Struts2的基本用法：

在Model 1模式下，整个Web应用几乎全部由JSP页面组成，JSP页面接收处理客户端请求，对请求处理后直接做出响应，用少量的JavaBean来处理数据库连接，数据库访问等操作。

Model 2已经是基于MVC架构的设计模式。在Model 2架构中，Servlet作为前端控制器，负责接收客户端发送的请求，在Servlet中只包含控制逻辑和简单的前端处理。然后，调用后端JavaBean来完成实际的逻辑处理，最后，转发到相应的JSP页面处理显示逻辑。

|  |
| --- |
| 浏  览  器 |

|  |
| --- |
| （控制器）Servlet |

1请求

企

业

数

据

库

4 2

|  |
| --- |
| （视图）  JSP |

5 响应

模型 JavaBean

MVC有如下特点

1）多个视图可以对应一个模型，按MVC设计模式，一个模型对应多个视图，可以减少代码的复制及代码的维护量，一旦模型发生改变，也易于维护。

2）模型返回的数据与显示逻辑分离，模型数据可以应用任何的显示技术，例如，使用JSP页面，Velocity模版或者直接产生Excel文档等。

3）应用被分隔为三层，降低了各层之间的耦合，提供了应用的可扩展性。

4）控制层的概念也很有效，由于它把不同的模型和不同的视图组合在一起，完成不同的请求，因此，控制层可以说是包含了用户请求权限的概念。

5）MVC更符合软件工程化管理的精神，不同的层次各司其职，每一层的组件具有相同的特征，有利于通过工程化和工具化产生管理程序代码。

下载Struts2 步骤

1）Full Distribution:下载struts2的完整版，包括Struts2的示例应用，核心库，源代码和文档等。

2）Example Applications:仅下载Struts2的示例应用。

3）Essential Dependencise:仅下载Struts2的核心库。

4）Documentation:仅下载Struts2的相关文档，包含Struts2的使用文档，参考手册和API文档等。

5）Source:下载Struts2的全部源代码。

包里面包含

apps：该文件夹包含了基于Struts 2的示例应用，

docs：该文件夹包含了Struts 2的相关文档

lib：该文件夹下包含Struts 2框架的核心类库。

src：该文件夹下包含了Struts2 框架的全部源码。

Struts2应用的开发步骤

1）在web.xml文件中定义核心Filter来拦截用户请求

<!-- 定义struts 2的核心Filter->

 <filter>

  <filter-name>struts2</filter-name>

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

 </filter>

  <!-- 让Struts 2的核心Filter拦截所有请求 -->

 <filter-mapping>

  <filter-name>struts2</filter-name>

  <url-pattern>/\*</url-pattern>

 </filter-mapping>

1. 如果需要以POST方式提交请求，则定义包含表单数据的JSP页面，如果仅仅只是以GET方法发送请求，则无须经过这一步。
2. 定义处理用户请求的Action：MVC框架的底层机制是：核心Servlet或Filter接收到用户请求后，通常会对用户请求进行简单的预处理，例如解析，封装参数等，然后通过反射来创建Action实例，

1）利用配置文件：例如配置login.action对应使用LoginAction类，这就可以让MVC框架知道创建哪个Action。

2）利用约定：这种用法可能是受Rails框架的启发，例如xxx.action总是对象XxxAction类。

在MVC框架中，控制器实际上由两部分组成，即拦截所有用户请求，处理请求的通用代码都由核心控制器完成，而实际的业务控制，则有Action处理。

1. 配置Action。

5）配置处理结果和物理视图资源之间的对应关系

<action name=”login” class=”com.yfy.action.LoginAction”>

<result name=”input” >/WEB-INF/content/login.jsp</result>

</action>

1. 编写视图资源。

Struts 2的常规配置

struts.xml：该文件应该放在Web应用的类加载路径下，通常就是放在WEB-INF/classes路径下。

struts.devMode:该常量设置Struts2应用是否使用开发模式。如果设置该常量为true，则可以在应用出错时显示更多，更友好。

struts.enable.SlashesInActionNames:该常量设置Struts2是否允许在Action名中使用斜线，默认值为false。

struts.enable.DynamicMethodInvocation:该常量设置Struts2是否支持动态方法调用，该常量的默认值时true。

struts.serve.static.browserCache:该常量设置浏览器是否缓存静态内容，当应用处于开发阶段时，如果希望每次请求都获得服务器的最新响应，则可设置该常量为false。

struts.i18n.encoding:指定Web应用的默认编码集

Struts 2默认会加载类加载路径下的struts.xml，struts-default.xml，struts-plugin.xml三类文件，其中struts.xml时开发者定义的默认配置文件，struts-default.xml时Struts 2框架自带的配置文件，而struts2-plugin.xml则是Struts 2插件的默认配置文件。

按照如下加载顺序加载

1）struts-default.xml:该文件保存在struts2-core.2.5.14.jar

2）struts-plugin.xml:该文件保存在struts2-xxx-plugin-2.5.14.jar文件中。

3）struts.xml:该文件是Web应用默认的Struts2配置文件

4）struts.properties:该文件是Struts2默认的配置文件。

5）web.xml:该文件是Web应用的配置文件。

Struts 2配置常量总共三种方式：

1）通过struts.properties文件

2）通过struts.xml配置文件

<constant name=”struts.custom.i18n,resources” value=”mess” />

3）通过Web应用的web.xml文件。

<filter>

<init-param>

<param-name>struts-custom.i18n,resources</param-name>

<param-value>mess</param-value>

</init-param>

</filter>

包含其他配置文件

1）通过include元素导入其他配置文件

<include file=” ” />

Struts2 提供了一个ActionContext类，Struts 2的Action可以通过该类来访问Servlet API，

1）Object get(Object key):该方法类似于调用HttpServletRequest的getArrtibute(String name);

2）Map getApplicatio()：返回一个Map对象，该对象模拟了该应用的ServletContext实例。

3）static ActionContext getContext()：静态方法，获取系统的ActionContext实例。

4）Map getParameters():获取所有的请求参数，类似于调用HttpServletRequest对象的getParameterMap()方法。

5）Map getSession():返回一个Map对象，该Map对象模拟了HttpSession实例。

6）void setApplication(Map application):直接传入一个Map实例，将该Map实例里的key-value对转换成application的属性名，属性值。

7）void setSession(Map session):直接传入一个Map实例，将该Map实例里的key-value对转换成session的属性名，属性值。

提供了一个ServletActionContext工具类

8)static PageContext getPageContext():取得Web应用的PageContext对象。

9)static HttpServletRequest getRequest():取得Web应用的HttpServletRequest对象。

10)static HttpServletResponse getResponse()：取得Web应用的HttpServletResponse对象。

11)static ServletContext getServletContext():取得Web应用的ServletContext对象。

1）ServletContextAware:实现该接口的Action可以直接访问Web应用的ServletContext实现。重写public void setServletContext(ServletContext context):通过该方法即可访问到代表Web应用的ServletContext对象。

2）ServletRequestAware:实现该接口的Action可以直接访问用户请求的HttpServletRequest实例。重写public void setServletRequest(HttpServletRequest request):通过该方法即可访问代表用户请求的HttpServletRequest对象。

3)ServletResponseAware:实现该接口的Action可以直接访问服务器响应的HttpServletResponse实例。重写public void setServletResponse(HttpServletResponse respon):实现这个方法时，该方法内有一个HttpServletResponse参数，该参数就代表了Web应用对客户端的响应，在setServletResponse(HttpServletResponse response)方法内访问到Web应用的响应对象，并将该对象设置成Action的实例变量，从而允许在execute方法中访问该HttpServletResponse对象。

包和命名空间

1）Struts2框架的核心组件就是Action，拦截器等，Struts2框架使用包来管理Action和拦截器等。每个包就是多个Action，多个拦截器，多个拦截器引用的集合，

2）配置<package.../>元素时必须指定name属性，这个属性是引用该包的唯一标识，除此之外，还可以指定一个可选的extends属性，extends属性值必须是另一个包的name属性。指定extends属性表示让该包继承另一个包，子包可以从一个或多个父包中继承到拦截器。拦截器栈，action等配置。

3）除此之外，Struts2还提供了一种所谓的抽象包，抽象包意味着该包不能包含Action定义，为了显示指定一个包是抽象包，可以为该<package.../>元素增加abstract=”true”属性。

4）<package.../>元素指定如下几个属性

1）name:必须属性，该属性指定该包的名字，该名字是该包被其他包引用的key

2）extends:可选属性，该属性指定该包继承其他包。继承其他包，可以继承其他包中的Action定义，拦截器定义等。

3）namespace:可选属性，该属性定义该包的命名空间。当某个包指定了命名空间后，该包下的所有Action处理的URL应该是 命名空间+Action名。不指定命名空间：默认命名空间。namespace=”/”:跟命名空间。

4）abstract:可选属性，它指定该包是否为一个抽象包。抽象包中不能包含Action定义。

Action只是一个逻辑控制器，它并不直接对浏览器生成任何响应，<action.../>元素时，需要指定name属性，通常name属性由字母和数字组成，如果需要在name属性中使用斜线(“/”),则需要指定Struts2允许Action name中出现斜线，设置允许Action name中出现斜线通过struts.enable.SlashesInActionNames常量指定，设置该常量值为true，即允许Action名中使用斜线。

使用Action的动态方法调用

1）DMI（Dynamic Method Invocation动态方法调用）来处理这种请求，

//表单原来action

action=”actionName:methodName”;

//触发另一个action

var targetForm = document.forms[0]; // 获取页面的第一个表单

targetForm.action = “login!regist”; //动态修改表单的action属性

//配置Action时需要指定<allowed-methods.../>子元素。由于Struts2 的DMI原来的设计存在安全隐患，现在必须使用，method方法中使用method=”{1}”

1. <allowed-methods>regist</allowed-methods>

在<package.../>元素下可以通过<global-allowed-methods.../> 指定全局的。

1. <global-allowed-methods>regist,add</global-allowed-methods>

SMI

1. strict-method-invocation="false"

配置<action.../>时，需要配置三个属性，name指定该Action处理怎样的请求，class属性指定该Action的处理类，该属性被省略，默认使用ActionSupport作为处理类，method属性指定使用哪个方法处理请求，如果省略该method属性，则默认使用execute方法处理请求。

Struts 2默认的校验文件名遵守如下规则：ActionName-validation.xml,即如果有类名为MyAction的Action类，则应该提供名为MyAction-validation.xml文件。

