# 学习方法：

做好笔记。Struts2内容比较多，比较杂。

多做实验，证明结论。

# Servlet PK Filter

Servlet：

void init(ServletConfig cfg);//初始化，读取Servlet的配置参数

void service(ServletRequest request,ServletResponse response);服务方法，用户的每次请求都会执行

void destory();//销毁方法

生命周期：单例的

----------------------------------------------------  
Filter：

void init(FilterConfig cfg);//初始化，读取Filter的配置参数

void doFilter(ServletRequest request,ServletResponse response,FilterChain chain);过滤方法，用户每次访问被过滤的资源，都会执行

void destory();//销毁方法

生命周期：单例的

总结：过滤器比Servlet强大，它能够决定是否让目标执行。

---------------------------

实际开发：Servlet在MVC开发模型中作为控制器存在，那么Filter也可以作为控制器。

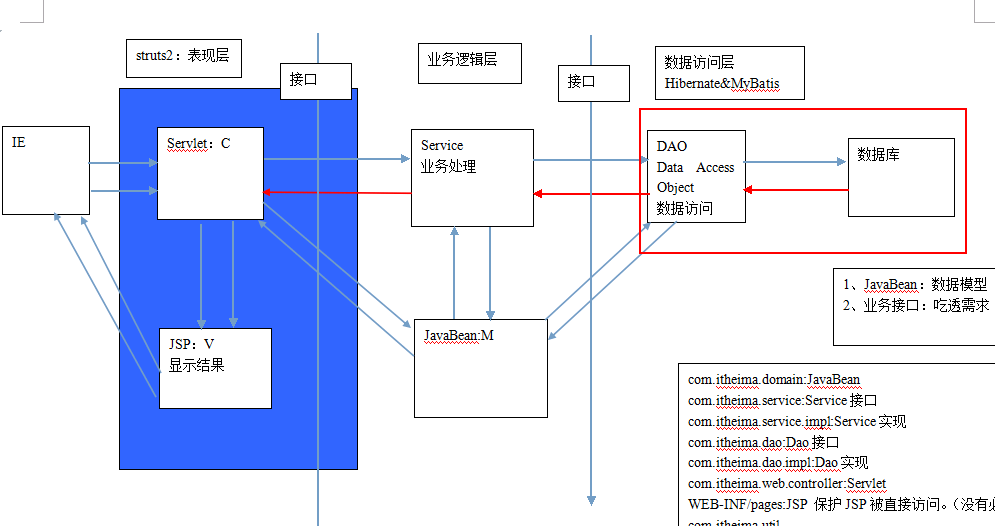
# Struts2简介

## **SUN推出的开发模型：**

模型1：JSP+JavaBean

模型2：MVC

## **Struts2是基于MVC开发模型的一个框架，属于表现层框架。（蓝色区域）**



## 分析之前的案例

JSP

Servlet:控制器

根据op执行对应的方法

用户的请求：

op=doXXX

框架都是半成品，程序员在此基础上开发实现自己业务功能的代码。框架是把程序员从繁琐的重复性的代码中解脱出来，提高开发效率。

JSP等

动作类：

doXXX方法

Filter：控制器

用户发出请求

## Struts2的小历史

Struts2：Struts+WebWork框架

Struts1:2003左右很火。

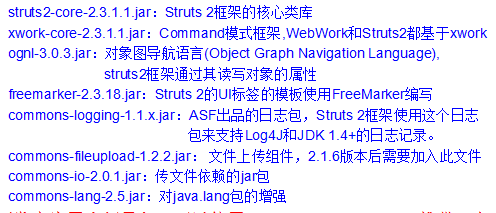
# 搭建开发环境

## 下载Struts2的发行包。

<http://struts.apache.org>



## 新建一个JavaWeb工程，拷贝以下jar包到lib目录中



小技巧：从Struts2发行包下面apps\struts2-blank\WEB-INF\lib,把其中jar都拷贝过来即可。

## 在构建路径的顶端，创建一个配置文件struts.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"

"http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>

</struts>

## 配置控制器映射：框架提供

在web.xml中配置：

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app version=*"2.5"*

xmlns=*"http://java.sun.com/xml/ns/javaee"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee*

*http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*>

<display-name></display-name>

<!-- 配置Struts2框架的核心控制器 -->

<filter>

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

<filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>

</filter>

<filter-mapping>

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

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

</filter-mapping>

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

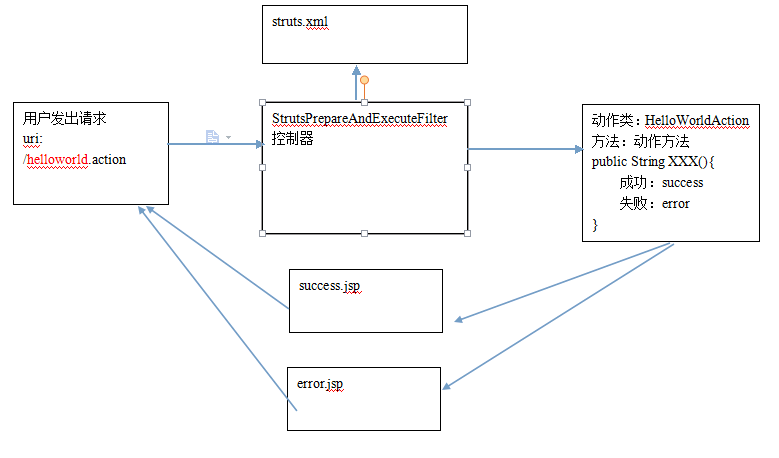
</web-app>

## 验证：

部署到Tomcat中，启动，如果没有错，成功

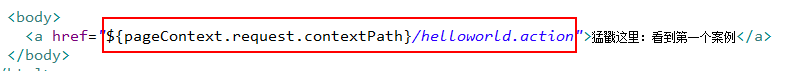
# 第一个Struts2案例

## Struts2的基本执行原理

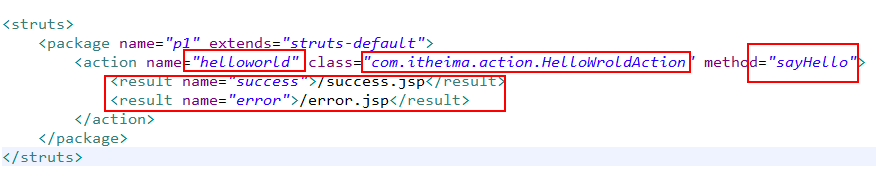


## 第一个案例的开发过程

### 发出请求给框架



### 修改struts.xml配置文件



<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"

"http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>

<package name=*"p1"* extends=*"struts-default"*>

<action name=*"helloworld"* class=*"com.itheima.action.HelloWroldAction"* method=*"sayHello"*>

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

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

</action>

</package>

</struts>

### 编写动作类和动作方法

**public** **class** HelloWroldAction {

**public** String sayHello(){

System.*out*.println("动作方法执行了");

**return** "error";

}

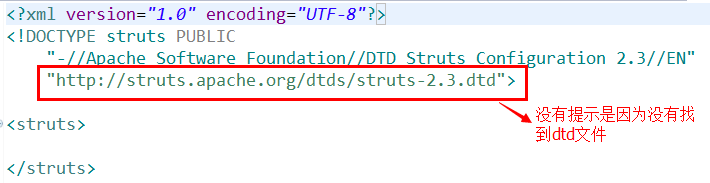
}

### 编写success.jsp或error.jsp页面

### 启动访问即可

# 编写Struts.xml配置文件没有提示

没有提示就是因为没有找到dtd或者schema造成的

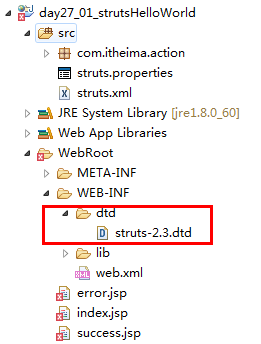


## 联网（联网一次即可）

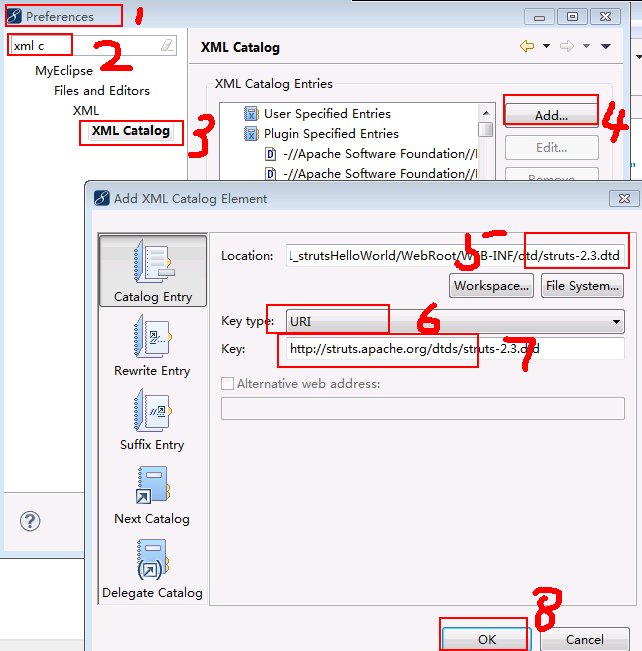
## 没有联网：手工配置一下

就是将dtd的网络地址与本地的dtd文件做个对应关系：

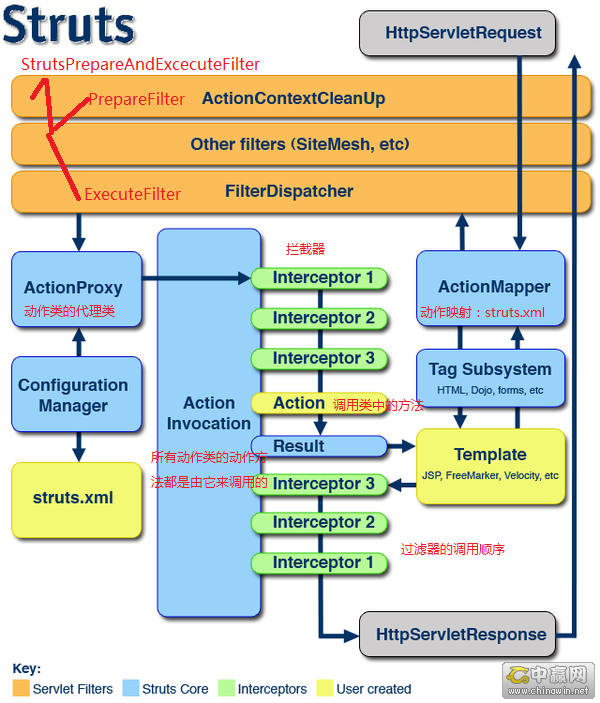
1.在项目中新建目录存放dtd文件



2.配置eclipse



# Struts2的执行过程：非常全面的图



# Struts2的配置文件

## **Struts2框架会按照以下顺序加载配置文件**

**a、default.properties:struts2-core\*\*.jar org.apache.struts包中（程序员只能看）**

**b、struts-default.xml：struts2-core\*\*.jar中（程序员只能看）**

**c、struts-plugin.xml:在插件的jar包中（程序员只能看）**

**d、struts.xml：在应用的构建路径顶端。自己定义的Struts配置文件（推荐）**

**e、struts.properties:在应用的构建路径顶端。程序员可以编写（不推荐）**

**f、web.xml:配置过滤器时，指定参数。程序员可以编写（不推荐）**

**特别注意：顺序是固定的。后面的配置会覆盖前面的同名配置信息。**

**代码验证：**

Struts2的默认请求路径是\*\*\*.action，改为\*\*\*.do

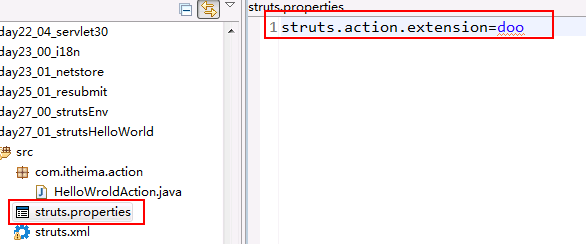
此处的默认配置在default.properties中。



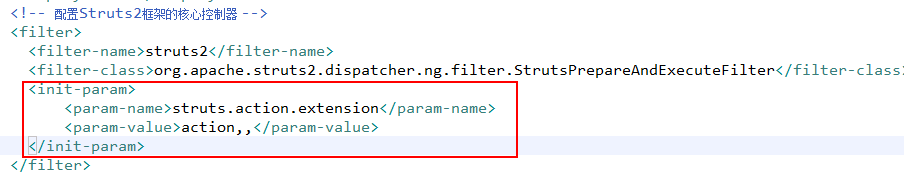
### struts.xml覆盖：



### struts.properties覆盖：



### web.xml覆盖：

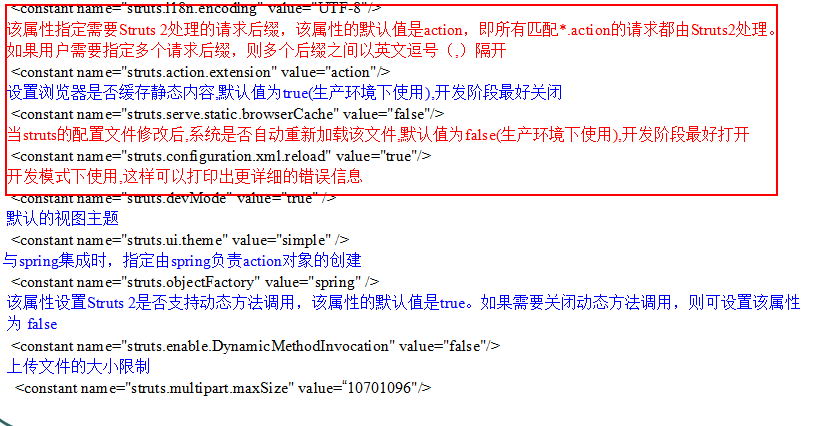


## 更改Struts2默认的参数

<!-- 开发模式：每次访问时都会重新读取xml配置文件信息 -->

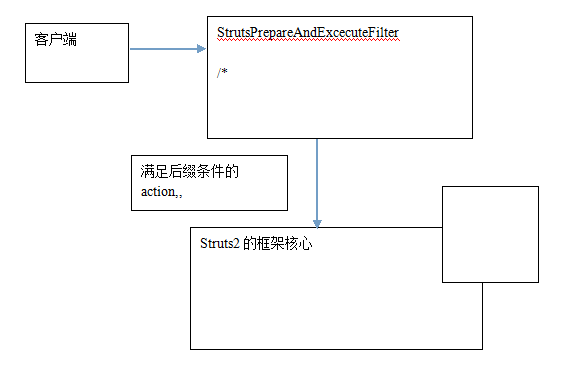
<constant name=*"struts.devMode"* value=*"true"*></constant>

<constant name=*"struts.action.extension"* value=*"do"*></constant>



## 后缀名的一点补充

**客户端所有的请求都经过过滤器的拦截，但是只有满足后缀条件的请求才会进入到Struts2的框架核心内部**



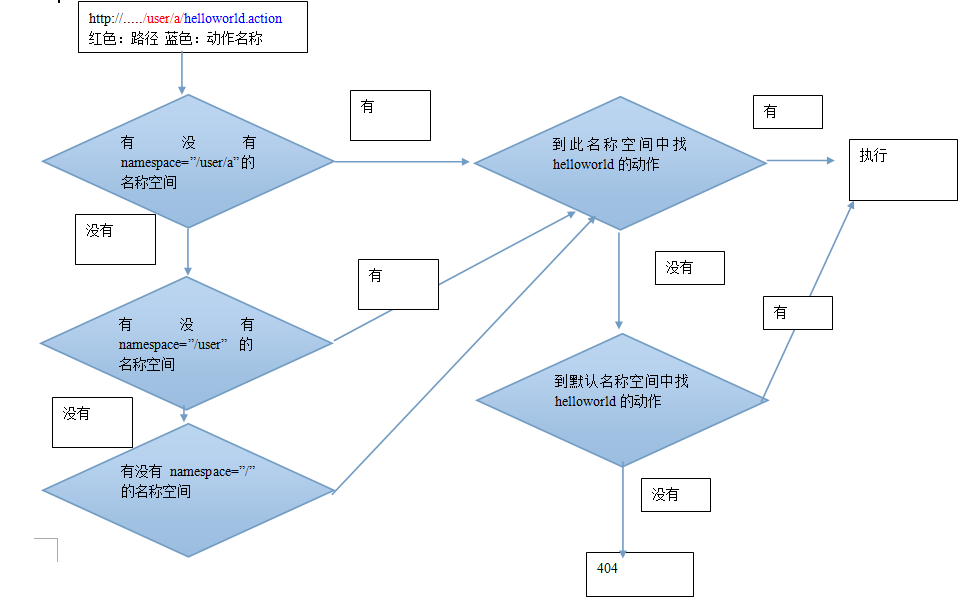
## package元素：（相当于java中的包）

意义：分模块开发。

属性：

* name:必须的。配置文件中要唯一。就是一个名字。
* extends:指定父包。会把父包中的配置内容继承下来。一般需要直接或间接的继承一个叫做“struts-default”的包(在struts-default.xml配置文件中)。如果不继承该包，那么Struts2中的核心功能将无法使用。
* abstract：是否是抽象包。没有任何action子元素的package可以声明为抽象包。
* namespace：指定名称空间。一般以”/”开头。该包中的动作访问路径：namesapce+动作名称。如果namespace=””，这是默认名称空间，和不写该属性是一样的。而namespace=”/”不是默认名称空间,它和namespace=””不同，是个实际的名称空间

