Flex sdk 4.0

Flash builder [4]

中文快速入门

从零基础，到案例实践的过程。

<mx:Label text="go to airia.cn just Flex it !"/>

OPEN

声明

根据《中华人民共和国著作权法》及《最高人民法院关于审理涉及计算机网络著作权纠纷案件适用法律若干问题的解释》的规定，特此声明：

未经AIRIA书面授权的情况下禁止转载、出版、印刷本电子书或电子书中的文章，以及禁止利用该文档从事各种商业行为。而破解和篡改本PDF文件也构成侵权行为。

完整未经修改的PDF文件是被允许在互联网转载、下载的，但请注明文档为AIRIA出品。

资源连接

版本修正：

2009/07/01 第一次 修正1处排版错误（p64）1处敏感字符(P48) 1处注释（P116）

第一章 ： Flash Builder 4 背 景

郭峰

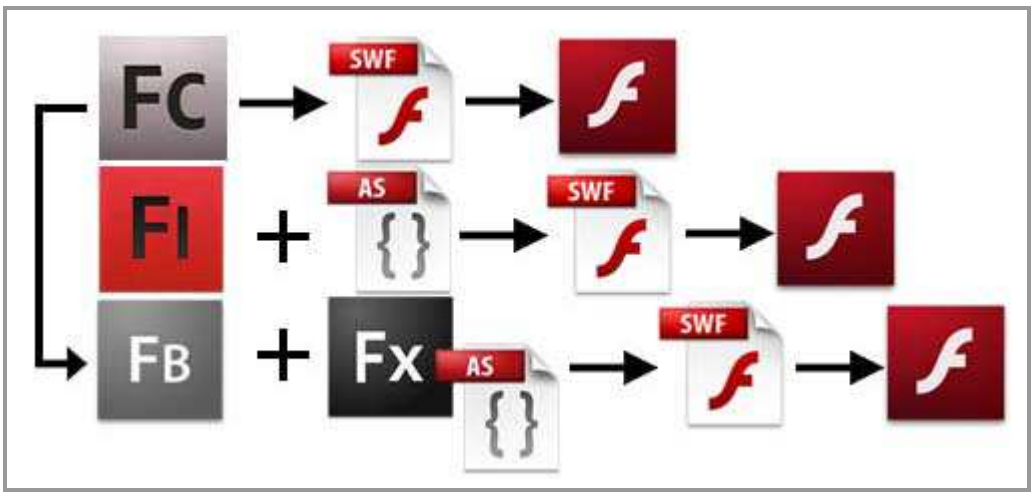
1.1 从Flex Builder到Flash Builder

2009年5月16日，Adobe 公司宣布“下一版本的 Flex Builder 将被命名为 Flash Builder”。官方的解释是，“这样可以使 Flash 家族工具的命名具有更好的一致性，并藉此将 Flash Builder 定位为开发工具”。

第一节　从Flex Builder到Flash Builder

2009年5月16日，Adobe 公司宣布“下一版本的 Flex Builder 将被命名为 Flash Builder”。官方的解释是，“这样可以使 Flash 家族工具的命名具有更好的一致性，并藉此将 Flash Builder 定位为开发工具”。

我们在 Flex Builder 中构建应用程序的时候，实际上需要用到开源的 Flex SDK 及 ActionScript，应用程序最终会被编译为运行在 Flash Player 中的 SWF 文件。但是，不管是 Flex SDK 还是 Flex Builder，看起来都与 Flash Player 没有什么关系。用户有时甚至会问到是否存在“Flex Player”这类软件。为了让用户能将 Flex 与 Flash 联系在一起，将 Flex Builder 命名为 Flash Builder 显然是一个不错的主意。下图展示了 Flash 家族主要工具与技术之间的关系。（图1）



图一

Flash 家族主要工具与技术之间的关系

（1）用 Flash Catalyst 生成 SWF 文件，并运行在 Flash Player 中。

（2）用 Flash Catalyst 创建的项目可以导入 Flash Builder 中。

（3）用 Flash CS4 IDE 和 AS3 生成 SWF 文件，并运行在 Flash Player 中。

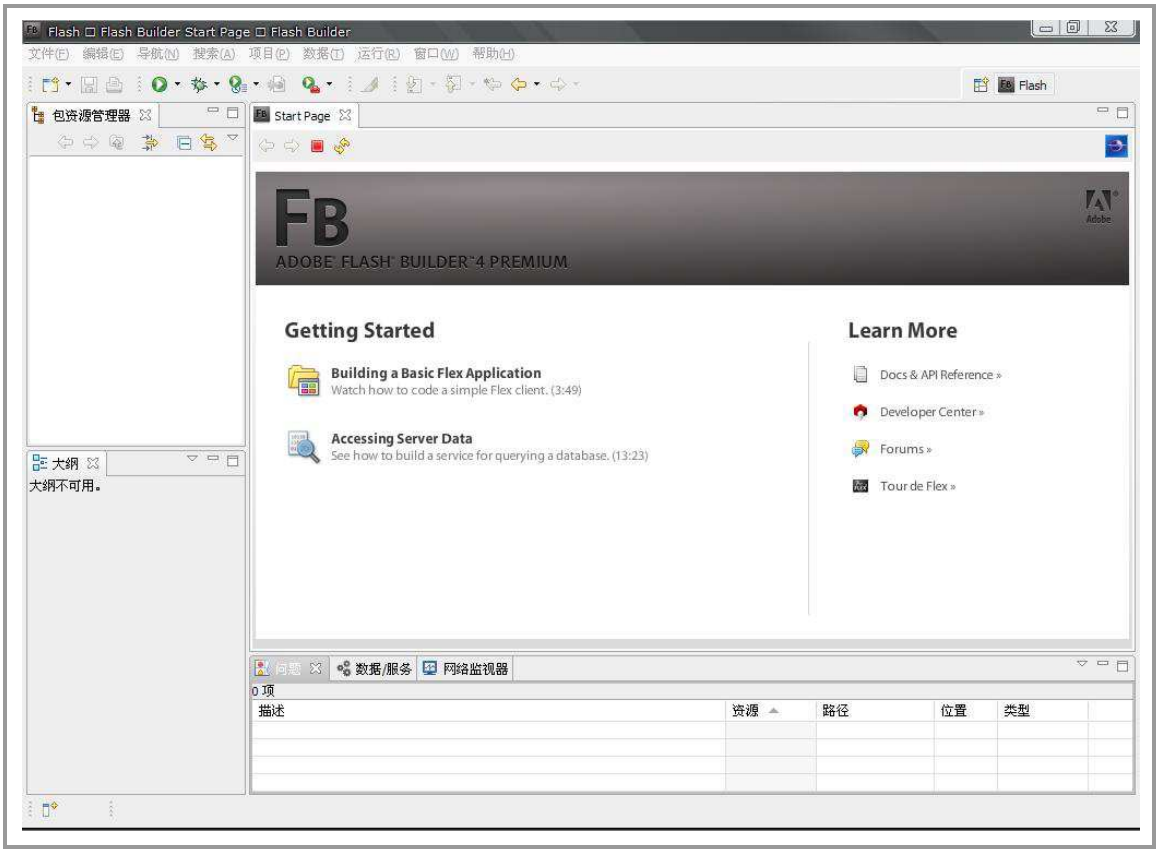
（4）用 Flash Builder 和 Flex SDK 及 AS3 生成 SWF 文件，并运行在 Flash Player 中。

第二章 ： Flash Builder 4 界面

唐凡

第一节　主界面

首先我们来看一看 Flash Builder 4 的启动画面。可以看到，在经历了更名以后，出现的不再是以前的“Fx”字样，而是“FB”。启动好了以后，可以看到主界面的整体 UI：（图二）



图二

Flash Builder 主界面预览

如果你使用过上一个版本，那么应该会发现，主界面的布局几乎没有改变。下一节我们来具体认识 Flash Builder 4主菜单。

第二节　主菜单

在 Flash Builder 中，我们并不经常使用到主菜单功能，大部分功能可以在工具栏找到，这节主要来介绍常用的主菜单。

菜单：文件 → 新建

在新建菜单中，我们可以新建Flex项目工程、包、文件夹和各种文件类型等 (图三)

图三

Flash Builder 新建菜单

菜单：导入 → 导出

选择“导入”可以将 Flash Builder 项目、主题和皮肤等工程添加到 Flash Builder 4 中进行编辑。（图四）

选择“导出”可以发布您的项目。

图四

Flash Builder 新建菜单

重构项目

当项目中删除了某些文件然后重新编译以后，在 bin-debug 下可能并不会动态删除对应文件，这会使项目不必要的增大。

所以我们需要对项目进行重构。

菜单：项目→ 清理（图五）

图五

Flash Builder 项目重构

在弹出的“清理”窗口中，勾选要重构的项目，再选择“清理下面所选项目”，然后确定。（图六）

这样你的项目会被重新构造，无用的资源会被去掉，保证了项目质量。

图六

Flash Builder 项目重构

数据

与上一个版本相比，Flash Builder 4 在“数据”菜单中加入了更多的连接方式来方便使用。（图七）

用户可以根据自己的实际需求来选择使用。具体方式请参考第六章 ：与服务端通信。

图七

数据

运行

在 Flash Builder 4 中，添加了 FlexUnit 测试，这是值得称赞的。通过“运行”菜单，可以对项目进行测试、运行以及追踪

调试

这些功能在实际开发中的使用是相当频繁的。（图八）

图八

运行菜单

以上是使用相对比较频繁的菜单，其他菜单功能实际使用频率较小，下一节主要介绍工具栏。

第三节　工具条

这是 Flash Builder 4 的常用工具栏，是我们比较常用的功能。常用工具栏比较简单，下图对工具条作用做出了简单注释。

图九

Flash Builder 工具条

第四节　主要窗口

本章最后一节，将向各位简单介绍 Flash Builder 4 的各个主要窗体。

包资源管理器

以树状结构的视图，来显示 Flex 工程下的所有包、文件夹、各类文件等。（图十）

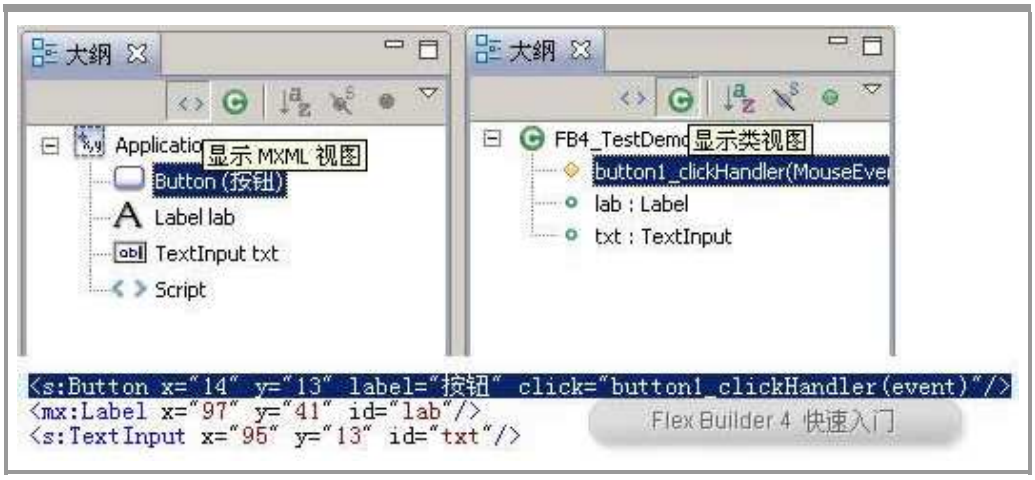
图十

包资源管理器

大纲窗口

大纲窗口分为“MXML 视图”和“类视图”，“MXML 视图”可以展现页面元素控件的总揽和包含关系，类视图可以展现页面 ActionScript 代码中的方法和属性的索引。在页面元素特别多的时候比较实用。

点击“MXML 视图”中的元素控件，可以快速定位到该元素位置。



问题窗口

当代码出现错误或者警告的时候，会把信息汇总在“问题”窗体显示。（图十一）

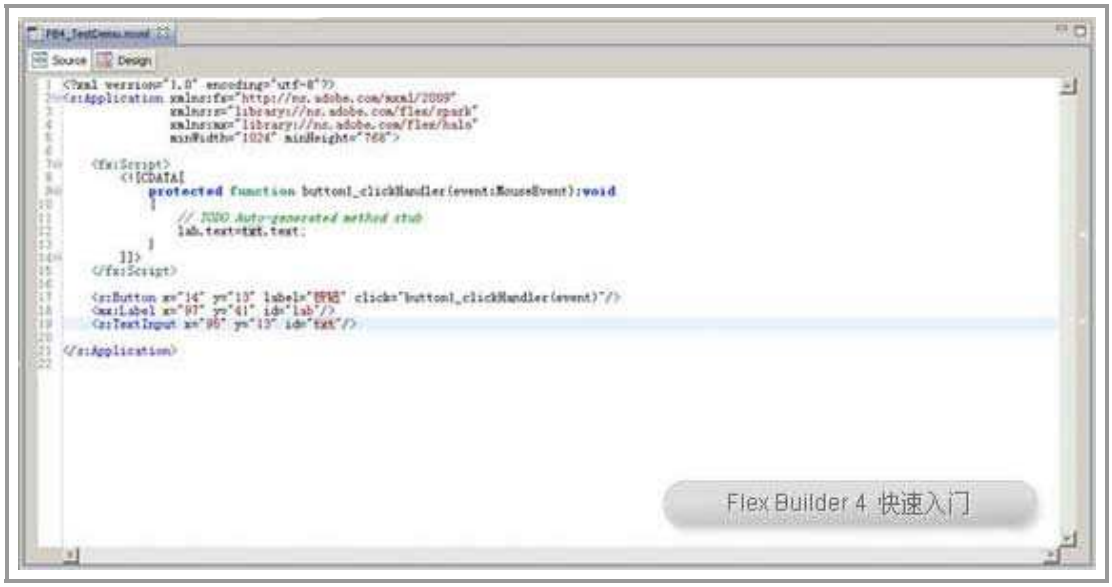
当页面存在错误，运行程序时系统会弹出提示，询问是否执行。（图十二）

图十一、图十二

问题窗口与错误提示

主窗体-代码视图

主窗体分为 Source（代码视图）和 Design（设计视图）。这是代码视图，我们使用最多的窗体，进行代码编写的地方。（图十三）

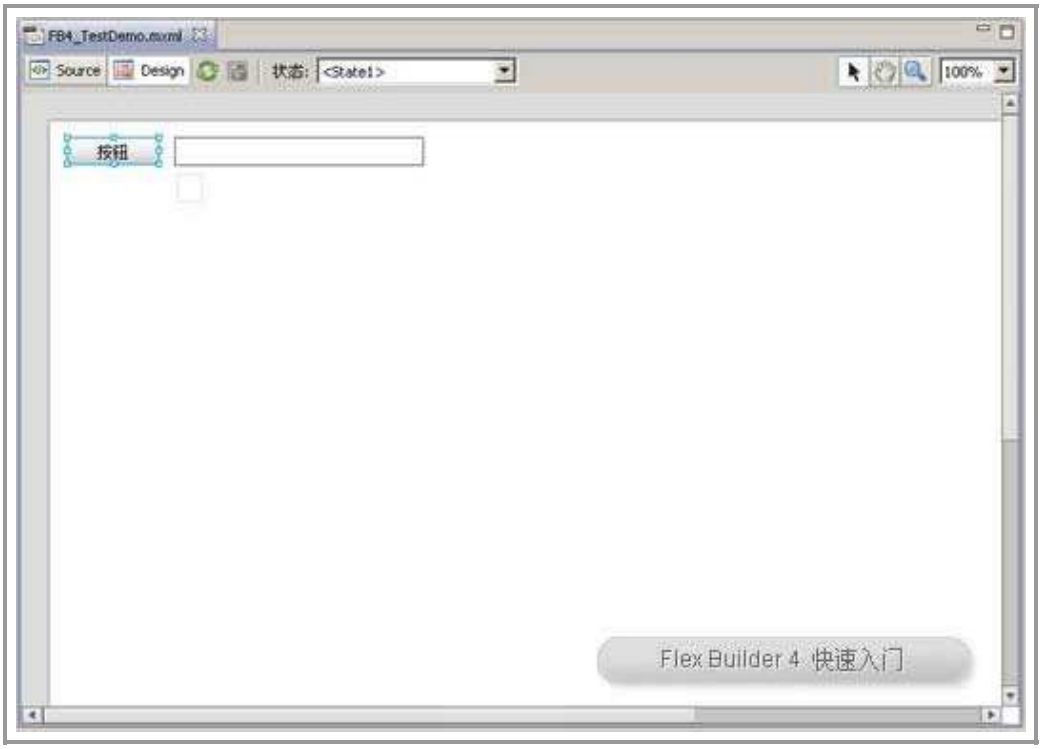


图十三

主窗体代码视图

主窗体-设计视图

这个是 Design 视图，也就是设计视图，我们可以在设计视图里对控件元素进行“所见即所得”的操作， 或者将其他控件元素直接拖动进来，而代码视图动态生成代码。（图十四）



图十四

主窗体设计视图

属性与外观窗口

在设计视图下，窗体右边会出现“属性”和“外观”窗体。

“属性”窗体分为“标准视图”和“类别视图”。

“标准视图”可以对当前所选择的控件元素的“公共”、“样式”和“布局”等进行编辑。（图十五）

图十五

属性与外观窗口

图十六

属性类别视图

“类别视图”可以分类显示各种属性，包括“公共”，“事件”，“大小”，“布局约束”，“效果”，“样式”，“其他”。（图十六）

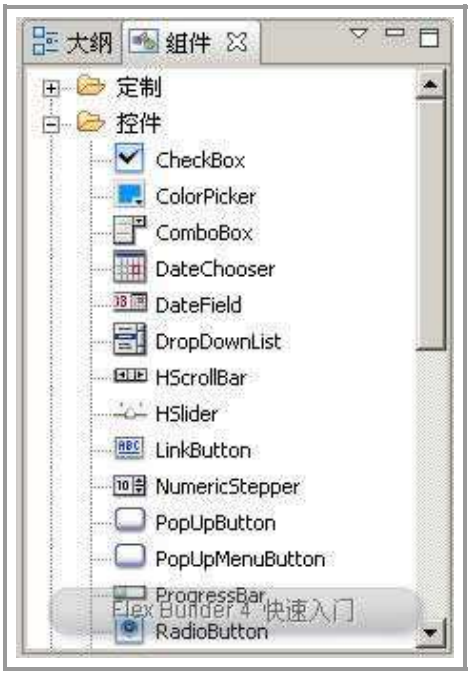
“外观”窗体可以对整个项目的外观风格进行编辑，这是 Flash Builder 4 新加入的一个功能。（图十七）

图十七

外观窗体

组建窗体

“组件” 窗体也是我们使用比较频繁的，里面列出了几乎所有的常用组件，我们可以直接把需要的组建直接拖拽到设计视图。



图十八

组建窗体

状态窗体

“状态”是 Flex 里面比较有特色的一个东西，我们可以新建一个状态，在新状态下改变界面的外观，通过切换状态，可以实现切换风格的功能。这里只是抛砖引玉，可以结合自己的需求来巧妙的使用状态。

所有的状态都会罗列在“状态”窗体里面，可以直接单击选中来切换状态。（图十九）

图十九

状态窗体

第三章 ： Flash Builder 4 新特性

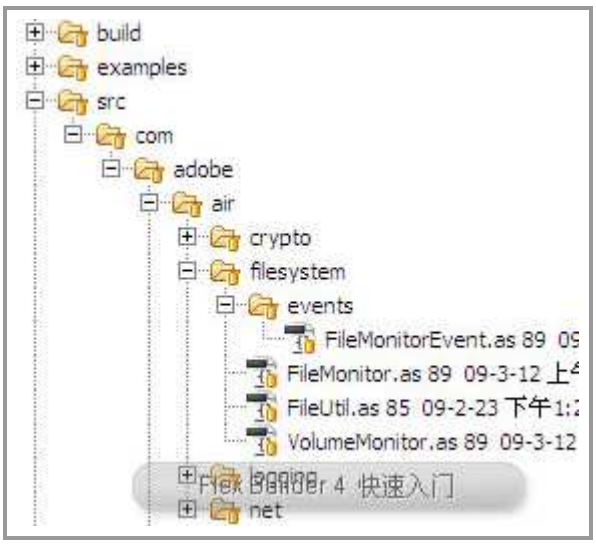
郑会宾	3.1 Package explorer
郑会宾	3.2 悬停时的ASDOC提示
唐凡	3.3 Getter & Setter
唐凡	3.4 自动生成EventHandler
郑会宾	3.5 条件断点

第一节 Package explorer

本小节介绍给大家的是 Flash Builder 4 中的一个非常细节的改进——包资源管理器（package explorer）。

在以往的Flex Builder 3 版本中Adobe并没秉承 Eclipse 中的一贯风格理所当然加入对包（package）的管理支持，以至于程序员在建立类包的过程中都要新建n多文件夹。程序员在开发过程中往往为了寻找一个类而要点开n层目录结构，十分不便。

下图是 Flex Builder 3 工程目录的截图，可以看到要打开多层目录。（图二十）



图二十

Flex Builder 3 的工程目录截图

在新的 Flash Builder 4 中要建立多层的包路径，我们只需在工程上右键->“新建” ->“包” ，便可添加包（package）。（图二十一）然后在弹出的对话框中输入包（package）的名字。(图 二十二)

图二十一

Flash Builder 添加包

图二十二 新建包命名

然后你便将看到在左侧的“包资源管理器”中的项目结构里添加了新的包（package）。（图二十三）

图二十三

第二节　悬停时的ASDOC提示

在多人协同开发时，如果我们往自己的代码中引用了别人开发的类包和方法，就会时不时切换到那些类的源码，去看同事们为类添加的注释或者 ASDoc以确定如何去使用这个方法，这样做十分不便。在这次Flash Builder 4的版本升级中，Adobe为我们新增了一项不错的功能——ASDoc 悬停提示，这项 新功能的主要作用就是使我们在编码过程中只用用鼠标悬停在你想要查看 ASDoc 的类，方法上，IDE 工具就会自动在一个视图中呈现这些 ASDoc 提示， 十分方便。

学习目标

了解 Flash Builder 4 的新特性——编码时的ASDoc悬停提示。

1、首先要打开 ASDoc 提示窗口，步骤如下： 工具栏->窗口->其他视图。（图二十四）

图二十四

打开ASDoc提示窗口

在弹出的窗口中的树形结构中选择 Flash Builder->ASDoc，点击“确定”。（图二十五）

图二十五

选择ASDos

然后，ASDoc 视图便会出现在 IDE 编辑器中。(图二十六)

图二十六

ASDos编辑器

2、我们首先写一个 Service 类，并添加 ASDoc 注释。ASDoc 的具体写法在下一章中会有详细介绍。

```
package cn.airia.fb4.asdoc
{
   public class UserService
   {
      public function UserService()
      {
      }
      /**
            * 根据用户id返回用户名
            * @param id 用户的id号
            * @return 用户名
            *
            */
           public function getUsernameById(id:String):String
      {
         return "momoko";
      }
   }
}
```

code

FlexBuilder 4 快速入门 代码片段

3、然后我们在主程序中调用这个 getUsernameById 方法，当我们需要查看这个方法的 ASDoc 注释时，只需将鼠标悬停在方法名上，ASDoc 窗口便会自动显示它的注释。（详见下图）

本章小结

Flash Builder 4为我们提供了这么好的工具，希望大家在开发过程，尤其是多人团队开发中可以多加利用，当然最主要的还是要养成写 ASDoc 注释的好习惯。

第三节 Getter & Setter

本节介绍Flash Builder4的Getter & Setter。

在Flash Builder4中提供了Getter & Setter（封装属性字段）的功能， 接下来我们一起来看看如何使用。

首先新建一个AS类，再写几个私有属性。如下：（图二十七）

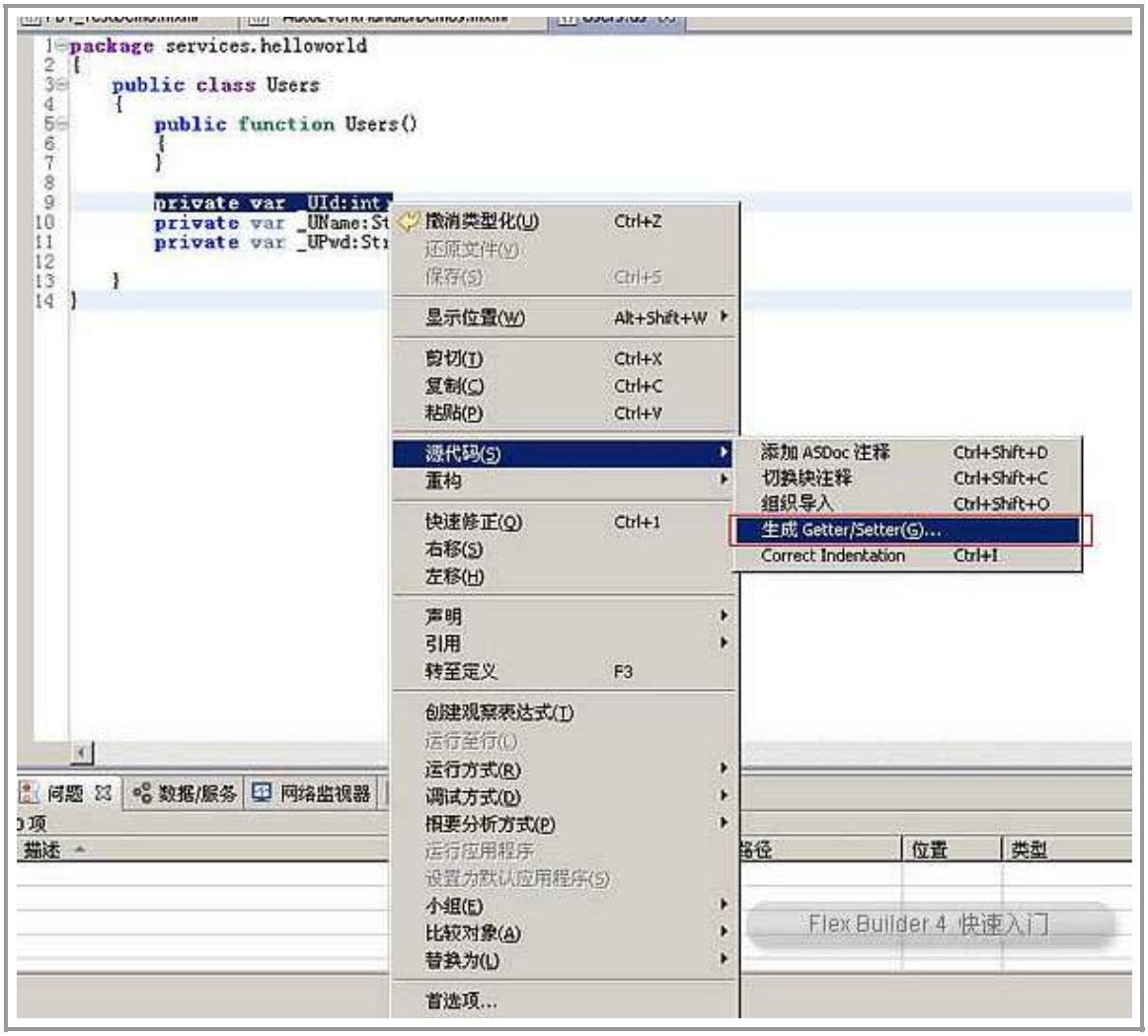
```
1  package services.helloworld
2  {
3      public class Users
4      {
5          public function Users()
6          {
7          }
8
9          private var _UId:int;
10         private var _UName:String;
11         private var _UPwd:String;
12
13     }
14 }
```


图二十七

私有属性

写了好属性以后，我们一起来给属性添加Getter & Setter把它们封装起来。

操作方法：高亮选择要封装的属性。右键菜单选择“源代码” --“生成Getter/Setter ” 。



图二十八

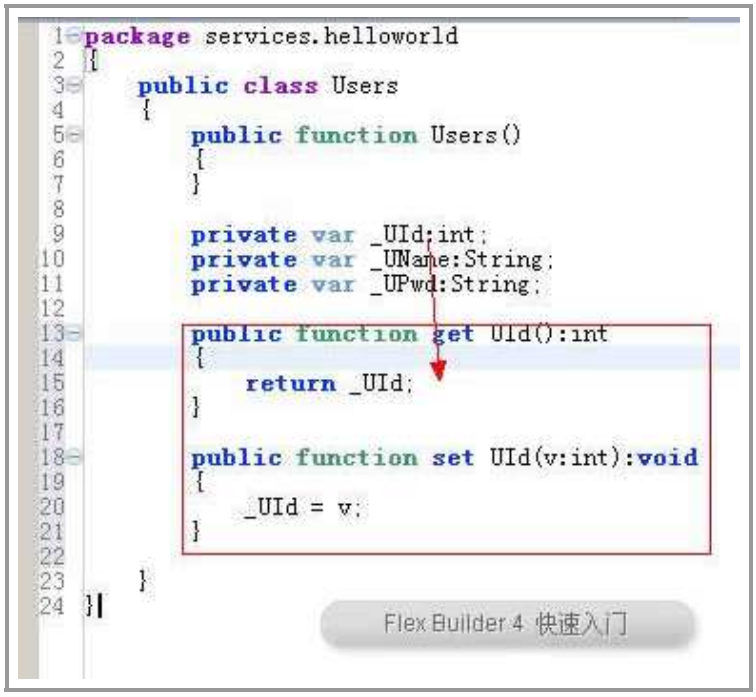
生成Getter/Setter

在弹出的对话框中，我们可以编辑Getter&Setter的一些信息， 可以单独生成Getter或者Setter，命名空间和位置等等。（图二十九）

图二十九

生成Getter/Setter

点击确定。我们就为这个属性生成好了Getter & Setter。（图三十）



图三十

生成Getter/Setter

该功能的缺陷则是每次只能为一个属性添加Getter & Setter， 不能批量生成，所以如有需要可以使用第三方插件或者CodeSmith之类的工具来生成实体类文件。

