# 1. 初识Struts

我们现在起开始学习Java企业级开发中常用的框架。框架是一种“半成品”，框架本身已经提供了部分功能，使用框架能提高开发效率。

Struts框架是Apache维护的产品，官网地址是：<http://struts.apache.org/> 。应用Struts框架能很好的实现MVC的软件模式，其中Struts就充当其中的“C”，即Controller控制器。

以前使用Servlet充当控制器，但是在使用Servlet时，目前我们感觉到有些“不便使用”之处，比如通常一个Servlet只能处理一个请求路径，若项目较复杂，新建的Servlet类会非常多。再比如需要自己手动获取请求的参数、手动处理文件的上传下载等。

而使用Struts框架，就能以较高效的办法解决上述问题，从而让开发者专注于开发具体的业务。

## 1.1 初次使用Struts

（1）首先在官网下载好Struts的开发包，这里使用的是Struts 2.5.10版本。在Web项目中引入下列jar包（jar包在Struts的lib目录中）：

|  |  |
| --- | --- |
| JAR包 | 描述 |
| commons-fileupload-1.3.2.jar | 文件上传相关包 |
| commons-io-2.4.jar | IO工具包 |
| commons-lang3-3.4.jar | java.lang的扩展包 |
| freemarker-2.3.23.jar | freemarker模板包 |
| log4j-api-2.7.jar | Log4J相关包 |
| ognl-3.1.12.jar | ONGL表达式支持包 |
| struts2-core-2.5.10.jar | Struts核心包 |

发现上述引用了log4j-api包，但是lib目录下并没有log4j-core包，实际上日志的实现包不一定就要用Log4J的，也可以使用其他的日志实现。这里我们还是自行引入以下log4j-core这个实现包（版本要对应）。同时我们在src目录下新建Log4J配置文件log4j2.xml，内容如下：

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**configuration status="error"**>  <**appenders**>  <**Console name="Console" target="SYSTEM\_OUT"**>  <**ThresholdFilter level="trace" onMatch="ACCEPT" onMismatch="DENY"**/>  <**PatternLayout pattern="%d{HH:mm:ss.SSS} %-5level %l - %msg%xEx%n"**/>  </**Console**>  <**File name="log" fileName="log/test.log" append="false"**>  <**ThresholdFilter level="trace" onMatch="ACCEPT" onMismatch="DENY"**/>  <**PatternLayout pattern="%d{HH:mm:ss.SSS} %-5level %class{36} %L %M - %msg%xEx%n"**/>  </**File**>  </**appenders**>  <**loggers**>  <**root level="trace"**>  <**appender-ref ref="Console"**/>  <**appender-ref ref="log"**></**appender-ref**>  </**root**>  </**loggers**> </**configuration**> |

（2）上述引入好了jar包，现在需要在web.xml中将Struts配置进来。Struts是通过过滤器配置到Web项目中的。web.xml中内容如下：

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"  version="3.1"**>  *<!-- 配置Struts核心过滤器 -->* <**filter**>  <**filter-name**>struts</**filter-name**>  *<!-- 核心过滤器类是StrutsPrepareAndExecuteFilter -->* <**filter-class**>org.apache.struts2.dispatcher.filter.StrutsPrepareAndExecuteFilter</**filter-class**>  </**filter**>  <**filter-mapping**>  <**filter-name**>struts</**filter-name**>  *<!-- 这会拦截所有的请求资源 -->* <**url-pattern**>/\*</**url-pattern**>  </**filter-mapping**> </**web-app**> |

（3）上述完成了Struts的导入。现在我们就用Struts进行开发。在Java Web中，一般称Controller中处理请求的类叫Action类，并且习惯请求连接以.action结尾。我们常说在Struts中写一个Action类，就是写一个类用于处理用户请求。

Struts中，一个Action类可处理多个请求，现在我们就创建一个Action来体验与Servlet的不同。新建一个名叫DemoAction的类作为Action类，在类中直接写自定义方法用于处理请求，该方法需要返回字符串类型数据。例如：

|  |
| --- |
| **package** com.zhang.action;  **import** org.apache.logging.log4j.LogManager; **import** org.apache.logging.log4j.Logger;  **public class** DemoAction {  Logger **logger** = LogManager.*getLogger*();  **public** String add() {  **logger**.info(**"执行了add业务方法。"**);  **return "success"**;  } } |