<!-- 定义一个通用Action -->

<action name="\*">

<result>/WEB-INF/content/{1}.jsp</result>

</action>

如果请求a.action,则进入a.jsp页面。

因为除非请求的URL与Action的name属性绝对相同，否则将按先后顺序来决定由哪个Action来处理用户请求。因此应该将name=”\*”的Action配置在最后，否则Struts 2将使用该Action来处理所有希望使用模式匹配请求。

默认Action处理类 2种写法，当用户请求找不到Action时，系统默认Action即将处理用户请求

1)<!-- 配置一个默认Action -->

    <default-action-ref name="simple" />

    <action name="simple"  class="com.yfy.error.SimpleViewResultAction" method="simple">

  <result name="error">/WEB-INF/error/simpleError.jsp</result>

</action>

2  <action name="\*">

    <result>/WEB-INF/error/simpleError.jsp</result>

    </action>

3.7配置处理结果

<result../>元素进行配置，该元素定义逻辑视图名和物理视图资源之间的映射关系。

name:该属性指定所配置的逻辑视图名

type:该属性指定结果类型。默认结果类型dispatcher

 <action name="simple"  class="com.yfy.error.SimpleViewResultAction" >

  <result name="error">/WEB-INF/error/simpleError.jsp</result>

  </action>

理解处理结果：Action处理完用户请求后，将返回一个普通字符串，整个普通字符串就是一个逻辑视图名。Struts2通过配置逻辑视图名和物理视图之间的映射关系，一旦系统收到Action返回的某个逻辑视图名，系统就会把这个对应的物理视图呈现给浏览者。



1）plainText用法：显示实际视图资源的源代码

 <result name="plaiTest" type="plainText">

<!-- 指定实际的视图资源 -->

<param name="location">/WEB-INF/content/welcome.jsp</param>

<!-- 指定输出页时所用的字符集 -->

<param name="charSet">UTF-8</param>

</result>

2）redirect结果类型

这种类型与dispatcher结果类型相对，dispatcher结果类型是将请求forward(转发)到指定的JSP资源：而redirect类型，则意味着将请求redirect(重定向)到指定的视图资源，会丢失所有的请求参数，请求属性——当然也丢失了Action的处理结果。不能重定向到/WEB-INF/路径下的任何资源。

<result name="redirectTest" type="redirect" >/redirect.jsp</result>

参数： location:该参数指定Action处理完用户请求后跳转的地址。

parse:该参数指定是否允许在location参数值中使用表达式，该参数默认为true。

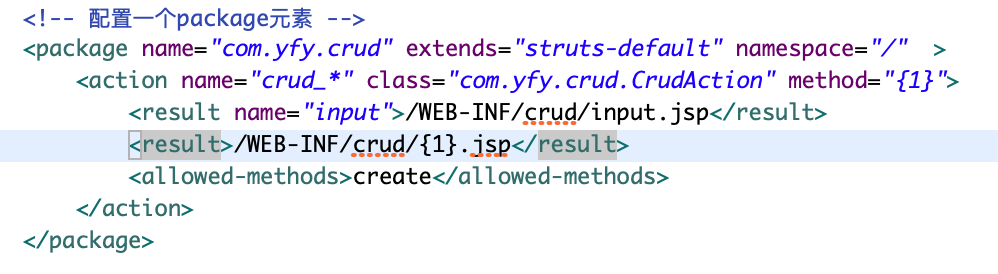
1. redirectAction使用ActionMapperFactory提供的ActionMapper来重定向请求。

参数：actionName:该参数指定重定向的Action名

namespace:该参数指定需要重定向的Action所在的命名空间。



3.7.7 动态结果

因为crud\_create匹配crud\_\*模式时，第一个星号（\*）的值是create,因此/WEB-INF/crud/{1}.jsp的{1}代表create，即对应/WEB-INF/crud/create.jsp,与配置class属性和method属性相比，配置<result.../>元素时，还允许使用OGNL表达式，这种用法允许根据Action属性值来定位物理视图资源。

全局结果

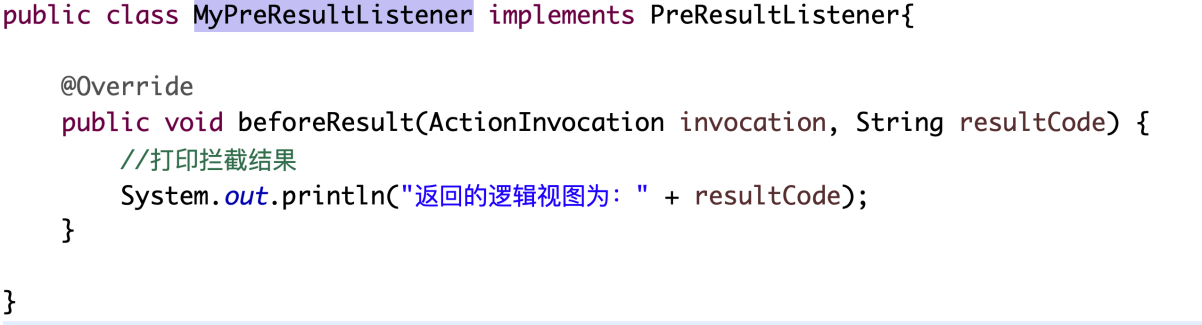
1）Struts 2 的<result.../>元素配置，也可放在<global-results.../>元素中配置



如果一个Action里包含了与全局结果里同名的结果，则Action里的局部Result会覆盖全局Result

PreResultListener是一个监听器接口，它可以在Action完成控制处理之后，系统转入实际的物理视图之间被回调。Struts2应用可由Action，拦截器添加PreResultListener监听器，添加PreResultListener通过ActionInvocation的addPreResultListener()方法完成。一旦为Action添加了PreResultListener监听器，该监听器就可以在应用转入实际物理视图之前回调该监听器的beforeResult()方法。

1. 实现PreResultListener接口



1. 将监听器注册到拦截器里面



1. 将拦截器注册到Action中





struts2的异常处理机制是通过在struts.xml文件中配置<exception-mapping.../>元素完成的，配置该元素，需要指定如下两个属性。(需要把异常在Action层一直抛)

1）exception:此属性指定该异常映射所设置的异常类型。

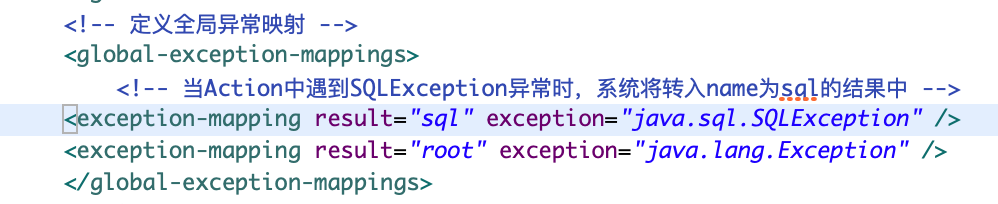
1. result:此属性指定Action出现该异常时，系统返回result属性值对象的逻辑视图名。

异常映射的种类

1）局部异常映射：将<exception-mapping.../>元素作为<action.../>元素的子元素配置。

2）全局异常映射：将<exception-mapping.../>元素作为<global-exception-mapping>元素的子元素配置。





输出异常异常信息：

1）<s:property value=”exception” /> 输出异常对象本身

2）<s:property value=”exceptionStack” /> 输出异常堆栈信息

3）<s:property value=”exception.message” /> 输出异常的message信息

Struts2国际化的步骤

1）让系统加载国际化资源文件，加载国际化资源文件有两种方式。

1）自动加载：Action范围的国际化资源文件，包范围的国际化资源文件由系统自动加载。

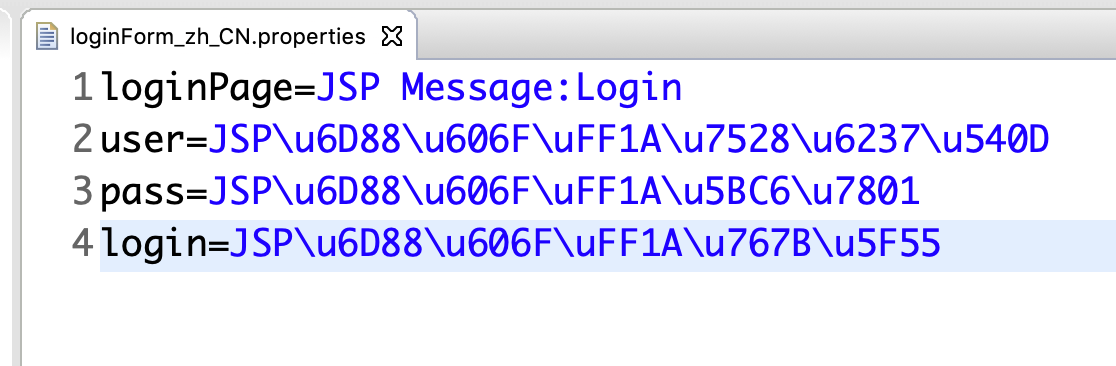
2）手动加载：JSP范围的国际化资源文件，全局访问的国际化资源文件，分别使用标签，配置常量的方式来手动加载。

1. 输出国际化。Struts2 输出国际化消息同样有两种方式。

1）在视图页面上输出国际化信息，需要使用Struts2的标签库。<s:il18n.../>

2）在Action类中输出国际化消息，需要使用ActionSupport的getText()方法来完成。





国际化语言：校验

1. 国际化资源文件的baseName与Action类的类名相同。
2. 国际化资源文件与\*.class文件保存在同一个路径下。

Action范围内的国际化资源消息可通过如下三种方式来使用

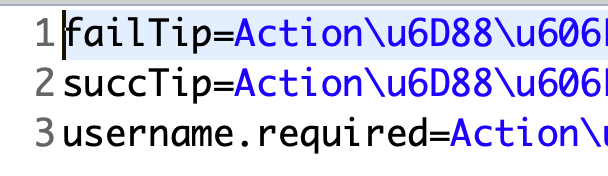
1）在JSP页面中输出国际化消息，可以使用Struts2的<s:text.../>标签，该标签可以指定一个name属性，该属性指定了国际化资源文件中的key。

