Flex Viewer 解析

作者	ropp
邮箱	fromirsa@gmail.com
文档版本	1.0

目　　录

1. Flex Viewer 简介 1

2. Flex Viewer 源码包结构 1

3. Flex Viewer 架构解析 3

   3.1 Flex Viewer 主席团 3

   3.2 Flex Viewer 松耦合的关键 5

   3.3 初始化那些事儿 6

   3.4 Widget 设计及实现 8

4. 自定义 Widget 10

5. Widget 之间通信 14

   5.1 直接通信 14

   5.2 间接通信 15

6. 让 Flex Viewer 为我所用 16

7. 后记 18

1. Flex Viewer 简介

Flex Viewer 是 ArcGIS Viewer for Flex 的简称，其 1.x 版本的名称为 Sample Flex Viewer，自 2.0 改称 ArcGIS Viewer for Flex。Flex Viewer 是基于 ArcGIS API for Flex（简称 AGS Flex API）设计的一个 WebGIS 应用程序，1.x 版本基于 Flex3， 2.x 版本基于 Flex4，当前最新版本是 2.3。Flex Viewer 的版本是随着 AGS Flex API 更新而更新的，并且版本号与其保持一致。每个版本的 Flex Viewer 都体现了其对应版本的 AGS Flex API 的新特性和新功能，同时，Flex Viewer 本身在每个版本中都会有一定程度的更新和改进。

Flex Viewer 在其官网有两种版本供下载（见本小节附录），一种是针对非开发人员的版本，即编译后的发布（release）版本。基于发布版本，用户通过修改配置文件，即可将自己的底图和业务图层通过 Flex Viewer 展现，并通过 Flex Viewer 自带的 Widget 实现查询、定位、空间分析等各种功能。另一种版本是针对开发人员的源代码，开发人员下载源代码后，可导入到 Flash Builder 开发环境，通过修改配置文件、自定义 UI、自定义 Widget 来实现特定的业务需求。目前已有一定数量基于 Flex Viewer 开发的在线系统，如辽宁省地理信息公共服务平台（见本小节附录）。

附录

1） Flex Viewer：http://help.arcgis.com/en/webapps/flexviewer/index.html

2） 辽宁省地理信息公共服务平台：http://www.lnditu.gov.cn/Service4People/index.html

2. Flex Viewer 源码包结构

Flex Viewer 源代码是 Flash Builder 中标准的 Flex 工程，可直接导入到 Flash Builder。下面分别介绍一下源代码包结构中的各个部分，包结构如图 2.1 所示。

1）src 根目录：包含 index.mxml、defaults.css 和 config.xml。index.mxml 是系统入口点，也就是创建 Flex Application 实例的地方；defaults.css 是 Flex Viewer 中用来定义组件样式的文件，所有对组件样式进行定义的 css 脚本都可以放到这个文件中，defaults.css 在 index.mxml 中被引用；config.xml 是缺省配置文件。

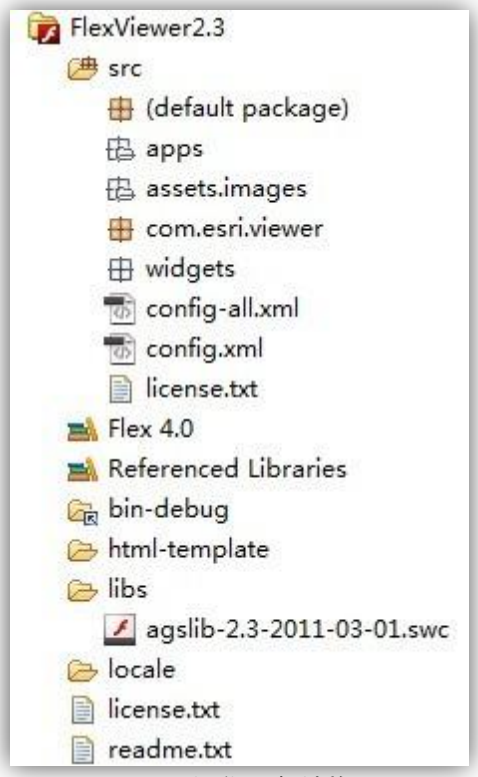


图 2.1 源代码包结构

2）apps：Flex Viewer 的良好设计使其具有很高的可配置性，包括底图、业务图层、各种服务器端资源、甚至是用户体验都是可配置的。apps 包中的内容是不同应用系统的配置文件及各种所需资源。Flex Viewer 可以在 url 参数中设置所需加载的配置文件，比如：在 Flex Viewer 的 url 后加上配置文件的信息“?config=apps/zh_CN/config.xml”，Flex Viewer 就会加载 apps/zh_CN 下的 config.xml 文件，根据该配置文件中的信息来配置整个 Flex Viewer 系统。也就是说，Flex Viewer 根据不同的配置文件可以展现完全不同的应用系统。

3）assets.images：Flex Viewer 使用的各种图片、flash 资源所在的目录。

4）com.esri.viewer：该包及其子包的内容是 Flex Viewer 的主体程序，这里所说的“主体程序”是相对于 Widget 而言的。主体程序中实现了 Widget 的基础、对 Widget 的管理、以及除自定义 Widget 之外的所有功能，这部分内容将在后边的小节介绍。

5）widgets：Flex Viewer 中所有的 Widget 都在此包中。Widget 基于 Module 开发，Module 是 Adobe 为解决 Flex 系统体积过大而提出的一种解决方案，较大的 Flex 系统可以通过 Module 将系统进行分割，从而减小系统初始化所需加载的体积。Flex Viewer 通过 Widget 将业务功能进行划分，每个 Widget 都是一个功能相对完善和

独立的组件，每个 Widget 可以独立完成一个或者一组相关操作。

6）libs：这是 Flash Builder 中 Flex 工程存放库文件的目录，agslib-2.3-2011-03-01.swc 是 AGS Flex API 2.3 的库文件。根据不同的业务需求，可能会用到更多的库文件，那么这些库文件都将放到这个目录下。

