Classpath下建立 xwork-conversion.properties

实现Validateable接口 其中有 validate方法

如果假设我们的action继承了ActionSupport

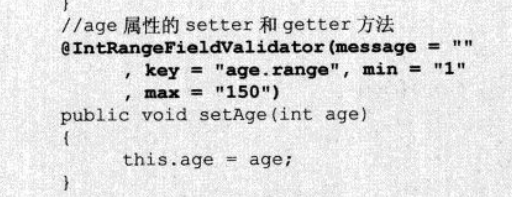
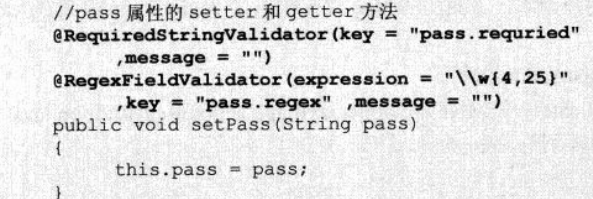
那么我们可以对于login方法 建立一个 void validateLogin(){…}方法

在validateLogin里 如果产生了错误(比如 this.addFieldErrors(…)); 那么此次验证就算失败

还可以在Action类中的字段的set方法上

打上Annotation

比如



基本上大部分验证器的效果 通过看名字就可以猜测出来

想要知道具体有哪些参数可以设置的话可以跟到相关的类去看一下

同Action包下建立

LoginAction-validation.xml

有几个坑爹的地方:

stringlength 用于验证字符串的长度是否介于 minLength maxLength之间

如果你字符串什么都没输入 那么还是可以通过验证的 有点坑爹

所以一定要和 requiredstring 一起用

如果你建立的是LoginAction-validation.xml那么每次不管调用LoginAction的什么方法(即与action对应的方法)

都会触发这些验证.所以这样有时候会带来不方便.

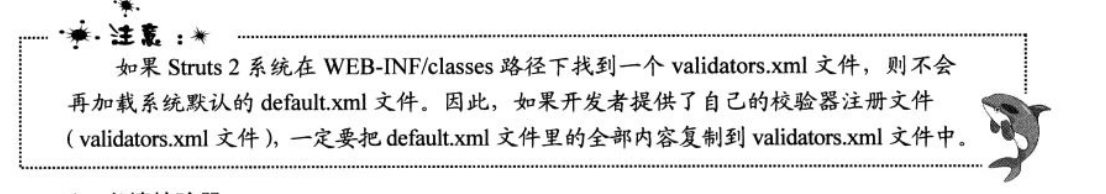
所以可以采用这样的办法:

LoginAction-login-validation.xml 其中login 是在struts.xml 上的 <action name="login" >…

这样当调用login方法的时候 会先触发 LoginAction-validation.xml 和 再触发LoginAction-login-validation.xml 两个一起验证

还有我们的action名字不要太"奇怪",比如叫做"input",我试了一下好像这两个验证文件就会失效 改成其他的名字就可以…

因为input,back,cancel,browse 这些方法是不会被validation拦截器所处理的 可以去查看struts-default.xml



<!DOCTYPE validators PUBLIC "-//Apache Struts//XWork Validator 1.0//EN" "http://struts.apache.org/dtds/xwork-validator-1.0.dtd">

<validators>

<field name="ndouble">

<field-validator type="double">

<param name="minInclusive">1.2</param>

<param name="maxInclusive">5.0</param>

<message>范围不对</message>

</field-validator>

</field>

<field name="age">

<field-validator type="int">

<param name="min">15</param>

<param name="max">20</param>

<message key="age\_error" />

</field-validator>

</field>

<field name="name">

<field-validator type="requiredstring">

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

<message>必须输入姓名</message>

</field-validator>

</field>

<field name="date">

<field-validator type="required">

<message>没有日期啊</message>

</field-validator>

<field-validator type="date">

<param name="min">1990-01-01</param>

<param name="max">2000-01-01</param>

<message>日期不对哦</message>

</field-validator>

</field>

<field name="email">

<field-validator type="requiredstring">

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

<message>邮箱啊</message>

</field-validator>

<field-validator type="email">

<message>%{email}是什么鸟啊</message>

</field-validator>

</field>

<field name="password">

<field-validator type="requiredstring">

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

<message key="password.required" />

</field-validator>

<field-validator type="stringlength">

<param name="minLength">3</param>

<param name="maxLength">10</param>

<message key="password.length" />

</field-validator>

</field>

<field name="password2">

<field-validator type="requiredstring">

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

<message key="password.required" />

</field-validator>

<field-validator type="stringlength">

<param name="minLength">3</param>

<param name="maxLength">10</param>

<message key="password.length" />

</field-validator>

</field>

<validator type="expression">

<param name="expression"><![CDATA[password==password2]]></param>

<message>密码不相等哦</message>

</validator>

</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"/>

除了这几个之外其他的都是好理解的:expression fieldexpression visitor conversion regex

expression定义一个表达式 如果该表达式不通过 则此次action的行动失败 属于actionErrors

fieldexpression类似上面 只是属于fielderrors 这些错误在页面显示的时是通过 <s:actionErrors/>和<s:fieldErrors/>的

visitor 格式如下

<field name="user">

<field-validator type="visitor">

<param name="context">addUser\_input</param>

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

<message>用户信息不正确!</message>

</field-validator>

</field>

意思是将user字段的验证工作交给user对应的类的验证器去验证

所以在user对应的类(假设就叫做User)的同包下必须要有:

User-addUser\_input-validation.xml

这样一个文件 内容的格式和上面的一样

addUser\_input被叫做上下文

可以定义多个上下文,每个上下文的验证工作不一样

conversion

regex 使用如下 显然一个字段可以同时有多个验证器