第四节　自动生成EventHandler

本节为大家介绍Flash Builder4的一个新功能，自动生成EventHandler。

在新版本中，Flash Builder4为我们提供了自动生成EventHandler功能，这个功能使用起来相当方便，

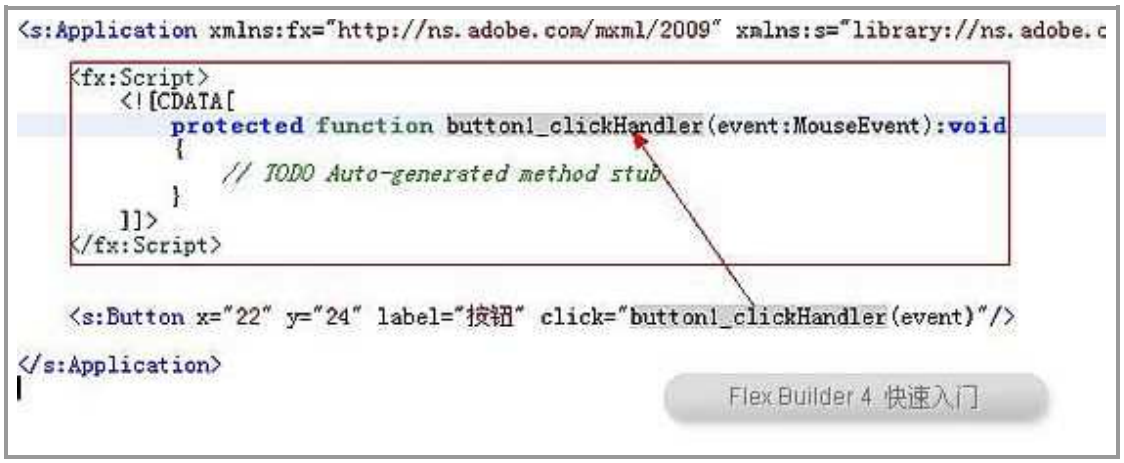
在一定程度上也提高了开发效率。

如下图(图三十一)所示，当我们为一个按钮添加“click”事件时，系统会自动提示生成EventHandler。

图三十一

提示生EventHandler

我们确认添加EventHandler，可以看到，自动生成了按钮的“button1_clickHandler”方法。（图三十二）



图三十二

生成按钮方法

这样系统就为我们自动生成好了按钮的“click”事件的执行方法。

小提示：

图例中并没有给按钮的“id”属性赋值，所以生成的方法名叫“button1_clickHandler”，这样并不符合命名规范，如果数量多了，可能会弄不清谁是谁。 如果按钮的“id='btnSubmit'”，生成的方法将是btnSubmit_clickHandler

所以推荐大家给按钮都命名，要符合命名规范，尽量给事件加上注释，说明其用途。

第五节　条件断点

本节要向大家隆重介绍的是个人认为最为在这次 IDE 工具的升级中（Flex Builder 3 ->Flash Builder 4）最为贴心的一处改进——条件断点。

准备工作

希望阅读此节的同学对程序的断点调试有一定的认识。 另外请确认系统中已经安装了 flash player debug 版本,你可以在下面的链接中根据自己的操作系统以及浏览器来选择适合自己的 player Flash Player 下载中心

使用过 Flex Builder 3 版本的同学在调试代码时可能都深有感触，debug 的断点设置在某一行语句，程序一旦执行到这一语句便会停止，但是并非每一次停止是要监控 都是我们所关心的，尤其是对于 for 循环的调试，于是我们便不断地按“process” 按钮（也就是那个绿色的小箭头）来“快进”我们的代码。这样做费时费力，有时候 不小心点快了又要重头来过。现在可好啦，在 Flash Builder 4版本中令人欣喜的增加了条件断点的功能，以前的烦恼一扫而去，循环再多也不怕。

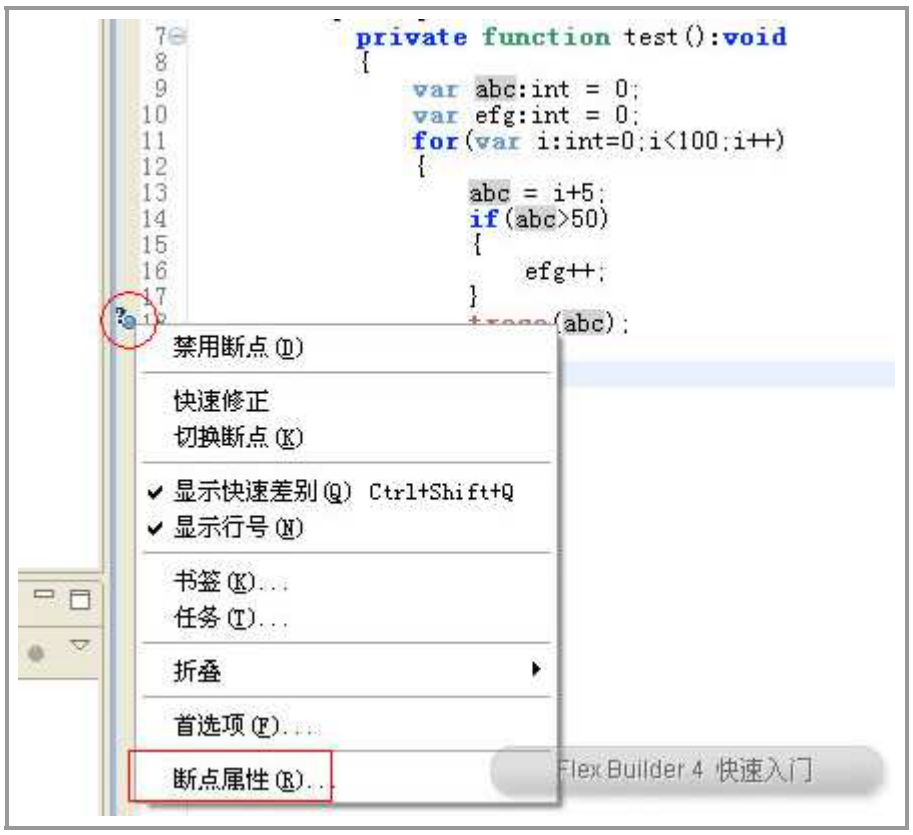
1、先来写一段 for 循环代码,接下去我们就利用这段代码来展示下条件断点的威力。

```
private function test():void
{
    var abc:int = 0;
    var efg:int = 0;
    for(var i:int=0;i<100;i++)
        {
        abc = i+5;
        if(abc>50)
            {
            efg++;
            }
        trace(abc);
        }
}
```

code

FlexBuilder 4 快速入门　代码片段

2、在 trace(abc); 这一行加入断点，然后在断点处点击右键，选择“断点属性”。（图三十三）



图三十三

选择断点属性

在弹出的对话框中我们可以看到条件断点的设置界面。 我们可以看到，断点大致分为3类：命中计数断点、条件为 true 断点、值改变断点。（图三十四）

图三十四

选择断点属性

3、命中计数断点 ，即断点所在行的代码被执行次数与设置值符合是，程序停止在断点所在位置。我们将计数设置为10，也就是说 trace(abc);被执行第十次的时候程序会暂停。（图三十五）

图三十五

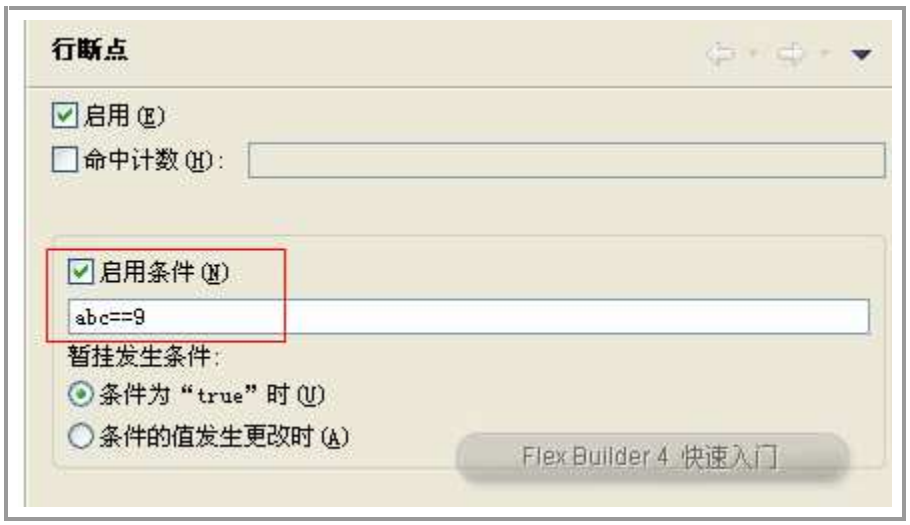
命种计数断点

3、调试程序 程序停止在断点的时候，看下此时的变量值，i 的值为9，说明 trace(abc);执行了10次。（图三十六）

图三十六

调试程序

4、条件为 true 断点，即当条件满足于设置的表达式，且运算返回值为 true 时，程序停止在断点所在位置。我们将 abc == 9 作为条件，“暂挂发生条件” 选择“条件为 true 时”。（图三十七）



图三十七

条件为true断点

调试程序，程序停止在断点的时候，看下此时的变量值，abc 的值正好为9。（图三十八）

图三十八

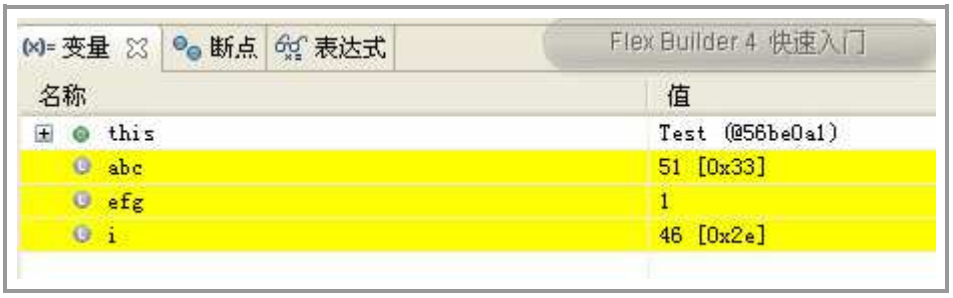
调试程序

5、值改变断点，即设置的变量一旦在程序运行过程中值发生改变，程序停止在断点所在位置。 我们将变量 efg 作为条件变量，“暂挂发生条件” 选择“条件的值更改时”。（图三十九）

图三十九

值改变断点

调试程序：程序停止在断点的时候，看下此时的变量值，efg 的值的确发生了改变，已由0变为为1。



图四十

调试程序

总结：

怎么样？十分方便吧，大大节省了调试程序时的时间精力。经过本人测试，条件断点还支持组合的设置 比如说“计数断点”分别和“条件为 true 断点”，“值改变断点”一起设置后，可以产生“与”的条件判断关系！这点使我们的调试断点设置更加灵活。

一个有经验的程序员，可以迅速的发现代码中的错误，Flash Builder 4的这次改进真可谓是令程序员能事半功倍啊！

第四章 ：Flex SDK 4 新特性

罗楷

Flex SDK 4 新特性 第一节 主题

我们来学习Flash builder 4的新特性 -- 主题

学习目标

1. 切换Flash Builder 4自带的多款主题。
2. 知道主题的存放位置。
3. 知道主题应用的原理。

实现步骤

在Flash Builder 4以前，Adobe默认的主题是Halo, 而从Flash Builder 4开始，默认的主题变成了Spark, Spark主题中使用了很多图片作为控件的皮肤，因此Spark只支持部分的色彩样式, 它们是baseColor, color, contentBackgroundColor, focusColor, symbolColor,selectionColor, and rollOverColor,如果同学们想改变一些控件的背景，比如Button的背景样式，我们需要重新定义背景的skin图片才可以做到。当然我们也可以使用原来的Halo主题。现在我们就来看看如何使用主题。

Flash Builder 4中包含了9款默认主题，其中两款Spark主题，七款Halo主题。这里有必要提一提它们的区别，在Flash Builder 4中，由于出现了新的library://ns.adobe.com/flex/spark (xmlns:s)名称空间,代表新的spark.*包中的控件，所以Flash Builder 4 中同时存在了原来的mx.*以及spark.*中的两组控件。因此在Flash Builder 4中css style也加入了名称空间的支持，比如

```
<fx:Style>
    @namespace s "library://ns.adobe.com/flex/spark";
    @namespace mx "library://ns.adobe.com/flex/halo";
    s|Button {
        color: #FF0000;
    }
    mx|DateChooser {
        color: #FF0000;
    }
 </fx:Style>
```

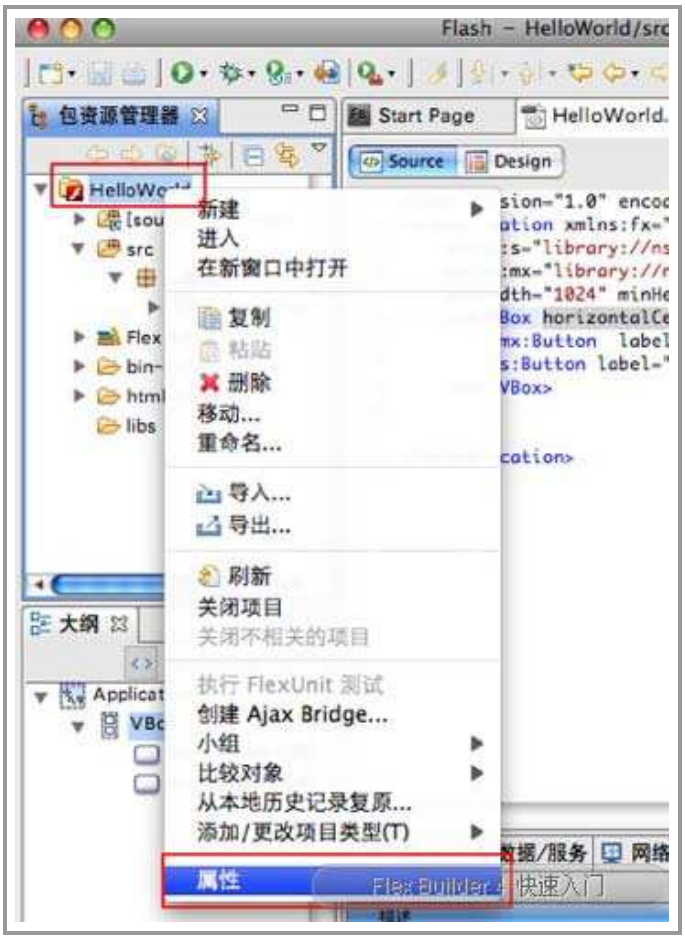
code

FlexBuilder 4 快速入门 代码片段

其中S|Button 代表了Spark包中Button的样式，mx|DateChooser则代表mx包中DateChooser的样式。

回到我们的主题选择，首先创建一个名为HelloWorld的Flex项目。

1. 右键点击项目，选择属性 （图四十一）



图四十一

创建项目

2. 选择Flex主题。

同学们可以看到右边显示了8款主题，其中第九款，也就是Halo.swc是没有显示在这里的，我们需要用导入主题的方式将其加入，我们在这里双击WireFrame主题。 （图四十二）

图四十二

FLEX主题

在HelloWorld.mxml中插入下面代码并编译运行 (代码中的布局方式也改变了，详情请查看布局章节):

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        xmlns:mx="library://ns.adobe.com/flex/halo"
        minWidth="1024" minHeight="768" preloader="mx.preloaders.DownloadProgressBar">
        <s:HGroup  gap="50" horizontalCenter="0" top="10">
        <s:Panel width="200" height="200" title="Spark Panel" >
        <s:layout>
            <s:HorizontalLayout paddingLeft="10" paddingTop="10" />
        </s:layout>
            <s:Button label="Spark 按钮"  />
        </s:Panel>
        </s:HGroup>

</s:Application>
```

code

FlexBuilder 4 快速入门 代码片段

左图是默认的spark样式，右图是切换wireframe样式后运行的结果 （图四十三）

图四十三

两种不同的FLEX主题预览

注: 如果同学们选择的是Halo系列的主题，那么这里运行后仍然看到的会是Spark主题，这是因为Halo主题只对mx.controls里的控件有效，稍后会为同学们演示。

3. 我们已经知道了如何切换主题，现在来看看主题的存放位置以及主题是如何被应用的

主题的默认存放位置在 Flash Builder 4安装目录/sdks/4.0.0/frameworks/themes/

打开后会看见9个文件夹，分别对应一个主题。我们在HelloWorld项目文件夹中打开配置文件.actionScriptProperties。

下图中红色框部分表示了主题的配置。 （图四十四）

```
<?xml version="1.0" encoding="UTF-8"?>
<actionScriptProperties mainApplicationPath="HelloWorld.mxml"
projectUUID="573a8c12-706d-4816-8c50-3dca480b54a0" v
ersion="6">
  <compiler additionalCompilerArguments="-locale en_US"
  autoRSLOrdering="true"
  copyDependentFiles="true"
  generateAccessible="false"
  htmlExpressInstall="true"
  htmlGenerate="true"
  htmlHistoryManagement="true"
  htmlPlayerVersionCheck="true"
  includeNetmonSwc="true"
  outputFolderPath="bin-debug"
  sourceFolderPath="src"
  strict="true"
  targetPlayerVersion="0.0.0"
  themeIsDefault="false"
  themeIsSDK="true"
  themeLocation="/Applications/Adobe Flash Builder Beta/sdks/4.0.0/frameworks/themes/Wireframe"
  themeMain="wireframe.swc"
  themeMaxSDK=""
  themeMinSDK="4.0.0"
  themeName="Wireframe"
  themePath="Wireframe"
  useApolloConfig="false"
  useDebugRSLSwfs="true"
  verifyDigests="true" warn="true">
    <themePath>Wireframe</themePath>
    <compilerSourcePath/>
    <libraryPath defaultLinkType="0">
      <libraryPathEntry kind="4" path=""/>
      <libraryPathEntry kind="1" linkType="1" path="libs"/>
    </libraryPath>
    <sourceAttachmentPath/>
  </compiler>
  <applications>
    <application path="HelloWorld.mxml"/>
  </applications>
  <modules/>
  <buildCSSFiles/>
</actionScriptProperties>
```


图四十四

FLEX主题配置

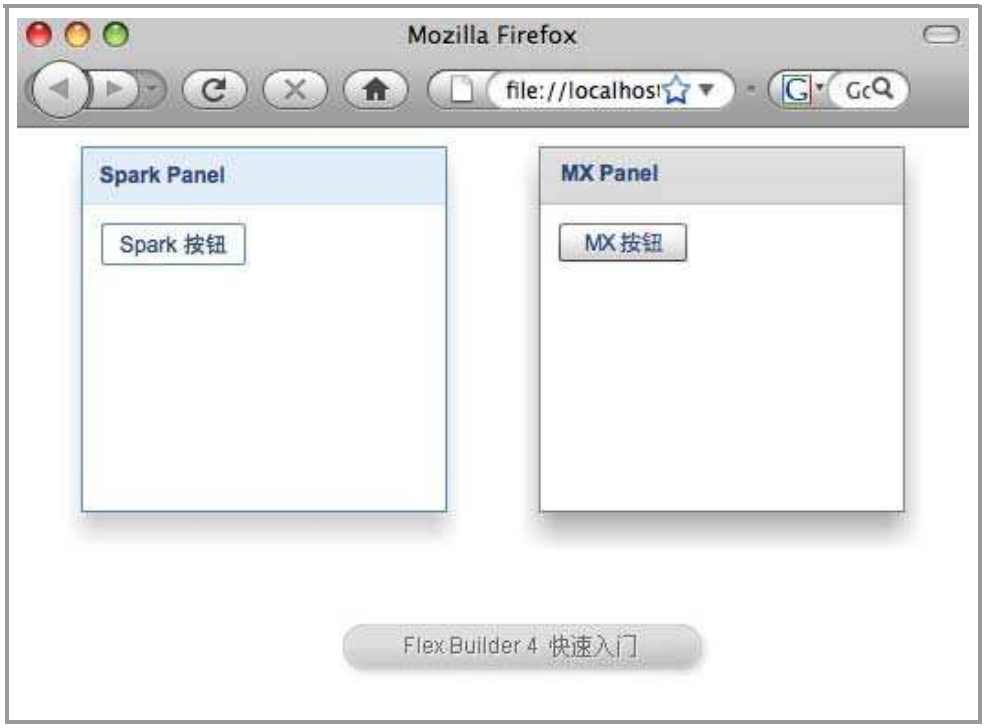
我们继续来看看主题包里都用什么，将wireframe.swc改名为wireframe.zip，解压。解压后可见下面这些文件。（图四十五）

图四十五

深入研究主题包

很明显default.css是主题的样式设置，我们可以对其修改以改变主题的配置。注意default.css中的名称空间 @namespace "library://ns.adobe.com/flex/spark"; 表示了他对spark系列的控件有效。

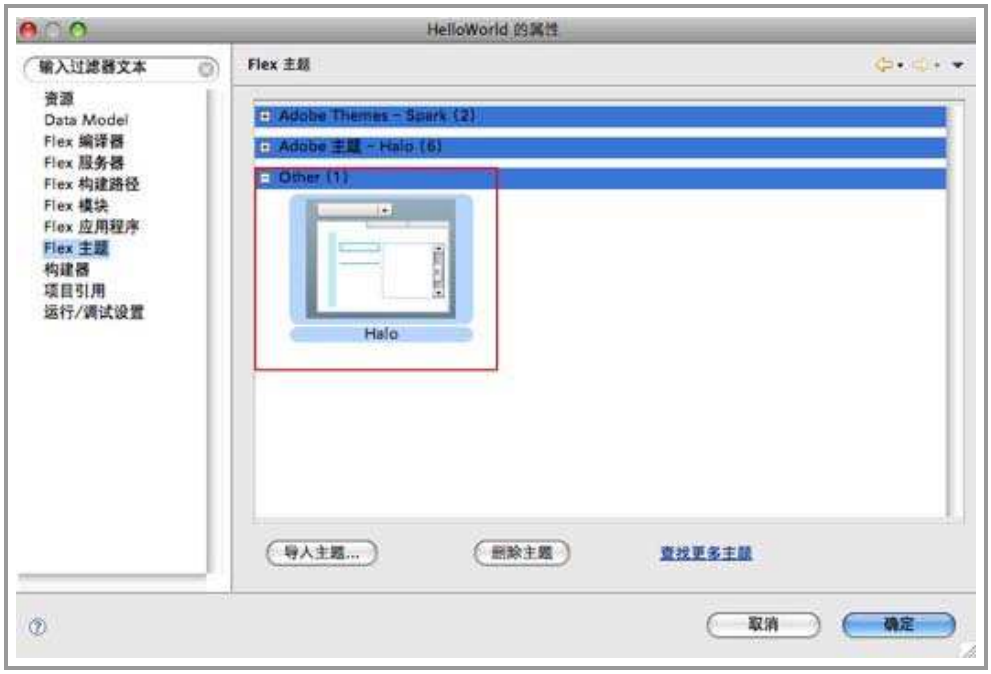
好了，以上就是Flash Builder 4中对主题的选择。现在我们来继续扩展一些内容。刚才我提到spark以及wireframe主题只对spark包中的控件有效，也就是对Flex Builder 3中的控件是没有效果的，我们来做个实验。将下面的代码放入HelloWorld.mxml，我在其中加入了mx:Panel以及mx: Button。现在同样选择wireframe主题，运行。我们可以看到下图的效果。（图四十六）



图四十六

主题扩展

右边的mx Panel以及Button显示了默认的Spark样式，说明了wireframe的设置对其无效。那么是否可以继续使用flex builder 3中的样式呢？当然可以，在themes文件夹中的Halo就代表了Flex 3中的默认样式，我们再次打开样式选择面板，点击“导入样式”，选择Halo文件夹中的halo.swc导入。之后我们在Other 一栏里会看见它，见下图（图四十七）



图四十七

导入样式

双击导入，再次运行刚才的代码（图四十八）

图四十八

导入样式、运行

同学们可以看到右侧已经显示了我们熟悉的Flex 3中的默认样式，而左侧的Spark控件仍然显示了Spark的默认样式。

总结：

本节中介绍了Flash Builder 4中主题的使用方法，现在的方式让主题的开发与使用更加的规范化，相信以后会出现越来越多更加实用，漂亮的主题

思考：

如何让Spark与mx控件同时应用一款主题?

Flex SDK 4 新特性 第二节　布局

这篇教程中我们将通过一些简单的例子初步的学习Flex SDK 4中新的布局方式

学习目标

1. 初步了解SDK4中布局有哪些变化。
2. 学会使用新的布局方式。

实现步骤

在之前的Flex SDK中，布局是在控件或者容器中单独定义的，因此我们的控件，比如List, TileList, and HorizontalList，他们全都有完全相同的方法，只是布局不同。在Flex 4中，布局已经从控件中剥离出来。这样我们可以更加灵活的对控件进行布局，比如在运行时将容器的布局方式由横向改为纵向。这在Flex3中是很难实现的，因为我们很难在运行时把List改为TileList 或者将Hbox改为Vbox。

在Flex 4的spark包中已经没有了原来的Hbox, Vbox, Box以及Canvas容器，取而代之的是Group，另一个新的容器是SkinnableContainer，它与Group的区别是我们可以为它定义皮肤，Group不能定义皮肤但有更高的执行效率以及使应用程序的体积更小。

我们现在来看一组例子，在Flex 4中定义List, TileList以及 HorizontalList

List `<s:List />`

HorizontalList

```
<s:List>

    <s:layout>
        <s:HorizontalLayout />
    </s:layout>
</s:List>
```

Tiled List

```
<s:List>

    <s:layout>
        <s:TileLayout />
    </s:layout>
</s:List>
```

code

FlexBuilder 4 快速入门　代码片段

上面的例子中我们通过设置layout属性中的HorizontalLayou与TileLayout实现了3种不同List的布局，可以看到虽然从形式上样式的定义变得更“复杂”了，但是布局的解耦让我们对控件的使用变得更加灵活。

除了控件的布局方式改变外，Flex 4中的滚动条也从控件中剥离了出来，默认的容器是没有滚动条以及virtualization的，我们来看下面显示滚动条的例子。

```
<?xml version="1.0" encoding="utf-8"?>

<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark" xmlns:mx="
library://ns.adobe.com/flex/halo">

<s:Panel title="Horizontal Panel" width="300" height="220" left="20" top="20">
    ______