2）如果想在该表单元素的label中输出国际化消息，可以为该表单标签指定一个key属性，该属性的值为国际化资源的key。

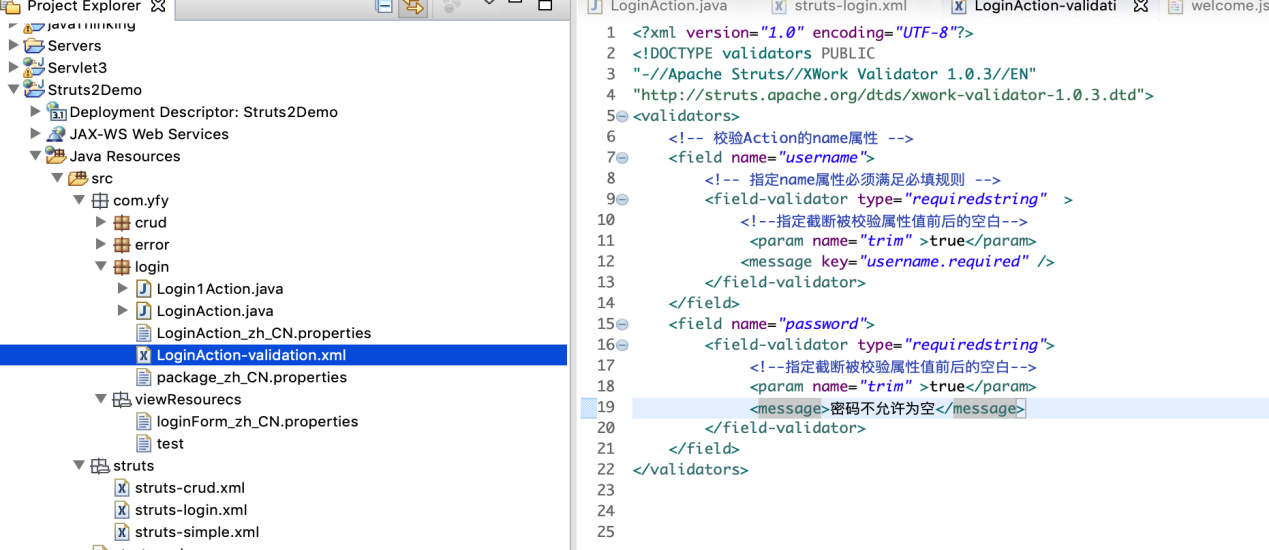
3）为了在Action类中访问国际化消息，可以使用ActionSupport类的getText()方法，该方法可以接受一个name参数，该参数指定了国际化资源文件中的key

Action国际化的命名：

1. 与Action在同一个目录



2）命名一定为LoginAction-validation.xml



3）<result name="input">/WEB-INF/viewResourecs/loginForm.jsp</result>在对应的struts.xml中配置这个语法。

4）getText("succTip")通过getText这个方法取出

包范围的国际化资源

命名package\_zh\_CN.properties

当Action范围的资源文件和包范围的资源文件同时存在，系统优先使用Action范围的资源。

使用全局国际化资源

不管在struts.xml文件中配置常量，只需要配置struts.custom.i18n.resources常量即可加载全局国际化资源文件。例如：

#指定Struts2 国际化资源文件的baseName为messageResource

struts.custom.i18n.resources=messageResource

或者#指定在struts.xml文件中配置如下的一个常量

<constant name=”struts.custom.i18n.resources” value=”messageResource” />

在JSP页面中使用<s:text />标签来输出国际化信息。

输出带占位符的国际化消息

1）如果需要在JSP页面中填充国际化消息里的占位符，则可以通过<s:text... />标签中使用多个<s:param... />标签来填充消息中的占位符。第一个<s:param.../>标签指定第一个占位符值，第二个<s:param.../>标签指定第二个占位符值。

2）如果需要在Action中填充国际化消息的占位符，则可以通过调用getText(String aTextName, List args)或getText(String key, String[] args)方法来填充占位符。该方法的第二个参数即可以是一个字符串数组，也可以是一个字符串组成的List对象，从而完成对占位符的填充。其中字符串数组，字符串集合中的第一个元素将填充第一个占位符，字符串数组，字符串集合中第二个元素将填充第二个占位符。

例如： welcomeMsg[0] ={0}, 您好！现在的时间是{1}

1)<s:i18n name="com.yfy.messageResourecs.mess">

<s:text name=”welcomeMsg”>

<s:param><s:property value=”username” /></s:param>

<s:param>${d}</s:param>

</s:text>

</s:i18n>

1. getText("succTip", new String[] {username})
2. 在资源中使用表达式: succTip=${username}

Struts2标签库

1）UI标签：主要用于生成HTML元素的标签。

1）表单标签：主要用于生成HTML页面的form元素，以及普通表单元素的标签

2）非表单标签：主要用于生成页面上的树，Tab页的等标签

2）非UI标签：主要用于数据访问，逻辑控制等的标签

1）流程控制标签：主要包含用于实现分支，循环等流程控制的标签。

2）数据访问标签：主要包含用于输出ValueStack中的值，完成国际化功能的标签。

导入Struts2标签库

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

1. parameters对象：用于访问HTTP请求参数。例如 #parameters[‘foo’]或#parameters.foo，用于返回调用HttServletRequest的getParameter(“foo”)方法的返回值。
2. request对象，用于访问HttpServletRequest的属性，例如#request[‘foo’]或#request.foo，用于返回调用HttpServletRequest的getAttribute(“foo”)方法的返回值。
3. session对象：用于访问HttpSession的属性，例如#session[‘foo’]或#session.foo,用于返回调用HttpSession的getAttribute(“foo”)方法的返回值。
4. application对象：用于访问ServletContext的属性，例如#application[“foo”]或#application.foo,用于返回调用ServletContext的getAttribute(“foo”)方法的返回值。
5. attr对象：该对象将依次搜索如下对象：PageContext,HttpServletRequest,HttpSession,ServletContext中的属性。

3.11.5 访问静态成员

<!-- 设置允许通过OGNL访问静态方法 -->

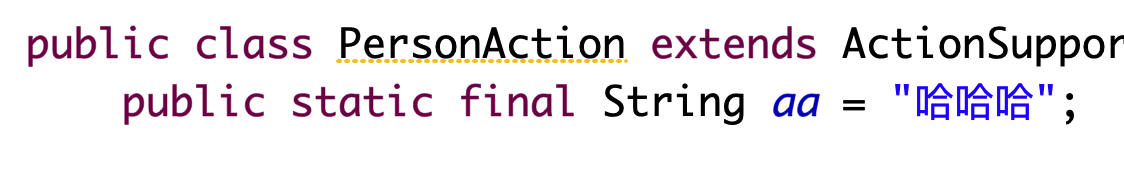
  <constant name="struts.ognl.allowStaticMethodAccess" value="true"></constant>

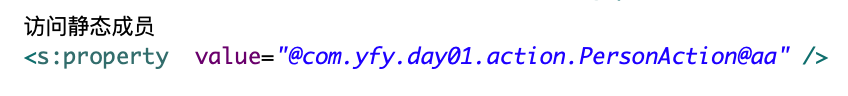
一旦设置上面所示的常量，OGNL表达式可以通过如下语法来访问静态变量

@className@staticField

@className@staticMethod(val...)

例如





3.11.7 控制标签

1）if：用于控制选择输出的标签

2）elseIf/elseif：与if标签结合使用，用于控制选择输出的标签。

3）else：与if标签结合使用，用于控制选择输出的标签

4）append：用于将多个集合拼接成一个新的集合。

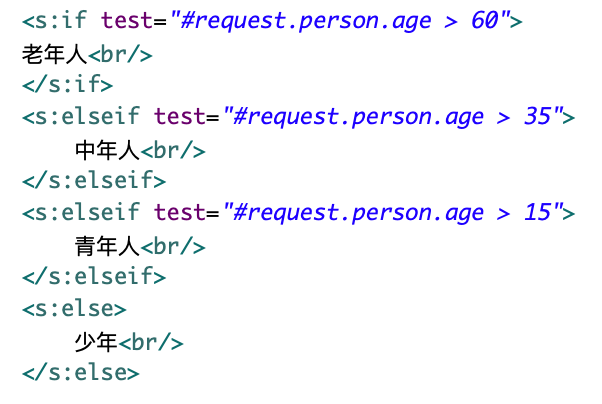
5）generator：它是一个字符串解析器，用于将一个字符串解析成一个集合。

6）iterator：它是一个迭代器，用于将集合迭代输出。

7）merge：用于将多个集合拼接成一个新的集合。但与append的拼接方式不同。

8）sort：这个标签用于对集合进行排序

9）subset：这个标签用于截取集合的部分元素，形成新的子元素。



s:iterator标签主要用于集合进行迭代

1）value：这是一个可选的属性，value属性用于指定被迭代的集合

2）var：这个是以可选的属性，该属性指定了集合里的元素的ID

3）status：这是一个可选的属性，该属性指定迭代时的IteratorStatus实例，通过该实例即可判断,即每次迭代时都会有一个IteratorStatus实例

1）int getCount():返回当前迭代了几个元素。

2）int getIndex():返回当前迭代元素的索引

3）boolean isEvent():返回当前被迭代的元素的索引是否是偶数

4）boolean isFirst():返回当前的元素是否是第一个元素。

5）boolean isLast():返回当前被迭代元素是否是最后一个元素

1. boolean isOdd():返回当前被迭代元素的索引是否是奇数。

List<String> list2 = new ArrayList<>();

<table border="1" width="200" >

<s:iterator value="#request.list2" var="name" status="st">

<tr <s:if test="#st.isOdd()">style="background-color:#bbbbbb"  </s:if>>

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

</tr>

</s:iterator>

</table>

Map<String, String> map2 = new HashMap<>();

<s:iterator value="#request.map2" var="score" status="st">

<tr <s:if test="#st.isEven()">style="background-color:#ff6600"</s:if>>

<td><s:property value="#st.getIndex() + 1" /></td>

<td><s:property value="key" /></td>

<td><s:property value="value" /></td>

</tr>

</s:iterator>

<s:textfield value="%{#request.bb}" />

append标签：用于将多个集合对象拼接起来，组成一个新的集合。

需要指定一个var属性，该属性确定拼接生成的新集合的名字，该新集合被防止在Stack Context中。 默认会去重的。

<s:append var="newList">

<s:param value="#request.list2"></s:param>

<s:param value="#request.list3"></s:param>

</s:append>

解析

<s:iterator value="#newList" var="a" status="st">

<tr>

<%-- <td>${st.count }</td>

<td>${a }</td> --%>

<td><s:property value="#st.count" /></td>

<td><s:property value="a" /></td>

</tr>

</s:iterator>

4 generator标签：可以将指定字符串按指定分隔符分隔成多个子串，临时生成的多个子串可以使用iteraot标签来迭代输出。generator将一个字符串转化成一个Iterator集合。很鸡肋的标签。

