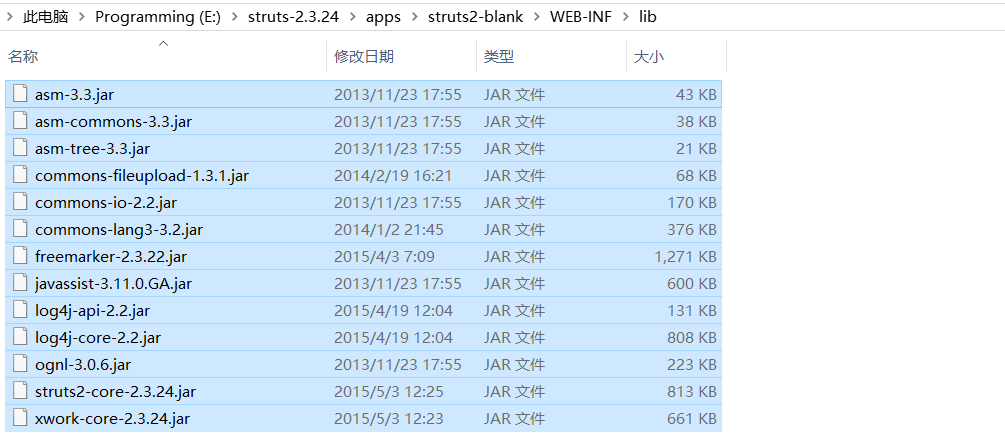
# 一、Struts2入门

## Struts2概述

1. Struts2是一个基于MVC设计模式的Web应用框架，它本质上相当于一个servlet，在MVC设计模式中，Struts2作为控制器(Controller)来建立模型与视图的数据交互；Struts2是在 struts 1和WebWork的技术基础上进行了合并的全新的Struts 2框架。
2. web层常见框架：Struts2和SpringMVC；

## Struts2框架入门

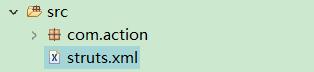
1. 导入jar包



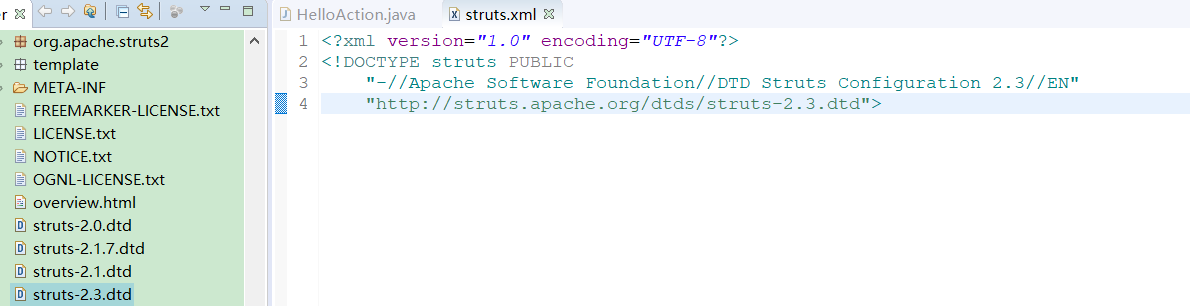
1. 创建Action



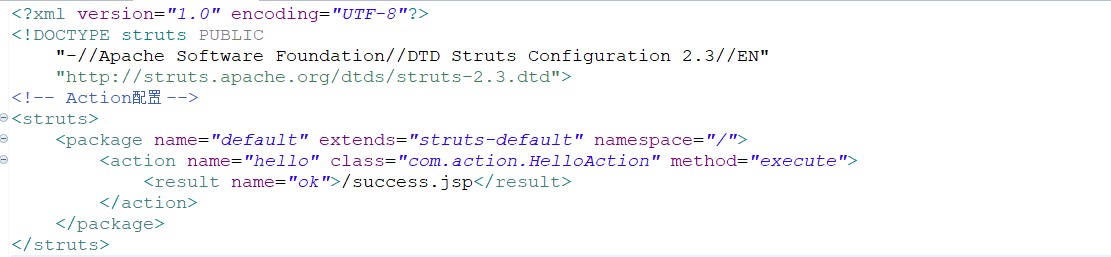
1. 配置Action的访问路径
2. 创建一个Struts2的核心配置文件：核心配置文件的名称和位置是固定的，位置必须在src下，名称struts.xml



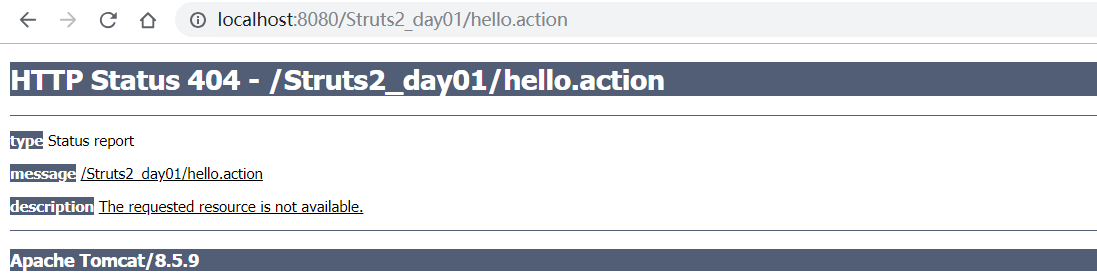
1. 引入dtd约束：



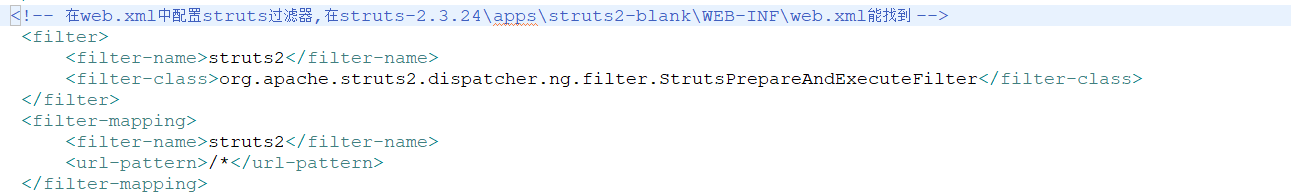
1. 配置struts.xml



1. 访问路径：<http://localhost:8080/项目名/hello.action>



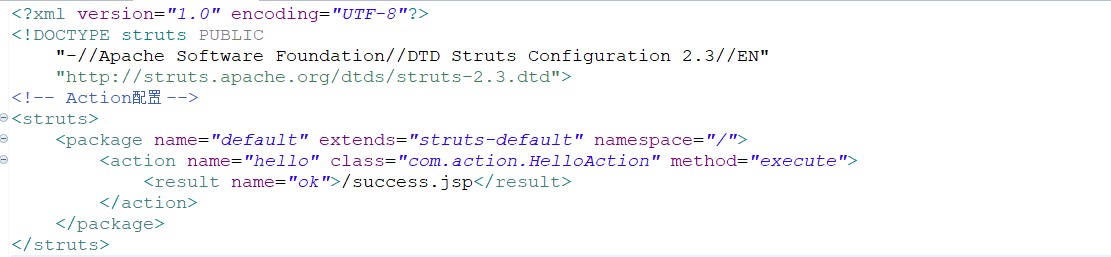
1. 配置struts2过滤器



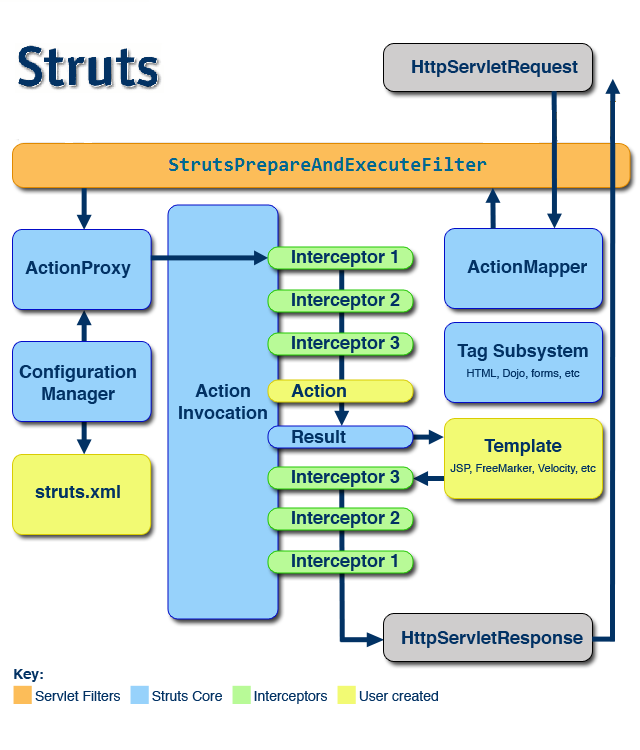
1. 过滤器在服务器启动的时候创建，创建过滤器的时候执行init方法；
2. 在init方法中主要加载配置文件（包含自己创建的配置文件和struts2自带的配置文件）；
3. 配置文件加载顺序(拦截器里57/74行)：default.properties(默认常量）,struts-default.xml(拦截器),struts-plugin.xml(struts依赖的插件),struts.xml(配置Action以及常量),struts.properties(配置常量),web.xml(核心过滤器及常量);

## Struts2执行过程

1. 原理：客户端向服务器发送一个请求，请求会先经过struts2核心过滤器；过滤器会解析strust.xml文件；根据请求的地址name找到对应的Action（class）；执行完毕根据struts.xml中的配置找到对应的结果result，并跳转到响应页面。



1. 底层原理：



（1）客户端初始化一个指向Servlet容器的请求（客户端提交一个HttpServletRequest请求）。

（2）经过 Struts 2 的核心过滤器StrutsPrepareAndExecuteFilter。

（3）StrutsPrepareAndExecuteFilter询问ActionMapper是否需要调用某个Action来处理这个（HttpServlet Request）请求，如果ActionMapper决定需要调用某个Action，FilterDispatche则把请求的处理交给ActionProxy。

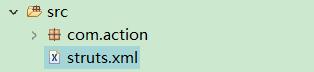
（4）ActionProxy通过Configuration Manager（struts.xml）询问框架的配置文件，找到需要调用的Action类。

（5）ActionProxy创建一个ActionInvocation实例，同时ActionInvocation通过代理模式调用Action。但在调用之前，ActionInvocation会根据配置加载Action相关的所有Interceptor（拦截器）。

