**Struts2第3天课程**

**【学习目标】**

## 1，能够理解请求参数合法性校验机制

1. 独立编写代码实现数据校验
2. 独立配置XML文件实现数据校验
3. 了解自定义校验规则

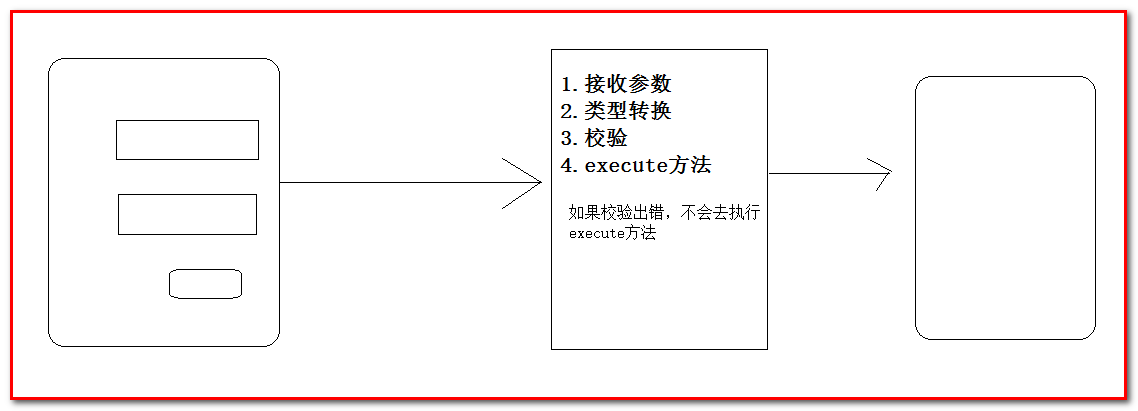
## 2.能够理解拦截器机制

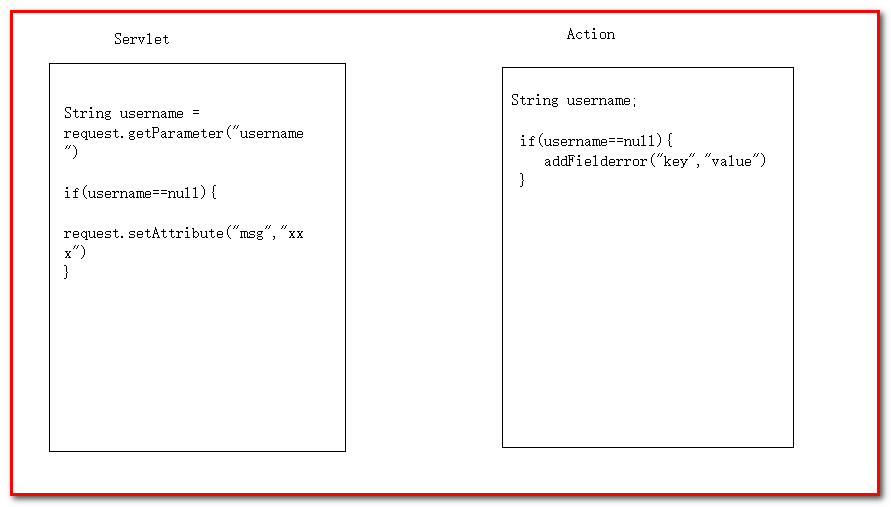
1. 说出拦截器的概念
2. 分析Struts2运行原理
3. 独立使用拦截器接口
4. 了解struts2内置拦截器
5. 独立编写代码实现自定义拦截器

## 3.能够理解struts2的上传功能

1. 独立编写代码实现文件上传

# 请求参数的合法性校验机制





## 表单校验的方式

请求参数的输入校验途径一般分两种：

* 客户端校验 ：通过JavaScript 完成 （jquery validation插件），目的：过滤正常用户的误操作。
* 服务器校验 ：通过java代码完成 ，目的：整个应用阻止非法数据的重要防线/最后防线

两者校验的特点：客户端校验，更加友好， 用户可以第一时间告知数据输入有错误(缺点：安全性差)，

服务器校验 更加安全校验机制 （考虑系统健壮性 ，服务器校验建议要做）

我们这里研究的是：在请求数据封装之后， 在struts2 实现请求数据合法性校验。

Struts2支持校验方式

* 代码中手动数据校验(了解)特点：高耦合、硬编码
* Xml配置规则进行校验
* 注解的方式校验(不研究)

## 代码中手动校验（了解）

需求：校验用户名和密码不能为空

这种方式是在Action中通过编写代码进行数据校验，由于需要编写代码，适用于小型项目或者表单数据不多的情况，有代码耦合的问题。

先搭建测试环境：

【第一步】创建struts2\_d03\_c01项目，在WebContent下面创建a\_validation文件夹，然后编写页面 login.jsp

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@taglib uri=*"/struts-tags"* prefix=*"s"* %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <title>My JSP 'login.jsp' starting page</title>  </head>  <body>  <h3>登录：请求数据校验--代码手动校验</h3>  <s:fielderror/>  <form action=*"${pageContext.request.contextPath }/user\_login.action"* method=*"post"*>  用户名：<input type=*"text"* name=*"username"*/><br/>  密 码：<input type=*"password"* name=*"pwd"*/><br/>  <input type=*"submit"* value=*"登录"*/>  </form>  </body>  </html> |

【第二步】：新建com.igeek\_01\_validation;包，编写LoginAction类

其中ActionSupport类已经实现了Validateable接口，所以可以使用validate方法进行校验。

|  |
| --- |
| **package** com.igeek\_01\_validation;  **import** org.apache.commons.lang3.StringUtils;  **import** com.opensymphony.xwork2.ActionSupport;  **public** **class** LoginAction **extends** ActionSupport{    **private** String username;  **private** String pwd;        @Override  **public** String login() **throws** Exception {    System.*out*.println("LoginAction执行了....");    System.*out*.println(toString());    **return** NOEN;  }  //需求：对username和pwd进行验证  //需要继承ActionSupport才有验证功能  //如果验证不通过，那么struts会把校验信息显示在login.jsp页面的<s:fielderror>标签中  //为什么验证不通过就会跳转到login.jsp中呢？  //答：需要指定验证不通过时跳转到login.jsp,strust中规定，验证不通过，跳转的是INPUT视图，  //需要在struts.xml的result结果中配置name属性为input  //重写validate是全局验证，验证这个Action中的所有方法  @Override  **public** **void** validate() {    //用户名不能为空，  // if(username==null||"".equals(username))  **if**(StringUtils.*isBlank*(username))  {  //添加错误信息  **this**.addFieldError("usernameError", "用户名不能为空");  }  //判断面不能为空  **if**(StringUtils.*isBlank*(pwd))  {  **this**.addFieldError("pwdError", "密码不能为空");  }    }  **public** String getUsername() {  **return** username;  }  **public** **void** setUsername(String username) {  **this**.username = username;  }  **public** String getPwd() {  **return** pwd;  }  **public** **void** setPwd(String pwd) {  **this**.pwd = pwd;  }  @Override  **public** String toString() {  **return** "LoginAction [username=" + username + ", pwd=" + pwd + "]";  }    } |

【第三步】配置struts.xml

|  |
| --- |
| <!-- 配置手动验证表单的Action -->  <action name=*"user\_\*"* class=*"com.igeek\_01\_validation.LoginAction"*  method=*"{1}"*>  <result name=*"input"*>/a\_validation/login.jsp</result>  </action> |

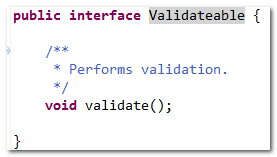
【第四步】测试环境，ok！

【用法原理分析】

如果要执行校验 ，Action必须继承ActionSupport 类 （因为它实现 Validateable接口）



Validateable接口中存在validate()方法



有两种校验范围：

* 全局校验 （对当前Action的所有方法进行校验 ）

在Action 提供 validate() 方法 --- 对Action中所有的方法都进行校验

* 局部校验 （校验Action中指定业务方法—校验一个方法 ）

在Action 添加 validateXxx 方法（约定）， 这里XXX 是要校验目标方法名 （只会对指定方法校验）

以下方法表示对login()方法进行校验。

|  |
| --- |
| **package** com.igeek\_01\_validation;  **import** org.apache.commons.lang3.StringUtils;  **import** com.opensymphony.xwork2.ActionSupport;  **public** **class** LoginAction **extends** ActionSupport{    **private** String username;  **private** String pwd;        @Override  **public** String execute() **throws** Exception {    System.*out*.println("LoginAction执行了execute....");    System.*out*.println(toString());    **return** NOEN;  }      **public** String login() **throws** Exception {    System.*out*.println("LoginAction执行了login....");    System.*out*.println(toString());    **return** **super**.execute();  }    **public** String find() **throws** Exception {    System.*out*.println("LoginAction执行了find....");    System.*out*.println(toString());    **return** NOEN;  }    //需求：对username和pwd进行验证  //需要继承ActionSupport才有验证功能  //如果验证不通过，那么struts会把校验信息显示在login.jsp页面的<s:fielderror>标签中  //为什么验证不通过就会跳转到login.jsp中呢？  //答：需要指定验证不通过时跳转到login.jsp,strust中规定，验证不通过，跳转的是INPUT视图，  //需要在struts.xml的result结果中配置name属性为input  //重写validate是全局验证，验证这个Action中的所有方法  // @Override  // public void validate() {  //  // //用户名不能为空，  //// if(username==null||"".equals(username))  // if(StringUtils.isBlank(username))  // {  // //添加错误信息  // this.addFieldError("usernameError", "用户名不能为空");  // }  // //判断面不能为空  // if(StringUtils.isBlank(pwd))  // {  // this.addFieldError("pwdError", "密码不能为空");  // }  //  // }  //全局校验不合理，需要使用局部校验  //局部校验的规则是validate+方法名，方法名的首字母大写，例如要验证login方法，，验证方法名就是validateLogin    **public** **void** validateLogin() {    //用户名不能为空，  // if(username==null||"".equals(username))  **if**(StringUtils.*isBlank*(username))  {  //添加错误信息  **this**.addFieldError("usernameError", "用户名不能为空");  }  //判断密码不能为空  **if**(StringUtils.*isBlank*(pwd))  {  **this**.addFieldError("pwdError", "密码不能为空");  }    }        **public** String getUsername() {  **return** username;  }  **public** **void** setUsername(String username) {  **this**.username = username;  }  **public** String getPwd() {  **return** pwd;  }  **public** **void** setPwd(String pwd) {  **this**.pwd = pwd;  }  @Override  **public** String toString() {  **return** "LoginAction [username=" + username + ", pwd=" + pwd + "]";  }    } |

|  |
| --- |
| <!-- a\_validation 中的代码中手动校验的action -->  <action name=*"user\_\*"* class=*"com.igeek\_01\_validation.LoginAction"* method=*"{1}"*>  <!-- 验证不通过 -->  <result name=*"input"*>/a\_validation/login.jsp</result>  </action> |

