|  |  |
| --- | --- |
| **Struts 2 UI Tags Tutorial**  In this example you wil see how to create a registration page using Struts 2 UI tags. You will also learn how to pre populate the form fields, set default values to it and to retrive the values back in the jsp page.  Struts 2 UI Tags are simple and easy to use. You need not write any HTML code, the UI tags will automatically generate them for you based on the theme you select. By default the XHTML theme is used. The XHTML theme uses tables to position the form elements.  In this example you wil see how to create a registration page using Struts 2 UI tags. You will also learn how to pre populate the form fields, set default values to it and to retrive the values back in the jsp page.  The register.jsp looks like this  http://www.tutorials4u.net/struts2-tutorial/images/UITag1Pic1.JPG | |
| The following code is used to create the register.jsp 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.<%@taglib uri="/struts-tags" prefix="s"%>  05.<html>  06.<head>  07.  08.<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">  09.<title>Register Page</title>  10.</head>  11.<body>  12.<s:form action="Register">  13.<s:textfield name="userName" label="User Name" />  14.<s:password name="password" label="Password" />  15.<s:radio name="gender" label="Gender" list="{'Male','Female'}" />  16.  17.<s:select name="country" list="countryList" listKey="countryId" listValue="countryName" headerKey="0" headerValue="Country" label="Select a country" />  18.<s:textarea name="about" label="About You" />  19.<s:checkboxlist list="communityList" name="community" label="Community" />  20.<s:checkbox name="mailingList"  21.label="Would you like to join our mailing list?" />  22.<s:submit />  23.</s:form>  24.</body>  25.</html>  If you view the source of this page you can see the HTML codes generated based on the XHTML theme.  **Struts 2 ValueStack**  Now lets understand how the UI tags work. In Struts 2 *ValueStack* is the place where the data associated with processing of the request is stored. So all the form properties will be stored on the *ValueStack*. The *name* attribute of the UI tag is the one which links the property on the *ValueStack*.  The next important attribute of the UI tag that you need to understand is the *value* attribute. If you like to populate some default value for a specific field then you need to set that *value* attribute to that value.  The following code will by default set the value of the textfield to "*Test*"  1.<s:textfield name="userName" label="User Name" value="Test"/>  Here we directly specify the value in the jsp page, instead if you want to set it throught Action then, you can have a property like *defaultName* and set its value to the desired name. In this case the code will look like this.  1.<s:textfield name="userName" label="User Name" value="defaultName"/>  The property *defaultName* is stored on the ValueStack so its value will be set to the textfield. If you think you don't need a seperate form property to do this, then you can set the *userName* property itself to the desired value. In this case you need not specify the *value* attribute seperately. In this example we populate the *community* in this way.  The value set in the *label* attribute of the UI tags will be used to render the label for that particular field while generating the HTML code.  Now lets see the flow of the example. First the index.jsp page will be invoked by the framework.  1.index.jsp  2.~~~~~~~~~  3.<META HTTP-EQUIV="Refresh" CONTENT="0;URL=populateRegister.action">  Here we forward the request to the *populateRegister* URL. Based on the mapping done in the *struts.xml* file the *populate()* method in the *RegisterAction* class will be called. Here the mapping is done using the dynamic method invocation feature of Struts 2. The struts.xml file contains the following 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="\*Register" method="{1}" class="com.tutorials4u.uitags.RegisterAction">  08.            <result name="populate">/register.jsp</result>  09.            <result name="input">/register.jsp</result>  10.  11.            <result name="success">/success.jsp</result>  12.        </action>  13.    </package>  14.</struts>  The register action class contains the form properties and the corresponding getter and setter methods. It also contains the *execute()* and *populate()* methods. In the populate method we first populate the values and then set the default values for the form fields. The *RegisterAction* class contains the following code.  001.package com.tutorials4u.uitags;  002.  003.import java.util.ArrayList;  004.  005.import com.opensymphony.xwork2.ActionSupport;  006.  007.public class RegisterAction extends ActionSupport {  008.  