

FlexBev 开发指导手册

作者：Flex 开发团队

公司：超图软件

时间：2012年7月

- 1. FlexBev2 概览..... 3**
 - 1.1 总体介绍..... 3
 - 1.2 界面预览..... 3
- 2. 架构设计解析 6**
 - 2.1 FlexBevLib 架构..... 6
 - 2.2 FlexBev2 UI 框架..... 7
- 3. 配置文件说明 8**
- 4. 面板与插件 10**
 - 4.1 面板初始化..... 11
 - 4.2 标签页操作..... 12
 - 4.3 面板停靠..... 14
 - 4.4 面板关闭..... 15
 - 4.5 插件通信机制 15
- 5. FlexBev2 查询功能剖析..... 16**
 - 5.1 注册事件类型到命令 17
 - 5.2 注入组件..... 17
 - 5.3 查询实现..... 18
 - 5.4 其他操作..... 18

1. FlexBev2 概览

本章将使您快速了解 Flex Bev 项目工程，界面组织结构以及快速搭建运行步骤。

1.1 总体介绍

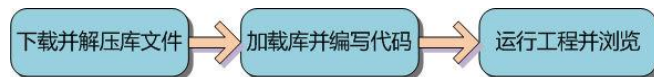
Flex Bev 是基于 SuperMap iClient for Flex 产品开发的一套支持用户自定义扩展功能集成显示框架，该框架旨在具备可定制，可装卸，配置方便，界面美观等系列特性，主要面向二次开发用户以及快速制作原型系统的人员。

Flex Bev 下载：

<http://support.supermap.com.cn/ProductCenter/DevelopCenterRefactoring/SuperMapiClientFlex.aspx>