提示：

1．被校验方法首字母 要大写。

2．全局校验一定会执行，如果出现局部校验。先执行的局部校验，再执行全局校验。

【注意点】

你的局部校验和全局校验不能同时对同一个表单内容进行矛盾的校验

局部校验：age：18--30

全局校验：age：40--50

## XML配置规则校验

手动代码校验，不适用于 表单元素非常多情况，造成代码量很大，且不容易维护（代码耦合性太强！！！）。企业中一般推荐使用xml文件来配置表单校验。

执行xml配置校验的要求： Action 必须继承ActionSupport类 （为了实现 Validateable接口）。

这里根据校验规则生效的范围分为全局校验和局部校验两种。--跟手动校验的两种方式是对应的

全局校验：对Action中的所有的方法都生效

局部校验：对Action中的某个方法生效

### 全局校验

需求：校验用户名和密码不能为空。

作用域范围：校验当前Action 所有方法

编写xml的方法：在Action类所在包，创建 Action类名-validation.xml

【第一步】在a\_validation文件夹，创建mylogin.jsp页面，用来完成xml的全局校验功能，页面代码：

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*  pageEncoding=*"UTF-8"*%>  <%@ taglib uri=*"/struts-tags"* prefix=*"s"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>Insert title here</title>  </head>  <body>  <s:fielderror></s:fielderror>  <form action=*"*${pageContext.request.contextPath }*/mylogin.action"* method=*"post"*>  用户名:<input type=*"text"* name=*"username"*><br/>  密 码:<input type=*"password"* name=*"pwd"*><br/>  <input type=*"submit"* value=*"登录"*>  </form>  </body>  </html> |

创建success.jsp页面

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>My JSP 'success.jsp' starting page</title>  </head>  <body>  This is my success JSP page. <br>  </body>  </html> |

【第二步】编写MyLoginAction

|  |
| --- |
| **package** com.igeek\_01\_validation;  **import** com.opensymphony.xwork2.ActionSupport;  **public** **class** MyLoginAction **extends** ActionSupport{    **private** String username;  **private** String pwd;      @Override  **public** String execute() **throws** Exception {  System.*out*.println("进来了");  **return** *SUCCESS*;  }  **public** String getUsername() {  **return** username;  }  **public** **void** setUsername(String username) {  **this**.username = username;  }  **public** String getPwd() {  **return** pwd;  }  **public** **void** setPwd(String pwd) {  **this**.pwd = pwd;  }  } |

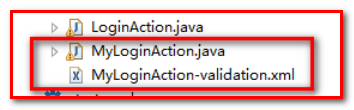
【第三步】struts.xml中的配置

|  |
| --- |
| <!-- 配置xml校验的Action -->  <action name=*"myLogin"* class=*"com.igeek\_01\_validation.MyLoginAction"*>  <result name=*"input"*>/a\_validation/mylogin.jsp</result>  <result>/a\_validation/success.jsp</result>  </action> |

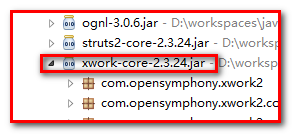
测试ok。

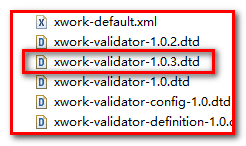
创建配置校验文件：

全局性的校验文件：直接是Action的名字-validation.xml



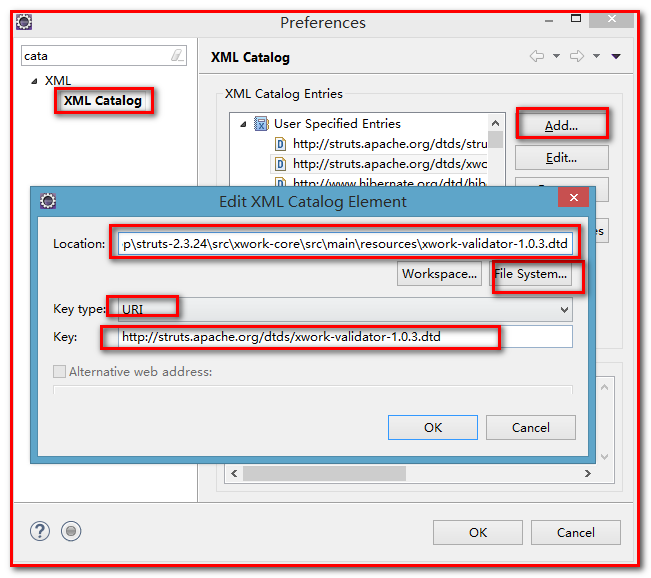
引入头信息 DTD （xwork-core包 最下面 xwork-validator-1.0.3.dtd）





|  |
| --- |
| <!DOCTYPE validators PUBLIC  "-//Apache Struts//XWork Validator 1.0.3//EN"  "http://struts.apache.org/dtds/xwork-validator-1.0.3.dtd"> |

配置提示:配置结束，要关闭xml，重新打开

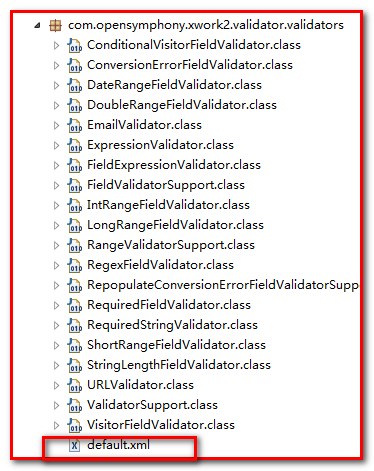


配置校验规则:

校验规则参考校验器，进行配置

（xwork core 提供 com.opensymphony.xwork2.validator.validators 下 default.xml ）







每个校验器的作用：

|  |  |
| --- | --- |
| 校验器 | 作用 |
| required | 必填校验器，要求被校验的属性不能为null |
| requiredstring | 必填字符串校验器，要求被校验的属性不能为null,长度必须大于0，默认情况下会对字符串首尾去空格 |
| stringlength | 字符串长度校验器，要求被校验的属性必须在指定的范围内，否则校验失败  minLength：指定最小长度  maxLength：指定最大长度  trim：指定校验属性被校验时，是否去除字符串前后空格 |
| regex | 正则表达式校验器，检查被校验的属性是否匹配某个正则表达式  expression：指定正则表达式  caseSensitive：指定进行正则表达式匹配时，是否区分大小写，默认值为true |
| Int | 整数校验器，要求field的整数值必须在指定范围内  min：指定最小值  max：指定最大值 |
| double | 双精度浮点型校验器，要求field的值必须在指定范围内  min：指定最小值  max：指定最大值 |
| fieldexpression | 字段ognl表达式校验器，要求field满足一个ognl表达式，  expression：指定ognl表达式  该逻辑表达式基于ValueStack进行求值，返回true时，验证通过 |
| email | 邮件地址校验器，如果field的内容不为空，则必须是合法邮件地址 |
| url | 网址校验器，如果field的内容不为空，则必须是合法的url地址 |
| date | 日期校验器，field的内容必须在某个范围内  min：指定最小值  max：指定最大值 |
| conversion | 转换校验器，指定在类型转换失败时，提示的错误信息 |
| visitor | 用于校验action中复合类型的属性，他指定一个校验文件，校验复合类型中属性 |
| expression | expression：指定ognl表达式，该逻辑表达式给予ValueStack进行求值，返回true是校验通过 |

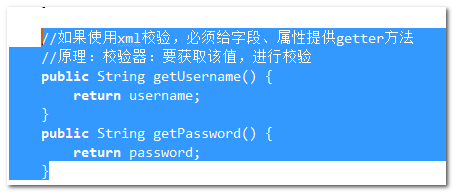
提示：这些都是struts2内置的校验器，每一种校验器都可以实现一种校验规则方式。只需要记住常用的几个就行。



|  |
| --- |
| <?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 name=*"username"*>  <field-validator type=*"requiredstring"*>  <message>用户名不能为空(xml)</message>  </field-validator>  </field>  <field name=*"pwd"*>  <field-validator type=*"requiredstring"*>  <message>密码不能为空(xml)</message>  </field-validator>  </field>  </validators> |

