# 类型转换

## 概述

从一个HTML表单到一个动作对象,类型转换将是从字符串到非字符串.因为HTTP没有“类型”的概念，所以一切表单输入都将以请求参数的形式被发送到服务器，而每一项表单输入之可能是一个String或一个String数组。在服务器端，必须先把这些String值转换为特定的数据类型，才能进行相应的处理。

把请求参数映射到动作属性的工作由Parameters拦截器负责，它是defaultStack拦截器栈的一员。所有的请求参数都是String类型，但并非所有的动作属性都是String类型，所以每一种非String类型的动作属性需要对相关的请求参数进行类型转换。

## struts2提供的转换解决方案

Struts2在类型转换失败时会发生错误，具体如何处理取决于你的动作类是否实现了com.opensymphony.xwork2.ValidationAware接口。

1. 没有实现该接口：struts在遇到类型转换错误时仍会继续调用其动作方法，就好像什么问题都没有发生过那样。
2. 如果实现了该接口：struts在遇到类型转换错误时不会继续调用其动作方法，它将检查相关动作元素中是否包含着一个input结果。如果是，struts将把控制权转交给那个result元素；如果找不到这样的结果，struts将抛出一个异常。

注：如果你的动作类继承自com.opensymphony.xwork2.ActionSupport，就相当于你间接实现了com.opensymphony.xwork2.ValidationAware接口。

## 类型转换错误消息的定制

conversionError拦截器负责添加与类型转换有关的出错消息和保存各请求参数的原始值，它是defaultStack中的一员。使用该拦截器的前提是你实现了com.opensymphony.xwork2.ValidationAware接口。

如果你用来呈现这个字段的标签使用的不是simple主题，有非法值的字段将导致一条有着以下格式的出错消息：

Invalid field value for field "*fieldName*".

我们可以通过以下方式来改变该消息：

1. 在动作类所在的包中创建一文件，文件的命名规则是*ClassName*.properties。
2. 在该文件中增加以下内容：invalid.fieldvalue.*fieldName*=Your message。*fieldName*是你动作类中队应的字段。

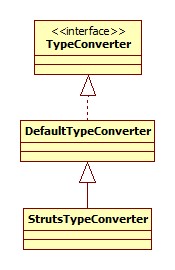
## 示例

## 自定义类型转换器

### TypeConverter接口

struts2内建的类型转换器并不能解决所有的问题，那就需要自己创建类型转换器。

自定义的类型转换器必须实现ognl.TypeConverter接口或对这个接口的某种具体实现做进一步扩展。



TypeConverter接口只有一个名为convertValue的方法，它的方法签名如下：

public Object convertValue(Map<String, Object> context, Object target, Member member, String propertyName, Object value, Class toType);

参数：

context：将在其中进行类型转换的OGNL上下文环境。

target：将在其中对有关属性进行设置的目标对象。

member：将被设置的类成员的名字。

propertyName：将被设置的属性的名字。

value：将被转换的值。

toType：转换结果的类型。

### DefaultTypeConverter

与自行实现TypeConverter接口相比，对该类扩展更容易一些，他是对TypeConverter金额接口的一种默认实现类，且他有简单的方法签名。

**public** Object convertValue(Map<String, Object> context, Object value, Class toType) {

**return** convertValue(value, toType);

}

**public** Object convertValue(Map<String, Object> context, Object target, Member member,

String propertyName, Object value, Class toType) {

**return** convertValue(context, value, toType);

}

### 自定义类型转换器的配置

在使用一个自定义的类型转换器之前，必须先对它进行配置。这种配置可以基于字段，也可以基于类。

#### 基于字段：

在动作类所在的包中创建*ActionClass*-conversion.properties的文件，其中的内容可能为下所示：

*fieldname*=customConverter1

#### 基于类：

在WEB-INF/classes子目录下创建一个conversion.properties文件，其中的内容可能为下所示：

*fullQualifieldClassName*=customerConverter1

## 与复杂对象的配合使用

### 与Collection配合使用

### 与Map配合使用

# 验证

Strut2验证可以通过一个XML配置文件和注解的方式来实现，当然手工验证（编码验证）也是支持的。同时也可以通过XML和注解共同使用的方式实现联合验证。

Struts2的验证是通过validation和workflow拦截器实现的，它们都属于default interceptor stack。validation拦截器用于验证并组织错误消息。workflow拦截器用于检测是否包含错误消息，假如有，它将返回结果为input所指向的页面，并将错误消息和原先输入的数据一同呈现给客户。

如果您的程序中使用了默认的验证（或者转换器）而没有提供为input的结果，则将出现错误。

## 注解验证

注解验证从struts2.1版本开始就不建议使用了。

## 示例

### 基础验证Basic Validation

让我们一步一步做一个基础验证的示例

1. 步骤一：创建输入表单

|  |
| --- |
| **create.jsp** |
| <body>  <s:form action="helloValidation">  <s:textfield name="name" label="姓名"></s:textfield>  <s:textfield name="age" label="年龄"></s:textfield>  <s:textfield name="address" label="籍贯"></s:textfield>  <s:submit value="提交"></s:submit>  </s:form>  </body> |

1. 步骤二：创建动作类

|  |
| --- |
| **HelloAction.java** |
| package cn.wzhting;  import com.opensymphony.xwork2.ActionSupport;  public class HelloAction extends ActionSupport {  private static final long serialVersionUID = 117358005790515177L;  private String name;  private Integer age;  private String address;  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }    public Integer getAge() {  return age;  }  public void setAge(Integer age) {  this.age = age;  }  public String getAddress() {  return address;  }  public void setAddress(String address) {  this.address = address;  }    } |

1. 步骤三：创建验证器。验证配置文件必须是以下两种形式之一

* <ActionClassName>-validation.xml
* <ActionClassName>-<ActionAliasName>-validation.xml

|  |
| --- |
| **HelloAction-validation.xml** |
| <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE validators PUBLIC  "-//OpenSymphony Group//XWork Validator 1.0.2//EN"  "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">  <validators>  <field name="name">  <field-validator type="requiredstring">  <message>请输入姓名</message>  </field-validator>  </field>  <field name="age">  <field-validator type="int">  <param name="min">13</param>  <param name="max">19</param>  <message>年龄为13~19周岁的才允许填写</message>  </field-validator>  </field>  </validators> |

1. 步骤四：请确认你的struts.xml文件中该动作有input的结果。

|  |
| --- |
| **struts.xml** |
| …  <action name="helloValidation" class="cn.wzhting.HelloAction">  <result name="success">/createConfirm.jsp</result>  <result name="error">/error.jsp</result>  <result name="input">/create.jsp</result>  </action>  … |

假如你没有这样设置，你将会得到“No result defined for action \*\*\* and result input”的错误提示。

### 客户端验证Client-side Validation

让我们一步一步做一个客户端验证的示例

* <s:form>标签的validate设置为true;
* 某些主题（themes）不支持客户端验证；

步骤一：创建输入表单

|  |
| --- |
| **create.jsp** |
| <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />  <title>Validation - Basic</title>  <s:head/>  </head>  <body>  <s:form action="helloValidation.action" validate="true">  <s:textfield name="name" label="姓名"></s:textfield>  <s:textfield name="age" label="年龄"></s:textfield>  <s:textfield name="address" label="籍贯"></s:textfield>  <s:submit value="提交"></s:submit>  </s:form>  </body> |