名称空间的搜索顺序图：



## action元素：

作用：定义一个动作。

属性：

* name：必须的。动作名称。用户用于发起请求。在包中要唯一。
* class：指定动作类的全名。框架会通过反射机制实例化。默认是：com.opensymphony.xwork2.ActionSupport。
* method：指定动作类中的动作方法。框架会执行该方法。默认是execute()。

-----------------

关于默认类：在struts-default.xml中进行了声明

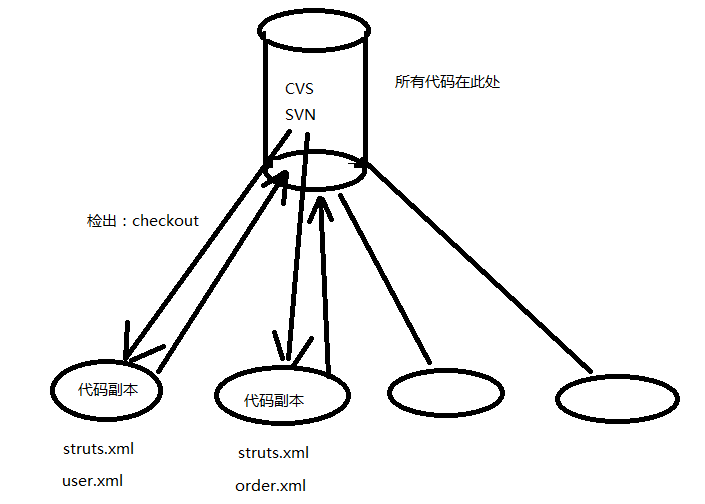


可以在自己的配置文件中覆盖它。



## include元素

分配置文件的意义：



配置：

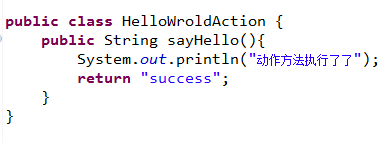




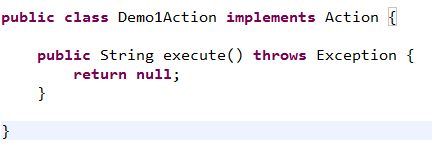
# 动作类：

## 编写动作类的三种方式：

### a、POJO（Plain Old Java Object）普通的JavaBean。



### b、实现com.opensymphony.xwork2.Action接口 (实现它的意义只是多了几个常量而已)



Action接口中的常量：

String SUCCESS:success。一切正常。

String NONE:none。动作方法执行后，不转向任何的结果视图。或者在动作方法中返回null。

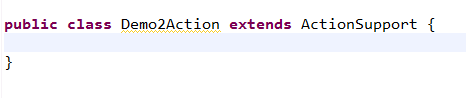
String ERROR:error。动作方法执行时遇到异常，转向错误提示页面。

String INPUT：input。验证、转换失败，转向输入页面。

String LOGIN：login。检测用户是否登录，没有登录转向此视图。

### c、继承com.opensymphony.xwork2.ActionSupport（**推荐**）

意义：提供了一些基本的功能。比如验证和国际化消息提示等。

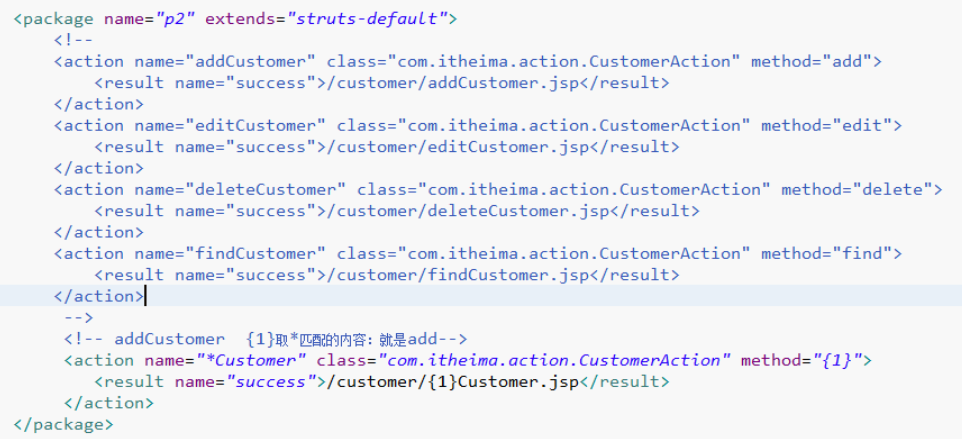


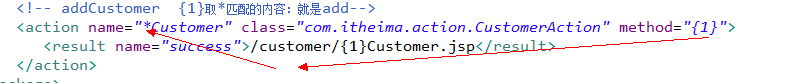
## 动作类中的动作方法

编写要求：public String XXX(){}

## 动作通配符映射

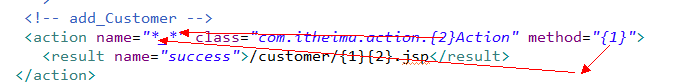
### 配置





可以使用通配符：\*,{1}匹配的就是第一个\*

### 可以匹配全部动作类的配置：

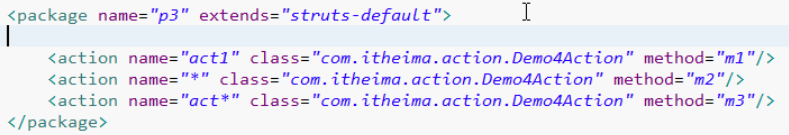


{1}匹配的是第一个\*，{2}匹配的是第二个\*

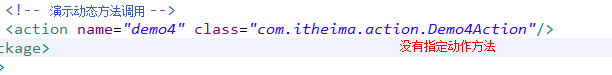
### 细节：

存在多个匹配的情况：

绝对匹配----->按照通配符匹配的配置顺序为准。



## 动态方法调用：DMI（Dynamic Method Invocation）



在访问时指定执行的动作方法：

<http://localhost:8080/XXX/demo4!m1> 就是执行demo4动作对应动作类的m1动作方法

默认情况下：会报错，说找不到demo4!m1这个动作方法。因为struts2框架默认是禁止动态方法调用的。

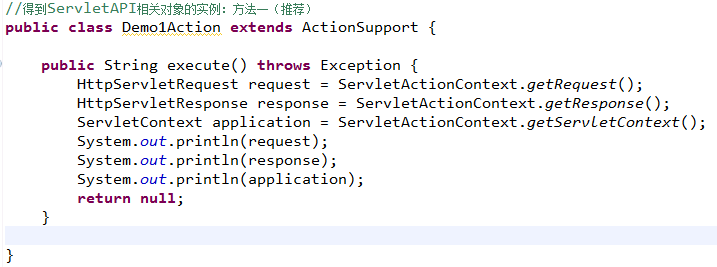
可以通过覆盖默认参数的方式来开启动态方法调用：



注意：不建议使用。容易暴露动作类中的动作方法，害怕非法执行。（用户可以自己执行一些不允许用户访问的方法）

# 动作类中获取ServletAPI

## 方式一：推荐。简单（使用ServletActionContext类）

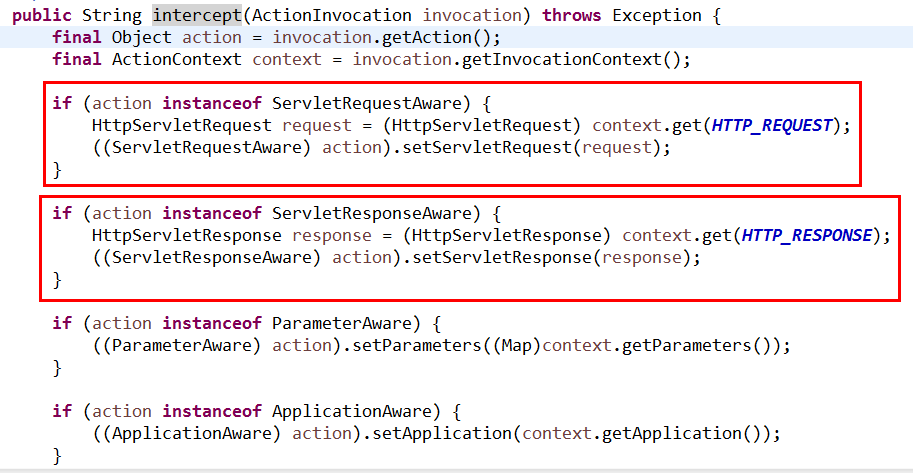


## 方式二：麻烦。（实现对应的XXXAware接口来得到实例）



原理：

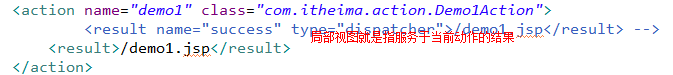
通过拦截器（<interceptor name=*"servletConfig"* class=*"org.apache.struts2.interceptor.ServletConfigInterceptor"*/>）给你注入的这些实例。



# 结果视图

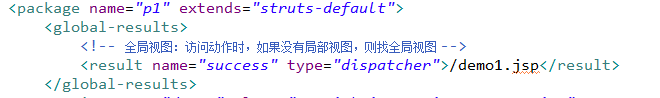
## 局部和全局结果视图

a、局部视图：就是只服务于当前动作的视图



b、全局视图：访问动作时，如果没有局部视图，则找全局视图（局部视图的优先级高，查找全局视图时也是按照名称查找）

多个动作对应同一个视图



全局视图可以用于指定项目的全局视图，比如访问不存在的路径时跳转到指定页面，可以定义一个抽象包，其他的包都继承它:

<package name=*"default"* extends=*"struts-default"* abstract=*"true"*>

<global-results>

<result name=*"success"* type=*"dispatcher"*>/demo1.jsp</result>

</global-results>

</package>

<package name=*"p1"* extends=*"default"*>

<action name=*"demo1"* class=*"com.itheima.action.Demo1Action"*>

<result>/demo1.jsp</result>

</action>

<action name=*"demo2"*>

</action>

</package>

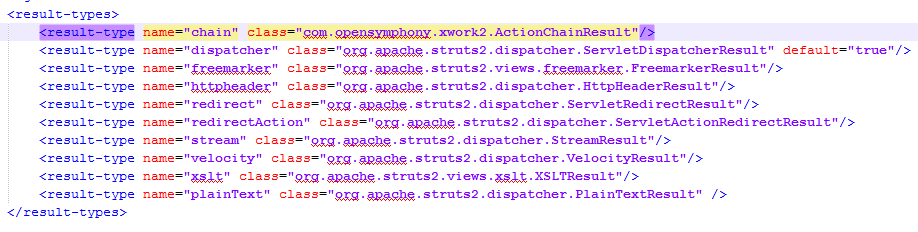
## result元素的配置：

**属性：**

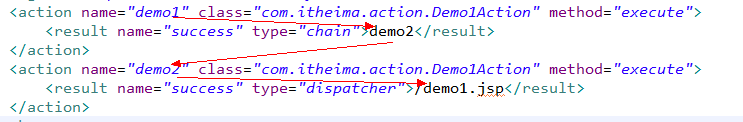
* name：逻辑视图名称。它对应的是动作方法的返回值。默认值：success。
* type：到达目标的形式。默认值：dispatcher。请求转发。

## Struts2提供的结果类型（result中的 type属性）

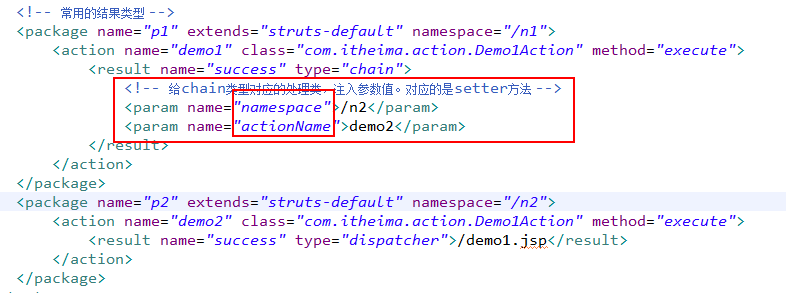
在struts-default.xml中有定义



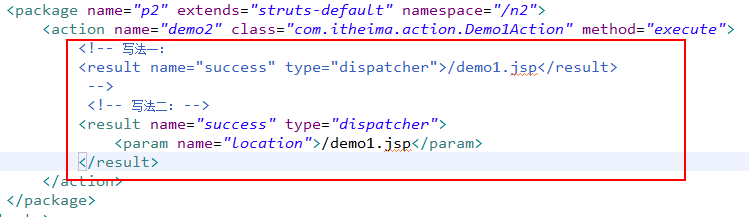
* chain:用于转发到另外一个动作。



转发到不同名称空间中的另外一个动作：



* dispatcher：用于转发到另外一个JSP页面。



* freemarker：用于转发到另外一个freemarker模板。（页面静态化）
* velocity：用于转发到另外一个velocity模板。
* httpheader：用于输出http协议的消息头。
* xslt：XML有关的样式
* redirect：用于重定向到另外一个JSP页面。
* redirectAction：用于重定向到另外一个动作。
* stream：用于文件下载（日后再讲。文件上传和下载）
* plainText：以纯文本的形式展现页面。输出源码。

## 自定义结果视图

随机验证码图片

### struts2的执行图

放行

用户请求

StrutsPrepareAndExcecuteFilter

执行

不放行

action

结果：result

框架核心

自己负责输出

测试：

<package name=*"p4"*>

<action name=*"demo3"* class=*"com.itheima.action.Demo3Action"* method=*"doSomething"*></action>

</package>

**public** **class** Demo3Action {

**public** String doSomething() **throws** IOException{

HttpServletResponse response = ServletActionContext.*getResponse*();

response.getWriter().write("yes");

**return** **null**;

}

}

上面的测试中，p4包没有继承struts-default包，从而没有使用result结果视图，但是也能输出页面，这就说明输出是由框架来做的，生成页面的内容是自己使用代码输出的，所以如果使用结果视图的话，就是将输出页面的代码放到结果类型所对应的处理类中的处理方法中

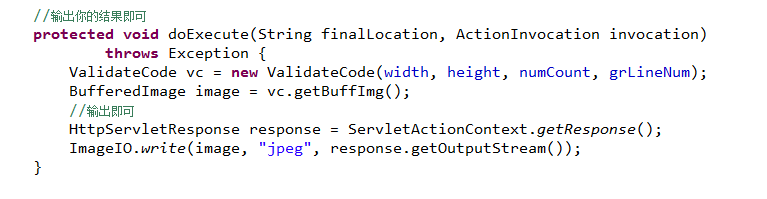
**如果是action结尾的就不放行，进入框架核心，输出是框架来输出的，是由结果类负责输出页面的**

**自定义结果视图就是继承视图类，将你要输出到页面的代码放入到实现的结果类中负责输出页面的方法中即可**

### 自定义结果视图步骤



a、编写一个类，直接或间接实现com.opensymphony.xwork2.Result接口。一般继承org.apache.struts2.dispatcher.StrutsResultSupport类。实现其中的抽象方法，方法中的内容就是你要输出的页面的代码



**public** **class** CaptachaResult **extends** StrutsResultSupport{

**private** **int** width = 120;

**private** **int** height = 80;

**private** **int** numCount = 4;

**private** **int** grLineNum = 100;

**public** **void** setWidth(**int** width) {

**this**.width = width;

}

**public** **void** setHeight(**int** height) {

**this**.height = height;

}

**public** **void** setNumCount(**int** numCount) {

**this**.numCount = numCount;

}

**public** **void** setGrLineNum(**int** grLineNum) {

**this**.grLineNum = grLineNum;

}

//输出你的结果即可

**protected** **void** doExecute(String finalLocation, ActionInvocation invocation)

**throws** Exception {

ValidateCode vc = **new** ValidateCode(width, height, numCount, grLineNum);

