// 学习的第一步是依葫芦画瓢

怎么说呢，对于我这种初学者而言，它的版本更新有点快。但别误会，我并没有它会影响我学习的意思在里头，只是当一项技术的版本迭代过快时，我就很难从根源上去了解它，很难从头至尾的掌握它。

要问为什么。。。。。。。我时间不够啊！

Struts是什么呢，一个基于MVC设计模式的Web应用框架（来源于百度百科），但说白了，就是处理HttpRequest与HttpResponse。但与Servlet不一样，使用Struts时，每一个请求都要先经过Struts的过滤器。

在web.xml中这样配置：

<filter>

<filter-name>Struts2</filter-name>

<filter-class>

org.apache.struts2.dispatcher.filter.StrutsPrepareAndExecuteFilter

</filter-class>

</filter>

<filter-mapping>

<filter-name>Struts2</filter-name>

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

</filter-mapping>

这里同时也是我讨厌它版本更新过快的地方了，因为这个<filter-class>中的类名会变动，初学者摸不着头脑。

与一般dynamic web project不同，除了web.xml外，还需要struts.xml文件。struts.xml文件位于Java Resources中的src文件夹下，注意，不是WebContent中的WEB-INF。这里扯一点题外话，我平时写的<servlet>、<filter>等标签，其实都是放在<web-app>标签内的。struts.xml也差不多，但它的最外围是<struts>标签。

<struts>

<package name=*"myPackage"* extends=*"struts-default"*>

<action name=*"myAction"* class=*"com.jojo.action.MyAction"*>

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

</action>

</package>

</struts>

应该找不出比上面更简单的struts.xml了，首先，<package>是什么完全不懂，为啥要继承也完全不懂。那个<action>可以类比Servlet里的那个“action”，它还有一个返回值，根据这个返回值（result）来发往对应的.jsp，默认用forward跳转。

它给我的感觉是把一个<servlet>、<servlet-mapping>拆成了多份。

Struts 2.5.8共有93个开发包，当我刚把它下下来的时候我以为全部都要引入。。。。。但核心包其实不多，一共也就那么几个，根据小峰视频及某网络博客整理如下：

* commons-fileupload （具体版本号我就不列出来了）
* commons-io
* commons-lang
* commons-logging
* freemarker
* javassist
* ognl
* struts2-core
* log4j-api

那个logging也是必须的，所以说版本变动伤脑筋呢，小峰视频里的那个版本不需要log也能启动\_(:з」∠)\_

2017.1.6

////////////////////////////////////////////////////////////////////////////////////////////

// The simplest case，大概。。。。

**public** **class** MyAction **extends** ActionSupport {

// 这个id我至今都不知道怎么用

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** String name;

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

@Override

**public** String execute() **throws** Exception {

**return** **null**;

}

}

首先是ActionSupport类，以后写的action类都要继承它，看名字就懂。这个类实现了Action接口（还有其他一大堆接口），功能很强大。

在action里头写getter与setter是有意义的，这样struts会自动获取数据，不像Servlet，还要写request.getParameter()。当然上面那个写法做例子讲还好，真这么写也是会出问题的，因为一般项目开发，一个model搞不好有几十个属性。替代方案是下面这个：

**Private** User user;

**Public** User getUser(){

return user;

}

**Public** void setUser(User user){

**this**.user = user;

}

然后前台页面处，假设有一个<form>表单：

<form action=*"myAction"* method=*"post"*>

<input type=*"text"* name=*"user.userName"* value=*""*>

</form>

像这样把name属性对应着写。想想挺神奇的，user根本就是个空引用，Struts不仅要设置数据，还要创建对象。

以上是获取单个javabean对象的方式，Struts还可以获取多个对象（同一类）的数据，这个功能在批量添加的时候非常有用。

前端这么写：

<input type=*"text"* name=*"user[0].userName"* value=*""*>

<input type=*"text"* name=*"user[1].userName"* value=*""*>

后端这么写：

**private** List<User> users;

**public** List<User> getUsers() {

**return** users;

}

**public** **void** setUsers(List<User> users) {

**this**.users = users;

}

还有一种情况是获取数目不定的字符串，比方说多选框。同样还是分成前端与后端来讲。

<input type=*"text"* name=*"hobby"* value=*""*>

<input type=*"text"* name=*"hobby"* value=*""*>

String[] hobby;