注意：

* 虽然使用了<s:head/>标签，此处我们只是利用其默认样式。
* 虽然struts2中的动作带不带action后缀效果一样，但是在此处最好加上action后缀，不然会报错。

步骤二、三、四通《基础验证 Basic Validation》一节，此处省略。

#### 动作和命名空间(Action and namespace)

如果表单提交到的动作不在默认命名空间里，在使用<s:form>时必须指定其namespace属性。例如，helloValidation在命名空间/ns内，可能的struts.xml如下：

|  |
| --- |
| **struts.xml** |
| …  <package name="p1" extends="struts-default" namespace="/ns">  <action name="helloValidation" class="cn.wzhting.HelloAction">  <result name="success">/createConfirm.jsp</result>  <result name="error">/error.jsp</result>  <result name="input">/create.jsp</result>  </action>  </package>  … |

输入表单如下：

|  |
| --- |
| **create.jsp** |
| <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />  <title>Validation - Basic</title>  <s:head/>  </head>  <body>  <s:form action="helloValidation.action" validate="true" namespace="/ns">  <s:textfield name="name" label="姓名"></s:textfield>  <s:textfield name="age" label="年龄"></s:textfield>  <s:textfield name="address" label="籍贯"></s:textfield>  <s:submit value="提交"></s:submit>  </s:form>  </body> |

看上去应该能正常运行，客户端验证将不能。struts必须准确的知道动作所在的命名空间（不是通过URL），因此正确的写法如下：

|  |
| --- |
| **create.jsp** |
| <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />  <title>Validation - Basic</title>  <s:head/>  </head>  <body>  <s:form action="/ns/helloValidation.action" validate="true">  <s:textfield name="name" label="姓名"></s:textfield>  <s:textfield name="age" label="年龄"></s:textfield>  <s:textfield name="address" label="籍贯"></s:textfield>  <s:submit value="提交"></s:submit>  </s:form>  </body> |

## 内建验证器

struts2为我们共内置了16个验证器，且全部是基于字段的验证器。

### required validator

#### 功能

用来验证某个给定的字段的值不是null。注意，空字符串不是null。

#### 参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名 | 类型 | 默认值 | 必须的 | 描述 |
| fieldName | String |  | no | 要验证的字段名 |

（用法见后面的说明）

#### 示例

页面：

<s:fielderror/>

<s:form action=*"validate"*>

<s:textfield name=*"userName"* label=*"用户名"*></s:textfield>

<s:submit value=*"登录"*></s:submit>

</s:form>

动作类:

**import** com.opensymphony.xwork2.ActionSupport;

**public** **class** ValidationAction **extends** ActionSupport {

**private** **static** **final** **long** *serialVersionUID* = 6877330242746547448L;

**private** String userName;

**private** String password;

**public** String getUserName() {

**return** userName;

}

**public** **void** setUserName(String userName) {

**this**.userName = userName;

}

**public** String getPassword() {

**return** password;

}

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

**this**.password = password;

}

}

验证配置文件:

<validators>

<field name=*"password"*>

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

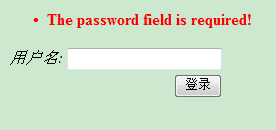
<message>The password field is required!</message>

</field-validator>

</field>

</validators>

运行结果:



#### 说明

验证配置文件的另外一种写法：

<validators>

<validator type=*"required"*>

<param name=*"fieldName"*>password</param>

<message>The password field is required!</message>

</validator>

</validators>

### requiredstring validator

#### 功能

验证给定的字段的值既不是null、也不是空白。

#### 参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名 | 类型 | 默认值 | 必须的 | 描述 |
| fieldName | String |  | no | 要验证的字段名 |
| trim | Boolean | true | no | 验证前是否要去掉前导和尾缀的空白字符 |

#### 示例

页面：

<s:form action=*"validate"*>

<s:textfield name=*"userName"* label=*"用户名"* required=*"true"* requiredposition=*"left"*></s:textfield>

<s:password name=*"password"* label=*"密码"* required=*"true"* requiredposition=*"left"*></s:password>

<s:submit value=*"登录"*></s:submit>

</s:form>

动作类:

**import** com.opensymphony.xwork2.ActionSupport;

**public** **class** ValidationAction **extends** ActionSupport {

**private** **static** **final** **long** *serialVersionUID* = 6877330242746547448L;

**private** String userName;

**private** String password;

**public** String getUserName() {

**return** userName;

}

**public** **void** setUserName(String userName) {

**this**.userName = userName;

}

**public** String getPassword() {

**return** password;

}

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

**this**.password = password;

}

}

验证配置文件:

<validators>

<field name=*"userName"*>

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

<message>Please input the userName!</message>

</field-validator>

</field>

<field name=*"password"*>

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

<param name=*"trim"*>false</param>

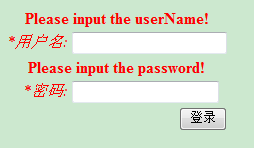
<message>Please input the password!</message>

</field-validator>

</field>

</validators>

运行结果:



#### 说明

验证配置文件的另外一种写法：

<validators>

<validator type=*"requiredstring"*>

<param name=*"fieldName"*>userName</param>

<message>Please input the userName!</message>

</validator>

<validator type=*"requiredstring"*>

<param name=*"fieldName"*>password</param>

<param name=*"trim"*>false</param>

<message>Please input the password!</message>

</validator>

</validators>

### int validator

#### 功能

用来验证某个字段的值是否可以被转换为一个整数。若指定参数，还验证是否在允许的范围内。

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 默认值 | 描述 |
| fieldName | String |  | 要验证的字段名 |
| min | Integer |  | 允许的最小值。若没有给出该参数则无限制 |
| max | Integer |  | 允许的最大值。若没有给出该参数则无限制 |

#### 示例

页面：

<s:form action=*"validate"*>

<s:textfield name=*"age"* label=*"年龄"*></s:textfield>

<s:submit value=*"登录"*></s:submit>

</s:form>

动作类:

**private** Integer age;

**public** Integer getAge() {

**return** age;

}

**public** **void** setAge(Integer age) {

**this**.age = age;

}

验证配置文件:

<validators>

<field name=*"age"*>

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

<param name=*"min"*>18</param>

<param name=*"max"*>60</param>

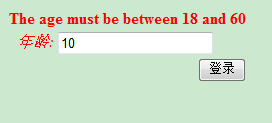
<message>The age must be between ${min} and ${max}</message>

</field-validator>

</field>

</validators>

运行结果:



#### 说明

验证配置文件的另外一种写法：

<validators>

<validator type=*"int"*>

<param name=*"fieldName"*>age</param>

<param name=*"min"*>18</param>

<param name=*"max"*>60</param>

<message>The age must be between ${min} and ${max}</message>

</validator>

</validators>

### long validator

#### 功能

用来验证某个字段的值是否可以被转换为一个长整数。若指定参数，还验证是否在允许的范围内。

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 默认值 | 描述 |
| fieldName | String |  | 要验证的字段名 |
| min | Long |  | 允许的最小值。若没有给出该参数则无限制 |
| max | Long |  | 允许的最大值。若没有给出该参数则无限制 |

#### 示例

页面：

动作类:

验证配置文件:

运行结果:

#### 说明

### short validator

#### 功能

用来验证某个字段的值是否可以被转换为一个短整数。若指定参数，还验证是否在允许的范围内。

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 默认值 | 描述 |
| fieldName | String |  | 要验证的字段名 |
| min | Short |  | 允许的最小值。若没有给出该参数则无限制 |
| max | Short |  | 允许的最大值。若没有给出该参数则无限制 |

#### 示例

页面：

动作类:

验证配置文件:

运行结果:

#### 说明

### double validator

#### 功能

用来验证某个字段的值是否可以被转换为一个双精度浮点数。若指定参数，还验证是否在允许的范围内。

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 默认值 | 描述 |
| fieldName | String |  | 要验证的字段名 |
| minInclusive | Double |  | 允许的最小值。若没有给出该参数则无限制（含最小值） |
| maxInclusive | Double |  | 允许的最大值。若没有给出该参数则无限制（含最大值） |
| minExclusive | Double |  | 允许的最小值。若没有给出该参数则无限制（不含最小值） |
| maxExclusive | Double |  | 允许的最大值。若没有给出该参数则无限制（不含最大值） |

#### 示例

页面：

<s:form action=*"validate"*>

<s:textfield name=*"percentage1"* label=*"百分比1"*></s:textfield>

<s:textfield name=*"percentage2"* label=*"百分比2"*></s:textfield>

<s:submit value=*"保存"*></s:submit>

</s:form>

动作类:

**private** Double percentage1;

**private** Double percentage2;

**public** Double getPercentage1() {

**return** percentage1;

}

**public** **void** setPercentage1(Double percentage1) {

**this**.percentage1 = percentage1;

}

**public** Double getPercentage2() {

**return** percentage2;

}

**public** **void** setPercentage2(Double percentage2) {

**this**.percentage2 = percentage2;

}

验证配置文件:

<validators>

<field name=*"percentage1"*>

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

<param name=*"minInclusive"*>20.1</param>

<param name=*"maxInclusive"*>50.1</param>

<message> The age must be between ${ *minInclusive* } and ${ *maxInclusive* }(含)</message>

</field-validator>

</field>

<field name=*"percentage2"*>

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

<param name=*"minExclusive"*>0.345</param>

<param name=*"maxExclusive"*>99.987</param>

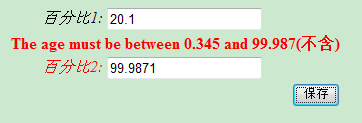
<message> The age must be between ${ *minExclusive* } and ${ *maxExclusive* }(不含)</message>

</field-validator>

</field>

</validators>

运行结果:



#### 说明

验证配置文件的另外一种写法：

<validators>

<validator type=*"double"*>

<param name=*"fieldName"*>percentage1</param>

<param name=*"minInclusive"*>20.1</param>

<param name=*"maxInclusive"*>50.1</param>

<message> The age must be between ${ minInclusive } and ${ maxInclusive }(含)</message>

</validator>

<validator type=*"double"*>

<param name=*"fieldName"*>percentage2</param>

<param name=*"minExclusive"*>0.345</param>

<param name=*"maxExclusive"*>99.987</param>

<message> The age must be between ${ minExclusive } and ${ maxExclusive }(不含)</message>

</validator>

</validators>

### date validator

#### 功能

用来确保给定的日期字段的值落在一个给定的范围内。

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 默认值 | 描述 |
| fieldName | String |  | 要验证的字段名 |
| min | java.util.Date |  | 允许的最小值。若没有给出该参数则无限制（含最小值） |
| max | java.util.Date |  | 允许的最大值。若没有给出该参数则无限制（含最大值） |

#### 示例

页面：

<s:form action=*"validate"*>

<s:textfield name=*"birthday"* label=*"出生日期"*></s:textfield>

<s:submit value=*"保存"*></s:submit>

</s:form>

动作类:

**private** Date birthday;

**public** Date getBirthday() {

**return** birthday;

}

**public** **void** setBirthday(Date birthday) {

**this**.birthday = birthday;

}

验证配置文件:

<validators>

<field name=*"birthday"*>

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

<param name=*"min"*>2011-01-01</param>

<param name=*"max"*>2011-12-31</param>

<message>日期必须为2011年</message>

</field-validator>

</field>

</validators>

运行结果:



#### 说明

验证配置文件的另外一种写法：

<validators>

<validator type=*"date"*>

<param name=*"fieldName"*>birthday</param>

<param name=*"min"*>2011-01-01</param>

<param name=*"max"*>2011-12-31</param>

<message>日期必须为2011年</message>

</validator>

</validators>

### expression validator

#### 功能

用于验证是否满足一个OGNL表达式。这是一个非字段的验证。只有给定的参数的返回值是true时才能验证通过。验证不通过时产生一个动作错误，因此要显示该错误，需要使用<s:actionerror/>标签。

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 默认值 | 描述 |
| expression | String |  | OGNL表达式，只有该表达式为true才能验证通过 |

#### 示例

页面：

<s:actionerror/>

<s:form action=*"validate"*>

<s:textfield name=*"minNumber"* label=*"最小值"*></s:textfield>

<s:textfield name=*"maxNumber"* label=*"最大值"*></s:textfield>

<s:submit value=*"保存"*></s:submit>

</s:form>

动作类:

**private** Integer minNumber;

**private** Integer maxNumber;

**public** Integer getMinNumber() {

**return** minNumber;

}

**public** **void** setMinNumber(Integer minNumber) {

**this**.minNumber = minNumber;

}

**public** Integer getMaxNumber() {

**return** maxNumber;

}

**public** **void** setMaxNumber(Integer maxNumber) {

**this**.maxNumber = maxNumber;

}

验证配置文件:

<validators>

<validator type=*"expression"*>

<param name=*"expression"*>

maxNumber>minNumber

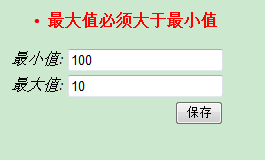
</param>

<message>最大值必须大于最小值</message>

</validator>

</validators>

运行结果:



#### 说明

该验证器没有字段形式的写法。要进行字段验证，请使用fieldexpression验证器。

### fieldexpression validator

#### 功能

用于验证某个字段是否满足一个OGNL表达式。这是一个基于字段的验证。只有给定的参数的返回值是true时才能验证通过。验证不通过时产生一个字段错误。

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 默认值 | 描述 |
| fieldName | String |  | 要验证的字段名 |
| expression | String |  | OGNL表达式，只有该表达式为true才能验证通过 |