7）locale：Flex Viewer 支持国际化，国际化所需的属性文件全部放在该目录下。在 Flash Builder 中，可通过指定编译参数来决定使用哪种语言，如图 2.2 所示：



图 2.2 设置编译参数

3. Flex Viewer 架构解析

OK！进入主题，本小节我们一起来探究一下 Flex Viewer 的庐山真面目！

3.1 Flex Viewer 主席团

在 com.esri.viewer.managers 包中，汇集了 Flex Viewer 中所有的 Manager。这些 Manager 各个肩负重任，他们虽隐身幕后，却是 Flex Viewer 良好运转的关键，我们不妨称之为主席团成员。

1) ConfigManager

Flex Viewer 通过配置文件来组织数据、功能和 UI。ConfigManager 的责任是适时读取配置文件，对配置文件进行解析，然后将解析结果分发出去，由其他需要使用配置文件数据的模块接收。

2) DataManager

Flex Viewer 各个部分之间需要共享数据，比如说 Widget 与 Widget 之间数据共享。DataManager 提供了一种数据共享的方案，任何模块都可以通过 DataManager 把数据贡献出来，供其它模块使用。 DataManager 的关键职责是把共享数据保存在内存中，任何时候都可以从 DataManager 获取需要的共享数据。

3) MapManager

Map 是 GIS 应用的基础。MapManager 解决了 Map 的问题。MapManager 不是对 Map 的简单封装，而是提供了所有与 Map 相关的操作，比如根据配置文件加载地图，放大、缩小这些基本操作，画图，在地图上显示信息框，图层控制等等。如果在某个自定义 Widget 中，想要画一个多边形，不必 new 一个 DrawTool，发个消息告诉 MapManager 你想画个多边形即可；如果你想在地图上某个点上显示一些信息，同样发个消息告诉 MapManager 就行了。MapManager 会很友好地帮我们完很多工作，我们只需发个消息知会一声。

4) ScriptingManager

预留，尚未使用。

5) SecurityManager

预留，尚未使用。

6) UIManager

Flex Viewer 用户体验之所以风格统一，是因为 UIManager 做了大量的工作。在配置文件中，有如下脚本，UIManager 会根据这一信息对 UI 的样式进行配置。

```
<style>
  <colors>
     0xFFFFFF,0x333333,0x101010,0x000000,0xFFD700
  </colors>
  <alpha>0.8</alpha>
 </style>
```

7) WidgetManager

顾名思义，WidgetManager 是对 Widget 进行管理的组件。WidgetManager 对 Widget 的管理包括根据配置文件创建 Widget 信息列表、加载 Widget、布局 Widget、关闭 Widget 等。WidgetManager 提供四种 Widget 布局方式：自由布局（float)、水平布局（horizontal)、垂直布局（vertical）和固定布局 (fix)。自由布局 Widget 可以拖动，水平布局、垂直布局和固定布局 Widget 完全有 WidgetManager 管理，不可拖动；四种布局方式中，固定布局 Widget 不可改变窗口大小。

3.2 Flex Viewer 松耦合的关键

系统耦合度是决定系统灵活性与可维护性的关键。Flex Viewer 的松耦合设计是其健壮的关键因素之一。那么，是什么保证了 Flex Viewer 的松耦合呢?

“事件！事件！还是事件!”

事件是 Flex Viewer 松耦合的关键。在 3.1 中多次提到了“消息”，物化到 Flex Viewer 中就是事件。不同的模块通过事件彼此交互、传递数据，保证了各模块之间的松耦合，彼此不必相知，却能紧密合作。ViewerContainer、EventBus、AppEvent 组成了 Flex Viewer 事件机制的基础。当然，Flex Viewer 事件机制的基础是 Flex 的事件机制。

1) EventBus

   继承自 EventDispatcher，使用了单例模式。EventBus 是全局的事件派发器，为 Flex Viewer 中的不同模块之间的交互提供便利。有了 EventBus，不同模块之间的交互无需彼此调用对方的方法，只需派发/监听消息即可。

2) AppEvent

   继承自 Event，在 Flex Viewer 中被用来当做消息和数据的载体，在不同的模块之间传递数据。AppEvent 定义了 Flex Viewer 中需要的所有事件类型，也就是那些公共的字符串常量。通常，系统层面需要添加的事件也都定义在 AppEvent 中。

3) ViewerContainer

   继承自 Group，使用了单例模式，是 Flex Viewer 中各个模块的容器。Flex Viewer 中调用 ViewerContainer 最多的三个方法是：addEventListener()、dispatchEvent()、showError()。我们可以在任何需要对某类 AppEvent 事件进行监听的地方通过 addEventListener()方法添加监听，可以在任何需要派发某类 AppEvent 事件的地方通过 dispatchEvent()方法派发事件，可以在任何需要显示 Error 信息的地方通过 showError()方法显示 Error 信息。如果通读 Flex Viewer 代码，会发现，AppEvent 的监听与派发随处可见。

Flex Viewer 松耦合的关键因素还有 Widget 的设计和实现，这部分内容将在后面

的小节涉及，在此按下不表。

至此，我们的讨论涉及到了 Flex Viewer 中的绝大部分模块，先来看一下 Flex Viewer 的整体结构，如图 3.1 所示。图中最下方的 Control Widgets 和 Business Widgets 尚未提及，我们将在 3.4 中涉及这部分内容。