  <s:Scroller width="100%" height="100%">

    <s:Group>

      <s:layout>

          <s:HorizontalLayout useVirtualLayout="true" />

       </s:layout>

       <s:TextInput />

       <s:Button label="clear" />

      <mx:DateChooser />

      <s:Button label="submit" />

    </s:Group>

  </s:Scroller>

</s:Panel>

</s:Application>
```

code

FlexBuilder 4 快速入门 代码片段

上面的代码中我们通过使用Scroller控件给Group容器加上了滚动条，如果把Scroll放在Panel的外面则滚动条出现在Panel的外部

编译运行上面的代码（图四十九）

图四十九

创建项目

布局的基本介绍就到这里，同学们可以到Spark Layout Specification 查看更详细的内容。

总结：

布局在Flex 4中有了很大的变化，就像我刚才提到的，布局的解耦让我们对容器，控件的界面操作变得更加灵活与合理。不过Flex 3中的布局方式仍然在mx控件中有效，所以我们也不必担心Flex 3程序向Flex 4移植的问题。

Flex SDK 4 新特性 第三节　特效

这一节中我们来学习Flex SDK 4中新的特效

学习目标

1.了解Flex SDK4中特效有了哪些变化。

2. 学会使用它们。

实现步骤

特效在RIA 应用中一直有举足轻重的地位，特效可以让程序中界面的变化变得平滑，流畅，甚至绚丽。合理的使用特效可以有效的提升用户体验。Flex 4中的特效有了很多的改变，从官方的描述来看它变得更加的强大以及有效率。

在Flex 3 中，特效只对继承于UIComponent的控件有效，但现在Spark.effects.*包中的特效对所有的对象都有效，包括原有的Halo 控件, 新的Spark 控件, 甚至graphic绘制的图像。Flex 4中的特效都继承于Animate类，Animate继承于Effect类，这样新的特效与Flex3的特效就完全是平行关系 (Flex 3的特效继承于TweenEffect类)，我们移植代码时就可以保留原有的代码不变了。

注：没有Flex 3经验直接学习Flash Builder 4的通可以直接学习这篇文章，不需要了解Flex 3的特效机制 

现在我们来通过一些简单例子展示Flex 4中的特效是如何执行的，在这里我对每个特效的具体属性就不做过多的解释，同学们通过实践以后可以通过Flash Builder 4中的帮助了解更详细的类容。

1. 使用Animate实现简单的特效。

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:d="http://ns.adobe.com/fxg/2008/dt"
    xmlns:s="library://ns.adobe.com/flex/spark">
    <fx:Declarations>
        <s:Animate id="mover" target="{button}" duration="1000">
            <s:SimpleMotionPath property="x" valueFrom="0" valueTo="100"/>
            <s:SimpleMotionPath property="y" valueTo="100"/>
            <s:SimpleMotionPath property="width" valueBy="20"/>
        </s:Animate>
    </fx:Declarations>
    <s:Button id="button" click="mover.play()"/>
</s:Application>
```

code

FlexBuilder 4 快速入门　代码片段

编译运行上面的代码，同学们可以看到按钮的动画效果。熟悉Flex 3 特效的同学可能已经注意到Animate与Flex

3中AnimateProperty的不同，AnimateProperty只能处理一个属性的动画。

使用Animate可以进行一些通用的特效动画，在Flex 4 中还有大量继承于Animate的特效类为同学们封装了很多标准的特效动画，下面我们使用Move与Rotate3D组合做一个按钮的360度旋转加移动。

编译并运行下面的代码

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:d="http://ns.adobe.com/fxg/2008/dt"
    xmlns:s="library://ns.adobe.com/flex/spark">
    <fx:Declarations>
        <s:Parallel id="transformer" target="{button}">
            <s:Move id="mover" xFrom="50" xTo="150" autoCenterTransform="true"/>
            <s:Rotate3D id="rotate" angleYTo="360" autoCenterTransform="true"/>
        </s:Parallel>
    </fx:Declarations>
    <s:Button id="button" x="50" y="100" label="你好"
        click="transformer.play()"/>
</s:Application>
```

code

FlexBuilder 4 快速入门 代码片段

上面的代码让按钮移动了100像素，并且以按钮的中心作为顶点沿Y轴旋转360度，我们使用了Parallel将两个特效打包执行，这也是比较常用的做法，同学们也可以单独对里面的move和rotate执行play()以单独执行他们，同学们也许注意到了这里我使用了中文，熟悉Flex 3的同学可能知道在Flex 3中实现中文的翻转效果是很麻烦的，但现在...试试上面的代码你就知道了:)

下面我继续给大家一些复杂一点的例子。

state的切换是Flex视图转换时比较常用的方法，state切换时加入动画可以让画面切换更加的平滑，自然。下面这个例子展示了Fade特效在state转换时的作用。

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/halo">
    <s:states>
        <s:State name="state1"/>
        <s:State name="state2"/>
    </s:states>
    <s:transitions>
        <s:Transition>
            <s:Fade targets="{[button0, button1, label2]}"/>
        </s:Transition>
    </s:transitions>
    <s:Button id="button0" label="按钮1" x="100" y="0" visible="true"  visible.state2="false"/>
    <s:Button id="button1" label="按钮2" x="100" y="50" alpha="0"  alpha.state2="1"/>
    <mx:Label id="label2" text="标签3" x="100" y="100" includeIn="state2"/>

    <s:Button label="Toggle State" click="currentState = (currentState=='state1')?'state2'
:'state1'"/>
</s:Application>
```

code

FlexBuilder 4 快速入门 代码片段

在这个例子中我们从state1切换到state2，这个过程中按钮1在state1时显示，按钮2与标签3在state2时显示。Transition用于在state转换是执行Fade特效。 targets="{[button0, button1, label2]}"表示Fade的对象是这3个对象。

里又有必要提一下Label控件，Flex3中我们要对Label执行动画效果需要特殊的设置，在 Flex 4中也已经不需要了。

最后一个例子我们来看一个Flex4中新的特效AnimateColor，用于颜色的变化。

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/halo">
    <s:states>
        <s:State name="state1"/>
        <s:State name="state2"/>
    </s:states>
    <s:transitions>
        <s:Transition>
            <s:AnimateColor target="{center}" duration="150"/>
        </s:Transition>
    </s:transitions>
    <s:Group mouseDown="currentState='state2'" mouseUp="currentState='state1'">
        <s:Ellipse x="50" y="50" width="100" height="100">
            <s:fill>
                <mx:RadialGradient>
                    <mx:GradientEntry id="center" color="0xf0f0f0" color.state2="0x808080"
                        ratio="0"/>
                    <mx:GradientEntry id="edge" color="0x404040"
                        ratio="1"/>
                </mx:RadialGradient>
            </s:fill>
        </s:Ellipse>
    </s:Group>
</s:Application>
```

code

FlexBuilder 4 快速入门 代码片段

对特效的介绍就到这里，同学们可以访问Chet Haase的文章Effects in Adobe Flex 4 SDK beta – Part 1: Basic effects 了解更详细的内容。

总结：

这里只介绍了Flex 4中很少的一部分内容，目的是让同学们从这些例子中学习特效的基本用法，对熟悉Flex3的同学来

说当然不会忘记那令人抓狂的特效用法以及让人更加抓狂的执行效率，在Flex 4中这一切都有了改善，让我们在实际的项目中检验

Flex 4绚丽的特效吧 :)

Flex SDK 4 新特性 第四节　样式

在Flex 4中，对Css的支持有了质的飞跃，我们就来学习有哪些变化

学习目标

1.了解Flex4 的对样式的设置有了哪些变化

2. 学会使用它们

实现步骤

在Flex 4以前，Flex对Css的支持是很别扭的，相对于HTML强大的Css功能，Flex只能说继承了很表面的一部分。在众多Flex 开发者的投票下，Flex4终于对Css支持做了很大的改进，虽然说仍然不能像HTML Css那样随心所欲，但也足够满足在任何项目中的需求了

在主题章节中我提到了在新的样式设置中增加了对名称空间的支持，这主要针对不同控件包中的控件设置样式，我们来回顾一下，看下面的css代码

```
<fx:Style>

    @namespace s "library://ns.adobe.com/flex/spark";
  @namespace mx "library://ns.adobe.com/flex/halo";

  s|Button {
    color: #FF0000;
  }

  mx|Button {
    color: #000000;
  }

</fx:Style>
```

code

FlexBuilder 4 快速入门　代码片段

第一排的名称空间申明表明了s代表spark的内容而mx代表halo也就是我们flex3中的所有控件。 s|Button表示将spark包下的Button样式改变，而mx|Button则表示设置mx包下Button的样式。如果同学们还有一点不理解也没关系，只要记住这种设置方式即可，在会用以后再通过看帮助了解名称空间更多的内容。

现在我们来就通过例子看看Flex4对Css的支持都有哪些提高

1. 同一控件多样式的设置 在Css文件中定义下面的样式

```
.blackButton{
        base-color:#000000;
}
.whiteFont{
        color:#ffffff;
}
```

code

FlexBuilder 4 快速入门 代码片段

主mxml文件中添加代码

```
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/halo" minWidth="1024" minHeight="768">

        <fx:Style source="css.css" />
    <s:Group horizontalCenter="0" verticalCenter="0">
            <s:Button label="样式设置" styleName="blackButton whiteFont" />
    </s:Group>
</s:Application>
```

code

FlexBuilder 4 快速入门 代码片段

对button设置了styleName="blackButton whiteFont" 两个样式，运行结果见 图五十（一）

2. 通过控件ID设置样式

在Css文件中定义下面的样式

```
#okButton{
        baseColor:#ae0000;
    color:#ffffff;
}
```

code

FlexBuilder 4 快速入门 代码片段

在上面的代码中加入另一个Button 运行结果见 图五十（二）

```
<s:Button id="okButton"  label=" Id设置样式"/>
```

code

FlexBuilder 4 快速入门 代码片段

3. 子控件样式设置

在Css中定义下面的样式

```
s|Panel s|Button{
        baseColor:blue;
        color:#ffffff;
}
```

code

FlexBuilder 4 快速入门 代码片段

在代码中再加入一个Panel

```
<s:Panel id="test" title="Panel" >
            <s:layout>
        <s:VerticalLayout  paddingLeft="10" paddingTop="10" paddingRight="10"
paddingBottom="10"
useVirtualLayout="true" />

        </s:layout>
            <s:Button  label=" 子控件设置" />
</s:Panel>
```

code

FlexBuilder 4 快速入门 代码片段

运行结果见 图五十（三）

图五十

样式

4.状态样式设置

在Flex3或者之前的版本中我们必须在一个样式里设置比如一个按钮的所有状态样式，up, down, over这样做很不方便的一点是如果我需要修改某一个状态的外观意味着我要重写整个样式，而现在不必了。

在Css中定义下面的样式

```
#cancelButton:up{
            baseColor   : #538787 ;
             color   : #304f68  ;
}
#cancelButton     :down {
            baseColor   : #917541 ;
             color   : #b8a553 ;
}
#cancelButton     :over   {
            baseColor   : #b8a553 ;
             color   : #333333 ;
}
```

code

FlexBuilder 4 快速入门 代码片段

这里我对id为cancelButton的按钮做了样式设置，我们同样可以针对所有的Button, 只需修改前缀为 s|Button:xx即可。

代码中加入一个Button

```
<s:Button id="cancelButton" label="状态设置"  />
```

code

FlexBuilder 4 快速入门 代码片段

运行结果如下（图五十一）

图五十一

状态样式设置

以上就是Flex4中对Css样式的改进，是不是要强大得多了呢？最后还有一点，同样在主题章节说到过，Spark主题中使用了很多图片作为控件的皮肤，因此Spark只支持部分的色彩样式, 它们是baseColor, color, contentBackgroundColor, focusColor, symbolColor, selectionColor, and rollOverColor,如果同学们想改变一些控件的背景，比如Button的背景样式，我们需要重新定义背景的skin图片才可以做到

总结：

Flex 4中的样式设置已经可以很好的满足项目中的需要，但仍然期待在下一个版本的Flex SDK中样式会变得更加的强大。

Flex SDK 4 新特性 第五节　状态

我们来了解状态(State)在Flex 4中的变化

学习目标

1.了解Flex SDK4中State有了哪些变化

2. 学会使用State

实现步骤

State在Flex SDK 4中将变得更加的灵活而且好用 (真的非常好用，试了这个再回到Flex3想死的感觉都有了，LOL)。我们将通过一组例子来了解State的改变以及使用方法。

1. States数组里现在只定义一组state, 不会再出现其他的标签.

```
<s : states   >
    <s:State name="submitState" />
</s:states>
```

code

FlexBuilder 4 快速入门　代码片段

上面的语法定义了submitState状态。

2. 既然在states数组中没有任何其他的标签那自然也就没有了Flex3中的AddChild与RemoveChild标签，取而代之的是每个控件中都有的includeIn(包含在)以及excludeFrom(从xx排除)属性。

接下来我们通过一个例子看看1,2具体的使用方法。

下面是Flex3中State的代码，作用是在submitState state中添加 submit 按钮并去掉firstTextInput 输入框，这个用法显得较为繁琐。

```
<mx:states>
    <mx:State name="submitState" basedOn="">
        <mx:AddChild relativeTo="{loginForm}" >
            <mx:Button label="submit" bottom="10" right="10"/>
        </mx:AddChild>
        <mx:RemoveChild target="{firstTextInput}"/>
    </mx:State>
</mx:states>
<mx:TextInput id="firstTextInput" />
<mx:Canvas id="loginForm" />
```

code

FlexBuilder 4 快速入门 代码片段

来看Flex4中实现上面状态的代码

```
<s:states>
    <s:State name="submitState" />
</s:states>
<s:TextInput id="firstTextInput" excludeFrom="submitState" />
<s:Group id="loginForm" >
    <s:Button label="submit" bottom="10" right="10" includeIn="submitState"/>
</s:Group>
```

code

FlexBuilder 4 快速入门 代码片段

这样是否更直接易读了呢？

用excludeFrom="submitState"表示在submitState中移除firstTextInput输入框，includein="submitState"表示在submitState状态中加入submit按钮。

3. SetProperty, SetStyle, 和 SetEventHandler 也被去掉，取而代之的是控件中的"."语法。

来看下面这个复杂点的例子，这个例子中我们在两个状态submitState与clearState之间切换，切换时改变按钮submit的属性，样式以及click事件

Flex3代码

```
<mx:states>
    <mx:State name="submitState" basedOn="">
        <mx:SetProperty target="{submitButton}" name="label" value="submit" />
        <mx:SetStyle target="{submitButton}" name="textDecoration" value="underline"/>
        <mx:SetEventHandler target="{submitButton}" name="click" handler="trace('done');"/>
    </mx:State>
    <mx:State name="clearState" basedOn="">
        <mx:SetProperty target="{submitButton}" name="label" value="clear" />
        <mx:SetEventHandler target="{submitButton}" name="click" handler="emptyDocument()" />
    </mx:State>
</mx:states>
<mx:Button id="submitButton" />
```

code

FlexBuilder 4 快速入门 代码片段

Flex4代码

```
<s:states>
    <s:State name="submitState" />
    <s:State name="clearState" />
</s:states>
<s:Button label.submitState="submit" textDecoration.submitState="underline"
   click.submitState="trace('done')" click.clearState="emptyDocument()"
   label.clearState="clear" textDecoration.clearState="none"/>
```

code

FlexBuilder 4 快速入门 代码片段

这个语法真是很有意思，也是非常的易读，textDecoration.submitState="underline" 表示textDecoration样式submitState状态时值等于underline， click.submitState="trace('done')" click.clearState="emptyDocument()。怎么解释就不用我在多说了吧 :)

同学们注意Flex 4 的state语法只能在MXML 2009名称空间里使用，Flex3的State语法只能在MXML 2006名称空间使用，所以我们不能混合两个状态的使用。

总结：

就像我开篇说过的，Flex4中的States使用方法与Flex3有了比较大的区别，变得直观，易用，实用性也更强。期待在新的项目中可以使用Flex4.

思考：

试试用States做控件皮肤的设置

Flex SDK 4 新特性 第六节　双向绑定

这一节介绍Flex4中双向绑定的使用方法

学习目标

学会使用双向绑定

实现步骤

Flex中一个很有用的功能是数据的绑定，比如我们有属性a,以及输入框b,我们可以把属性a与输入框b绑定起来，这样改变a的值时，输入框b的值也会相应变化。这种绑定是单项的。在Flex SDK4以前没有直接的双向绑定，所以当我们想反过来通过设置b的值来改变a时就会比较麻烦。而Flex4为我们提供了双向绑定的方法，下面我们来举个例子。

场景中有t1与t2两个输入框，无论我们改变哪一个输入框的值，另一个都会跟着变化。

code

```
<s:TextInput id="t1" text="@{t2.text}" />
<s:TextInput id="t2" />
```

FlexBuilder 4 快速入门　代码片段

代码中的text="@{t2.text}"就实现了双向绑定。下面是另一个双向绑定的方法

code

```
<fx:Binding source="input1.text" destination="input2.text" twoWay="true"/>
```

FlexBuilder 4 快速入门　代码片段

twoWay="true"表示双向绑定。

同学们要注意在样式，特效，数据服务以及远程对象的使用时是不能使用双向绑定的。

总结：

很简单的内容，很实用的功能。

Flex SDK 4 新特性 第七节　ASDoc

Flex4中的ASDoc已经完全支持MXML控件

学习目标

学会如何使用ASDoc生成我们的控件说明书

实现步骤

ASDoc一款用于生成代码说明书的工具，他能自动将我们书写在程序中的注释转换为说明书中控件属性，方法等的说明。所以如果我们非常规范的书写了自定义控件的注释，在最后我们可以通过一个简单的命令生成一本控件的说明书。

在Flex 3中ASDoc只能完美的生成ActionScript控件的说明书，而对MXML控件只对<scripit>中的注释有效。在Flex 4中，ASDoc已经能完美生成MXML控件的说明。我们现在就通过一个例子学习如何使用ASDoc

任意创建一个Flex项目，之后新建一个名为MyVBoxComplex.mxml的文件，拷贝下面的代码到文件中

```
<?xml version="1.0"?>
<!-- MyVBoxComplex.mxml -->
<!---
    The class level comment for the component.
    This tag supports all ASDoc tags,
    and does not require a CDATA block.
-->
<mx:VBox xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:mx="library://ns.adobe.com/flex/halo"
    xmlns:s="library://ns.adobe.com/flex/spark">
    <!---
        Comment for language element - this comment will be ignored.
    -->
    <fx:Script>
        <![CDATA[
            import flash.events.MouseEvent;
            /**
             * For a method in an &lt;Script&gt; block,
             * same rules as in an AS file.
             *
             * @param eventObj The event object.
             */
            public function handleClickEvent(eventObj:MouseEvent):void {
                dispatchEvent(eventObj);
            }
            /**
             * For a property in an &lt;Script&gt; block,
             * same rules as in an AS file.
```

```
            */
            public var myString:String = new String();
        ]]>
    </fx:Script>
    <!---
        Comment for first button appears in the output.
     -->
    <s:Button id="myButton" label="This button has a comment"
        click="handleClickEvent(event);"/>
    <s:Button id="myButton2"
        label="Has id but no comment so appears in output"/>
    <!---
        Comment for button with no id is ignored by ASDoc.
     -->
    <s:Button label="This button has no id"/>
</mx:VBox>
```

code

FlexBuilder 4 快速入门 代码片段

asdoc命令存在于"FlashBuilder4安装目录/sdks/4.0.0/bin"

我们现在通过asdoc在bin目录下创建document文件夹并在其中生成上面MyVBoxComplex的说明书。进入bin目录，在命令行中输入下面命令。我的命令是在Mac下运行的，所以我的MyVboxComplex.mxml路径是linux格式，Windows下的同学直接输入windows路径代替即可，比如C:\flex\projects\demo\src\mx\controls\MyVBoxComplex.mxml

