佟刚Struts2笔记

从Struts1升级到Struts2：

1. Struts1里使用ActionServlet作为控制器；Struts2使用了一个过滤器作为控制器
2. Struts1中每个HTML表单都对应一个ActionForm实例；Struts2中，HTML的表单将被直接映射到一个POJO（简单Java对象）
3. Struts1的验证逻辑编写在ActionForm中；Struts2中的验证逻辑编写在Action中
4. Struts1中，Action类必须继承org.apache.struts.action.Action类；Struts2中任何一个POJO都可以是Action类
5. Struts2在页面里使用OGNL来显示各种对象模型，可以不再使用EL和JSTL

action和Action类

1. action：一个action代表一个Struts2的请求，包括超链接等请求
2. Action类：能够处理Struts2请求的类（如：Product类）

Action类是普通的Java类，可以有属性和方法，同时必须遵守下面的这些规则：

1）属性的名字必须遵守与JavaBean属性名相同的命名原则。属性的类型可以是任意类型，从字符串到非字符串(基本数据类型)之间的数据转换可以自动发生

2）必须有一个不带参数的构造器

3）至少有一个供Struts2在执行这个action时调用的方法（如Product类中的save()方法，默认是重写execute()方法）。

4）同一个Action类可以包含多个action方法（如：execute()、save()）

5）Struts2会为每一个HTTP请求创建一个新的Action实例(Action对象)

在Action中访问web资源：

1. 如何访问？
2. 和Servlet API解耦的方式：为了避免与Servlet API耦合在一起，方便Action做单元测试，Struts2对HttpServletRequest、HttpSession和ServletContext进行了封装，构造了3个Map对象来替代这3个对象，在Action中可以直接使用HttpServletRequest、HttpSession、ServletContext对应的Map对象来保存和读取数据。这种方式只能访问有限的Servlet API对象，且只能访问其有限的方法。

>>使用ActionContext类：ActionContext 是 Action 执行的上下文对象, 在 ActionContext 中保存了 Action 执行所需要的所有对象, 包括 parameters, request, session, application 等。

【参考struts2/index**.**jsp、TestActionContextAction**.**java、test-actionContext.jsp】

>>实现XxxAware接口

【参考struts2/index**.**jsp、TestAwareAction**.**java、test-aware**.**jsp】

【选用建议：若一个Action类中有多个action方法，且多个方法都需要使用域对象的Map或parameters，则建议使用Aware接口的方式】

1. 和Servlet API耦合的方式：可以访问更多的Servlet API对象，且可以调用其原生的方法。

>>使用ServletActionContext类：

【参考index.jsp、TestServletActionContextAcion.java、test-ServletActionContext.jsp】

>>实现ServletXxxAware接口：

【参考index.jsp、TestServletAwareAction.java、test-ServletAwareAction.jsp】

【

知识点：

1、session对应的Map实际上是SessionMap类型的，强转后若调用其invalidate()方法，可使其session失效

】

【注：最常用的方式是extends ActionSupport类并且实现XxxAware接口，即：public class ActionTest1 extends ActionSupport implements RequestAware(){...}】

Struts2请求的拓展名：

在org.apache**.**struts2包下的default**.**properties中配置了Struts2应用的一些常量。

struts**.**action**.**extension定义了当前Struts2应用可以接受的请求的拓展名。

可以在struts**.**xml文件中以常量配置的方式修改default**.**properties所配置的常量，方法如下：

|  |
| --- |
| <!-- 配置Struts2可以受理的请求的拓展名 -->  <constant name=*"struts.action.extension"* value=*"action,do,"*></constant> |

注意：

在web**.**xml中有下面的设置

|  |
| --- |
| <!-- 配置Struts2的Filter -->  <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> |

这是一个总的控制器，这表示所有的请求都可以通过总控制器，若是设置成<url-pattern>\*.do</url-pattern>置，那么就只有扩展名为 **.**do 的请求才能通过总控制器。【注意：默认是 \*.action，我自己修改成了 /\* 】

而在struts**.**xml中还可以有这样的配置：

|  |
| --- |
| <!-- 配置Struts2可以受理的请求的拓展名 -->  <constant name=*"struts.action.extension"* value=*"action,do,"*></constant> |

这里配置的常量是通过总控制器的请求(可以通过总控制器的请求在web**.**xml中配置)交给action处理的拓展名只能是 （*action,do,* ）

案例： 【查看demo1目录下文件、index**.**jsp的 <a href="login-ui.do">LoginUI</a>】

本案例实现了简易的在线人数统计，使用不同的浏览器的确可以获取不同的session对象，从而实现数量+1，但是有一个很严重的问题，那就是当不断的刷新浏览器的情况下，数量还是会+1，这就有问题。这涉及的就是如何防止表单的重复提交。

