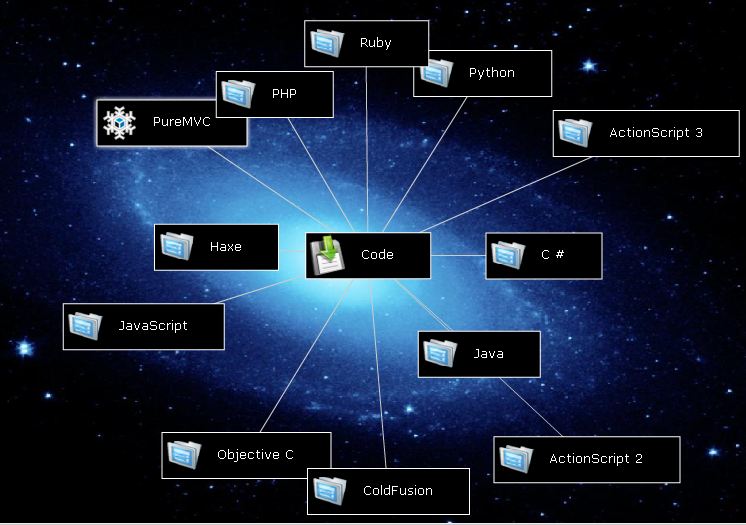
**[puremvc框架之hello world!](http://www.cnblogs.com/yjmyzz/archive/2010/07/27/1786123.html)**

[puremvc](http://www.puremvc.org/)是一个可应用于多种语言、多种平台的mvc框架。根据[官网](http://www.puremvc.org/" \t "_blank)上的描述，现在已经支持下列语言：



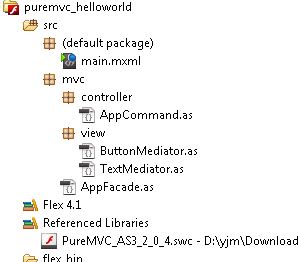
官方也推出了[最佳实践的中文文档](http://bbs.9ria.com/thread-58666-1-1.html)，当然，园子里也有兄弟说它烂的 :)

跟asp.net mvc框架有所不同，在asp.net mvc中，一个http请求过来，controller会自动去取得数据，最终转化为model，然后选取一个view进行呈现，同时把model传到view中，一切还算比较简单。

然而puremvc除mvc模式外，渗杂了更多的模式：比如facade(门面)，observer(观察者)，singleton(单件)等等

先不管其它，来一个flex版本的Hello World吧。(注：以下内容参考了[Pure MVC第一步：最简PureMVC](http://bbs.9ria.com/thread-58666-1-1.html))

整个项目结构：



这个项目最终就是要在界面上显示一个文本框，一个按钮，点击按钮时，文本框内容发生改变。(注：该项目纯粹是为了puremvc而puremvc，不具实践意义，仅仅只是出于学习目的，刻意简化的一个示例) 另：因为这个项目不涉及具体的数据实体类，所以model这部分就省去了。

在puremvc中，model/view/controller统一是由Facade类的[单件实例](http://www.cnblogs.com/yjmyzz/archive/2010/07/27/1785984.html" \t "_blank)来统筹管理的。

大致流程可理解为：通过Facade类的单件实例（即：统一的门面） 启动 puremvc环境，启动同时注册Command对象(相当于asp.net mvc中的controller)，然后Command通过与之关联的facade(即前面的单件实例)来注册Mediator(中介者：用于把View与Command联系起来)。

当UI界面(即View)上有动静时（比如按钮点击了之类），与之关联的Mediator(中介者)会发送通知给facade，然后facade会调用command对象执行相关的处理。(即：消息响应)

应该不难理解吧：

1、先在flash builder中创建一个flex项目，同时添加puremvc相关的swc引用，然后主界面main.mxml内容如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <?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/mx" minWidth="300" minHeight="200" height="42" width="300" >      <fx:Declarations>          <!-- Place non-visual elements (e.g., services, value objects) here -->      </fx:Declarations>        <s:TextInput x="10" y="10" width="206" id="txtResult"/>      <s:Button x="224" y="9" label="Button" id="btnSend"/>  </s:Application> |