（6）Action执行完毕后，ActionInvocation负责根据struts.xml中的配置找到对应的返回结果result。

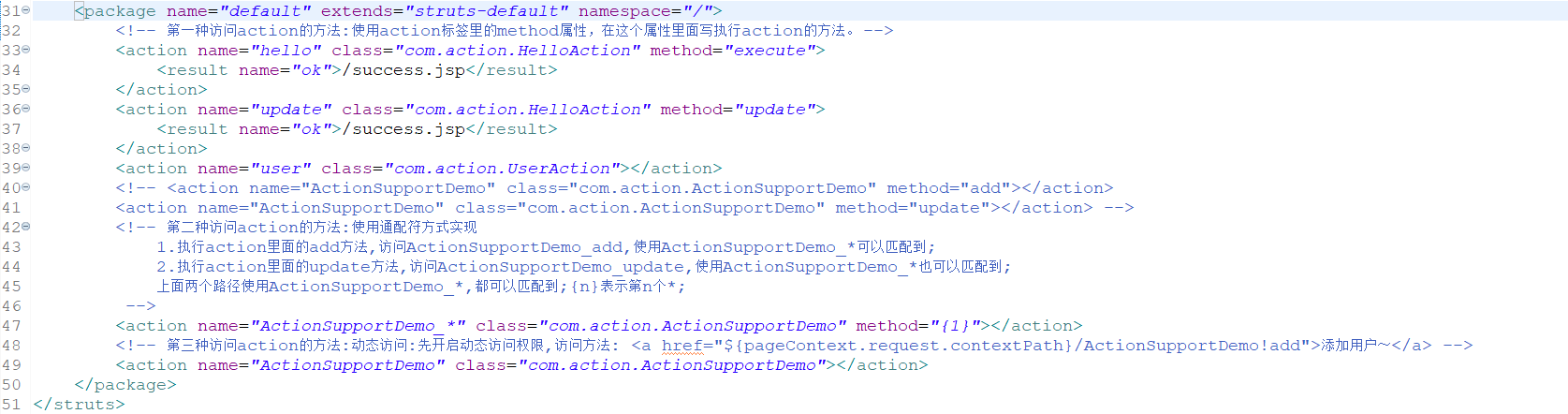
## Struts2配置

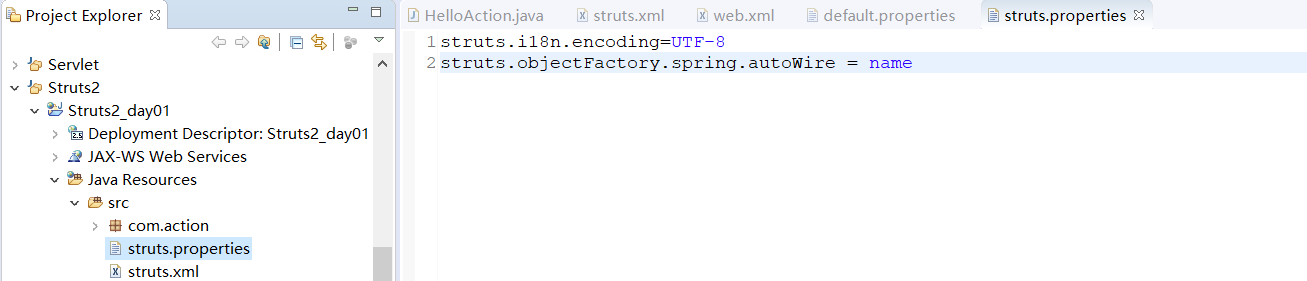
Struts2的核心配置文件：

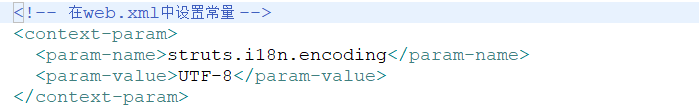


1. 名称和位置是固定的
2. 在配置文件中有三个标签：package,action，result









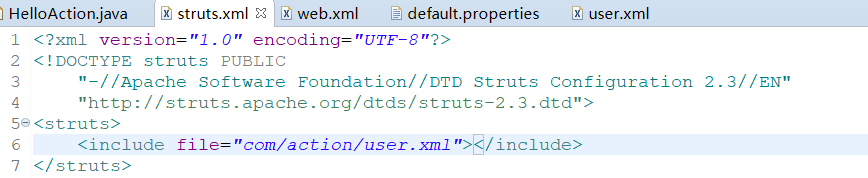
1. struts.i18n.encoding=UTF-8作用：

表单提交数据到action，在action中可以获得表单提交的数据；

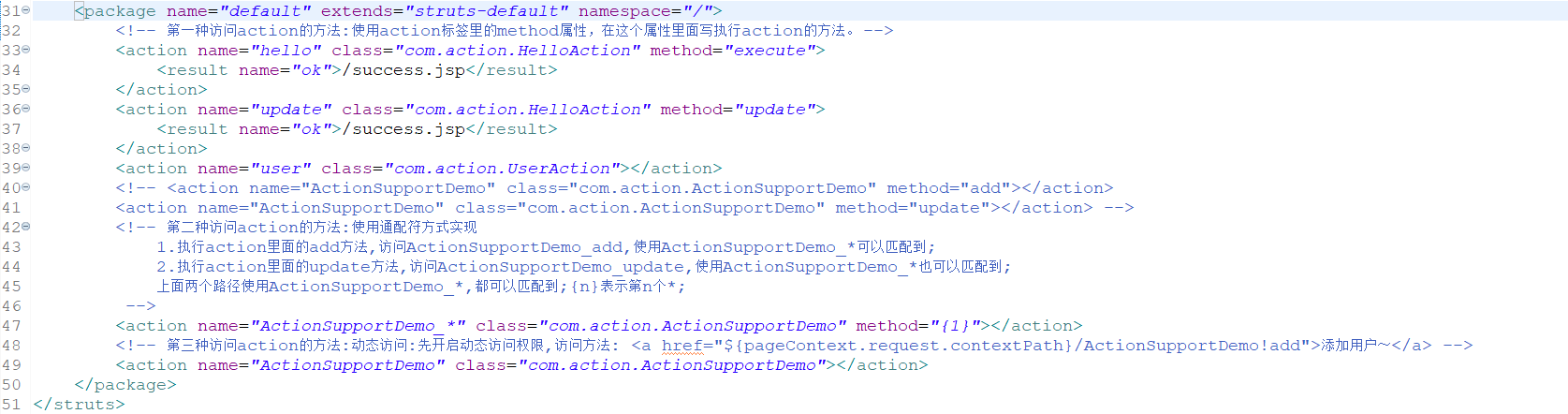
表单提交数据有中文，则会有乱码问题，解决：post提交直接设置编码，get提交做编码转换；

如果在action中获得表单，通过post方式提交中文，中文乱码帮解决了，不需要自己处理；

1. 分模块开发：单独写配置文件，然后把配置文件引入到核心配置文件中。





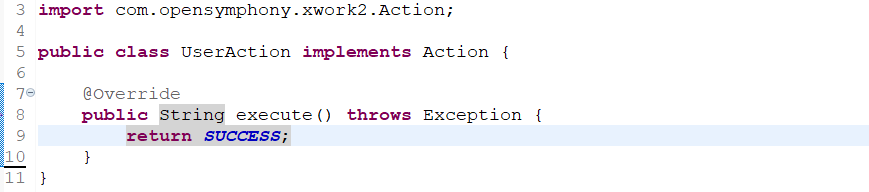


## Action编写方式

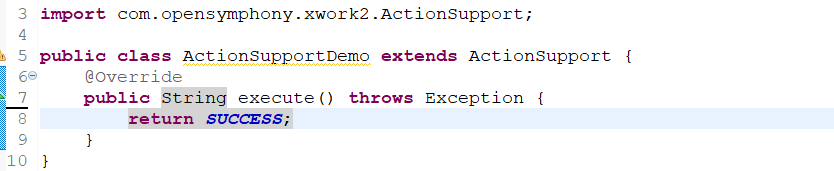
1. Action编写有三种方式：

第一种：创建普通类，不继承任何类，不实现任何接口，**public** **class** HelloAction

第二种：创建类，实现接口



第三种（常用）：创建类，继承ActionSupport



1. 返回值：
2. 在action里面的方法，如果有返回值的时候类型必须是String。
3. action里面的方法可以没有返回值，没有返回值的时候，在result标签不需要配置。把方法写成void或让返回值返回NONE或”none”；

## 访问action的方法（重点）

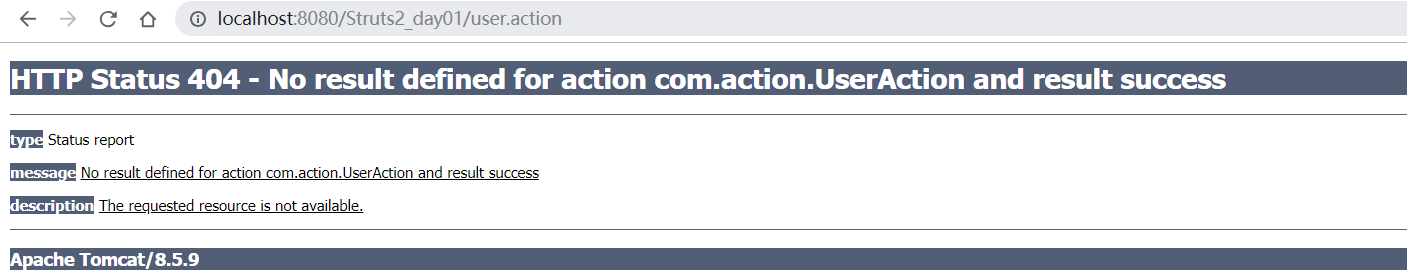
1. 有三种方式实现

第一种：使用action标签里的method属性，在这个属性里面写执行action的方法。

第二种：使用通配符方式实现。

第三种：动态访问（一般不用）。

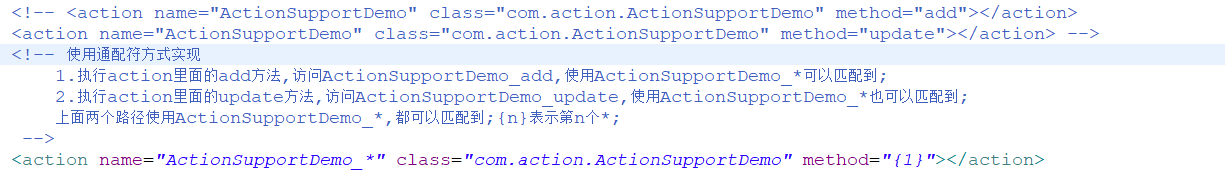
1. 错误演示：
2. 如果action方法有返回值，在配置文件中没有相应的result配置，则出现错误



1. 创建action，创建多个方法，使用method属性，缺陷action每个方法都需要配置，如果action里面有多个方法，需要配置很多的action。



1. 解决方法：使用通配符方式实现。
2. 使用通配符方式实现：
3. 在action标签的name属性值里面写符号\*。
4. \*号表示匹配任意内容。



1. 动态访问：



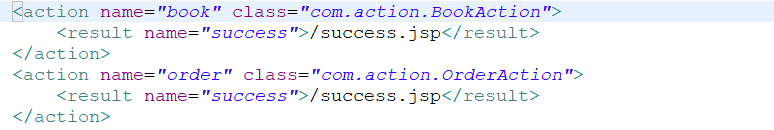


# 二、Struts2数据操作

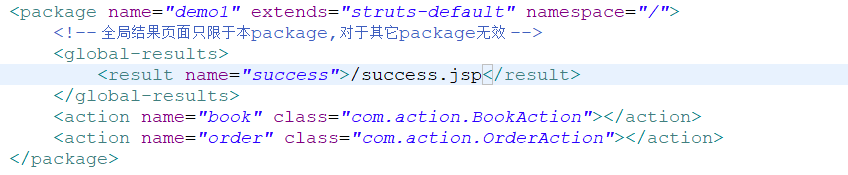
## 结果页面配置

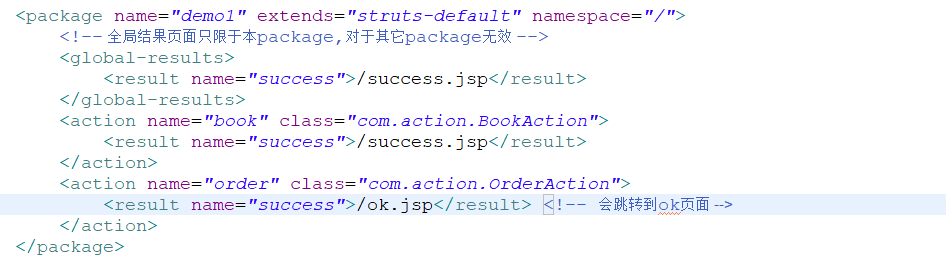
### 全局结果页面

1. result标签配置action方法的返回值到不同的路径里面。
2. 创建两个action，默认执行execute方法，让两个action的方法都返回success，返回success之后配置到同一个页面。
3. 如果多个action，方法里面返回值相同，并访问相同的页面，这个时候可以使用全局结果页面。



