|  |  |
| --- | --- |
| **Domain Object as JavaBeans Property Tutorial**  In this tutorial you will learn how to use a application domain object as the FormBeans property. You will see how the params interceptor help in automatically transferring the form property values to the application domain object and how to retrieve the values back in the jsp page using OGNL expression language.  In Struts 2 transferring the data to the domain object is automatically done by the **params** interceptor.  You just need to create a domain object as a JavaBeans property and the corresponding getter and setter methods.  The framework will automatically initializes the domain object and transfers the form data. The UserAction class contains the following code. | |
| 01.public class UserAction extends ActionSupport{  02.  03.    private User user;  04.  05.    public UserAction() {  06.    }  07.  08.    public String execute() {  09.        return SUCCESS;  10.    }  11.  12.    public User getUser() {  13.        return user;  14.    }  15.  16.    public void setUser(User user) {  17.        this.user = user;  18.    }  19.} |  |
| To refer the user attributes like name, age etc. we need to first get the user object and then access its properties. For example to access the user's age in the Action you need to use the following syntax.  1.getUser().getAge();  The User class contains the following attributes and the corresponding getter and setter methods.  01.public class User {  02.  03.    private String name;  04.    private int age;  05.    private String sex;  06.    private String[] hobby;  07.    private String country;  08.  09.  10.    public String getName() {  11.        return name;  12.    }  13.  14.    public void setName(String name) {  15.        this.name = name;  16.    }  17.  18.    public int getAge() {  19.        return age;  20.    }  21.  22.    public void setAge(int age) {  23.        this.age = age;  24.    }  25.  26.    public String getSex() {  27.        return sex;  28.    }  29.  30.    public void setSex(String sex) {  31.        this.sex = sex;  32.    }  33.  34.    public String[] getHobby() {  35.        return hobby;  36.    }  37.  38.    public void setHobby(String[] hobby) {  39.        this.hobby = hobby;  40.    }  41.  42.    public String getCountry() {  43.        return country;  44.    }  45.  46.    public void setCountry(String country) {  47.        this.country = country;  48.    }  49.}  In the jsp page the user attributes cannot be directly referenced. Since the attributes we refer in the jsp page belongs to the User object we need to go one level deeper to reference the attributes. To refer the user's age, the value of the name attribute should be  1.name="user.age"  The index.jsp page contains the following code.  01.<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  02."http://www.w3.org/TR/html4/loose.dtd">  03.<%@taglib uri="/struts-tags" prefix="s" %>  04.<html>  05.<head>  06.  07.<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  08.<title>User Details</title>  09.</head>  10.<body>  11.    <s:form action="UserAction" >  12.  13.        <s:textfield name="user.name" label="User Name" />  14.        <s:textfield name="user.age" label="Age" />  15.  16.        <s:radio name="user.sex" label="Sex" list="{'M','F'}" />  17.        <s:checkboxlist name="user.hobby" label="Hobby" list="{'Music','Art','Dance'}" />  18.  19.        <s:select name="user.country" label="Country" list="{'Select','India','USA','France', 'Spain'}"  />  20.        <s:submit />  21.    </s:form>  22.  23.</body>  24.</html>  The result.jsp page contains the follwing code.  01.<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  02."http://www.w3.org/TR/html4/loose.dtd">  03.<%@taglib uri="/struts-tags" prefix="s" %>  04.  05.<html>  06.<head>  07.<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  08.<title>User Details</title>  09.</head>  10.<body>  11.  12.    <h2>User Details</h2>  13.    <hr>  14.    User Name :<s:property value="user.name" /><br>  15.    Age :<s:property value="user.age" /><br>  16.  17.    Hobbies :<s:property value="user.hobby" /><br>  18.    Country :<s:property value="user.country" /><br>  19.</body>  20.  21.</html>  On executing the example the following page will be displayed to the user.  http://www.tutorials4u.net/struts2-tutorial/images/UserDetails1Pic1.JPG  On entering the user details and clicking the Submit button the following page will be dispalyed.  http://www.tutorials4u.net/struts2-tutorial/images/UserDetails1Pic2.JPG |  |
| **Struts 2 ModelDriven Action Tutorial**  In this tutorial you will lean how to create an Action implementing the ModelDriven interface. You will see how to implement the getModel() method and to retrieve the values back in the jsp page using the OGNL expression language.  To create a ModelDriven Action our Action class should implement the **ModelDriven interface** and should include the **modelDriven interceptor**. The modelDriven interceptor is already included in the default stack.  The next step is to implement the **getModel()** method in such a way that it returns the application domain object, in our example we return the User object.  When using the ModelDriven method we need to initialize the User object ourselves.  The framework will automatically transfers the form data into the User object. | |
