程序所需要的属性信息。

#### 23.1.3. 讨论

用Flex框架开发基于AIR的程序非常类似于开发Web用途的Flex程序，AIR程序提供与操作系统通信的能力。当开发基于 Web 的 Flex 程序时，主应用程序文件包含的根标签是<mx:Application>。因为AIR程序使用本地操作系统窗体运行在桌面上，所以主应用程序文件的根标签是<mx:WindowedApplication> 。

mx.core.WindowedApplication类扩展自mx.core.Application并提供了与本地操作系统窗体相關联的属性和方法。WindowedApplication是AIR程序的主窗口，进而生成其他本地窗体，访问文件系统，与操作系统交互等你所期望的大多数桌面程序应有的功能。

下面的例子是一个简单的AIR程序主文件：

```
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical" title="Hello World">
    <mx:Label text="Welcome to AIR" />
```

```
</mx:WindowedApplication>
```

当程序运行时，标题Hello World将会显示在程序窗口的标题栏以及系统任务栏上。文字Welcome to AIR被显示在屏幕上。

通过Flex 3 SDK的bin目录下的amxmlc命令行工具来编译air源程序。它使用起来和mxmlc工具差不多，只不过后者是编译Flex以及基于ActionScript的程序。当你运行mxmlc工具时，它会使用默认的frameworks目录下的flex-config.xml配置文件。而amxmlc工具使用air-config.xml配置文件和libs/ari目录下的airglobal.swc 库文件。

把/bin目录添加到系统路径后，可直接打开命令行窗口输入以下命令：

```
>amxmlc HelloWorld.mxml
```

这会生成一个HelloWorld.swf 文件。如果你点击这个SWF文件，除了背景颜色你看不到任何东西，这是因为它现在运行在Flash Player之上。对于AIR程序，你需要让它运行在Adobe AIR运行时环境下。生成的SWF文件是部署和打包AIR程序的必要文件之一，还有一个文件就是程序描述文件。

应用程序描述符是一个XML文件，规定了一些被用来安装，运行，识别AIR程序所需的属性信息。这个文件在开发部署和打包分发阶段是必须的。除应用程序安装的具体目录和关联的图标文件以外，程序描述符让你在程序运行后变为只读之前先设置好应用程序窗体的窗体属性。

下面是一个很基本的应用程序描述文件：

```
<application xmlns="http://ns.adobe.com/air/application/1.0">

    <id>com.oreilly.flexcookbook.HelloWorld</id>
    <filename>HelloWorld</filename>
    <name>Hello World</name>
    <version>0.1</version>
    <description>A Hello World Application</description>

    <initialWindow>
        <content>HelloWorld.swf</content>
        <systemChrome>standard</systemChrome>
        <transparent>false</transparent>
        <visible>true</visible>
        <width>400</width>
        <height>400</height>
    </initialWindow>

</application>
```

这些参数集只是所有属性中的一部分，不过这些都是很常用的配置信息。`<application>` 标签是描述文件的根标签且有个属性是指向AIR名称空间的URI。名称空间最后面的(1.0)是指该程序运行所需要的AIR运行时版本。

描述文件所需的参数有：`<id>`, `<filename>`, `<version>`, 和`<initialWindow>` 标签以及`<content>`

子节点。Id是程序的标志符且是唯一的，因此，我们推荐用反写的域名作为其值。ID还被用作安装时程序的存储目录。Filename被用来指向操作系统中的该程序，包括程序的执行名称以及安装目录名。Version指明程序的版本号，一般不需要，但是在升级程序时是必需的。Content是<initialWindow> 的节点，指出将被AIR载入的主应用程序SWF文件，这个SWF文件正是用amxmlc命令行工具生成的。

除此之外，在这个例子中还添加了<name>, <description>, name和description值将在程序安装界面显示，<initialWindow>的systemChrome, transparent, 和visible节点和程序窗口的外观有关系。这些属性可以在描述文件里设置，也可以在根标签<mx:WindowedApplication>里作为属性设置，但是在程序运行后，这些属性将都变为只读状态。Width和height属性指明窗口大小，这是可选的属性。

## 23.2节. 理解AIR命令行工具

### 23.2.1. 问题

我想使用Flex 3 SDK中的命令行工具部署，调式和打包AIR程序。

### 23.2.2. 解决办法

使用/bin目录下的amxmlc, adl, 和 adt工具。

### 23.2.3. 讨论

Flex 3 SDK包含的命令行工具涉及编译，运行和打包等等。如果要编译一个AIR程序，可使用amxmlc工具并加上文件名路径作为参数，文件可以是HTML, ActionScript, 或者MXML。本章的例子重点讲解利用Flex Framework来开发AIR程序，因此文件类型一般为.mxml格式。Adl工具被用来在开发期间运行测试AIR程序，而adt工具则是用于打包AIR程序为安装文件。

Amxmlc工具用于生成AIR程序主SWF文件，这和mxmlc命令行有些相似。事实上amxmlc命令是调用mxmlc命令并指示编译器使用/frameworks目录下的air-config.xml配置文件，所有mxmlc命令行参数都可用，比如指定默认或额外的配置文件，不过这一章的例子使用最简单的选项生成SWF文件。

Amxmlc的文件参数就是指主应用程序文件，其父标签为<mx:WindowedApplication>。下面的例子是一个很简单的AIR程序主文件：

```
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical" title="Hello World">

    <mx:Label text="Welcome to AIR" />
```

</mx:WindowedApplication>  
mx.core.WindowedApplication类基于Flex的AIR应用程序容器。WindowedApplication类继承自mx.core.Application类，且是AIR程序的最主要的本地桌面窗口。

要编译此程序并在AIR环境下运行，可像mxmlc那样使用amxmlc工具：

```
>amxmlc -output HelloWorld.swf  HelloWorld.mxml
```

这会在当前目录下生成名为HelloWorld.swf的SWF文件。Output选项是可选的，没有它仍可获得相同结果，如果没有这个output选项，编译器会根据文件名来命名生成的SWF文件，利用这个选项可以重命名生成的SWF文件。

在前面的章节中已经讲到，应用程序描述符是一个XML文件用于存储和安装，运行和识别AIR程序所需特定属性，如果在Flash Player下运行amxmlc生成的SWF文件，你会看到只有背景颜色，这是因为该SWF文件只能运行在AIR环境之下，要想运行程序，你需要一个指向此程序的应用程序描述文件。

下面的例子是一个简单的应用程序描述文件，保存为HelloWorld-app.xml，指定之前生成的SWF文件为程序content文件：

```
<application xmlns="http://ns.adobe.com/air/application/1.0">
    <id>com.oreilly.flexcookbook.HelloWorld</id>
    <filename>HelloWorld</filename>
    <name>Hello World</name>
    <version>0.1</version>
    <description>A Hello World Application</description>
    <initialWindow>
        <content>HelloWorld.swf</content>
        <systemChrome>standard</systemChrome>
        <transparent>false</transparent>
        <visible>true</visible>
        <width>400</width>
        <height>400</height>
    </initialWindow>
</application>
```

在描述文件中的初始化应用程序窗口(WindowedApplication实例)有些和程序相关的属性和参数，你会注意到有个content元素值就是指向之前用amxmlc工具生成的SWF文件路径。

在开发AIR程序期间，你可能用到Adobe Debug Launcher (adl)工具用于运行程序，在打包和安装前进行预览。Adl工具也支持trace语句输出的控制台的打印信息，要想通过adl预览AIR程序，输入下面的命令并加上描述文件即可：

```
>adl HelloWorld-app.xml
```

如果命令运行成功的话，我们会看到下面的画面 **Figure 23-1:**

**Figure 23-1. Hello World AIR application**



程序按照预定设想运行的话，下一步就可以通过AIR Developer Tool (adt)打包为AIR程序为安装文件。AIR安装文件需要数字证书签名来验证程序的真伪，要生成一个AIR安装文件，你需要从安全的证书管理机构如VeriSign或Thawte获取证书，或者你使用自签名证书。证书颁发机构的代码签名在安装程序时验证签名者的身份以及保证最终用户所得到的程序没有被恶意修改。通过adt工具创建的自签名证书也可以保证签名后程序没有被修改过，但是不能用来确认签名者的身份并且发布者属性在安装窗口显示为unknown。

我们还是推荐从证书颁发机构获取的证书对AIR程序进行代码签名，不过在这章的例子中，我们将使用自签名证书生成安装文件。

用adt工具输入以下命令创建一个自签名证书：

```
>adt -certificate -cn HelloWorld 1024-RSA certificate.pfx password
```

这个命令在当前目录下生成名为certificate.pfx的文件，还有相应的证书和密匙，这里的证书名为HelloWorld，密匙类型被设定为1024-RSA。可用的密匙类型有1024-RSA 和2048-RSA，扩展名为.pfx 和.p12，这就是个人信息交换文件类型。

被adt工具创建的证书和密匙被存储在PKCS12-type 类型的密匙存储器文件。证书和密匙用来签名和打包AIR程序。输入下面的命令对程序打包为安装文件：

Code View:

```
>adt -package -storetype pkcs12 -keystore certificate.pfx HelloWorld.air HelloWorld-app  
.xml HelloWorld.swf
```

Storetype参数指向Public-Key Cryptography Standards (PKCS) 的版本密匙存储器。密匙存储器是通过之前的adt工具创建的。最后三个参数值分别代表安装文件的名称，描述文件和SWF文件。在末尾你还可以继续添加程序所需的其他资源。这里的定位是相对的即当前目录，不过你可以使用-C选项改变引入文件的具体位置。

运行此命令后，终端会提示你输入密码用于生成证书；对于这个例子，如果命令运行成功，则密码被接受，然后在当前目录下生成一个叫HelloWorld.air的文件。这个文件就是Hello World程序的安装文件。要想安装这个程序，只要双击它，你会看到一个熟悉的窗口Figure 23-2.

**Figure 23-2.** 初始安装窗口

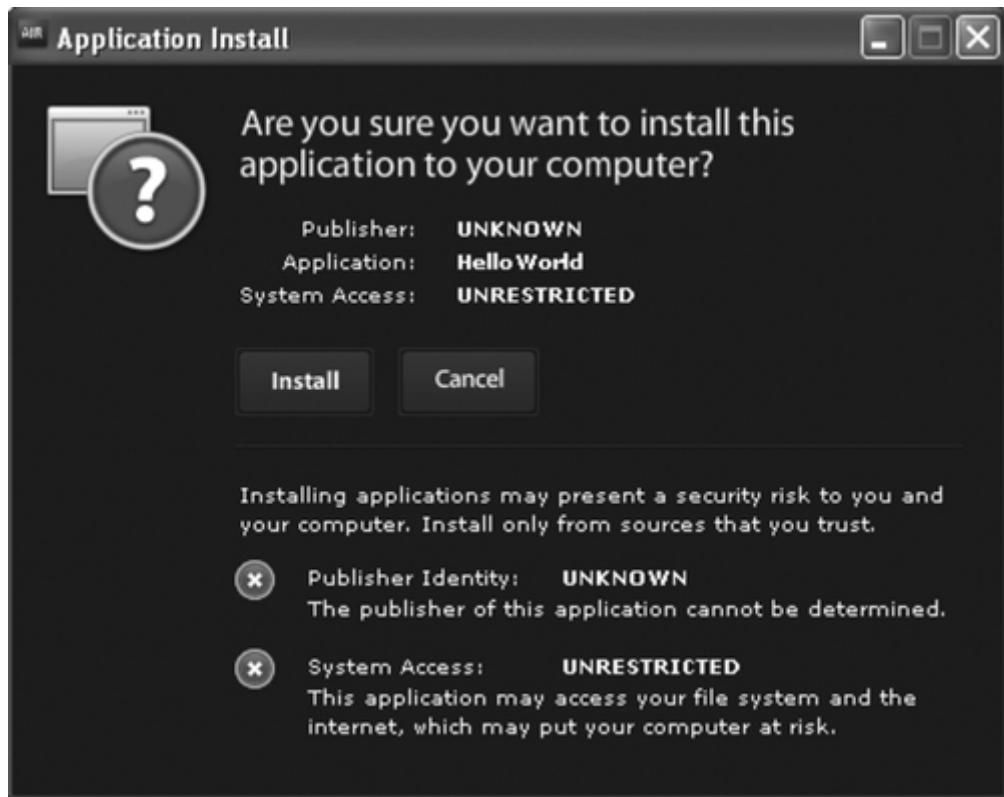
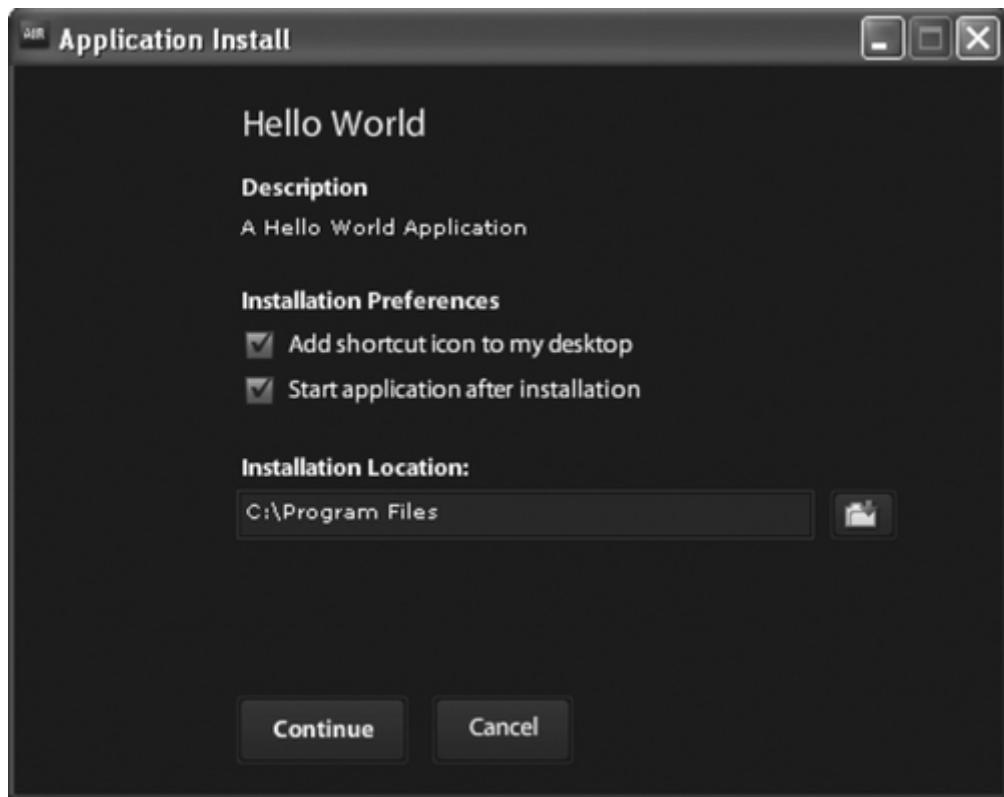


Figure 23-2 是一个最初的安装窗口，应用程序的名称通过描述文件的<name>属性给出，publisher身份被认为是unknown，这是因为在打包时使用了自签名证书，如果你点击Install按钮，你会看到下一个窗口 Figure 23-3.

**Figure 23-3.** 第二个安装窗口



点击第二个窗口的Continue按钮将安装程序并添加快捷方式到桌面上。你可以指定安装目录，如果默认的话，程序会安装到C:\Program Files\HelloWorld而Mac OS X会安装在HD/Applications/HelloWorld。应用程序安装目录就是程序目录，储存文件的目录被称为存储目录，可在以下目录中找到程序，Windows的C:\Documents and Settings\<username>\Application Data\com.oreilly.flexcookbook.HelloWorld以及Mac OS X的在/Users/<username>/Library/Preferences/com.oreilly.flexcookbook.HelloWorld。这些目录名称就是来自描述文件的ID--用反写的域名。在安装期间创建的程序目录和存储目录都是通过AIR的system API写成的，你还可以读写存储目录，但程序目录是只读的。

AIR程序的删除操作和其他桌面程序一样，通过控制面板的添加删除工具，在Mac OS中直接拖程序目录到回收站中即可，但是这不会删除安装后生成的辅助文件，包括存储目录。

## 23.3节. 打开和管理本地窗体

### 23.3.1. 问题

我想在AIR程序中创建本地窗体

### 23.3.2. 解决办法

使用flash.display.NativeWindow 和 mx.core.Window 类

### 23.3.3. 讨论

你可以依照操作系统的约定创建本地窗体，不仅仅是它们的功能还有它们的外观。而且除了

可以轻松创建和其他桌面程序一样的外观和矩形形状的本地窗体，还可以通过style属性和自定义图像创建自定义皮肤，自定义窗体仍拥有操作窗口的控件，通过这些控件可以监听本地窗体发出的事件。

AIR程序的MXML主标签为<mx:WindowedApplication>，这个在程序运行后就是程序的初始窗口。通过应用程序描述文件可进行自定义设置或者直接在<mx:WindowedApplication>标签内申明。WindowedApplication窗口扮演者作为flash.display.NativeWindow 类实例的容器，让你可直接在 MXML 中添加 Flex 组件到显示列表。你可通过 nativeWindow 属性访问 NativeWindow实例。

NativeWindow类扮演者一个接口，用于控制本地桌面窗口。要通过Flex Framework创建本地窗口，可用mx.core.Window类实例创建，就像WindowedApplication类，Window类扮演者底层本地窗口实例的容器，可通过nativeWindow属性访问，window的初始化属性可在根标签<mx:Window>中进行设置。

下面的例子是一个继承自Window类的自定义本地窗口：

```
<mx:Window xmlns:mx="http://www.adobe.com/2006/mxml"
    title="Hello" width="200" height="200">

    <mx:Label text="I am a Window!" />

</mx:Window>
```

当打开时，窗口将被显示在桌面上，大小为宽200，高200，窗口标题栏显示Hello，其他窗口的chrome 和transparency属性可在<mx:Window>标签中设置，以及NativeWindow实例的事件处理函数。下面的例子删除标准系统样式，约束窗口最大化和最小化控件，添加窗体控件的事件监听器：

Code View:

```
<mx:Window xmlns:mx="http://www.adobe.com/2006/mxml"
    title="Hello" systemChrome="none" transparent="true"
    maximizable="false" minimizable="false"
    width="200" height="200"
    windowComplete="completeHandler();"
    closing="closingHandler(event);">

<mx:Script>
    <![CDATA[
        private function completeHandler():void
        {
            nativeWindow.addEventListener(
                NativeWindowBoundsEvent.RESIZING,
                resizeHandler );
        }

        private function resizeHandler(
            evt:NativeWindowBoundsEvent):void
        {
            trace( evt.beforeBounds + " : " + evt.afterBounds );
        }
    ]]>
</mx:Script>
```

```

        }
    private function closingHandler( evt:Event ):void
    {
        trace( "goodbye!" );
    }
} ]>
</mx:Script>

<mx:Label text="I am a Window!" />

</mx:Window>

```

当窗口打开后，窗体样式被替换为标准的Flex样式，且禁止了最大化和最小化控件，事件处理器响应NativeWindow实例发出的creation, resizing, 和closing事件。

API中包含一些选项用于设置本地窗体类型。这些类型属性值在NativeWindowTypes类中，窗体被认为是由自定义边框和控件组成，默认，一个窗体的类型为normal。utility 和lightweight类型不能显示在Windows操作系统的任务栏或Mac OS X 菜单中。Utility窗体被认为是精简的normal窗体，因为它只有简单的边框和一个关闭按钮，Lightweight窗体需要一个systemChrome属性设置为false，它的行为像一个通知窗口，就像其他桌面程序的信息通知窗口那样。

标题栏，状态栏和resizing处理都可通过 <mx:Window> 组件的showTitleBar, showStatusBar, 和showGripper属性进行设置。要想删除标准的Flex边框，只要把<mx:Window> 标签的showflexChrome 属性设为false，这样窗体只显示内容而没有标准的系统边框控件；在这个例子中只显示文本I am a Window! 而没有任何背景或窗体边框。

要显示一个<mx:Window>组件，你首先要创建window实例并调用open方法：

Code View:

```

<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    windowComplete="initHandler();"
    closing="closeHandler();">

    <mx:Script>
        <! [CDATA[
            import com.oreilly.flexcookbook.CustomWindow;

            private var window:CustomWindow;
            private function initHanlder():void
            {
                window = new CustomWindow();
                window.alwaysInFront = true;
                window.open();
        ]
    
```

```

        }

private function closeHandler():void
{
    if ( !window.closed ) window.close();
}
] ]>
</mx:Script>

</mx:WindowedApplication>

```

当程序完成初始化布局并打开底层NativeWindow实例，windowComplete事件被触发，initHandler方法被调用。在initHandler方法内，CustomWindow 实例被创建并打开。设置alwaysInFront属性窗口被锁定在Z轴最上层，当然你也可以自己使用Window类的orderInFrontOf, orderInBackOf, orderToFront 和orderToBack 方法自己设置。

需要注意的是，当关闭主应用程序窗口之前，关闭其他自定义窗口都不会导致主窗口关闭，closing事件监听器的目的就是退出程序确定所有的窗口要关闭。

## 23.4节. 创建本地菜单

### 23.4.1. 问题

我想提供一个本地菜单供用户执行特殊的命令。

### 23.4.2. 解决办法

使用本地的菜单API 创建程序和窗体菜单。

### 23.4.3. 讨论

本地菜单API类提供了操作系统本地菜单的相关特性。你可以添加菜单项以及监听菜单项选择事件。有好几种本地菜单类型，如何创建和交互这取决于你的AIR程序所运行的操作系统，因此你添加菜单时要确认是否支持所有目标操作系统。

Mac OS X操作系统支持应用程序菜单。应用程序菜单是个全局菜单，可被程序工具栏所访问，且不依赖于当前获得焦点的本地窗口。应用程序菜单是MAC OS X操作系统自动创建的，你可以往上面添加菜单项或子菜单以及事件处理器，成为标准菜单。

Windows操作系统支持的本地菜单本认为是窗体菜单。Window菜单显示在本地窗体对象上，刚好在标题栏下面，且只支持具有系统边框的本地窗体，因此当<mx:Window>实例的systemChrome属性被设为none时Window菜单是不能被添加进去的。

要检测AIR程序运行在什么操作系统上，你可以使用NativeWindow.supportsMenu和NativeApplication.supportsMenu属性：

```
if( NativeWindow.supportsMenu )  
{  
    // Windows  
}  
  
else if( NativeApplication.supportsMenu )  
{  
    // Mac OS X  
}
```

如果NativeWindow.supportsMenu属性值为true，运行在Windows操作系统下的程序具有本地窗口菜单。flash.desktop.NativeApplication类提供程序级信息和函数，其中有个静态的supportsMenu属性。如果NativeApplication对象支持程序级菜单，则说明当前运行在Mac OS X环境中。

要添加本地菜单对象作为根菜单，可先实例化NativeMenu，并赋值给NativeWindow或NativeApplication实例的menu属性：

```
if( NativeWindow.supportsMenu )  
{  
    stage.nativeWindow.menu = new NativeMenu();  
}  
  
else if( NativeApplication.supportsMenu )  
{  
    NativeApplication.nativeApplication.menu = new NativeMenu();  
}
```

设置好根菜单后，接着就可以添加菜单或子菜单到NativeMenu对象上了。下面的例子在<mx:WindowedApplication>上创建一个应用程序菜单或窗体菜单，这取决于什么操作系统上：

#### Code View:

```
<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    windowComplete="initHandler();">

<mx:Script>
    <![CDATA[
        private function initHandler():void
        {
            var fItem:NativeMenuItem =
                new NativeMenuItem( "File" );
        }
    ]]>

```

```

var fileMenu:NativeMenuItem;
if( NativeWindow.supportsMenu )
{
    stage.nativeWindow.menu = new NativeMenu();
    fileMenu = stage.nativeWindow.menu.addItem( fItem );
}
else if( NativeApplication.supportsMenu )
{
    NativeApplication.nativeApplication.menu =
        new NativeMenu();
    fileMenu =
        NativeApplication.nativeApplication.menu.addItem( fItem );
}

NativeApplication.nativeApplication.menu.addItem( fItem );
}
fileMenu.submenu = createFileMenu();
}

private function createFileMenu():NativeMenu
{
    var menu:NativeMenu = new NativeMenu();

    var openItem:NativeMenuItem =
        new NativeMenuItem( "Open" );
    var openCmd:NativeMenuItem = menu.addItem( openItem );
    openCmd.addEventListener( Event.SELECT, openHandler );

    var saveItem:NativeMenuItem =
        new NativeMenuItem( "Save" );
    var saveCmd:NativeMenuItem = menu.addItem( saveItem );
    saveCmd.addEventListener( Event.SELECT, saveHandler );

    return menu;
}
private function openHandler( evt:Event ):void
{
    printOut.text += "You selected open.\n";
}
private function saveHandler( evt:Event ):void
{
}

```

```

        printOut.text += "You selected save.\n";
    }
] ]>
</mx:Script>

<mx:TextArea id="printOut" width="100%" height="100%" />
</mx:WindowedApplication>
```

当应用程序被创建，布局初始化完成后，initHandler方法被调用，新的File菜单被添加到WindowedApplication或NativeApplication实例的menu属性上，这取决于你的操作系统(Windows 或Mac OS X)。通过createFileMenu方法子菜单被添加到File菜单中，事件监听器被添加用于响应每个目录项的select事件。

当应用程序运行在Windows操作系统时，主应用程序窗口将在标题栏下面显示File菜单。当运行在Mac OS X 操作系统时，File菜单将会显示在应用程序工具栏上。选择File菜单后会打开拥有子菜单选项的上下文菜单。点击任何菜单项会调用注册了的事件监听器并在<mx:TextArea>对象上打印出文本。

NativeMenuItem类还提供分割线支持，构造器默认的isSeparator参数值为false，如果设为true则会显示一条水平线：

```
var rule:NativeMenuItem = new NativeMenuItem("Line", true);
```

你还可以在运行期间通过NativeMenuItem类的enabled属性启动和禁止菜单项：

```
var saveItem:NativeMenuItem = new NativeMenuItem("Save");
saveItem.enabled = false;
```

通过在运行期间启动和禁止菜单，可以设定哪些菜单命令可用，这依赖于你的程序需求。

本地菜单的概念不仅仅局限于应用程序菜单和窗体菜单，还有如上下文菜单----鼠标右键打开，或者是窗口中命令打开的菜单以及系统托盘打开的菜单都可以认为是本地菜单，实际上flash.ui.ContextMenu和flash.ui.ContextMenuItem类都是继承自 AIR native menu API.

