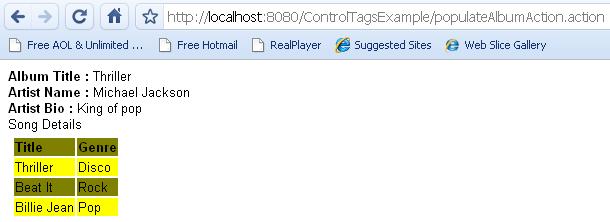
**Struts 2 Control Tags Tutorial**

In this example you will learn how to use the iterator tag and the if and else tags.

In this example you will see how display the following details using the Struts 2 iterator tag and the if and else tags.



In our AlbumInfoAction class we populate the artist and the song details using the following code.

01.package tutorials4u;

02.

03.import java.util.ArrayList;

04.import java.util.List;

05.

06.

07.public class AlbumInfoAction{

08.

09.private String title;

10.private Artist artist;

11.private static List<Song> songs = new ArrayList<Song>();

12.

13.    static {

14.        songs.add(new Song("Thriller","Disco"));

15.        songs.add(new Song("Beat It","Rock"));

16.        songs.add(new Song("Billie Jean","Pop"));

17.    }

18.

19.    public String populate()

20.    {

21.        title = "Thriller";

22.        artist = new Artist("Michael Jackson","King of pop");

23.        return "populate";

24.    }

25.

26.    public String execute()

27.    {

28.        return "success";

29.    }

30.

31.    public String getTitle() {

32.        return title;

33.    }

34.    public void setTitle(String title) {

35.        this.title = title;

36.    }

37.    public Artist getArtist() {

38.        return artist;

39.    }

40.    public void setArtist(Artist artist) {

41.        this.artist = artist;

42.    }

43.

44.    public List<Song> getSongs() {

45.        return songs;

46.    }

47.

48.}

The song class contains the title and the genre attributes.

01.package tutorials4u;

02.

03.public class Song {

04.

05.    private String title;

06.    private String genre;

07.

08.    Song(String title, String genre)

09.    {

10.        this.title = title;

11.        this.genre = genre;

12.    }

13.    public String getTitle() {

14.            return title;

15.    }

16.    public void setTitle(String title) {

17.            this.title = title;

18.    }

19.    public String getGenre() {

20.            return genre;

21.    }

22.    public void setGenre(String genre) {

23.            this.genre = genre;

