|  |  |
| --- | --- |
| **DispatchAction Functionality in Struts 2 Tutorial**  In this tutorial you will learn how to group a set of related functionalities in a single Action class.  In Struts 1 DispatchAction helps us in grouping a set of related functions into a single action. In Struts 2 all the Actions by default provide this functionality. To use this functionality we need to create different methods with the similar signature of the execute() method, only the name of the method changes.  In our example the UserAction class contains all the functions related to a User like addUser(), updateUser() and deleteUser(). | |
| package com.tutorials4u.user;  import com.opensymphony.xwork2.ActionSupport;  public class UserAction extends ActionSupport{  private String message;  public String execute()  {  message = "From execute method";  return SUCCESS;  }  public String add()  {  message = "From add method";  return SUCCESS;  }  public String update()  {  message = "From update method";  return SUCCESS;  }  public String delete()  {  message = "From delete method";  return SUCCESS;  }  public String getMessage() {  return message;  }  public void setMessage(String message) {  this.message = message;  }  }  Unlike Struts 1 you can even have the execute() method along with the other methods in the Action class. We need to specify which method in the action class will be called while configuring the action mapping. A separate action mapping needs to be created for each method in the action class. The following action mapping is done in the struts.xml file.  <!DOCTYPE struts PUBLIC  "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"  "http://struts.apache.org/dtds/struts-2.0.dtd">  <struts>  <package name="default" extends="struts-default">  <action name="User" class="com.tutorials4u.user.UserAction">  <result name="success">/success.jsp</result>  </action>  <action name="addUser" method="add" class="com.tutorials4u.user.UserAction">  <result name="success">/success.jsp</result>  </action>  <action name="updateUser" method="update" class="com.tutorials4u.user.UserAction">  <result name="success">/success.jsp</result>  </action>  <action name="deleteUser" method="delete" class="com.tutorials4u.user.UserAction">  <result name="success">/success.jsp</result>  </action>  </package>  </struts>  Note that we use the same action class in all the action mappings. When the request URL is "User" the execute() method in the UserAction class will be invoked. When the request URL is "addUser" the add() method in the UserAction class will be invoked, this is specified using the method attribute of the action tag. Similarly for update and delete request the updateUser() and deleteUser() methods will be invoked respectively.  Configuring a seperate action mapping for each method in the action class can be avoided by using an another feature of Struts 2 called Dynamic Method Invocation. The next example explains how to do this. ( [Dynamic Method Invocation Example](http://www.tutorials4u.net/struts2-tutorial/struts2_dynamic_method_invocation_example.html))  In the index.jsp page we create four different buttons to invoke the different methods in the UserAction class. The submit tag is used to create the buttons in the jsp page.  <%@ page language="java" contentType="text/html; charset=ISO-8859-1"  pageEncoding="ISO-8859-1"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <%@taglib uri="/struts-tags" prefix="s" %>  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">  <title>Insert title here</title>  </head>  <body>  <s:form action="User" >  <s:submit />  <s:submit action="addUser" value="Add" />  <s:submit action="updateUser" value="Update" />  <s:submit action="deleteUser" value="Delete" />  </s:form>  </body>  </html>  Now lets execute the example. On running the example the following page will be displayed to the user. When the user click a button the appropriate method in the UserAction class gets invoked.  http://www.tutorials4u.net/struts2-tutorial/images/DispatchAction1Pic1.JPG  When the user clicks the Add button the addUser() method in the UserAction class gets executed and the following page is displayed to the user.  http://www.tutorials4u.net/struts2-tutorial/images/DispatchAction1Pic2.JPG |  |
| **Dynamic Method Invocation Tutorial**  In this example you will see how you can avoid configuring a seperate action mapping for each method in the Action class by using the wildcard method.  This is a continuation of the previous example ( [DispatchAction functionality in Struts 2](http://www.tutorials4u.net/struts2-tutorial/dispatch_action_in_struts2_example) ). In this example you will see how you can avoid configuring a seperate action mapping for each method in the Action class by using the wildcard method. Look at the following action mapping. | |
| 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.        <action name="\*User" method="{1}" class="tutorials4u.UserAction">  08.            <result name="success">/success.jsp</result>  09.  10.        </action>  11.    </package>  12.</struts>  Note the similarity between the action mapping and the following request URLs in this page.  01.<%@ page language="java" contentType="text/html; charset=ISO-8859-1"  02.    pageEncoding="ISO-8859-1"%>  03.<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  04.  05.<%@taglib uri="/struts-tags" prefix="s" %>  06.<html>  07.<head>  08.<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">  09.<title>Insert title here</title>  10.</head>  11.<body>  12.    <s:form action="User" >  13.  14.        <s:submit />  15.        <s:submit action="addUser" value="Add" />  16.        <s:submit action="updateUser" value="Update" />  17.        <s:submit action="deleteUser" value="Delete" />  18.    </s:form>  19.</body>  20.  21.</html>  As you can see we have replaced all the method names with an asterisk symbol. The word that matches for the first asterisk will be substituted for the method attribute. So when the request URL is "addUser" the add() method in the UserAction class will be invoked |  |