**public** String[] getHobby() {

**return** hobby;

}

**public** **void** setHobby(String[] hobby) {

**this**.hobby = hobby;

}

这个我写代码测试过，是可以嵌套的，比方说某javabean类中包含数组对象，那么前端处写成“user.hobby”这种形式就可以，后端只要写user的getter与setter，还是挺方便的。

除此之外还有一种设置获取数据的方式，模型驱动，这种方式只能设置获取一种model类数据，上面介绍的那两种可以设置复数个model类。用这种方式，action类还要在实现一个接口ModelDriven<T>，他奶奶的还是带泛型的。唉，以后有空再补吧。

2017.1.9

////////////////////////////////////////////////////////////////////////////////////////////

struts.xml 的详细配置，还是看初始的xml。

<struts>

<package name=*"myPackage"* extends=*"struts-default"*>

<action name=*"myAction"* class=*"com.jojo.action.MyAction"*>

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

</action>

</package>

</struts>

1. 先看<package>，name属性就是名字，没什么特别含义，开发时方便分类，就这样。extends属性，继承，但到底干嘛的现在还不清楚。。。。<package>里有一个比较特殊的属性namespace。

<package name=*"myPackage”*namespace=*"/fore”*extends=*"struts-default"*>

一旦加了namespace，再访问该package下的action就要在url里加入“/fore”。好处肯定有的，这样一来分类更加明确，最关键的是，省得起名字了。有namespace在action的url就是重名也无所谓。有好处的同时他也有陷阱，加了namespace后就不能随意使用相对路径了，因为url中多了namespace的值，像action中的result以后都得用绝对路径，不过好在struts.xml里也是能用el表达式的。

还有一个特殊的属性叫做abstract（抽象包），不写的话其值默认false。添加这个属性主要是为了自定义一些过滤器。和Java的abstract关键字类似，如果<package>的abstract属性为true，那么这个<package>里是不能放<action>的，只能用来继承。

2. <action name=*"myAction"* class=*"com.jojo.action.MyAction"*>

name和class也没什么好说，url及对应的处理类。特殊点的是method属性，如果没有明确指出method属性的话，那么action对应的处理类会默认执行execute()方法。一旦这么写：

<action name=*"my"* class=*"com.jojo.action.MyAction"* method=*"show"*>

那么针对这个请求MyAction类只会调用show()方法，所以show()方法不能随便写，返回值什么的要参照execute()方法。

3. <result>标签，这玩意其实也蛮复杂的。

<result name=*"success"*>

${pageContext.request.contextPath}/success.jsp

</result>

name属性要与Action类返回的字符串相对应，中间的内容就是要跳转的jsp页面，用不用绝对路劲要看<package>标签有没有namespace属性，不过我这边统一用绝对路径好了，省得烦。

除了name属性外，还有type属性。我个人是理解成转发方式，默认的话type=”dispatcher”就相当于request.getRequestDispathcer().forward(); 我曾经很纠结，因为Struts好像是把放在Action类里的数据统统转发了，而原本Servlet转发要自己显示调用request.setAttribute(); 其他的还有：

type=”redirect”客户端跳转，没什么好说的。

type=”chain”跳转至另一个Action类数据共享，相应的，<result>标签里不能写jsp页面，要用action的url。

type=”redirectAction”顾名思义，重定向至另一个Action类，数据不共享。

记住，type是绝对不可以少的。

Servlet里，转发通过request与response，这里头包含了jsp页面与Servlet类，Struts是把这两种转发方式，统统拆成4个来处理。

有些result是全局共用的，像错误页面，不可能给每一个<action>都添加错误跳转，会死人的。

<global-results>

<result name=*"error"*>error.jsp</result>

</global-results>

用法上没有太大区别，只是顺序问题没研究过，目前是把它写在<package>的最顶上。我还是选择相信和顺序没关系，都什么年代了，只不过按顺序写看的舒心。

4. <include file=*"otherModule.xml"*></include>

这个东西简称分模块调用，<include>标签可以类比JSP的静态包含，其存在意义为让开发人员过得舒坦点。。。。。一个大型项目（商业用）的struts.xml里搞不好有几十乃至上百个action，这时就算用<package>标签来分类看着也很吃力，因此将不同模块放到不同的xml文件里，并用<include>标签关联。应该算是为开发效率而牺牲运行效率吧。

