**第三部分：Strut2**

**配置步骤（eclipse）：**

1、把strut2解压的apps下strut2-black.war压缩包的WEB-INF\lib下所有jar包复制到自己web应用的WEB-INF\lib下。 若是自己新建的USER LIBERARY,则需要deploy到WEB-INF\lib下，这一步需要手动设置，否则有些jar包运行会出错。

2、修改web.xml添加strut2的核心filter：

<filter>

<filter-name>struts2</filt/er-name> //只是起的一个名字，无实际意义

<filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class> //代表所用的filter具体位置，一般不会变化

</filter>

<filter-mapping> //映射关系，哪个filter可以过滤哪个url

<filter-name>struts2</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

3、把strut2解压的apps下strut2-black.war压缩包的WEB-INF\classes下的struts.xml复制到Web下的src目录下，并且修改：

<struts>

<constant name="struts.enable.DynamicMethodInvocation" value="false" />

<constant name="struts.devMode" value="true" /> //开发模式

这个开发模式使得可以在页面中看到strut错误的具体信息**（在发布的时候必须关闭，漏洞）**

<package name="crazyit" namespace="/\*\*" extends="struts-default"> //包名字只是为了区别，namespace则是url必须是/\*\*/xxx.action才会响应，原来是xxx.acton会响应，用来区分名字相同的action。

<action name="\*" class=”package.Action”> //过滤器工作原理就是根据action name去找对应class action类来处理

<result name=”success” type=”dispatcher”>/pages/success.jsp</result>//根据action返回值确定result，type为视图类型（dispatcher为转发，保留原来的request对象和数据；redirect为重定向，会失去这次请求的数据，当然包括action处理结果，其本质上是重新发送一次请求，所以不能制定WEB-INF下资源；redirectAction重定向到另一个在xml中配置的action）。result路径/是绝对路径，不加/是相对路径(相对于action)。action到action(两者没什么联系，**输入错误要求重新输入**)用重定向，使传递的数据不丢失路径写\*\*.action?data=${data}。chain用来将Action执行完之后链接到另一个Action中继续执行(两者是前后相关的,**连续处理一件事**)，新的Action使用上一个Action处理后的上下文（ActionContext）。

</action>

</package>

</struts>

**Action类工作原理：**

Filter类调用对应的action处理用户http请求（url最后和action名字一样的则会调用此action处理此url，url以action为后缀的自动略去action匹配），在此action类中需要有和http请求参数name一样的成员变量和对应的get/set方法，然后自动识别，最后默认调用execute()方法返回处理结果。

Action默认就是一个request（难怪可以自动识别http中的参数，两者都是request），即所有里面的参数都可以通过request获取，访问其余Servlet api通过ActionContext类获取函数才行。

Ps:只有配置文件.xml修改之后需要重启tomcat,其余都是热启动，自动识别。

**让一个Action可以作多种处理，不单是execute()方法:**

1. 动态调用： url的最后结尾设置成 login！regist name是login，但作用是调用login这个action下regist()函数处理，而不是execute()。
2. 通配符：在strut2.xml文件中action name=\*Action,method=”{1}”。就会根据你传的\*来定位你说需要的方法，这里没有默认execute,所有方法都需要自己定义名字。
3. 设置默认处理action，但请求的url没有匹配的action时，自动调用默认action。但是这个现在不能和通配符一起使用（bug）。

**<result>标签实现动态调用：**

可以使用通配符，实现动态调用，但这个是根据用户url确定的。

使用OGNL--${属性名}等操作，实现根据action处理的属性值来决定result中操作。

可以使用全局<global-results>标签结合<result>，实现action匹配但是action下面没有result时，启用默认result。

**Package标签必须按照顺序写：**

default-action-ref  
default-class-ref  
global-results  
global-exception-mappings  
action\*(就是所有的action放到最后)

**异常处理：但现在有了ajax之后，完全可以在js中完成操作，这个异常处理基本不需要了**

在action方法中尽管抛出不同的异常，在stuts2.xml配置处理信息，主要依靠<exception-mapping exception=“//exception类型”result=”//抛给哪个result接受”>和<global-exception-mappings>这两个标签。