属性 ： count:该属性是一个可选的属性，该属性指定生成集合中元素的总数

separator:这是一个必填的属性，该属性指定用于解析字符串的分隔符

val:这是一个必填的属性，该属性指定被破解的字符串

converter:这是一个可选属性，该属性指定一个转换器，该转换器负责将集合中的每个字符串转换为对象，通过该转换器可以将一个字符串解析成对象集合。

var:该属性是一个可选属性，如果指定了该属性，则将生成的Iterator对象放入到Stack context中。

5 merge标签,与append相似，元素排列顺序不同。

6 subset标签用于取得集合的子集

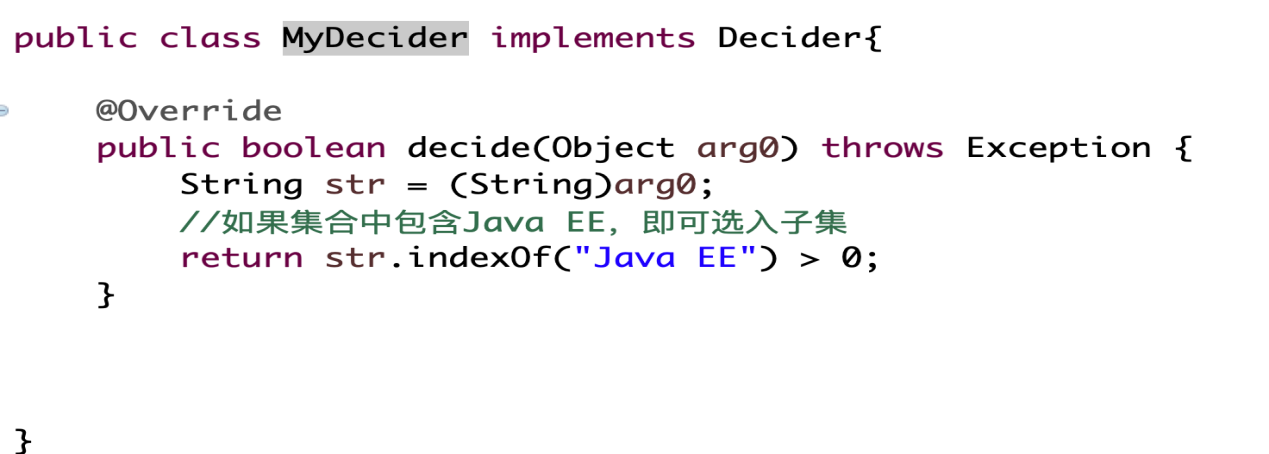
count：这是一个可选属性，该属性指定子集中元素的个数。如果不指定该属性，则默认取得源集合的全部元素。

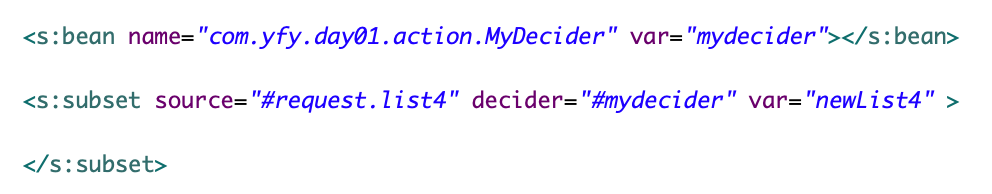
source：这是一个可选属性，该属性指定源集合。如果不指定该属性，则默认取得ValueStack栈顶的集合。

start：这是一个可选属性，该属性指定子集从源集合的第几个元素开始截取。默认从第一个元素（即0）开始截取。

decider：这是一个可选属性，该属性指定由开发者自己决定是否选中该元素。该属性必须指定一个org.apache.struts2.util.SubsetIteratorFilter.Decider对象

var：这是一个可选属性，如果指定了该属性，则将生成的Iterator对象设置成page范围的属性。放入到pageScope内。





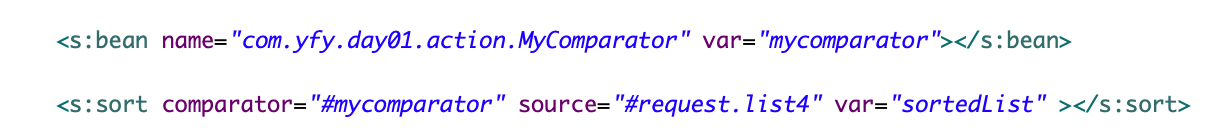
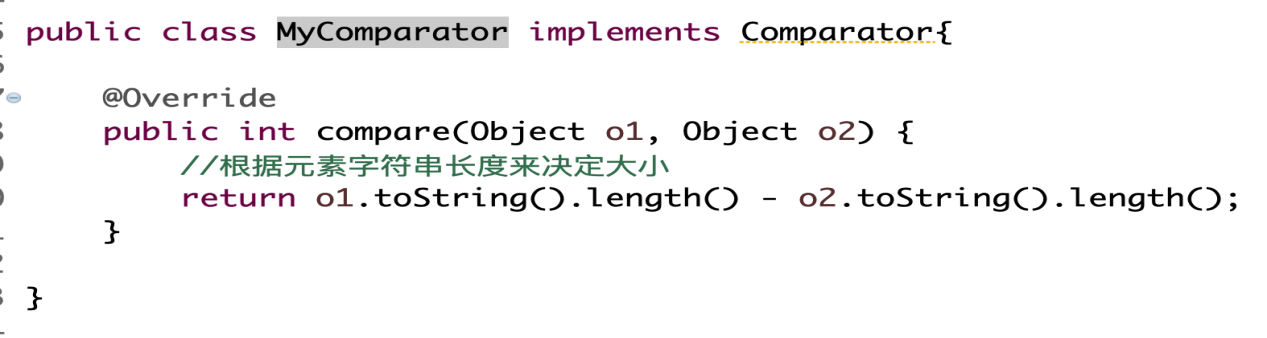


7 sort标签用于对指定的集合元素进行排序。pageScope中的用#attr

comparator:这是一个必填的属性，该属性指定进行排序的Comparator实例。

source:这是一个可选的属性，该属性指定被排序的集合，如果不指定该属性，则对ValueStack栈顶的集合进行排序。

var:这是一个可选的属性，如果指定了该属性，则将生成的Iterator对象设置成page范围的属性，不放入Stack Context中。该属性的作用与subset标签中var属性作用相同。





3.11.8 数据标签

1）action：该标签用于在JSP页面直接调用一个Action，通过指定executeResult参数，还可以将Action的处理结果包含到本页面中

1）var：这是一个可选属性，一旦定义了该属性，该Action将被放入Stack Context中。

2）name：这是一个必填属性，通过该属性指定标签调用哪个Action。

3）namespace：这是一个可选属性，通过该属性指定该标签调用哪个Action。

4）executeResult：这是一个可选属性，该属性指定是否要将Action的处理结果页面包含到本页面。该属性值默认是false，即不包含。

5）ignoreContextParams：这是一个可选参数，它指定该页面中的请求参数是否需要传入调用的Action。该属性默认是false，即将本页面的请求参数出入被调用的Action。

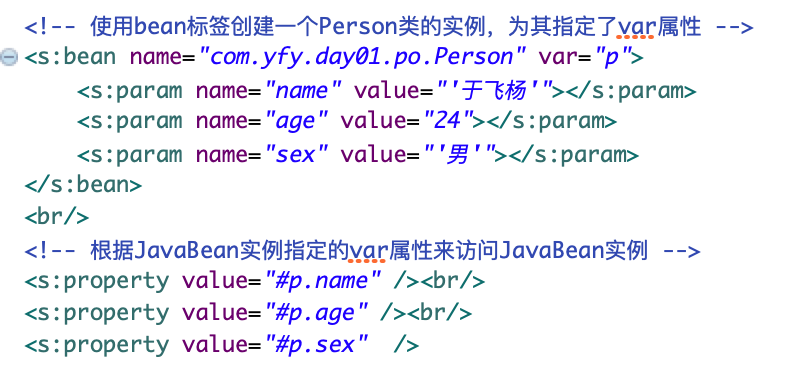
//下面调用第一个Action，并将结果包含到本页面中<br/>

<s:action name="tag\_list" executeResult="true"></s:action>

1. bean：该标签用于创建一个JavaBean实例，如果指定var属性，则可以将创建的JavaBean实例放入Stack Context中。

1）name:该属性是一个必填属性，该属性指定要实例化的JavaBean的实现类。

2）var:该属性是一个可选属性。如果指定了该属性，则该JavaBean实例会被放入Stack Context中，并让入requestScope中。 注意里面的String类型加上 ‘ ’



3）date：用于格式化输出一个日期。

1)format:这是一个可选属性，如果指定了该属性，将根据该属性指定的格式来格式化日期。

2）nice：这是一个可选属性，该属性只能true或这false，它用来指定是否输出指定日期和当前时刻的时间差。该属性默认为false，即表示不输出时间差。

3）name:这是一个必填属性，该属性指定要格式化的日期值。

4）var:这是一个可选属性，如果指定了该属性，格式化后的字符串将被放入Stack Context中，并放入requestScope中，但不会在页面中输出。

用法<s:bean name="java.util.Date" var="now"></s:bean>

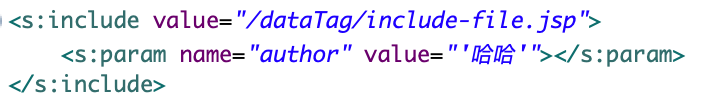
<s:date name="#now"  format="dd/MM/yyyy" />

4）il18n：用于指定国际化资源文件的baseName。

5）include：用于在JSP页面中包含其他的JSP或Servlet。

1）value：这是一个必填属性，该属性指定需要被包含的JSP页面，或者Servlet。

param传递字符串要用双引号。



在 include-file.jsp中${param.author }

6）param：用于设置一个参数，通常是用做bean标签，url标签的子标签。可以与<s:bean>或者<s:include>一起使用。

1）name:这是一个可选属性，指定需要设置参数的参数名。

2）value:这是一个可选属性，指定需要设置参数的参数值。

<param name=”color” value=”’blue’” />

如果想指定color参数的值为blue字符串。

7）push：用于将某个值放入ValueStack的栈顶。

1）value:这是一个必填属性，该属性指定需要放到ValueStack栈顶的值。只有在push标签内时，被push标签放入ValueStack中的对象才存在；一旦离开了push标签，则立即移除。