1. 全局结果和局部结构代码：配置全局结果页面，也配置了局部结构页面，最终以局部结构页面为准。比如图二中的order，访问的是ok，而不是success页面。

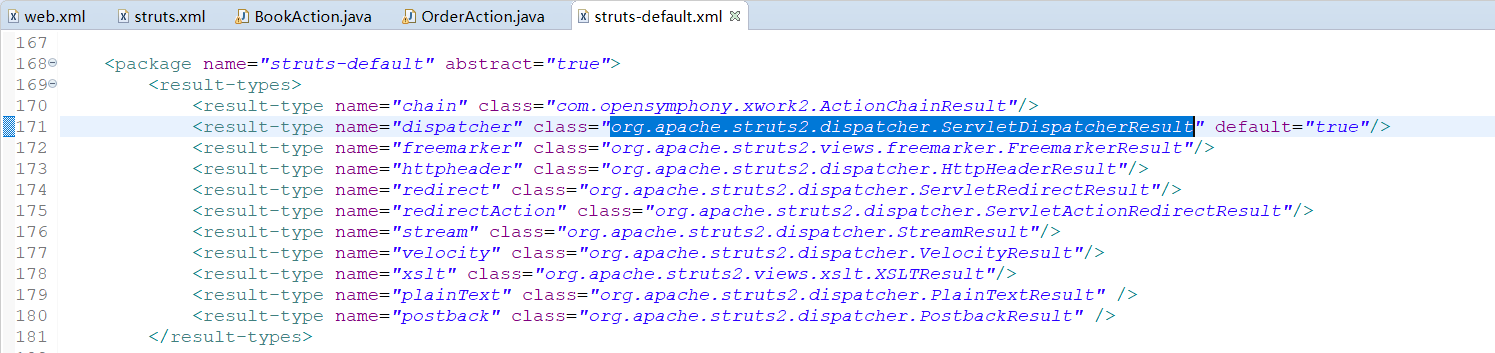




### result标签的type属性：

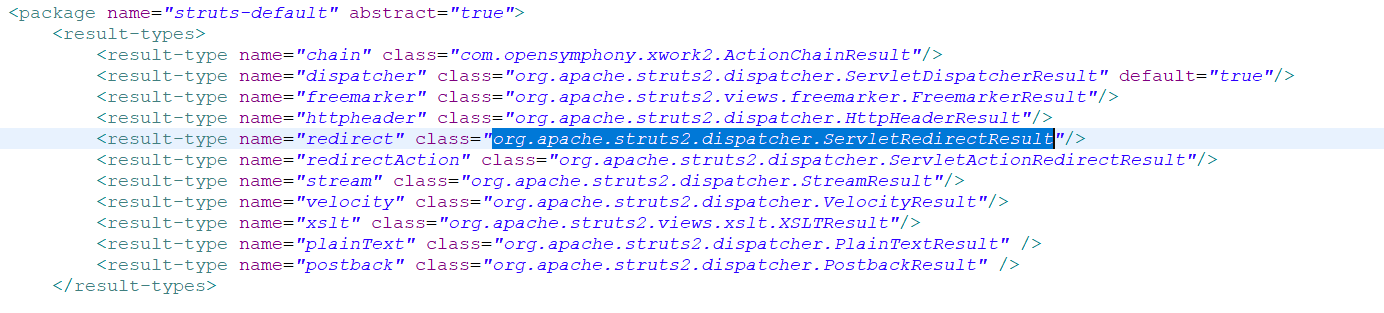
1. result标签处理name属性外，还有个type属性
   1. type属性：如何到路径里面（转发/重定向）
2. type属性值：
   1. 默认做转发操作，值是dispatcher。

结果处理器类：



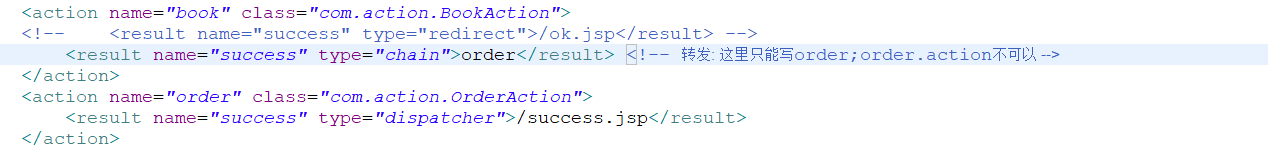
* 1. 做重定向操作，值是redirect。

结果处理器类：

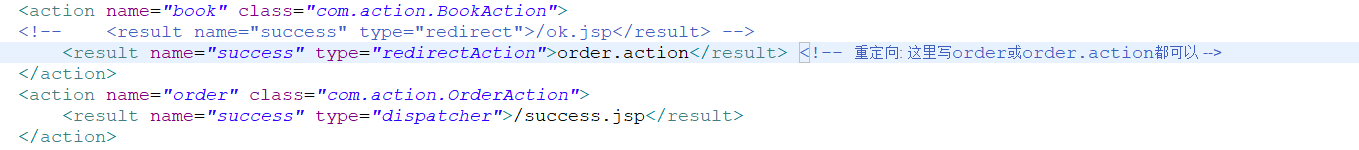


1. 上面两个type的值一般针对跳转到页面中的配置。以下是配置到其它action里面：

chain：转发到action，一般不用，因为缓存问题；



redirectAction：重定向到action；



## Action获取表单数据及操作数据

1. 之前web阶段，提交表单到servlet，在servlet里面使用request.getParameter或getParameterValues方法获取；
2. 提交表单action但是action没有request对象，不能直接使用request对象。以下是action获取表单提交数据的三种主要方式：

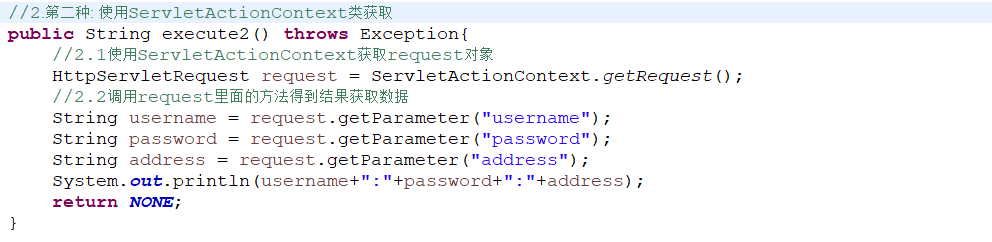
### 使用ActionContext类获取

* 1. 创建一个表单，提交表单数据到action里面
  2. 在action里面使用ActionContext获取数据（解耦合：因为代码表面上和request没关系，虽然底层还是request实现的）

