

*The Legend Java NATARAJ Sir*  
*Now with DURGASOFT...*

# STRUTS

## Course Work Book

*By*  
**Mr. NATRAJ**



**AN ISO 9001:2008 CERTIFIED**



**Software Solutions®**

# **DURGASOFT**

# **STRUITS**

## INTRODUCTION STRUTS

The Struts framework is written entirely in Java using standard J2EE APIs. In addition, it uses several well-known J2EE design patterns, such as Model-view-controller and FrontController.

### Model-view-controller (MVC)

Model-view-controller (MVC) is a design pattern that defines a clear separation between the following three application tiers:

- The *model* is the set of data and business rules for the application. This is commonly called the application's business logic.
- The *view* is the application's user interface.
- The *controller* defines the way that an application interacts with user input and the model. This is called the application logic.

By promoting a clear separation between tiers, MVC allows loose coupling between components that comprise each tier. This allows more flexibility and code reuse. For example, if you develop several user interfaces for one application, you need to develop the view component because of the loosely coupled application tiers.

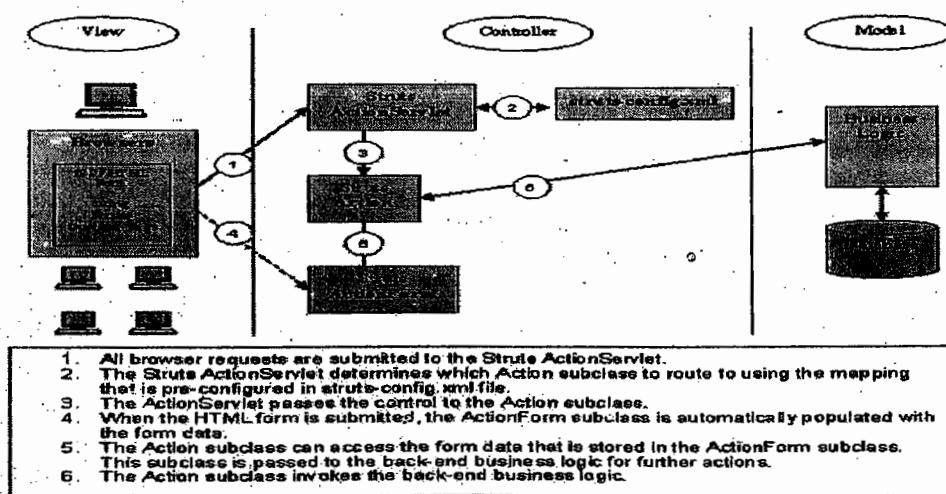
The Struts framework is the View and Controller components of MVC. The following shows how Struts maps to the MVC framework. Struts has three major components:

- Action beans
- ActionServlet
- ActionForm beans and custom tags.

### Action beans and ActionServlet

Struts provides a single ActionServlet (`org.apache.struts.action.ActionServlet`) to handle all browser requests. This type of framework is called the FrontController pattern. Each browser request is handled by the Struts Action subclass (subclass of `org.apache.struts.action.Action`). Each browser request is mapped to an Action subclass in the `struts-config.xml` file. The ActionServlet loads the mapping during initialization. To configure the Web project to pass all browser requests to the ActionServlet, map all URIs that end with `.do` (for example, `*.do`) to the ActionServlet in the Web deployment descriptor. You can then provide the actual Action subclass mappings in the Struts configuration file for individual request URI, such as `/submit.do`.

**Figure 1. MVC and the Struts framework**



## STRUTS

DURGASOFT

### ActionForm beans

Browser requests can come with parameters. When a user submits a HTML form, the Struts framework encloses the parameters in an org.apache.struts.action.ActionForm bean. You can also use the ActionForm bean to pre-populate a form with default values that are obtained from database or other backend systems. If the user enters incorrect values in the form, the ActionForm may perform validations. You can re-display the form with the previous input.

### Custom tags

Struts provides a large set of JSP custom tags that support the ActionForm beans. The custom tags support:

- Pre-population of a HTML form with values taken from an ActionForm subclass.
- Internationalization, such as providing text that is determined by the user's locale.
- Logic, such as showing a different title for a page based on how it is used.

## STRUTS 1.X

The framework provides several components that make up the **Control** layer of a MVC style application. These include a controller component (servlet), developer-defined request handlers, and several supporting objects.

The Struts Taglib component provides direct support for the **View** layer of a MVC application. Some of these tags access the control-layer objects. Others are generic tags found convenient when writing applications. Other taglibs, including JSTL, can also be used with the framework. Other presentation technologies, like Velocity Templates and XSLT can also be used with the framework.

The **Model** layer in a MVC application is often project-specific. The framework is designed to make it easy to access the business-end of your application, but leaves that part of the programming to other products, like JDBC, Enterprise Java Beans, Object Relational Bridge, or iBATIS, to name a few.

Let's step through how this all fits together.

When initialized, the controller parses a configuration file (struts-config.xml) and uses it to deploy other control layer objects. Together, these objects form the **Struts Configuration**. The Configuration defines (among other things) the collection of ActionMappings[org.apache.struts.action.ActionMappings] for an application.

The controller component consults the ActionMappings as it routes HTTP requests to other components in the framework. Requests may be forwarded to JavaServer Pages or Action[org.apache.struts.action.Action] subclasses provided by the application developer. Often, a request is first forwarded to an Action and then to a JSP (or other presentation page). The mappings help the controller turn HTTP requests into application actions.

An individual ActionMapping[org.apache.struts.action.ActionMapping] will usually contain a number of properties including:

- a **request path** (or "URI"),
- the **object type** (Action subclass) to act upon the request, and
- other properties as needed.

The Action object can handle the request and respond to the client (usually a Web browser) or indicate that control should be forwarded elsewhere. For example, if a login succeeds, a login action may wish to forward the request onto the mainMenu page.

Action objects have access to the application's controller component, and so have access to that member's methods. When forwarding control, an Action object can indirectly forward one or more shared objects, including JavaBeans, by placing them in one of the standard contexts shared by Java Servlets.

## STRUTS

DURGASOFT

For example, an Action object can create a shopping cart bean, add an item to the cart, place the bean in the session context, and then forward control to another mapping. That mapping may use a JavaServer Page to display the contents of the user's cart. Since each client has their own session, they will each also have their own shopping cart.

Most of the business logic in an application can be represented using JavaBeans. An Action can call the properties of a JavaBean without knowing how it actually works. This encapsulates the business logic, so that the Action can focus on error handling and where to forward control.

JavaBeans can also be used to manage input forms. A key problem in designing Web applications is retaining and validating what a user has entered between requests. You can define your own set of input bean classes, by subclassing `ActionForm[org.apache.struts.action.ActionForm]`. The `ActionForm` class makes it easy to store **and validate** the data for your application's input forms. The `ActionForm` bean is automatically saved in one of the standard, shared context collections, so that it can be used by other objects, like an Action object or another JSP.

The form bean can be used by a JSP to collect data from the user ... by an Action object to validate the user-entered data ... and then by the JSP again to re-populate the form fields. In the case of validation errors, the framework has a shared mechanism for raising and displaying error messages.

Another element of the Configuration are the `ActionFormBean [org.apache.struts.action.ActionFormBeans]`. This is a collection of descriptor objects that are used to create instances of the `ActionForm` objects at runtime. When a mapping needs an `ActionForm`, the servlet looks up the form-bean descriptor by name and uses it to create an `ActionForm` instance of the specified type.

Here is the sequence of events that occur when a request calls for an mapping that uses an `ActionForm`:

- The controller servlet either retrieves or creates the `ActionForm` bean instance.
- The controller servlet passes the bean to the Action object.
- If the request is being used to submit an input page, the Action object can examine the data. If necessary, the data can be sent back to the input form along with a list of messages to display on the page. Otherwise the data can be passed along to the business tier.
- If the request is being used to create an input page, the Action object can populate the bean with any data that the input page might need.

The Struts Taglib component provides custom tags that can automatically populate fields from a JavaBean. All most JavaServer Pages really need to know is the field names to use and where to submit the form.

Other tags can automatically output messages queued by an Action or `ActionForm` and simply need to be integrated into the page's markup. The messages are designed for localization and will render the best available message for a user's locale.

The framework and Struts Taglib were designed from the ground-up to support the internationalization features built into the Java platform. All the field labels and messages can be retrieved from a message resource. To provide messages for another language, simply add another file to the resource bundle.

Internationalism aside, other benefits to the message resources approach are consistent labeling between forms, and the ability to review all labels and messages from a central location.

For the simplest applications, an Action object may sometimes handle the business logic associated with a request. **However, in most cases, an Action object should invoke another object, usually a JavaBean, to perform the actual business logic.** This lets the Action focus on error handling and control flow, rather than business logic. To allow reuse on other platforms, business-logic JavaBeans should not refer to any Web application objects. The

## STRUTS

DURGASOFT

Action object should translate needed details from the HTTP request and pass those along to the business-logic beans as regular Java variables.

In a database application, for example:

- A business-logic bean will connect to and query the database,
- The business-logic bean returns the result to the Action,
- The Action stores the result in a form bean in the request,
- The JavaServer Page displays the result in a HTML form.

Neither the Action nor the JSP need to know (or care) from where the result comes. They just need to know how to package and display it.

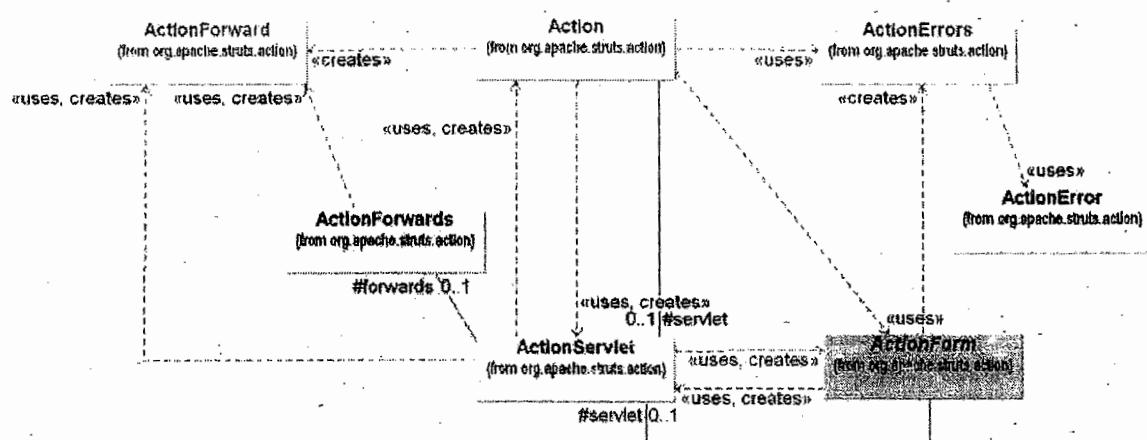
Other sections in this document cover the various framework components in greater detail. The Struts Taglib component includes several Developer Guides covering various aspects of the custom tags. A number of sample applications are bundled with the distribution that show how it all comes together.

The framework is distributed under the Apache Software Foundation license. The code is copyrighted, but is free to use in any application.

### Struts details

Displayed in Figure 6 is a stripped-down UML diagram of the org.apache.struts.action package. Figure 6 shows the minimal relationships among ActionServlet (Controller), ActionForm (Form State), and Action (Model Wrapper).

**Figure 6. UML diagram of the relationship of the Command (ActionServlet) to the Model (Action & ActionForm)**



### The ActionServlet class

Do you remember the days of function mappings? You would map some input event to a pointer to a function. If you were slick, you would place the configuration information into a file and load the file at run time. Function pointer arrays were the good old days of structured programming in C.

Life is better now that we have Java technology, XML, J2EE, and all that. The Struts Controller is a servlet that maps events (an event generally being an HTTP post) to classes. And guess what--the Controller uses a configuration file so you don't have to hard-code the values. Life changes, but stays the same.

## **STRUTS**

## **DURGASOFT**

ActionServlet is the Command part of the MVC implementation and is the core of the Framework. ActionServlet (Command) creates and uses Action, an ActionForm, and ActionForward. As mentioned earlier, the struts-config.xml file configures the Command. During the creation of the Web project, Action and ActionForm are extended to solve the specific problem space. The file struts-config.xml instructs ActionServlet on how to use the extended classes. There are several advantages to this approach:

- The entire logical flow of the application is in a hierarchical text file. This makes it easier to view and understand, especially with large applications.
- The page designer does not have to wade through Java code to understand the flow of the application.
- The Java developer does not need to recompile code when making flow changes.

Command functionality can be added by extending ActionServlet.

### **The ActionForm class**

ActionForm maintains the session state for the Web application. ActionForm is an abstract class that is sub-classed for each input form model. When I say *input form model*, I am saying ActionForm represents a general concept of data that is set or updated by a HTML form. For instance, you may have a UserActionForm that is set by an HTML Form. The Struts framework will:

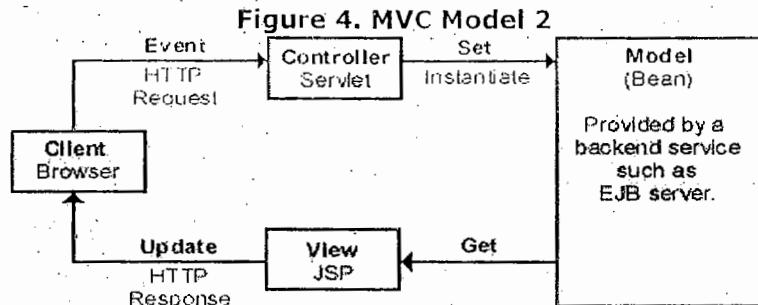
- Check to see if a UserActionForm exists; if not, it will create an instance of the class.
- Struts will set the state of the UserActionForm using corresponding fields from the HttpServletRequest. No more dreadful request.getParameter() calls. For instance, the Struts framework will take fname from request stream and call UserActionForm.setFname().
- The Struts framework updates the state of the UserActionForm before passing it to the business wrapper UserAction.
- Before passing it to the Action class, Struts will also conduct form state validation by calling the validation() method on UserActionForm. **Note:** This is not always wise to do. There might be ways of using UserActionForm in other pages or business objects, where the validation might be different. Validation of the state might be better in the UserAction class.
- The UserActionForm can be maintained at a session level.

#### **Notes:**

- The struts-config.xml file controls which HTML form request maps to which ActionForm.
- Multiple requests can be mapped UserActionForm.
- UserActionForm can be mapped over multiple pages for things such as wizards.

### **The Action class**

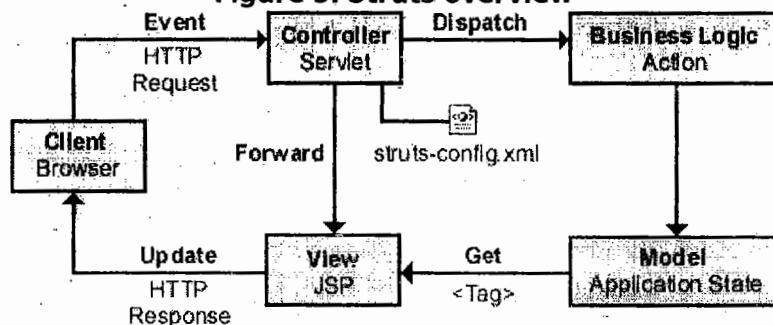
The Action class is a wrapper around the business logic. The purpose of Action class is to translate the HttpServletRequest to the business logic. To use Action, subclass and overwrite the process() method. The ActionServlet (Command) passes the parameterized classes to ActionForm using the perform() method. Again, no more dreadful request.getParameter() calls. By the time the event gets here, the input form data (or HTML form data) has already been translated out of the request stream and into an ActionForm class.



Struts, an MVC 2 implementation

Struts is a set of cooperating classes, servlets, and JSP tags that make up a reusable MVC 2 design. This definition implies that Struts is a framework, rather than a library, but Struts also contains an extensive tag library and utility classes that work independently of the framework. Figure 5 displays an overview of Struts.

**Figure 5. Struts overview**



### Struts overview

- **Client browser:** An HTTP request from the client browser creates an event. The Web container will respond with an HTTP response.
- **Controller:** The Controller receives the request from the browser, and makes the decision where to send the request. With Struts, the Controller is a command design pattern implemented as a servlet. The struts-config.xml file configures the Controller.
- **Business logic:** The business logic updates the state of the model and helps control the flow of the application. With Struts this is done with an Action class as a thin wrapper to the actual business logic.
- **Model state:** The model represents the state of the application. The business objects update the application state. ActionForm bean represents the Model state at a session or request level, and not at a persistent level. The JSP file reads information from the ActionForm bean using JSP tags.
- **View:** The view is simply a JSP file. There is no flow logic, no business logic, and no model information -- just tags. Tags are one of the things that make Struts unique compared to other frameworks like Velocity.

Note: "Think thin" when extending the Action class. The Action class should control the flow and not the logic of the application. By placing the business logic in a separate package or EJB, we allow flexibility and reuse.

Another way of thinking about Action class is as the Adapter design pattern. The purpose of the Action is to "Convert the interface of a class into another interface the clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interface" (from *Design*

## STRUTS

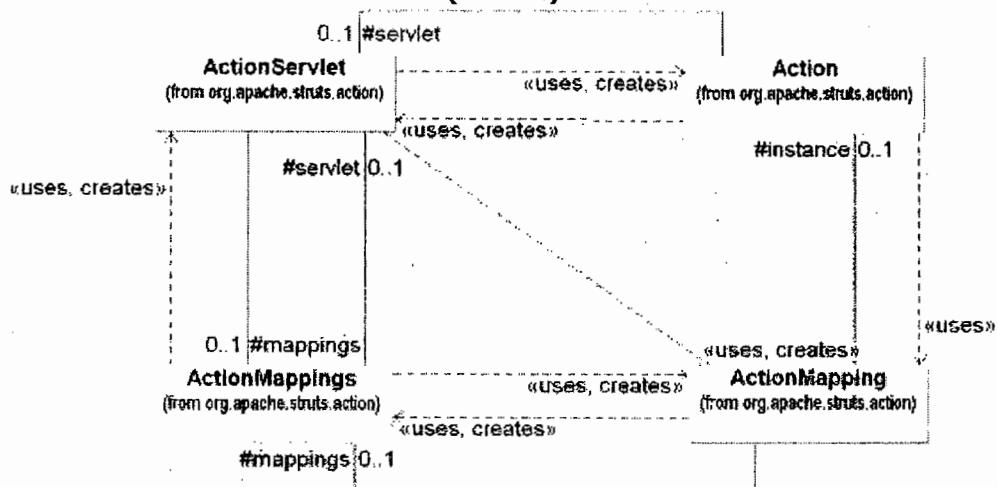
## DURGASOFT

Patterns - *Elements of Reusable OO Software* by Gof). The client in this instance is the ActionServlet that knows nothing about our specific business class interface. Therefore, Struts provides a business interface it does understand, Action. By extending the Action, we make our business interface compatible with Struts business interface. (An interesting observation is that Action is a class and not an interface. Action started as an interface and changed into a class over time. Nothing's perfect.)

### The Error classes

The UML diagram (Figure 6) also included ActionError and ActionErrors. ActionError encapsulates an individual error message. ActionErrors is a container of ActionError classes that the View can access using tags. ActionErrors is Struts way of keeping up with a list of errors.

**Figure 7. UML diagram of the relationship of the Command (ActionServlet) to the Model (Action)**



### The ActionMapping class

An incoming event is normally in the form of an HTTP request, which the servlet Container turns into an HttpServletRequest. The Controller looks at the incoming event and dispatches the request to an Action class. The struts-config.xml determines what Action class the Controller calls. The struts-config.xml configuration information is translated into a set of ActionMapping, which are put into container of ActionMappings. (If you have not noticed it, classes that end with s are containers)

The ActionMapping contains the knowledge of how a specific event maps to specific Actions. The ActionServlet (Command) passes the ActionMapping to the Action class via the perform() method. This allows Action to access the information to control flow.

### ActionMappings

ActionMappings is a collection of ActionMapping objects.

```

1 App1(StrutsBasicApplication)
2 -----
3 -----register.jsp-----
4 <%@ taglib uri="demo" prefix="html" %>
5
6 <html:form action="register">
7   UserName :<html:text property="username"/> <br>
8   Password :<html:password property="password"/><br>
9   <html:submit value="Register"/>
10  </html:form>
11 -----
12 <web-app>
13   <servlet>
14     <servlet-name>action</servlet-name>
15     <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
16     <init-param>
17       <param-name>config</param-name>
18       <param-value>/WEB-INF/struts-config.xml</param-value>
19     </init-param>
20     <load-on-startup>1</load-on-startup>
21   </servlet>
22
23   <servlet-mapping>
24     <servlet-name>action</servlet-name>
25     <url-pattern>*.do</url-pattern>
26   </servlet-mapping>
27
28   <taglib>
29     <taglib-uri>demo</taglib-uri>
30     <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
31   </taglib>
32 </web-app>
33
34 -----
35 <!DOCTYPE struts-config PUBLIC
36   "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
37   "http://struts.apache.org/dtds/struts-config_1_3.dtd">
38
39 <struts-config>
40   <form-beans>
41     <form-bean name="rf" type="app.RegisterForm"/>
42   </form-beans>
43
44 <action-mappings>
45   <action path="/register"
46     type="app.RegisterAction"
47     name="rf">
48     <forward name="success" path="/success.jsp"/>
49     <forward name="failure" path="/failure.jsp"/>
50   </action>
51
52 </action-mappings>
53 </struts-config>
54 -----
55 package app;
56
57 import org.apache.struts.action.*;
58 import javax.servlet.http.*;
59
60 public class RegisterForm extends ActionForm
61 {

```

```

62  private String username;
63  private String password;
64  public void setUsername(String uname)
65  {
66      username=uname;
67  }
68  public void setPassword(String pwd)
69  {
70      password=pwd;
71  }
72  public String getUsername()
73  {
74      return username;
75  }
76  public String getPassword()
77  {
78      return password;
79  }
80  }
81 -----RegisterAction.java-----
82 package app;
83 import org.apache.struts.action.*;
84 import javax.servlet.http.*;
85
86 public class RegisterAction extends Action
87 {
88     public ActionForward execute(ActionMapping mapping,ActionForm form,
89             HttpServletRequest req,HttpServletResponse res) throws Exception
90     {
91         System.out.println("RegisterAction:execute(-,-,-,-)");
92         RegisterForm fm=(RegisterForm)form;
93
94         String uname=fm.getUsername();
95         String pwd=fm.getPassword();
96
97         if (uname.equals("durga")&&pwd.equals("soft"))
98             return mapping.findForward("success");
99         else
100            return mapping.findForward("failure");
101    }// execute
102 }/class
103 -----success.jsp-----
104 <HTML>
105   <HEAD>
106     <TITLE> About Loigin </TITLE>
107   </HEAD>
108
109  <BODY>
110    <center>Login Success</center>
111  </BODY>
112 </HTML>
113 -----failure.jsp-----
114 <HTML>
115   <HEAD>
116     <TITLE> About Loigin </TITLE>
117   </HEAD>
118
119  <BODY>
120    <center>Login Failure</center>
121    <p><a href="register.jsp">Try again</a></p>
122 </BODY>

```

## STRUTS

## DURGASOFT

```
123 </HTML>
124 =====
125 App>>>>>>>>> Working with Diff types of comps >>>>>>>>>>>>>
126 -----input.jsp-----
127 <%@taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
128 <%@taglib uri="http://struts.apache.org/tags-logic" prefix="logic" %>
129 <%@taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
130
131 <body bgcolor="#777788">
132 <html:form action="ipath" >
133 <table border="3" width="1" bgcolor="#993300" align="left" cellspacing="1" cellpadding="1">
134 <tr>
135   <td>Name:</td>
136   <td><html:text property="name" /></td>
137 </tr>
138
139 <tr>
140   <td>Age:</td>
141   <td><html:text property="age"/></td>
142 </tr>
143 <tr>
144   <td>Address:</td>
145   <td><html:textarea property="addrs" cols="20" rows="5" >
146     Enter Address
147   </html:textarea>
148 </td>
149 </tr>
150
151 <tr>
152   <td>Gender:</td>
153   <td><html:radio property="gender" value="M"/> Male
154   <html:radio property="gender" value="F"/> Female
155 </td>
156 </tr>
157 <tr>
158   <td>Marital Status:</td>
159   <td><html:checkbox property="ms" value="Married" /> Married </td>
160 </tr>
161 <tr>
162   <td>Qualifications:</td>
163   <td><html:select property="qlfy" >
164     <html:option value="engg">Engineering </html:option>
165     <html:option value="arts">Arts </html:option>
166     <html:option value="cmrs">Commerce </html:option>
167     <html:option value="Medico">Medical </html:option>
168   </html:select>
169 </td>
170 </tr>
171 <tr>
172   <td>Courses:</td>
173   <td><html:select multiple="yes" property="crs">
174     <html:option value="JAVA">JAVA Package </html:option>
175     <html:option value=".net">.Net package</html:option>
176     <html:option value="oracle"> Oracle package</html:option>
177     <html:option value="php">PHP </html:option>
178   </html:select>
179 </td>
180 </tr>
181 <tr>
182   <td> Hobbies :</td>
183   <td><html:checkbox property="hb" value="stamps" /> Stamps Collection<br>
```