<s:push value="#p">

<s:property value="name" />

<s:property value="age" />

<s:property value="sex" />

<s:property value="time" />

</s:push>

8）set：用于设置一个新变量，并可以将新变量放入指定的范围内。

可以理解为定义来一个新变量，且将一个已有的值复制给新变量，并且可以将新变量放到指定的范围内。

1）scope:这是一个可选属性，指定新变量被放置的范围，该属性可以接受application,session,request,page或action 5个值。该属性默认值是action。

2）value;这是一个可选属性，指定将赋给变量的值。如果没有指定该属性，则将ValueStack栈顶的值赋给新变量。

3）var:这是一个可选属性，如果指定了该属性，则会将该值放入ValueStack中。

<!-- 使用bean标签定义一个JavaBean实例 -->

<s:bean name="com.yfy.day01.po.Person" var="q" >

<s:param  name="name" value="'小飞飞飞呀飞'" />

<s:param name="age" value="24" />

<s:param  name="sex" value="'男吗'" />

</s:bean>

action范围取值：<s:property value="#xxx.name" /><s:property value="#xxx.age" />${xxx.name }

request范围取值：${requestScope.xxx.name } ${requestScope.xxx.age }

<s:set var="yyy" value="#q" scope="application" ></s:set>

application范围取值：${applicationScope.yyy.name } ${applicationScope.yyy.age }

<s:set var="zzz"  value="#q" scope="session" ></s:set>

session范围取值：${sessionScope.zzz.name }${sessionScope.zzz.age }

9）text：用于输出国际化消息。

1. url：用于生成一个url地址。
2. action:这是一个可选属性，指定生成URL的地址为哪个Action，如果Action不提供，就使用value作为URL的地址值。
3. anchor：这是一个可选属性，指定URL的锚点。
4. encode:这是一个可选属性，指定是否需要对参数进行编码，默认为true。
5. escapeAmp：这是一个可选参数，指定是否需要对&符号进行编码，默认是true。
6. forceAddSchemeHostAndPort:这是一个可选参数，指定是否需要在URL对应的地址里强制添加scheme，主机和端口。
7. includeConext:这是一个可选属性，指定是否需要将当前上下文包含在URL地址中。
8. includeParams:这是一个可选属性，该属性指定是否包含请求参数，该属性的属性值只能为none，get或者all。该属性默认是get。
9. method:这是一个可选属性，该属性指定Action的方法，当使用Action来生成URL时，如果指定了该属性，则URL将链接到指定Action的特定方法。
10. namespace：这是一个可选属性，该属性指定命名空间，当使用Action来生成URL时，如果指定了该属性，则URL将链接到此namespace的指定Action处。
11. portletMode:这是一个可选属性，指定结果页面的portlet模式。
12. scheme:这是一个可选属性，用于设置scheme属性。
13. value:这是一个可选属性，指定生成URL的地址值，如果value不提供就用action属性指定的Action作为URL地址。
14. var:这是一个可选属性，如果指定了该属性，将会把该链接放入到Struts2的ValueStack中。
15. windowState:这是一个可选属性，指定结果页面的portlet的窗口状态，
16. peoperty：用于输出某个值，包括输出ValueStack， Stack Context和Action Context的值。

1）value:这是一个可选属性，指定需要输出的属性值，如果没有指定该属性，则默认输出ValueStack栈顶的值。

2）default:这是一个可选属性，如果需要输出的属性值为null，则显示default属性指定的值。

3.11.11 表达标签

form标签：1）cssClass：设置该表单元素的class

1. cssStyle：设置该表单元素的style属性，使用内联的CSS样式

3）title：设置表单的title属性。

4）disabled：设置表单元素的disabled属性。

5）label：设置表单元素的label属性。

6）labelPosition：设置表单元素的label所在位置，可接受的值为top(上面)和left(左边)，默认在左边。

7）requiredposition/requiredLabel：定义必填标记（\*）位于label元素的位置，left或right。

1. name：定义表单元素的name属性，该属性值用于与Action的属性形成对应。

9）required：定义是否在表单元素上增加必填项，设置为true时增加必填标记，否则不增加。

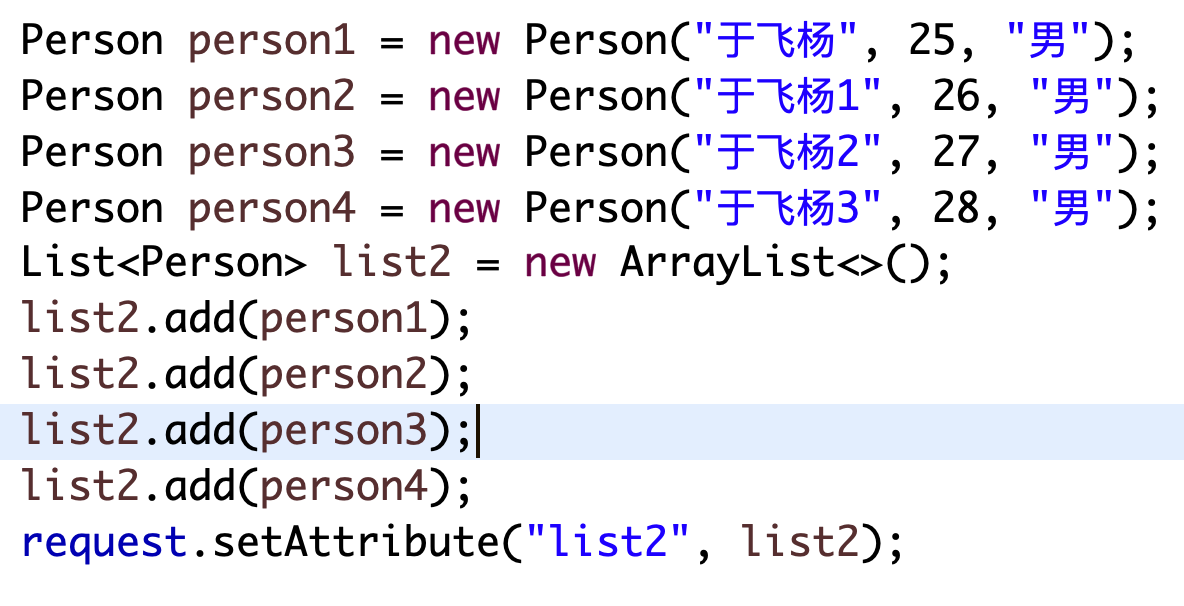
1. value：设置表单元素的value属性。

2.checkboxlist标签：可以创建多个复选框，用于生成多个<input type=”checkbox” />

1）listKey:该属性指定集合元素中的某个元素（例如集合元素为Person实例，指定Person实例的name属性）作为复选框的value。如果集合是Map，则可以使用key或value值指定Map对象的key或value作为复选框的value。

2）listValue：该属性指定集合元素中的某个属性（例如集合元素为Person实例，指定Person实例的name属性）作为复选框的标签。如果集合是Map，则可以使用key或value值指定Map对象的key或value作为复选框的标题。







提供get和set方法。

1. radio标签

radio标签与checkboxlist的用法几乎完全相同。



1. select标签：生成一个下拉列表，使用该标签必须指定list属性。

1）listKey：该属性指定集合元素中的某个元素作为复选框的value。如果集合是Map，则可以使用key或value值分别代表Map对象的key或value作为复选框的标签。

2）listValue：该属性指定集合元素中的某个属性，作为复选框的标签，如果集合是Map，则可以使用key或value值分别代表Map对象的key或value作为复选框的标签。

3）multiple：设置该列表框是否允许多选。



可多选。

8（省略一部分下拉列表）doubleselect标签：会生成一个级联列表框（会生成两个下拉列表），当选择第一个下拉列表框时，第二个下拉列表框内容会随之改变。

1)list：指定用于输入第一个下拉列表框中的选项集合。

2)listKey：该属性指定集合元素中的某个属性作为第一个下拉列表框的value，如果集合是Map，则可以使用key或value值分别代表Map对象的key或value值分别代表Map对象的key或value作为第一个下拉列表框的value。

3)listValue:该属性指定集合元素中的某个属性作为复选框的标签。如果集合是Map，则可以使用key或value值分别代表Map对象的key或value作为第一个下拉列表框的标签。