再次提示：配置校验器，可以参考com.opensymphony.xwork2.validator.validators 下 default.xml

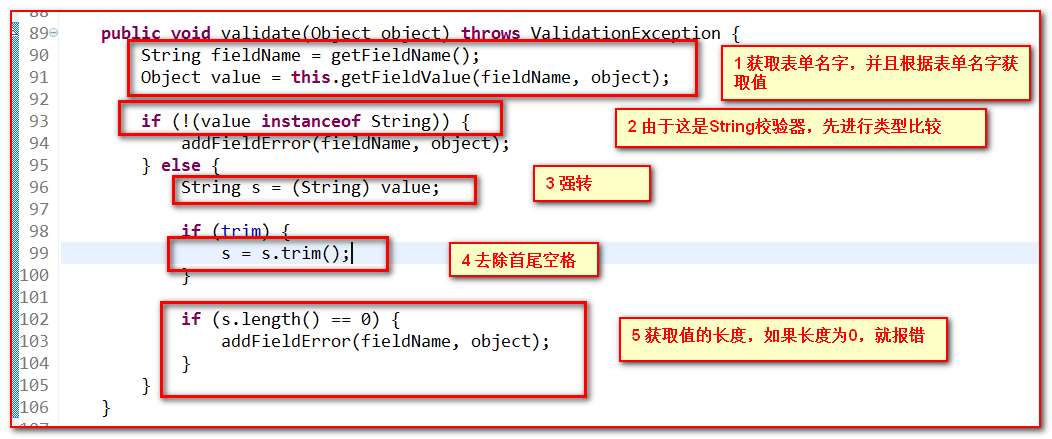
注意：使用xml配置校验的字段属性必须都有getter（因为需要将值取出来校验）



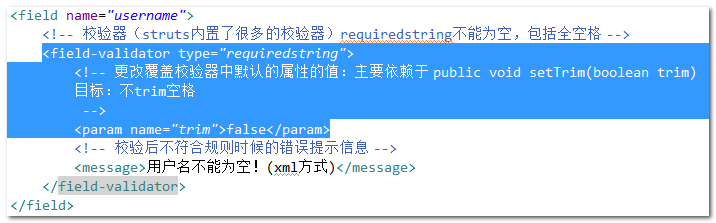
内置校验器小分析：

RequiredStringValidator校验器的作用是必须存在，必须是字符串，且默认值不能是空格。

为什么requiredstring能表示被校验的字段不能是空值呢？：原因是



【示例】如果你将trim的值设置为false的时候，校验的时候，不去除空格



【注意】属性必须提供getter方法。否则校验器无法得到数据，进行校验。

### 局部校验

不管是全局校验还是局部校验：校验规则都是固定的，区别在于作用范围，局部校验只对某个请求（方法）生效

|  |  |
| --- | --- |
| 校验器 | 作用 |
| required | 必填校验器，要求被校验的属性不能为null |
| requiredstring | 必填字符串校验器，要求被校验的属性不能为null,长度必须大于0，默认情况下会对字符串首尾去空格 |
| stringlength | 字符串长度校验器，要求被校验的属性必须在指定的范围内，否则校验失败  minLength：指定最小长度  maxLength：指定最大长度  trim：指定校验属性被校验时，是否去除字符串前后空格 |
| regex | 正则表达式校验器，检查被校验的属性是否匹配某个正则表达式  regex：指定正则表达式  caseSensitive：指定进行正则表达式匹配时，是否区分大小写，默认值为true |
| int | 整数校验器，要求field的整数值必须在指定范围内  min：指定最小值  max：指定最大值 |
| double | 双精度浮点型校验器，要求field的值必须在指定范围内  min：指定最小值  max：指定最大值 |
| fieldexpression | 字段ognl表达式校验器，要求field满足一个ognl表达式，  expression：指定ognl表达式  该逻辑表达式基于ValueStack进行求值，返回true时，验证通过 |
| email | 邮件地址校验器，如果field的内容不为空，则必须是合法邮件地址 |
| url | 网址校验器，如果field的内容不为空，则必须是合法的url地址 |
| date | 日期校验器，field的内容必须在某个范围内  min：指定最小值  max：指定最大值 |
| conversion | 转换校验器，指定在类型转换失败时，提示的错误信息 |
| visitor | 用于校验action中复合类型的属性，他指定一个校验文件，校验复合类型中属性 |
| expression | expression：残水指定ognl表达式，该逻辑表达式给予ValueStack进行求值，返回true是校验通过 |

区别：

作用域范围：校验当前Action的指定方法。

编写xml的方法：在Action类所在包，创建 类名-<Action>URL访问路径-validation.xml

【第一步】注册表单myRegist.jsp（常用校验）

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@ taglib uri=*"/struts-tags"* prefix=*"s"* %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>My JSP 'regist.jsp' starting page</title>  </head>    <body>    <s:fielderror />  <form action=*"*${pageContext.request.contextPath }*/myRegist.action"* method=*"post"*>  username:<input type=*"text"* name=*"username"*>(非空，且长度为3-10位)<br>  password:<input type=*"password"* name=*"pwd"*>(必须，且长度为6-12)<br>  repwd：<input type=*"password"* name=*"repwd"*>(必须和密码一致)<br>  age:<input type=*"text"* name=*"age"*> (年龄在18-90之间)<br>  phone:<input type=*"text"* name=*"phone"*>(手机号规则，11位数字)<br>  email:<input type=*"text"* name=*"email"*>(邮箱格式)<br>  <input type=*"submit"* value=*"注册"*>  </form>  </body>  </html> |

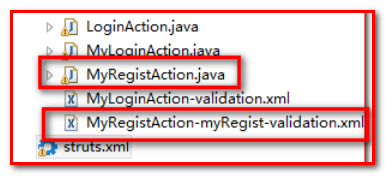
【第二步】MyRegistAction类对象定义属性，用来接收页面传递的值。

|  |
| --- |
| **package** com.igeek\_01\_validation;  **import** com.opensymphony.xwork2.ActionSupport;  **public** **class** MyRegistAction **extends** ActionSupport{  //用户名  **private** String username;  //密码  **private** String pwd;  //重复密码  **private** String repwd;  //年龄  **private** **int** age;  //手机号码:在实际项目中，手机号码都是使用String类型  **private** String phone;  // 邮箱  **private** String email;    **public** String regist() **throws** Exception {    System.*out*.println("MyRegistAction的myRegist方法被执行了...");  System.*out*.println(toString());    **return** *SUCCESS*;  }  **public** String getUsername() {  **return** username;  }  **public** **void** setUsername(String username) {  **this**.username = username;  }  **public** String getPwd() {  **return** pwd;  }  **public** **void** setPwd(String pwd) {  **this**.pwd = pwd;  }  **public** String getRepwd() {  **return** repwd;  }  **public** **void** setRepwd(String repwd) {  **this**.repwd = repwd;  }  **public** **int** getAge() {  **return** age;  }  **public** **void** setAge(**int** age) {  **this**.age = age;  }  **public** String getPhone() {  **return** phone;  }  **public** **void** setPhone(String phone) {  **this**.phone = phone;  }  **public** String getEmail() {  **return** email;  }  **public** **void** setEmail(String email) {  **this**.email = email;  }  @Override  **public** String toString() {  **return** "MyRegistAction [username=" + username + ", pwd=" + pwd  + ", repwd=" + repwd + ", age=" + age + ", phone=" + phone  + ", email=" + email + "]";  }  } |

【第三步】配置struts.xml

|  |
| --- |
| <!-- 配置xml的局部校验器的Action -->  <action name=*"myRegist"* class=*"com.igeek\_01\_validation.MyRegistAction"* method=*"regist"*>  <result name=*"input"*>/a\_validation/myRegist.jsp</result>  <result name=*"success"*>/a\_validation/success.jsp</result>  </action> |

【第四步】编写局部校验器文件 ，校验MyRegistAction的execute方法。

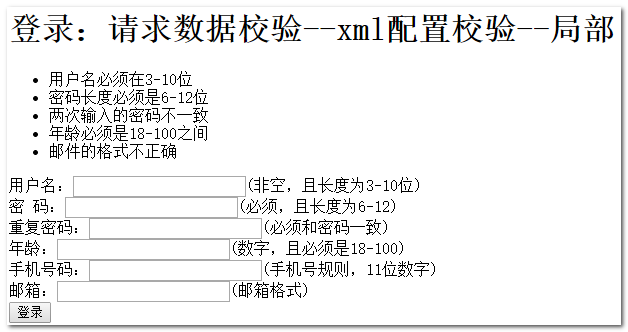


【第五步】MyRegistAction-myRegist-validation.xml文件的配置。

使用常用校验器进行校验：

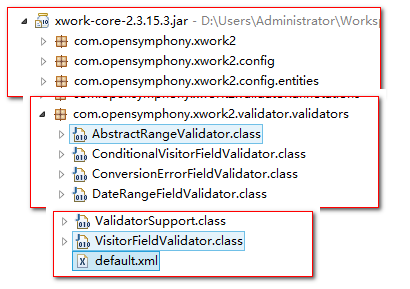
|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE validators PUBLIC  "-//Apache Struts//XWork Validator 1.0.3//EN"  "http://struts.apache.org/dtds/xwork-validator-1.0.3.dtd">  <validators>  <!-- 验证用户名 :(非空，且长度为3-10位)-->  <field name=*"username"*>  <!-- 非空 -->  <field-validator type=*"requiredstring"*>  <message>用户名不能为空</message>  </field-validator>  <!-- 长度验证 -->  <field-validator type=*"stringlength"*>  <param name=*"maxLength"*>10</param>  <param name=*"minLength"*>3</param>  <message>用户名必须是3-10位</message>  </field-validator>  </field>    <!-- 验证密码:(必须，且长度为6-12) -->  <field name=*"pwd"*>  <field-validator type=*"requiredstring"*>  <message>密码不能为空</message>  </field-validator>  <field-validator type=*"stringlength"*>  <param name=*"maxLength"*>12</param>  <param name=*"minLength"*>6</param>  <message>密码必须是6-12位</message>  </field-validator>  </field>    <!-- 重复密码：(必须和密码一致)  fieldexpression:专门用来比较表单字段的  -->  <field name=*"repwd"*>  <field-validator type=*"fieldexpression"*>  <param name=*"expression"*><![CDATA[pwd==repwd]]></param>  <message>两次密码输入不一致</message>  </field-validator>  </field>    <!-- 验证年龄：(年龄在18-90之间) -->  <field name=*"age"*>  <field-validator type=*"int"*>  <param name=*"min"*>18</param>  <param name=*"max"*>90</param>  <message>年龄必须在18-90之间</message>  </field-validator>    </field>    <!-- 验证手机号码：(手机号规则，11位数字) -->  <field name=*"phone"*>  <field-validator type=*"regex"*>  <param name=*"regex"*><![CDATA[^\d{11}$]]></param>  <message>手机号码必须是11位</message>  </field-validator>  </field>    <!-- email:符合邮箱的格式 -->  <field name=*"email"*>    <field-validator type=*"email"*>  <message>邮箱不符合格式</message>  </field-validator>  </field>  </validators> |