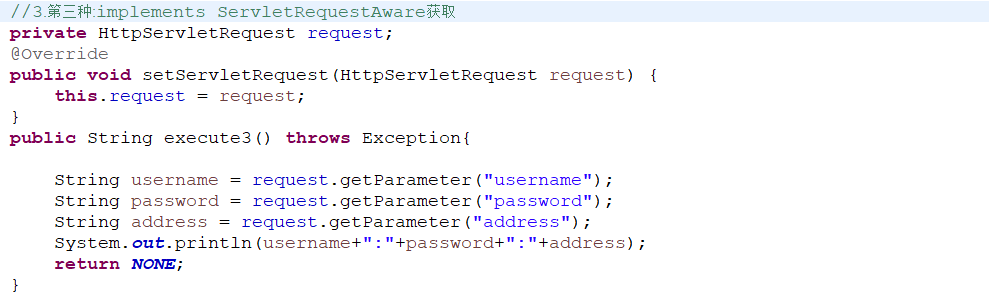


### 使用ServletActionContext类获取

* 1. 调用ServletActionContext类里面的静态方法，得到request对象



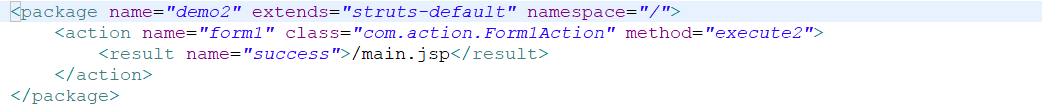
### 使用接口注入（了解）

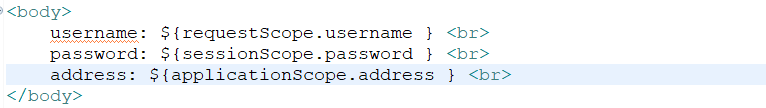


### 在action中操作域对象

request/session/servletContext，使用ServletActionContext类操作。

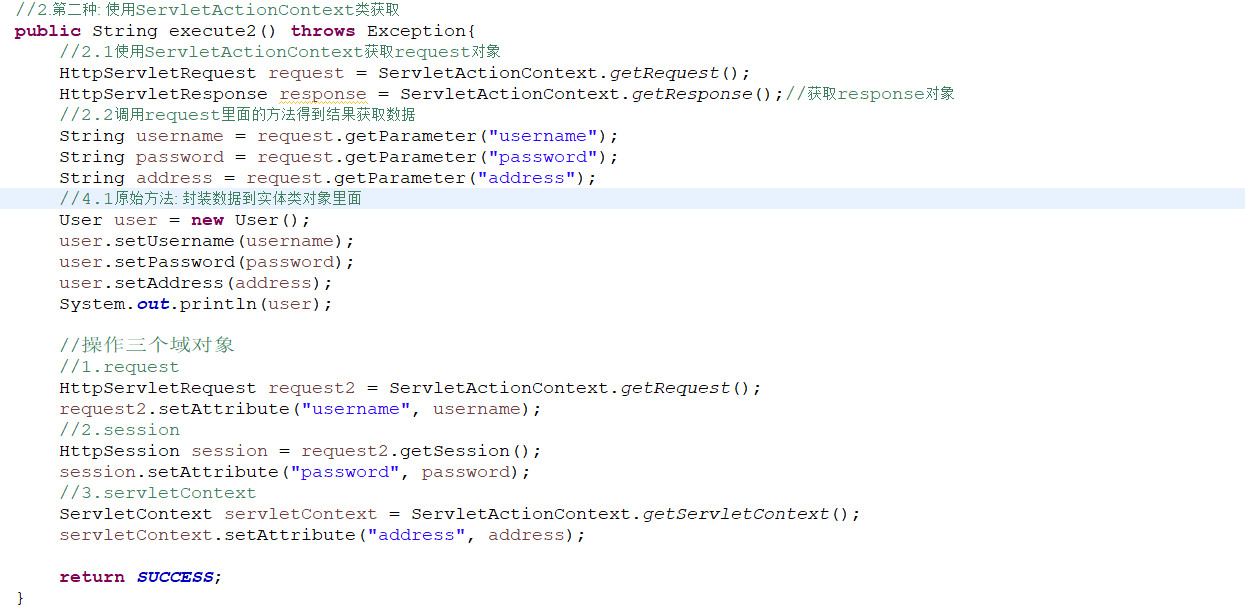






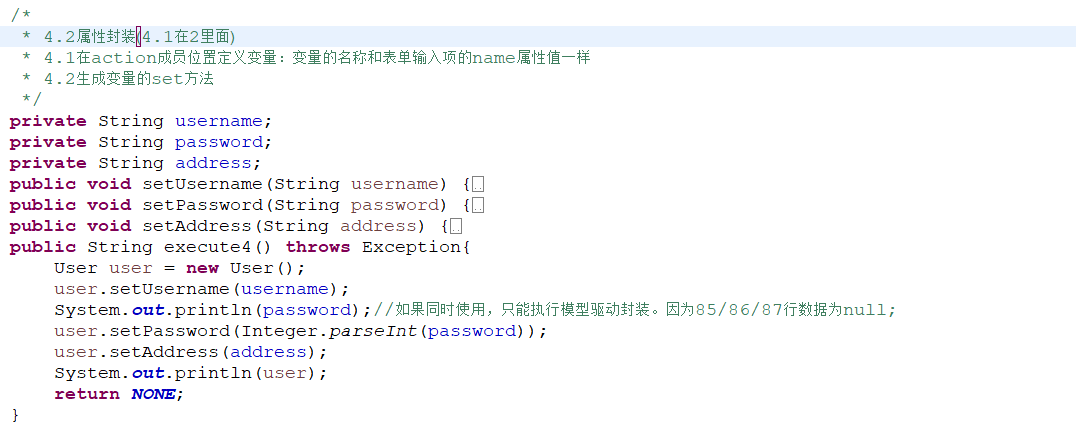
## Struts2封装表单数据到实体类(也叫属性类)的方式

### 原始方式获取表单数据封装到实体类

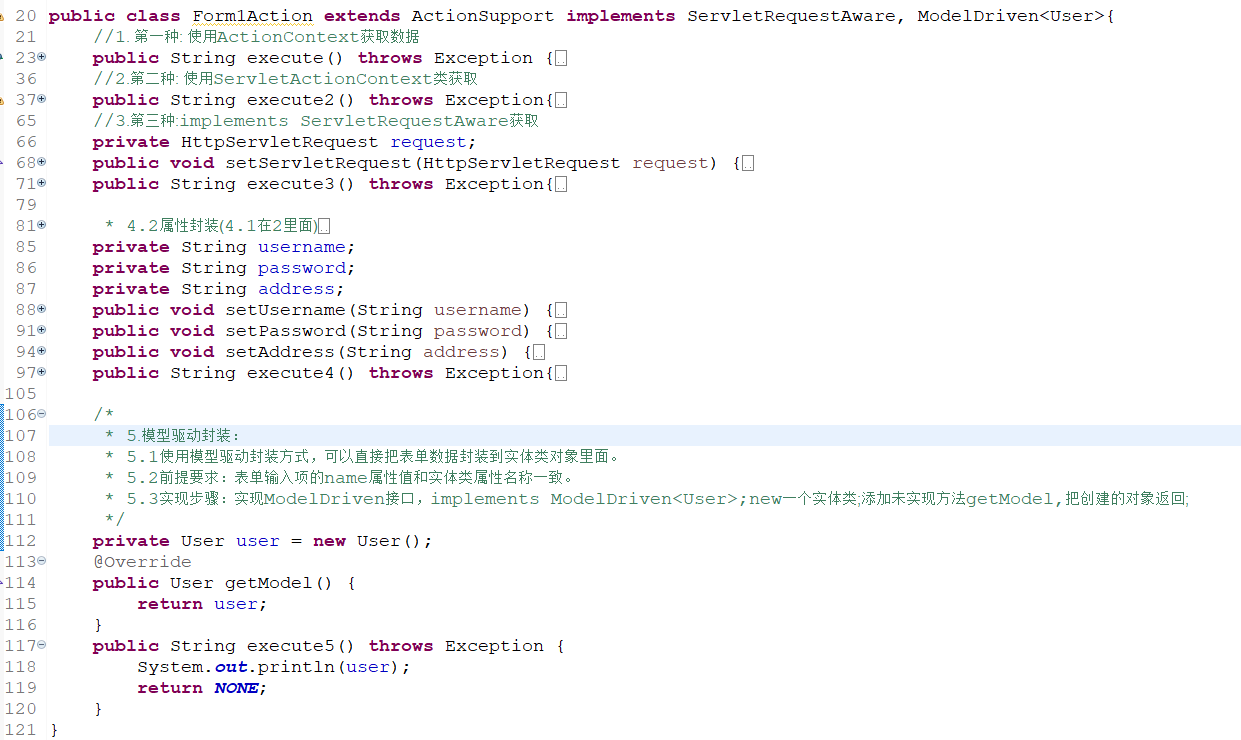


### 属性封装

直接把表单提交属性封装到action的属性里面，缺点：不能把数据直接封装到实体类里，只能手动封装。



### 模型驱动封装（重点）



### 使用模型驱动和属性封装需要注意的问题

在一个action中，获得表单数据可以使用属性封装，可以使用模型驱动封装，但是不能同时使用属性封装和模型驱动封装获取同一个表单数据，如果同时使用，只能执行模型驱动封装。

### 模型驱动与属性驱动是什么？模型驱动使用时注意什么问题：

1. 模型驱动与属性驱动 都是用来封装数据的；
2. 模型驱动：在实体类中实现了ModelDriven<T>接口，使用泛型把属性类封装起来，重写getModel方法，然后在实体类里创建一个属性(bean)类的实例；
3. 属性封装：在实现类里定义属性，生成get与set方法，通过属性来取值；
4. 注意：模型驱动使用时，先把属性(bean)类实例化，否则会出现空指针异常；

## 比较表达式封装和模型驱动封装

### 表达式封装

使用表达式封装可以把表达数据封装到实体类对象里面；实现过程：





### 比较

使用表达式封装和模型驱动封装都可以把表单数据封装到实体类里面里面；使用模型驱动封装只能把数据封装到一个实体类对象里面，在一个action里面不能使用模型驱动封装到不同的实体类对象里面。使用表达式封装可以把数据封装到不同的实体类对象里面。

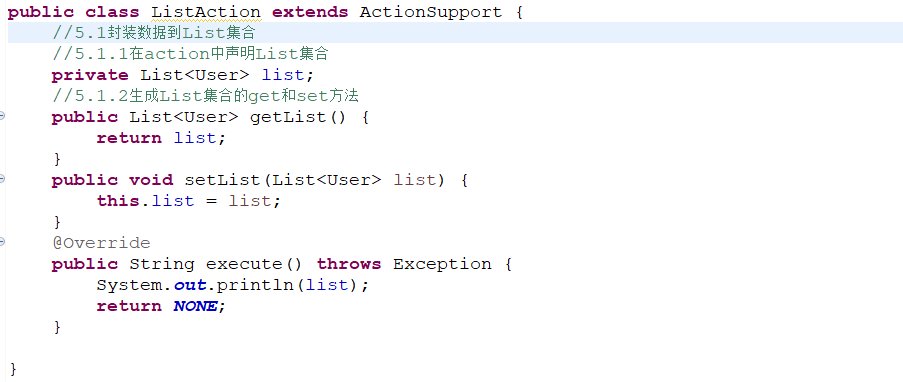
## 封装到List/Map集合里面

### 封装数据到List集合

第一步：在action中声明List集合

第二步：生成List集合的get和set方法

第三步：在form里重写name的表达式



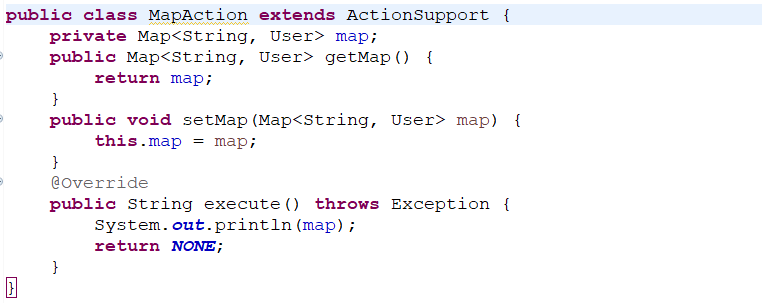


### 封装数据到Map集合

第一步：在action中声明Map集合

第二步：生成Map集合的get和set方法

第三步：在form里重写name的表达式





# 三、Struts2值栈

## OGNL概述（对象图导航语言）

OGNL是Object-Graph Navigation Language的缩写，它是一种功能强大的表达式语言，通过它简单一致的表达式语法，可以存取对象的任意属性，调用对象的方法，遍历整个对象的结构图，实现字段类型转化等功能。它使用相同的表达式去存取对象的属性。

1. 之前web阶段，学过EL表达式，EL表达式只能在jsp页面中获取域对象里面的值
2. OGNL是一种表达式，这个表达式功能更加强大；在struts2里面和struts2标签一起操作值栈中的数据；OGNL本身不是struts2的一部分，使用时需要先导包，只不过struts2将其引入了进来（已提供，所以不需要手动导包了）
3. 然后在jsp页面中引入标签库。

## OGNL的要素

表达式：规定了做什么；

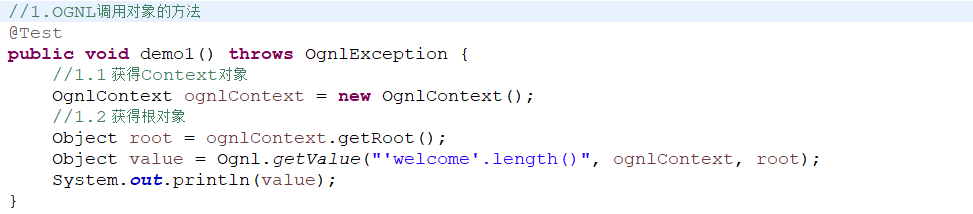
根对象（Root）：Root对象规定了对谁操作；

Context对象：上下文环境，规定了OGNL的操作在哪里执行；

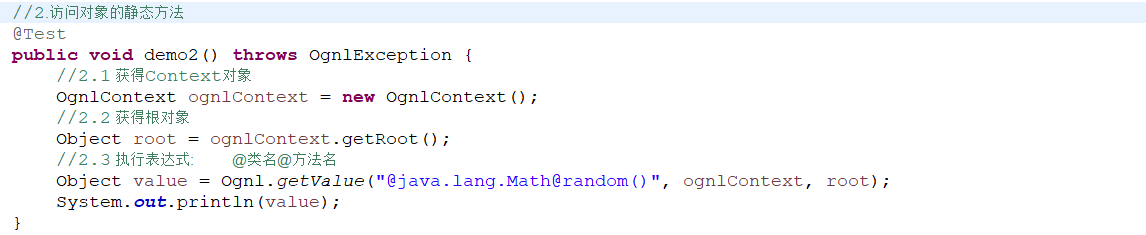
## OGNL入门案例

### OGNL的java环境入门：

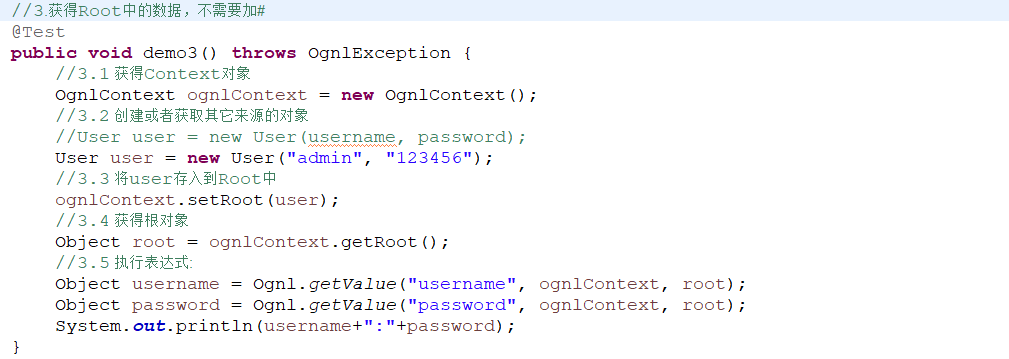
1. OGNL调用对象的方法



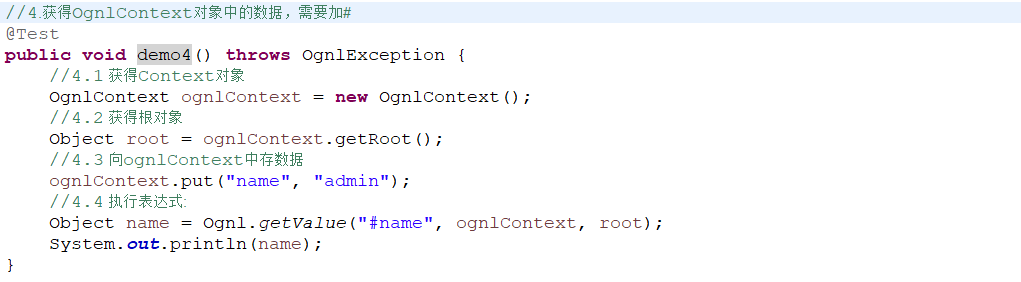
1. 访问对象的静态方法



1. 获得Root中的数据，不需要加#



1. 获得OgnlContext对象中的数据，需要加#



### OGNL在struts2环境入门

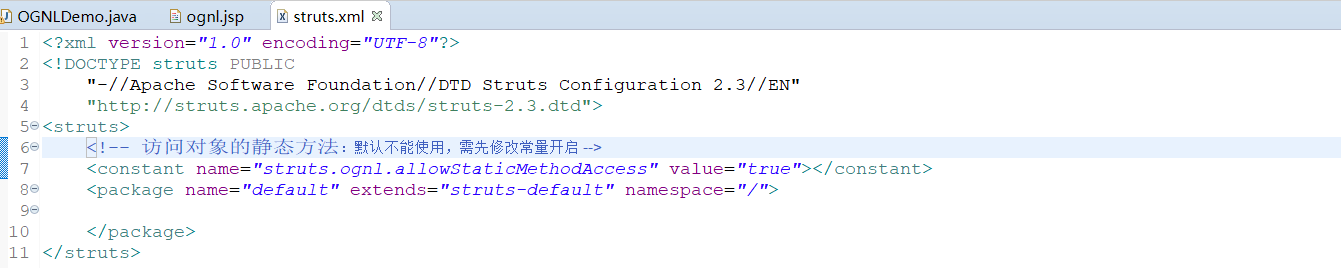
1. 引入标签库：



1. 访问对象的方法：



1. 访问对象的静态方法：默认不能使用，需先修改常量开启。





## 什么是值栈

1. 之前WEB阶段，在Servlet里面进行操作，把数据放到域对象里面，可以在页面中使用el表达式获取到，域对象在一定范围内存值和取值。
2. 在action里面，提供本身一种存储机制，类似于域对象，是值栈，可以存值和取值。
   1. 在action里面把数据存到值栈里面，在页面中获取到值栈数据