## STRUTS

DURGASOFT

```
184      <html:checkbox property="hb" value="reading" /> Reading Books<br>
185      <html:checkbox property="hb" value="roaming" /> Travelling
186    </td>
187  </tr>
188  <tr>
189    <td><html:submit value="send"/></td>
190    <td> <html:reset value="Reset"/></td>
191  </tr>
192
193  </table>
194 </html:form>
195 </body>
196 -----web.xml-----
197 <web-app>
198 <servlet>
199   <servlet-name>action</servlet-name>
200   <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
201   <init-param>
202     <param-name>config</param-name>
203     <param-value>/WEB-INF/struts-config.xml</param-value>
204   </init-param>
205   <load-on-startup>2</load-on-startup>
206 </servlet>
207 <servlet-mapping>
208   <servlet-name>action</servlet-name>
209   <url-pattern>*.do</url-pattern>
210 </servlet-mapping>
211 <welcome-file-list>
212   <welcome-file>input.jsp</welcome-file>
213 </welcome-file-list>
214 </web-app>
215 -----struts-config.xml-----
216 <!DOCTYPE struts-config PUBLIC
217   "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
218   "http://jakarta.apache.org/struts/dtds/struts-config_1_3.dtd">
219 <struts-config>
220 <form-beans>
221   <form-bean name="InputForm" type="p1.InputForm"/>
222 </form-beans>
223
224 <action-mappings>
225   <action name="InputForm" path="/ipath" scope="request" type="p1.InputAction">
226     <forward name="result" path="/result.jsp" />
227   </action>
228 </action-mappings>
229 </struts-config>
230 -----InputForm.java-----
231
232 package p1;
233
234 import javax.servlet.http.HttpServletRequest;
235 import org.apache.struts.action.ActionMapping;
236
237 public class InputForm extends org.apache.struts.action.ActionForm {
238
239   private String name;
240   private int age;
241   private String addrs;
242   private String gender;
243   private String ms;
244   private String qif;
```

## STRUTS

## DURGASOFT

S

```
245  private String[] crs;          30
246  private String[] hb;          30
247
248 /* public void reset(ActionMapping mapping,HttpServletRequest request) 30
249 {
250     ms="Single";
251     crs=new String[1];
252     crs[0]="No Courses Selected";
253     hb=new String[1];
254     hb[0]="no Hobbies selected";
255 } */          31
256
257 public String getAddrs() {          31
258     return addrs;
259 }
260
261 public void setAddrs(String addrs) { 32
262     this.addrs = addrs;
263 }
264
265 public int getAge() {          32
266     return age;
267 }
268
269 public void setAge(int age) { 32
270     this.age = age;
271 }
272
273 public String[] getCrs() {          33
274     return crs;
275 }
276
277 public void setCrs(String[] crs) { 33
278     this.crs = crs;
279 }
280
281 public String getGender() {          34
282     return gender;
283 }
284
285 public void setGender(String gender) { 34
286     this.gender = gender;
287 }
288
289 public String[] getHb() {          35
290     return hb;
291 }
292
293 public void setHb(String[] hb) { 35
294     this.hb = hb;
295 }
296
297 public String getMs() {          35
298     return ms;
299 }
300
301 public void setMs(String ms) { 36
302     this.ms = ms;
303 }
304
305 public String getName() {          36
306 }
```

## STRUTS

DURGASOFT

```
306     return name;
307 }
308
309 public void setName(String name) {
310     this.name = name;
311 }
312
313 public String getQlfy() {
314     return qlfy;
315 }
316
317 public void setQlfy(String qlfy) {
318     this.qlfy = qlfy;
319 }
320
321 }
322 -----InputAction.java-----
323 package p1;
324 import javax.servlet.http.HttpServletRequest;
325 import javax.servlet.http.HttpServletResponse;
326 import org.apache.struts.action.ActionForm;
327 import org.apache.struts.action.ActionForward;
328 import org.apache.struts.action.ActionMapping;
329
330 public class InputAction extends org.apache.struts.action.Action {
331
332     public ActionForward execute(ActionMapping mapping, ActionForm form,
333         HttpServletRequest request, HttpServletResponse response)
334         throws Exception {
335
336         //read form data from inputForm
337         InputForm fm=(InputForm)form;
338         String name=fm.getName();
339         int age=fm.getAge();
340         String gen=fm.getGender();
341         String addrs=fm.getAddrs();
342         String ms=fm.getMs();
343         String qlfy=fm.getQlfy();
344         String crs[] =fm.getCrss();
345         String hb[] =fm.getHbs();
346
347         //check age status ..
348         if(age>=18)
349             request.setAttribute("msg","you are elegible for Vote");
350         else
351             request.setAttribute("msg", "you are not elegible for Vote");
352         //keep form data in request attributes...
353         request.setAttribute("name",name);
354         request.setAttribute("age", age);
355         request.setAttribute("gender", gen);
356         request.setAttribute("addrs", addrs);
357         request.setAttribute("ms", ms);
358         request.setAttribute("qlfy", qlfy);
359         request.setAttribute("crs", crs);
360         request.setAttribute("hb", hb);
361
362
363     return mapping.findForward("result");
364
365 } //execute
366 } //class
```

```

367 -----result.jsp-----
368 <%@taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
369 <%@taglib uri="http://struts.apache.org/tags-logic" prefix="logic" %>
370 <%@taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
371
372 <body bgcolor="#779988" text="red">
373 <center>
374
375 <logic:notEmpty name="name">
376   Name:<bean:write name="name" /><br>
377 </logic:notEmpty>
378
379 <logic:notEmpty name="age">
380   Age:<%=request.getAttribute("age") %><br>
381 </logic:notEmpty>
382
383 <logic:notEmpty name="gender">
384   Gender:<%=request.getAttribute("gender") %><br>
385 </logic:notEmpty>
386
387 <logic:notEmpty name="addrs">
388   Address:<%=request.getAttribute("addrs") %><br>
389 </logic:notEmpty >
390
391 <logic:notEmpty name="ms">
392   Marital status:<%=request.getAttribute("ms") %><br>
393 </logic:notEmpty >
394
395 <logic:notEmpty name="qlfy">
396   Qualification:<%=request.getAttribute("qlfy") %><br>
397 </logic:notEmpty>
398
399 <logic:notEmpty name="crs">
400   Course:
401     <logic:iterate id="item" collection="<%=request.getAttribute("crs")%>">
402       <bean:write name="item"/>.....
403     </logic:iterate>
404   </logic:notEmpty ><br>
405
406 <logic:notEmpty name="hb">
407   Hobbies:
408     <logic:iterate id="item" collection="<%=request.getAttribute("hb")%>">
409       <bean:write name="item"/>.....
410     </logic:iterate>
411   </logic:notEmpty ><br>
412
413 <logic:notEmpty name="msg">
414   Result is :<%=request.getAttribute("msg") %>
415 </logic:notEmpty >
416
417   </center>
418 </body>
419
420
421

```

```

1 >>>>>>>> Struts App with Server Side and Client Side Programmatic Form Validations
2 >>>>>>>>
3 -----register.jsp-----
4 <%@taglib uri="http://struts.apache.org/tags-html" prefix="html"%>
5 <%@taglib uri="http://struts.apache.org/tags-bean" prefix="bean"%>
6 <%@taglib uri="http://struts.apache.org/tags-logic" prefix="logic"%>
7 <html:html>
8   <head>
9     <script language="JavaScript">
10    function myValidate(frm)
11    {
12      //read form data
13      var un=frm.username.value;
14      var pwd=frm.password.value;
15
16      //write client side programmatic form validation logic
17      if(un=="") // required rule
18      {
19        alert("user name is mandatory");
20        frm.username.focus();
21        return false;
22      }//if
23      else // first char must be an alphabet
24      {
25        var val=un.charCodeAt(0); //gives ascii value of first char
26        if( !(val>=65 && val<=90) && !(val>=97 && val<=122))
27        {
28          alert("First char must be an alphabet");
29          frm.username.focus();
30          return false;
31        }//if
32      }//else
33      if(pwd=="") // required rule
34      {
35        alert("password is mandatory");
36        frm.password.focus();
37      }
38      frm.vflag.value="true"; // to indicate that Client side validations are done
39      return true;
40    }/myvalidate()
41  </script>
42 </head>
43
44 <center> <h1> <bean:message key="my.title" /></h1> </center>
45 <html:form action = "register" onsubmit="return myValidate(this)">
46   <bean:message key="my.user"/><html:text property = "username"/>
47   <html:errors property="err1"/> <br>
48
49   <bean:message key="my.pwd"/><html:password property = "password"/>
50   <html:errors property="err2"/> <br>
51
52   <html:submit>
53     <bean:message key="my.btn.cap"/>
54   </html:submit>
55
56   <html:hidden property="vflag" value="false"/>
57 </html:form>
58
59   <logic:notEmpty name="msg" >
60     <bean:message key="my.res"/> <bean:write name="msg" />

```

## STRUTS

DURGASOFT

```
61 </logic:notEmpty>
62
63 <br>
64 <bean:message key="my.footer"/>
65 </html:html>
66 -----web.xml-----
67 <web-app>
68 <servlet>
69 <servlet-name>action</servlet-name>
70 <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
71 <init-param>
72 <param-name>config</param-name>
73 <param-value>/WEB-INF/struts-config.xml</param-value>
74 </init-param>
75 <load-on-startup>2</load-on-startup>
76 </servlet>
77
78 <servlet-mapping>
79 <servlet-name>action</servlet-name>
80 <url-pattern>*.do</url-pattern>
81 </servlet-mapping>
82 </web-app>
83 -----struts-config.xml-----
84 <!DOCTYPE struts-config PUBLIC
85   "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
86   "http://struts.apache.org/dtds/struts-config_1_3.dtd">
87
88 <struts-config>
89
90 <form-beans>
91   <form-bean name="rf" type="app.RegisterForm"/>
92 </form-beans>
93
94 <action-mappings>
95   <action path="/register" type="app.RegisterAction" name="rf" input="/register.jsp">
96     <forward name="result" path="/register.jsp"/>
97   </action>
98 </action-mappings>
99   <message-resources parameter="myfile"/>
100 </struts-config>
101 -----myfile.properties-----
102 # presentation lables
103 my.title=<center><h1> Login Page </h1></center>
104 my.user= UserName
105 my.pwd= Password
106 my.btn.cap=Login
107 my.res=Result is
108 my.footer=@copy; copy rights reserved 2011-12
109
110 #form validation error msgs
111 my.un.err=username is required
112 my.un.fchar=user name must begin with an alphabet
113 my.pwd.err=password is required
114 # To apply styles on form validation error msgs
115 errors.header=<font color=red size=1>
116 errors.footer=</font>
117 -----RegisterForm.java-----
118 package app;
119 import org.apache.struts.action.*;
120 import javax.servlet.http.*;
121
```

```

122 public class RegisterForm extends ActionForm
123 {
124     private String username="raja";
125     private String password="hyd";
126
127     public RegisterForm()
128     {
129         System.out.println("RegisterForm: 0-param constructor");
130     }
131
132     public void setUsername(String username)
133     {
134         System.out.println("RegisterForm:setUsername(-)");
135         this.username = username;
136     }
137     public String getUsername()
138     {
139         System.out.println("RegisterForm:getUsername()");
140         return username;
141     }
142     public void setPassword(String password)
143     {
144         System.out.println("RegisterForm:setPassword(-)");
145         this.password = password;
146     }
147     public String getPassword()
148     {
149         System.out.println("RegisterForm:setPassword(-)");
150         return password;
151     }
152
153     public ActionErrors validate(ActionMapping mapping,HttpServletRequest req)
154     {
155
156         System.out.println("RegisterForm:validate(-,-)");
157         ActionErrors errs=new ActionErrors();
158         // read vflag value
159         String flag=req.getParameter("vflag"); // gives Client Side form validation logic execution status
160         if(flag.equals("false"))
161         {
162             System.out.println("server side form validation is going on.....");
163             // write server side programmatic form validation logic(required rule) (java code)
164             if(username==null || username.length()==0 || username.equals(""))
165             {
166                 errs.add("err1",new ActionMessage("my.un.err"));
167             }
168             else // to check weather first char is alphabet or not
169             {
170                 // get First char from username
171                 char fchar=username.charAt(0);
172                 if(!Character.isUpperCase(fchar) && !Character.isLowerCase(fchar))
173                 {
174                     errs.add("err1",new ActionMessage("my.un.fchar"));
175                 } //if
176             } //else
177
178             if(password==null || password.length()==0 || password.equals(""))
179             {
180                 errs.add("err2",new ActionMessage("my.pwd.err"));
181             }
182             System.out.println("errs obj size"+errs.size());

```

## STRUTS

```
183 }//if
184 return errs;
185 }//validate(,-)
186 }//class
187 -----RegisterAction.java-----
188 package app;
189 import org.apache.struts.action.*;
190 import javax.servlet.http.*;
191 import javax.servlet.*;
192
193
194 public class RegisterAction extends Action
195 {
196
197 public RegisterAction()
198 {
199     System.out.println("RegisterAction: 0-param constructor");
200 }
201 public ActionForward execute(ActionMapping mapping,ActionForm form,
202                         HttpServletRequest req,HttpServletResponse res)throws Exception
203 {
204     System.out.println("RegisterAction:execute(,-,-,-)");
205     RegisterForm rf = (RegisterForm)form;
206
207     String user = rf.getUsername();
208     String pass = rf.getPassword();
209
210     if(user.equals("Durga") && pass.equals("soft"))
211     {
212         req.setAttribute("msg","Valid Credentials");
213     }
214     else
215     {
216         req.setAttribute("msg","InValid Credentials");
217     }
218
219     return mapping.findForward("result");
220 }
221 }//class
222
223
224
225
226
227
228
```

## DURGASOFT

```

1 App2(StrutsApp-with form validations)
2 =====
3 -----register.jsp-----
4 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
5 <%@ taglib uri="http://struts.apache.org/tags-logic" prefix="logic" %>
6 <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
7
8   <bean:message key="my.Title"/>
9   <html:form action="register" method="GET">
10    <bean:message key="my.un" /><html:text property="username"/><br>
11    <html:errors property="username"/><br>
12
13   <bean:message key="my.pwd" /><html:text property="password"/><br>
14   <html:errors property="password"/><br>
15
16   <html:submit>
17     <bean:message key="my.btn.cap"/>
18   </html:submit>
19 </html:form>
20
21   <logic:notEmpty name="msg" scope="request">
22     The Result is :<bean:write name="msg" scope="request"/>
23 </logic:notEmpty>
24
25 -----web.xml-----
26 <web-app>
27   <servlet>
28     <servlet-name>action</servlet-name>
29     <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
30     <init-param>
31       <param-name>config</param-name>
32       <param-value>/WEB-INF/struts-config.xml</param-value>
33     </init-param>
34     <load-on-startup>2</load-on-startup>
35   </servlet>
36
37   <servlet-mapping>
38     <servlet-name>action</servlet-name>
39     <url-pattern>*.do</url-pattern>
40   </servlet-mapping>
41
42 </web-app>
43 -----struts-config.xml-----
44 <!DOCTYPE struts-config PUBLIC
45   "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
46   "http://struts.apache.org/dtds/struts-config_1_3.dtd">
47 <struts-config>
48   <form-beans>
49     <form-bean name="rf" type="app.RegisterForm"/>
50   </form-beans>
51   <action-mappings>
52     <action path="/register" type="app.RegisterAction" name="rf" input="/register.jsp">
53       <forward name="result" path="/register.jsp"/>
54     </action>
55   </action-mappings>
56   <message-resources parameter="myfile"/>
57
58   <plug-in className="org.apache.struts.validator.ValidatorPlugin">
59     <set-property property="pathnames"
60       value="/WEB-INF/validator-rules.xml,
61       /WEB-INF/validation.xml"/>

```

## STRUTS

## DURGASOFT

```
62    </plug-in>
63  </struts-config>
64  -----validation.xml-----
65 <!DOCTYPE form-validation PUBLIC
66   "-//Apache Software Foundation//DTD Commons Validator Rules Configuration 1.3.0//EN"
67   "http://jakarta.apache.org/commons/dtds/validator_1_3_0.dtd">
68
69 <form-validation>
70  <formset>
71    <form name="rf">
72      <field property="username" depends="required">
73        <arg0 key="my.un"/>
74      </field>
75      <field property="password" depends="required">
76        <arg0 key="my.pass"/>
77      </field>
78    </form>
79  </formset>
80 </form-validation>
81 -----RegisterForm.java-----
82 // RegisterForm.java
83 package app;
84 import org.apache.struts.action.*;
85 import org.apache.struts.validator.*;
86 import javax.servlet.http.*;
87
88 public class RegisterForm extends ValidatorForm
89 {
90     // FormBean properties
91     String username="xyz",password;
92
93     public RegisterForm()
94     {
95         System.out.println("0-arg constructor: RegisterForm"+this.hashCode());
96     }
97
98     public void setUsername(String username)
99     {
100         System.out.println("setUsername: RegisterForm");
101         this.username=username;
102     }
103
104     public void setPassword(String password)
105     {
106         System.out.println("setPassword: RegisterForm");
107         this.password=password;
108     }
109
110     public String getUsername()
111     {
112         System.out.println("getUsername: RegisterForm");
113         return username;
114     }
115
116     public String getPassword()
117     {
118         System.out.println("getPassword: RegisterForm");
119         return password;
120     }
121 } //class
122 -----RegisterAction.java-----
```

## STRUTS

DURGASOFT

```
123 // RegisterAction.java
124
125 package app;
126
127 import javax.servlet.http.*;
128 import org.apache.struts.action.*;
129
130 public class RegisterAction extends Action
131 {
132
133     public RegisterAction()
134     {
135         System.out.println("0-arg constructor:RegisterAction");
136     }
137
138
139     public ActionForward execute(ActionMapping mapping,
140                               ActionForm form,
141                               HttpServletRequest req,
142                               HttpServletResponse res) throws Exception
143     {
144         System.out.println("B.logic in execute(-,-,-,-) : RegisterAction"+this.hashCode());
145         RegisterForm fm=(RegisterForm) form;
146
147         // read form data
148         String uname=fm.getUsername();
149         String pwd=fm.getPassword();
150
151         System.out.println("control is leaving execute(-,-,-,-) of RegisterAction");
152         //write b.logic
153         if(uname.equals("Durga")&& pwd.equals("soft"))
154         {
155             req.setAttribute("msg","valid Credentials");
156             return mapping.findForward("result");
157         }
158         else
159         {
160             req.setAttribute("msg","Invalid Credentials");
161             return mapping.findForward("result");
162         }
163     }//execute
164 } //class
165 -----myfile.properties-----
166 #my properties file
167 my.un=Enter Username
168 my.pwd=Enter Password
169
170 my.btn.cap=Login
171 my.Title=<b><center>Welcome to DURGA </center></b>
172
173 #Default errors msgs of validator rules
174     errors.required={0} is required.
175     errors.minLength={0} can not be less than {1} characters.
176     errors.maxLength={0} can not be greater than {1} characters.
177     errors.invalid={0} is invalid.
178
179     errors.byte={0} must be a byte.
180     errors.short={0} must be a short.
181     errors.integer={0} must be an integer.
182     errors.long={0} must be a long.
183     errors.float={0} must be a float.
```

STRUSTS

```
184     errors.double={0} must be a double.  
185  
186     errors.date={0} is not a date.  
187     errors.range={0} is not in the range {1} through {2}.  
188     errors.creditcard={0} is an invalid credit card number.  
189     errors.email={0} is an invalid e-mail address.  
190  
191     errors.header=<font color=red>  
192     errors.footer =</font>  
193  
194
```

DURGASOFT

```

1  >>>>>App4 (Struts App having Client Side , Server Side Declarative Form validations)
2  -----register.jsp-----
3  <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
4  <html>
5  <head>
6  <script language="JavaScript">
7  function myValidate(frm)
8  {
9  //call ValidatorPlugin Generated Dynamic Javascript function.
10 var flag=validateRf(frm);
11 if(flag==false)
12 {
13 return false;
14 }
15 else
16 {
17 //write programmatic from validation logic
18 var unval=frm.username.value;
19 var pwdval=frm.password.value;
20 if(unval.length!=pwdval.length)
21 {
22 alert("username and password must have same length");
23 frm.password.focus();
24 return false;
25 }
26 return true;
27 }
28 }//myValidate(-,-)
29 </script>
30 </head>
31 <html:javascript formName="rf"/>
32
33 <html:form action = "register" method="get" onsubmit="return myValidate(this)">
34 Username:<html:text property = "username"/><br>
35 Password:<html:password property = "password"/><br>
36 <html:submit value="checkDetails"/>
37 </html:form>
38
39 <html:errors/>
40 -----web.xml-----
41 <web-app>
42 <servlet>
43 <servlet-name>action</servlet-name>
44 <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
45 <init-param>
46 <param-name>config</param-name>
47 <param-value>/WEB-INF/struts-config.xml</param-value>
48 </init-param>
49 <load-on-startup>2</load-on-startup>
50 </servlet>
51
52 <servlet-mapping>
53 <servlet-name>action</servlet-name>
54 <url-pattern>*.do</url-pattern>
55 </servlet-mapping>
56 </web-app>
57 -----struts-config.xml-----
58 <!DOCTYPE struts-config PUBLIC
59      "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
60      "http://struts.apache.org/dtds/struts-config_1_3.dtd">
61 <struts-config>

```

**STRUTS****DURGASOFT****S**

```
62 <form-beans> 12
63   <form-bean name="rf" type="app.RegisterForm"/> 12
64 </form-beans> 12
65 12
66 <action-mappings> 12
67   <action path="/register" type="app.RegisterAction" name="rf" input="/register.jsp" 12
       validate="true"> 12
68     <forward name="success" path="/success.jsp"/> 12
69     <forward name="failure" path="/failure.jsp"/> 12
70   </action> 13
71 </action-mappings> 13
72 13
73 <message-resources parameter="myfile"/> 13
74 13
75 <plug-in className="org.apache.struts.validator.ValidatorPlugin"> 13
76   <set-property property="pathnames" 13
       value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/> 13
77 </plug-in> 13
78 </struts-config> 13
79 -----myfile.properties----- 14
80 # Struts Validator Default Error Messages 14
81 errors.required={0} is required. 14
82 errors.minLength={0} can not be less than {1} characters. 14
83 errors.maxLength={0} can not be greater than {1} characters. 14
84 errors.invalid={0} is invalid. 14
85 errors.byte={0} must be a byte. 14
86 errors.short={0} must be a short. 14
87 errors.integer={0} must be an integer. 14
88 errors.long={0} must be a long. 14
89 errors.float={0} must be a float. 15
90 errors.double={0} must be a double. 15
91 errors.date={0} is not a date. 15
92 errors.range={0} is not in the range {1} through {2}. 15
93 errors.creditcard={0} is an invalid credit card number. 15
94 errors.email={0} is an invalid e-mail address. 15
95 15
96 # user-defined msgs supplying {0} value for required rule error msg 157
97 my.un=Login Username 158
98 my.pass=Login Password 159
99 160
100 my.mask.err.msg=Username must contain only alphabets 161
101 my.unpwd.len=Username and password must have same length 162
102 my.custrule.msg={0} must contain same first and last characters 163
103 164
104 # for styles on form validation error nisgs 165
105 errors.header=<ul><font color=red> 166
106 errors.footer=</ul></font> 167
107 errors.suffix=</li> 168
108 errors.prefix=<li> 169
109 -----RegisterForm.java----- 170
110 package app; 171
111 172
112 import org.apache.struts.action.*; 173
113 import org.apache.struts.validator.*; 174
114 import javax.servlet.http.*; 175
115 176
116 public class RegisterForm extends DynaValidatorForm 177
117 { 178
118   // no FormBean properties 179
119 180
120   // no setXxx(-) and getXxx() 181
```

## STRUTS

## DURGASOFT

```
121     public ActionErrors validate(ActionMapping mapping,HttpServletRequest req)
122     {
123         // call super.validate(-,-) to perform Validator Plguin based
124         //form validations(approach2)
125         ActionErrors errs=super.validate(mapping,req);
126
127         // read form data from FormBean properties
128         String user=(String)get("username");
129         String pwd=(String)get("password");
130
131         //develop custom form validation logic(programmatic)
132         //check weather username and passwords are having same length or not
133         if(errs.size()==0)
134         {
135             if(user.length()!=pwd.length())
136             {
137                 errs.add("unpwderr",new ActionMessage("my.unpwd.len"));
138             }//if
139         }//if
140         System.out.println("size of errs is"+errs.size());
141         return errs;
142
143     }//validate(-,-)
144 }
145 -----validation.xml-----
146 <!DOCTYPE form-validation PUBLIC
147     "-//Apache Software Foundation//DTD Commons Validator Rules Configuration 1.3.0//EN"
148     "http://jakarta.apache.org/commons/dtds/validator_1_3_0.dtd">
149
150 <form-validation>
151     <formset>
152         <form name="rf">
153
154             <field property="username" depends="required,mask,myrule">
155                 <arg position="0" key="my.un"/>
156
157                 <msg name="mask" key="my.msk.msg"/>
158                 <var>
159                     <var-name>mask</var-name>
160                     <var-value>^[A-Za-z0-9]*$</var-value>
161                 </var>
162             </field>
163
164             <field property="password" depends="required,myrule">
165                 <arg position="0" key="my.pass"/>
166             </field>
167         </form>
168     </formset>
169 </form-validation>
170
171 -----validator-rules.xml-----
172 <!DOCTYPE form-validation PUBLIC
173     "-//Apache Software Foundation//DTD Commons Validator Rules Configuration 1.3.0//EN"
174     "http://jakarta.apache.org/commons/dtds/validator_1_3_0.dtd">
175
176 <form-validation>
177     <global>
178         <validator name="myrule"
179             classname="MyCustRule"
180             method="validateMyRule"
181             mcthodParams="java.lang.Object,
```

```

182         org.apache.commons.validator.ValidatorAction,
183         org.apache.commons.validator.Field,
184         org.apache.struts.action.ActionMessages,
185         org.apache.commons.validator.Validator,
186         javax.servlet.http.HttpServletRequest"
187         msg="my.custrule.msg"
188         jsFunction="validateMyRule"/>
189
190 .....
191 .....
192 </global>
193 </form-validation>
194 -----RegisterAction.java-----
195 package app;
196 import org.apache.struts.action.*;
197 import org.apache.struts.validator.*;
198
199 import javax.servlet.http.*;
200
201 public class RegisterAction extends Action
202 {
203
204     public RegisterAction()
205     {
206         System.out.println("RegisterAction:0-arg constructor");
207     }
208
209
210     public ActionForward execute(ActionMapping mapping,ActionForm form,
211                             HttpServletRequest request,
212                             HttpServletResponse response) throws Exception
213     {
214         //type casting
215         RegisterForm fm = (RegisterForm)form;
216
217         //read data from FormBean class obj
218         String user =(String) fm.get("username");
219         String pass =(String)fm.get("password");
220
221         // b.logic
222         if(user.equals("durga") && pass.equals("soft"))
223             return mapping.findForward("success");
224         else
225             return mapping.findForward("failure");
226     } //execute
227 } //class
228
229 -----MyCustRule.java-----
230 //MyCustRule.java (Dévelope based on FieldChecks class of struts-core-1.3.8.jar file)
231 import javax.servlet.http.*;
232 import org.apache.commons.validator.*;
233 import org.apache.commons.validator.util.*;
234 import org.apache.struts.action.*;
235 import org.apache.struts.validator.*;
236
237 public class MyCustRule
238 {
239     public static boolean validateMyRule(Object bean, ValidatorAction va,
240                                         Field field, ActionMessages errors,
241                                         Validator validator, HttpServletRequest request)
242     {