为了能使用户请求可访问此方法，还需要对Struts的访问进行专门的配置。需要在src文件夹下新建一个struts.xml配置文件，框架会自动加载这个文件。其中就用于配置访问路径对应的Action类和方法，以及返回路径。例如：

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* **<!DOCTYPE struts PUBLIC  "-//Apache Software Foundation//DTD Struts Configuration 2.5//EN"  "http://struts.apache.org/dtds/struts-2.5.dtd"*>*** <**struts**>  *<!-- 所有的action都是在package中配置的。package可管理不同的action -->  <!-- package的name要求是唯一的名称，即如果配置多个package，名称不能重复。extends表示继承，一般都继承自struts-default -->* <**package name="demo" extends="struts-default"**>  *<!--在package中可写多个action的配置-->  <!--一个action标签用于配置一个action请求。action有如下属性：  name：表示访问此action的名称，就是在浏览器中输入的访问地址。  class：指示该action的处理类，这里写为DemoAction。  method：指示该action对应的方法，就是add方法。  通过上述配置，就确定了如何访问到Action类的一个处理方法。  -->* <**action name="expAdd" class="com.zhang.action.DemoAction" method="add"**>  *<!-- action中可配置result节点。该节点用于配置请求跳转到哪个页面。  result节点是根据方法的返回值来配置跳转的页面的。  name属性用于匹配方法的返回值，若返回值匹配，则会跳转到执行的页面，若找不到匹配的返回值，则会报错。  例如下面配置了success返回值将会跳转到index.jsp页面。我们在add方法中返回的也是success，所以能匹配。当然，可配置多个result节点，这样当程序返回不同值时，就能跳转到不同页面。  -->* <**result name="success"**>/index.jsp</**result**>  </**action**>  </**package**> </**struts**> |

这样，第一个Struts例子就编写完毕了。启动Tomcat后，我们就能通过地址：localhost:8080/应用名/expAdd 访问了（或者加上.action后缀，即localhost:8080/应用名/expAdd.action）。可以看到相应的日志输出。因此我们以后会直接编写Action类，而不再写Servlet类了。因为一个Action类中可写多个方法，我们只需在struts里多配置action请求即可。

## 1.2 Struts的执行流程

（1）当服务器启动时，Tomcat会加载项目的web.xml文件，其中配置了Struts过滤器，则同时会创建Struts核心过滤器对象，并执行过滤器的init()方法。