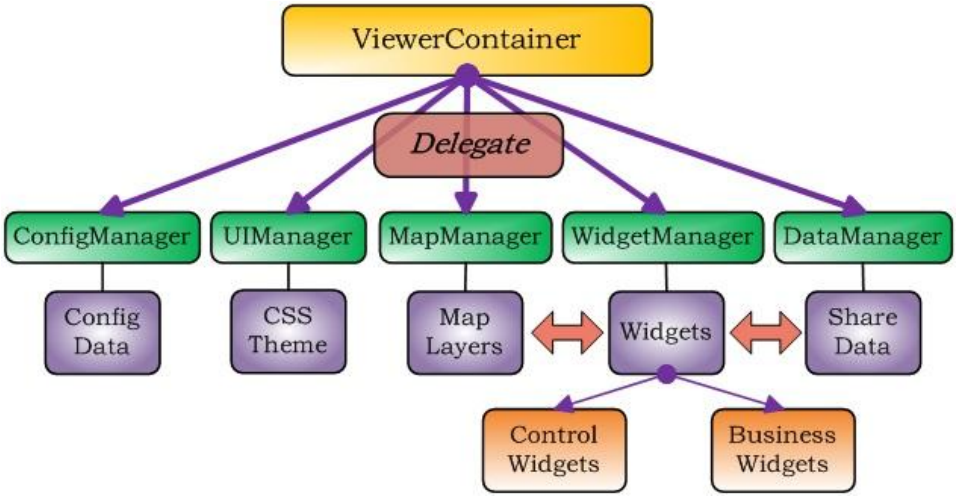


图 3.1 Flex Viewer 整体结构

3.3 初始化那些事儿

在浏览器地址栏输入 Flex Viewer 的地址，经过短暂等待，当她华丽丽地展现在我们眼前，你是否想过在这短暂的等待中，Flex Viewer 都做了哪些事情呢？本小节我们来探讨 Flex Viewer 初始化那些事儿。注意，我们这里所说的 Flex Viewer 初始化，不是 Flex 概念中组件生命周期的初始化部分，而是指 Flex Viewer 在可以与用户交互之前，所做的准备工作。

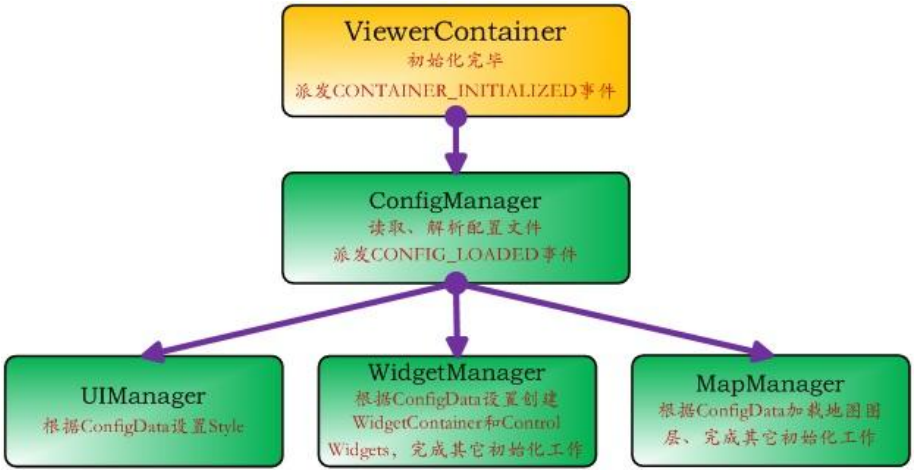


图 3.2 Flex Viewer 初始化过程

3.2 小节中，我们强调通过使用事件，Flex Viewer 将各模块之间充分解耦。实际上，事件也伴随着 Flex Viewer 初始化的整个过程。Flex Viewer 的初始化过程如图 3.2 所示。

1) 首先观察一下 ConfigManager 的构造方法，其中有这样一句代码：

```
ViewerContainer.addEventListener(ViewerContainer.CONTAINER_INITIALIZED, init);
```

也就是说，ConfigManager 实例在构造期间就通过 ViewerContainer 在 EventBus 的单例添加了对 CONTAINER_INITIALIZED 事件的监听，一旦 ViewerContainer 在别的地方派发了 CONTAINER_INITIALIZED 事件，ConfigManager 将响应该事件，开始读取配置文件、解析配置文件，构造 ConfigData 数据，最后将 ConfigData 连同 CONFIG_LOADED 事件派发出去，见下面代码：

```
ViewerContainer.dispatchEvent(new AppEvent(AppEvent.CONFIG_LOADED, configData));
```

2) 我们再观察一下 UIManager、WidgetManager 和 MapManager 的代码，在各自的初始化代码中，都可以找到下面一句代码，此处不再解释：

```
ViewerContainer.addEventListener(AppEvent.CONFIG_LOADED, config);
```

3) 最后，问题的关键在于 CONTAINER_INITIALIZED 事件何时何地被派发？我们观察一下 ViewerContainer 的 init()方法，其中有下面这句代码：

```
ViewerContainer.dispatch(ViewerContainer.CONTAINER_INITIALIZED);
```

而 init()方法是 ViewerContainer 的 creationComplete 事件的响应方法。也就是说 ViewerContainer 初始化结束才是 Flex Viewer 初始化的开始。

Flex Viewer 初始化的脉络已经相当清晰，ConfigManager、UIManager、WidgetManager、MapManager 在各自的初始化过程中对相应的事件进行监听，一旦 ViewerContainer 初始化完成，派发出 CONTAINER_INITIALIZED 事件，其它的准备工作将顺其自然，水到渠成。

我们注意到，初始化过程所涉及的各个模块都是相互透明的，彼此并不知道有对方的存在，而这些模块之所以能够亲密协作，关键就在于它们都只关心事件，事件机制使得这些模块之间实现松耦合。实际上，事件充斥着 Flex Viewer 的各个角落，

随着对 Flex Viewer 理解的逐渐深入，对这一点的体会将更加明显。

3.4 Widget 设计及实现

一个 Widget 是对一组相关操作的封装，这些相关操作完成某项特定业务功能。Widget 不仅可以调用地图资源，也可以访问服务器端资源。Widget 基于 Flex Module，编译后是独立的 swf 文件，这样一来，将具体的业务逻辑封装在 Widget 中，减小了系统主体的体积，在业务功能较多时，可显著缩短系统加载时间。加之面向接口编程和依赖注入思想的应用，Flex Viewer 中的 Widget 可以独立开发，通过配置文件决定提供哪些 Widget，从而决定系统提供哪些业务功能。可以把 Widget 理解为“插件”。这种灵活的“插件”机制是如何实现的呢？下面我们了解一下 Widget 编程模型，如图 3.3 所示。