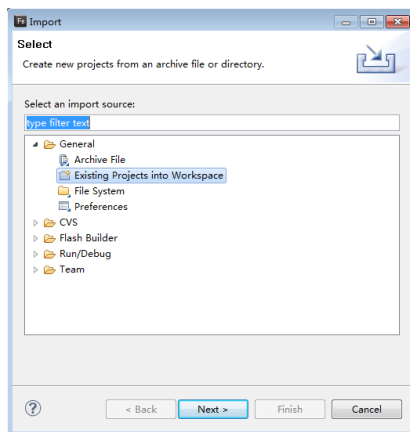
1.2 界面预览

首先，配置流程如下：

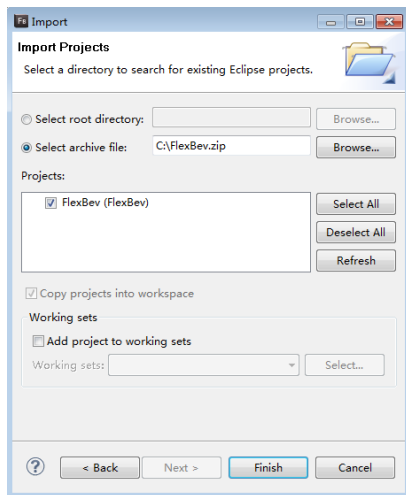


第一步：下载 Flex Bev 源代码工程包（下载地址参见1.1）。

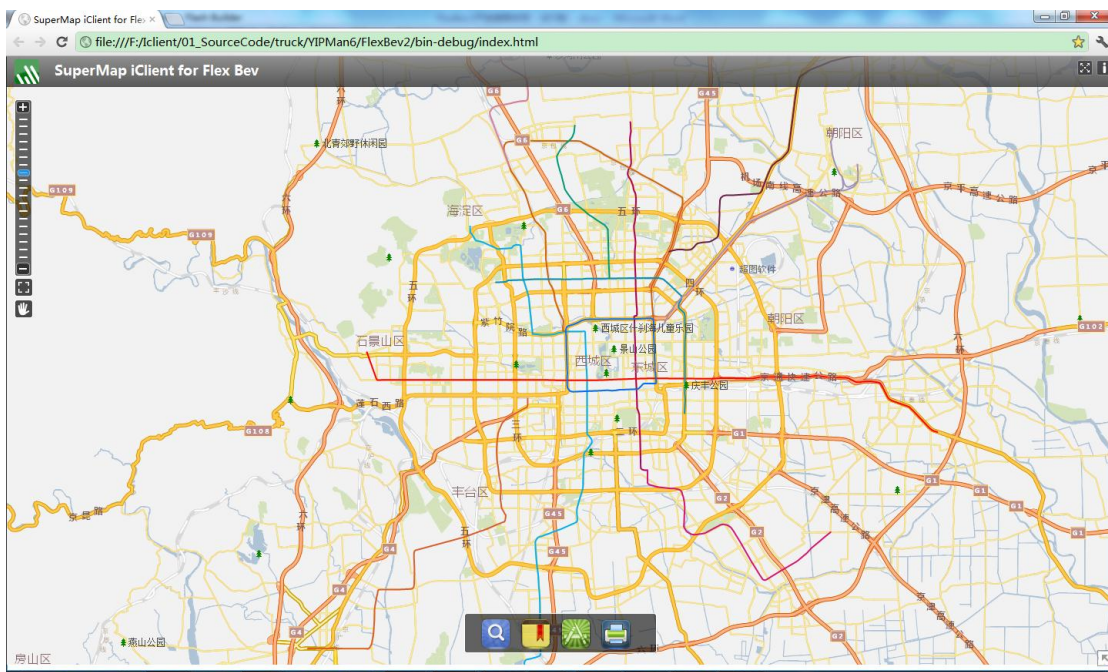
第二步：在 Flash Builder 4.6 中，进入**包资源管理器**界面，空白处右键选择“**import**”，如图：



双击“Existing Projects into Workspace”，然后选择“Select archive file”对应的“Browse...”按钮：



点击 Finish 完成项目添加。直接运行即可看到如下界面：



其次，下图标示了主界面上各个部件的位置以及名称：



从上图可知主界面主要有如下**五部分**组成：

标题栏：显示基本信息。该组件相关代码文件为：

com\supermap\skins\customSkins\HeaderbarSkin.mxml

导航条：左侧上方的地图缩放条与下面的全幅平移按钮。 该组件相关代码文件为：

com\supermap\gears\navigator\NavigationGear.mxml

功能管理组件：位于图示的正下方，该面板负责面板的初始化工作，点击即可创建对应的面板。该组件相关代码文件为：

com\supermap\containers\component\PanelConfigBar.as

浮动面板：如上图中的“要素检索”功能面板。该 面板允许停靠多个标签页，每个标签页对应一个插件。

有关该部分的详细解说，请看第三部分——面板与插件。该组件相关代码文件为：

com\supermap\gears\query\QueryContainer.mxml

停靠区域：上图中右上方的停靠位置。该部分是点击浮动面板右上方的停靠按钮触发显示的，一个停靠条对应一个浮动面板。

该组件相关代码文件在 FlexBevLib 库中，主要是 com\supermap\framework\dock\supportClasses\DockBarMananger.as