（2）init()方法会加载以下配置文件：

struts-default.xml：struts的默认配置，此文件在struts的包中，用于完成核心功能的初始化，

struts-plugin.xml：Struts相关插件配置，此文件也在struts包中，

struts.xml：这是我们开发者自行编写的配置文件。

（3）当浏览器用户初次访问时，服务器会根据访问路径找到相应的action配置，并创建action对象。

（4）Struts会执行默认拦截器栈中定义的拦截器，最后执行业务处理方法。每次请求都会走这一步。关于Struts拦截器下面讲。

## 1.3 Struts配置文件

首先，我们首先看一下struts的默认配置文件struts-default.xml，此文件在struts-core.jar中。大概内容是：

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"** *?> <!--  注释中是对文件的说明 -->* **<!DOCTYPE struts PUBLIC  "-//Apache Software Foundation//DTD Struts Configuration 2.5//EN"  "http://struts.apache.org/dtds/struts-2.5.dtd"*>*** <**struts**>   *<!-- constant节点用于定义常量 -->* <**constant name="struts.excludedClasses"  value="  java.lang.Object,  java.lang.Runtime,  java.lang.System,  java.lang.Class,  java.lang.ClassLoader,  java.lang.Shutdown,  java.lang.ProcessBuilder,  ognl.OgnlContext,  ognl.ClassResolver,  ognl.TypeConverter,  ognl.MemberAccess,  ognl.DefaultMemberAccess,  com.opensymphony.xwork2.ognl.SecurityMemberAccess,  com.opensymphony.xwork2.ActionContext"** />   <**constant name="struts.excludedPackageNames" value="java.lang.,ognl,javax,freemarker.core,freemarker.template"** />   *<!-- 很多的bean节点用于Struts所需对象的创建。这和以后要学习的Spring框架有关系。 -->* <**bean class="com.opensymphony.xwork2.ObjectFactory" name="struts"**/>  <**bean type="com.opensymphony.xwork2.factory.ResultFactory" name="struts" class="org.apache.struts2.factory.StrutsResultFactory"** />  *<!-- ...很多的bean配置，省略...-->   <!-- 这里就是struts-default默认包，它的abstract为true，说明只能被别的包继承。包中定义了result结果类型和拦截器 -->* <**package name="struts-default" abstract="true" strict-method-invocation="true"**>  *<!-- 各种结果类型 -->* <**result-types**>  <**result-type name="chain" class="com.opensymphony.xwork2.ActionChainResult"**/>  <**result-type name="dispatcher" class="org.apache.struts2.result.ServletDispatcherResult" default="true"**/>  <**result-type name="freemarker" class="org.apache.struts2.views.freemarker.FreemarkerResult"**/>  <**result-type name="httpheader" class="org.apache.struts2.result.HttpHeaderResult"**/>  <**result-type name="redirect" class="org.apache.struts2.result.ServletRedirectResult"**/>  <**result-type name="redirectAction" class="org.apache.struts2.result.ServletActionRedirectResult"**/>  <**result-type name="stream" class="org.apache.struts2.result.StreamResult"**/>  <**result-type name="velocity" class="org.apache.struts2.result.VelocityResult"**/>  <**result-type name="xslt" class="org.apache.struts2.views.xslt.XSLTResult"**/>  <**result-type name="plainText" class="org.apache.struts2.result.PlainTextResult"** />  <**result-type name="postback" class="org.apache.struts2.result.PostbackResult"** />  </**result-types**>   *<!-- 定义拦截器interceptor -->* <**interceptors**>  <**interceptor name="alias" class="com.opensymphony.xwork2.interceptor.AliasInterceptor"**/>  <**interceptor name="autowiring" class="com.opensymphony.xwork2.spring.interceptor.ActionAutowiringInterceptor"**/>  *<!-- ...很多interceptor... -->   <!-- 以下定义了很多的拦截器栈。拦截器栈就是为了方便引用很多的拦截器。 -->  <!-- Basic stack -->* <**interceptor-stack name="basicStack"**>  *<!-- 内部都是引用了上面的拦截器 -->* <**interceptor-ref name="exception"**/>  <**interceptor-ref name="servletConfig"**/>  <**interceptor-ref name="prepare"**/>  <**interceptor-ref name="checkbox"**/>  <**interceptor-ref name="datetime"**/>  <**interceptor-ref name="multiselect"**/>  <**interceptor-ref name="actionMappingParams"**/>  <**interceptor-ref name="params"**/>  <**interceptor-ref name="conversionError"**/>  </**interceptor-stack**>   *<!-- Sample validation and workflow stack -->* <**interceptor-stack name="validationWorkflowStack"**>  <**interceptor-ref name="basicStack"**/>  <**interceptor-ref name="validation"**/>  <**interceptor-ref name="workflow"**/>  </**interceptor-stack**>   *<!-- 默认的拦截器栈 -->* <**interceptor-stack name="defaultStack"**>  <**interceptor-ref name="exception"**/>  <**interceptor-ref name="alias"**/>  <**interceptor-ref name="servletConfig"**/>  <**interceptor-ref name="i18n"**/>  <**interceptor-ref name="prepare"**/>  <**interceptor-ref name="chain"**/>  <**interceptor-ref name="scopedModelDriven"**/>  <**interceptor-ref name="modelDriven"**/>  <**interceptor-ref name="fileUpload"**/>  <**interceptor-ref name="checkbox"**/>  <**interceptor-ref name="datetime"**/>  <**interceptor-ref name="multiselect"**/>  <**interceptor-ref name="staticParams"**/>  <**interceptor-ref name="actionMappingParams"**/>  <**interceptor-ref name="params"**/>  <**interceptor-ref name="conversionError"**/>  <**interceptor-ref name="validation"**>  <**param name="excludeMethods"**>input,back,cancel,browse</**param**>  </**interceptor-ref**>  <**interceptor-ref name="workflow"**>  <**param name="excludeMethods"**>input,back,cancel,browse</**param**>  </**interceptor-ref**>  <**interceptor-ref name="debugging"**/>  </**interceptor-stack**>  </**interceptors**>   *<!-- 指示默认使用的拦截器栈就是上面定义的defaultStack -->* <**default-interceptor-ref name="defaultStack"**/>   *<!-- 指示默认执行的action类就是ActionSupport -->* <**default-class-ref class="com.opensymphony.xwork2.ActionSupport"** />  <**global-allowed-methods**>execute,input,back,cancel,browse,save,delete,list,index</**global-allowed-methods**>  </**package**> </**struts**> |