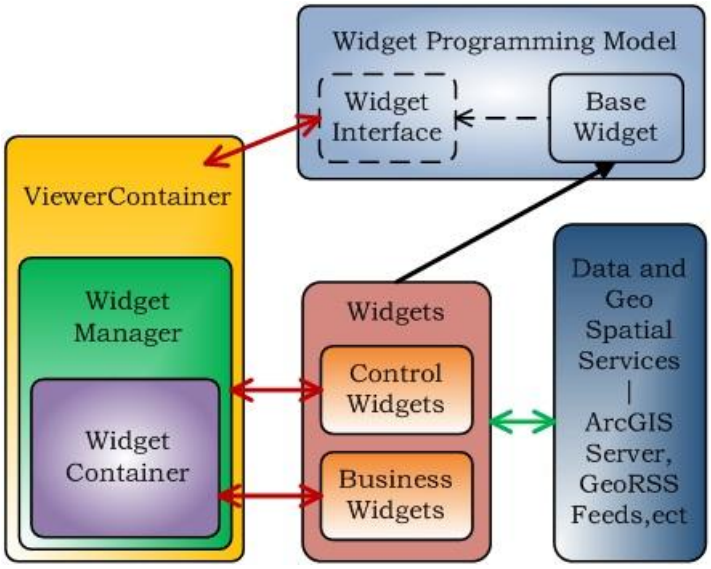


图 3.3 Widget 编程模型

Widget 分为两种，Control Widget 和 Business Widget。Control Widget 是指控制组件，比如 NavigationWidget、HeaderControllerWidget、OverviewMapWidget 等，这些 Widget 提供系统级别的功能，不针对具体业务功能；Business Widget 是指业务组件，比如 SearchWidget、BookmarWidget、GeoRSSWidget 等，这些 Widget 提供具体业务功能。两种 Widget 都继承自 BaseWidget，都由 WidgetManager 来管理，不同的是，在 WidgetManager 中有一个 WidgetContainer 来专门管理 Business Widget。

两种Widget有共同的父类BaseWidget，BaseWidget实现了接口IBaseWidget，两者也就有了共同的接口。WidgetManager 通过 IBaseWidget 来管理 Widget，与具体的 Widget 解耦。Flex Viewer 此处使用面向接口编程和依赖注入，实现了主体系统与 Widget 的松耦合。

- *IBaseWidget*

  该接口定义了 WidgetManager 与 Widget 进行交互的方法，BaseWidget 实现了这个接口。

- *BaseWidget*

  该类是所有 Widget 的基类。只要某一组件继承自 BaseWidget，WidgetManager 即可对其进行管理。由于 BaseWidget 继承自 Module，每一个 Widget 都将编译成独立的 swf 文件。

但两种 Widget 在具体实现上有所不同。Flex Viewer 为 Business Widget 提供了统一的 UI 基类和接口，也就是 WidgetTemplate 和 IWidgetTemplate。通常情况下，Control Widget 都会使用自定义 UI，但是这不是必须的。

- *IWidgetTemplate*

  该接口定义了 BaseWidget 与 WidgetTemplate 交互的方法。WidgetTemplate 实现了这一接口。

- *WidgetTemplate*

  该类是 IWidgetTemplate 的一种默认实现，在具体的 Widget 实现中，同样可以使用自定义的WidgetTemplate，只要实现IWidgetTemplate接口即可。WidgetTemplate 为 Widget 提供基本 UI 控件，包括窗口面板、带有图片按钮的标题栏等。使用 Flex Viewer 实现的 WidgetTemplate，开发人员可以将更多的精力和时间放在实现业务逻辑上。

需要说明的是，Widget 的设计在 Flex Viewer 1.x 版本和 2.x 版本中有所区别。在 1.x 中不存在 Control Widget，比如菜单组件，菜单项是可配置的，但是菜单组件本身是不可配置的。2.x 版本以后，Widget 的概念扩大了，Flex Viewer 中，凡是能看到的组件都是 Widget，这样一来，控制组件可以像业务组件一样可配置，

比如 HeaderControllerWidget，灵活性大大提高。

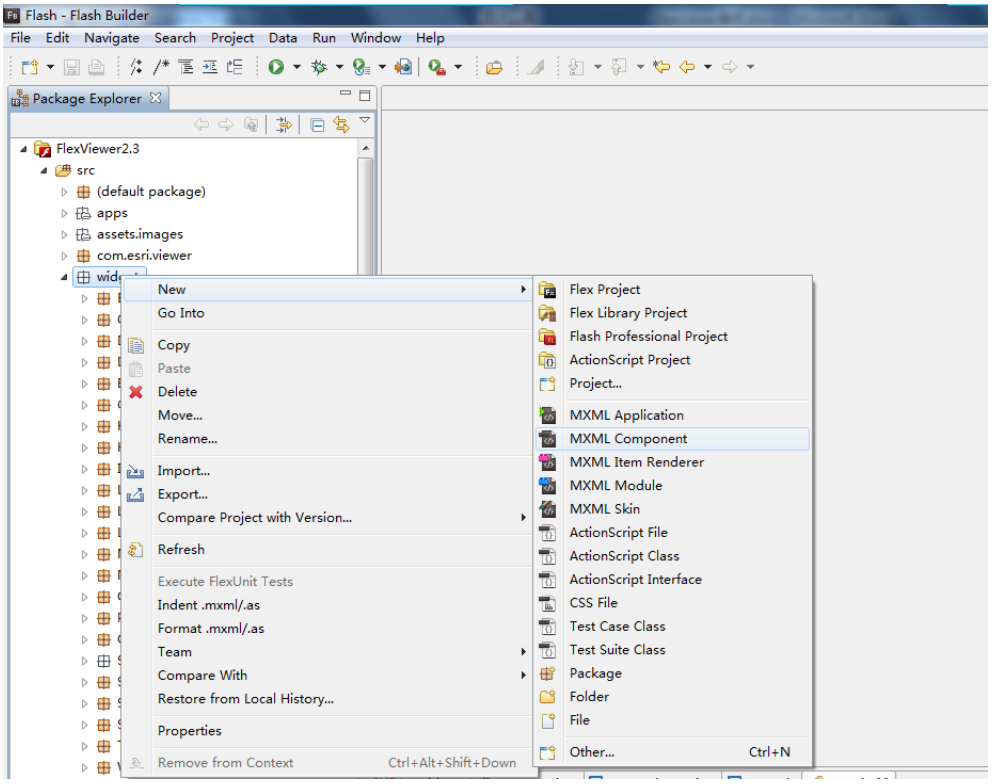
4. 自定义 Widget

终于可以实现第一个 Widget 了，按照惯例，我们通过一个 Hello World Widget 来说明如何在 Flex Viewer 中开发、编译、配置、部署和使用自定义 Widget。Flex Viewer 的源代码中已经包含了一个 HelloWorld Widget，我们还是亲手尝试一下吧。