#### 示例

页面：

<s:form action=*"validate"*>

<s:textfield name=*"minNumber"* label=*"最小值"*></s:textfield>

<s:textfield name=*"maxNumber"* label=*"最大值"*></s:textfield>

<s:submit value=*"保存"*></s:submit>

</s:form>

动作类:

**private** Integer minNumber;

**private** Integer maxNumber;

**public** Integer getMinNumber() {

**return** minNumber;

}

**public** **void** setMinNumber(Integer minNumber) {

**this**.minNumber = minNumber;

}

**public** Integer getMaxNumber() {

**return** maxNumber;

}

**public** **void** setMaxNumber(Integer maxNumber) {

**this**.maxNumber = maxNumber;

}

验证配置文件:

<validators>

<field name=*"maxNumber"*>

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

<param name=*"expression"*>

maxNumber>minNumber

</param>

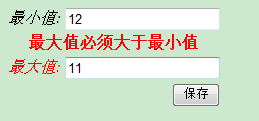
<message>最大值必须大于最小值1</message>

</field-validator>

</field>

</validators>

运行结果:



#### 说明

验证配置文件的另外一种写法：

<validators>

<validator type=*"fieldexpression"*>

<param name=*"fieldName"*>maxNumber</param>

<param name=*"expression"*>

maxNumber>minNumber

</param>

<message>最大值必须大于最小值</message>

</validator>

</validators>

### email validator

#### 功能

用来验证给定的字段是否符合一个Email的规范。它的正则表达式为

\\b(^[\_A-Za-z0-9-](\\.[\_A-Za-z0-9-])\*@([A-Za-z0-9-])+((\\.com)|(\\.net)|(\\.org)|(\\.info)|(\\.edu)|(\\.mil)|(\\.gov)|(\\.biz)|(\\.ws)|(\\.us)|(\\.tv)|(\\.cc)|(\\.aero)|(\\.arpa)|(\\.coop)|(\\.int)|(\\.jobs)|(\\.museum)|(\\.name)|(\\.pro)|(\\.travel)|(\\.nato)|(\\..{2,3})|(\\..{2,3}\\..{2,3}))$)\\b

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 默认值 | 描述 |
| fieldName | String |  | 要验证的字段名 |

#### 示例

页面：

<s:form action=*"validate"*>

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

<s:submit value=*"保存"*></s:submit>

</s:form>

动作类:

**private** String email;

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

验证配置文件:

<validators>

<field name=*"email"*>

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

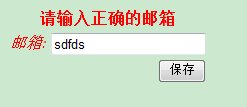
<message>请输入正确的邮箱</message>

</field-validator>

</field>

</validators>

运行结果:



#### 说明

验证配置文件的另外一种写法：

<validators>

<validator type=*"email"*>

<param name=*"fieldName"*>email</param>

<message>请输入正确的邮箱</message>

</validator>

</validators>

### url validator

#### 功能

用来验证给定的字段值是否是一个合法的URL地址。

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 默认值 | 描述 |
| fieldName | String |  | 要验证的字段名 |

#### 示例

页面：

<s:form action=*"validate"*>

<s:textfield name=*"url"* label=*"个人主页"*></s:textfield>

<s:submit value=*"保存"*></s:submit>

</s:form>

动作类:

**private** String url;

**public** String getUrl() {

**return** url;

}

**public** **void** setUrl(String url) {

**this**.url = url;

}

验证配置文件:

<validators>

<field name=*"url"*>

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

<message>请输入正确的地址</message>

</field-validator>

</field>

</validators>

运行结果:



#### 说明

验证配置文件的另外一种写法：

<validators>

<validator type=*"url"*>

<param name=*"fieldName"*>url</param>

<message>请输入正确的地址</message>

</validator>

</validators>

### visitor validator

#### 功能

该验证程序可以提高代码的可重用性，你可以利用它把同一个验证程序配置文件用于多个动作。

#### 参数

#### 示例

页面：

<s:form action=*"customer\_save"*>

<s:textfield name=*"address.streetName"* label=*"街道"*></s:textfield>

<s:submit></s:submit>

</s:form>

动作类:

**public** **class** Customer **extends** ActionSupport {

**private** Address address;

**public** Address getAddress() {

**return** address;

}

**public** **void** setAddress(Address address) {

**this**.address = address;

}

}

**public** **class** Address {

**private** String streetName;

**public** String getStreetName() {

**return** streetName;

}

**public** **void** setStreetName(String streetName) {

**this**.streetName = streetName;

}

}

验证配置文件:

Address-validation.xml

<validators>

<field name=*"streetName"*>

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

<message>请输入正确街道地址</message>

</field-validator>

</field>

</validators>

Customer-validation.xml

<validators>

<field name=*"address"*>

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

<message>Address:</message>

</field-validator>

</field>

</validators>

运行结果:



#### 说明

### conversion validator

#### 功能

#### 参数

#### 示例

页面：

动作类:

验证配置文件:

运行结果:

#### 说明

### stringlength validator

#### 功能

用来验证一个非空的字段值是不是有足够的长度。

#### 参数

#### 示例

页面：

动作类:

验证配置文件:

运行结果:

#### 说明

### regex validator

#### 功能

用来检查给定字段是否与给定的正则表达式相匹配。正则表达式的详细内容可以参考JDK的java.util.regex.Pattern类。

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 默认值 | 描述 |
| fieldname | String |  | 要验证的字段名 |
| expression | String |  | 正则表达式。此参数是必须的 |
| caseSensitive | Boolean | true | 是否区分大小写的情况 |
| trim | Boolean | true | 验证前是否要去掉前导和尾缀的空白字符 |

#### 示例

页面：

<s:form action=*"validate"*>

<s:textfield name=*"userName"* label=*"用户名"*></s:textfield>

<s:submit value=*"保存"*></s:submit>

</s:form>

动作类:

**private** String userName;

**public** String getUserName() {

**return** userName;

}

**public** **void** setUserName(String userName) {

**this**.userName = userName;

}

验证配置文件:

<validators>

<field name=*"userName"*>

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

<param name=*"expression"*><![CDATA[([aAbBcCdD][123][eEfFgG][456])]]></param>

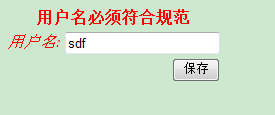
<message> 用户名必须符合规范</message>

</field-validator>

</field>

</validators>

运行结果:



#### 说明

验证配置文件的另外一种写法：

<validators>

<validator type=*"regex"*>

<param name=*"fieldName"*>userName</param>

<param name=*"expression"*><![CDATA[([aAbBcCdD][123][eEfFgG][456])]]></param>

<message> 用户名必须符合规范</message>

</validator>

</validators>

### conditionalvisitor

#### 功能

#### 参数

#### 示例

页面：

动作类:

验证配置文件:

运行结果:

#### 说明

## 编写自定义的验证程序

### 编写步骤：

* 实现特定的接口，常用的是继承已经实现相应接口的类，如com.opensymphony.xwork2.validator.validators.FieldValidatorSupport或com.opensymphony.xwork2.validator.validators.ValidatorSupport
* 注册该验证程序