## 23.5节. 读写文件

### 23.5.1. 问题

我想在文件系统上创建，访问和写文件。

### 23.5.2. 解决办法

使用AIR的file system API的File, FileStream, 和 FileMode类。

### 23.5.3. 讨论

一个File对象是一个指针，代表一个文件或一个目录。要读写文件到硬盘驱动器，你可以使

用FileStream把File对象放入一个缓冲区，通过FileStream类的同步和异步方法读取和写入。当使用的同步的FileStream.open方法时，文件被当作一个ByteArray对象，其他任何操作将被暂停直到文件被读取或写入。而异步的FileStream.openAsync方法类似于URLStream对象，数据被放置在缓冲区。使用同步还是异步方法这取决于你的程序需求，但是有点需要注意，就是所有操作完成后要记得关闭流。

当调用FileStrema.open和FileStream.openAsync方法时可通过 FileMode 类的字符串常量设定文件的具体操作方式。 FileMode.WRITE 常量指当打开一个文件流，如果文件不存在则创建文件并写入数据，如果文件存在则覆盖所有数据。 FileMode.APPEND 常量指把缓冲区的数据追加到文件末尾， FileMode.UPDATE 常量则既可以读也可以写文件。所有的写指令当文件不存在时都会创建新文件，当使用 FileMode.READ 时，文件事先必须存在，并把文件数据读取到缓冲区中。

为了读写文件，你需要把File对象指向一个用户电脑中的文件。 File类有一些静态属性和方法对应操作系统文件系统的标准目录以及应用程序目录和应用程序存储目录。

下面的例子使用同步的FileStream.open方法在桌面上写文件：

```
var file:File = File.desktopDirectory.resolvePath( "test.txt" );
var stream:FileStream = new FileStream();
stream.open( file, FileMode.WRITE );
stream.writeUTFBytes( "Hello World" );
stream.close();
```

名为test.txt的文件被创建或打开，使用FileStream.writeUTFBytes方法写入文本Hello World。要读取文件数据，可使用FileStream.readUTFBytes方法：

```
var file:File = File.desktopDirectory.resolvePath( "test.txt" );
var stream:FileStream = new FileStream();
stream.open( file, FileMode.READ );
trace( stream.readUTFBytes( stream.bytesAvailable ) );
stream.close();
```

当使用 FileMode.READ 参数打开文件时， FileStream 对象会立即读取数据到缓冲区，通过 FileStream.bytesAvailable 属性访问缓冲区数据。当使用同步的 FileStream.open 方法时，其他所有操作都暂停直到数据都被放入缓冲区，因为这个操作占据了主应用程序线程。你可以使用异步 FileStream.openAsync 方法代替同步方法打开文件，并监听 progress 和 complete 事件。如果使用 FileMode.WRITE 参数将会覆盖任何其他数据，要检测是否已经有文件存在，可使用 File 类的 exists 属性：

```
var file:File = File.desktopDirectory.resolvePath( "test.txt" );
if( file.exists )
{
    trace( "File created: " + file.creationDate );
    file = File.desktopDirectory.resolvePath( "test2.txt" );
}
var stream:FileStream = new FileStream();
stream.open( file, FileMode.WRITE );
stream.writeUTFBytes( "Hello World" );
```

```
stream.close();
```

FileStream.writeUTFBytes和FileStream.readUTFByte方法只是FileStream类中读写方法的一小部分。

讨论读写文件中，需要特别注意的是当AIR程序安装时所创建的目录的读写权限。在程序安装时有个主要的目录被创建，它们是应用程序目录和应用程序存储目录。File.applicationDirectory和File.applicationStorageDirectory分别指向这两个目录，应用程序目录是只读的，而应用程序存储目录可读写，你可以读取也可以写入需要的文件数据。

## 23.6节. 对象序列化

### 23.6.1. 问题

我想对自定义对象进行序列化并保存到硬盘驱动器的文件中。

### 23.6.2. 解决办法

通过class-alias注册自定义类，使用ActionScript Message Format (AMF) 编码序列化对象，并使用FileStream.writeObject方法把对象存储到文件上。

### 23.6.3. 讨论

使用AIR文件系统API，你可以把经过AMF编码序列化过的对象写入文件流缓冲区中。ActionScript语言中大多数内建对象类型比如String和Boolean都自动支持序列化。这些类型同通过AMF被编码为二进制格式，当经过反序列化可以恢复原来的值。然而自定义对象是不自动支持序列化的，要启动自定义对象序列化支持，你需要用registerClassAlias 方法用一个类别名注册此类，或者在类定义前申明这个[RemoteClass]元数据标签。

假定你的程序要把用户信息作为一个对象进行显示，可过一段时间被同一个程序或者被其他程序载入，这些程序应已知道如何处理该对象，要表现用户信息的自定义对象看起来大概是下面这样：

```
package com.oreilly.flexcookbook
{
    [RemoteClass]
    [Bindable]
    public class UserData
    {
        public var firstName:String;
        public var lastName:String;
        public var age:Number;
        public var sessionLength:Number;

        public function UserData( firstName:String = "",
```

```

        lastName:String = "" )
    {
        this.firstName = firstName;
        this.lastName = lastName;
    }

}
}

```

每次你的程序被用户打开时，一个新的UserData对象被创建，等待用户输入信息，通常可以保存这些信息以特定的格式保存到文本文件上，但是你也可以序列化对象到文件上，从而可轻易的以对象类型读取，不必要解析字符串，重置UserData新实例属性。要序列化对象且通过反序列化保留其属性值，需要使用[RemoteClass]元数据标签注册它。

在类申明前插入[RemoteClass] 元数据标签，可启动此类在使用AMF序列化和反序列化中能保持类型信息和属性值。你也可以在变量定义前加上[Transient]元数据标签，指示此变量不会被序列化。比如上面例子中的sessionLength属性每次程序运行时都会更新，因为没必要进行保存，你可以把此属性标记为transient属性拒绝被序列化：

```

package com.oreilly.flexcookbook
{
    [RemoteClass]
    public class UserData
    {
        public var firstName:String;
        public var lastName:String;
        public var age:Number;

        [Transient]
        public var sessionLength:Number;

        public function UserData( firstName:String = "",
                                lastName:String = "" )
        {
            this.firstName = firstName;
            this.lastName = lastName;
        }
    }
}

```

使用FileStream.writeObject方法把序列化自定义对象写到文件里，当使用writeObject和readObject写入和读取二进制数据默认所用的编码格式是AMF 3 规范。通过设置FileStream.objectEncoding属性，你可以使用AMF 0 格式，下面的例子保存用户信息到扩展名为.user的文件上：

Code View:

```

<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"

```

```

layout="vertical">

<mx:Script>
<! [CDATA[
import com.oreilly.flexcookbook(userData;

private var userData:userData = new userData();
private static const EXT:String = ".user";
private function submitHandler():void
{
    userData.firstName = firstField.text;
    userData.lastName = lastField.text;
    userData.age = ageField.value;

    saveUserData();
}
private function saveUserData():void
{
    var fnm:String = userData.firstName + "_" +
        userData.lastName + EXT;
    var file:File =
        File.desktopDirectory.resolvePath( fnm );
    var stream:FileStream = new FileStream();
    stream.open( file, FileMode.WRITE );
    stream.writeObject( userData );
    stream.close();
}

]]>
</mx:Script>

<mx:Form>
<mx:FormItem label="First Name:">
<mx:TextInput id="firstField"
    change="{submitBtn.enabled = firstField.text != ''}"
/>
</mx:FormItem>
<mx:FormItem label="Last Name:">
<mx:TextInput id="lastField"
    change="{submitBtn.enabled = lastField.text != ''}"
/>
</mx:FormItem>
<mx:FormItem label="Age:">
<mx:NumericStepper id="ageField"

```

```

        minimum="18" maximum="110" />
    </mx:FormItem>
    <mx:Button id="submitBtn" label="submit"
        enabled="false"
        click="submitHandler();"/>
</mx:Form>

</mx:WindowedApplication>

```

当用户输入并提交信息后，submitHandler事件处理器被调用，UserData对象被更新，接着调用saveUserData方法。在saveUserData方法中，用UserData实例的firstName和lastName属性组成文件路径，文件被保存到桌面上，扩展名为.user。当反序列化时，文件被再次打开，对象属性值将被恢复。下面的例子扩展了之前的程序，提供对打开.user文件的支持：

Code View:

```

<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">

    <mx:Script>
        <![CDATA[
            import com.oreilly.flexcookbook.UserData;

            [Bindable]
            private var userData:UserData = new UserData();
            private var file:File = File.desktopDirectory;
            private var filter:FileFilter =
                new FileFilter("User File", "*.user");
            private static const EXT:String = ".user";

            private function submitHandler():void
            {
                userData.firstName = firstField.text;
                userData.lastName = lastField.text;
                userData.age = ageField.value;

                saveUserData();
            }
            private function saveUserData():void
            {
                var fnm:String = userData.firstName + "_" +
                    userData.lastName + EXT;
                var file:File =
                    File.desktopDirectory.resolvePath( fnm );

```

```

        var stream:FileStream = new FileStream();
        stream.open( file, FileMode.WRITE );
        stream.writeObject( userData );
        stream.close();
    }

    private function openHandler():void
    {
        file.browseForOpen( "Open User", [filter] );
        file.addEventListener( Event.SELECT, selectHandler );
    }
    private function selectHandler( evt:Event ):void
    {
        var stream:FileStream = new FileStream();
        stream.open( file, FileMode.READ );
        userData = stream.readObject();
    }
} ]>
</mx:Script>

<mx:Form>
    <mx:FormItem label="First Name:>
        <mx:TextInput id="firstField"
            text="{userData.firstName}"
            change="{submitBtn.enabled = firstField.text != ''}"
        />
    </mx:FormItem>
    <mx:FormItem label="Last Name:>
        <mx:TextInput id="lastField"
            text="{userData.lastName}"
            change="{submitBtn.enabled = lastField.text != ''}"
        />
    </mx:FormItem>
    <mx:FormItem label="Age:>
        <mx:NumericStepper id="ageField"
            value="{userData.age}"
            minimum="18" maximum="110"
        />
    </mx:FormItem>
    <mx:Button id="submitBtn" label="submit"
        enabled="false"
        click="submitHandler();"
    />
</mx:Form>

```

```

<mx:HRule width="100%" />
<mx:Button label="open" click="openHandler();" />

</mx:WindowedApplication>

```

当用户点击open按钮时，File.browseForOpen方法会打开一个对话框，并设置了过滤器只显示.user扩展名的文件。当一个文件被选中后，selectHandler方法被调用，通过FileStream.readObject方法将文件读入到缓冲区，readObject方法将反序列化UserData对象，通过数据绑定将保存的数据显示到表单。

FileStream.writeObject和FileStream.readObject方法提供了很多方法用于序列化数据对象到文件系统上。通过[RemoteClass]元数据标签注册类即可用AMF将类进行编码为二进制格式。如果你是从文本文件中处理载入和解析键--值对自定义对象，那么序列化自定义对象将大大减少开发开销。

## 23.7节. 使用加密的本地存储区

### 23.7.1. 问题

我想把数据存储到用户硬盘上且不可被其他程序所读写。

### 23.7.2. 解决办法

使用AIR 程序中加密的本地存储区以加密存储信息。

### 23.7.3. 讨论

当一个AIR程序被安装后，一个加密的本地存储区被创建，用于存储一些需要保密的信息。使用Windows的Data Protection API (DPAPI) 和 Mac OS X的密匙链for AIR applications on Windows and Keychain for those on Mac OS X，数据被加密且只有在相同的安全沙箱中可用，加密本地存储区最大空间为10MB。

数据被存储在哈希表中，你可以使用键字符串设置和读取加密本地存储区中的数据。数据被序列化为ByteArray对象，这样可以存储大多数内建对象类型以及通过类别名注册的自定义对象。通过flash.data.EncryptedLocalStore类的静态方法访问加密的本地存储区。getItem和setItem方法通过一个键字符串读取相关联的数据，你也可以通过键字符串删除相关联的数据或使用EncryptedLocalStore类的方法清除整个存储区数据。

下面的例子使用EncryptedLocalStore存储用户数据：

Code View:

```

<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical" windowComplete="completeHandler();">
    <mx:Script>

```

```

<! [CDATA[

    import com.carlcalderon.arthropod.Debug;
    import com.lxy.flexcookbook(userData;
    [Bindable]
    public var userData:userData;

    private function submitHandler():void{
        userData =
            new userData(firstField.text,lastField.text);
        var bytes:ByteArray = new ByteArray();
        bytes.writeObject(userData);
        EncryptedLocalStore.setItem("user",bytes);

        views.selectedChild = userCanvas;
    }

    private function completeHandler():void{
        var user:ByteArray =
            EncryptedLocalStore.getItem("user");
        if(user != null){
            userData = user.readObject() as userData;
            views.selectedChild = userCanvas;
        }
    }
]]>
</mx:Script>
<mx:ViewStack id="views" width="300" height="300"
    backgroundColor="0xeeeeee">
    <mx:Form id="inputForm">
        <mx:FormItem label="First Name:>
            <mx:TextInput id="firstField" />
        </mx:FormItem>
        <mx:FormItem label="Last Name:>
            <mx:TextInput id="lastField"/>
        </mx:FormItem>
        <mx:Button label="submit" click="submitHandler();"/>
    </mx:Form>
    <mx:VBox id="userCanvas">
        <mx:Label text="Hello,"/>
        <mx:HBox>
            <mx:Label text="{userData.firstName}"/>
            <mx:Label text="{userData.lastName}"/>
        </mx:HBox>
    </mx:VBox>
</mx:ViewStack>
```

```
</mx:WindowedApplication>
```

当应用程序运行完成初始化布局后，completeHandler方法被调用检测加密本地存储区和user相关的数据，如果有，一个ByteArray对象被返回并被反序列化回溯为UserData对象。如果没有相关数据，则用户可以输入并提交数据到存储区，在submitHandler方法中，用户数据被序列化，存到ByteArray对象，并通过ByteArray.writeObject方法和一个键字符串关联后存储到存储区，通过ByteArray方法任何对象都可被序列化，例如UTF-编码的字符串，Boolean值，numbers，都可以被存储到加密的本地存储区中。

## 23.8节. 浏览本地文件

### 23.8.1. 问题

我想使用打开对话框和保存对话框用于打开和保存文件。

### 23.8.2. 解决办法

使用flash.filesystem.File类的browse开头的那些方法。

### 23.8.3. 讨论

File类提供了一个对话框窗口用于打开一个或多个文件。使用File.browseForOpen方法选择一个文件时 select 事件被触发，使用 File.browseForOpenMultiple 方法选择多个文件时 selecteMutiple事件被触发。

下面的例子打开一个对话框选择一个特定扩展名的文件：

Code View:

```
private var file:File = new File();
private var filter:FileFilter = new FileFilter( "Text", "*.txt;
*.xml; *.html" );

private function initHandler():void
{
    file.browseForOpen( "Open File", [filter] );
    file.addEventListener( Event.SELECT, selectHandler );
    file.addEventListener( Event.CANCEL, cancelHandler );
}

private function selectHandler( evt:Event ):void
{
    var stream:FileStream = new FileStream();
    stream.open( file, FileMode.READ );
    trace( stream.readUTFBytes( stream.bytesAvailable ) );
}
```

```

        stream.close();
    }
private function cancelHandler( evt:Event ):void
{
    trace( "Browse cancelled." );
}

```

一个新的File对象被创建，File.browseForOpen方法打开一个对话框，使用FileFilter类指定特定的文件类型可被选择，每次只能选择一个文件。监听select事件，在selectHandler事件处理函数中，FileStream对象把File实例放置到读取缓冲区中。

File.browseForOpenMultiple方法打开的对话框可以一次选择多个文件，selectMultiple事件被触发，事件类型变成了FileListEvent：

Code View:

```

private var file:File;
private var filter:FileFilter = new FileFilter( "Text", "*.txt;  
*.xml; *.html" );

private function initHandler():void
{
    file = File.desktopDirectory;
    file.browseForOpenMultiple( "Open File", [filter] );
    file.addEventListener( FileListEvent.SELECT_MULTIPLE,
        selectHandler );
}

private function selectHandler( evt:FileListEvent ):void
{
    trace( "Selected files from: " + file.url + "\n" );
    var files:Array = evt.files;
    for( var i:int = 0; i < files.length; i++ )
    {
        trace( ( files[i] as File ).name + "\n" );
    }
}

```

在这个例子中，打开文件对话框被打开并指向用户的桌面目录，选择文件后，selectHandler 被调用并打印出被选择的文件名。

File类也支持保存对话框，在保存对话框里，用户选择一个目录，输入文件名保存文件。下面的例子打开保存文件对话框，写入文本Hello World到目标文件：

```
private var file:File;
```

```

private function initHandler():void
{
    file = File.desktopDirectory;
    file.browseForSave( "Save As" );
    file.addEventListener( Event.SELECT, selectHandler );
}

private function selectHandler( evt:Event ):void
{
    var stream:FileStream = new FileStream();
    stream.open( evt.target as File, FileMode.WRITE );
    stream.writeUTFBytes( "Hello World." );
    stream.close();
}

```

Browse系列方法会适应不同的操作系统，对于用户选择和保存文件提供了极大的方便。

## 23.9节. 使用File System控件

### 23.9.1. 问题

我想添加控件用于浏览和显示文件系统特定目录下的内容。

### 23.9.2. 解决办法

使用SDK的文件系统控件。

### 23.9.3. 讨论

Flex 3 SDK的AIR API提供了很多控件方便浏览电脑文件系统的目录。 这些控件只可用在AIR程序中，是由Framework的基于列表的组件组合而成。如Tree, List, 和DataGrid。虽然这些文件系统控件的外观和交互和Flex组件复本是一样的，但是这些控件的数据内容是由directory属性提供而非传统的dataProvider属性。

下面的例子使用FileSystemComboBox和FileSystemList组件浏览和显示计算机文件系统：

Code View:

```

<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical" windowComplete="initHandler();">
    <mx:Script>
        <! [CDATA[
            import mx.events.FileEvent;
            private function initHandler():void{
                fileCB.directory = File.documentsDirectory;
            }
        
```

```

private function changeHandler(e:FileEvent):void{
    trace(e.file.nativePath);
}
]]>
</mx:Script>
<mx:FileSystemComboBox id="fileCB"
    directory="{fileList.directory}"
    directoryChange="changeHandler(event);" />
<mx:FileSystemList id="fileList"
    directory="{fileCB.directory}" />
</mx:WindowedApplication>

```

当点击mx.controls.FileSystemComboBox控件的下拉按钮时会显示目录的层级结构，程序载入初始化布局完成后，组合框初始目录为用户的文档目录，如果你点击下拉按钮，你会看到整个目录的层级结构。不过FileSystemComboBox并没有显示文档目录的所有子目录，比选中的目录层级还低的目录是不会被显示的。

在这个例子中，当FileSystemComboBox控件选中一个目录时，这个目录的内容就会在FileSystemList控件中显示出来，并且changeHandler方法被调用directoryChange事件传递的事件对象是FileEvent对象。FileEvent对象的file属性指出当前选中的目录，且目录路径被打印到控制台。FileSystemComboBox 和FileSystemList的directory属性进行双向绑定，当前目录发生变化时每个组件都会被更新。

除了可显示隐藏文件和特定扩展名文件外，FileSystemList类还有些属性用于浏览历史记录，你可以使用FileSystemHistoryButton类启动历史记录导航功能。

下面的例子使用FileSystemHistoryButton组件来轻松导航到以前选择过的目录：

Code View:

```

<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">

    <mx:HBox width="100%">
        <mx:FileSystemHistoryButton label="Back"
            dataProvider="{fileList.backHistory}"
            enabled="{fileList.canNavigateBack}"
            click="fileList.navigateBack();"
            itemClick="fileList.navigateBack(event.index)" />
        <mx:FileSystemHistoryButton label="Forward"
            dataProvider="{fileList.forwardHistory}"
            enabled="{fileList.canNavigateForward}"
            click="fileList.navigateForward();"
            itemClick="fileList.navigateForward(event.index)" />
    </mx:HBox>

```

```

<mx:FileSystemList id="fileList"
    width="100%" height="250"
    directory="{File.documentsDirectory}" />

</mx:WindowedApplication>

```

每个历史按钮的数据提供者是之前选择过的目录，一个File对象数组。FileSystemList有内部的管理方法更新backHistory, forwardHistory, canNavigateBack, 和canNavigateForward 属性。你可以从FileSystemHistoryButton 组件的下拉按钮中选择FileSystemList 中特定的目录。FileSystemHistoryButton 的itemClick事件触发的对象实例是一个MenuEvent。你可以使用事件对象的index属性进行浏览历史定位以及更新FileSystemList组件的目录显示列表。

虽然FileSystemList控件可显示指定目录下的文件和目录，但是它不能显示目录在文件系统中的层级关系。这个就要通过FileSystemTree控件来做了。FileSystemTree类可以指示是否显示隐藏文件以及特定扩展名的文件。File对象只表示目录，还有其他过滤选项以及如何自定义浏览和导航目录。FileSystemTree组件相关事件有directoryClosing, directoryOpening事件和directoryChange事件。

下面的例子使用FileStream类显示文件系统根目录：

```

<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">

    <mx:FileSystemTree id="fileTree"
        width="100%" height="100%"
        directory="{FileSystemTree.COMPUTER}" />

</mx:WindowedApplication>

```

FileSystemTree 控件可以很方便的从目录层级中浏览目录。FileSystemTree组件显示的每一项标签是文件或目录的名称。要想显示文件或目录的更多信息，就需要FileSystemDataGrid类。FileSystemDataGrid控件通过DataGridColumn实例自动显示文件名，类型，大小，创建时间，修改时间。使用 FileSystemDataGrid控件，双击某一项浏览目录。下面的例子使用FileSystemDataGrid控件浏览文件系统的桌面目录：

```

<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">

    <mx:FileSystemDataGrid id="fileGrid"
        width="100%" height="100%"
        directory="{File.desktopDirectory}" />

</mx:WindowedApplication>

```

双击某一项就会刷新列表显示被选择目录下的文件和目录。这个动作会导致进入目录并显示

其内容，但是组件并没有提供其他控件使用户返回到先前的目录。和FileSystemList一样，FileSystemDataGrid内部管理着浏览历史记录，你仍可以使用FileSystemHistoryButton类浏览历史记录。

使用AIR API的文件系统控件可方便的浏览计算机文件系统。但是需要注意的是AIR并不会对文件系统通知做出反应，也就是说如果文件和目录被删除，则AIR控件不会自动刷新。要确保不会对已删除的文件和目录进行操作，记得先调用FileSystemList, FileSystemTree, 和FileSystemDataGrid控件的refresh方法。

## 23.10节. 使用本地拖拽(Drag-and-Drop) API

### 23.10.1. 问题

我想在应用程序内外拖动数据。

### 23.10.2. 解决办法

添加数据到剪贴板，使用NativeDragManager类管理拖拽操作。

### 23.10.3. 讨论

通过本地的drag-and-drop API文件系统和AIR应用程序之间可以进行数据传输。当一个拖拽姿势启动后，指定格式的数据被添加到剪贴板并传递给NativeDragManager的doDrag方法。你可以注册事件监听器监听NativeDragManager类拖拽操作发出的completion事件。事件对象类型是NativeDragEvent类实例，通过NativeDragEvent.clipboard属性可访问剪贴板数据。

当用户用鼠标选择了应用程序的某一个元素，即一个拖拽姿势被启动。当用户按住鼠标，操作进入“拖阶段”，所有继承自flash.display.InteractiveObject类的注册组件都可接受拖拽动作。当用户松开鼠标即表示拖拽姿势完成。启动拖拽姿势的组件被认为是“拖动源”，接受拽操作的InteractiveObject实例被认为是“拖拽目标”。

Flex Framework本身就支持应用程序内的拖拽操作，而本地的拖拽API运行在文件系统和AIR程序之间拖拽数据。我们推荐使用Flex FrameWork拖拽API。如果你熟悉mx.managers.DragManager类在Flex程序内传输数据的话，那么你会发现flash.desktop.NativeDragManager类与此非常相似。有个重要方面需要注意的是两个类的DragSource对象被添加到DragManager实例中，而Clipboard对象被添加到NativeDragManager实例中。

下面的例子使用NativeDragManager添加一个图片到剪贴板，并使用拖拽姿势传输数据到文件系统的一个目录：

Code View:

```
<mx:WindowedApplication  
    xmlns:mx="http://www.adobe.com/2006/mxml"  
    layout="vertical"  
    windowComplete="initHandler();"
```

```

closing="closeHandler();">

<mx:Script>
<! [CDATA[
import mx.graphics.codec.PNGEncoder;

private var tempDir:File = File.createTempDirectory();
private var imageData:BitmapData;

private function initHandler():void
{
    imageData = new BitmapData( image.width,
        image.height );
    imageData.draw( image );
}

private function closeHandler():void
{
    tempDir.deleteDirectory();
}

private function clickHandler():void
{
    var transfer:Clipboard = new Clipboard();
    transfer.setData( ClipboardFormats.FILE_LIST_FORMAT,
        [getImageFile()], false );

    NativeDragManager.dropAction =
        NativeDragActions.COPY;
    NativeDragManager.doDrag(this,transfer,imageData);
}

private function getImageFile():File
{
    var tempFile:File = tempDir.resolvePath( "img.png" );

    var png:ByteArray =
        new PNGEncoder().encode( imageData );
    var stream:FileStream = new FileStream();
    stream.open( tempFile, FileMode.WRITE );
    stream.writeBytes( png );
    stream.close();

    return tempFile
}

```

```

        }

    ]]>
</mx:Script>

<mx:Image id="image"
    source="@Embed(source='assets/bigshakey.png')"
    buttonMode="true" useHandCursor="true"
    mouseDown="clickHandler();"
/>

</mx:WindowedApplication>

```

当Image控件的mouseDown 事件触发时，clickHandler方法被调用，Clipboard对被创建。 使用Clipboard.setData方法创建被移动的数据副本。 Clipboard对象的数据格式是一个文件列表：里面只有一个图像文件，放置在文件系统的临时目录里。当用户拖动图片离开应用程序到文件系统的任何目录，松开鼠标按钮，图片文件便会被移动到目标位置。

你还可以拖动数据到另一个程序中。要让你的程序接受本地拖拽操作。需要监听 NativeDragManager 类发出的拖拽动作事件。下面的例子是一个接受图片文件的应用程序：