（1）常用的跳转结果类型。可在result节点中通过type设置：

dispatcher：转发，这是默认的。转发就意味着结果页面能获取request域对象数据。

redirect：重定向

redirectAction：重定向到action资源。比如可重定向到本命名空间的Action，若要重定向到其他名称空间的Action，则要写上nameSpace参数，比如：

|  |
| --- |
| <**result name="list" type="redirectAction"**>  <**param actionName="add"** />  <**param nameSpace="/user"** /> </**result**> |

重定向到Action也很常用，比如访问“add”这个Action时，可实现添加一个用户的功能，若要求添加用户后显示所有人的列表，则要访问“list”这个action。若采用转发方式访问list.action，那么浏览器地址不会改变，若用户刷新页面，则很可能又会添加一个用户，这时就应该使用重定向。

stream：流，文件下载时需要用到这个结果类型。

在Web中，外部不能直接访问WEB-INF下的资源，也就是不能通过浏览器输入地址的方式或者重定向方式访问WEB-INF下的资源，只能通过内部转发的方式访问。所以，为了安全性，一般将JSP文件放在WEB-INF目录下。此时struts配置为：

|  |
| --- |
| <**result name="login" type="dispatcher"**>/WEB-INF/login.jsp</**result**> |

这样，login请求就能返回WEB-INF下的资源。但是如果把type改成redirect，则报错，因为WEB-INF下内容不能重定向访问。

（2）Struts的拦截器：

Struts定义了很多拦截器和拦截器栈。如果引用了某个拦截器栈，其中的所有拦截器都会被引用。

Struts的拦截器和Servlet中的过滤器功能类似，都用于拦截资源，可增加很多功能。拦截器和过滤器的区别如下：

（1）拦截器只能拦截action请求，过滤器可拦截所有资源；

（2）拦截器是struts中的概念，只能在struts中用，而过滤器在Web项目中都能使用用。

## 1.4 参数拦截器

在默认拦截器中，提供了参数拦截器。这个拦截器使用的非常多，因为该拦截器能把请求参数直接封装到Action类的属性中。使用十分方便。

例1：使用表单提交用户名和密码参数。

（1）JSP页面：

|  |
| --- |
| <%@ **page pageEncoding**="**UTF-8**" **language**="**java**" **isELIgnored**="**false**" %> <**html**> <**head**>  <**meta charset="UTF-8"** />  <**title**>登录</**title**> </**head**> <**body**>  <**form action="${**pageContext.request.contextPath**}/login.action" method="GET"**>  <**p**>用户名：<**input type="text" name="userName"**></**p**>  <**p**>密码：<**input type="password" name="pwd"**></**p**>  <**p**><**input type="submit"**></**p**>  </**form**> </**body**> </**html**> |

