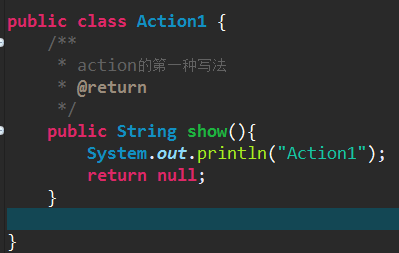
# Struts基础

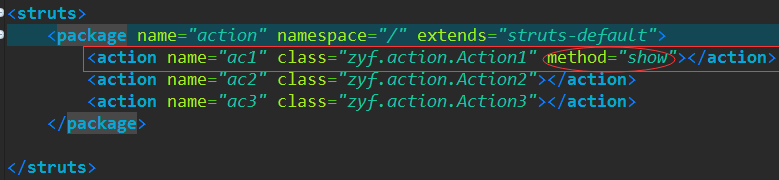
Svn插件安装地址：http://subclipse.tigris.org/update\_1.8.x

一、**Action类的三种写法**：

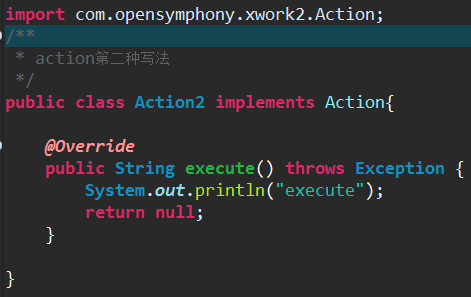
1. 普通类（在struts.xml文件中配置method属性，不加括号）

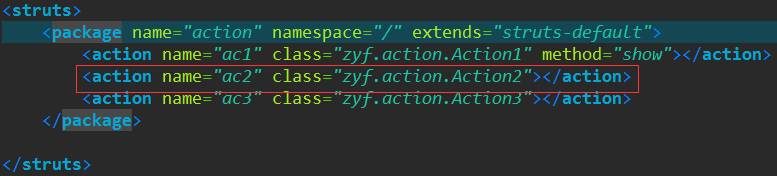


在struts.xml中的配置为：（method一定要配置，里面的值是Action类里面需要执行的方法名称）



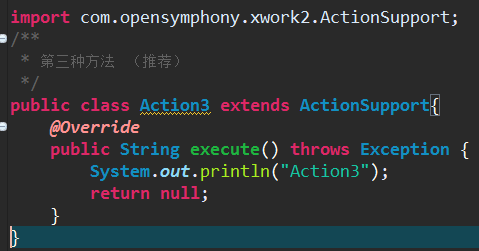
1. 实现Action接口



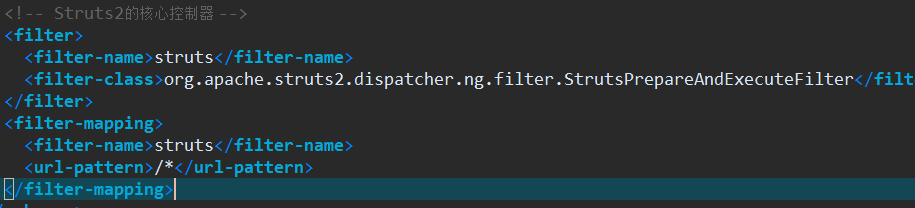


这里method的设置可以省略，默认为execute方法，下同

1. 继承ActionSupport类（推荐使用）



Web.xml的配置为：



服务器启动的时候，需要先加载Strues2环境，所以要在web.xml文件中配置Strues2环境

web.xml配置完成之后，Action的配置都在struts.xml文件中完成，且struts.xml文件必须放在src根目录下

<**struts**>

<!--

以package为一个整体

name:自定义的名字

namespace：访问时的虚拟路径

extends:必须写为struts-default

-->

<**package** name=*"user"* namespace=*"/user"* extends=*"struts-default"*>

<!-- name:请求的地址（相当于servlet里面的url-patten） -->

<!--

name:访问的url （要注意的是，在页面写路径时要在前面添加上项目名）

class:编写好的action类

method:action类中调用的方法

-->

<**action** name=*"user\_login"* class=*"zyf.action.UserAction"* method=*"login"*>

