JavaServer Faces – Validation & Error Handling

This tutorial explains the validation and the error handling in JSF and shows a step by step example application. The tutorials shows the standard validators like

`<f:validateLength>`

`<f:validateLongRange>`

`<f:validateDoubleRange>`

and custom validators. With custom validators you can validate any kind of special input data you get.

General

Author:
Sascha Wolski
Sebastian Hennebrueder
http://www.laliluna.de/tutorials.html – Tutorials for Struts, EJB, xdoclet and eclipse.

Date:
March, 10 2005
PDF

http://www.laliluna.de/download/jsf-validation-error-handling-en.pdf

Sources

http://www.laliluna.de/download/jsf-validation-error-handling-source.zip

Validation

JavaServer Faces has an extensive support for validation of input data. It is possible to use standard validators of JSF or to write own validator methods or classes. If the value of a component isn't accepted, an error message is generated for that specific component and the associated attritbute of the class is not modified. These error messages can then be displayed to users so that they can correct their input.

There is no explicit support for client-side validation in JSF' s validation model. Any validator methods you write in backing beans, as well as the standard validators, or custom validators will not generate JavaScript to check a component's value on the client-side. You can support client-side validation in your own validators.

Standard validators

JSF includes a few standard validators. The list below shows the standard validators, which are located in the core JSF tag library.

Validator	*Custom Tag*	*Attributes*	*Behavoir*
Length	`<f:validateLength>`	minimum, maximum	Ensures that the length of the control's value is greater than or equal to *minimum* (if specified) and less than or equal to *maximum* (if specifed)
LongRange	`<f:validateLongRange>`	minimum, maximum	Ensures that the control's value can be converted to a *long* and is greater than or equal to *minimum* (if specifed) and less than or equal to *maximum (if specifed)*

Validator	*Custom Tag*	*Attributes*	*Behavoir*
DoubleRange	`<f:validateDoubleRange>`	minimum, maximum	Ensures that the control's value can be converted to double, and its greater than or equal to *minimum* (if specifed) and less than or equal *maximum (if specifed)*.

Examples:

Length validator

```
<h:inputText id="lengthInput">
   <f:validateLength minimum="2" maximum="5" />
</h:inputText>
```

LongRange validator

```
<h:inputText id="longInput">
   <f:validateLongRange minimum="6" maximum="10" />
</h:inputText>
```

DoubleRange validator

```
<h:inputText id="doubleInput">
   <f:validateDoubleRange minimum="4.1" maximum="9.78" />
</h:inputText>
```

Validator methods

An input control can also be associated with a single validation method on a backing bean. Validator methods are generally used for application-specific validation and aren't reused in other applications. For example we had a backing bean called *myBean* and a validation method *validateCreditCard*.

The following code shows an validator method on a backing bean:

```
public void validateLength(FacesContext context,
   UIComponent component,
   Object input) throws ValidatorException {

   String var = (String)input;

   if(var.length() < 3){
      throw new ValidatorException(
             new FacesMessage(message, null));
   }
}
```

In the JSP we can associate an input control with the method like below:

```
<h:inputText id="creditCardInput"
          value="#{myBean.creditCard}"
          validator="#{myBean.validateLength}" />
```

Validator classes

Validators (Validator classes) are generic and designed for use in different types of applications. The class you create implements the Validator interface of JSF. A validator must be specified in the application configuration file (*faces-config.xml*) of the project and can be used by the logical name.

The following code shows an example validator class called *CustomerValidator*:

```
public class CustomValidator implements Validator{

   /**
    * Method validate
    */
```

```
   public void validate(FacesContext context,
            UIComponent component,
            Object object)
            throws ValidatorException {

      String var = object.toString();

      if(var.length() < 3){
          throw new ValidatorException(
                   new FacesMessage(message, null));
      }
   }
}
```

Example, how to specify a custom validator in the faces-config.xml:

```
<validator>
   <validator-id>customerValidator</validator-id>
   <validator-class>my.package.CustomerValidator</validator-class>
</validator>
```

In JSP you can associate an input control with the validator like below:

```
<h:inputText id="creditCardInput"
      value="#{myBean.creditCard}">
   <f:validator validatorId="customerValidator" />
</h:inputText>
```

Output error messages

The error messages can be displayed back to users with a *<h:messages>* or *<h:message>* element on the JSP. The *<h:messages>* element display all errors holding in the application scope and the *<h:message>* element display only the error message for a specified component.

The example application

Now let's start with a small example application.

Create a new project named *JSFValidation* and add the JavaServer Faces capabilities.

The backing bean

Create a new Java class *MyBean* in the package *de.laliluna.tutorial.validation.bean*.

Add a property var of type String and provide a getter and setter methods.

Add a validator method *validateLength(..)*. Check the length of the component value and if it is less than 3 throw a *ValidatorException*.

The following source code shows the class *MyBean*:

```
public class MyBean {

   //properties
   private String var;


   // getter and setter methods
   public String getVar() {
      return var;
   }
```

```
   public void setVar(String var) {
      this.var = var;
   }


   /**
    * Validator Method
    * (validate the name property)
    * @param context
    * @param input
    */
   public void validateLength(FacesContext context,
               UIComponent component,
               Object input) throws ValidatorException {

        String var = (String)input;

        System.out.println(var.length());

        if(var.length() < 3){
          throw new ValidatorException(
                   new FacesMessage("Length must be greater than 3", null));
        }
   }
}
```

Custom validator class

Create a java class *LengthValidator* in the package de.laliluna, .which implements the interface *Validator* of JSF.