【第六步】测试：效果如图：



提示：

在编写校验的时候，可参考default.xml和源码来配置。位置如下：

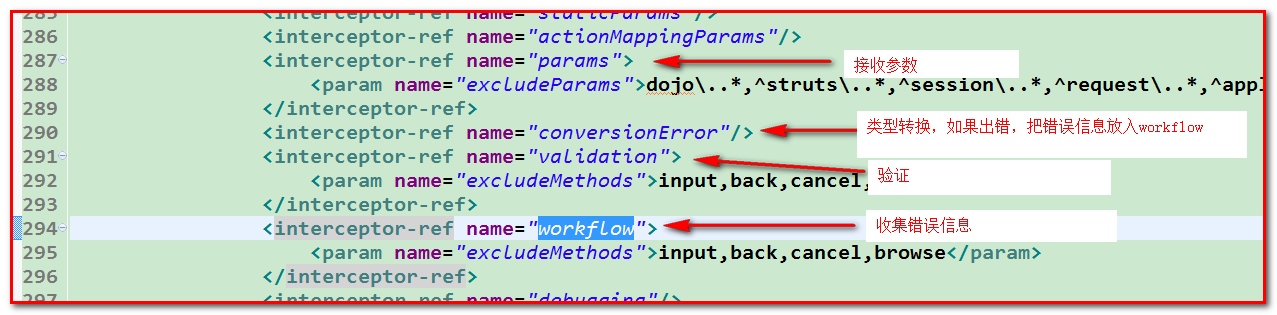


提示：在Action 执行xml 校验时， 必须要为 变量提供 getter方法！！！

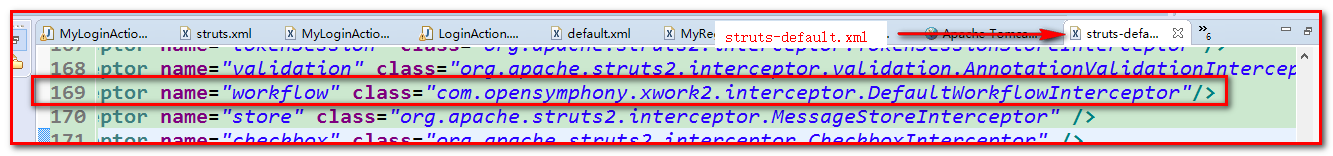
1. 必须要先接收参数（属性驱动、模型驱动）
2. 校验文件规则Action名-url-validation.xml（url不是方法名是Action的name属性）

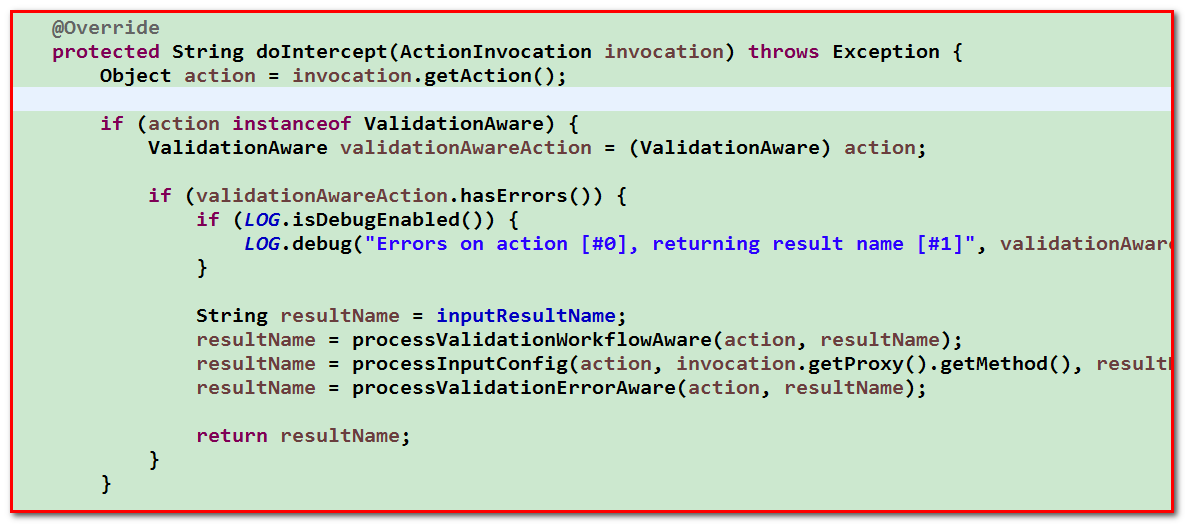
在所有的校验规则中，最负责的，无非是正则表达式校验。

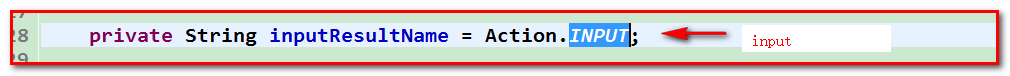
Strust2框架接收参数--》类型转换--》校验--》显示错误信息



Strust2框架验证不通过的信息怎么会跳转到input视图？



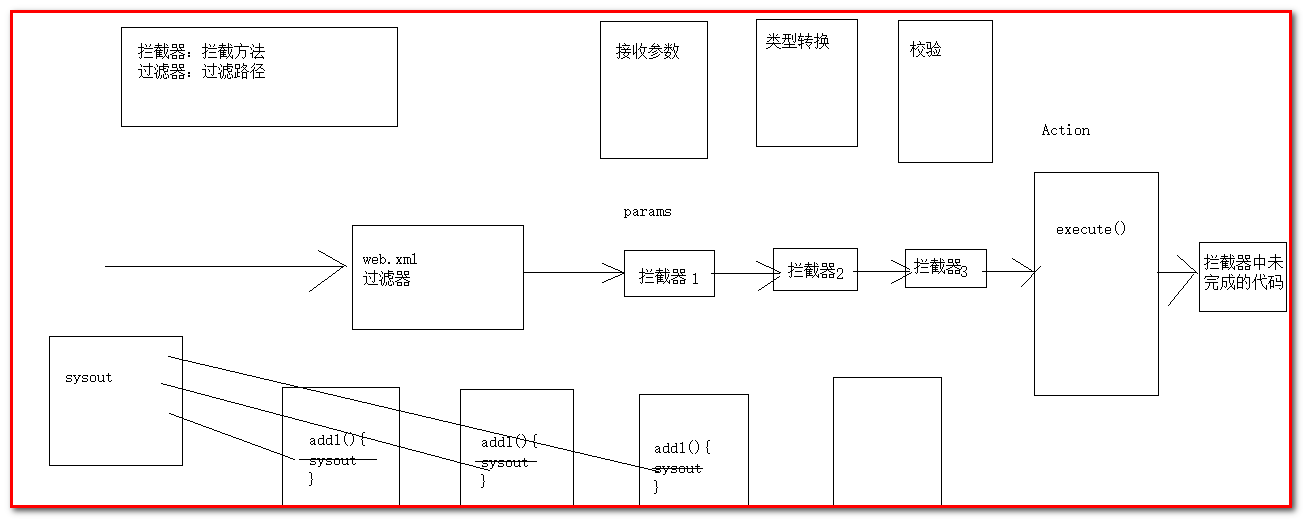




总结：一般情况下我们使用xml方式的局部校验来校验表单信息。

# 拦截器机制（重点）

## 拦截器的概念



什么是strut2的拦截器？

拦截器就是一个类，它能够拦截Action的请求，并进行预处理。

Struts2 拦截器在访问某个 Action 方法之前或之后实施拦截,（在action之前调用的称之为前置拦截器，之后也称之为后置拦截器）

拦截器是可插拔的，是一种AOP 实现 （AOP Spring 面向切面编程 ）

----- AOP 理解为 代理思想 ,使用代理模式

aop思想简单理解：在不改变原来代码的情况下，对原来的代码功能进行控制和增强（增加或减少）

新建struts2\_d03\_c02项目

【第一步】 新建一个Action，MyAction.java,放在com.igeek.action\_01包中

|  |
| --- |
| **package** com.igeek.action\_01;  **import** com.opensymphony.xwork2.ActionSupport;  /\*\*  \* **@author** jerry  \* **@version** v1.0 2017年3月29日 上午8:53:49  \*/  **public** **class** MyAction **extends** ActionSupport{  @Override  **public** String execute() **throws** Exception {  System.***out***.println("执行了MyAction");  **return** ***NONE***;  }  } |