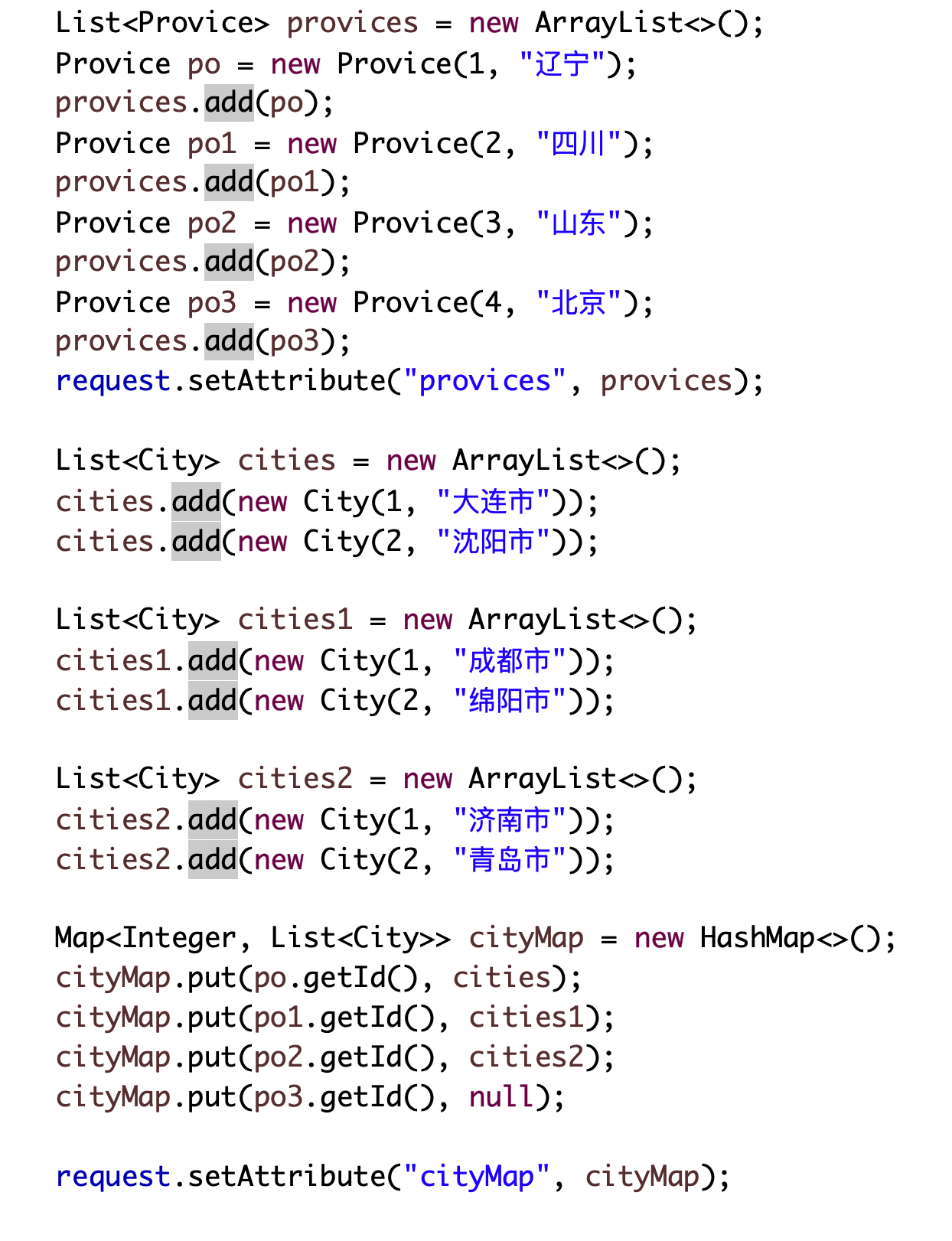
4)doubleList:指定用于输出第二个下拉列表框中选项的集合。

5)doubleListKey：该属性指定集合元素中的某个属性作为第二个下拉列表框中的value。如果集合是Map，则可以使用key或value值指定Map对象的key或value作为第二个下拉列表框的value。

6)doubleListValue:该属性指定集合元素中的某个属性作为第二个下拉框的标签。如果集合是Map，则可以使用key或value值来指定Map对象的key或value作为第二个下拉列表框的标签。

7)doubleName:指定第二个下拉框的name属性。





10 token标签：这是一个用于防止重复提交表单的标签。如果需要该标签起作用，则应该在Struts2的配置文件中启用TokenInterceptor拦截器或TokenSessionStoreInterceptor拦截器。

token标签的实现原理是在表单中增加一个隐藏域，每次加载页面时，该隐藏域的值都不同。而TokenInterceptor拦截器则拦截所有用户请求，如果两次请求时该token对应隐藏域的值相同，则阻止表单提交。

在默认情况下，token标签生成的隐藏域的name为struts.token.因此，不要在表单中另外在定义一个名为struts.token的表单域。

<!-- 表单重复提交的校验 -->

<action name="token\_\*"  class="com.yfy.day03.action.TokenAction" method="{1}" >

    <!-- 使用系统默认的拦截器 -->

<interceptor-ref name="defaultStack" />

<!-- 使用防刷新的token拦截器 -->

<interceptor-ref name="token" >

<param name="includeMethods">addToken</param> //哪个方法需要拦截重复提交

</interceptor-ref>

<!-- 定义重复提交转向的视图，该路基视图名必须为 invalid.token -->

<result name="invalid.token">/WEB-INF/day03/refresh.jsp</result>

<result>/index.jsp</result>

<allowed-methods>addToken</allowed-methods>

</action>

在界面中加入 <s:token />

1. 深入使用Struts2

表现层另一个数据处理是数据校验，数据校验可分为客户端和服务器端校验两种。客户端校验和服务器端校验都是必不可少的，二者分别完成不同的过滤。

客户端校验进行基本校验，如检验非空字段是否为空，数据格式是否正确等。客户端校验主要用来过滤用户的误操作，客户端校验的作用，拒绝误操作输入提交到服务器处理，降低服务器端负担。

服务器端校验也必不可少，服务器端校验防止非法数据进入程序，导致程序异常，底层数据库异常。服务器端校验是保证程序有效运行及数据完整的手段。

自定义类型转换器

1)在OGNL项目中有一个TypeConverter接口，这个接口就是自定义类型转换器必须实现的接口。

由于上面的接口比较复杂，所以OGNL项目还为该接口提供了一个实现类，DefaultTypeConverter，通常都采用扩展该类来实现自定义类型转换器。

2）

convertValue方法的作用最简单，该方法负责完成类型的转换，不过这种转换是双向的。

第一个参数：context是类型转换环境的上下文。

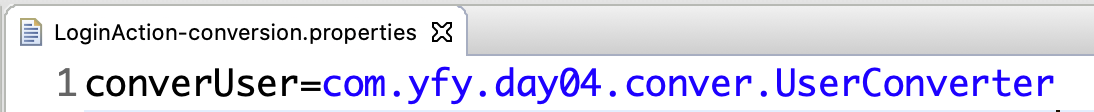
第二个参数：value需要转换的参数，随着转换方向的不同，value参数的值也是不一样的。

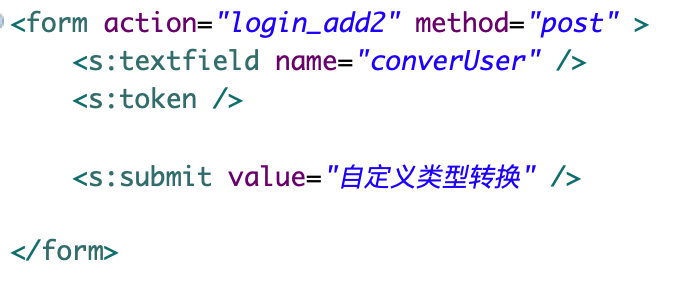
第三个参数：toType是转换后的目标类型。

用法：1）private User converUser添加get和set方法

2）public class UserConverter extends DefaultTypeConverter:类继承

3）



1. 表单

4.2 使用Struts2的输入校验

类型转换中的错误处理

在struts-default.xml中有如下配置

 <interceptor-ref name="conversionError"/> //类型转换错误拦截器

该拦截器将负责将对应错误封装成表单域错误(FieldError),并将这些错误信息放入ActionContext中。

1）处理类型转换错误

当类型转换出现异常时，conversionError拦截器会处理该异常，然后转入名为input的逻辑视图，因此应该为该Action增加名为input的逻辑视图定义。在页面中使用<s:fielderror/>标签即可输出类型转换错误信息。默认的提示是英文，只需要在国际化资源文件中添加如下代码

xwork.default.invalid.fieldvalue={0}字段类型转换失败!

在某些时候，可能还需要对特定字段指定特别的提示：

invalid.fieldvalue.<propName>=<tipMsg>:其中<propName>替换成需要进行类型转换的属性名。例如user属性里的name，应写成 invalid.fieldvalue.user.name=xxx

编写校验规则文件：

1）校验文件名：Action名字-validation.xml或者Action名字-方法名字-validation.xml,

与Action放在统一目录。

2）编写校验文件： 跟元素<validators.../>元素

字段校验风格 short-circuit=”true”:成为短路校验，就是按照按照顺序校验

<field name=”被校验的字段”>

<field-validator type=”校验器名” short-circuit=”true” >

<!-- 此处需要为不同的校验器指定数量不等的校验参数 -->

<param name=”参数名” >参数值</param>

<!-- 校验失败后的提示信息，此处key指定国际化信息的key -->

<message key=”I18NKey” >校验后的提示信息</message>

</field-validator>

</field>



1. 在struts的配置文件中：当校验失败后，自动返回名为input

<result name = "input" >/WEB-INF/day05/registForm.jsp</result>

1. 在界面中用<s:fielderror fieldName="info.getName()" />

内置校验器：如果开发者自己添加了校验器，则可以通过添加一个validators.xml文件(该文件放在WEB-INF/classes路径下)来注册校验器。

<validators>

  <validator name="校验器名字" class="校验器的实现类"/>

</validators>

如果在WEB-INF/classes路径下找到了validators.xml文件，则不会在加载默认系统的default.xml文件，所以，一定要把default.xml文件里的全部内容复制到validators.xml里。

1)必填校验器 required:该校验器要求指定的字段必须有值，非空。

<validators>

<field name=”username” >

<field-validator type=”required” >

<message/>

</field-validator>

</field>

</validators>

1. 必填字符串校验器 requiredstring:该校验器要求字段非空且长度大于0

<field-validator type=”requiredstring” >

<!-- 指定截断被校验属性值前后的空白 -->

<param name=”trim” >true</param>

<message/>

</field-validator>

3)整数校验器 包括 int, long, short该校验器要求字段的整数值必须在指定范围内

<field-validator type=”int” >

<!-- 指定最小值和最大值 -->

<param name=”min” >20</param>

<param name=”max” >100</param>

<message/>

</field-validator>

1. 日期校验器 date：要求字段的日期必须在指定范围内

<field-validator type=”date” >

<!-- 指定最小值和最大值 -->

<param name=”min” >1990-01-01</param>

<param name=”max” >2010-01-01</param>

<message/>

</field-validator>

1. 表达式校验器 expression：不可在字段校验器中使用，该表达式要求OGNL表单是返回true。

<validators>

<validator type=”expression” >

<!-- 指定表达式校验 -->

<param name=”expression’ ></param>

<message />

</validator>

</validators>

1. 字段表达式校验器 fieldexpression:它要求指定字段满足一个逻辑表达式

<field-validator type=”fieldexpression” >

<!-- 指定逻辑表达式 -->

<param name=”expression” ><![CDATA[(pass == rpass)]]</param>

<message/>

</field-validator>

7)邮件地址校验器 email：建议使用正则表达式校验

<field-validator type=”email” >

<message/>

</field-validator>

8)网址校验器 url：建议使用正则表达式校验

<field-validator type=”url” >

<message/>

</field-validator>

9)Vistor校验器 Visitor校验器用于检测Action里的复合属性。例如Action里包含了一个User类型的属性。

<!-- 指定校验的字段 -->

<field name="info">

  <field-validator type="visitor">

<!-- 指定校验文件的名字 Info-infoContext-validation.xml -->

  <param name="context">infoContext</param>

<!-- 指定校验失败后提示信息是否添加前缀 -->

  <param name="appendPrefix">true</param>

<!-- 指定校验失败的提示信息的前缀 -->

  <message>用户的:</message>

  </field-validator>

  </field>

在 Info-infoContext-validation.xml 里面填写info的属性校验信息即可。

1. 转换校验器 conversion:它检查被校验字段在类型转换中是否出现错误

<field-validator type=”conversion” >

<!-- 指定类型转换失败后，返回输入页面依然保留原来的错误输出 -->

<param name=”repopulateField” >true</param>

<message/>

</field-validator>

其中在国际化语言中：建议 invalid.fieldvalue.age= 置为空

11)字符串长度校验器 stringlength:要求被校验字段的长度必须在指定的范围内。

<field-validator type=”stringlength” >

<!-- 指定字符串的最小和最大长度 -->

<param name=”minLength” >4</param>

<param name=”maxLength” >10</param>

<message/>

</field-validator>

1. 正则表达式校验器 regex:被校验字段是否匹配一个正则表达式

<field-validator type=”regex” >

<!-- 指定逻辑表达式 -->

<param name=”regex” ><![CDATA[(\w{4, 20})]]</param>

<message/>

</field-validator>

手动完成输入校验

1. 重写validate()方法：通过重写ActionSupport类的validate()方法来进行这种校验。

 public void validate() {

String name = info.getName();