009.    private String userName;  010.  011.    private String password;  012.  013.    private String gender;  014.  015.    private String about;  016.  017.    private String country;  018.  019.    private ArrayList<Country> countryList;  020.  021.    private String[] community;  022.  023.    private ArrayList<String> communityList;  024.  025.    private Boolean  mailingList;  026.  027.    public String populate() {  028.  029.        countryList = new ArrayList<Country>();  030.        countryList.add(new Country(1, "India"));  031.        countryList.add(new Country(2, "USA"));  032.        countryList.add(new Country(3, "France"));  033.  034.        communityList = new ArrayList<String>();  035.        communityList.add("Java");  036.        communityList.add(".Net");  037.        communityList.add("SOA");  038.  039.        community = new String[]{"Java",".Net"};  040.        mailingList = true;  041.  042.        return "populate";  043.    }  044.  045.    public String execute() {  046.        return SUCCESS;  047.    }  048.  049.    public String getUserName() {  050.        return userName;  051.    }  052.  053.    public void setUserName(String userName) {  054.        this.userName = userName;  055.    }  056.  057.    public String getPassword() {  058.        return password;  059.    }  060.  061.    public void setPassword(String password) {  062.        this.password = password;  063.    }  064.  065.    public String getGender() {  066.        return gender;  067.    }  068.  069.    public void setGender(String gender) {  070.        this.gender = gender;  071.    }  072.  073.    public String getAbout() {  074.        return about;  075.    }  076.  077.    public void setAbout(String about) {  078.        this.about = about;  079.    }  080.  081.    public String getCountry() {  082.        return country;  083.    }  084.  085.    public void setCountry(String country) {  086.        this.country = country;  087.    }  088.  089.    public ArrayList<Country> getCountryList() {  090.        return countryList;  091.    }  092.  093.    public void setCountryList(ArrayList<Country> countryList) {  094.        this.countryList = countryList;  095.    }  096.  097.    public String[] getCommunity() {  098.        return community;  099.    }  100.  101.    public void setCommunity(String[] community) {  102.        this.community = community;  103.    }  104.  105.    public ArrayList<String> getCommunityList() {  106.        return communityList;  107.    }  108.  109.    public void setCommunityList(ArrayList<String> communityList) {  110.        this.communityList = communityList;  111.    }  112.  113.    public Boolean getMailingList() {  114.        return mailingList;  115.    }  116.  117.    public void setMailingList(Boolean mailingList) {  118.        this.mailingList = mailingList;  119.    }  120.  121.}  **Textfield and Password Tags**  Now lets see each UI tag in detail. The *textfiled* tag is used to create a textfield and *password* tag is used to create a password field. These tags are simple and uses only the common attributes discussed before.  1.<s:textfield name="userName" label="User Name" />  2.<s:password name="password" label="Password" />  **Radio Tag**  To create radio buttons we use *radio* tag. The *list* attribute of the *radio* tag is used to specify the option values. The value of the *list* attribute can be a Collection, Map, Array or Iterator. Here we use Array.  1.<s:radio name="gender" label="Gender" list="{'Male','Female'}" />  **Select Tag**  We dispaly the country dropdown using the *select* tag. Here we specify the option values using the *countryList* property of the *RegisterAction* class. The *countryList* is of type ArrayList and contain values of type *Country*. The *Country* class has *countryId* and *countryName* attribute. The *countryName* holds the country value to be display in the frontend and the *countryId* holds the id value to store it in the backend. Here *countryId* is the key and the *countryName* is the value. We specify this in the select tag using the *listKey* and *listValue* attribute. The first value can be specified using the *headerValue* attribute and the corresponding key value is specified using the *headerKey* attribute.  1.<s:select name="country" list="countryList" listKey="countryId" listValue="countryName" headerKey="0" headerValue="Country"  2.label="Select a country" />  **Textarea Tag**  The *textarea* tag is used to create a textarea.  1.<s:textarea name="about" label="About You" />  **Checkboxlist Tag**  The *checkboxlist* tag is similar to that of the *select* tag, the only difference is that it displays boxes for each option instead of a dropdown. It returns an array of String values.  1.