| 01.public class UserAction extends ActionSupport implements ModelDriven {  02.  03.    private User user = new User();  04.  05.    public UserAction() {  06.    }  07.  08.    public Object getModel() {  09.        return user;  10.    }  11.  12.    public String execute() {  13.        return SUCCESS;  14.    }  15.  16.    public User getUser() {  17.        return user;  18.    }  19.  20.    public void setUser(User user) {  21.        this.user = user;  22.    }  23.} |  |
| You can directly access the user attributes like name, age etc in Action use the following syntax.  1.user.getXXX();  The User class contains the following attributes and the corresponding getter and setter methods.  01.public class User {  02.  03.    private String name;  04.    private int age;  05.    private String sex;  06.    private String[] hobby;  07.    private String country;  08.  09.  10.    public String getName() {  11.        return name;  12.    }  13.  14.    public void setName(String name) {  15.        this.name = name;  16.    }  17.  18.    public int getAge() {  19.        return age;  20.    }  21.  22.    public void setAge(int age) {  23.        this.age = age;  24.    }  25.  26.    public String getSex() {  27.        return sex;  28.    }  29.  30.    public void setSex(String sex) {  31.        this.sex = sex;  32.    }  33.  34.    public String[] getHobby() {  35.        return hobby;  36.    }  37.  38.    public void setHobby(String[] hobby) {  39.        this.hobby = hobby;  40.    }  41.  42.    public String getCountry() {  43.        return country;  44.    }  45.  46.    public void setCountry(String country) {  47.        this.country = country;  48.    }  49.}  In the jsp page the user attributes can be accessed directly. To refer the user's age, the value of the name attribute should be  1.name = "age"  The index.jsp page contains the following code.  01.<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  02."http://www.w3.org/TR/html4/loose.dtd">  03.<%@taglib uri="/struts-tags" prefix="s" %>  04.  05.<html>  06.<head>  07.<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  08.<title>User Details</title>  09.</head>  10.<body>  11.  12.<s:form action="UserAction" >  13.    <s:textfield name="name" label="User Name" />  14.    <s:textfield name="age" label="Age" />  15.  16.    <s:radio name="sex" label="Sex" list="{'M','F'}" />  17.    <s:checkboxlist name="hobby" label="Hobby"  18.  19.    list="{'Music','Art','Dance'}" />  20.    <s:select name="country" label="Country"  21.    list="{'Select','India','USA','France','Spain'}"  />  22.  23.    <s:submit />  24.</s:form>  25.</body>  26.</html>  The result.jsp page contains the follwing code.  01.<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  02.  03."http://www.w3.org/TR/html4/loose.dtd">  04.<%@taglib uri="/struts-tags" prefix="s" %>  05.<html>  06.<head>  07.<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  08.  09.<title>User Details</title>  10.</head>  11.<body>  12.    <h2>User Details</h2>  13.    <hr>  14.    User Name :<s:property value="name" /><br>  15.  16.    Age :<s:property value="age" /><br>  17.    Hobbies :<s:property value="hobby" /><br>  18.    Country :<s:property value="country" /><br>  19.  20.</body>  21.</html>  On executing the example the following page will be displayed to the user.  http://www.tutorials4u.net/struts2-tutorial/images/UserDetails1Pic1.JPG  On entering the user details and clicking the Submit button the following page will be dispalyed.  http://www.tutorials4u.net/struts2-tutorial/images/UserDetails1Pic2.JPG |  |

|  |  |
| --- | --- |
| **Struts 2 File Upload Tutorial**  Learn how to upload a file in Struts 2 with the help of the built-in FileUploadInterceptor. You will also see how to validate a file based on the file size and the content type.  In this example you will learn how to do file upload with the help of the built-in FileUploadInterceptor. To do this first we need to get the file form the user. We use the Struts 2 tags to build our form. The encoding type of the form should be set to multipart/form-data and the HTTP method should be set to post. The index.jsp page contains the following code. | |
| 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.<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">  08.  09.<title>Insert title here</title>  10.<s:head />  11.</head>  12.<body>  13.    <s:form action="fileUpload" method="post" enctype="multipart/form-data" >  14.        <s:file name="userImage" label="User Image" />  15.        <s:submit />  16.  17.    </s:form>  18.</body>  19.</html>  Here our file name is userImage. Next we will create our FileUploadAction class. We need to have a File attribute and the corresponding getter and setter methods in the action to receive the userImage file.  You will also get information regarding the file name and content type of the file if you implement the following setter methods. This step is optional if you implement the setter methods you will get more details regarding the file.  1.public void setUserImageContentType(String userImageContentType) {  2.    this.userImageContentType = userImageContentType;  3.}  4.public void setUserImageFileName(String userImageFileName) {  5.    this.userImageFileName = userImageFileName;  6.}  The FileUploadAction extends ActionSupport and contains the following code.  01.package vaannila;  02.  