ActionSupport：

1. ActionSupport是默认的Action类，若某个action节点没有配置class属性，则ActionSupport即为待执行的Action类，而execute()方法即为要默认执行的方法。

|  |
| --- |
| <action name=*"login-ui"* method=*"execute"* class=*"com.opensymphony.xwork2.ActionSupport"*>  <result name=*"success"* type=*"dispatcher"*>/demo1/login.jsp</result>  </action> |

与下面的配置是一样的：

|  |
| --- |
| <action name=*"login-ui"*>  <result>/demo1/login.jsp</result>  </action> |

1. 在手工完成字段验证、显示错误消息、国际化等情况下，推荐extends ActionSupport

result：

1. result是action的子节点
2. Result代表action方法执行后，可能去的一个目的地
3. 一个action节点可以配置多个result子节点
4. result的name属性值对应着action方法可能有的一个返回值
5. result的type属性表示结果的响应类型【struts-default**.**xml】中定义了result-type

|  |
| --- |
| <result-types>  <result-type name=*"chain"* class=*"com.opensymphony.xwork2.ActionChainResult"*/>  <result-type name=*"dispatcher"* class=*"org.apache.struts2.dispatcher.ServletDispatcherResult"* default=*"true"*/>  <result-type name=*"freemarker"* class=*"org.apache.struts2.views.freemarker.FreemarkerResult"*/>  <result-type name=*"httpheader"* class=*"org.apache.struts2.dispatcher.HttpHeaderResult"*/>  <result-type name=*"redirect"* class=*"org.apache.struts2.dispatcher.ServletRedirectResult"*/>  <result-type name=*"redirectAction"* class=*"org.apache.struts2.dispatcher.ServletActionRedirectResult"*/>  <result-type name=*"stream"* class=*"org.apache.struts2.dispatcher.StreamResult"*/>  <result-type name=*"velocity"* class=*"org.apache.struts2.dispatcher.VelocityResult"*/>  <result-type name=*"xslt"* class=*"org.apache.struts2.views.xslt.XSLTResult"*/>  <result-type name=*"plainText"* class=*"org.apache.struts2.dispatcher.PlainTextResult"* />  </result-types> |

常用的有dispatcher(默认)、redirect、redirectAction、chain：

1. dispatcher：转发，同Servlet中的转发，只有一个请求（地址栏不变），不能转发到一个action
2. redirect：重定向，同Servlet中的重定向，有两个请求（地址栏变化），不能重定向到一个action
3. redirectAction：重定向到一个Action，有两个请求（地址栏变化）
4. chain：转发到一个Action，只有一个请求（地址栏不变）

【参考demo2目录下的文件、struts**.**xml文件】

<!--

总结：

1、使用<result name="c" type="redirect">/atguigu/testAction.do</result>、<result name="c" type="redirectAction">/atguigu/testAction.do</result>能实现一样的功能

2、使用<result name="c" type="dispatcher">/atguigu/testAction.do</result>、<result name="c" type="chain">/atguigu/testAction.do</result不能实现一样的功能。

-->

通配符映射：【参考demo3目录下的文件、struts.xml】

1. 一个web应用可能有成百上千个action声明，可以利用Struts提供的通配符映射机制把多个彼此相似的映射关系简化为一个映射关系
2. 通配符映射规则：
3. 若找到多个匹配，没有通配符(精确匹配)的那个将胜出
4. 若指定的动作不存在，Struts将会尝试把这个URI与任何一个包含着通配符\*的动作名及进行匹配
5. 被通配符匹配的URI字符串的子串可以用{1}、{2}来引用。{1}匹配第一个子串、{2}匹配第二个子串
6. {0}匹配整个URI
7. 若Struts找到的带有通配符的匹配不止一个，则按先后顺序进行匹配
8. \*可以匹配零个或多个字符，但不包括 / 字符，如果想把 / 字符包括在内，需要使用\*\*，如果需要对某个字符进行转义，需要使用 \ 。

动态方法调用：

1. 动态方法调用：通过url动态调用Action中的方法
2. 默认情况下，Struts的动态方法调用处于禁用状态

值栈：ValueStack【参考valueStack/Product.java、valueStack/Test.java、valueStack/input.jsp、valueStack/details.jsp】

1. 值栈贯穿整个Action（一个请求一个Action，每一次请求都需要创建一个值栈）的生命周期（每个Action类的对象实例都拥有一个valueStack对象），这个值栈对象就相当于一个数据的中转站，在其中保存当前Action对象和其他相关对象的相关信息。
2. 可以从ActionContext中获取值栈对象
3. Struts框架把ValueStack对象保存在名为“struts.valueStack”的请求属性中。
4. 值栈可以分为两个部分：

1）Map栈：实际上是OgnlContext类型，是个Map，也是对ActionContext的一个引用，里面保存着各种Map：parametersMap、requestMap、SessionMap、applicationMap、attr