BufferedImage image = vc.getBuffImg();

//输出即可

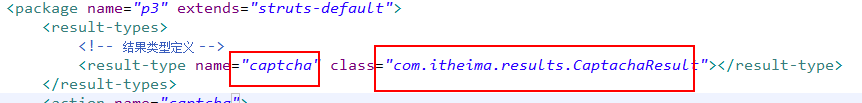
HttpServletResponse response = ServletActionContext.*getResponse*();

ImageIO.*write*(image, "jpeg", response.getOutputStream());

}

}

b、声明结果类型，然后才能使用



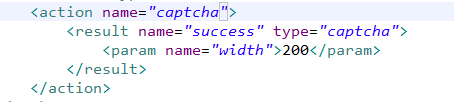
<result-types>

<!-- 结果类型定义 -->

<result-type name=*"captcha"* class=*"com.itheima.results.CaptachaResult"*></result-type>

</result-types>

c、使用



<action name=*"captcha"*>

<result name=*"success"* type=*"captcha"*>

<param name=*"width"*>200</param>

</result>

</action>

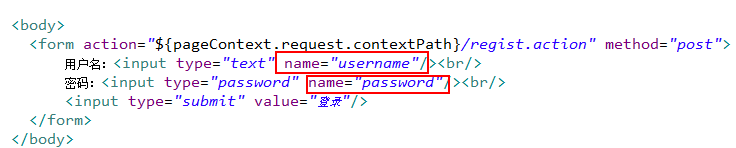
# 封装请求参数（特别重要）

## 动态参数注入：（请求参数，用户输入的）

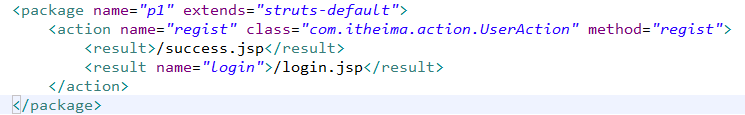
### 方式一：用Action动作类作为模型对象。

在动作类中加入javabean中的属性，并提供setter方法，框架会按照表单的name值，调用对应的setter属性

页面：



配置文件：



<package name=*"p1"* extends=*"struts-default"*>

<action name=*"regist"* class=*"com.itheima.action.UserAction"* method=*"regist"*>

<result>/success.jsp</result>

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

</action>

</package>

动作类：



//封装参数：方式一：动作类当做模型对象

**public** **class** UserAction **extends** ActionSupport {

**private** String username;

**private** String password;

**public** String getUsername() {

**return** username;

}

//框架会按照表单的name值，调用对应的setter属性

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getPassword() {

**return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

**public** String regist(){

System.***out***.println(username+":"+password);

**if**("wzhting".equals(username)&&"sorry".equals(password))

**return** ***SUCCESS***;

**else**

**return** ***LOGIN***;

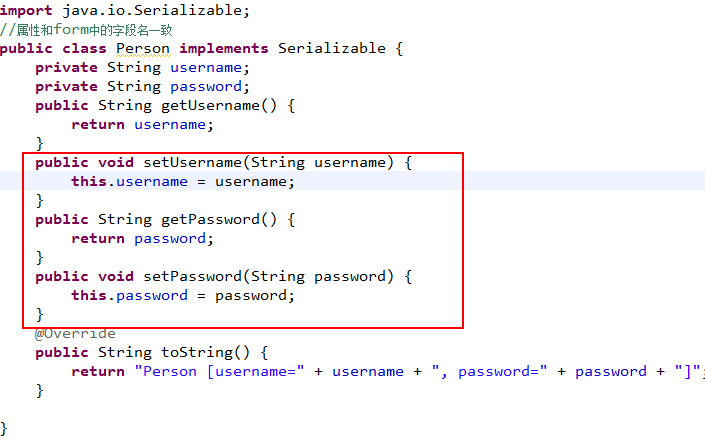
}

}

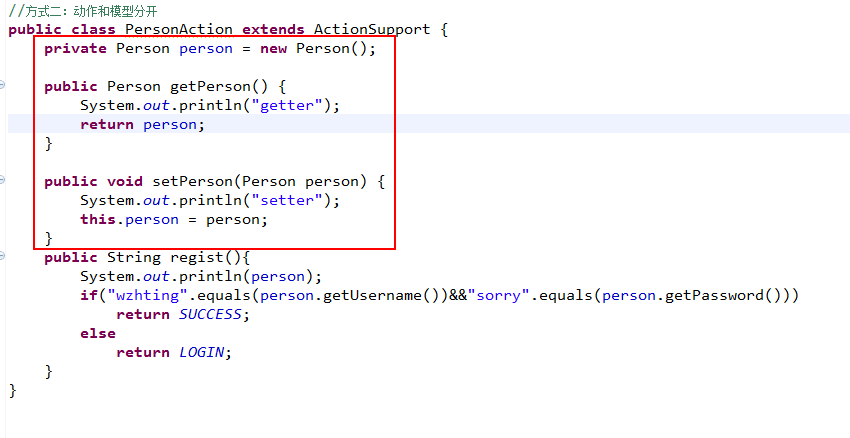
### 方式二：动作类和模型分开（推荐）

这种方式在页面中的写法要注意，采用的是ognl表达式person.username，是先调用动作类中getPerson方法，再调用setUsername方法，调用getPerson方法的时候，会调用两次，第一次调用是判断对象是否为空，如果为空会调用setter方法设置对象，最后调用getter方法得到对象引用

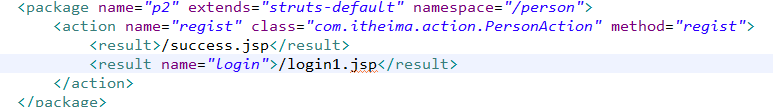
模型：



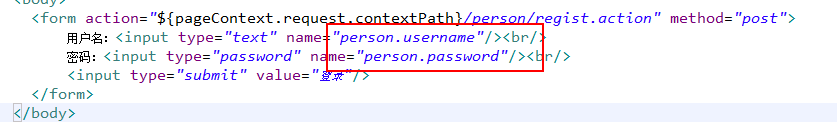
动作类：



配置文件：



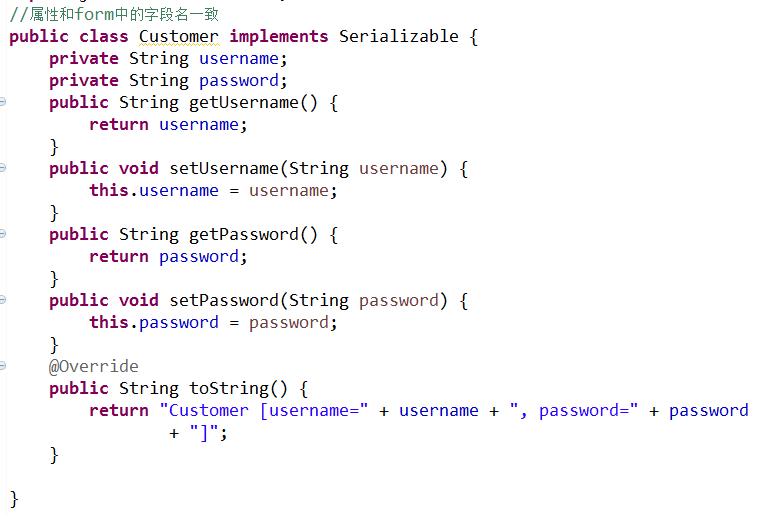
表单页面：



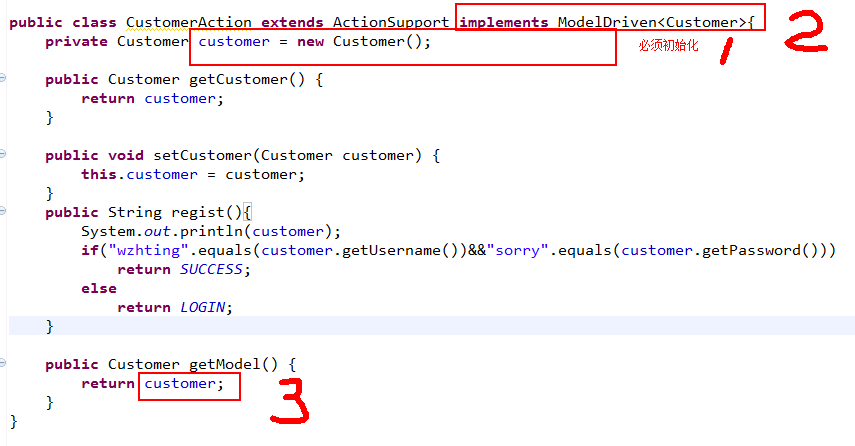
### 方式三：**模型驱动**（面试题，推荐）动作和模型分开

**模型驱动：ModelDriven**

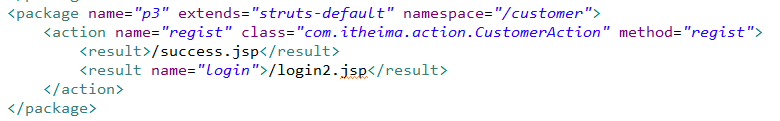
模型：



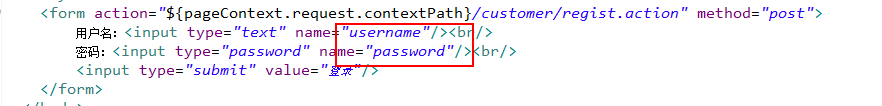
\*动作类：



配置文件：

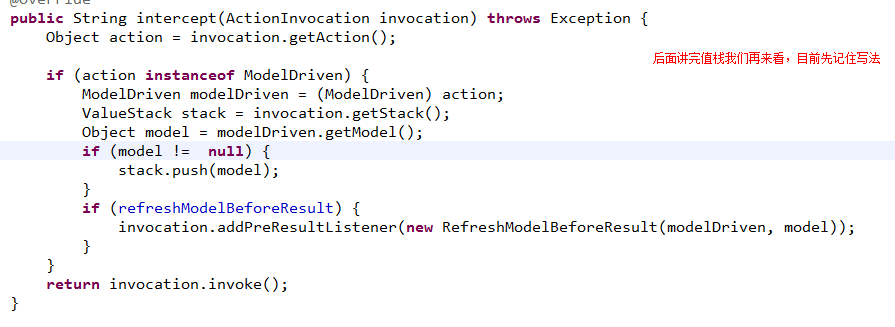


JSP页面：



原因：与Struts2的值栈有关

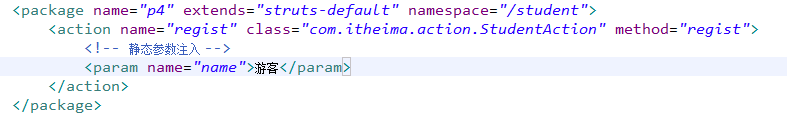
该功能是由一个叫做modelDriven的拦截器完成的：

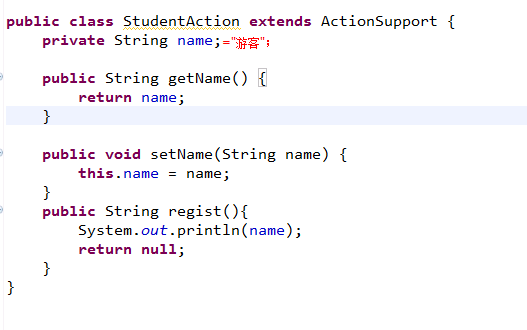


**动作类中的getModel方法会在动作方法执行前先执行；得到当前对应的模型对象并把模型对象压入值栈的栈顶。setUsername先调用栈顶对象的该方法。**

## 静态参数注入（相当于给参数设置一个默认值）

配置：





## 动态参数和静态参数注入功能实现（知道）

是由两个拦截器来完成。

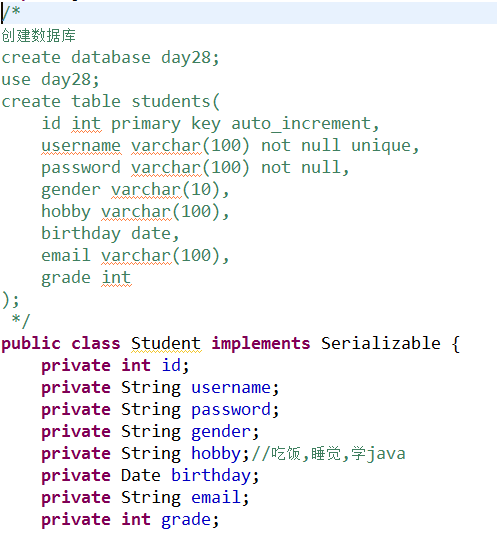
静态参数注入：staticParams

动态参数注入：params

# 学生的注册案例

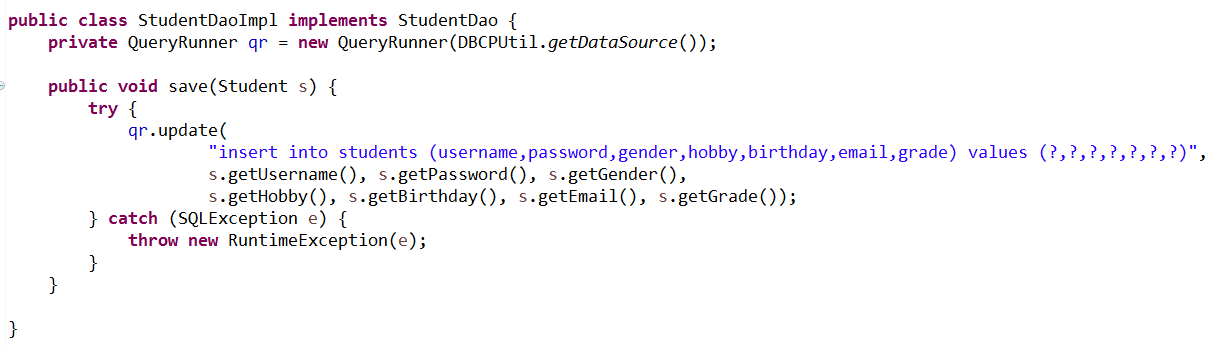
## 代码示例：

模型：

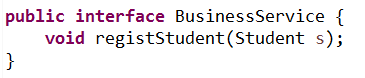


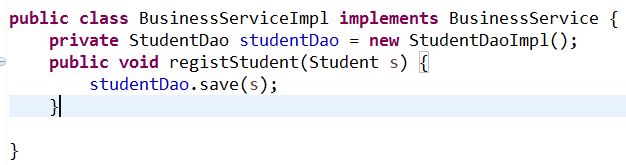
Dao层：



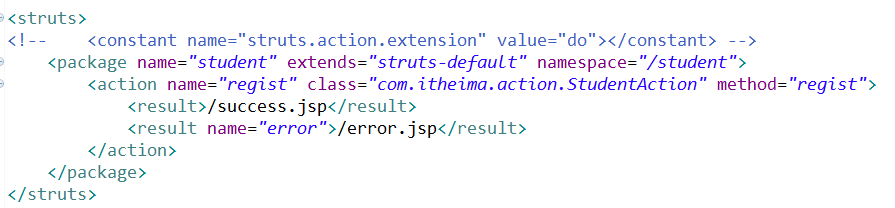


service层：





配置：



注册页面：

<form action=*"*${pageContext.request.contextPath}*/student/regist"* method=*"post"*>

<table border=*"1"* width=*"438"*>

<tr>

<td>用户名：</td>

<td>

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

</td>

</tr>

<tr>

<td>密码：</td>

<td>

<input type=*"password"* name=*"password"*/>

</td>

</tr>

<tr>

<td>性别：</td>

<td>

<input type=*"radio"* name=*"gender"* value=*"male"* checked=*"checked"*/>男性

<input type=*"radio"* name=*"gender"* value=*"female"*/>女性

</td>

</tr>

<tr>

<td>爱好：</td>

<td>

<input type=*"checkbox"* name=*"hobbies"* value=*"吃饭"*/>吃饭

<input type=*"checkbox"* name=*"hobbies"* value=*"睡觉"*/>睡觉

<input type=*"checkbox"* name=*"hobbies"* value=*"学java"*/>学java

</td>

</tr>

<tr>

<td>出生日期：(yyyy-MM-dd)</td>

<td>

<input type=*"text"* name=*"birthday"*/>

</td>

</tr>

<tr>

<td>邮箱：</td>

<td>

<input type=*"text"* name=*"email"*/>

</td>

</tr>

<tr>

<td>成绩：</td>

<td>

<input type=*"text"* name=*"grade"*/>

</td>

</tr>

<tr>

<td colspan=*"2"*>

<input type=*"submit"* value=*"注册"*/>

</td>

</tr>

</table>

</form>

动作类：

**public** **class** StudentAction **extends** ActionSupport **implements** ModelDriven<Student> {

**private** Student student = **new** Student();//模型

**private** BusinessService s = **new** BusinessServiceImpl();

**private** String [] hobbies;//小特殊

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

**return** hobbies;

}

**public** **void** setHobbies(String[] hobbies) {

**this**.hobbies = hobbies;

}

**public** String regist(){

**try** {

//单独处理爱好

**if**(hobbies!=**null**&&hobbies.length>0){

StringBuffer sb = **new** StringBuffer();