（2）Action类：只要写好属性即可：

|  |
| --- |
| **package** com.zhang.action;  **import** org.apache.logging.log4j.LogManager; **import** org.apache.logging.log4j.Logger;  **public class** DemoAction {  Logger **logger** = LogManager.*getLogger*();  *// 在Action中写好属性，属性名和表单提交的参数名一致，且提供setter和getter方法，Struts就能自动完成参数的接收* **private** String **userName**; *// 用户名属性* **private** String **pwd**; *// 密码属性  // 业务处理方法* **public** String login() {  **logger**.info(**"开始执行login()方法"**);  **logger**.info(**"接收到userName参数值：{}"**, **userName**);  **logger**.info(**"接收到pwd参数值：{}"**, **pwd**);  **return "success"**;  }  *// getter和setter* **public** String getUserName() {  **return userName**;  }  **public void** setUserName(String userName) {  **this**.**userName** = userName;  }  **public** String getPwd() {  **return pwd**;  }  **public void** setPwd(String pwd) {  **this**.**pwd** = pwd;  } } |

（3）struts.xml的配置：

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* **<!DOCTYPE struts PUBLIC  "-//Apache Software Foundation//DTD Struts Configuration 2.5//EN"  "http://struts.apache.org/dtds/struts-2.5.dtd"*>*** <**struts**>  <**package name="demo" extends="struts-default"**>  <**action name="login" class="com.zhang.action.DemoAction" method="login"**>  <**result name="success" type="redirect"**>/index.jsp</**result**>  </**action**>  </**package**> </**struts**> |

我们发现这样获得参数方便很多，并且我们使用POST或者GET方式请求，接收到的参数都没有乱码，这也是因为Struts默认采用UTF-8编码。

例2：添加一个人员信息。希望Struts能自动将提交的参数值封装成一个对象。

Struts是可以实现这样的功能的，只需要Action中属性的类型是相关的类即可。struts.xml自行配置。

（1）User类：

|  |
| --- |
| **package** com.zhang.action;  **public class** User {  **private** String **name**; *// 姓名* **private int age**; *// 年龄* **private** String **gender**; *// 性别  // 必须写getter和setter方法。自行写。还有写好toString()方法。* } |

（2）Action类：

|  |
| --- |
| **package** com.zhang.action;  **import** org.apache.logging.log4j.LogManager; **import** org.apache.logging.log4j.Logger;  **public class** DemoAction {  Logger **logger** = LogManager.*getLogger*();  *// User类型属性。* **private** User **userinfo**;  *// 业务处理方法* **public** String add() {  *// 其中能接收到user信息* **logger**.info(**"接收到用户信息：{}"**, **userinfo**.toString());  **return "success"**;  }  *// getter和setter* **public** User getUserinfo() {  **return userinfo**;  }  **public void** setUserinfo(User userinfo) {  **this**.**userinfo** = userinfo;  } } |

（3）JSP文件中，参数名必须用“对象名.属性名”方式，即：

|  |
| --- |
| <%@ **page pageEncoding**="**UTF-8**" **language**="**java**" **isELIgnored**="**false**" %> <**html**> <**head**>  <**meta charset="UTF-8"** />  <**title**>添加</**title**> </**head**> <**body**>  <**form action="${**pageContext.request.contextPath**}/add.action" method="POST"**>  <**p**>用户名：<**input type="text" name="userinfo.name"**></**p**>  <**p**>年龄：<**input type="number" name="userinfo.age"**></**p**>  <**p**>  性别：  <**select name="userinfo.gender"**>  <**option value="男"**>男</**option**>  <**option value="女"**>女</**option**>  </**select**>  </**p**>  <**p**><**input type="submit"**></**p**>  </**form**> </**body**> </**html**> |

## 1.5 开发Action的几种方式

方式1：Action类继承ActionSupport，这样可以使用SUCCESS等常量。这是最常用的方法。

方式2：实现Action接口，也可以使用SUCCESS等常量。

方式3：直接写Action类，不继承任何类，也不继承任何接口。

# 2. Struts常用配置

## 2.1 路径配置

struts.xml中，package标签用于定义一个包，包的作用就是用来管理action。在package中还可配置namespace属性，就是名称空间。名称空间默认是“/”，如果配置了名称空间，那么名称空间也会作为action路径的一部分。即此包中的所有action都在这个“路径”中。

例如：

|  |
| --- |
| <**struts**>  <**package name="demo" extends="struts-default" namespace="/user"**>  <**action name="add" class="com.zhang.action.DemoAction" method="add"**>  <**result name="success" type="redirect"**>/index.jsp</**result**>  </**action**>  </**package**> </**struts**> |