1. ContextMap：Struts把各种各样的映射关系（一些Map类型的对象）压入ContextMap中，实际上就是ActionContext的一个引用。

Struts会把下面的这些映射压入ContextMap中

>>parameters：该Map中包含当前请求的请求参数

>>request：该Map中包含当前request对象中的所有属性

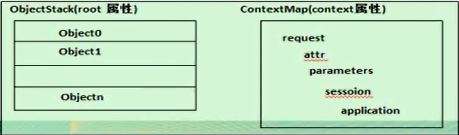
>>session：该Map中包含当前session对象的所有属性

>>application：该Map中包含当前application对象中的所有属性

>>attr：该Map按如下顺序来检索某个属性：request、session、application

2）对象栈（root）：实际上是compoundRoot类型，是一个使用ArrayList定义的栈，里面保存着各种和当前Action实例相关的对象，是一个数据结构意义的栈。

1. ObjectStack：Struts把Action和相关对象压入ObjectStack中。



1. 在写HelloWorld时，${productDesc}读取productDesc值，实际上该属性并不在request等域对象中，而是从值栈中获取的。

|  |
| --- |
|  |

5、Struts的property标签和OGNL表达式来读取值栈中的属性值

1）property标签属性：



1. 值栈中的属性值：

>>对于对象栈：指的是对象栈中某一个对象的属性值

>>对于Map栈：指的是request、session、application的一个属性值或者一个 请求参数的值

1. 读取对象栈（Object栈）里的对象的属性【参考valueStack目录下的文件】
2. 若想访问ObjectStack里的某个对象的属性，可以使用一下几种形式之一：

>>object**.**propertyName

>>object['propertyName']

>>object["propertyName"]

1. object栈里的对象可以通过一个从零开始的下标来引用

ObjectStack里的栈顶对象可以用【0】来引用，他下面的那个对象可以用【1】 来引用；若希望返回栈顶对象的message属性值，可以用【0】**.**message 或 【0】【“message”】 或【0】【‘message’】

1. 若在指定的对象里没有找到指定的属性，则到指定对象的下一个对象里继续搜索。 即【n】的含义是从第n个开始搜索，而不是只搜索到第n个对象
2. 若从栈顶对象开始搜索，则可以省略下标部分。如获取栈顶对象的message属性：

>><s:property value=*"[0].message"*/>

>><s:property value=*"message"* />

5）默认情况下，Action（Product）对象会被Struts2自动的放在值栈的栈顶

7、读取Map栈里的对象的属性

【参考valueStack/Product.java、valueStack/Test.java、valueStack/input.jsp、valueStack/details.jsp】

8、之所以能在页面上获取一个一个的属性，并不是在请求域中真的有这个属性，而是因为 从值栈里面获取的。

9、Struts2自动把Action对象放入值栈中：

1）放入的时

A）先创建一个StrutsActionProxy对象

B）在创建StrutsActionProxy对象之后，对其初始化时，把Action对象放入值 栈中。

间点为：Struts2终将调用Action类的Action方法，但在调用该方法之 前：

OGNL：Object Graph Navigator Language

对象图导航语言

1. 在JSP页面上可以利用OGNL访问到值栈（ValueStack）里的对象属性
2. 若希望访问值栈中ContextMap中的数据，需要给OGNL表达式加上一个前缀字符#，如果没有前缀字符#，搜索将在ObjectStack中进行

3、ActionContext就是将OgnlContext包装了一下，可以认为OgnlContext就是 ActionContext。

使用OGNL调用字段和方法：【参考valueStack/details**.**jsp】

1、使用OGNL调用public类的public类型的静态字段和静态方法：

1）默认不允许调用静态方法，在org**.**apache**.**struts2包下的default**.**properties文件 下有这样一个设置struts.ognl.allowStaticMethodAccess=false， 此时我们只需要将它设置为true就能调用静态方法。【注意：不是在这个文件中设置， 而是在struts**.**xml文件中设置】

2、调用对象栈的方法为一个属性赋值【参考valueStack/details**.**jsp】

OGNL访问数组类型的属性：【参考valueStack/details**.**jsp】

1. 有些属性返回一个对象数组而不是单个对象，可以像读取任何其他对象属性那样读取它 们，这种数组型属性的各个元素以逗号分隔，并且不带方括号
2. 可以使用下标访问数组中指定的元素：colors[0]
3. 可以调用其length字段查出给定数组中有多少个元素：colors**.**length

OGNL访问List类型的属性：

1. isEmpty或者isEmpty()判断给定的List是否为空

OGNL访问Map类型的属性：【参考valueStack/details**.**jsp】

1. 读取一个Map类型的属性将以如下所示的格式返回它所有的键值对



1. 若希望检索出某个Map的值，需要使用如下格式：map[key]
2. 可以使用size或者size()得出某个给定的Map的键值对的个数
3. 可以使用isEmpty或isEmpty()检查某给定的Map是不是为空
4. 可以使用如下语法来创建一个Map：



【

总结：能用EL的就一定能用OGNL，但是能够OGNL的不一定能用EL

】

声明式异常处理：【参考valueStack/exceptionTest.jsp、valueStack/exceptionPage.jsp、valueStack/ExceptionTest.java、】