Ps:全局异常的result不要抛给局部的result，局部异常可以给全局。

**Action解析http请求（基于LGNL表达式：页面中的name可被Action可识别成成员变量）：**

http请求中参数全是string类型，struts2中利用自己内建类型转换器可以自动识别http请求参数并转化为Action中对应成员变量，按照javabean格式，就可以通过request（action中成员变量就是request中map）获得key/value。

http请求参数还可以写成User.name User.age等复合类形式，struts2照样能够识别，只不过在对应的Action中需要有User类。还可以是Map集合，User[“one”].name User[“one”].age这种形式（但必须是OGNL表达式）🡺Map<String,User>。或者List<User>，User[0].name User[0].age。

但是若是去掉泛型，则Action无法自动识别，需要添加一个properties的配置文件，放在Action相同位置，strut2框架可以自动识别。

但若是http请求中不是基于OGNL表达式，但是也想实现String🡺User，则需要自己写类型转换器（默认若是可以转则不会使用自定义的转换器，即一个成员变量优先响应系统默认转换，无法默认转换时才会启用自定义），extends DefaultTypeConverter类重写convertValue方法，或者利用更好的StrutTypeConverter,重写convertFromString convertToStrin逻辑更加清楚。然后部署这个转换器:

1. 局部部署（针对某个Action）：在Action对应的properties文件中写user=conventer.UserConverter。
2. 全局部署（针对所有Action中User类型，让User变的和List、Map、String一样可以自动识别）：在src下添加xwork-conversion.properties文件，beans.User=converter.UserConverter。

转换失败后，系统自动启用<result name=” input “>的逻辑视图。

**Struts2输入校验（服务器端校验，即对http请求转化为Action中对应属性校验）: 但是必须结合标签才可以，在前后端分离时代不起作用**

配置ActionClassName-validation.xml文件，放在与Action同一个目录下。分为字段校验和非字段两种方式，仅排版格式不同，下面是字段校验格式：

<validators>

<field name="username">

<field-validator type="requiredstring">

<param name="trim">true</param>//去掉空格

<message>用户名不能为空!</message>

</field-validator>

</field>

</validators>

问题：一个Action可能有许多不同处理逻辑，会需要不同的校验方式，则可配置ActionClassName-ActionAlisaName-validation.xml文件是针对struts2.xml中<action name=” ActionAlisaName”>的校验方式，放在与Action同一个目录下。校验顺序是全局🡺个体。

这个作为此xml文件开头必写:

<!DOCTYPE validators PUBLIC

"-//Apache Struts//XWork Validator 1.0.3//EN"

"http://struts.apache.org/dtds/xwork-validator-1.0.3.dtd">

若是需要对复合类型校验User等，则在对应的Action目录下的validation.xml中添加

<field name="User">

<field-validator type="visitor">

<param name="context">userContext</param>

<message>User校验信息错误</message>

</field-validator>

</field>

但是这并没有实现User中信息校验，为此还需要在User.java目录下添加

User-userContext-validation.xml文件，里面内容和ActionClassName-validation.xml这种一样。

校验顺序: 首先执行类型转换 ，然后执行校验框架(validation.xml) ,执行特定方法对应的validate验证（test,validateTest） 编码验证 ,最后执行validate(); 编码验证。如果在以上所有过程之中发现任何错误，都不会再去执行execute()，页面转向struts.xml中input对应的页面。要在jsp中获取校验失败信息<message>，则需要使用<s:fielderror>标签或者表单直接是利用标签生成的则自动显示。

**Struts2的拦截器机制：只能对Action有用，而filter可以对所有请求过滤**

拦截器完成了struts2 70%的工作（Action是处理核心），从接收到http请求，解析http中参数，类型转换，Action方法调用，数据校验，异常处理，一直到调用Action中对应方法全是由内建的拦截器完成（在struts-default.xml文件中）

**配置拦截器：**

自己配置对action拦截器需要在struts2.xml中定义，然后再对应的<action>中引用，如果显示配置拦截器则默认拦截器（默认对所有Action都能用）失去作用，为了仍可以用默认拦截器，显示声明使用defaultStack: 拦截器栈