【第二步】 新建一个拦截器类MyInterceptor implements Interceptor 接口

|  |
| --- |
| **package** com.igeek.action\_01;  **import** com.opensymphony.xwork2.ActionInvocation;  **import** com.opensymphony.xwork2.interceptor.Interceptor;  /\*\*  \* **@author** jerry  \* **@version** v1.0 2017年3月29日 上午8:56:36  \*/  **public** **class** MyInterceptor **implements** Interceptor{  //销毁  @Override  **public** **void** destroy() {  // **TODO** Auto-generated method stub    }  //初始化  @Override  **public** **void** init() {  // **TODO** Auto-generated method stub    }  //执行拦截器功能，拦截Action方法，可以在方法之前和之后编写代码  //ActionInvocation invocation 参数可以调用action中的方法  @Override  **public** String intercept(ActionInvocation invocation) **throws** Exception {  //在方法之前执行代码  System.***out***.println(" method begin");    String result = invocation.invoke();//调用action中的方法  System.***out***.println(result);  //在方法之后执行代码  System.***out***.println("method after");  **return** result;  }  } |

【第三步】 在struts.xml中配置自定义的拦截器，并且把自定义的拦截器插入Action节点中

|  |
| --- |
| <?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=*"default01"* extends=*"struts-default"* namespace=*"/interceptor\_01"*>  <!-- 给自定义的拦截器起一个名字 -->  <interceptors>  <interceptor name=*"myceptor"* class=*"com.igeek.action\_01.MyInterceptor"*></interceptor>  </interceptors>  <action name=*"myAction"* class=*"com.igeek.action\_01.MyAction"*>  <!-- 配置自定义的拦截器 -->  <!-- 如果action中没有配置任何拦截器，这个action使用strust-default.xml中  的defaultStack拦截器栈定义的18个拦截器，  如果配置了自定义拦截器，那么使用自定义的拦截器，不会使用默认的的18个拦截器  结论：我们实际开发过程中，不能只配置一个自定义的拦截器-->  <interceptor-ref name=*"myceptor"*></interceptor-ref>  </action>  </package>  </struts> |

【第四步】 测试

<http://localhost:8080/strust2_d03_c01/interceptor_01/myAction.action>

Filter：字符集编码的过滤器

Struts2 将拦截器定义拦截器栈，作用于目标Action

Struts core 包 struts-default.xml --链条...



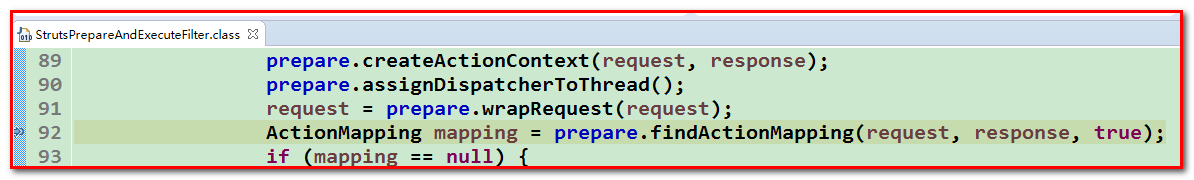
默认执行拦截器栈 defaultStack

为什么defaultStack会被默认执行呢？

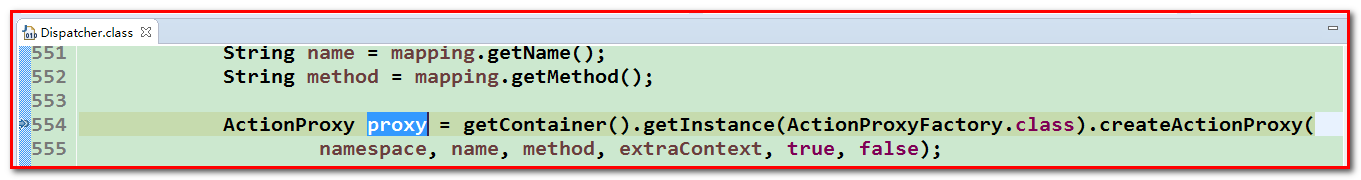


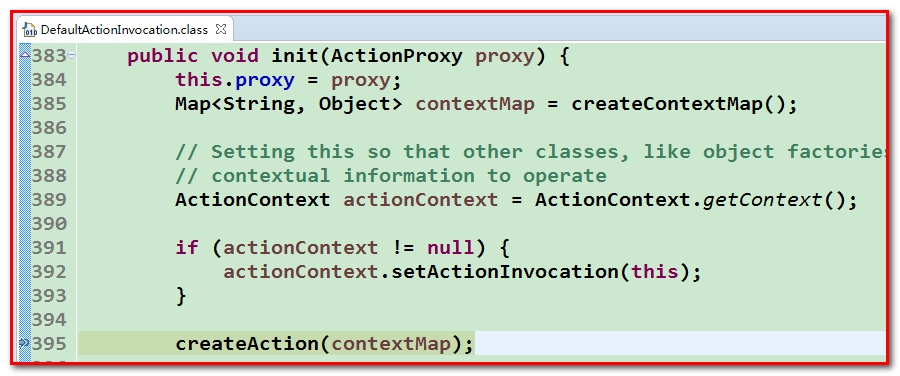
## Struts2拦截器运行原理的底层分析(了解)