```
asdoc -doc-sources ~/Documents/Adobe\ Flash\ Builder\
Beta/ASDocDemo/src/MyVBoxComplex.mxml -output document
```

生成的文件列表见“图五十二”；双击index.html，看到的页面见"图五十三"。

图五十二

ASDoc生成列表

图五十三

ASDoc生成列表

同学们可以清楚看到MyVBoxComplex的说明页面。

ASDoc的简单介绍就到这里，目的是让大家通过这个例子对ASDoc有一定的了解。ASDoc细讲起来有很多内容，这里就不详述了。感兴趣的同学可以在Flash Builder4的帮助文档中输入ASDoc查看详细说明

思考：

如何生成多个控件的说明？

哪些注释在ASDoc中会被忽略？

Flex SDK 4 新特性 第八节　SWFObject 与 HTML Template

了解Flex SDK 4中的Html Template与swfobject 2

学习目标

了解什么是html template和swfobject 2，它们在项目中起到什么作用 。

实现步骤

在每个Flex 4项目中都有一个html-template文件夹，里面的内容如下：（图五十四）

图五十四

html-template文件夹

1. 我们先来了解history文件夹中的内容

history.js, history.css以及historyFrame.html用于记录Flex程序中每一个state的地址，比如我们从state1,切换到state2,从Tab1跳转到Tab2时它为我们保存对应这些state的浏览器地址。如果我们在项目中激活 “允许继承浏览器导航功能的选项” ，那就意味着我们可以:

1. 通过前进和后退返回在程序中放问过的state。 2. 将某一state的地址保存进收藏夹，以后可以直接通过地址打开的这个state的页面。 3. 将某一state的地址给其他人，他们可以直接链接到这个states页面。

换句话说如果我们整个程序的试图切换都用state方式，那么我们可以像浏览普通网页一样在浏览器中访问我们程序的任意指定页面。

激活“允许继承浏览器导航功能的选项”的方法是

“选中项目-->属性 -->Flex编译器 --> 允许继承浏览器导航功能的选项。 ”

提示：如果想进一步了解historyFrame.html的工作机制，请同学们在Flash Builder 4的帮助中输入deep linking 。

2. html-template以及swfobject

html-template的作用是在编译生成项目时生成一个Html页面，用户通过访问Html页面来访问程序，而不是直接访问生成的swf文件。为什么不直接访问swf文件？因为html-template除了显示swf程序外还帮我们做了很多其他事，比如

1. 引用history系列文件来支持浏览器导航。
2. 检测用户Flash player版本，一键自动安装。
3. 页面标题，编码，宽度，程序质量，等等。

在Flex 4之前，是直接使用一段javascript加入到html-template文件中实现上述功能，而从Flex 4开始，正式引入了swfobject 2(Flash CS4中发布也使用swfobject 2)，它是一个开源的标准库，用于将swf嵌入到html页面，以及实现一些其他的功能。html-template并不是不可改变的，同学们可以更具自己项目的需要修改里面的内容，如果你使用了自己的html页面，你也可以通过引用swfobject.js以及history的方法实现html-template中提供的功能。

提示：如果想进一步了解html-template的详细信息，请同学们在Flash Builder 4的帮中中输入swfobject或者html-template。

第五章 ：自定义组件开发

郭峰

5.1 自定义Flex组件

陆仕桑

5.2 MXML组件开发

郭峰

5.3 ActionScript组件开发

自定义组件开发 第一节　自定义Flex组件

如下图所示，组件窗口中的“控件”是什么？为什么会听到“Flex 控件开发”和“Flex 组件开发”这两种不同的说法呢？控件和组件有什么关系？（图五十五）

图五十五

组建与控件的关系

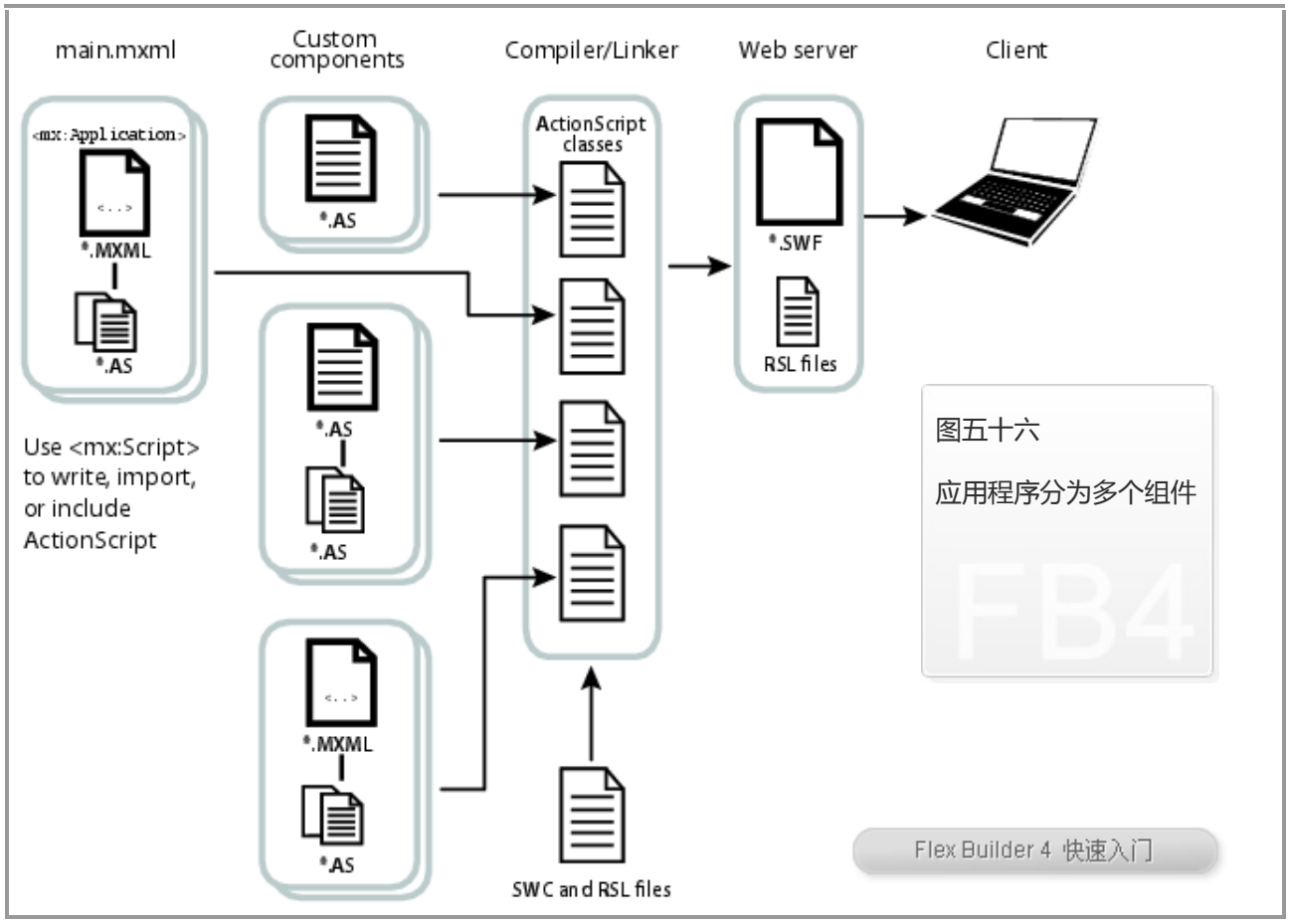
Adobe Flex 支持组件开发模型，通常情况下，我们使用 Flex 中预定义的组件，根据需求开发应用程序。当预定义组件不能满足需求时，我们还可以创建自定义组件来构建应用程序。在软件行业中，一般把“component”翻译为“组件”，把“control”翻译为“控件”。其中，术语“组件”指任何可复用的、可以与其他对象交互的对象，如 Flex 中的 validators、formatters、effects、managers、controls、containers 等等；术语“控件”则指能够在界面上看到的组件，也称为可视组件，如 Flex 中的 controls、containers、borders 等等。所有的控件都是组件，反之，则不一定。Flex 中的所有类都可以被称为组件，其中能够在界面上看到的组件都可以被称为控件。

为什么要创建自定义组件

- 通过组件，我们可以将应用程序分为能够独立开发和维护的模块。通过在自定义组件中实现通用逻辑，我们可以创建一系列可重用的组件，实现多个应用程序间的代码共用。
- 此外，通过让自定义组件继承 Flex 中预定义的类，我们可以扩展 Flex 中的类，给现有的组件添加更多的行为，或者实现具有全新行为的组件。比如，我们可以创建自定义版本的 Flex 可视组件及非可视组件。

我们可以通过在一个单独的 MXML 文件中包含所有 MXML 代码和支持性的 ActionScript 代码来构建整个应用程序。随着应用程序不断变大，这个 MXML 文件的大小和复杂度不断增长，应用程序很快会变得难以理解和调试，多个开发人员同时的话将十分困难。为了解决这个问题，我们可以将整个应用程序分为互不关联的功能单元，也称为模块。从而，不同的开发人员或者开发小组可以独立的开发和调试各个模块；可以在不同的应用程序中使用开发好的模块，从而避免重复性的工作；可以更快的分离、调试应用程序中产生的错误。

在 Flex 中，一个模块对应着一个在 MXML 文件或者 ActionScript 文件中实现的组件。下图显示了如何将一个应用程序分为多个组件。（图五十六）



图五十六

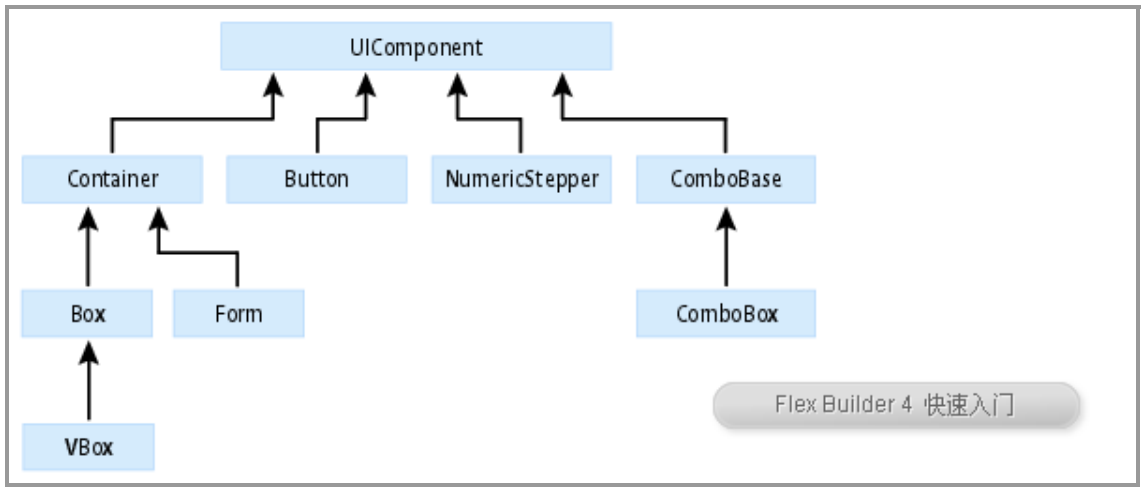
应用程序分为多个组件

怎样创建自定义组件

Flex 是以多个 ActionScript 类层次结构的形式来实现的。这些类层次结构中定义了组件类（各种控件）、管理类（Manager 类）、数据服务类（HttpService、WebService 等）以及实现 Flex 其他特性的类。通过继承类层次结构中相应的类，我们可以扩展现有的组件或者创建新的组件来实现自定义组件。

在 Flex 中，所有的可视组件都继承自 UIComponent ActionScript 类。常用的非可视组件有 Validator、Formatter 和 Effect。我们通过使用 MXML 和 ActionScript 语言扩展类层次结构来创建自定义组件。这些组件将继承超类中的属性、方法、事件、样式和特效。（组图五十七）

UIComponent 类层次结构

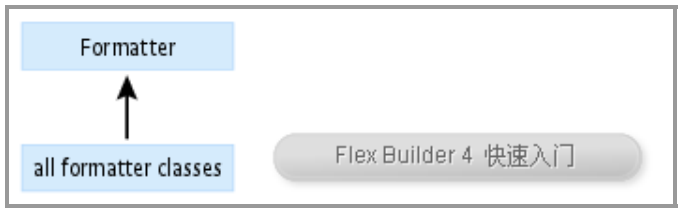


图五十七（组）

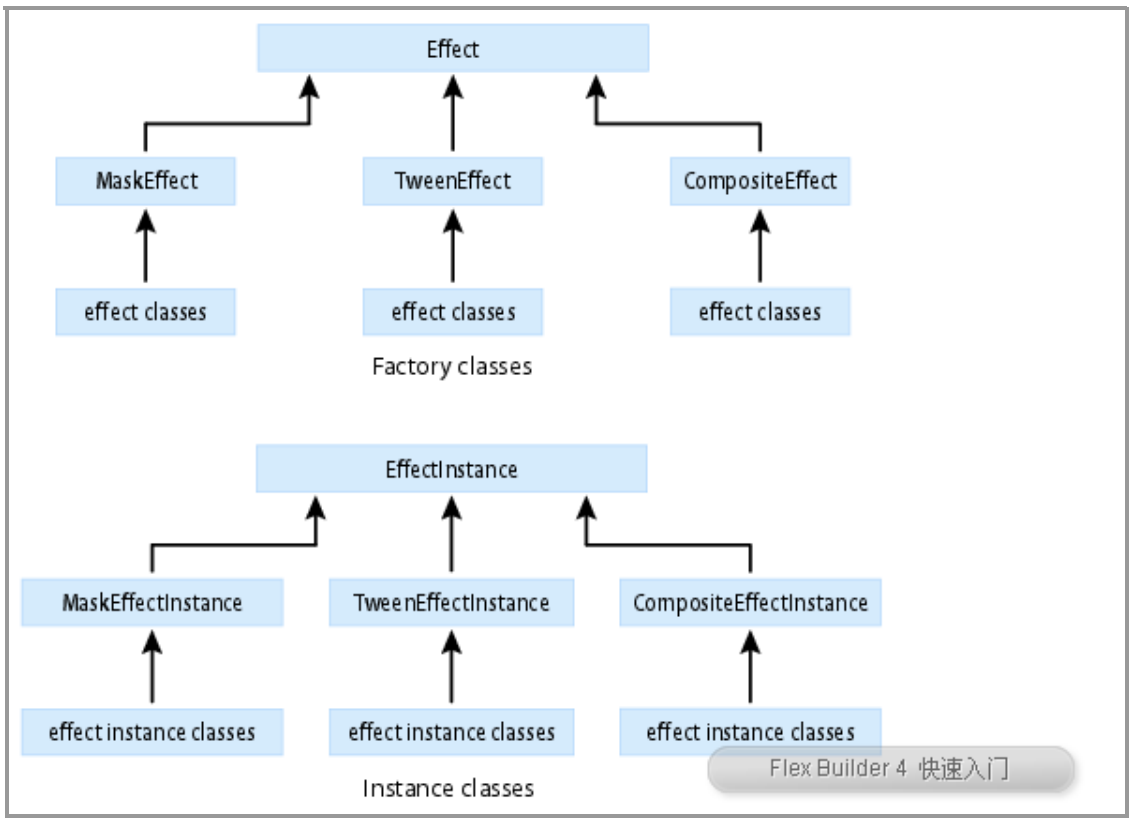
常用的非可视组件

Validator 类层次结构

Formatter 类层次结构



Effect 类层次结构



通过扩展现有的组件来创建自定义组件：

- 我们可以定制现有的组件，使其满足应用程序的要求。比如，我们可以创建一个继承自 Button 的自定义组件，将其“Label”属性设定为“提交”，用来统一所有表单上的提交按钮。
- 我们可以修改现有组件的行为。比如，让所有 VBox 中的组件都按从下至上的顺序排列。
- 为现有的组件添加新的逻辑和行为。比如，我们可以让 TextInput 控件支持组合按键来删除输入的内容。

当扩展现有的组件无法满足应用程序要求时，我们需要创建新的自定义组件。在 Flex 中，如果创建的自定义组件是可视组件，并希望能够被 addChild 及相关方法所调用（这意味着所创建的自定义组件可以添加到容器中去，可以参与界面的布局），所创建的自定义组件需继承 UIComponent 类。

在实现自定义组件之前，我们需要决定是在 MXML 文件中实现还是在 ActionScript 文件中实现，这取决于应用程序的需求。

- MXML 组件和 ActionScript 组件两者都定义新的 ActionScript 类。
- 基本上所有在 ActionScript 组件中能做的事情都可以在 MXML 组件中做。对于简单的组件，比如修改现有组件的行为或者为其添加新的基本特性，在 MXML 中实现起来更快、更简单。
- 如果组件是一个包含其他组件的组合组件，而且需要用布局容器来设置这些组件的大小和位置，使用 MXML 来定义组件。
- 如果需要修改组件的行为，例如对子组件进行布局，应当使用 ActionScript。
- 如果是通过继承 UIComponent 来创建一个可视组件，使用 ActionScript。
- 如果是创建非可视组件，例如 fomatter、validator 或者 effect，使用 ActionScript。
- 如果是添加日志支持，使用 ActionScript。

用 ActionScript 创建自定义组件时，需要创建一个继承自 Flex 类层次结构中相关类的子类。这个子类的名称（例如 MyAsButton），必须同 ActionScript 文件相同（比如，MyAsButton.as）。子类将继承超类中的所有属性和方法。该例中，我们可以通过<MyAsButton> 标签在 MXML 中引用这个组件。

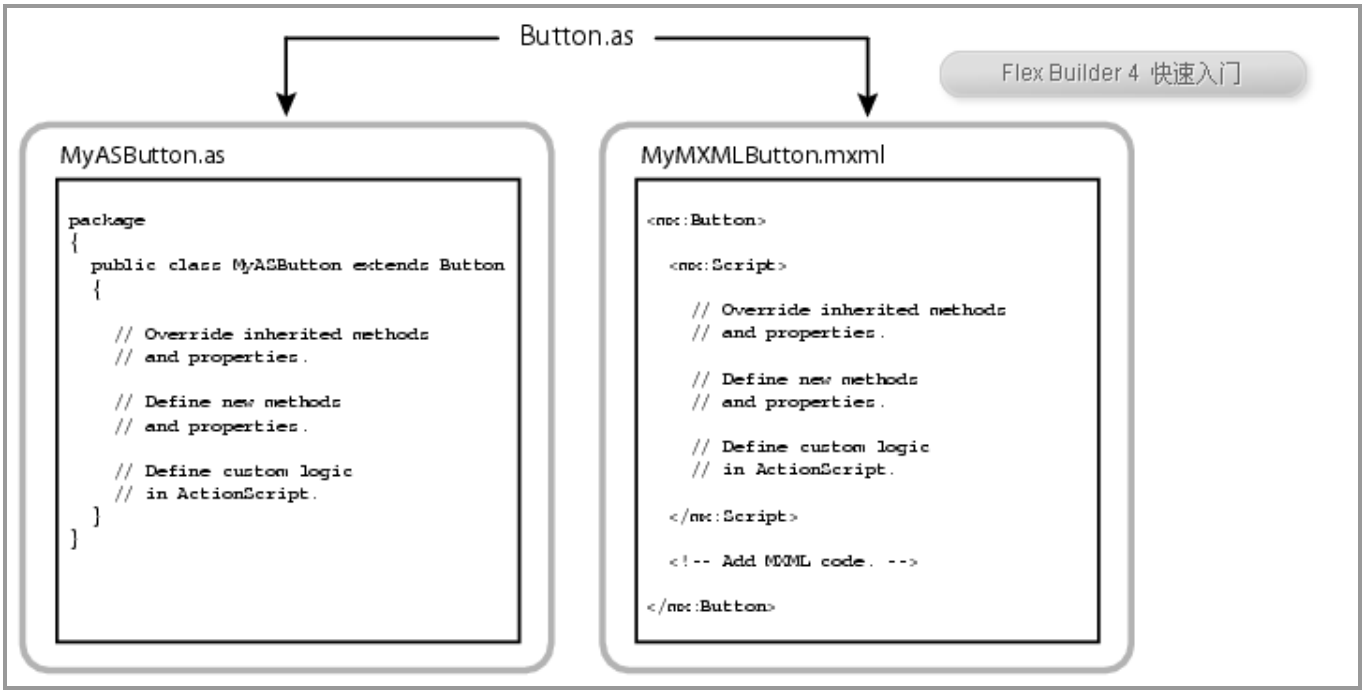
当我们使用 MXML 来定义组件时，Flex 编译器自动创建一个 ActionScript 类，MXML 文件的名称（比如，MyMXMLButton.mxml）对应自动创建的 ActionScript 类的名称。该例中，自动生成的 ActionScript 类名称为

MyMXMLButton，我们可以通过<MyMXMLButton> 标签在 MXML 中引用这个组件。

下图显示了两个继承自 Flex Button 组件的组件，一个定义在 ActionScript 中，一个定义在 MXML中。（图五十八）

图五十八

应用程序分为多个组件



两个组件都继承自 Button 类。因此，两个组件都继承了 Button 类中标记为 public 和 protected 的属性、方法及其他元素。在任意一个组件中我们都可以重写继承的元素、定义新的元素以及添加自定义的逻辑。

我们无法重写由变量定义的属性，但是可以重写由 getter 和 setter 方法定义的属性。

我们可以重新设置该属性的值。通常情况下，我们可以在 ActionScript 组件的构造函数重新设置该属性的值；

因为 MXML 组件不支持构造函数，因此我们需要在事件处理程序中设置该属性的值。

此外，当我们使用 MXML 自定义组件的时候，Flex 编译器将帮助我们完成大量的工作，例如调用 addChild 将子组件添加到自定义组件中。这使得在 MXML 中创建自定义组件比在 ActionScript 容易的多。

如何在项目中使用自定义组件

创建好的自定义组件以特定类型的文件形式存在于文件系统中，我们需将这些文件引入我们的项目，然后才可以在项目中使用这些组件。

Flex 组件的文件类型

文件类型	扩展名	说明
MXML	.mxml	在 MXML 文件中实现的组件。
ActionScript	.as	在 ActionScript 类中实现的组件。
SWC	.swc	在 MXML 或 ActionScript 文件中实现组件后，将其打包进 SWC 文件中。SWC 文件中包含打包好的可以在多个应用程序间复用的组件。当生成 SWF 文件时，SWC 文件会被编译进应用程序中。
RSL	.swc	在 MXML 或 ActionScript 文件中实现组件后，可以通过 RSL 文件来部署。RSL 是一种独立文件，能够被 SWF 文件分别下载并缓存在客户端供多个应用程序的 SWF 文件使用。

对于 MXML 和 ActionScript 文件

当使用 Flex SDK 编译应用程序的时候，我们需要设置定义了自定义组件的 MXML 或者 ActionScript 文件是位于应用程序的目录结构中，还是位于由多个应用程序共享的目录结构中。

例如，当我们创建只有一个应用程序使用的组件的时候，通常情况下将组件存放在应用程序的目录结构中。我们可以在应用程序主文件所在的目录下创建子目录，然后将定义了组件的 MXML 或者 ActionScript 文件放置在创建的子目录中。当 Flex 编译应用程序的时候，组件将随同整个程序被编译进产生 SWF 文件中。

或者，当组件由多个应用程序共享的时候，我们将定义了组件的 MXML 或者 ActionScript 文件放在一个共享的目录结构中，并将其包含在应用程序的 ActionScript 源路径中。当 Flex 编译应用程序的时候，也会编译包含进 ActionScript 源路径的组件。

编译时，通过下列方法设置共享组件所在的目录：

- 对于 Flash Builder，打开项目属性对话框，然后选择构建路径设置 ActionScript 源路径。
- 对于 mxmlc 编译器，设置 source-path 选项来设定包含共享的 MXML 和 ActionScript 文件的目录的位置。

对于 SWC 文件

SWC 文件是 Flex 组件的存档文件。SWC 文件使得开发者之间能够很容易的交换组件。只需要交换一个单独的文件便可以了，而不是一大堆的 MXML 或者 ActionScript 文件、图片或其他资源文件。此外，SWC 文件中的 SWF 文件是编译过的，这意味着其中的代码在通常情况下是隐藏了的。最后，将组件编译为 SWC 文件能够很容易的实现名字空间的分配。

SWC 文件包含一个或多个组件，并且按 PKZIP 格式压缩或解压缩。我们可以用 WinZip、JAR 或者其他存档工具打开。然而，不要手动改变 SWC 文件中的内容，不要尝试在外部运行的 SWC 文件中包含的 SWF 文件。

编译时，通过下列方法设置 SWC 文件所在的目录：

- 对于 Flash Builder，打开项目属性对话框，然后选择构建路径设置包含 SWC 文件的库路径。
- 对于 mxmlc 编译器，设置 library-classpath 选项来设定包含 SWC 文件的目录的位置。

对于 RSL 文件 一种减少应用程序的 SWF 文件的方法便是将共享的资源放置在独立的文件中。这些独立的文件可以被分别下载并缓存在客户端。这些共享的资源只需要在客户端下载一次，就能够被多个应用程序在运行时加载。这些共享的文件被成为运行时共享库（RSLs）。

如果我们有多个应用程序， 并且这些应用程序共享一些列核心的组件和类，我们的用户只需要以 RSL 形式下载这些共享资源一次。只要应用程序在同一个域下，那么这些应用程序将使用同一个缓存过的 RSL。通过使用 RSL，我们能够减少最终生成的 SWF 文件的体积。使用 RSL 的最终效果，取决于使用该 RSL 的应用程序的数量。如果只有一个应用程序使用该 RSL，那么不会减少下载的总量，反而可能增加下载量。

编译时，通过下列方法设置 RSL 文件所在的目录：

- 对于 Flash Builder，打开项目属性对话框，然后选择构建路径设置包含 SWC 文件的库路径。
- 对于 mxmlc 编译器，设置 external-library-path 选项来设定编译时 RSL 文件的路径。设置 runtime-shared-libraries 来设定应用程序发布时 RSL 文件的相对位置。

总结：

本节主要介绍了为什么自定义组件以及如何自定义组件。为了实现模块化编程及代码的重用，我们通过继承 Flex 类层次结构中相关的类，在 MXML 文件或者 ActionScript 文件中实现自定义组件，并通过 MXML 文件、ActionScript 文件、SWC 文件将其引入我们的应用程序中。为了让大家分清楚组件和控件的区别，本节还简单提及了组件和控件的概念。此外我们可以看到，如果一个组件是定义在MXML文件中的，我们可以称其为 MXML 组件；如果是定义在 ActionScript 文件中的，我们则称其为 ActionScript 组件。接下来的内容中，我们将通过两个例子，来看看具体是如何实现自定义组件的。

思考：

编译使用 MXML 定义的组件的时候，如何知道 Flex 编译器自动创建了哪些代码？

自定义组件开发 第二节　MXML组件开发

本节，我们以“自定义播放器”为实例，来学习 MXML 组件开发的方法。

学习目标

1. 了解 MXML 组件开发的特点。

2. 学习使用 MXML 语言开发组件，逻辑部分要结合 ActionScript 语言。

3. 开发组件“自定义播放器”。

MXML 组件开发的特点

为什么要开发组件？简而言之，是因为组件有利于代码的维护和复用。组件可以被定义在 MXML 文件（以 .mxml 为后缀的文件）或者 ActionScript 文件（以 .as 为后缀的文件）中。因此，有两种组件开发的形式：MXML 组件开发及 ActionScript 组件开发。凡是定义在 MXML 文件中的组件都可以转化为定义在 ActionScript 文件中的组件。Flex SDK 中的大部分组件都是定义在 ActionScript 文件中的。

然而，MXML 组件开发也有其适用之处。MXML 组件比较适合“组合模式”的组件，通过 MXML，可以很方便的进行子组件的布局及数据绑定。 以下是一些 MXML 组件开发的示例，更多的内容还需读者在实践中体会摸索。

MXML 组件开发的优点：

1. 可以利用“设计”模式，进行所见即所得的界面开发。
2. 可以快速的添加子组件。不需要申明一个实例，然后再调用 addChild()，将其添加到父组件的布局中去。
3. 可以很方便地进行数据绑定。使用 “{ binding_expression }”可以快速将任意可绑定的数据源绑定到指定位置。
4. 可以很方便的定义类实例，不需要显示的初始化。例如定义<fx:Binding/>后，应用程序运行时会自动初始化该实例。

MXML 语言的缺点：

1. 没有 ActionScript 的辅助，无法完成复杂的逻辑。
2. MXML 组件默认都是 public 的，没有访问限制。
3. MXML 标签中定义的事件是不可以被移除的。如<mx:Button click="doClick()"/>，click 事件是不可以被移除的。
4. 不能自定义构造函数。

使用 Flash Builder 4 和 MXML 语言开发组件的步骤：

1. 新建 Flex 库项目。Flex 库项目编译后可以产生 swc 文件，可以作为组件发布的主要形式。
2. 新建 MXML 组件文件，文件名即为组件名。
3. 继承现有组件。文件内部第一级标签即为所继承组件名。
4. 在 MXML 文件主标签里加 implements 属性可以继承接口。
5. 在 MXML 文件里添加各种标签，包括可视组件，数据服务，验证，特效等。如果是非可视的标签，如特效等要

加到 <fx:Declarations> 标签里，这是 Flex 4 的新特点。

6. 可以在 <fx:Script> 标签里添加 ActionScript 代码，以实现逻辑，比如事件处理等。代码要被 <![CDATA[和]]> 包围。
7. 可以在 <fx:Style> 标签里添加CSS样式代码，以设置组件样式。
8. 可以添加 <fx:XML>、<fx:XMLList>、<fx:Array>、<fx:Model> 等实用标签，以提供更好的功能。这些标签都要加在 <fx:Declarations> 标签里。

开发实例

本节通过开发一个“自定义播放器”组件，来讲解 MXML 组件开发技术。界面布局、组件使用等部分采用 MXML 语言，逻辑部分采用 ActionScript 语言。因为是组合式组件，不涉及组件生命周期等，所以可以利用 MXML 来开发，方便布局和属性设置等。

先看一下最终使用效果图：（图五十九）

图五十九

播放器预览

上部即是我们要做的“自定义播放器”组件，中间和下部是文字介绍和播放列表，不属于“自定义播放器”组件。

接下来，和我一起一步一步开发“自定义播放器”。

1. 设计“自定义播放器”组件功能

播放器是一种很常用的组件，既可以是一个独立产品，也可以集成到大系统中，如教学系统、会议系统、娱乐网站等，主要分为音频播放器和视频播放器。

对于视频播放器，Flex 3 框架的 mx.controls.VideoDisplay 类提供了视频播放的视频显示和基本控制，但是没有提供控制界面等播放器必需的元素。Flex 4 框架保留了此类，但是在新增的 spark 组件体系中，有功能更加完善的视频播放器支持类。spark.primitives.VideoElement 类提供了基本控制和显示功能，但是没有外壳，spark.components.VideoPlayer 类是带完整外壳的视频播放器组件，spark.skins.default.VideoPlayerSkin 是 spark.components.VideoPlayer 类的默认皮肤，还有支持组件：VideoPlayerScrubBar（时间条），VideoPlayerVolumeBar（音量条），VideoPlayerVolume-BarMuteButton（静音按钮）。可以说功能非常完善了。

对于音频播放器，Flex 4 框架还没有提供现成的完整组件，但是 flash.media 包提供了对声音的基本控制。互联网上有封装好的开源音频播放器类库，接口类似于 Flex 视频播放器，可以拿来复用。
地址为： http://rojored.com/#simple-flex-audio-player

我们的“自定义播放器”就是要结合视频播放器和音频播放器的功能，可以同时播放视频和音频，自动切换显示区的有无。

详细功能列表：

1. 可以加载视频和音频文件，自动判断并播放。

2. 如果是视频，则有显示区，是音频，则无显示区。

3. 播放/暂停按钮，具有播放/暂停控制功能；停止按钮，具有停止并返回开头控制功能。

4. 时间条，可以显示播放进度；拖动控制播放进度。

5. 已播放时间、总时间显示。

6. 音量条，可以控制声音大小。切换视频和音频，音量不变。

7. 静音按钮，可以切换静音。静音时，音量条可以调节，可以设置音量，以便在不静音时生效。

8. 信息，可以显示播放内容的额外信息，如内容描述。

9. 播放视频时，单击显示区切换视频播放和暂停状态。

2. 新建“自定义播放器”组件库项目及文件、示例项目及文件

a. 新建 Flex 库项目，项目名：CustomPlayer。其它选项默认即可。构建后，Flash Builder 4 会自动在 bin 文件夹中生成 swc 文件，即编译打包的组件。

b. 在 src 文件夹中新建包，名称：cn.airia.fb4.customPlayer。Flash Builder 4 会自动创建相应文件夹。

c. 在包 cn.airia.fb4.customPlayer 中新建 MXML 组件 CustomPlayer，继承自 spark.components.Group。宽度 100%，高度不填，布局改为 spark.layouts.VerticalLayout。

d. 新建 Flex 项目，项目名：CustomPlayerSample，作为示例项目，默认新建CustomPlayer.mxml 文件作为主应用程序文件。示例项目需要使用组件，有多种方法，我们采用这种方法：“库路径” > “添加项目” > “CustomPlayer”。在发布时，可以将最终 CustomPlayer 组件的 swc 文件复制到 CustomPlayerSample 项目的 libs 文件夹中，并从库路径中删除 CustomPlayer 项目，这样就可以独立运行 CustomPlayerSample 了。

e. 在CustomPlayerSample项目的“属性” > “项目引用”里添加对 CustomPlayer 项目的引用。在某些时候有用，比如打

开CustomPlayerSample项目时也打开CustomPlayer 项目。

3. 编写示例程序类 CustomPlayerSample

示例程序用来使用和测试“自定义播放器”组件。可以简单处理，这里不详解了。

设计界面如下：（图六十）

图六十

播放器设计界面

上部是“自定义播放器”组件。各位读者因为还没编好“自定义播放器”，所以只会显示空白，没有播放控件。 中间是一段介绍文字。下部是播放列表，一个 List，选择项目后，“自定义播放器”会播放相应视音频。

我把示例程序 CustomPlayerSample 的主题设为了Wireframe，如果主题不同，组件在示例程序中的表现也不同，但是组件本身是一样的。

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/halo"
               xmlns:customPlayer="cn.airia.fb4.customPlayer.*"
               minWidth="800"
               minHeight="600"
               viewSourceURL="srcview/index.html">
    <!-- 这是 Flex 4 的新布局组件 -->
    <s:layout>
        <s:BasicLayout/>
    </s:layout>
    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;
            // 播放列表数据，可绑定
            [Bindable]
            private var playListDP:ArrayCollection = new ArrayCollection([
                {label: "10秒钟的视频", url: "videos/10秒.flv"},
                {label: "黑猩猩打空手道", url: "videos/空手道.flv"},
                {label: "电话来了铃声", url: "audios/dddd.mp3"}]);
            /**
             * 给播放器要播放文件的源路径和额外信息，播放
             */
            private function play(event:Event):void {
                var item:Object = (event.target as List).selectedItem;
                player.source = item.url;
                player.info = item.label;
                player.play();
            }
        ]]>
    </fx:Script>
    <!-- 这就是我们要编写的"自定义播放器"组件 -->
    <customPlayer:CustomPlayer id="player"
                               horizontalCenter="0"
                               width="450"
                               bottom="250"
                               top="20">
    </customPlayer:CustomPlayer>
    <!-- 文字介绍 -->
    <mx:Text y="360"
             width="450"
             horizontalCenter="0"
             color="#EA9FA4"
             text="上面是一个简单的"自定义播放器"组件，可用于会议系统、教学系统、娱乐系统等。选择播放下面列表里的
视频或音频："/>
    <!-- 播放列表，选择项目后播放对应视音频 -->
    <s:List id="playList"
            dataProvider="{playListDP}"
            labelField="label"
            y="400"
            width="450"
            height="100"
            horizontalCenter="0"
            selectionChanged="play(event);"/>
</s:Application>
```

code

FlexBuilder 4 快速入门 代码片段

4. 编写组件类 CustomPlayer

CustomPlayer 组合了官方视频类——VideoElement 类，以及开源音频类——rojored 的 Audio 类，并加上统一的控件控制两者，和额外的功能、改进，如点击视频切换播放暂停，静音后还能设置音量值等。

具体讲解可以参见代码注释，结合代码讲解更加直观，易于理解。

设计界面如下：

图六十一

CustomPlayer

在播放视频时会自动显示视频显示区。

代码如下: (代码片段包含了本例讲解、注释)

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 这是 Flex 4 的新容器组件，没有皮肤，性能更好 -->
<s:Group xmlns:fx="http://ns.adobe.com/mxml/2009"
         xmlns:s="library://ns.adobe.com/flex/spark"
         xmlns:mx="library://ns.adobe.com/flex/halo"
         xmlns:rojored="com.rojored.view.controls.*"
         width="100%"
         creationComplete="init();">

    <!-- 这是 Flex 4 的新布局组件，这里采用垂直布局 -->
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            /**
             * 标示播放类型的常量
             */
            public static const VIDEO:String = "video";
            public static const AUDIO:String = "audio";

            /**
             * 设置的音量
             * 不等同于播放对象的音量，在静音时不为0，仍然保持设置值
             */
            [Bindable]
            private var volume:Number;

            /**
             * 播放对象
             * 可选 VideoElement 和 Audio 两种类
             * 但是调用的接口方法类似，所以不限定类型
             */
```

```
        [Bindable]
        private var playObject:*;

        /**
         * 源属性
         * 直接和播放对象的源挂钩
         */
        [Bindable]
        public function get source():Object {
            return playObject.source;
        }

        public function set source(value:Object):void {
            // 设置源后，要设置播放对象，以便选择播放视频或音频
            setPlayable(value);
        }

        /**
         * 显示在控制栏的额外信息
         */
        private var _info:String;

        [Bindable]
        public function get info():String {
            return _info;
        }

        public function set info(value:String):void {
            _info = value;
        }

        /**
         * 初始化
         * 默认播放是视频，音量值 0.7
         */
        private function init():void {
            playObject = video;
            volume = 0.7;
        }

        /**
         * 播放
         * 因为视音频播放对象播放方法一样，所以可以不用判断类型
         */
        public function play():void {
            playObject.play();
        }

        /**
         * 暂停
         * 因为视音频播放对象暂停方法一样，所以可以不用判断类型
         */
        public function pause():void {
            playObject.pause();
        }

        /**
         * 切换播放和暂停状态
         * 如果已播放，则暂停，如果已暂停，则播放
```

```
         */
        public function togglePlayPause():void {
            playObject.playing ? pause() : play();
        }

        /**
         * 停止
         * 因为视音频播放对象停止方法一样，所以可以不用判断类型
         */
        public function stop():void {
            playObject.stop();
        }

        /**
         * 根据源来设置播放对象
         * 先判断类型，然后停止非此类型的播放器，切换到此类型的播放器
         * 如果是视频类型，显示视频容器，如果是音频类型，隐藏视频容器
         */
        private function setPlayable(source:Object):void {
            switch (judgeType(source)) {
                case VIDEO:
                    audio.source = null;
                    audio.stop();
                    // 显示视频容器
                    videoContainer.visible = true;
                    // 标记显示列表失效，这样程序会自动更新绘制显示列表，否则视频位置和大小会有问题
                    videoContainer.invalidateDisplayList();
                    playObject = video;
                    playObject.source = source;
                    break;
                case AUDIO:
                    video.source = null;
                    video.stop();
                    // 隐藏视频容器
                    videoContainer.visible = false;
                    playObject = audio;
                    playObject.source = source;
                    break;
            }
        }

