Jakarta Struts
A beginner's tutorial

Isabelle HURBAIN

21st October 2002

Contents

Chapter 1

Introduction

Struts is an open source framework useful in building web applications with Java Servlet and JavaServer Pages (JSP) technology. It encourages software development following the MVC design pattern.

1.1 A brief overview

Many web applications are JSP-only or Servlets-only. With JSP, Java code is embedded in the HTML code; with Servlets the Java code calls println methods to generate the HTML code.

Both approaches have their advantages and drawbacks; Struts gathers their strengths to get the best of their association.

1.2 The MVC design pattern and its application to Struts

The MVC design pattern divides applications into three components:

- the model maintains the state and data that the application represents
- the view allows the display of information about the model to the user
- the controller allows the user to manipulate the application

In Struts, the view is handled by JSPs and presentation components, the model is represented by Java Beans and the controller uses Servlets to perform its action.

1.3 Who should read this tutorial

This tutorial is for people who want to learn Struts from scratch - that is, from server installation to operational knowledge. It will explain how to setup a whole Struts application using Tomcat 4.0.4, Struts 1.0.2 and Eclipse 2.0.

I do not guarantee that what I explain in this tutorial is the best way or even a correct way to do things. Particularly, I'm not an Eclipse guru and my way to do things can seem weird. It is at least my way works;-)

Chapter 2

Installation

Before we begin an application, we must install all the needed stuff and configure it so that it is useable.

2.1 JDK

In order to do any Java development you need the Java Developer Kit. You can find a JDK1.4 at http://java.sun.com.

2.2 Win32 installation

Just execute the installation executable and follow the instructions.

You will also need to setup the PATH and the JAVA_HOME variables; to do this, you can write a little batch file called for example javasetup.bat and put it into a directory in your execution path.

Mine looks like this:

```
set JAVA_HOME=C:\j2sdk1.4.0_01

set PATH=%PATH%;%JAVA_HOME%\bin
```

2.2.1 Linux installation

Just execute the installation .bin and follow the instructions. (You may need to su root).

You will also need to setup the PATH and JAVA_HOME variables; to do this, you can write a little shell script or put it into your .profile.

My shell script looks like this (in Bash):

```
export JAVA_HOME=/usr/java/j2sdk1.4.0_01

export PATH=$PATH:$JAVA_HOME/bin
```

2.3 Tomcat

Tomcat is a JSP/Servlet container - a web server that will allow you to use JSPs and servlets in your application. You will find it as a subproject of the Jakarta project (http://jakarta.apache.org).

2.3.1 Win32 Installation

To install Tomcat on a Win32 platform, you just have to download the binary distribution and to decompress it into a directory of your choice.

To run it, check that your JAVA_HOME and PATH variables are set and launch the "startup.bat" script in the bin directory of your Tomcat directory. You might need to add a CATALINA_HOME variable to your batch file pointing to your Tomcat directory (I will call this directory this way from now on).

2.3.2 Linux Installation

To install Tomcat on a Linux platform, you can also use the binary distribution and un-tar.gz it into a directory of your choice. You can also compile it from sources - to do that, read the documentation provided with Tomcat.

Then check that your JAVA_HOME and PATH variables are set and launch the "startup.sh" script in the bin directory of your Tomcat directory. You might need to add a CATALINA_HOME variable to your shell script pointing to your Tomcat directory (I will call this directory this way from now on).

2.3.3 Configuration

You'll have to add a line to the file tomcat-users.xml in the conf subdirectory of your $CATALINA_HOME. This line will allow you to access Tomcat's application manager. The line looks like this:

```
<user name="myname"    password="mypassword" roles="standard,manager" />
```

where `myname` and `mypassword` are values you choose for your manager login and password.

After this modification, you will have to relaunch Tomcat by calling the shutdown script (.bat or .sh, depending on your system), waiting a bit for the whole Tomcat to close, and relaunch startup.

Check that everything is okay by accessing http://localhost:8080/ with your favourite browser - you should see a Tomcat page. If you don't, please refer to Tomcat's installation documentation.

Also try http://localhost:8080/manager/ to check that your login and password are working. It should display

```
FAIL - Unknown command /
```

IMPORTANT WARNING

The configuration as described here is only a *development* configuration and should be hardened on a production server!