1. 通配符

<action name=*"user\_\*"* class=*"comm.jojo.action.UserAction"* method=*"{1}"*>

每一个模块都会有增删查改，操作固然不同，但url却很相似。因此，使用通配符可以大大提升该部分的开发效率。这边的重点不在 \* 而是method=*"{1}"* 这句话的意思是取第一个通配符的内容，为什么要有编号呢，因为有些很变态的写法 \*\_\* （莫名喜感。。。。），1就是取第一个，2就是取第二个。

取到通配符内容后进入action执行相应的方法并返回，返回时还是有缺陷的，因为共用一个action的缘故，result上体现不出独立性来，所以通配符要考虑清楚后再用。不过，一般增删查改，有很复杂的页面处理。

2017.1.14

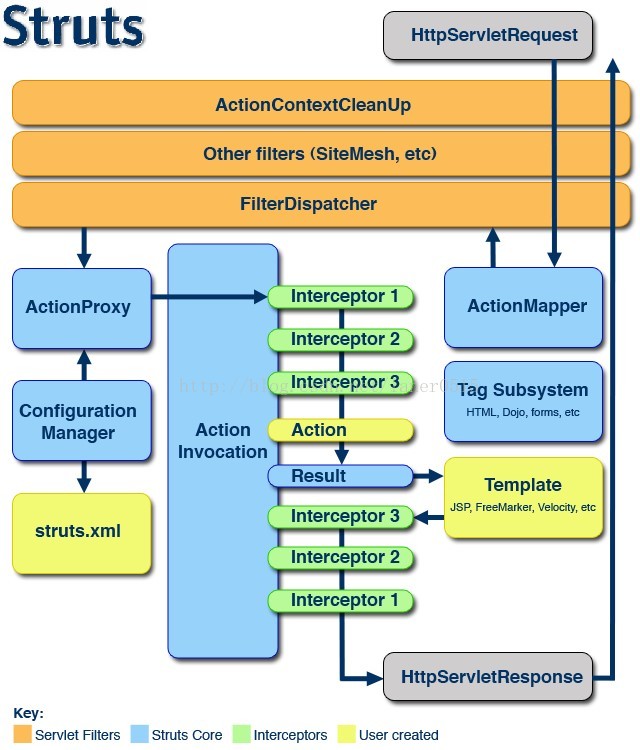
////////////////////////////////////////////////////////////////////////////////////////

生命周期

每请求一次Action，Struts就会创建一个Action实例，再请求，再创建。它不像Servlet，Servlet使用单例模式，创建一次后就一直存在，关闭程序后销毁。

////////////////////////////////////////////////////////////////////////////////////////

我觉得现在是时候贴这张图了。



// 那个interceptor1,2,3到底是啥？

这是Struts2的运行原理图，可以看到request与response之间有很多步骤，其实里面我还有一堆东西不懂，总之现在看interceptor（拦截器）。看图，它是在action之前及之后发生作用的，和过滤器是两个概念。

Struts2的拦截器是可插拔的，是AOP（Aspect Oriented Program）的体现。

// 这些理论我觉得蛮玄乎的，我理解为在程序运行过程中修改代码，但应该只是切换方法之类的，不可能真的“修改”，程序不编译就能运行？

Struts本身定义了很多拦截器，统统都在struts-default.xml里。顺带一提，这个xml文件里定义了很多东西，像result-type什么的。除了单个的拦截器外，Struts还定义了不少拦截器栈<interceptor-stack>。