**for**(**int** i=0;i<hobbies.length;i++){

**if**(i>0)

sb.append(",");

sb.append(hobbies[i]);

}

student.setHobby(sb.toString());

}

s.registStudent(student);

**return** ***SUCCESS***;

} **catch** (Exception e) {

e.printStackTrace();

**return** ***ERROR***;

}

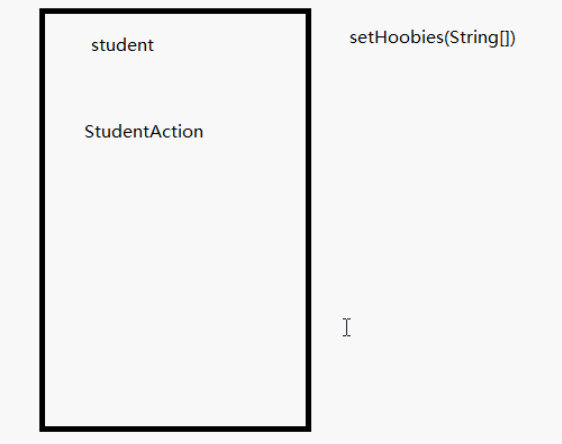
}

**public** Student getModel() {

**return** student;

}

}



## 表单标签

页面引入Struts2标签库：

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

使用Struts2的表单标签：



<!-- 推荐使用Struts2的标签库 -->

<s:form action=*"regist"* namespace=*"/student"*>

<s:textfield name=*"username"* label=*"用户名"* requiredLabel=*"true"* requiredPosition=*"left"*></s:textfield>

<s:password name=*"password"* label=*"密码"* requiredLabel=*"true"* requiredPosition=*"left"*></s:password>

<!-- list的取值是一个ONGL表达式，创建了一个Map对象。必须有name -->

<s:radio name=*"gender"* list=*"*#{'male':'男性','female':'女性'}*"* label=*"性别"* requiredLabel=*"true"* requiredPosition=*"left"*></s:radio>

<!-- list的取值是一个ONGL表达式，创建了一个List对象。必须有name -->

<s:checkboxlist name=*"hobbies"* list=*"{'吃饭','睡觉','学java'}"* label=*"爱好"*></s:checkboxlist>

<s:textfield name=*"birthday"* label=*"出生日期：yyyy-MM-dd"*></s:textfield>

<s:textfield name=*"email"* label=*"邮箱"*></s:textfield>

<s:textfield name=*"grade"* label=*"成绩"*></s:textfield>

<s:submit value=*"注册"*></s:submit>

</s:form>

## 批量添加表单的写法

页面：



<body>

<form action=*"*${pageContext.request.contextPath}*/customer/regist.action"* method=*"post"*>

用户名：<input type=*"text"* name=*"customers[0].username"*/>&nbsp;&nbsp; 密码：<input type=*"password"* name=*"customers[0].password"*/><br/>

用户名：<input type=*"text"* name=*"customers[1].username"*/>&nbsp;&nbsp; 密码：<input type=*"password"* name=*"customers[1].password"*/><br/>

用户名：<input type=*"text"* name=*"customers[2].username"*/>&nbsp;&nbsp; 密码：<input type=*"password"* name=*"customers[2].password"*/><br/>

<input type=*"submit"* value=*"注册"*/>

</form>

<form action=*"*${pageContext.request.contextPath}*/customer/regist1.action"* method=*"post"*>

用户名：<input type=*"text"* name=*"customers1['c1'].username"*/>&nbsp;&nbsp; 密码：<input type=*"password"* name=*"customers1['c1'].password"*/><br/>

用户名：<input type=*"text"* name=*"customers1['c2'].username"*/>&nbsp;&nbsp; 密码：<input type=*"password"* name=*"customers1['c2'].password"*/><br/>

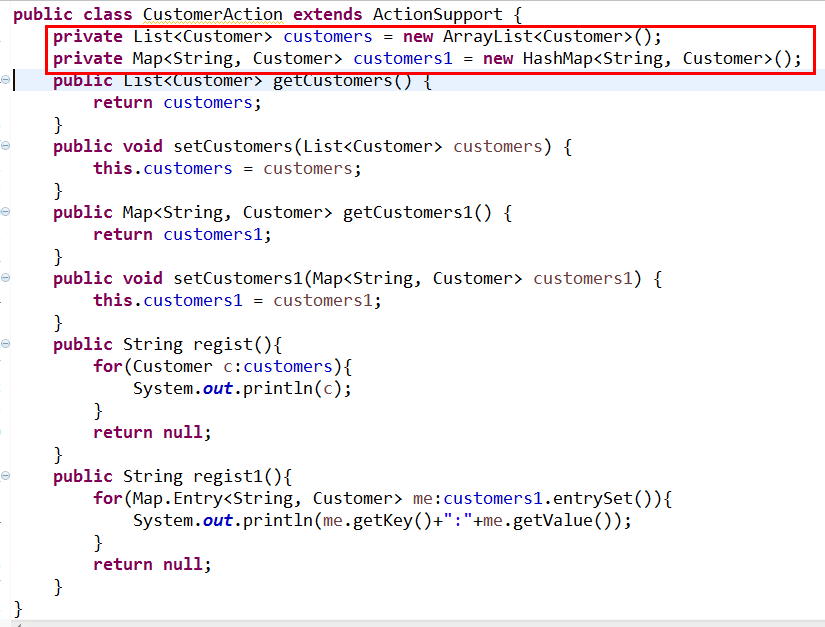
用户名：<input type=*"text"* name=*"customers1['c3'].username"*/>&nbsp;&nbsp; 密码：<input type=*"password"* name=*"customers1['c3'].password"*/><br/>

<input type=*"submit"* value=*"注册"*/>

</form>

</body>

动作类：



# 类型转换（学，开发中几乎不用管）

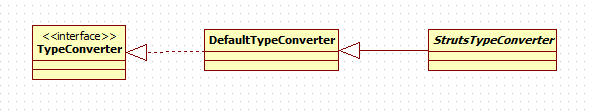
String----->java.util.Date：输入

java.util.Date------>String:输出：查询显示，输出错误回显

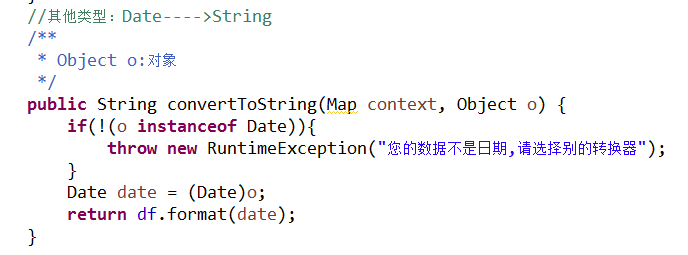
Struts2会按照本地默认（与操作系统的区域有关yyyy-MM-dd）格式自动转换

## 目标：实现MM/dd/yyyy<------>java.util.Date互相转换

### a、编写一个类，继承StrutsTypeConverter





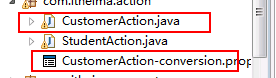


### b、注册类型转换器

* 局部类型转换器：只为当前动作使用



1. 动作类作为模型：

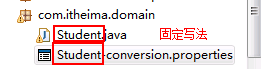


文件内容：



1. 动作类和模型分开：（经常用）

在模型所在的包中，建立以下配置文件



文件内容：



* 全局类型转换器：所有动作使用



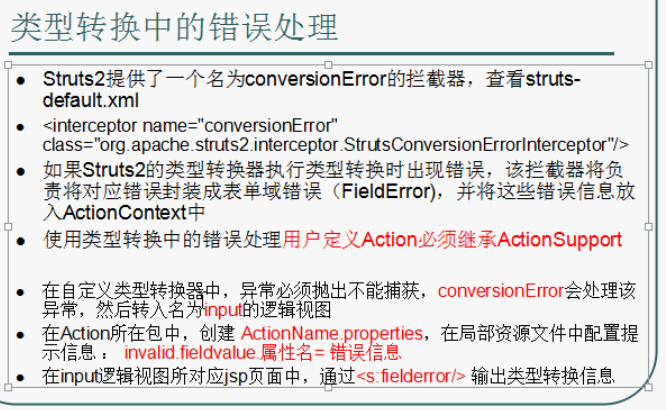
在构建路径顶端，建立固定名称为xwork-conversion.properties的配置文件:



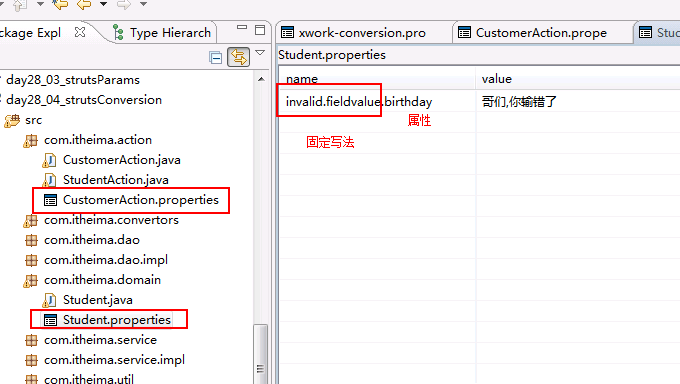
## 类型转换失败后事

a、转换失败，会自动转到一个name=input的逻辑视图，一般指向输入的那个页面，目的回显（建议使用struts2的表单标签）

b、错误消息提示中文版本



前提：动作类继承ActionSupport才能使用。



该功能是由一个叫做conversionError拦截器负责处理的。

# 输入验证

用户输入验证：开发中工作量极大，产出比较低。

客户端验证：用js。防君子不防高手。

服务器端验证：无论何时都必须写。

实际开发：客户端验证+服务器端验证

以下内容：属于服务器端验证

## 编程式验证

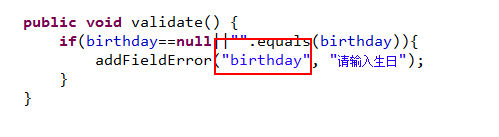
编写验证代码:

缺点：验证规则写到了代码中，硬编码。

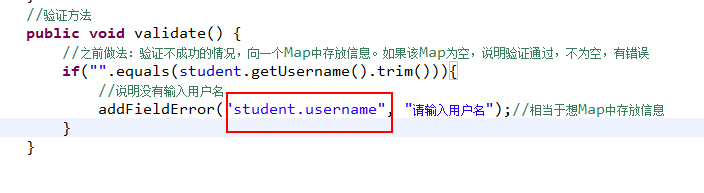
优点：验证时可以控制的更加精细。（用的少）

1. 针对动作类中的所有动作方法进行验证
2. 动作类需要实现ActionSupport, 覆盖掉public void validate(){}方法
3. 方法内部：编写你的验证规则，不正确的情况调用addFieldError添加错误信息

动作类就是模型：



动作类和模型分开：



不使用student.username这种方式添加错误信息s:fieldError标签中和普通标签中都有错误提示消息,使用了,将只会在s:fieldError中显示错误消息