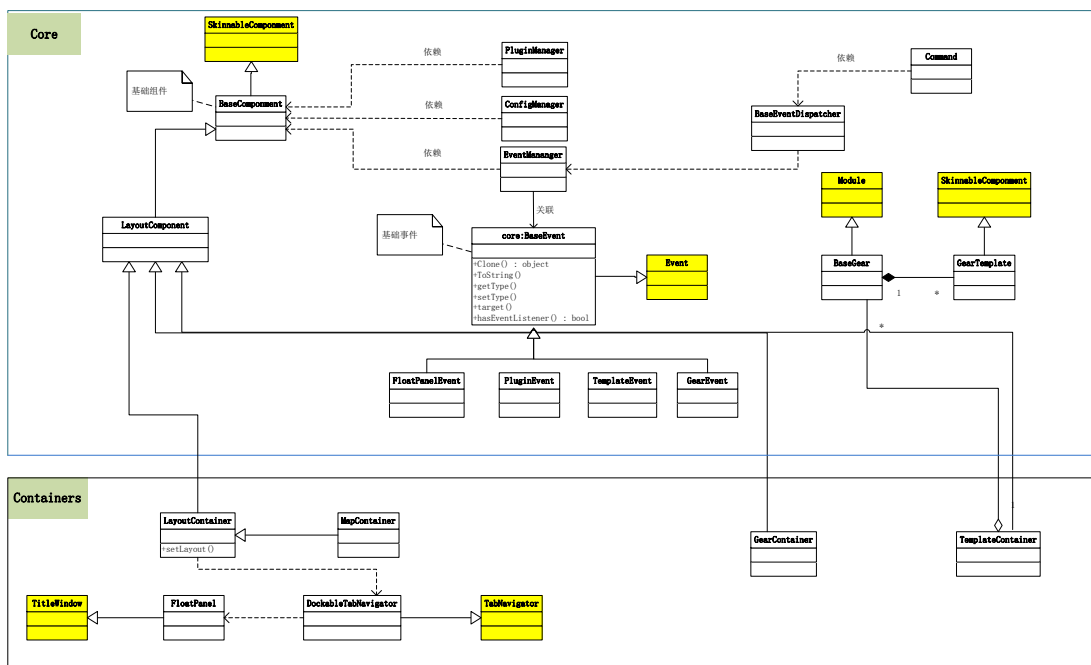
2. 架构设计解析

在初步了解 FlexBev2 界面结构后，我们来了解下 FlexBevLib 库的架构图，在熟悉 FlexBevLib 架构前提下能够提升大家对 FlexBev2 的理解程度，以便深入扩展定制，更好的满足需求。

这里涉及两个图示，前者为 FlexBevLib 的架构图，后者为 FlexBevLib 与 FlexBev2 UI 框架的扩展机制简图，这两个图有机结合就能够提供一套灵活易用的插件扩展开发机制。

2.1 FlexBevLib 架构

FlexBevLib架构图



该图主要分两部分，一是核心 core 部分，另一个是容器 Containers 部分。

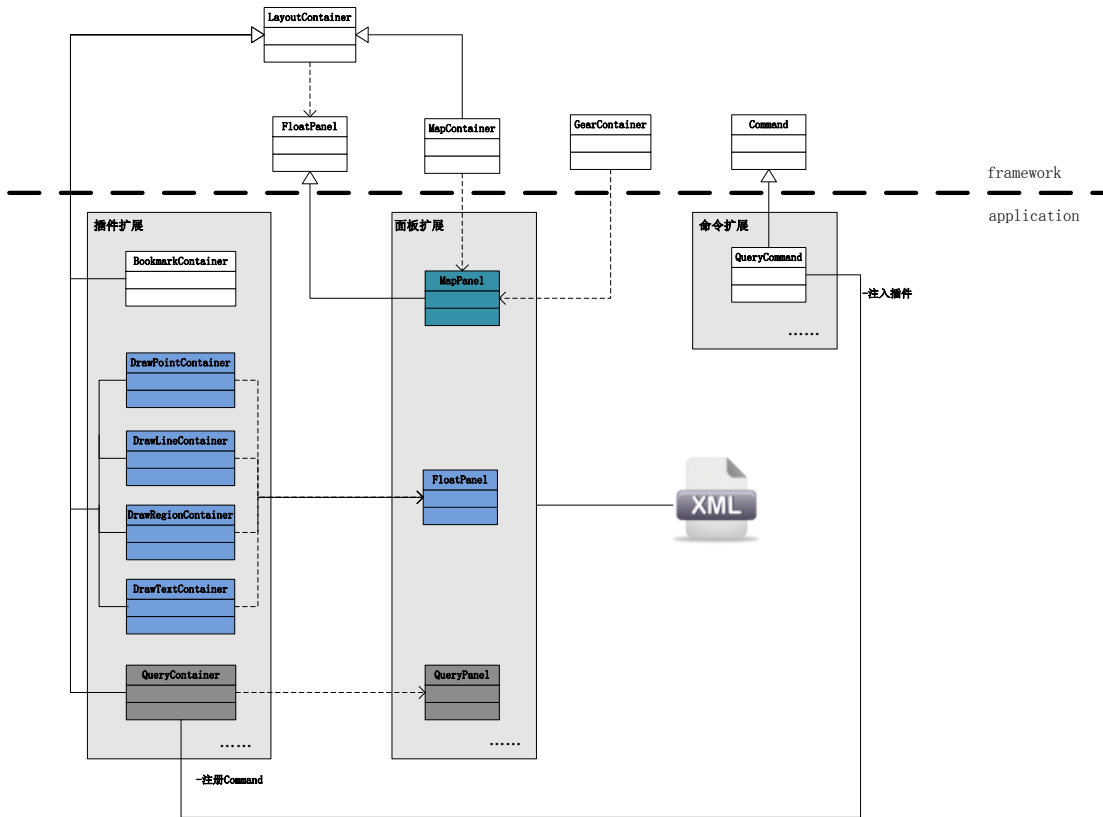
Core 部分主要涵盖了两块：基础组件 `BaseComponent`，基础模块 `BaseGear`。前者是整个 `FlexBevLib` 的框架组件基类，在它基础之上会封装到容器组件部分，最终到 `FlexBev2 UI` 框架里覆写为自己的插件类型。