1. exception-mapping元素：配置当前action的声明式异常
2. Exception-mapping元素中有两个属性
3. exception：指定需要捕获的异常类型
4. result：指定一个相应结果，该结果将在捕获到指定异常时执行，既可以来自当前 action的声明，也可以来自global-results的声明。
5. 可以通过global-exception-mappings元素为应用程序提供一个全局性的异常捕获映 射，但在global-exception-mappings元素下声明的任何exception-mapping元素 只能引用在global-results元素下声明的某个result元素
6. 声明式异常处理机制由ExceptionMappingInterceptor拦截器负责处理，当某个 exception-mapping元素声明的异常被捕获到时，ExceptionMappingInterceptor 拦截器就会向ValueStack对象栈栈顶中添加两个对象
7. exceptionStack：包含着被捕获异常的栈
8. exception对象：表示被捕获异常的Exception对象

通用标签：【参考struts2-3】

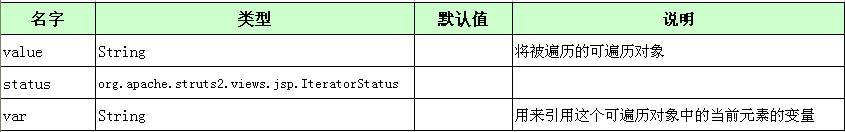
1. property标签：用来输出值栈中的属性值
2. 对于对象栈：打印值栈中对应的属性值
3. 对于Map栈：打印request、session、application的某个属性值 或 某个请求 参数的值(parameters)



1. Url标签：用来动态创建一个URL



1. set标签：用来在以下Map对象里创建一个键值对：
2. push标签：把一个对象压入ValueStack而不是压入ContextMap；push标签在起始 时把一个对象压入栈，标签结束时将对象弹出。
3. Iterator标签：用来遍历一个数组，Collection或一个Map，并把这个可遍历对象里的 每一个元素依次压入和弹出ValueStack栈。在开始执行时，Iterator标签会把 IteratorStatus类的一 个实例压入ContextMap，并在每次遍历循环时更新它，可以 将一个指向 IteratorStatus对象的变量赋值给status属性

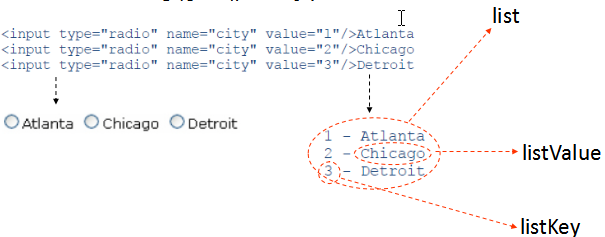


1. sort标签：用来对一个可遍历对象里的元素进行排序



Form标签：【参考struts2-3】

1. 标签的属性可以被赋值为一个静态的值或一个OGNL表达式。如果在赋值时使用了一个 OGNL表达式并把它用 %{} 括起来，这个表达式将会被求值。
2. Checkbox标签在HTML中是多选，但是在Struts2中却不是多选，在Struts2中， checkbox标签通常用于提交一个布尔值
3. List、listKey、listValue这3个属性对radio、select、checklist（多选）等标签非常 重要。



主题theme:

1、主题：为了让所有的UI标签能够产生同样的视觉效果而归集到一起的一组模板，即风 格相近的模板被打包成一个主题。

2、属性：

A）simple：把UI标签翻译成最简单的HTML对应元素，而却会忽视行标属性

B）xhtml：xhtml是默认的主题，这个主题的模板通过使用一个布局表格提供了一种 自动化的排版机制。

C）css\_xhtml: 这个主题里的模板与 xhtml 主题里的模板很相似, 但它们将使用 css 来进行布局和排版

D）ajax: 这个主题里的模板以 xhtml 主题里德模板为基础, 但增加了一些 Ajax 功 能。

1. 在显示错误消息的时候，为了消除Struts框架默认添加的ul、li标签，可以使用两种方 法：
2. 自己通过css样式控制
3. 覆盖template模板

Parameters拦截器：

Parameters 拦截器将把表单字段映射到 ValueStack 栈的栈顶对象的各个属性中. 如果某个字段在模型里没有匹配的属性, Param 拦截器将尝试 ValueStack 栈中的下一个对象

ModelDriven：（模型驱动）

1. 如果Action类实现了ModelDriven接口，该拦截器将把ModelDriven接口的 getModel()方法返回的对象置于值栈栈顶。
2. Action实现ModelDriven接口后的运行流程
3. 先执行ModelDrivenInterceptor的intercept方法

|  |
| --- |
| **public** String intercept(ActionInvocation invocation) **throws** Exception{  //获取Action对象  Object action = invocation.getAction();  //判断是否是ModelDriven的实例  **if** (action **instanceof** ModelDriven){  //强制转换为ModelDriven类型  ModelDriven modelDriven = (ModelDriven) action;  //获取值栈  ValueStack stack = invocation.getStack();  //调用ModelDriven的getModel()接口  Object model = modelDriven.getModel();  **if** (model != **null**){  //把getModel()方法的返回值压入值栈栈顶  stack.push(model);  }  **if** (**this**.refreshModelBeforeResult){  invocation.addPreResultListener(**new**  RefreshModelBeforeResult(modelDriven, model));  }  }  **return** invocation.invoke();  } |