2.4 Eclipse

Eclipse is a Java IDE, you can find it at http://www.eclipse.org.

On Linux and Win32, you'll just have to unzip / untargz the package and execute eclipse(.exe on Win32) to complete the installation (basically the creation of the workspace).

Eclipse isn't a requirement at all for Struts development, it is just the IDE I use. If you feel more comfortable with another one or without one, it should be relatively easy to adapt the tutorial to it.

2.5 Struts

Struts can also be found as a subproject of the Jakarta project (http://jakarta.apache.org). There is no Struts installation for the moment, just uncompress it in a convenient directory.

Chapter 3

Learning by example: your first Struts application

3.1 Presentation

We will develop a basic web application: it will allows users to register and to see their personal data . This application will have 5 pages:

- a main menu
- a registration form
- a confirmation of registration
- a request information form
- a display of the wanted information

The data will be held in a Vector within the application's scope, so that we do not have to bother with a database or falt files (this is just an example and the aim of this tutorial is to concentrate on Struts, not Java).

3.2 Creating the application workspace

Prior to launching Eclipse, we still have a few things to do in order to setup the web application:

- Create a directory in the webapps directory of your $CATALINA_HOME (which we will refer to $APP_BASE). Call it registeruser.
- In your $APP_BASE, create a directory WEB-INF
- In $APP_BASE/WEB-INF/, create a directory classes and a directory lib
- Copy struts.jar from the struts distribution (in the lib directory) to the $APP_BASE/WEB-INF/lib/ directory
- Copy all the struts*.tld files and the struts-config_1_0.dtd file from the struts distribution (in the lib directory) to the $APP_BASE/WEB-INF/ directory
- Copy $CATALINA_HOME/common/lib/servlets.jar to the $APP_BASE/WEB-INF/lib/ directory

- Create a $APP_BASE/src directory.

Now it's time to open Eclipse. When you first open Eclipse, you have 4 windows, called views. I will not speak any further about views and perspectives - each view and what it does are pretty well documented in Eclipse documentation.

The main view contains a help file, you can read it or close it:).

We want to create a new application. To do this, click on File → New → Project... . A window opens, in which we select Java in the first frame and Java Project in the second frame. Click on Next.

Type a project name (like registeruser) in the Project Name field. Uncheck the box "Use defaults" for the Project Content, and browse your disk to set the directory to $APP_BASE. Click on Next, a message box asks you if you want to create the project now, click on Yes.

On the Source tab, set the "Build output folder" to $APP_BASE/WEB-INF/classes. Click on "Use source folders contained in the project", and then on "Add existing folders" and there select $APP_BASE/src.

On the Librairies tab, click on Add external JARs and add the two JARs of $APP_BASE/WEB-INF/lib/. Click on Finish.

Open the Resource perspective with Window → Open Perspective → Resource.

We also have to import the classes for Struts and for Java Servlets. Right-click on $APP_BASE/src in the navigator and select Import. Select Zip File and Next. Go to $APP_BASE/WEB-INF/lib and select struts.jar. Click on Next. Repeat the operation with the file $CATALINA_HOME/common/lib/servlet.jar.

You also have to create a web.xml file, which should look like this:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

<web-app>

  <!-- Action Servlet Configuration -->
  <servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>

    <!-- Resources bundle base class -->
    <init-param>
      <param-name>application</param-name>
      <param-value>ApplicationResources</param-value>
    </init-param>

    <!-- Context-relative path to the XML resource containing Struts configuration information -->
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>

    <!-- The debugging detail level for this servlet, which controls how much information is logged. -->
    <init-param>
      <param-name>debug</param-name>
      <param-value>2</param-value>
    </init-param>

    <load-on-startup>2</load-on-startup>
  </servlet>


  <!-- Action Servlet Mapping -->
  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>


  <!-- The Welcome File List -->
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

  <!-- Application Tag Library Descriptor -->
  <taglib>
    <taglib-uri>/WEB-INF/app.tld</taglib-uri>
    <taglib-location>/WEB-INF/app.tld</taglib-location>
  </taglib>

  <!-- Struts Tag Library Descriptors -->
  <taglib>
    <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
  </taglib>

  <taglib>
    <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
  </taglib>

  <taglib>
    <taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
  </taglib>

