Struts Code Peaces – <html:options> element

We explain the struts <html:options> element and illustrate the usage with some small examples.

Generals

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

Datum:
February 22th 2005

The <html:options> element

The *<html:options>* element is only valid when nested inside a *<html:select>* element. It renders a HTML *<option>* element. The element is used to display data of lists (arrays, collections) inside a select element. This tag can be used multiple times within a single <html:select> element.

The following example shows the source code of the JSP file.

```
<html:select property="selectedItem">
      <html:options collection="customers" property="id" labelProperty="name" />
</html:select>
```

The following HTML source code is rendered at runtime.

```
<select name="selectedItem">
      <option value="1">Marie</option>
      <option value="2">Klaus</option>
</select>
```

Attributes of the <html:select> element

Now the most important attribute will be explained. You find a complete list of all available attributes for this tag in the API of the HTML tag library.

http://struts.apache.org/userGuide/struts-html

Name	Beschreibung
collection	Name of an property which holds a collection of beans.
labelName	Name of the bean which contain a collection of labels
labelProperty	Name of the collection inside the bean specified by the labelName attributes
name	Name of the bean which contain the properties.
property	Specified the property of the bean, specified by the name attribute, which holds the collection of values. This values will be submitted to the server.

Usage of the <html:options> element

Create a new struts project to illustrate the usage of the <*html:options*> element.

Create a object class

Create a new java class customer in the package *de.laliluna.tutorial.options.object*. This class represents a customer.

Create two properties, *id* of type int and *name* of type String and provide a getter and setter method for each of this properties.

Define a constructor which allows you to set the both properties if you initialize the class.

The object class looks like the following:

```
public class Customer {

	private int id;
	private String name;

	public Customer(){}

	public Customer(int id, String name){
		this.id = id;
		this.name = name;
	}

	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
}
```

Create a new form bean

Create a action form class *ExampleForm* in the package *de.laliluna.tutorial.options.form*.
Define a property of type String, which is associated with *<html:select>* element. This property holds the value, which is submitted by the element.

Define two properties of type Collection. The property *customerValues* contains the values of the customers. The property *customerLabel* contains the labels of the customers.

Provide some dummy data in the getter method for each of the properties.

The following source code shows the action form class:

```
public class ExampleForm extends ActionForm {

	private String selectedItem;
	private Collection customerLabels;
	private Collection customerValues;

	public String getSelectedItem() {
		return selectedItem;
	}
	public void setSelectedItem(String selectedItem) {
		this.selectedItem = selectedItem;
	}

	public Collection getCustomerLabels() {

		// define some dummy labels
		customerLabels = new ArrayList();
		customerLabels.add("Marie");
		customerLabels.add("Klaus");
		customerLabels.add("Peter");

		return customerLabels;
	}
	public void setCustomerLabels(Collection customerLabels) {
		this.customerLabels = customerLabels;
```

```
      }
      public Collection getCustomerValues() {

            // define some dummy names
            customerValues = new ArrayList();
            customerValues.add("1");
            customerValues.add("2");
            customerValues.add("3");

            return customerValues;
      }
      public void setCustomerValues(Collection customerValues) {
            this.customerValues = customerValues;
      }
}
```

Create a new action class

Create a new action class *ExampleAction* in the package *de.laliluna.tutorial.options.action* an. We provide some dummy data as collection.

```
public class ExampleAction extends Action {

      public ActionForward execute(
            ActionMapping mapping,
            ActionForm form,
            HttpServletRequest request,
            HttpServletResponse response) {
            ExampleForm exampleForm = (ExampleForm) form;

            //define a dummy collection
            Collection customers = new ArrayList();
            customers.add(new Customer(1, "Marie"));
            customers.add(new Customer(2, "Klaus"));
            customers.add(new Customer(3, "Peter"));

            //set the collection in the request
            request.setAttribute("customers", customers);


            return mapping.findForward("success");
      }
}
```

Create the struts-config.xml

Now open the struts-config.xml and specify the form bean and the action mapping.

```
<struts-config>
   <form-beans>
      <form-bean name="exampleForm"
type="de.laliluna.tutorial.options.form.ExampleForm" />
   </form-beans>

   <action-mappings>
      <action
         name="exampleForm"
         path="/example"
         scope="request"
         type="de.laliluna.tutorial.options.action.ExampleAction">
         <forward name="success" path="/form/example.jsp" />
      </action>
   </action-mappings>
</struts-config>
```

Create the jsp file

Create a jsp file named *example.jsp* in the folder */WebRoot/form/*.

Open the jsp file *example.jsp* and add the following source code.

```
<%@ page language="java"%>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-bean" prefix="bean"%>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html"%>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-logic" prefix="logic"%>

<html>
      <head>
            <title>example.jsp</title>
      </head>
      <body>
            <html:form action="/example">

            .... sample code ...

            </html:form>
      </body>
</html>
```

Within the *<html:form>* element add the first example.

Example 1

The attribute *property* of the *<html:select>* element specifes the associated property *selectedItem* of the form bean. This property holds the value of the selected option, when the form is submitted. The attribute *collection* of the *<html:options>* element refers to the collection, which is saved in the request.

The attribute *property* specifies the value, which is submitt to the server if the option is selected. In our case the property is the field *id* of the object class. The attribute *labelProperty* refers to the property *name* of the object class and is used to display a label for this option.

```
<h4>Use collection attribute of the &lt;html:options&gt; Tag</h4>

<html:select property="selectedItem">
      <html:options collection="customers" property="id" labelProperty="name" />
</html:select>

<html:submit/>
```

Example 2

In the second example we do not use the collection attribute. We refer directly to a collection of the form bean with the attribute *property*, which holds a list of values. The attribute *labelProperty* specifes the collection, which holds the label for each element.

```
<h4>Use property attribute of the &lt;html:options&gt; Tag</h4>

<html:select property="selectedItem">
      <html:options property="customerValues" labelProperty="customerLabels" />
</html:select>

<html:submit/>
```

Now you can test the project. We use a jboss or tomcat installation. Test the project with the following link.

http://localhost:8080/OptionsTag/example.do