验证的实现全是靠验证器(validators)完成的，这些验证器必须在ValidatorFactory(利用registerValidator方法)中进行注册。自定义的验证器可以通过一种很简单的方式注册进去，那就是在构建路径中（/WEB-INF/classes）中创建validators.xml文件，在该文件中声明你要注册的验证器。

以下列出了struts2框架中默认的验证器，自定义的验证器的定义语法可以参考以下内容。

|  |
| --- |
| **com.opensymphony.xwork2.validator.validators.default.xml** |
| <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE validators PUBLIC  "-//OpenSymphony Group//XWork Validator Config 1.0//EN"  "http://www.opensymphony.com/xwork/xwork-validator-config-1.0.dtd">  <!-- START SNIPPET: validators-default -->  <validators>  <validator name="required" class="com.opensymphony.xwork2.validator.validators.RequiredFieldValidator"/>  <validator name="requiredstring" class="com.opensymphony.xwork2.validator.validators.RequiredStringValidator"/>  <validator name="int" class="com.opensymphony.xwork2.validator.validators.IntRangeFieldValidator"/>  <validator name="long" class="com.opensymphony.xwork2.validator.validators.LongRangeFieldValidator"/>  <validator name="short" class="com.opensymphony.xwork2.validator.validators.ShortRangeFieldValidator"/>  <validator name="double" class="com.opensymphony.xwork2.validator.validators.DoubleRangeFieldValidator"/>  <validator name="date" class="com.opensymphony.xwork2.validator.validators.DateRangeFieldValidator"/>  <validator name="expression" class="com.opensymphony.xwork2.validator.validators.ExpressionValidator"/>  <validator name="fieldexpression" class="com.opensymphony.xwork2.validator.validators.FieldExpressionValidator"/>  <validator name="email" class="com.opensymphony.xwork2.validator.validators.EmailValidator"/>  <validator name="url" class="com.opensymphony.xwork2.validator.validators.URLValidator"/>  <validator name="visitor" class="com.opensymphony.xwork2.validator.validators.VisitorFieldValidator"/>  <validator name="conversion" class="com.opensymphony.xwork2.validator.validators.ConversionErrorFieldValidator"/>  <validator name="stringlength" class="com.opensymphony.xwork2.validator.validators.StringLengthFieldValidator"/>  <validator name="regex" class="com.opensymphony.xwork2.validator.validators.RegexFieldValidator"/>  <validator name="conditionalvisitor" class="com.opensymphony.xwork2.validator.validators.ConditionalVisitorFieldValidator"/>  </validators>  <!-- END SNIPPET: validators-default --> |

注意：

在struts2.0.7和之前的版本中，如果加入了自定义的验证器，你必须同时还得拥有一份默认验证器的拷贝，即在类路径中的validators.xml的验证器会覆盖掉默认的验证器。但之后的版本则避免了此问题。

### 示例

用来检查密码强度的验证程序。规则：至少包含一个数字、一个小写字母和一个大写字母。此外该验证程序还可以接受一个minLength参数，用户可以通过设置该参数来设置一个可接受的口令的最小长度。

1、编写验证器

**package** wiva.struts2.train.validator;

**import** com.opensymphony.xwork2.validator.ValidationException;

**import** com.opensymphony.xwork2.validator.validators.FieldValidatorSupport;

**public** **class** StrongPasswordValidator **extends** FieldValidatorSupport {

**private** **int** minLength = -1;

**public** **int** getMinLength() {

**return** minLength;

}

**public** **void** setMinLength(**int** minLength) {

**this**.minLength = minLength;

}

**public** **void** validate(Object object) **throws** ValidationException {

String feildName = getFieldName();

String value = (String)getFieldValue(feildName, object);

**if**(value == **null**||value.length()==0){

addFieldError(feildName, object);

}**else** **if**((minLength>-1)&&(value.length()<minLength)){

addFieldError(feildName, object);

}**else** **if**(!isPasswordStrong(value)){

addFieldError(feildName, object);

}

}

**private** **static** **final** String *GROUP1* = "abcdefghijklmnopqrstuvwxyz";

**private** **static** **final** String *GROUP2* = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

**private** **static** **final** String *GROUP3* = "0123456789";

**protected** **boolean** isPasswordStrong(String password) {

**boolean** ok1 = **false**;

**boolean** ok2 = **false**;

**boolean** ok3 = **false**;

**int** length = password.length();

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

**if**(ok1&&ok2&&ok3)

**break**;

String character = password.substring(i,i+1);

**if**(*GROUP1*.contains(character)){

ok1 = **true**;

**continue**;

}

**if**(*GROUP2*.contains(character)){

ok2 = **true**;

**continue**;

}

**if**(*GROUP3*.contains(character)){

ok3 = **true**;

**continue**;

}

}

**return** ok1&&ok2&&ok3;

}

}

2、注册验证器（WEB-INF/classes/validators.xml）

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

<!DOCTYPE validators PUBLIC

"-//OpenSymphony Group//XWork Validator Config 1.0//EN"

"http://www.opensymphony.com/xwork/xwork-validator-config-1.0.dtd">

<validators>

<validator name=*"strongpassword"* class=*"wiva.struts2.train.validator.StrongPasswordValidator"*></validator>

</validators>

3、使用（\*Action-validation.xml）

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

<!DOCTYPE validators PUBLIC

"-//OpenSymphony Group//XWork Validator 1.0.2//EN"

"http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">

<validators>

<field name=*"password"*>

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

<param name=*"minLength"*>6</param>

<message>少包含一个数字、一个小写字母和一个大写字母且不能少于6个</message>

</field-validator>

</field>

</validators>

## 编程验证

之前所使用和编写的验证程序都是声明性的，先声明，后使用。在某些场合，可能因为验证规则过于复杂，用声明性验证会困难一些，因而需要为它们编写必要的验证代码，即需要进行“编程验证”。

Struts2提供了一个com.opensymphony.xwork2.Validateable接口，可以在自己的动作类内通过实现该接口以提供编程验证的功能。

package com.opensymphony.xwork2;

public interface Validateable {

void validate();

}

如果动作类实现了该接口，Struts会调用它的validate方法，所以应把用来验证用户输入的代码编写在这个方法内。ActionSupport实现了该接口，因此动作类继承ActionSupport就不需要直接实现该接口了。

### 示例1

用户注册程序。如果用户输入的用户名已经在数据库中存在了，则要求用户换另外一个用户名进行注册。

动作类

**package** wiva.struts2.train.action;

**import** java.util.LinkedList;

**import** java.util.List;

**import** com.opensymphony.xwork2.ActionSupport;

**public** **class** UserRegAction **extends** ActionSupport {

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 2227569782574386186L;

**private** String userName;

**private** String password;

**private** **static** List<String> *userNames* = **new** LinkedList<String>();

**static**{

//实际业务中,userNames中的内容应该从数据库中查询出来

*userNames*.add("admin");

*userNames*.add("wzhting");