</web-app>
```

The project is now created, it's time to check that everything is all right.

Restart Tomcat and go to http://localhost:8080/registeruser/ (if $CATALINA_HOME/webapps/registeruser is your $APP_BASE, else adapt it to your case. I will use http://localhost:8080/registeruser/). You should see a directory listing with files .classpath, .project, and a src directory. It is normal that you don't see the WEB-INF directory.

3.3 The first page

3.3.1 First draft

Back to Eclipse, right-click on registeruser to add a new file. Call it index.jsp, and edit it to get the following result:

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>

<html:html>
<head>
    <title>My first Struts application!</title>
    <html:base/>
</head>

<body>
This is my first Struts application.
</body>
</html:html>
```

Do not forget to close all your tags! You can do it either with a closing tag (like in `<title></title>`) or with a closed tag (like in `<html:base/>`).

Save the file, reload http://localhost:8080/registeruser/ and admire the result;-)

OK, it's not that impressive. Let's talk about the code. The first line is a requirement to tell Tomcat that this page is a JSP and that it should treat it as such. The `taglib` lines import new tags, such as the `<html:html>` that you can see below. The html taglib could be the only one to insert - however I just copy and paste the whole beginning of the document so that I'm sure I do not forget anything;-)

The rest of the script is rather classical, except maybe for `<html:html>` (that renders a `<html>` tag) and `<html:base>` (that renders a `<base>` tag to point the absolute reference of the current file).

3.3.2 A bit of internationalization

It is really easy to create internationalized web application that display things according to the language of the client browser. You have seen in the web.xml file a section concerning resource bundles - that's what we will use now.

Create a file ApplicationResources.properties in $APP_BASE/WEB-INF/classes, and edit it so that it looks like that:

```
index.title = My first Struts application!
index.text1 = This is my first Struts application.
```

Now, edit your index.jsp file to replace

- `<html:html>` with `<html:html locale="true">`
- `My first Struts application!` (in title) with `<bean:message key="index.title" />`
- `This is my first Struts application.` (in body) with `<bean:message key="index.text1" />`

Now you can reload you application by going to http://localhost:8080/manager/reload?path=/registeruser. As you can see, there is not much difference.

Now create a file ApplicationResources_fr.properties $APP_BASE/WEB-INF/classes, and edit it so that it looks like that:

```
index.title = Ma premiere application Struts!
index.text1 = Ceci est ma premiere application Struts.
```

Reload your application. In most browsers there is a way to configure the language of it, so do it - put French (fr) in first and reload index.jsp.

So we've seen a new tag: <bean:message>. This tag is used to get messages form resource bundles (like ApplicationResources.properties) and to display them. If you have a package for your application, you can put your resource bundle in the directory of your package (if you have a package com.ihurbain.registeruser it will be in $APP_BASE/WEB-INF/classes/com/ihurbain/registeruser for example), provided that you modify your web.xml file to fit it (in this example, you should replace ApplicationResources.properties with com.ihurbain.registeruse
After this little deviation, let's go back to our application.

3.4 Enter the application and create the data source

Let's suppose you want to make an authentification at login. To do this, you need a little form with a login and a password. We will do a very basic identification, just for example purpose, with a login ihurbain and a password foobar. We also want the registration utility to be available from now on.

IMPORTANT NOTICE

Doing an authentification the way it's done here is generally not a good idea! It is just for example purpose...

So we'll need four things:

- a JSP form
- something to store the login information
- something to check the authentification
- something to create the data source

Struts uses a configuration file called struts-config.xml to define the behaviour of an application. In this file, we define how actions demanded by the client must be treated.