2、创建门面AppFacade类

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | package mvc  {      import mvc.controller.AppCommand;        import org.puremvc.as3.interfaces.IFacade;      import org.puremvc.as3.patterns.facade.Facade;        public class AppFacade extends Facade implements IFacade      {          public static const START\_UP:String="start\_up";          public static const CHANGE\_TEXT:String="change\_text";            public function AppFacade()          {            }            public static function getInstance():AppFacade          {              if (instance == null)              {                  instance=new AppFacade();              }              return instance as AppFacade;          }            protected override function initializeController():void          {              super.initializeController();              this.registerCommand(START\_UP, AppCommand);//入口：注册Command          }            //启动puremvc环境          public function startup(\_main:main):void          {              //注：这里的main即为UI界面main.mxml对应的类              this.sendNotification(START\_UP, \_main);//启动时，发送通知          }      }  } |

当然，这不是一个严格意义的单件(构造函数仍然可调用，不过这里不想把代码弄得太复杂，大家暂时把它当做一个单件来看待吧) 。

注：注意这里加注释的部分，在initializeController中，注册了一个Command对象(AppCommand在代码下面会贴出)，这样就把门面Facade与命令Command联系起来了，另外在startup里，把主界面main.mxml对应的类实例使用\_main传递进来，这样puremvc环境就跟程序主类联系起来了。

3、创建AppCommand类

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | package mvc.controller  {      import mvc.view.ButtonMediator;      import mvc.view.TextMediator;        import org.puremvc.as3.interfaces.INotification;      import org.puremvc.as3.patterns.command.SimpleCommand;        public class AppCommand extends SimpleCommand      {          public function AppCommand()          {              super();          }            public override function execute(inote:INotification):void          {              var \_main:main=inote.getBody() as main;              //注册：中介者(用于连接view与command)              facade.registerMediator(new TextMediator(\_main.txtResult));              facade.registerMediator(new ButtonMediator(\_main.btnSend));          }      }  } |

注：这里的execute方法中，通过facade引用，得到puremvc中的门面Facade唯一实例，然后注册Mediator中介者，把界面上的元素(即输入框与按钮)跟门面也关联起来了。

4、中介者(TextMediator与ButtonMediator)

这也是跟asp.net mvc不同的一个地方，puremvc并没有直接操作视图(View)上的元素，而是引入了中介者(Mediator)这个概念，它相当于“View上的具体元素”与Command之间的桥梁，这样的好处，就是进一步解耦，从一定程度上消除对具体界面的依赖。

ButtonMediator.as

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | package mvc.view  {      import spark.components.Button;      import flash.events.MouseEvent;      import mvc.AppFacade;        import org.puremvc.as3.interfaces.IMediator;      import org.puremvc.as3.patterns.mediator.Mediator;        public class ButtonMediator extends Mediator implements IMediator      {          public static const NAME:String = "ButtonMediator";            public function ButtonMediator(viewComponent:Button)          {              super(NAME,viewComponent);              this.btnInstance.addEventListener(MouseEvent.CLICK,btnClick);            }            private function btnClick(e:MouseEvent):void{              this.sendNotification(AppFacade.CHANGE\_TEXT,"Hello PureMVC !");          }            //获取UI界面上的“按钮实例"          private function get btnInstance():Button{              return viewComponent as Button;          }      }  } |

可以看到，ButtonMediator通过调用sendNotification方法向puremvc环境发送消息CHANGE\_TEXT就完事了，至于谁去接收消息，它不关心。

TextMediator.as

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | package mvc.view  {      import spark.components.TextInput;      import mvc.AppFacade;        import org.puremvc.as3.interfaces.IMediator;      import org.puremvc.as3.interfaces.INotification;      import org.puremvc.as3.patterns.mediator.Mediator;        public class TextMediator extends Mediator implements IMediator      {          public static const NAME:String = "TextMediator";            public function TextMediator(viewComponent:TextInput)          {              super(NAME,viewComponent);          }            public override function listNotificationInterests():Array{              return [AppFacade.CHANGE\_TEXT];          }            //响应消息          public override function handleNotification(notification:INotification):void          {              switch(notification.getName()){                  case AppFacade.CHANGE\_TEXT:                      this.txtInstance.text = notification.getBody() as String;                      break;              }          }            //获取与之关联的UI界面上的“文本输入框”          private function get txtInstance():TextInput{              return viewComponent as TextInput;          }      }  } |