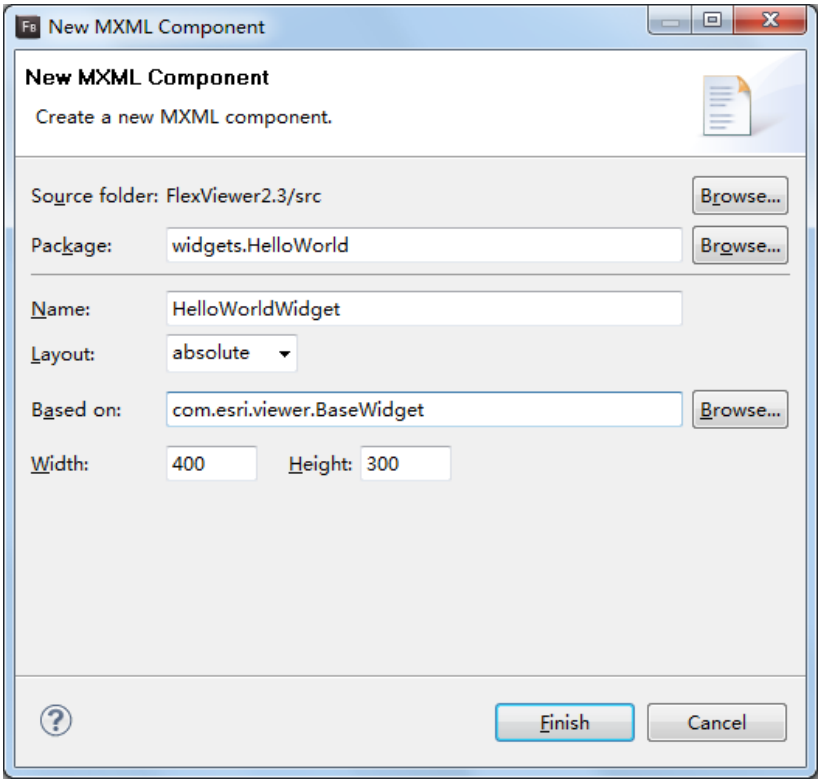
1） 安装 Flash Builder，下载 Flex Viewer 源码，我们使用最新的 2.3；

2） 打开 Flash Builder，导入 Flex Viewer 2.3 的源代码；

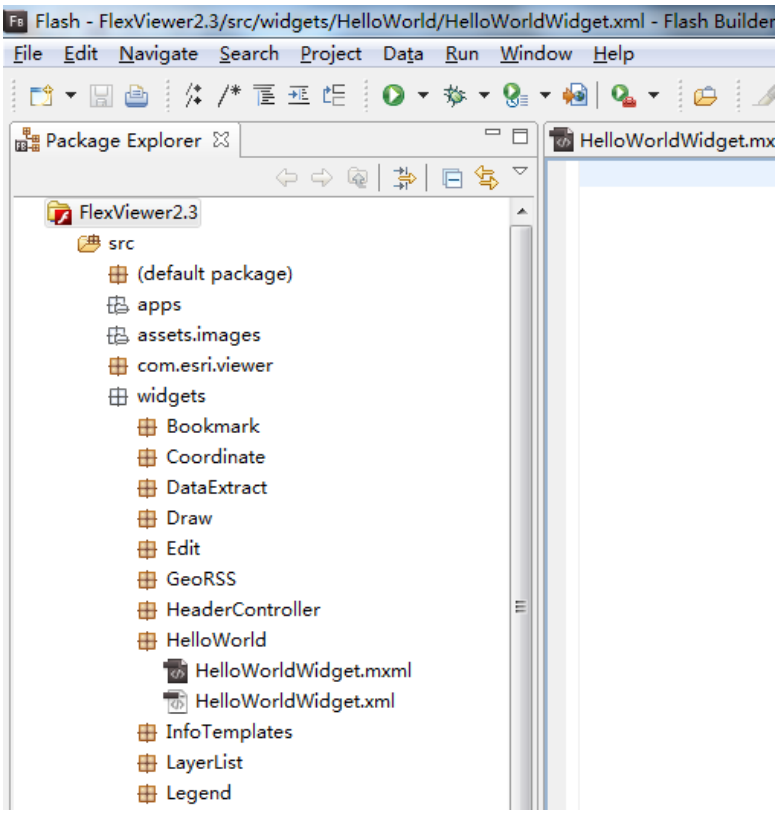
3） 鼠标放在 widgets 包上，单击右键，在弹出的菜单中选择 New，然后单击 MXML Component；



4） 在 New MXML Component 对话框中，输入包名“widgets.HelloWorld”,填写 Widget 名称“HelloWorldWidget”，并选择基类 BaseWidget，单击 Finish；

