**[struts2采用convention-plugin实现零配置](http://javeye.javaeye.com/blog/358744)**

最近开始关注struts2的新特性，从这个版本开始，Struts开始使用convention-plugin代替codebehind-plugin来实现struts的零配置。  
配置文件精简了，的确是简便了开发过程，但是，我们熟悉的配置突然disappear了，真是一下很不适应。跟着潮流走吧，看看该怎样来搞定convention-plugin。  
使用Convention插件，你需要将其JAR文件放到你应用的WEB-INF/lib目录中，你也可以在你Maven项目的POM文件中添加下面包依赖

**Xml代码**

1. **<dependency>**
2. **<groupId>**org.apache.struts**</groupId>**
3. **<artifactId>**struts2-convention-plugin**</artifactId>**
4. **<version>**2.1.6**</version>**
5. **</dependency>**

零配置并不是没有配置，而是通过约定大于配置的方式，大量通过约定来调度页面的跳转而使得配置大大减少。所以，首先应该了解下convention-plugin的约定：  
1. 默认所有的结果页面都存储在WEB-INF/content下，你可以通过设置struts.convention.result.path这个属性的值来改变到其他路径。如：

**Xml代码**

1. **<constant** name="struts.convention.result.path" value="/WEB-INF/page" **/>**

则将路径配置到了WEB-INF/page 下。  
2. 默认包路径包含action,actions,struts,struts2的所有包都会被struts作为含有Action类的路径来搜索。你可以通过设置struts.convention.package.locators属性来修改这个配置。如：

**Xml代码**

1. **<constant** name="struts.convention.package.locators" value="web,action" **/>**

则定义了在项目中，包路径包含web和action的将被视为Action存在的路径来进行搜索。  
Com.ustb.web.\*/com.ustb.action.\*都将被视为含有Action的包路径而被搜索。  
3. 接着，Convention从前一步找到的package以及其子package中寻找 com.opensymphony.xwork2.Action 的实现以及以Action结尾的类：

**Java代码**

1. com.example.actions.MainAction
2. com.example.actions.products.Display (**implements** com.opensymphony.xwork2.Action)
3. com.example.struts.company.details.ShowCompanyDetailsAction

4. 命名空间。从定义的.package.locators标示开始到包结束的部分，就是命名空间。举个例子：  
Com.ustb.web.user.userAction的命名空间是：”/user”。Com.ustb.web.user.detail.UserAction的命名空间是：”/user/detail”  
5. Convention通过如下规则确定URL的具体资源部分:去掉类名的Action部分。然后将将每个分部的首字母转为小写，用’-’分割，你可以设置struts.convention.action.name.separator 如

**Xml代码**

1. **<constant** name="struts.convention.action.name.separator" value="-" **/>**

还是举个例子：  
UserAction->user  UserDetailAction ->user-detail。结合上面的。对于com.ustb.web.user.detail.UserDetailAction，映射的url就是/WEB-INF/content/user/detail/user-detail.jsp  
6. struts支持.jsp .html .htm .vm格式的文件。  
下面是actiong和结果模版的映射关系：

|  |  |  |  |
| --- | --- | --- | --- |
| **URL** | **Result** | **File that could match** | **Result Type** |
| /hello | success | /WEB-INF/content/hello.jsp | Dispatcher |
| /hello | success | /WEB-INF/content/hello-success.htm | Dispatcher |
| /hello | success | /WEB-INF/content/hello.ftl | FreeMarker |
| /hello-world | input | /WEB-INF/content/hello-world-input.vm | Velocity |
| /test1/test2/hello | error | /WEB-INF/content/test/test2/hello-error.html | Dispatcher |

以上的内容来自struts2的文档[http://struts.apache.org/2.1.6/docs/convention-plugin.html](http://struts.apache.org/2.1.6/docs/convention-plugin.html" \t "_blank)

当然，简单的通过默认的方式来进行配置不能完全满足实际项目的需要。所幸，convention的零配置是非常灵活的。  
**通过@Action注释**   
对如下例子：

**Java代码**

1. **package** com.example.web;
3. **import** com.opensymphony.xwork2.Action;
4. **import** com.opensymphony.xwork2.ActionSupport;
6. **public** **class** HelloAction **extends** ActionSupport {
7. @Action("action1")
8. **public** String method1() {
9. **return** SUCCESS;
10. }
12. @Action("/user/action2")
13. **public** String method2() {
14. **return** SUCCESS;
15. }
16. }

|  |  |  |
| --- | --- | --- |
| **方法名** | **默认调用路径** | **默认映射路径** |
| method1 | /hello!method1.action . | /WEB-INF/content/hello.jsp |
| method2 | /hello!method2.action. | /WEB-INF/content/hello.jsp |

通过@Action注释后

|  |  |  |
| --- | --- | --- |
| **方法名** | **@Action注释后调用路径** | **@Action注释** **后映射路径** |
| method1 | /action1!method1.action. | /WEB-INF/content/action1.jsp |
| method1 | /user/action2!method2.action | /WEB-INF/content/user/action2.jsp |