<s:checkboxlist list="communityList" name="community" label="Community" />  **Checkbox Tag**  The *checkbox* tag returns a boolean value. If the checkbox is checked then true is returned else false is returned.  1.<s:checkbox name="mailingList" label="Would you like to join our mailing list?" />  **Submit Tag**  The *submit* tag is used to create the Submit button  1.<s:submit />  Now lets enter the details and submit the form. The *execute()* method in the *RegisterAction* class will be invoked this time and the user will be forwarded to the *success.jsp* page.  01.success.jsp  02.-----------  03.<%@ page language="java" contentType="text/html; charset=ISO-8859-1"  04.pageEncoding="ISO-8859-1"%>  05.<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  06.<%@taglib uri="/struts-tags"  prefix="s"%>  07.<html>  08.<head>  09.<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">  10.<title>Details Page</title>  11.  12.</head>  13.<body>  14.User Name: <s:property value="userName" /><br>  15.Gender: <s:property value="gender" /><br>  16.Country: <s:property value="country" /><br>  17.About You: <s:property value="about" /><br>  18.  19.Community: <s:property value="community" /><br>  20.Mailing List: <s:property value="mailingList" />  21.</body>  22.</html>  Now lets enter the following details and submit the form.  http://www.tutorials4u.net/struts2-tutorial/images/UITag1Pic2.JPG  The following registration details will be displayed to the user.  http://www.tutorials4u.net/struts2-tutorial/images/UITag1Pic3.JPG |  |
| **Struts 2 Data Tags Tutorial**  In this tutorial you will learn about the ValueStack and the ActionContext and then see how to use the property tag, set tag and push tag to store and retrieve data.  In this example you will learn how to use the property tag, the set tag and the push tag. These tags are part of the Struts 2 Data Tags. Before we see the syntax of each tag you need to know what an ActionContext and a ValueStack is.  http://www.tutorials4u.net/struts2-tutorial/images/ValueStackPic1.gif   * The ActionContext is a global storage area that holds all the data associated with the processing of a request. * The ActionContext is thread local this makes the Struts 2 actions thread safe. * The ValueStack is the part of the ActionContext. In Struts 2 actions resides on the ValueStack. | |
| The Property Tag The property tag is used to retrive the data from the ValueStack or some other object in the ActionContext like application or seesion. Let's see how to display the following details using the property tag.  http://www.tutorials4u.net/struts2-tutorial/images/DataTagPic1.JPG  Our Action class AlbumInfoAction contains the following piece of code.  01.package tutorials4u;  02.  03.public class AlbumInfoAction{  04.  05.    private String title;  06.    private Artist artist;  07.  08.    public String populate()  09.    {  10.        title = "Thriller";  11.        artist = new Artist("Michael Jackson","King of pop");  12.        return "populate";  13.    }  14.  15.    public String execute()  16.    {  17.        return "success";  18.    }  19.  20.    public String getTitle() {  21.        return title;  22.    }  23.    public void setTitle(String title) {  24.        this.title = title;  25.    }  26.    public Artist getArtist() {  27.        return artist;  28.    }  29.    public void setArtist(Artist artist) {  30.        this.artist = artist;  31.    }  32.  33.}  Our Artist data class contains the following code.  01.package tutorials4u;  02.  03.public class Artist {  04.  05.    private String name;  06.    private String bio;  07.  08.    Artist(String name, String bio)  09.    {  10.        this.name = name;  11.        this.bio = bio;  12.    }  13.    public String getName() {  14.        return name;  15.    }  16.    public void setName(String name) {  17.        this.name = name;  18.    }  19.    public String getBio() {  20.        return bio;  21.    }  22.    public void setBio(String bio) {  23.        this.bio = bio;  24.    }  25.  26.}  Let's see how we can access the action class attributes using the property tag in the jsp page. The albumDetails.jsp page contains the following code.  01.<%@taglib uri="/struts-tags" prefix="s"%>  02.<html>  03.<head>  04.<s:head />  05.<style type="text/css">  06.@import url(style.css);  07.</style>  08.<title>Album Details</title>  09.  10.</head>  11.<body>  12.<div class="content">  13.    <b>Album Title :</b>  14.    <s:property value="title" /> <br>  15.    <b>Artist Name :</b>  16.  17.    <s:property value="artist.name" />  18.    <br>  19.    <b>Artist Bio :</b>  20.    <s:property value="artist.bio" />  21.    <br>  22.  23.</div>  24.</body>  25.</html> |  |