        /**
         * 判断播放类型
         * 利用正则表达式判断文件名后缀，如果是.flv，则为视频，是.mp3，则为音频
         * 还可以添加其它文件类型的判断，现在暂时不考虑
         */
        private function judgeType(source:Object):String {
            if (new RegExp("(\.flv)$", "i").test(source.toString())) {
                return VIDEO;
            } else if (new RegExp("(\.mp3)$", "i").test(source.toString())) {
                return AUDIO;
            }
            return null;
        }

        /**
         * 格式化时间
         * 将
```

```
            * VideoElement 返回的是秒，Audio 返回的是毫秒，要处理好
            */
           private function format(value:Number):String {

               var seconds:Number;

               if (playObject is Audio) {
                   seconds = Math.floor(value / 1000);
               } else if (playObject is VideoElement) {
                   seconds = value;
               }

               var result:String = Math.floor(seconds % 60).toString();

               // 如果是秒数是 1 位数，则在前面添加 1 个 0
               if (result.length == 1) {
                   result = Math.floor(seconds / 60).toString() + ":0" + result;
               } else {
                   result = Math.floor(seconds / 60).toString() + ":" + result;
               }
               return result;
           }
       ]]>
   </fx:Script>

   <!-- 将播放对象的音量绑定为设置音量，如果静音了，则为0 -->
   <fx:Binding destination="video.volume"
               source="muteBtn.selected ? 0 : volume"/>
   <fx:Binding destination="audio.volume"
               source="muteBtn.selected ? 0 : volume"/>

   <!-- 音频播放对象，因为是非可视对象，所以要放在 <fx:Declarations> 标签内 -->
   <fx:Declarations>
       <rojored:Audio id="audio"/>
   </fx:Declarations>

   <!-- 视频播放对象，因为 VideoElement 不能接受 click 事件，所以用一个 Group 包装 -->
   <s:Group id="videoContainer"
            click="togglePlayPause();"
            width="100%"
            height="100%">

       <s:VideoElement id="video"
                       width="100%"
                       height="100%"
                       autoRewind="true"/>
       <!-- 停止后自动回复到开头 -->

   </s:Group>

   <!-- 播放时间条，将值和播放对象的播放头时间双向绑定，这样就可以同步了 -->
   <!-- 这里不采用 s:HSlider 组件，否则 value 的双向绑定会导致播放问题 -->
   <mx:HSlider id="timeSlider"
               width="100%"
               minimum="0"
               maximum="{playObject.totalTime}"
               value="@{playObject.playheadTime}"
               liveDragging="false"/>
```

```
    <s:Group width="100%">

        <!-- 播放暂停按钮，采用切换按钮，切换播放和暂停 -->
        <s:ToggleButton id="playPauseBtn"
                        label="{playObject.playing ? '暂停' : '播放'}"
                        width="54"
                        x="0"
                        selected="{playObject.playing}"
                        click="togglePlayPause();"/>

        <!-- 停止按钮，停止播放 -->
        <s:Button id="stopBtn"
                  label="停止"
                  click="stop();"
                  x="60"
                  width="52"/>

        <!-- 播放时间显示，包括当前播放头时间和总时间 -->
        <mx:Label id="time"
                  x="119"
                  width="76"
                  text="{format(playObject.playheadTime)}/{format(playObject.totalTime)}"/>

        <!-- 静音按钮，采用切换按钮，切换后会自动影响绑定它的播放对象音量值 -->
        <s:ToggleButton id="muteBtn"
                        label="静"
                        width="33"
                        x="218"/>

        <!-- 音量条，将值和设置的音量值双向绑定，而非播放对象的当前音量，因此不受静音影响 -->
        <!-- 因为 valueInterval 默认为1，不合要求，要改为0，也就是无值间隔 -->
        <s:HSlider id="volumeSlider"
                   width="86"
                   x="252"
                   y="5"
                   minimum="0"
                   maximum="1"
                   valueInterval="0"
                   value="@{volume}"
                   liveDragging="true"/>

        <!-- 信息标签，显示赋给的额外信息 -->
        <mx:Label id="infoLabel"
                  text="{info}"
                  x="346"
                  width="104"/>

    </s:Group>

</s:Group>
```

总结：

本节使用了官方组件和开源类库自定义播放器组件，满足自己的特殊需要。比如可以同时播放视频和音频、点击视频能暂停等。 MXML 组件开发利用 MXML 做界面布局，ActionScript 做逻辑运算，各得其所，方便高效。

开发时，经常要自定义组件，这时可以充分利用已有的组件，减少重复开发带来的浪费。所以，要多多关注全世界的类库，同时，也可以把自己的成果贡献给世界。

自定义组件开发 第三节　ActionScript组件开发

本节重点介绍了如何开发 Flex 可视组件。通过对 UIComponent 中各种与组件开发有关的方法的学习，我们将了解到 Flex 组件的基本结构，了解 Flex 组件开发中所蕴藏的秘密。最后本节以一个 ActionScript 组合组件的开发为例，向大家展示具体是如何实现自定义 Flex 可视组件的。

可视组件的基本结构

所有的 Flex 可视组件都是 UIComponent 类的子类。因此，可视组件将继承 UIComponent 中定义的方法、属性、样式、事件和特效。为了创建一个可视组件，我们必须实现类的构造函数。此外，我们还可以有选择的重写一个或多个下列 UIComponent 中定义的保护方法。

UIComponent 方法	说明
commitProperties()	提交组件属性的任何变化，或者确认属性的变化是同时发生的，或者确保属性值按一定顺序设置的。
createChildren()	创建组件的子组件。例如，在 ComboBox 组件里包含一个 TextInput 控件以及一个 Button 控件作为其子组件。
layoutChrome()	为容器类的子类定义容器周围的边框区域。
measure()	设置组件的默认大小及默认的最小尺寸。
updateDisplayList()	根据组件的属性和样式，设置组件子元素的大小和位置，绘制组件使用的皮肤或者图形元素。组件父容器的大小决定了组件的大小。

组件的使用者并不会直接调用这些方法。这些方法将作为初始化过程的一部分，在创建组件时被 Flex 所调用，或者在调用其他方法时被调用。

综上所述，Flex 中，一个可视组件的基本结构如下：

```
package com.custom.controls
{
public class VisualComponent extends UIComponent
{
    ////////////////////////////////////////
    // 构造函数
    ////////////////////////////////////////
    public function VisualComponent()
    {
    }

    ////////////////////////////////////////
    // 重写 createChildren 方法 [可选]
    ////////////////////////////////////////
    override protected function createChildren():void
    {
    }

    ////////////////////////////////////////
    // 重写 commitProperties 方法 [可选]
    ////////////////////////////////////////
    override protected function commitProperties():void
    {
    }

    ////////////////////////////////////////
    // 重写 measure 方法 [可选]
    ////////////////////////////////////////
    override protected function measure ():void
    {
    }

    ////////////////////////////////////////
    // 重写 layoutChrome 方法 [可选]
    ////////////////////////////////////////
    override protected function layoutChrome():void
    {
    }

    ////////////////////////////////////////
    // 重写 updateDisplayList 方法 [可选]
    ////////////////////////////////////////
    override protected function updateDisplayList(unscaledWidth:Number, unscaledHeight:Number
):void
    {
    }
} }
```

code

FlexBuilder 4 快速入门 代码片段

我们必须将 ActionScript 自定义组件定义在包中。包反映了应用程序目录结构中组件所在目录的位置。示例中的类，表明在应用程序根目录下的 com/custom/controls/ 目录中，以 ActionScript 文件的形式定义了 VisualComponent

组件，其文件名为 VisualComponent.as。 其中，包 com.custom.controls 对应目录结构 com/custom/controls/。

组件的类定义必须以 public 关键字为前缀。如果以 internal 为关键字，则该类对包外的类是不可见的，即我们无法在包外使用该类。虽然一个 ActionScript 文件中能定义多个 internal 类，但是能定义且只能定义一个 public 类。 任何 internal 类的定义都必须放在包定义的最后一个花括号之后。下面代码演示了在目录结构com/custon/controls/所含的文件 VisualComponent.as 中定义了一个 public 类和两个 internal 类。

```
package com.custom.controls
{
import mx.core.UIComponent;

public  class VisualComponent extends UIComponent
{
}
}


internal class SampleClass1
{
}

internal class SampleClass2
{
}
```

code

FlexBuilder 4 快速入门 代码片段

失效验证机制

在组件未被销毁期间，应用程序可能通过，改变组件的大小和位置、更改控制组件显示的某个属性、更改组件的某个样式或者皮肤属性，来更改组件。例如，我们可能更改组件中显示的文本的字体大小。因为字体大小的改变，组件的大小也可能需要改变，这需要 Flex 更新整个应用程序的布局。布局操作可能需要 Flex 调用组件的 commitProperties()、measure()、layoutChrome() 以及 updateDisplayList() 方法。

我们的应用程序能够以编程方式改变组件的字体大小，这比 Flex 更新整个应用程序的布局要快得多。因此，我们应该在确定了最终的字体大小之后再来更新整个应用程序的布局。

在另一个情景下，当我们设置组件的多个属性的时候，例如设置 Button 控件的 label 和 icon 属性，我们希望所有属性都设置完毕后，commitProperties()、measure() 以及 updateDisplayList() 方法只执行一次。我们不希望在设置 label 属性的时候这些方法执行一次，在设置 icon 属性的时候，这些方法又执行一次。

同样的，一些组件可能同时改变他们的字体大小。我们希望 Flex 调整布局操作以去除多余的处理，而不是在每个组件改变其字体大小后都重新布局一次。

Flex 使用失效验证机制来同步组件的更改。Flex 使用一系列方法来实现这种机制。通过调用这些方法，Flex 可以发出信号，表明组件发生了改变，需要调用组件的 commitProperties()、measure()、layoutChrome() 或者 updateDisplayList() 方法。

下面的表格描述了这些失效验证方法：

失效验证方法	说明
invalidateProperties()	标记组件以使组件的 commitProperties() 方法在下次屏幕刷新期间被调用。
invalidateSize()	标记组件以使组件的 measure() 方法在下次屏幕刷新期间被调用。
invalidateDisplayList()	标记组件以使组件的 layoutChrome() 及 updateDisplayList() 方法在下次屏幕刷新期间被调用。

当组件调用失效验证方法的时候，它向 Flex 发出信号表明组件必须被更新。当多个组件调用失效验证方法的时候，Flex 调整所有更新，使它们在下次屏幕更新期间一起发生。

通常情况下，组件使用者并不直接调用失效验证方法，而是在必要的时候，通过组件的 setter 方法或者组件的其他方法来调用。

基本结构详解

构造函数

如果一个 ActionScript 类的父类是 UIComponent，或者是UIComponent 的某个子类，那么该类应该定义一个 public 的构造函数，且该构造函数有如下特点：

- 没有返回类型的定义（方法后面没有 :void 之类的定义）。
- 声明为 public。
- 没有参数。
- 需要调用 super() 方法来调用父类的构造函数。

每个类只能有一个构造函数；ActionScript 不支持重载构造函数。

在构造函数中可以初始化类属性的值，例如，我们可以设置属性和样式的默认值，或者初始化数据结构，例如数组。

不要在构造函数中创建子组件；应该只在构造函数中初始化组件的属性。如果需要创建子组件，在 createChildren() 方法中创建他们。

createChildren() 方法

如果一个组件中创建了其他组件或者可视对象，那么该组件可以被称为组合组件。例如，Flex ComboBox 控件包含一个 TextInput 控件来定义 ComboBox 的文本区域，以及一个 Button 控件来定义 ComboBox 的箭头。组件通过实现 createChildren() 方法来创建子对象（例如其他组件）。

我们不会直接调用 createChildren() 方法，当我们调用 addChild()，将组件加入到其父对象之中的时候，Flex 会调用该方法。createChildren() 方法没有对用的失效方法，这意味着在我们将组件添加到其父对象之中后，我们不需要再次调用该方法（子组件作为组件的一部分，在组件未被销毁期间都是存在的，因此不需要多次创建）。

commitProperties() 方法

我们使用 commitProperties() 方法来协调对组件属性的更改，大多数情况下，我们协调的是影响到组件如何在屏幕上显示的属性。

当调用 invalidateProperties() 方法的时候，Flex 会安排一个对 commitProperties（ ）方法的调用。当我们使用 addChild() 方法将一个组件添加到一个容器中的时候，Flex 会自动调用 invalidateProperties() 方法。

对 commitProperties() 方法的调用在 measure() 方法之前，这使得我们可以在 commitProperties() 方法中预先设置 measure() 方法中可能用到的属性的值。

使用 commitProperties() 方法的主要优点在于：

- 协调对多个属性的更改，使其同时产生效果。

  例如，我们可能定义了多个属性来控制组件中显示的文本，例如控制组件中文本的对齐方式。文本或者文本对齐方式的变化都需要 Flex 更新组件的外观。然而，如果我们同时更改了文本和对齐方式，当刷新屏幕的时候，我们只想让 Flex 执行一次为了重设组件大小或位置而进行的运算。因此，我们使用 commitProperties() 方法，根据多个属性间的关系来计算任意所需的值。

- 协调对同一属性的多次更改。

  我们不希望每次更改同一属性的时候都不必要的执行一次复杂运算。例如，用户更改 Button 控件的 icon 属性来改变 Button 上显示的图片。根据 icon 的形状和大小来计算标签的位置可能是一个计算代价高昂的操作，这使得我们只想在有必要的时候才执行它。为了避免这种行为，我们使用 commitProperties() 方法来执行这些运算。当 Flex 刷新显示内容时候，它会调用 commitProperties() 方法。这意味着屏幕刷新期间，无论我们更改了多少次 icon 属性， Flex 只在刷新屏幕的时候执行一次上述操作。

measure() 方法

measure() 方法用来设置默认的组件大小，以像素为单位，我们也可以在这里设置组件默认的最小尺寸。

当我们调用 invalidateSize() 方法的时候，Flex 会安排一个对 measure() 方法的调用。在调用了 invalidateSize() 方法之后，measure() 方法将在下一次渲染界面的时候执行。当我们使用 addChild() 方法将组件添加到容器中的时候，Flex 会自动调用 invalidateSize() 方法。

当我们给组件设定高度和宽度的时候，Flex 不会调用 measure() 方法，即使我们显式的调用 invalidateSize() 方法。这是因为，Flex 只在 explicitWidth 属性或者 explicitHeight 属性为 NaN 的时候才调用 measure() 方法。

下面的示例中，因为我们显式的设置了 Button 控件的大小，Flex 不会调用 measure() 方法：

```
<mx:Button height="10" width="10"/>
```

FlexBuilder 4 快速入门 代码片段

在现有组件的子类中，只有当我们需要修改超类中设置组件大小的规则的时候，我们才实现 measure() 方法。因此，为了设置新的默认大小，或者为了在运行时执行运算来决定设置组件大小的规则，我们需要重写 measure() 方法。

我们通过在 measure() 方法中设定下列属性的值，来设置组件的默认大小：

属性	说明
measuredHeight measureWidth	指定组件默认的高度和宽度，以像素为单位。在 measure() 方法被调用之前，这些属性的值都为0。虽然我们可以保持这些属性值都为0，但是这样会使组件在默认情况下不可见。
measuredMinHeight measureMinWidth	指定组件默认的最小高度和最小宽度，以像素为单位。Flex 不能将组件的大小设置的比其指定的最小尺寸还小。

measure() 方法仅能够设置组件的默认大小。在 updateDisplayList() 方法中，组件的父容器将指定组件的真实大小，这可能与其默认尺寸不相同。

组件使用者在应用程序中可以通过下列方式来重写组件默认大小的设置：

- 设置 explicitHeight 和 exlicitWidth 属性
- 设置 width 和 height 属性
- 设置 percentWidth 和 percentHeight 属性

如何计算默认大小呢？一些 Flex 组件使用静态的大小。例如，无论TextArea 控件里面包含什么样的文本，它的默认大小都为100像素宽、44像素高。如果文本所占的空间比 TextArea 控件大，TextArea 控件上就会显示出滚动条来。

通常情况下，我们根据组件的特征或者传递给组件的信息来设置组件的大小。例如，Button 控件的 measure() 方法检查其 label 的文本、边界设置和字体特点来决定其默认大小。

layoutChrome() 方法

Container 类，以及一些 Container 类的子类，使用 layoutChrome() 方法来定义容器周围的边框区域。

当我们调用 invalidateDisplayList() 方法的时候，Flex 会安排一个对 layoutChrome() 方法的调用。在调用了 invalidateDisplayList() 方法之后，layoutChrome() 方法将在下一次渲染界面的时候执行。当我们使用 addChild() 方法将组件添加到容器中的时候，Flex 会自动调用 invalidateDisplayList() 方法。

通常情况下，我们使用 RectangularBorder 类来定义容器的边框区域。例如，我们可以创建一个 RectangularBorder 对象，然后在重写的 createChildren() 方法中将其作为子组件添加到组件中。

当我们创建 Container 类的子类的时候，我们可以使用 createChildren() 方法来创建组件的子内容；子内容是容器中容纳的组件。然后我们可以在 updateDisplayList() 方法中设置子内容的位置。

通常情况下我们使用 layoutChrome() 方法定义容器的边框区域及设置边框区域的位置，还可以添加其他任何希望出现在边框区域的元素。例如，Panel 容器使用 layoutChrome() 方法来定义其标题区域，其中包括标题的文本和关闭按钮。

将容器的内容区域和边框区域分开处理的主要原因在于为了处理 Container.autoLayout 属性被设置为 false 的情况。当 autoLayout 属性被设置为 true，无论何时，只要容器里面某个子内容的大小或者位置发生了改变，Flex 都会重新计算容器和所有子内容的大小及位置。autoLayout 属性的默认值为 true。当 autoLayout 属性被设为 false，容器及其子内容的大小或位置只会在添加子内容或者移除子内容的时候计算一次。然而，两种情况下，Flex 都会执行 layoutChrome() 方法。因此，即使 autoLayout 属性被设为 false，容器依然可以刷新其边框区域。

updateDisplayList() 方法

组件的 updateDisplayList() 方法根据前面指定的属性和样式来设置组件的子对象的大小及位置，绘制组件中使用的任何皮肤及图形元素。组件的父容器负责指定组件自身的大小和位置（例如，在容器类的 updateDisplayList() 方法中设置其子内容的大小和位置）。

直到其 updateDisplayList() 方法被调用，组件是不会显示在屏幕上的。当我们调用 invalidateDisplayList() 方法的时候，Flex 会安排一个对 updateDisplayList() 方法的调用。在调用了 invalidateDisplayList() 方法之后，updateDisplayList() 方法将在下一次渲染界面的时候执行。当我们使用 addChild() 方法将组件添加到容器中的时候，Flex 会自动调用 invalidateDisplayList() 方法。

updateDisplayList() 方法的主要用途如下：

- 设置组件中显示出来的元素的大小和位置。

  许多组件由一个或多个子组件构成，或者有一些属性用来控制组件中信息的显示。例如，Button 控件让我们能够指定一个可选的 icon，以及使用 labelPlacement 属性来指定按钮文本相对图标的显示位置。Button.updateDisplayList() 方法使用 icon 及 labelPlacement 属性来控制按钮的显示。 对于容纳了子控件的容器，updateDisplayList() 方法控制如何设置其子内容的位置。例如，HBox 容器的 updateDisplayList() 方法将其子内容按从左到右的顺序排为单独的一行；VBox 容器的 updateDisplayList() 方法将其子内容按从上到下的顺序排为单独的一列。 在 updateDisplayList() 方法中设置子对象的大小的时候，我们需要使用 setActualSize() 方法，而不是通过设置控制大小的属性，例如高度和宽度。设置子对象的位置的时候，通过 move() 方法，而不是设置 x 和 y 属性。

- 绘制组件中必须的可视元素。

  组件支持许多类型的可视元素，例如皮肤、样式和边框。在 updateDisplayList() 方法中，我们可以添加这些可视元素，使用 Flash 的绘图 API 绘图，执行对组件外观的额外控制。

updateDisplayList() 方法有如下签名：

```
protected function updateDisplayList(unscaledWidth:Number,unscaledHeight:Number):void
```

FlexBuilder 4 快速入门 代码片段

unscaledWidth 在组件自身的坐标系中，以像素为单位，指定了组件的宽度，与组件的 scaleX 属性无关。这是由组件的父容器决定的宽度。

unscaledHeight 在组件自身的坐标系中，以像素为单位，指定了组件的高度，与组件的 scaleY 属性无关。这是由组件的父容器决定的高度。

缩放发生在 Flash Player 或者 AIR 中，在 updateDisplayList() 方法执行之后。例如，一个组件的 unscaledWidth 为 100，它的 scaleY 为2.0，则在 Flash Player 或者 AIR 中，它显示出来的宽度为200.

每一个 Flex 可视组件都是 Flash Sprite 类的子类，因此它们继承了 Sprite.graphics 属性。Sprite.graphics 属性指定了一个 Graphics 对象。我们可以使用这个对象在组件中绘制矢量图形。

初始化生命周期

组件初始化生命周期描述了我们从一个组件类创建组件对象的时候产生的一系列步骤。作为初始化生命周期的一部分，Flex 自动调用组件的方法、派发事件以及使组件可见。

下面的示例用 ActionScript 创建了一个 Button 控件并将其添加到一个容器中：

```
// 创建 Box 容器。
var boxContainer:Box = new Box();
// 配置 Box 容器。

// 创建 Button 控件。
var b:Button = new Button();
// 配置 Button 控件。
b.label = "Submit";
...
// 将 Button 控件添加到 Box 容器中。
boxContainer.addChild(b);
```

code

FlexBuilder 4 快速入门 代码片段

下面的步骤显示了当我们执行代码创建 Button 控件并将其添加到容器中时发生了什么：

1. 调用组件的构造函数。

```
// 创建 Button 控件。
var b:Button = new Button();
```

code

FlexBuilder 4 快速入门 代码片段

2. 通过设置组件的属性来配置组件。

```
// 配置 Button 控件。
b.label = "Submit";
```

code

FlexBuilder 4 快速入门 代码片段

3. 组件的 setter 方法可能调用 invalidateProperties()、invalidateSize() 或者 invalidateDisplayList() 方法。

我们调用 addChild() 方法将组件添加到容器中去。

```
// 将 Button 控件添加到 Box 容器中。
boxContainer.addChild(b);
```

code

FlexBuilder 4 快速入门 代码片段

1. Flex 将执行以下操作：

2. 设置组件的 parent 属性来引用其父容器。
3. 计算组件的样式设置。
4. 在组件上派发 preinitialize 事件。
5. 调用组件的 createChildren() 方法。
6. 调用 invalidateProperties()、invalidateSize() 和 invalidateDisplayList() 方法来引起随后在下次渲染界面期间对 commitProperties()、measure() 或者 updateDisplayList() 方法的调用。

   唯一例外的是当用户设置了组件的高度和宽度的时候，Flex 不会调用 measure() 方法。

7. 在组件上派发 initialize 事件。这个时候，该组件所有的子对象都已经被初始化，但是组件本身并没有因为布局而被设置大小或者处理。我们可以在组件被计算进布局之前使用该事件执行额外的处理。
8. 在父容器上派发 childAdd 事件。
9. 在父容器上派发 initialize 事件。
10. 在下次渲染界面期间，Flex 将执行以下操作：
    a.调用组件的 commitProperties() 方法。
    b.调用组件的 measure() 方法。
    c.调用组件的 layoutChrome() 方法。
    d.调用组件的 updateDisplayList() 方法。
    e.在组件上派发 updateComplete 事件。
11. 如果 commitProperties()、measure() 或者 updateDisplayList() 方法调用了 invalidateProperties()、invalidateSize 者 invalidateDisplayList() 方法，Flex 将派发额外的渲染事件。
12. 最后一个渲染事件发生了之后，Flex 将执行以下操作：
    a.通过设置 visible 属性为 true 使组件变得可见。
    b.在组件上派发 creationComplete 事件。组件将因为布局的需要而被设置大小或被处理。该事件在组件创建的时候只会派发一次。
    c.在组件上派发 updateComplete 事件。

大多数配置组件的工作发生在我们使用 addChild() 方法将组件添加到容器中的时候。这是因为在我们将组件添加到容器中之前，Flex 无法决定组件的大小，决定其继承来的样式属性，或者将其绘制在屏幕上。

我们也可以在 MXML 中定义我们的应用程序：

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

code

FlexBuilder 4 快速入门 代码片段

在 MXML 中创建组件时所产生的一系列步骤和上述在 ActionScript 中创建组件时产生的步骤是相同的。

我们可以使用 removeChild() 方法将组件从容器中移除。如果不存在任何对组件的引用，组件最终将被 Flash Player 或者 AIR 的垃圾回收机制从内存中删除。

创建组件的步骤

当我们实现可视组件的时候，我们重写组件的相关方法，定义新的属性，派发新的事件，或者执行应用程序所需

的任何其他自定
义组件的操作。

为了实现我们的组件，可以遵循下列步骤：

1. 如果有必要，为组件创建任何所需的皮肤。
2. 创建一个 ActionScript 文件。
   a.扩展一个基类，例如 UIComponent 类或者其他 UIComponent 的子类。
   b.指定用户可以通过 MXML 标签属性设置的属性。
   c.嵌入任何所需的图形及皮肤文件。
   d.实现构造函数。
   e.实现 UICommponent.createChildren() 方法。
   f.实现 UIComponent.commitProperties() 方法。
   g.实现 UIComponent.measure() 方法。
   h.实现 UIComponent.layoutChrome() 方法。
   i.实现 UIComponent.updateDisplayList() 方法。
   j.添加属性、方法、样式、事件和元数据。
3. 将组件作为 ActionScript 文件或者 SWC 文件部署。

我们不需要重写所有的组件方法来定义一个新的组件。我们只重写实现组件功能所需的方法。如果我们创建了一个继承自现有组件，例如 Button 控件或者 VBox 容器，的子类，为了实现新功能，我们必须实现那些必须重写的方法。

例如，我们可以实现一个使用新机制来定义其默认大小的自定义 Button 控件。这种情况下，我们只需要重写 measure() 方法就可以了。

或者我们可能实现一个 VBox 容器的子类。该类使用来自 VBox 的所有设置子对象大小的逻辑，但是容器将按从下到上的顺序来排列子内容，而不是从上至下。这种情况下，我们只需要重写 updateDisplayList() 方法就可以了。

可视组件需实现的接口

Flex 通过接口将组件的基本功能分为可一点点实现的不相关联的元素。例如，为了让组件能够接收焦点，我们必须实现 IFocusable 接口；为了让组件能够参与布局过程，组件必须实现 ILayoutClient 接口。

为了简化接口的使用，UIComponent 类实现了下表中除 IFocusManagerComponent 以及 IToolTipManagerClient 的所有接口。然而，许多 UIComponent 类的子类实现了 IFocusManagerComponent 和 IToolTipManagerClient 接口。

因此，如果我们创建了一个 UIComponent 的子类，或者创建了一个继承自 UIComponent 的类的子类，我们不需要实现这些接口。但是，如果我们创建了一个不是继承自 UIComponent 的组件，并且我们想在 Flex 中使用它，我们可能需要实现一个或多个下列接口。

> 注意: 对于 Flex，Adobe 推荐所有的组件都扩展自 UIComponent 类或者 UIComponent 的子类。

接口	用途
IChildList	表明容器中子内容的数量。
IDeferredInstantiationUIComponent	表明组件或者对象的实例化将被延迟。
IFlexDisplayObject	为皮肤元素指定接口。
IFocusManagerComponent	表明组件或对象是可接受焦点的，这意味着组件可以从 FocusManager 收到焦点。 UIComponent 类并没有实现该接口，因为它的一些扩展类不需要收到焦点。
IInvalidating	表明组件或者对象可以利用验证失效机制执行延迟的属性提交、测量大小以及绘图或者布局操作。
ILayoutManagerClient	表明组件或者对象可以参与到 LayoutManager 的一系列提交、测量和更新操作中去。
IPropertyChangeNotifier	表明组件支持特殊形式的事件传播机制。
IRepeaterClient	表明组件或对象可以与 Repeater 类一起使用。
IStyleClient	表明组件能够从其他对象继承样式，并且支持 setStyle() 和 getStyle() 方法。
IToolTipManagerClient	表明组件或由一个 toolTip 属性，因此它会被 ToolTipManager 所监控。
IUIComponent	定义我们必须实现一系列基本的 API，以使组件能够称为布局容器或列表的子元素。
IValidatorListener	表明组件可以监听验证事件，因此可以显示出一个验证状态，例如红色的边框及错误提示。

一个组合组件的具体实现

以下代码片段包含了讲解与注释

```
package com.custom.controls
{
    ////////////////////////////////////
    //
    // 导入所有必须的类。
    //
    ////////////////////////////////////
    import mx.core.UIComponent;
    import mx.controls.Button;
    import mx.controls.TextArea;
    import flash.events.Event;
```

```
import flash.text.TextLineMetrics;

//////////////////////////////////////
//
// 当子 TextArea 组件的 text 发生改变的时
// 候，ModalText 派发出一个 change 事件。
//
//////////////////////////////////////
[Event(name="change", type="flash.events.Event")]

//////////////////////////////////////
//
// 当设置 ModalText 的 text 属性的值的时候，
// 派发一个 textChanged 事件。
//
//////////////////////////////////////
[Event(name="textChanged", type="flash.events.Event")]