}

**public** String getUserName() {

**return** userName;

}

**public** **void** setUserName(String userName) {

**this**.userName = userName;

}

**public** String getPassword() {

**return** password;

}

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

**this**.password = password;

}

**public** **void** validate() {

**if**(*userNames*.contains(userName))

addFieldError("userName", "用户名"+userName+"已经存在");

}

}

### 示例2

若针对某个动作方法单独进行验证,你需要编写一个方法public void validate方法名().方法名的第一个字母大写

<s:form action=*"addAction"*>

<s:textfield name=*"userName"* label=*"用户名"*></s:textfield>

<s:submit value=*"添加"*></s:submit>

</s:form>

<s:form action=*"editAction"*>

<s:textfield name=*"userName"* label=*"用户名"*></s:textfield>

<s:submit value=*"修改"*></s:submit>

</s:form>

动作类

**package** wiva.struts2.train.action;

**import** com.opensymphony.xwork2.ActionSupport;

**public** **class** ValidateAction **extends** ActionSupport {

**private** **static** **final** **long** *serialVersionUID* = -739143505400361615L;

**private** String userName;

**public** String getUserName() {

**return** userName;

}

**public** **void** setUserName(String userName) {

**this**.userName = userName;

}

**public** String add(){

**return** *SUCCESS*;

}

**public** String edit(){

**return** *SUCCESS*;

}

**public** **void** validateAdd() {

**if**(userName==**null**||userName.length()==0)

addFieldError("userName","请输入用户名");

}

}

# 本地化（Localization）

## 概述

struts2框架在以下方面支持国际化(i18n):

* UI标签
* 来源于ValidationAware接口的消息和错误（ActionSupport、ValidationAwareSupport实现了该接口）
* 通过动作类的getText()方法获得的文本内容（动作类继承了ActionSupport）

## 绑定消息的搜索顺序

* 1. 动作类的属性文件。该文件名与动作类的名字一样且存放在同一个包中。例如：某动作类为train.struts2.actions.SomeAction，它的默认属性文件就是/WEB-INF/classes/train/struts2/actions/SomeAction.properties。
  2. 动作类所实现的各个接口（含间接实现的接口）的属性文件。例如SomeAction实现了接口SomeInterface，那么其对应的属性文件就是SomeInterface.properties。如果第1步中未找到对应的key，则会在此处进行搜索。
  3. 动作类所继承的父类（含间接父类）的属性文件。如果在第2步中还未找到，则会搜索其父类的属性文件并上溯到java.lang.Object。
  4. 如果动作类实现了com.opensymphony.xwork2.ModelDriven接口，则搜寻其模型对象且按照从第1步开始的搜索顺序。
  5. 包属性文件。如果前面还没有找到，则搜索包里的默认属性文件（含其父包中的属性文件），该文件名为package.properties。
  6. ？？
  7. 全局资源文件。

## 访问消息资源文件

我们可以通过在标签中使用getText方法、text标签、i18n标签等形式访问资源文件。

### 使用getText方法

<s:property value=”getText(‘some.key’)”/>

### 使用text标签

第一种：<s:text name=”some,key”/>

如果some.key没有找到，则显示“some.key”；若找到则显示其对应的值。

第二种：<s:text name=”some,key”>

默认值

</s:text>

如果some.key没有找到，则显示“默认值”；若找到则显示其对应的值。

### 使用i18n标签

<s:i18n name="StudentAction">

<s:text name="student.name"></s:text>

</s:i18n>

### 使用UI标签中的key属性

<s:textfield name="name" key="student.name"></s:textfield>

## i18n拦截器

## 日期和数字的格式化

# 文件上传

编写步骤

前提：上传文件表单的enctype必须是multipart/form-data;method必须是post

1. 在你的上传文件的表单里使用<s:file>标签，给他起一个名字。如果需要一次上传多个文件，就必须使用多个file标签，但它们的名字必须是相同的。
2. 编写动作类。此类必须有以下3个属性

|  |  |  |
| --- | --- | --- |
| 属性类型 | 属性名 | 说明 |
| java.io.File | [inputName] | 代表要上传的文件 |
| java.lang.String | [inputName]FileName | 代表要上传的文件文件名 |
| java.lang.String | [inputName]ContentType | 代表要上传的文件内容类型 |
| 注：适用于单文件上传 | | |

1. 多文件上传(暂略)[[查看示例](http://localhost:8080/Upload/multipleUpload.jsp)]

三、File Upload拦截器

struts2应用程序里，File Upload拦截器和Jakarta Commons FileUpload库可以帮助分析并负责上传文件。它是defaultStack拦截器组的成员。

程序员一般会设置以下两个File Upload拦截器的属性:

maximumSize:上传文件的最大长度(单位字节),默认值2MB.

allowedTypes:允许上传得内容类型的清单,各类型之间以逗号分隔.

**public** **class** SingleFileUploadAction **extends** ActionSupport **implements** ServletContextAware{

**private** **static** **final** **long** *serialVersionUID* = -8952900773638870479L;

**private** File attachment;

**private** String attachmentFileName;

**private** String attachmentContentType;

**private** String description;

**private** ServletContext servletContext;

**public** String upload(){

**if**(attachment!=**null**){

String dataDir = servletContext.getRealPath("/WEB-INF/");

File saveFile = **new** File(dataDir,attachmentFileName);

**boolean** r = attachment.renameTo(saveFile);

**return** *SUCCESS*;

}**else**{

**return** *INPUT*;

}

}

**public** **void** setServletContext(ServletContext servletContext) {

**this**.servletContext = servletContext;

}

**public** File getAttachment() {

**return** attachment;

}

**public** **void** setAttachment(File attachment) {

**this**.attachment = attachment;

}

**public** String getAttachmentFileName() {

**return** attachmentFileName;

}

**public** **void** setAttachmentFileName(String attachmentFileName) {

**this**.attachmentFileName = attachmentFileName;

}

**public** String getAttachmentContentType() {

**return** attachmentContentType;

}

**public** **void** setAttachmentContentType(String attachmentContentType) {

**this**.attachmentContentType = attachmentContentType;

}

**public** String getDescription() {

**return** description;

}

**public** **void** setDescription(String description) {

**this**.description = description;

}

}

<struts>

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

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

<action name=*"singleFileUploadInput"*>

<result>/SingleUpload.jsp</result>

</action>

<action name=*"singleFileUpload"* class=*"cn.sdxhce.action.SingleFileUploadAction"* method=*"upload"*>

<interceptor-ref name=*"fileUpload"*>

<param name=*"maximumSize"*>1048576</param>

</interceptor-ref>

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

<result name=*"input"*>/SingleUpload.jsp</result>

<result>/SingleConfirm.jsp</result>

</action>

<action name=*"uploadAction"* class=*"cn.sdxhce.action.FileUpload"*>

<result>/uploadResult.jsp</result>

<result name=*"input"*>/upload.jsp</result>

</action>

<action name=*"multipleUploadAction"* class=*"cn.sdxhce.action.MultipleUploadAction"* method=*"upload"*>