03.import java.io.File;  04.  05.import com.opensymphony.xwork2.ActionSupport;  06.  07.public class FileUploadAction extends ActionSupport{  08.  09.    private File userImage;  10.  11.    private String userImageContentType;  12.  13.    private String userImageFileName;  14.  15.    public String execute()  16.    {  17.        return SUCCESS;  18.    }  19.  20.    public File getUserImage() {  21.        return userImage;  22.    }  23.  24.    public void setUserImage(File userImage) {  25.        this.userImage = userImage;  26.    }  27.  28.    public String getUserImageContentType() {  29.        return userImageContentType;  30.    }  31.  32.    public void setUserImageContentType(String userImageContentType) {  33.        this.userImageContentType = userImageContentType;  34.    }  35.  36.    public String getUserImageFileName() {  37.        return userImageFileName;  38.    }  39.  40.    public void setUserImageFileName(String userImageFileName) {  41.        this.userImageFileName = userImageFileName;  42.    }  43.  44.}  When the file is uploaded successful, the user will be forwarded to the success page, where the details regarding the file upload will be displayed. The success.jsp page contains the following code.  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.<html>  05.<head>  06.<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">  07.<title>File Upload</title>  08.</head>  09.<body>  10.  11.    You have uploaded the following file.  12.    <hr>  13.    File Name : ${userImageFileName} <br>  14.    Content Type : ${userImageContentType}  15.</body>  16.</html>  Now let's run the example.  http://www.tutorials4u.net/struts2-tutorial/images/FileUpload1Pic1.JPG  When the user selects an image file and upload it. The following success page will be displayed to the user.  http://www.tutorials4u.net/struts2-tutorial/images/FileUpload1Pic4.JPG  We only need an image file, what if the user uploads a pdf document? Let's see how to validate this in the next page.  We can validate either programmatically or declaratively. Let's see how to do programmatically first. Validate the file in the validate() method of the FileUploadAction class. You can get the file using the getX() method and the file type type using the getXContentType() method.  In this example we validate declaratively. To do this we need to create our own interceptor stack. The only change we need to do is to add few parameters to the fileUpload interceptor. So we copy the defaultStack from the struts-default.xml and paste it in our struts.xml file and rename the stack to fileUploadStack. Our 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="fileUploadPackage" extends="struts-default">  07.        <interceptors>  08.            <interceptor-stack name="fileUploadStack">  09.                <interceptor-ref name="exception" />  10.                <interceptor-ref name="alias" />  11.  12.                <interceptor-ref name="servletConfig" />  13.                <interceptor-ref name="prepare" />  14.                <interceptor-ref name="i18n" />  15.                <interceptor-ref name="chain" />  16.                <interceptor-ref name="debugging" />  17.                <interceptor-ref name="profiling" />  18.  19.                <interceptor-ref name="scopedModelDriven" />  20.                <interceptor-ref name="modelDriven" />  21.                <interceptor-ref name="fileUpload">  22.                    <param name="maximumSize">10240</param>  23.                    <param name="allowedTypes"> image/jpeg,image/gif,image/png</param>  24.  25.                </interceptor-ref>  26.                <interceptor-ref name="checkbox" />  27.                <interceptor-ref name="staticParams" />  28.                <interceptor-ref name="actionMappingParams" />  29.                <interceptor-ref name="params">  30.                    <param name="excludeParams"> dojo\..\*,^struts\..\*</param>  31.  32.                </interceptor-ref>  33.                <interceptor-ref name="conversionError" />  34.                <interceptor-ref name="validation">  35.                    <param name="excludeMethods"> input,back,cancel,browse</param>  36.                </interceptor-ref>  37.  38.                <interceptor-ref name="workflow">  39.                    <param name="excludeMethods"> input,back,cancel,browse</param>  40.                </interceptor-ref>  41.            </interceptor-stack>  42.        </interceptors>  43.  44.        <action name="fileUpload" class="vaannila.FileUploadAction">  45.            <interceptor-ref name="fileUploadStack" />  46.            <result name="input">/index.jsp</result>  47.            <result name="success">/success.jsp</result>  48.  49.        </action>  50.    </package>  51.</struts>  The maximumSize value is set in bytes. Here we set the maximumSize to 10kb. The allowedTypes indicate the file types that can be uploaded. Here we set it to only image files like image/jpeg,image/gif,image/png.  Now lets see how the validation works. We will upload a text file. The following error message is displayed to the user.  http://www.tutorials4u.net/struts2-tutorial/images/FileUpload1Pic2.JPG  Now we will upload a file greater than 10kb, the following error message will be displayed.  http://www.tutorials4u.net/struts2-tutorial/images/FileUpload1Pic3.JPG |  |