```

```

243     // read formbean property data
244     String value = ValidatorUtils.getValueAsString(bean, field.getProperty());
245
246     // gather first ,last characters
247     char fchar=value.charAt(0);
248     char lastchar=value.charAt(value.length()-1);
249
250     //form validation logic (check weather first and last characters are same or not)
251     if(fchar!=lastchar)
252     {
253         // add Validation error to ActionError class obj
254         errors.add(field.getKey(), Resources.getActionMessagevalidator, request, va, field));
255         return false;
256     }
257     else
258     {
259         return true;
260     }
261 } //validateMyRule
262 } //class
263 -----ValidateMyRule.js-----
264 //develop this code by taking validateRequired.js file as reference file
265
266 function validateMyRule(form) {
267     var isValid = true;
268     var focusField = null;
269     var i = 0;
270     var fields = new Array();
271
272     var oRequired = eval('new ' + jcv_retrieveFormName(form) + '_myrule()');
273
274     for (var x in oRequired) {
275         if (!jcv_verifyArrayElement(x, oRequired[x])) {
276             continue;
277         }
278         var field = form[oRequired[x][0]];
279
280         if (!jcv_isFieldPresent(field)) {
281             fields[i++] = oRequired[x][1];
282             isValid=false;
283         }
284         else if (( field.type == 'text' ||
285             field.type == 'textarea' ||
286             field.type == 'password')) {
287             //gathers comp value
288             var value = field.value;
289
290             //form validation logic
291             var fchar=value.charAt(0);
292             var lchar=value.charAt(value.length-1);
293
294             if (fchar!=lchar) {
295                 if ((i == 0) ) {
296                     focusField = field;
297                 } //iff
298                 fields[i++] = oRequired[x][1];
299                 isValid = false;
300             } //if
301         } //else
302
303     } //for

```

## STRUTS

DURGASOFT

```
304     if (fields.length > 0) {
305         jcv_handleErrors(fields, focusField);
306     }/if
307     return isValid;
308 }/validateMyRule
310 -----success.jsp-----
311 <% System.out.println("from success.jsp"); %>
312 <html>
313     <body>
314         <center>
315             <font size = 5 color = green>Login Successfull</font>
316         </center>
317     </body>
318 </html>
319 -----failure.jsp-----
320 <% System.out.println("from failure.jsp"); %>
321 <html>
322     <body>
323         <center>
324             <font size = 5 color = red>Login Failure</font><br>
325             <a href = "register.jsp">Try Again</a>
326         </center>
327     </body>
328 </html>
329
330
331
332
333
334
335
336
337
338
339
340
341
```

```

1 App4(Struts to DB s/w communication + Declarative form validations+Dynamic Form Beans)
2 =====
3 -----Register.jsp-----
4 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
5 <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
6 <html:form action="/insert">
7     <center>
8         <h1><u> Registration Screen</u></h1>
9         <h2>
10            <table border="0" width="100%" align="center">
11                <tr>
12                    <td><bean:message key="registration.id" /></td>
13                    <td><html:text property="id" /></td>
14                </tr>
15                <tr>
16                    <td><bean:message key="registration.name" /></td>
17                    <td><html:text property="name" /></td>
18                </tr>
19                <tr>
20                    <td><bean:message key="registration.address" /></td>
21                    <td><html:text property="address" /></td>
22                </tr>
23                <tr>
24                    <td><bean:message key="registration.doj" /></td>
25                    <td><html:text property="doj" /></td>
26                </tr>
27                <tr>
28                    <td><bean:message key="registration.age" /></td>
29                    <td><html:text property="age" /></td>
30                </tr>
31                <tr>
32                    <td><bean:message key="registration.email" /></td>
33                    <td><html:text property="email" /></td>
34                </tr>
35            </table><br><br><center>
36
37            <html:submit>
38                <bean:message key="btn.cap"/>
39            </html:submit>
40        </h2>
41    </html:form>
42 -----web.xml-----
43 <web-app>
44     <servlet>
45         <servlet-name>action</servlet-name>
46         <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
47         <init-param>
48             <param-name>config</param-name>
49             <param-value>/WEB-INF/struts-config.xml</param-value>
50         </init-param>
51         <load-on-startup>1</load-on-startup>
52     </servlet>
53
54     <servlet-mapping>
55         <servlet-name>action</servlet-name>
56         <url-pattern>*.do</url-pattern>
57     </servlet-mapping>
58
59     <welcome-file-list>
60         <welcome-file>Register.jsp</welcome-file>
61     </welcome-file-list>

```

## STRUTS

## DURGASOFT

```
62 </web-app>
63 -----struts-config.xml-----
64 <!DOCTYPE struts-config PUBLIC
65   "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
66   "http://struts.apache.org/dtds/struts-config_1_3.dtd">
67 <struts-config>
68   <form-beans>
69     <form-bean name="bean" type="org.apache.struts.validator.DynaValidatorForm">
70       <form-property name="id" type="java.lang.String" />
71       <form-property name="name" type="java.lang.String" />
72       <form-property name="address" type="java.lang.String" initial="hyd"/>
73       <form-property name="email" type="java.lang.String" />
74       <form-property name="doj" type="java.lang.String" />
75       <form-property name="age" type="java.lang.String" />
76     </form-bean>
77   </form-beans>
78   <action-mappings>
79     <action name="bean"
80       path="/insert"
81       type="RegisterAction"
82       input="/Failure.jsp"
83       validate="true">
84       <forward name="ok" path="/Success.html"/>
85       <forward name="fail" path="/Failure.html"/>
86     </action>
87   </action-mappings>
88
89   <message-resources parameter="ApplicationResources"/>
90
91   <plug-in className="org.apache.struts.validator.ValidatorPlugin">
92     <set-property property="pathnames"
93       value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
94   </plug-in>
95
96 </struts-config>
97 -----validation.xml-----
98 <?xml version="1.0" encoding="ISO-8859-1" ?>
99
100 <!DOCTYPE form-validation PUBLIC
101   "-//Apache Software Foundation//DTD Commons Validator Rules Configuration 1.3.0//EN"
102   "http://jakarta.apache.org/commons/dtds/validator_1_3_0.dtd">
103
104 <form-validation>
105   <formset>
106     <form name="bean">
107       <field property="id" depends="required,integer">
108         <arg position="0" key="my.error.id" />
109       </field>
110
111       <field property="name" depends="required,mask">
112         <arg position="0" key="my.error.name" />
113         <msg name="mask" key="my.error.mask.name" />
114         <var>
115           <var-name>mask</var-name>
116           <var-value>^a-zA-Z*$</var-value>
117         </var>
118       </field>
119
120       <field property="address" depends="required,minlength">
121         <arg position="0" key="my.error.address" />
122         <arg position="1" name="minlength" key="${var:minlength}" resource="false"/>
```

```

123      <var>
124          <var-name>minlength</var-name>
125          <var-value>10</var-value>
126      </var>
127  </field>
128
129  <field property="email" depends="required, email">
130      <arg position="0" key="my.error.email" />
131  </field>
132
133  <field property="age" depends="required,integer,range" >
134      <arg position="0" key="my.error.age" />
135
136      <arg position="1" name="range" key="${var:min}" resource="false" />
137      <arg position="2" name="range" key="${var:max}" resource="false"/>
138      <var>
139          <var-name>min</var-name>
140          <var-value>20</var-value>
141      </var>
142      <var>
143          <var-name>max</var-name>
144          <var-value>50</var-value>
145      </var>
146  </field>
147
148  <field property="doj" depends="required,date" >
149      <arg position="0" key="my.error.doj" />
150      <var>
151          <var-name>datePattern</var-name>
152          <var-value>yyyy-MM-dd</var-value>
153      </var>
154  </field>
155  </form>
156  </formset>
157 </form-validation>
158 -----RegisterAction.java-----
159 import javax.servlet.http.*;
160 import org.apache.struts.action.*;
161 import java.sql.*;
162 import org.apache.struts.validator.DynaValidatorForm;
163
164 public class RegisterAction extends Action
165 {
166     public ActionForward execute(ActionMapping mapping,
167         ActionForm form,
168         HttpServletRequest request,
169         HttpServletResponse response)
170     {
171
172         Connection con = null;
173         PreparedStatement ps = null;
174         String status = null;
175
176         try
177         {
178             System.out.println("Action class execute() method is called");
179             DynaValidatorForm b=(DynaValidatorForm)form;
180
181             Class.forName("oracle.jdbc.driver.OracleDriver");
182             con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl", "scott", "tiger");
183

```

## STRUTS

DURGASOFT

```
184     ps = con.prepareStatement("INSERT INTO STRUTS_STUDENT VALUES(?,?,?,?,?,?)");
185
186     ps.setInt(1, Integer.parseInt((String)b.get("id")));
187     ps.setString(2, (String)b.get("name"));
188     ps.setString(3, (String)b.get("address"));
189     ps.setString(4, (String)b.get("email"));
190     ps.setDate(5, java.sql.Date.valueOf((String)b.get("doj")));
191     ps.setInt(6, new Integer((String)b.get("age")).intValue());
192
193     status = null;
194     if(ps.executeUpdate() == 0)
195         status = "fail";
196     else
197         status = "ok";
198 //try
199     catch(ClassNotFoundException ce)
200 {
201     ce.printStackTrace();
202 }
203     catch(SQLException se)
204 {
205     se.printStackTrace();
206 }
207     finally
208 {
209     try
210     {
211         if(ps != null)
212             ps.close();
213     }
214     catch(SQLException se)
215     {}
216     try
217     {
218         if(con != null)
219             con.close();
220     }
221     catch(SQLException se)
222     {}
223 } //finally
224
225     return mapping.findForward(status);
226 } // execute()
227 } // class
228 -----success.html-----
229 <HTML>
230   <BODY>
231     <CENTER><BR><BR>
232     <H1>Registration Successful !!!</H1>
233   </BODY>
234 </HTML>
235 -----failure.jsp-----
236 <HTML>
237   <BODY>
238     <CENTER><BR><BR>
239     <H1>Registration Failure !!!</H1>
240   </BODY>
241 </HTML>
242 -----ApplicationResources.properties-----
243 # Struts Validator Error Messages
244 errors.header=<h1>List of error(s)</h1><h2><hr>
```

## STRUTS

DURGASOFT

```
245 errors.footer=</h2><hr>
246 errors.required={0} is required.
247 errors.minLength={0} can not be less than {1} characters.
248 errors.maxLength={0} can not be greater than {1} characters.
249 errors.invalid={0} is invalid.
250
251 errors.byte={0} must be a byte.
252 errors.short={0} must be a short.
253 errors.integer={0} must be an integer.
254 errors.long={0} must be a long.
255 errors.float={0} must be a float.
256 errors.double={0} must be a double.
257
258 errors.date={0} is not a date.
259 errors.range={0} is not in the range {1} through {2}.
260 errors.creditcard={0} is an invalid credit card number.
261 errors.email={0} is an invalid e-mail address.
262
263 my.error.id=<br><u>ID</u>
264 my.error.name=<br><u>Name</u>
265 my.error.address=<br><u>Address</u>
266 my.error.email=<br><u>Email</u>
267 my.error.age=<br><u>Age</u>
268 my.error.doj=<br><u>Date of Joining</u>
269 my.error.mask.name=<br>Please provide only alphabets for <u>Name</u>
270
271 registration.id=Student ID
272 registration.name=First Name
273 registration.email=E-Mail Address
274 registration.address=Resedential Address
275 registration.doj=Date of Joining (YYYY-MM-DD)
276 registration.age=Age
277 -----sql.txt-----
278 SQL> desc struts_student;
279 Name          Type
280 -----
281
282 ID           NUMBER
283 NAME         VARCHAR2(20)
284 ADDRESS      VARCHAR2(30)
285 EMAIL        VARCHAR2(40)
286 DOJ          DATE
287 AGE          NUMBER
288
289
290 create table struts_student (
291 ID           NUMBER,
292 NAME         VARCHAR2(20),
293 ADDRESS      VARCHAR2(30),
294 EMAIL        VARCHAR2(40),
295 DOJ          DATE,
296 AGE          NUMBER);
```

## ACTION CLASSES

### **DispatchAction:**

DispatchAction provides a mechanism for grouping a set of related functions into a single action, thus eliminating the need to create separate actions for each function. In this example we will see how to group a set of user related actions like add user, update user and delete user into a single action called UserAction.

The class UserAction extends org.apache.struts.actions.DispatchAction. This class does not provide an implementation of the execute() method as the normal Action class does. The DispatchAction uses the execute method to manage delegating the request to the individual methods based on the incoming request parameter. For example if the incoming parameter is "method=add", then the add method will be invoked. These methods should have similar signature as the execute method.

#### ***org.apache.struts.actions***

##### ***Class DispatchAction***

`java.lang.Object`

  └ `org.apache.struts.action.Action`

    └ `org.apache.struts.actions.BaseAction`

      └ `org.apache.struts.actions.DispatchAction`

##### **Direct Known Subclasses:**

`EventDispatchAction, LookupDispatchAction, MappingDispatchAction`

public abstract class **DispatchAction**

extends `BaseAction`

An abstract **Action** that dispatches to a public method that is named by the request parameter whose name is specified by the parameter property of the corresponding ActionMapping. This Action is useful for developers who prefer to combine many similar actions into a single Action class, in order to simplify their application design.

To configure the use of this action in your struts-config.xml file, create an entry like this:

```
<action path="/saveSubscription" type="org.apache.struts.actions.DispatchAction"
name="subscriptionForm" scope="request" input="/subscription.jsp" parameter="method"/>
```

which will use the value of the request parameter named "method" to pick the appropriate "execute" method, which must have the same signature (other than method name) of the standard Action.execute method. For example, you might have the following three methods in the same action:

- `public ActionForward delete(ActionMapping mapping, ActionForm form,
HttpServletRequest request, HttpServletResponse response) throws Exception`
- `public ActionForward insert(ActionMapping mapping, ActionForm form, HttpServletRequest
request, HttpServletResponse response) throws Exception`
- `public ActionForward update(ActionMapping mapping, ActionForm form,
HttpServletRequest request, HttpServletResponse response) throws Exception`

and call one of the methods with a URL like this:

`http://localhost:8080/myapp/saveSubscription.do?method=update`

## **STRUTS**

## **DURGASOFT**

**NOTE** - All of the other mapping characteristics of this action must be shared by the various handlers. This places some constraints over what types of handlers may reasonably be packaged into the same DispatchAction subclass.

**NOTE** - If the value of the request parameter is empty, a method named unspecified is called. The default action is to throw an exception. If the request was cancelled (a html:cancel button was pressed), the custom handler cancelled will be used instead. You can also override the getMethodName method to override the action's default handler selection.

### **Advanced Action Classes**

In this chapter, we dig further into the Controller components of the Struts framework by covering the built-in Action classes that come with Struts. Our goal is to provide you with a solid understanding of the Struts built in actions and how they can be used to facilitate the design of your struts applications.

The Struts Framework provides several built in actions. A few of these are essential for any Struts applications. Others are necessary to add cohesion to that would normally be a granular collection of related actions.

### **ForwardAction and Beyond**

Generally speaking it is a bad idea to place within you JSP pages any direct links to other JSP pages. For one reason it is not a good design decision; the struts-config.xml file, which is part of the Controller, should contain the entire flow of your application.

In the Model-View-Controller(MVC) architecture, it is the role of the Controller to select the next View. The Controller consists of Struts configuration, the ActionServlet, and the actions. If you add a link directly to another JSP, you are violating the architectural boundaries of the Model 2 architecture. (Model 2 is the instantiation of the MVC architecture for Web applications).

At times, however, all you really need is just a plain link; you don't want (or need) an action to execute first. Perhaps there are no objects from the domain that need to be mapped into scope in order for the view to display. Perhaps the page is very simple. In this case, a better approach is to use the Forward Action.

The ForwardAction acts as a bridge from the current View (JSP) and the pages it links to. It uses the RequestDispatcher to forward to a specified Web resource. It is the glue that allows you to link to an action instead of directly to a JSP.

Later, if you need to, you can change the ActionMapping in the Struts configuration file so that every page that linked to that action will link to the new action. Also, you can change the action to a custom one that you write instead of using the ForwardAction that Struts provides.

### **To use the ForwardAction, follow these steps:**

1. Using the html:link tag with the action attribute, add a link to the JSP page that points to the action.
2. Create an action mapping in the Struts configuration file that uses the ForwardAction with the parameter attribute to specify the JSP path.

Let's say you have a JSP page that has a direct link to another JSP page:

```
<html:link page="/index.jsp">Home</html:link>
```

You have recently converted to the MVC/Model 2 architecture religion and you want to change the html:link tag to link to an action. Because you already have a link, simply changes it as follows:

```
<html:link action="home">Home</html:link>
```

## **STRUTS**

All you do is remove the page attribute and add an action attribute that points to the home action. Now you have to create the home action mapping. (The link in the previous code snippet would be expanded to a URL, link <http://localhost:8080/actgions/home.do>.)

To add an action mapping to the home action that you referenced in your html:link tag, use this code:

```
<action  
    Path="/home"  
    Type="org.apache.struts.actions.ForwardAction"  
    Parameter="/index.jsp"  
/>
```

The ForwardAction uses the parameter attribute so that you can specify the path. Notice the parameter is set to /index.jsp, which was what the page attribute of the html:link tag was originally set to. Thus, the parameter attribute indicated to where you what the ForwardAction to forward.

## **Linking to JSP Directly: More than Just Bad Design**

In addition to being a bad design decision linking directly to JSP pages may result in errors. The Controller maps in the correct ModuleConfig based On the request. Having the correct module implies that the resource bundlers for that module are loaded. If the JSP page uses the resource bundles to display massages (which is required for internationalized applications), then you need to link to the action directly because that is the only way the JSP page is guaranteed to work properly.

## **Linking to Global Forwards Instead of Actions**

From a purely design perspective, there are other alternatives to using the ForwardAction. Rather than linking to an action or a page, you could link to a global forward:

```
<html:link forward ="home">Home</html:link>
```

Thus, the best approach for our example is to add this in the JSP:

```
<html:link action="home">Home</html:link>
```

Then add this in the global forwards section of the Struts configuration file:

```
<forward name="home" path="/home.do"/>
```

And this the action mappings section of the Struts configuration file:

```
<action path="/home" forward="/index.jsp"/>
```

Concise, straightforward, and well designed. Exactly what we needed!

## **Don't Bleed Your V into Your C**

If you find that you have a lot of links to JSP pages from other JSP pages, you may not understand MVC very well. Using ActionForward may mask the fact that your design is inherently messed up. JSPs linking to other JSPs, whether they use ForwardAction or not, works only for the simplest of Web applications in MVC. If it works for your application, you are probably doing something wrong.

Complex MVC Web applications typically need an action to execute first; it is the job of the action to map Model items into scope so that the View (typically JSP) can display them. If this is not the case for your nontrivial Web application, you are probably putting too much logic in the JSP pages (or in custom tags that JSP pages use) - which can be the source of huge maintenance issues.

## **DURGASOFT**

## **STRUTS**

## **DURGASOFT**

The JSP pages and custom tags should contain only display logic, and this logic should be kept to a bare minimum. Thus, the JSP pages and custom tags should not talk directly to the Model-they only display different pieces of the Model. In essence, they should speak only to the Value object and Data Transfer objects. Again the action talks directly to the Model and delegates the display of the Model objects to the View. The Model in turn implements the persistence and business rules for the application.

Adopting a strict aMVC architecture is generally a good idea; it keeps your JSP smaller and more focused. JSP are harder to test than actions, so adopting MVC increases the Liquidity and flexibility of your code base.

In short, if you overuse the ForwardAction, you need to evaluate your understanding of the MVC architrave.

### **Forwards for Legacy Access**

A lot of folks wrote Web applications before Struts existed. A lot of other folks have had to integrate with commercial portal implementations and other frameworks that are built on top of servlets (legacy and otherwise) using the ForwardAction (or the forward attribute) is a good way to encapsulate this legacy integration of your View components (JSP and custom tags). In this manner, the ForwardAction acts as "legacy glue" to other Web resource.

Let's say that you have a legacy Web resource that you want to use with the form validation of Struts. However, your legacy resources are part of an elaborate MVC based Web application framework that you created before the dawn of the Struts Framework.

This legacy resource does all types of neat things that you have not ported to the Struts Framework yet. For some reason (time and money come to mind), you do not want to rewrite this legacy resource-at least not yet.

For the purpose of simplicity, our sample legacy Web resource will be a servlets. Essentially, you want the servlets do Get method to be called only if the ActionForm validate. Here is our example servlets:

```
public class LegacyServlet extends HttpServlet
{
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException
    {
        //Get the mapping
        ActionMapping mapping=(ActionMapping)request.getAttribute(Globals.MAPPING_KEY);
        //Get the UserForm
        UserForm form=(UserForm)request.getAttribute(mapping.getName());
        //Do some fly, super, whiz
        //bang legacy stuff
        //Generate some output
        response.getWriter().println("User name"+form.getFirstName());
    }
};
```

## STRUTS

## DURGASOFT

Notice how the servlet can access the context that the Struts Framework mapped into request scope. Now suppose this servlet was mapped into our Web application like so:

```
<servlet>
    <servlet-name>legacy</servlet-name>
    <servlet-class>legacy.LegacyServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>legacy</servlet-name>
        <url-pattern>legacy/roar</url-pattern>
    </servlet-mapping>
```

Thus, posts to /legacy/roar would cause this servlet's doPost() method to be called. Now, to map this servlet as an action that acts as a form handler, you need to do this:

```
<action
    path="/register"
    forward="/legacy/roar"
    input="/form/userForm.jsp"
    name="userForm"
    parameter="/legacy/roar"
    validate="true"
    scope="request"
/>
```

Of course, the previous code assumes that you have a from bean:

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

Your input JSP(/from/userForm.jsp) would look like this:

```
.....
<h1>Legacy: Struts Form for userFroms </h1>
<html:form action="/legacy">
    Email :
    <html:text property="email" /><br>
```

First Name :

```
<html:text property="firstName" /><br>
```

Last Name :  
    <html:text property="lastName" /></br>

Password :  
    <html:password property="password" /></br>

Password Check:  
    <html:password property="passwordCheck"/></br>

UserName :  
    <html:text property="userName" /></br>  
    <html:submit/> <html:cencel/>  
</html:form>

Notice that the html:form tag points to an action path, just as you expect. Because the RequestDispatcher.forward method to the Servlet is invoked only if the ForwardAction's execute method is invoked, the depots method of the legacy servlet is called only if the ForwardAction's execute method is called, we set the validate attribute to true in the action mapping of this servlet, so the execute method of the ForwardAction is called only if the ActionForm (UserForm) validates (returns no ActionError objects).

**InculeAction:**

To understand the InculeAction, be sure to read about the ForwardAction in the previous section. The InculeAction is similar to the ForwardAction but is not used as much. You could rewrite the last example using the InculeAction as follows:

```
<action  
    path="/register"  
    type="org.apache.struts.actions.IncludeAction"  
    input="/form/userForm2.jsp"  
    name="userForm"  
    parameter="/legacy/roar"  
    validate="true"  
    scope="request"  
/>
```

This shorter form uses the include attribute:

```
<action  
    path="/register"  
    include="/legacy/roar"  
    input="/form/userForm2.jsp"  
    name="userForm"  
    parameter="/legacy/roar"  
    validate="true"  
    scope="request"  
/>
```

## **STRUTS**

## **DURGASOFT**

So what is the difference between the includeAction and the ForwardAction? The difference is that you need to use the includeAction only if the action is going to be included by another action or JSP.

Therefore, if you have code in your JSP that looks like this:

```
<jsp:include page="/someWebApp/someModule/someAction.do"/>
```

The action could not use a ForwardAction because it would forward control to the new rather than including its output within the output of the JSP -or throw a nasty IllegalStateException if the output buffer was already committed.

This discussion also applies to using the Tiles framework. A Tile can be Web resource, and an action can be a Web resource. This , you could define a Tile as an action. Refer to Chapters 13 and 17 for more details.

## **DispatchAction**

Oftentimes actions seem to be too plentiful and too small. It would be nice to group related actions into one class.