| As you can see title is the property of the AlbumInfoAction so we can access it directly. But name and bio are properties of the Artist class so to access them we need to go one step deeper. We need to use a second-level OGNL expression to access them.  **Struts 2 Set Tag Example**  The set tag is used to assign a property value to another name. This helps in accessing the property in a faster and easier way. To access the artist name we need to go one level deeper and fetch it when we used the property tag, instead you can assign the value **to another property in the ActionContext and access it directly**. The following code shows how to do this.  1.<s:set name="artistName" value="artist.name" />  2.<s:set name="artistBio" value="artist.bio" />  3.<b>Album Title :</b> <s:property value="title" /> <br>  4.<b>Artist Name :</b> <s:property value="#artistName" /> <br>  5.  6.<b>Artist Bio :</b> <s:property value="#artistBio" /> <br> |  |
| The property artistName and artistBio will now be stored in the ActionContext. To refer then you need to use the following syntax **#objectName**.  You can also place the property value in the session map in the following way. Now the value artistName and artistBio will persist throughout the session.  1.<s:set name="artistName" value="artist.name" scope="session" />  2.<s:set name="artistBio" value="artist.bio" scope="session" />  3.<b>Album Title :</b> <s:property value="title" /> <br>  4.<b>Artist Name :</b> <s:property value="#session['artistName']" /> <br>  5.  6.<b>Artist Bio :</b> <s:property value="#session['artistBio']" /> <br>  In the same way you can also store the values in other maps avaliable in the ActionContext.  **Struts 2 Push Tag Example**  You can **push a value into the ValueStack** using the push tag. The value we pushed using push tag will be on top of the ValueStack, so it can be easily referenced using the first-level OGNL expression instead of a deeper reference. The following code show how to do this.  1.<b>Album Title :</b> <s:property value="title" /> <br>  2.<s:push value="artist">  3.<b>Artist Name :</b> <s:property value="name" /> <br>  4.  5.<b>Artist Bio :</b> <s:property value="bio" /> <br>  6.</s:push> |  |
| **Struts 2 Bean Tag Tutorial**  In this tutorial you will see how to use the Struts 2 Bean Tag with the help of a CurrencyConverter bean example.  We will see how the bean tag works using a currency converter example. In this example we will convert dollars to rupees. We do this using the CurrencyConverter JavaBeans class.  The CurrencyConverter class. | |
| 01.package tutorials4u;  02.  03.public class CurrencyConverter {  04.  05.    private float rupees;  06.    private float dollars;  07.  08.    public float getRupees() {  09.            return dollars \* 50;  10.    }  11.    public void setRupees(float rupees) {  12.            this.rupees = rupees;  13.    }  14.    public float getDollars() {  15.            return rupees/50 ;  16.    }  17.    public void setDollars(float dollars) {  18.            this.dollars = dollars;  19.    }  20.  21.}  The next step is to create an instance of the CurrencyConverter bean in the jsp page using the bean tag. We can either use the bean tag to push the value onto the ValueStack or we can set a top-level reference to it in the ActionContext. Let's see one by one.  First we will see how we can do this by pushing the value onto the ValueStack. The index.jsp page contains the following code.  01.<html>  02.<head>  03.<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">  04.<title>Bean Tag Example</title>  05.  06.</head>  07.<body>  08.<s:bean name="tutorials4u.CurrencyConverter">  09.    <s:param name="dollars" value="100" />  10.    100 Dollars = <s:property value="rupees" /> Rupees  11.</s:bean>  12.  13.</body>  14.</html>  The name attribute of the bean tag hold the fully qualified class name of the JavaBean. The param tag is used to set the dollar value. We now use the property tag to retrive the equivalent value in rupees.  The CurrencyConverter bean will resides on the ValueStack till the duration of the bean tag. So its important that we use the property tag within the bean tag.  When you execute the example the following page is displayed.  http://www.tutorials4u.net/struts2-tutorial/images/BeanTagPic1.JPG  Now we will see how we can do the same by creating a top-level reference to the bean in the ActionContext. We do this using the following code.  01.<html>  02.<head>  03.<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">  04.<title>Bean Tag Example</title>  05.</head>  06.<body>  07.  08.<s:bean name="tutorials4u.CurrencyConverter" var="converter">  09.    <s:param name="dollars" value="100"></s:param>  10.</s:bean>  11.100 Dollars = <s:property value="#converter.rupees" /> Rupees  12.  13.</body>  14.</html> |  |