1. 验证失败：

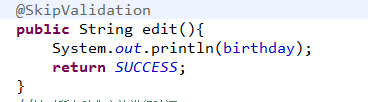
视图：会自动转向一个name=input的逻辑视图

错误消息提示：建议使用struts2标签库。如果没有显示请使用<s:fielderror>标签



1. 针对动作类中的指定动作方法进行验证

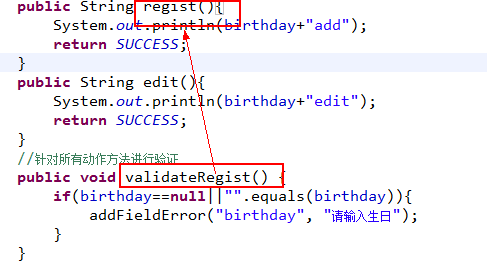
方式一：简单。使用一个注解。跳过验证



方式二：

如果一个动作方法名叫做regist

只针对该方法进行验证，请编写public void validateRegist(){}



## 声明式验证

优点：把验证规则写到了配置文件中。（用得多）

缺点：不是很精细。

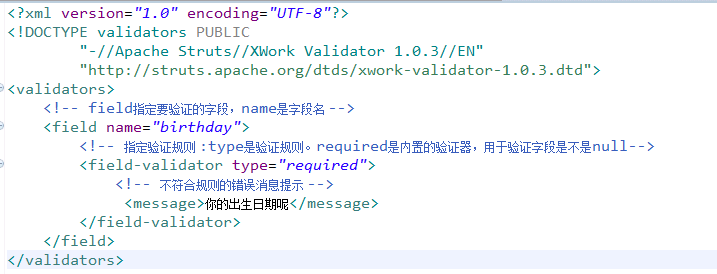
错误视图和消息提示和编程式一致。

1. 针对动作类中的所有动作方法进行验证

* 动作类就是模型：

在动作类所在的包中建立：动作类名-validation.xml配置文件。

文件内容：



<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE validators PUBLIC

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

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

<validators>

<!-- field指定要验证的字段，name是字段名 -->

<field name=*"birthday"*>

<!-- 指定验证规则 :type是验证规则。required是内置的验证器，用于验证字段是不是null-->

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

<!-- 不符合规则的错误消息提示 -->

<message>你的出生日期呢</message>

</field-validator>

</field>

</validators>

* + - 动作类和模型分开：

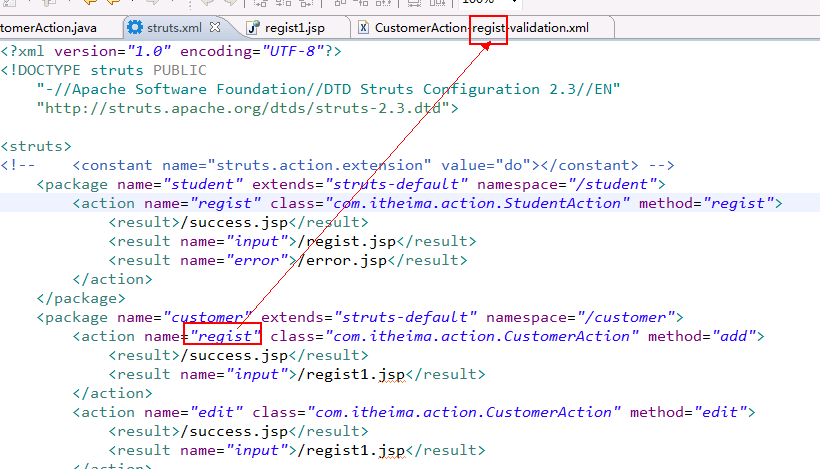
在动作类所在的包中建立：动作类名-validation.xml配置文件。

文件内容：

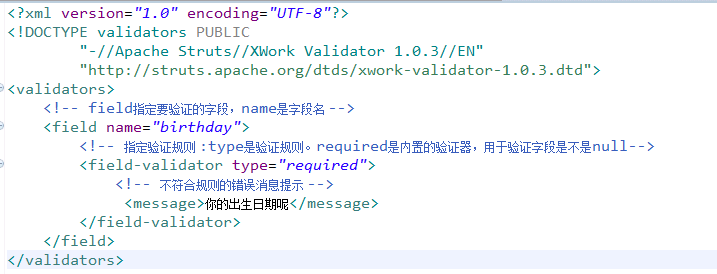


b、针对动作类中的指定动作方法进行验证

在动作类所在的包中建立：动作类名-动作名（是请求的struts.xml配置文件中的动作名）-validation.xml配置文件。



文件内容：



说明：验证功能是由validation拦截器来负责处理的。回显错误信息是由workflow拦截器来负责处理的。

## Struts2中提供的内置声明式验证器的使用

Struts2提供的声明式验证器在**xwork-core-\*\*.jar**包的

**com\opensymphony\xwork2\validator\validators\default.xml**配置文件中。

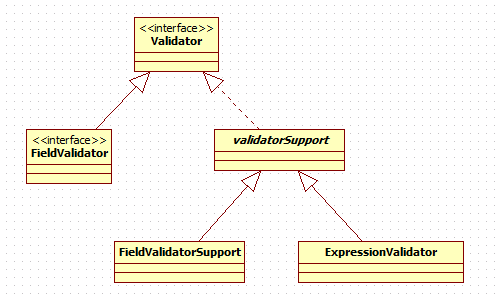


示例：



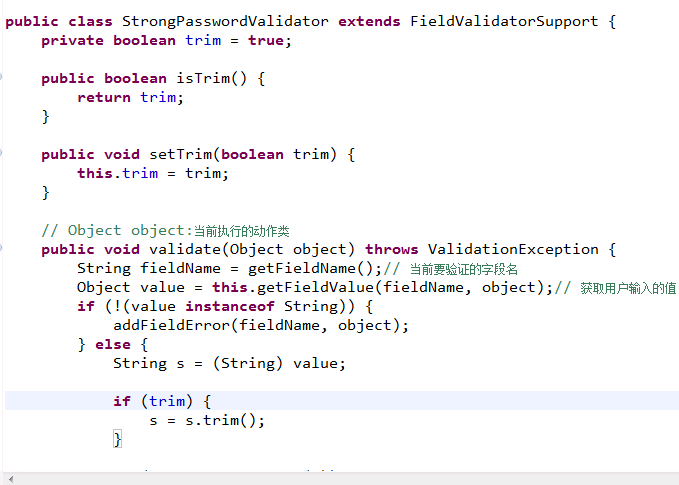


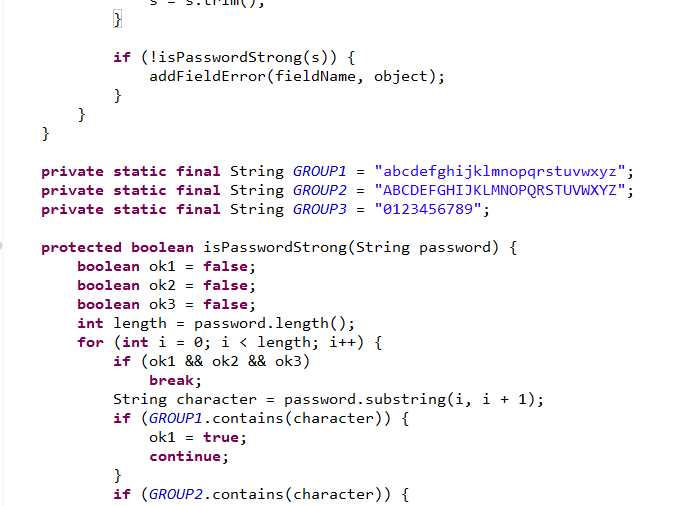
## 自定义声明式验证(了解怎么运作的)

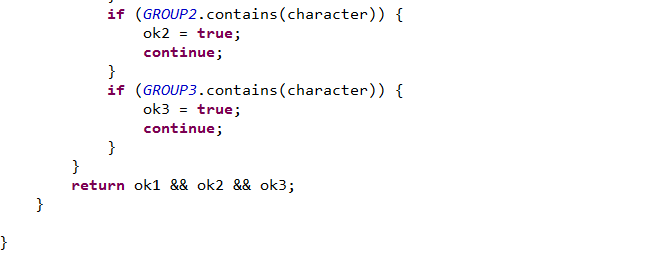


验证密码的强度：至少一个大写、小写、数字组成。

1. 编写一个类，继承FieldValidatorSupport

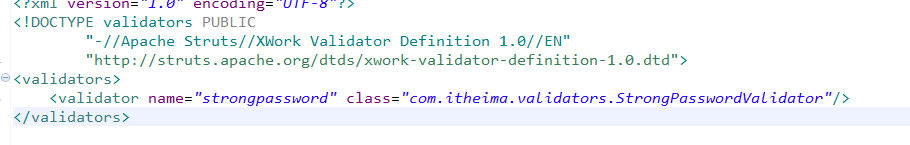






b、声明自定义的验证器

在构建路径顶端，建立一个固定名称为validators.xml的配置文件



<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE validators PUBLIC

"-//Apache Struts//XWork Validator Definition 1.0//EN"

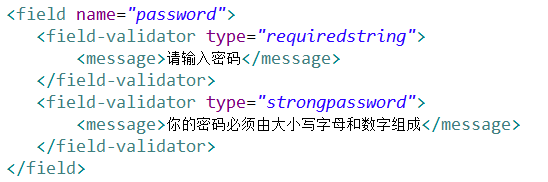
"http://struts.apache.org/dtds/xwork-validator-definition-1.0.dtd">

<validators>

<validator name=*"strongpassword"* class=*"com.itheima.validators.StrongPasswordValidator"*/>

</validators>

c、可以像使用struts2内置的声明式验证器一样来使用自己定义的验证器了

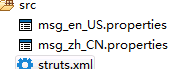


# Struts2的国际化：鸡肋

## 配置全局消息资源文件

1.1配置

资源文件位置：



文件内容：



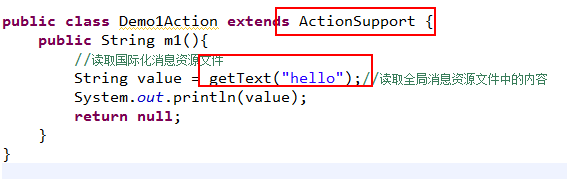
在struts.xml中配置全局消息资源文件



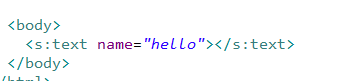
1.2使用

1. 在动作类中使用

前提：必须经过动作方法。动作类要继承ActionSupport



b、在页面上使用



## 配置局部消息资源文件

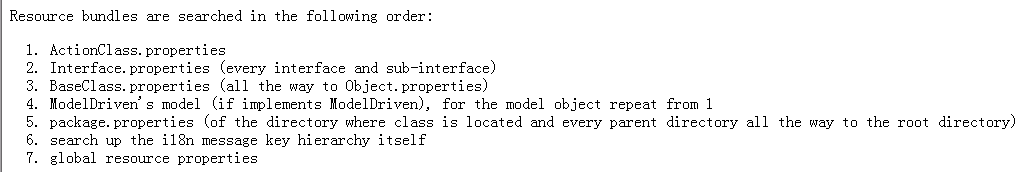
1. 消息资源文件命名规则：动作类名\_zh\_CN.properties
2. 访问：

a、在JSP上使用<s:text/>，直接访问的JSP，显示全局消息资源文件的内容。因为没有经过动作类。

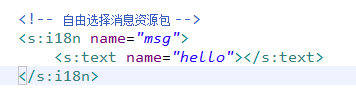
b、通过访问动作，转发到JSP，那么显示局部的。局部的优先级高于全局的

c、通过动作访问，转发到JSP，如果动作类没有继承ActionSupport，那么会显示全局的。

资源文件搜索顺序：



## 3、自由指定资源包



资源包在某个目录中：



# Struts2中的拦截器：（很重要）

常用的拦截器（面试题）：Struts2中功能核心。是一种AOP编程思想的具体应用。

modelDriven：模型驱动

servletConfig：获取ServletAPI

staticParams：静态参数注入

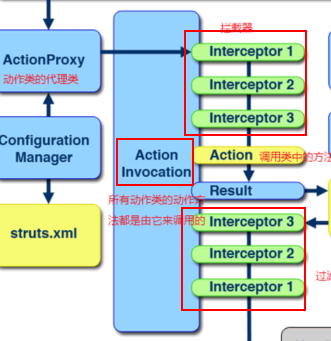
params：动态参数注入

validation：输入验证，声明式验证。

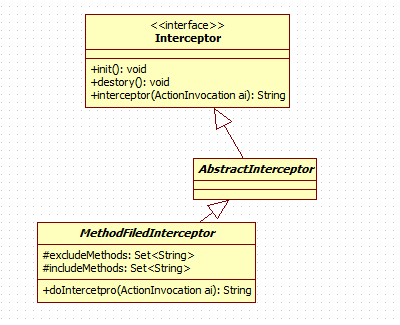
等等

拦截器执行过程图：

**拦截器会在动作方法执行之前按照正的顺序执行一遍，动作方法执行之后，结果视图显示之后，再按反的顺序执行一遍，在这之间有个ActionInvocation，它里面保存了当前动作的相关信息，比如参数，动作类名等**



类图：



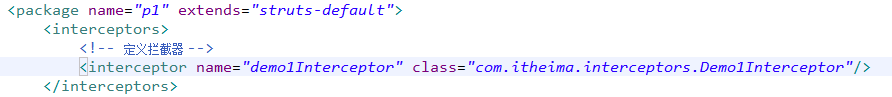
## 自定义拦截器（和过滤器功能差不多，保安）开发步骤

1. 编写一个类，继承AbstractInterceptor抽象类

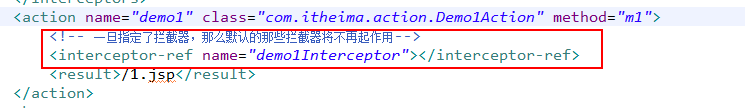


拦截器的返回值是最终目标的返回值。就是一个结果逻辑视图。对应struts.xml中的result

b、配置拦截器

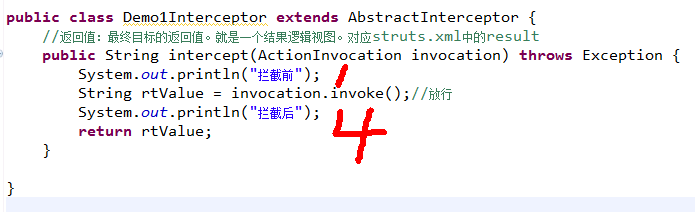


c、使用拦截器



注意：在action中一旦配置了拦截器，那么默认的那些拦截器将不再起作用

必须明确：执行顺序：先执行拦截器中放行前面的代码，再执行动作方法，再执行结果页面中的内容，最后执行拦截器中放行后面的代码

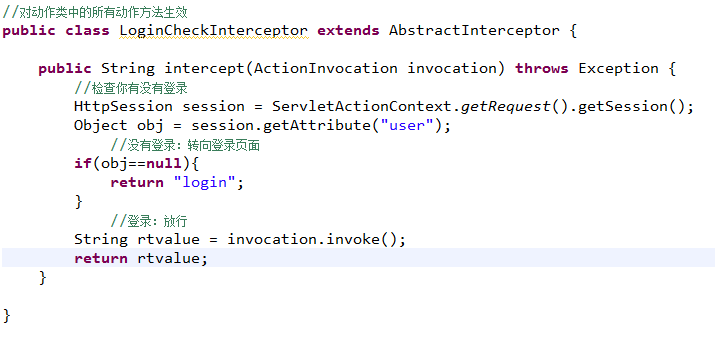


2是动作类中的动作方法；3是结果视图

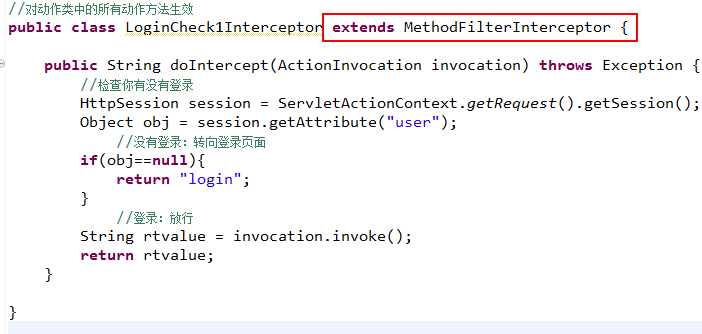
## 编写一个执行动作方法前验证用户是否登录的拦截器

（细粒度权限控制）

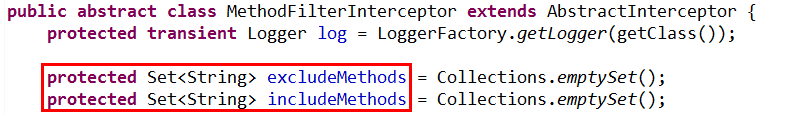
a、针对所有动作方法进行拦截的拦截器（继承AbstractInterceptor抽象类）



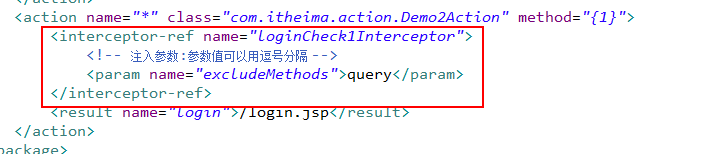
b、开发功能更加强大的拦截器（可以指定哪些方法执行，哪些方法不执行）（继承MethodFilterInterceptor抽象类）



MethodFilterInterceptor抽象类中的属性：包含哪些方法，不包含哪些方法



配置哪些方法执行，哪些方法不执行:



## 拦截器栈

定义：

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

<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"*/>

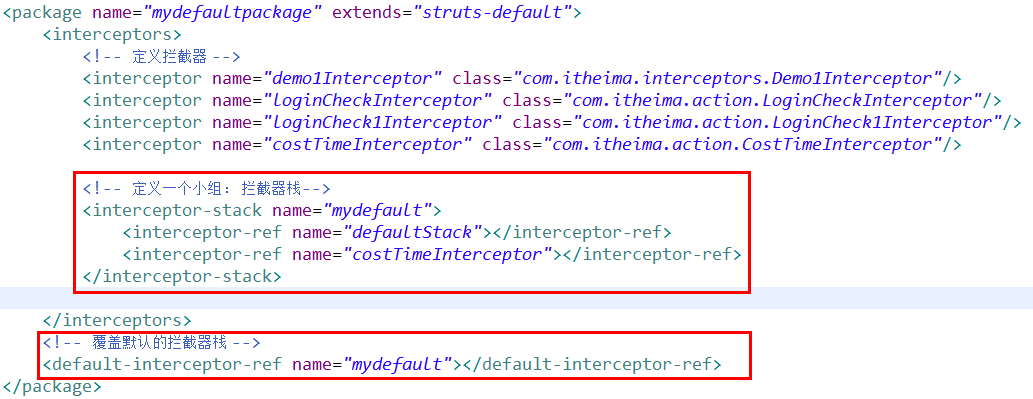
<interceptor-ref name=*"deprecation"*/>

</interceptor-stack>

在struts-default.xml文件中配置了默认的拦截器栈：

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

自定义拦截器栈并覆盖默认的拦截器栈：



# 文件的上传和下载

## 文件上传

必要前提：

表单的method必须是post

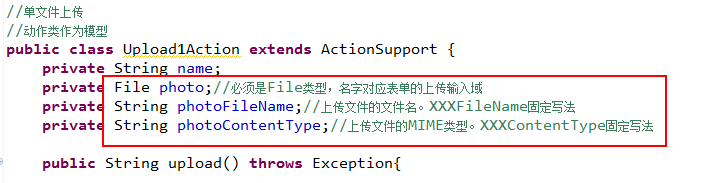
表单的enctype必须是multipart/form-data

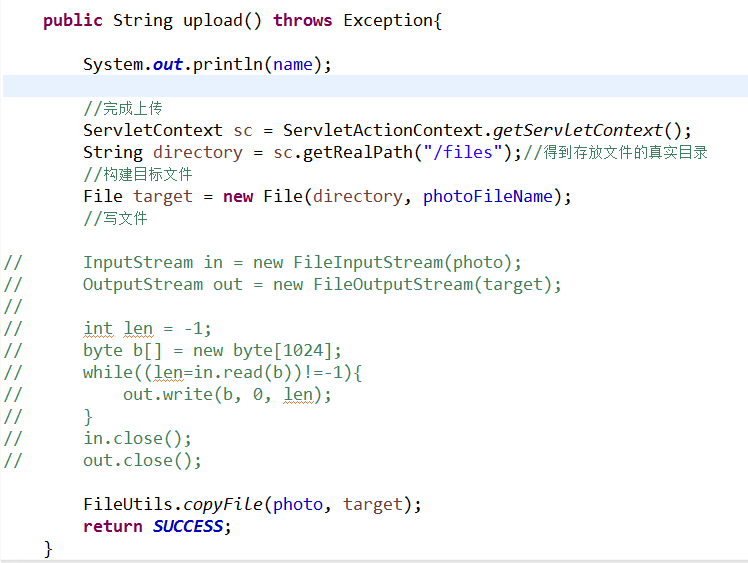
提供input type=”file”类型上传输入域

在Struts2中，文件上传是由一个叫做fileUpload的拦截器完成的。

### 单文件上传

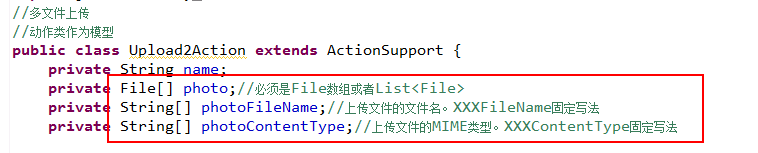
动作方法：





### 多文件上传

动作类：



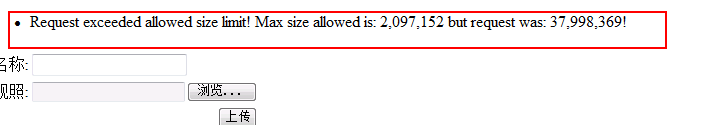
表单：



### 上传中出现的问题

* 上传失败时，会转向一个input的视图。
* 页面中显示错误提示信息的配置：





* 修改上传文件的大小限制：



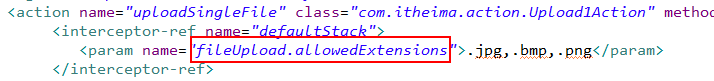
* 限制上传文件的类型：

fileUpload拦截器中有以下属性：

allowedTypesSet：允许上传的文件的MIME类型，多个用逗号分隔

allowedExtensionsSet：允许上传的文件的扩展名，多个用逗号分隔

给fileUpload拦截器设置参数：



* 修改错误消息提示：与国际化有关

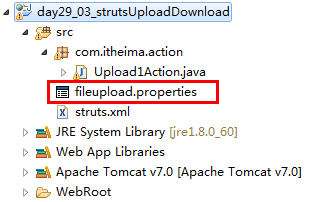
文件上传所有的默认消息提示在：

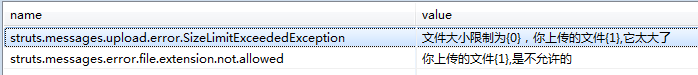
**struts2-core.jar org\apache\struts2\struts-messsages.properties**文件中