For example, let's say that a group of related actions all work on the same set of objects in the user session (HttpSession)- a shopping cart, for example. Another example is a group of actions that are all involved in the same use case. Yet another example is a group of actions that are all involved in CRUD operations on domain objects. (CRUD stands for create, read, update, and delete. Think of an add/edit/delete/listing of products for an online e-commerce store)

If you can group related actions into one class, you can create helper methods that they all use, thus improving reuse (or at least facilitating it). Also if these helper methods are only used by these related actions and these actions are in the same class then the helper methods can be encapsulated (hidden) inside this one class.

The DispatchAction class is used to group related actions into one class. DispatchAction is an abstract class, so you must override it to use it. It extends the Action class.

Rather than having a single execute method, you have a method for each logical action. The DispatchAction dispatches to one of the logical actions represented by the methods. It picks a method to invoke based on an incoming request parameter. The value of the incoming request parameter is the name of the method that the DispatchAction will invoke.

To use the DispatchAction, follow these steps:

1. Create an action handler class that subclasses DispatchAction.
2. Create a method to represent each logical related Action.
3. Create an action mapping for this action handler using the parameter attribute to specify the request parameter that carries the name of the method you want to invoke.
4. Pass the action a request parameter that refers to the method you want to invoke.

First, you create an action handler class that subclasses DispatchAction:

```
public class UserDispatchAction extends DispatchAction {  
    ...  
}
```

Then, you create a method to represent each logical related action:

```
public class UserDispatchAction extends DispatchAction {
```

**STRUTS****DURGASOFT**

```
public ActionForward remove(
    ActionMapping mapping,
    ActionForm form,
    HttpServletRequest request,
    HttpServletResponse response)
    throws Exception

{
    System.out.println("Remove User");
    Return mapping.findForward("success");
}

public ActionForward save(
    ActionMapping mapping,
    ActionForm form,
    HttpServletRequest request,
    HttpServletResponse response)
    throws Exception

{
    System.out.println("Save User");
    return mapping.findForward("success");
}

}
```

Notice these methods have the same signature (other than the method name) of the standard Action. execute method.

```
ActionMapping mapping,
ActionForm form,
HttpServletRequest request,
HttpServletResponse response)
throws Exception

{
    System.out.println("SAVE USER (LOOKUP)");
    return mapping.findForward("success");
}

.....
```

## STRUTS

Notice these methods have the same signature of the standard Action.execute() method (except for the method name).

Third, you must implement the getKeyMethodMap() method to map the resource keys to method names:

```
<action
    path="/lookupDispatchUserSubmit"
    type="action.UserLookupDispatchAction"
    input="/form/userForm.jsp"
    name="userForm"
    parameter="method"
    validate="true"
    scope="request">
    <forward name="success" path="/success.jsp"/>
</action>
```

The fifth step is to set up the messages in the resource bundle for the labels and values of the buttons. Inside your bundle (e.g . application.properties), add the following two entries:

```
userForm.save=save
userForm.remove=remove
```

Finally, use bean:message to display the labels of the button in the bean:

```
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean"%>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html"%>
...
<html:link action="home">Home</html:link>
<html:form action="/lookupDispatchUserSubmit">
...
    <html:submit property="method">
        <bean:message key="userForm.remove"/>
    </html:submit>
    <html:submit property="method">
        <bean:message key="userForm.save"/>
    </html:submit>

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

## **STRUTS**

## **DURGASOFT**

When the user clicks the Remove button on the HTML form, the remove method is called on the action.

When the user clicks the Save button, the save method is called on the action.

### **SwitchAction**

The SwitchAction class is used to support switching from module to module. Let's say you have an action that wants to forward to an action in another module. You perform the following steps:

1. Map in a SwitchAction into your Struts Configuration file.
2. Create a forward or link to the SwitchAction that passes the page parameter and the module prefix parameter.

Let's break this down. Say you have a Web application that uses Struts. Struts has two modules: the default and a module called admin, as follows:

```
<servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
        <param-name>config</param-name>
        <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <init-param>
        <param-name>config/admin</param-name>
        <param-value>/WEB-INF/struts-config-admin.xml</param-value>
    </init-param>
    <init-param>
        <param-name>debug</param-name>
        <param-value>3</param-value>
    </init-param>
    <init-param>
        <param-name>detail</param-name>
        <param-value>3</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
```

The init parameter config/admin defines the admin module. The config init parameter defines the default module.

Now let's say you have an action in the default module that edits users and you want to delegate the display of those users to an action in the admin module. First, map a SwitchAction into the default module as shown here:

```
<action
    path="/switch"
    type="org.apache.struts.actions.SwitchAction">
```

```
</action>
```

Now you can set up a forward in the action that edits the users as follows:

```
<action
    path="/userSubmit"
    attribute="userform"
    input="/form/userForm.jsp"
    name="userform"
    scope="request"
    type="action.UserAction">
    <forward name="success"
        path="/switch.do?page=/listUsers.do&prefix=/admin"/>
</action>
```

Notice that this forward passes two request parameters. The page parameter specifies the module relative action. The second parameter specifies the prefix of the module—in this case admin. You don't have to use forwards to use the SwitchAction; any JSP can link to the SwitchAction to move to any module. The listUser.do action is not defined in the default module; it is defined in the admin. The forward in this example forwards to the action at the path /admin/listUsers.do. The listUser.do is defined in /WEB-INF/struts-config-admin.xml, and the userSubmit.do action is defined in /WEB-INF/struts-config.xml.

```

1 App5(Basic DispatchAction Application)
2 -----
3 -----Index.jsp-----
4 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
5
6 <html:html>
7   <h2><center>
8     Please click on a link below, to perform your desired activity.
9     <br><br>
10    <html:link action="/add">Add User</html:link><br>
11    <html:link action="/modify">Modify User</html:link><br>
12    <html:link action="/delete">Delete User</html:link>
13  </center></h2>
14 </html:html>
15 -----web.xml-----
16 <?xml version="1.0" encoding="ISO-8859-1"?>
17 <!DOCTYPE web-app
18 PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
19 "http://java.sun.com/j2ee/dtds/web-app\_2\_3.dtd">
20
21 <web-app>
22   <servlet>
23     <servlet-name>Dummy</servlet-name>
24     <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
25     <init-param>
26       <param-name>config</param-name>
27       <param-value>/WEB-INF/struts-config.xml</param-value>
28     </init-param>
29     <load-on-startup>1</load-on-startup>
30   </servlet>
31
32   <servlet-mapping>
33     <servlet-name>Dummy</servlet-name>
34     <url-pattern>*.do</url-pattern>
35   </servlet-mapping>
36   <welcome-file-list>
37     <welcome-file>Index.jsp</welcome-file>
38   </welcome-file-list>
39 </web-app>
40 -----struts-config.xml-----
41 <!DOCTYPE struts-config PUBLIC
42   "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
43   "http://struts.apache.org/dtds/struts-config\_1\_3.dtd">
44
45 <struts-config>
46   <form-beans>
47     <form-bean name="bean" type="org.apache.struts.action.DynaActionForm">
48       <form-property name="id" type="java.lang.String"/>
49       <form-property name="pass" type="java.lang.String"/>
50       <form-property name="function" type="java.lang.String"/>
51     </form-bean>
52   </form-beans>
53
54   <global-forwards>
55     <forward name="success" path="/success.jsp" />
56     <forward name="failure" path="/failure.jsp" />
57   </global-forwards>
58
59   <action-mappings>
60     <action path="/controller1"
61           name="bean">

```

## STRUTS

## DURGASOFT

```
62      type="OurAction"
63      parameter="function"
64      scope="request"/>
65
66  <action path="/add" type="org.apache.struts.actions.ForwardAction"
67    parameter="/AddUser.jsp"/>
68  <action path="/delete" type="org.apache.struts.actions.ForwardAction"
69    parameter="/DeleteUser.jsp"/>
70  <action path="/modify" type="org.apache.struts.actions.ForwardAction"
71    parameter="/ModifyUser.jsp"/>
72  <action path="/home" type="org.apache.struts.actions.ForwardAction"
73    parameter="/Index.jsp"/>
74  </action-mappings>
75  <message-resources parameter="ApplicationResources"/>
76  </struts-config>
77 -----ApplicationResources.properties-----
78 form.id=<b><u>User Name</u></b>
79 form.pass=<b><u>Password</u></b>
80 -----AddUser.jsp-----
81 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
82 <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
83
84 <html:html>
85   <html:form action="/controller1">
86     <center><br><br>
87     <table>
88       <caption><font size = 6><u>Adding User</u></font></h2></u></caption>
89       <tr>
90         <td><bean:message key="form.id"/></td>
91         <td><html:text property="id"/></td>
92       </tr>
93       <tr>
94         <td><bean:message key="form.pass"/></td>
95         <td><html:password property="pass"/></td>
96       </tr>
97     </table>
98     <br>
99     <html:submit value="AddUser"/>
100    <html:hidden property="function" value="add"/>
101  </center>
102 </html:form>
103 -----ModifyUser.jsp-----
104 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
105 <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
106
107 <html:html>
108   <html:form action="/controller1">
109     <center><br><br>
110     <table>
111       <caption><font size = 6><u>Modifying User</u></font></h2></u></caption>
112       <tr>
113         <td><bean:message key="form.id"/></td>
114         <td><html:text property="id"/></td>
115       </tr>
116       <tr>
117         <td><bean:message key="form.pass"/></td>
118         <td><html:password property="pass"/></td>
119       </tr>
120     </table>
121     <br><br>
```

```

119      <html:submit value="UpdateUser"/>
120
121      <html:hidden property="function" value="modify"/>
122      </center>
123  </html:form>
124 </html:html>
125 -----DeleteUser.jsp-----
126 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
127 <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
128
129 <html:html>
130  <html:form action="/controller1">
131  <center><br><br>
132  <table>
133    <caption><font size = 6><u>Deleting User</font></h2></u></caption>
134    <tr>
135      <td><bean:message key="form.id"/></td>
136      <td><html:text property="id"/></td>
137    </tr>
138  </table>
139  <br><br>
140  <html:submit value="DeleteUser"/>
141
142  <html:hidden property="function" value="remove"/>
143 </center>
144 </html:form>
145 </html:html>
146 -----OurAction.java-----
147 import java.io.*;
148 import org.apache.struts.action.*;
149 import org.apache.struts.actions.*;
150 import java.sql.*;
151 import javax.servlet.http.*;
152 public class OurAction extends DispatchAction
153 {
154     private Connection getConnection()
155     {
156         Connection con = null;
157         System.out.println("getConnection() of OurAction");
158         try
159         {
160             Class.forName("oracle.jdbc.driver.OracleDriver");
161             con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl", "Scott",
162             "Tiger");
163         }
164         catch(Exception e)
165         {
166             return con;
167         } //getConnection()
168     private void releaseResources(Connection con, PreparedStatement pst)
169     {
170         System.out.println("releaseResources() of OurAction");
171         if(con != null)
172         {
173             try
174             {
175                 con.close();
176             }
177             catch(Exception e)
178             {

```

## STRUTS

```
179     }
180 }//if
181 if(pst != null)
182 {
183     try
184     {
185         pst.close();
186     }
187     catch(Exception e)
188     {
189     }
190 }//if
191 //releaseResources()
192
193 public ActionForward add(ActionMapping mapping,
194                         ActionForm f,
195                         HttpServletRequest request,
196                         HttpServletResponse response)
197 {
198     Connection con = null;
199     PreparedStatement pst = null;
200     System.out.println("add() of OurAction");
201     try
202     {
203         DynaActionForm form=(DynaActionForm)f;
204         con = this.getConnection();
205         pst=con.prepareStatement("INSERT INTO STRUTS_DISP_ACTION_TEST VALUES(?, ?)");
206         pst.setString(1,(String)form.get("id"));
207         pst.setString(2,(String)form.get("pass"));
208
209         int i = pst.executeUpdate();
210         request.setAttribute("Action", "Insert ");
211
212         if(i != 0)
213         {
214             return mapping.findForward("success");
215         }//if
216         else
217         {
218             return mapping.findForward("failure");
219         }
220     } // try
221     catch(Exception e)
222     {
223         return mapping.findForward("failure");
224     }
225     finally
226     {
227         releaseResources(con,pst);
228     }//finally
229 } // add()
230
231 public ActionForward remove(ActionMapping mapping,
232                            ActionForm f,
233                            HttpServletRequest request,
234                            HttpServletResponse response)
235 {
236     Connection con = null;
237     PreparedStatement pst = null;
238     System.out.println("remove() of OurAction");
239     try
```

## DURGASOFT

## STRUTS

DURGASOFT

```
240  {
241      DynaActionForm form=(DynaActionForm)f;
242      con=getConnection();
243      pst=con.prepareStatement("DELETE FROM STRUTS_DISP_ACTION_TEST WHERE ID=?");
244
245      pst.setString(1,(String)form.get("id"));
246
247      int i=pst.executeUpdate();
248
249      request.setAttribute("Action", "Deletion");
250
251      if(i != 0)
252      {
253          return mapping.findForward("success");
254      }
255      else
256      {
257          return mapping.findForward("failure");
258      }
259  } //try
260  catch(Exception e)
261  {
262      return mapping.findForward("failure");
263  }
264  finally
265  {
266      releaseResources(con, pst);
267  }
268 } // remove()
269
270 public ActionForward modify(ActionMapping mapping,
271                         ActionForm f,
272                         HttpServletRequest request,
273                         HttpServletResponse response)
274 {
275     Connection con = null;
276     PreparedStatement pst = null;
277     System.out.println("modify of OurAction");
278
279     try
280     {
281         DynaActionForm form=(DynaActionForm)f;
282         con = this.getConnection();
283         pst=con.prepareStatement("UPDATE STRUTS_DISP_ACTION_TEST SET PASS=? WHERE
284         ID=?");
285         pst.setString(1,(String)form.get("pass"));
286         pst.setString(2,(String)form.get("id"));
287
288         int i = pst.executeUpdate();
289         request.setAttribute("Action", "Updation");
290         if(i != 0)
291             return mapping.findForward("success");
292         else
293             return mapping.findForward("failure");
294     } //try
295     catch(Exception e)
296     {
297         return mapping.findForward("failure");
298     }
299     finally
```

```
300     releaseResources(con,pst);
301 }
302 } // modify()
303 } // OurAction class
304 -----succes.jsp-----
305 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
306 <html>
307   <body>
308     <div align="center">
309       <font color="#0000FF" size="5"><em><strong>
310         <%= (String)request.getAttribute("Action")%>Operation Successful <br>
311       </strong></em></font>
312     </div>
313     <center>
314       <br><br>
315       <html:link action="/home">HOME</html:link><br>
316     </center>
317   </body>
318 </html>
319 -----failure.jsp-----
320 <html>
321   <body>
322     <p align="center"><strong><font size="5"><em>
323       Operation failed !!! </em></font></strong>
324     </p>
325   </body>
326 </html>
327
```

```

1 App6(Mini Project)
2 =====
3 -----Index.jsp-----
4 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
5
6 <html:html>
7   <h2><center>
8     Please click on a link below, to perform your desired activity.
9   <br><br>
10    <html:link action="/add">Add User</html:link><br>
11    <html:link action="/modify">Modify User</html:link><br>
12    <html:link action="/delete">Delete User</html:link>
13  </center></h2>
14 </html:html>
15 -----web.xml-----
16 <?xml version="1.0" encoding="ISO-8859-1"?>
17
18 <!DOCTYPE web-app
19 PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
20 "http://java.sun.com/j2ee/dtds/web-app\_2\_3.dtd">
21
22 <web-app>
23   <servlet>
24     <servlet-name>Dummy</servlet-name>
25     <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
26     <init-param>
27       <param-name>config</param-name>
28       <param-value>/WEB-INF/struts-config.xml</param-value>
29     </init-param>
30     <load-on-startup>1</load-on-startup>
31   </servlet>
32   <servlet-mapping>
33     <servlet-name>Dummy</servlet-name>
34     <url-pattern>*.do</url-pattern>
35   </servlet-mapping>
36
37   <welcome-file-list>
38     <welcome-file>Index.jsp</welcome-file>
39   </welcome-file-list>
40 </web-app>
41 -----struts-config.xml-----
42 <!DOCTYPE struts-config PUBLIC
43   "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
44   "http://struts.apache.org/dtds/struts-config\_1\_3.dtd">
45
46
47 <struts-config>
48
49   <form-beans>
50     <form-bean name="bean" type="org.apache.struts.validator.DynaValidatorActionForm">
51       <form-property name="id" type="java.lang.String"/>
52       <form-property name="pass" type="java.lang.String"/>
53       <form-property name="function" type="java.lang.String"/>
54     </form-bean>
55   </form-beans>
56
57   <global-forwards>
58     <forward name="success" path="/success.jsp" />
59     <forward name="failure" path="/failure.jsp" />
60   </global-forwards>
61

```

```

62 <action-mappings>
63   <action path="/controller1" name="bean" type="OurAction"
64     parameter="function" validate="true" input="/error.jsp" scope="request"/>
65
66   <action path="/controller2" name="bean" type="OurAction"
67     parameter="function" validate="true" input="/error.jsp" scope="request"/>
68
69   <action path="/add" type="org.apache.struts.actions.ForwardAction"
70     parameter="/AddUser.jsp"/>
71   <action path="/delete" type="org.apache.struts.actions.ForwardAction"
72     parameter="/DeleteUser.jsp"/>
73   <action path="/modify" type="org.apache.struts.actions.ForwardAction"
74     parameter="/ModifyUser.jsp"/>
75   <action path="/home" type="org.apache.struts.actions.ForwardAction"
76     parameter="/Index.jsp"/>
77 </action-mappings>
78 <message-resources parameter="ApplicationResources"/>
79 <plug-in className="org.apache.struts.validator.ValidatorPlugIn">
80   <set-property property="pathnames"
81     value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
82 </plug-in>
83 </struts-config>
-----validation.xml-----
84 <?xml version="1.0" encoding="ISO-8859-1" ?>
85
86 <!DOCTYPE form-validation PUBLIC
87   "-//Apache Software Foundation//DTD Commons Validator Rules Configuration 1.0//EN"
88   "http://jakarta.apache.org/commons/dtds/validator_1_3.dtd">
89 <form-validation>
90   <formset>
91     <form name="/controller1">
92       <field property="id" depends="required, maxlength">
93         <arg position="0" key="bean.id"/>
94         <arg position="1" name="maxlength" key="${var:maxlength}" resource="false"/>
95         <var>
96           <var-name>maxlength</var-name>
97           <var-value>15</var-value>
98         </var>
99       </field>
100      <field property="pass" depends="required, minlength, maxlength, mask">
101        <arg position="0" key="bean.pass"/>
102        <arg position="1" name="minlength" key="${var:minlength}" resource="false"/>
103        <arg position="1" name="maxlength" key="${var:maxlength}" resource="false"/>
104
105        <var>
106          <var-name>minlength</var-name>
107          <var-value>5</var-value>
108        </var>
109
110        <var>
111          <var-name>maxlength</var-name>
112          <var-value>15</var-value>
113        </var>
114
115        <var>
```

```

116      <var-name>mask</var-name>
117      <var-value>^[0-9a-zA-Z]*$</var-value>
118      </var>
119      <msg name="mask" key="errors.invalid"/>
120
121      </field>
122      </form>
123
124      <form name="/controller2">
125          <field property="id" depends="required, maxlength">
126              <arg position="0" key="bean.id"/>
127
128              <arg position="1" name="maxlength" key="${var:maxlength}" resource="false"/>
129              <var>
130                  <var-name>maxlength</var-name>
131                  <var-value>15</var-value>
132              </var>
133          </field>
134      </form>
135  </formset>
136
137 </form-validation>
-----ApplicationResources.properties-----
138 errors.required={0} is required.<br>
139 errors.minLength={0} can not be less than {1} characters.<br>
140 errors.maxLength={0} can not be greater than {1} characters.<br>
141 errors.invalid={0} is invalid.It can take only alphanumerics.<br>
142
143
144 errors.byte={0} must be a byte.<br>
145 errors.short={0} must be a short.<br>
146 errors.integer={0} must be an integer.<br>
147 errors.long={0} must be a long.<br>
148 errors.float={0} must be a float.<br>
149 errors.double={0} must be a double.<br>
150
151 errors.date={0} is not a date.<br>
152 errors.range={0} is not in the range {1} through {2}.<br>
153 errors.creditcard={0} is an invalid credit card number.<br>
154 errors.email={0} is an invalid e-mail address.<br>
155
156 bean.id=<b><u>User Name</u></b>
157 bean.pass=<b><u>Password</u></b>
158
159
160 form.id=<b><u>User Name</u></b>
161 form.pass=<b><u>Password</u></b>
162 form1.pass=<b><u>New Password</u></b>
-----AddUser.jsp-----
163 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
164 <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
165
166
167 <html:html>
168     <html:form action="/controller1">
169         <center><br><br>
170             <table>
171                 <caption><font size = 6><u>Adding User</u></font></h2></u></caption>
172                 <tr>
173                     <td><bean:message key="form.id"/></td>
174                     <td><html:text property="id"/></td>
175                 </tr>
176             <tr>

```

## STRUTS

## DURGASOFT

```
177 <td><bean:message key="form.pass"/></td>
178 <td><html:password property="pass"/></td>
179 </tr>
180 </table>
181 <br>
182 <html:submit value="AddUser"/>
183
184 <html:hidden property="function" value="add"/>
185 </center>
186 </html:form>
187 </html:html>
188
189 -----ModifyUser.jsp-----
190 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
191 <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
192
193 <html:html>
194 <html:form action="/controller1">
195 <center><br><br>
196 <table>
197 <caption><font size = 6><u>Modifying User</font></h2></u></caption>
198
199 <tr>
200 <td><bean:message key="form.id"/></td>
201 <td><html:text property="id"/></td>
202 </tr>
203
204 <tr>
205 <td><bean:message key="form1.pass"/></td>
206 <td><html:password property="pass"/></td>
207 </tr>
208 </table>
209 <br><br>
210 <html:submit value="UpdateUser"/>
211 <html:hidden property="function" value="update"/>
212 </center>
213 </html:form>
214 </html:html>
215 -----DeleteUser.jsp-----
216 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
217 <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
218
219 <html:html>
220 <html:form action="/controller2">
221 <center><br><br>
222 <table>
223 <caption><fcnt size = 6><u>Deleting User</font></h2></u></caption>
224 <tr>
225 <td><bean:message key="form.id"/></td>
226 <td><html:text property="id"/></td>
227 </tr>
228 </table>
229 <br><br>
230 <html:submit value="DeleteUser"/>
231
232 <html:hidden property="function" value="delete"/>
233 </center>
234 </html:form>
235 </html:html>
236 -----OurAction.java-----
237 import java.io.*;
```

```
238 import org.apache.struts.action.*;
239 import org.apache.struts.actions.*;
240 import java.sql.*;
241 import javax.naming.*;
242 import javax.sql.*;
243 import javax.servlet.http.*;
244 import org.apache.struts.validator.*;
245
246 public class OurAction extends DispatchAction
247 {
248     private Connection getConnection()
249     {
250         Connection con = null;
251         try
252         {
253             Class.forName("oracle.jdbc.driver.OracleDriver");
254             con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl", "Scott", "Tiger");
255         } //try
256         catch(Exception e)
257         {
258             return con;
259         } //getConnection()
260
261     private void releaseResources(Connection con, PreparedStatement pst)
262     {
263         if(con != null)
264         {
265             try
266             {
267                 con.close();
268             }
269             catch(Exception e)
270             {}
271         } //if
272
273         if(pst != null)
274         {
275             try
276             {
277                 pst.close();
278             } //try
279             catch(Exception e)
280             {}
281         } //if
282     } //releaseResources(-,-)
283     public ActionForward add(ActionMapping mapping,
284                             ActionForm f,
285                             HttpServletRequest request,
286                             HttpServletResponse response)
287     {
288         Connection con = null;
289         PreparedStatement pst = null;
290         System.out.println("add () of OurAction");
291         try
292         {
293             DynaValidatorForm form=(DynaValidatorForm)f;
294             con = this.getConnection();
295
296             pst=con.prepareStatement("INSERT INTO STRUTS_DISP_ACTION_TEST VALUES(?, ?)");
297             pst.setString(1,(String)form.get("id"));


```

```
299     pst.setString(2,(String)form.get("pass"));
300
301     int i = pst.executeUpdate();
302
303     request.setAttribute("Action", "Insert ");
304
305     if(i != 0)
306     {
307         return mapping.findForward("success");
308     }
309     else
310     {
311         return mapping.findForward("failure");
312     }
313 } // try
314 catch(Exception e)
315 {
316     return mapping.findForward("failure");
317 }
318 finally
319 {
320     releaseResources(con,pst);
321 }
322 } // addUser()
323
324 public ActionForward delete(ActionMapping mapping,
325     ActionForm f,
326     HttpServletRequest request,
327     HttpServletResponse response)
328 {
329     Connection con = null;
330     PreparedStatement pst = null;
331     System.out.println("delete() of OurAction");
332     try
333     {
334         DynaValidatorForm form=(DynaValidatorForm)f;
335         con=getConnection();
336
337         pst=con.prepareStatement("DELETE FROM STRUTS_DISP_ACTION_TEST WHERE ID=?");
338         pst.setString(1,(String)form.get("id"));
339
340         int i=pst.executeUpdate();
341
342         request.setAttribute("Action", "Deletion");
343         if(i != 0)
344         {
345             return mapping.findForward("success");
346         }
347         else
348         {
349             return mapping.findForward("failure");
350         }
351     }//try
352     catch(Exception e)
353     {
354         return mapping.findForward("failure");
355     }
356     finally
357     {
358         releaseResources(con, pst);
359     }
```

## STRUTS

DURGASOFT

