# 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。我们首先学习Struts框架。

## 1.1 从Servlet到Struts

在使用Servlet开发时，通常需要为每个请求写一个Servlet处理类，若项目较复杂，新建的Servlet类会非常多。如果不想这么麻烦，可以请求同一个Servlet，那么要规定请求时额外携带一个参数，Servlet根据该参数执行不同的逻辑处理。

而且我们发现，Servlet中一般就是“获得参数 - 处理 - 返回视图（如JSP）”这样固定的写法。也就是说，Servlet在MVC开发模型中充当的是控制器M，我们这里介绍的Struts就是一个基于MVC模式开发的框架（属于三层架构表现层、业务逻辑层和数据访问层中的表现层框架），使用Struts框架就可较好地解决上述提到的问题。开发者可以基于框架开发实现自己的业务功能代码，框架的目的是把开发者从繁琐的、重复性的代码中解脱出来，提高开发效率，使开发者专注于业务。

所谓框架，可以说是一种“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/> 。

## 1.2 Filter在Struts中的应用

我们先对比一下Servlet与Filter。Servlet可作为控制器，提供了init()、service()和destroy()等方法，其中通过service方法能拿到请求和响应对象进行处理。并且，Servlet是单例的。

Filter是过滤器，提供了init()、doFilter()和destroy()方法，其doFilter方法中不仅能拿到请求和响应对象，还能控制请求是否放行（即继续执行）。Filter也是单例的，可以说，Filter完全能实现Servlet的功能，甚至比Servlet功能更加强大。在Struts框架中，实际上就是使用了Filter作为控制器，进而实现了各种各样的功能。在下面我们将会看到，使用Struts首先要配置好它的过滤器，其中执行的“Action”其实也是由框架封装产生的，本质上使用的是Filter作为控制器。

# 2. Struts第一个案例

（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 | OGNL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，现在就用Struts进行开发。在Struts中，一般通过编写动作类（Action类）来处理请求和响应，开发动作类相当于编写控制器代码。

Struts中，一个Action类可处理多个请求。例如此时新建一个名叫DemoAction的动作类，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"**;  } } |

为了能使用户请求连接可访问到此方法（请求连接映射到此方法），还需要对Struts的访问进行专门的配置。需要在src文件夹下（应用的构建路径顶端）新建一个struts.xml配置文件，框架会自动加载这个文件。其中可配置访问路径及其动作类的方法，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可用于管理不同模块的action，分模块开发时常用。每个package的name要求在应用中是唯一的，即配置多个package时名称不能重复，否则导致访问出错。  extends表示继承，一般继承自struts-default，否则某些功能不可用 -->* <**package name="demo" extends="struts-default"**>  *<!--在package中可写多个action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，就是浏览器的访问路径。  class：指示该action的动作类，这里写为DemoAction。  method：指示该action对应的方法，就是add方法（DemoAction中的）。  通过这样的配置，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中的result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，则会跳转到对应的页面，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，这样当处理方法返回不同值时，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（访问Struts请求时一般会加上.action后缀，即localhost:8080/应用名/expAdd.action）。访问后可以看到相应的日志输出。从该例子可看出Struts解决了我们开头提到的两个问题，简化了开发。

# 3. Struts的执行流程

（1）服务器启动时，Tomcat加载应用的web.xml文件。由于web.xml中配置了Struts过滤器，因此Tomcat将会创建该应用的Struts核心过滤器对象，并执行初始化方法。

（2）进行初始化时，Struts2会按需加载以下配置文件：

default.properties：默认配置文件，在org.apache.struts包中（只能看），

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（只能看），

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

struts.xml：开发者自行编写的配置文件，在应用的构建路径顶端，

struts.properties：开发者自行编写的配置文件，在应用的构建路径顶端，但推荐在上面的struts.xml中配置，

web.xml：最后加载web.xml中的给过滤器设置的参数，可以在这里设置参数，但不推荐，推荐一律在struts.xml中编写。

需要注意：后面的配置会覆盖前面的同名配置。

以上配置的加载顺序有助于后面知识的理解（例如配置Spring和Struts2集成时）。

（3）每次浏览器访问时，服务器会根据访问路径找到相应的action，并创建动作类对象（用户每次访问都会创建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。

下面附上Struts中的默认配置文件struts-default.xml，此文件在struts-core.jar中。大概内容和解释如下：

|  |
| --- |
| *<?***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。注意，没有action子元素的package才可声明为抽象包。struts-default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="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="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**> |

说明：

上述文件中，result-types节点中定义了许多的结果类型，这个在下面的结果视图类型中做详细讲解。此外，文件中还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。关于拦截器，后面也会做点解释。