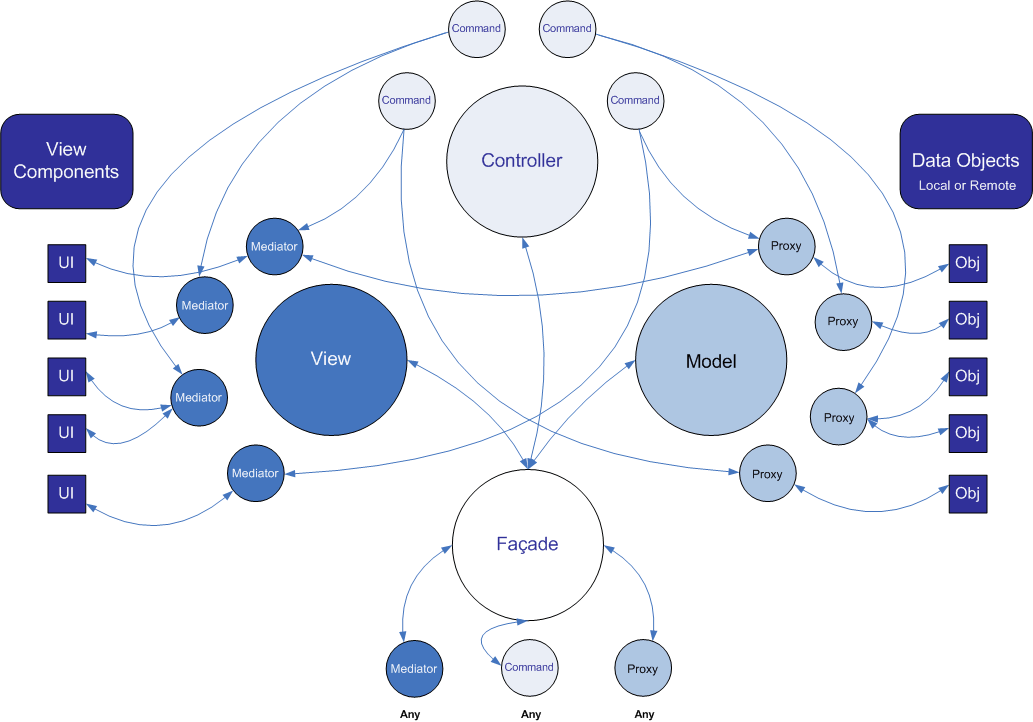
在这段代码中，TextMediator负责监听CHANGE\_TEXT消息，同时得到消息后，会做出反应。这样跟上面的ButtonMediaor一唱一合，完成了整个消息的处理。

ok，代码准备得差不多了，回过头在main.mxml中正式启用puremvc环境吧：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | <?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/mx" minWidth="300" minHeight="200" height="42" width="300" creationComplete="application1\_creationCompleteHandler(event)">      <fx:Declarations>          <!-- Place non-visual elements (e.g., services, value objects) here -->      </fx:Declarations>        <fx:Script>          <![CDATA[              import mvc.AppFacade;              import mx.events.FlexEvent;                protected function application1\_creationCompleteHandler(event:FlexEvent):void              {                  //启动puremvc环境                  AppFacade.getInstance().startup(this);              }          ]]>      </fx:Script>      <s:TextInput x="10" y="10" width="206" id="txtResult"/>      <s:Button x="224" y="9" label="Button" id="btnSend"/>  </s:Application> |

[示例源代码下载](http://cid-2959920b8267aaca.office.live.com/self.aspx/flex/puremvc%5E_helloworld.rar) (flash builder4环境)

最后再附上一张puremvc的结构图：



[**puremvc框架之Command**](http://www.cnblogs.com/yjmyzz/archive/2010/07/31/1789634.html)

在前一篇 [puremvc框架之hello world!](http://www.cnblogs.com/yjmyzz/archive/2010/07/27/1786123.html) 里，已经对这个框架有了一个大概的认识，不过在消息的处理上，有一个不太适合的地方：

为了完成响应消息，TextMediator亲自去监听自己感兴趣的消息类型，然后亲自来处理。要知道：Mediator属于View层(即：MVC中的V)，它最好是什么也不干，仅仅与界面保持联系即可，对于如何响应消息这类粗活，应该交由Controller层(即MVC中的C)来处理最适合不过，所以这一章介绍如何把消息处理由V(View)转移到C(Controller)上来。

对于每类消息，我们可以创建一个与之对应的Command，以上回的CHANGE\_TEXT消息类型来说，我们可以创建一个ChangeTextCommand，代码如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | package mvc.controller  {      import mvc.view.TextMediator;        import org.puremvc.as3.interfaces.INotification;      import org.puremvc.as3.patterns.command.SimpleCommand;        public class ChangeTextCommand extends SimpleCommand      {          public function ChangeTextCommand()          {              super();          }            public override function execute(notification:INotification):void{                (facade.retrieveMediator(TextMediator.NAME) as TextMediator).txtInstance.text = notification.getBody() as String;          }      }  } |