2）接下来 Parameters 拦截器将把表单字段映射到 ValueStack 栈的栈顶对象的各 个属性中. 因为此时 ValueStack 栈的栈顶元素是刚被压入的模型(Employee)对 象, 所以该模型将被填充. 如果某个字段在模型里没有匹配的属性, Param 拦截器 将尝试 ValueStack 栈中的下一个对象

类型转换：【参考struts2-4】

1. 从一个HTML表单到一个Action对象，类型转换是从字符串到非字符串
2. 在Struts2中，把请求参数映射到action属性的工作由Parameters拦截器负责，它是 默认的defaultStack拦截器中的一员，Parameters拦截器可以自动完成字符串到基本 数据类型之间的转换。
3. 类型转换错误：
4. 如果类型转换错误：
5. 若Action类没有实现ValidationAware接口：Struts在遇到类型转换错误时 仍会继续调用其Action方法，就好像什么都没有发生一样
6. 若Action类实现了ValidationAware接口：Struts在遇到类型转换错误时， 将不会继续调用其Action方法： Struts将检查相关action元素的声明是否 包含一个name=input的result。如果有，Struts将控制权讲给那个result 元素；若没有Struts将抛出一个异常。
7. 因为ActionSupport实现了ValidationAware接口，所以我们开发的时候直 接继承ActionSupport就可以了。
8. 定义类型转换器【参考TestConversionAction.java、DateConverter.java、】
9. 为什么需要自定义的类型转换器？

因为Struts不能自动完成字符串到引用类型的转换

1. 如何定义类型转换器：
2. 开发类型转换器类：继承StrutsTypeConverter
3. 配置类型转换器：

两种方式：

①基于字段的配置：

》在字段所在的Model（可能是Action，也可能是一个JavaBean）的 包下，新建一个ModelClassName-conversion**.**properties文件。

》在该文件中输入键值对： fieldName=类型转换器的全类名

》第一次使用该转换器时创建实例（单实例）

②基于类型的配置：

》在src下新建xwork-conversion**.**properties文件

》键入： 待转换的类型=类型转换器的全类名

》在当前Struts应用被加载时创建实例

1. 复杂属性

国际化：【参考struts2-5】

1. 如何配置国际化资源文件：
2. Action范围资源文件：

在Action类文件所在的路径建立ActionName\_language\_country.properties 的文件，这是该指定的Action才能访问该资源文件。

1. 包范围资源文件：

在包的根路径下建立文件名为 package\_language\_country.properties 的属 性文件，一旦建立，处于该包下的所有 Action都可以访问该资源文件。注意： 包范围资源文件的 basename 就是package， 不是Action所在的包名。

1. 全局资源文件：
2. 命名方式： basename\_language\_country.properties
3. 在struts.xml文件中配置：

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

C）在struts.properties文件中配置

Struts.custom.i18n.resouces=basename

1. 临时指定资源文件：

<s:i18n ... />标签的name属性指定国际化资源文件

5）国际化资源文件加载顺序：离当前Action较近的将被优先加载

1. 如何在页面上和Action类中访问国际化资源文件中的value值：
2. 在Action类中，若Action实现了TextProvider接口（通过继承ActionSupport 来实现该接口），则可以调用其getText()方法来获取value值。
3. 在页面上可以使用s:text标签
4. 实现通过超链接切换语言：（将Locale对象存放在session中）
5. 关键之处在于知道Struts2框架是如何确定Locale对象的。
6. 可以通过阅读 I18N 拦截器知道

|  |
| --- |
| <a href=*"testI18n?request\_locale=en\_US"*>English</a>  <a href=*"testI18n?request\_locale=zh\_CN"*>中文</a> |

只需要上面的两行代码，就可切换中英文。

1. 具体确定Locale对象的过程：

> Struts2 使用 i18n 拦截器 处理国际化，并且将其注册在默认的拦截器栈中

> i18n拦截器在执行Action方法前，自动查找请求中一个名为 request\_locale 的参数。如果该参数存在，拦截器就将其作为参数，转换成Locale对象，并 将其设为用户默认的Locale(代表国家/语言环境)。并把其设置为 session 的 WW\_TRANS\_I18N\_LOCALE 属性

> 若 request 没有名为request\_locale 的参数，则 i18n 拦截器会从 Session 中获取 WW\_TRANS\_I18N\_LOCALE 的属性值，若该值不为空，则将该属性 值设置为浏览者的默认Locale