//////////////////////////////////////
//
// 当设置 ModalText 的 textPlacement 属性
// 的值的时候，派发一个 placementChanged
// 事件。
//
//////////////////////////////////////
[Event(name="placementChanged", type="flash.events.Event")]

/*** a) 扩展 UIComponent。 ***/
public class ModalText extends UIComponent
{

    /*** b) 实现构造函数。 ***/
    public function ModalText()
    {
        super();
    }


    /*** c) 为两个子组件定义变量。 ***/
    //////////////////////////////////////
     //
    // 为子组件声明变量。
     //
     //////////////////////////////////////
    private var text_mc:TextArea;
    private var mode_mc:Button;


    /*** d) 实现 createChildren() 方法。 ***/
```

```
        ////////////////////////////////////////
         //
        // 在创建子组件之前检查它们是否存在，这样在
         // 组件的子类里可以创建不同的子组件来代替它
         // 父类中定义的子组件。
         //
         ////////////////////////////////////////
        override protected function createChildren():void
        {
            super.createChildren();

            ////////////////////////////////////////
            //
            // 创建并初始化 TextArea 控件。
              //
            ////////////////////////////////////////
            if (!text_mc)
            {
                text_mc = new TextArea();
                text_mc.explicitWidth = 80;
                text_mc.editable = false;
                text_mc.text= _text;
                text_mc.addEventListener("change", handleChangeEvent);
                addChild(text_mc);
            }

            ////////////////////////////////////////
            //
            // 创建并初始化 Button 控件。
              //
            ////////////////////////////////////////
            if (!mode_mc)
            {    mode_mc = new Button();
                mode_mc.label = "Toggle Editing Mode";
                mode_mc.addEventListener("click", handleClickEvent);
                addChild(mode_mc);
            }
        }


        /*** e) 实现 commitProperties() 方法。 ***/
        override protected function commitProperties():void
        {
            super.commitProperties();

            if (bTextChanged) {
                bTextChanged = false;
                text_mc.text = _text;
```

```
            invalidateDisplayList();
        }
    }


    /*** f) 实现 measure() 方法。 ***/
     ////////////////////////////////////////
     //
    // 默认宽度等于文本的宽度加上按钮的宽度。
     // 高度由按钮指定。
     //
    ////////////////////////////////////////
    override protected function measure():void
    {
        super.measure();

         ////////////////////////////////////////////
         //
        // 既然 Button 控件使用了皮肤，我们就可以获取
          // 已计算好的 Button 控件的大小。
          //
        ////////////////////////////////////////////
        var buttonWidth:Number = mode_mc.getExplicitOrMeasuredWidth();
        var buttonHeight:Number = mode_mc.getExplicitOrMeasuredHeight();

        ////////////////////////////////////////////
         //
        // 默认的和最小的宽度等于 TextArea 控件的
          // measuredWidth 加上 Button 控件的
          // measuredWidth。
          //
        ////////////////////////////////////////////
        measuredWidth = measuredMinWidth =
            text_mc.measuredWidth + buttonWidth;

        ////////////////////////////////////////////
         //
        // 默认的和最小的高度等于 TextArea 控件的高度
          // 和 Button 控件的高度中较大的那个再加上文本
          // 周围的边框所产生的10像素。
          //
        ////////////////////////////////////////////
        measuredHeight = measuredMinHeight =
            Math.max(mode_mc.measuredHeight,buttonHeight) + 10;
    }


    /*** g)实现 updateDisplayList() 方法。 ***/
    ////////////////////////////////////////
```

```
        //
        // 根据 Button 控件的 label 文本及预留的10
        // 像素边框区域来设置 其大小。
         // 根据余下的组件区域设置 TextArea 的大小。
         // 根据 textPlacement 属性设置子组件的位置。
         ////////////////////////////////////////
        override protected function
updateDisplayList(unscaledWidth:Number,unscaledHeight:Number):void
        {
            super.updateDisplayList(unscaledWidth, unscaledHeight);
             ////////////////////////////////////////
            //
            // 左边和右边的边框各需要1像素，还需要在左边
              // 和右边各留3像素的空白。
              //
            ////////////////////////////////////////
            var usableWidth:Number = unscaledWidth - 8;

                        ////////////////////////////////////////
            //
            // 顶部和底部的边框各需要1像素，还需要在顶部
              // 和底部各留3像素的空白。
              //
            ////////////////////////////////////////
            var usableHeight:Number = unscaledHeight - 8;
            ////////////////////////////////////////
            //
            // 根据 Button 控件的文本计算其大小。
              //
            ////////////////////////////////////////
            var lineMetrics:TextLineMetrics = measureText(mode_mc.label);
            ////////////////////////////////////////
            //
            // 在文本周围添加10像素的边框区域。
              //
            ////////////////////////////////////////
            var buttonWidth:Number = lineMetrics.width + 10;
            var buttonHeight:Number = lineMetrics.height + 10;
            mode_mc.setActualSize(buttonWidth, buttonHeight);

            ////////////////////////////////////////
            //
            // 计算文本的大小。在 Button 和 TextArea
            // 之间设置5像素的间隔。
              //
            ////////////////////////////////////////
            var textWidth:Number = usableWidth - buttonWidth - 5;
            var textHeight:Number = usableHeight;
```

```
            text_mc.setActualSize(textWidth, textHeight);

            ////////////////////////////////////////
            //
            // 根据 textPlacement 属性设置控件的位置。
              //
            ////////////////////////////////////////
            if (textPlacement == "left")
            {
                text_mc.move(4, 4);
                mode_mc.move(4 + textWidth + 5, 4);
            }
            else
            {
                mode_mc.move(4, 4);
                text_mc.move(4 + buttonWidth + 5, 4);
            }

            ////////////////////////////////////////
            //
            // 在子组件周围绘制简单的边框。
              //
            ////////////////////////////////////////
            graphics.lineStyle(1, 0x000000, 1.0);
            graphics.drawRect(0, 0, unscaledWidth, unscaledHeight);
        }


        /*** h) 添加方法、属性和元数据。 ***/
        ////////////////////////////////////////
         //
        // 一般情况下，为属性指定一个私有的持有变量。
         //
        ////////////////////////////////////////
        private var _textPlacement:String = "left";

        ////////////////////////////////////////
         //
        // 为 textPlacement 属性创建一对 getter/setter。
         //
        ////////////////////////////////////////
        public function set textPlacement(p:String):void
        {
            _textPlacement = p;
            invalidateDisplayList();
            dispatchEvent(new Event("placementChanged"));
        }

        ////////////////////////////////////////
```

```
        //
        // textPlacement 属性支持数据绑定。
        //
        ////////////////////////////////////////
        [Bindable(event="placementChanged")]
        public function get textPlacement():String
        {
            return _textPlacement;
        }

        private var _text:String = "ModalText";
        private var bTextChanged:Boolean = false;

        ////////////////////////////////////////
        //
        // 为 text 属性创建一对 getter/setter。
        //
        ////////////////////////////////////////
        public function set text(t:String):void
        {
            _text = t;
            bTextChanged = true;
            invalidateProperties();
            dispatchEvent(new Event("textChanged"));
        }

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

        ////////////////////////////////////////
                //
        // 处理子组件派发的事件。
        //
        ////////////////////////////////////////
        private function handleChangeEvent(eventObj:Event):void
        {
               dispatchEvent(new Event("change"));
        }

        ////////////////////////////////////////
        //
        // 处理子组件派发的事件。
        //
        ////////////////////////////////////////
        private function handleClickEvent(eventObj:Event):void
```

```
        {
                text_mc.editable = !text_mc.editable;
        }
    }
}
```

code

FlexBuilder 4 快速入门 代码片段

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application
        xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        xmlns:mx="library://ns.adobe.com/flex/halo"
        xmlns:controls="com.custom.controls.*"
        width="600"
        height="480">
        <s:Panel title="自定义可视化组件演示"
                width="80%"
                height="80%"
                horizontalCenter="0"
                verticalCenter="0">
                <controls:ModalText/>
        </s:Panel>
</s:Application>
```

code

FlexBuilder 4 快速入门 代码片段

总结

本节详细介绍了实现 ActionScript 可视组件的种种细节。一般情况下，我们通过继承 UIComponent 或者其子类来创建自定义的可视组件。我们需要重写特定的方法来实现我们所需的功能或逻辑。我们需要根据应用程序的需求及组件的初始化生命周期来确定组件最终该如何实现。此外我们还可以通过直接实现一些既定接口，而不是继承现有的类来创建能够在 Flex 中使用的可视组件。最后，通过创建 ModalText 组合组件，本节向大家展示了具体是如何实现自定义 ActionScript 可视组件的。

本节大部分内容都翻译自 Adobe 官方文档 ActionScript Custom Components。因作者水平有限，因此有不当的地方，望大家能发邮件告知，以互相交流。此外，所有内容应以官方文档为准。

思考

- UIComponent 中失效验证机制是如何实现的？
- UIComponent 中 LayoutManager 是如何工作的？
- 使用 MXML 创建的组件和使用 ActionScript 创建的组件，它们占用的内存都一样多吗？

第六章 ：与服务端通信

郑会宾	6.1 通过Http Service与服务端通信
唐凡	6.2 通过Web Service与服务端通信
郑会宾	6.3 通过Remoting与服务端通信
郑会宾	6.4 与Flash Media Server交互

与服务端通信 第一节 通过Http Service与服务端通信

在此，我们将利用 Flash Builder 4 带给我们的新特性——“数据/服务/控件绑定” ，通过 Http Service 来实现客户端与服务端之间的通信。

学习目标

先来看看我们最终要实现什么：

在文本框输入字符串“Sakura Momoko” ，点击“send”按钮。客户端通过 Http Service 将字符串“Sakura Momoko”发送到服务端，服务端收到该字符串后返回一条欢迎信息。（图六十二）

图六十二

服务端返回欢迎信息

准备工作

工欲善其事必先利其器，本节使用 Java Servelt 实现服务端的 Http Service，因此我们要准备相关的 Java 开发工具，以及 Servlet 容器同时，希望本节的读者对 Java Servlet 有一定的了解。

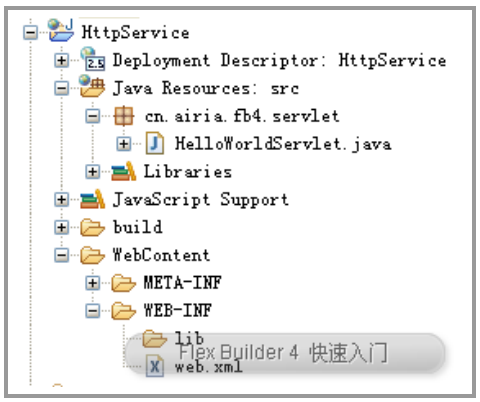
工具下载：
Java 开发工具：Eclipse 3.4
Servlet 容器 ：Tomcat 6.0

实现步骤

接下去我们就一步步的来实现这一应用。

服务端：

1、首先使用 Eclipse 3.4 建立一个 Dynamic Web Project 来实现一个简单的 servlet ，工程结构如下图所示。（六十三）



图六十三

Dynamic Web Project

2、编写 servelt 实现类“HelloWorldServlet.java”。

```
package cn.airia.fb4.servlet;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class HelloWorldServlet extends HttpServlet {

        private static final long serialVersionUID = -525713750766666340OL;

        @Override

        protected void doPost(HttpServletRequest req, HttpServletResponse resp)

                        throws ServletException, IOException {

                String name = req.getParameter("name");

                PrintWriter pw = resp.getWriter();

                pw.write("hello "+name+",welcome to Flash Builder's world");

                pw.close();

        }

}
```

code

FlexBuilder 4 快速入门 代码片段

3、配置“web.xml”文件。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
id="WebApp_ID" version="2.5">
  <display-name>HttpService</display-name>
  <servlet>
        <servlet-name>helloWorld</servlet-name>
        <servlet-class>cn.airia.fb4.servlet.HelloWorldServlet</servlet-class>
  </servlet>
  <servlet-mapping>
        <servlet-name>helloWorld</servlet-name>
        <url-pattern>/hello.do</url-pattern>
  </servlet-mapping>
</web-app>
```

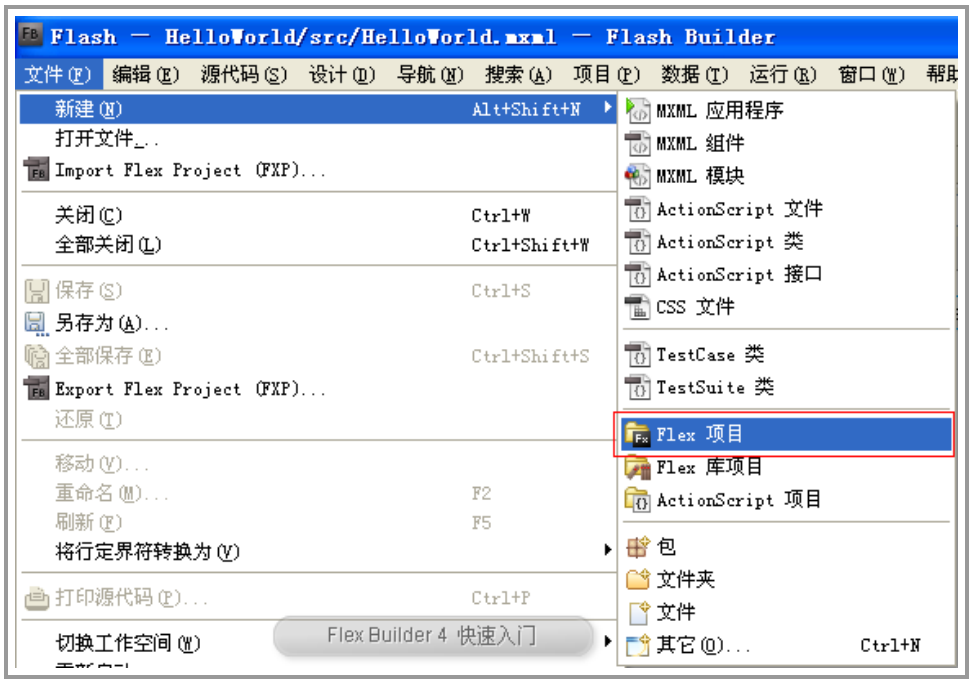
code

FlexBuilder 4 快速入门 代码片段

4、启动 web 容器 ，这里使用的是 Tomcat 6.0，在浏览器地址栏输入“http://localhost:8080/HttpService/hello.do” ，若浏览器显示“hello null,welcome to Flash Builder's world”则表示服务端工作完成。

客户端：

1、新建一个 Flex 项目。（图六十四）



图六十四

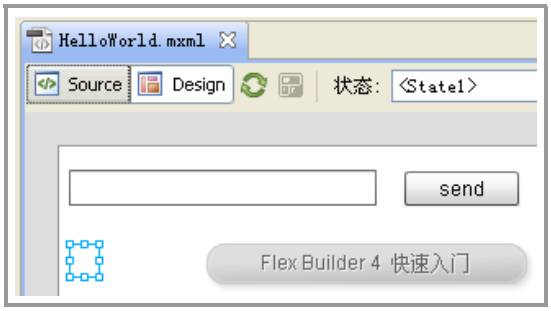
新建Flex项目

2、为项目起名“HelloWorld”后按“完成”。（图六十五）

图六十五

新建Flex项目

3、在“HelloWorld.mxml”的“Design”视图中拖拉控件，绘制界面。这里用到的控件有 TextInput、Button、以及 Label。（图六十六）



图六十六

设计界面

4、接下来几步就是本文的核心内容了，实现数据服务绑定的设置都在这里，是 Flash Builder 4 中特有的新功能，请同学们注意观看。点击菜单栏“数据”项，在下拉菜单中选择“连接Http...”（即 “连接Httpservice”）。（图六十七）

图六十七

连接Http

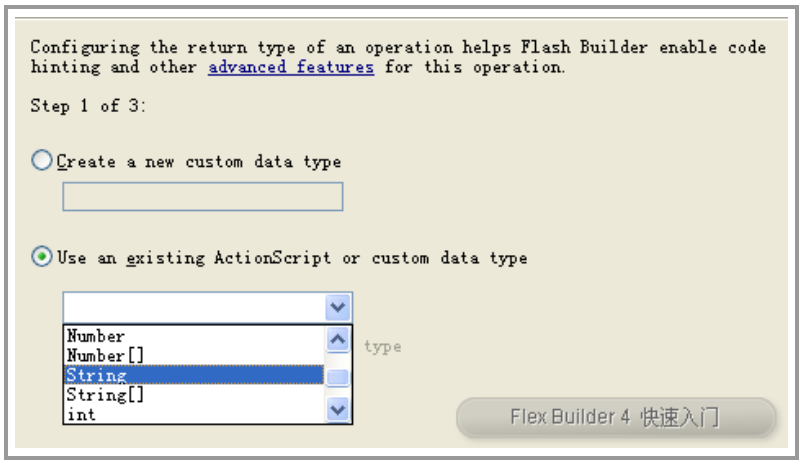
5、在弹出窗口中进行如下设置。（图六十八）

- 为 servlet 起一个操作名称这里为“sayHello”
- 在URL栏输入之前的 servlet 访问地址
- 方式采用“POST”
- 添加一个类型为“String”名称为“name”的传入参数
- 设置包名，这里设置为“cn.airia.fb4.services.helloservice”
- 点击“完成”

图六十八
配置HTTP服务

6、在 数据/服务”视图中操作的配置返回类型。

这里我们选择第二项 已经存在的 ActionScrip数据类型或者是之前已经自定义的类型”，选择“String类型。（图七十）



图七十
选择 String 类型

7、回到界面的“Design”视图，选中“Button”组件，在“属性”面板中对其进行设置，点击 选择单击时“生成服务调用”。（图七十一）

图七十一

生成服务调用

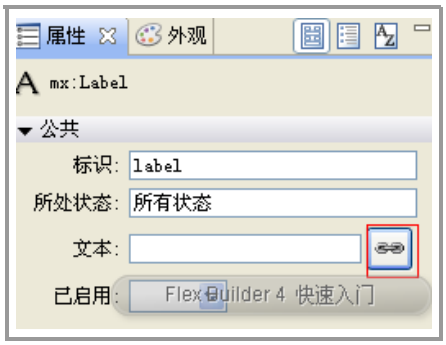
在弹出窗口中进行如下设置后“确定“。

8、此时界面跳转至“Source”视图，函数的参数“name”呈高亮显示。

我们将这个“name”设置为 TextInput 控件的内容（为 TextInput 控件设置一个标示，这里起名为“textInputName”）。

```
protected function button_clickHandler(event:MouseEvent):void
{
    sayHelloResult.token = helloService.sayHello(textInputName.text);
}
```

9、接下去将返回的结果与 Label 控件进行绑定，选中 Label 后，在“属性”面板中点击“超链接”图标。（图七十四）



图七十四

文本超链接

在弹出窗体中进行如下设置后点击“确定”按钮。（图七十五）

图七十五

绑定到数据

10、编码结束（其实我们基本没有手动编写代码噢 :) ），运行我们的程序看看效果吧

总结

本节教会了大家使用 Flash Builder 4 中新的“数据服务与组件绑定”解决方案中的 Http Service 方式来进行快速开发，同学们可以发现这种方式几乎不必编写任何代码，减轻了程序员的工作，同时也提高了开发效率，非常适合新人上手。但是这里我同样要指出的是，这种傻瓜式的开发方式也会使自动生成代码过于冗余，对今后的代码维护造成不便，组件与服务的直接绑定也会造成项目代码层次结构不清诸多影响。

思考

- 使用 GET 方式传递数据，这个项目应该做如何变更?
- 如何传递和处理复杂数据类型?

与服务端通信 第二节　通过Web Service与服务端通信

本节将为大家介绍如何用Flash Builder4调用WebService获得服务端信息。WebService在网络中运行十分广泛，我们可以获得天气预报、证券信息等等的各类信息。这里为了方便各位初学者理解，就由我们自己来编写一个简单的WebService来调用。

学习目标

在Flash Builder4中输入一个字符串，点击按钮调用WebService，返回：“XXX，Hello World！”

WebService端

首先我们要编写一个WebService来调用，本例中使用VisualStudio2008来编写一个WebService。

打开VS2008，新建一个项目，选择ASP.NET Web服务应用程序。选择好路径后确定。（图七十七）

图七十七

常用的非可视组件

写一个简单的WebService方法，接收一个string类型的字符串参数str，返回“str+'Hello World！'”。

方法如下：

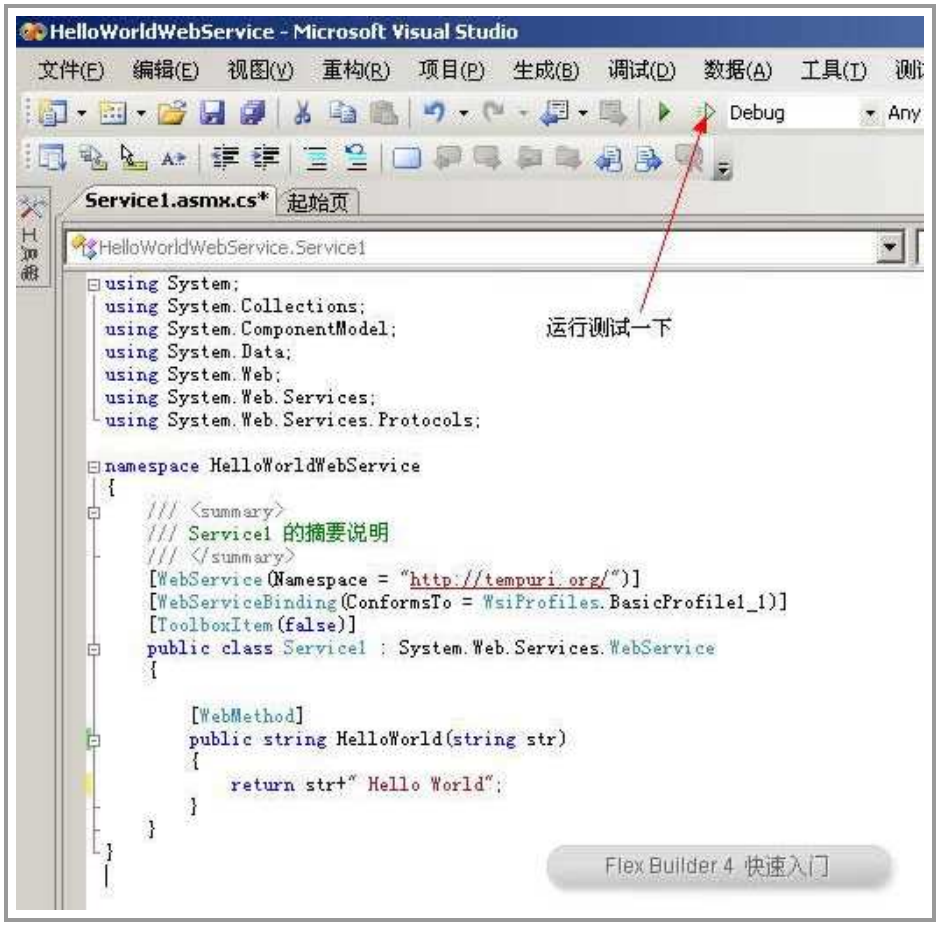
```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

namespace HelloWorldWebService
{
    /// <summary>
    /// Service1 的摘要说明
    /// </summary>
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [ToolboxItem(false)]
    public class Service1 : System.Web.Services.WebService
    {