Code View:

```

<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    windowComplete="initHandler();">

    <mx:Script>
        <! [CDATA[
            import mx.controls.Image;

            private var loader:Loader;
            private var xposition:Number;
            private var yposition:Number;

            private function initHandler():void
            {
                addEventListener( NativeDragEvent.NATIVE_DRAG_ENTER,
                    dragEnterHandler );
                addEventListener( NativeDragEvent.NATIVE_DRAG_DROP,
                    dragDropHandler );
            }

            private function dragEnterHandler( evt:NativeDragEvent
                ):void
            {

```

```

if( evt.clipboard.hasFormat( ClipboardFormats.FILE_LIST_FORMAT ) )
    NativeDragManager.acceptDragDrop( this );
}

private function dragDropHandler( evt:NativeDragEvent
) :void
{
    var pt:Point = globalToLocal(new Point( evt.localX,
evt.localY ));
    xposition = pt.x;
    yposition = pt.y;
    var files:Array = evt.clipboard.getData(
        ClipboardFormats.FILE_LIST_FORMAT ) as Array;
    loader = new Loader();
    loader.contentLoaderInfo.addEventListener(
        Event.COMPLETE, completeHandler );
    loader.load( new URLRequest( files[0].url ) );
}

private function completeHandler( evt:Event ) :void
{
    var bmp:Bitmap = loader.content as Bitmap;
    var image:Image = new Image();
    image.source = bmp;
    image.x = xposition;
    image.y = yposition;
    addChild( image );
}

] ] >
</mx:Script>

</mx:WindowedApplication>

```

当应用程序初始化完成后，开始注册监听nativeDragEnterEvent和nativeDragDropEvent事件。当一个文件被拖到程序上，调用NativeDragManager.acceptDragDrop方法。当dragDropEvent动作发生后，移动的数据被认为是一组文件对象，数组中的第一个文件被载入到Image控件。虽然本节的例子演示了使用NativeDragManager 使AIR程序如何和文件系统交换数据，但是并不意味着只能是文件对象，你还可以交换位图数据和格式化字符串如HTML, URL, 或者是简单文本。

### 23.11节. 与操作系统剪贴板交互

### 23.11.1. 问题

我想与操作系统剪贴板进行数据交换。

### 23.11.2. 解决办法

使用Clipboard类的静态generalClipboard属性。

### 23.11.3. 讨论

除了通过拖拽操作和系统剪贴板交换数据外，还可通过flash.desktop.Clipboard类的静态属性generalClipboard与剪贴板交换数据。AIR程序支持的数据交换格式有：位图数据，标准文本数据，HTML，和URL格式。当数据可用时，可分别被转换为BitmapData对象，File对象数组和String对象。你还可以用自定义数据与Clipboard对象进行交换，只不过这些数据只能在知道如何处理它的其他AIR程序中可用。

在下面的例子中，你可以添加，删除，访问操作系统剪贴板中的数据：

## Code View:

```

        ClipboardFormats.TEXT_FORMAT      )
    as String;
}

]

]]>
</mx:Script>

<mx:TextArea id="textField"
    width="100%" height="100%" />
<mx:Button label="add to clipboard"
    click="addHandler();" />
<mx:Button label="remove from clipboard"
    click="removeHandler();" />
<mx:Button label="past from clipboard"
    click="pasteHandler();" />

</mx:WindowedApplication>

```

addHandler事件处理函数把<mx:TextArea>控件中显示的内容传送到系统剪贴板。这样你就可以粘贴这些数据到其他非AIR应用程序中了。其他程序拷贝到剪贴板的String数据也可以粘贴到<mx:TextArea>控件中。在pasteHandler方法中，先使用hasFormat方法检测剪贴板中的数据格式，如果是String数据，则被转换为ActionScript String对象并粘贴到组件。removeHandler方法中使用clear方法清除剪贴板。

关于数据格式类型，其实并不局限于ClipboardFormats类所列出的那几种，你可以指定任何可被多个AIR程序所识别的字符串值，比如可能有两个或更多程序处理用户信息，要利用系统剪贴板在应用程序直接共享这些信息，你可以用类似下面的代码把数据放置到剪贴板上：

Code View:

```
Clipboard.generalClipboard.setData( "userObject", new UserObject( 'Ted', 'Henderson' ) );
```

任何知道如何处理userObject格式的AIR应用程序都可以访问和处理剪贴板中的UserObject实例对象。

## 23.12节. 添加HTML内容

### 23.12.1. 问题

我想在应用程序中显示HTML内容。

### 23.12.2. 解决办法

使用<mx:HTML>控件载入并显示HTML内容。

### 23.12.3. 讨论

Adobe AIR 运行时支持在基于SWF的应用程序中渲染HTML。渲染引擎是基于WebKit技术构建，功能和基于WebKit引擎的浏览器一样。比如Safari。这个引擎的核心就是AIR HTML API 中的flash.html.XMLLoader类，它继承自flash.display.Sprite类，XMLLoader对象可被添加到基于ActionScript和基于Flex的程序中。

该API中还包括<mx:HTML>控件，它可以轻松的将HTML内容显示到Flex框架的容器中。mx.controls.HTML类内部的接口是XMLLoader类实例并提供对HTML Document Object Model (DOM)的访问。这一能力类似于HTML页面中操作DOM和CSS的JavaScript。你可以无限制的载入远程服务器上HTML页面，也可以载入本地沙箱中的HTML页面甚至是HTML格式的字符串。

通过location属性值将HTML页面载入到<mx:HTML>控件，location字符串被传给内部的XMLLoader实例，它才载入和渲染所需的页面。下面的例子载入web网页到<mx:HTML> 控件，还有些控件用于导航到其他页面：

Code View:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">

    <mx:Script>
        <! [CDATA[

            [Bindable]
            public var urlLocation:String = "http://www.adobe.com";

        ]]>
    </mx:Script>

    <mx:Form width="100%">
        <mx:FormItem width="100%">
            <mx:HBox width="100%">
                <mx:TextInput id="urlField"
                    width="100%"
                    text="{html.location}" />
                <mx:Button label="go"
                    click="{urlLocation = urlField.text}"
                    />
            </mx:HBox>
        </mx:FormItem>
    </mx:Form>
</mx:WindowedApplication>
```

```

</mx:Form>
<mx:HTML id="html"
    width="100%" height="100%"
    location="{urlLocation}"
/>

</mx:WindowedApplication>

```

这个应用程序从 <http://www.adobe.com> 载入 web 页面作为开始页并提供控件通过改变 <mx:HTML> 控件的 location 属性值导航到其他 HTML 页面。页面载入后，你可以跟用其他浏览器那样与 HTML 内容进行操作。HTMLLoader 实例也是在内部对浏览历史进行管理，并提供一个简单的方式浏览历史记录。下面的例子添加控件用于向前或向后浏览历史记录：

Code View:

```

<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">
    <mx:Script>
        <![CDATA[
            [Bindable]
            public var urlLocation:String = "http://www.adobe.com";
        ]]>
    </mx:Script>
    <mx:Form width="100%">
        <mx:FormItem width="100%">
            <mx:HBox width="100%">
                <mx:Button label="back"
                    click="html.historyBack();"/>
                <mx:Button label="forward"
                    click="html.historyForward();"/>
                <mx:TextInput id="urlField"
                    width="100%"
                    text="{html.location}"/>
                <mx:Button label="go"
                    click="{urlLocation = urlField.text}"/>
            </mx:HBox>
        </mx:FormItem>
    </mx:Form>
    <mx:HTML id="html"
        width="100%" height="100%"
        location="{urlLocation}"/>
</mx:WindowedApplication>

```

HTML.historyBack和HTML.historyForward方法用于导航HTMLLoader对象中的历史记录。如果你熟悉Ajax应用程序的构建，那HTMLLoader类的历史记录管理非常类似于window.history JavaScript对象。除了这两个方法外，你还可以访问历史记录数多少，以及当前所处的位置，以及用historyGo方法直接载入指定的历史记录。从这个例子中不难发现，开发一个基于SWF的个性化Web浏览器是如此简单。

这里还有些HTMLLoader实例需要关注的事件，需要注意的是要搞清楚这些事件的发生顺序：

```
<mx:HTML id="html"
    width="100%" height="100%"
    location="http://www.adobe.com"
    htmlDOMInitialize="initHandler();"
    locationChange="changeHandler();"
    complete="completeHandler();"
    htmlRender="renderHandler();"
/>
```

locationChange 事件是在内部HTMLLoader实例的 location 属性更新后触发的，htmlDOMInitialized事件是在文档被创建且你与HTML DOM交互之前触发的，complete事件触发后，你就可以操作HTML DOM了。而htmlRender是在渲染HTML内容时触发的。

## 23.13节. 在ActionScript和JavaScript之间跨脚本操作

### 23.13.1. 问题

我想访问HTML元素节点，访问JavaScript变量和函数，操作CSS 样式。

### 23.13.2. 解决办法

监听complete事件，使用<mx:HTML>控件的domWindow属性访问HTML DOM。

### 23.13.3. 讨论

HTMLLoader类支持访问HTML文档的DOM对象。你可以访问HTML的节点元素，也可以和页面里的javascript交互，访问其变量和方法，也可以在JavaScript中调用ActionScript方法。这种在JavaScript和ActionScript直接的调用叫跨脚本。

通过<mx:HTML>控件的domWindow属性可以访问HTML文档的全局JavaScript对象。domWindow属性是一个通用的对象类型，在其他ActionScript对象上通过点运算符访问HTML DOM的属性。使用domWindow属性，你可以访问HTML节点元素，JavaScript变量和函数以及任何CSS样式表。假定下面的HTML文档将被载入HTML控件中：

```
<html>
<body>
<p id="helloField">Hello World</p>
```

```

</body>
</html>

```

这是个简单的页面，只显示文本Hello World。当内容完全载入后就可以getElementId方法访问helloField元素，就像JavaScript代码那样：

Code View:

```

<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">

    <mx:Script>
        <![CDATA[

            private function completeHandler():void
            {
                var p1:String =
                    html.domWindow.document.getElementById('helloField').innerHTML;
                trace( p1 );
            }
        ]]>
    </mx:Script>

    <mx:HTML id="html"
        width="100%" height="100%"
        location="test.html"
        complete="completeHandler();"
    />

</mx:WindowedApplication>

```

当内容被完全载入，且DOM可用时Complete事件从<mx:HTML>控件的HTMLLoader实例内部发出。在completeHandler方法内，应用程序访问test.html文档的helloField元素节点值并打印字符串到调试控制台上。<mx:HTML>控件的domWindow属性是HTML DOM window的ActionScript表示形式。访问载入的HTML文档并不局限于只读权限，你还可以像下面那样改变helloField元素的文本值：

```
html.domWindow.document.getElementById('helloField').innerHTML = "Hola!";
```

使用styleSheets属性可访问和操作CSS样式表。styleSheets属性是一个数组对象，根据层叠样式表的申明顺序而创建。下面的代码中helloField元素加入CSS样式：

```

<html>
    <style>
        #helloField {

```

```

        font-size: 24px;
        color: #FF0000;
    }
</style>
<body>
    <p id="helloField">Hello World</p>
</body>
</html>

```

通过styleSheets属性访问和操作样式表：

```

var styleSheets:Object = html.domWindow.document.styleSheets;
trace( styleSheets[0].cssRules[0].style.fontSize );
styleSheets[0].cssRules[0].style.color = "#FFCCFF";

```

CSS样式表的每个样式申明都被放置在一个对象数组中，通过cssRules属性访问。可以通过属性访问和更新样式。

注意：

对样式的修改并不会保存在HTMLLoader实例的历史记录中。如果<mx:HTML>控件的location属性被更新，即使使用HTML.historyBack方法，之前所作的任何改变也不会被恢复。

使用domWindow属性还可以访问HTML文档的JavaScript变量和函数。

```

<html>
    <script>
        var age = 18;
        function setAge( num )
        {
            age = num;
            handleAgeChange( age );
        }

        function sayHello()
        {
            return "Hello";
        }
    </script>
    <body>
        <p id="helloField">Hello World</p>
    </body>
</html>

```

你可能已经注意到在setAge方法内调用了一个不存在的函数handleAgeChange。虽然ActionScript方法可以被指定为在HTML DOM的JavaScript函数调用的委托，但是如果setAge方法被调用，既没有JavaScript函数也没有ActionScript函数委托，将会抛出

ReferenceError异常。同样道理，如果domWindow 属性调用了一个不存在的JavaScript对象，也会抛出错误。这使得ActionScript 和JavaScript之间的桥接成为一个依赖。但是这个属性是一个通用的对象类型，松耦合的引入使得应用程序不会依赖于特定的HTML文档。

下面的例子载入test.html文档，更新age属性，调用sayHello方法：

Code View:

```
<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">

    <mx:Script>
        <![CDATA[

            private function completeHandler():void
            {
                trace( html.domWindow.sayHello() );

                html.domWindow.handleAgeChange = ageHandler;
                trace( "Current Age: " + html.domWindow.age );
                html.domWindow.setAge( 30 );
            }

            private function ageHandler( value:Object ):void
            {
                trace( "Age changed = " + value );
            }
        ]]>
    </mx:Script>

    <mx:HTML id="html"
        width="100%" height="100%"
        location="test.html"
        complete="completeHandler();"
    />

</mx:WindowedApplication>
```

当setAge JavaScript方法被调用时并没有抛出异常，因为handleAgeChange函数已经委托给ActionScript方法ageHandler。这样你可以在JavaScript和ActionScript之间传递任何类型的数据，不过如果不是简单类型(比如Date对象)，则必须转换成适当的类型。

## 23.14节. 本地SQL数据库

### 23.14.1. 问题

我想让应用程序在本地保存和接收数据。

### 23.14.2. 解决办法

在用户硬盘上创建数据库文件，执行SQL语句。

### 23.14.3. 讨论

Adobe AIR 运行时引入了SQL数据库引擎，使我们可以为创建本地数据库存储信息。一个数据库被保存为一个文件，且没有限定存放于特定的目录下，这样可允许任何应用程序都可访问数据库中的数据。AIR的SQL引擎可创建关系型数据库，用标准SQL语句存储和接受复杂数据。

SQL 数据库API 包含几个类，提供创建和打开数据库，生成语句，监听操作事件，检索有关数据库的架构信息。使用 `flash.filesystem.File` 类用合法的扩展名创建数据库文件。使用 `Flash.dataSQLConnection` 类的 `open` 和 `openAsync` 方法建立一个数据库连接。如果你没有传递 `File` 引用给 `open` 和 `openAsync` 方法，则会在内存中创建一个数据库并允许执行SQL语句。SQL语句可被同步和异步执行。在执行SQL语句前可监听成功和失败相关事件，只不过异步执行操作是在主程序线程之外，这样在SQL执行时可以运行其他代码。

下面的例子演示在数据库存在时打开数据库，反之创建新的数据库文件：

```
var db:File =
    File.applicationStorageDirectory.resolvePath("Authors.db");

var sqlConn:SQLConnection = new SQLConnection();
sqlConn.addEventListener( SQLEvent.OPEN, openHandler );
sqlConn.addEventListener( SQLErrorEvent.ERROR, errorHandler );
sqlConn.openAsync( db );

private function openHandler( evt:SQLEvent ):void
{
    trace( "Database created." );
}

private function errorHandler( evt:SQLErrorEvent ):void
{
    trace( "Error " + evt.error.message + " :: " +
        evt.error.details );
}
```

如果之前没有创建过Authors.db，则`SQLConnection`实例的`openAsync`方法会在应用程序存储目录创建这个数据库文件。如果操作成功，一个`SQLEvent`对象发出并调用`openHandler`方法。如果操作中出现错误，一个`SQLErrorEvent`被发出。`SQLErrorEvent`对象的`error`属性是一个

SQLError对象继承自flash.errors.Error类。

要执行SQL语句，先赋值申明的SQL字符串值传递给SQLStatement的text属性并调用execute方法。下面的例子使用SQL语言的CREATE TABLE语句在数据库里创建一个新表：

Code View:

```
var db:File =
    File.applicationStorageDirectory.resolvePath( "Authors.db" );

var sqlConn:SQLConnection = new SQLConnection();
sqlConn.addEventListener( SQLEvent.OPEN, openHandler );
sqlConn.addEventListener( SQLErrorEvent.ERROR, errorHandler );
sqlConn.openAsync( db );

private function openHandler( evt:SQLEvent ):void
{
    var sql:String = "CREATE TABLE IF NOT EXISTS authors (" +
                    "authorId      INTEGER      PRIMARY KEY," +
                    "firstName      TEXT        NOT NULL," +
                    "lastName       TEXT        NOT NULL" +
                    ");";

    var statement:SQLStatement = new SQLStatement();
    statement.sqlConnection = sqlConn;
    statement.text = sql;
    statement.addEventListener( SQLEvent.RESULT, resultHandler );
    statement.addEventListener( SQLErrorEvent.ERROR,errorHandler );
    statement.execute();
}

private function resultHandler( evt:SQLEvent ):void
{
    trace( "Table created." );
}

private function errorHandler( evt:SQLErrorEvent ):void
{
    trace( "Error " + evt.error.message +
          " :: " + evt.error.details );
}
```

当数据库连接被打开时，SQL语句被执行，创建了一个新的authors表，列名分别为authorId, firstName, 和lastName。SQLStatement对象通过其sqlConnection属性获得SQLConnection实例，在执行之前要注册相应的事件监听器。因为这里使用的是异步的openAsync方法，这样在执行异步方法时，其他操作也会继续被执行，不会被阻塞。

使用SQL声明性语言执行查询，除了CREATE TABLE 命令外，还有INSERT, SELECT,

UPDATE, 和DELETE查询语句。下面的代码执行INSERT语句添加数据：

```
private var insertQuery:SQLStatement = new SQLStatement();

private function addAuthor( fName:String, lName:String ):void
{
    var sql:String = "INSERT INTO authors VALUES (" +
        "null," +
        "'" + fName + "'," +
        "'" + lName + "'"+
    ")";

    insertQuery.sqlConnection = sqlConn;
    insertQuery.text = sql;
    insertQuery.addEventListener( SQLEvent.RESULT, insertHandler );
    insertQuery.addEventListener( SQLErrorEvent.ERROR,
        errorHandler );
}

insertQuery.execute();
}

private function insertHandler( evt:SQLEvent ):void
{
    var result:SQLResult = insertQuery.getResult();
    trace( "Row ID : " + result.lastInsertRowID + " / " +
        "# Rows Affected : " + result.rowsAffected );
}
```

使用INSERT INTO SQL语句Author数据被添加到authors数据库表中。如果操作成功，insertHandler方法被调用，SQLStatement实例的getResult方法收到SQLResult对象。

要简化和增强执行查询语句的性能，SQLStatement类提供了一个parameters属性和itemClass属性。像这里的情况，一个同样的Insert语句可能使用不同的插入内容调用很多次，这时候就可以使用SQLStatement的parameters属性，parameters属性是一个关联数组，使用命名的和未命名的参数存储键---值对。

下面的例子使用命名参数：

```
var insertSql:String = "INSERT INTO authors VALUES (" +
    "null,:firstName,:lastName);";
insertQuery.sqlConnection = sqlConn;
insertQuery.text = insertSql;
insertQuery.parameters[":firstName"] = fName;
insertQuery.parameters[":lastName"] = lName;
```

当语句被执行前，这些值被替换，你可以在属性名前使用`:`或`@`符号，也可以是数字索引作为属性名，SQL语句里用`?`取代：

```

var insertSql:String = "INSERT INTO authors VALUES (" +
    "null,?,?);";

insertQuery.sqlConnection = sqlConn;
insertQuery.text = insertSql;
insertQuery.parameters[1] = lName;

```

当执行语句时每个索引下的值会替换掉SQL语句中的? 符号。这样做的好处是不仅提高了执行效率，而且提高了抵御如SQL注入等恶意攻击。

如果你的程序有个数据对象关联到本地数据库的表数据，那么通过SQLStatement的itemClass属性可以很方便的映射SELECT查询获得的行数据结果。下面的例子代码使用itemClass属性映射com.oreilly.flexcookbook.Author类返回的author数据：

```

import com.oreilly.flexcookbook.Author;

private var selectQuery:SQLStatement = new SQLStatement();

private function getAuthors():void
{
    var sql:String = "SELECT authorId, firstName, lastName FROM
authors";
    selectQuery.sqlConnection = sqlConn;
    selectQuery.text = sql;
    selectQuery.itemClass = Author;
    selectQuery.addEventListener( SQLEvent.RESULT, selectHandler );
    selectQuery.addEventListener( SQLErrorEvent.ERROR,
        errorHandler );

    selectQuery.execute();
}

private function selectHandler( evt:SQLEvent ):void
{
    var authors:Array = selectQuery.getResult().data;
    for( var i:int = 0; i < authors.length; i++ )
    {
        var author:Author = authors[i] as Author;
        trace( author.firstName + " " + author.lastName );
    }
}

```

当查询成功时，selectHandler方法被调用，SQLStatement实例的getResult方法得到SQLResult对象，SQLResult对象的数据属性是一个数据库行记录数组，数组中的每个元素被映射为Author对象，因为Author类已提供给itemClass属性。

关于SQL语言已经有很多书可以查阅，本节重点介绍在本地数据库执行查询语句。

## 23.15节. 检测和监控网络连接

### 23.15.1. 问题

我想检测Internet连接和监视器是否可用。

### 23.15.2. 解决办法

使用AIR monitoring API的ServiceMonitor, SocketMonitor, 和URLMonitor类。

### 23.15.3. 讨论

Adobe AIR包含的类可检测网络资源是否可用，还有当连接改变时实时通知用户。这些类使得开发人员能够创建支持偶尔连接(occasional connectivity)的应用程序：当资源可用时，通过联机服务形成无缝的工作环境，当资源不可用时本地数据可被保存和读取。本地数据包括序列化对象：来自不同的文件，或加密的本地存储区或本地数据库。

你可以监听NativeApplication实例发出的networkChange事件判断网络连接是否改变。当连接可用或不可用时触发该事件，事件本身并没有很多关于连接的相关信息。因此，你需要使用事件处理器检测应用程序可工作在请求服务模式。

下面的例子为networkChange事件创建事件监听器：

```
<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    networkChange="networkChangeHandler() ;">

    <mx:Script>
        <! [CDATA[
            private function networkChangeHandler( evt:Event ):void
            {
                // check connection
            }
        ]]>
    </mx:Script>

</mx:WindowedApplication>
```

这个例子中<mx:WindowedApplication>根标签申明networkChange事件处理器，当网络连接发生变化时即会调用此事件处理器。 应用程序启动不会立即去检查网络资源，因此如果有需要，你需要执行所需的操作以检查是否连接已可用。

networkChange事件提示应用程序只有在网络发生变化时才会发生，而不管所需服务是否可用。根据这个需求，你可以使用SocketMonitor和URLMonitor类检测所需服务是否可用。

ServiceMonitor类是所有监视类的基类，提供了简便的方法轮询服务的可用性。要检测HTTP连接是否改变，你可以使用URLMonitor对象，它是ServiceMonitor类的子类。下面的代码通过HTTP头检测web站点是否可用：

```
private var monitor:URLMonitor;

private function startMonitor():void
{
    var req:URLRequest = new URLRequest( "http://www.adobe.com" );
    req.method = URLRequestMethod.HEAD;

    monitor = new URLMonitor( req );
    monitor.pollInterval = 30000;
    monitor.addEventListener( StatusEvent.STATUS, statusHandler );
    monitor.start();
}

private function statusHandler( evt:StatusEvent ):void
{
    trace( "Available: " + monitor.available );
    trace( "Event code: " + evt.code );
}
```

URLMonitor 对象的pollInterval属性值单位为毫秒。这个例子中，服务每30秒查询次HTTP头，此服务一直持续下去，但是只有在第一次查询和网络连接发生变化时才会发出StatusEvent事件。这个例子中，startMonitor方法被调用后30秒statusHandler方法被调用，以后只有在URLMonitor实例的available属性发生变化时才会被调用，例如网络连接改变。

使用SocketMonitor实例检测socket连接和检测HTTP是类似的，只不过需要主机和端口参数：

```
socketMonitor = new SocketMonitor( "www.adobe.com", 1025 );
socketMonitor.addEventListener( StatusEvent.STATUS, statusHandler );
socketMonitor.start();
```

## 23.16节. 检测用户是否在线

### 23.16.1. 问题

我想检测用户是否在线。

### 23.16.2. 解决办法

设置NativeApplication的idleThreshold属性，监听userIdle和userPresent事件。

### 23.16.3. 讨论

检测用户在线的原理是基于键盘和鼠标的活动状态，离线是指键盘和鼠标在一段时间内不处于活动状态。你可以设置一个时间期限判断用户是否在线。

下面的例子使用NativeApplication实例检测用户在线：

Code View:

```
<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    windowComplete="completeHandler();">

    <mx:Script>
        <![CDATA[

            private function completeHandler():void
            {
                NativeApplication.nativeApplication.idleThreshold =
                    10;

                NativeApplication.nativeApplication.addEventListener(
                    Event.USER_IDLE, idleHandler );
            }

            NativeApplication.nativeApplication.addEventListener(
                Event.USER_PRESENT, presenceHandler );
            }

            private function idleHandler( evt:Event ):void
            {
                trace( "Hello?!?!" );
            }

            private function presenceHandler( evt:Event ):void
            {
                trace( "Welcome Back!" );
            }
        ]]>
    </mx:Script>

</mx:WindowedApplication>
```

当应用程序运行并完成初始化后，completeHandler方法被调用，它设置阈值，创建presence事件监听器。idleThreshold属性单位为秒，10秒后用户没动作，则idleHandler方法被调用，处于空闲状态后当鼠标和键盘事件再次被检测到时，presenceHandler被调用欢迎用户回来。使用presence事件的优点就是可以证明当前正在电脑前。

## 23.17节. 创建系统托盘图标

### 23.17.1. 问题

我想让应用程序运行在后台，不需要主界面。

### 23.17.2. 解决办法

在<mx:WindowedApplication>根标签和描述文件里设置应用程序的可见性为 false，使用 [DockIcon](#) 和 [SystemTrayIcon](#) 类，添加自定义程序图标。

### 23.17.3. 讨论

是可以创建没有主界面的应用程序，且运行在后台。这些应用程序出现在系统托盘或停靠栏里。Mac OS X 和Windows操作系统都支持应用程序图标，只是规定有些差别。因此AIR 针对不同操作系统提供图标显示类。DockIcon类是运行在Mac OS X系统下而SystemTrayIcon类是在Windows下。要检测哪一个图标被操作系统所支持，你可以使用NativeApplication 类的 supportsDockIcon 和supportsSystemTrayIcon属性。

DockIcon 和 SystemTrayIcon 类都是继承自 flash.desktop.InteractiveIcon 抽象基类。NativeApplication实例的icon属性是指向操作系统支持的应用程序图标类引用。你可以赋值图形给icon的bitmaps属性。Bitmaps数组里的元素都是BitmapData对象，其大小是操作系统所规定的大小。如果bitmaps属性为空，则Mac OS X下默认的图标会被使用而Windows系统下不会显示图标在系统托盘上。

除了可以添加自定义应用程序图标外，你还可以添加当用户点击图标时显示的本地的上下文菜单。在上下文菜单上，可以监听选择项和运行相应的命令。下面的例子将运行在系统托盘或停靠栏上，并提供一个命令用于关闭程序：

Code View:

```
<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    visible="false"
    windowComplete="completeHandler();">

    <mx:Script>
        <! [CDATA[
            [Embed(source='assets/AIRApp_16.png')]
            private var icon16:Class;
            [Embed(source='assets/AIRApp_32.png')]
            private var icon32:Class;
            [Embed(source='assets/AIRApp_48.png')]
            private var icon48:Class;
        ]>
    
```

```

[Embed(source='assets/AIRApp_128.png')]
private var icon128:Class;

private function completeHandler():void
{
    var shellMenu:NativeMenu = createShellMenu();

    var icon:InteractiveIcon =
        NativeApplication.nativeApplication.icon;
    if( NativeApplication.supportsDockIcon )
    {
        ( icon as DockIcon ).menu = shellMenu;
    }
    else
    {
        ( icon as SystemTrayIcon ).menu = shellMenu;
        ( icon as SystemTrayIcon ).tooltip = "My App";
    }

    var bitmaps:Array = [new icon16(), new icon32(),
                         new icon48(), new icon128()];
    icon.bitmaps = bitmaps;
}

private function createShellMenu():NativeMenu
{
    var menu:NativeMenu = new NativeMenu();
    var quitCmd:NativeMenuItem =
        new NativeMenuItem( "Quit" );
    quitCmd.addEventListener(Event.SELECT,quitHandler );
    menu.addItem( quitCmd );
    return menu;
}

private function quitHandler( evt:Event ):void
{
    NativeApplication.nativeApplication.exit();
}

]]>
</mx:Script>

</mx:WindowedApplication>

```

当应用程序启动和初始化完成后，通过NativeApplication.supportsDockIcon属性检测当前操作系统，如果为true则表明是Mac OS X系统，NativeApplication的InteractiveIcon类型的icon属性是一个DockIcon对象。通过createShellMenu方法创建本地上下文菜单赋值给icon的menu 属性。当用户点击quit命令时调用quitHandler方法。

这里有一个很重要的属性，那就是<mx:WindowedApplication>根标签的visible属性被设置为false，这样隐藏了程序主窗口，使所有用户必须通过系统托盘图标来访问程序。在描述文件里也同样设置visible属性达到相同效果。下面是一个描述文件例子：

```
<application xmlns="http://ns.adobe.com/air/application/1.0">

    <id>SystTrayApp</id>
    <name>SystTrayApp</name>
    <filename>SystTrayApp</filename>
    <version>0.1</version>
    <initialWindow>
        <content>SystTrayApp.swf</content>
        <systemChrome>none</systemChrome>
        <transparent>true</transparent>
        <visible>true</visible>
    </initialWindow>

</application>
```

## 第二十四章. FlexUnit 单元测试 (常青)

随着 Flex 应用程序变得越来越庞大和负责，单元测试已逐渐被广大 Flex 开发者所接受和欢迎。所谓单元测试，就是一个确保项目中新增条件或改变不会引入更多的 BUGs 或不会修改预期行为，使得大型团队能够不引入 bugs 的情况下协调工作，确认小的独立的部分程序向特定的方法返回预期结果。这使得 bugs 和异常能被迅速定位，因为正确的单元测试将能测试单个方法或小块功能的行为。

单元测试的核心是单元测试用例(Test Case)，它将传递值给应用程序方法并报告该方法是否有正确的值被返回。简单的如某个操作方法是否返回正确的整数值或复杂到确认一些显示逻辑是否被正确执行或某服务能返回正确的对象类型。多个测试用例可归类为一个测试集合，即一组测试整个应用程序或大型应用的特点部分的测试用例。测试集合将显示所有测试用例的测试结果，包括通过的和失败的。作为一个开发者，添加新代码的同时也要添加相应的测试用例，以确保新的代码不会干扰原有的代码并且能得到预期的结果。

FlexUnit 框架允许你创建测试用例和异步测试和评估测试工具，提供测试集合的所以测试的可视化显示。本章的这些小节都是由 Daniel Rinehart 编写，展示如何开发有意义的测试用例，整合到测试集合，以及如何使用更高级的工具如 Antennae 库为你的应用程序自动生成测试。

### 24.1节. 创建应用 FlexUnit 框架的应用程序

#### 24.1.1. 问题

我使用 FlexUnit 框架类为应用程序创建测试并运行之。

#### 24.1.2. 解决办法

下载和解压缩 FlexUnit，把 flexunit.swc 文件引入到应用程序编译路径中。

#### 24.1.3. 讨论

FlexUnit 框架包括一个图形化的测试运行器和用于创建自定义测试的基类。你可以从 <http://code.google.com/p/as3flexunitlib/> 下载到，确认下载的是最新的版本。

然后解压缩 ZIP 文件，只要把其中的 flexunit.swc 引入到应用程序的编译路径中即可，如果你使用 Flex Builder，选择 Project → Properties → Flex Build Path → Library Path → Add SWC，然后找到文件 flexunit/bin/flexunit.swc。如果你喜欢用命令行，修改 mxmlc 参数，引入 -library-path+=flexunit/bin/flexunit.swc，根据的情况调整下文件路径即可。

## 24.2节. 运行 FlexUnit 单元测试

### 24.2.1. 问题

我需要创建应用程序运行 FlexUnit 测试并显示测试结果。

### 24.2.2. 解决办法

使用 TestSuite 实例和 TestRunnerBase 组件运行测试。

### 24.2.3. 讨论

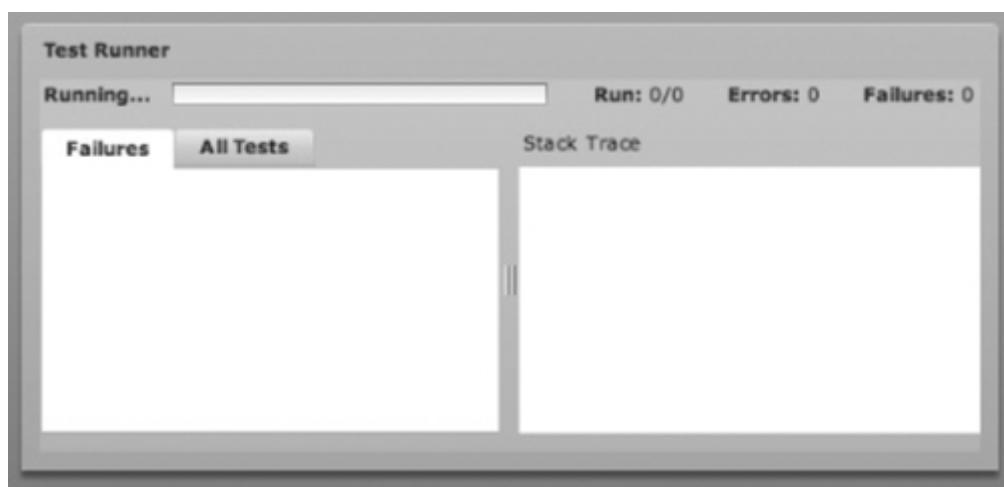
TestRunnerBase 是默认的包含 FlexUnit 框架的图形化测试运行器。想要用 TestRunnerBase 测试创建的应用程序，编辑 MXML 文件，加入如下内容：

Code View:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:flexui="flexunit.flexui.*">
    <flexui:TestRunnerBase id="testRunner" width="100%"
        height="100%"/>
</mx:Application>
```

编译并运行程序，输出结果像下面这样 [Figure 24-1](#)。

**Figure 24-1. The initial appearance of a FlexUnit test application**



然后，创建测试集合保存所有测试。在同一个 MXML 文件中添加<mx:Script>块：

```

<mx:Script>
<! [CDATA[
    import flexunit.framework.TestSuite;

    private function createTestSuite():TestSuite
    {
        var testSuite:TestSuite = new TestSuite();
        return testSuite;
    }
]]>

</mx:Script>

```

注意这个例子并没有添加任何测试到测试集合实例中，它只是创建了对象，以便往此添加 TestCase 实例。

最后，通过 handleCreationComplete 函数把 TestSuite 实例赋值给 TestRunnerBase 实例：

```

private function handleCreationComplete():void
{
    testRunner.test = createTestSuite();
    testRunner.startTest();
}

```

这个函数将在程序载入时自动启动测试。

Code View:

```

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:flexui="flexunit.flexui.*"
    creationComplete="handleCreationComplete();">

```

编译和运行程序后，输出面板如图 [Figure 24-1](#)，现在你可以添加测试用例(TestCase)实例到 TestSuite 实例中，当程序开始时它们将被运行。

这是最终的 MXML 文件：

```

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:flexui="flexunit.flexui.*"
    creationComplete="handleCreationComplete();">
<mx:Script>
<! [CDATA[
    import flexunit.framework.TestSuite;

    private function createTestSuite():TestSuite

```

```

    {
        var testSuite:TestSuite = new TestSuite();
        return testSuite;
    }

    private function handleCreationComplete():void
    {
        testRunner.test = createTestSuite();
        testRunner.startTest();
    }
} ]>
</mx:Script>
<flexui:TestRunnerBase id="testRunner" width="100%" height="100%"/>
</mx:Application>

```

## 24.3节. 创建FlexUnit测试用例

### 24.3.1. 问题

我需要创建FlexUnit TestCase类实例测试代码。

### 24.3.2. 解决办法

创建TestCase子类，包含一个或多个以test开头的方法。

### 24.3.3. 讨论

当创建继承自TestCase的ActionScript类，一般的命名方法是在类名后面添加Test作为后缀。例如，如果被测试类叫RegExp，则TestCase类将被命名为RegExpTest。另外一般把TestCase类和被测试类放在同一个包中。如一个类叫做mx.core.UITextFormat，那TestCase类叫mx.core.UITextFormatTest。

举例，创建一个继承自TestCase的叫RegExpTest的ActionScript类，如下代码：

```

package
{
    import flexunit.framework.TestCase;

    public class RegExpTest extends TestCase
    {
    }
}

```

FlexUnit框架使用反射确定TestCase的哪些方法可以运行。函数名称以test开头的指示该函数将包含要运行的代码，例如这节的例子，添加方法testRegExp到RegExpTest：

```
public function testRegExp():void
{
}
```

下一步就是生成一个或多个断言 (assertions)。一个断言就是以编程方式确认事实语句。一般的做法是通过一些操作将期望值与实际值相比较，返回结果。FlexUnit包含一系列断言类型用于不同的测试情形。常见的断言和函数如下：

#### assertEquals

比较 ==.

#### assertTrue

检测条件是否为 true.

#### assertNull

检测条件是否为 null.

#### assertStrictlyEquals

比较 ===.

FlexUnit还提供各种方便的断言用于测试相反的条件，例如assertFalse和assertNotNull. (具体请看FlexUnit文档中有关断言的完整列表)

每个断言函数都可接受可选的字符串作为第一个参数。如果断言失败时，该字符串将被作为默认的"expected X but was Y" 信息的前缀。编写断言时，请注意如果一个断言失败，剩下的测试方法将不会被执行。

TestCase继承自Assert，它定义了所有的断言函数。这允许TestCase子类可直接调用断言函数。下面的代码演示各种断言方法，这些方法应该被添加到testRegExp函数中：

Code View:

```
var regExp:RegExp = new RegExp("a", "i");
assertFalse(regExp.test("b"));
assertFalse("regExp doesn't match", regExp.test("b"));

assertNull(regExp.exec("b"));
assertNull("regExp doesn't match", regExp.exec("b"));

assertNotNull(regExp.exec("Apple"));
assertNotNull("regExp matches", regExp.exec("Apple"));

assertTrue(regExp.exec("Apple") is Array);
assertTrue("regExp exec returned an Array",
    regExp.exec("Apple") is Array);
```

```

assertEquals("A", regExp.exec("Apple") [0]);
assertEquals("regExp matched A in Apple", "A",
    regExp.exec("Apple") [0]);

assertStrictlyEquals(regExp, regExp);
assertStrictlyEquals("RegExp object identity", regExp,
    regExp);

```

你可以在TestCase中添加新的测试方法测试其他逻辑。习惯我们把每个测试方法集中测试特定的操作或任务。例如，当测试create, retrieve, update, 和delete操作时，每个操作都对应到它自己的测试方法，如 testCreate, testRetrieve等等。这种方式中，如果这些断言失败，将会报告多个故障，这将帮助你快速诊断问题。

请注意，TestCase中的测试方法的执行顺序是随机的。每个测试方法创建它自己的数据，和其他已运行的测试无联系。下面是完整的ActionScript文件：

Code View:

```

package
{
    import flexunit.framework.TestCase;

    public class RegExpTest extends TestCase
    {
        public function testRegExp():void
        {
            var regExp:RegExp = new RegExp("a", "i");
            assertFalse(regExp.test("b"));
            assertFalse("RegExp doesn't match", regExp.test("b"));

            assertNull(regExp.exec("b"));
            assertNull("RegExp doesn't match", regExp.exec("b"));

            assertNotNull(regExp.exec("Apple"));
            assertNotNull("RegExp matches", regExp.exec("Apple"));

            assertTrue(regExp.exec("Apple") is Array);
            assertTrue("RegExp exec returned an Array",
                regExp.exec("Apple") is Array);

            assertEquals("A", regExp.exec("Apple") [0]);
            assertEquals("regExp matched A in Apple", "A",
                regExp.exec("Apple") [0]);
        }
    }
}

```

```

        assertStrictlyEquals(regExp, regExp);
        assertStrictlyEquals("RegExp object identity", regExp,
            regExp);
    }
}

}

```

最后一步就是添加新创建的TestCase到TestSuite，这放在下一节讲解。

## 24.4节. 添加测试用例到测试集合

### 24.4.1. 问题

我想添加测试用例到现有的测试集合中。

### 24.4.2. 解决办法

使用TestSuite的addTestSuite方法

### 24.4.3. 讨论

使用addTestSuite方法将测试用例添加到测试集合中，该方法接受一个TestCase类引用作为参数。在后台FlexUnit使用反射找到所有以test开头的方法并执行它们。

下面的例子更新自 [24.2节](#) 的createTestSuite方法，添加RegExpTest到测试集中：

```

private function createTestSuite():TestSuite
{
    var testSuite:TestSuite = new TestSuite();
    testSuite.addTestSuite(RegExpTest);
    return testSuite;
}

```

如果TestCase没有在默认的包内，请确认导入类路径：

```

import mx.core.UITextFormatTest;

private function createTestSuite():TestSuite
{
    var testSuite:TestSuite = new TestSuite();
    testSuite.addTestSuite(RegExpTest);
    testSuite.addTestSuite(UITextFormatTest);
    return testSuite;
}

```

当添加多个测试用例后，可根据添加的顺序运行它们。

## 24.5节. 在测试前后运行代码

### 24.5.1. 问题

我需要在每个测试用例测试前或测试后运行特定的代码。

### 24.5.2. 解决办法

重写TestCase类的setUp和tearDown方法。

### 24.5.3. 讨论

默认情况下，每个TestCase中的测试方法都会在自己的TestCase实例中运行。如果多个测试方法需要同一个系统状态或数据，你可以使用setUp方法统一进行设置而不用在每个测试开始前显式调用某个设置方法。同理，如果需要在每个测试后清理某个对象或测试断言，无论是否有断言失败或错误，tearDown方法必须被运行。请切记一旦有断言失败或产生错误，测试方法将会停止执行。tearDown方法还有点好处就是如果测试使用到外部资源或对象的话可以进行释放。

setUp方法很常用，比如用来保持当前正常状态数据到系统中。对于复杂的测试，可能需要挂接多个对象或连接到外部资源。如要在测试之前创建代码，像下面那样重写setUp方法：

```
override public function setUp():void
{
}
```

你可以在这里放置任何代码，包括断言。如果某些资源或对象不存在的话，在setUp方法里使用断言可直接快速取消测试。注意如果setUp方法内出现断言失败或抛出异常，预订的测试方法或tearDown方法都不会被调用。这是tearDown方法唯一不被调用的情况。

类似setUp，tearDown方法是在每个测试后运行，无论是否有失败断言或异常。可以把它理解成try...catch...finally块的finally部分。按这种理解，也就是说tearDown方法并不是必需的。请记住默认下每个测试方法都是运行在自己的TestCase实例中，这就意味着类变量将被设置为实例值，取消之前测试方法所作的修改。常见的如tearDown方法包括执行每个测试方法运行后生成的贡献断言或对外部资源的释放，比如断开Socket。要在每个测试方法后面运行代码，像下面那样重写tearDown方法：

```
override public function tearDown():void
{
}
```

下面的代码演示每个测试方法何时运行，调用setUp，测试代码和tearDown方法：

Code View:

```
package
{
    import flexunit.framework.TestCase;

    public class SetUptearDownTest extends TestCase
    {
```

```

private var _phase:String = "instance";

override public function setUp():void
{
    updatePhase("setUp()");
}

override public function tearDown():void
{
    updatePhase("tearDown()");
}

public function testOne():void
{
    updatePhase("testOne()");
}

public function testFail():void
{
    updatePhase("testFail()");
    fail("testFail() always fails");
}

public function testError():void
{
    updatePhase("testError()");
    this["badPropertyName"] = "newValue";
}

private function updatePhase(phase:String):void
{
    trace("Running test", methodName, "old phase", _phase,
          "new phase", phase);
    _phase = phase;
}
}

```

输出信息如下：

```

Running test testFail old phase instance new phase setUp()
Running test testFail old phase setUp() new phase testFail()
Running test testFail old phase testFail() new phase tearDown()

```

```
Running test testError old phase instance new phase setUp()
Running test testError old phase setUp() new phase testError()
Running test testError old phase testError() new phase tearDown()
Running test testOne old phase instance new phase setUp()
Running test testOne old phase setUp() new phase testOne()
Running test testOne old phase testOne() new phase tearDown()
```

注意每个测试都以实例的\_phase值开头，无论是否有断言失败或异常，setUp和tearDown 方法都被执行。

## 24.6节。测试用例之间共享测试数据

### 24.6.1. 问题

我想在多个测试用例之间共享数据，包括简单的或复杂的测试数据实例。

### 24.6.2. 解决办法

创建能生成所需测试数据实例的工厂类。

### 24.6.3. 讨论

一般的单元测试都有多个测试用例，彼此之间需共享相同或类似的测试数据。这些数据可能很简单，比如是一个表示地址的object，也可能会很复杂，比如是以特定方式建立的相关实体的命令。不是通过剪贴和粘帖代码或从外部载入资源来为每个TestCase创建和初始化对象，而是通过工厂来集中创建。这种集中处理数据的类型方式我们称之为ObjectMother 设计模式。

简单为例，ObjectMother是个简单的工具类，只提供一个静态方法用于创建需要的类型对象。该方法通常有两种形式：一种是需要传递一个参数值用于设置每个属性，该方法只装配对象。第二种需要很少或不需要参数，该方法为每个字段提供实际的智能的默认值。如需要其他的对象类型，还可以使用较低级别的创建方法来生成更复杂的对象。

下面的例子是一个简单的ObjectMother实现：

Code View:

```
package
{
    public class ObjectMother
    {
        public static const SHIPPING_ZIP_CODE:String = "0123";
```

```

public static function createAddress(line:String,
    city:String, state:String, zip:String):Address
{
    var address:Address = new Address();
    address.line = line;
    address.city = city;
    address.state = state;
    address.zip = zip;
    return address;
}

public static function createAddressShipping():Address
{
    return createAddress("123 A Street", "Boston", "MA",
        SHIPPING_ZIP_CODE);
}

public static function createAddressBilling():Address
{
    return createAddress("321 B Street", "Cambridge", "MA",
        "02138");
}

public static function createOrder(lineItems:Array =
null):Order
{
    var order:Order = new Order();
    order.shippingAddress = createAddressShipping();
    order.billingAddress = createAddressBilling();
    for each (var lineItem:LineItem in lineItems)
    {
        addLineItemToOrder(order, lineItem);
    }
    return order;
}

public static function addLineItemToOrder(order:Order,
lineItem:LineItem):void
{
    order.addLineItem(lineItem);
}
}

```

从简单的Address对象开始，定义了标准的参数化创建方法createAddress。两个辅助类函数createAddressShipping和createAddressBilling，为TestCase方法提供快速方式访问更充实的Address实例。辅助类函数建立在通用的createAddress函数值上。分层的创建策略可方便的创建出更复杂类型的对象，如这里的createOrder例子。

因为每次调用一个方法都会生成新的对象实例，所以由一个TestCase的修改不会影响到另一个TestCase。同一时刻，测试数据都被集中化处理，在ObjectMother中修改数据以支持新的测试可能会破坏现有的测试。相对于能轻松访问到测试数据等优点这只是一个小小问题。

## 24.7节. 处理测试用例事件

### 24.7.1. 问题

我需要等待TestCase中的某个事件。

### 24.7.2. 解决办法

使用addAsync FlexUnit方法

### 24.7.3. 讨论

TestCase的测试行为经常会出现等待异步事件问题。如果TestCase方法只涉及同步事件，比如当改变属性时property change事件立即触发，不需要特殊的处理。但是当涉及到异步事件时你就要特别小心了。一般需要在测试中监听异步事件，如URLLoad是否完成或UIComponent 是否已创建完毕。这一节将讨论在TestCase中使用URLLoad类和假设的配置对象时如何处理事件。

在TestCase中事件需要被特别对待，因为除非FlexUnit被通知需要等待一个事件，否则测试方法将会理解完成。FlexUnit会认为该方法已通过测试，并开始运行下一个测试方法。这将导致不一致的结果，如FlexUnit会显示一个绿色状态栏而后台的测试却是失败的或更糟的是会显示错误信息。

需要先通知FlexUnit应该等待触发的事件完成后再决定一个测试是通过还是失败，传递给addEventListener的监听器必须替换为addAsync，addAsync的前两个参数是必须的，剩下的都是可选的，第一个参数是监听器，当事件触发时被调用。第二个参数为单位为毫秒的超时时间，当超过这个时间事件还没触发，FlexUnit将认为该测试失败并继续运行其他的测试方法。

下面的例子是典型的addAsync用法：

Code View:

```
package
{
    import flash.events.Event;
    import flash.net.URLLoader;
    import flash.net.URLRequest;
```

```

import flexunit.framework.TestCase;

public class ConfigurationTest extends TestCase
{
    public function testParse():void
    {
        var urlLoader:URLLoader = new URLLoader();
        urlLoader.addEventListener(Event.COMPLETE,
            addAsync	verifyParse, 1000));
        urlLoader.load(new URLRequest("sample.xml"));
    }

    private function verifyParse(event:Event):void
    {
        var configuration:Configuration = new Configuration();
        assertFalse(configuration.complete);
        configuration.parse(new XML(event.target.data));
        assertTrue(configuration.complete);
    }
}

```

Configuration的parse方法被分为两个方法，第一个方法为等待事件所需的对象进行构造并初始化动作。第二个验证方法则使用事件结果执行处理过程和进行断言。通常情况下，verifyParse方法应该是直接作为addEventListener的监听器参数的，但是这里却被addAsync和给出的1000毫秒超时有包装了一次。请注意这里的监听器函数的名称不是以test开头的；如果是这样的话，那FlexUnit就会试图把它作为额外的测试方法来运行，这可不是我们所期望的结果。

对于FlexUnit来说事件类型是很重要的。且目标监听器可被强制转换为其参数规定类型。上面的例子中使用了通用的Event而不是FlexEvent或其他Event子类，这样监听器可安全地确认其参数类型。不会发生事件不匹配，在运行时会报告这种类型强制转换故障。

在这一点上，关于addAsync有两个需要特别注意的问题。首先，决不能出现同时有两个addAsync处于等待状态，这样FlexUnit将不能正确处理检测和测试。不过把这些AddAsync调用串联起来是可以的，这样只有一个addAsync调用监听器，新的addAsync将在随后的代码中被创建。其次，在测试期间不能多次调用addAsync。因为addAsync机制是作为汇合点让FlexUnit知道何时测试会完成或失败，多次调用addAsync会导致误报或不正常现象。

用addAsync来代替使用闭包或创建实例变量，这样通过addAsync的可选的第三个参数传递给监听器。传递的参数可以是任意的，可灵活使用。例如，上面代码的定义的测试可以在创建确认完成标志Configuration对象之前先初始化XML载入。这样的做法是遵循单元测试的快速故障检测模式，帮助你尽可能的节省整个测试集合的测试时间。下面是使用传递数据的方式修改的测试方法：

Code View:

```
public function testComplete():void
{
    var configuration:Configuration = new Configuration();
    assertFalse(configuration.complete);

    var urlLoader:URLLoader = new URLLoader();
    urlLoader.addEventListener(Event.COMPLETE,
        addAsync(verifyComplete, 1000, configuration));
    urlLoader.load(new URLRequest("sample.xml"));
}

private function verifyComplete(event:Event,
    configuration:Configuration):void
{
    configuration.parse(new XML(event.target.data));
    assertTrue(configuration.complete);
}
```

测试方法中创建的对象作为verifyComplete的第二个参数被传递进来。使用这种机制还可以传递通用的Object或想int这样的基本数据类型。

默认，超过指定的时间后事件将不会被触发，FlexUnit生成失败报告。如果事件没有触发打还有些事情需要处理，这时可以使用addAsync的第四个参数指定将被调用的函数。当测试期间事件没有触发或在测试中执行特殊的对象清理时定义自定义失败处理函数是非常有用的。自定义失败处理函数总是接受传递的数据，即便它是null。下面我们假设没有触发complete事件来测试下Configuration对象：