3.4.1 The JSP form

Let's first create the login form. Edit your index.jsp to get a file like this:

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>

<html:html locale="true">
<head>
    <title><bean:message key="index.title" /></title>
    <html:base/>
</head>

<body>
<html:form action="/login">
    <bean:message key="prompt.login" />
    <html:text property="login" />
    <br />
    <bean:message key="prompt.password" />
    <html:password property="password" />
    <br />
    <html:submit>
        <bean:message key="index.login" />
    </html:submit>
</html:form>
</body>
</html:html>
```

and your ApplicationResources.properties to get

```
index.title = Welcome to RegisterUser
index.login = Login

prompt.login=Login:
prompt.password=Password:
```

It is a good idea to always externalize your strings; it is not much more work to do when you create a JSP and it can save you a precious amount of time if you want to modify them.

There are several new HTML tags here:

- `<html:form action=>` renders a `<form>` tag
- `<html:text property=>` renders a `<input type="text">` tag
- `<html:password property=>` renders a `<input type="password">` tag
- `<html:submit>` renders a `<input type="submit">` tag.

If you reload the application and your JSP, you have a nice error message beginning with

```
javax.servlet.ServletException: Cannot find ActionMappings or ActionFormBeans collection
```

This is normal. Look at the `<html:form>` tag: we define an action, /login. But where is this action defined? At the moment, nowhere.

It is time to look at the struts-config.xml file.

3.4.2 The struts-config.xml file

This file must be created into the $APP_BASE/WEB-INF/ directory. It is a XML file, divided into 3 main sections.

- The form-beans section. The form beans are used to store the information from the form and to validate them. A form bean is associated with each form.
- The global-forwards section. Once an action is performed, its results are forwarded to another page.
- The action-mappings section. This maps an action to a name and describes what should be done.

Let's edit this file.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE struts-config PUBLIC
          "-//Apache Software Foundation//DTD Struts Configuration 1.0//EN"
          "http://jakarta.apache.org/struts/dtds/struts-config_1_0.dtd">
```

```
<struts-config>

  <form-beans>

    <!-- Logon form bean -->
    <form-bean      name="loginForm"
                    type="LoginForm"/>
  </form-beans>


  <global-forwards>
    <forward   name="mainmenu" path="/mainmenu.jsp" />
  </global-forwards>


  <action-mappings>

    <!-- Process a user logon -->
    <action    path="/login"
               type="LoginAction"
               name="loginForm"
               scope="session"
               input="/index.jsp">
    </action>

  </action-mappings>

</struts-config>
```

The first lines are XML definitions lines - they define the document type.

The real configuration begins with `<struts-config>`.

We first define an action (in the `<action-mappings>` section). The action path corresponds to what we used for the form action, that is to say ”/login”. This is the path to be mapped to the action. Then there is a type: this is the name of the Servlet class behind the JSP. The name corresponds to a bean defined in `<form-beans>`. The scope defines in which context the login information will be available; we want it to be session-wide, so we put ”session”. The input property defines from where the action is called.

The `<form-beans>` section also contains an element: it is the bean associated to the login form. Its name corresponds to the ”name” field of the action mapping and its type is the name of the used class.

The forward maps a name to the following JSP in the application flow.

When we reload the application and the index.jsp, we have another error:

```
javax.servlet.ServletException: Exception creating bean of class LoginForm:
                                java.lang.ClassNotFoundException: LoginForm
```

This is logical, as we didn't create the LoginForm class. So let's do it.

3.4.3 Create the ActionForm and Action

Create a new class in your project by right-clicking on src, New → Other, then choosing Java Class. Put LoginForm in the name field, and org.apache.struts.action.ActionForm in Superclass, and click Finish. A new Java file has been created. The packages `org.apache.struts.action.*` and `javax.servlet.http.*` should also be imported.

Edit this file to add a String login property and a password property:

```
    private String login;
    private String password;
```

You can generate the accessors with Source $\rightarrow$ Generate getter and setter. You have to define a getter and a setter for each property of a formbean class.

Save this file, reload the application and the index.jsp and you can see a nice login form. But if you fill in the form and validate, you still have an error:

```
The requested service (Servlet action is currently unavailable) is not currently available.
```

Still nothing abnormal, as we didn't create any Action Servlet. So create the class LoginAction in your src directory; this class must have org.apache.struts.action.Action for superclass. Edit it to have something like this:

```
import java.io.IOException;
import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.struts.action.*;

public class LoginAction extends Action {
    public ActionForward perform(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException {

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

This class must contain the `perform` method as defined above; at the moment, perform only returns a forward to the next JSP. Save this file, reload the application, fill in the form and validate: the error is now

```
The requested resource (/mainmenu.jsp) is not available.
```

So let's create a mainmenu.jsp file:

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>

<html:html locale="true">
<head>
    <title><bean:message key="mainmenu.title" /></title>
    <html:base/>
</head>

<body>
<bean:message key="mainmenu.presentation" />
</body>
</html:html>
```

and create the corresponding messages in ApplicationResources.properties:

```
mainmenu.title=Main Menu
mainmenu.presentation=This is the main menu!
```

Now reload the application and you should not see any error. Great, isn't it? Well, actually, not exactly. We do have a form, but it doesn't perform much. You can enter with any login/pass.