# 4. 封装请求参数

在默认拦截器中，提供了参数拦截器（params），这个参数拦截器的作用就是把请求参数直接封装到动作类的属性中，使开发者能十分方便地获取到请求参数。下面就讲解如何在Struts中封装请求参数，共有三种方式，掌握它们十分重要。

## 4.1 使用动作类作为模型对象

使用动作类作为模型对象，就是直接将要封装的参数写在动作类的成员变量中，并且要提供getter、setter方法。这和第一个例子中的做法是相同的，成员变量不需要通过new进行初始化，由Struts自动完成。

例：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**> |

这样获得参数很方便，POST请求接收到的参数也无乱码，这是因为Struts默认采用UTF-8编码。

## 4.2 动作类和模型分开

动作类和模型分开就是不直接在动作类中写参数属性，而是用一个模型类接收，模型类中的属性和表单中要提交的属性是对应的。这样动作类看上去更加清晰，推荐使用。

例：添加一个人员信息。

这里让Struts将提交的参数封装成一个模型对象（即人员信息）。首先编写模型类User：

（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）动作类中使用上述模型，提供getter和setter方法：

|  |
| --- |
| **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）表单中，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**> |

（4）自行写struts.xml中的action配置。

通过这样，动作类也能成功接收到参数。这里，动作类中的模型也不需要通过new先进行创建，Struts框架会先调用模型的getter方法判断对象是否为null，如果是null就自动创建对象。

## 4.3 动作类和模型分开（使用模型驱动）

封装参数第三种方式就是使用模型驱动（ModelDriven）。使用模型驱动也是将动作类和模型属性分开，也是需要模型类对象的，但需要满足以下条件：

（1）动作类要实现ModelDriven<T>接口，其中的泛型T就是模型对象类型。

（2）动作类中提供模型对象属性及其setter、getter方法，这点和方式二类似，注意的是，这时模型对象需要使用new创建好，并且需要实现getModel()方法，这个getModel()方法中只要将模型对象返回即可。

（2）表单的提交参数直接写模型对象的属性，这点和方式一相同。

例如我们同样完成上面方式二中的案例。

首先模型类、struts.xml自行编写，基本是相同的，但动作类写为：

|  |
| --- |
| *// 动作类实现ModelDriven<T>。T就是模型对象类型。如果想使用ActionSupport类中的功能，也能再继承该类* **public class** UserAction **extends** ActionSupport **implements** ModelDriven<User> {  Logger **logger** = LogManager.*getLogger*();   *// 模型对象，要创建好，并提供getter、setter方法。* **private** User **user** = **new** User();   **public** User getUser() {  **return user**;  }   **public void** setUser(User user) {  **this**.**user** = user;  }   *// 实现的getModel方法要返回模型对象* @Override  **public** User getModel() {  **return user**;  }  *// action方法* **public** String add() {  *// 其中可拿到封装好的user对象* **logger**.info(**"接收到用户信息：{}"**, **user**.toString());  **return "success"**;  } } |

同样，JSP中的表单直接写为模型的属性：

|  |
| --- |
| <**form action="${**pageContext.request.contextPath**}/addUser.action" method="post"**>  姓名：<**input type="text" name="name"**>  性别：<**input type="text" name="age"**>  年龄：<**input type="text" name="gender"**>  <**input type="submit"**> </**form**> |

这时动作类也是能拿到封装好参数的。

上述三种方法在实际开发中都是可以使用的，在一个动作类中也能混用上述三种方法，遵循简单、代码简洁的原则使用。

总结开发Action的几种方式（不是封装参数的几种方式，虽然也是三种）：

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。常用的常量：

（1）SUCCESS：即“success”，表示一切正常；

（2）ERROR：即“error”，动作方法执行遇到异常，转向错误提示页面；

（3）NONE：即“none”，动作方法执行后，不转向任何的结果视图。当然也可在动作方法中返回null，例如文件下载、AJAX数据返回就是不需要转向视图的。

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。

## 4.4 封装数组和集合类型的数据

批量处理数据时，常会在表单中上传多个同名的参数，例如批量删除学生时，需要上传多个学生的编号id，但都是通过一个参数名id上传的，例如：

|  |
| --- |
| <%@ **page contentType**="**text/html;charset=UTF-8**" **language**="**java**" %> <**html**> <**head**>  <**title**>批量删除学生</**title**> </**head**> <**body**> <**form action="${**pageContext.request.contextPath**}/deleteStudent.action" method="post"**>  <**table**>  <**tr**>  <**td**>  学号：<**input type="text" name="id" value="001"**>  </**td**>  <**td**>  姓名：<**input type="text" name="name" value="张三"**>  </**td**>  </**tr**>  <**tr**>  <**td**>  学号：<**input type="text" name="id" value="002"**>  </**td**>  <**td**>  姓名：<**input type="text" name="name" value="李四"**>  </**td**>  </**tr**>  </**table**>  <**input type="submit"**> </**form**> </**body**> </**html**> |