### Servlet和action的区别

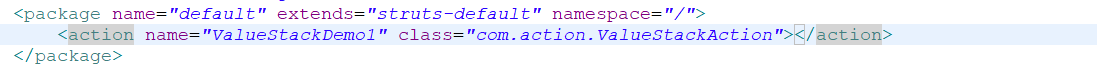
* 1. Servlet：默认第一次访问的时候创建，创建一次，单实例对象。
  2. Action：访问的时候创建，并且每次访问Action的时候，都会创建Action对象（可以通过无参构造方法进行验证），创建多次，多实例对象。

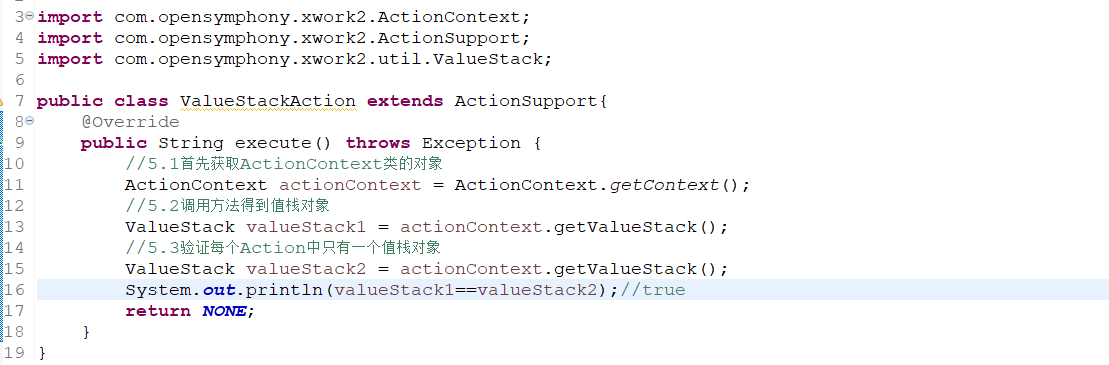
### 值栈存储位置

* 1. 每次访问Action的时候都会创建值栈对象。
  2. 在每个Action对象里面，都会有一个值栈对象。

## 获取值栈对象

1. 使用ActionContext类里面的方法得到值栈。
2. 每个Action对象中只有一个值栈对象。



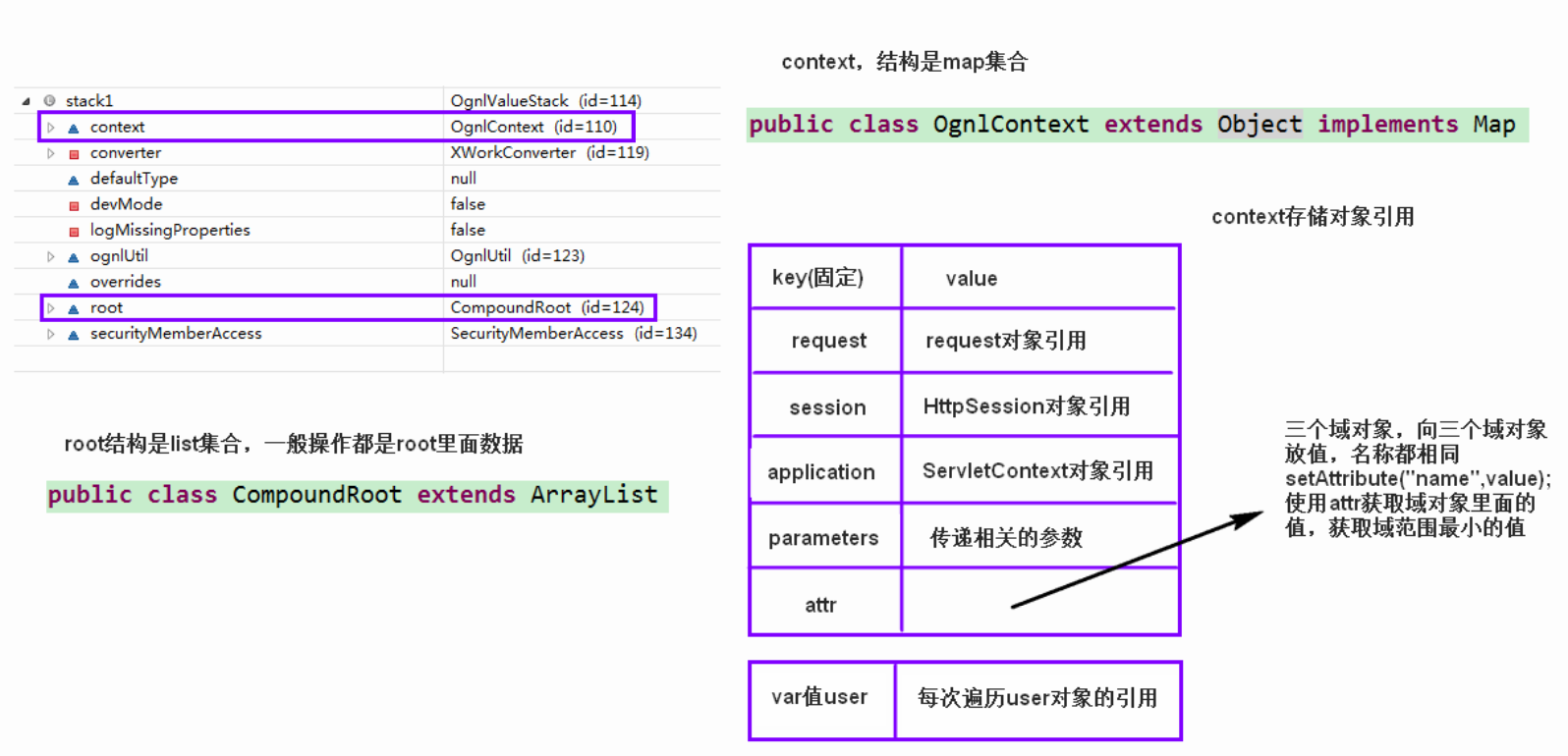


## 值栈内部结构

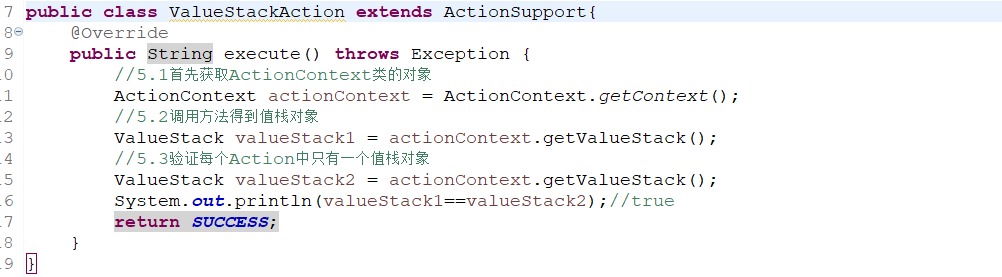
1. 值栈分为两部分

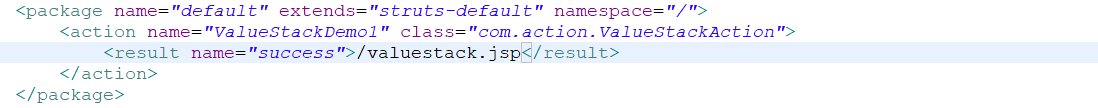
第一部分：Root部分，是一个List集合；一般操作的都是Root里面的数据

第二部分：Context部分，是一个Map集合；因为这个map集合，才可以解耦合使用request等域对象。



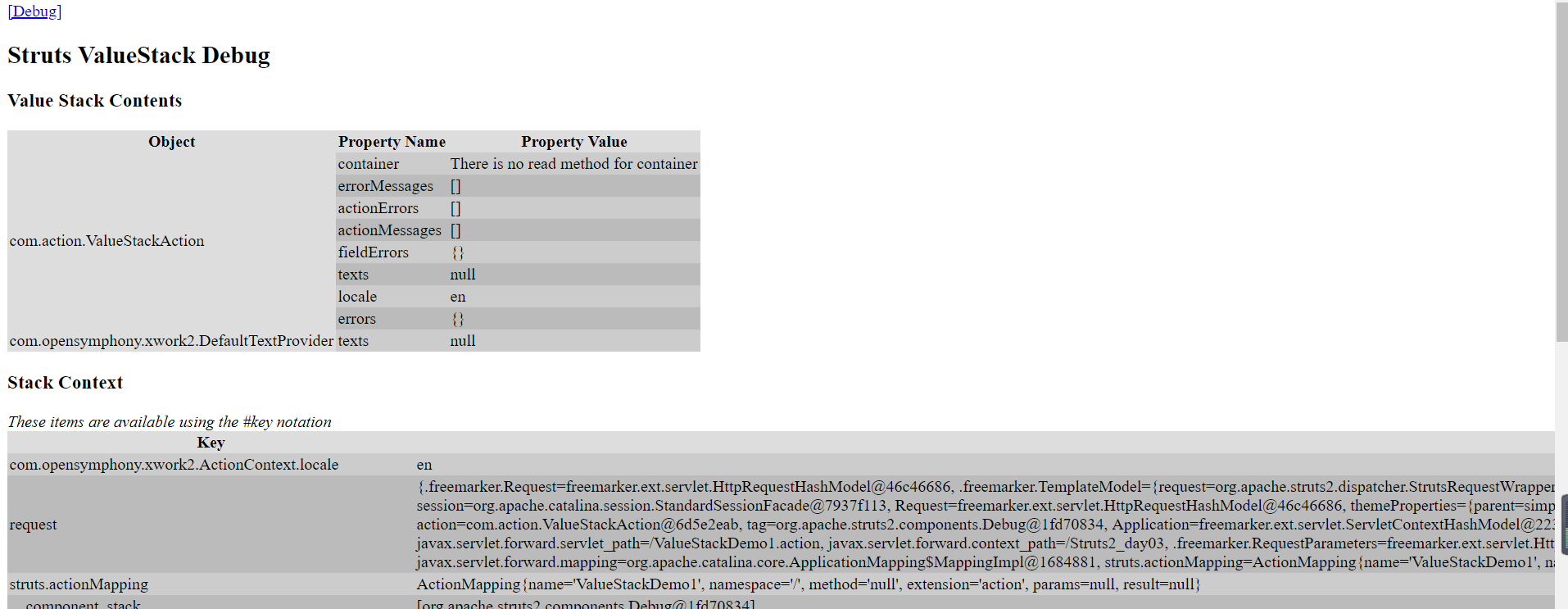
1. struts2里面的标签<s：debug>，使用这个标签可以查看值栈的结构和存储的值。
   1. 访问Action，执行Action的方法，有返回值，配置返回值到jsp页面，在jsp页面中使用这个标签。











* 1. 在action中没有做任何操作，栈顶元素是action的引用



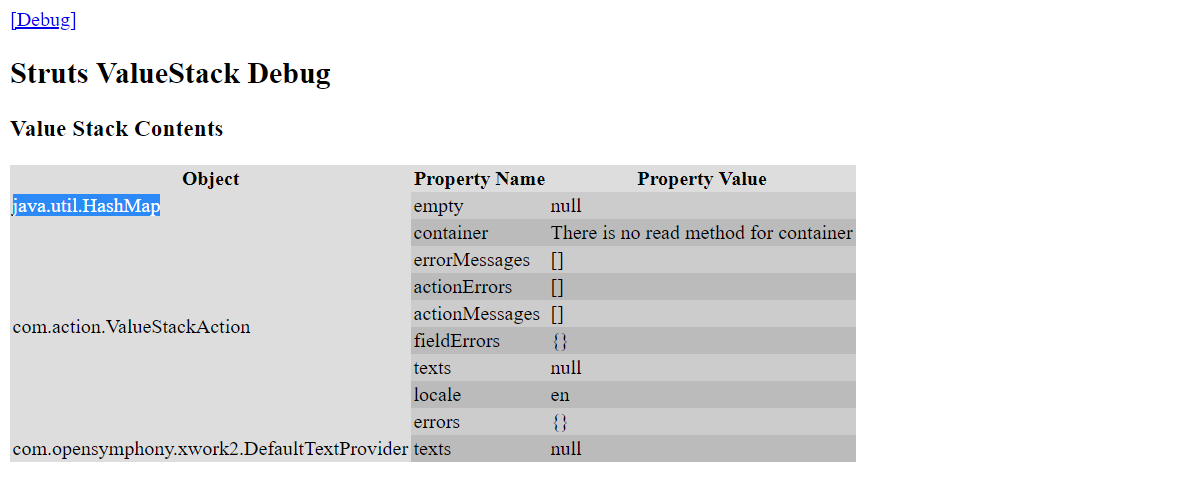
Action对象里面有值栈对象，值栈对象里面有action的引用（为了取值等操作更方便）