```
360 } // deleteUser()
361
362 public ActionForward update(ActionMapping mapping,
363     ActionForm f,
364     HttpServletRequest request,
365     HttpServletResponse response)
366 {
367     Connection con = null;
368     PreparedStatement pst = null;
369     System.out.println("update() of OurAction");
370
371     try
372     {
373         DynaValidatorForm form=(DynaValidatorForm)f;
374
375         con = this.getConnection();
376         pst=con.prepareStatement("UPDATE STRUTS_DISP_ACTION_TEST SET PASS=? WHERE
377 ID=?");
378         pst.setString(1,(String)form.get("pass"));
379         pst.setString(2,(String)form.get("id"));
380
381         int i = pst.executeUpdate();
382         request.setAttribute("Action","Updation");
383
384         if(i != 0)
385             return mapping.findForward("success");
386         else
387             return mapping.findForward("failure");
388     }
389     catch(Exception e)
390     {
391         return mapping.findForward("failure");
392     }
393     finally
394     {
395         releaseResources(con,pst);
396     }
397 } // updateUser()
398 } // OurAction class.
-----success.jsp-----
399 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
400
401 <html>
402 <body>
403 <div align="center">
404 <font color="#0000FF" size="5"><em><strong>
405 <%= (String)request.getAttribute("Action")%> Operation Successful <br>
406 and values are <%= (String)request.getParameter("id")%>
407 and <%= (String)request.getParameter("pass")%>
408 </strong></em></font>
409 </div>
410
411 <center>
412 <br><br>
413 <html:link action="/home">HOME</html:link><br>
414 </center>
415 </body>
416 </html>
417
418 -----failure.jsp-----
419 <html>
```

```
420 <body>
421   <p align="center"><strong><font size="5"><em>
422     Operation failed !!! </em></font></strong>
423   </p>
424 </body>
425 </html>
426 -----error.jsp-----
427 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
428
429 <html:html>
430   <center><br>
431   <html:errors />
432   <br><br>
433
434   <html:link action="/home">HOME</html:link><br>
435   </center>
436 </html:html>
437
```

## Advanced Action Classes

In this chapter, we dig further into the Controller components of the Struts framework by covering the built-in Action classes that come with Struts. Our goal is to provide you with a solid understanding of the Struts built in actions and how they can be used to facilitate the design of your struts applications.

The Struts Framework provides several built in actions. A few of these are essential for any Struts applications. Others are necessary to add cohesion to that would normally be a granular collection of related actions.

### ForwardAction and Beyond

Generally speaking it is a bad idea to place within you JSP pages any direct links to other JSP pages. For one reason it is not a good design decision; the struts-config.xml file, which is part of the Controller, should contain the entire flow of your application.

In the Model-View-Controller(MVC) architecture, it is the role of the Controller to select the next View. The Controller consists of Struts configuration, the ActionServlet, and the actions. If you add a link directly to another JSP, you are violating the architectural boundaries of the Model 2 architecture. (Model 2 is the instantiation of the MVC architecture for Web applications).

At times, however, all you really need is just a plain link; you don't want (or need) an action to execute first. Perhaps there are no objects from the domain that need to be mapped into scope in order for the view to display. Perhaps the page is very simple. In this case, a better approach is to use the Forward Action.

The ForwardAction acts as a bridge from the current View (JSP) and the pages it links to. It uses the RequestDispatcher to forward to a specified Web resource. It is the glue that allows you to link to an action instead of directly to a JSP.

Later, if you need to, you can change the ActionMapping in the Struts configuration file so that every page that linked to that action will link to the new action. Also, you can change the action to a custom one that you write instead of using the ForwardAction that Struts provides.

### To use the ForwardAction, follow these steps:

1. Using the html:link tag with the action attribute, add a link to the JSP page that points to the action.
2. Create an action mapping in the Struts configuration file that uses the ForwardAction with the parameter attribute to specify the JSP path.

Let's say you have a JSP page that has a direct link to another JSP page:

```
<html:link page="/index.jsp">Home</html:link>
```

You have recently converted to the MVC/Model 2 architecture religion and you want to change the html:link tag to link to an action. Because you already have a link, simply changes it as follows:

```
<html:link action="home">Home</html:link>
```

All you do is remove the page attribute and add an action attribute that points to the home action. Now you have to create the home action mapping. (The link in the previous code snippet would be expanded to a URL, link <http://localhost:8080/actions/home.do>.)

To add an action mapping to the home action that you referenced in your html:link tag, use this code:

```
<action  
    Path="/home"  
    Type="org.apache.struts.actions.ForwardAction"
```

```
Parameter="/index.jsp"  
/>>
```

The `ForwardAction` uses the `parameter` attribute so that you can specify the path. Notice the parameter is set to `/index.jsp`, which was what the `page` attribute of the `html:link` tag was originally set to. Thus, the `parameter` attribute indicated to where you what the `ForwardAction` to forward.

### **Linking to JSP Directly: More than Just Bad Design**

In addition to being a bad design decision linking directly to JSP pages may result in errors. The Controller maps in the correct `ModuleConfig` based On the request. Having the correct module implies that the resource bundlers for that module are loaded. If the JSP page uses the resource bundles to display massages (which is required for internationalized applications), then you need to link to the action directly because that is the only way the JSP page is guaranteed to work properly.

### **Linking to Global Forwards Instead of Actions**

From a purely design perspective, there are other alternatives to using the `ForwardAction`. Rather then linking to an action or a page, you could link to a global forward:

```
<html:link forward ="home">Home</html:link>
```

Thus, the best approach for our example is to add this in the JSP:

```
<html:link action="home">Home</html:link>
```

Then add this in the global forwards section of the Struts configuration file:

```
<forward name="home" path="/home.do"/>
```

And this the action mappings section of the Struts configuration file:

```
<action path="/home" forward="/index.jsp"/>
```

Concise, straightforward, and well designed. Exactly what we needed!

### **Don't Bleed Your V into Your C**

If you find that you have a lot of links to JSP pages from other JSP pages, you may not understand MVC very well. Using `ActionForward` may mask the fact that your design is inherently messed up. JSPs linking to other JSPs, whether they use `ForwardAction` or not, works only for the simplest of Web applications in MVC. If it works for your application, you are probably doing something wrong.

Complex MVC Web applications typically need an action to execute first; it is the job of the action to map Model items into scope so that the View (typically JSP) can display them. If this is not the case for your nontrivial Web application, yhou are probably putting too much logic in the JSP pages (or in custom tags that JSP pages use) - which canbe the source of huge maintenance issues.

The JSP pages and custom tags should contain only display logic, and this logic should be kept to a bare minimum. Thus, the JSP pages and custom tags should bot talk directly to the Model-they only display different pices of the Model. In essence, they should speak only to the Value object and Data Transfer objects. Again the action talks directly to the Model and delegates the display of the Model objects to the View. The Model in turn implements the persistence and business rules for the application.

Adopting a strict aMVC architecture is generally a good idea; it keeps your JSP smaller and more focused. JSP are harder to test than actions, so adopting MVC increases the Liquidity and flexibility of your code base.

## STRUTS

## DURGASOFT

In short, if you overuse the `ForwardAction`, you need to evaluate your understanding of the MVC architrave.

### Forwards for Legacy Access

A lot of folks wrote Web applications before Struts existed. A lot of other folks have had to integrate with commercial portal implementations and other frameworks that are built on top of servlets (legacy and otherwise) using the `ForwardAction` (or the `forward` attribute) is a good way to encapsulate this legacy integration of your View components (JSP and custom tags). In this manner, the `ForwardAction` acts as "legacy glue" to other Web resource.

Let's say that you have a legacy Web resource that you want to use with the form validation of Struts. However, your legacy resources are part of an elaborate MVC based Web application framework that you created before the dawn of the Struts Framework.

This legacy resource does all types of neat things that you have not ported to the Struts Framework yet. For some reason (time and money come to mind), you do not want to rewrite this legacy resource—at least not yet.

For the purpose of simplicity, our sample legacy Web resource will be a servlets. Essentially, you want the servlets do Get method to be called only if the `ActionForm` validate. Here is our example servlets:

```
public class LegacyServlet extends HttpServlet
{
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException
    {
        //Get the mapping
        ActionMapping
        mapping = (ActionMapping) request.getAttribute(Globals.MAPPING_KEY);
        //Get the UserForm
        UserForm form = (UserForm) request.getAttribute(mapping.getName());
        //Do some fly, super tricky, whiz
        //bang legacy stuff
        //Generate some output
        response.getWriter().println("User name" + form.getFirstName());
    }
}
```

Notice how the servlet can access the context that the Struts Framework mapped into request scope. Now suppose this servlet was mapped into our Web application like so:

```
<servlet>
    <servlet-name>legacy</servlet-name>
    <servlet-class>legacy.LegacyServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>legacy</servlet-name>
        <url-pattern>legacy/roar</url-pattern>
    </servlet-mapping>
```

Thus, posts to `/legacy/roar` would cause this servlet's `doPost()` method to be called. Now, to map this servlet as an action that acts as a form handler, you need to do this:

```
<action
    path="/register"
    forward="/legacy/roar"
    input="/form/userForm.jsp"
    name="userForm"
```

```
parameter="/legacy/roar"
validate="true"
scope="request"
/>>
```

Of course, the previous code assumes that you have a from bean:

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

Your input JSP(/from/userForm.jsp) would look like this:

```
.....
<h1>Legacy: Struts Form for userFroms </h1>
<html:form action="/legacy">
  Email :
    <html:text property="email" /><br>

  First Name :
    <html:text property="firstName" /><br>
  Last Name :
    <html:text property="lastName" /><br>
  Password :
    <html:password property="password" /><br>
  Password Check:
    <html:password property="passwordCheck"/><br>
  UserName :
    <html:text property="userName" /><br>
  <html:submit/> <html:cencel/>
</html:form>
```

Notice that the `html:form` tag points to an action path, just as you expect. Because the `RequestDespactcher` forward method to the Servlet is invoked only if the `ForwardAction`'s `execute` method is invoked, the `depots` method of the legacy servlet is called only if the `ForwardAction`'s `execute` method is called, we set the `validate` attribute to true in the action mapping of this servlet, so the `execute` method of the `ForwardAction` is called only if the `ActionForm` (`UserForm`) validates (returns no `ActionError` objects).

**InculcAction:**

To understand the `InculcAction`, be sure to read about the `ForwardAction` in the previous section. The `InculcAction` is similar to the `ForwardAction` but is not used as much. You could rewrite the last example using the `IriculeAction` as follows:

```
<action
  path="/register"
  type="org.apache.struts.actions.IncludeAction"
  input="/form/userForm2.jsp"
  name="userForm"
  parameter="/legacy/roar"
  validate="true"
  scope="request"
/>>
```

This shorter form uses the include attribute:

```
<action
    path="/register"
    include="/legacy/roar"
    input="/form/userForm2.jsp"
    name="userForm"
    parameter="/legacy/roar"
    validate="true"
    scope="request"
/>
```

So what is the difference between the includeAction and the ForwardAction? The difference is that you need to use the includeAction only if the action is going to be included by another action or JSP.

Therefore, if you have code in your JSP that looks like this:

```
<jsp:include page="/someWebApp/someModule/someAction.do"/>
```

The action could not use a ForwardAction because it would forward control to the new rather than including its output within the output of the JSP -or throw a nasty IllegalStateException if the output buffer was already committed.

This discussion also applies to using the Tiles framework. A Tile can be Web resource, and an action can be a Web resource. This , you could define a Tile as an action. Refer to Chapters 13 and 17 for more details.

### **DispatchAction**

Oftentimes actions seem to be too plentiful and too small. It would be nice to group related actions into one class.

For example, let's say that a group of related actions all work on the same set of objects in the user session (HttpSession)- a shopping cart, for example. Another example is a group of actions that are all involved in the same use case. Yet another example is a group of actions that are all involved in CRUD operations on domain objects. (CRUD stands for create, read, update, and delete. Think of an add/edit/delete/listing of products for an online e-commerce store)

If you can group related actions into one class, you can create helper methods that they all use, thus improving reuse (or at least facilitating it). Also if these helper methods are only used by these related actions and these actions are in the same class then the helper methods can be encapsulated (hidden) inside this one class.

The DispatchAction class is used to group related actions into one class. DispatchAction is an abstract class, so you must override it to use it. It extends the Action class.

Rather than having a single execute method, you have a method for each logical action. The DispatchAction dispatches to one of the logical actions represented by the methods. It picks a method to invoke based on an incoming request parameter. The value of the incoming request parameter is the name of the method that the DispatchAction will invoke.

To use the DispatchAction, follow these steps:

1. Create an action handler class that subclasses DispatchAction.
2. Create a method to represent each logical related Action.
3. Create an action mapping for this action handler using the parameter attribute to specify the request parameter that carries the name of the method you want to invoke.
4. Pass the action a request parameter that refers to the method you want to invoke.

## STRUTS

DURGASOFT

First, you create an action handler class that subclasses DispatchAction:

```
public class UserDispatchAction extends DispatchAction {  
    ....  
}
```

Then, you create a method to represent each logical related action:

```
public class UserDispatchAction extends DispatchAction {  
    ....  
    public ActionForward remove(  
        ActionMapping mapping,  
        ActionForm form,  
        HttpServletRequest request,  
        HttpServletResponse response)  
        throws Exception  
    {  
        System.out.println("Remove User");  
        return mapping.findForward("success");  
    }  
  
    public ActionForward save(  
        ActionMapping mapping,  
        ActionForm form,  
        HttpServletRequest request,  
        HttpServletResponse response)  
        throws Exception  
    {  
        System.out.println("Save User");  
        return mapping.findForward("success");  
    }  
}
```

Notice these methods have the same signature (other than the method name) of the standard Action.execute method.

```
    ActionMapping mapping,  
    ActionForm form,  
    HttpServletRequest request,  
    HttpServletResponse response)  
    throws Exception  
{  
    System.out.println("SAVE USER (LOOKUP)");  
    return mapping.findForward("success");  
}  
.....  
}
```

Notice these methods have the same signature of the standard Action.execute() method (except for the method name).

Third, you must implement the getKeyMethodMap() method to map the resource keys to method names:

```
<action  
    path="/lookupDispatchUserSubmit"  
    type="action.UserLookupDispatchAction"  
    input="/form/userForm.jsp"  
    name="userForm"  
    parameter="method"
```

```
validate="true"
scope="request"
<forward name="success" path="/success.jsp"/>
</action>
```

The fifth step is to set up the messages in the resource bundle for the labels and values of the buttons. Inside your bundle (e.g . application.properties), add the following two entries:

```
userForm.save=save
userForm.remove=remove
```

Finally, use bean:message to display the labels of the button in the bean:

```
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean"%>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html"%>
...
<html:link action="home">Home</html:link>
<html:form action="/lookupDispatchUserSubmit">
...
<html:submit property="method">
<bean:message key="userForm.remove"/>
</html:submit>
<html:submit property="method">
<bean:message key="userForm.save"/>
</html:submit>
<html:cancel>
</html:cancel>
</html:form>
<body>
</html>
```

When the user clicks the Remove button on the HTML form, the remove method is called on the action.

When the user clicks the Save button, the save method is called on the action.

### **SwitchAction**

The SwitchAction class is used to support switching from module to module. Let's say you have an action that wants to forward to an action in another module. You perform the following steps:

1. Map in a SwitchAction into your Struts Configuration file.
2. Create a forward or link to the SwitchAction that passes the page parameter and the module prefix parameter.

Let's break this down. Say you have a Web application that uses Struts. Struts has two modules: the default and a module called admin, as follows:

```
<servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
  <init-param>
    <param-name>config</param-name>
    <param-value>/WEB-INF/struts-config.xml</param-value>
  </init-param>
  <init-param>
    <param-name>config/admin</param-name>
```

## STRUTS

DURGASOFT

```
<param-value>/WEB-INF/struts-config-admin.xml</param-value>
</init-param>
<init-param>
    <param-name>debug</param-name>
    <param-value>3</param-value>
</init-param>
<init-param>
    <param-name>detail</param-name>
    <param-value>3</param-value>
</init-param>
<load-on-startup>1</load-on-startup>
</servlet>
```

The init parameter config/admin defines the admin module. The config init parameter defines the default module.

Now let's say you have an action in the default module that edits users and you want to delegate the display of those users to an action in the admin module. First, map a SwitchAction into the default module as shown here:

```
<action
    path="/switch"
    type="org.apache.struts.actions.SwitchAction">
</action>
```

Now you can set up a forward in the action that edits the users as follows:

```
<action
    path="/userSubmit"
    attribute="userform"
    input="/form/userForm.jsp"
    name="userform"
    scope="request"
    type="action.UserAction">
    <forward name="success"
        path="/switch.do?page=/listUsers.do&prefix=/admin"/>
</action>
```

Notice that this forward passes two request parameters. The page parameter specifies the module relative action. The second parameter specifies the prefix of the module—in this case admin. You don't have to use forwards to use the SwitchAction; any JSP can link to the SwitchAction to move to any module. The listUser.do action is not defined in the default module; it is defined in the admin. The forward in this example forwards to the action at the path /admin/listUsers.do. The listUser.do is defined in /WEB-INF/struts-config-admin.xml, and the userSubmit.do action is defined in /WEB-INF/struts-config.xml.

```

1 App7(Application on LookupDispatchAction)
2 =====
3 -----User.jsp-----
4 <%@ taglib uri="http://jakarta.apache.org/struts/tags-bean" prefix="bean"%>
5 <%@ taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html"%>
6
7 <html>
8   <head>
9     <title>JSP for EmpForm form</title>
10    </head>
11    <body>
12      <html:form action="/emp">
13        userid : <html:text property="userid"/><br/>
14        firstname : <html:text property="firstname"/><br/>
15        lastname : <html:text property="lastname"/><br/>
16        address : <html:text property="address"/><br/>
17        <html:submit property="function">
18          <bean:message key="btn.cap1"/>
19        </html:submit>
20        <html:submit property="function">
21          <bean:message key="btn.cap2"/>
22        </html:submit>
23        <html:submit property="function">
24          <bean:message key="btn.cap3"/>
25        </html:submit>
26        <html:cancel/>
27      </html:form>
28      <%if (request.getAttribute("operation")!=null)
29      {
30      %>
31
32      <%=request.getAttribute("operation") %> is success.
33      <%} %>
34    </body>
35  </html>
36 -----ApplicationResources.properties-----
37 btn.cap1=insert
38 btn.cap2=Modify
39 btn.cap3=Remove
40 -----web.xml-----
41 <?xml version="1.0" encoding="UTF-8"?>
42 <web-app xmlns="http://java.sun.com/xml/ns/j2ee"
43   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
44   version="2.4" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
45   http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
46   <servlet>
47     <servlet-name>action</servlet-name>
48     <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
49     <init-param>
50       <param-name>config</param-name>
51       <param-value>/WEB-INF/struts-config.xml</param-value>
52     </init-param>
53     <load-on-startup>0</load-on-startup>
54   </servlet>
55
56   <servlet-mapping>
57     <servlet-name>action</servlet-name>
58     <url-pattern>*.do</url-pattern>
59   </servlet-mapping>

```

## STRUTS

DURGASOFT

```
60      <welcome-file>User.jsp</welcome-file>
61      </welcome-file-list>
62  </web-app>
63  -----struts-config.xml-----
64 <?xml version="1.0" encoding="UTF-8"?>
65 <!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration
1.2//EN"
66 "http://struts.apache.org/dtds/struts-config_1_2.dtd">
67
68 <struts-config>
69   <form-beans >
70     <form-bean name="ef" type="EmpForm" />
71   </form-beans>
72   <action-mappings >
73     <action attribute="ef" input="/User.jsp" name="ef" parameter="function" path="/emp"
type="EmpAction">
74       <forward name="success" path="/User.jsp" />
75       <forward name="failure" path="/failure.jsp" />
76     </action>
77   </action-mappings>
78
79   <message-resources parameter="com.durga.struts.ApplicationResources" />
80
81 </struts-config>
82 -----EmpForm.java-----
83 import org.apache.struts.action.ActionForm;
84
85 public class EmpForm extends ActionForm
86 {
87 private String lastname;
88 private String userid;
89 private String address;
90 private String firstname;
91
92 public String getLastname()
93 {
94 System.out.println("getLastname: EmpForm");
95 return lastname;
96 }
97
98 public void setLastname(String lastname)
99 {
100 System.out.println("setLastname: EmpForm");
101 this.lastname = lastname;
102 }
103 public String getUserid()
104 {
105 System.out.println("getUserid: EmpForm");
106 return userid;
107 }
108 public void setUserid(String userid)
109 {
110 System.out.println("setUserid: EmpForm");
111 this.userid = userid;
112 }
113 public String getAddress()
114 {
115 System.out.println("getAddress: EmpForm");
116 return address;
117 }
118 public void setAddress(String address)
```

```

119 {
120 System.out.println("setAddress: EmpForm");
121 this.address = address;
122 }
123
124 public String getFirstname()
125 {
126 System.out.println("getFirstname: EmpForm");
127 return firstname;
128 }
129
130 public void setFirstname(String firstname)
131 {
132 System.out.println("setFirstname: EmpForm");
133 this.firstname = firstname;
134 }
135 }//class
136 -----EmpAction.java-----
137 import java.sql.Connection;
138 import java.sql.PreparedStatement;
139 import java.util.HashMap;
140 import java.util.Map;
141
142 import javax.servlet.http.HttpServletRequest;
143 import javax.servlet.http.HttpServletResponse;
144
145 import org.apache.struts.action.ActionForm;
146 import org.apache.struts.action.ActionForward;
147 import org.apache.struts.action.ActionMapping;
148 import org.apache.struts.actions.LookupDispatchAction;
149
150 import com.durga.connmanager.DBConnection;
151
152 public class EmpAction extends LookupDispatchAction
153 {
154 protected Map getKeyMethodMap()
155 {
156 System.out.println("getKeyMethodMap(): EmpAction");
157 Map map=new HashMap();
158 map.put("btn.cap1","add");
159 map.put("btn.cap2","update");
160 map.put("btn.cap3","delete");
161 return map;
162 }
163
164 public ActionForward add(ActionMapping mapping,
165 ActionForm form,
166 HttpServletRequest request,
167 HttpServletResponse response)
168 {
169
170 String output=null;
171 Connection con=null;
172 PreparedStatement ps=null;
173 DBConnection dbcon=null;
174
175 try
176 {
177 System.out.println("add(): EmpAction");
178 EmpForm ef = (EmpForm) form;
179 dbcon=new DBConnection();

```

```

180
181     con=dbcon.getConnection();
182     ps=con.prepareStatement("insert into employee_info values(?,?,?,?,?)");
183
184     ps.setString(1,ef.getUserid());
185     ps.setString(2, ef.getFirstname());
186     ps.setString(3, ef.getLastname());
187     ps.setString(4, ef.getAddress());
188
189     int result=ps.executeUpdate();
190
191
192     if(result!=0)
193         output="success";
194     else
195         output="failure";
196
197     request.setAttribute("operation","Insert operation ");
198 //try
199     catch(Exception e)
200     {
201         e.printStackTrace();
202     }
203     finally
204     {
205         dbcon.releaseResouc(ps,con);
206     }
207     return mapping.findForward(output);
208 } //add
209
210
211     public ActionForward update(ActionMapping mapping,
212                               ActionForm form,
213                               HttpServletRequest request,
214                               HttpServletResponse response)
215     {
216
217     String output=null;
218     Connection con=null;
219     PreparedStatement ps=null;
220     DBConnection dbcon=null;
221     try
222     {
223         System.out.println("update(): EmpAction");
224         EmpForm ef = (EmpForm) form;
225         dbcon=new DBConnection();
226
227
228         con=dbcon.getConnection();
229         ps=con.prepareStatement("update employee_info set firstname=?,lastname=?,address=?"
230         "where empid=?");
231
232         ps.setString(1,ef.getFirstname());
233         ps.setString(2, ef.getLastname());
234         ps.setString(3, ef.getAddress());
235         ps.setString(4, ef.getUserid());
236
237         int result=ps.executeUpdate();
238
239         if(result!=0)

```

```
240     output="success";
241 else
242     output="failure";
243
244     request.setAttribute("operation","Update operation ");
245 }//try
246 catch(Exception e)
247 {
248     e.printStackTrace();
249 }
250 finally
251 {
252     dbcon.releaseResouc(ps,con);
253 }
254 return mapping.findForward(output);
255 }//update
256
257 public ActionForward delete(ActionMapping mapping,
258                 ActionForm form,
259                 HttpServletRequest request,
260                 HttpServletResponse response)
261 {
262
263     String output=null;
264     Connection con=null;
265     PreparedStatement ps=null;
266     DBCConnection dbcon=null;
267
268     try
269     {
270         System.out.println("delete(): EmpAction");
271         EmpForm ef = (EmpForm) form;
272         dbcon=new DBCConnection();
273
274         con=dbcon.getConnection();
275         ps=con.prepareStatement("delete from employee_info where empid=?");
276         ps.setString(1,ef.getUserid());
277
278         int result=ps.executeUpdate();
279
280         if(result!=0)
281             output="success";
282         else
283             output="failure";
284
285         request.setAttribute("operation","Delete operation ");
286     } //try
287     catch(Exception e)
288     {
289         e.printStackTrace();
290     }
291 finally
292 {
293     dbcon.releaseResouc(ps,con);
294 }
295 return mapping.findForward(output);
296 } //delete
297
298 } //class
299 -----DBConnection.java-----
300 package com.durga.conmmanager;
```

## STRUTS

## DURGASOFT