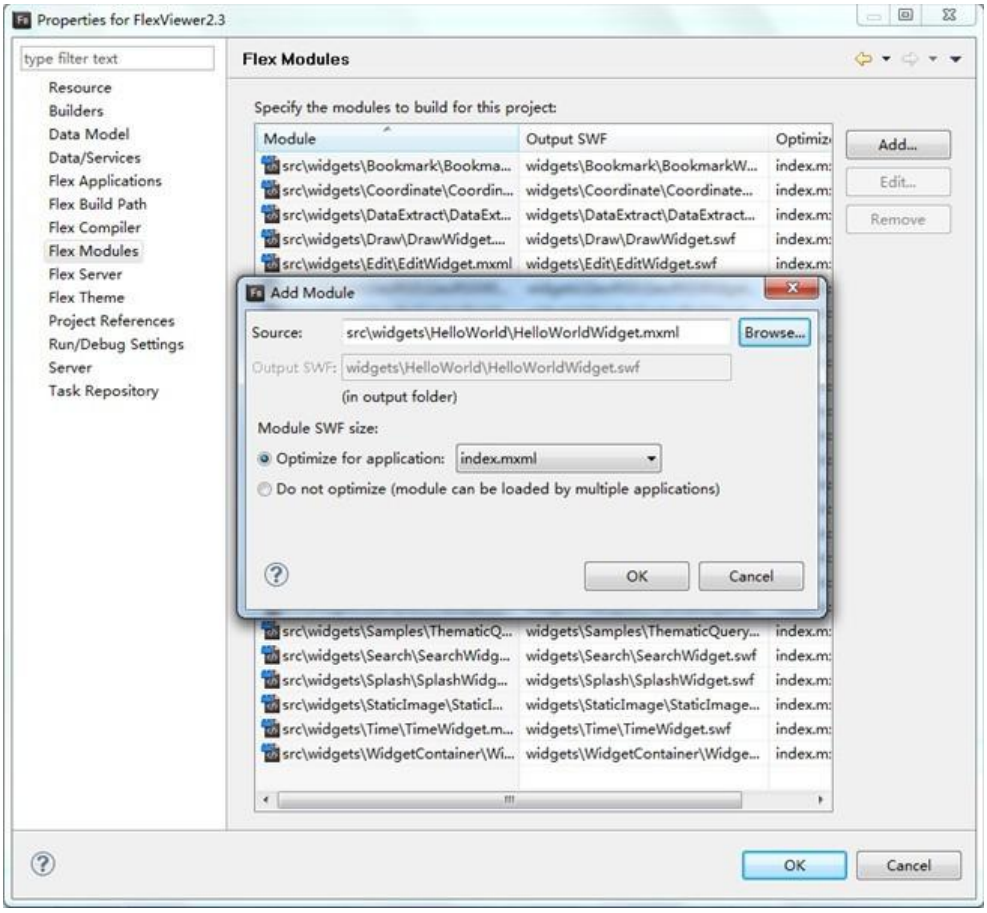


5） 此时，HelloWorldWidget 已经创建完毕，按照 Flex Viewer 提倡的做法，在其包下新建一个同名 xml 配置文件，即“HelloWorldWidget.xml”；

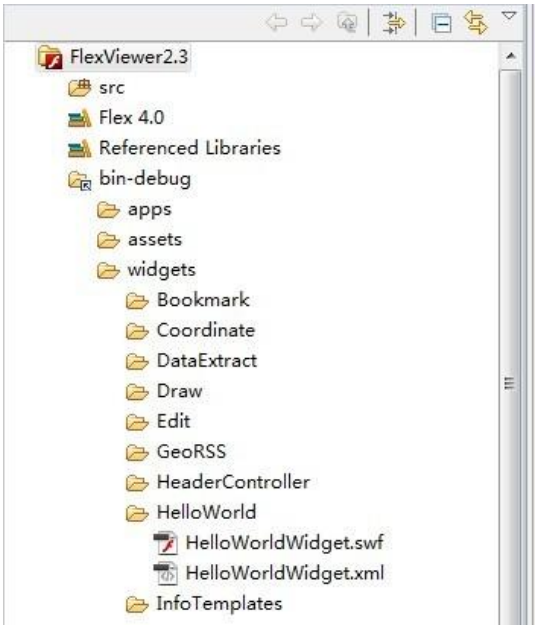


6） 此时，HelloWorldWidget 不会被编译，因为还未把它加入到 Module 列表。打开

工程的属性窗口，点击 Flex Modules，点击 Add 键，将 HelloWorldWidget 加入到 Module 列表中。点击 OK，会发现 HelloWorldWidget 的图标已经和其他的 Widget 一样；



7） 启动编译，编译后会发现在 bin-debug 目录下，HelloWorldWidget 已经被编译成 swf 文件；



8） 下面开始实现 HelloWorldWidget 的业务逻辑，在 Widget 面板中显示出配置文件中的“Hello，Flex Viewer!”字符串。配置文件内容：

```
<?xml version="1.0" ?>
<configuration>
    <hellocontent>Hello, Flex Viewer!</hellocontent>
</configuration>
```

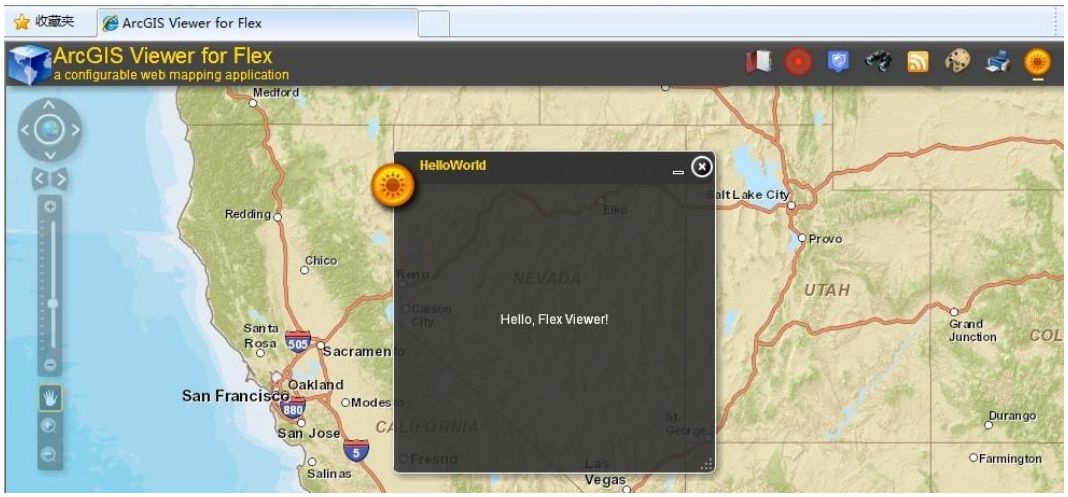
HelloWorldWidget 代码：

```
<?xml version="1.0" encoding="utf-8"?>
<viewer:BaseWidget xmlns:fx="http://ns.adobe.com/mxml/2009"
                   xmlns:s="library://ns.adobe.com/flex/spark"
                   xmlns:mx="library://ns.adobe.com/flex/mx"
                   xmlns:viewer="com.esri.viewer.*"
                   layout="absolute"
                   width="400"
                   height="300"
                   widgetConfigLoaded="init()">
    <fx:Script>
        <![CDATA[
            [Bindable]
            private var helloContent:String;
            private function init():void{
                if (configXML){
                    helloContent=String(configXML.hellocontent);
                }
            }
        ]]>
    </fx:Script>
    <viewer:WidgetTemplate>
        <s:HGroup width="100%"
                  height="100%"
                  horizontalAlign="center"
                  verticalAlign="middle">
            <s:Label text="{helloContent}"/>
        </s:HGroup>
    </viewer:WidgetTemplate>
</viewer:BaseWidget>
```