<!--

name:是UserAction类中login方法返回的值

type:默认是转发（不写的情况下也是转发）

标签中的文本：跳转的页面

-->

<**result** name=*"ok"* type=*"dispatcher"*>/index.jsp</**result**>

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

</**action**>

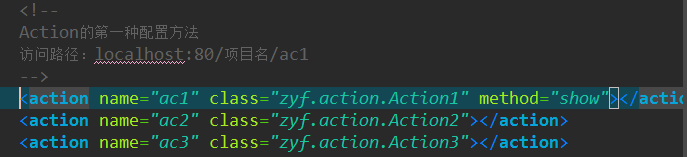
</**package**>

</**struts**>

访问一个Action时，在地址栏显示的地址为：域名:ip/项目名/命名空间（workspace）/url(action的name属性)

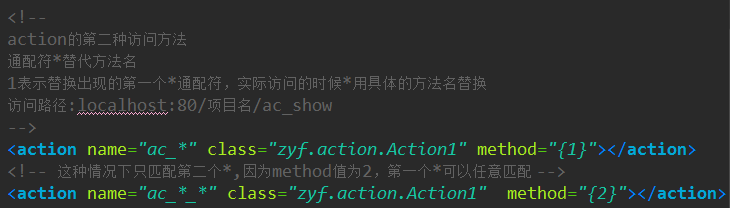
需要注意一点的是：在jsp页面中访问Action时，不能直接填写action的name属性，必须在前面加上项目名，可以用**${pageContext.request.contextPath}**动态方式表示项目名

1. **Action的三种配置方式**
2. 普通方法



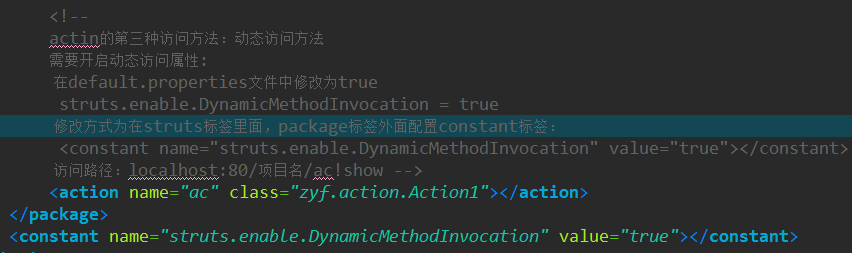
这种方法必须每一个方法配置一个action，比较麻烦 ，不推荐

1. 通配符替换方法



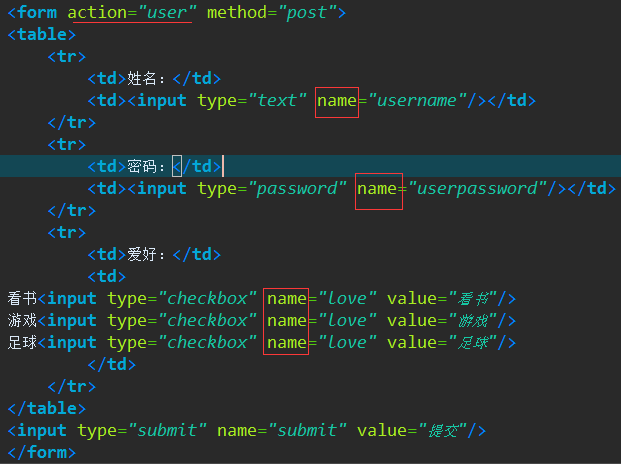
Method中的数字代表替换前面出现的第几个通配符，访问的时候用的是action中具体的方法，上面ac\_show中的show代表方法名

1. 动态访问方法



1. **在Action中属性封装的3种方式**
2. 使用ServletActionContext对象

a.前端页面：



b. struts.xml



c. Action类中的实现代码



1. 封装属性（属性驱动）（第一种：普通属性封装）

strust2属性封装的规范

\* 1.前段页面中name属性值必须与action中定义的属性名相同

\* 例子

\* <input type="text" name="username"/>

\* public XXXAction{

\* private String username;

\* }

\* 2.action中的属性必须要有setXXX()方法

\* 例子

\* <input type="text" name="username"/>

\* public XXXAction{

\* private String username;

\* public void setUsername(String username){

\* this.username=username;

\* }

\* }

\* 集合的封装

\* 1.action中定义集合，名字自定义（action中与前端页面必须一

\* 致，并且在集合名词定义之后必须创建对象，否则会出错） \* 同时还要有setXXX()、getXXX()方法

\* 2.★★★页面中的写法★★★

\* name=action中集合的名字[下标]

\* 例如：name=love[0] name=love[1]