**通过@Actions注释**

**Java代码**

1. **package** com.example.web;
3. **import** com.opensymphony.xwork2.ActionSupport;
4. **import** org.apache.struts2.convention.annotation.Action;
5. **import** org.apache.struts2.convention.annotation.Actions;
7. **public** **class** HelloAction **extends** ActionSupport {
8. @Actions({
9. @Action("/different/url"),
10. @Action("/another/url")
11. })
12. **public** String method1() {
13. **return** “error”;
14. }

我们可以通过：***/different/url!method1.action***或 ***/another/url!method1.action***来调用**method1** 方法。  
对应的映射路径分别是***/WEB-INF/content/different/url-error.jsp; /WEB-INF/content/another/url-error.jsp***  
可能误导了大家，一个方法被@Action注释后，只是多了一种调用方式，而不是说覆盖了原来的调用方式。比如对于如下例子：

**Java代码**

1. **package** com.example.web;
3. **import** com.opensymphony.xwork2.ActionSupport;
4. **import** org.apache.struts2.convention.annotation.Action;
5. **import** org.apache.struts2.convention.annotation.Actions;
7. **public** **class** HelloAction **extends** ActionSupport {
8. @Action("/another/url")
9. **public** String method1() {
10. **return** “error”;
11. }

我们调用method1方法可以通过两种方式：  
1  **/hello!method1.action** 映射 url：**/WEB-INF/content/hello-error.jsp**  
2 **/another/url!method1.action**映射 url：**/WEB-INF/content/another/url-error.jsp**  
可见，两种方式均可对method1方法进行调用，唯一的区别就是，两种调用的映射是不一样的，所以，想跳转到不同的界面，这是一个非常好的选择。  
  
  
**通过@Namespace 注释**

**Java代码**

1. **package** com.example.web;
3. **import** com.opensymphony.xwork2.ActionSupport;
4. **import** org.apache.struts2.convention.annotation.Action;
5. **import** org.apache.struts2.convention.annotation.Actions;
6. @Namespace("/other")
7. **public** **class** HelloWorld **extends** ActionSupport {
9. **public** String method1() {
10. **return** “error”;
11. }
12. @Action("url")
13. **public** String method2() {
14. **return** “error”;
15. }
17. @Action("/different/url")
18. **public** String method3() {
19. **return** “error”;
20. }
21. }

通过 **/other/hello-world!method1.action**访问**method1** 方法。  
通过 **/other/url!method2.action**访问**method2** 方法  
通过 **/different /url!method3.action**访问**method3** 方法   
与@Action 注释不同的是，该注释覆盖了默认的namespace(这里是’/’)，此时再用hello!method1.action 已经不能访问**method1** 了.  
**@Results和@Result**   
**1 全局的（global）。**   
全局results可以被action类中所有的action分享，这种results在action类上使用注解进行声明。

**Java代码**

1. **package** com.example.actions;
3. **import** com.opensymphony.xwork2.ActionSupport;
4. **import** org.apache.struts2.convention.annotation.Action;
5. **import** org.apache.struts2.convention.annotation.Actions;
6. **import** org.apache.struts2.convention.annotation.Result;
7. **import** org.apache.struts2.convention.annotation.Results;
9. @Results({
10. @Result(name="failure", location="/WEB-INF/fail.jsp")
11. })
12. **public** **class** HelloWorld **extends** ActionSupport {
13. **public** String method1() {
14. **return** “failure”;
15. }
16. @Action("/different/url")
17. **public** String method2() {
18. **return** “failure”;
19. }
21. }

当我们访问 **/hello -world !method1.action**时，返回 **/WEB-INF/fail.jsp**   
当我们访问 **/hello -world !method2.action**时，返回 **/WEB-INF/fail.jsp**  
当我们访问 **/different/url!method2.action**时，返回 **/WEB-INF/fail.jsp**  
  
**2 本地的（local）。**   
本地results只能在action方法上进行声明。

**Java代码**

1. **package** com.example.actions;
3. **import** com.opensymphony.xwork2.ActionSupport;
4. **import** org.apache.struts2.convention.annotation.Action;
5. **import** org.apache.struts2.convention.annotation.Actions;
6. **import** org.apache.struts2.convention.annotation.Result;
7. **import** org.apache.struts2.convention.annotation.Results;
9. **public** **class** HelloWorld **extends** ActionSupport {
10. @Action(value="/other/bar",results={@Result(name = "error", location = "www.baidu.com",type="redirect")})
11. **public** String method1() {
12. **return** “error”;
13. }
14. }

当我们调用 **/hello -world !method1.action**时，返回 **/WEB-INF/content/hello-error.jsp**  
当我们调用 **/other/bar!method1.action**时，返回 **www.baidu.com**  
  
夜深了，想学习的，靠你们自己了哈……