Implement the *validate(..)* method of the interface.

Within the *validate(..)* method check the length of the component value and if it is less than 3 throw a *ValidatorException*.

The following source code shows the class *LengthValidator:*

```
public class LengthValidator implements Validator{

   /**
    * Method validate
    */
   public void validate(FacesContext context,
                   UIComponent component,
                   Object object)
                   throws ValidatorException {

      String var = object.toString();

      if(var.length() < 3){
          throw new ValidatorException(
                   new FacesMessage("Length must be greater than 3", null));
      }


   }
}
```

The JSP file

Create a new JSP file *example.jsp* in the folder */WebRoot* of the project.

The following source code shows the content of the JSP file *example.jsp*:

```
<%@ page language="java" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>


<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
   <title>Validator Example</title>
</head>

<body>
   <f:view>
      <h:form id="standard">
         <table border="1" width="100%">
            <tr>
               <td colspan="4">
                  <h3>Using standard validators</h3>
               </td>
            </tr>
            <tr>
               <td width="25%"><b>Validator</b></td>
               <td width="25%"><b>Comments</b></td>
               <td width="25%"><b>Input</b></td>
               <td width="25%"><b>Errors</b></td>
            </tr>

            <tr>
               <td>None</td>
               <td>required="true"</td>
               <td>
                  <h:inputText
                        id="required"
                        value="#{myBean.var}"
                        required="true"/>
               </td>
               <td>
                  <h:message for="required" />
               </td>
            </tr>

            <tr>
               <td>Length</td>
               <td>minimum="3"<br>maximum="5"</td>
               <td>
                  <h:inputText
                        id="length"
                        value="#{myBean.var}">
                        <f:validateLength minimum="3" maximum="5" />
                  </h:inputText>
               </td>
               <td>
                  <h:message for="length" />
               </td>
            </tr>

            <tr>
               <td>LongRange</td>
               <td>minimum="4"<br>maximum="10"</td>
               <td>
                  <h:inputText
                        id="longRange"
```

```
                value="#{myBean.var}">
                <f:validateLongRange minimum="4" maximum="10" />
            </h:inputText>
        </td>
        <td>
            <h:message for="longRange" />
        </td>
    </tr>

    <tr>
        <td>DoubleRange</td>
        <td>minimum="4.1"<br>maximum="9.86"</td>
        <td>
            <h:inputText
                id="doubleRange"
                value="#{myBean.var}">
                <f:validateDoubleRange minimum="4.1" maximum="9.86" />
            </h:inputText>
        </td>
        <td>
            <h:message for="doubleRange" />
        </td>
    </tr>

    <tr>
        <td>
            Length <br>
            LongRange
        </td>
        <td>
            Length minimum="2"<br>
            Length maximum="3"<br>
            LongRange minimum="10"<br>
            LongRange maximum="15"
        </td>
        <td>
            <h:inputText
                id="combined"
                value="#{myBean.var}">
                <f:validateLength minimum="2" maximum="3" />

                <f:validateLongRange minimum="10" maximum="15" />
            </h:inputText>
        </td>
        <td>
            <h:message for="combined" />
        </td>
    </tr>

    <tr>
        <td colspan="4">
            <h3>Using validator method</h3>
        </td>
    </tr>

    <tr>
        <td>Validation Method</td>
        <td>validator="#{myBean.validateName}</td>
        <td>
            <h:inputText
                id="validateMethod"
                value="#{myBean.var}"
                validator="#{myBean.validateLength}"/>
```

```
               </td>
               <td>
                  <h:message for="validateMethod" />
               </td>
            </tr>

            <tr>
               <td colspan="4">
                  <h3>Using custom validator class</h3>
               </td>
            </tr>

            <tr>
               <td>Validator Class</td>
               <td>validatorId="lengthValidator"</td>
               <td>
                  <h:inputText
                        id="validatorClass"
                        value="#{myBean.var}">
                        <f:validator validatorId="lengthValidator"/>
                  </h:inputText>
               </td>
               <td>
                  <h:message for="validatorClass" />
               </td>
            </tr>
         </table>

         <br>

         <h:commandButton
            id="submit"
            value="Submit" />
      </h:form>
   </f:view>
</body>
</html>
```

Configure the application configuration file (faces-config.xml)

Open the faces-config.xml.

Add the navigation rule.

Add the mapping for the backing bean MyBean.

Register the custom validator class LengthValidator.

The following source code shows the content of the *faces-config.xml*:

```
<faces-config>
   <navigation-rule>
      <from-view-id>example.jsp</from-view-id>
      <navigation-case>
          <from-outcome>success</from-outcome>
          <to-view-id>example.jsp</to-view-id>
      </navigation-case>
   </navigation-rule>

   <managed-bean>
      <managed-bean-name>myBean</managed-bean-name>
      <managed-bean-class>de.laliluna.tutorial.validation.bean.MyBean</managed-
bean-class>
      <managed-bean-scope>request</managed-bean-scope>
   </managed-bean>
```

```
   <validator>
      <validator-id>lengthValidator</validator-id>
      <validator-
class>de.laliluna.tutorial.validation.validator.LengthValidator</validator-
class>
   </validator>
</faces-config>
```

Now that's all, you can test the application. We use a Jboss or Tomcat installation.

Call the project with the following link:

http://localhost:8080/JSFValidation/example.faces