9） 在 config.xml 对 HelloWorldWidget 进行配置，如下：

```
<widgetcontainer>
   <widget label="HelloWorld" icon="assets/images/i_solar.png"
       config="widgets/HelloWorld/HelloWorldWidget.xml"
       url="widgets/HelloWorld/HelloWorldWidget.swf"/>
</widgetcontainer>
```

10）编译，运行！Hello，Flex Viewer!



5. Widget 之间通信

虽然每个 Widget 都是封装良好的一个组件，提供一组针对特定业务功能的操作，但是有时候需要 Widget 之间的彼此协作来完成一个粒度更大的业务逻辑。此时就需要 Widget 之间的交互，或者说通信。经常看到这样的问题“一个 Widget 如何调用另外一个 Widget 的方法？”。Widget 之间彼此相互独立，互不知晓，“一个 Widget 调用另一个 Widget 的方法”意味着两个 Widget 紧密地耦合在了一起，这不符合“松耦合”的要求。那么 Widget 之间该如何交互呢？答案还是事件！还记得 ViewerContainer 的 addEventListener() 和 dispatchEvent() 方法么？ViewerContainer 通过 EventBus 添加监听和派发事件，使各模块默契协作。

5.1 直接通信

直接通信是指 Widget A 监听某一事件 E，Widget B 在任何时间派发了事件 E，Widget A 都能够捕捉到这个事件，并进行响应。当然，这里对事件的监听和捕捉都是通过 ViewerContainer 代理的 EventBus 实现的，参见图 5.1。我们注意到，Widget A 与 Widget B 除了通过全局的 EventBus 监听、派发、捕捉事件外，没有发生任何直接的联系。

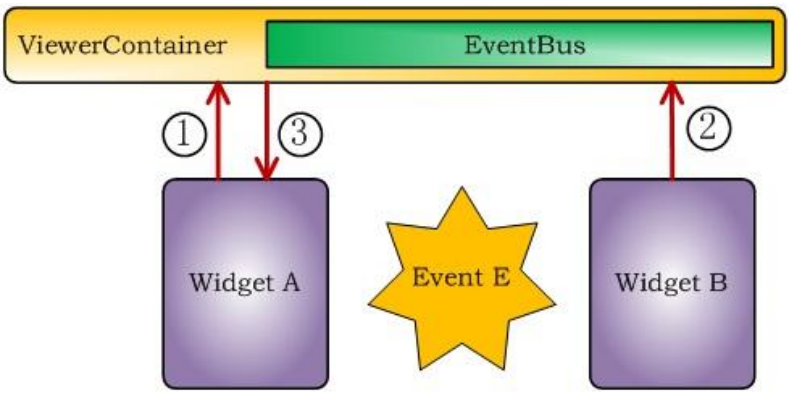


图 5.1 Widget 之间直接通信

① Widget A 通过 EventBus 监听*事件 E*;

② Widget B 通过 EventBus 派发*事件 E*;

③ Widget A 通过 EventBus 捕捉*事件 E*，并进行响应。

5.2 间接通信

直接通信有一个前提条件，那就是必须交互的两个 Widget 都已经被加载。Widget 采用 lazy-load 的加载方式，也就是只有第一次打开的时候才会加载。如果 Widget B 已经加载并派发了事件 E，而此时 Widget A 还并未加载，Widget A 是无法捕捉并响应事件 E 的，因为 Widget A 在内存中还不存在。

此时的解决方法是通过 DataManager 间接通信。DataManager 响应 Widget B 派发的事件，并数据存储起来，Widget A 在第一次加载时通过 DataManager 获取 Widget B 共享的数据，以此可以达到通信的目的，参见图 5.2。同样的，Widget A 与 Widget B，以及两者与 DataManager 之间并未直接交互，所有的协作都通过事件来完成。

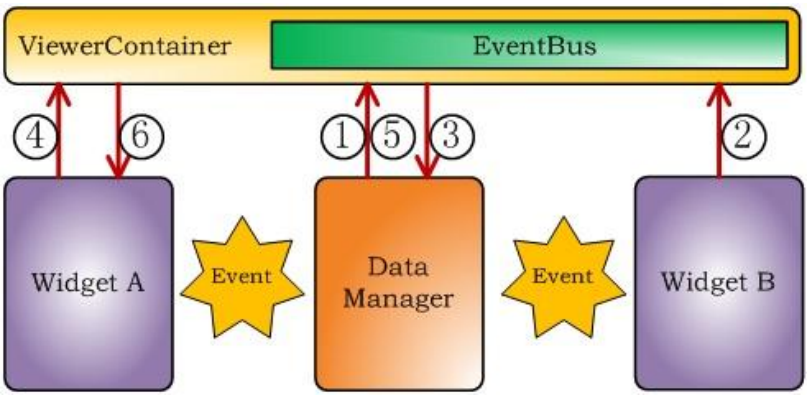


图 5.2 Widget 之间间接通信

① DataManager 通过 EventBus 监听*共享数据事件*和*获取数据事件*;