<result name=*"input"*>/multipleUpload.jsp</result>

<result>/multipleUploadResult.jsp</result>

</action>

</package>

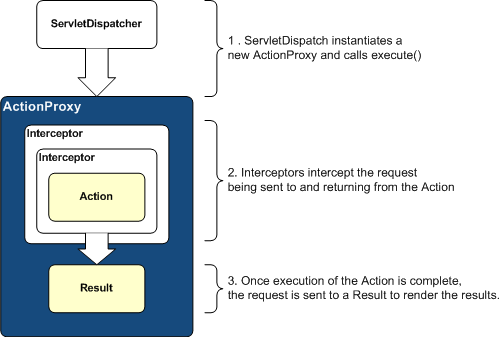
</struts>

# 自定义拦截器

Struts自带的拦截器有35个之多。例如：输入验证是由名为validation拦截器处理的，如果禁用该拦截器，输入验证将停止工作；文件上传如此简单和顺利，要感谢名为fileUpload的拦截器。

Struts自带的默认拦截器足以满足绝大多数的应用程序的需要，但迟早会遇到需要自己建立一个拦截器的时候，这就是自定义拦截器。

## 了解拦截器



## 二、Struts自带的拦截器

拦截器的使用必须先遵循先编写拦截器、再定义后使用的原则。我们所使用的默认拦截器其实都是Struts之前已经编写好了的，并且在某个配置文件中进行了定义，这个文件便是存放在Struts的发行包中，具体位置为struts2-core-\*.jar(\*为版本号)中的struts-default.xml文件中。

<interceptors>

<interceptor name="alias" class="com.opensymphony.xwork2.interceptor.AliasInterceptor"/>

<interceptor name="autowiring" class="com.opensymphony.xwork2.spring.interceptor.ActionAutowiringInterceptor"/>

<interceptor name="chain" class="com.opensymphony.xwork2.interceptor.ChainingInterceptor"/>

<interceptor name="conversionError" class="org.apache.struts2.interceptor.StrutsConversionErrorInterceptor"/>

<interceptor name="cookie" class="org.apache.struts2.interceptor.CookieInterceptor"/>

<interceptor name="clearSession" class="org.apache.struts2.interceptor.ClearSessionInterceptor" />

<interceptor name="createSession" class="org.apache.struts2.interceptor.CreateSessionInterceptor" />

<interceptor name="debugging" class="org.apache.struts2.interceptor.debugging.DebuggingInterceptor" />

<interceptor name="externalRef" class="com.opensymphony.xwork2.interceptor.ExternalReferencesInterceptor"/>

<interceptor name="execAndWait" class="org.apache.struts2.interceptor.ExecuteAndWaitInterceptor"/>

<interceptor name="exception" class="com.opensymphony.xwork2.interceptor.ExceptionMappingInterceptor"/>

<interceptor name="fileUpload" class="org.apache.struts2.interceptor.FileUploadInterceptor"/>

<interceptor name="i18n" class="com.opensymphony.xwork2.interceptor.I18nInterceptor"/>

<interceptor name="logger" class="com.opensymphony.xwork2.interceptor.LoggingInterceptor"/>

<interceptor name="modelDriven" class="com.opensymphony.xwork2.interceptor.ModelDrivenInterceptor"/>

<interceptor name="scopedModelDriven" class="com.opensymphony.xwork2.interceptor.ScopedModelDrivenInterceptor"/>

<interceptor name="params" class="com.opensymphony.xwork2.interceptor.ParametersInterceptor"/>

<interceptor name="actionMappingParams" class="org.apache.struts2.interceptor.ActionMappingParametersInteceptor"/>

<interceptor name="prepare" class="com.opensymphony.xwork2.interceptor.PrepareInterceptor"/>

<interceptor name="staticParams" class="com.opensymphony.xwork2.interceptor.StaticParametersInterceptor"/>

<interceptor name="scope" class="org.apache.struts2.interceptor.ScopeInterceptor"/>

<interceptor name="servletConfig" class="org.apache.struts2.interceptor.ServletConfigInterceptor"/>

<interceptor name="sessionAutowiring" class="org.apache.struts2.spring.interceptor.SessionContextAutowiringInterceptor"/>

<interceptor name="timer" class="com.opensymphony.xwork2.interceptor.TimerInterceptor"/>

<interceptor name="token" class="org.apache.struts2.interceptor.TokenInterceptor"/>

<interceptor name="tokenSession" class="org.apache.struts2.interceptor.TokenSessionStoreInterceptor"/>

<interceptor name="validation" class="org.apache.struts2.interceptor.validation.AnnotationValidationInterceptor"/>

<interceptor name="workflow" class="com.opensymphony.xwork2.interceptor.DefaultWorkflowInterceptor"/>

<interceptor name="store" class="org.apache.struts2.interceptor.MessageStoreInterceptor" />

<interceptor name="checkbox" class="org.apache.struts2.interceptor.CheckboxInterceptor" />

<interceptor name="profiling" class="org.apache.struts2.interceptor.ProfilingActivationInterceptor" />

<interceptor name="roles" class="org.apache.struts2.interceptor.RolesInterceptor" />

<interceptor name="jsonValidation" class="org.apache.struts2.interceptor.validation.JSONValidationInterceptor" />

<interceptor name="annotationWorkflow" class="com.opensymphony.xwork2.interceptor.annotations.AnnotationWorkflowInterceptor" />

<interceptor name="multiselect" class="org.apache.struts2.interceptor.MultiselectInterceptor" />

<!-- Basic stack -->

<interceptor-stack name="basicStack">

<interceptor-ref name="exception"/>

<interceptor-ref name="servletConfig"/>

<interceptor-ref name="prepare"/>

<interceptor-ref name="checkbox"/>

<interceptor-ref name="multiselect"/>

<interceptor-ref name="actionMappingParams"/>

<interceptor-ref name="params">

<param name="excludeParams">dojo\..\*,^struts\..\*</param>

</interceptor-ref>

<interceptor-ref name="conversionError"/>

</interceptor-stack>

<!-- Sample validation and workflow stack -->

<interceptor-stack name="validationWorkflowStack">

<interceptor-ref name="basicStack"/>

<interceptor-ref name="validation"/>

<interceptor-ref name="workflow"/>

</interceptor-stack>

<!-- Sample JSON validation stack -->

<interceptor-stack name="jsonValidationWorkflowStack">

<interceptor-ref name="basicStack"/>

<interceptor-ref name="validation">

<param name="excludeMethods">input,back,cancel</param>

</interceptor-ref>

<interceptor-ref name="jsonValidation"/>

<interceptor-ref name="workflow"/>

</interceptor-stack>

<!-- Sample file upload stack -->

<interceptor-stack name="fileUploadStack">

<interceptor-ref name="fileUpload"/>

<interceptor-ref name="basicStack"/>

</interceptor-stack>

<!-- Sample model-driven stack -->

<interceptor-stack name="modelDrivenStack">

<interceptor-ref name="modelDriven"/>

<interceptor-ref name="basicStack"/>

</interceptor-stack>

<!-- Sample action chaining stack -->

<interceptor-stack name="chainStack">

<interceptor-ref name="chain"/>

<interceptor-ref name="basicStack"/>

</interceptor-stack>

<!-- Sample i18n stack -->

<interceptor-stack name="i18nStack">

<interceptor-ref name="i18n"/>

<interceptor-ref name="basicStack"/>

</interceptor-stack>

<!-- An example of the paramsPrepareParams trick. This stack

is exactly the same as the defaultStack, except that it

includes one extra interceptor before the prepare interceptor:

the params interceptor.