3.4.4 Refining the login process

To correct it, we should add a `validate` method to the ActionForm. The `validate` method will check that the login and pass are OK before it allows the user to enter the application. Edit the LoginForm file and add this method:

```
public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {
    if (login.equals("ihurbain") && password.equals("foobar"))
        return null;
    else {
        ActionErrors errors = new ActionErrors();
        errors.add("login", new ActionError("error.login"));
        return errors;
    }
}
```

When the login is validated, this method returns null, so that no error is returned. When the login is not validated, an error is returned. This error has an "error.login" key, that we must define in the resources bundle:

```
error.login=Login failed. Please check you login and password.
```

When an error is detected, the forward is done to the input read in the action tag of struts-config.xml - here, back to login.jsp.

To display the errors, you can add a `<html:errors />` tag in the body of index.jsp. This displays the error, but lacks formatting. You can add something like

```
errors.header=<h3><font color="red">Validation Error</font></h3>
                 You must correct the following error(s) before proceeding:<UL>
errors.footer=</ul><hr>
```

in your resource bundle; it will be much nicer.

This is still not perfect: if someone directly types the mainmenu.jsp, he can enter anyway. We can correct it by adding

```
<logic:notPresent name="loginForm">
    <logic:redirect href="/registeruser/index.jsp" />
</logic:notPresent>
```

in mainmenu.jsp. This little piece of code checks the existence of the loginForm bean (which is not generated if someone enters the application directly with mainmenu.jsp) and redirects the request to index.jsp if it does not exist.

So now we have a login system. We also wanted to create a data source.

3.4.5 Creating a data source

We want the application to have a global users pool. To do that, we can simply create a vector of users with an application scope. You can edit your LoginAction.java to add

```
if(request.getSession().getServletContext().getAttribute("users") == null){
    Vector users = new Vector();
    request.getSession().getServletContext().setAttribute("users", users);
}
```

in the perform method. This way, if the users vector does not exist, it is created and set as a parameter of the application. To ensure there is no problem, you can add a

```
<logic:notPresent name="users">
    <logic:redirect href="/registeruser/index.jsp" />
</logic:notPresent>
```

in mainmenu.jsp.

3.5 The main menu

This will be a very short section. The menu allows the user to choose between adding a new user and displaying a user's personal data. So we only need two links on mainmenu.jsp. Your mainmenu.jsp should now look like this:

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>

<html:html locale="true">
<logic:notPresent name="loginForm">
    <logic:redirect href="/registeruser/index.jsp" />
</logic:notPresent>
<logic:notPresent name="users">
    <logic:redirect href="/registeruser/index.jsp" />
</logic:notPresent>
<head>
    <title><bean:message key="mainmenu.title" /></title>
    <html:base/>
</head>

<body>
<html:link href="/registeruser/adduser.jsp"><bean:message key="mainmenu.adduser" /></html:link>
<br />
<html:link href="/registeruser/viewuser.jsp"><bean:message key="mainmenu.viewuser" /></html:link>
</body>
</html:html>
```

You should also add in your ApplicationResources.properties

