|  |  |
| --- | --- |
| **Struts 2 Validation Tutorial**  In this example you will see how to perform validation in Struts 2. You will learn how the validate() method is called by the workflow interceptor and how the errors are stored using the ValidationAware interface. You will also see how to retrieve error messages from the properties file.  In this example we will see how we can validate a login page using Struts 2. Let's first create the login page. We use Struts UI tags to create the login page. The <s:head /> tag should be placed in the head section of the HTML page. The s:head tag automatically generates links to the css and javascript libraries that are necessary to render the form elements.  The s:form tag contains all the form elements. The action attribute contains the action name to wich the form should be submitted. This action name should be same as the one specified in the XML declarative architecture. In this example we use struts.xml file to do the configuration.  The textfield tag is used create a text box. The label attribute of the textfield tag contains the name to be displayed on the page and the name attribute contains the name of the property in the action class to be mapped. The password tag is same as the textfield tag except that the input value is masked. The submit tag is used to create a submit button, the value "Login" represents the label of the button.  Note that the code is simple without any HTML tables, this is because Struts 2 will automatically create the necessary tables for the page based on the theme selected. By default the XHTML theme is selected. | |
| 01.<%@taglib uri="/struts-tags" prefix="s" %>  02.<html>  03.<head>  04.<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  05.  06.<title>Login Page</title>  07.<s:head />  08.</head>  09.<body>  10.    <s:form action="Login">  11.        <s:textfield name="userName" label="User Name" />  12.  13.        <s:password name="password" label="Password" />  14.        <s:submit value="Login" />  15.    </s:form>  16.  17.</body>  18.</html>  When the user clicks the Login button the request will be forwarded to the Login action.  We do the action mapping using the struts.xml file. First we need to create a package for our action.  01.<struts>  02.    <package name="default" extends="struts-default">  03.  04.        <action name="Login" class="tutorials4u.Login">  05.            <result name="input">/login.jsp</result>  06.            <result name="success">/success.jsp</result>  07.  08.        </action>  09.    </package>  10.</struts>  Here our "default" package extends "struts-default" package. By extending the "struts-default" package the action will by default inherit the set of interceptors defined in the defaultstack. The "struts-default" package is defined in the struts-default.xml file.  All the common tasks done by the Actions are seperated and placed in different interceptors. You can define an interceptor stack for each action. Most commonly used interceptors are grouped in defaultstack of the struts-default package. The defaultstack will be sufficient in most cases. The inteceptors will be fired in the order in which they are declared in the stack both before and after the action is executed.  Here the "Login" action is mapped to the "Login" class in the "tutorials4u" package. The results are defined using the "<result>" element. If any validation errors occur the user will be forwarded to the login.jsp page. If the login is successfull then the user will be forwarded to the success.jsp page.  The defaultstack contains the following interceptors.  01.<interceptor-stack name="defaultStack">  02.     <interceptor-ref name="exception"/>  03.     <interceptor-ref name="alias"/>  04.     <interceptor-ref name="servletConfig"/>  05.  06.     <interceptor-ref name="prepare"/>  07.     <interceptor-ref name="i18n"/>  08.     <interceptor-ref name="chain"/>  09.     <interceptor-ref name="debugging"/>  10.  11.     <interceptor-ref name="profiling"/>  12.     <interceptor-ref name="scopedModelDriven"/>  13.     <interceptor-ref name="modelDriven"/>  14.     <interceptor-ref name="fileUpload"/>  15.  16.     <interceptor-ref name="checkbox"/>  17.     <interceptor-ref name="staticParams"/>  18.     <interceptor-ref name="actionMappingParams"/>  19.     <interceptor-ref name="params">  20.  21.     <param name="excludeParams">dojo\..\*,^struts\..\*</param>  22.     </interceptor-ref>  23.     <interceptor-ref name="conversionError"/>  24.     <interceptor-ref name="validation">  25.  26.     <param name="excludeMethods">input,back,cancel,browse</param>  27.     </interceptor-ref>  28.     <interceptor-ref name="workflow">  29.     <param name="excludeMethods">input,back,cancel,browse</param>  30.  31.     </interceptor-ref>  32. </interceptor-stack>  Our Login Action class extends ActionSupport. It is good to extend ActionSupport class as it provides default implementation for most common tasks.  01.public class Login extends ActionSupport {  02.  03.    private String userName;  04.    private String password;  05.  06.    public Login() {  07.    }  08.  09.    public String execute() {  10.        return SUCCESS;  11.    }  12.  13.    public void validate() {  14.        if (getUserName().length() == 0) {  15.            addFieldError("userName", "User Name is required");  16.        } else if (!getUserName().equals("tutorials4u")) {  17.            addFieldError("userName", "Invalid User");  18.        }  19.        if (getPassword().length() == 0) {  20.            addFieldError("password", getText("password.required"));  21.        }  22.    }  23.  24.    public String getUserName() {  25.        return userName;  26.    }  27.  28.    public void setUserName(String userName) {  29.        this.userName = userName;  30.    }  31.  32.    public String getPassword() {  33.        return password;  34.    }  35.  36.    public void setPassword(String password) {  37.        this.password = password;  38.    }  39.}  The ActionSupport class implements **Action** interface which exposes the **execute()** method.  