Code View:

```
public function testCompleteEvent():void
{
    var configuration:Configuration = new Configuration();
    assertFalse(configuration.complete);
    configuration.addEventListener(Event.COMPLETE,
        addAsync(verifyEvent, 250, configuration,
            verifyNoEvent));
}

private function verifyEvent(event:Event,
    configuration:Configuration):void
{
    fail("Unexpected Event.COMPLETE from Configuration
        instance");
}
```

```

private function
verifyNoEvent(configuration:Configuration) :void
{
    assertFalse(configuration.complete);
}

```

还是很有必要为此事件定义一个监听器，如本例所示，当事件触发时，它显示错误状态，当事件没有触发时，自定义失败处理函数仍然会在适当的状态确认Configuration。

为建立或测试一个对象如果有多个异步事件触发，要保证同一时刻只有一个addAsync处于活动状态，如前所述，为了解决此限制，你可以创建另一个addAsync。扩充前面的例子，如果configuration的解析需要载入外部文件，complete状态可能不会立即发生变化。下面的例子就是演示如何把两个事件串联起来：

Code View:

```

public function testComplexComplete() :void
{
    var configuration:Configuration = new Configuration();
    assertFalse(configuration.complete);

    var urlLoader:URLLoader = new URLLoader();
    urlLoader.addEventListener(Event.COMPLETE,
addAsync(verifyComplexParse, 10 00, configuration));
    urlLoader.load(new URLRequest("complex.xml"));
}

private function verifyComplexParse(event:Event,
configuration:Configuration) : void
{
    configuration.addEventListener(Event.COMPLETE,
addAsync(verifyComplexComplete, 1000, configuration));
    configuration.parse(new XML(event.target.data));
    assertFalse(configuration.complete);
}

private function verifyComplexComplete(event:Event,
configuration:Configuration) :void
{
    assertTrue(configuration.complete);
}

```

在verifyComplexParse函数是为第一个addAsync进行配置，第二个调用addAsync是为监听串

联事件中的下一个事件，这样的串联可以根据需要继续增加事件。

## 24.8节。用FlexUnit测试可视化组件

### 24.8.1. 问题

我需要测试可视化组件

### 24.8.2. 解决办法

展示将组件放在可视体系中然后测试它。

### 24.8.3. 讨论

有人认为可视组件的测试已偏离了单元测试的目的，因为它们很难被独立出来进行测试，以便能控制测试条件。测试功能丰富的Flex框架组件是很复杂的，比如怎样确定某个方法是否被正确调用。样式和父容器也会影响一个组件的行为。因此，你最好是用自动化功能测试的方法来测试可视组件。

在测试一个可视组件之前，该组件必须通过了各种生命周期步骤。当组件被添加到显示体系后Flex框架会自动进行处理。TestCases虽然不是一个可视组件，这意味着组件必须与外部的TestCase相关联。这种外部关联意味着无论测试失败还是成功你都必须小心清理善后工作；否则可能会影响到其他组件的测试。

#### 24.8.3.1. 组件测试模式

获得显示对象引用最简单的方法是使用Application.application。因为TestCase是运行Flex应用程序之上，它是一个单例实例。可视组件的创建和激活并不是一个同步的行为；在可被测试之前，TestCase需要等待组件进入一个已知的状态。通过使用addAsync等待FlexEvent.CREATION\_COMPLETE事件是最简单的方法知道新创建的组件已进入已知状态。要确保一个TestCase方法不会影响到其他正在运行的TestCase方法，被创建的组件在被移除前必须清理和释放任何外部引用。使用tearDown方法和类实例变量是完成这两个任务的最好方法。下面的例子代码演示Tile组件的创建，连接，激活和清理：

Code View:

```
package mx.containers
{
    import flexunit.framework.TestCase;

    import mx.core.Application;
    import mx.events.FlexEvent;

    public class TileTest extends TestCase
    {
        // class variable allows tearDown() to access the instance
```

```

private var _tile:Tile;

override public function tearDown():void
{
    try
    {
        Application.application.removeChild(_tile);
    }
    catch (argumentError:ArgumentError)
    {
        // safe to ignore, just means component was never
added
    }
    _tile = null;
}

public function testTile():void
{
    _tile = new Tile();
    _tile.addEventListener(FlexEvent.CREATION_COMPLETE,
        addAsync(verifyTile, 1000));
    Application.application.addChild(_tile);
}

private function verifyTile(flexEvent:FlexEvent):void
{
    // component now ready for testing
    assertTrue(_tile.initialized);
}
}

```

这里需要注意的关键点是定义了一个类变量，允许tearDown方法引用这个被创建和添加到Application.application的实例对象。另外组件被添加到Application.application可能还没有成功，这就是为什么tearDown方法中的removeChild调用被放置在try...catch块中以防止抛出任何异常。测试方法使用addAsync在运行测试之前进行等待，直到组件进入一个稳定的状态。

#### 24.8.3.2. 组件创建测试

虽然你可以手动调用测试和各种其他Flex框架组件方法，测试将更好的模拟对象所运行的环境。不像单元测试，组件所连接外部环境将不能被严格控制，这意味着你必须集中于组件的测试而不能顾及周围环境。例如，早先创建的Tile容器的布局逻辑可被测试：

Code View:

```
public function testTileLayout():void
{
    _tile = new Tile();
    var canvas:Canvas = new Canvas();
    canvas.width = 100;
    canvas.height = 100;
    _tile.addChild(canvas);
    canvas = new Canvas();
    canvas.width = 50;
    canvas.height = 50;
    _tile.addChild(canvas);
    canvas = new Canvas();
    canvas.width = 150;
    canvas.height = 50;
    _tile.addChild(canvas);
    _tile.addEventListener(FlexEvent.CREATION_COMPLETE,
        addAsync(verifyTileLayout, 1000));
    Application.application.addChild(_tile);
}

private function verifyTileLayout(flexEvent:FlexEvent):void
{
    var horizontalGap:int =
        int(_tile.getStyle("horizontalGap"));
    var verticalGap:int =
        int(_tile.getStyle("verticalGap"));
    assertEquals(300 + horizontalGap, _tile.width);
    assertEquals(200 + verticalGap, _tile.height);
    assertEquals(3, _tile.numChildren);
    assertEquals(0, _tile.getChildAt(0).x);
    assertEquals(0, _tile.getChildAt(0).y);
    assertEquals(150 + horizontalGap,
        _tile.getChildAt(1).x);
    assertEquals(0, _tile.getChildAt(1).y);
    assertEquals(0, _tile.getChildAt(2).x);
    assertEquals(100 + verticalGap, _tile.getChildAt(2).y);
}
```

这个例子中，三个大小不同的子组件被添加到Tile。根据Tile布局逻辑，这个例子应该创建一个 $2 \times 2$  的网格并使每个网格的宽度和高度最大化以容纳子组件。Verify方法断言默认逻辑将会生产的结果。重要的是要注意到测试只注重于组件所使用到的逻辑。这不是测试布局是否看起来足够好，而只是其行为与文档相匹配。另外重要的一点需要注意，就是关于当前层

级的组件测试会影响到在其之上的组件样式。这个测试方法在它创建实例时可以设置样式值以便确认该值是否被使用过。

#### 24.8.3.3. Postcreation 测试

组件创建后，额外的变化会让测试变得很困难。通常使用FlexEvent.UPDATE\_COMPLETE事件，但组件的一个简单变化会多次触发此事件。虽然可以建立逻辑以正确处理这多个事件，但是TestCase除了测试组件内逻辑并不会区分Flex框架事件和UI更新逻辑。因此集中于组件逻辑的测试设计确实是一门艺术。这就是为什么要在这个级别进行组件测试而不是单元测试。

下面的例子添加了另外的子组件到先前创建的Tile，检测发生的变化：

Code View:

```
// class variable to track the last addAsync() Function
instance
private var _async:Function;

public function testTileLayoutChangeAfterCreate():void
{
    _tile = new Tile();
    var canvas:Canvas = new Canvas();
    canvas.width = 100;
    canvas.height = 100;
    _tile.addChild(canvas);
    canvas = new Canvas();
    canvas.width = 50;
    canvas.height = 50;
    _tile.addChild(canvas);
    canvas = new Canvas();
    canvas.width = 150;
    canvas.height = 50;
    _tile.addChild(canvas);
    _tile.addEventListener(FlexEvent.CREATION_COMPLETE,
addAsync(verifyTileLayoutAfterCreate, 1000));
    Application.application.addChild(_tile);
}

private function
verifyTileLayoutAfterCreate(flexEvent:FlexEvent):void
{
    var horizontalGap:int =
int(_tile.getStyle("horizontalGap"));
    var verticalGap:int =
int(_tile.getStyle("verticalGap"));
}
```

```

        assertEquals(300 + horizontalGap, _tile.width);
        assertEquals(200 + verticalGap, _tile.height);
        assertEquals(3, _tile.numChildren);
        assertEquals(0, _tile.getChildAt(0).x);
        assertEquals(0, _tile.getChildAt(0).y);
        assertEquals(150           + horizontalGap,
_tile.getChildAt(1).x);
        assertEquals(0, _tile.getChildAt(1).y);
        assertEquals(0, _tile.getChildAt(2).x);
        assertEquals(100 + verticalGap, _tile.getChildAt(2).y);

    var canvas:Canvas = new Canvas();
    canvas.width = 200;
    canvas.height = 100;
    _tile.addChild(canvas);
    _async = addAsync(verifyTileLayoutChanging, 1000);
    _tile.addEventListener(FlexEvent.UPDATE_COMPLETE,
_async);
}
}

private function
verifyTileLayoutChanging(flexEvent:FlexEvent):void
{
    _tile.removeEventListener(FlexEvent.UPDATE_COMPLETE,
_async);
    _tile.addEventListener(FlexEvent.UPDATE_COMPLETE,
addAsync
(verifyTileLayoutChangeAfterCreate, 1000));
}

private function
verifyTileLayoutChangeAfterCreate(flexEvent:FlexEvent):void
{
    var horizontalGap:int =
        int(_tile.getStyle("horizontalGap"));
    var verticalGap:int =
        int(_tile.getStyle("verticalGap"));
    assertEquals(400 + horizontalGap, _tile.width);
    assertEquals(200 + verticalGap, _tile.height);
    assertEquals(4, _tile.numChildren);
    assertEquals(0, _tile.getChildAt(0).x);
    assertEquals(0, _tile.getChildAt(0).y);

    assertEquals(200 + horizontalGap,

```

```

        _tile.getChildAt(1).x);
assertEquals(0, _tile.getChildAt(1).y);

assertEquals(0, _tile.getChildAt(2).x);
assertEquals(100 + verticalGap, _tile.getChildAt(2).y);

assertEquals(200 + horizontalGap,
        _tile.getChildAt(3).x);
assertEquals(100 + verticalGap, _tile.getChildAt(3).y);

}

```

事件处理逻辑现在使用一个类变量来跟踪最后通过addAsync添加的异步函数，这是为了允许改监听器可被移除并添加一个不同的监听器来处理第二次触发的同类事件。如果这时另一个变化发生，将会触发另一个FlexEvent.UPDATE\_COMPLETE，verifyTileLayoutChanging方法也必须存储它的addAsync函数为了它能被移除。如果Flex框架逻辑改变了如何触发事件，那这一链式事件处理就显得很脆弱了，整个代码测试将会导致失败。这个测试没有处理这两个触发的FlexEvent.UPDATE\_COMPLETE事件为了组件能顺利完成子组件的布局任务；在这个级别试图捕捉组件逻辑会产生意想不到的效果。如果在中间状态verifyTileLayoutChanging中捕捉组件逻辑，在这个方法中的断言将发挥作用，如果事件没有被正确触发，这些变化的事件将会保证此测试失败。

虽然组件也会触发额外的事件，如Event.RESIZE，但是该事件所在的组件状态通常是不稳定的。正如处在Event.RESIZE的Tile那样，组件的宽度发生变化，但是其子组件的位置却还没有。另外还可能有这样的排队操作，当要移除显示层级中的组件时操作队列中却要试图访问该组件，这将会导致错误。当测试那些采用同步方式更新逻辑的组件，移除其他监听器还需要的组件时要尽量避免发生这些问题。换句话说，被测试组件发出的事件要清晰表明组件的变化已完全实现。不管你选择什么方法处理这种情况，请记住有哪些方法是可靠的，有哪些测试行为是脱离组件的。

#### 24.8.3.4. 根据时间测试

如果组件一下子产生了很多复杂的变化，维持事件的数量和顺序将会非常困难。除了等待特定的事件外，另一个办法就是去等待一段时间。这个方法可以轻松的处理多个被更新的对象或组件(使用Effect实例，在某个已知时间内播放)。基于时间的测试最主要的缺点就是如果测试环境的速度和资源发生改变的话可能会导致误报。等待一个固定时间也意味着整个TestSuite的所花时间将比添加异步或事件驱动的测试多出不少。

下面的代码是把之前的Tile例子用基于时间的触发器改写一边：

Code View:

```

private function waitToTest(listener:Function,
waitTime:int):void
{
    var timer:Timer = new Timer(waitTime, 1);

```

```

        timer.addEventListener(TimerEvent.TIMER_COMPLETE,
            addAsync(listener, waitTime + 250));
        timer.start();
    }

public function testTileLayoutWithTimer():void
{
    _tile = new Tile();
    var canvas:Canvas = new Canvas();
    canvas.width = 100;
    canvas.height = 100;
    _tile.addChild(canvas);
    canvas = new Canvas();
    canvas.width = 50;
    canvas.height = 50;
    _tile.addChild(canvas);
    canvas = new Canvas();
    canvas.width = 150;
    canvas.height = 50;
    _tile.addChild(canvas);
    Application.application.addChild(_tile);
    waitToTest	verifyTileLayoutCreateWithTimer, 500;
}

private function
verifyTileLayoutCreateWithTimer(timerEvent:TimerEvent):void
{
    var horizontalGap:int =
        int(_tile.getStyle("horizontalGap"));
    var verticalGap:int =
        int(_tile.getStyle("verticalGap"));
    assertEquals(300 + horizontalGap, _tile.width);
    assertEquals(200 + verticalGap, _tile.height);
    assertEquals(3, _tile.numChildren);
    assertEquals(0, _tile.getChildAt(0).x);
    assertEquals(0, _tile.getChildAt(0).y);
    assertEquals(150 + horizontalGap,
        _tile.getChildAt(1).x);
    assertEquals(0, _tile.getChildAt(1).y);
    assertEquals(0, _tile.getChildAt(2).x);
    assertEquals(100 + verticalGap, _tile.getChildAt(2).y);

    var canvas:Canvas = new Canvas();
    canvas.width = 200;
}

```

```

        canvas.height = 100;
        _tile.addChild(canvas);
        waitToTest(verifyTileLayoutChangeWithTimer, 500);
    }

private function
verifyTileLayoutChangeWithTimer(timerEvent:TimerEvent):void
{
    var horizontalGap:int =
        int(_tile.getStyle("horizontalGap"));
    var verticalGap:int =
        int(_tile.getStyle("verticalGap"));
    assertEquals(400 + horizontalGap, _tile.width);
    assertEquals(200 + verticalGap, _tile.height);
    assertEquals(4, _tile.numChildren);
    assertEquals(0, _tile.getChildAt(0).x);
    assertEquals(0, _tile.getChildAt(0).y);
    assertEquals(200 + horizontalGap,
        _tile.getChildAt(1).x);
    assertEquals(0, _tile.getChildAt(1).y);
    assertEquals(0, _tile.getChildAt(2).x);
    assertEquals(100 + verticalGap, _tile.getChildAt(2).y);
    assertEquals(200 + horizontalGap,
        _tile.getChildAt(3).x);
    assertEquals(100 + verticalGap, _tile.getChildAt(3).y);
}

```

和之前的测试例子，能快速响应触发的事件不同，这个版本的测试将最少要1秒时间，更多的时间被用在定时器延时上，在这期间调用 addAsync 处理。之前例子中包装 FlexEvent.UPDATE\_COMPLETE 监听器的中间方法被移除了，但在其他测试代码中保持一致。

#### 24.8.3.5. 使用程序化的视觉断言

获取渲染组件的原始位图数据能力能很方便的以编程方式来验证可视组件的某个方面。这里有个例子将测试组件的背景和边框样式是如何改变的。创建组件实例后，可以捕捉其位图数据并进行检查。下面的例子通过添加Canvas边框来测试能产生预期效果：

Code View:

```

package mx.containers
{
    import flash.display.BitmapData;

    import flexunit.framework.TestCase;

```

```

import mx.core.Application;
import mx.events.FlexEvent;

public class CanvasTest extends TestCase
{
    // class variable allows tearDown() to access the instance
    private var _canvas:Canvas;

    override public function tearDown():void
    {
        try
        {
            Application.application.removeChild(_canvas);
        }
        catch (argumentError:ArgumentError)
        {
            // safe to ignore, just means component was never
            added
        }
        _canvas = null;
    }

    private function captureBitmapData():BitmapData
    {
        var bitmapData:BitmapData = new
            BitmapData(_canvas.width, _canvas.height);
        bitmapData.draw(_canvas);
        return bitmapData;
    }

    public function testBackgroundColor():void
    {
        _canvas = new Canvas();
        _canvas.width = 10;
        _canvas.height = 10;
        _canvas.setStyle("backgroundColor", 0xFF0000);
        _canvas.addEventListener(FlexEvent.CREATION_COMPLETE,
            addAsync(verifyBackgroundColor, 1000));
        Application.application.addChild(_canvas);
    }

    private function
    verifyBackgroundColor(flexEvent:FlexEvent):void

```

```

{
    var bitmapData:BitmapData = captureBitmapData();
    for (var x:int = 0; x < bitmapData.width; x++)
    {
        for (var y:int = 0; y < bitmapData.height; y++)
        {
            assertEquals("Pixel (" + x + ", " + y + ")",
                0xFF0000, bitmapData.getPixel(x, y));
        }
    }
}

public function testBorder():void
{
    _canvas = new Canvas();
    _canvas.width = 10;
    _canvas.height = 10;
    _canvas.setStyle("backgroundColor", 0xFF0000);
    _canvas.setStyle("borderColor", 0x00FF00);
    _canvas.setStyle("borderStyle", "solid");
    _canvas.setStyle("borderThickness", 1);
    _canvas.addEventListener(FlexEvent.CREATION_COMPLETE,
        addAsync	verifyBorder, 1000));
    Application.application.addChild(_canvas);
}

private function verifyBorder(flexEvent:FlexEvent):void
{
    var bitmapData:BitmapData = captureBitmapData();
    for (var x:int = 0; x < bitmapData.width; x++)
    {
        for (var y:int = 0; y < bitmapData.height; y++)
        {
            if ((x == 0) || (y == 0) || (x == bitmapData.width - 1) ||
                (y == bitmapData.height - 1))
            {
                assertEquals("Pixel (" + x + ", " + y + ")",
                    0x00FF00,bitmapData.getPixel(x, y));
            }
            else
            {
                assertEquals("Pixel (" + x + ", " + y + ")",
                    0xFF0000,bitmapData.getPixel(x, y));
            }
        }
    }
}

```

```
        }
    }
}
}
```

testBackgroundColor方法验证已设置背景颜色的Canvas的所有像素。testBorder方法验证何时边框被添加到Canvas，外边框的像素值转换为边框颜色而其他所以像素仍保持背景色。捕获位图数据是在captureBitmapData方法中进行，使用它可以绘制任何Flex组件到BitmapData实例上。这是一项很强大的技术，可以用来验证程序化的皮肤或其他很难进行单元测试的可视组件。

还有另一种方法测试组件的外观。请到<http://code.google.com/p/visualflexunit/>看看Visual FlexUnit。

#### 24.8.3.6. 隐藏被测试组件

添加测试组件到Application.application的一个副作用就是它将被渲染处理。这将导致当测试正在运行，组件正在被添加和移除时FlexUnit测试很难进行调整和重定位。要排除这个情况，你可以通过设置组件的visible和includeInLayout属性为false来隐藏被测试组件。例如，如果要隐藏之前代码中的Canvas的话，添加下面的代码：

```
_canvas.visible = false;
_canvas.includeInLayout = false;
Application.application.addChild(_canvas);
```

## 24.9节. 安装和配置Antennae

### 24.9.1. 问题

我想要自动构建和测试Flex应用程序。

### 24.9.2. 解决办法

下载，解压缩开源的Antennae模板并为指定系统进行配置。

### 24.9.3. 讨论

Antennae是一个专用于自动构建和测试Flex应用程序的开源项目。它使用Ant和Java提供跨平台工具来编译Flex库和Flex应用程序，生成FlexUnit测试集，并以自动化方式运行FlexUnit测试。Antennae也定义了一个框架用于构建具有多依赖的复杂项目并智能化的进行重编译。你

可以从<http://code.google.com/p/antennae/>. 下载它。确保下载的是最新版本的Antennae-\* .zip。解压缩后，Antennae ZIP包的所有内容都会放在Antennae目录下。.

Antennae有多个目录组成，大致如下：

#### *lib*

包括生成TestSuite所需的已编译的Java和Flex工具，FlexUnit运行和报告的命令行, FlexUnit库以及FlexUnit程序模板。

#### *src*

Java和Flex工具源代码，不包括FlexUnit (它在另外个地方)。

#### *templates*

构建Flex库，Flex应用程序，FlexUnit应用程序和具有复杂依赖的Flex项目所需的Antennae模板。

#### *tools*

生成Ant目标，任务和属性进行自动构建和测试。

#### *tutorial*

如何使用基础Ant特性好Antennae模板的教程。

另外还包括文档和用于在各种平台上配置Antennae的示例文件。

首先需要正确配置才能运行和实验Antennae所有的模板和教程。为了提供跨平台和多开发者支持，Antennae把所有可能变化的属性都集中到一个文件中，这样你可以根据需要进行自定义设置。在根目录中有两个文件，build-user.properties.mac和build-user.properties.win，是配置Antennae的入口。Linux用户应该可以使用build-user.properties.mac文件，但是需要注意可能由于Flex框架的限制有些特性将不可用。

复制相应系统的build-user.properties文件。下一步就是修改这个文件。你可以用任何文本编辑器编辑这个文件。

flex2.dir关键字属性是必须要设置的：把它指向Flex 2或Flex 3的目录(就是bin, lib和player的父目录)。可以是Flex Builder中包含的Flex SDK或独立版的Flex SDK。

如果你想使用命令行方式的FlexUnit自动化工具，你必须设置flex2.standalone.player属性。建议设置为调试版的播放器。使用调试版可以跟踪输出信息，帮助你确定测试正确性。

Tomcat.webapps.dir属性需要被设置为tutorial/multi/app/例子的部署目标。

最后，以air开头的各种属性是针对编译，运行和打包基于Flex的AIR应用程序的。关于这些属性的配置请看Antennae维基文档<http://code.google.com/p/antennae/w/list>

请浏览下tutorial目录快速看一下Antennae是如何构建和测试项目的。运行构建目标的Ant或测试将通过tutorial目录下的所有项目，调用每个项目的正确目标，为每个测试目标演示

FlexUnit运行是成功还是失败。

根目录下包含一个README.txt文件，说明了如何建立Antennae和构建和测试项目基本原理。Tutorial和template目录下都有一个README.txt文件介绍每个项目的目标和使用方法。

## 24.10节. 生成自动测试集

### 24.10.1. 问题

我想自动生成包含所有测试用例的测试集。

### 24.10.2. 解决办法

使用Antennae TestSuite-generation工具。

### 24.10.3. 讨论

TestCase必须包含进TestSuite才可以运行。创建新的TestCase并添加到TestSuite已经成为一种体力活。为了代替这种手工添加每个TestCase到TestSuite，你可以让TestSuite自动生成。开源的Antennae项目包含一个工具自动检测源代码目录和包含的测试自动生成TestSuite。

Antennae包中的lib子目录包含一个JAR文件叫arc-flexunit2.jar，里面有类叫com.allurent.flexunit2.framework.AllTestsFileGenerator。当AllTestsFileGenerator在源代码目录运行时，它会寻找所有命名为Test\*.as或\*Test.as并创建包含它们的TestSuite。这个工具可以在被重新定位的标准输出上创建TestSuite。生成的TestSuite文件被叫做FlexUnitAllTests。假设Antennae被解压到~/Antennae和C:\Antennae，你可以像下面那样调用此工具：

```
java -cp ~/Antennae/lib/arc-flexunit2.jar  
com.allurent.flexunit2.framework.AllTestsFileGenerator  
~/FlexCookbook/src/ > ~/FlexCookbook/src/FlexUnitAllTests.as
```

```
java -cp C:\Antennae\lib\arc-flexunit2.jar  
com.allurent.flexunit2.framework.AllTestsFileGenerator  
C:\FlexCookbook\src\ > C:\FlexCookbook\src\FlexUnitAllTests.as
```

第一个例子中，~/Antennae/lib/arc-flexunit2.jar是JAR文件的位置。运行的类名为com.allurent.flexunit2.framework.AllTestsFileGenerator，~/FlexCookbook/src/是源代码目录位置，~/FlexCookbook/src/FlexUnitAllTests.as是生成文件的位置。

生成的TestSuite文件大概是这样：

```
package  
{  
    import flexunit.framework.*;
```

```

import mx.containers.CanvasTest;
import mx.containers.TileTest;

public class FlexUnitAllTests
{
    public static function suite() : TestSuite
    {
        var testSuite:TestSuite = new TestSuite();
        testSuite.addTestSuite(mx.containers.CanvasTest);
        testSuite.addTestSuite(mx.containers.TileTest);
        return testSuite;
    }
}
}

```

当FlexUnit应用程序被编译之前，总是需要自动生成FlexUnitAllTests文件(请看Antennae文档关于Flex Builder下使用AllTestsFileGenerator工具的更多细节)

为了代替在主应用程序中手动构建TestSuite，可直接用FlexUnitAllTests类来代替TestSuite运行。每次FlexUnitAllTests类被重新生成时，所有包含的测试都将被重新编译和运行。使用FlexUnitAllTests的FlexUnit应用程序如下：

```

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:flexui="flexunit.flexui.*"
    creationComplete="handleCreationComplete();">
<mx:Script>
    <![CDATA[
        import flexunit.framework.TestSuite;

        private function handleCreationComplete():void
        {
            testRunner.test = FlexUnitAllTests.suite();
            testRunner.startTest();
        }
    ]]>
</mx:Script>
<flexui:TestRunnerBase id="testRunner" width="100%"
    height="100%"/>
</mx:Application>

```

按规定所有名为Test\*.as 或 \*Test.as都被包括进生成的TestSuite中。你可以创建一个过滤文件用正则表达式重新指定哪些文件将被包括进去。过滤文件的每一行指定一个独立的正则表达式，对每个文件进行匹配。如果文件名匹配任何一个正则表达式，那该文件将被包括进去。过滤文件的例子如下：