## 向值栈放数据(set/push/ get)

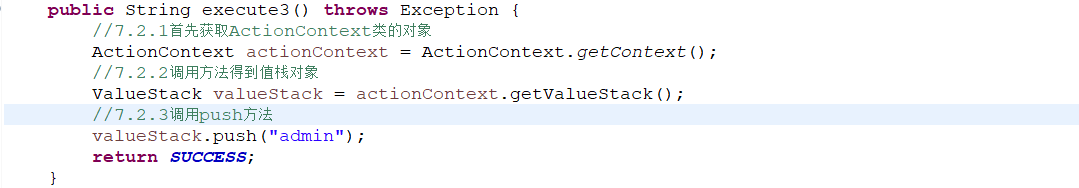
### 向值栈放字符串

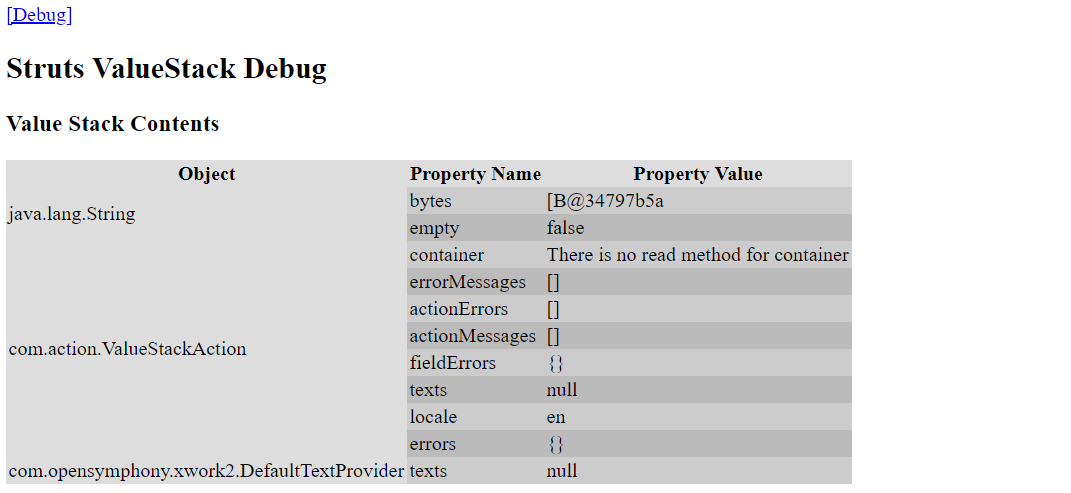
1. 获取值栈对象，调用值栈对象里面的set方法



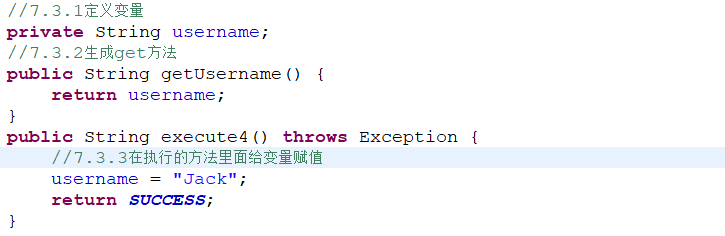


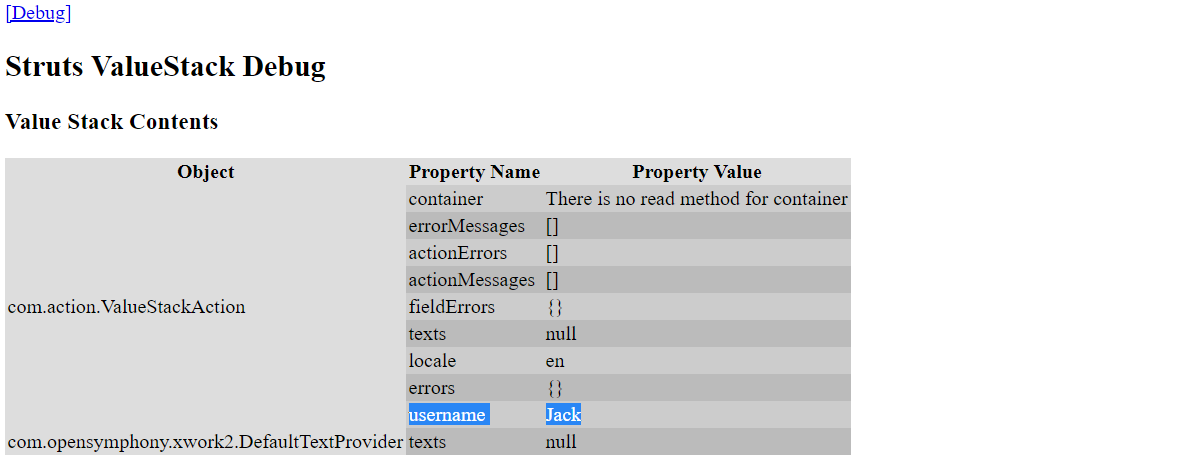
1. 获取值栈对象，调用值栈对象里面的push方法





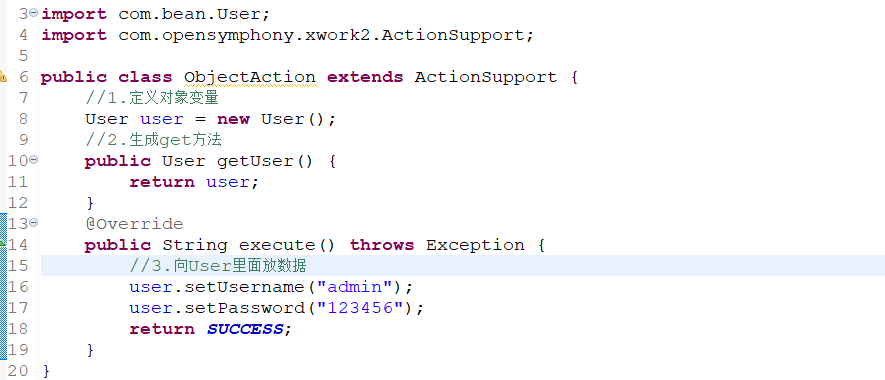
1. 在action中定义变量，生成get方法并给其赋值（可在任意地方赋值）；get方法的作用是Struts2底层得到此属性或对象，并将其放入值栈中。