```
mainmenu.adduser=Add user
mainmenu.viewuser=View  user's personal data
```

As you modified the resources bundle, do not forget to reload the application.

3.6 Adding a user

We want to gather some information about a new user, so we define a new form, called adduser.jsp. You could do something like this:

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>

<html:html locale="true">
<logic:notPresent name="loginForm">
    <logic:redirect href="/registeruser/index.jsp" />
</logic:notPresent>
<logic:notPresent name="users">
    <logic:redirect href="/registeruser/index.jsp" />
</logic:notPresent>
<head>
    <title><bean:message key="adduser.title" /></title>
    <html:base/>
</head>

<body>
<html:errors />
<bean:message key="adduser.presentation" />
<html:form action="/adduser">
<table>
<tr>
    <td><bean:message key="adduser.mrms" /></td>
    <td>
        <html:radio property="title" value="Mr">Mr</html:radio>
        <html:radio property="title" value="Ms">Ms</html:radio>
    </td>
</tr>
<tr>
<tr>
    <td><bean:message key="adduser.firstname" /></td>
    <td><html:text property="firstName" /></td>
</tr>
<tr>
    <td><bean:message key="adduser.lastname" /></td>
    <td><html:text property="lastName" /></td>
</tr>
<tr>
    <td><bean:message key="adduser.email" /></td>
    <td><html:text property="email" /></td>
</tr>
<tr>
    <td><bean:message key="adduser.birthdate" /></td>
    <td><html:text property="birth" /></td>
</tr>
</table>
<html:submit />
</html:form>
<html:link href="/registeruser/mainmenu.jsp"><bean:message key="mainmenu.back" /></html:link>
</body>
</html:html>
```

Then, you have to configure Struts, by adding

```
<form-bean  name="userForm"
            type="UserForm" />

<forward    name="registration"   path="/registration.jsp" />

<action     path="/adduser"
            type="UserAction"
            name="userForm"
            scope="request"
            input="/adduser.jsp" />
```

in their corresponding sections.
The form bean, called UserForm.java, would be something like this:

```
import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;

public class UserForm extends ActionForm {
    private String title;
    private String firstName;
    private String lastName;
    private String email;
    private String birth;

    public String getBirth() {
        return birth;
    }

    public String getEmail() {
        return email;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public String getTitle() {
        return title;
    }

    public void setBirth(String birth) {
        this.birth = birth;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {
        ActionErrors errors = new ActionErrors();
        if(title == null || title.equals("")) {
            errors.add("title", new ActionError("errors.title"));
        }
        if(firstName == null || firstName.equals("")) {
            errors.add("firstname", new ActionError("errors.firstname"));
        }
        if(lastName == null || lastName.equals("")) {
            errors.add("lastname", new ActionError("errors.lastname"));
        }
        return errors;
    }
}
```

As you can see, the validation is made on required fields. Of course, we could do something a bit more sophisticated, like checking the validity of the format of the email address or the validity of the birthdate.

The Action gets the UserForm and adds it to the users vector:

```
import java.io.IOException;
import java.util.Vector;
import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.struts.action.*;

public class UserAction extends Action {
    public ActionForward perform(
    ActionMapping mapping,
    ActionForm form,
    HttpServletRequest request,
    HttpServletResponse response)
    throws IOException, ServletException {
        Vector users =
            (Vector) (request
            .getSession()
            .getServletContext()
            .getAttribute("users"));
        users.add(request.getAttribute("userForm"));
        request.getSession().getServletContext().setAttribute("users", users);
        return (mapping.findForward("registration"));
    }
}
```

And a last page to tell the user that all is OK is a good idea too. This page is called registration.jsp:

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>

<html:html locale="true">
<head>
    <title><bean:message key="adduser.title" /></title>
    <html:base/>
</head>

<body>
<html:errors/>
<bean:message key="registration.ok" />
```

```
<br />
<html:link href="/registeruser/adduser.jsp"><bean:message key="mainmenu.adduser" /></html:link>
<br />
<html:link href="/registeruser/mainmenu.jsp"><bean:message key="mainmenu.back" /></html:link>
</body>
</html:html>
```

Do not forget to edit your ApplicationResources.properties file to match the messages:

```
mainmenu.back = Back to main menu

errors.title=Title is a required field. Please fill it in.
errors.firstName=First name is a required field. Please fill it in.
errors.lastName=Last name is a required field. Please fill it in.