> 若session中的WW\_TRANS\_I18N\_LOCALE的属性值为空，从ActionContext 中获取 Locale 对象。

4）注意：超链接必须是一个Struts2的请求，即：使 i18n 拦截器工作。

Struts2验证：【参考struts2-7】

1. Struts2验证方式有两种
2. 声明式验证：
3. 对哪个Action或者Model的哪个字段进行验证
4. 使用什么验证规则
5. 如果验证失败，转向哪一个页面，显示什么错误消息
6. 编程式验证：Action类中重写validate()方法
7. 声明式验证：【参考validation.jsp、TestValidationAction.java、success.jsp、TestValidationAction-testValidation-validation.xml、】
8. 字段验证：判断某个字段属性的输入是否有效
9. 非字段验证：不只针对某个字段，而是针对多个字段的输入值之间的逻辑关系进行 校验。例如对再次输入密码确认的判断。
10. 声明式验证框架的原理：
11. Struts2的默认拦截器中提供了一个validation拦截器
12. 每个具体的验证规则都会对应具体的验证器。有一个配置文件把验证规则和验证器 关联起来。而实际上验证的是那个验证器。该文件位于 com.opensymphony.xwork2.validator.validators下的default**.**xml

4、注意：若一个Action类可以应答多个action请求，每个action请求又是使用不同的 验证规则，怎么办？

》为每一个不同的action请求定义其对应的验证文件：

ActionClassName**-**别名**-**validation**.**xml

》不带别名的配置文件：ActionClassName-validation**.**xml中的验证规则依然会起作 用。可以把各个action共有的验证规则配置在其中，但需要注意的是，只适用于 某一个action请求的验证规则就不要配置在这里了。

【参考validation.jsp、validation2.jsp、TestValidationAction.java、success.jsp、TestValidationAction-testValidation-validation.xml、TestValidationAction-testValidation2-validation.xml、】

1. Struts2验证框架验证规则
2. required：确保某给定字段的值不是空值null
3. requiredstring：确保某给定字段的值既不是空值null，也不是空白

参数（空白字符串）：

trim参数：默认为true，表示Struts在验证该字段之前先去除前后空格

1. stringlength：验证一个非空的字段值是不是有足够的长度

参数：

minLength：相关字段的最小长度。若没有给出这个参数，该字段将没有最小长 度限制

maxLength：相关字段的最大长度。若没有给出这个参数，该字段将没有最大长 度限制

trim：在验证之前是否去掉前后空格

1. date：确保某给定日期字段的值落在一个给定的范围内。

参数：

max：相关字段的最大值。若没有给出这个参数，该字段将没有最大长度限制

min：相关字段的最小值。若没有给出这个参数，该字段将没有最小长度限制

1. email：检查给定String值是否是一个合法的email
2. url：检查给定的String值是否是一个合法的url
3. regex：检查某给定字段的值是否与一个给定的正则表达式模式匹配

参数：

expression\*：用来匹配的正则表达式

caseSensitive：是否忽视字母的大小写，默认为true

trim：是否除去前后空格

1. int：检查给定整数字段值是否在某一个范围内

max：相关字段的最大值。若没有给出这个参数，该字段将没有最大长度限制

min：相关字段的最小值。若没有给出这个参数，该字段将没有最小长度限制

1. conversion：检查给定Action属性进行的类型转换是否会导致一个转换错误。该 验证器还可以在默认的类型转换消息的基础上添加一条自定义的消息。
2. expression和fieldExpression：用来验证一个给定字段是否满足一个OGNL表 达式。

①前者是一个非字段验证程序，后者是一个字段验证程序

②前者是在验证失败时将生成一个action错误，而后者在验证失败时会生成一个 字段错误。

③expression\*：用来进行验证的OGNL表达式

1. 短路验证器：

1）若对一个字段使用多个验证器，默认情况下会执行所有的验证，哪怕前面的验证器 验证不通过，后面的验证器仍会执行；如是想改变这种情况，即前面的验证没有通过的 时候，后面的验证不再执行，那这时就需要使用短路验证。

2）<validator .../>元素和<field-validator ... />元素可以指定一个可选的 short-circuit属性。该属性指定该验证器是否是短路验证器，默认值是false。

3）对同一个字段内的多个验证器，如果一个短路验证器失败，其他验证器不会继续校 验。

7、若类型转换（params拦截器）失败，默认情况下还会执行后面的拦截器（conversionError 拦截器、validation拦截器、workflow拦截器），因此仍会继续进行调用validator