```
301 import java.sql.Connection;
302 import java.sql.DriverManager;
303 import java.sql.PreparedStatement;
304
305 public class DBConnection
306 {
307
308     public Connection getConnection()
309     {
310         try
311         {
312             System.out.println("getConnection(): DBConnection");
313             Class.forName("oracle.jdbc.driver.OracleDriver");
314             Connection
315             con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
316             return con;
317         }
318         catch(Exception e)
319         {
320             return null;
321         }
322
323     } //method
324
325
326     public void releaseResouc(PreparedStatement ps,Connection con)
327     {
328         System.out.println("releaseResouc(): DBConnection");
329         try
330         {
331             ps.close();
332         }
333         catch(Exception e)
334         {
335             e.printStackTrace();
336         }
337
338         try
339         {
340             con.close();
341         }
342         catch(Exception e)
343         {
344             e.printStackTrace();
345         }
346
347     } //method
348
349 } //class
350 -----failure.jsp-----
351 <%=request.getAttribute("operation") %> is failed
352
```

## SWITCH ACTION

### 5.3 Configuring Your Application for Modules

Very little is required in order to start taking advantage of the module feature. Just go through the following steps:

1. Prepare a configuration file for each module.
2. Inform the controller of your module.
3. Use Actions to refer to your pages.

#### 5.3.1 Module Configuration Files

Back in version 1.0, a few "boot-strap" options were placed in the web.xml file, and the bulk of the configuration was done in a single struts-config.xml file. Obviously, this wasn't ideal for a team environment, since multiple users had to share the same configuration file.

Since version 1.1, you have two options: you can list multiple struts-config files as a comma-delimited list, or you can subdivide a larger application into modules.

With the advent of modules, a given module has its own configuration file. This means each team (each module would presumably be developed by a single team) has their own configuration file, and there should be a lot less contention when trying to modify it.

#### 5.3.2 Informing the Controller

Since version 1.0, you listed your configuration file as an initialization parameter to the action servlet in web.xml. This is still done since version 1.1, but the parameter can be extended. In order to tell the framework machinery about your different modules, you specify multiple 'config' initialization parameters, with a slight twist. You'll still use 'config' to tell the ActionServlet about your "default" module, however, for each additional module, you will list an initialization parameter named "config /module ", where /module is the prefix for your module (this gets used when determining which URIs fall under a given module, so choose something meaningful!).

For example:

```
<init-param>
    <param-name>config</param-name>
    <param-value>/WEB-INF/conf/struts-default.xml</param-value>
</init-param>
<init-param>
    <param-name>config/module1</param-name>
    <param-value>/WEB-INF/conf/struts-module1.xml</param-value>
</init-param>
...

```

Here we have two modules. One happens to be the "default" module, identified by the param-name of "config", and the other will be using the module prefix "/module1" based on the param-name it was given ("config/module1"). The controller is configured to find the respective configuration files under /WEB-INF/conf/ (which is the recommended place to put all configuration files). Pretty simple!

(The struts-default.xml would be equivalent to what most folks call struts-config.xml. I just like the symmetry of having all my module configuration files being named struts-*module.xml* )

If you'd like to vary where the pages for each module are stored, see the forwardPattern setting for the Controller.

### **5.3.3 Switching Modules**

There are three approaches for switching from one module to another.

1. You can use the org.apache.struts.actions.SwitchAction from the Extras JAR,
2. you can use a <forward> (global or local) and specify the contextRelative attribute with a value of true,
3. or you can specify the "module" parameter as part of any of the Struts JSP hyperlink tags (Include, Img, Link, Rewrite, or Forward).

You can use org.apache.struts.actions.SwitchAction like so:

```
...
<action-mappings>
<action path="/toModule"
type="org.apache.struts.actions.SwitchAction"/>
...
</action-mappings>
...
```

Now, to change to ModuleB, we would use a URI like this:

<http://localhost:8080/toModule.do?prefix=/moduleB&page=/index.do>

If you are using the "default" module as well as "named" modules (like "/moduleB"), you can switch back to the "default" module with a URI like this:

<http://localhost:8080/toModule.do?prefix=&page=/index.do>

Here's an example of a global forward:

```
<global-forwards>
  ...
    <forward name="toModuleB"
      contextRelative="true"
      path="/moduleB/index.do"
      redirect="true"/>
  ...
</global-forwards>
```

\* You could do the same thing with a local forward declared in an ActionMapping:

```
<action-mappings>
<action ... >
  <forward name="success"
```

```
    contextRelative="true"
        path="/moduleB/index.do"
        redirect="true"/>
</action>
...
</action-mappings>
```

Using the module parameter with a hyperlink tag is even simpler:

```
<html:link module="/moduleB" path="/index.do"/>
```

That's all there is to it! Happy module-switching!

#### 5.4 The Web Application Deployment Descriptor

The final step in setting up the application is to configure the application deployment descriptor (stored in file WEB-INF/web.xml ) to include all the framework or Taglib components that are required. Using the deployment descriptor for the example application as a guide, we see that the following entries need to be created or modified.

##### 5.4.1 Configure the ActionServlet Instance

Add an entry defining the action servlet itself, along with the appropriate initialization parameters. Such an entry might look like this:

```
<servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>
        org.apache.struts.action.ActionServlet
    </servlet-class>
    <init-param>
        <param-name>config</param-name>
        <param-value>
            /WEB-INF/struts-config.xml
        </param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
```

The initialization parameters supported by the action servlet are described below. (You can also find these details in the [Javadocs](#) for the ActionServlet class.) Square brackets describe the default values that are assumed if you do not provide a value for that initialization parameter.

- **chainConfig** - Comma-separated list of either context-relative or classloader path(s) to load commons-chain catalog definitions from. If none specified, the default catalog that is provided with the framework will be used. (Since version 1.3)
- **config** - Context-relative path to the XML resource containing the configuration information for the default module. This may also be a comma-delimited list of configuration files. Each file is loaded in turn, and its objects are appended to the internal data structure. [/WEB-INF/struts-config.xml].  
**WARNING** - If you define an object of the same name in more than one configuration file, the last one loaded quietly wins.  
**WARNING** - Plugins are not designed to be loaded more than once in the same module. The set of configuration files for a module should load a plugin only once.
- **config/\${module}** - Context-relative path to the XML resource containing the configuration information for the application module that will use the specified prefix (/\${module}). This can be repeated as many times as required for multiple application modules. (Since version 1.1)
- **configFactory** - The Java class name of the ModuleConfigFactory used to create the implementation of the ModuleConfig interface. (Since version 1.3) [org.apache.struts.config.impl.DefaultModuleConfigFactory]
- **convertNull** - Force simulation of the version 1.0 behavior when populating forms. If set to "true", the numeric Java wrapper class types (like java.lang.Integer ) will default to null (rather than 0). (Since version 1.1) [false]
- **rulesets** - Comma-delimited list of fully qualified classnames of additional org.apache.commons.digester.RuleSet instances that should be added to the Digester that will be processing struts-config.xml files. By default, only the RuleSet for the standard configuration elements is loaded. (Since version 1.1)
- **validating** - Should we use a validating XML parser to process the configuration file (strongly recommended)? [true]

**WARNING** - The framework will not operate correctly if you define more than one <servlet> element for a controller servlet, or a subclass of the standard controller servlet class. The controller servlet **MUST** be a web application wide singleton.

## STRUTS

DURGASOFT

```
1 App8(SwitchAction class based Application having multiple modules)
2 =====
3 -----index.jsp-----
4 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
5
6 <html:html>
7   <html:link action="/switch?prefix=/mod1&page=/Student.jsp">Student Search</html:link><br>
8
9   <html:link action="/switch?prefix=/Faculty&page=/Faculty.jsp">Faculty
10  Search</html:link><br>
11 </html:html>
12 -----web.xml-----
13 <?xml version="1.0" encoding="UTF-8"?>
14 <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc./DTD Web Application 2.3//EN"
15   "http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">
16 <web-app>
17   <servlet>
18     <servlet-name>action</servlet-name>
19     <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
20     <init-param>
21       <param-name>config</param-name>
22       <param-value>/WEB-INF/struts-config.xml</param-value>
23     </init-param>
24     <init-param>
25       <param-name>config/mod1</param-name>
26       <param-value>/WEB-INF/struts-config-student.xml</param-value>
27     </init-param>
28     <init-param>
29       <param-name>config/Faculty</param-name>
30       <param-value>/WEB-INF/struts-config-faculty.xml</param-value>
31     </init-param>
32     <load-on-startup>1</load-on-startup>
33   </servlet>
34
35   <servlet-mapping>
36     <servlet-name>action</servlet-name>
37     <url-pattern>*.do</url-pattern>
38   </servlet-mapping>
39
40   <welcome-file-list>
41     <welcome-file>index.jsp</welcome-file>
42   </welcome-file-list>
43 </web-app>
44 -----struts-config.xml-----
45 <!DOCTYPE struts-config PUBLIC
46   "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
47   "http://struts.apache.org/dtds/struts-config_1_3.dtd">
48
49 <struts-config>
50   <action-mappings>
51     <action path="/switch" scope="request" type="org.apache.struts.actions.SwitchAction"/>
52   </action-mappings>
53 </struts-config>
54 -----struts-config-faculty.xml-----
55 <!DOCTYPE struts-config PUBLIC
56   "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
57   "http://struts.apache.org/dtds/struts-config_1_3.dtd">
58 <struts-config>
59   <form-beans>
60     <form-bean name="FacultyForm" type="FacultyFormBean"/>
61   </form-beans>
```

## STRUTS

DURGASOFT

```
61 <action-mappings>
62   <action name="FacultyForm" path="/faculty" scope="request" type="FacultyAction">
63     <forward name="succ" path="/Faculty.jsp"/>
64     <forward name="fail" path="/Fail1.jsp"/>
65   </action>
66 </action-mappings>
67 </struts-config>
68 -----struts-config-student.xml-----
69 <!DOCTYPE struts-config PUBLIC
70   "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
71   "http://struts.apache.org/dtds/struts-config_1_3.dtd">
72 <struts-config>
73 <form-beans>
74   <form-bean name="studentform" type="StudentFormBean"/>
75 </form-beans>
76 <action-mappings>
77   <action name="studentform" path="/student" type="StudentAction">
78     <forward name="succ" path="/Student.jsp"/>
79     <forward name="fail" path="/fail.jsp"/>
80   </action>
81 </action-mappings>
82
83 </struts-config>
84 -----Student.jsp-----
85 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
86 <%@ taglib uri="http://struts.apache.org/tags-logic" prefix="logic" %>
87 <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
88
89 <center><font size=6>Search Criteria of Student</font></center>
90 <html:form action="student">
91   <table>
92     <tr>
93       <td>StudentID</td>
94       <td><html:text property="id" /></td>
95       <td><html:submit value="Send" /></td>
96     </tr>
97   </table>
98 </html:form>
99
100 <table border=1>
101 <%
102   ArrayList al=(ArrayList)request.getAttribute("res");
103   %>
104
105 <logic:notEmpty name="res" scope="session">
106   <tr>
107     <td>ID</td>
108     <td>Name</td>
109     <td>Course</td>
110     <td>Qualifacation</td>
111   </tr>
112
113 <logic:iterate id="id1" collection="<%="al%>">
114   <tr>
115     <td><bean:write name="id1" property="id" /></td>
116     <td><bean:write name="id1" property="name" /></td>
117     <td><bean:write name="id1" property="course" /></td>
118     <td><bean:write name="id1" property="qualification" /></td>
119   </tr>
120
121 </logic:iterate>
```

```

122      </logic:notEmpty>
123      </table>
124
125 -----Faculty.jsp-----
126      <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
127      <%@ taglib uri="http://struts.apache.org/tags-logic" prefix="logic" %>
128      <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
129
130      <center><font size=6>Search Criteria of Faculty</font></center>
131      <html:form action="faculty">
132          <table>
133              <tr>
134                  <td>FacultyID</td>
135                  <td><html:text property="id" /></td>
136                  <td><html:submit value="Send" /></td>
137              </tr>
138          </table>
139      </html:form>
140
141      <table border=1>
142      <%
143          ArrayList al=(ArrayList)request.getAttribute("resf");
144      %>
145
146      <logic:notEmpty name="resf" scope="session">
147          <tr>
148              <td>ID</td>
149              <td>Name</td>
150              <td>Subject</td>
151              <td>Qualifacation</td>
152          </tr>
153
154      <logic:iterate id="id1" collection="<%="al%>">
155          <tr>
156              <td> <bean:write name="id1" property="id" /></td>
157              <td> <bean:write name="id1" property="name" /></td>
158              <td><bean:write name="id1" property="subject" /></td>
159              <td><bean:write name="id1" property="qualification" /></td>
160          </tr>
161      </logic:iterate>
162
163      </logic:notEmpty>
164      </table>
165 -----StudentFormBean.java-----
166 import javax.servlet.http.HttpServletRequest;
167 import org.apache.struts.action.ActionError;
168 import org.apache.struts.action.ActionErrors;
169 import org.apache.struts.action.ActionForm;
170 import org.apache.struts.action.ActionMapping;
171
172 public class StudentFormBean extends org.apache.struts.action.ActionForm
173 {
174
175     public StudentFormBean()
176     {
177         System.out.println("StudentFormBean().....constructor is called");
178     }
179     private String id;
180     public void setid(String id)
181     {
182         System.out.println("setid() :StudentFormBean");

```

```
183     this.id=id;
184 }
185 public String getid()
186 {
187     System.out.println("getid() :StudentFormBean");
188     return id;
189 }
190 }
191 }-----FacultyFormBean.java-----
192 import javax.servlet.http.HttpServletRequest;
193 import org.apache.struts.action.ActionError;
194 import org.apache.struts.action.ActionErrors;
195 import org.apache.struts.action.ActionForm;
196 import org.apache.struts.action.ActionMapping;
197
198 public class FacultyFormBean extends org.apache.struts.action.ActionForm
199 {
200     private String id;
201     public FacultyFormBean()
202     {
203         System.out.println("FacultyFormBean().....constructor is called");
204     }
205
206     public void setid(String id)
207     {
208         System.out.println("setid() :FacultyFormBean");
209         this.id=id;
210     }
211     public String getid()
212     {
213         System.out.println("getid() :FacultyFormBean");
214         return id;
215     }
216 }
217 }-----Student.java-----
218 package bean;
219 public class Student
220 {
221     String id,name,course,qualification;
222     public Student()
223     {
224         System.out.println("student().....constructor is called");
225     }
226     public void setid(String id)
227     {
228         this.id=id;
229         System.out.println("setId() :Student");
230     }
231     public String getid()
232     {
233         System.out.println("getId() :Student");
234         return this.id;
235     }
236     public void setname(String name)
237     {
238         System.out.println("setname() :Student");
239         this.name=name;
240     }
241     public String getname()
242     {
```

## STRUCTS

DURGASOFT

```
244     System.out.println("getname() :Student");
245     return this.name;
246   }
247   public void setcourse(String course)
248   {
249     System.out.println("setcourse() :Student");
250     this.course=course;
251   }
252   public String getcourse()
253   {
254     System.out.println("getcourse() :Student");
255     return this.course;
256   }
257   public void setqualification(String qualification)
258   {
259     System.out.println("setqualification() :Student");
260     this.qualification=qualification;
261   }
262   public String getqualification()
263   {
264     System.out.println("getqualification() :Student");
265     return this.qualification;
266   }
267
268 }
-----Faculty.java-----
270 package bean;
271 public class Faculty
272 {
273   String id,name,subject,qualification;
274   public Faculty()
275   {
276     System.out.println("Faculty().....constructor is called");
277   }
278
279   public void setid(String id)
280   {
281     this.id=id;
282     System.out.println("setId() :Faculty");
283   }
284   public String getid()
285   {
286     System.out.println("getId() :Faculty");
287     return this.id;
288   }
289   public void setname(String name)
290   {
291     System.out.println("setname() :Faculty");
292     this.name=name;
293   }
294   public String getname()
295   {
296     System.out.println("getname() :Faculty");
297     return this.name;
298   }
299
300   public void setssubject(String subject)
301   {
302     System.out.println("setssubject() :Faculty");
303     this.subject=subject;
304   }
```

## STRUCTS

```
305     public String getsubject()
306     {
307         System.out.println("getsubject() :Faculty");
308         return this.subject;
309     }
310
311     public void setqualification(String qualification)
312     {
313         System.out.println("setqualification() :Faculty");
314         this.qualification=qualification;
315     }
316     public String getqualification()
317     {
318         System.out.println("getqualification() :Faculty");
319         return this.qualification;
320     }
321
322 }
```

---

```
323 -----StudentAction.java-----
```

```
324 import java.io.*;
325 import javax.servlet.RequestDispatcher;
326 import javax.servlet.ServletException;
327 import javax.servlet.http.HttpServletRequest;
328 import javax.servlet.http.HttpServletResponse;
329 import javax.servlet.http.HttpSession;
330 import org.apache.struts.action.Action;
331 import org.apache.struts.action.ActionError;
332 import org.apache.struts.action.ActionErrors;
333 import org.apache.struts.action.ActionForm;
334 import org.apache.struts.action.ActionForward;
335 import org.apache.struts.action.ActionMapping;
336 import org.apache.struts.action.ActionServlet;
337 import org.apache.struts.action.DynaActionForm;
338 import org.apache.struts.util.MessageResources;
339 import java.sql.*;
340 import java.util.ArrayList;
341 public class StudentAction extends org.apache.struts.action.Action
342 {
343
344     public ActionForward execute(ActionMapping mapping,
345                               ActionForm form,
346                               HttpServletRequest request,
347                               HttpServletResponse response) throws Exception
348     {
349         try
350         {
351             System.out.println("execute() :StudentAction");
352
353             StudentFormBean sfb=(StudentFormBean)form;
354             String id=sfb.getid();
355
356             Class.forName("oracle.jdbc.driver.OracleDriver");
357             Connection
358             con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
359
360             PreparedStatement ps=con.prepareStatement("select * from students where id = ?");
361             ps.setString(1,id);
362             ResultSet rs=ps.executeQuery();
363
364             int count=0;
365             ArrayList al=new ArrayList();
```

## DURGASOFT

```

365      while(rs.next())
366      {
367          count=1;
368          bean.Student st=new bean.Student();
369
370          st.setid(rs.getString(1));
371          st.setname(rs.getString(2));
372          st.setcourse(rs.getString(3));
373          st.setqualification(rs.getString(4));
374          al.add(st);
375      }
376      if(count==0)
377          return mapping.findForward("fail");
378
379      request.setAttribute("res",al);
380  } //try
381  catch(Exception e)
382  {
383      e.printStackTrace();
384  }
385  return mapping.findForward("succ");
386 } //execute
388 } //class
-----fail.jsp-----
390 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
391
392 <center><font size=6>No Data Found</font></center>
393 <%request.removeAttribute("res");%>
394 -----FacultyAction.java-----
395 import java.io.*;
396 import javax.servlet.RequestDispatcher;
397 import javax.servlet.ServletException;
398 import javax.servlet.http.HttpServlet;
399 import javax.servlet.http.HttpServletRequest;
400 import javax.servlet.http.HttpSession;
401 import org.apache.struts.action.Action;
402 import org.apache.struts.action.ActionError;
403 import org.apache.struts.action.ActionErrors;
404 import org.apache.struts.action.ActionForm;
405 import org.apache.struts.action.ActionForward;
406 import org.apache.struts.action.ActionMapping;
407 import org.apache.struts.action.ActionServlet;
408 import org.apache.struts.action.DynaActionForm;
409 import org.apache.struts.util.MessageResources;
410 import java.sql.*;
411 import java.util.*;
412
413 public class FacultyAction extends org.apache.struts.action.Action
414 {
415     public FacultyAction()
416     {
417         System.out.println("FacultyAction().....constructor is called");
418     }
419
420     public ActionForward execute(ActionMapping mapping,
421                               ActionForm form,
422                               HttpServletRequest request,
423                               HttpServletResponse response) throws Exception
424     {
425         try

```

```
426    {
427        System.out.println("execute() :FacultyAction");
428        FacultyFormBean sfb=(FacultyFormBean)form;
429        String id=sfb.getid();
430
431        Class.forName("oracle.jdbc.driver.OracleDriver");
432        Connection
433        con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
434
435        PreparedStatement ps=con.prepareStatement("select * from faculty where fid = ?");
436        ps.setString(1,id);
437        ResultSet rs=ps.executeQuery();
438
439        int count=0;
440        ArrayList al=new ArrayList();
441
442        while(rs.next())
443        {
444            count=1;
445            bean.Faculty st=new bean.Faculty();
446            st.setid(rs.getString(1));
447            st.setname(rs.getString(2));
448            st.setsubject(rs.getString(3));
449            st.setqualification(rs.getString(4));
450            al.add(st);
451        }
452        if(count==0)
453            return mapping.findForward("fail");
454
455        request.setAttribute("resf",al);
456    }
457    catch(Exception e)
458    {
459        e.printStackTrace();
460    }
461    return mapping.findForward("succ");
462 }
463 -----fail.jsp-----
464 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
465 <center><font size=6>No Data Found</font></center>
466 <% request.removeAttribute("resf"); %>
```

```

1 App10(I18N)
2 =====
3 -----Customer.jsp-----
4 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
5 <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
6
7 <html:form action="Customer">
8 <bean:message key="label1"/><html:text property="cno"/> <br>
9 <bean:message key="label2"/><html:text property="cname"/> <br>
10 <bean:message key="label3"/><html:text property="bill"/> <br>
11 <html:submit value="Submit"/>
12 </html:form>
13 -----web.xml-----
14 <?xml version="1.0" encoding="UTF-8"?>
15 <web-app xmlns="http://java.sun.com/xml/ns/j2ee"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
16 version="2.4" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd">
17 <servlet>
18   <servlet-name>action</servlet-name>
19   <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
20   <init-param>
21     <param-name>config</param-name>
22     <param-value>/WEB-INF/struts-config.xml</param-value>
23   </init-param>
24   <load-on-startup>0</load-on-startup>
25 </servlet>
26 <servlet-mapping>
27   <servlet-name>action</servlet-name>
28   <url-pattern>*.do</url-pattern>
29 </servlet-mapping>
30 <welcome-file-list>
31   <welcome-file>Customer.jsp</welcome-file>
32 </welcome-file-list>
33 </web-app>
34 -----struts-config.xml-----
35 <?xml version="1.0" encoding="UTF-8"?>
36 <!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration
1.2//EN"
37 "http://struts.apache.org/dtds/struts-config\_1\_3.dtd">
38
39 <struts-config>
40   <form-beans>
41     <form-bean name="CustForm" type="CustFormBean"></form-bean>
42   </form-beans>
43
44   <action-mappings>
45     <action path="/Customer" name="CustForm" scope="request" type="CustomerAction">
46       <forward name="success" path="/Success.jsp"></forward>
47     </action>
48   </action-mappings>
49
50   <message-resources parameter="App"/>
51   <message-resources parameter="App_de"/>
52   <message-resources parameter="App_fr"/>
53 </struts-config>
54 -----CustFormBean.java-----
55 import org.apache.struts.action.ActionForm;
56
57 public class CustFormBean extends ActionForm
58 {

```

## STRUTS

DURGASOFT

```
59  private String cno,cname,bill;
60
61  public CustFormBean()
62  {
63      System.out.println("constructor is called:CustFormBean()");
64  }
65
66  public String getCno()
67  {
68      System.out.println("getCno():CustFormBean");
69      return cno;
70  }
71
72  public void setCno(String cno)
73  {
74      System.out.println("setCno():CustFormBean");
75      this.cno = cno;
76  }
77
78  public String getName()
79  {
80      System.out.println("getName():CustFormBean");
81      return cname;
82  }
83
84  public void setName(String cname)
85  {
86      System.out.println("setName():CustFormBean");
87      this.cname = cname;
88  }
89
90  public String getBill()
91  {
92      System.out.println("getBill():CustFormBean");
93      return bill;
94  }
95
96  public void setBill(String bill)
97  {
98      System.out.println("setBill():CustFormBean");
99      this.bill = bill;
100 }
101 }
102 -----CustomerAction.java-----
103 import javax.servlet.http.*;
104 import org.apache.struts.action.*;
105
106 public class CustomerAction extends Action
107 {
108
109     public CustomerAction()
110     {
111         System.out.println("CustomerAction():CustomerAction");
112     }
113     public ActionForward execute(ActionMapping mapping,ActionForm form,
114                               HttpServletRequest request,
115                               HttpServletResponse response)throws Exception
116     {
117
118         CustFormBean cfb=new CustFormBean();
119     }
```

## STRUTS

DURGASOFT

```
120     String cno=cfb.getCno();
121     String cname=cfb.getcname();
122     String bill=cfb.getBill();
123
124     HttpSession session=request.getSession(true);
125     session.setAttribute("cno",cno);
126     session.setAttribute("cname",cname);
127     session.setAttribute("bill",bill);
128
129     return mapping.findForward("success");
130   }
131 }
-----Success.jsp-----
132 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
133 <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
135
136 <center>
137 <bean:message key="label1"/><bean:write property="cno" scope="session"/> <br>
138 <bean:message key="label2"/><bean:write property="cname" scope="session"/> <br>
139 <bean:message key="label3"/><bean:write property="bill" scope="session"/> <br>
140 </center>
141 -----App.properties-----
142 label1=Customer No
143 label2=Customer Name
144 label3=Customer Bill
145 -----App_de.properties-----
146 label1=JCustomer No
147 label2=JCustomer Name
148 label3=JCustomer Bill
149 -----App_fr.properties-----
150 label1=Customero Noo
151 label2=Customero Nameo
152 label3=Customero Billo
153
```

## STRUTS

DURGASOFT