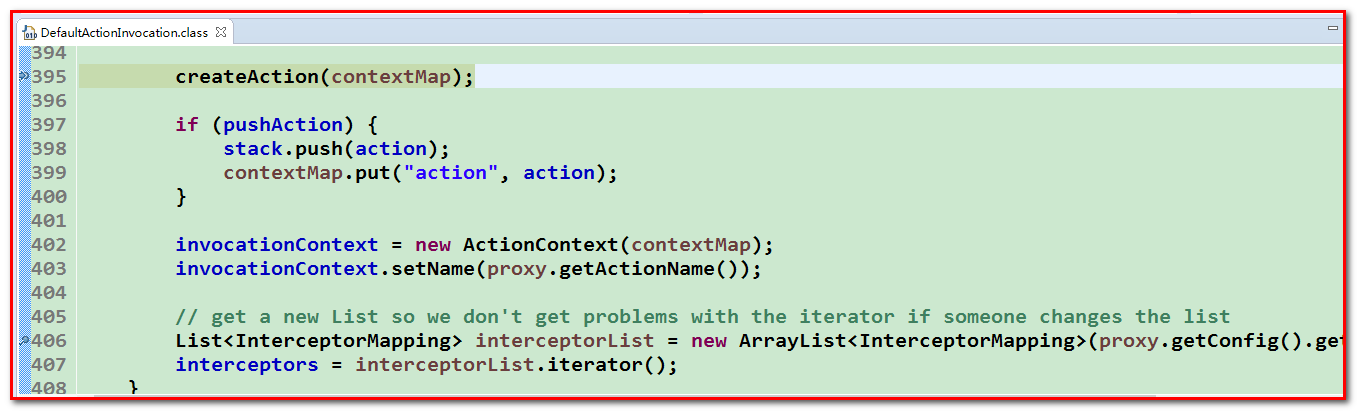
1. 在Struts过滤器中通过ActionMapping来搜集url请求信息



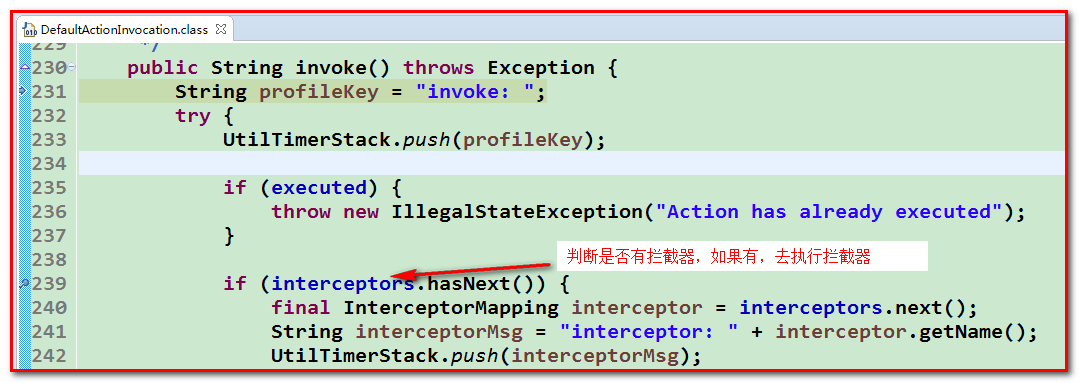
1. 通过ActionProxy 去创建Action和添加拦截器

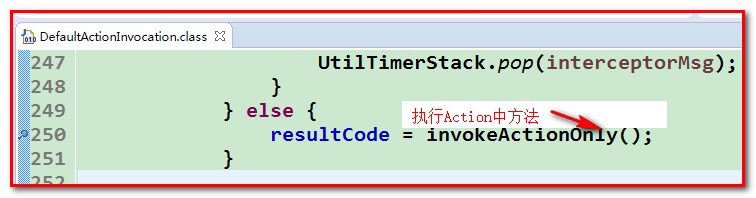


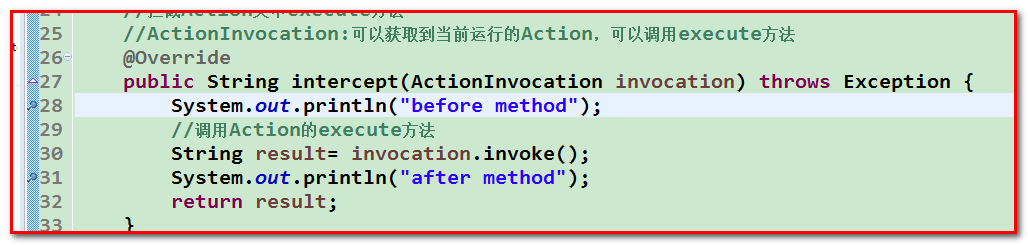


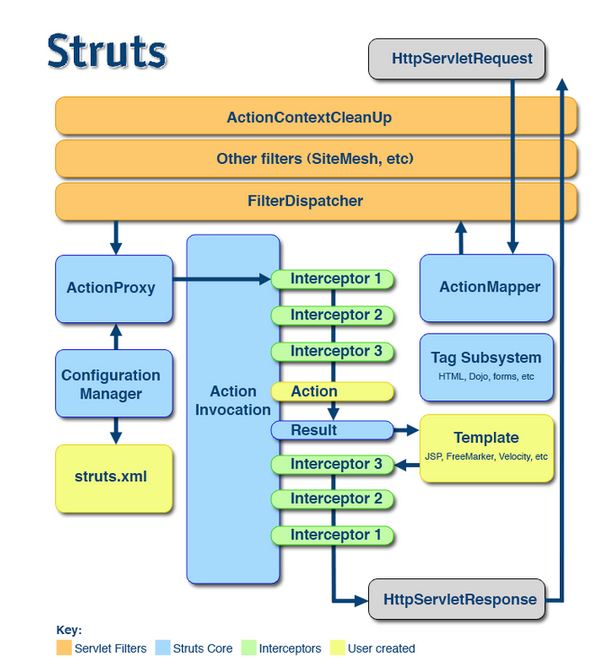


1. 先去执行拦截器，然后执行Action，最后执行拦截器中没有执行完的代码



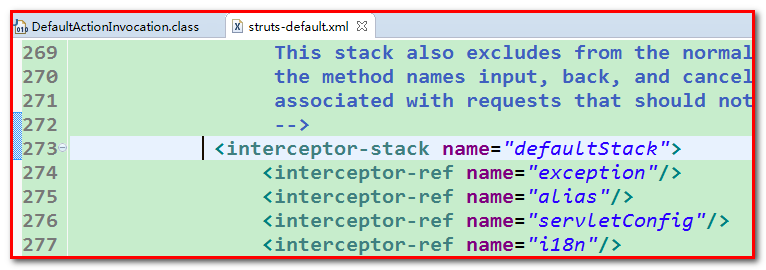






（了解：struts2的2.1.6这个版本以前，核心控制器是FilterDispatcher，从2.1.6开始，使用是StrutsPrepareAndExecuteFilter）

【原理分析】



1.当web.xml被加载之后，会初始化StrutsPrepareAndExecuteFilter，它会调用init方法初始化，准备struts相关的环境，加载相应配置文件（6个--包括struts.xml）--会将所有的action的name都加载到环境中。

2.访问/\*,----StrutsPrepareAndExecuteFilter—-默认会执行doFilter，

ActionMapping mapping = prepare.findActionMapping(request, response, true);

找你访问的这个action有没有配置（是不是存在）

如果不存在，chain.doFilter(request, response); 直接过滤拦截，忽略后面的所有的过滤器和action的执行，并告诉你说action不存在，那么就不往下走。

如果存在，execute.executeAction(request, response, mapping);准备执行action

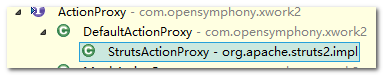
3.准备执行action，

ActionProxy proxy = config.getContainer().getInstance(ActionProxyFactory.class).createActionProxy(

namespace, name, method, extraContext, true, false);

生成action的代理对象----增强---使用过滤器增强，

proxy.execute();代理



invocation.invoke();



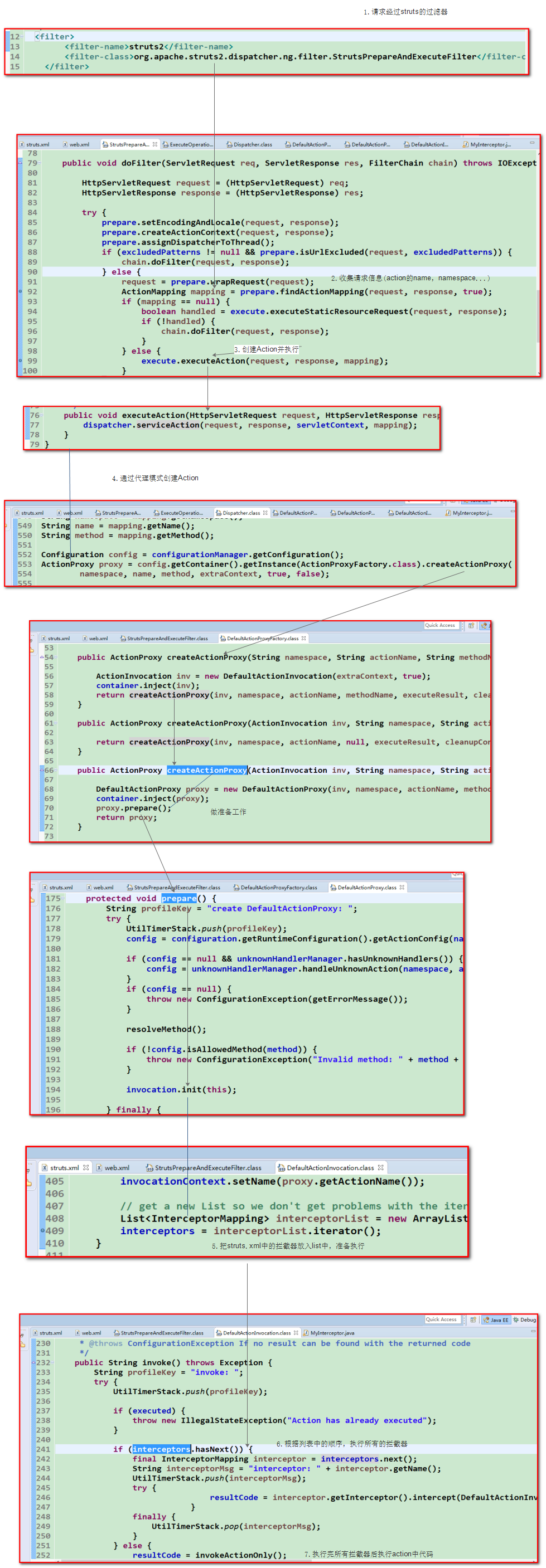
ActionInvocation增强器里面的invoke方法，判断 if (interceptors.hasNext())-配置的那些拦截器有没有执行完，如果没有执行完，就执行resultCode = interceptor.getInterceptor().intercept(DefaultActionInvocation.this);

具体的拦截器，执行之后 return invocation.invoke();，返回到原来的调用对象，原来的调用对象又会自动调用invoke方法。--（链式递归）递归调用

4.当拦截器都执行完成之后（增强完成之后），resultCode = invokeActionOnly();--》执行具体的action：invokeAction(getAction(), proxy.getConfig());返回了结果集。

【源码跟踪】

【跟踪一】：初始化xml配置文件等信息





## 拦截器的接口

* 程序中每个拦截器 都必须实现 Interceptor 接口
* 开发人员 也可以继承 AbstractInterceptor 只需要覆盖 intercept 方法
* 开发人员 也可以继承 MethodFilterInterceptor ，只需要覆盖 doIntercept 方法

excludeMethods 放行的方法（不去拦截的方法）

includeMethods 包含的方法（拦截的方法）

## 内置拦截器(了解)



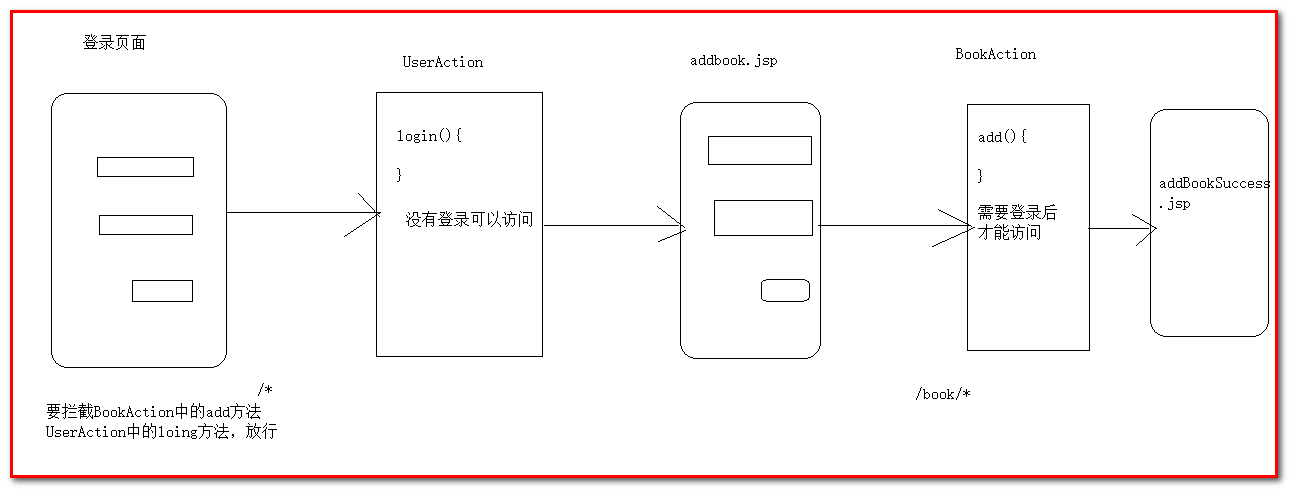


涉及到的：

* Exception ： 异常处理机制拦截器
* i18n ： 处理国际化问题
* modelDriven ： 将请求参数，封装model对象 （Action 实现ModelDriven接口）
* fileUpload ：文件上传
* params ： 请求参数转换封装
* conversionError ： 将类型转换异常进行处理
* validation ： 请求参数校验
* workflow ： 判断fieldError 是否存在，如果存在，自动跳转input视图