关键在于(facade.retrieveMediator(TextMediator.NAME) as TextMediator).txtInstance.text = notification.getBody() as String;这一句，它就是用来处理消息的，同时我们也注意到txtInstance原来为private属性，为了能让外部访问，必须改成public方法，修改后的TextMediator.as如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | package mvc.view  {      import spark.components.TextInput;      import mvc.AppFacade;        import org.puremvc.as3.interfaces.IMediator;      import org.puremvc.as3.interfaces.INotification;      import org.puremvc.as3.patterns.mediator.Mediator;        public class TextMediator extends Mediator implements IMediator      {          public static const NAME:String = "TextMediator";            public function TextMediator(viewComponent:TextInput)          {              super(NAME,viewComponent);          }            /\*          public override function listNotificationInterests():Array{              return [AppFacade.CHANGE\_TEXT];          }            //响应消息          public override function handleNotification(notification:INotification):void          {              switch(notification.getName()){                  case AppFacade.CHANGE\_TEXT:                      this.txtInstance.text = notification.getBody() as String;                      break;              }          }          \*/            //获取与之关联的UI界面上的“文本输入框”          public function get txtInstance():TextInput{              return viewComponent as TextInput;          }      }  } |

有了ChangeTextCommand处理CHANGE\_TEXT消息后，TextMediator中就不需要亲自来监听并处理该消息了，所以把这部分去掉，同时把txtInstance改成public即可。

好了，最后一个问题：如何把ChangeTextCommand跟puremvc中的facade实例联系起来呢？

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | package mvc.controller  {      import mvc.view.ButtonMediator;      import mvc.view.TextMediator;        import org.puremvc.as3.interfaces.INotification;      import org.puremvc.as3.patterns.command.SimpleCommand;        import mvc.AppFacade;        public class AppCommand extends SimpleCommand      {          public function AppCommand()          {              super();          }            public override function execute(inote:INotification):void          {              var \_main:main=inote.getBody() as main;                facade.registerMediator(new TextMediator(\_main.txtResult));              facade.registerMediator(new ButtonMediator(\_main.btnSend));                //注册ChangeTextCommand              facade.registerCommand(AppFacade.CHANGE\_TEXT,ChangeTextCommand);          }      }  } |

注意加注释的行：facade.registerCommand(AppFacade.CHANGE\_TEXT,ChangeTextCommand); 这样就行了，因为AppCommand是在AppFacade中被注册的，所以把ChangeTextCommand的注册加到AppCommand中后，ChangeTextCommand就跟facade联系起来了。

注：本文内容参考了<http://bbs.9ria.com/viewthread.php?tid=58719>

源文件下载：[http://cid-2959920b8267aaca.office.live.com/self.aspx/flex/puremvc^\_command.fxp](http://cid-2959920b8267aaca.office.live.com/self.aspx/flex/puremvc%5E_command.fxp) (下载后，直接用Flash Builder 4导入即可)

[**puremvc框架之proxy**](http://www.cnblogs.com/yjmyzz/archive/2010/08/01/1789769.html)

上一篇 [puremvc框架之Command](http://www.cnblogs.com/yjmyzz/archive/2010/07/31/1789634.html) 里，已经学习了如何利用Command来解耦View层与业务逻辑的依赖，但是仍然有二个问题:

1、ButtonMediator中发送消息时，仍然采用硬编码的方式，将消息内容写死在代码中：

|  |  |
| --- | --- |
| 1  2  3 | private function btnClick(e:MouseEvent):void{              this.sendNotification(AppFacade.CHANGE\_TEXT,"Hello PureMVC !");          } |

这显然不是一个好的设计，不够灵活

2、我们一直在说puremvc是一个mvc框架，至今为止 controller(即Command)、view(即Mediator)都已经出现过了，但是model层还是不见踪影，puremvc中的model层在哪里？

在asp.net mvc中，model层通常是定义数据实体的部分，可以选用的技术有很多，比如linq to sql，linq to entity ，nhibernate之类，这个概念在puremvc中仍然是相通的，只不过换了个名字，我们称之为Proxy！