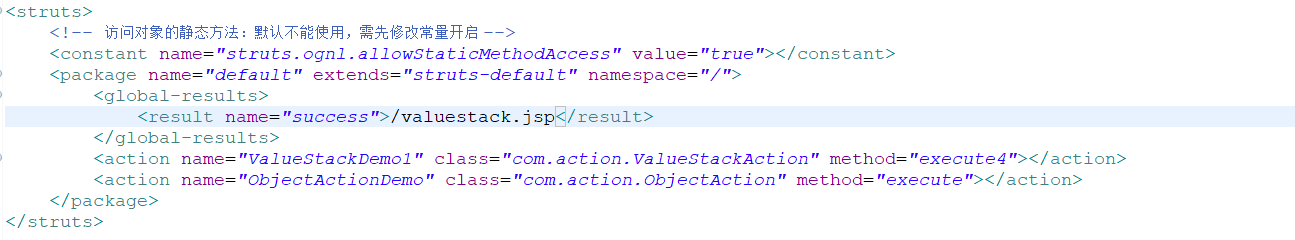


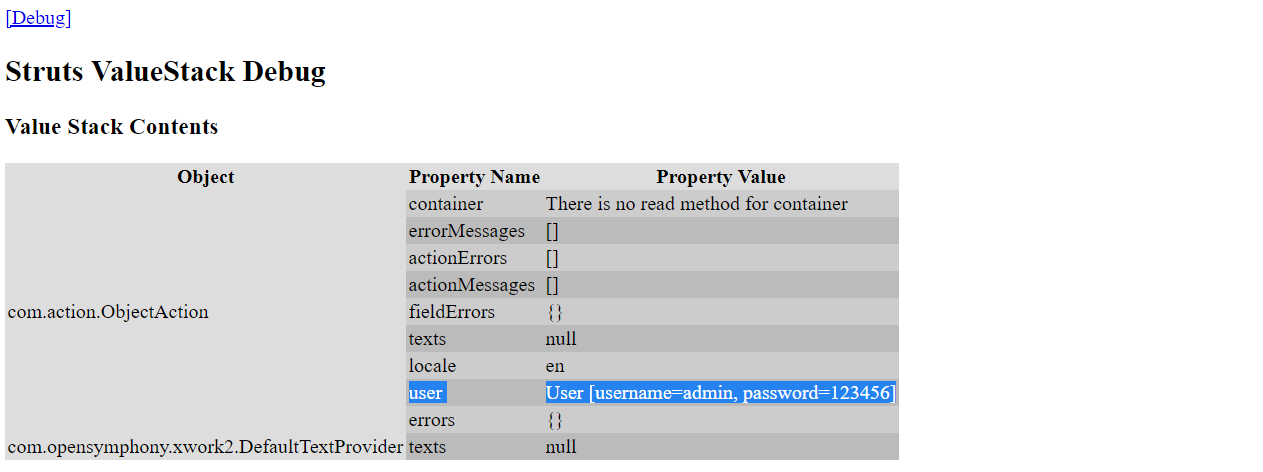


### 向值栈放对象

* 1. 定义对象变量
  2. 生成变量的get方法
  3. 在执行的方法里面向对象中设置值

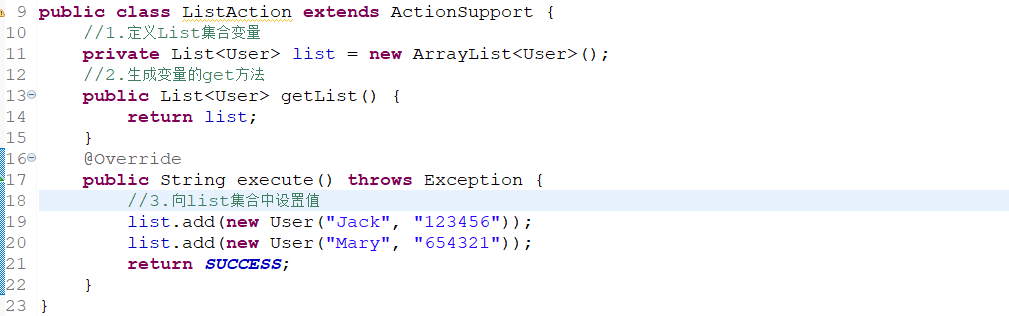


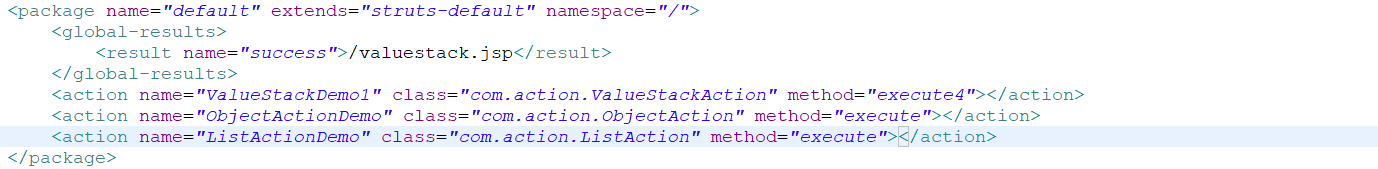


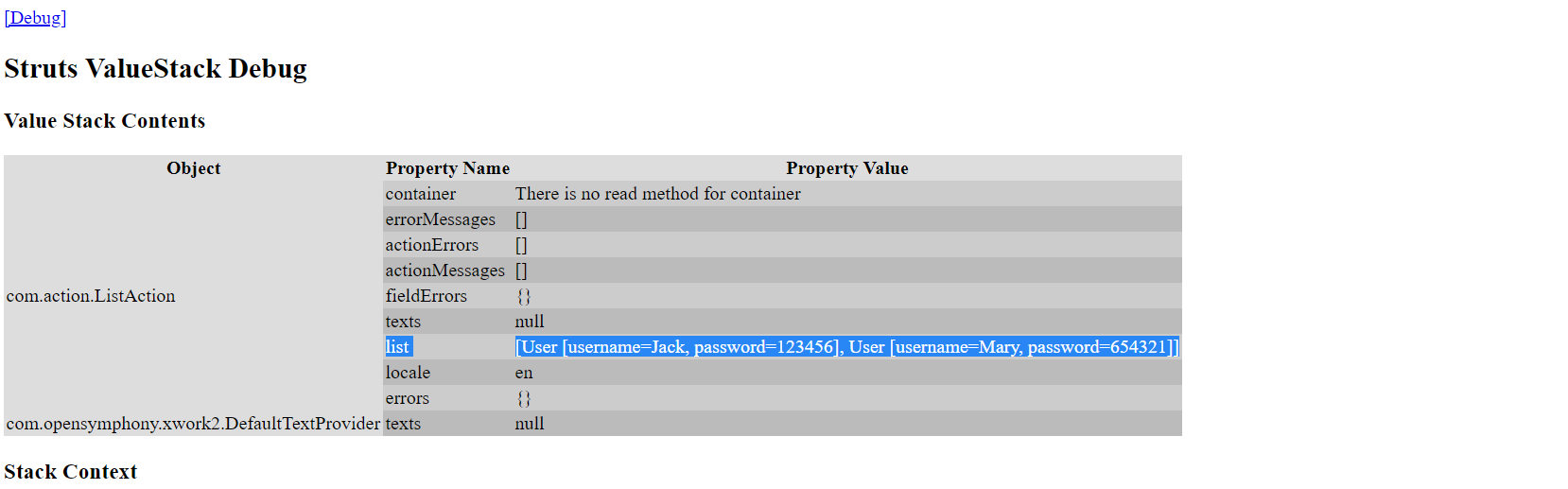


### 向值栈放List集合

1. 定义List集合变量
2. 生成变量的get方法
3. 在执行的方法里面向List集合设置值







## 从值栈获取数据(获取get保存的方式)

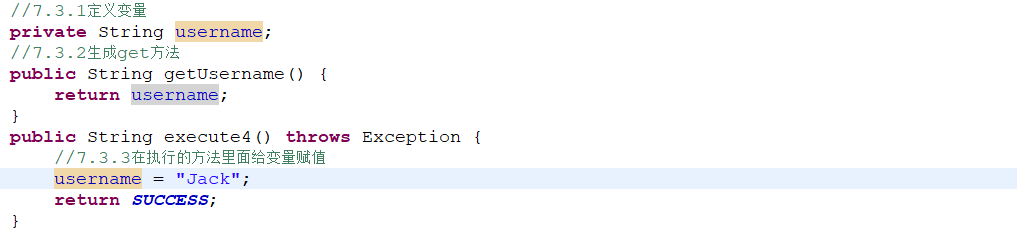
### 使用Struts2的标签加OGNL表达式获取值栈数据

<s:property value=”ognl表达式” />

### 获取字符串

向值栈放字符串

在jsp中使用struts标签+ognl表达式

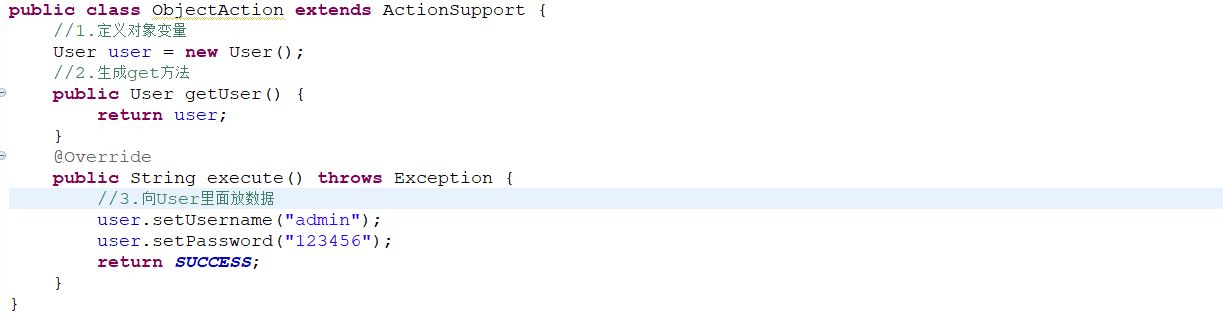


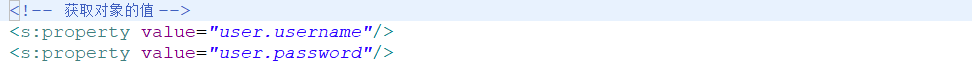


### 获取对象

向值栈放对象

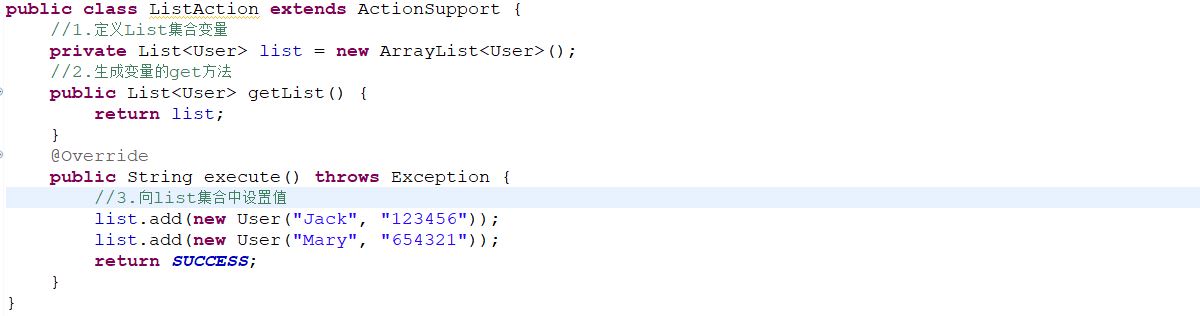
在jsp中使用struts标签+ognl表达式

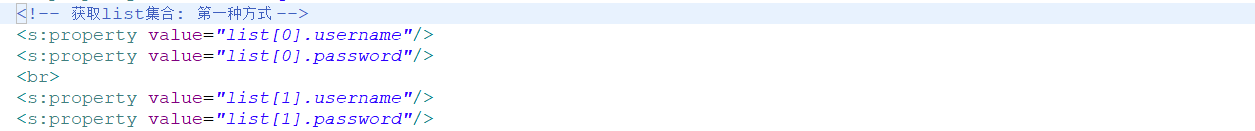




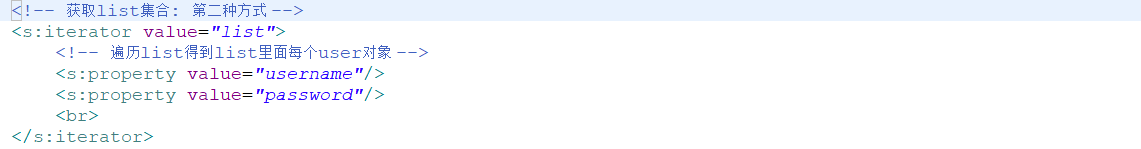
### 获取List集合（三种）

1. 获取list集合: 第一种方式

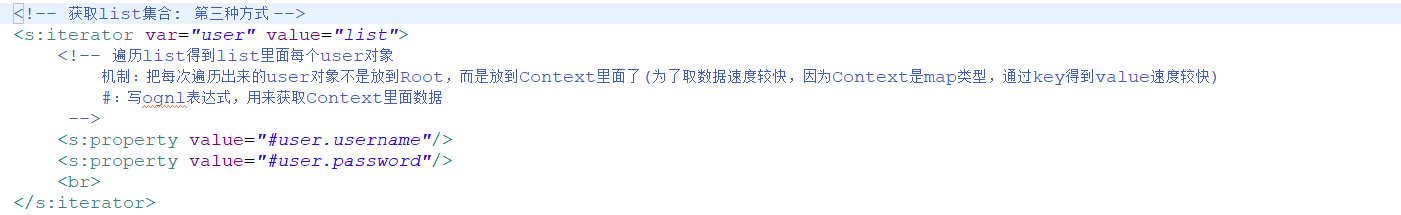




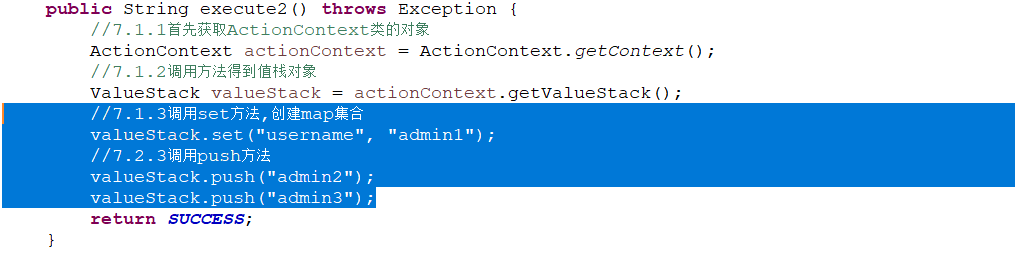
1. 获取list集合: 第二种方式



1. 获取list集合: 第三种方式



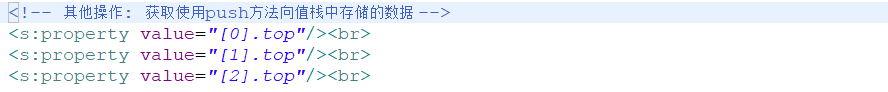
### 其他操作(获取set/push保存数据的方式)

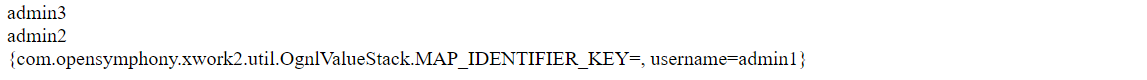


1. 获取使用set方法向值栈中存储的数据

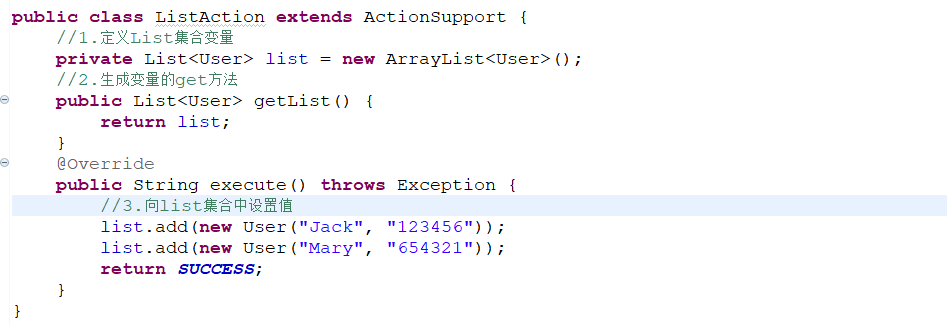


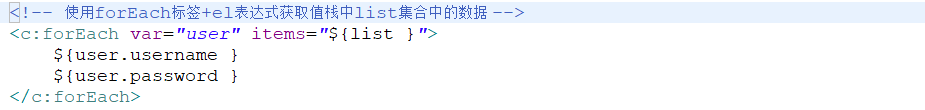
1. 获取使用push方法向值栈中存储的数据(后进先出)
   1. 使用push 方法设置值，没有名称，只有设置的值。
   2. 向值栈中放数据，把值栈中放的数据，存到数组里面，数组名称为top，根据数组获取值。





## EL表达式获取值栈数据





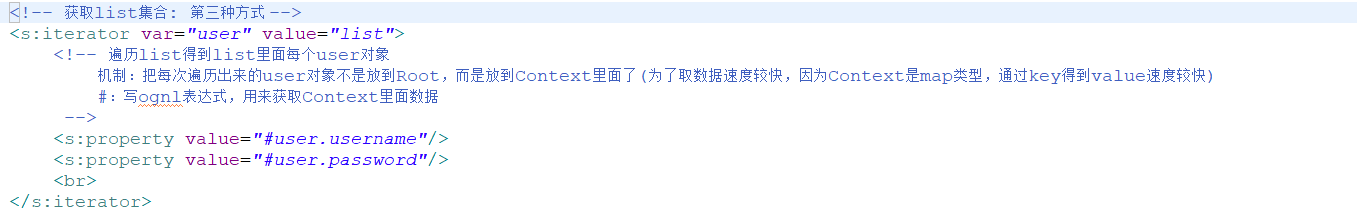
EL表达式获取值栈数据的原理步骤：

1. EL表达式获取域对象的值。
2. 向域对象里面放数据使用setAttribute方法，取值使用getAttribute方法。
3. 这里底层增强了request对象里面的getAttribute方法
   1. 首先，从request域中获取到值，如果获取到，直接返回
   2. EL表达式如果从request域中获取不到值，到值栈中把值获取出来，然后放到域对象中。但是效率低，建议使用ognl表达式。