\*/

/\*

\* 通用的属性封装：

\* 1.action 中提供属性，并提供getXXX()、setXXX()方法

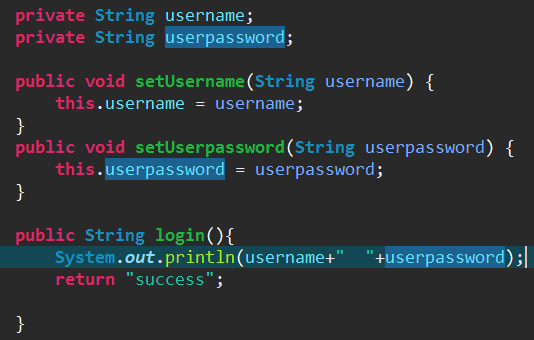
\* 2.页面中的name属性值和action中的属性名一致

\* 3.如果action中的属性是集合类型，页面中的name属性书写规范：name=action中集合的名字[下标]

\*

\*/

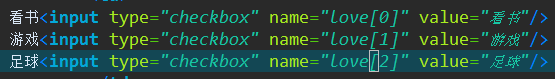
private ArrayList<String> love=new ArrayList<String>();



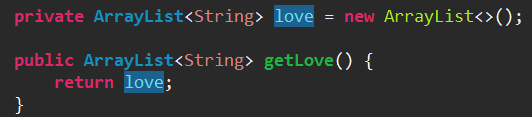
如上图所示，调用login可以直接得到前端页面传过来的普通属性的值（前提是前段页面中name属性值必须与action中定义的属性名相同）

如果前端传递过来的是一个数组或者集合，集合声明时一定要进行初始化，并且前端name属性值和action中集合名称要对应

前端name属性的写法为：

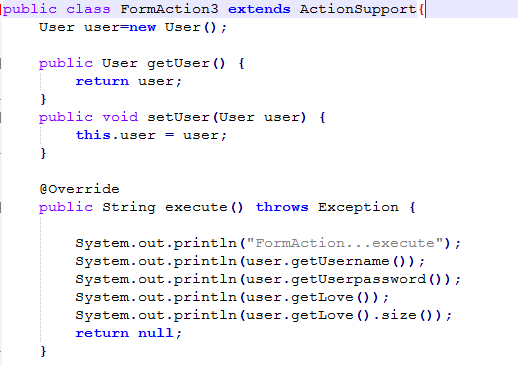


Action类中：



上面这种封装属性的方法是讲所有的属性都封装在action类中，很麻烦而且不容易阅读，所以模型驱动的另一个方式是讲所有的属性抽取出来变成一个实体类（这个实体类的属性名和前端参数名相同且实现set/get方法），但是在action必须有实体类声明的属性并且要实现该实体类的set/get方法，页面的属性写法为Action类中的实体类对象名.实体类中相应的属性名

属性驱动的第二种方式（对象属性封装）：



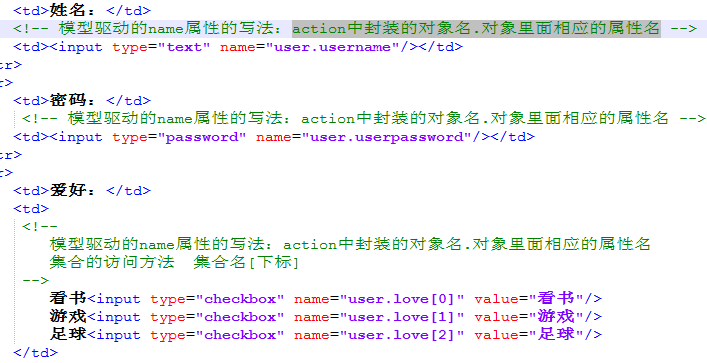


1. 使用模型（模型驱动）





模型驱动中页面的写法可以直接写实体类中相应的属性，因为实现了接口之后会自动加载模型（实体类）,页面提交的参数系统会自动在模型里面找，也可以像属性驱动一样在用 action中封装的对象名.对象里面相应的属性名 的方法



属性驱动与模型驱动的区别：

1. 属性驱动只能用的方法传递参数，模型驱动既可以用的方法，也可以直接用属性名传递参数（最好直接用属性名）
2. **分模块开发时struts.xml文件的配置**