/mx/containers/\*Test.as

RegExpTest.as

第一行包含 /mx/containers/ 目录下的任何目录下的所有测试。第二行包括一个名为

RegExpTest.as的测试文件。

如上面的规则被存储在filters.txt文件，可以这样调用：

```
java -cp ~/Antennae/lib/arc-flexunit2.jar  
com.allurent.flexunit2.framework.AllTestsFileGenerator  
~/FlexCookbook/src/ filters.txt >  
~/FlexCookbook/src/FlexUnitAllTests.as
```

```
java -cp C:\Antennae\lib\arc-flexunit2.jar  
com.allurent.flexunit2.framework.AllTestsFileGenerator  
C:\FlexCookbook\src\ filters.txt >  
C:\FlexCookbook\src\FlexUnitAllTests.as
```

第二十五章. 编译与调试 (ASer@欢乐学)

编译Flex程序最常用的方法是使用Flex Builder或者是在命令行调用MXML编译器(mxmlc)。不过，还有不少其他的工具也可以完成编译程序、移动文件或者调用应用程序的任务。比如，make、Ant、及Rake，你可以仅仅使用一个命令就能调用它们完成整个编译和部署的任务。

Flex中的调试是借助于Debug版的Flash Player完成的，因为debug版的Flash Player能让我们看到trace语句的输出结果。在Flex Builder 3中，你可以逐行执行并查看变量的值。另外，Flex Builder 3还提供了一个新的视图窗口用来显示内存的使用情况以及对象的生成和销毁。除了Flex Builder 3，还有一些开源工具供你选择。比如，使用Firebug + Console.as或者Xray，你可以监视对象的值；使用FlashTracer或者Output Panel工具你一样可以看到trace语句的输出结果。本章将讲述使用Flex Builder和Xray、FlashTracer调试以及查看输出结果。

## 25.1节. 在Flex Builder外使用Trace语句

### 25.1.1. 问题

我想借助trace语句帮助调试程序，但是我没有Flex Builder 3。

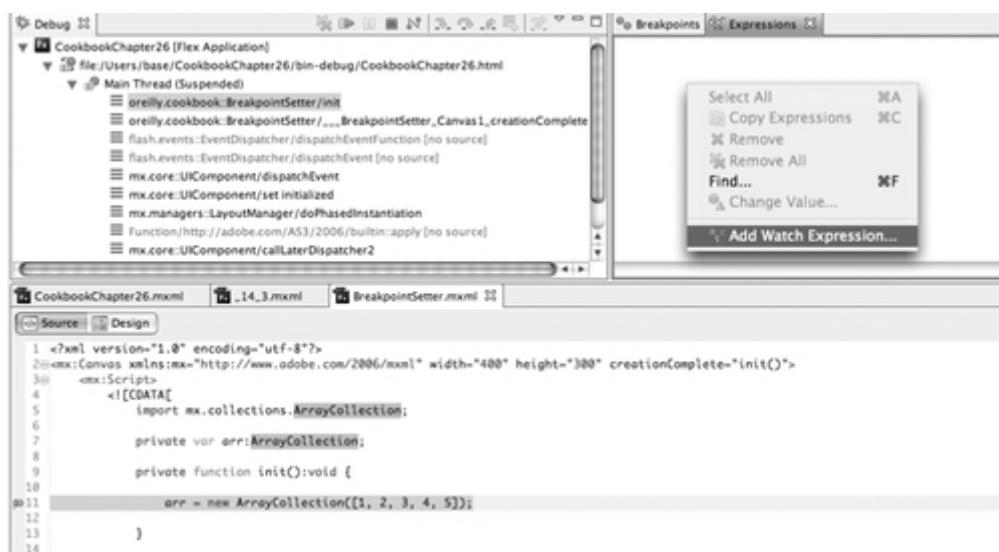
### 25.1.2. 解决办法

下载并使用一个开源的trace工具。

### 25.1.3. 讨论

自从Adobe开放Flex 3de库和编译器后，开发者有了更多查看trace语句输出结果的选择。已经不再限制在必须使用Flash IDE或Flex Builder IDE了；现在你可以使用下面几种工具。比如，Xray（John Grden开发）在Flash内创建trace语句视图。Xray不仅仅显示应用程序运行中的trace结果，还能查看运行中的对象（图25-1）。

图 25-1. 使用Xray查看输出结果



第三种选择是使用FlashTrace (Alessandro Crugnola开发)。它是安装在Firefox浏览器中的插件，可以让你看到运行中应用程序的输出结果。如果需要，还可以把结果输出到文件中。

可以中<http://osflash.org/xray#downloads>下载Xray，从<http://www.sephiroth.it/firefox>下载FlashTrace。

## 25.2节. 使用组件编译器

### 25.2.1. 问题

我想把一个Flex组件编译到SWC文件中，以便用于运行时共享库 (RSL)。

### 25.2.2. 解决办法

使用组件编译器 (compc)，然后使用命令行参数或者把一个XML配置文件作为加载配置参数传递给compc编译。

### 25.2.3. 讨论

使用下面的语法调用组件编译器compc：

代码：

```
>compc -source-path . -include-classes oreilly.cookbook.foo -output example.swc
```

下面是compc用到的一些重要的参数：

**-benchmark**

输出性能标准。

**-compiler.debug**

指定生成的SWC是否应该包含调试信息和功能。

**-compiler.external-library-path [path-element] [...]**

指定编译时要进行链接检查的SWC文件或目录。

**-compiler.include-libraries [library] [...]**

指定库 (SWC)。

**-compiler.library-path [path-element] [...]**

指定包含要编译的SWC文件的SWC文件或目录。

**-compiler.locale [locale-element] [...]**

为国际化指定本地地点。

**-compiler.optimize**

允许优化SWF。

**-compiler.services <filename>**

给出Flex数据服务配置文件的路径。

**-compiler.theme [filename] [...]**

列出应用程序中所有当作主题使用的CSS或者SWC文件。

**-compiler.use-resource-bundle-metadata**

指定资源包是否包含到应用程序中。

**-include-classes [class] [...]**

指定RSL中应该包含的所有类；可以重复多次或使用通配符列出路径。

**-include-file <name><path>**

指定RSL中应该包含的所有文件；可以重复多次或者使用通配符列出路径。

**-include-resource-bundles [bundle] [...]**

设定是否应该包含本地化资源包。

**-load-config <filename>**

加载配置文件。

**-output <filename>**

设定compc生成的文件的名称和位置。

**-runtime-shared-libraries [url] [...]**

指示本次编译中compc生成的RSL中应该包含的所有外部RSL。

**-runtime-shared-library-path [path-element] [rsl-url] [policy-file-url] [rsl-url] [policy-file-url]**

设定应用程序使用的RSL的位置和其他信息。

**-use-network**

指示SWC是否能访问网络资源。

把多个类编译到运行时共享库中的命令可能会很长。为了简化，你可以使用配置文件或者清单文件。

像MXML编译器（mxmcl）一样，你可以通过load-config参数为compc指定使用的配置文件。同样跟mxmcl一样，compc也会自动加载一个名字叫做flex-config.xml的默认配置文件。如果你想使用flex-config.xml的所有内容（它里面的许多参数都是必须的），那么可以使用+=运算符在默认的配置基础上添加新的配置文件：

`>compc -load-config+=configuration.xml`

所有的参数都可以设定在XML文档里，并用–load-config传递给compc：

`<include-sources>src/.</include-sources>`

## 25.3节. 安装Flex Ant Tasks

### 25.3.1. 问题

我想使用Flex 3 SDK中的Flex Ant。

### 25.3.2. 解决办法

拷贝flex\_ant/lib/flexTasks.jar到Ant的库目录 ({ANT\_root}/lib)。

### 25.3.3. 讨论

为了确保Ant总是能够访问由Flex 3 SDK提供的Flex Ant Tasks库中的所有任务，你必须把任务拷贝到Ant安装目录下的lib目录下。如果你不把文件拷到lib目录下，那么当你制作项目XML文件时必须使用Ant的-lib参数设定它。

## 25.4节。在Flex Ant Tasks中使用compc和mxmclc任务

### 25.4.1. 问题

我想使用包含在Flex Ant Tasks中的mxmclc或compc任务来简化应用程序的编译和使用Ant。

### 25.4.2. 解决办法

把Flex Ant tasks安装到你的Ant库中，然后使用<mxmclc>或<compc>标签，并把要传递给编译器的参数都放到标签中。

### 25.4.3. 讨论

Flex Ant tasks通过提供给开发者预置的常用任务使用，大大的简化了使用Ant编译Flex应用程序的过程。mxmclc或compc命令行使用的所有参数都可以传递给Flex Ant Task。比如，在声明了mxmclc任务后，你可以像下面这样声明输出文件：

代码：

```
<mxmclc file="C:/Flex/projects/app/App.mxml" output="C:/Flex/projects/bin/App.swf">
```

使用mxmclc Ant 任务后就不用再指定mxmclc的位置和所有运行需要的参数了，这样就节省了时间，并且使得你的构建文件更易读。更多参数可以像下面这样设置：

代码：

```
<!-- Get default compiler options. -->
<load-config filename="${FLEX_HOME}/frameworks/flex-config.xml"/>
<!-- List of path elements that form the roots of ActionScript class hierarchies.
-->
<source-path path-element="${FLEX_HOME}/frameworks"/>
<!-- List of SWC files or directories that contain SWC files. -->
<compiler.library-path dir="${FLEX_HOME}/frameworks" append="true">
    <include name="libs" />
    <include name=".bundles/{locale}" />
</compiler.library-path>
</mxmclc>
```

Flex Ant Tasks的<compc>任务也是如此；compc的所有设置都可以传递给<compc>任务：

```
<compc output="${output}/mylib.swc" locale="en_US">
```

## 25.5节. 编译和部署使用RSL的Flex应用程序

### 25.5.1. 问题

我需要部署一个使用了一个或多个运行时共享库（RSL）的Flex程序。

### 25.5.2. 解决办法

在应用程序编译后使用external-library-path指定RSL的位置。

### 25.5.3. 讨论

当Flex程序初始化时，它需要知道所有需要的运行时共享库的位置。external-library-path包含了这些信息；把它传递给编译器，这样Flash Player就能在实例化组件或类之前马上加载RSL的字节而不需要加载一个独立的SWF。

在使用RSL文件前，你必须先创建一个RSL。RSL保存在应用程序运行期访问的SWC文件中。SWC RSL文件由compc编译，SWF文件由mxmlc编译。为了让应用程序能用RSL，必须通过runtime-shared-libraries指定RSL的位置并传递给mxmlc编译器。在本例中，使用Ant编译SWC以及将使用该SWC的SWF，也就是说，我们将使用compc和mxmlc。在Ant将要使用的build.xml文件中，需要以变量的形式声明这两个编译器：

```
<property name="mxmlc" value="C:\FlexSDK\bin\mxmlc.exe"/>
<property name="compc" value="C:\FlexSDK\bin\compc.exe"/>
```

然后，使用compc编译应用程序将要访问的RSL，使用move任务将它放到application/rsl目录下：

```
<target name="compileRSL">
  <exec executable="${compc}">
    <arg line="-load-config+=rsl/configuration.xml" />
  </exec>
  <mkdir dir="application/rsl" />
  <move file="example.swc" todir="application/rsl" />
  <unzip src="application/rsl/example.swc" dest="application/rsl/" />
</target>
```

然后使用mxmlc编译SWF。注意我们将把一个名为configuration.xml的XML文件通过-load-config参数传递给编译器。该文件包含了应用程序的所有设置，包括RSL的位置：

```
<target name="compileApplication">
  <exec executable="${mxmlc}">
    <arg line="-load-config+=application/configuration.xml" />
  </exec>
</target>
```

```
<target name="compileAll" depends="compileRSL,compileApplication">
</target>
```

注意两种命令行调用都使用包含运行时共享库路径的configuration.xml文件:

```
<flex-config>
<compiler>
<external-library-path>
<path-element>example.swc</path-element>
</external-library-path>
</compiler>
<file-specs>
<path-element>RSLClientTest.mxml</path-element>
</file-specs>
<runtime-shared-libraries>
<url>example.swf</url>
</runtime-shared-libraries>
</flex-config>
```

代替命令行调用mxmlc添加的external-library-path项:

```
mxmlc -external-library-path=example.swc
```

使用load-config把configuration.xml传递给编译器，每个设置都从这个XML文件中读取。

传递给compc类似下面的文件:

```
<flex-config>
<compiler>
<source-path>
<path-element>.</path-element>
</source-path>
</compiler>
<output>example.swc</output>
<include-classes>
<class>oreilly.cookbook.shared.*</class>
</include-classes>
</flex-config>
```

本例中的Ant文件完整内容如下:

```
<?xml version="1.0"?>
```

```
<project name="useRSL" basedir=".">  
  
    <property name="mxmlc" value="C:\FlexSDK\bin\mxmlc.exe"/>  
    <property name="compc" value="C:\FlexSDK\bin\compc.exe"/>  
  
    <target name="compileRSL">  
        <exec executable="\${compc}">  
            <arg line="-load-config+=rsl/configuration.xml" />  
        </exec>  
        <mkdir dir="application/rsl" />  
        <move file="example.swc" todir="application/rsl" />  
        <unzip src="application/rsl/example.swc" dest="application/rsl/" />  
    </target>  
  
    <target name="compileApplication">  
        <exec executable="\${mxmlc}">  
            <arg line="-load-config+=application/configuration.xml" />  
        </exec>  
    </target>  
  
    <target name="compileAll" depends="compileRSL,compileApplication">  
    </target>  
  
</project>
```

## 25.6节. 在Flex Builder调试中创建和监视表达式

### 25.6.1. 问题

在Flex程序运行时我想跟踪一个值的变化。

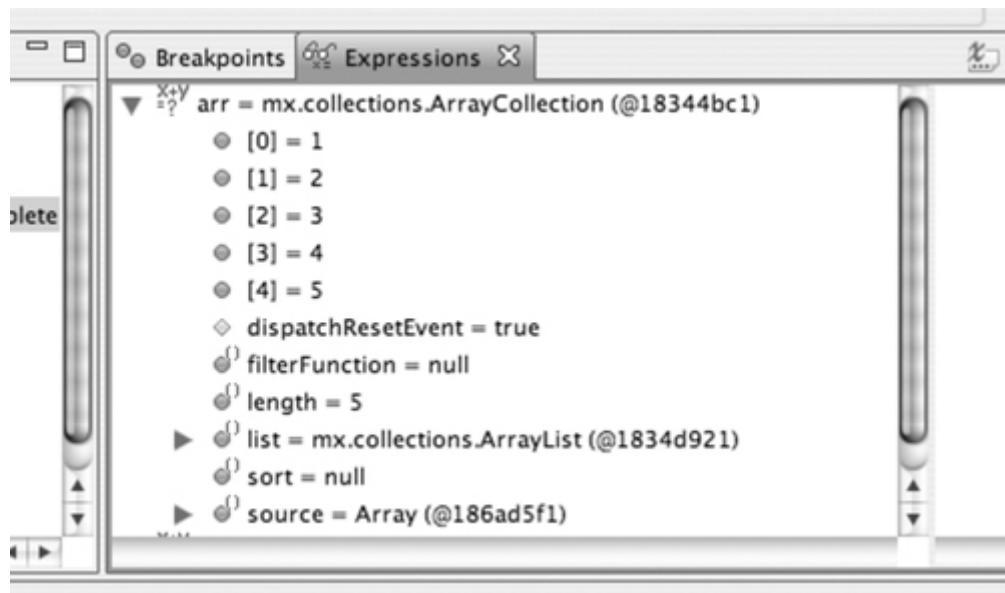
### 25.6.2. 解决办法

使用Flex Builder调试器运行你的程序，并在你要监视的变量的地方设置断点。在Flex Builder调试器的表达式窗口创建一个新的表达式。

### 25.6.3. 讨论

表达式是一个非常强大的调试工具，它能让你看到范围内的所有变量的值。设置断点的地方的所有对象都可以通过创建表达式来查看其值，像图25-2。

图 25-2. 创建一个Expression



比如，如果你在数组实例化的地方设置断点，

代码：

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300" creationComplete="init()">
<mx:Script>
<! [CDATA[
import mx.collections ArrayCollection;

private var arr:ArrayCollection;

private function init():void {

```

```

        arr = new ArrayCollection([1,2,3,4, 5]);
        //breakpoint here
    }

    private function newFunc():void {
        var newArr:ArrayCollection = new ArrayCollection([3,
            4, 5, 6]);
    }

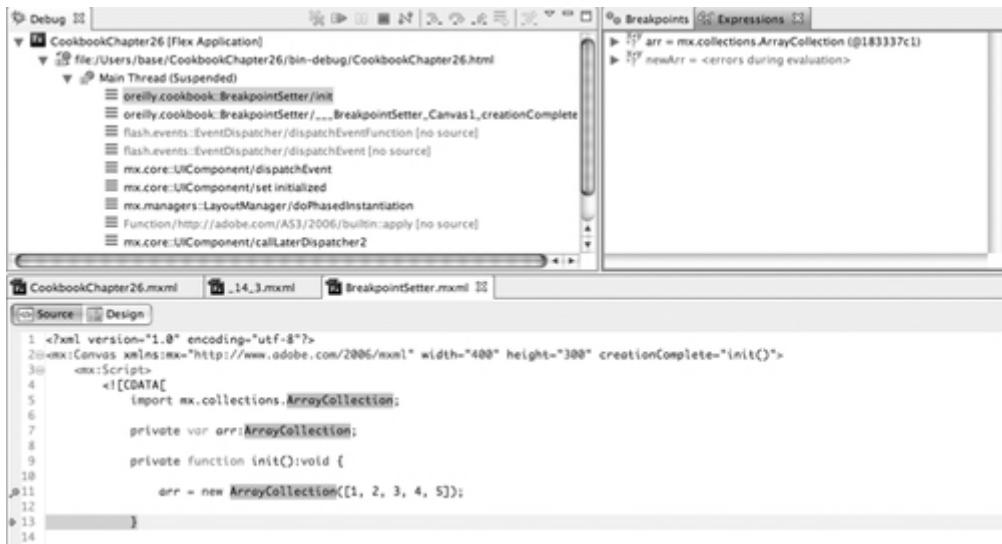
} ]>
</mx:Script>

</mx:Canvas>

```

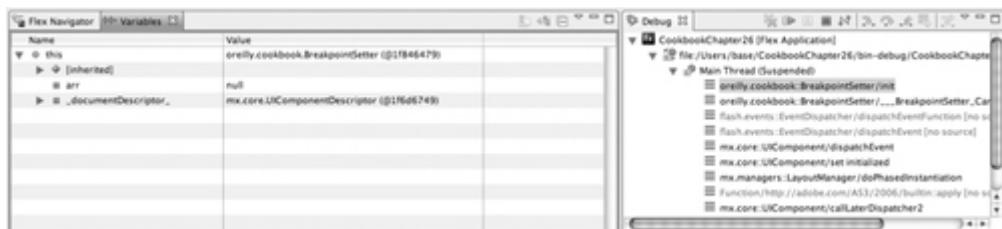
arr 的值将被认为是 null。如果你按下 F6 继续执行程序，这个表达式将被认为是 ArrayCollection，它包含一个由 5 个整数组成的数组。(图 25-3)。

图 25-3. 该表达式显示了变量的值



但是 newArr 被认为是 null，因为它不在范围内 (图 25-4)。

图 25-4. 只能计算范围内的变量的值



如果你把断点设置在 17 行，那么 newArr 和 arr 都被计算为 ArrayCollection，因为它们都在当前范围内。

## **25.7节. 在Flex Builder中安装Ant窗口**

由Ryan Taylor贡献

### **25.7.1. 问题**

在独立版本的Flex Builder中找不到Ant窗口。

### **25.7.2. 解决办法**

安装Eclipse Java开发工具。

### **25.7.3. 讨论**

要在独立版本的Flex Builder中使用Ant，你必须安装Eclipse Java开发工具。像下面这样做：

在Flex Builder菜单条中选择Help->Software Updates->Find and Install.

选择Search for New Features to Install然后点击Next。

在对话框中选择Eclipse Project Updates，然后点击Finish。

会出现一个菜单，问你从哪里下载文件。可以选择任意位置，不过最好是离你近的地方，这样下载速度快，节省时间，然后点击OK。

查看Eclipse Project Updates树形结构中各种SDK版本，直到你找到Eclipse Java Development Tools。勾选它旁边的复选框然后点Next。

当更新管理器下载完所有必需的文件后，会提示你有新功能。点击Install All。

安装完毕后，重启Flex Builder。

现在通过Window->Other Views->Ant，你就能在Flex Builder中看到Ant窗口了。

## **25.8节. 为自动通用任务创建一个Ant构建文件**

由Ryan Taylor贡献

### **25.8.1. 问题**

我想借助Ant的能力自动化常用任务，比如编译和生成文档。

### **25.8.2. 解决办法**

在需要自动化的地方创建Ant构建文件。

### **25.8.3. 讨论**

创建Ant构建文件非常简单，并且是使用Ant自动化常用任务的第一步。新建一个名为

build.xml的XML文档，并把它保存到你的项目目录下的build目录下。并不是非要把它保存到这个文件夹下，但是通常都这么做。

你的构建文件的根节点应该像下面这样：

```
<project name="MyAntTasks" basedir="."> </project>
```

你应该想为你的项目设置一个唯一的名称。这个名字会显示在Eclipse中的Ant窗口里。对于basedir属性，应该确保它指向你的项目目录。当你定义你项目文件夹里的其他文件或目录时将会经常用到basedir属性。

接下来，你可能想添加其他属性。例如，创建一个指向你项目源文件目录的属性，可以像下面这样做：

```
<project name="MyAntTasks" basedir="."> <property name="src" value="${basedir}/src" /> </project>
```

上面的例子也演示了如何使用定义过的属性，就是用\${property}。

如果你发现你需要定义很多属性，但是你又想让你的构造文件尽量清晰，那么你可以把属性定义到单独的文件中。要做到这样，新建一个名为build.properties的文本文件，并把它保存到跟build.xml同一目录下。在这个文件中，声明属性就想下面一样简单：

```
src="${basedir}/src"
```

这就是需要做的所有事情。一些有用的属性是声明src目录、bin目录和Flex 3 SDK目录的路径。你会很快理解你需要的是什么。现在，你就可以开始为你的构建文件添加任务了。

## 25.9节. 使用mxmcl和Ant编译Flex应用程序

由Ryan Taylor贡献

### 25.9.1. 问题

我想为Ant建造文件添加编译程序的任务。

### 25.9.2. 解决办法

为你的使用MXML编译器编译文件的Ant建造文件添加可执行的任务。

### 25.9.3. 讨论

编译对象是你要添加到你的Ant建造文件中的最常见和有用的对象。Flex应用程序是用Flex 3 SDK中的免费的命令行编译器mxmcl编译的。在你建造文件中添加了要编译的目标后，就可以自动化建造过程了：不用你再打开命令提示符或终端，Ant会自动编译所有的文件。

MXML编译器（mxmcl）有多种格式。你可以生成一个目标，然后使用可执行版的：

代码：

```

<!-- COMPILE MAIN -->

<target name="compileMain" description="Compiles the main application files.">
    <echo>Compiling '${bin.dir}/main.swf...</echo>
    <exec executable="${FLEX_HOME}/bin/mxmlc.exe" spawn="false">
        <arg line="-source-path '${src.dir}'" />
        <arg line="-library-path ${FLEX_HOME}/frameworks" />
        <arg line="${src.dir}/main.mxml" />
        <arg line="-output '${bin.dir}/main.swf'" />
    </exec>
</target>

```

或者，你可以编写一个类似下面的任务，然后使用Java版的：

代码：

```

<!-- COMPILE MAIN -->

<target name="compileMain" description="Compiles the main application files.">
    <echo>Compiling '${bin.dir}/main.swf...</echo>
    <java jar="${FLEX_HOME}/lib/mxmlc.jar" fork="true" failonerror="true">
        <arg value="+flexlib=${FLEX_HOME}/frameworks" />
        <arg value="-file-specs='${src.dir}/main.mxml'" />
        <arg value="-output='${bin.dir}/main.swf'" />
    </java>
</target>

```

最后（或许是最好的）方式是使用Flex 3 SDK中的mxmlc任务。在25.3节中已经讲述了如何安装。为了能访问你的建造文件中的信息，你应该首先添加一个任务定义：

代码：

```

<!-- TASK DEFINITIONS -->

<taskdef resource="flexTasks.tasks" classpath="${FLEX_HOME}/ant/lib/flexTasks.jar" />

```

导入Flex任务后，你可以使用更加直观的语法编译，在诸如Eclipse一样的工具中显示错误提示。比如：

代码：

```

<!-- COMPILE MAIN -->

<target name="compileMain" description="Compiles the main application files.">

```

```

<echo>Compiling ${bin.dir}/main.swf...</echo>
<mxmlc file="${src.dir}/main.mxml" output="${bin.dir}/main.swf">
    <source-path path-element="${src.dir}" />
</mxmlc>
</target>

```

在所有的例子中，应用了相同的基本规则。你应该定义指向项目src和bin目录的属性，也需要定义指向Flex 3 SDK的属性。例子中的所有属性的名字都可以改变，除了FLEX\_HOME这个名字是固定的。FLEX\_HOME属性必须在使用mxmlc任务前设置为Flex 3 SDK的根目录。如果你使用EXE或JAR版本的mxmlc，你可以使用FLEX\_HOME以外的属性名称。

使用Ant编译你的项目的真正好处是它能把所有的目标链在一起。例如，你可能会创建一个名为compileAll的目标，它会一个接一个的调用一些单独编译的目标：

代码：

```

<!-- COMPILE ALL -->
<target name="compileAll" description="Compiles all application files." depends="compileMain, compileNavigation, compileGallery, compileLibrary">
    <echo>Finishing compile process...</echo>
</target>

```

在一开始这看起来有些恐怖；但是当你熟悉Ant和配置文件后，你会发现它会大大提高你的工作效率。通过使用第三方工具比如Ant来自动化你的编译过程，你已经不会被困在只能使用单一的开发工具上了。你可以很容易的在你选择的开发工具，比如Flex Builder，FDT，TextMate，或者FlashDevelop上调用Ant来创建你的项目。

## 25.10节. 使用ASDoc和Ant生成文档

由Ryan Taylor贡献

### 25.10.1. 问题

我想为应用程序生成文档。

### 25.10.2. 解决办法

首先创建一个使用ASDoc（包含在Flex 3 SDK中）生成文档的Ant建造文件，然后为它添加一个可执行的任务。

### 25.10.3. 讨论

ASDoc是Flex 3 SDK中的一个免费的命令行工具。如果你看过Adobe的在线帮助文档，那么你已经熟悉了ASDoc生成的文档风格。虽然在命令提示符下使用它并不十分的困难，但是为你的Ant建造文件添加一个任务来自动化流程会更好一些。

在为生成文档创建目标之前，先添加一个清理你的文档目录的目标会非常有用。你定义docs.dir属性时，把它指向你的项目的docs目录就可以了：

代码：

