Apache Struts 2 Documentation

Validation

Added by Cuong Tran, last edited by Matthieu Chase Heimer on May 25, 2008 (view change) SHOW COMMENT

Struts Action 2 relies on a validation framework provided by [XWork] to enable the application of input validation rules to your Actions before they are executed. This section only provides the bare minimum to get you started and focuses on our extension of the XWork validators to support client-side validation.

> There is also an option for Client Side (Javascript and/or AJAX) based validation, please see Client Side Validation for more information.

Using Annotations

Annotations can be used as an alternative to XML for validation.

Examples

In all examples given here, the validation message displayed is given in plain English - to internationalize the message, put the string in a properties file and use a property key instead, specified by the 'key' attribute. It will be looked up by the framework (see Localization).

1. Basic Validation
2. Client Validation
3. AJAX Validation
4. Using Field Validators
5. Using Non Field Validators
6. Using Visitor Field Validator

Bundled Validators

> **Note**
> When using a Field Validator, Field Validator Syntax is ALWAYS preferable than using the Plain Validator Syntax as it facilitates grouping of field-validators according to fields. This is very handy especially if a field needs to have many field-validators which is almost always the case.

1. conversion validator
2. date validator
3. double validator
4. email validator
5. expression validator
6. fieldexpression validator
7. int validator
8. regex validator
9. required validator
10. requiredstring validator
11. stringlength validator
12. url validator
13. visitor validator

Registering Validators

Validation rules are handled by validators, which must be registered with the ValidatorFactory (using the registerValidator method). The simplest way to do so is to add a file name validators.xml in the root of the classpath (/WEB-INF/classes) that declares all the validators you intend to use.

This list declares all the validators that comes with the framework. It is provided as an example of the syntax, you should not need to declare any of these validators unless you are replacing the defaults.

```
<?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="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>
```

Struts 2.0.7 and Prior
The `validators.xml` containing custom validators needs to contain a copy of the default validators. No DTD was used in validators.xml. See: http://struts.apache.org/2.x/docs/release-notes-208.html#ReleaseNotes2.0.8-MigrationfrompreviousReleases

Turning on Validation

The `defaultStack` already has validation turned on. When creating your own interceptor-stack be sure to include the `validation` interceptor. From `struts-default.xml`:

```
<interceptor-stack name="defaultStack">
   ...
   <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>
```

Beginning with version 2.0.4 Struts provides an extension to XWork's `com.opensymphony.xwork2.validator.ValidationInterceptor` interceptor.

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

Using this validator you can turn off validation for a specific method by using the `@org.apache.struts2.interceptor.validation.SkipValidation` annotation on your action method.

Validator Scopes

Field validators, as the name indicate, act on single fields accessible through an action. A validator, in contrast, is more generic and can do validations in the full action context, involving more than one field (or even no field at all) in validation rule. Most validations can be defined on per field basis. This should be preferred over non-field validation whereever possible, as field validator messages are bound to the related field and will be presented next to the corresponding input element in the respecting view.

Non-field validators only add action level messages. Non-field validators are mostly domain specific and therefore offer custom implementations. The most important standard non-field validator provided by XWork is ExpressionValidator.

Notes

Non-field validators takes precedence over field validators regardless of the order they are defined in *-validation.xml. If a non-field validator is short-circuited, it will causes its non-field validator to not being executed. See validation framework documentation for more info.

Defining Validation Rules

Validation rules can be specified:

1. Per Action class: in a file named ActionName-validation.xml
2. Per Action alias: in a file named ActionName-alias-validation.xml
3. Inheritance hierarchy and interfaces implemented by Action class: XWork searches up the inheritance tree of the action to find default validations for parent classes of the Action and interfaces implemented

Here is an example for SimpleAction-validation.xml:

```
<!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="bar">
      <field-validator type="required">
          <message>You must enter a value for bar.</message>
      </field-validator>
      <field-validator type="int">
          <param name="min">6</param>
          <param name="max">10</param>
          <message>bar must be between ${min} and ${max}, current value is ${bar}.</message>
      </field-validator>
  </field>
  <field name="bar2">
      <field-validator type="regex">
          <param name="regex">[0-9],[0-9]</param>
          <message>The value of bar2 must be in the format "x, y", where x and y are between 0 and 9</message>
     </field-validator>
  </field>
  <field name="date">
      <field-validator type="date">
          <param name="min">12/22/2002</param>
          <param name="max">12/25/2002</param>
          <message>The date must be between 12-22-2002 and 12-25-2002.</message>
      </field-validator>
  </field>
  <field name="foo">
      <field-validator type="int">
```

```
            <param name="min">0</param>
            <param name="max">100</param>
            <message key="foo.range">Could not find foo.range!</message>
        </field-validator>
    </field>
    <validator type="expression">
        <param name="expression">foo lt bar </param>
        <message>Foo must be greater than Bar. Foo = ${foo}, Bar = ${bar}.</message>
    </validator>
</validators>
```

Here we can see the configuration of validators for the SimpleAction class. Validators (and field-validators) must have a type attribute, which refers to a name of an Validator registered with the ValidatorFactory as above. Validator elements may also have <param> elements with name and value attributes to set arbitrary parameters into the Validator instance. See below for discussion of the message element.