<package name=*"myPackage"* extends=*"struts-default"*>

<interceptors>

<interceptor name=*"myInterceptor"*

class=*"com.jojo.interceptor.MyInterceptor"*>

</interceptor>

</interceptors>

<action name=*"my"* class=*"com.jojo.action.MyAction"*>

<result name=*"success"*>

${pageContext.request.contextPath}/success.jsp

</result>

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

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

</action>

</package>

其实吧，我觉得更多的还是一个配置问题。首先在<package>里配置自定义的拦截器类，然后在<action>里配置要用的拦截器

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

这一句不能忘，没了就会出错，那这是为什么呢？因为计算机很笨！别笑，我认真的，这个拦截器栈原本是默认使用的，但当我们显示的配置了自定义的拦截器时，Struts就不会再调用默认的拦截器栈了，这时就需要我在手动添上去。

2017.1.15

////////////////////////////////////////////////////////////////////////////////////////

// 学的东西越来越多了，头也越来越晕，Struts和Servlet还是有很多不同点的。我现在不明白啊，Action和Service到底什么关系，DAO又写在哪里？为什么光验证个用户就要这样跳来跳去的。

还，差的远呢。。。。。。。

使用Struts 2实现用户登录功能，共三种情况，登录成功；用户名与密码不能为空；用户名或密码错误。需要的类及jsp页面有：UserLoginAction，UserService，User，login.jsp，success.jsp，error.jsp。

步骤说白了都一样：

login.jsp->UserloginAction(调用UserService)->success.jsp/error.jsp

难点有三个，一是登录失败时如何将错误信息传入jsp页面，Servlet传值直接调用setAttribute()方法就好，Struts要烦一点，在Action内准备一个error字符串并配备相应的getter和setter，在自动传入jsp页面。

二是登录成功后要如何保存用户信息，Servlet操作很简单啊，从request处获取session对象然后session.setAttribute()就好。Struts烦一点，我就直接贴代码了：

ActionContext actionContext = ActionContext.*getContext*();

Map<String, Object>session = actionContext.getSession();

session.put("currentUser", **this**.user);

要先获取ActionContext对象（action上下文，我自己瞎翻的），看代码是单例模式因此获取方法是静态的。然后在从actionContext中获取session，这个session是个map，也跟Servlet不同。

三是struts.xml中<result>的配置，error还是放在<global-results>中比较好，到时候统一展示错误信息。

现在加功能了，验证登录权限，使用Struts拦截器来实现未登录用户不能访问的功能。说起来倒也不烦LoginInterceptor及struts.xml配置。

第一个，检测session中是否存在currentUser，在就放行，不在就转到错误页面，想好看点的话可以在错误页面添加一个登录按钮。所以这里又来了一个问题，这里不是Action不能自动传值，要怎么把错误信息传给error.jsp？

String error = "警告，尚未登录<br/><a href=\"userLogin\">点击登录</a>";

ActionContext actionContext2 = invocation.getInvocationContext();

// 。。。。。。。这句子好长

HttpServletRequest request =

(HttpServletRequest) actionContext2.get(ServletActionContext.***HTTP\_REQUEST***);

request.setAttribute("error", error);

首先要注意的是invocation.getInvocationContext();没有使用ActionContext的静态方法是因为要获取与拦截器相关联的ActionContext，然后就是ActionContext的get()方法，这里暂时不想多讲。

第二个，struts.xml的配置

<interceptors>

<interceptor name=*"login"*

class=*"com.jojo.interceptor.LoginInterceptor"*></interceptor>

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

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

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

</interceptor-stack>

</interceptors>

<default-interceptor-ref name=*"myStack"*></default-interceptor-ref>

首先在<interceptor>标签内部配置自定义的拦截器，然后在<action>内部通过<interceptor-ref name=””></interceptor>来使用自定义的拦截器，同时还要加上 <interceptor-ref name=”defaultStack”></interceptor> ，这一句是必须要加的。但由于LoginInterceptor的特殊性（大部分动作在执行前都需要登录验证），当<action>过多时使用<interceptor-ref>标签会很累。

因此，上面的选择是将自定义的拦截器与defaultStack合成一个新的拦截器栈，并将其设为默认，如此一来只要给不需要登录验证的动作加<interceptor-ref name=”defaultStack”></interceptor> 这一句就好。