<interceptors>

<interceptor name=*"mySimple"* class=*"interceptor.SimpleInterceptor"*>

<param name=*"name"*>简单拦截器</param>

</interceptor>

<interceptor-stack name=*"mySimpleStack"*>

<interceptor-ref name=*"defaultStack"*>

</interceptor-ref>

<interceptor-ref name=*"mySimple"*>

<param name=*"name"*>改名后的拦截器</param>

</interceptor-ref>

</interceptor-stack>

</interceptors>

<action name=*"\*Action"* class=*"actions.LoginAction"* method=*"{1}"*>

<interceptor-ref name=*"mySimpleStack"*></interceptor-ref>

<result name=*"success"*>/pages/success.jsp</result>

<result name=*"input"*>/pages/loginform.html</result>

</action>

**编写自定义拦截器类：**

拦截器类编写需要实现接口（涉及到初始化数据）或者继承类AbstractInterceptor（只需要重写intercept(ActionInvocation arg0)方法，arg0参数是被拦截的方法）

这个拦截器运行在类型转换之后，Action所有方法调用之前，所以在拦截器中可以获取Action所有数据,包括ActionContext实例，这样通过ActionContext实例又可以访问4个全局的Servlet Api获取里面的数据。

但上述拦截是对Action下所有方法都拦截，想要对特定方法拦截，需继承MethodFilterInterceptor类重写doInterceptor,处理逻辑不变，只是需要在struts2.xml中配置哪些方法不拦截，哪些拦截：

<interceptor name=*"mySimple"* class=*"interceptor.SimpleInterceptor"*>

<param name=*"includeMethods"*>regist</param>

<param name=*"excludeMethods"*>login</param>

</interceptor>

**拦截器实用功能：权限检查，基于拦截器可以获取Action和ActionContext数据才能实现**

可以在每个Action开始执行逻辑代码之前先进行权限处理（比如判断username.equal()，age==），但这不利于代码复用，所以采用配置方式，即把这个功能绑定在拦截器上。

注意：拦截器中若是调用了invoke()，控制权已经失去了，即Action中的返回值才会决定<result>的逻辑资源，若是没调用invoke()，则由拦截器的返回值决定<result>

**Ajax和JSON支持：两者在struts中完美结合**

* jquery中发送ajax请求写法：

$.ajax({

type:"post",

url:"excuteAjaxJsonAction",//需要用来处理ajax请求的action

data: //设置数据源

* {name:'xyh',age:22} //简单的键值对，Action中变量name,age
* {array:[1,2,3,4,5,6]}//数组传递，Action中变量Integer[] array对应，

此时(traditional: true需要开启)，阻止深度序列化，否则变成array=Object

* 复杂形势的json数据，{data: JSON.stringify(data)},包装成json字符串,action中先通过request.getParamer(’data’)获得json字符串，通过JSONObejct/Array.fromObject()解析得到json对象,再通过json对象操作得到想要的数据。

dataType:"json",//设置需要返回的数据类型

success:function(data){

var d = $.parseJSON(data);//将字符串数据转换成json类型，得到d是js对象

},

error:function(){

alert("系统异常，请稍后重试！");

}//这里不要加","

});

* Action中接受json数据，并返回写法：

1. HttpServletRequest request = ServletActionContext.getRequest();

request.getParameter("name") 这种获取方法和jsp一样

1. 或者直接写到成员变量里，有对应的setter方法，即可自动对应，成员变量User，ajax中data:{user.name / user.age}都能识别，和jsp操作相同
2. Java中利用JSONObject/Array.fromObject(Bean,String,List,Map)，把对应的对象转化为JSON对象，或者直接自己新建JSONObject/JSONArray，最后.toString()传给result(在action中需要有对应的setter/getter方法)。

* Struts.xml配置方法：

1. <package name="HNSystem" extends="json-default">//json-default继承struts-default

<action name="\*JsonAction" class="actions.JsonAction" method="{1}">

<result type="json">

<param name="root">result</param>

</result>

</action>

1. 这里需要导入Struts2中额外支持JSON的包