In this context, "Action Alias" refers to the action name as given in the Struts configuration. Often, the name attribute matches the method name, but they may also differ.

Localizing and Parameterizing Messages

Each Validator or Field-Validator element must define one message element inside the validator element body. The message element has 1 attributes, key which is not required. The body of the message tag is taken as the default message which should be added to the Action if the validator fails. Key gives a message key to look up in the Action's ResourceBundles using getText() from LocaleAware if the Action implements that interface (as ActionSupport does). This provides for Localized messages based on the Locale of the user making the request (or whatever Locale you've set into the LocaleAware Action). After either retrieving the message from the ResourceBundle using the Key value, or using the Default message, the current Validator is pushed onto the ValueStack, then the message is parsed for \${...} sections which are replaced with the evaluated value of the string between the \${ and }. This allows you to parameterize your messages with values from the Validator, the Action, or both.

If the validator fails, the validator is pushed onto the ValueStack and the message - either the default or the locale-specific one if the key attribute is defined (and such a message exists) - is parsed for ${...} sections which are replaced with the evaluated value of the string between the ${ and }. This allows you to parameterize your messages with values from the validator, the Action, or both.

NOTE: Since validation rules are in an XML file, you must make sure you escape special characters. For example, notice that in the expression validator rule above we use ">" instead of ">". Consult a resource on XML for the full list of characters that must be escaped. The most commonly used characters that must be escaped are: & (use &), > (user >), and < (use <).

Here is an example of a parameterized message:

This will pull the min and max parameters from the IntRangeFieldValidator and the value of bar from the Action.

```
bar must be between ${min} and ${max}, current value is ${bar}.
```

Another notable fact is that the provided message value is capable of containing OGNL expressions. Keeping this in mind, it is possible to construct quite sophisticated messages.

See the following example to get an impression:

```
<message>${getText("validation.failednotice")}! ${getText("reason")}: ${getText("validation.inputrequired")}</message>
```

Validator Flavor

The validators supplied by the XWork distribution (and any validators you might write yourself) come in two different flavors:

1. Plain Validators / Non-Field validators
2. FieldValidators

Plain Validators (such as the ExpressionValidator) perform validation checks that are not inherently tied to a single specified field. When you declare a plain Validator in your -validation.xml file you do not associate a fieldname attribute with it. (You should avoid using plain Validators within the syntax described below.)

FieldValidators (such as the EmailValidator) are designed to perform validation checks on a single field. They require that you specify a fieldname attribute in your -validation.xml file. There are two different (but equivalent) XML syntaxes you can use to declare FieldValidators (see " vs. syntax" below).

There are two places where the differences between the two validator flavors are important to keep in mind:

1. when choosing the xml syntax used for declaring a validator (either or)
2. when using the short-circuit capability

NOTE: Note that you do not declare what "flavor" of validator you are using in your -validation.xml file, you just declare the name of the validator to use and WebWork will know whether it's a "plain Validator" or a "FieldValidator" by looking at the validation class that the validator's programmer chose to implement.

Non-Field Validator Vs Field-Validator

There are two ways you can define validators in your -validation.xml file:

1. <validator>
2. <field-validator>

Keep the following in mind when using either syntax:

Non-Field-Validator The <validator> element allows you to declare both types of validators (either a plain Validator a field-specific FieldValidator).

```
<!-- Declaring a plain Validator using the <validator> syntax: -->

   <validator type="expression>
         <param name="expression">foo gt bar</param>
         <message>foo must be great than bar.</message>
   </validator>
```

```
<!-- Declaring a field validator using the <validator> syntax; -->

   <validator type="required">
        <param name="fieldName">bar</param>
        <message>You must enter a value for bar.</message>
   &lt/validator>
```

field-validator The <field-validator> elements are basically the same as the <validator> elements except that they inherit the fieldName attribute from the enclosing <field> element. FieldValidators defined within a <field-validator> element will have their fieldName automatically filled with the value of the parent <field> element's fieldName attribute. The reason for this structure is to conveniently group the validators for a particular field under one element, otherwise the fieldName attribute would have to be repeated, over and over, for each individual <validator>.