拦截器是有其局限性在的，它只能对Action进行操作，比如我直接访问某jsp页面，拦截器是管不了的，这要靠过滤器，但Struts的核心控制器已经把所有的url都给过滤了，所以要怎么加呢？

还有一个解决方案是把某些jsp页面放在web-inf文件夹下。

小结：

1. 大部分内容还是放在<package>中配置的。
2. 不要把字打错。

2017.1.17

////////////////////////////////////////////////////////////////////////////////////////

值栈

值栈是对应每个请求对象的一套内存数据的封装，Struts 2 会给每个请求都创建一个值栈。值栈能够线程安全的为每个请求提供公共的数据存取服务。

把request，session，application什么的统统封装起来，这个理解起来还是蛮抽象的，特别是还存在这么一个接口com.opensymphony.xwork2.util.ValueStack

这是狭义上的值栈，以键值对的形式保存数据。

OGNL（Object Graph Navigation Language）

对象导航图语言，原本以为有多叼，搞半天还是一jsp标签库。。。。。神秘感瞬间没了。

<%@ taglib uri=*"/struts-tags"* prefix=*"s"*%>

// taglib，话说c标签库要怎么引入来着的？

<s:property value=*"name"*/>

// 获取值，这种写法是从ValueStack取值，那个狭义上的值栈

<s:property value=*"#parameters.name"*/>

// 这个写法是从请求参数处获取值，即url后跟的数据，那么问题来了，post方式提交的数据呢？

<s:property value=*"#request.name"*/>

// 从request.setAttribute()处取值，脑子里开始和EL表达式打架了

<s:property value=*"#session.name"*/>

// session.setAttribute()

<s:property value=*"#application.name"*/>

// application.setAttribute()

<s:property value=*"#attr.name"*/>

// 这个比较特殊吧，按顺序取值page,request,session,application。最好还是要加上范围，不然太混乱。

访问复杂对象

<s:property value=*"user.userName"*/> // 对象名加属性，访问javabean

<s:property value=*"users[0].name"*/> // 访问集合对象，主要是list

<s:property value=*"map[‘key’].name"*/>

// 访问Map，用键来访问。

唉，到目前为止与EL表达式都没啥区别，让人不禁感叹：它，到底，有什么卵用。然后来了，特殊功能来了，OGNL可以访问类的静态变量及静态方法（返回值）。

<s:property [value=](mailto:value=\"@com.jojo.common.MyStatic@myVarible\"/>)*["@com.jojo.common.MyStatic@myVarible"](mailto:value=\"@com.jojo.common.MyStatic@myVarible\"/>)*[/>](mailto:value=\"@com.jojo.common.MyStatic@myVarible\"/>)

// 访问静态变量，注意完整类名

<s:property [value=](mailto:value=\"@com.jojo.common.MyStatic@myFunc\"/>)*["@com.jojo.common.MyStatic@myFunc()"](mailto:value=\"@com.jojo.common.MyStatic@myFunc\"/>)*[/>](mailto:value=\"@com.jojo.common.MyStatic@myFunc\"/>)

// 虽然说是说访问静态方法，但本质上还是访问变量。而且访问静态方法比较烦，因为Struts 2默认不允许访问静态方法（涉及到安全，暂时不懂），要想成功访问要在struts.xml里配置

<constant name=*"struts.ognl.allowStaticMethodAccess"* value=*"true"*>

</constant>

附赠一个开启动态方法调用的配置语句：

<constant name=*"struts.enable.DynamicMethodInvocation"* value=*"true"* >

</constant>

小结：

1. 注意作用域，开头要加“#”

2017.1.18

////////////////////////////////////////////////////////////////////////////////////////

Struts 标签

// 然而我JSTL标签还没背全啊。。。。我还是喜欢JAVA代码，可惜换了Struts后写不了了，所以这个还是目前必学的。还有，为什么，在jsp里写Java代码不好呢？虽然确实蛮丑的。