这样，add方法的访问路径就为：localhost:8080/应用名/user/add。注意，namespace一定要加上斜杠。

## 2.2 Struts常量

可在struts.xml中对Struts本身进行一些配置，很多配置可通过Struts常量来控制。Struts默认的常量配置在struts-core.jar中，进入org.apache.struts2包中即可看到一个default.properties文件，此文件中就定义了所有常量和默认值。

比如定义了“struts.action.extension=action,,”，这就是设置action后缀的。设置成“action,,”的意思是，请求既能加上action后缀，也能什么都不加。因此我们的访问路径既能写成xxx.action，也能直接写xxx。

我们可以在struts.xml中使用constant节点自行配置这些Struts常量。例如：

|  |
| --- |
| <**struts**>  <**constant name="struts.action.extension" value="html,aspx"**></**constant**>  <**package name="demo" extends="struts-default" namespace="/user"**>  ...  </**package**> </**struts**> |

上面就设置了action的后缀是html和aspx。此时，访问action必须加上html或者aspx，比如：localhost:8080/应用名/user/add.html。这样可以“模拟”应用是“静态页面”和.NET应用。

Struts常量都可用上述方法自行配置：在constant节点中写name属性和value属性，name写常量名，value写自定义值。下面是常设置的一些常量：

|  |
| --- |
| *<!-- 设置请求数据的编码，默认就是UTF-8 -->* <**constant name="struts.i18n.encoding" value="UTF-8"**/> *<!-- 设置访问后缀 -->* <**constant name="struts.action.extension" value="action,,"**/> *<!-- 设置struts.xml自动重载。推荐在开发时使用。这样当修改struts.xml时，不用重启Tomcat -->* <**constant name="struts.configuration.xml.reload" value="true"** /> *<!-- 上传文件大小限制，单位是字节 -->* <**constant name="struts.multipart.maxSize" value="10701096"**/> *<!-- 开启开发者模式，可在页面打印出详细错误信息 -->* <**constant name="struts.devMode" value="true"** /> *<!-- 与spring集成时，指定由spring负责action对象的创建 -->* <**constant name="struts.objectFactory" value="spring"** /> |

## 2.3 导入其他struts配置文件

若项目较大，把所有的action都配置在一个struts.xml中会显得混乱并且不好协同工作。Struts支持在struts配置文件中导入其他地方的struts配置文件。我们可以在每个action包下新建xml文件来配置本包所有的action。然后在struts.xml中使用include来引入这些文件。

例如：

（1）com.action1下面的action1.xml文件：

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"** *?>* **<!DOCTYPE struts PUBLIC  "-//Apache Software Foundation//DTD Struts Configuration 2.5//EN"  "http://struts.apache.org/dtds/struts-2.5.dtd"*>*** <**struts**>  <**package name="user" extends="struts-default" namespace="/"**>  <**action name="login" class="com.action1.Action1" method="login"**>  <**result name="login"**>/index.jsp</**result**>  </**action**>  </**package**> </**struts**> |

（2）com.action2下面的action2.xml文件：

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"** *?>* **<!DOCTYPE struts PUBLIC  "-//Apache Software Foundation//DTD Struts Configuration 2.5//EN"  "http://struts.apache.org/dtds/struts-2.5.dtd"*>*** <**struts**>  <**package name="teacher" extends="struts-default" namespace="/"**>  <**action name="viewList" class="com.action2.Action2" method="viewList"**>  <**result name="success"**>/success.jsp</**result**>  </**action**>  </**package**> </**struts**> |

（3）struts.xml中就可引入上述文件。struts.xml中存放一些公用配置，比如常量的配置。

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* **<!DOCTYPE struts PUBLIC  "-//Apache Software Foundation//DTD Struts Configuration 2.5//EN"  "http://struts.apache.org/dtds/struts-2.5.dtd"*>*** <**struts**>  *<!-- 公用配置 -->* <**constant name="struts.action.extension" value="action,,"**/>  *<!-- 引入配置 -->* <**include file="com/action1/action1.xml"** />  <**include file="com/action2/action2.xml"** /> </**struts**> |