```
1 App11(Struts File Uploading Application)
2 =====
3 -----upload.jsp-----
4 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
5
6 <html:form action="upload.do" enctype="multipart/form-data">
7   Enter File1:<html:file property="file1" /><br /><br />
8   Enter File2:<html:file property="file2" /><br /><br />
9   <html:submit />
10 </html:form>
11 -----display.jsp-----
12 <center> The uploaded files are <center>
13
14 <table>
15   <tr>
16     <td><b>The File name1</b>:</td>
17     <td><%= request.getAttribute("fileName1") %></td>
18   </tr>
19   <tr>
20     <td><b>The File name2</b>:</td>
21     <td> <%= request.getAttribute("fileName2") %> </td>
22   </tr>
23 </table>
24 -----web.xml-----
25 <?xml version="1.0" encoding="ISO-8859-1"?>
26 <!DOCTYPE web-app
27   PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
28   "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
29
30 <web-app>
31   <servlet>
32     <servlet-name>action</servlet-name>
33     <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
34   <init-param>
35     <param-name>config</param-name>
36     <param-value>/WEB-INF/struts-config.xml</param-value>
37   </init-param>
38   <load-on-startup>1</load-on-startup>
39 </servlet>
40
41 <servlet-mapping>
42   <servlet-name>action</servlet-name>
43   <url-pattern>*.do</url-pattern>
44 </servlet-mapping>
45
46 <welcome-file-list>
47   <welcome-file>upload.jsp</welcome-file>
48 </welcome-file-list>
49 </web-app>
50 -----struts-config.xml-----
51 <!DOCTYPE struts-config PUBLIC
52   "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
53   "http://struts.apache.org/dtds/struts-config_1_3.dtd">
54
55 <struts-config>
56
57   <form-beans>
58     <form-bean name="uf" type="UploadForm"/>
59   </form-beans>
60
61   <action-mappings>
```

## STRUTS

DURGASOFT

```
62      <action path="/upload" type="UploadAction" name="uf">
63          <forward name="display" path="/display.jsp" />
64      </action>
65  </action-mappings>
66
67 </struts-config>
-----UploadForm.java-----
68 import org.apache.struts.upload.FormFile;
69 import org.apache.struts.action.ActionForm;
70
71 public class UploadForm extends ActionForm
72 {
73     protected FormFile file1,file2;
74
75     public FormFile getFile1()
76     {
77         System.out.println("getFile1():UploadForm");
78         return file1;
79     }
80
81     public void setFile1(FormFile file1)
82     {
83         System.out.println("setFile1():UploadForm");
84         this.file1 = file1;
85     }
86
87     public FormFile getFile2()
88     {
89         System.out.println("getFile2():UploadForm");
90         return file2;
91     }
92
93     public void setFile2(FormFile file2)
94     {
95         System.out.println("setFile2():UploadForm");
96         this.file2 = file2;
97     }
98 } //class
-----UploadAction.java-----
100 import java.io.InputStream;
101 import java.io.IOException;
102 import java.io.OutputStream;
103 import java.io.FileOutputStream;
104 import java.io.ByteArrayOutputStream;
105 import java.io.FileNotFoundException;
106
107 import javax.servlet.http.HttpServletRequest;
108 import javax.servlet.http.HttpServletResponse;
109
110 import org.apache.struts.upload.FormFile;
111 import org.apache.struts.action.Action;
112 import org.apache.struts.action.ActionForm;
113 import org.apache.struts.action.ActionMapping;
114 import org.apache.struts.action.ActionForward;
115 import org.apache.struts.action.ForwardingActionForward;
116
117 public class UploadAction extends Action
118 {
119     public ActionForward perform(ActionMapping mapping,
120                               ActionForm form,
121                               HttpServletRequest request,
```

```
123             HttpServletRequest response)
124     {
125
126     try
127     {
128         System.out.println("perform():UploadAction");
129         UploadForm uf = (UploadForm) form;
130
131         FormFile file1 =uf.getFile1();
132         FormFile file2 =uf.getFile2();
133
134         InputStream stream1=file1.getInputStream();
135         InputStream stream2=file2.getInputStream();
136
137         String fname1=file1.getFileName();
138         String fname2=file2.getFileName();
139
140
141         OutputStream bos1 = new FileOutputStream(fname1);
142         int bytesRead = 0;
143         byte[] buffer = new byte[1024];
144         while ((bytesRead = stream1.read(buffer, 0,1024)) != -1)
145         {
146             bos1.write(buffer, 0, bytesRead);
147         }
148         bos1.close();
149
150
151
152         OutputStream bos2 = new FileOutputStream(fname2);
153         bytesRead = 0;
154         while ((bytesRead = stream2.read(buffer, 0, 1024)) != -1)
155         {
156             bos2.write(buffer, 0, bytesRead);
157         }
158         bos2.close();
159
160         request.setAttribute("fileName1", fname1);
161         request.setAttribute("fileName2", fname2);
162
163         return mapping.findForward("display");
164
165     } //try
166     catch (Exception e)
167     {
168         return mapping.findForward("display");
169     }
170
171     }//method
172 } //class
```

```

1 App (Application on File Downloading(Resource Downloading)
2 -----
3 -----MyJsp.jsp-----
4 <%@taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
5 <html:link action="/dpath">Download Word Doc</html:link>
6 -----web.xml-----
7 <?xml version="1.0" encoding="UTF-8"?>
8 <web-app xmlns="http://java.sun.com/xml/ns/j2ee"
9 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
10 version="2.4" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
11 http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
12 <servlet>
13 <servlet-name>action</servlet-name>
14 <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
15 <init-param>
16   <param-name>config</param-name>
17   <param-value>/WEB-INF/struts-config.xml</param-value>
18 </init-param>
19 <load-on-startup>0</load-on-startup>
20 </servlet>
21
22 <servlet-mapping>
23   <servlet-name>action</servlet-name>
24   <url-pattern>*.do</url-pattern>
25 </servlet-mapping>
26
27 <welcome-file-list>
28   <welcome-file>index.jsp</welcome-file>
29 </welcome-file-list>
30 </web-app>
31 -----struts-config.xml-----
32 <?xml version="1.0" encoding="UTF-8"?>
33 <!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration
1.2//EN"
34 "http://struts.apache.org/dtds/struts-config_1_2.dtd">
35 <struts-config>
36   <action-mappings >
37     <action path="/dpath" type="MyAction" parameter="/abc.doc" />
38   </action-mappings>
39 </struts-config>
40 -----MyAction.java-----
41 //MyAction.java
42 import java.io.*;
43 import javax.servlet.*;
44 import javax.servlet.http.*;
45 import org.apache.struts.action.*;
46 import org.apache.struts.actions.*;
47
48 public class MyAction extends DownloadAction {
49
50   protected StreamInfo getStreamInfo(ActionMapping mapping,
51                                     ActionForm form,
52                                     HttpServletRequest request,
53                                     HttpServletResponse response)
54   throws Exception {
55
56     // get the File Name from parameter attribute of <action> tag.
57     String fileName=mapping.getParameter();
58     // Set the content disposition
59     response.setHeader("Content-disposition","attachment;filename="+fileName);
60     // set the content type(MIME type)

```

## STRUTS

```
61     String contentType="application/msword";
62     //get Access to ServletContext obj
63     ServletContext sc=servlet.getServletContext();
64     // return Stream obj pointing to the file to be downloaded.
65     return new ResourceStreamInfo(contentType,sc,fileName);
66   } //method
67 } //class
68
69
70
71
72
73
```

DURGASOFT

```
1 App12(TilesFramework based Application)
2 =====
3 -----index.jsp-----
4 <%
5 response.sendRedirect("mytiles.do");
6 %>
7 -----web.xml-----
8 <?xml version="1.0" encoding="ISO-8859-1"?>
9
10 <!DOCTYPE web-app
11 PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
12 "http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">
13
14 <web-app>
15   <!-- Action Servlet Configuration -->
16   <servlet>
17     <servlet-name>action</servlet-name>
18     <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
19     <init-param>
20       <param-name>config</param-name>
21       <param-value>/WEB-INF/struts-config.xml</param-value>
22     </init-param>
23
24
25
26
27
28   <load-on-startup>1</load-on-startup>
29 </servlet>
30
31   <!-- Action Servlet Mapping -->
32   <servlet-mapping>
33     <servlet-name>action</servlet-name>
34     <url-pattern>*.do</url-pattern>
35   </servlet-mapping>
36   <!-- The Usual Welcome File List -->
37   <welcome-file-list>
38     <welcome-file>index.jsp</welcome-file>
39   </welcome-file-list>
40
41 </web-app>
42 -----welcome.jsp-----
43 <b>
44   <center>
45     Hello Welcom To Tiles Examples
46   </center>
47 </b>
48 -----update.jsp-----
49 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
50
51 <html:form action="updatedata">
52   <table>
53     <tr>
54       <td> Employee Id </td><td><html:text property="eid"/></td>
55     </tr>
56     <tr>
57       <td> Employee Name </td><td><html:text property="name" /></td>
58     </tr>
59     <tr>
60       <td> Employee Address </td><td><html:textarea property="address"/> </td>
61     </tr>
```

## STRUCTS

## DURGASOFT

```
62 <tr>
63   <td colspan=2 align="center"><html:submit/><html:reset/> </td>
64 </tr>
65 </table>
66 </html:form>
67 -----insert.jsp-----
68 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html"%>
69
70 <html:form action="insertdata">
71   <table>
72     <tr>
73       <td> Employee Id </td><td><html:text property="eid"/></td>
74     </tr>
75     <tr>
76       <td> Employee Name </td><td><html:text property="name"/> </td>
77     </tr>
78     <tr>
79       <td> Employee Address </td><td><html:textarea property="address"/></td>
80     </tr>
81     <tr>
82       <td colspan=2 align="center"><html:submit/><html:reset/> </td>
83     </tr>
84   </table>
85 </html:form>
86 -----delete.jsp-----
87 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html"%>
88
89 <html:form action="deletedata">
90   <table>
91     <tr>
92       <td> Employee Id </td><td><html:text property="eid"/></td>
93     </tr>
94     <tr>
95       <td colspan=2 align="center"><html:submit/><html:reset/></td>
96     </tr>
97   </table>
98 </html:form>
99 -----MyTilesHome.jsp-----
100 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
101 <%@ taglib uri="http://struts.apache.org/tags-tiles" prefix="tiles" %>
102
103 <table width="100%" height="100%" cellspacing="0" cellpadding="4" >
104   <tr>
105     <td align="center" colspan="2"><tiles:insert attribute="header"/></td>
106   </tr>
107
108   <tr>
109     <td width="20%"><tiles:insert attribute="left-content"/></td>
110     <td width="80%"><tiles:insert attribute="main-content" /></td>
111   </tr>
112
113   <tr>
114     <td colspan="2"><tiles:insert attribute="footer" /></td>
115   </tr>
116 </table>
117 -----BaseHeader.jsp-----
118 <div align="center" style="color:#000033; background-color: #CCD9F9; font-size: 25px;
119   font-weight: 900;
120   text-align: center; border-width: 1px; border-style: solid; height: 50px"> Tiles Sample
121   Application<br>
122   <font style="color:#000033; background-color: #CCD9F9; font-size: 11px; font-weight: 100;
```

```
121     text-align: center;">Developed in Struts Exadel Studio</font></div>
122 -----BaseFooter.jsp-----
123 <div align="center" style="color:#000033; background-color: #CCD9F9; font-size: 11px;
124 font-weight: 900;
125 text-align: center; border-width: 1px; border-style: solid; height:30px"> All Rights Are Reserved By
126 Sathya Technologies @ 2010<br>
127         <font style="color:#000033; background-color: #CCD9F9; font-size: 11px; font-weight: 100;
128 text-align: center;">Developed in Struts Exadel Studio</font></div>
129 -----LeftNavigation.jsp-----
130 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
131 <table width="100%" style="color:#000033; background-color: #CCD9F9; font-size: 15px;
132 font-weight: 900;
133     text-align: center; border-width: 1px; border-style: solid; border-color:#000033">
134 <tr>
135     <td align="center" height=30></td>
136 <tr>
137     <td align="center" height=30>
138         <html:link action="insert" >insert</html:link>
139     </td>
140 </tr>
141     .
142 <tr>
143     <td align="center" height=30>
144         <html:link action="update" >update<br></html:link>
145     </td>
146 </tr>    .
147 <tr>
148     <td align="center" height=30>
149         <html:link action="delete" >delete</html:link>
150     </td>
151 </tr>
152 .
153 <tr>
154     <td align="center" height=30></td>
155 </tr>
156 </table>
157 -----success.jsp-----
158 <table width="100%">
159 <tr>
160     <td align="center"></td>
161 </tr>
162 .
163 <tr>
164     <td align="center"></td>
165 </tr>
166 .
167 <br><br><b>
168 <%=String)request.getAttribute("Action")%>
169     Operation has Successfully Completed</b></td></tr>
170 </table>
171 .
172 -----failure.jsp-----
173 <table width="100%">
174 <tr>
175     <td align="center"><b>
176         <font color="red">Warning</font>
177         <%=String)request.getAttribute("Action")%>
178     </b></td></tr>
```

## STRUSTS

## DURGASOFT

```
179      <b>Sorry The Operation You Have Done is Failed</b>
180      </td>
181      </tr>
182  </table>
183  -----struts-config.xml-----
184  <!DOCTYPE struts-config PUBLIC
185      "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
186      "http://struts.apache.org/dtds/struts-config_1_3.dtd">
187  <struts-config>
188      <form-beans>
189          <form-bean name="dataForm" type="org.apache.struts.action.DynaActionForm">
190              <form-property name="eid" type="java.lang.String"/>
191              <form-property name="name" type="java.lang.String"/>
192              <form-property name="address" type="java.lang.String"/>
193          </form-bean>
194
195          <form-bean name="delForm" type="org.apache.struts.action.DynaActionForm">
196              <form-property name="eid" type="java.lang.String"/>
197          </form-bean>
198      </form-beans>
199
200      <action-mappings>
201          <action path="/mytiles" forward="basedef"/>
202          <action path="/insert" forward="insertTile"/>
203          <action path="/update" forward="updateTile" />
204          <action path="/delete" forward="deleteTile"/>
205
206          <action path="/updatedata" name="dataForm" type="updateAction" scope="session">
207              <forward name="success" path="tile.main.success"/>
208              <forward name="failure" path="tile.main.failure"/>
209          </action>
210
211          <action path="/insertdata" name="dataForm" type="insertAction" scope="session">
212              <forward name="success" path="tile.main.success"/>
213              <forward name="failure" path="tile.main.failure"/>
214          </action>
215
216          <action path="/deletedata" name="delForm" type="delAction" scope="session">
217              <forward name="success" path="tile.main.success"/>
218              <forward name="failure" path="tile.main.failure"/>
219          </action>
220      </action-mappings>
221
222      <plug-in className="org.apache.struts.tiles.TilesPlugin">
223          <set-property property="definitions-config" value="/WEB-INF/tiles-defs.xml"/>
224          <set-property property="moduleAware" value="true"/>
225      </plug-in>
226
227  </struts-config>
228
229  -----tiles-defs.xml-----
230  <!DOCTYPE tiles-definitions PUBLIC
231      "-//Apache Software Foundation//DTD Tiles Configuration//EN"
232      "http://jakarta.apache.org/struts/dtds/tiles-config_1_3.dtd">
233
234  <tiles-definitions>
235
236      <definition name="basedef" path="/pages/MyTilesHome.jsp">
237          <put name="header" value="/pages/BaseHeader.jsp" />
238          <put name="left-content" value="/pages/LeftNavigation.jsp"/>
239          <put name="main-content" value="/pages/welcome.jsp"/>
```

```

240     <put name="footer" value="/pages/BaseFooter.jsp" />
241   </definition>
242
243   <definition name="tile.main.success" extends="basedef">
244     <put name="main-content" value="/pages/success.jsp" />
245   </definition>
246
247   <definition name="tile.main.failure" extends="basedef">
248     <put name="main-content" value="/pages/failure.jsp" />
249   </definition>
250
251   <definition name="insertTile" extends="basedef">
252     <put name="main-content" value="/pages/insert.jsp" />
253   </definition>
254
255   <definition name="deleteTile" extends="basedef">
256     <put name="main-content" value="/pages/delete.jsp" />
257   </definition>
258
259   <definition name="updateTile" extends="basedef">
260     <put name="main-content" value="/pages/update.jsp" />
261   </definition>
262
263 </tiles-definitions>
264
265 -----DatabaseBean.java-----
266 package bean;
267 import java.sql.*;
268 import java.io.*;
269
270 public class DatabaseBean
271 {
272     Connection con;
273     Statement st;
274     PreparedStatement prest;
275     int i;
276     public void setConnection()
277     {
278         try
279         {
280             System.out.println("setConnection():DatabaseBean");
281             Class.forName("oracle.jdbc.driver.OracleDriver");
282             con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
283             System.out.println("Connection Successfully Created "+con);
284         } //try
285         catch(Exception e)
286         {
287             con=null;
288             e.printStackTrace();
289             System.out.println("Connection Not Established");
290         }
291     } //setConnection()
292
293     public int insertData(int eid,String name,String add)
294     {
295         System.out.println("insertData():DatabaseBean");
296         try
297         {
298             System.out.println("Connection "+con);
299             st=con.createStatement();
300             String qry="insert into StrutsEmployee values("+eid+",'"+name+"','"+
301             add+"')";

```

```
301     i=st.executeUpdate(qry);
302     System.out.println("Data Inserted Successfully");
303
304     st.close();
305     return 1;
306 } //try
307 catch(Exception e)
308 {
309     e.printStackTrace();
310 }
311     return 0;
312 } //insertData()
313
314 public int updateData(int eid,String name,String add)
315 {
316     try
317     {
318         System.out.println("updateData():DatabaseBean");
319         st=con.createStatement();
320         String query="update StrutsEmployee set name='"+name+"',address='"+add+"' where
321         eid='"+eid;
322
323         i=st.executeUpdate(query);
324         System.out.println("Data Updated Successfully");
325         st.close();
326         if(i==0)
327             return 0;
328         else
329             return 1;
330     } //try
331     catch(Exception e)
332     {
333         e.printStackTrace();
334     }
335     return 0;
336 } // updateData()
337
338 public int deleteData(int eid)
339 {
340     try
341     {
342         System.out.println("deleteData():DatabaseBean");
343         st=con.createStatement();
344         String query="delete from StrutsEmployee where eid='"+eid;
345
346         i=st.executeUpdate(query);
347         System.out.println("Data Deleted Successfully");
348         st.close();
349         if(i==0)
350             return 0;
351         else
352             return 1;
353
354     }
355     catch(Exception e)
356     {
357         e.printStackTrace();
358     }
359     System.out.println("del Qry : The Value of i "+i);
```

## STRUTS

DURGASOFT

```
361     return 0;
362 } // deleteData()
363
364 public void closeConnection()
365 {
366     try
367     {
368         System.out.println("closeConnection():DatabaseBean");
369         con.close();
370     }
371     catch(Exception e)
372     {
373         e.printStackTrace();
374     }
375 }
376 } //closeConnection()
377 } //DatabaseBean
378 -----insertAction.java-----
379 import javax.servlet.http.*;
380 import org.apache.struts.action.*;
381 import org.apache.struts.action.DynaActionForm;
382 import bean.DatabaseBean;
383 public class insertAction extends Action
384 {
385     DatabaseBean dbean;
386     public ActionForward execute(ActionMapping mapping, ActionForm form,
387             HttpServletRequest request, HttpServletResponse response)
388     {
389         System.out.println("execute():insertAction");
390
391         dbean = new DatabaseBean();
392         dbean.setConnection();
393
394         DynaActionForm myForm = (DynaActionForm)form;
395         String eid=(String)myForm.get("eid");
396         int empid=Integer.parseInt(eid);
397         String name=(String)myForm.get("name");
398         String add=(String)myForm.get("address");
399
400         int result=dbean.insertData(empid,name,add);
401         dbean.closeConnection();
402
403         request.setAttribute("Action","Insert");
404
405         if(result==1)
406             return mapping.findForward("success");
407         else
408             return mapping.findForward("failure");
409     }
410 }
411 -----deleteAction.java-----
412 import javax.servlet.http.*;
413 import org.apache.struts.action.*;
414 import org.apache.struts.action.DynaActionForm;
415 import bean.DatabaseBean;
416
417 public class delAction extends Action
418 {
419     DatabaseBean dbean;
420     public ActionForward execute(ActionMapping mapping, ActionForm form,
```

```

422             HttpServletRequest request, HttpServletResponse response)
423         {
424             System.out.println("delAction():DelAction");
425             dbean = new DatabaseBean();
426             dbean.setConnection();
427
428             DynaActionForm myForm = (DynaActionForm)form;
429
430             String eid=(String)myForm.get("eid");
431             int empid=Integer.parseInt(eid);
432             int result=dbean.deleteData(empid);
433
434             dbean.closeConnection();
435             request.setAttribute("Action","Delete");
436
437             if (result==1)
438                 return mapping.findForward("success");
439             else
440                 return mapping.findForward("failure");
441         }
442     }
443 -----updateAction.java-----
444 import javax.servlet.http.*;
445 import org.apache.struts.action.*;
446 import org.apache.struts.action.DynaActionForm;
447 import bean.DatabaseBean;
448 public class updateAction extends Action
449 {
450     DatabaseBean dbean;
451     public ActionForward execute(ActionMapping mapping, ActionForm form,
452                                 HttpServletRequest request, HttpServletResponse response)
453     {
454         System.out.println("execute():updateAction");
455         dbean = new DatabaseBean();
456         dbean.setConnection();
457
458         DynaActionForm myForm = (DynaActionForm)form;
459
460         String eid=(String)myForm.get("eid");
461         int empid=Integer.parseInt(eid);
462         String name=(String)myForm.get("name");
463         String add=(String)myForm.get("address");
464
465         int result=dbean.updateData(empid,name,add);
466
467
468         dbean.closeConnection();
469         request.setAttribute("Action","update");
470
471         if (result==1)
472             return mapping.findForward("success");
473         else
474             return mapping.findForward("failure");
475     }
476 }
```

## TILES FRAMEWORK

Typically during Web application development, the user interface (UI) group creates the site's look and feel. Based on that look and feel, the group creates HTML pages that represent the application's functionality and navigation. With a servlets and JavaServer Pages (JSPs)-based implementation, where HTML pages are converted into servlets and JSPs, UI developers identify common HTML and JSP view components, such as header, footer, body, menu, and search. This article presents various solutions to effectively and efficiently organize HTML and JSP view components. I evaluate each solution using specific criteria, such as page number, code repetition, and layout control.

To explore templating and layout solutions, we will use the Tiles framework. The Tiles framework's view components are known as *tiles*. The framework uses an XML configuration file to organize those tiles. This framework not only enables you to reuse tiles, but also the layouts that organize them.

To explore the more powerful and flexible solutions, we will investigate the synergy between the Tiles and Struts frameworks. Struts is an open source framework for developing Web applications using the popular Model-View-Controller (MVC) or Model 2 architectural pattern. Struts comes with a large set of reusable tags for which the Tiles tag library makes an excellent enhancement.

### Evaluation criteria

I will evaluate each solution based on the criteria below. The criteria are not mutually exclusive. For a specific situation and particular application, you must always balance between the strengths and weaknesses of each solution with respect to these factors.

#### Page number

A solution should strive to minimize the number of HTML and JSP pages. As the page number increases, the complexity of developing, managing, and maintaining an application increases drastically.

#### Code repetition

Under most circumstances, repetition is bad. The more repeated HTML and JSP code, the more difficult it is to develop and maintain an application. A simple change can result in a cascade of changes in many different pages with unpredictable consequences. A concrete and practical way of attaining reuse is to avoid code repetition.

#### Layout control

While code repetition is bad, repetition of layout logic and code can be worse. Spreading the logic and behavior of view component organization over several JSPs can be a recipe for disaster. Attaining reuse of templating and layout logic is a better form of reuse than only reusing view components. Thus, you can achieve a higher level of reuse with effective layout control.

#### Coupling

Coupling is the degree of interactivity between entities. Software engineers are taught again and again to minimize coupling between unrelated classes, packages, and so on. We can apply the same principle to view components. Even though there are distinct view components from a user perspective, in the JSP implementation, the components might be intricately coupled. A solution should reduce coupling between unrelated view components.

#### Complexity

Complexity brings increased development and maintenance costs, making a more complex solution less suitable. Complexity grows fast as well, and what might originally look simple and innocuous can quickly turn into a big mess as you add more pieces.

I'll evaluate several solutions using a basic example of JSPs with common view components, like header and footer. I'll present these solutions in order of increasing complexity, and then I'll measure in detail each one against the evaluation criteria.

### **Solution 1: Basic JSP**

Consider the following JSP for a.jsp:

```
<html>
<body>
Header
<p>
a's body...
<p>
Footer
<p>
</body>
</html>
```

Consider the following JSP for b.jsp:

```
<html>
<body>
Header
<p>
b's body...
<p>
Footer
<p>
</body>
</html>
```

In many cases, the developers obtain the code from the UI group and literally convert it into a JSP as necessary. As shown above, each JSP has a duplicate header and footer. Solution 1 is undesirable because changes in common view components, like header and footer, require changes in all relevant pages, as each page is responsible for laying out the view components. This simple solution lacks foresight. With so much HTML and JSP code duplication, we minimize the number of pages but at a heavy maintenance cost. There is strong coupling between the different view components, which, as I explained earlier, is undesirable.

### **Solution 2: JSP include**

Consider the following JSP for a.jsp:

```
<html>
<body>
<%-- include header --%>
<jsp:include page="/header.jsp" />
```

## STRUTS

a's body...