24.    }

25.}

|  |  |
| --- | --- |
| **Struts 2 OGNL Expression Language Tutorial**  In this tutorial you will learn more about the OGNL expression language and the syntax for accessing the Java Collections like Arrays, Lists and Maps.  In this example you will learn the different syntaxes for using the **Object-Graph Navigation Language (OGNL)**. OGNL expression language is simple and powerful. OGNL expression language helps in accessing the values stored on the ValueStack and in the ActionContext. | |
| First let's see how to access an array of String variables using OGNL. In the action class we create a string array and initialize it as shown below.  01.package tutorials4u;  02.  03.public class SampleAction {  04.  05.    private  String[] sampleArray;  06.    {  07.        sampleArray =  new String[]{"item1","item2","item3"};  08.    }  09.    public String execute()  10.    {  11.        return "success";  12.    }  13.    public String[] getSampleArray() {  14.        return sampleArray;  15.    }  16.    public void setSampleArray(String[] sampleArray) {  17.        this.sampleArray = sampleArray;  18.    }  19.}  You can access the array values in the jsp page using the OGNL expression language in the following way.  01.<b>Array Usage Examples</b>  02.  03.<br><hr>  04.<b>sampleArray :</b> <s:property value="sampleArray"/> <br>  05.<b>sampleArray.length :</b> <s:property value="sampleArray.length"/> <br>  06.  07.<b>sampleArray[0] :</b> <s:property value="sampleArray[0]"/> <br>  08.<b>[0].sampleArray :</b> <s:property value="[0].sampleArray"/> <br>  09.<b>top.sampleArray :</b> <s:property value="top.sampleArray"/> <br>  The figure shows the syntax and the corresponding output.  http://www.tutorials4u.net/struts2-tutorial/images/OgnlPic1.JPG  Since our object is on top of the ValueStack we can access it using [0] notation. If our object was in the second position from the top, we will access it using the [1] notation.  We can also do this using the top keyword. top returns the value on top of the ValueStack.  Now let's see how to access an ArrayList using the OGNL expression language. In the action class we create and initialize the ArrayList as shown below.  01.package tutorials4u;  02.  03.import java.util.ArrayList;  04.import java.util.List;  05.  06.public class SampleAction {  07.  08.    private  List<String> sampleList = new ArrayList<String>();  09.    {  10.        sampleList.add("listItem1");  11.        sampleList.add("listItem2");  12.        sampleList.add("listItem3");  13.    }  14.    public String execute()  15.    {  16.        return "success";  17.    }  18.    public List<String> getSampleList() {  19.        return sampleList;  20.    }  21.    public void setSampleList(List<String> sampleList) {  22.        this.sampleList = sampleList;  23.    }  24.}  You can access the ArrayList values in the jsp page using the following syntax.  1.<b>List Usage Examples</b>  2.<br><hr>  3.<b>sampleList :</b> <s:property value="sampleList"/> <br>  4.  5.<b>sampleList.size :</b> <s:property value="sampleList.size"/> <br>  6.<b>sampleList[0] :</b> <s:property value="sampleList[0]"/> <br>  The figure shows the syntax and the corresponding output.  http://www.tutorials4u.net/struts2-tutorial/images/OgnlPic2.JPG  Now let's see how to access a Map using the OGNL expression language. In the action class we create and initialize the HashMap as shown below.  01.package tutorials4u;  02.  03.import java.util.HashMap;  04.import java.util.Map;  05.  06.public class SampleAction {  07.  08.    private  Map<Integer,String> sampleMap = new HashMap<Integer,String>();  09.    private  String carMake;  10.    {  11.        sampleMap.put(new Integer(1), "one");  12.        sampleMap.put(new Integer(2), "two");  13.        sampleMap.put(new Integer(3), "three");  14.    }  15.    public String execute()  16.    {  17.        return "success";  18.    }  19.    public Map<Integer, String> getSampleMap() {  20.        return sampleMap;  21.    }  22.    public void setSampleMap(Map<Integer, String> sampleMap) {  23.        this.sampleMap = sampleMap;  24.    }  25.    public String getCarMake() {  26.        return carMake;  27.    }  28.    public void setCarMake(String carMake) {  29.        this.carMake = carMake;  30.    }  31.  32.}  You can access the Map values in the jsp page using the following OGNL expression language syntax.  1.<b>Map Usage Examples</b>  2.<br><hr>  3.<b>sampleMap[1] :</b> <s:property value="sampleMap[1]"/> <br>  4.  5.<b>sampleMap.size :</b> <s:property value="sampleMap.size"/> <br>  You can also create a map in the jsp page using the following syntax.  1.<s:select list="#{'make1':'Ford', 'make2':'Honda', 'make3':'Toyota'}" name="carMake" label="Select "></s:select>  The figure shows the syntax and the corresponding output.  http://www.tutorials4u.net/struts2-tutorial/images/OgnlPic3.JPG  Now let's see how to access the name property of the User object in the Action class using the OGNL expression language. The SampleAction contains the following code.  01.package tutorials4u;  02.  03.public class SampleAction {  04.  05.    private  User user = new User();  06.    {  07.        user.setName("Eswar");  08.    }  09.    public String execute()  10.    {  11.        return "success";  12.    }  13.  14.    public String getQuote()  15.    {  16.        return "Don't think, just do";  17.    }  18.    public User getUser() {  19.        return user;  20.    }  21.    public void setUser(User user) {  22.        this.user = user;  23.    }  24.  25.}  You need to use the second-level OGNL expression language to access the user name property.  1.<b>user.name :</b> <s:property value="user.name"/> <br>  You can also invoke a method in the action class in the following way. This will invoke the quote() method in the action class.  1.<b>quote() :</b> <s:property value="quote()"/> <br>  The figure shows the syntax and the corresponding output.  http://www.tutorials4u.net/struts2-tutorial/images/OgnlPic4.JPG |  |