覆盖掉默认的消息提示：局部消息资源文件无效。

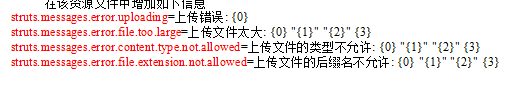
全局资源文件有效：







常用的错误提示信息：



## 文件下载：结果类型的使用

**文件下载是由名字为stream的结果类型处理文件下载。**

1. 动作类的书写遵守一定的规范



1. struts.xml配置文件

**

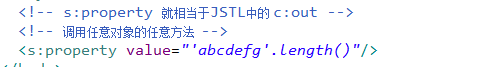
# OGNL表达式简介

前提：在struts2中要使用OGNL表达式，必须放到Struts2的标签中。

1、OGNL是Object Graphic Navigation Language（对象图导航语言）的缩写，它是一个开源项目。webwork用它作为表达式语言。

1. 其他重要的功能

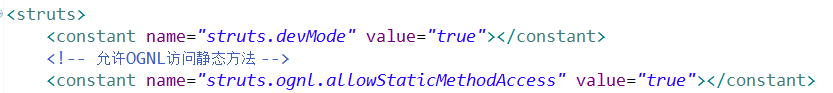
a、支持对象方法调用，如xxx.doSomeSpecial()：



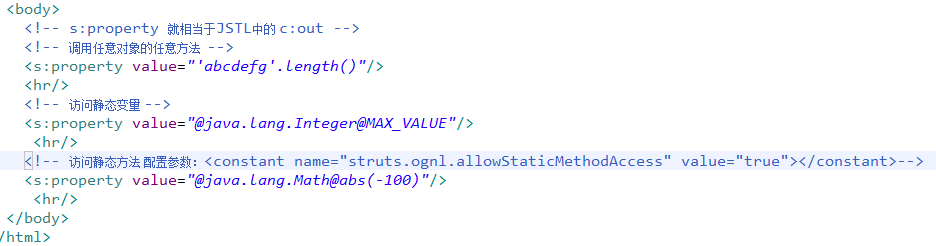
1. 支持类静态的方法调用和值访问

表达式的格式:@[类全名（包括包路径）]@[方法名 |  值名]，例如：@java.lang.String@format('foo %s', 'bar')或@tutorial.MyConstant@APP\_NAME；

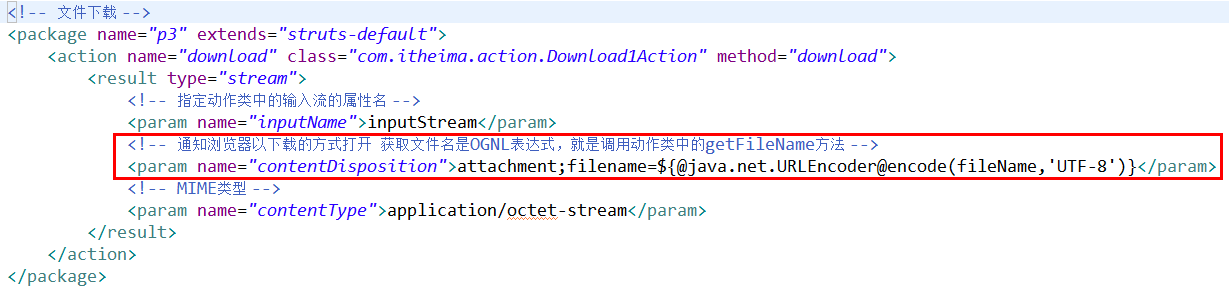
调用静态方法需要配置参数：



使用：



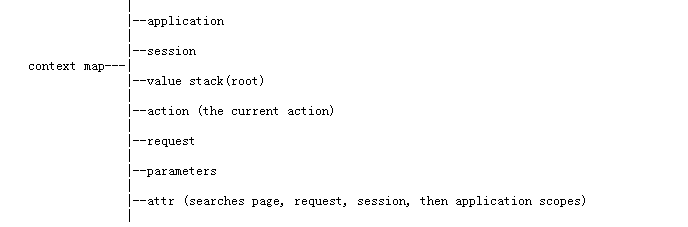
应用示例：文件下载示例



c、访问OGNL上下文（OGNL context）和ActionContext；

d、操作集合对象。

## context上下文：数据中心

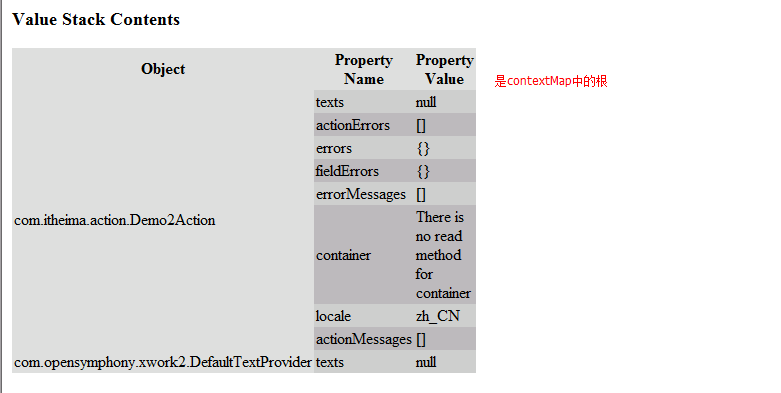


contextmap中的数据：

|  |  |  |
| --- | --- | --- |
| key | value | 备注 |
| application | ServeltContext中的所有属性attributes | 相当于EL的内置对象applicationScope |
| session | HttpSession中的所有属性attributes | 相当于EL的内置对象sessionScope |
| value stack(根） | 他是一个List |  |
| action | 动作类 |  |
| request | ServletRequest中的所有属性 | 相当于EL的内置对象requestScope |
| parameters | Map map= request.getParameterMap() | 相当于EL的内置对象paramValues |
| attr | 从四大域范围中搜索 | ${} |

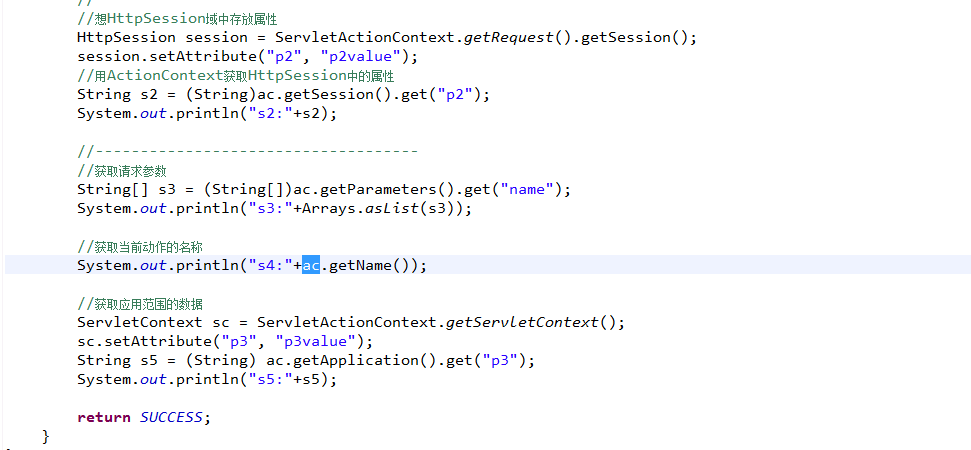


动作类的生命周期：每次访问都会重新创建新的动作类的实例。还会创建ActionContext和ValueStack的实例。ActionContext和ValueStack一直保持在你的线程中。（ThreadLocal）。

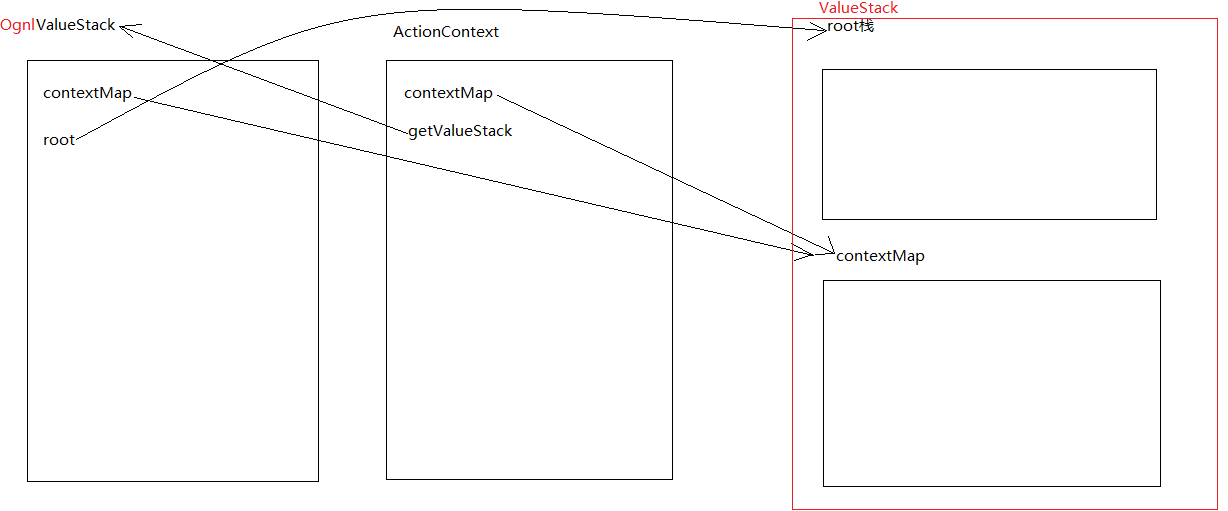


## ActionContext的API：操作contextMap中的数据的

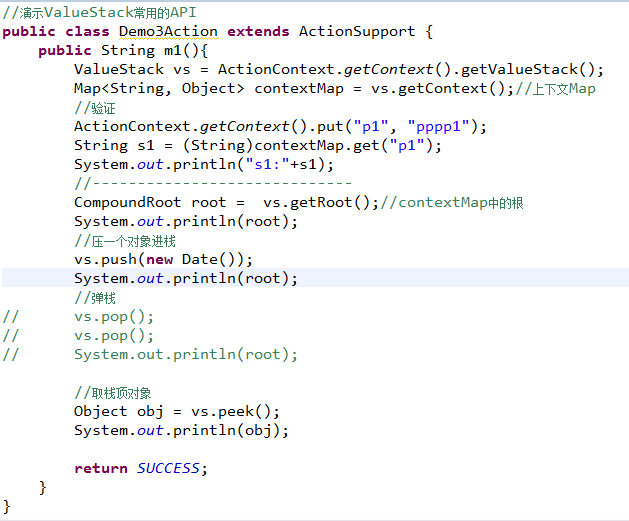




PS:ContextMap是xwork中的东西,struts2的核心就是xwork中的ognl,但是为了方便使用,所以就在ContextMap的基础上添加了root栈,ContextMap和root栈组成了值栈,值栈如下图所示:



## ValueStack的API：操作contextMap中根的数据

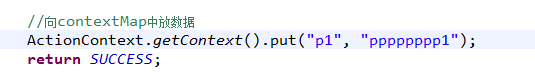


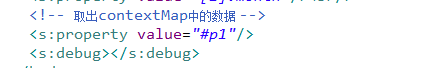
ValueStack的常用方法：



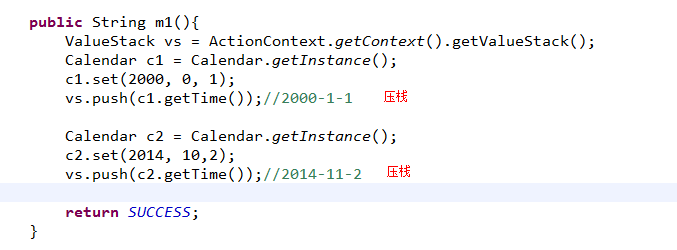
## 在JSP中使用ONGL显示contextMap/根中的数据

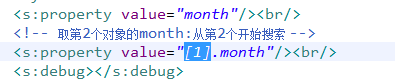
获取contextMap中的数据，OGNL表达式要用#开头





获取根（List）中的对象的属性，直接写属性的名称。会从栈顶的对象一直往下找。





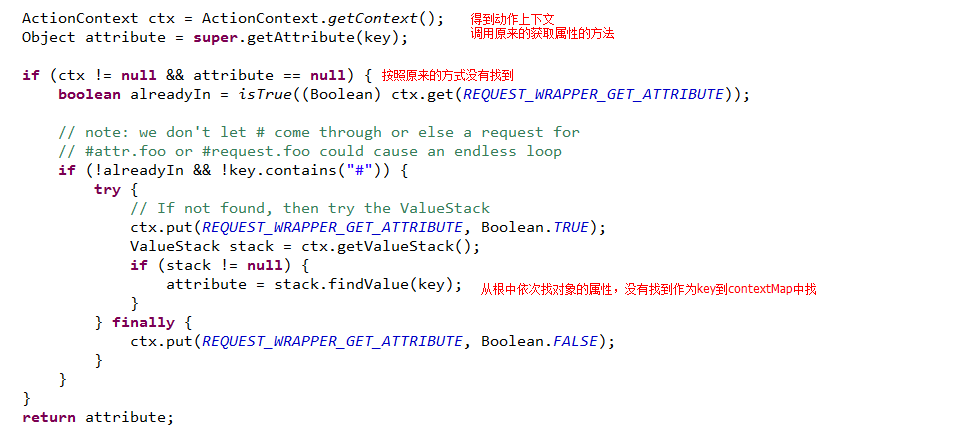
## OGNL的其他用法

### a、在JSP中取数据，可以使用OGNL表达式，也可以使用EL表达式

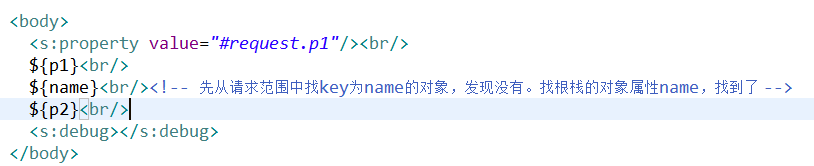
EL表达式在Struts2中，被做了小小的改动。

Struts2对原始的HttpServletRequest对象（服务器提供）进行了包装：org.apache.struts2.dispatcher.StrutsRequestWrapper

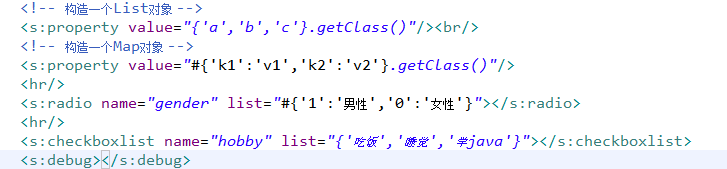
对request中的getAttribute方法重新修改了内容：先按照原来的方式找，没找到的话再调用ValueStack的findValue方法，按照OGNL表达式的方式来找:



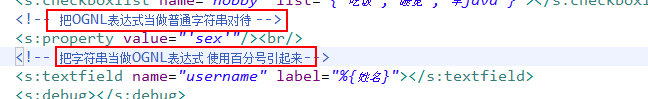




### b、在JSP页面中利用OGNL构造List对象和Map对象



### c、字符串和OGNL互换



### d、在配置文件中xml或properties也可以使用OGNL表达式，请使用${ognl}



# Struts2中的常用标签

## 1、常用的通用标签









## 2、常用的UI标签



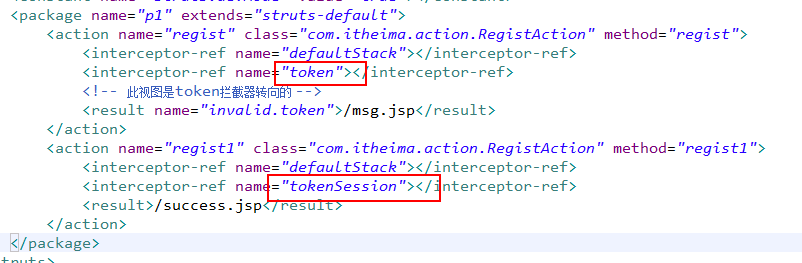
# Struts2中的UI主题

1. 提供的主题：simple（实际用）、xhtml（默认）、css\_xhtml、ajax
2. 修改struts2使用的主题：配置全局参数

<constant name=*"struts.ui.theme"* value=*"simple"*></constant>

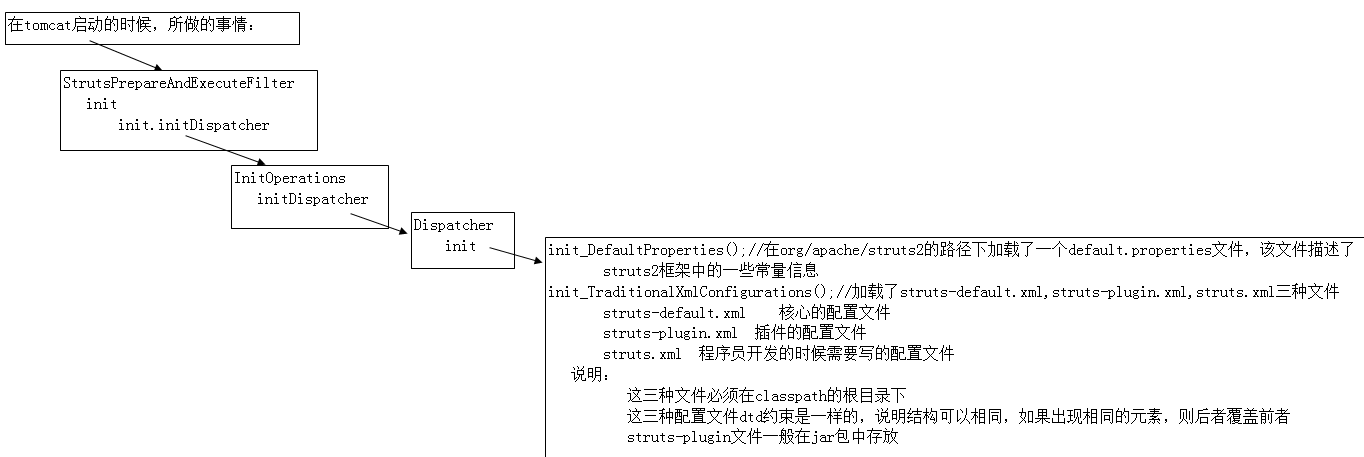
# 防止表单重复提交

1. 在表单中添加一个s:token
2. 向HttpSession中存放了一个令牌
3. 向表单中产生一个隐藏域，存了令牌
4. 执行动作方法前，有一个叫做token的拦截器负责处理



# 插件机制

## 启动的时候加载的配置文件：



## 对象工厂



ObjectFactory的说明：

1、该对象称为对象工厂：生成对象的工厂，struts2的很关键的组件都可以是一个对象（如Result，Action等）,也就是说对象工厂会生产struts2的关键组件

2、在default.properties文件中有一个常量struts.objectFactory，该值就是对象工厂

## 静态注入

静态：只有一次

注入：往struts2容器中注入东西

在struts2的配置文件中，只要标注了bean都为静态注入

<bean type="com.opensymphony.xwork2.util.ValueStackFactory" name="struts" class="com.opensymphony.xwork2.ognl.OgnlValueStackFactory" />

特点：1，当tomcat启动的时候就被加载了，所以加载一次

<bean type="com.opensymphony.xwork2.ObjectFactory" name="struts" class="org.apache.struts2.impl.StrutsObjectFactory" />

StrutsObjectFactory该bean一开始的时候就被加载了

改变action的产生方式：

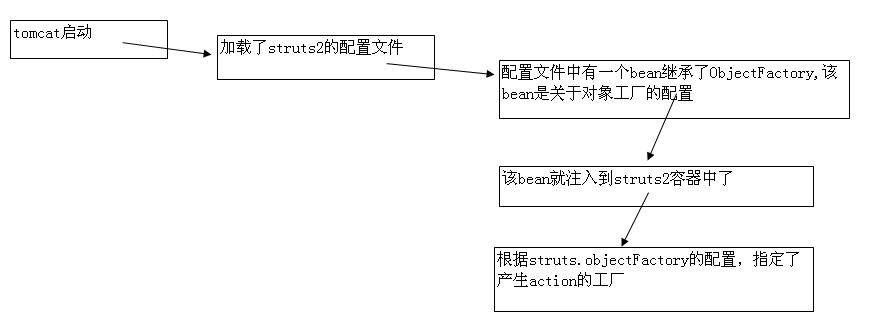
1、写一个类，该类继承了ObjectFactory, 重写buildAction方法

2、在struts2的配置文件中进行配置

<bean type="com.opensymphony.xwork2.ObjectFactory" name="itheima12" class="com.itheima12.struts.action.CreateAction" />

<constant name="struts.objectFactory" value="itheima12"></constant>

## 运行流程



## struts2与spring的整合：

在struts2-spring-plugin-2.3.1.2.jar包中有一个struts-plugin.xml文件

<bean type="com.opensymphony.xwork2.ObjectFactory" name="spring" class="org.apache.struts2.spring.StrutsSpringObjectFactory" />

<constant name="struts.objectFactory" value="spring" />

所以当请求一个action时，创建action对象的方式就改成StrutsSpringObjectFactory的buildAction方法

## 插件机制的解释

**插件机制的解释：**

在一个工程的lib下，肯定有很多个struts2-plugin-xxx.jar包，在这些jar包下有一个文件：struts-plugin.xml文件，在该文件下面可以配置很多内容

1、静态注入 2、配置结果集 3、配置拦截器

只需要把这个jar包放到lib下，当tomcat启动的时候就被加载了，如果不需要这些功能，只需要把这些jar包移除掉就可以了。

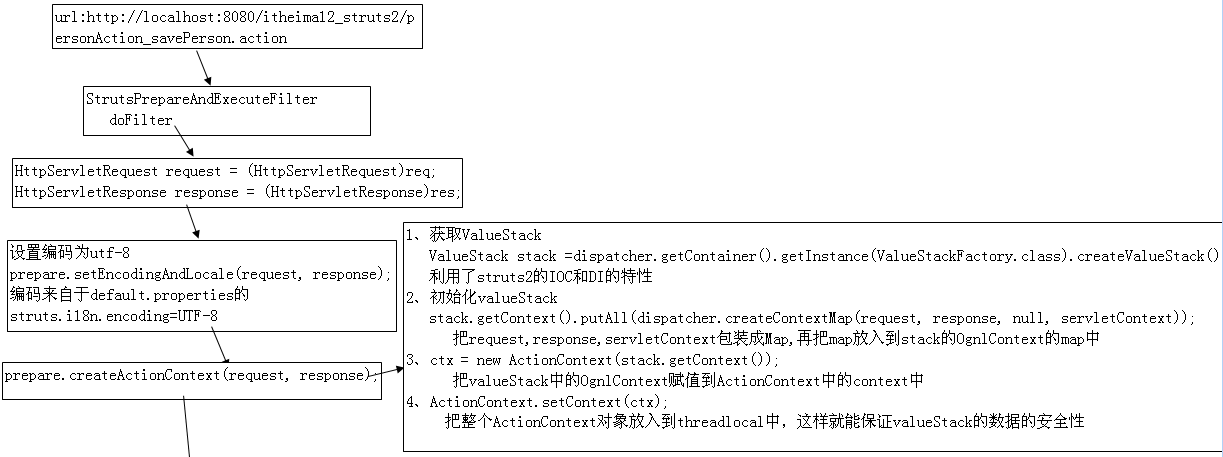
**深度剖析：**

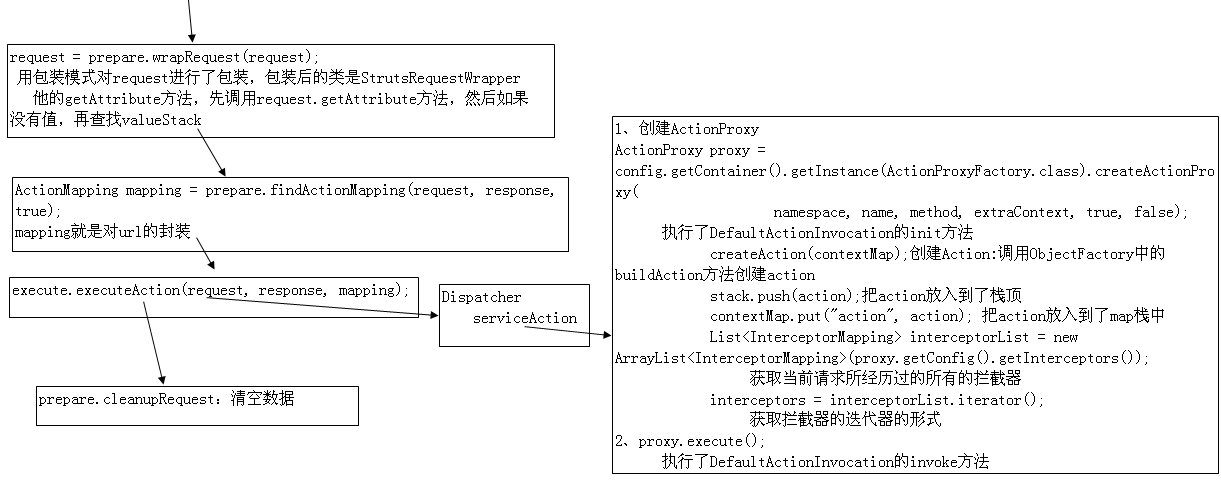
1、在struts2的最底层是用objectFactory.buildAction方法来创建action的

2、只要是继承了ObjectFactory的类都可以看做是objectFactory

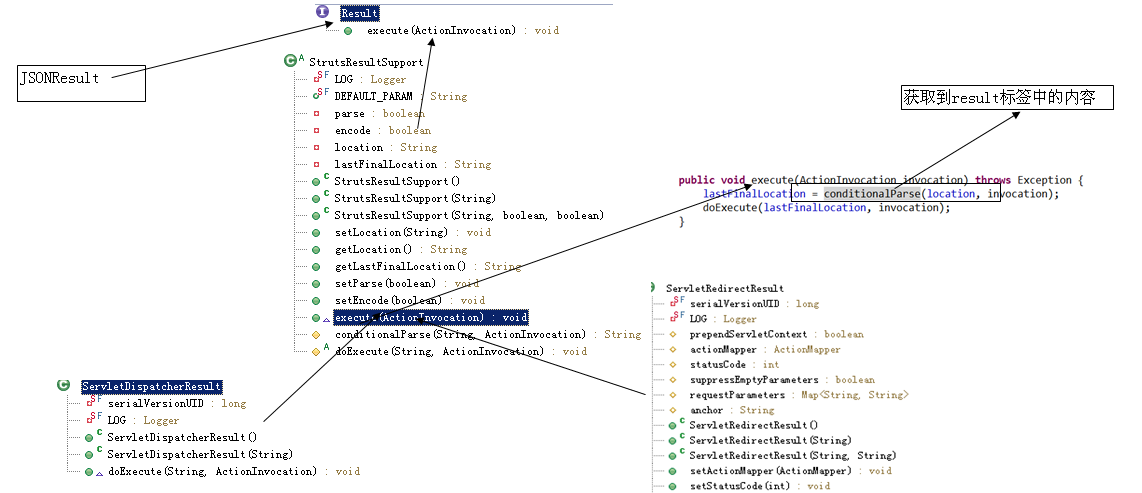
3、struts2利用了静态注入和常量的方式接收了程序员自己写的ObjectFactory

# Struts2的核心流程





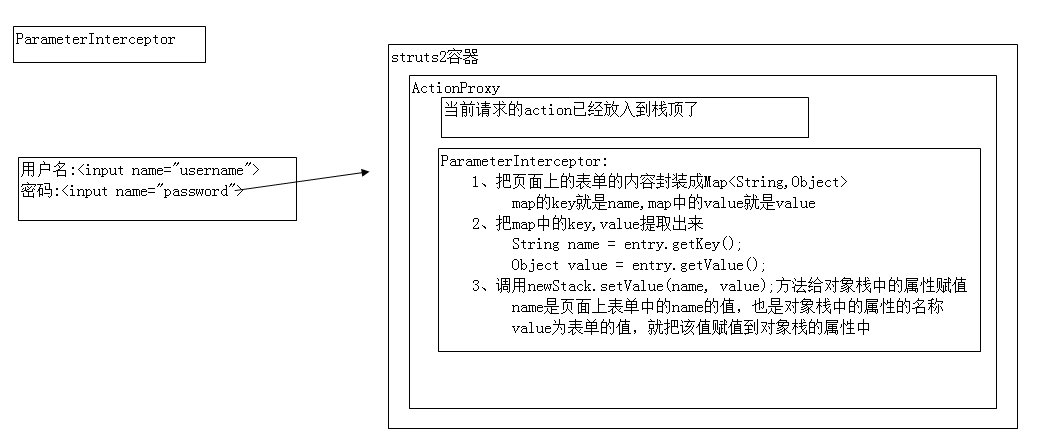
# Struts2的结果集



Result是接口，所有结果集都要实现Result接口，StrutsResultSupport是抽象类，它实现了Result接口中的execute方法，方法中的lastFinalLocation = conditionalParse(location, invocation)是获取result标签中的内容，结果集从表现形式上可以分为刷新和不刷新，struts2中的json结果集的插件是不刷新的，因此不需要获取result标签中的内容，所以JSONResult类是直接实现Result接口的，结果集是直接实现Result接口还是继承StrutsResultSupport抽象类就看这个结果集需不需要获取result标签中的内容

# struts2几个关键的拦截器

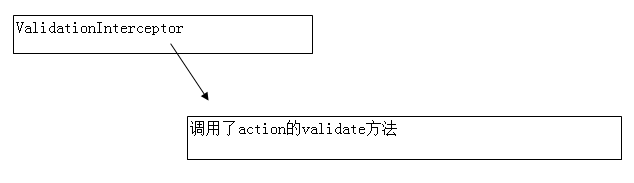
## ParameterInterceptor



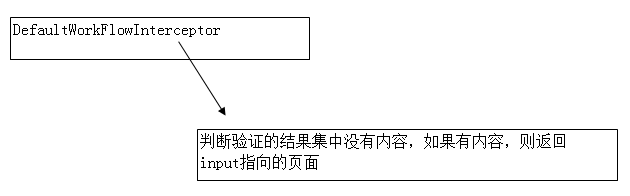
## ModelDrivenInterceptor



## ValidationInterceptor

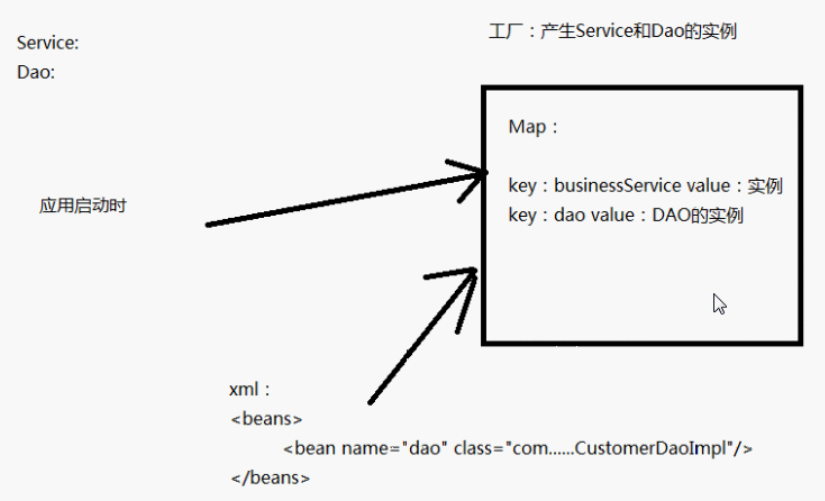


## DefaultWorkflowInterceptor



# Struts2和Spring进行整合

## Spring简介



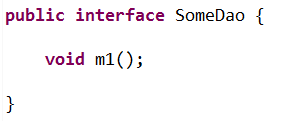
1. 搭建Spring（核心）的开发环境

spring.jar commons-logging.jar

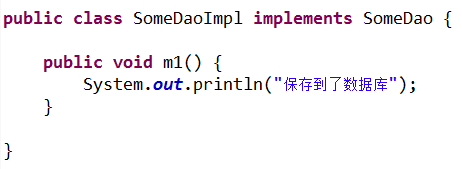
1. Spring是一个对象容器

在应用的构建路径中建立一个application.xml（随意）配置文件

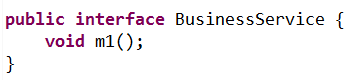
Dao:



DaoImpl:



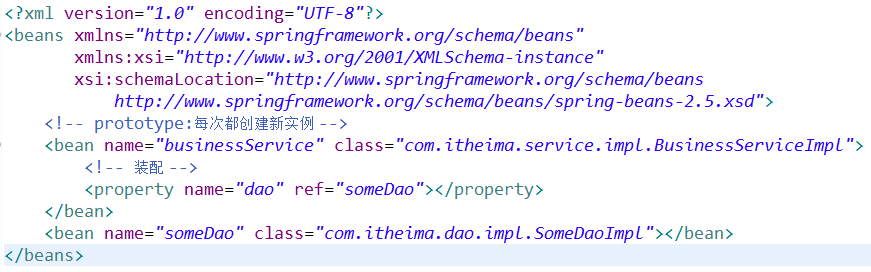
Service:



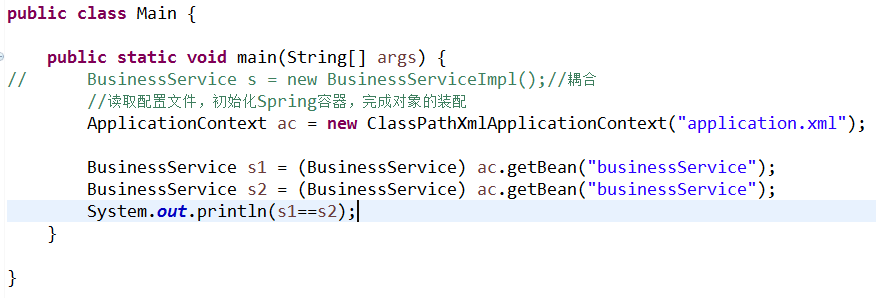
ServiceImpl:



application.xml:



控制层测试：

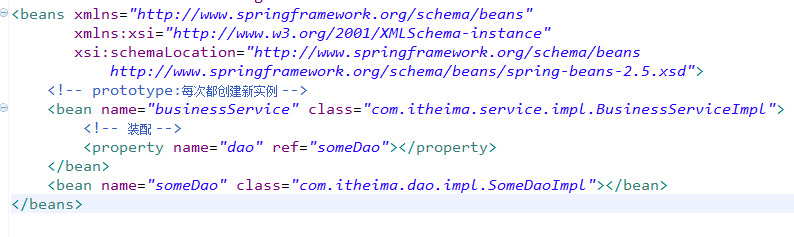


## JavaWeb中集成Spring

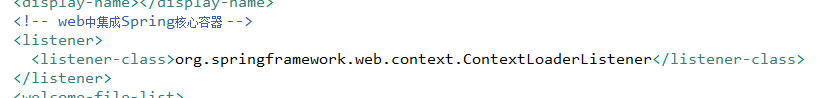
1、搭建Spring（核心）的开发环境

spring.jar commons-logging.jar

1. 在WEB-INF目录下建立固定名称applicationContext.xml配置文件



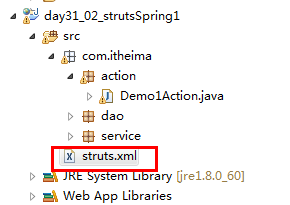
1. 在web.xml中注册监听器:



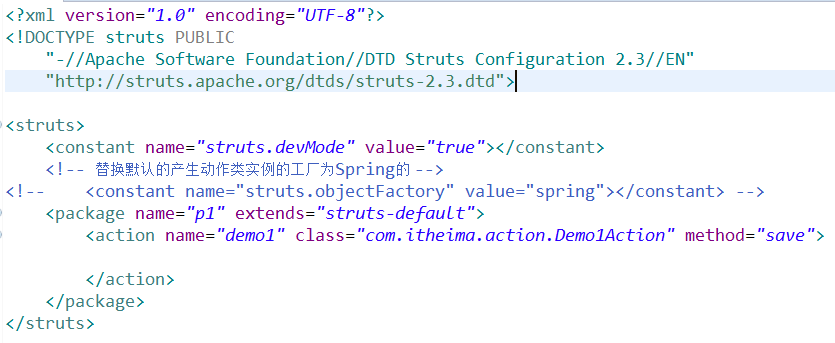
## Struts2整合Spring

### 整合方式一：动作类还是Struts2负责管理，只是向Spring容器要service的实例

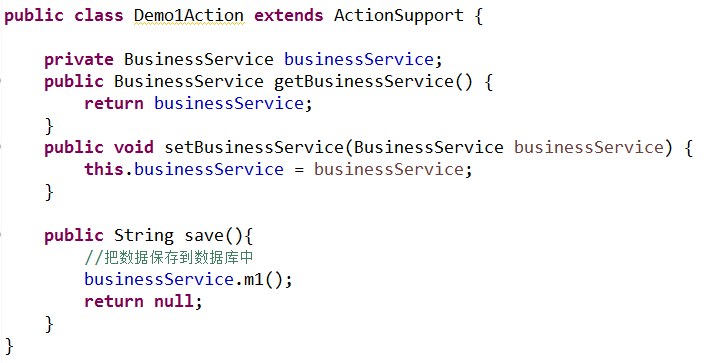
1. 搭建Spring的web环境
2. 搭建Struts2的开发环境



struts.xml:



Demo1Action:



web.xml:



1. 拷贝Struts2和Spring的插件过来。

struts2-spring-plugin-2.3.15.3.jar

通过配置文件替换掉ObjectFactory：不用做（在struts2-spring-plugin-2.3.15.3.jar中已经替换了）

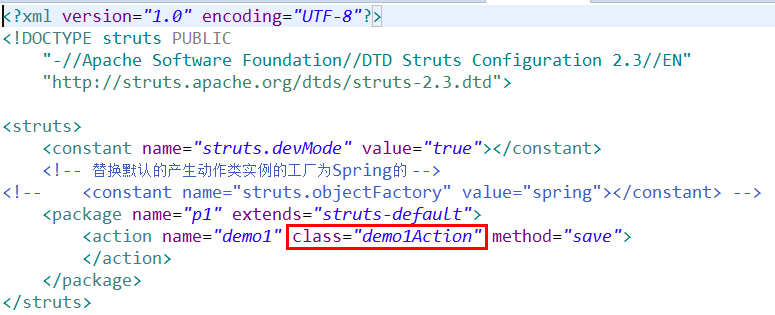


### 整合方式二：动作类也交给Spring管理

applicationContext.xml:



struts.xml: **action的类名不能为全名，应该是在spring容器中的名称**



# 案例：

## struts.xml:

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"

"http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>

<constant name=*"struts.devMode"* value=*"true"*></constant>

<constant name=*"struts.ui.theme"* value=*"simple"*></constant>

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

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

<!-- 用户登录 -->

<action name=*"login"* class=*"com.itheima.action.UserAction"* method=*"login"*>

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

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

</action>

<!-- 显示所有的用户 -->

<action name=*"list"* class=*"com.itheima.action.UserAction"* method=*"list"*>

<result>/user/list.jsp</result>

</action>

<!-- 条件查询 -->

<action name=*"search"* class=*"com.itheima.action.UserAction"* method=*"search"*>

<result>/user/list.jsp</result>

</action>

<!-- 添加新用户 -->

<action name=*"add"* class=*"com.itheima.action.UserAction"* method=*"add"*>

<!-- 添加成功后，返回用户列表 -->

<result name=*"success"* type=*"chain"*>list</result>

<result name=*"input"*>/user/add.jsp</result>

</action>

<!-- 删除用户 -->

<action name=*"del"* class=*"com.itheima.action.UserAction"* method=*"del"*>

<!-- 添加成功后，返回用户列表 -->

<result name=*"success"* type=*"chain"*>list</result>

</action>

<!-- 查询一个用户 -->

<action name=*"view"* class=*"com.itheima.action.UserAction"* method=*"view"*>

<result>/user/view.jsp</result>

</action>

<!-- 文件下载 -->

<action name=*"download"* class=*"com.itheima.action.UserAction"* method=*"download"*>

<result type=*"stream"*>

<param name=*"inputName"*>inputStream</param>

<param name=*"contentDisposition"*>attachment;filename=${@java.net.URLEncoder@encode(user.filename.substring(user.filename.indexOf('\_')+1),"UTF-8")}</param>

</result>

</action>

<!-- 显示修改页面 -->

<action name=*"editUI"* class=*"com.itheima.action.UserAction"* method=*"editUI"*>

<result>/user/edit.jsp</result>

</action>

<!-- 保存修改信息 -->

<action name=*"edit"* class=*"com.itheima.action.UserAction"* method=*"edit"*>

<!-- 添加成功后，返回用户列表 -->

<result name=*"success"* type=*"chain"*>list</result>

<result name=*"input"*>/user/edit.jsp</result>

</action>

</package>

</struts>

## UserAction:

**public** **class** UserAction **extends** ActionSupport **implements** ModelDriven<User> {

**private** User user = **new** User();//模型

**private** List<User> users = **new** ArrayList<User>();

**private** String isUpload;//是否上传简历

**private** String[] interests;//用户选择的爱好

**private** File upload;//文件

**private** String uploadFileName;//文件名

**private** InputStream inputStream;//下载的输入流

**private** BusinessService businessService;

//-------------------------------------------------

**public** String login(){

User logonUser = businessService.login(user.getLogonName(), user.getLogonPwd());

**if**(logonUser==**null**){

addActionError("错误的用户名或密码");//添加一个动作错误。放到一个Map中

**return** ***LOGIN***;

}

//必须把logonUser放到HttpSession中

ServletActionContext.*getRequest*().getSession().setAttribute("user", logonUser);

**return** ***SUCCESS***;

}

**public** String list(){

users = businessService.findAllUsers();

**return** ***SUCCESS***;

}

**public** String search(){

users = businessService.findUsersByCondition(user.getUserName(), user.getSex(), user.getEducation(), isUpload);

**return** ***SUCCESS***;

}

**public** String add() **throws** IOException{

//处理爱好

**if**(interests!=**null**||interests.length>0){

StringBuffer sb = **new** StringBuffer();

**for**(**int** i=0;i<interests.length;i++){

**if**(i>0){

sb.append(",");

}

sb.append(interests[i]);

}

user.setInterest(sb.toString());

}

//存放目录

String storeDirectory = ServletActionContext.*getServletContext*().getRealPath("/files");

String uuidFileName = UUID.*randomUUID*().toString()+"\_"+uploadFileName;

//给模型设置：path filename

user.setFilename(uuidFileName);

//计算子目录

String path = Genertor.*genClildDir*(storeDirectory, uuidFileName);

user.setPath(path);

//文件上传

FileUtils.*copyFile*(upload, **new** File(storeDirectory+File.***separator***+path,uuidFileName));

//保存模型

businessService.addUser(user);

**return** ***SUCCESS***;

}

**public** String del(){

businessService.delUser(user);

//**TODO** 最好把他的简历找到删除

**return** ***SUCCESS***;

}

**public** String view(){

user = businessService.findUserById(user.getUserID());

**return** ***SUCCESS***;

}

**public** String download() **throws** FileNotFoundException{

user = businessService.findUserById(user.getUserID());

String storeDirectory = ServletActionContext.*getServletContext*().getRealPath("/files");

inputStream = **new** FileInputStream(storeDirectory+File.***separator***+user.getPath()+File.***separator***+user.getFilename());

**return** ***SUCCESS***;

}

**public** String editUI(){

user = businessService.findUserById(user.getUserID());

ValueStack vs = ActionContext.*getContext*().getValueStack();

vs.push(user);

**return** ***SUCCESS***;

}

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

User dbUser = businessService.findUserById(user.getUserID());//数据库中的原来的数据

//处理爱好

**if**(interests!=**null**||interests.length>0){

StringBuffer sb = **new** StringBuffer();

**for**(**int** i=0;i<interests.length;i++){

**if**(i>0){

sb.append(",");

}

sb.append(interests[i]);

}

user.setInterest(sb.toString());

}

//判断有没有上传新的文件

**if**("".equals(uploadFileName)){

//没有重新上传

user.setPath(dbUser.getPath());

user.setFilename(dbUser.getFilename());

}**else**{

//存放目录

String storeDirectory = ServletActionContext.*getServletContext*().getRealPath("/files");

String uuidFileName = UUID.*randomUUID*().toString()+"\_"+uploadFileName;

//给模型设置：path filename

user.setFilename(uuidFileName);

//计算子目录

String path = Genertor.*genClildDir*(storeDirectory, uuidFileName);

user.setPath(path);

//文件上传

FileUtils.*copyFile*(upload, **new** File(storeDirectory+File.***separator***+path,uuidFileName));

}

//保存模型

businessService.editUser(user);

**return** ***SUCCESS***;

}

**public** User getModel() {

**return** user;

}

//--------------------------------------------------

**public** BusinessService getBusinessService() {

**return** businessService;

}

**public** **void** setBusinessService(BusinessService businessService) {

**this**.businessService = businessService;

}

**public** User getUser() {

**return** user;

}

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

**this**.user = user;

}

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

**return** users;

}

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

**this**.users = users;

}

**public** String getIsUpload() {

**return** isUpload;

}

**public** **void** setIsUpload(String isUpload) {

**this**.isUpload = isUpload;

}

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

**return** interests;

}

**public** **void** setInterests(String[] interests) {

**this**.interests = interests;

}

**public** File getUpload() {

**return** upload;

}

**public** **void** setUpload(File upload) {

**this**.upload = upload;

}

**public** String getUploadFileName() {

**return** uploadFileName;

}

**public** **void** setUploadFileName(String uploadFileName) {

**this**.uploadFileName = uploadFileName;

}

**public** InputStream getInputStream() {

**return** inputStream;

}

**public** **void** setInputStream(InputStream inputStream) {

**this**.inputStream = inputStream;

}

}

## Login.jsp:





## list.jsp:













## view.jsp:



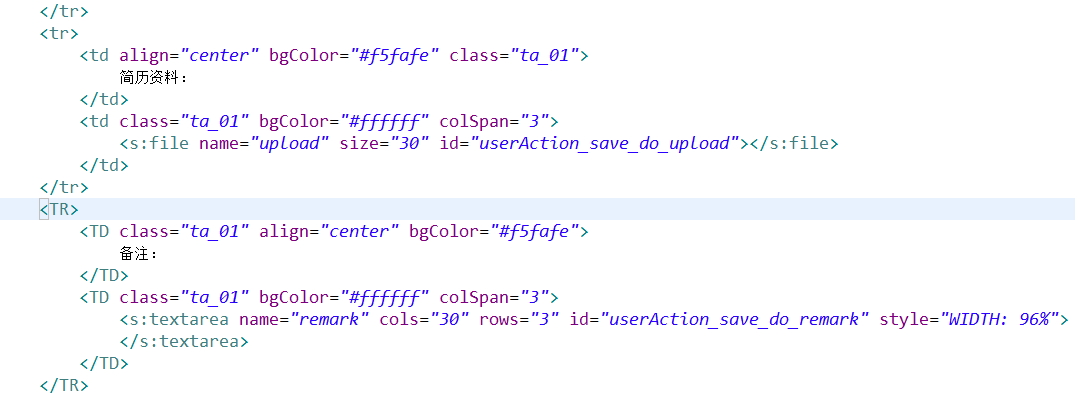




## add.jsp







## edit.jsp