System.out.println("进入validate方法进行校验" + name);

//要求用户必须包含yfy子串

if(!name.contains("yfy")) {

addFieldError("userIf", "您的用户名必须含有yfy");

}

} 确定，初始化的时候就会校验，不推荐

<s:fielderror fieldName="userIf" />

1. 重写validateXxx()方法

 public void validateAdd() {

String name = info.getName();

System.out.println("进入validate方法进行校验" + name);

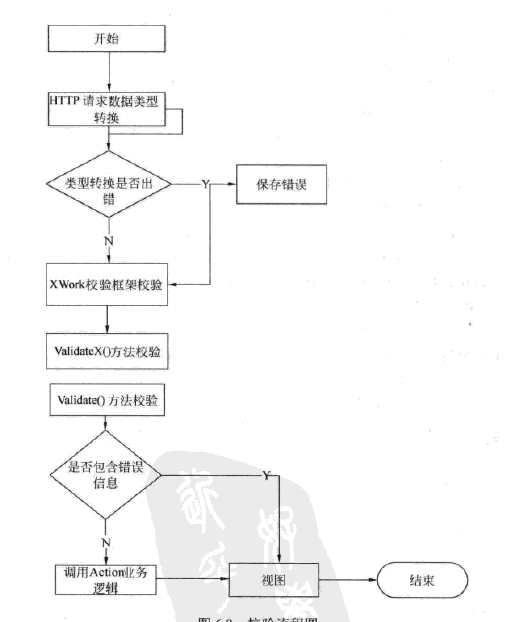
//要求用户必须包含yfy子串

if(!name.contains("yfy")) {

addFieldError("userIf", "您的用户名必须含有yfy");

}

}



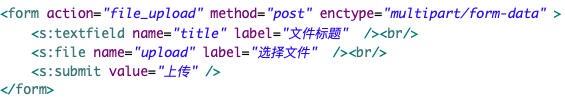
Struts2 执行数据校验的流程图

Struts2 控制文件上传

必须将表单method设置为post，将enctype设置为multipart/form-data。浏览器会把用户选择的文件的二进制数据发送给服务器。

需要两个jar包 1）commons-fileupload-1.4.jar 2)commons-io-2.6.jar

例如：



封装文件域

1）类型为File的xxx成员变量封装了该文件域对应的文件内容。

2）类型为String的xxxFileName成员变量封装了该文件域对应的文件名字。

3）类型为String的xxxContentType成员变量封装了该文件域对应的文件的文件类型。

例子 //文件，二进制字符串,对应前端的name属性

private File upload;

//封装上传文件类型

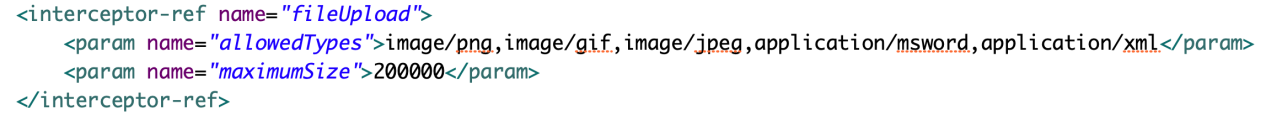
private String uploadContentType;

//封装上传文件名

private String uploadFileName;

//直接在struts.xml文件里面配置成员变量

private String savePath;



获取上传路径在action中配置param

<!-- 动态设置Action的属性值 -->

<param name="savePath">/uploadFiles</param>

在对应的Action中声明一个变量savePath并提供get和set方法。

//获取文件上传的保存位置。

public String getSavePath() {

return ServletActionContext.getServletContext().getRealPath(savePath);

}

手动实现文件过滤

1）在Action中定义一个专用于进行文件过滤的方法，该方法的方法名字是任意的，逻辑是判断上传文件的类型是否为允许类型。

public  String fileterTypes(String[] types) {

//获得允许上传的所有文件类型

String fileType = getUploadContentType();

for(String type : types) {

if(type.equals(fileType)) {

return null;

}

}

return ERROR;

}

2）利用Struts 2 的输入校验来判断用户输入的文件是否符合要求。

在struts.xml中

<!-- 设置允许上传的文件类型，在Action中提供get和set方法进行获取 -->

<param name="allowTypes">image/png,image/gif,image/jpeg</param>

<!-- 文件类型校验失败时返回input逻辑视图 -->

<result name="input">/WEB-INF/day06/uploadForm.jsp</result>

//过滤加上方法名，只校验这个方法

public void validateUpload() {

String fileterResult = fileterTypes(getAllowTypes().split(","));

//如果当前文件类型不允许上传

if(fileterResult != null) {

addFieldError("upload", "您上传的文件类型不正确");

}

}

拦截器实现文件校验

Struts2提供了一个文件上传拦截器，通过配置该拦截器便可拦截，Struts2 中文件上传的拦截器是fileUpload,为了让该拦截器起作用，只需要在Action中配置该拦截器的引用即可

1)在对应Struts.xml中的action配置拦截器

<interceptor-ref name="fileUpload">

<!-- 配置允许上传的文件类型 -->

<param name="allowedTypes">image/png,image/gif,image/jpeg</param>

<!-- 配置允许上传文件的大小 -->

<param name="maximumSize">200000</param>

</interceptor-ref>

<!-- 配置默认拦截器 -->

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

<!-- 过滤失败跳转界面 -->

<result name="input">/WEB-INF/day06/uploadForm.jsp</result>

2）默认上传跳转input界面是英文的，需要修改国际化资源文件

#改变文件类型不允许的提示信息

struts.messages.error.content.type.not.allowed=

#改变上传文件太大的提示信息

struts.messages.error.file.too.large=

#上传文件失败，未知错误

struts.messages.error.uploading

使用Struts 2控制文件下载

1）public class FileDownloadAction extends ActionSupport{

//该成员可以在配置文件中动态指定该值，指定下载文件目录

private String inputPath;

提供get和set方法

<!-- 动态设置Action的属性值 -->

<param name="inputPath">/uploadFiles</param>

2）private String fileName;获取下载文件的名称

提供get和set方法。

/\*\*

\* 定义一个返回InputStream的方法。

\* 该方法将做为下载文件的入口，

\* 且需要配置stream类型结果时指定inputName参数

\* inputName参数的值就是方法去掉get前缀，首字母小写的字符串

\*/

public InputStream getTargetFile() {

//显示中文名字必须转换，否则中文名称为action的名字

this.downFileName=new String(downFileName.getBytes(),"iso-8859-1");

//返回指定文件对应的输入流

return ServletActionContext.getServletContext().getResourceAsStream(inputPath + “/” + fileName);

}

1. private String downFileName; 文件的下载名字，中文不显示的问题

提供get和set方法。

4）

<result type="stream" >

<!-- 指定下载文件的文件类型 -->

<param name="contentType">image/png,image/gif,image/jpeg</param>

<!-- 指定getTargetFile()方法返回被下载文件的InputStream，指定被下载的入口流入流 -->

<param name="inputName">targetFile</param>

<!-- 指定该文件必须下载 -->

<param name="contentDisposition">attachment;filename=${downFileName}</param>

<!-- 指定下载文件的大小 -->

<param name="bufferSize">4096</param>

</result>

属性名称

1）contentType:指定被下载文件的文件类型。

2）inputName:指定被下载文件的入口输入流。

3）contentDisposition:指定下载文件的名字，和响应响应报头。

4）bufferSize:指定下载文件时的缓冲大小。

stream：结果类型的逻辑视图是返回给客户端一个输入流，因此无须指定location属性。

下载前的授权控制

实现Action这个接口，重写execute这个方法。会自动进入校验

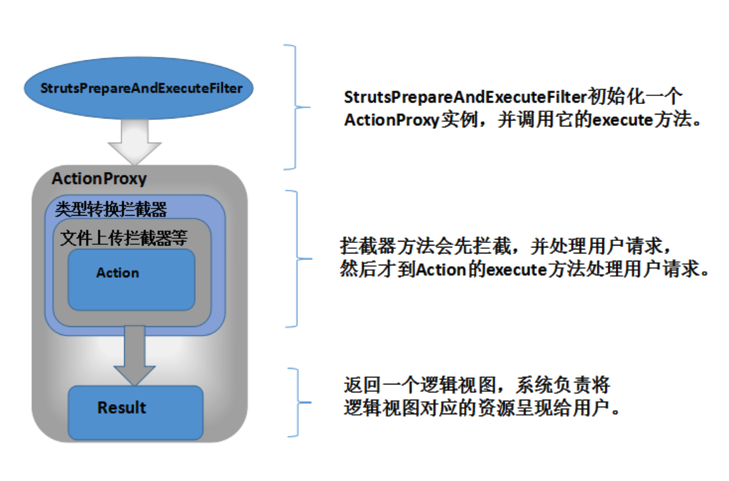
详解Struts2 的拦截器机制

拦截器是Struts2框架重要组成部分，可以把Struts2理解成一个空容器，而大量的内建拦截器完成了该框架的大部分操作。如果，params拦截器负责解析HTTP请求的参数，并设置Action的属性；servlet-config拦截器直接将HTTP请求中的HttpServletRequest实例和HttpServletResponse实例传给Action；fileUpload拦截器则负责解析请求参数中的文件域，并将一个文件域设置成Action的三个属性。

Struts 2拦截器是可插拔式的设计；如果需要使用某个拦截器只需要在配置文件中应用该拦截器即可。

拦截器的应用：当StrutsPrepareAndExecuteFilter接收到用户请求之后，大量拦截器将会对用户请求进行处理，然后才会调用用户开发的Action实例的方法来处理请求。