adduser.title = Add a user
adduser.presentation=Please complete the form. All the fields marked with * are required.
adduser.mrms = * Title
adduser.mr = Mr.
adduser.ms = Ms.
adduser.firstname=* First name
adduser.lastname=* Last name
adduser.birthdate = Birth date (dd/mm/yyyy)
adduser.email= Email

registration.ok = The user has been registred.
```

With this, you can register as many users you want (well, to be honest, as many users your computer supports;-))

3.7 Get information about a user

To get information about a user, you first want to know which user you want to know about. So we need another form! This one is very simple - it uses a `<html:select>` and a `<html:option>` to render a `<select>` component from a vector.

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>

<html:html locale="true">
<logic:notPresent name="loginForm">
    <logic:redirect href="/registeruser/index.jsp" />
</logic:notPresent>
<logic:notPresent name="users">
    <logic:redirect href="/registeruser/index.jsp" />
</logic:notPresent>
<head>
    <title><bean:message key="adduser.title" /></title>
    <html:base/>
</head>

<body>
```

```
<html:form action="/viewuser">
<html:select property="lastName">
    <html:options collection="users" property="lastName" labelProperty="lastName" />
</html:select>
<html:submit />
</html:form>
</body>
</html:html>
```

Add parameters to struts-config.xml:

```
    <form-bean name="userViewForm"
     type="UserViewForm" />

    <forward  name="viewinfo"    path="/viewinfo.jsp" />

    <action   path="/viewuser"
           type="UserViewAction"
           name="userViewForm"
           scope="request"
       input="/adduser.jsp" />
```

You can easily see that the form-bean information will be the same as the UserForm, except that the validation is not correct anymore. So you can inherit the new bean (UserViewForm.java) from UserForm and override the validate method so that it returns null.

```
import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionMapping;

public class UserViewForm extends UserForm {
    public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {
        return null;
    }
}
```

The associated Action gets the corresponding user in the users vector and sets it as a request attribute.

```
import java.io.IOException;
import java.util.Vector;

import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.struts.action.*;

public class UserViewAction extends Action {
    public ActionForward perform(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException {

        UserViewForm user = (UserViewForm)(request.getAttribute("userViewForm"));
        Vector users = (Vector)(request.getSession().getServletContext().getAttribute("users"));
        for(int i=0; i<users.size(); i++) {
            if(((UserForm)(users.get(i))).getLastName().equals(user.getLastName())){
                request.setAttribute("user", users.get(i));
            }
        }
        return mapping.findForward("viewinfo");
    }
}
```

The last page I will describe here is the page that displays the information about the user (viewinfo.jsp):

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>

<html:html locale="true">
<logic:notPresent name="loginForm">
    <logic:redirect href="/registeruser/index.jsp" />
</logic:notPresent>
<logic:notPresent name="users">
    <logic:redirect href="/registeruser/index.jsp" />
</logic:notPresent>
<head>
<title><bean:message key="viewuser.title" /></title>
<html:base/>
</head>

<body>
<html:errors />


<bean:write name="user" property="title" />
<bean:write name="user" property="firstName" />
<bean:write name="user" property="lastName" />
<table>
<tr>
    <td><bean:message key="adduser.email" /></td>
    <td><bean:write name="user" property="email" /></td>
</tr>
<tr>
    <td><bean:message key="adduser.birthdate" /></td>
    <td><bean:write name="user" property="birth" /></td>
</tr>
</table>
<html:link href="/registeruser/viewuser.jsp"><bean:message key="mainmenu.viewuser" /></html:link>
<br />
<html:link href="/registeruser/mainmenu.jsp"><bean:message key="mainmenu.back" /></html:link>
</body>
</html:html>
```

As you can see, the information is displayed thanks to `<bean:write>` tags; these tags print the property "property" of the bean (in any scope) with the name "name".

Do not forget to edit the ApplicationResources.properties file:

```
viewuser.title = User information
```

Reload the application... you now have finished this tutorial!

Chapter 4

Conclusion

I hope that this tutorial will have given you a good overview of Struts. You can get more information (particularly about the tags library) on Struts website (http://jakarta.apache.org/struts/).

I've tried to make this tutorial as clear and bug-free as I could. Also keep in mind that Struts, Eclipse and the J2EE specification keeps on evolving every day. However, if you have any question or if you find a bug, please do not hesitate to email me at isabelle.hurbain@free.fr.

Appendix A

GNU Free Documentation License

0. Preamble

The purpose of this License is to make a manual, textbook, or other written document ”free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of ”copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. Applicability and definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The ”Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as ”you”.

A ”Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A ”Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The ”Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The ”Cover Texts” are certain short passages of text that are listed, as Front- Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A ”Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not ”Transparent” is called ”Opaque”.

2. Verbatim copying

3. Copying in quantity

4. Modifications

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back- Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. Combining documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. Collections of documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.