基础组件本身支持了事件总线机制与依赖注入机制。即在子类里你可以直接取到 `EventBus` 单一实例，该实例最终其实是指向了一个基础事件派发器（`BaseEventDispatcher`），通过该事件总线，你可以在各个插件之间完成各种通信与数据传递。

而注入是与事件总线结合使用的一个命令机制，事件总线可以注册事件类型到命令（command），当事件类型派发时，事件流便会到达 command 的回调函数里去，而在 command 里我们可以使用 Inject 来注入各种插件，这也就实现了插件的通信与数据传递。

Containers 层，则是在基础组件基础上覆写下来的基础容器类，这里需要注意 LayoutContainer 类。更详细的说明请结合第四部分---查询功能解析来深入理解运用。

2.2 FlexBev2 UI 框架



该图则展现了 FlexBevLib 与 FlexBev2 ui 框架的内在联系与扩展机制。

插件方面都是继承自 LayoutContainer，如查询插件 QueryContainer (com\supermap\gears\query\QueryContainer.mxml)。

注意：框架里的 LayoutContainer 不是 Flex SDK 里的 LayoutContainer，前者是 com\supermap\containers\LayoutContainer.as 后者是 mx.Core.LayoutContainer。

而面板则是继承自 FloatPanel 面板基类，对面板的使用如果不涉及皮肤样式的修改这个类基本上是不需覆写的，能满足大多需求。

然后通过配置把面板与插件联系起来即可运行整个程序了。

另外 `command` 是必须要覆写的，上图举了一个例子，`QueryCommand` 里可以通过 `Inject` 标签页实现对 `QueryContainer` 的注入。
具体解析请参见第四部分---查询功能解析。

3. 配置文件说明

打开 `FlexBev2` 工程项目源码，在 `src` 根目录下有 `Application-config.xml` 配置文件，该配置文件是整个工程功能配置的核心，下面我们逐一看看其节点含义及其使用注意事项：

1 `Panel` 节点：

- `Id`: 面板的唯一标识。必设属性。
- `X`: 面板显示的初始X位置。可设置。
- `Y`: 面板显示的初始Y位置。可设置。
- `Width`: 面板默认显示宽度。
- `Height`: 面板默认显示高度。
- `selectedIndex`: 标签页的显示索引。即面板打开时优先显示的标签页索引位置。
- `isDragAble`: 是否可拖拽。默认为`true`。
- `isResizeAble`: 是否可缩放。默认为`true`。
- `maxTabNum`: 允许停放的最大标签页数量。
- `Name`: 面板的完全限定类名。
- `iconBar`: 面板的图标。用在面板管理组件上。
- `Describe`: 面板的描述性说明。用在面板管理组件上。
- `Title`: 面板的标题。用在停靠时候的标题显示。