* 数据类

<s:property value=””default=””escapeHTML=””>

// 这个标签见得多了，关系到OGNL。Value就是要输出的值，default为默认值，escapeHTML的值为true/false，不将内容当成HTML来处理的意思，默认为true，好伤心。属性不止上面那几个，懒得列了，用到再说。

<s:set var=””value=””scope=””>

// 设置值，唉。

<s:bean name=””var=””>

<s:param name=””value=””></s:param>

</s:bean>

// 创建bean对象，外层name是完整包名，var是变量名。内层name是类里头那些属性的名字，value是具体值，相当于构造函数了。bean与set标签设的值取的时候必须要加 # 这个我觉得蛮奇怪的。好像只有ValueStack（狭义值栈）里的值取时不用加 #

<s:date name=””format=””>

// 时间转换，name处取值，formate定义要转换的格式，如：yyyy-MM-dd

<s:a action=””namespace=””>

<s:param name=””value=””></s:param>

</s:a>

// 链接，还有一个<s:url>标签也是设链接的，但它太烦了，先按<s:a>套路写一串代码，在取个id，然后在用<s:a>来使用。还不如直接用<s:a>。

<s:include value=””></s:include>

// 动态包含，里妈个比的一会name一会value。。。。

* 控制类

<s:if test=””></s:if>

<s:elseif test=””></s:elseif>

<s:else></s:else>

// 这些标签比JSTL好用的多

<s:iterator value=””status=””></s:iterator>

// 话说c标签的<c:forEach>我用的也不熟啊

太多了。。。。。完全不想记啊

* 界面标签

把<form>，<select>，<radio>这些标签统统封装了一遍

// 我说，这玩意真的用得到嘛？

* 其它标签

<s:updownselect> <s:optiontransferselect>

小结：

1. struts标签不同于JSTL，是必学的。
2. 查文档去吧，小傻逼~

2017.1.19

////////////////////////////////////////////////////////////////////////////////////////

* 验证框架

// 主要用来验证表单内容，不得不说这个东西还是蛮有用的。JSP&Servlet开发，验证表单要么前端JS代码，要么后端Servlet写Java代码。现在Struts把验证这个过程单独列出来，解耦（我现在也只会这么理解）。

// 太久没看英文文档了，头晕，还是按照小峰视频慢慢学吧，等自己开始写项目了，在慢慢深入，话说我这手指头什么时候开始好啊？

所谓验证呢，我最熟悉的就是登录验证，用户名或密码不能为空。只不过举这个例子不太对，因为登录时需要进数据库查找用户名和密码，所以用户名空不空的直接交给action去做也没问题。注册倒是个好例子，比方说用户名得是那个格式，密码不得少于X位，等等。

<action name=*"userRegister"* class=*"com.jojo.action.UserRegisterAction"*>

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

<result name=*"input"*>register.jsp</result>

</action>

多了一个<result name=*"input"*>，这个是必须的，验证不通过时会通过这个<result>进入register.jsp并提示错误信息。

// 还是要多查文档啊。

然后是验证的关键，搞一个xml，命名规则是xxx-validation.xml（和某Action相关联）。

<validators>

<field name=*"user.userName"*>

<field-validator type=*"requiredstring"*>

<message>必须输入姓名</message>

</field-validator>

<field-validator type=*"stringlength"*>

<param name=*"minLength"*>6</param>

<param name=*"maxLength"*>19</param>

<message>用户名必须在${minLength}与${maxLength}之间 </message>

</field-validator>

</field>

</validators>

这里用的是字段型验证器，还有一个叫非字段型验证器，格式是这样的：

<validator type=*"email"*>

<param name=*"fieldName"*>user.Email</param>

<message>请按格式填写</message>

</validator>

我原本以为这个非字段型验证器中间可以加多个fieldName，结果不行，那还不如用字段型验证器，看着也清楚。

最后，除了Action类，前端jsp页面的表单必须用Struts标签，至于<s:fielderror/>和<s:actionerror/>，打印错误信息并显示在相应的地方。