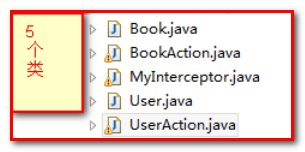
面试题：请你说说你了解的struts2中的常用拦截器？

## 自定义登录拦截器（掌握）



【业务目标】：登录拦截：如果未登录，则无法访问到业务相关的action

【具体描述】：项目中提供三个页面：login.jsp，addbook.jsp，addbooksuccess.jsp，启动项目的时候，直接进入login.jsp，用户进行登录，如果登录信息正确，则可以进入bookadd.jsp页面，进行信息的添加;如果登录失败，则返回login.jsp，并且提示相关信息；对于bookadd.jsp的提交功能，必须要在登录成功之后，才可以添加成功，如果没有登录， 是无法进入的，这里采用拦截器进行拦截，从session中获取user信息进行判断。



【示例代码】：在没有拦截器的时候，请求过程是这样的：

【第一步】创建a\_interceptor文件夹，编写login.jsp页面、addbook.jsp页面和addbooksuccess.jsp页面

<1>login.jsp页面：

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@taglib uri=*"/struts-tags"* prefix=*"s"* %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <title>My JSP 'login.jsp' starting page</title>  </head>    <body>  <s:fielderror />  <h3>登录界面</h3>  <form action="${pageContext.request.contextPath }/interceptor/user\_login.action*"* method=*"post"*>  username:<input type=*"text"* name=*"username"*><br>  password:<input type=*"password"* name=*"pwd"*><br>  <input type=*"submit"* value=*"提交"*>  </form>  </body>  </html> |

<2>图书添加页面：addbook.jsp

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <title>My JSP 'bookadd.jsp' starting page</title>  </head>  <body>  <h3>书籍信息添加页面</h3>  <form action=*"*${pageContext.request.contextPath }*/interceptor/book\_add.action"* method=*"post"*>  bookname:<input type=*"text"* name=*"bookname"*><br>  price:<input type=*"text"* name=*"price"*><br>  <input type=*"submit"* value=*"提交"*>    </form>  </body>  </html> |

<3>addbooksuccess.jsp页面

|  |
| --- |
| <body>  This is my book add success JSP page. <br>  </body> |

【第二步】创建com.igeek.interceptor包，编写模型类：User.java类和Book.java类

<1>User.java类

|  |
| --- |
| **package** com.igeek.interceptor;  //用户模型  **public** **class** User {  //用户名  **private** String username;  //密码  **private** String pwd;  **public** String getUsername() {  **return** username;  }  **public** **void** setUsername(String username) {  **this**.username = username;  }  **public** String getPwd() {  **return** pwd;  }  **public** **void** setPwd(String pwd) {  **this**.pwd = pwd;  }  } |

<2>Book类：

|  |
| --- |
| **package** com.igeek.interceptor;  **public** **class** Book {  // 书名  **private** String bookname;  // 价格  **private** **double** price;  **public** String getBookname() {  **return** bookname;  }  **public** **void** setBookname(String bookname) {  **this**.bookname = bookname;  }  **public** **double** getPrice() {  **return** price;  }  **public** **void** setPrice(**double** price) {  **this**.price = price;  }  @Override  **public** String toString() {  **return** "Book [bookname=" + bookname + ", price=" + price + "]";  }  } |

【第三步】编写UserAction类和BookAction类：

<1>编写UserAction.java类：

|  |
| --- |
| **package** com.igeek.interceptor;  **import** org.apache.struts2.ServletActionContext;  **import** com.opensymphony.xwork2.ActionSupport;  **import** com.opensymphony.xwork2.ModelDriven;  **public** **class** UserAction **extends** ActionSupport **implements** ModelDriven<User> {  **private** User user = **new** User();  **public** String login() **throws** Exception {  // 应该调用底层业务，从数据库中来查找数据  **if** ("admin".equals(user.getUsername()) && "123".equals(user.getPwd())) {  // 将数据信息存入到session中  ServletActionContext.*getRequest*().getSession()  .setAttribute("user", user);  // 跳转到成功页面  **return** "login\_success";  } **else** {  // 存储失败信息  **this**.addFieldError("userInfoError", "用户信息错误");  // 跳转回失败页面  **return** "login\_fail";  }  }  @Override  **public** User getModel() {  **return** user;  }  } |

<2>编写BookAction类：

|  |
| --- |
| **package** com.igeek.interceptor;  **import** com.opensymphony.xwork2.ActionSupport;  **import** com.opensymphony.xwork2.ModelDriven;  **public** **class** BookAction **extends** ActionSupport **implements** ModelDriven<Book>{  //创建模型  **private** Book book=**new** Book();    **public** String add() **throws** Exception {  //简单的打印书籍信息  System.*out*.println(book);  //直接返回成功  **return** *SUCCESS*;  }    @Override  **public** Book getModel() {  // **TODO** Auto-generated method stub  **return** book;  }  } |

【第四步】配置struts.xml文件：

|  |
| --- |
| <package name=*"interceptor"* extends=*"struts-default"* namespace=*"/interceptor"*>  <action name=*"user\_\*"* class=*"com.igeek.interceptor.UserAction"* method=*"{1}"*>  <result name=*"login\_success"*>/a\_interceptor/addbook.jsp</result>  <result name=*"login\_fail"*>/a\_interceptor/login.jsp</result>  </action>    <action name=*"book\_\*"* class=*"com.igeek.interceptor.BookAction"* method=*"{1}"*>  <result>/a\_interceptor/addbooksuccess.jsp</result>  </action>    </package> |

【测试】测试一下上述配置是否OK

【第五步】自定义的拦截器，MyInterceptor.java类：

|  |
| --- |
| **package** com.igeek.interceptor;  **import** org.apache.struts2.ServletActionContext;  **import** com.opensymphony.xwork2.ActionInvocation;  **import** com.opensymphony.xwork2.ActionSupport;  **import** com.opensymphony.xwork2.interceptor.Interceptor;  **import** com.opensymphony.xwork2.interceptor.MethodFilterInterceptor;  /\*\*  \* 自定义拦截器，继承MethodFilterInterceptor类  \* **@author** jerry  \*  \*/  **public** **class** MyInterceptor **extends** MethodFilterInterceptor{  //拦截Action方法  //要把UserAction的login()方法拦截，判断session中是否有user对象（是否登录）  @Override  **protected** String doIntercept(ActionInvocation invocation) **throws** Exception {  User user= (User) ServletActionContext.*getRequest*().getSession().getAttribute("user");  //判断  **if**(user!=**null**){  //执行  **return** invocation.invoke();  }**else**{  //跳转到登录页面  //获取当前执行的Action对象  ActionSupport support = (ActionSupport) invocation.getAction();  support.addFieldError("error", "请登录...");  //返回一个未登录的结果集  **return** "no\_login";  }  }  } |

【第六步】配置struts.xml:

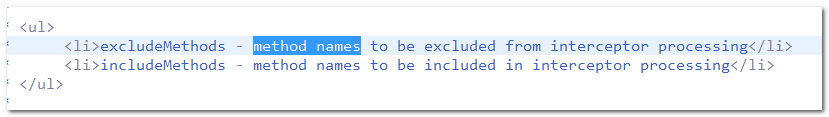
|  |
| --- |
| <package name=*"interceptor"* extends=*"struts-default"* namespace=*"/interceptor"*>  <!-- 注册拦截器 -->  <interceptors>  <!-- name:名字 class：拦截器对应的完整的包路径 -->  <interceptor name=*"myInterceptor"* class=*"com.igeek.interceptor.MyInterceptor"*>  <!-- 登录的方法放行呢？  excludeMethods：放行的方法：  includeMethods：只拦截的方法  -->  <param name=*"excludeMethods"*>login</param>  </interceptor>  <!-- 设置拦截器栈 -->  <interceptor-stack name=*"myStack"*>  <interceptor-ref name=*"myInterceptor"*></interceptor-ref>  <interceptor-ref name=*"defaultStack"*></interceptor-ref>  </interceptor-stack>  </interceptors>    <!-- 告诉系统使用自定义的拦截器栈 -->  <default-interceptor-ref name=*"myStack"*></default-interceptor-ref>    <!-- 配置全局结果集 -->  <global-results>  <result name=*"no\_login"*>/a\_interceptor/login.jsp</result>  </global-results>    <!-- UserAction -->  <action name=*"user\_\*"* class=*"com.igeek.interceptor.UserAction"* method=*"{1}"*>  <result name=*"login\_success"*>/a\_interceptor/addbook.jsp</result>  <result name=*"login\_fail"*>/a\_interceptor/login.jsp</result>  </action>  <!-- BookAction -->  <action name=*"book\_\*"* class=*"com.igeek.interceptor.BookAction"* method=*"{1}"*>  <result>/a\_interceptor/addbooksuccess.jsp</result>  </action>    </package> |

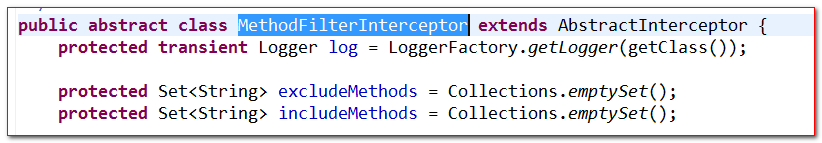
排除方法：

在MethodFilterInterceptor中定义了两个属性，一个属性是拦截器需要拦截的方法，还有一个是拦截器放行的方法

excludeMethods：拦截器拦截所有的方法，在是exclude中的方法不拦截

IncludeMethods：拦截器只拦截这个里面定义的方法





【第六步】测试

面试题： 拦截器和过滤器 区别 ？

过滤器 javaweb学习，拦截服务器端所有资源的访问 （静态、 动态）。在web.xml

拦截器 struts2 学习，在struts2框架内部，只对Action访问进行拦截 （默认拦截器 ，无法拦截静态web资源（jsp、html）， 可以将静态web资源放入WEB-INF， 通过Action间接访问）