拦截器进行验证。可以通过修改 ConversionErrorInterceptor 源代码的方式使当类型

转换失败时，不再执行后续的拦截器，而直接返回input的result

|  |
| --- |
|  |
| 上面的第1个错误“Invalid field value for field "age".”其实就是params拦截器调用后的类型转换错误；而第2个错误“age类型转换错误”才是validator拦截器调用后的验证错误。（这是我们自己定义的错误信息） |
| 我们的目标就是params拦截器进行类型转换错误后，不再调用后面的conversionError拦截器。 |

|  |
| --- |
| 下面是默认执行的拦截器栈，每一个Action都会顺序的调用下面的拦截器 |
| <interceptor-stack name=*"defaultStack"*>  <interceptor-ref name=*"exception"*/>  <interceptor-ref name=*"alias"*/>  <interceptor-ref name=*"servletConfig"*/>  <interceptor-ref name=*"i18n"*/>  <interceptor-ref name=*"prepare"*/>  <interceptor-ref name=*"chain"*/>  <interceptor-ref name=*"debugging"*/>  <interceptor-ref name=*"scopedModelDriven"*/>  <interceptor-ref name=*"modelDriven"*/>  <interceptor-ref name=*"fileUpload"*/>  <interceptor-ref name=*"checkbox"*/>  <interceptor-ref name=*"multiselect"*/>  <interceptor-ref name=*"staticParams"*/>  <interceptor-ref name=*"actionMappingParams"*/>  <interceptor-ref name=*"params"*>  <param name=*"excludeParams"*>dojo\..\*,^struts\..\*</param>  </interceptor-ref>  <interceptor-ref name=*"conversionError"*/>  <interceptor-ref name=*"validation"*>  <param name=*"excludeMethods"*>input,back,cancel,browse</param>  </interceptor-ref>  <interceptor-ref name=*"workflow"*>  <param name=*"excludeMethods"*>input,back,cancel,browse</param>  </interceptor-ref>  </interceptor-stack> |

1. 非字段验证：不是针对于某一个字段的验证
2. 代码：

|  |
| --- |
| <!-- 非字段验证 -->  <!-- 要求：password与confirm一致 -->  <validator type=*"expression"*>  <param name=*"expression"*><![CDATA[password==confirm]]></param>  <message>password is not equals confirm</message>  </validator> |

1. 如何显示错误消息：使用<s:actionerror />标签【因为它是action的错误，而不是field的错误。】

【参考validation3.jsp、success.jsp、TestValidationAction3.java、TestValidationAction3-testValidation3-validation.xml】

1. 自定义验证器：
2. 怎么自定义验证器？

①定义一个验证器类

》自定义的验证器都需要实现Validator接口

》可以选择继承ValidatorSupport或者FieldValidatorSupport类

》若希望实现一个一般的验证器，则可以继承ValidatorSupport

》若希望实现一个字段验证器，则可以继承FieldValidatorSupport

②在配置文件中配置验证器

》默认情况下，Struts2会在类路径的根目录下加载validators**.**xml文件。在 该文件中加载验证器，该文件的定义方式同默认的验证器的那个配置文 件：位于com.opensymphony.xwork2.validator.validatos下的 default.xml。

》若类路径下没有指定的验证器，则从 com.opensymphony.xwork2.validator.validatos下的default.xml中的验 证器加载

1. 案例：自顶一个验证18位身份证的验证器

【参考IDCard**.**java、IDCardValidator**.**java、validation4.java、Validators.xml、TestValidationAction4-testValidation4-validation.xml、TestValidationAction4.java】

错误消息的重用：

1、多个字段使用同样的验证规则，可以使用同一条验证消息

Struts文件上传：【参考struts2-8】

1. Struts2进行文件上传使用了fileUpload拦截器
2. 一个表单提交的file文件域需要在提交的action中对应的三个属性字段

在upload.jsp文件提交的form表单中有下面的一个文件上传域

|  |
| --- |
| <s:file name=*"ppt"* label=*"PPTFile"*></s:file> |

需要在TestUploadAction.jsp中对应下面的三个属性

|  |
| --- |
| **private** File ppt; //文件对应的file对象  **private** String pptContentType; //文件类型  **private** String pptFileName; //文件名 |

【注：是一个file上传域需要对应三个属性字段】

【参考 upload.jsp、TestUploadAction.java、success.jsp】

1. 使用IO流上传到指定的文件夹
2. 一次上传多个文件怎么办？

若上传多个文件，则上述的3个属性可以改为List类型。注意：多个文件的name 需要一致。

【参考 upload2.jsp、TestUpload2.java、success.jsp】

1. 可以对上传的文件进行限制吗？例如拓展名、内容类型、文件大小？如果可以，若 出错，显示什么错误消息？消息可以定制吗？

可以通过配置FileUploadInterceptor拦截器的参数的方式来对上传的文件限制。

1. maximumSize：单个文件上传的最大大小，默认值是2M
2. allowedTypes：允许上传的类型，使用逗号分隔
3. allowedExtensions：允许上传文件的拓展名，多个使用逗号分隔

可以在国际化资源文件中定义如下如下的消息来定制错误消息

1. Struts.messages.error.uploading：文件上传出错的消息
2. Struts.messages.error.file.too.lager：文件超过最大值的消息
3. Struts.messages.error.content.type.not.allowed：文件内容类型不合法消息
4. Struts.messages.error.file.extension.not.allowed：文件拓展名不合法消息