上面表单中就可能要传递两个学生信息，且参数名都是id或者name。

这时，只能让动作类中封装成数组类型或者集合类型。这里为了演示，把id信息封装成String数组，把name信息封装成List<String>集合。

如果使用“动作类作为模型对象”的方式，则直接在Action类中这么写：

|  |
| --- |
| **public class** StudentAction {  Logger **logger** = LogManager.*getLogger*();   *// id数组* String[] **id**;  *// name集合* List<String> **name**;   **public** String delete() {  **logger**.info(**"接收的id学号数组：{}"**, Arrays.*toString*(**id**));  **logger**.info(**"接收的name姓名集合：{}"**, **name**.toString());  **return "success"**;  }   *// 提供getter和setter方法* **public** String[] getId() {  **return id**;  }   **public void** setId(String[] id) {  **this**.**id** = id;  }   **public** List<String> getName() {  **return name**;  }   **public void** setName(List<String> name) {  **this**.**name** = name;  } } |

如果使用动作类和模型分开的方式，则先定义好模型类：

|  |
| --- |
| **public class** Student {  **private** String **id**;  **private** String **name**; // setter getter toString... } |

动作类为：

|  |
| --- |
| **public class** StudentAction {  Logger **logger** = LogManager.*getLogger*();   **private** List<Student> **studentList** = **new** ArrayList<>();   **public** String delete() {  **logger**.info(**"接收的id学号数组：{}"**, **studentList**);  **return "success"**;  }   *// getter setter* **public** List<Student> getStudentList() {  **return studentList**;  }   **public void** setStudentList(List<Student> studentList) {  **this**.**studentList** = studentList;  } } |

则此时的表单参数写为：

|  |
| --- |
| <**form action="${**pageContext.request.contextPath**}/deleteStudent.action" method="post"**>  <**table**>  <**tr**>  <**td**>  学号：<**input type="text" name="studentList[0].id" value="001"**>  </**td**>  <**td**>  姓名：<**input type="text" name="studentList[0].name" value="张三"**>  </**td**>  </**tr**>  <**tr**>  <**td**>  学号：<**input type="text" name="studentList[1].id" value="002"**>  </**td**>  <**td**>  姓名：<**input type="text" name="studentList[1].name" value="李四"**>  </**td**>  </**tr**>  </**table**>  <**input type="submit"**> </**form**> |

# 5. 结果视图类型

配置action时，result节点中可通过type属性设置跳转的类型，这些类型实际上都在struts-default.xml文件中定义了（是在result-type节点中定义的）。常用如下结果类型：

（1）redirect：重定向到视图

（2）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，则要为该result节点设置好参数，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。

（3）chain：用于转发到另一个动作。注意这里转发到的是动作，而不能是JSP。例如：

|  |
| --- |
| <**action name="delete" class="studentAction" method="delete"**>  <**result name="success" type="chain"**>deleteStudent</**result**> </**action**> <**action name="deleteStudent" class="studentAction" method="deleteStudent"**>  <**result name="success"**>/index.jsp</**result**> </**action**> |

则访问delete后，delete处理成功则会转发到deleteStudent这个action再处理，即经过两个action。chain和redirect是不同的，chain转发会获取到之前的request和response对象。同样，若要转发到其他名称空间的Action，也要加上actionName和namespace参数。

（4）dispatcher：转发到视图，这是默认的。转发就意味着结果页面能获取request域对象数据。还可以这样设置转发到的视图：

|  |
| --- |
| <**result name="success" type="plainText"**>  <**param name="location"**>/index.jsp</**param**> </**result**> |

即设置location属性。为什么能设置location，以及前面chain中的param等属性呢？这是因为在每个result-type结果类型定义对应的类中（例如ServletDispatcherResult类），就有相应的属性（即setLocation），因此可通过设置location属性来进行“注入”。

那为什么能在dispatcher类型的result标签内容中直接写视图路径呢？这是因为location属性是默认的（可以在源码中看到default），因此这样写就默认设置的是location属性。

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。

（5）stream：流，文件下载时需要用到这个结果类型。

（6）plainText：以纯文本的形式展现页面。可用于输出源码。如果我们将type设置为这个，那么JSP页面就直接输出文本代码了。

（7）httpheader：用于输出http协议的消息头。（即没有消息正文输出）

# 6. Struts常用配置

## 6.1 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。

## 6.2 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，且此时上面的属性struts.configuration.xml.reload会自动设置为true自动重载 -->* <**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"** />  *<!-- 设置浏览器是否缓存静态内容，默认为true，但在开发环境中最好设置成false -->* <**constant name="struts.serve.static.browserCache" value="false"** /> |

## 6.3 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**> |