| **Struts 2 Interceptors Example**  In this example you will see how the interceptors are invoked both before and after the execution of the action and how the results are rendered back to the user.  In this example you will see how the interceptors are invoked both before and after the execution of the action and how the results are rendered back to the user. Let's understand this with the help of the following diagram.  http://www.tutorials4u.net/struts2-tutorial/images/Interceptor1Pic1.gif | |
| The following actions happen in a sequence when a request comes to the Struts 2 framework.   * Framework first finds which **Action class** to invoke for this request and **discovers the interceptors** associated with the action mapping. * Now the Framework creates an instance of **ActionInvocation** and calls its **invoke()** method. At this point the Frameworks hands the control over to the ActionInvocation for further processing of the request. * ActionInvocation is the one which encapsulates the action and the associated intercteptors. ActionInvocation knows in which sequence the interceptors should be invoked. * ActionInvocation now invokes the **intercept()** method of the first interceptor in the stack. We will understand this with the help of an example. Our example is very simple it uses only one interceptor for logging details. * The LoggingInterceptor's intercept() method contains the following code.   01.public String intercept(ActionInvocation invocation) throws Exception  02.{  03.    //Pre processing  04.    logMessage(invocation, START\_MESSAGE);  05.  06.    String result = invocation.invoke();  07.  08.    //Post processing  09.    logMessage(invocation, FINISH\_MESSAGE);  10.  11.    return result;  12.}   * As you can see, first the logMessage() method is called and the message is logged, this is the pre processing done by the logger interceptor, then the invoke() method of the ActionInvocation is again called, this time the ActionInvocation will call the next intercetor in the stack and this cycle will continue till the last interceptor in the stack. * After the execution of all the interceptors the action class will be invoked. Finally a result string will be returned and the corresponding view will be rendered. This is the normal flow of events. * But what if an validation error occurs, in this case the request processing will be stopped. No further interceptors will be invoked. Action will not be executed. The control flow changes, now the interceptors executed so far will be invoked in the reverse order to do the post processing if any and finally the result will be rendered to the user. * Let's come back to the normal flow. In our case the logger interceptor is the only interceptor in the stack, so after logging the "START\_MESSAGE", the ActionInvocation's invoke() method will invoke the action. Our action simply returns "success", then again the logger interceptor will be invoked to do the post processing, this time the "FINISH\_MESSAGE" is logged and the result is returned. Based on the result the corresponding view will be rendered to the user.   We get the following benefits by using the interceptors.   * Extremely flexiable. * Cleaner and focused Action classes. * Provides code readability and code reuse. * Testing process becomes easier. * We can add only the interceptors we need to the stack and customising the action processing for each request.   Now lets see the flow of the example. In the index.jsp page we forward the request to the "TestLogger" URL.  1.<META HTTP-EQUIV="Refresh" CONTENT="0;URL=TestLogger.action">  The TestLogger URL is mapped to the TestLoggerAction class in the struts.xml file.  01.<!DOCTYPE struts PUBLIC  02."-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"  03."http://struts.apache.org/dtds/struts-2.0.dtd">  04.  05.<struts>  06.    <package name="test" extends="struts-default">  07.        <action name="TestLogger" class="com.tutorials4u.interceptor.TestLoggerAction">  08.            <interceptor-ref name="logger" />  09.            <result name="success">/success.jsp</result>  10.  11.        </action>  12.    </package>  13.</struts>  The interceptor-ref element is used to add a interceptor reference to the action. All the interceptors are defined in the struts-default package of the struts-default.xml file.  Now the execute() method of the TestLoggerAction class will be invoked. The execute() method just prints a statement and returns success.  01.package com.tutorials4u.interceptor;  02.  03.public class TestLoggerAction {  04.  05.    public String execute()  06.    {  07.        System.out.println("From Action");  08.        return "success";  09.    }  10.}  Based on the mapping in the XML configuration file the user will be forwarded to the success page.  The following log messages are logged in the console.  1.INFO: Starting execution stack for action //TestLogger  2.From Action  3.INFO: Finishing execution stack for action //TestLogger |  |