可以参考org.apache.struts2下的struts-messages.properties,可以提供更多的定制信息

【特别注意：在org.apache.struts2下的default.properties文件中还有下面的一个配置struts.multipart.maxSize=2097152这个配置规定了上传文件的总大小不能超过2M，如果想要改变，要做下面的两种配置。】

|  |
| --- |
| <interceptor-ref name=*"defaultStack"*>  <!-- 单个文件上传的最大大小,默认值是2M,这里指定5M,即 5242880byte-->  <param name=*"fileUpload.maximumSize"*>5242880</param>  <!-- 上传的文件的内容类型 -->  <param name=*"fileUpload.allowedTypes"*>text/html,text/xml,application/msword</param>  <!-- 上传文件的拓展名 -->  <param name=*"fileUpload.allowedExtensions"*>html,xml</param>  </interceptor-ref> |
| <constant name=*"struts.multipart.maxSize"* value=*"5242880"*></constant> |

Struts2文件下载：【参考struts2-8】

1. Struts专门为下载提供了一种Stream结果类型，在使用一个Stream结果时，不 必准备一个JSP页面。
2. Stream类型可以设置如下参数：
3. contentType：被下载文件的MIME类型。默认值为text/plain
4. contentLength：被下载文件的大小，以字节为单位
5. contentDisposition：可以设置下载文件名的contentDisposition响应头， 默认值为inline，通常设置为如下格式：attachment：filename=“XXX”
6. inputName：Action中提供的文件的输入流。默认值为inputStream
7. bufferSize：文件下载时缓冲区的大小。默认值为1024
8. allowCaching：文件下载时是否允许使用缓存。默认值为true
9. contentCharSet：文件下载时的字符编码（注意这里s是大写）

【注：1）、2）、3）是动态声明的，需要在Action中定义成员变量】

3、Stream结果类型的参数可以在Action中以getter方法的方式覆盖。

表单的重复提交：【参考struts2-9】

1、Struts2解决表单重复提交问题：

1. 在s:form中添加s:token子标签

①生成一个隐藏域

②在session中添加一个属性值

注：隐藏域的值和session的属性值是一致的。

1. 使用Token或TokenSession拦截器

①这两个拦截器都不在defaultStack拦截器栈中。所以需要手工配置一下

②若使用Token拦截器，则需要配置一个invalid.token的result

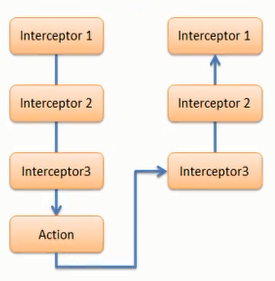
③若使用TokenSession拦截器，则不需要配置任何其他的result

1. Token拦截器 与 TokenSession拦截器的区别
2. 都是解决表单重复提交的问题
3. 使用Token拦截器，会转到invalid.token的result
4. 使用TokenSession拦截器还会响应那个目标页面，但不会执行 TokenSession 的后续拦截器，就像什么也没发生一样。

自定义拦截器：【参考struts2-9】

1. 拦截器是Struts2的核心组成部分
2. Struts2很多功能都是构建在拦截器基础之上的，例如文件上传和下载、国际化、 数据类型转换和数据校验等。
3. Struts2拦截器在访问某个Action方法之前或之后实施拦截
4. Struts2拦截器是可插拔的，拦截器是AOP（面向切面编程）的一种实现
5. 拦截器栈：将拦截器按一定的顺序联结成一条链，在访问被拦截的方法时，Struts2 拦截器链中的拦截器就会按其之前定义的顺序依次调用。

6、拦截器执行顺序：



1. Interceptor接口
2. 每个拦截器都直接或间接的实现了Interceptor接口。
3. Struts会依次调用某个Action而注册的每一个拦截器的intercept方法
4. 每次调用intercept方法时，Struts会传递一个ActionInvocation接口的实例。
5. ActionInvocation：代表一个给定Action的执行状态，拦截器可以从该类的 对象里获得与该Action相关联的Action对象和Result对象。在完成拦截器 自己的任务之后，拦截器将调用ActionInvocation的invoke()方法前进到 Action处理流程的下一个环节。
6. AbstractInterceptor类实现了Interceptor接口，并为init、destroy提供了 一个空白的实现。
7. Interceptor接口的方法：
8. init()方法：该方法将在拦截器被创建后立即被调用，它在拦截器的生命周期内 只被调用一次。可以在该方法中对相关资源进行必要的初始化。
9. intercept()方法：每拦截一个请求，该方法就会被调用一次。
10. destroy()方法：该方法将在拦截器被销毁之前被调用，他在拦截器的生命周期 内也只被调用一次。
11. 自定义拦截器：
12. 步骤：

①定义一个拦截器的类：可以实现Interceptor接口，也可以继承AbstractInterceptor类。

②在struts.xml文件中配置

2）案例：【参考MyInterceptor.java、struts.xml、】

3）注意：在自定义的拦截器中可以选择不调用ActionInvocation的invoke()方 法，那么后续的拦截器和Action方法不会被调用。Struts渲染自定义拦截器 intercept()方法返回值对应的result