The following constants are declared in the Action interface which can be used as return values in the execute() method.  public static final String ERROR = "error"  public static final String INPUT = "input"  public static final String LOGIN = "login"  public static final String NONE = "none"  public static final String SUCCESS = "success"  **ERROR** is returned when the action execution fails.  **INPUT** is returned when the action requires more input from the user.  **LOGIN** is returned when the user is not logged into the system.  **NONE** is returned when the action execution is successfull and there are no views to display.  **SUCCESS** is returned when the action executed successfully and the corresponding result is displayed to the user.  Now lets see the roles played by the different interceptors.  The **params** interceptor helps in transfering the request data onto the action object.  The **workflow** interceptor controls the flow of cotrol.  The workflow interceptor checks whether the action implements the **Validateable** interface , if it does, the workflow interceptor will invoke the **validate()** method of the Action class.  In the validate() method we validate the user name and the password. If the validation fails an error is added using the **addFiledError()** method.  The validate() method doesn't return any errors, instead it stores all the errors with the help of the **ValidationAware** interface.  Now the workflow interceptor will check any validation errors has occured. If any error has occured the workflow interceptor will stop the request processing and transfer the control to the input page with the appropriate error messages.  On executing the sample example the following page will be displayed to the user.  http://www.tutorials4u.net/struts2-tutorial/images/LoginPage1Pic1.JPG  For each field the error messages can be added using the addFieldError() method. The error messages can either be added directly or it can be specified in a seperate properties file.  The properties files should have the same name as the Action class. In our case the properties file name is "Login.properties" and the Action name is "Login.java".  The Login.properties file contains the following entry.  1.password.required = Password is required.  The getText() method provided by the TextProvider interface can be used to retrive the error messages.  http://www.tutorials4u.net/struts2-tutorial/images/LoginPage1Pic2.JPG  You can also do the business validations in the validate() method.  http://www.tutorials4u.net/struts2-tutorial/images/LoginPage1Pic3.JPG  If there are no errors, then the execute() method will be invoked by the workflow interceptor.  In our execute() method we simply return "success". The user will be forwarded to the success page.  http://www.tutorials4u.net/struts2-tutorial/images/LoginPage1Pic4.JPG |  |
| **Struts 2 Validation Using XML File Tutorial**  In this example you will see how to perform validations using XML validation file. You will learn the naming convention for the XML validation file and see how to do field validations.  In this example we will see how we can perform validations using XML validation file. The naming convention of the XML validation file should be **ActionClass-Validation.xml**. Here our Action Class name is "Login.java" and the XML validation file name is "Login-Validation.xml".  The Login-Validation.xml file contains the following code. | |
| 01.<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"  02."http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">  03.  04.<validators>  05.    <field name="userName">  06.  07.        <field-validator type="requiredstring">  08.            <message>User Name is required.</message>  09.        </field-validator>  10.    </field>  11.    <field name="password">  12.  13.        <field-validator type="requiredstring">  14.            <message key="password.required" />  15.        </field-validator>  16.    </field>  17.</validators>  The field element contains the name of the form property that needs to be validated. The filed-validator element inside the field element contains the type of validation that needs to be performed.  Here you can either specify the error message directly using the message element or you can use the properties file to define all the errror messages and use the key attribute to specify the error key.  Note the properties file should also have the same name as the Action class.  The **Login Action** class contains the following code.  01.public class Login extends ActionSupport {  02.  03.    private String userName;  04.    private String password;  05.  06.    public Login() {  07.    }  08.  09.    public String execute() {  10.        return SUCCESS;  11.    }  12.  13.    public String getUserName() {  14.        return userName;  15.    }  16.  17.    public void setUserName(String userName) {  18.        this.userName = userName;  19.    }  20.  21.    public String getPassword() {  22.        return password;  23.    }  24.  25.    public void setPassword(String password) {  26.        this.password = password;  27.    }  28.}  The **login.jsp** page contains the following code.  01.<%@page contentType="text/html" pageEncoding="UTF-8"%>  02.<%@taglib uri="/struts-tags" prefix="s" %>  03.  04.<html>  05.<head>  06.<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  07.<title>Login Page</title>  08.<s:head />  09.</head>  10.  11.<body>  12.    <s:form action="LoginAction">  13.        <s:textfield name="userName" label="User Name" />  14.        <s:password name="password" label="Password" />  15.  16.        <s:submit value="Login" />  17.    </s:form>  18.</body>  19.</html>  The <s:head /> tag is used to include the required css and js file for the selected theme. By default the **xhtml** theme is used.  Execute the example and click the Login button without entering the user name and password the following page will be displayed.  http://www.tutorials4u.net/struts2-tutorial/images/LoginPage1Pic2.JPG  Enter a user name and password and click the Login button the following success page will be displayed.  http://www.tutorials4u.net/struts2-tutorial/images/LoginPage1Pic4.JPG |  |