        [WebMethod]
        public string HelloWorld(string str)
        {
            return str+"Hello World";
        }
    }
}
```

Flex Builder 4 快速入门

图七十八

WebService方法

测试一下：（图七十九）

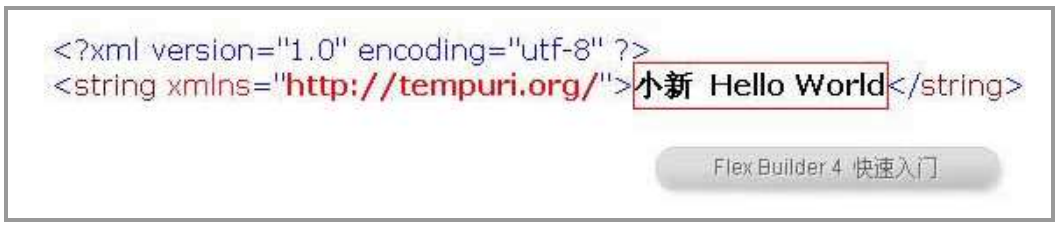


图七十九（组）

WebService

图七十九（组）

WebService

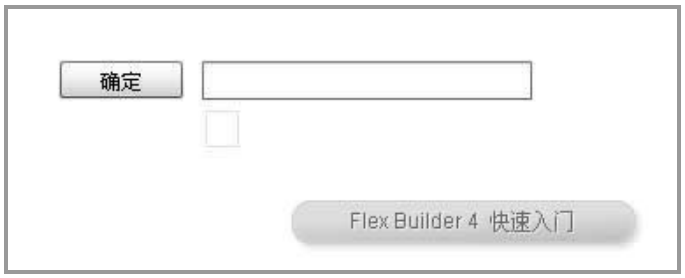


可以看到已经测试成功，运行以后出现了HelloWorld方法，输入“小新”，返回了“小新 Hello World”。 至此WebService编写完成。

Flex端

接下来我们用Flash Builder4编写Flex端。

我们只需要按钮、输入框和一个文本框就可以了，布局如下：（图八十）



图八十

WebService FLEX断布局

接下来是重点，也是你看本章的目的所在。

选择菜单里的“数据--连接WebService”：（图八十一）

图八十一

连接WebService

在弹出的对话框中，服务器名称中输入HelloWorld，这个可以自定义，下方会自定生成包名。

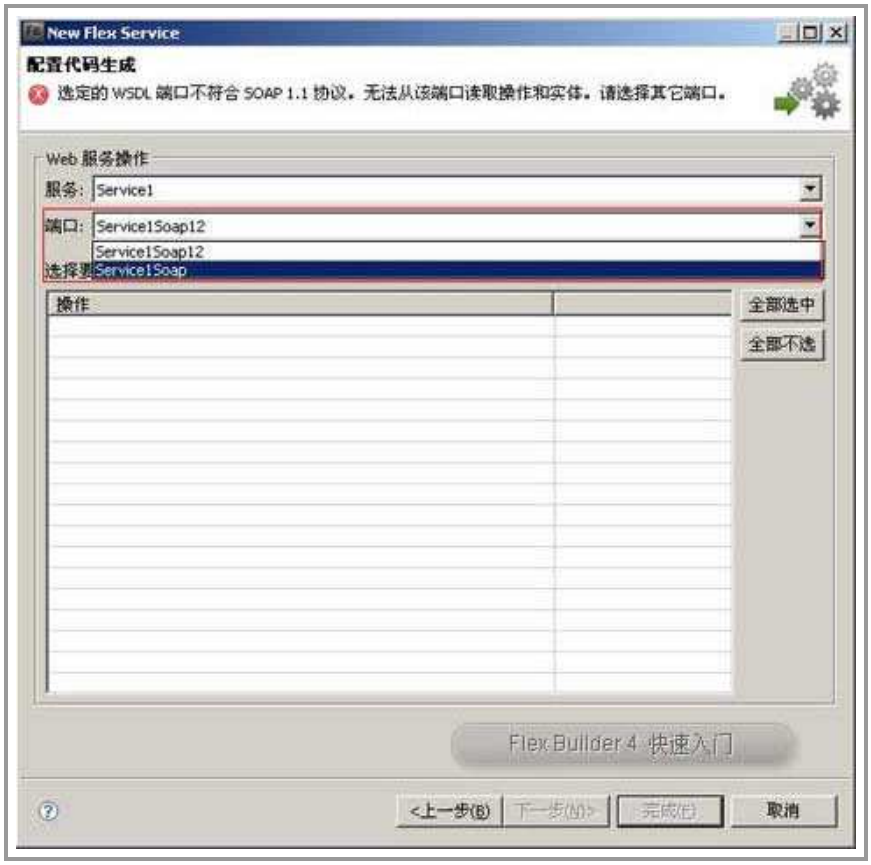
在“WSDL URL：” 中输入WebService的地址，然后在后面加上“?wsdl”以获得WebService的XML格式。（注意不要遗漏）

然后点击“下一步”：（图八十二）

图八十二

获得WebService的XML格式

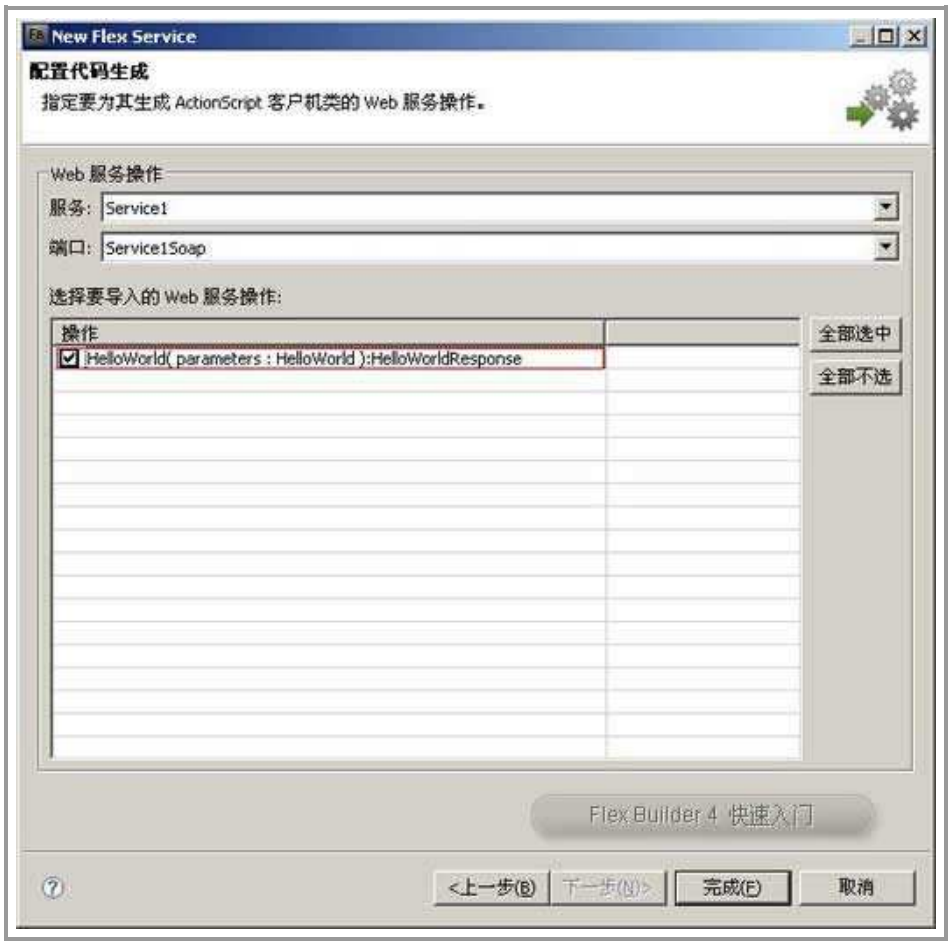
如果没有列出你的WebService的方法，就换一下端口：（图八十三）



图八十三

Web服务操作

成功的列出了我们的HelloWorld（ ）方法，勾选它点击完成：（图八十四）



图八十四

WebService配置代码生成

这时，系统会自动为我们生成一些as文件，同时你可以在“数据/服务”窗体看到HelloWorld方法的信息。

图八十五

WebService配置生成

接下来在设计视图选择按钮，在右边的属性窗体给按钮添加“单击时”的事件，选择“生成服务调用”（图八十六）

在弹出的“生成服务调用”对话框中，我们看到了HelloWorld方法，直接点击确定。（图八十六）

图八十六

生成服务调用

成功以后，工程会自动引入WebService的命名空间，自动给按钮添加一个点击事件来调用我们的HelloWorld（ ）方法。

方法需要传入一个参数，我们把输入框的text值放进去。

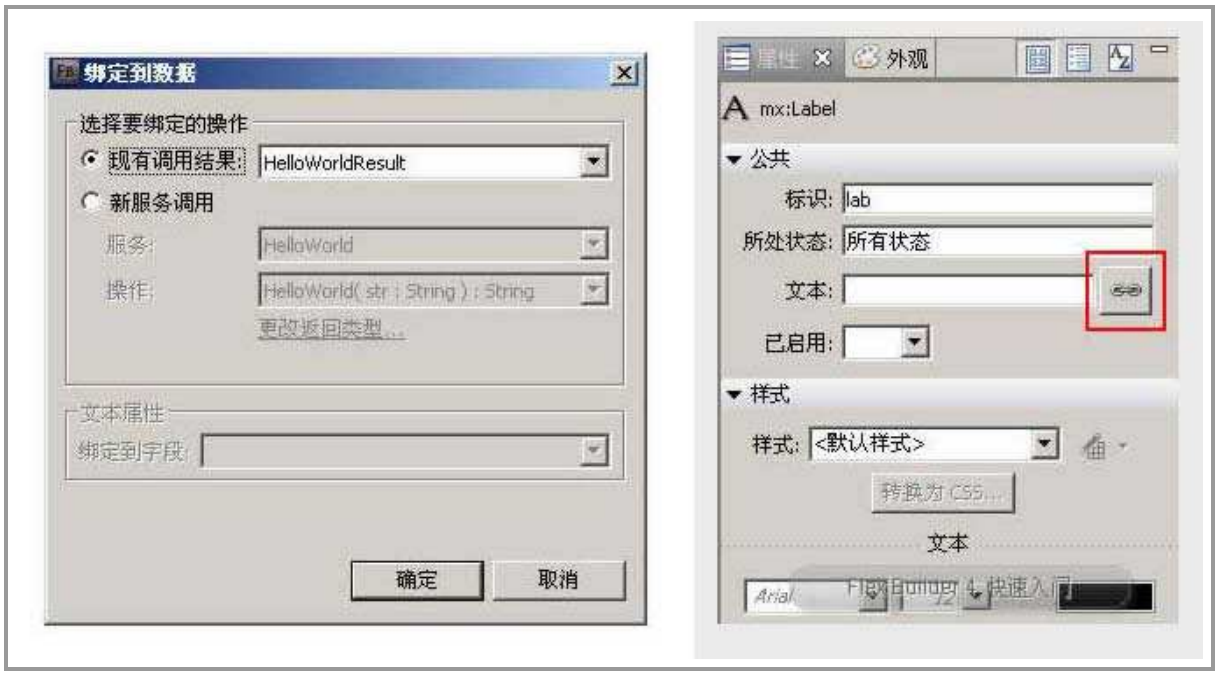
```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/halo"
               minWidth="1024" minHeight="768" xmlns:helloworld="services.helloworld.*">

    <fx:Script>
        <![CDATA[
            import mx.controls.Alert;
            protected function button1_clickHandler(event:MouseEvent):void
            {
                // TODO Auto-generated method stub
                HelloWorldResult.token = helloWorld.HelloWorld(txt.text);
            }
        ]]>
    </fx:Script>
    <fx:Declarations>
        <s:CallResponder id="HelloWorldResult"/>
        <helloworld:HelloWorld id="helloWorld" fault="Alert.show(event.fault.faultString)" showBusyCursor="true"/>
    </fx:Declarations>

    <s:Button id="btn" x="14" y="13" label="确定" click="button1_clickHandler(event)"/>
```


最后我们需要把返回的值放在文本框里。在设计视图选中文本框， 然后在右边的“属性”窗体里选择“文本”右边的“绑定数据”。

在弹出的“绑定到数据”对话框中，又看到了HelloWorld，直接确定：（图八十七）

图八十七

生成服务调用



好了，一切就绪，再来测试一下最终结果！

在输入框输入，点击按钮，服务端成功的返回了我们想要的！

图八十八

WebService返回成功

总结：

至此你又学会了如何和WebService通信！在高级应用中，如果WebService返回一个泛型集合，在Flash Builder里可以自动生成相应的AS实体类型来接受，相当于对象之间的传递，十分方便。

与服务端通信 第三节　通过Remoting与服务端通信

本小节我们来学习借助 BlazeDS 使用 Remoting 方式与后台服务器通信，为了活跃下学习气氛我先用一个大家耳熟能详的例子来说明下什么是 Remoting 什么是 BlazeDS。

场景一：熟悉的音乐响起

“噔~噔噔 噔 噔 噔 噔，噔~噔噔 噔 噔 噔 噔”

龟田小队长带领鬼子来到马家河子李家村，把村民们召集到打谷场。龟田喊话了

“八路軍の行方を言い出すのでさえすれば、皇軍がとても大きくて賞があります”。

好，Remoting 的意向建立了，鬼子想知道八路的下落，而村民中汉奸恰巧能提供这个服务（Service），一对供应关系产生。需方：鬼子（前台 Flex），供方：汉奸（后台）。

场景二：村民议论纷纷

“这孙子说什么呢？”

好，Remoting 关系虽然建立，但是由于语言不通（Flex 为 AS3 语言，后台为 Java 等语言），无法使双赢交易进行，汉奸急啊。

场景三：就在这时，肩负中日两国人民友谊桥梁之重任的翻译官出场

“太君说了，只要你们说出八路的下落，皇军大大滴有赏。”　“太君，我说，八路就在王二麻子家养伤，我给您带路。”

翻译官在龟田耳边一番耳语，龟田脸上露出了狡黠的笑容。

“にで、りょしんだだでほうや”（你滴，良心大大滴好呀）。

好，至此一次调用结束，这时我要是问“BlazeDS 在 Remoting 里是干嘛的呀”有悟性的朋友肯定会回答我“翻译官啊”。

学习目标

这一节我们采用一个用户登录的示例来演示通过Remoting与服务端通信。以下是完成后的截图，经过后台验证给出登陆信息。 (图九十)

图九十

Remoting与服务端通信

准备工作

希望你对Java有一定了解，会写简单的Servlet。

BlazeDS 4 下载地址：下载
Java 开发工具：Eclipse 3.4
Servlet 容器：Tomcat 6.0

实现步骤

接下去我们就一步步的来实现这一应用。

Server 端：

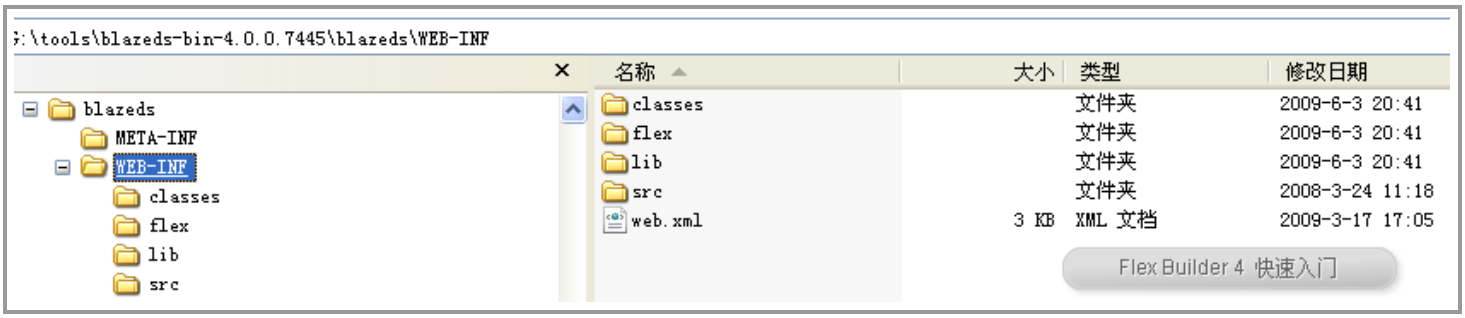
1、使用 Eclipse 3.4 建立一个 Dynamic Web Project。

2、将下载好的“blazeds-bin-4.x.x.xxxxx.zip”文件解压，得到以下两个文件。（图九十一）

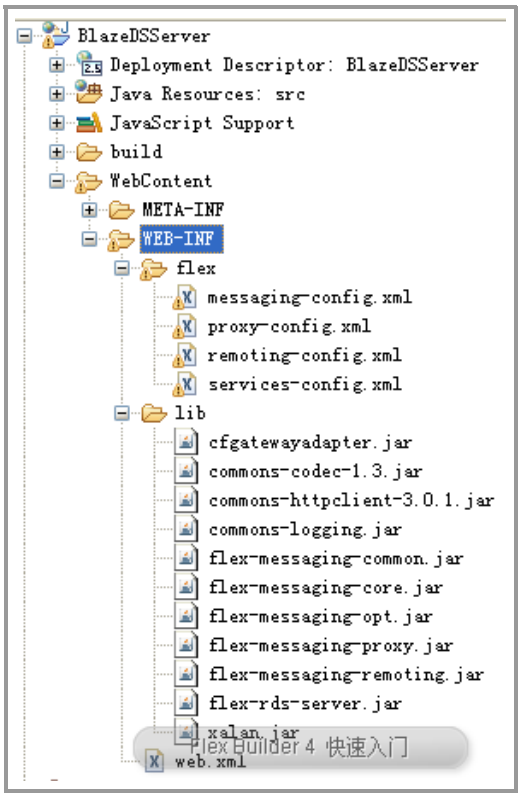
图九十一

需要的blazeds文件

接下来我们再对其中名为“blazeds.war”的文件进行再次解压，得到以下目录结构。



3、将上图中flex,lib两个文件夹以及web.xml三样东西复制到我们的 Web Project 的 WEB-INF 下并覆盖同名文件，得到以下的工程结构。（图九十三）



图九十三

复制文件到 WEB-INF

4、打开 web.xml 文件。将其中被注释掉的代码解禁。注意，还要将 useAppserverSecurity 的 value 由 true 改为 false。（图九十四）

修改前

```
<!-- begin rds
    <servlet>
        <servlet-name>RDSDispatchServlet</servlet-name>
        <display-name>RDSDispatchServlet</display-name>
        <servlet-class>flex.rds.server.servlet.FrontEndServlet</servlet-class>
        <init-param>
            <param-name>useAppserverSecurity</param-name>
            <param-value>true</param-value>
        </init-param>
        <load-on-startup>10</load-on-startup>
    </servlet>

    <servlet-mapping id="RDS_DISPATCH_MAPPING">
        <servlet-name>RDSDispatchServlet</servlet-name>
        <url-pattern>/CFIDE/main/ide.cfm</url-pattern>
    </servlet-mapping>
end rds -->
```


图九十四

修改web.xml 的

useAppserverSecurity

该截图基于bds4

修改后

```
<servlet>
    <servlet-name>RDSDispatchServlet</servlet-name>
    <display-name>RDSDispatchServlet</display-name>
    <servlet-class>flex.rds.server.servlet.FrontEndServlet</servlet-class>
    <init-param>
        <param-name>useAppserverSecurity</param-name>
        <param-value>false</param-value>
    </init-param>
    <load-on-startup>10</load-on-startup>
</servlet>

<servlet-mapping id="RDS_DISPATCH_MAPPING">
    <servlet-name>RDSDispatchServlet</servlet-name>
    <url-pattern>/CFIDE/main/ide.cfm</url-pattern>
</servlet-mapping>
```

Flex Builder 4 快速入门

5、为了体现 BlazeDS 的优势——即 AS3 对象和 Java 对象之间的自由映射转换，我们在这里定一个 Java Bean “User”用以维护用户名和密码等信息。

```
package cn.airia.fb4.vo;
public class User {
    private String username;
    private String password;
    public String getUsername() {
       return username;
    }
    public void setUsername(String username) {
       this.username = username;
    }
    public String getPassword() {
       return password;
    }
    public void setPassword(String password) {
       this.password = password;
    }
}
```

code

FlexBuilder 4 快速入门 代码片段

6、撰写核心代码 UserService.java，这个类会成为前台所调用的 RemoteClass，在这个类里我们使用 Hard Code 的方式实现用户登陆的验证。

```
package cn.airia.fb4.service;

import cn.airia.fb4.vo.User;

public class UserService {
    public String login(User user)
    {
       if(user.getUsername().equals("momoko") && user.getPassword().endsWith("123"))
       {
           return "Welcome "+user.getUsername();
       }
       else
       {
           return "Login failed";
       }
    }

}
```

code

FlexBuilder 4 快速入门 代码片段

7、打开 WEB-INF/flex/remoting-config.xml，在<server>...</server>标签中加入如下配置信息。用 source 来标记之前我们定义的类 UserService，来表示这个类被开放出来以供前台调用，同时我们也要为其设置一个 id 以标示这个类。

```
<destination id="userService">
    <properties>
        <source>cn.airia.fb4.service.UserService</source>
    </properties>
</destination>
```

code

FlexBuilder 4 快速入门 代码片段

以下是完整的remoting-config.xml （图九十五）

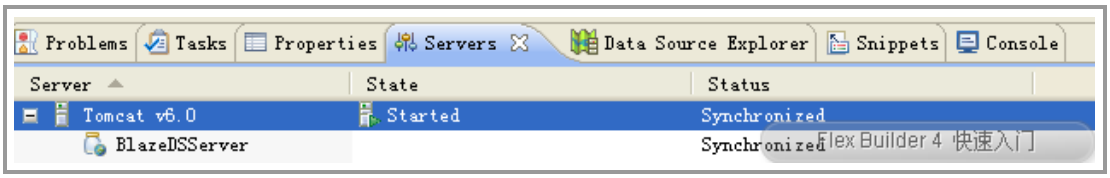
```
<?xml version="1.0" encoding="UTF-8"?>
<service id="remoting-service"
    class="flex.messaging.services.RemotingService">
    <adapters>
        <adapter-definition id="java-object"
        class="flex.messaging.services.remoting.adapters.JavaAdapter"
        default="true"/>
    </adapters>
    <default-channels>
        <channel ref="my-amf"/>
    </default-channels>
    <destination id="userService">
        <properties>
            <source>cn.airia.fb4.service.UserService</source>
        </properties>
    </destination>
</service>
```

Flex Builder 4 快速入门

图九十五

remoting-config.xml

8、将项目添加到 Tomcat 中并启动，至此 Sever 端的操作全部完成，等待 Client 端调用。（图九十六）



图九十六

项目添加至Tomcat

Client 端：

1、新建一个 Flex Project，注意急性子的同学手不要太快，Remoting 访问方式和 Httpservice 以及 webservice 有所区别需额外设置后方能“下一步”，我们在“应用服务器类型”一栏中选择“J2EE”和“BlazeDS”后点击“下一步”。（图九十七）

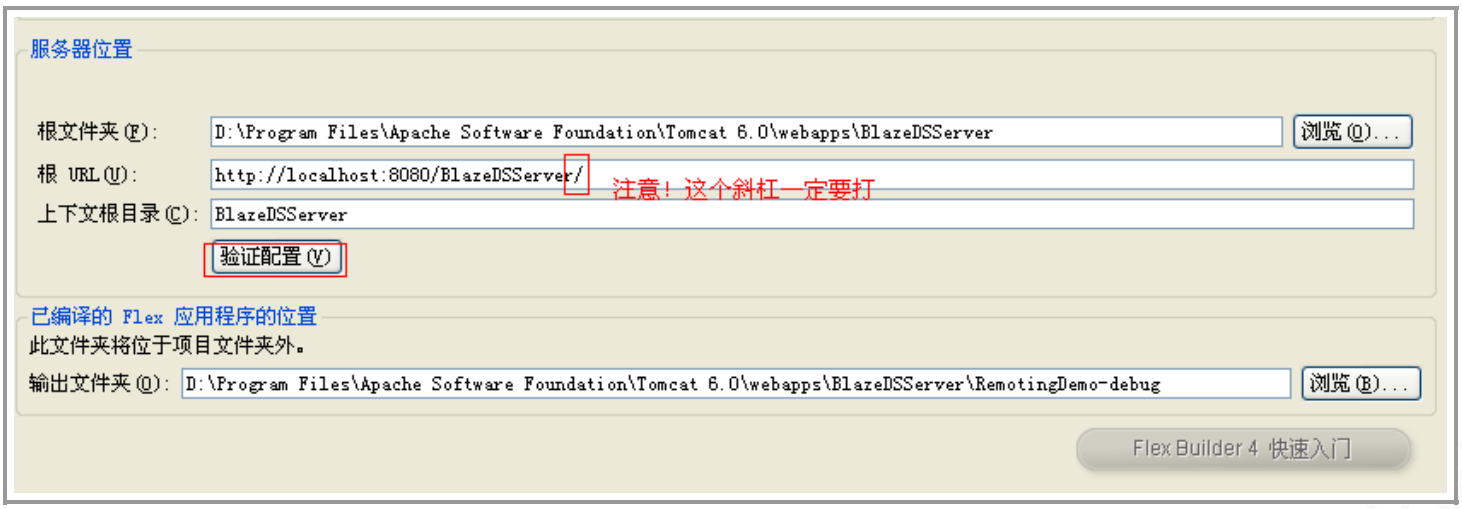
图九十七

新建FLEX项目

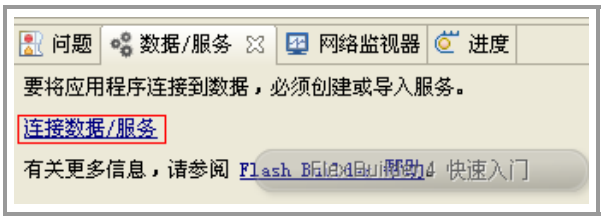
在弹出窗体中进行设置

- 根文件夹：定义为Server端程序的发布路径
- 根URL：定义为 Server 端程序的URL访问路径
- 上下文根目录：定义为 BlazeDSServer
- 输出文件夹：系统会自动生成

点击“验证配置” 按钮，若无错误提示，即可进入下一步后直接按“完成” 。



2、在“数据/服务”视图，我们来配置之前后台发布的 RemoteClass，点击“链接数据/服务”link。（图九十九）



图九十九

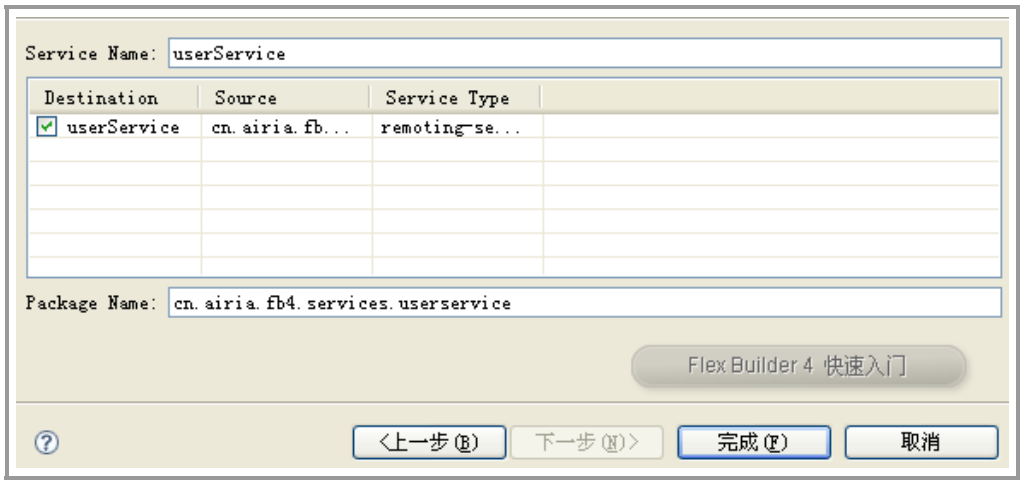
连接数据服务

在弹出窗体中选择 BlazeDS 后下一步。（图一百）

图一百

连接数据服务

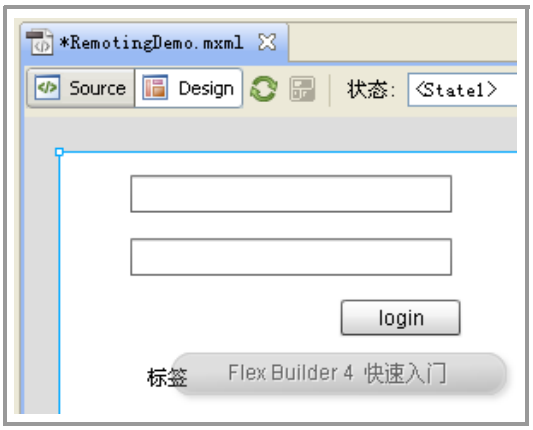
有可能会询问你RDS的密码，我们可以选择不需要密码直接跳过这步，在接下来的弹出窗体中可以看到后台发布的 RemoteClass，简单设置下Service Name以及 Package Name 后，点击“完成”按钮。（图一百零一）



图一百零一

设置Service

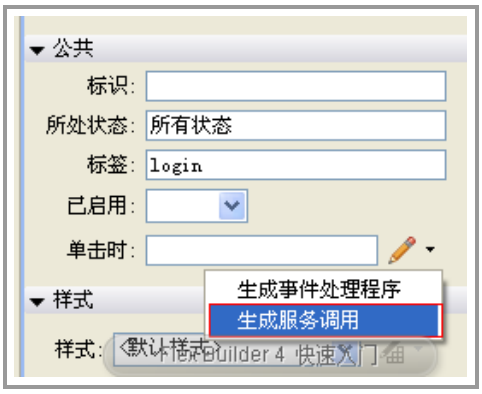
3、在 RemotingDemo.mxml 的 Design 视图中绘制界面，这里使用了两个 TextInput，一个 Button 和一个 Label 控件，这里分别为两个 TextInput 控件定义标示“textInputUsername”和“textInputPassword”。（图一百零二）



图一百零二

绘制界面

4、在Design视图中选中 Button，在属性面板里点击铅笔按钮，选择“生成服务器调用”。（图一百零三）



图一百零三

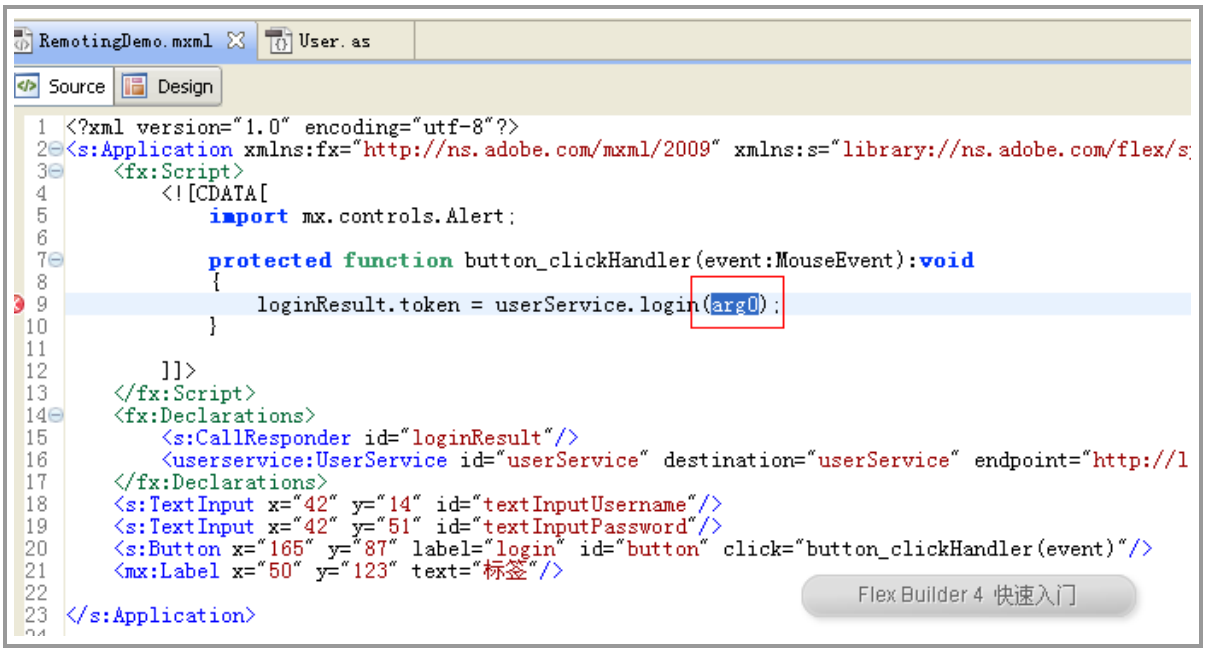
生成服务器调用

在弹出窗口中选择服务和操作后，点击“确定”按钮。（图一百零四）

图一百零四

生成服务器调用

此时视图切换到 RemotingDemo.mxml 的 Source 视图。并自动生成了“login”按钮的点击处理事件代码段，我们对这段代码进行编辑，将“arg0”替换为一个 User 对象。



修改后的代码如下，我们可以看到这里的 User 类，是由系统参照 Server 端的定义自动为生成的。