This is useful for when you wish to apply parameters directly

to an object that you wish to load externally (such as a DAO

or database or service layer), but can't load that object

until at least the ID parameter has been loaded. By loading

the parameters twice, you can retrieve the object in the

prepare() method, allowing the second params interceptor to

apply the values on the object. -->

<interceptor-stack name="paramsPrepareParamsStack">

<interceptor-ref name="exception"/>

<interceptor-ref name="alias"/>

<interceptor-ref name="i18n"/>

<interceptor-ref name="checkbox"/>

<interceptor-ref name="multiselect"/>

<interceptor-ref name="params">

<param name="excludeParams">dojo\..\*,^struts\..\*</param>

</interceptor-ref>

<interceptor-ref name="servletConfig"/>

<interceptor-ref name="prepare"/>

<interceptor-ref name="chain"/>

<interceptor-ref name="modelDriven"/>

<interceptor-ref name="fileUpload"/>

<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>

<!-- A complete stack with all the common interceptors in place.

Generally, this stack should be the one you use, though it

may do more than you need. Also, the ordering can be

switched around (ex: if you wish to have your servlet-related

objects applied before prepare() is called, you'd need to move

servletConfig interceptor up.

This stack also excludes from the normal validation and workflow

the method names input, back, and cancel. These typically are

associated with requests that should not be validated.

-->

<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>

<!-- The completeStack is here for backwards compatibility for

applications that still refer to the defaultStack by the

old name -->

<interceptor-stack name="completeStack">

<interceptor-ref name="defaultStack"/>

</interceptor-stack>

<!-- Sample execute and wait stack.

Note: execAndWait should always be the \*last\* interceptor. -->

<interceptor-stack name="executeAndWaitStack">

<interceptor-ref name="execAndWait">

<param name="excludeMethods">input,back,cancel</param>

</interceptor-ref>

<interceptor-ref name="defaultStack"/>

<interceptor-ref name="execAndWait">

<param name="excludeMethods">input,back,cancel</param>

</interceptor-ref>

</interceptor-stack>

</interceptors>

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

## 三、Interceptor接口

从技术角度上讲，每个拦截器都是直接或间接地实现了com.opensymphony.xwork2.interceptor.Interceptor接口的java类，该接口的定义如下：

package com.opensymphony.xwork2.interceptor;

import com.opensymphony.xwork2.ActionInvocation;

import java.io.Serializable;

public interface Interceptor extends Serializable {

void destroy();

void init();

String intercept(ActionInvocation invocation) throws Exception;

}

方法说明：

1. init（）：这个方法将在拦截器被创建出来后立刻被调用，它在拦截器的生命周期内只会调用一次。自定义的拦截器必须实现这个方法对相关资源进行必要的初始化。
2. destroy():这个方法将在拦截器被销毁之前被调用，它在拦截器的生命周期内只会调用一次。自定义的拦截器必须实现这个方法来释放拦截器曾经占用的资源。
3. intercept(ActionInvocation invocation):每拦截到一个动作请求，这个方法就会被调用一次，给拦截器一个机会在该动作开始之前或执行完毕后做些事情。

Struts会依次调用程序员为某个动作而注册的每一个拦截器的intercept方法。在每次调用该方法的时候，都会向它传递一个com.opensymphony.xwork2.ActionInvocation （接口）的实例。一个ActionInvocation对象代表一个给定动作的执行状态,拦截器可以从这个对象获得与该动作相关联的Action对象和Result对象.在完成自己的任务后,拦截器将调用ActionInvocatio的invoke方法前进到动作处理流程的下一个环节.com.opensymphony.xwork2.interceptor.AbstractInterceptor类实现了Interceptor接口,并为init和destroy方法分别提供了一个空白的实现.并非所有的拦截器都需要对某些资源进行初始化和销毁,因此我们自定义的拦截器一般选择继承该类,以节省开发时间.

## 四.自定义拦截器示例

利用拦截器用于登陆用户是否超时

### 拦截器

**public** **class** SessionInterceptor **extends** AbstractInterceptor {

**private** **static** **final** **long** *serialVersionUID* = 6072772019059317328L;

**public** String intercept(ActionInvocation invocation) **throws** Exception {

HttpServletRequest request = ServletActionContext.*getRequest*();

HttpSession session = request.getSession();

Object o = session.getAttribute("user");

**if**(o==**null**)

**return** "sessionInvalided";

**else**

**return** invocation.invoke();

}

}

### Struts配置文件

<struts>

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

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

<interceptors>

<interceptor name=*"sessionCheck"* class=*"cn.sdxhce.interceptor.SessionInterceptor"*></interceptor>

</interceptors>

<global-results>

<result name=*"sessionInvalided"*>/sessionInvalid.jsp</result>

</global-results>

<action name=*"register"* class=*"cn.sdxhce.action.StudentAction"*>

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

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

<result>/result.jsp</result>

</action>

</package>

</struts>

# 使用标准插件

## JFreeChart使用

### 所需jar包:

/jfreechart-1.0.13.jar/jcommon-1.0.16.jar/struts2-jfreechart-plugin-2.1.8.1.jar

### 动作类

**public** **class** GetChartAction **extends** ActionSupport {

**private** **static** **final** **long** *serialVersionUID* = -7814290464584999876L;

**private** JFreeChart chart;

**public** JFreeChart getChart() {

**return** chart;

}

**public** String execute(){

ValueAxis xAxis = **new** NumberAxis("年度");

ValueAxis yAxis = **new** NumberAxis("产值");

XYSeries xySeries = **new** XYSeries("绿豆");

xySeries.add(0,300);

xySeries.add(1,200);

xySeries.add(2,400);

xySeries.add(3,500);

xySeries.add(4,600);

xySeries.add(5,500);

xySeries.add(6,800);

xySeries.add(7,1000);

xySeries.add(8,1100);

XYSeriesCollection xyDataset = **new** XYSeriesCollection(xySeries);

XYPlot xyPlot = **new** XYPlot(xyDataset,xAxis,yAxis,**new** StandardXYItemRenderer(StandardXYItemRenderer.*SHAPES\_AND\_LINES*));

chart = **new** JFreeChart(xyPlot);

**return** *SUCCESS*;

}

}

### 配置文件

<package name=*"p2"* extends=*"jfreechart-default"*>

<action name=*"chart"* class=*"wiva.struts2.train.action.GetChartAction"*>

<result type=*"chart"*>

<param name=*"width"*>600</param>

<param name=*"height"*>400</param>

</result>

</action>

</package>

### 页面

<body>

<s:url action=*"chart"* var=*"url"*></s:url>

<img src="<s:property value=*"url"*/>" alt=*"hello"* />

</body>