Struts验证表达式，差不多是这种感觉。那个逻辑表达式貌似是OGNL，其结果必须是boolean型。

<validators>

      <validator type="expression">

         <param name="expression"> <![CDATA[(逻辑表达式)]]>

</param>

         <message>Failed to meet Ognl Expression  .... </message>

      </validator>

</validators>

* 自定义验证器

上面介绍的验证器，像email，int，requiredstring都是Struts自带的。不多，但这边蛮奇怪的，文档上说自带验证器15个，validators.xml里只有14个，少了short。

先不管他，自定义验证器如何生效呢？你果然懂了，xml呗。（xml满天飞啊~）首先建一个包com.jojo.validator把validators.xml复制进去，并添加自己写的验证器，格式如下：

<validator name=*""* class=*""*></validator>

验证器，比如说SensitiveWordValidator要继承FieldValidatorSupport类，看到support我就明白了。重写validate(Object object)方法。最后在相应的XXX-validation.xml中配置就完了。

2017.1.23

/////////////////////////////////////////////////////////////////

// 2017.2.2过去整10天了，10天，他妈的感觉像是1个月。时间其实很多，关键看把握。

Struts2其实我还没学完，草草收场。刚好，来回忆一下。Struts2给我最大的印象是各种xml文件，其中最重要的是struts.xml，这里头最基础的有<package>，<action>，<result>。然后是<interceptors>拦截器，这里头又有<interceptor>，<interceptor-stack>设置一个拦截器栈，以及<interceptor-ref>拦截器映射和强大的<default-interceptor-ref->。

// 完全不知道怎么回忆，我可能学了假的Struts 2……

根据文档，Struts 2通过interceptors，actions，results这三个核心来处理请求。每一个都由相应的 .xml文件或注解来配置。

下面看Struts2的运行图，跟着箭头走。

***1.*** 首先HttpServletRequest -> FilterDispatcher

org.apache.struts2.dispatcher.filter.StrutsPrepareAndExecuteFilter

我对这个类的最大印象就是它好长\_(:з」∠)\_。这是Struts 2的核心控制器，过滤一切请求，当然看名字也看出来了。这个过滤器之上还有两个ActionContextCleanUp与Other filters (Sitemesh, etc)，虽说我都没用过。Other filters是自己定义的，如果要用自定义过滤器，那么就必须在加上ActionContextCleanUp，形成一个三层过滤体系。我目前只引用过FilterDispatcher

***2.*** HttpServletRequest -> ActionMapper

ActionMapper是一个接口，它提供了HTTP requests与action invocation requests之间的映射（我实在不知道怎么翻，动作调用请求？），反之亦然。它会根据提供的HttpServletRequest来返回ActionMapping，ActionMapping是一个类，包含了<action>的所有信息，此类由框架调用。

ActionMapper不需要确保返回的ActionMapping是一个实在的action，除非你提供的请求是有效的。

***3.*** ActionMapper-> FilterDispatcher->ActionProxy

如果ActionMapper确认该请求需要由Struts进行处理，FilterDispatcher就会关闭之后的过滤器，并创建ActionProxy对象继续处理。

***4.*** ActionProxy-> ConfigurationManager->ActionInvocation

为了确定到底有哪一个action来处理，ActionProxy向ConfigurationManager查找，该对象通过struts.xml产生。这里个人猜测，每修改struts.xmlxml都要重启项目就是为了重新创建CofigurationManager，其他的像web.xml同理。确定了是哪个action后便创建相应的ActionInvocation。

***5.*** Interceptor-> Action->result->Interceptor

这部分内容统统都是由ActionInvocation代为调用，我猜。因为，这些东西都是在struts.xml中配置的，那么创建ActionInvocation时也一定都将信息传了进去。然后就是一一调用了。具体的配置就不写了，虽然记得也不是太清，上面都有

***6.*** Interceptor-> HttpServletResponse

最后一步，这里我应该好好看看HttpServletRequest是怎么转换为HttpServletResponse的。

**小结：**

这才是最为主要的功能，是Struts2的核心结构，其他的像文件上传，防止多次提交都是次要的。理解了核心才好办事，不过不先学那些功能，一上来就看核心体系也是扯淡，哪来那么大耐性。

2017.2.7

////////////////////////////////////////////////////////////////////////////////////////