HINT: It is always better to defined field-validator inside a <field> tag instead of using a <validator> tag and supplying fieldName as its param as the xml code itself is clearer (grouping of field is clearer)

NOTE: Note that you should only use FieldValidators (not plain Validators) within a block. A plain Validator inside a <field> will not be allowed and would generate error when parsing the xml, as it is not allowed in the defined dtd (xwork-validator-1.0.2.dtd)

```
Declaring a FieldValidator using the <field-validator> syntax:

<field name="email_address">
  <field-validator type="required">
      <message>You cannot leave the email address field empty.</message>
  </field-validator>
  <field-validator type="email">
      <message>The email address you entered is not valid.</message>
  </field-validator>
</field>
```

The choice is yours. It's perfectly legal to only use elements without the elements and set the fieldName attribute for each of them. The following are effectively equal:

```
<field name="email_address">
  <field-validator type="required">
      <message>You cannot leave the email address field empty.</message>
  </field-validator>
  <field-validator type="email">
      <message>The email address you entered is not valid.</message>
  </field-validator>
</field>


<validator type="required">
  <param name="fieldName">email_address</param>
  <message>You cannot leave the email address field empty.</message>
</validator>
<validator type="email">
  <param name="fieldName">email_address</param>
  <message>The email address you entered is not valid.</message>
</validator>
```

Short-Circuiting Validator

Beginning with XWork 1.0.1 (bundled with WebWork 2.1), it is possible to short-circuit a stack of validators. Here is another sample config file containing validation rules from the Xwork test cases: Notice that some of the <field-validator> and <validator> elements have the short-circuit attribute set to true.

```
<!DOCTYPE validators PUBLIC
        "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
        "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
  <!-- Field Validators for email field -->
  <field name="email">
      <field-validator type="required" short-circuit="true">
          <message>You must enter a value for email.</message>
      </field-validator>
      <field-validator type="email" short-circuit="true">
          <message>Not a valid e-mail.</message>
      </field-validator>
  </field>
  <!-- Field Validators for email2 field -->
  <field name="email2">
     <field-validator type="required">
          <message>You must enter a value for email2.</message>
      </field-validator>
     <field-validator type="email">
          <message>Not a valid e-mail2.</message>
      </field-validator>
  </field>
  <!-- Plain Validator 1 -->
  <validator type="expression">
      <param name="expression">email.equals(email2)</param>
      <message>Email not the same as email2</message>
  </validator>
```

```
    <!-- Plain Validator 2 -->
    <validator type="expression" short-circuit="true">
        <param name="expression">email.startsWith('mark')</param>
        <message>Email does not start with mark</message>
    </validator>
  </validators>
```

short-circuiting and Validator flavors

Plain validator takes precedence over field-validator. They get validated first in the order they are defined and then the field-validator in the order they are defined. Failure of a particular validator marked as short-circuit will prevent the evaluation of subsequent validators and an error (action error or field error depending on the type of validator) will be added to the ValidationContext of the object being validated.

In the example above, the actual execution of validator would be as follows:

1. Plain Validator 1
2. Plain Validator 2
3. Field Validators for email field
4. Field Validators for email2 field

Since Field Validator 2 is short-circuited, if its validation failed, it will causes Field validators for email field and Field validators for email2 field to not be validated as well.

Usefull Information: More complecated validation should probably be done in the validate() method on the action itself (assuming the action implements Validatable interface which ActionSupport already does).

A plain Validator (non FieldValidator) that gets short-circuited will completely break out of the validation stack no other validators will be evaluated and plain validator takes precedence over field validator meaning that they get evaluated in the order they are defined before field validator gets a chance to be evaludated again according to their order defined.

Short cuircuiting and validator flavours

A FieldValidator that gets short-circuited will only prevent other FieldValidators for the same field from being evaluated. Note that this "same field" behavior applies regardless of whether the or syntax was used to declare the validation rule. By way of example, given this -validation.xml file:

```
  <validator type="required" short-circuit="true">
    <param name="fieldName">bar</param>
    <message>You must enter a value for bar.</message>
  </validator>

  <validator type="expression">
    <param name="expression">foo gt bar</param>
    <message>foo must be great than bar.</message>
  </validator>
```

both validators will be run, even if the "required" validator short-circuits. "required" validators are FieldValidator's and will not short-circuit the plain ExpressionValidator because FieldValidators only short-circuit other checks on that same field. Since the plain Validator is not field specific, it is not short-circuited.

How Validators of an Action are Found

As mentioned above, the framework will also search up the inheritance tree of the action to find default validations for interfaces and parent classes of the Action. If you are using the short-circuit attribute and relying on default validators higher up in the inheritance tree, make sure you don't accidentally short-circuit things higher in the tree that you really want!

Resources

WebWork Validation

Next: Localization

Children Show Children