# Struts2的文件的上传

## 回顾文件的上传

原理：

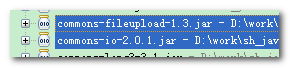
服务端需要接收 文件类型、文件名、文件大小

|  |
| --- |
| * 客户端   <form> 设置enctype编码类型(MIME类型) multipart/form-data  <form> 设置 method 提交方式 post  <input type=”file”> 元素，必须提供name属性   * 服务器   apache commons-fileupload 组件  jsp-smartupload 组件  Servlet3.0 以后 API内置文件上传API  COS 文件上传组件 |

## Struts2的文件上传机制

Struts2 内部文件上传，默认采用 apache commons-fileupload

Struts2 默认导入文件上传jar包



defaultStack 默认拦截器栈，提供 fileUpload的拦截器，用于实现文件上传



## Struts2的文件上传的实现

【提示】上传的表单一定要添加 enctype=*"multipart/form-data"*这个值

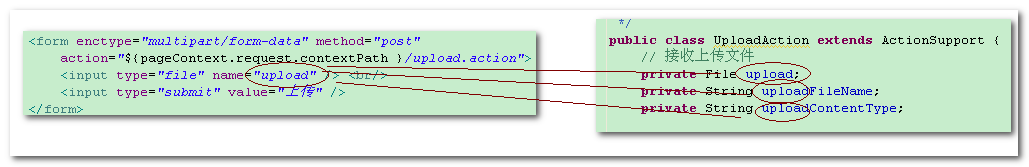
【第一步】编写页面：新建项目struts2\_d03\_c03在WebContent下面创建b\_upload文件夹，编写upload.jsp 上传文件表单

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@ taglib uri=*"/struts-tags"* prefix=*"s"*%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>My JSP 'upload.jsp' starting page</title>  </head>    <body>  <s:fielderror></s:fielderror>  <form action=*"*${pageContext.request.contextPath }*/upload.action"* method=*"post"* enctype=*"multipart/form-data"*>  <input type=*"file"* name=*"upload"*><br>  <input type=*"submit"* value=*"提交"*>  </form>  </body>  </html> |

【第二步】编写UploadAction 接收上传后的文件

|  |
| --- |
| **package** com.igeek\_01;  **import** java.io.File;  **import** org.apache.commons.io.FileUtils;  **import** com.opensymphony.xwork2.ActionSupport;  /\*\*  \* **@author** jerry  \* **@version** v1.0 2017年3月29日 下午12:03:13  \*/  **public** **class** UploadAction **extends** ActionSupport{  /\*\*  \* 通过FileUploadInterceptor拦截器可知，struts2文件上传需要3个参数，这3个参数分别是  File对象 ,名称与input框中的name对应，例如：<file name="upload"></file> -->File对象名 upload  ContentType:File对象名+ContentType，例如：uploadContentType  FileName：File对象ming+FileName，例如：uploadFileName  \*/  **private** File upload;//request.getParameter("upload");  **private** String uploadContentType;  **private** String uploadFileName;  //生成属性的gett和sett方法    @Override  **public** String execute() **throws** Exception {  //具体的文件上传工作交给struts框架去完成  //通过FileUtils工具类设置保存文件的路径  //第一个参数是上传的文件，第二个参数是要保存的文件  FileUtils.*copyFile*(upload, **new** File("d://"+uploadFileName));  **return** ***NONE***;  }  **public** File getUpload() {  **return** upload;  }  **public** **void** setUpload(File upload) {  System.***out***.println("setUpload");  **this**.upload = upload;  }  **public** String getUploadContentType() {  **return** uploadContentType;  }  **public** **void** setUploadContentType(String uploadContentType) {  System.***out***.println("setUploadContentType");  **this**.uploadContentType = uploadContentType;  }  **public** String getUploadFileName() {  **return** uploadFileName;  }  **public** **void** setUploadFileName(String uploadFileName) {  System.***out***.println("setUploadFileName");  **this**.uploadFileName = uploadFileName;  }  } |

【注意】成员变量的名字必须和页面对应：



【第三步】配置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>  <package name=*"default"* extends=*"struts-default"* namespace=*"/"*>  <action name=*"upload"* class=*"com.igeek\_01.UploadAction"*>  <result name=*"input"*>/b\_upload/upload.jsp</result>  </action>  </package>    </struts> |

【第四步】访问

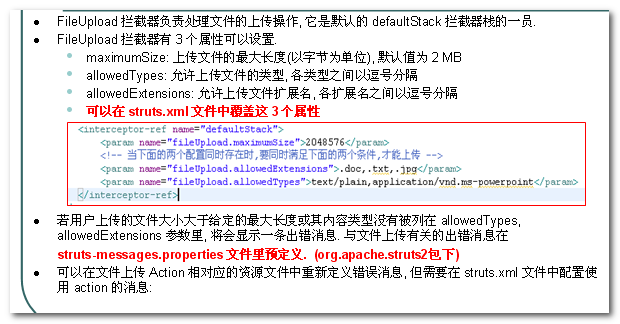
【提示】文件会默认上传到：盘符:\apache-tomcat-7.0.69\work\Catalina\localhost\struts2\_d03\_c03

因为这是个临时的存储位置，一旦系统运行结束，就自动删除

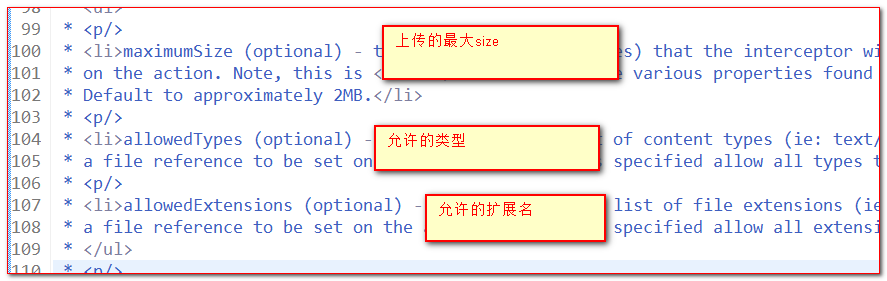
## 文件上传的参数设置

默认只能上传2m

在struts2 文件上传，存在一些限制参数，当违背参数，跳转 input 视图



【知识点1】设定允许上传的文件后缀名：

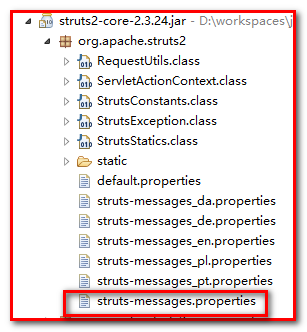


【示例】

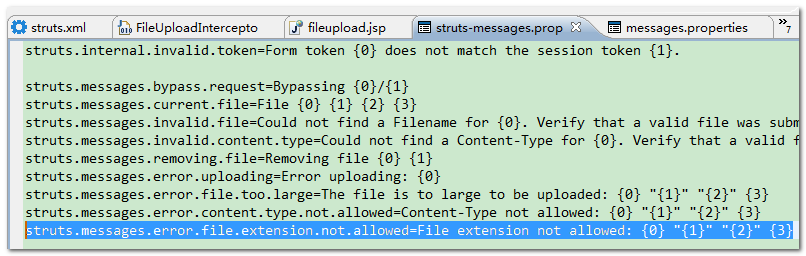
通过在action中配置拦截器栈中的fileUpload拦截器的参数， 注意：该配置只影响当前的action的文件上传。

|  |
| --- |
| <?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=*"default"* extends=*"struts-default"* namespace=*"/"*>  <action name=*"upload"* class=*"com.igeek\_01.UploadAction"*>  <result name=*"input"*>/b\_upload/upload.jsp</result>  <!-- 设置拦截器参数 -->  <interceptor-ref name=*"defaultStack"*>  <!-- 设置允许上传的文件扩展名 -->  <param name=*"fileUpload.allowedExtensions"*>.txt</param>  </interceptor-ref>  </action>  </package>    </struts> |

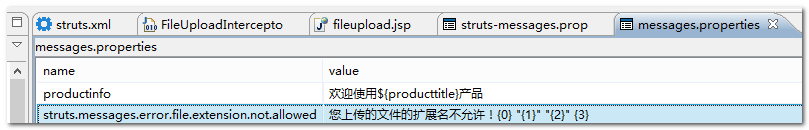
当如果页面上传的文件不符合上图中的设置的时候，系统会报错，给出的提示信息默认是英文提示，但是我们可以改变，如何改变呢？打开struts2的核心包，找到struts-messages.properties属性文件，如下图：



打开之后，如下图所示：



覆盖默认的国际化提示信息：



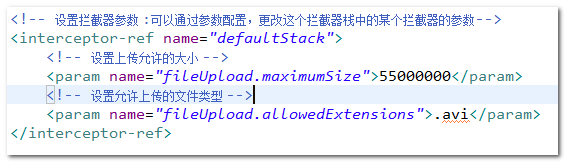
【知识点2】在文件上传的过程中，会出现大小问题，这里我们进行大小设定

struts有两个地方来限制文件上传大小的！

1）核心配置文件中的常量（最常用最有效的方式）



2）在action中配置：（第二种方式的大小其实受第一种方式限制）



【问题】：这两种有什么区别：



重点和小结

1 实现自定义拦截器，实现未登录访问控制

2 文件上传实现和配置（了解）