先来定义一个AppProxy类吧(放到mvc.model包中)，代码如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | package mvc.model  {      import org.puremvc.as3.interfaces.IProxy;      import org.puremvc.as3.patterns.proxy.Proxy;      import flash.events.IOErrorEvent;      import flash.events.Event;      import flash.net.URLLoader;      import flash.net.URLRequest;          public class AppProxy extends Proxy implements IProxy      {          public static const NAME:String="AppProxy"; //名称常量            public function AppProxy(proxyName:String=null, data:Object=null)          {              super(NAME, data);//将本proxy的名称常量传入其中              var \_loader:URLLoader=new URLLoader;              \_loader.addEventListener(Event.COMPLETE, onComplete);              \_loader.addEventListener(IOErrorEvent.IO\_ERROR, onError);              \_loader.load(new URLRequest("data.xml"));          }            private function onComplete(e:Event):void          {                var \_xml:XML=XML(e.target.data);              setData(\_xml); //将xml内容保存进data          }            private function onError(e:IOErrorEvent):void          {              trace("数据获取失败！");          }      }  } |

这里，我们用xml做为数据源来提供数据，data.xml放到根目录下，内容如下：

|  |  |
| --- | --- |
| 1  2  3  4 | <?xml version="1.0" encoding="UTF-8"?>  <message>          <msg>Hello World!</msg>  </message> |

ok，这一步做好后，老问题又来了：如何让它跟puremvc环境中的facade实例挂上勾？类似上一篇的处理，还是放到AppCommand中来处理

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | package mvc.controller  {      import mvc.AppFacade;      import mvc.model.AppProxy;      import mvc.view.ButtonMediator;      import mvc.view.TextMediator;        import org.puremvc.as3.interfaces.INotification;      import org.puremvc.as3.patterns.command.SimpleCommand;        public class AppCommand extends SimpleCommand      {          public function AppCommand()          {              super();          }            public override function execute(inote:INotification):void          {              var \_main:main=inote.getBody() as main;                //注册proxy              facade.registerProxy(new AppProxy());                facade.registerMediator(new TextMediator(\_main.txtResult));              facade.registerMediator(new ButtonMediator(\_main.btnSend));                facade.registerCommand(AppFacade.CHANGE\_TEXT,ChangeTextCommand);          }      }  } |

注意加注释的部分facade.registerProxy(new AppProxy());这样就ok了，这一步执行后，puremvc环境中就已经有data.xml的数据了

现在就可以把原来ButtonMediator中硬编码的部分去掉了，改成下面这样：

|  |  |
| --- | --- |
| 1  2  3  4 | private function btnClick(e:MouseEvent):void{          //this.sendNotification(AppFacade.CHANGE\_TEXT,"Hello PureMVC !");          sendNotification(AppFacade.CHANGE\_TEXT);      } |

即：view层只发送消息(类型)，通知puvrmvc环境--“CHANGE\_TEXT消息我已经发出去了！”，至于数据在哪，谁去处理，关我P事!

OK，有人发了消息，自动就要有人处理，接下来折腾ChangeTextCommand.as

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public override function execute(notification:INotification):void{                var \_proxy:AppProxy=facade.retrieveProxy(AppProxy.NAME) as AppProxy;              var \_xml:XML=XML(\_proxy.getData());              trace(\_xml);              (facade.retrieveMediator(TextMediator.NAME) as TextMediator).txtInstance.text = \_xml.msg;              //(facade.retrieveMediator(TextMediator.NAME) as TextMediator).txtInstance.text = notification.getBody() as String;          } |

这里，我们把原来的方法注释掉了，改成用Proxy的getData获取刚才data.xml中的数据，然后该数据赋值为TextMediator相关联的文本框.

至此，M(proxy)-V(mediator)-C(command)全都登场了，相互之间也实现了完全解耦！

最后再从头回顾一下主要的处理细节：

1、man.mxml中通过 AppFacade.getInstance().startup(this) 启动puvemvc环境

2、而AppFacade又通过this.registerCommand(START\_UP, AppCommand) 注册AppCommand

3、AppCommand中又通过

|  |  |
| --- | --- |
| 1  2  3  4 | facade.registerProxy(new AppProxy());  facade.registerMediator(new TextMediator(\_main.txtResult));  facade.registerMediator(new ButtonMediator(\_main.btnSend));  facade.registerCommand(AppFacade.CHANGE\_TEXT,ChangeTextCommand); |

把mediator、proxy以及消息CHANGE\_TEXT相关的ChangeTextCommand给扯进来

4、然后ButtonMediator中又通过sendNotification(AppFacade.CHANGE\_TEXT)来发送自己感兴趣的消息

5、最后CHANGE\_TEXT消息被与之关联的ChangeTextCommand得到，并在execute方法中处理以更新UI界面。

源文件下载：[http://cid-2959920b8267aaca.office.live.com/self.aspx/flex/PureMVC^\_Proxy.fxp](http://cid-2959920b8267aaca.office.live.com/self.aspx/flex/PureMVC%5E_Proxy.fxp) （用FB4导入即可）