2 `Plugin`

- `Id`: 插件的唯一标识，必设属性。
- `Name`: 插件的完全限定类名。
- `iconBar`: 停靠时候显示图标。
- `Label`: 名称。停靠时候的悬停提示内容。
- `Describe`: 当插件本身拖拽出去重新生成一个新的面板的时候，该描述作为新面板的描述内容。

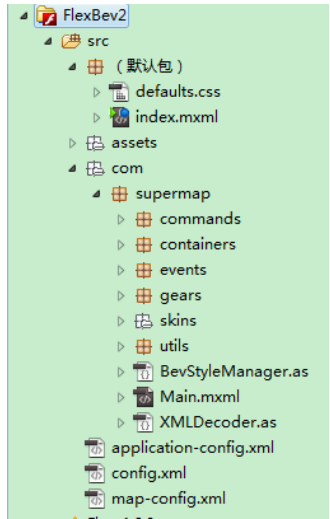
3 `GearContainer`

- `Id`: 插件的唯一标识，必设属性。
- `Left`: 距离父对象左侧的长度。
- `Top`: 距离父对象顶部的长度。
- `url`: `Gear` 功能模块的 `url` 地址。

4 `PanelManangerBar`

- Id:**面板管理组件的 id。
- Left:**距离舞台左侧的长度。
- Bottom:**距离舞台底部的长度。
- Name:**完全限定类名。

接下来是 FlexBev2 工程的文件列表，如下图:



Defaluts.css 是全局样式文件。

Index.mxml 是整个工程的启动文件。

Assets 文件夹是图片资源文件。

Com.supermap.commands 包是命令包，里面放置的是各种功能逻辑的 **command**。如 **QueryCommand** 便是继承自 **Command** 类(该类在 **flexbeplib** 库中)。

Com.supermap.containers 包是组件包，里面放置了各种自定义的组件，主要的界面 **ui** 组件。

Com.supermap.events 包是事件包。

Com.supermap.gears 包是插件包，里面包含了各种功能组件，这些组件都是以插件形式存在于对应的面板容器里。

Com.supermap.skins 包是皮肤包，里面包含了各种皮肤 **skin**。

Com.supermap.utils 包是工具包，里面包含了两个类: **PanelUtil** 与 **ReflectUtil**。前者提供面板置顶显示的全局静态方法，后者提供对工程启动时候需要反射的类的配置方法。

BevStyleMananger.as 类处理 **ui** 框架的样式，设置通用组件的统一外观。

Main.mxml 是主界面组件，负责获取 **Application-config.xml** 配置文件里的信息并派发出去。主要在面板管理组件 (**com.supermap.containers.component.PanelConfigBar**)里处理。

XMLdecoder.as 类是用来解析 **Application-config.xml** 配置文件，并组织数据派发出去。主要在 **Main.mxml** 里进行监听处理。

提示:在 Gears 包里扩展功能模块，只需继承自 LayoutContainer(FlexbeveLib 中的类)写自己的组件即可，然后在 Application-config.xml 配置文件里按照 plugin 节点的方式配置到 panel 面板节点里即可。

4. 面板与插件

了解上一部分的配置文件节点信息后，下面来看常用的面板扩展配置方式:

1. 带有标签页的面板配置
2. 没有标签页面板配置

第一种의常用配置为:

```
<panel id="printPanel" title="打印"

    name="com.supermap.framework.dock.FloatPanel"

    iconBar="assets/panel/print.png"

    height="510" width="1050" x="10" y="10"

    describe="打印">

    <plugin id="printContainer"

        name="com.supermap.gears.print.PrintContainer"

        label="打印"

        icon="assets/bank.png"

        iconBar="assets/panel/bookmark.png"

        describe="打印面板"

    />

</panel>
```

在配置文件根节点下添加一个 panel 节点，然后里面放置一个或者多个 plugin 节点即可。

其呈现方式如下:



这里只有一个面板，它包含了一个“打印”标签页，其他是该标签页里的内容展示。

第二种常用配置为:

```
<panel id="lanel" icon="assets/accordion/style.png"
      iconBar="assets/panel/bank.png"
      title="面板"
      describe="面板示例"
      height="120" width="230" x="20" y="550"
      isDock="true"/>
```

这里直接添加一个 panel 节点即可。其呈现方式如下:



4.1 面板初始化

配置文件里配置的若干面板，是通过主界面上的面板管理组件(PanelConfigBar.as)进行管理的，如下:



点击各自的的面板图标便可打开对应的面板，该部分逻辑在PanelConfigBar.as里。

这里只负责面板的初始化工作，这个要区别于停靠状态。停靠状态是保留了面板的即时状态的，类似于最小化的状态，只是这

里把最小化和布局两者整合在一起，称为停靠。当且仅当面板关闭后，面板管理组件点击才会初始化一个新的面板。
还要注意的是，由于面板标签页是可拖拽的，因此面板拖拽后，相当于面板数据变化，这个时候点击面板管理组件，也是可创建新的面板的，这并不冲突。

注意:默认的面板FloatPanel使用的是com.supermap.framework.skins.FloatPanelSkin;
不带标签页的面板使用的则是com.supermap.framework.skins.FloatPanelNoTabSkin。

4.2 标签页操作

为了更好的管理各种业务逻辑窗口，框架本身提供了一个基于标签页管理的综合管理面板。该面板上可停靠用户自定义数量的标签页，一个标签页对应一个业务功能容器，同时标签页支持拖拽停放，可灵活的拆分与组装。类似于Chrome(谷歌浏览器)上面的标签页操作。



1 标签页的拖拽

主要有如下几种情况:

鼠标放在某个标签页上直接拖拽出该标签页的显示范围。这里鼠标有两种状态，如图:



这个时候释放鼠标的话，会创建一个新的面板，如下图所示：



同理，当我们拖拽该标签页到整个面板的区域外部的时候，也会生出一个新的面板。

2 标签页上悬停

为了更好的说明操作的流程，这里直接拖“线标绘”这个标签页，拖拽到“面标绘”(也可以是其他任何一个标签页)上方，那么“面标绘”标签页就会检测到“线标绘”标签页，然后根据在其上的具体位置(靠左边还是右边)来进行标签页的嵌入。




当拖拽的标签页位于面板顶部其他位置时候，会动态监测位置并默认提示在最后一个标签页的位置上，如果此时释放鼠标则会添加到最后一个标签页上。如下图：



以上就是最常用的标签页交互方式了，这里还有其他的一些细节处理，用户自己可以多做尝试，以便更好的管理面板。

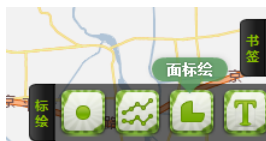
4.3 面板停靠



通过点击图片实现该功能，用户可以在保留面板当前数据状态(如坐标位置，输入信息等)的同时以一个图片的显示方式悬停在舞台（可以理解为浏览器内容窗口范围）的右侧，当然点击该图片即可还原面板之前的数据状态。



当鼠标滑过的时候，会显示停靠面板里的所有的标签页信息，如下图：




这里的图标对应着面板里的标签页索引位置，因为当点击不同的图标就会在展开面板后显示该索引位置的标签页，如这里点击第三个图标（面标绘提示信息），即可看到如下面板：



当面板打开时，右侧的停靠栏就移除了。

4.4 面板关闭



当点击图标，框架内部会判断当前面板的标签页数目，如果多于一个，那么会弹出如下窗口提示：



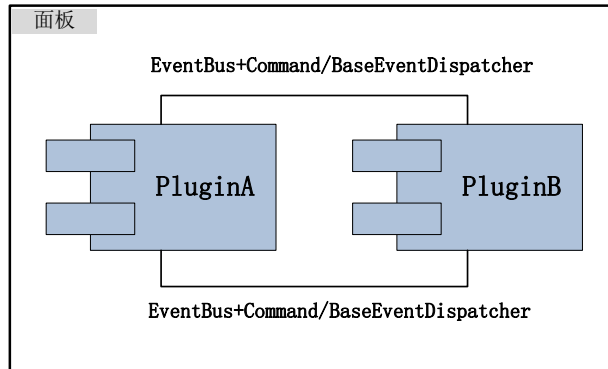
点击“关闭”则关闭当前面板，点击取消或者提示窗口的关闭图标会直接关闭这个模式的提示窗口，当前面板不受影响。