② Widget B 派发*共享数据事件*;

③ DataManager 响应*共享数据事件*，将数据以 key-value 的形式保存;

④ Widget A 监听*分发数据事件*，并且派发*获取数据事件*;

⑤ DataManager 响应*获取数据事件*，查询 Widget A 所需数据，派发*分发数据事件*;

⑥ Widget A 响应*分发数据事件*，并做相应的响应动作。

DataManager 是 Flex Viewer 为模块之间共享数据和保存共享数据设计的一种解决方案。DataManager 将各个模块需要共享出来的数据存储在内存中，一旦某个模块需要获取共享数据，派发一个消息即可，DataManager 会响应这个消息，并共享数据作为消息的载体派发出去。

DataManager 响应的事件

- `DATA_FETCH_ALL`: 获取数据事件（全部数据）
- `DATA_PUBLISH`: 共享数据事件
- `DATA_FETCH`: 获取数据事件（根据 key 获取数据）

DataManager 派发的事件

- `DATA_SENT`: 分发数据事件
- `DATA_NEW_PUBLISHED`: 当有新的数据被共享时，将新数据分发出去

6. 让 Flex Viewer 为我所用

Flex Viewer 框架的设计者说“Flex Viewer 就是为了让人能将其大卸八块，然后组装新意，GIS 是可以玩的。”Flex Viewer 可以帮助我们快速开发系统原型，并慢慢演变成为成熟的系统。但或许，Flex Viewer 带给我们的最重要的东西不是 Flex Viewer 本身的代码，而是隐藏在纷繁复杂，让人昏昏欲睡的代码中的架构思想。这种思想值得我们借鉴、扩展、应用，甚至是推翻，然后重新设计更适合具体应用场景的 Viewer，让 Flex Viewer 真正为我所用。

此处，列举两个拓展 Flex Viewer 的示例。

◆ 示例一

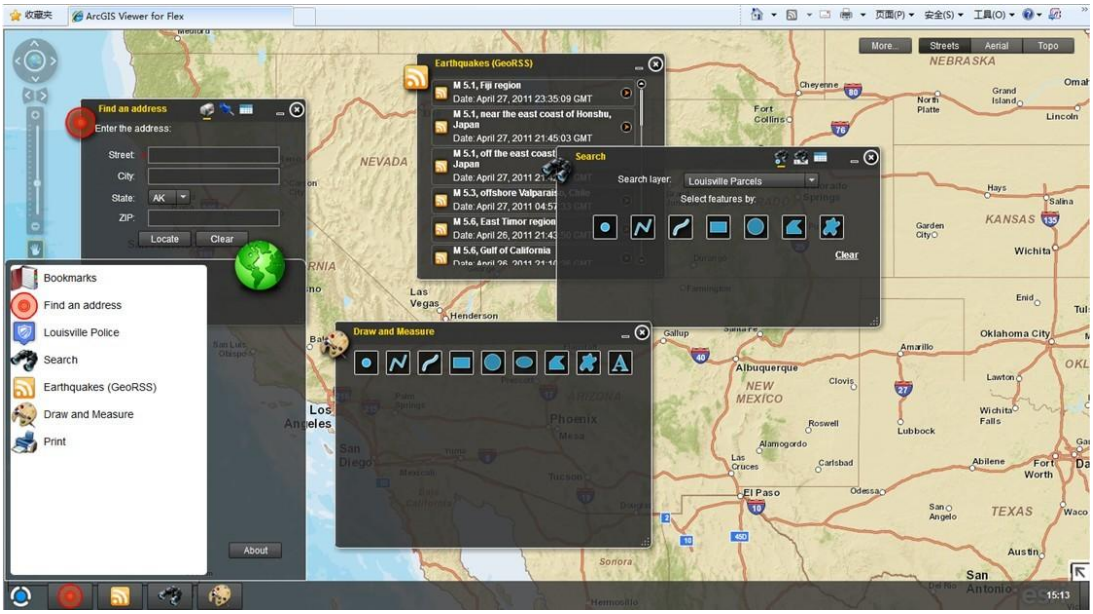
第一个示例很简单，简单到只有一行代码。

进入 Flex Viewer，打开两个 Widget，将一个 Widget 拖拽至另一个 Widget 的上方，部分重叠。用鼠标在处于下方的 Widget 上按下，你会发现没有任何反应。这跟我们平时的体验是不一样的，我们更愿意看到鼠标按下时，处于下方的 Widget 能神奇的移动到所有的 Widget 的上面。

实现这一点很简单，打开 WidgetTemplate.as，找到 mouse_downHandler()方法，修改如下：

```
public function mouse_downHandler(event:MouseEvent):void{
    // focus this widget when mouse down
    ViewerContainer.dispatchEvent(new
         AppEvent(AppEvent.WIDGET_FOCUS, widgetId));
    // end
    if (_draggable && enableDraging){
         header.addEventListener(MouseEvent.MOUSE_MOVE,
             mouse_moveHandler);
         widgetFrame.addEventListener(MouseEvent.MOUSE_MOVE,
             mouse_moveHandler);
        }
}
```

◆ 示例二



第二个示例是在 Flex Viewer 的基础上，实现类似于 Windows 风格的用户体验，如图 6.1 所示。源代码会与本文档一并提供下载。

7. 后记

Flex Viewer 已被广泛应用，但至今尚没有一个中文版的材料，希望本文档能够给使用 Flex Viewer 的 GISer 们提供一些启发和帮助。文档没有针对具体的代码细节进行讲解，而是试图讲清楚 Flex Viewer 的架构思想，虽然很可能没有讲清楚:)。本文档可能对那些了解 ArcGIS Flex API，有一些 Flex 基础，阅读了一定量的 Flex Viewer 源码，但还没有理清其思路的 GISer 比较有帮助。对于刚刚接触 Flex Viewer 的 GISer，建议先尝试读一读 Flex Viewer 的代码，尝试开发几个 Widget，先了解清楚其事件机制。

如发现文档中存在错误或不妥之处请发送邮件至 fromirsa@gmail.com，文档会及时更新和修正。