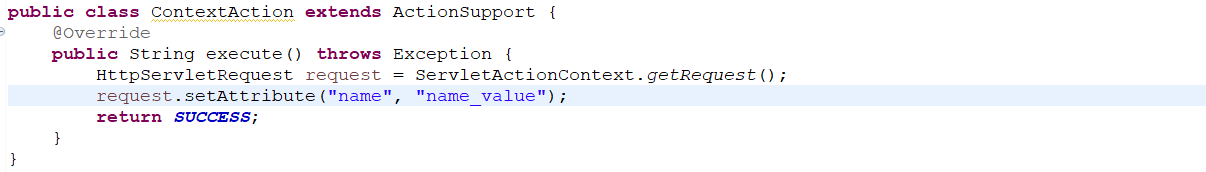
## OGNL的#和%的使用

### #的使用

* + 1. 使用#获取Context里面的数据



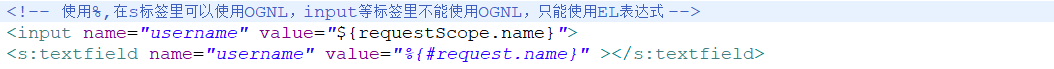
* + 1. 向request放值，然后在页面使用OGNL获取，因为在Context里有request等域对象的引用。





### %的使用

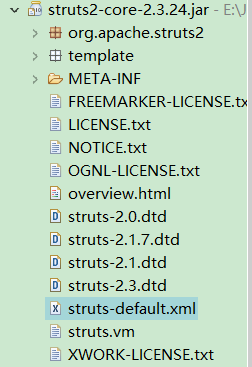
在struts2的表单标签中使用，要加%{………}，否则不识别OGNL表达式



# 四、Struts2拦截器

## 1. 拦截器概述

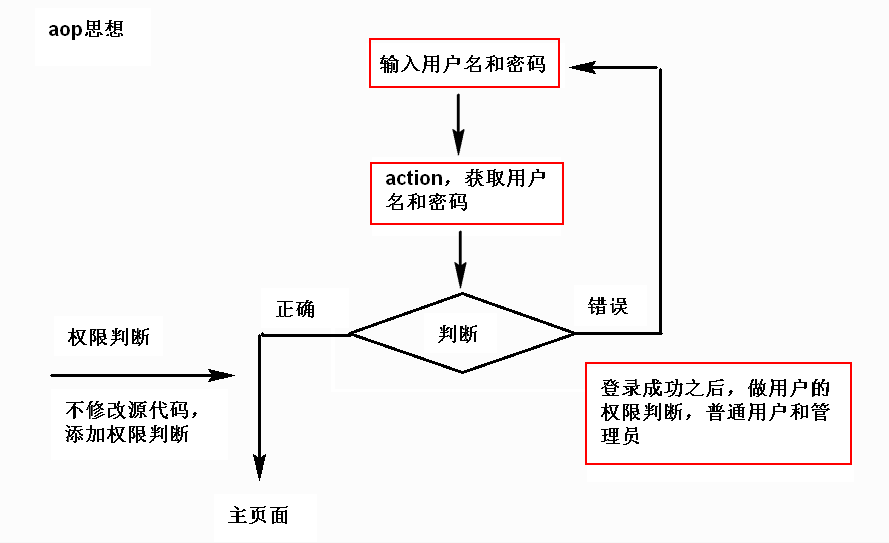
1. 在struts2框架中封装了很多功能，struts2里面封装的功能都是在拦截器里面帮助实现的。
2. Struts2里面有很多个拦截器，但不是每次这些拦截器都执行，每次执行默认的拦截器。
3. Struts2里面默认拦截器的位置。

1. 拦截器都是在action对象被创建后，方法执行之前执行的。

## 拦截器底层原理(两个)

1. AOP思想：面向切面编程，有基本功能，扩展功能，不修改源代码。



1. 责任链模式：责任链模式是java众多设计模式的一种，它和过滤链很相似。
   * 1. 责任链：要执行多个操作(添加/修改/删除)：首先执行添加，执行添加之后，做类似于放行的操作；执行修改，修改操作执行之后，做类似于放行的操作；执行删除。
     2. 过滤器：一个请求可以有多个过滤器进行过滤，每个过滤器只有做放行才能执行下一个过滤器。
2. AOP思想和责任链模式如何应用到拦截器里面：拦截器是在action对象创建之后，方法执行之前执行的；在action方法执行之前，执行默认拦截器，执行过程使用aop思想；在action中没有直接调用拦截器的方法，而是使用配置文件方式进行操作；在执行拦截器时执行多个拦截器，这个过程使用责任链。
3. 查看源代码，底层执行过程
   * + 1. 执行action



* + - 1. 创建action代理对象（代理对象不是真正的对象，代理对象具有真正对象所有的功能；真正的对象是new出来的）



* + - 1. 执行action里的方法



* + - 1. 遍历执行多个拦截器



* + - 1. 类似于放行的操作



## 过滤器和拦截器的区别

1. 拦截器只能对action的请求起作用，而过滤器则可以对几乎所有的请求起作用。因此登陆功能最好使用过滤器。
2. 拦截器不依赖于Servlet容器，过滤器依赖于Servlet容器。
3. 拦截器可以访问action上下文和值栈里的对象而过滤器不能访问。
4. 在action的生命周期中，拦截器可以多次调用，而过滤器只能在容器初始化时被调用一次。

## 自定义登陆拦截器

1. 在Struts2里面有很多的拦截器，这些拦截器是Struts2封装的功能，但是实际开发中，Struts2里面的拦截器中可能没有要使用的功能，这时需要自己写拦截器实现功能。
2. 拦截器结构：
3. 继承类



1. 实现接口



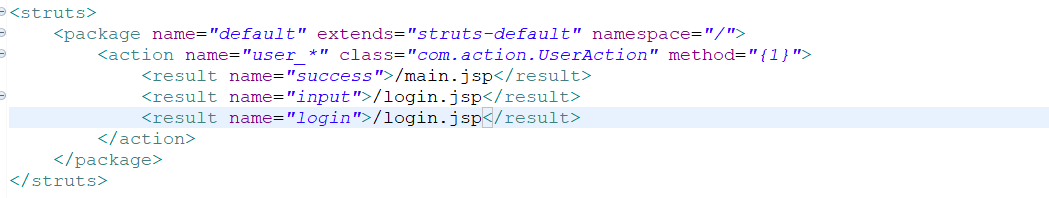
在接口里面有三个方法：

初始化init()，销毁destroy()，逻辑操作intercept(ActionInvocation invocation)

1. 在实际开发中建议使用第三种方式，创建一个继承MethodFilterInterceptor类的实现类，因为它可以较为简单的让action里的某个方法不进行拦截（前两种比较麻烦）。
2. 让action和拦截器产生关联，不是通过action调用拦截器的方法，而是通过配置文件让其建立联系的。
3. 添加登陆拦截器功能
4. 判断session里面是否有名称是username的值
5. 拦截器实现过程：
   1. 创建类，继承MethodFilterInterceptor类。
   2. 重写MethodFilterInterceptor类里面的方法，写拦截器逻辑。
6. 代码：





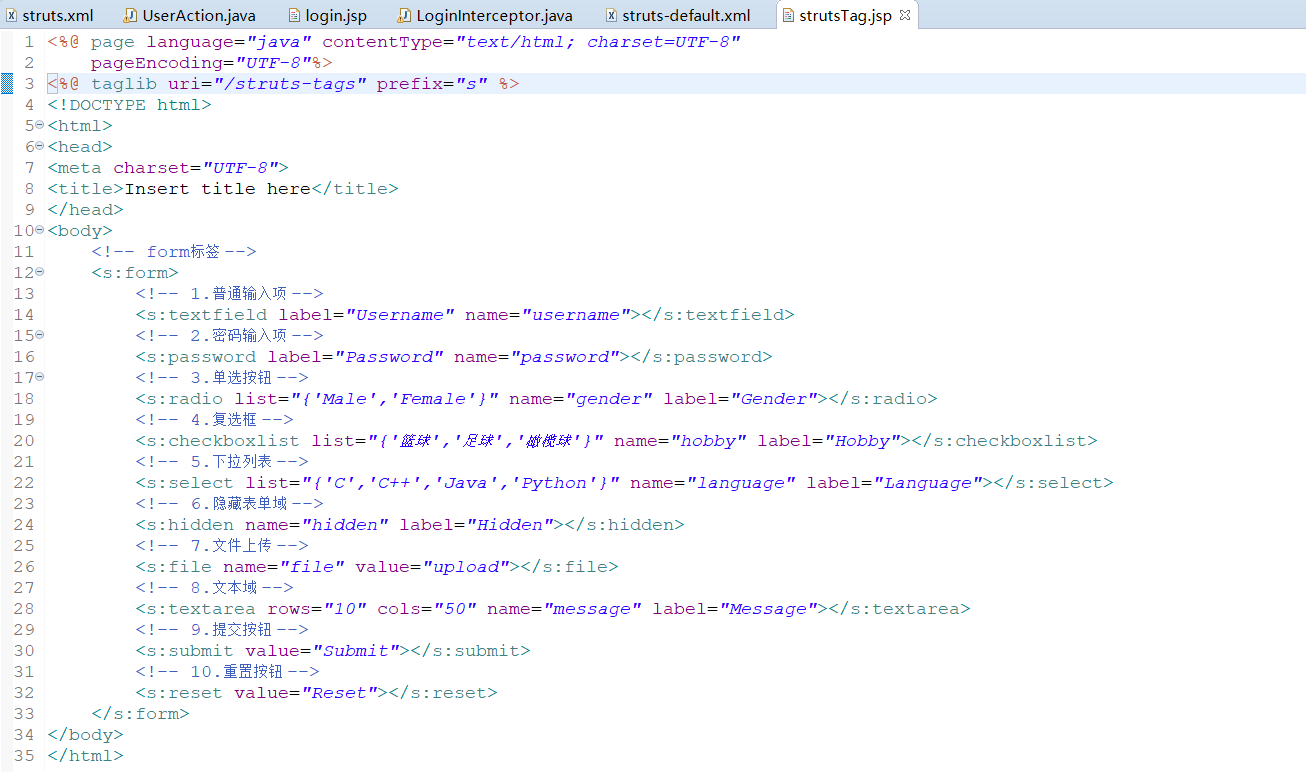


1. 注册拦截器（配置action和拦截器之间的关联关系）
   1. 在要拦截的action标签所在的package标签里面声明拦截器；
   2. 在具体的标签中使用已经声明的拦截器；
   3. struts2里面执行很多的默认拦截器，但是如果在action里面配置了自定义拦截器，默认的拦截器不会被执行；
   4. 配置action中某些方法，使其不进行拦截；因为拦截器在login方法前运行，所以Object一直为空。



## Struts2标签库

1. <s:property>标签：和ognl表达式使用，获取值栈数据；
2. <s:debug>标签：查看值栈结构和数据；
3. <s:iterator>标签：遍历值栈集合
4. Struts2表单标签





# 五、Struts2总结

## 1. Action

1. Action的创建（通常采用继承ActionSupport类）。
2. 配置访问Action的方法（通配符访问）。
3. Action获取表单或Ajax等提交的数据（建议使用模型驱动，注意模型驱动与属性封装使用的问题）。
4. Action操作域对象。
5. Result标签里面的type属性(转发/重定向到页面/Action)

## 值栈

1. 值栈结构
2. 向值栈放数据（主要是get方法）
3. 从值栈中获取数据

## 拦截器

1. 拦截器只能拦截Action，不能拦截页面。
2. 拦截器原理：AOP和责任链模式（不是放行操作，是类似放行的操作）
3. 自定义拦截器
4. 继承MethodFilterInterceptor类；
5. 写拦截器逻辑；
6. 配置拦截器与Action的关联；
7. 配置Action某些方法不进行拦截；