如果当前面板标签页只有一个，那么就直接关闭该面板。

注意：面板的添加与移除都是在舞台上操作的。对应的操作对象都是FlexGlobals.topLevelApplication。

4.5 插件通信机制

在同一个面板中或者不同的面板之间，插件之间的通信如下图所示：



这里主要有两种通信方式:

通过事件派发器BaseEventDispatcher:

这种方式类似于Flex SDK里处理方式, 自定义事件派发出去, 事件派发器通过getInstance方法取到一个单例, 然后在回调里就可以完成数据的获取或者分发了。

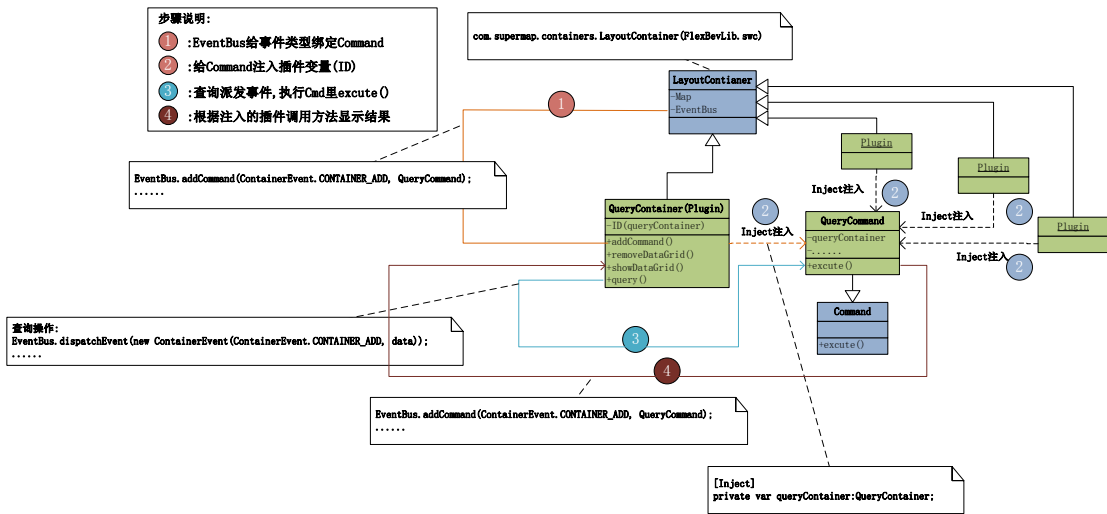
通过命令与注入的方式:

该主要是利用事件总线来进行事件类型到命令的绑定, 一旦交互或者触发事件, 那么就会到命令的回调函数里, 由于命令具有Inject标签注入的特性, 这里需要什么组件接受数据, 只要注入进来即可, 剩下的就是在command里的回调函数excute里进行数据的传递了。

综合比较上两种通信方式, 如果是数据的分发涉及到的组件不多, 仅需要一两个组件之间进行通信, 那么第一种通信方式即可满足需求, 如果是数据分发到各个组件, 涉及的数据调度流程比较复杂, 推荐使用第二种通信方式, 这种方式能很好的降低UI组件与逻辑的耦合度, 易于系统级别的维护。

5. FlexBev2 查询功能剖析

下图就是查询功能描述图:



从上图我们来深入剖析该功能的实现细节:

首先:LayoutContainer与Command类是FlexbevLib库里的类, 前者是插件容器基类, 本身封装了Map属性与EventBus事件总线。

其次:功能实现类主要有两个, 一个是QueryContainer组件, 一个是QueryCommand类。

最后:请留意图示左上方的步骤说明。

好, 接下来, 来看第一步:

5.1 注册事件类型到命令

在QueryContainer组件(继承自LayoutContainer)的创建完成回调函数(CreationComplete)里添加了command绑定, 代码如下:

```
protected function bordercontainer1_creationCompleteHandler(event:FlexEvent):void
{
    EventBus.addCommand(ContainerEvent.CONTAINER_ADD, QueryCommand);
    EventBus.addCommand(ContainerEvent.CONTAINER_DELETE, QueryCommand);
    EventBus.addCommand(QueryPanelEvent.QUERY_PANEL_REMOVE, QueryCommand);
}
```

这三行代码实现了类似自定义事件添加监听的功能, 只不过这里没有直接写明监听回调, 而是通过框架把回调注册到了QueryCommand命令里的excute方法里了。也就是说主要其他地方派发该事件类型, 如EventBus.dispatchEvent(new ContainerEvent(ContainerEvent.CONTAINER_ADD, data)), 那么程序就会执行到excute方法体里去。

5.2 注入组件

当自定义组件添加对事件绑定之后, 需要到对应的command里注入相关组件, 当然也可以注入其他组件, 该查询QueryCommand注入组件如下:

```
[Inject]
public var qContainer:QueryContainer;
```

注意:这里一定要设置组件的 id, 把组件 id 作为变量的名称, 否则注入不起作用。

这样是可以在 excute 方法体里使用该组件了, 框架内部已经做了 command 对事件类型绑定的映射, 因此这里可以注入任何已经添加到舞台上的组件唯一 id。本例我们添加了查询组件 QueryContainer 的 id--- qContainer。

5.3 查询实现

查询面板的 UI 呈现如下图:



该面板是 QueryPanel, 内部放置了一个插件—QueryContainer 组件, 就是查询功能组件。那么我们点击“学校”后, 在 QueryContainer 里可知派发了如下事件:

```
EventBus.dispatchEvent(new ContainerEvent(ContainerEvent.CONTAINER_ADD, tabAddedTitle, recordGrid ));
```

根据上面第一步所述, 该事件类型已经绑定到了 QueryCommand, 那么此时程序会到 QueryCommand 里的 excute 方法体里:

```
override public function execute(event:BaseEvent):void
{
    if(event.type == ContainerEvent.CONTAINER_ADD)
    {
        qContainer.showDataGridHandler(event as ContainerEvent);
    }
    else if(event.type == QueryPanelEvent.QUERY_PANEL_REMOVE)
    {
        qContainer.queryPanelRemoveHandler(event as QueryPanelEvent);
        qContainer.deleteAllDataGridHandler();
    }
    else if(event.type == ContainerEvent.CONTAINER_DELETE)
        qContainer.deleteDataGridHandler(event as ContainerEvent);
}
```

由于 QueryContainer 已经通过 Inject 标签注入的方式注入进来, 那么就直接执行 showDataGridHandler 方法了。

直接在 QueryContainer 里监听派发该事件不也行!但是通过 Inject+command 的方式处理事件流, 会使得整个回调流程更加清晰, 这里只是注入了查询组件, 如需把查询的数据传递给其他组件只需要注入组件 id, 这种依赖注入的方式避免了一个组件里开多个接口链接到多个组件, 同时事件流程更加清晰明了, 便于维护扩展。

5.4 其他操作

通过“学校”复选框取消勾选后关闭查询结果。

这里会派发一个事件:

```
EventBus.dispatchEvent(new ContainerEvent(ContainerEvent.CONTAINER_DELETE, tabTitle));
```

之后执行 excute 方法里 qContainer.deleteDataGridHandler 方法。

总结:查询功能本身不仅仅包含了组件本身以及查询逻辑，也包含了附带的查询结果的展示与清除的功能，这些都是查询功能的展示范畴，而这些分隔开来的方法体通过Inject+Command的方式来区分事件类型进行调用，就很好的把逻辑与呈现解耦开来。