添加额外的struts-XX.xml文件

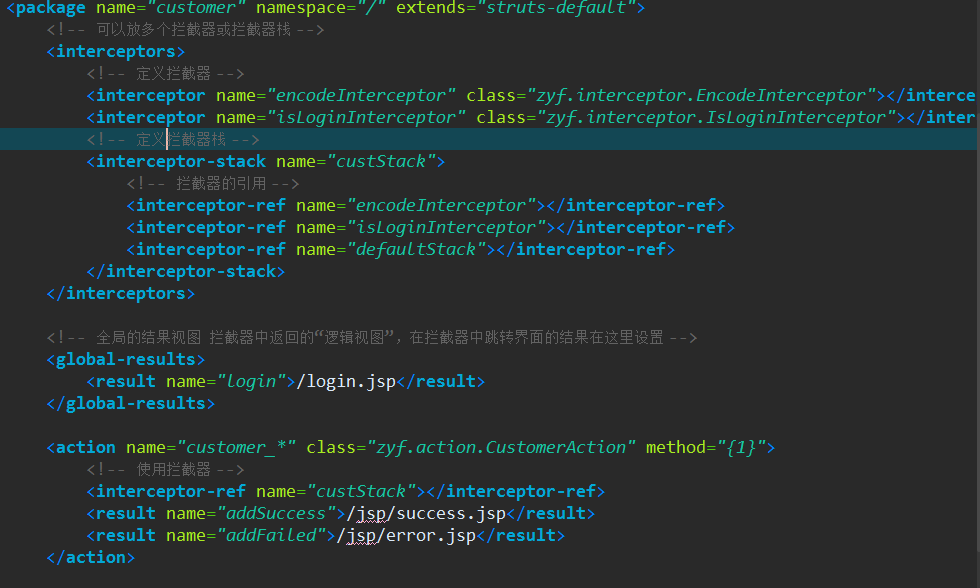
1.创建struts-xx.xml文件（自己定义，位置不限,但是格式要与struts.xml一致）

2.在主struts.xml中通过下面的标签来加载额外的struts-xx.xml文件

<include file="cn/oracle/oaec/struts/customer/struts-customer.xml"></include>

可以在src下任意文件夹下面创建任意名字的.xml配置文件，里面的内容格式必须与struts.xml配置文件格式一致，但是主struts.xml文件必须在src根目录下，并且名字必须为struts.xml，可以在struts.xml通过<include >标签引入，标签放的位置在<struts>标签当中

1. **拦截器**



**在<struts>标签中，只有常量、引入和包是平级的（constant、package、include），其他的标签都要写在package中**

**<interceptors>标签中可以放多个<interceptor> <interceptor-stack>**

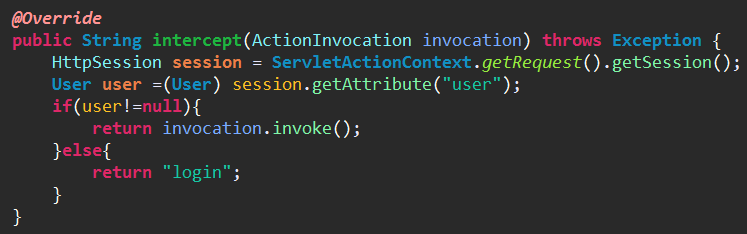
**在<interceptor>标签中配置拦截器，配置好拦截器之后，如果这个拦截器需要在多个Action中用到，就可以将该拦截器配置到拦截器栈中，使用时直接使用拦截器栈就可以了 （需要注意的是：如果不定义自己的拦截器，系统有默认的拦截器，一旦自己配置了拦截器，系统默认的拦截器就失去作用了，所以除了自己的拦截器外，要将系统默认的拦截器也配置进来，以便系统能正常工作）；如果拦截器只需要使用，就可以不用配置拦截器栈，直接在action中使用该拦截器（同时也要使用系统默认的拦截器）**

**拦截器返回的‘逻辑视图’不能在action的result中配置，需要在<global-results>标签的result中进行配置**

**需要注意的是:标签是有先后顺序的，这三个标签从前到后依次为：**

**Interceptors---->global-results----->action**

**具体参考以上截图**



实现拦截器有两种方式：**implements** Interceptor 和**extends** **MethodFilterInterceptor** ，在action中使用拦截器时，如果action的配置方式是用通配符配置或者动态加载配置，一旦使用实现接口的方式， 则该Action类中所有的方法都会被拦截；如果只需要拦截该Action类中的一部分方法，那么就不能用实现接口的方式了，而要用到**extends** **MethodFilterInterceptor** 的方式，并且要在Action类中使用的时候在**<param name=”excludeMethods”>需要过滤的方法名，多个方法名之间用“，”隔开</param>**  中配置不需要拦截的方法（默认所有的方法都拦截），<param>建议配置在action中使用的<**interceptor-ref>** 标签中，也可全局配置但不建议，如果在action中使用的是拦截器栈，则需要用 **具体的拦截器名.excludeMethods方式设置，如** **<param name=”拦截器名.excludeMethods”>需要过滤的方法名，多个方法名之间用“，”隔开</param>**