```
<!-- CLEAN DOCS -->

<target name="cleanDocs" description="Cleans out the documentation directory.">
    <echo>Cleaning '${docs.dir}'...</echo>
    <delete includeemptydirs="true">
        <fileset dir="${docs.dir}" includes="**/*" />
    </delete>
</target>
```

完成上面的工作后，就要创建用来生成文档的目标了。注意，在本例中的depends属性表示在生成文档前会先执行cleanDocs：

代码：

```
<!-- GENERATE DOCUMENTATION -->

<target name="generateDocs" description="Generates application documentation using ASD
oc." depends="cleanDocs">
    <echo>Generating documentation...</echo>
    <exec executable="${FLEX_HOME}/bin/asdoc.exe" failOnError="true">
        <arg line="-source-path ${src.dir}" />
        <arg line="-doc-sources ${src.dir}" />
        <arg line="-main-title ${docs.title}" />
        <arg line="-window-title ${docs.title}" />
        <arg line="-footer ${docs.footer}" />
        <arg line="-output ${docs.dir}" />
    </exec>
</target>
```

FLEX\_HOME属性应该指向你机器上的Flex 3 SDK的根目录。src.dir和doc.dir属性分别代表你的项目的src和docs目录。最后但并非不重要的是docs.title和docs.footer属性，它们用来设置文档的标题和底部。通常文档的标题是你的项目名称。底部是放置版权和URL的好地方。

即便你没有写任何注释，ASDoc会根据你的代码成功的生成文档。当然，强烈推荐按照Java文档注释的格式在你的代码中写上注释。这样不仅能生成详尽的文档，而且能帮助不熟悉你代码的开发者深入理解代码。

## 25.11节. 使用Rake编译Flex应用程序

### 25.11.1. 问题

我想使用Rake（Ruby工具）编译Flex应用程序。

### 25.11.2. 解决办法

如果你还没有Ruby 1.9，那么先下载并安装它，然后下载并安装Rake。

### 25.11.3. 讨论

虽然是在Ruby中编写的，但是C++和C程序员会非常熟悉Ruby的功能。在下载并安装Ruby和Rake后，你可以编写一个类似下面的简单的Rake文件：

代码：

```
task :default do
  DEV_ROOT = "/Users/base/flex_development"
  PUBLIC = "#{DEV_ROOT}/bin"
  FLEX_ROOT = "#{DEV_ROOT}/src"
  system "/Developer/SDKs/Flex/bin/mxmlc --show-actionscript-warnings=true --strict=true -file-specs #{FLEX_ROOT}/App.mxml"
  system "cp #{FLEX_ROOT}/App.swf #{PUBLIC}/App.swf"
end
```

Rake中的所有任务都跟Ant中的相似，意思是，它们都定义一个要完成的动作。默认的动作总是会被执行，额外的、在其他不同任务中的动作可以被有选择的执行。在任务中，可以声明变量，可以使用system参数，如下：

代码：

```
system "/Developer/SDKs/Flex/bin/mxmlc --show-actionscript-warnings=true --strict=true -file-specs #{FLEX_ROOT}/App.mxml"
```

这实际上是调用MXML编译器来生成SWF文件。由于在Rake任务中，如果前一个任务没有结束，那么后面的任务不会被执行，下面的一行可以假设已经生成了SWF文件并可以拷贝

到新位置：

```
system "cp #{FLEX_ROOT}/App.swf #{PUBLIC}/App.swf"
```

这个Rake文件还声明了一个变量，用来指定存放文件的正确位置。现在可以用任意名字保存这个文件，并使用Rake命令在命令行执行它。如果你把它保存为Rakefile，你可以通过键入下面的内容运行它：

```
rake Rakefile
```

## 25.12节. 使用ExpressInstall

### 25.12.1. 问题

你想为没有安装Flash Player正确版本的用户安装Flash Player。

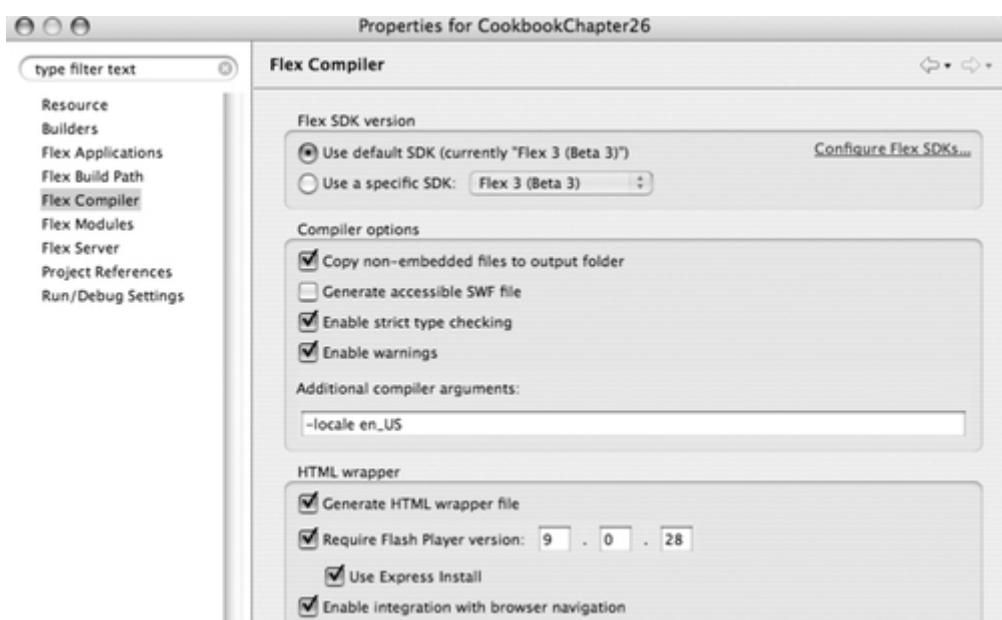
### 25.12.2. 解决办法

在编译的时候设置ExpressInstall，这样SWF文件会重定向到Adobe的网站，在那里就可以安装Flash Player。

### 25.12.3. 讨论

要使用ExpressInstall，你可以在Flex Builder中设置使用ExpressInstall。(图25-5)。

图 25-5. 设置Express Install 参数



如果不是使用Flex Builder开发，那么可以在HTML页面中的Object标签和embed标签中设置pluginspage属性：

```
pluginspage="http://www.adobe.com/go/getflashplayer"
```

对于基于Netscape的浏览器比如FireFoxe，<embed>语句像下面一样：

代码：

```
<embed src="CookbookChapter26.swf" id="CookbookChapter26" quality="high"
bgcolor="#869ca7" name="CookbookChapter26" allowscriptaccess="sameDomain"
pluginspage="http://www.adobe.com/go/getflashplayer" type="application/x-shockwave-flash" align="middle"
height="100%" width="100%">>
```

## 25.13节. 使用**Flex Builder 3**的**Memory Profiling**查看内存快照

### 25.13.1. 问题

我想在运行时查看Flash Player内存中的所有对象。

### 25.13.2. 解决办法

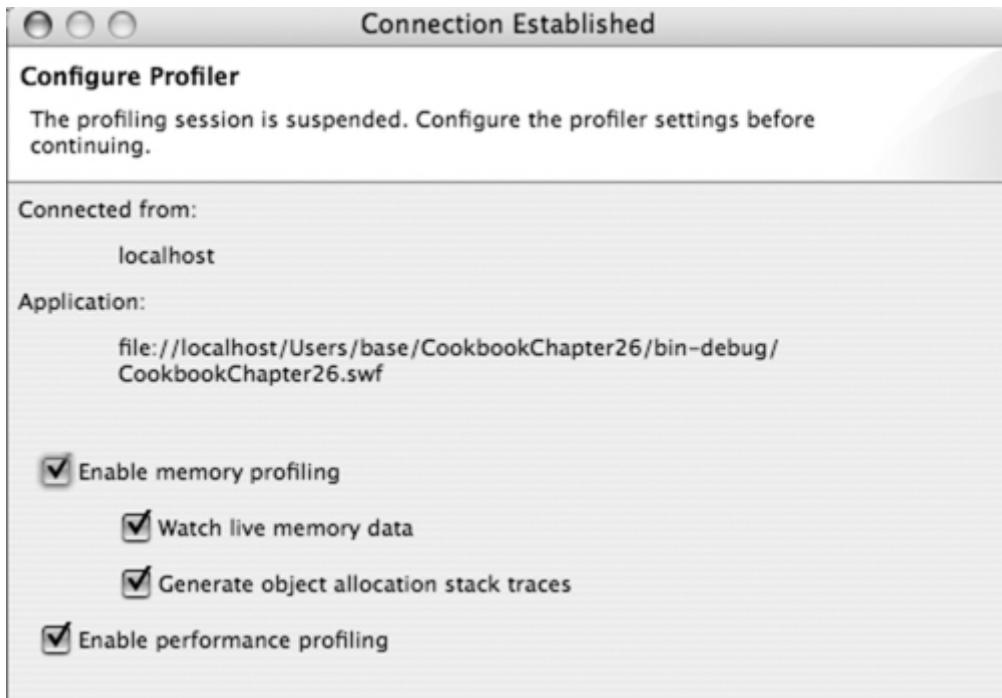
使用Flex Builder 3的Memory模拟视图运行你的程序并观察对象的创建和销毁。

### 25.13.3. 讨论

Flex Profiler是Flex Builder 3中新增的、允许你查看分配和清理内存及对象的强大工具。它通过一个本地的Socket连接到你的应用程序。如果你的防毒软件禁用Socket通信的话，你需要关闭它才能正常使用。

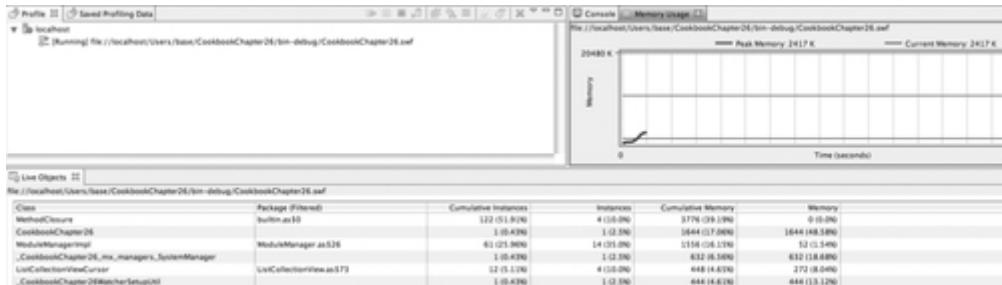
当运行Profiler后，它每隔几毫秒就生成一次数据快照，并记录Flash Player的状态，就像抽样过程一样。通过分析这些数据，Profiler可以显示你的应用程序中的每个操作。Profiler记录了这些操作的执行时间，以及Flash Player中对象使用内存的情况。当一个应用程序在Profiler中运行时，你会看到建立连接的对话框 (图25-6)。在这里你可以允许内存模拟以便查找发生问题的区域，允许性能模拟可以帮助改善应用程序的性能。

图 25-6. 选择一个模仿类型



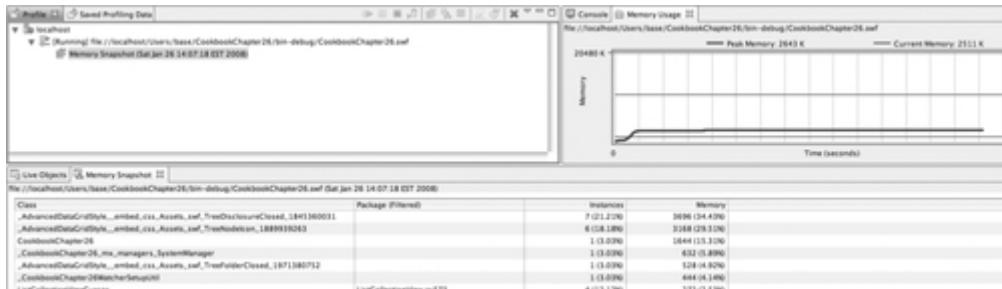
如果你勾选了Watch Live Memory Data复选框，模拟视图会显示Flash Player中的对象的实时图 (图25-7)。

图 25-7. Flex模拟窗口中的实时对象和内存数据



Profiler提供了内存快照，这些快照可以在任何时间获得，并提供关于任意对象的实例数量及占用内存情况的详细资料 (图25-8)。

图 25-8. 在一幅内存快照中查看实例数量和内存使用情况



最后，通过比较不同时间的内存快照，你可以找出Loitering对象，就是那些在第一个内存快照后创建，在第二快照中已经存在的对象。关于类名称、内存大小、实例数量的信息都包含在Loitering Object窗口中(图25-9)。

图 25-9. 在Loitering Object窗口中查看所有生成的对象



# 第二十六章. 配置，国际化和打印 (常青)

为了确保你的应用程序能适应更广泛的用户群，Flex 3提供了辅助功能，国际化支持和打印选项。比如，如果你的项目需要遵循辅助功能标准，你会发现屏幕阅读器和键盘Tab顺序将帮助那些视障人士，因为他们使用“点击”设备是很困难的。Flex的国际化和本地化工具集在Flex3中有了很大改进。新的本地化特性引入了内建国际化资源管理器，运行时区域检测和运行时资源模块请求等。Flex 3 可以重复打印Flex组件及其数据，以及多页内容。

这一章包含的小节将带你浏览各种各样的打印输出格式，包括使用PrintDataGrid打印跨页的自定义项渲染器的组件。一些小节介绍如何显示非西方字符，检测屏幕读者，定义Tab顺序，使你的程序适合视障人士使用。最后介绍一些应用程序本地化技术。

## 26.1节. 在程序中添加国际化字符

### 26.1.1. 问题

我想在程序中显示字为基础的语言文本，比如中文或韩文。

### 26.1.2. 解决办法

在Flash Player中嵌入合适的字体。

### 26.1.3. 讨论

Flex应用程序能显示非西方字符，包括Unicode编码文本，比如汉字或韩文字符，支持这些字符在Flash Player里显示。和西方字体一样，开发者也可以嵌入这些字体到程序中，但是你也要知道，这样做有个后果，就是数量众多的字符会使得你的SWF文件变得很庞大。要达到这种效果，你需要在SWF的大小和完美的文字渲染中做出平衡。

下面的例子，ChineseFonts.mxml，例举说明。

注意

当你打开光盘中的ChineseFonts.mxml文件时，只有在你的系统中包含这些字体时才能看到汉字。而嵌入字体可以在任何系统里正常显示。

Code View:

```
<Application xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute">
<mx:Style>
@font-face
{
    src: local("LiSong Pro");
    fontFamily: EmbeddedChinese;
    fontStyle: normal;
```

```

        fontWeight: normal;
    }
</mx:Style>
<mx:Form>
<mx:FormItem label="System Font">
    <mx:Label text="快的棕色狐狸慢慢地跳過了懶惰灰色灰鼠" />
</mx:FormItem>
<mx:FormItem label="Embedded Font">
    <mx:Label fontFamily="EmbeddedChinese" text="快的棕色狐狸慢慢地跳  
過了懶惰灰色灰鼠" />
</mx:FormItem>
</mx:Form>
</mx:Application>

```

虽然MXML源文件是以Unicode编码，但是载入的XML数据包含简体中文字符：

Code View:

```

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute">
    <mx:Style>
        @font-face
        {
            src: local("LiSong Pro");
            fontFamily: EmbeddedChinese;
            fontStyle: normal;
            fontWeight: normal;
        }
    </mx:Style>
    <mx:XML source="books.xml" id="booksData" />
    <mx:VBox fontFamily="EmbeddedChinese">
        <mx:Repeater id="iterator" dataProvider="{booksData.book}">
            <mx:VBox backgroundColor="#0xffffffff">
                <mx:Label text="{iterator.currentItem.@title}" />
                <mx:Text width="200"
                    text="{iterator.currentItem.toString()}" />
                <mx:HRule width="200" />
            </mx:VBox>
        </mx:Repeater>
    </mx:VBox>
</mx:Application>

```

下面的就是例子中用到的文档：

```
<books>
```

<book title="阿波罗为Adobe 导电线开发商口袋指南">

现在您能建立和部署基于闪光的富有的互联网应用(RIAs) 对桌面使用Adobe 的导电线框架。

由阿波罗产品队的成员写，这是正式指南对于Adobe 阿波罗，新发怒平台桌面运行时间阿尔法发行从

Adobe 实验室。众多的例子说明怎么阿波罗工作因此您可能立即开始大厦RIAs 为桌面。

</book>

<book title="编程的导电线2">

编程的导电线2 谈论导电线框架在上下文。作者介绍特点以告诉读者不仅怎样，而且原因为什么

使用一个特殊特点，何时使用它，和何时不是的实用和有用的例子。这本书被写为发展专家。当书不假设观众早先工作了以一刹那技术，读者最将受益于书如果他们早先建立了基于互联网，n

tiered 应用。

</book>

<book title="ActionScript 3.0 设计样式">

如果您是老练的闪光或屈曲开发商准备好应付老练编程技术与ActionScript 3.0，这实践介绍逐步设计样式作为您通过过程。您得知各种各样的类型设计样式和修建小抽象例子在尝试

您的手之前在大厦完全的运作的应用被概述在书。

</book>

</books>

## 26.2节。使用本地化资源包

### 26.2.1. 问题

我想在应用程序中支持少量的可选语言。.

### 26.2.2. 解决办法

使用编译的资源包提供本地资源。

### 26.2.3. 讨论

对于基础的Flex应用程序本地化需求，你可以使用资源包。资源包是一些ActionScript对象，提供一个接口通过数据绑定和ActionScript代码访问由属性文件中定义的本地化内容。应用程序的每个捆绑包表示一个单独的本地化属性文件。属性文件是一个文本文件，包含本地化

属性键和关联值的列表。键值对以key=value的格式，属性文件以.properties扩展名保存。

本地化值是文本字符串，嵌入的资源可以是图像，ActionScript类引用。当本地化应用程序时，为属性文件的每一项定义一个实体以完全支持替代语言用于更新应用程序。下面的属性文件例子定义了一些英语(美国)的一些属性值：

```
#Localization resources in American English  
pageTitle=Internationalization Demo  
language=American English  
flag=Embed("assets/usa.png")  
borderSkin=ClassReference("skins.en_US.LocalizedSkin")
```

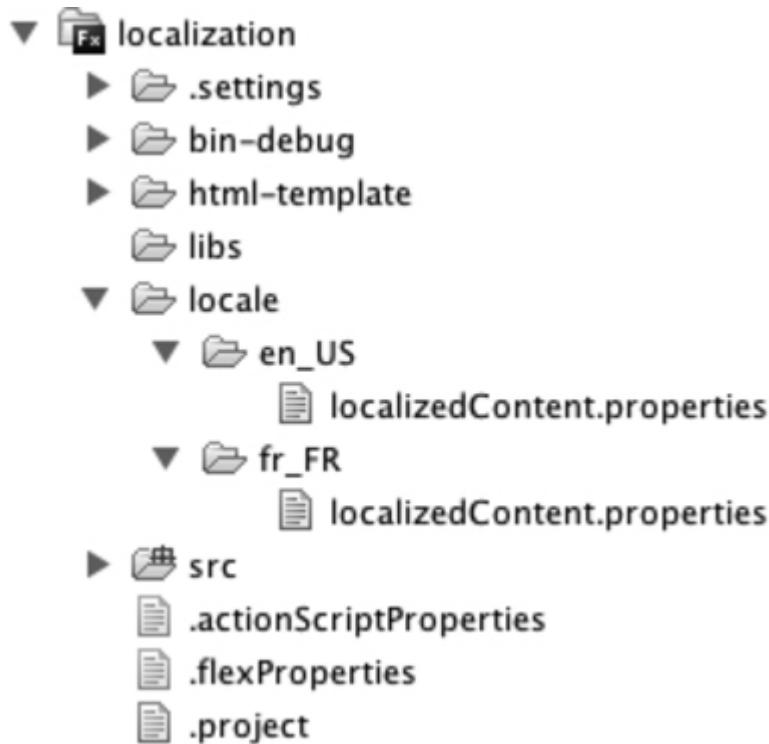
当本地化应用程序时，你需要创建一个属性文件副本，它包含对应语言的相关内容。如果你的应用程序需支持美国英语和法语，在这个例子中，你需要创建两个属性文件，包含各自的翻译文本，一个代替美国国旗的法国国旗图片引用，以及一个边框皮肤引用：

```
#Localization resources, En Francais  
pageTitle=Demo d'internationalisation  
language=Francais  
flag=Embed("assets/france.png")  
borderSkin=ClassReference("skins.fr_FR.LocalizedSkin")
```

当设置你的属性文件时，需要考虑几个因素，其中最重要的当属文件大小和应用程序复杂度。你可能希望为每个自定义组件创建一个属性文件，例如，共享资源里的相关组件包。你可能要定义一个用于全局范围的属性文件，比如包含自定义应用程序错误信息，一些按钮上使用的通用标签。

不管你如何分散本地化属性，你都需要创建一个目录结构来组织这些文件，如Figure 26-1所示，最好的方法就是用locale或localization作为目录名，下面包含用各自区域标识命名的子目录放置各自的属性文件，这样编译器就能轻松的找到这些属性文件。

**Figure 26-1. Directory structure for localization properties files**



当你构建程序时，编译器会为每个属性文件创建一个**ResourceBundle**类的子类。访问属性文件中定义项的最简单方法是使用**@Resource**指令。使用这个方法，编译器会寻找对应的属性值用以替换。使用**@Resource**指令的好处是你不必编写关于**ResourceBundle**实例的任何代码，编译器已经帮你做了所有事情。**@Resource**指令接受两个参数，一个捆包标识和一个键，它用来寻找相应属性文件中对应的属性值。例如，要引用名为**localizationProperties.properties**属性文件的**applicationTitle**属性名，使用下面的命令：

```
@Resource(key='applicationTitle', bundle='localizationProperties')
```

看下面一个完整的例子，**LocalizationResource.mxml**，定义了一个小应用程序：

Code View:

```

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute">
    <mx:Metadata>
        [ResourceBundle("localizedContent")]
    </mx:Metadata>
    <mx:VBox horizontalCenter="0"
        verticalCenter="0"
        horizontalAlign="center"
        borderSkin="@Resource(key='borderSkin',
        bundle='localizedContent')">
        <mx:Label fontSize="24" text="@Resource(key='pageTitle',
        bundle='localizedContent')"/>
        <mx:Label fontSize="24" text="@Resource(key='language',
        bundle='localizedContent')"/>
        <mx:Image source="@Resource(key='flag',
        bundle='localizedContent')"/>
    </mx:VBox>
</mx:Application>

```

```

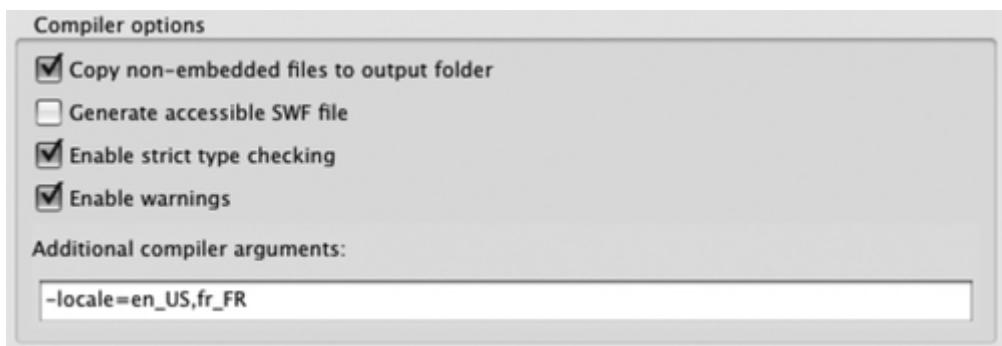
</mx:VBox>
</mx:Application>

```

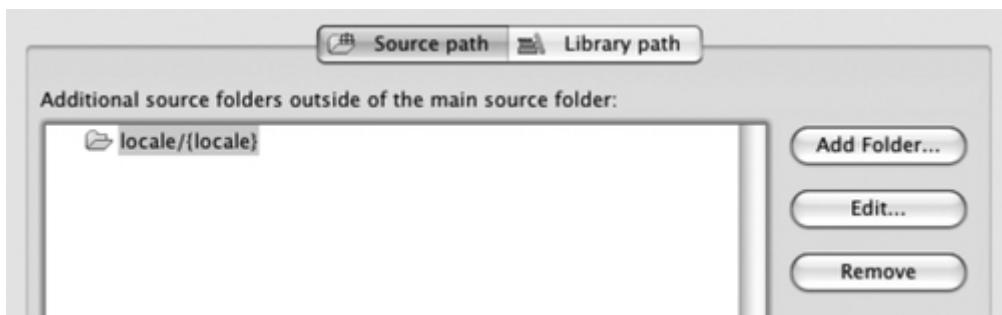
元标签[ResourceBundle]告诉编译器需要哪个资源包。这点非常重要，因为资源包在编译期被构建，支持语言所相关的所有资源都必须被编译进SWF。

编译器也需要为特定的区域构建本地化支持进行配置。在Flex Builder 3中，在工程的属性对话框中设置。在Flex Build Path面板中，在source path中加入本地化文件路径。如果你按照上面的工程目录中使用locale目录的话，那这里应该是locale/{locale}，如图Figure 26-3. 另外，在Flex Compiler面板中添加参数标识希望支持哪种语言。例如支持美国英语和法语，输入**-locale en\_US,fr\_FR** (Figure 26-2). 当编译应用程序时，编译器会寻找对应的属性文件，并用locale/en\_US和locale/fr\_FR中的属性文件替换掉source path中的表达式值。

**Figure 26-2. Localization build path entry**



**Figure 26-3. Localization compiler arguments**



在构建本地化应用程序之前你必须先本地化相关的Flex Framework内容，如错误信息。Adobe提供了一个叫copylocale工具拷贝文件到新的locale中。你需要重复此步为每个locale都拷贝一份。注意copylocale实际上没有拷贝本地化文件，它只是用于编译应用程序目的。要运行此命令，简单的传入默认本地化标识和目标本地化标识：

[Copylocale.exe en\\_US fr\\_FR](#)

Flex 3之前，你不能在运行时改变区域设置，这就意味着你需要为每个区域需求使用相应的资源包都编译一份应用程序副本。下面我们看看LocalizationResource.mxml例子编译后的效果：

**Figure 26-4. LocalizationResource.mxml with compiler argument "-locale=en\_US"**



Figure 26-5. LocalizationResource.mxml with compiler argument "-locale=fr\_FR"



## 26.3节。使用本地化资源管理器

### 26.3.1. 问题

我想支持小数量的本地化需求或者在运行时通过编程决定区域，或者由用户选择区域。

### 26.3.2. 解决办法

使用[ResourceManager](#)类支持多个区域，允许应用程序在运行时改变区域设置。

### 26.3.3. 讨论

ResourceManager类是编译器根据本地化属性文件创建资源包所用的最主要的ActionScript接口。它可以接受各种类型的资源包资源以及提供一种机制可动态设置所需区域。资源管理器是一个单例模式，管理着整个程序的区域设置。每个继承自UIComponent的类都有个protected属性名为resourceManager，它就是一个资源管理器单例的引用。

虽然@Resource指令可方便的绑定本地化内容到MXML标签上，但是对于ActionScript 组件或ActionScript方法涉及到本地化资源时它就没什么用武之地了。这种情况下，最好使用资源管理器，因为它提供了一些方法用于访问本地化数据或者数据绑定表达式的目标使用。下面的例子引用之前的LocalizationManager.mxml，用 ResourceManager方法替换26.2节的LocalizationResource.mxml中使用的@Resource 指令：

Code View:

```
<mx:VBox horizontalCenter="0" verticalCenter="0"
```

```

        horizontalAlign="center"
        borderSkin="{resourceManager.getClass('localizedContent',
            'borderSkin')}">
<mx:Label fontSize="24"
    text="{resourceManager.getString('localizedContent', 'page
        Title')}" />
<mx:Label fontSize="24"
    text="{resourceManager.getString('localizedContent', 'lang
        usage')}" />
<mx:Image source="{resourceManager.getClass('localizedContent',
            'flag')}" />
</mx:VBox>
```

这些方法的名称非常接近于载入的资源类型，其中的参数即资源绑定名称和所需属性的键值也和@Resource指令相似。

使用资源管理器绑定本地化属性值到它们的目标上还有另一个好处：Flex 3可以在运行时动态改变区域设置，现在不需要强制为每个区域都编译一个SWF文件了。方法的绑定属性可以使得应用程序能自己更新改变后的区域设置。在LocalizationManager.mxml例子中，提供按钮让用户选择英语或法语：

Code View:

```

<mx:HBox>
    <mx:Button label="In English"
        click="resourceManager.localeChain = ['en_US']" />
    <mx:Button label="En Francais"
        click="resourceManager.localeChain = ['fr_FR']" />
</mx:HBox>
```

在这个例子中，localeChain属性根据用户选择的按钮而被重置。localeChain属性是一个字符串数组，表示所需区域的顺序列表。这是非常有用的，例如，应用程序将从浏览器通过Accept-Language HTTP 标头或主机操作系统的语言首选项接收有关用户的语言首选项的信息，英国的用户首选的区域设置是 en\_GB，但用户也可以接受 en\_US。当资源管理器的方法被调用时，它将根据在localeChain中定义的指定名称顺序寻找资源绑定包，因此，要这样设置localeChain属性提供en\_US给英国用户选择：

```
resourceManager.localeChain = ["en_GB", "en_US"];
```

## 26.4节. 使用本地化资源模块

### 26.4.1. 问题

我的应用程序需要支持多种区域设置。

## 26.4.2. 解决办法

在运行时根据需要使用资源模块载入特定本地化资源。

## 26.4.3. 讨论

根据程序需要支持的语言，将每个本地化资源包都编译进应用程序将使得SWF文件大小倍增，而绝大多数用户其实只需要一种区域语言，这将浪费大量的时间用在SWF文件的下载上。还好Flex 3添加一种功能可以编译资源包，这被称作资源模块，它可以在运行时动态加载到应用程序中。你可以先检测用户的区域设置，然后载入相应的资源模块。

要想根据本地化属性文件构建资源模块，首先要检测应用程序需要什么资源，这不仅包括你定义的资源，而且也包括Flex框架所需的资源。你可以使用mxmlc编译器分析应用程序，输出所需资源列表。这可以修改Flex Builder 3的项目属性对话框中的额外编译器参数达到目的。不过命令行模式下也照样可做到，而且更方便快捷。当调用编译器时，没有指定区域，一个编译器分析结果输出的保存文件和MXML文件名称：

```
mxmlc -locale= -resource-bundle-list=resources.txt ResourceModules.mxml
```

当命令完成后，resources.txt的内容类似这样：

```
bundles = containers controls core effects localizedContent skins styles
```

这个输出内容告诉编译器哪些资源包将被构建到资源模块中。如果使用资源模块编译应用程序，必须使用命令行编译。指定source path为本地化属性文件，资源包列表和上面一样。例如为26.4的 ResourceModules.mxml 例子编译：

Code View:

```
mxmlc -locale=en_US -source-path=.,locale/{locale}  
-include-resource-bundles=containers,controls,core,effects,localizedContent,skins,styles  
-output en_US_resources.swf
```

编译为法语模块：

Code View:

```
mxmlc -locale=fr_FR -source-path=.,locale/{locale}  
-include-resource-bundles=containers,controls,core,effects,localizedContent,skins,styles  
-output fr_FR_resources.swf
```

这个命令有几项内容。起先，使用-locale参数定义区域，其二，source path参数可能已经很熟悉了，这个例子中包括当前目录(.)。这个例子中localizedContent属性文件包含一个嵌入类引用，如果source path中没有应用程序根目录，编译器可能就找不到类引用。注意假定你在工程源文件目录下调用了mxmlc，接着include-resource-bundles 参数设置为之前例子生成的列表内容，这个列表有逗号分割符和未含空格的包名组成。最后，这个例子告诉编译器输出文件名称为en\_US\_resources.swf。你可以命名为其他的。但最好是根据区域标识取名称，这样你就可以根据文件名编程实现载入合适的资源模块了。

当你使用mxmlc编译资源模块后，嵌入资源如图片的引用路径将是相对于本地化属性文件的位置。也就是说，如果应用程序使用编译的资源包资源路径是相对于工程源文件目录的。

程序代码中，使用资源管理器的loadResourceModule方法载入资源模块。给此方法传递一个

标识你要使用的资源模块URL，这个方法和其他ActionScript载入方法机理类似，比如SWFLoader或传统模块。一个需要SWF文件的请求发送到服务器，然后SWF被浏览器下载。如果所需资源模块来自其他域这需要cross-domain策略文件。你必须等资源模块下载完成后才可使用，当子模块可以使用时，ResourceEvent事件会被触发。你可以监听这些事件如ResourceEvent.COMPLETE。loadResourceModule方法返回一个实现IEventDispatcher接口的对象引用。下面摘录自ResourceModules.mxml例子代码演示如何载入和使用资源模块：

Code View:

```
import mx.events.ResourceEvent;
import mx.resources.ResourceManager;

private var selectedLocale:String;

private function setAppLocale(locale:String):void
{
    this.selectedLocale = locale;
    if (resourceManager.getLocales().indexOf(locale) == -1)
    {
        var dispatcher:IEventDispatcher =
        resourceManager.loadResourceModule(locale + "_resources.swf");
        dispatcher.addEventListener(ResourceEvent.COMPLETE,
onResourceLoaded);
    }
    else
    {
        onResourceLoaded(null);
    }
}

private function onResourceLoaded(e:ResourceEvent):void
{
    resourceManager.localeChain = [this.selectedLocale];
    views.selectedIndex = 1;
    contentBackground.setStyle("borderSkin",
        resourceManager.getClass('localizedContent', 'borderSkin'));
    contentBackground.invalidateDisplayList();
    contentBackground.validateNow();
}
```

在这个例子中，用户可在美国英语和法语中做出选择。当用户选择一个语言后，setAppLocale方法被调用，载入所需资源模块。这个方法首先通过资源管理器的getLocales方法输出结果检测所需资源是否已经被载入。这样做很有好处，如果资源没有被载入过，再调用loadResourceModule方法载入资源模块，并且监听complete事件以便可以知道资源模块是否已经准备好。

在complete事件的响应中，应用程序设置localeChain属性为最近使用的资源模块。接着调用

了contentBackground对象的三个方法。使用资源模块更新样式。

loadResourceModule方法还可接收一些可选的参数。如果程序需要载入多个资源模块，将会载入所有模块，但最后个update参数需设置为false。这样就不会多次重复调用资源管理器的update而节省开销。

## 26.5节. 支持IME设备

### 26.5.1. 问题

我想分发使用日文，中文或韩文等多字节字符的应用程序

### 26.5.2. 解决办法

使用Capabilities类检测输入法编辑器，使用IME类控制如何与Flex应用程序交互。

### 26.5.3. 讨论

东方的语言如汉字就是以象形字组成而不是用拉丁字符组成。拉到语言的字符是有限的，可被轻松的映射到键盘上。但这方法对于东方语言就不可能，因为这需要成千上万个键盘按键。输入法编辑器(IMEs)是软件工具，允许字符由多个按键组合而成。IME是运行在操作系统级别，在Flash Player外部。

Capabilities类有个属性叫hasIME，你可以使用它检测用户系统是否安装有IME。使用flash.system.IME对象检测是否已启动IME以及设置了什么转换模式。下面的例子测试IME，如果找到，启动IME并设置转换模式：

Code View:

```
private function detectIME():void
{
    if (Capabilities.hasIME == true)
    {
        output.text = "Your system has an IME installed.\n";
        if (flash.system.IME.enabled == true)
        {
            output.text += "Your IME is enabled. and set to " +
                flash.system.IME.conversionMode;
        }
        else
        {
            output.text += "Your IME is disabled\n";
            try
            {
                flash.system.IME.enabled = true;
                flash.system.IME.conversionMode =
            }
        }
    }
}
```

```

        IMEConversionMode.JAPANESE_HIRAGANA;
        output.text += "Your IME has been enabled
                        successfully";
    }
    catch (e:Error)
    {
        output.text += "Your IME could not be enabled.\n"
    }
}
else
{
    output.text = "You do not have an IME installed.\n";
}

```

当尝试创作IME设置时必须使用try...catch代码块，因为如果IME不支持指定设置就会发生错误。

有时候你可能希望禁用IME，比如某个文本框只输入数字。这时你可以在组件获得焦点时触发一个函数禁用IME，当组件失去焦点时重新启动IME：

Code View:

```

<mx:Script>
<[ [
    private function enableIME(enable:Boolean):void
    {
        if (Capabilities.hasIME)
        {
            try
            {
                flash.system.IME.enabled = enable;
                trace("IME " + (enable ? "enable" : "disable"));
            }
            catch (e:Error)
            {
                Alert.show("Could not " + (enable ? "enable" :
                    "disable") + " IME");
            }
        }
    }
]]>
</mx:Script>
<mx:VBox horizontalCenter="0" verticalCenter="0" >
    <mx:TextInput id="numericInput" focusIn="enableIME(false)"
        focusOut="enableIME(true)" />
    <mx:TextInput id="textInput" />

```

```
</mx:VBox>
```

如果你想知道用户是否组合一个字符，你可以监听System.ime对象事件：

```
System.ime.addEventListener(IMEEEvent.IME_COMPOSITION, onComposition);
```

## 26.6节. 检测屏幕阅读器

### 26.6.1. 问题

我想自定义应用程序以适应有视力障碍的用户

### 26.6.2. 解决办法

使用Accessibility类的静态属性active检测屏幕阅读器

### 26.6.3. 讨论

富互联网应用程序带给用户体验的标志之一就是丰富的媒体功能和。但不幸的是视力障碍人士使用Flex应用程序的这些功能。屏幕阅读器的支持对于视力障碍人士来说是非常重要的，因为这是他们唯一的方法能与你的程序进行交互。如果要适应视力障碍人士的需求，你可能希望专门为屏幕阅读器用户更改用户体验。Accessibility 类的active属性可被用来测试用户是否是使用屏幕阅读器。下面来自ScreenReader.mxml的代码块使用Accessibility.active确定是否要播放一个动画：

```
private function showNextPage():void
{
    if (Accessibility.active == false)
    {
        page2.visible = true;
        pageChangeAnimation.play();
    }
    else
    {
        page1.visible = false;
        page2.alpha = 1;
    }
}
```

## 26.7节. 创建标签以设置访问顺序

### 26.7.1. 问题

我想支持那些使用指针设备有困难的用户(指针设备就是鼠标之类)

### 26.7.2. 解决办法

为组件定义一个标签顺序以便用户可以不使用指针设备而能导航应用程序

### 26.7.3. 讨论

标签顺序是应用程序非常重要的易用性指标。它能使用户无需频繁切换键盘和指针设备就能轻松导航整个应用程序。对于使用指针设备难得用户，标签顺序是很必要的。你可以设置每个对象的tabIndex属性以指定组件的标签顺序。下面的例子TabOrder.mxml, 设置了标签顺序，这样用户就能不用鼠标也能轻松导航：

Code View:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute" creationComplete="firstName.setFocus()">
    <mx:Canvas width="228" height="215" x="50" y="50"
        backgroundColor="#FFFFFF">
        <mx:Label x="10" y="10" text="First Name" tabIndex="1" />
        <mx:TextInput x="10" y="36" width="100" id="firstName"
            tabIndex="2"/>
        <mx:Label x="118" y="10" text="Last Name" tabIndex="3" />
        <mx:TextInput x="118" y="36" width="100" id="lastName"
            tabIndex="4"/>
        <mx:Label x="10" y="69" text="Address" tabIndex="5" />
        <mx:TextInput x="10" y="95" width="208" id="address"
            tabIndex="6"/>
        <mx:Label x="10" y="125" text="City" tabIndex="7"/>
        <mx:TextInput x="10" y="151" width="100" id="city"
            tabIndex="8"/>
        <mx:Label x="118" y="125" text="State" tabIndex="9"/>
        <mx:TextInput x="118" y="151" width="34" id="state"
            tabIndex="10"/>
        <mx:Label x="160" y="125" text="Zip" tabIndex="11"/>
        <mx:TextInput x="160" y="151" width="58" id="zip"
            tabIndex="12"/>
        <mx:Button x="153" y="181" label="Submit" id="submit"
            tabIndex="13"/>
    </mx:Canvas>
</mx:Application>
```

## 26.8节。打印选择项

### 26.8.1. 问题

我想打印应用程序。

### 26.8.2. 解决办法

使用mx.printing包中的相关类，格式化和生成打印输出。

### 26.8.3. 讨论

mx.printing包有多个实现类用户生成打印输出。例如FlexPrintJob类定义一个打印任务，添加选项到任务中，发送任务给打印机。下面的BasicPrintJob.mxml例子创建一个打印任务，添加两页输出内容，发送此任务给打印机：

Code View:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="300">
    <mx:Script>
        <![CDATA[
            import mx.printing.FlexPrintJob;

            public function print():void
            {
                var printJob:FlexPrintJob = new FlexPrintJob();
                if (printJob.start())
                {
                    printJob.addObject(pageContainer1);
                    printJob.addObject(pageContainer2);
                    printJob.send();
                }
            }
        ]]>
    </mx:Script>
    <mx:VBox width="380" height="260" verticalCenter="-20"
        horizontalCenter="0">
        <mx:VBox id="pageContainer1">
            <mx:Label text="Page 1" />
            <mx:TextArea id="page1" width="100%" height="100%" />
        </mx:VBox>
        <mx:VBox id="pageContainer2">
            <mx:Label text="page 2" />
            <mx:TextArea id="page2" width="100%" height="100%" />
        </mx:VBox>
    </mx:VBox>
```

```

</mx:VBox>
<mx:Button bottom="5" right="10" label="Print"
    click="print();" />

```

当start方法被调用时，操作系统会显示一个打印对话框。执行会被暂停直到用户完成打印任务的配置。如果用户决定取消这次打印任务，start方法将返回false。否则此函数会调用addObject方法添加文本框到打印任务中，并调用send方法发送任务到打印机。

每次调用addObject，被添加项及其子对象都会被放置在新的一页里。如上面的打印例子，pageContainer1和pageContainer2被打印在不同的页面里。

addObject方法还接受可选的参数告诉打印机任务如何缩放添加的打印项。如果打印项太大，打印机任务将会渲染到多个页面。默认下打印项会根据页面宽度进行缩放。但是其他选项参数也是可用的，这些定义好的静态常量都在FlexPrintJobScaleType类中。你可以，比如缩放图表以匹配单个页面的高度：

Code View:

```

Public function print():void
{
    if (printJob.start())
    {
        printJob.addObject(columnChart,
                           FlexPrintJobScaleType.MATCH_HEIGHT);
        printJob.send();
    }
}

```

如果这个图表太宽，超出的部分会被打印到新页面上，作为一个例子，ScaleExample.mxml演示了各种缩放类型的效果。

## 26.9节. 格式化打印内容

### 26.9.1. 问题

我想按指定的格式打印内容

### 26.9.2. 解决办法

构建自定义打印渲染组件来格式化打印内容

### 26.9.3. 讨论

通常，你想要打印输出的内容不同于在应用程序中显示出的那样。你可能希望创建一个不

通过程序展现给用户的可打印应用程序对象版本或生成数据报告。打印渲染器就是为此目的而设计的，它是一个输出指定打印内容的组件。

在26.8节的BasicPrintJob.mxml例子中，你可能不想打印page标签或文本框控件的边框，也就是你可能只想打印文本框中的输入内容，就像文字处理器生成的那样，填充页面而不缩放文字。要打印出文本块，使用下面的BasicTextRenderer.mxml组件：

Code View:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml"  
    backgroundColor="#000000">  
    <mx:String id="textToPrint" />  
    <mx:Text width="100%" text="{textToPrint}" />  
</mx:Canvas>
```

当你使用打印渲染器来格式化输出时，首先要添加渲染器到显示列表以便Flex对组件进行可视化方面的布局：

```
public function print():void  
{  
    var printJob:FlexPrintJob = new FlexPrintJob();  
    if (printJob.start())  
    {  
        var printRenderer:BasicTextRenderer =  
            new BasicTextRenderer();  
        printRenderer.width = printJob.pageWidth;  
        printRenderer.textToPrint = page1.text;  
        printRenderer.visible = false;  
        Application.application.addChild(printRenderer);  
        printJob.addObject(printRenderer);  
        printJob.send();  
        Application.application.removeChild(printRenderer);  
    }  
}
```

我们注意到这个例子使用了打印任务对象的pageWidth属性。pageWidth和pageHeight 属性是在start方法返回时被设置的。当编写打印渲染器组件，当组件大小变化时需要注意这些属性值。通过这些属性，你可以确定即便更换打印机或纸张大小的情况下你的渲染器仍能正常工作在纵向和横向模式。

## 26.10节. 控制打印未知长度的多页内容

### 26.10.1. 问题

我想控制超过多页打印内容的布局，但是我不知道到底有多少数据将被打印以及组件的尺寸。

### 26.10.2. 解决办法

如果你要打印表格式数据，需要使用PrintDataGrid组件控制多页内容的打印。PrintDataGrid组件可控制变化的重复的多页打印内容。

### 26.10.3. 讨论

如果你有表格式的数据，比如数据报表，就可以使用PrintDataGrid组件格式化数据打印多页内容了。PrintDataGrid组件是一个特定的数据表格，设计用于打印多页内容的数据，下面的例子MultipageDataGrid.mxml，利用PrintDataGrid打印报表数据：

Code View:

```
public function print():void
{
    var printJob:FlexPrintJob = new FlexPrintJob();
    if (printJob.start())
    {
        var printGrid:PrintDataGrid = new PrintDataGrid();
        printGrid.width = printJob.pageWidth;
        printGrid.height = printJob.pageHeight;
        printGrid.columns = populationGrid.columns;
        printGrid.dataProvider = populationData.state;
        printGrid.visible = false;
        Application.application.addChild(printGrid);
        printJob.addObject(printGrid);
        while (printGrid.validNextPage)
        {
            printGrid.nextPage();
            printJob.addObject(printGrid);
        }
        printJob.send();
        Application.application.removeChild(printGrid);
    }
}
```

当使用PrintDataGrid时，你需要设置大小以匹配页面尺寸。添加表格到打印任务将添加第一页，使用validNextPage属性测试是否还有额外页面的数据，通过nextPage方法准备好下一页打印数据。

灵活使用PrintDataGrid组件可以帮你格式化各种类型的打印数据。PrintDataGrid并没有限制只用于打印表格式文本，PrintDataGrid可与项渲染器组合生成如图表，图像或复杂的组件。下面的例子GridSquares.mxml，PrintDataGrid与项渲染器组合生成相同的红色方框集合- ManualMultiPage.mxml：

Code View:

```
public function print(itemSize:int, itemCount:int):void
{
    var printData:Array = new Array();
    for (var i:int = 0; i < itemCount; i++)
    {
        printData.push(itemSize);
    }

    var column:DataGridColumn = new DataGridColumn();
    column.headerText = "";
    column.itemRenderer = new ClassFactory(SquareRenderer);

    var printGrid:PrintDataGrid = new PrintDataGrid();
    printGrid.showHeaders = false;
    printGrid.visible = false;
    printGrid.setStyle("horizontalGridLines", false);
    printGrid.setStyle("verticalGridLines", false);
    printGrid.setStyle("borderStyle", "none");
    printGrid.columns = [column];
    printGrid.dataProvider = printData;
    Application.application.addChild(printGrid);

    var printJob:FlexPrintJob = new FlexPrintJob();
    if (printJob.start())
    {
        printGrid.width = printJob.pageWidth;
        printGrid.height = printJob.pageHeight;
        printJob.addObject(printGrid);
        while (printGrid.validNextPage)
        {
            printGrid.nextPage();
            printJob.addObject(printGrid);
        }
        printJob.send();
    }

    Application.application.removeChild(printGrid);
}
```

## 26.11节. 打印页眉和页脚

### 26.11.1. 问题

我想打印出页眉和页脚

### 26.11.2. 解决办法

创建打印渲染器组件控制页面布局

### 26.11.3. 讨论

结合打印渲染器的PrintDataGrid比PrintDataGrid自身具备更多的布局控制能力。常见的任务就是打印页眉和页脚。这个技术涉及是否在布局中包含页眉和页脚以及PrintDataGrid的validNextPage属性的测试结果。下面的代码，HeaderFooterPrintRenderer.mxml，定义了一个打印渲染器生成多页打印内容，包括在适当位置的页眉和页脚：

Code View:

```
<?xml version="1.0"?>
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml"
    backgroundColor="#ffffff" horizontalAlign="center">
    <mx:Script>
        <! [CDATA [
            public function startJob():void
            {
                //Try to print this on a single page
                header.visible = true;
                header.includeInLayout = true;
                footer.visible = true;
                footer.includeInLayout = true;

                this.validateNow();

                if (printGrid.validNextPage)
                {
                    //The grid is too big to fit on a single page
                    footer.visible = false;
                    footer.includeInLayout = false;
                    this.validateNow();
                }
            }

            public function nextPage():Boolean
            {
                header.visible = false;
```

```

        header.includeInLayout = false;

        printGrid.nextPage();

        footer.visible = !printGrid.validNextPage;
        footer.includeInLayout = !printGrid.validNextPage;

this.validateNow();

        return printGrid.validNextPage;
    }
] ]>
</mx:Script>
<mx:DateFormatter id="formatter" formatString="M/D/YYYY" />
<mx:Canvas id="header" height="80" width="100%">
    <mx:Label text="Population by State"
        fontSize="24"
        color="0x666666"
        horizontalCenter="0"
        verticalCenter="0"
        width="100%"
        textAlign="center" />
</mx:Canvas>
<mx:VBox height="100%" width="80%">
    <mx:PrintDataGrid id="printGrid" width="100%" height="100%">
        <mx:columns>
            <mx:DataGridColumn dataField="@name"
                headerText="State" />
            <mx:DataGridColumn dataField="@population"
                headerText="Population"/>
        </mx:columns>
    </mx:PrintDataGrid>
</mx:VBox>
<mx:DateFormatter id="format" formatString="m/d/yyyy" />
<mx:Canvas id="footer" height="80" width="100%">
    <mx:Label text="{ formatter.format(new Date()) }"
        left="20" bottom="5" />
</mx:Canvas>
</mx:VBox>
```

这个组件定义了一个包含标题的页眉和显示日期的页脚。startJob方法为第一页初始化打印布局。它首先尝试把整个数据都容纳在单个页面里，然后调用validateNow强制更新组件布局，你可以通过PrintDataGrid的validNextPage属性测试其结果。如果返回值为false，则正好容纳，否则布局将被调整以隐藏页脚，重新更新布局。到此为止，无聊内容是多页还是单页，

第一页都将被添加到打印机任务中。如果需要多页，nextPage方法将准备相应的布局。它将隐藏页眉（因此这已经不是第一页，不需要页眉了）以及根据情况打印页脚。

把构建智能页面布局放到渲染器里将大大简化实际打印任务。下面的代码块HeaderFooter.mxml，演示如何使用打印渲染器：

Code View:

```
public function print():void
{
    var printJob:FlexPrintJob = new FlexPrintJob();
    if (printJob.start())
    {
        var printRenderer:HeaderFooterPrintRenderer =
            new HeaderFooterPrintRenderer();
        printRenderer.visible = false;
        this.addChild(printRenderer);
        printRenderer.width = printJob.pageWidth;
        printRenderer.height = printJob.pageHeight;
        printRenderer.dataProvider = populationData.state;
        printRenderer.startJob();

        do
        {
            printJob.addObject(printRenderer);
        }
        while (printRenderer.nextPage());

        //Send the last page
        printJob.addObject(printRenderer);
        printJob.send();

        this.removeChild(printRenderer);
    }
}
```

Print方法开始一个打印任务并建立一个打印渲染器。这部分代码完成渲染器startJob方法的调用，准备好第一页的打印。下一部分代码，do...while块继续添加页面到打印任务，直到nextPage方法返回false，指示已没有页面可打印。但是do...while块是在最后调用nextPage方法，因此当循环结束最后一页还没有加入到打印任务中，需额外加入，发送打印任务，移除显示列表中的渲染器。