**Struts 2 Interceptors Tutorial**

In this tutorial you will see different ways to create you own interceptor stack and associate it with the action class.

Struts 2 comes with a set of pre defined interceptors and interceptor stacks which you can use out of the box. The struts-default.xml file contains the struts-default package which defines all the interceptors and the interceptor stacks. You can use the stack that meets your need.

When you extend your package from the struts-default package by default the defaultStack will be used for all the actions in your package. This is configured in the struts-default.xml file in the following way.

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

Let's now create our own interceptor stack. The interceptor-stack element is used to create an interceptor stack. A stack contains a group of interceptors. Each interceptor in the stack is defined using the interceptor-ref element. In this example we will create a stack similar to the defaultStack and customise the validation interceptor according to our need.

We have three methods in our SampleAction class, populate() ,execute() and validate(). Since we extend our class from ActionSupport which inturn implements the Validateable interface, the validate() method of the action class will be called by the workflow interceptor. By default the validate() method will be called during the execution of both populate() and execute() methods but we need to validate only when the execute() method is invoked.

We do this by specifying the populate method in the excludeMethods parameter of the validation interceptor.

The struts.xml file contains the following code.

01.<!DOCTYPE struts PUBLIC

02."-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"

03."http://struts.apache.org/dtds/struts-2.0.dtd">

04.

05.<struts>

06.    <package name="default" extends="struts-default">

07.

08.        <interceptors>

09.            <interceptor-stack name="exampleStack">

10.                    <interceptor-ref name="exception" />

11.                    <interceptor-ref name="alias" />

12.                    <interceptor-ref name="servletConfig" />

13.                    <interceptor-ref name="prepare" />

14.

15.                    <interceptor-ref name="i18n" />

16.                    <interceptor-ref name="chain" />

17.                    <interceptor-ref name="debugging" />

18.                    <interceptor-ref name="profiling" />

19.                    <interceptor-ref name="scopedModelDriven" />

20.                    <interceptor-ref name="modelDriven" />

21.

22.                    <interceptor-ref name="fileUpload" />

23.                    <interceptor-ref name="checkbox" />

24.                    <interceptor-ref name="staticParams" />

25.                    <interceptor-ref name="actionMappingParams" />

26.                    <interceptor-ref name="params">

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

28.

29.                    </interceptor-ref>

30.                    <interceptor-ref name="conversionError" />

31.                    <interceptor-ref name="validation">

32.                            <param name="excludeMethods">populate</param>

33.                    </interceptor-ref>

34.

35.                    <interceptor-ref name="workflow">

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

37.                    </interceptor-ref>

38.            </interceptor-stack>

39.        </interceptors>

40.

41.        <action name="\*Sample" method="{1}" class="tutorials4u.SampleAction">

42.            <interceptor-ref name="exampleStack" />

43.            <result name="populate">/first.jsp</result>

44.            <result name="success">/success.jsp</result>

45.        </action>

46.

47.    </package>

48.</struts>

If you see our exampleStack the only change that we have done is, we have changed the excludeMethods of the validation interceptor, rest all is similar to the defaultStack. This is just to show you how to create your own interceptor stack, you can also achieve the same in a much simpler way.

You can extend your stack from the defaultStack and override the excludeMethods parameter of the validation interceptor in the following way to achieve the same result.

01.<!DOCTYPE struts PUBLIC

02."-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"

03."http://struts.apache.org/dtds/struts-2.0.dtd">

04.

05.<struts>

06.    <package name="default" extends="struts-default">

07.

08.        <action name="\*Sample" method="{1}" class="com.tutorials4u.interceptor.SampleAction">

09.            <interceptor-ref name="defaultStack" >

10.                <param name="validation.excludeMethods"> populate</param>

11.            </interceptor-ref>

12.            <result name="populate">/first.jsp</result>

13.

14.            <result name="success">/success.jsp</result>

15.        </action>

16.    </package>

17.</struts>

Our SampleAction class contains the following code.

01.package com.tutorials4u.interceptor;

02.

03.import com.opensymphony.xwork2.ActionSupport;

04.

05.public class SampleAction extends ActionSupport{

06.

07.    private static final long serialVersionUID = 1L;

08.

09.    public void validate()

10.    {

11.        System.out.println("validate() method called");

12.    }

13.

14.    public String populate()

15.    {

16.    System.out.println("populate() method called");

17.        return "populate";

18.    }

19.

20.    public String execute()

21.    {

22.        System.out.println("execute() method called");

23.        return SUCCESS;

24.    }

25.}

When you run the code using the defaultStack without any changes. The following messages gets printed in the console.

1.validate() method called

2.populate() method called

3.validate() method called

4.execute() method called

When you run the code the using the exampleStack we just created. The follwing messages gets printed in the console.

1.populate() method called

2.validate() method called

3.execute() method called

As you can see the validate() method is not invoked during populate. In this way you can customise the stack base on your requirement