实际上struts2已经默认启动了大量通用功能的拦截器，只要配置Action时所在的package继承了struts-default包，这些拦截器就会起作用。



拦截器与Action之间的关系

配置拦截器

在struts.xml文件中定义拦截器只需为拦截器类指定一个拦截器名，就完成了拦截器的定义。

<interceptor name=”拦截器名” class=”拦截器的实现类” >

<!-- 可以出现0次或多次参数 -->

<param name=”参数值”>参数值</param>

</interceptor>

除此之外，还可以把多个拦截器连在一起组成拦截器栈，例如，如果需要在Action执行前同时做登录检查，安全检查和记录日志，则可以把这三个动作对应的拦截器组成一个拦截器栈。

<interceptor-stack name=”拦截器栈名” >

<interceptor-ref name=”拦截器1” />

<interceptor-ref name=”拦截器2” />

<!-- 因为拦截器和拦截器栈的功能是完全 -->

<interceptor-stack name=”拦截器栈名1” >

<interceptor-ref name=”拦截器1” />

<interceptor-ref name=”拦截器2” />

</interceptor-stack>

</interceptor-stack>

配置默认拦截器：当配置一个包时，可以为其指定默认拦截器。一旦为某个包指定了默认的拦截器，如果该包中Action没有显式指定拦截器，则默认的拦截器将会起作用。配置默认拦截器使用<default-interceptor-ref />元素，name属性是一个已经存在拦截器(栈)的名字，每个<package ../>元素只能有一个默认拦截器，该元素作为<package ../>元素的子元素使用，为该包下的所有Action配置默认的拦截器。

<package name=”包名”>

<!-- 所有拦截器和拦截器栈 -->

<interceptors>

<!-- 定义拦截器 -->

<interceptor .../>

<!-- 定义拦截器栈 -->

<interceptor-stack .../>

</interceptors>

<!-- 配置该包下的默认拦截器 -->

<default-interceptor-ref name=”拦截器名或拦截器栈名” />

<!-- 配置多个Action -->

<action .../>

</package>

在struts-default.xml文件中

<struts>

<!-- 配置Struts2配置文件的跟目录 -->

<package name=”struts-default” >

<!-- 指定struts 2的默认拦截器栈 -->

<default-interceptor-ref name=”defaultStack” />

</package>

</struts>

配置相关元素

1）<interceptors .../>元素：该元素用于定义拦截器，所有的拦截器和拦截器栈，该元素包含<interceptor .../>和<interceptor-stack .../>子元素，分别用于定义拦截器和拦截器栈。

2）<interceptor.../>元素，该元素用于定义单个的拦截器，定义拦截器只需要指定两个属性，即name和class，分别指定拦截器的名字和实现类。

3）<interceptor-stack .../>元素：该元素用于定义拦截器栈，该元素中包含多个<interceptor-ref.../>元素，用于将多个拦截器或拦截器栈组合成一个新的拦截器栈。

4）<interceptor-ref .../>元素，该元素引用一个拦截器或拦截器栈，表明应用指定拦截器，该元素只需指定一个name属性，该属性值为一个已经定义的拦截器和拦截器栈。该元素可以作为<interceptor-stack.../>和<action.../>元素的子元素使用。

5）<param .../>:该元素用于为拦截器指定参数，可以作为<interceptor .../>和<interceptor-ref .../>元素的子元素使用。

6）<default-interceptor-ref .../>：该元素为指定包配置默认拦截器，该元素作为<package .../>元素的子元素使用。

实现拦截器类

实现 Interceptor接口，

1）init():该拦截器被实例化后，在该拦截器执行拦截之前，系统将回调该方法。对于每个拦截器而言，其init()方法只执行一次，因此，该方法的方法体主要用于初始化资源，例如数据库连接等。

2）destroy():该方法与init()方法对应。在拦截器实例销毁之前，系统将回调该拦截器destroy方法，该方法用于销毁在init()方法里打开的资源。

3）intercept(ActionInvocation invocation):该方法是用户需要实现的拦截动作。就像Action的execute方法一样，intercept方法会返回一个字符串作为逻辑视图。如果该方法直接返回一个字符串，系统将会跳转到该逻辑视图对应的实际视图资源，不会调用被拦截的Action。该方法的ActionInvocation参数包含了被拦截的Action的引用，可以通过该参数的invoke方法，将控制权转给下一个拦截器，或者转给Action的execute方法。

除此之外，Struts2还提供了一个AbstractInterceptor类，该类提供了一个init()和destroy()方法的空实现，如果实现的拦截器不需要打开资源，则可以无须实现这两个方法，

<package name="com.yfy.day04.action" extends="struts-default" namespace="/" >

<!-- 应用所需使用的拦截器都在该元素下配置 -->

  <interceptors>

  <!-- 配置拦截器 -->

      <interceptor name="mySimple" class="com.yfy.day09.interceptor.SimpleInterceptor">

      <!-- 为拦截器指定参数值 在SimpleInterceptor声明一个name的私有方法和get/set获取值-->

      <param name="name">简单的拦截器</param>

      </interceptor>

  </interceptors>

<action name="login\_\*"  class="com.yfy.day04.action.LoginAction" method="{1}" >

<interceptor-ref name="defaultStack" />

<interceptor-ref name="token" />

<!-- 应用自定义拦截器，会拦截所有方法 -->

<interceptor-ref name="mySimple">

<param name="name">该名后的简单的拦截器</param>

</interceptor-ref>

<result name="invalid.token">/WEB-INF/day03/refresh.jsp</result>

<result name="success">/WEB-INF/day04/succ.jsp</result>

<result name="error">/WEB-INF/day04/error.jsp</result>

<allowed-methods>add,add1,add2</allowed-methods>

</action>

</package>

拦截方法的拦截器

为了实现方法过滤的特性，Struts2 提供了一个MethodFilterInterceptor类，该类是AbstractInterceptor类的子类，如果用户需要自己实现的拦截器支持方法过滤特性，则应该继承MethodFilterInterceptor,该类是AbstractInterceptor类的子类，如果用户需要自己实现的拦截器支持方法过滤特性，则应该继承MethodFilterInterceptor.

1)MethodFilterInterceptor类重写了AbstractInterceptor类的intercept(ActionInvocation invocation)方法，但提供了一个doIntercept(ActionInvocation invocation)抽象方法，

2) public void setExcludeMethods(String excludeMethods) {

        this.excludeMethods = TextParseUtil.commaDelimitedStringToSet(excludeMethods);

}

排除需要过滤的方法——设置方法“黑名单”，所有在excludeMethods字符串中列出的方法都不会被拦截

3)public void setIncludeMethods(String includeMethods) {

        this.includeMethods = TextParseUtil.commaDelimitedStringToSet(includeMethods);

    }

设置需要过滤的方法——设置方法“白名单”，所有在includeMethods字符串中列出的方法都会被拦截。

<package name="com.yfy.day04.action" extends="struts-default" namespace="/" >

<!-- 应用所需使用的拦截器都在该元素下配置 -->

  <interceptors>

      <!-- 配置拦截器 -->

  <interceptor name="mySimpleMehtod" class="com.yfy.day09.interceptor.MyFilterInterceptor">

  <param name="name">拦截方法的拦截器</param>

  </interceptor>

  </interceptors>

<action name="login\_\*"  class="com.yfy.day04.action.LoginAction" method="{1}" >

<!-- 加上默认拦截器，以便与参数都可以传递 -->

<interceptor-ref name="defaultStack" />

<interceptor-ref name="token" />

<!-- 自定以方法拦截器 -->

<interceptor-ref name="mySimpleMehtod">

<param name="name">该名后的方法拦截器</param>

<!-- 不需要拦截的方法 -->

<param name="excludeMethods">add1</param>

<!-- 需要拦截的方法 -->

<param name="includeMethods">add,add1</param> //add1,二者会冲突，但是后面这个取胜

</interceptor-ref>

<result name="invalid.token">/WEB-INF/day03/refresh.jsp</result>

<result name="success">/WEB-INF/day04/succ.jsp</result>

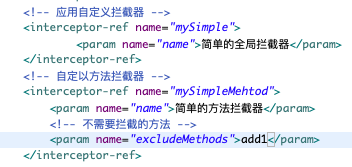
<result name="error">/WEB-INF/day04/error.jsp</result>

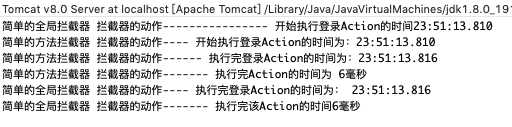
<allowed-methods>add,add1,add2</allowed-methods>

</action>

拦截器执行的顺序

例子如上：在Action的控制方法执行之前，位于拦截器链前面的拦截器将先发生作用，在Action的控制方法执行之后，位于拦截器链前面的拦截器将后发生作用





监听拦截器：

拦截器的监听接口：PreResultListener,只是之前是Action中注册该监听器，那么该监听器将只对指定Action有效

1）

//resultCode:这个参数就是被拦截Action的被拦截方法的返回值。

public void beforeResult(ActionInvocation invocation, String resultCode) {

}

//将拦截结果的监听器注册给该拦截器

invocation.addPreResultListener(new MyPreResultListener());

覆盖拦截器栈里特定拦截器的参数

<interceptors>

<!-- 配置mysimple拦截器 -->

<interceptor name=”mySimple” class=”com.yfy.interceptor.SimpleInterceptor” >

<param name=”name”>简单的拦截器</param>

</interceptor>

<interceptor name=”second” class=”com.yfy.interceptor.SecondInterceptor” />

<!-- 配置拦截器栈 -->

<interceptor-stack name=”myStack” >

<interceptor-ref name=”mySimple”>

<param name=”name”>第一个</param>

</interceptor-ref>

<interceptor-ref name=”second”>

<param name=”name”>第二个</param>

</interceptor-ref>

</interceptor-stack>

</interceptors>

<action>

<!-- 配置默认拦截器 -->

<interceptor-ref name=”defaultStack” />

<interceptor-ref name=”myStack” >

<!-- 覆盖指定拦截器的指定参数 -->

<param name=”second.name” >该名后的拦截器</param>

</interceptor-ref>

</action>

JSON的基本知识

数据结构：

1)由key-value对组成的数据结构，这种数据结构在不同的语言中有不同的实现，例如JavaScript中是一个对象，在Java中是一个Map对象。

2)有序集合。这种数据结构在不同语言中，可能有list，vector，数组和序列等实现。