```
protected function button_clickHandler(event:MouseEvent):void
{
    var user:User = new User();
    user.username = textInputUsername.text;
    user.password = textInputPassword.text;
    loginResult.token = userService.login(user);
}
```

code

FlexBuilder 4 快速入门 代码片段

5、切换到的 Design 视图，选择 Label 控件，在右侧的属性面板中，点击 按钮将其与返回结果绑定，点击“确定”按钮。（图一百零六）

图一百零六

Label绑定返回结果

图一百零七

绑定到数据

6、编码结束（其实我们基本没有手动编写代码噢：）），运行我们的程序看看效果吧。（图一百零八）

图一百零八

完成测试

总结：

BlazeDS 真的非常方便，建议会 J2EE 的同学从它开始上手。缺点嘛，BlazeDS 的 license 是基于 GNU GPL 协议，该协议大致意思就是，如果你在你的项目中使用了 BlazeDS，而你的项目又是以产品的形式销售给客户，那么你就必须把你的项目全部开源。如果你的客户是比较正规的国外公司，而你又不想开源，那么请慎用 BlazeDS 吧。

思考：

BlazeDS 可以与 Spring Bean 进行整合，有兴趣的同学可以回家试试。 BlazeDS 提供了 Message 服务，想全面了解 BlazeDS 的同学不妨用它的 Message 服务来实现一个简单聊天室。

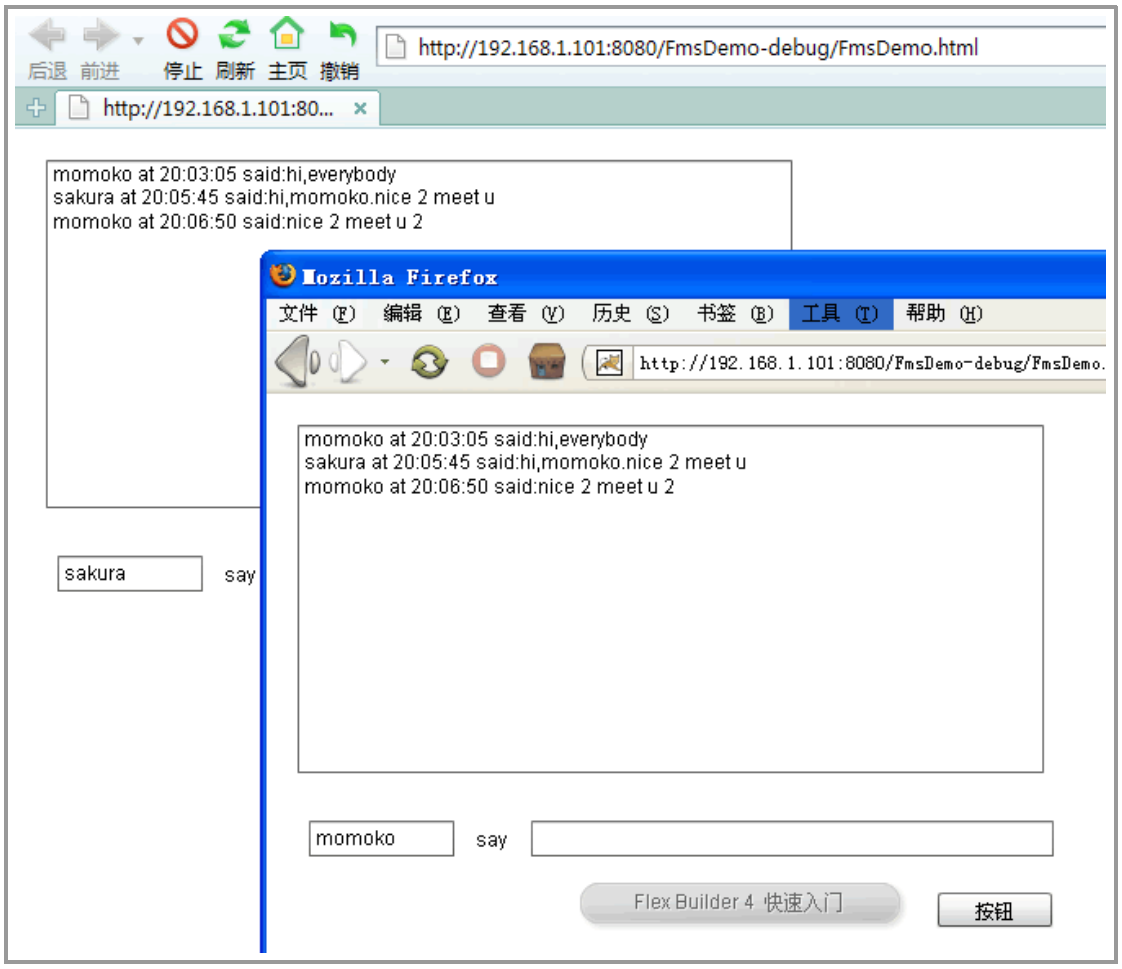
与服务端通信 第四节 与Flash Media Server交互

Flash Media Server 简称 FMS，目前版本号已经是3.5，FMS 可以作为提供诸如视频会议，网络游戏等交互式应用的服务平台，尤其在多媒体方面表现

尤为出色，Adobe提供了免费的开发版供大家学习，开发版支持10个客户端。本节内容就是教会大家如何使用 FMS 制作一个简单的聊天室。

学习目标

这一节要教会大家使用 FMS 通过SharedObject方式来实现聊天室的功能，先来看看完成后的作品。（图一百零九）



图一百零九

学习目标

工作原理

本例中 FMS 聊天室的工作原理很简单，关键在于一个名为SharedObject的对象，我们可以把它看成所有聊天用户共享的全局对象，谁都可以读写它，如果A

用户发言就等于把一条对话追加到这全局变量中去，这个变量一旦发生改变便会通知所有其他在线聊天用户说：“同志们我的内容被更新啦”，其它客户便

可以去从这个变量 中取得最新的聊天记录，从而知道刚才A说了什么。稍微有点复杂，看一遍没看懂的同学，请认真地逐字逐句的把前面这段话好好理解，

这是聊天室现原理所在。

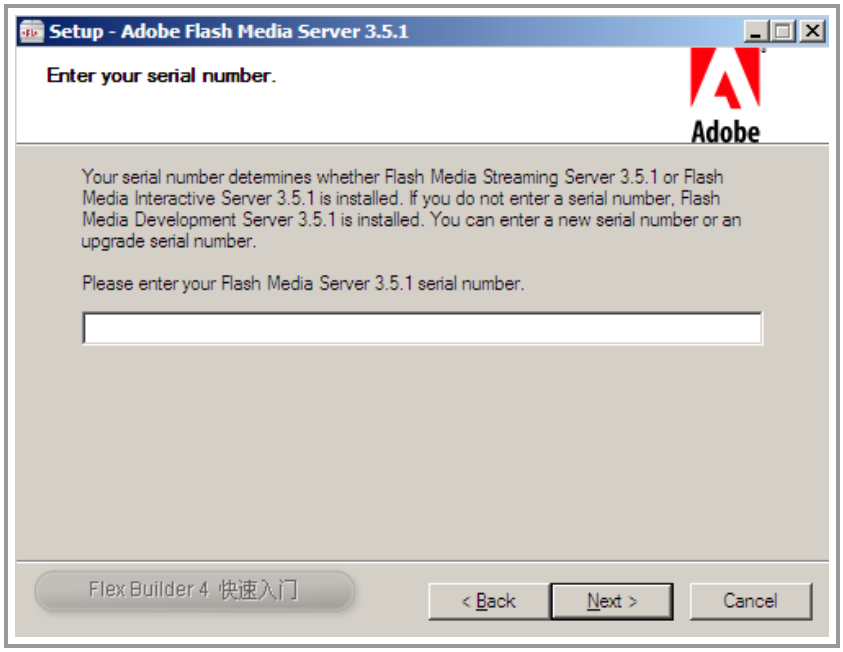
准备工作

1、下载工具

FMS ：~~FMS 3.5(需要注册一个Adobe的账号)~~ Servlet 容器：~~Tomcat 6.0~~

2、安装 FMS 3.5。

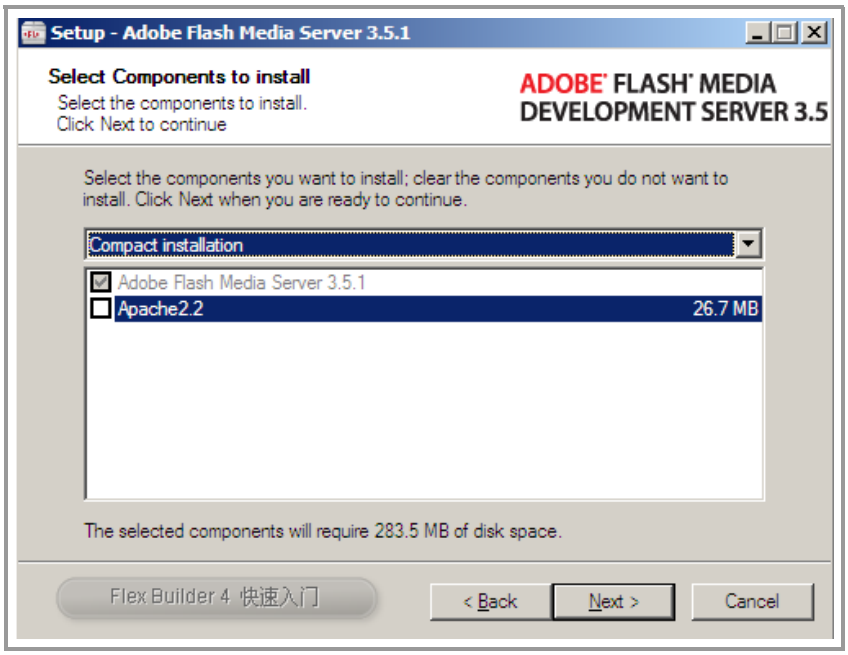
遇到下图这一步，是让你输入序列号，当然我们肯定没有，所以空着直接 Next。不输入序列号我们的 FMS就会成为开发版可以永久使用，但是最多只能支持10个客户端。（图一百一十）



图一百一十

安装FMS 3.5

之后会让你选择安装 Apache 服务器，由于我们在前几节课程中都是使用的 Tomcat，因此这一步就不安装了。（图一百一十一）



图一百一十一

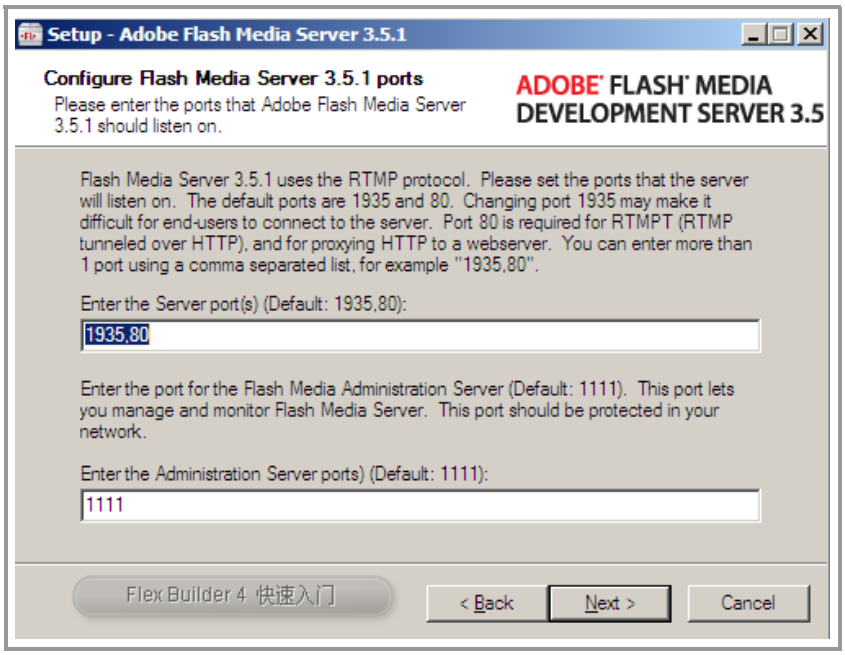
安装FMS 3.5

再接下去就是要配置 FMS 的管理员账号了。（图一百一十二）

图一百一十二

安装FMS 配置账号

最后是配置 FMS Server 的端口以及 Server 控制台的访问端口，系统默认是 1935,80,1111这3个端口，请确保没有被其它程序占用。（图一百一十三）



图一百一十三

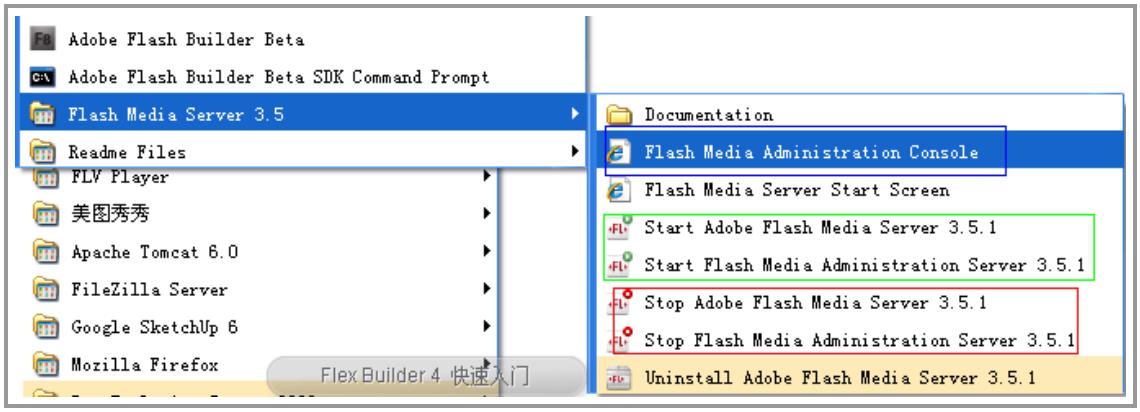
安装FMS 配置端口

3、在 FMS 安装目录下的“applications”文件夹下建立一个名为“myApp”的文件夹，作为 Server 端项目根路径。以后我们的 SharedObject 对象便会存放在这个位置。（图一百一十四）

图一百一十四

建立myApp文件夹

4、在开始菜单中可以看见一些按钮，蓝色部分是打开服务器控制台，红色部分是关闭 FMS 服务，绿色部分是打开 FMS 服务。



点击蓝色部分按钮输入之前设置的账号可进入控制台。

实现步骤

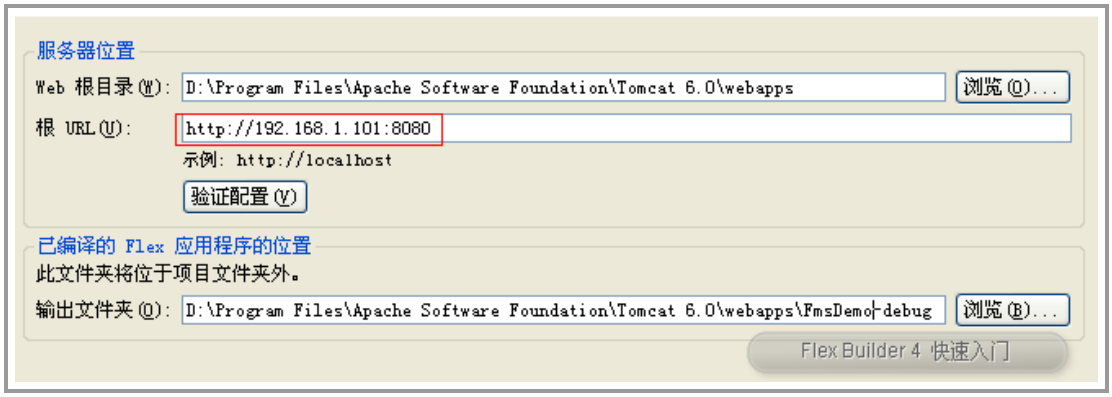
1、建立 Flex 项目，因为我们做的是聊天室，这个工程架设在 Web 容器中（如 Tomcat 等）中会更加便于我们调试，所以我们在服务器技术里选择“J2EE”，由于这里服务端技术使用的是 fms，因此我们将“使用远程对象访问服务”给勾去后点击“下一步”。（图一百一十七）

一百一十七

建立FLEX项目

2、之后我们对项目进行服务器配置。

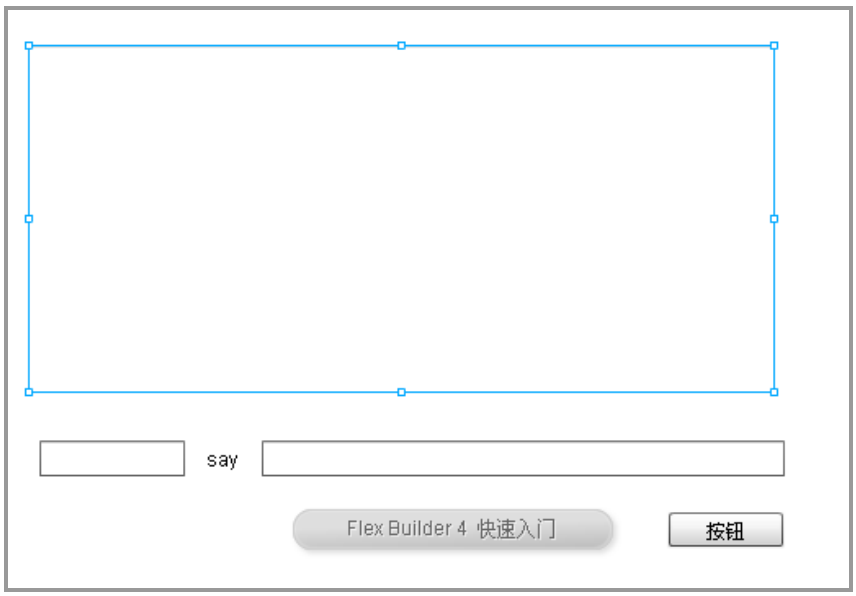
- Web 根目录：设置为 Tomcat 的项目发布目录。
- 根 URL ：设置为访问 Tomcat 的根地址，注意！为了便于局域网内的其他同学观摩你的作品，请务必使用ip地址，而非localhost。
- 输出路径：系统会自动指定，无需自己设置。
- 最后点击“验证配置”，待没有错误提示后便可以点击“完成”，结束工程的设置。（图一百一十八）



一百一十八

服务器配置

3、绘制简单的聊天界面，一个 TextArea 用于显示对话，一个 TextInput 用于输入用户昵称，一个 TextInput 用于输入消息，一个 Button 按钮发送对话。（图一百一十九）



一百一十九

绘制界面

设置完后我们顺便为这些控件设置下标示。“textAreaContent”，“textInputName”，“textInputMessage”，“buttonSend”等。

4、我们定一个 Message Bean 来记录每一条对话。结构如下，分别是昵称，消息内容，时间。

```
package cn.airia.fb4.vo
{
    public class Message
    {
        public function Message()
        {
        }
        public var nickname:String;
        public var msg:String;
        public var time:Date;
    }
}
```

code

FlexBuilder 4 快速入门 代码片段

5、接下去开始编码，首先我们定义一些变量。

```
private var netConnection:NetConnection;
private var serverApp:String = "rtmp://192.168.1.101/myApp";
private var talkSO:SharedObject;
```

code

FlexBuilder 4 快速入门 代码片段

netConnection:可以认为它是 Flex 与 FMS 通讯的桥梁。
serverApp: 配置的是 FMS 服务的地址，注意这里使用的是rtmp协议。
talkSO: 一个SharedObject对象，可以把它看成一个供所有聊天室用户存放聊天记录的全局对象。

6、在程序初始化执行一些操作。

```
private function init():void
{
    netConnection = new NetConnection();
    buttonSend.addEventListener(MouseEvent.CLICK,sendMessage);
    netConnection.addEventListener(NetStatusEvent.NET_STATUS,netStatusHandler);
    netConnection.addEventListener(AsyncErrorEvent.ASYNC_ERROR,asyncErrorHandler);
    netConnection.connect(serverApp);
}
```

code

FlexBuilder 4 快速入门 代码片段

NetStatusEvent.NET_STATUS: 用于监听连接状态。
AsyncErrorEvent.ASYNC_ERROR: 是用于监听一些连接中的错误信息。
netConnection.connect(serverApp): 这句的作用是正式开始连接 FMS 服务器。

这里主要是加一些监听。

NetStatusEvent.NET_STATUS的回调函数netStatusHandler用于根据服务器返回状态判断是否连接成功。

```
private function netStatusHandler(evt:NetStatusEvent):void
{
    if(evt.info.code == "NetConnection.Connect.Success")
    {
        talkSO = SharedObject.getRemote("talk",netConnection.uri,false);
        talkSO.addEventListener(SyncEvent.SYNC,talkSOHandler);
        talkSO.addEventListener(AsyncErrorEvent.ASYNC_ERROR,asyncErrorHandler);
        talkSO.connect(netConnection);
    }
    else
    {
        Alert.show("connect failed "+evt.info.code);
    }
}
```

code

FlexBuilder 4 快速入门 代码片段

我们可以看到当服务器返回的 code 为“NetConnection.Connect.Success”时表示连接成功，那么我们就可以开始对这个全局共享变量talkSO进行一些操作了。

SharedObject.getRemote("talk",netConnection.uri,false);这句的意思是，跑到 FMS 服务器下找“talk”这个文件，如果没有则新建一个用以存放我们的全局变量，另外第三个参数的false的意思，就是说这个文件是临时的，当 FMS 服务器关闭时这个“talk”就被删除了。

小测试1：思考以下2个问题，如果回答不出，请重新阅读或放弃本教程。

netConnection.uri的内容及作用是什么?
第三个参数如果是 true 是代表什么?

talkSO.addEventListener(SyncEvent.SYNC,talkSOHandler);这句的作用比较关键，它监听的是talkSO变量也就是“talk”文件的内容有没有被其它用户改写。

talkSO.connect(netConnection);这句例行公事，把前面对于 talkSO 的设置通过桥梁 netConnection 与 FMS 服务器扯上关系，大家死记硬背即可。

小测试2：思考以下2个问题，如果回答不出，请重新阅读或放弃本教程。

```
talkSO.addEventListener(AsyncErrorEvent.ASYNC_ERROR,asyncErrorHandler);
这句意思是什么?
```

7、桥梁，全局变量什么的都设置好了，接下去就是核心代码了。

```
private function sendMessage(evt:MouseEvent):void
{
    var array:ArrayCollection = new ArrayCollection();
    if(talkSO.data.msgList != null)
    {
        convertArrayCollection(array,talkSO.data.msgList as ArrayCollection);
    }
    var message:Message = new Message();
    message.nickname = textInputName.text;
    message.msg = textInputMessage.text;
    message.time = new Date();
    array.addItem(message);
    talkSO.setProperty("msgList",array);
    textInputMessage.text = "";
}
```

code

FlexBuilder 4 快速入门 代码片段

“talkSO”有个data属性，是用来存放数据的，我们需要它来存放我们所有的聊天记录，这里给聊天记录的集合起个名字叫做“msgList”，这段代码的作用就是把“msgList”拿下来，然后把用户的发言追加进去，然后通过talkSO.setProperty("msgList",array);这句把它写回 FMS 服务器去。

```
private function talkSOHandler(evt:SyncEvent):void
{
    textAreaContent.text = "";
    if(talkSO.data.msgList != null)
    {
        var tmp:ArrayCollection = new ArrayCollection();
        convertArrayCollection(tmp,talkSO.data.msgList as ArrayCollection);
        var formatter:DateFormatter = new DateFormatter();
        formatter.formatString = "HH:NN:SS";
        for(var i:int = 0;i<tmp.length;i++)
        {
            var message:Object = tmp.getItemAt(i) as Object;
            var timeString:String = formatter.format(message.time);
            var fullMsg:String = message.nickname + " at " + timeString + " said:" + message.msg;
            textAreaContent.text = textAreaContent.text+fullMsg+"\n";
        }
    }
}
```

code

FlexBuilder 4 快速入门 代码片段

这段代码是对于“SyncEvent.SYNC”事件所作出的响应，当有用户更新了聊天记录时候就会触发执行。主要功能就是把聊天记录“msgList”从服务器拿下来，然后遍历下所有对话，把它们打印在 TextArea 中。

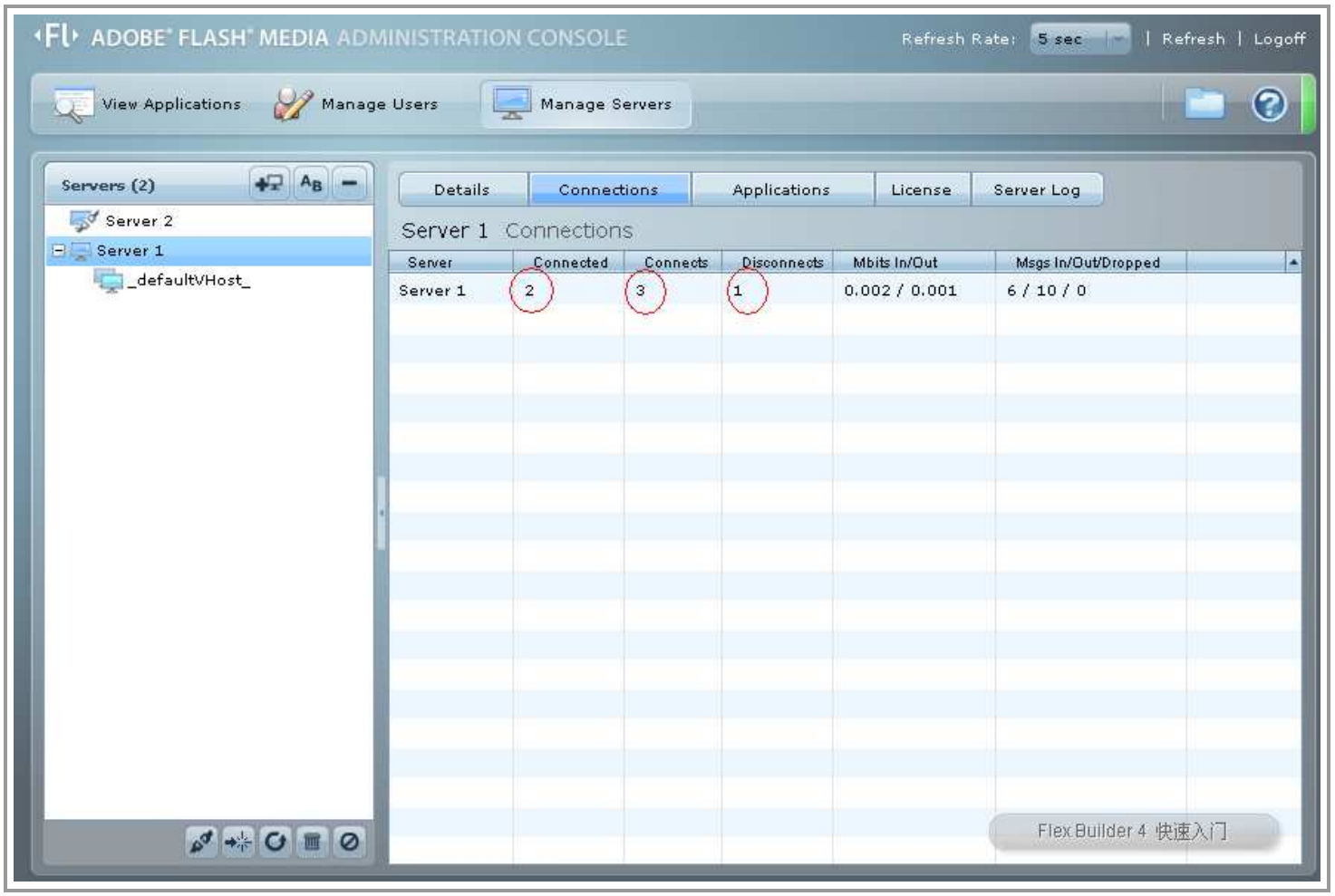
```
private function convertArrayCollection(arrNew:ArrayCollection,arrOld:ArrayCollection):void
{
    arrNew.removeAll();
    for(var i:int = 0;i<arrOld.length;i++)
    {
        arrNew.addItemAt(arrOld.getItemAt(i),i);
    }
}
```

code

FlexBuilder 4 快速入门 代码片段

这段代码作用是用于 ArrayCollection 的复制。

8、运行代码，叫上你局域网的朋友一起测试吧，你也可以自己开2个浏览器进行测试。当有客户端连接上，你跑去 FMS 的控制台可以看到一些连接信息。



总结：

这节我们学习了在 FMS 中利用 SharedObject 方式实现了简单聊天室，目的在于让各位同学了解 FMS 的工作机制，包括今后你们在实现视频会议的时候， 其实其中原理也是一样的。本人在以往 Flex3 的开发中发现 SharedObject 的性能和可靠性实在不敢恭维（ Flex4 中尚未验证），因此请大家在实际项目 中慎用 SharedObject 进行频繁的读写操作。

思考：

使用 FMS 实现简单的视频录制和播放。 使用 FMS 实现简单的多人视频。

业界声音（排名不分先后）

王磊：

（Flex 第一步 作者）

《Flash Builder 4 入门教学》是什么？它是AIRIA论坛里面无数无私奉献的高手聚合而已成的一本属于Flash开发者快速上手Flash Builder 4的入门级“书籍”。在AIRIA论坛编辑团队的共同努力下，短短的20多天里面就能做出这么一份图文并茂的入门教学来，的确不易。

《Flash Builder 4 入门教学》阐述了这款最新的IDE的一些显著特点：代码模板、包重构、Setter 与 Getter自动生成等，而无论是初学者还是经验者，都应该仔细的研究一下，希望更多的朋友从中获取到自己想要的内容。

杨占坡：

（Flex 3 RIA开发详解与精深实践 作者）

千呼万唤始出来，Flash Builder终于发布了Beta版本。FB4从命名上就表明了Adobe的理念，一方面突出了Flex开发的最终目的和对大优势；另一方面，全面推动动画设计、软件开发、Web、RIA领域的全能型技术平台。

让人吃惊的是，AIRIA的斗士们第一时间就开始筹划FB4中文教程。这对于Flex领域的Developer来说是非常之利好。AIRIA作为中国最大的Flex RIA社区有着深厚的技术积淀，可以预期这份教程将带给Flex Developer最有价值的开发路线图。

在下一页发出你的声音...

艾睿网
airia.cn

教学编辑团队 出品