```
<p>
<%-- include footer --%>
<jsp:include page="/footer.jsp" />
</body>
</html>
```

Consider the following JSP for b.jsp:

```
<html>
<body>
<%-- include header --%>
<jsp:include page="/header.jsp" />
b's body...
<p>
<%-- include footer --%>
<jsp:include page="/footer.jsp" />
</body>
</html>
```

Note that common view components, like header and footer, are split up using the JSP include mechanism.

Consider this header.jsp:

Header

```
<p>
```

Consider this footer.jsp:

Footer

```
<p>
```

Solution 2 nicely addresses some of Solution 1's major shortcomings. You only need to change common view components once. Hence, this solution greatly eliminates HTML and JSP code repetition, significantly improving application maintainability. It increases the page number a bit, but drastically reduces the tight coupling between common view components and other pages. On the complexity scale, this solution is simple and readily implemented on many real-world applications. However, it has one major drawback: if you change how and where you organize the view components (i.e., by changing the component layout), then you would need to update every page -- resulting in an expensive and prohibitive change. Solution 2 achieves view component reuse, but does not achieve the reuse of layout and templating logic.

## Solution 3: Tiles insert

Consider this JSP for a.jsp:

```
<%@ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>
<html>
<body>
<%-- include header --%>
```

## DURGASOFT

## **STRUTS**

```
<tiles:insert page="/header.jsp" flush="true"/>  
a's body...  
<p>  
<%-- include footer --%>  
<tiles:insert page="/footer.jsp" flush="true"/>  
</body>  
</html>
```

## **DURGASOFT**

Consider this JSP for b.jsp:

```
<%@ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>  
<html>  
<body>  
<%-- include header --%>  
<tiles:insert page="/header.jsp" flush="true"/>  
b's body...  
<p>  
<%-- include footer --%>  
<tiles:insert page="/footer.jsp" flush="true"/>  
</body>  
</html>
```

Instead of using the JSP include mechanism, Solution 3 uses the Tiles insert mechanism. Using the Tiles insert tag, you include the view components in the appropriate positions. In all other aspects, the solution mirrors the JSP include solution (Solution 2) exactly, with the same advantages and disadvantages.

## **Solution 4: Splitting bodies**

Consider this a.jsp:

```
<%@ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>  
<html>  
<body>  
<%-- include header --%>  
<tiles:insert page="/header.jsp" flush="true"/>  
<%-- include body --%>  
<tiles:insert page="aBody.jsp" flush="true"/>  
<%-- include footer --%>  
<tiles:insert page="/footer.jsp" flush="true"/>  
</body>  
</html>
```

Consider this b.jsp:

```
<%@ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>
<html>
<body>
<%-- include header --%>
<tiles:insert page="/header.jsp" flush="true"/>
<%-- include body --%>
<tiles:insert page="bBody.jsp" flush="true"/>
<%-- include footer --%>
<tiles:insert page="/footer.jsp" flush="true"/>
</body>
</html>
```

Solution 4 differs slightly from the Tiles insert solution. Solution 4 separates the core bodies into their individual pages, like aBody.jsp and bBody.jsp.

Consider the following JSP for aBody.jsp:

a's body...

```
<p>
```

Consider the following JSP for bBody.jsp:

b's body...

```
<p>
```

Solution 4's advantage: it limits body changes to the respective pages. Also, it lets you reuse the bodies in other places, eliminating the need for repetition and duplication. Thus, the solution further diminishes the coupling between common view components and other application components. Creating and managing each body component introduces an additional complexity level. As with other solutions, each page still does its own layout. Hence, there is no overarching layout policy or scheme.

### **Solution 5: Templating tiles**

Using Tiles's templating feature, you can define the following layout (from the layout.jsp file shown below) as a template. Since this is a layout, you insert placeholders instead of the actual view components using the Tiles insert tag. Thus, for all components, this page defines one reusable layout:

```
<%@ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>
<html>
<body>
<%-- include header --%>
<tiles:insert attribute="header"/>

<%-- include body --%>
<tiles:insert attribute="body"/>
```

```
<%-- include footer --%>
<tiles:insert attribute="footer"/>
</body>
</html>
```

Other content pages, like a.jsp and b.jsp, use the above layout for arranging components. In the actual page, you insert the layout using the Tiles insert tag. Using the Tiles put tag, you can specify the actual view components for all placeholders specified in the layout.

Consider this a.jsp:

```
<%@ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>
<tiles:insert page="/layout.jsp" flush="true">
  <tiles:put name="header" value="/header.jsp"/>
  <tiles:put name="body" value="/aBody.jsp"/>
  <tiles:put name="footer" value="/footer.jsp"/>
</tiles:insert>
```

Consider this b.jsp:

```
<%@ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>
<tiles:insert page="/layout.jsp" flush="true">
  <tiles:put name="header" value="/header.jsp"/>
  <tiles:put name="body" value="/bBody.jsp"/>
  <tiles:put name="footer" value="/footer.jsp"/>
</tiles:insert>
```

Solution 5's most significant advantage is that it encapsulates the layout scheme or mechanism, drastically reducing the coupling between common view components and other content bodies. However, it increases complexity by introducing another layout page. Understanding and implementing templating can also be difficult at first.

#### **Solution 6: Struts and Tiles**

The above layout page, layout.jsp, contains the HTML and JSP code for organizing the components. The content pages, a.jsp and b.jsp, do not contain any HTML code; they just contain the Tiles tags to insert the necessary components. Wouldn't it be nice to specify all the content pages in one XML configuration file?

Let's name that file tileDefinitions.xml and specify its pages as:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<component-definitions>
  <definition name="aDef" path="/layout.jsp">
    <put name="header" value="/header.jsp"/>
    <put name="footer" value="/footer.jsp"/>
    <put name="body" value="/aBody.jsp"/>
  </definition>
  <definition name="bDef" path="/layout.jsp">
```

```
<put name="header" value="/header.jsp"/>
<put name="footer" value="/footer.jsp"/>
<put name="body" value="/bBody.jsp"/>
</definition>
<definition name="cDef" path="/layout.jsp">
<put name="header" value="/header.jsp"/>
<put name="footer" value="/footer.jsp"/>
<put name="body" value="/cBody.jsp"/>
</definition>
</component-definitions>
```

Solution 6 eliminates all the content pages, like a.jsp and b.jsp, by putting their definitions in the XML file. Since a resource like a.jsp no longer exists, how can we request it? More importantly, how can we request the definitions in the tileDefinitions.xml file?

The powerful and synergistic integration of Struts and Tiles comes to the rescue. Besides the regular Struts configuration parameters, we specify the configuration file's location as another parameter in the web.xml file, as shown below. Specifying the definitions-config parameter enables Struts to find and know about the Tiles definitions:

```
<!-- Standard Action Servlet Configuration (with debugging) -->
<servlet>
<servlet-name>action</servlet-name>
<!--
<servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
-->
<servlet-class>org.apache.struts.tiles.ActionComponentServlet</servlet-class>
<init-param>
<param-name>definitions-config</param-name>
<param-value>/WEB-INF/tileDefinitions.xml</param-value>
</init-param>
...
</servlet>
```

Now, we define a Struts action, which returns a definition specified in the configuration file upon success. The Struts action DoFirst is a nonoperational action, as shown below:

```
package com.malani.struts.action;
import org.apache.struts.action.*;
import javax.servlet.http.*;
public class DoFirst extends Action {
    public ActionForward perform(
        ActionMapping aMapping,
        ActionForm aForm,
        HttpServletRequest aRequest,
```

## STRUTS

```
HttpServletResponse aResponse  
) {  
    return aMapping.findForward("success");  
}  
}
```

You cannot invoke a definition directly from the browser, but you can invoke one from Struts as if it is an actual resource. Define the Struts actions in the struts-config.xml file as shown below:

```
<action path="/a"  
       type="com.malani.struts.action.DoFirst"  
>  
    <forward name="success" path="aDef"/>  
</action>  
<action path="/b"  
       type="com.malani.struts.action.DoFirst"  
>  
    <forward name="success" path="bDef"/>  
</action>  
<action path="/c"  
       type="com.malani.struts.action.DoFirst"  
>  
    <forward name="success" path="cDef"/>  
</action>
```

Now, invoke the Struts action by requesting a.do and b.do actions respectively to return the desired resource.

Solution 6's main advantage is that it consolidates all definitions in an XML configuration file. Eliminating the content pages drastically reduces the total page number. By introducing Struts, we turn up the complexity another notch.

## Solution 7: Tiles inheritance

In the definitions configuration file, observe that each page's definition looks similar. Each definition has three components, two of which are fixed as header and footer. A powerful Tiles feature enables inheritance between definitions. Hence, you can define a base definition and let the original definitions inherit from that definition. The original definitions must only supply their unique part. The following shows the XML configuration file with inheritance between definitions:

```
<?xml version="1.0" encoding="ISO-8859-1"?>  
<component-definitions>  
    <definition name="baseDef" path="/layout.jsp">  
        <put name="header" value="/header.jsp"/>  
        <put name="footer" value="/footer.jsp"/>  
        <put name="body" value="" />  
    </definition>
```

## STRUTS

DURGASOFT

```
<definition name="aDef" extends="baseDef">
    <put name="body" value="/aBody.jsp"/>
</definition>

<definition name="bDef" extends="baseDef">
    <put name="body" value="/bBody.jsp"/>
</definition>

<definition name="cDef" extends="baseDef">
    <put name="body" value="/cBody.jsp"/>
</definition>

</component-definitions>
```

Elimination of duplicate and redundant information in the configuration file is an advantage of this solution. Overall, the advantages and disadvantages of this solution are identical to the Struts and Tiles solution.

### Solution summary

The following table summarizes the different solutions with respect to the evaluation criteria. I encourage you to add other creative solutions as well as other important evaluation criteria, such as extensibility, maintainability, and performance.

### Evaluate various solutions per specified criteria

Solution	Page - number	Code repetition	Layout control	Coupling	Complexity
1: Basic	*	***	*	***	
2: JSP include	**	**	*	**	*
3: Tiles insert					
4: Splitting bodies	***	**	**	**	**
5: Templating tiles			***		
6: Struts and Tiles	**	*	***	*	***
Interactions					

**Scale:** High: \*\*\* Medium: \*\* Low: \*

The table shows that each solution's complexity level gradually increases. It also shows that as you increase complexity, you reduce code repetition, increase layout-control flexibility, and

## **STRUTS**

## **DURGASOFT**

diminish coupling between unrelated view components. The page number initially increases as various view components split, but as you define more pages in a definitions configuration file, consolidation will occur.

### **What solution is best?**

The best solution depends on your project's needs and requirements and your skills and knowledge in developing and maintaining a Web application. The Basic solution is too simple; I don't recommend it because it goes against the grain of good software engineering practices. If your Web application is complex, then templating offers great layout control. Hence, you may want to research and utilize a templating framework like Tiles. If you already use Struts, then you should leverage the synergy between Tiles and Struts for a powerful solution.

### **Divine design**

In this article, I evaluated various solutions for organizing view components in HTML and JSPs. I also explored the synergy between the Struts and Tiles frameworks. These strategies and solutions will help you make informed design and architectural decisions regarding your Web applications.

```

1 App13(TilesFramework based Application)
2 =====
3 -----index.jsp-----
4 <%
5 response.sendRedirect("mytiles.do");
6 %>
7 -----web.xml-----
8 <?xml version="1.0" encoding="ISO-8859-1"?>
9
10 <!DOCTYPE web-app
11 PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
12 "http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">
13
14 <web-app>
15 <!-- Action Servlet Configuration -->
16 <servlet>
17   <servlet-name>action</servlet-name>
18   <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
19   <init-param>
20     <param-name>config</param-name>
21     <param-value>/WEB-INF/struts-config.xml</param-value>
22   </init-param>
23
24   <load-on-startup>1</load-on-startup>
25 </servlet>
26
27 <!-- Action Servlet Mapping -->
28 <servlet-mapping>
29   <servlet-name>action</servlet-name>
30   <url-pattern>*.do</url-pattern>
31 </servlet-mapping>
32 <!-- The Usual Welcome File List -->
33 <welcome-file-list>
34   <welcome-file>index.jsp</welcome-file>
35 </welcome-file-list>
36
37 </web-app>
38 -----welcome.jsp-----
39 <b>
40 <center>
41   Hello Welcom To Tiles Examples
42 </center>
43 </b>
44 -----update.jsp-----
45 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
46
47 <html:form action="updatedata">
48 <table>
49   <tr>
50     <td> Employee Id </td><td><html:text property="eid"/></td>
51   </tr>
52   <tr>
53     <td> Employee Name </td><td><html:text property="name" /></td>
54   </tr>
55   <tr>
56     <td> Employee Address </td><td><html:textarea property="address"/> </td>
57   </tr>
58   <tr>
59     <td colspan=2 align="center"><html:submit/><html:reset/> </td>
60   </tr>
61 </table>

```

```

62 </html:form>
63 -----insert.jsp-----
64 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html"%>
65
66 <html:form action="insertdata">
67 <table>
68 <tr>
69   <td> Employee Id </td><td><html:text property="eid"/></td>
70 </tr>
71 <tr>
72   <td> Employee Name </td><td><html:text property="name"/> </td>
73 </tr>
74 <tr>
75   <td> Employee Address </td><td><html:textarea property="address"/></td>
76 </tr>
77 <tr>
78   <td colspan=2 align="center"><html:submit/><html:reset/> </td>
79 </tr>
80 </table>
81 </html:form>
82 -----delete.jsp-----
83 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html"%>
84
85 <html:form action="deletedata">
86 <table>
87 <tr>
88   <td> Employee Id </td><td><html:text property="eid"/></td>
89 </tr>
90 <tr>
91   <td colspan=2 align="center"><html:submit/><html:reset/></td>
92 </tr>
93 </table>
94 </html:form>
95 -----MyTilesHome.jsp-----
96 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
97 <%@ taglib uri="http://struts.apache.org/tags-tiles" prefix="tiles" %>
98
99 <table width="100%" height="100%" cellspacing="0" cellpadding="4" >
100 <tr>
101   <td align="center" colspan="2"><tiles:insert attribute="header"/></td>
102 </tr>
103
104 <tr>
105   <td width="20%"><tiles:insert attribute="left-content"/></td>
106   <td width="80%"><tiles:insert attribute="main-content" /></td>
107 </tr>
108
109 <tr>
110   <td colspan="2"><tiles:insert attribute="footer" /></td>
111 </tr>
112 </table>
113 -----BaseHeader.jsp-----
114 <div align="center" style="color:#000033; background-color: #CCD9F9; font-size: 25px;
font-weight: 900;
115   text-align: center; border-width: 1px; border-style: solid; height: 50px"> Tiles Sample
Application<br>
116   <font style="color:#000033; background-color: #CCD9F9; font-size: 11px; font-weight: 100;
text-align: center;">Developed in Struts Exadel Studio</font></div>
117
118 -----BaseFooter.jsp-----
119 <div align="center" style="color:#000033; background-color: #CCD9F9; font-size: 11px;
font-weight: 900;

```

## STRUTS

## DURGASOFT

```
120 text-align: center; border-width: 1px; border-style: solid; height: 30px" > All Rights Are Reserved By  
121 Sathya Technologies @ 2010<br>  
122 <font style="color: #000033; background-color: #CCD9F9; font-size: 11px; font-weight: 100;  
123 text-align: center;"> Developed in Struts Exadel Studio</font></div>  
124 -----LeftNavigation.jsp-----  
125 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>  
126 <table width="100%" style="color: #000033; background-color: #CCD9F9; font-size: 15px;  
127 font-weight: 900;  
128 text-align: center; border-width: 1px; border-style: solid; border-color: #000033">  
129 <tr>  
130 <td align="center" height=30></td>  
131 </tr>  
132 <tr>  
133 <td align="center" height=30>  
134 <html:link action="insert" >insert</html:link>  
135 </td>  
136 </tr>  
137 <tr>  
138 <td align="center" height=30>  
139 <html:link action="update" >update<br></html:link>  
140 </td>  
141 </tr>  
142 <tr>  
143 <td align="center" height=30>  
144 <html:link action="delete" >delete</html:link>  
145 </td>  
146 </tr>  
147 </table>  
148 -----success.jsp-----  
149 <table width="100%">  
150 <tr>  
151 <td align="center" height=30></td>  
152 </tr>  
153 </table>  
154 -----failure.jsp-----  
155 <table width="100%">  
156 <tr>  
157 <td align="center"></td>  
158 </tr>  
159 <tr>  
160 <td align="center"></td>  
161 </tr>  
162 <br><br><b>  
163 <%= (String) request.getAttribute("Action") %>  
164 Operation has Successfully Completed</b></td></tr>  
165 </table>  
166 -----failure.jsp-----  
167 <table width="100%">  
168 <tr>  
169 <td align="center"><b>  
170 <font color="red">Warning</font>  
171 <%= (String) request.getAttribute("Action") %>  
172 <b>Sorry The Operation You Have Done is Failed</b>  
173 </td>  
174 </tr>  
175 </table>
```

```

179 -----struts-config.xml-----
180 <!DOCTYPE struts-config PUBLIC
181   "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
182   "http://struts.apache.org/dtds/struts-config_1_3.dtd">
183 <struts-config>
184   <form-beans>
185     <form-bean name="dataForm" type="org.apache.struts.action.DynaActionForm">
186       <form-property name="eid" type="java.lang.String"/>
187       <form-property name="name" type="java.lang.String"/>
188       <form-property name="address" type="java.lang.String"/>
189     </form-bean>
190
191     <form-bean name="delForm" type="org.apache.struts.action.DynaActionForm">
192       <form-property name="eid" type="java.lang.String"/>
193     </form-bean>
194   </form-beans>
195
196   <action-mappings>
197     <action path="/mytiles" forward="basedef"/>
198     <action path="/insert" forward="insertTile"/>
199     <action path="/update" forward="updateTile" />
200     <action path="/delete" forward="deleteTile"/>
201
202     <action path="/updatedata" name="dataForm" type="updateAction" scope="session">
203       <forward name="success" path="tile.main.success"/>
204       <forward name="failure" path="tile.main.failure"/>
205     </action>
206
207     <action path="/insertdata" name="dataForm" type="insertAction" scope="session">
208       <forward name="success" path="tile.main.success"/>
209       <forward name="failure" path="tile.main.failure"/>
210     </action>
211
212     <action path="/deletedata" name="delForm" type="delAction" scope="session">
213       <forward name="success" path="tile.main.success"/>
214       <forward name="failure" path="tile.main.failure"/>
215     </action>
216   </action-mappings>
217
218   <plug-in className="org.apache.struts.tiles.TilesPlugin">
219     <set-property property="definitions-config" value="/WEB-INF/tiles-defs.xml"/>
220     <set-property property="moduleAware" value="true"/>
221   </plug-in>
222
223 </struts-config>
224
225 -----tiles-defs.xml-----
226 <!DOCTYPE tiles-definitions PUBLIC
227   "-//Apache Software Foundation//DTD Tiles Configuration//EN"
228   "http://jakarta.apache.org/struts/dtds/tiles-config_1_3.dtd">
229
230 <tiles-definitions>
231
232   <definition name="basedef" path="/pages/MyTilesHome.jsp">
233     <put name="header" value="/pages/BaseHeader.jsp" />
234     <put name="left-content" value="/pages/LeftNavigation.jsp"/>
235     <put name="main-content" value="/pages/welcome.jsp"/>
236     <put name="footer" value="/pages/BaseFooter.jsp" />
237   </definition>
238
239   <definition name="tile.main.success" extends="basedef">

```

```

240    <put name="main-content" value="/pages/success.jsp" />
241  </definition>
242
243  <definition name="tile.main.failure" extends="basedef">
244    <put name="main-content" value="/pages/failure.jsp" />
245  </definition>
246
247  <definition name="insertTile" extends="basedef">
248    <put name="main-content" value="/pages/insert.jsp" />
249  </definition>
250
251  <definition name="deleteTile" extends="basedef">
252    <put name="main-content" value="/pages/delete.jsp" />
253  </definition>
254
255  <definition name="updateTile" extends="basedef">
256    <put name="main-content" value="/pages/update.jsp" />
257  </definition>
258
259 </tiles-definitions>
260
261 -----DatabaseBean.java-----
262 package bean;
263 import java.sql.*;
264 import java.io.*;
265
266 public class DatabaseBean
267 {
268     Connection con;
269     Statement st;
270     PreparedStatement prest;
271     int i;
272     public void setConnection()
273     {
274         try
275         {
276             System.out.println("setConnection():DatabaseBean");
277             Class.forName("oracle.jdbc.driver.OracleDriver");
278             con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
279             System.out.println("Connection Successfully Created "+con);
280             }/try
281             catch(Exception e)
282             {
283                 con=null;
284                 e.printStackTrace();
285                 System.out.println("Connection Not Established");
286             }
287     }//setConnection()
288
289     public int insertData(int eid,String name,String add)
290     {
291         System.out.println("insertData():DatabaseBean");
292         try
293         {
294             System.out.println("Connection "+con);
295             st=con.createStatement();
296             String qry="insert into StrutsEmployee values("+eid+",'"+name+"','"+"+add+"')";
297             i=st.executeUpdate(qry);
298
299             System.out.println("Data Inserted Successfully");
300

```

## STRUTS

```
301     st.close();
302     return 1;
303   } //try
304   catch(Exception e)
305   {
306     e.printStackTrace();
307   }
308   return 0;
309 } //insertData()
310
311
312 public int updateData(int eid,String name,String add)
313 {
314   try
315   {
316     System.out.println("updateData():DatabaseBean");
317     st=con.createStatement();
318     String query="update StrutsEmployee set name='"+name+"',address='"+add+"' where
319     eid='"+eid;
320     i=st.executeUpdate(query);
321     System.out.println("Data Updated Successfully");
322     st.close();
323     if(i==0)
324       return 0;
325     else
326       return 1;
327   } //try
328   catch(Exception e)
329   { e.printStackTrace();}
330
331   return 0;
332 } // updateData()
333
334 public int deleteData(int eid)
335 {
336   try
337   {
338     System.out.println("deleteData():DatabaseBean");
339
340     st=con.createStatement();
341     String query="delete from StrutsEmployee where eid='"+eid;
342
343     i=st.executeUpdate(query);
344     System.out.println("Data Deleted Successfully");
345     st.close();
346     if(i==0)
347       return 0;
348     else
349       return 1;
350
351   }
352   catch(Exception e)
353   {
354     e.printStackTrace();
355   }
356   System.out.println("del Qry : The Value of i "+i);
357
358   return 0;
359 } // deleteData()
360
```

## DURGASOFT

```

361     public void closeConnection()
362     {
363         try
364         {
365             System.out.println("closeConnection():DatabaseBean");
366             con.close();
367         }
368         catch(Exception e)
369         {
370             e.printStackTrace();
371         }
372     }/closeConnection()
373 }//DatabaseBean
374 -----insertAction.java-----
375 import javax.servlet.http.*;
376 import org.apache.struts.action.*;
377 import org.apache.struts.action.DynaActionForm;
378 import bean.DatabaseBean;
379 public class insertAction extends Action
380 {
381     DatabaseBean dbean;
382     public ActionForward execute(ActionMapping mapping, ActionForm form,
383                                 HttpServletRequest request, HttpServletResponse response)
384     {
385         System.out.println("execute():insertAction");
386
387         dbean = new DatabaseBean();
388         dbean.setConnection();
389
390         DynaActionForm myForm = (DynaActionForm)form;
391         String eid=(String)myForm.get("eid");
392         int empid=Integer.parseInt(eid);
393         String name=(String)myForm.get("name");
394         String add=(String)myForm.get("address");
395
396
397         int result=dbean.insertData(empid,name,add);
398         dbean.closeConnection();
399
400         request.setAttribute("Action","Insert");
401
402         if(result==1)
403             return mapping.findForward("success");
404         else
405             return mapping.findForward("failure");
406     }
407 }
408 -----deleteAction.java-----
409 import javax.servlet.http.*;
410 import org.apache.struts.action.*;
411 import org.apache.struts.action.DynaActionForm;
412 import bean.DatabaseBean;
413
414 public class delAction extends Action
415 {
416     DatabaseBean dbean;
417     public ActionForward execute(ActionMapping mapping, ActionForm form,
418                                 HttpServletRequest request, HttpServletResponse response)
419     {
420         System.out.println("delAction():DelAction");
421         dbean = new DatabaseBean();

```

```
422     dbean.setConnection();
423
424     DynaActionForm myForm = (DynaActionForm)form;
425
426     String eid=(String)myForm.get("eid");
427     int empid=Integer.parseInt(eid);
428     int result=dbean.deleteData(empid);
429
430     dbean.closeConnection();
431     request.setAttribute("Action","Delete");
432
433     if (result==1)
434         return mapping.findForward("success");
435     else
436         return mapping.findForward("failure");
437     }
438 }
439 -----updateAction.java-----
440 import javax.servlet.http.*;
441 import org.apache.struts.action.*;
442 import org.apache.struts.action.DynaActionForm;
443 import bean.DatabaseBean;
444 public class updateAction extends Action
445 {
446     DatabaseBean dbean;
447     public ActionForward execute(ActionMapping mapping, ActionForm form,
448                               HttpServletRequest request, HttpServletResponse response)
449     {
450         System.out.println("execute():updateAction");
451         dbean = new DatabaseBean();
452         dbean.setConnection();
453
454         DynaActionForm myForm = (DynaActionForm)form;
455
456         String eid=(String)myForm.get("eid");
457         int empid=Integer.parseInt(eid);
458         String name=(String)myForm.get("name");
459         String add=(String)myForm.get("address");
460
461         int result=dbean.updateData(empid,name,add);
462
463         dbean.closeConnection();
464         request.setAttribute("Action","update");
465
466         if (result==1)
467             return mapping.findForward("success");
468         else
469             return mapping.findForward("failure");
470     }
471 }
472 }
```

## RequestProcessor

### How a Request is Processed

ActionServlet is the only servlet in Struts framework, and is responsible for handling all of the requests. Whenever it receives a request, it first tries to find a sub-application for the current request. Once a sub-application is found, it creates a RequestProcessor object for that sub-application and calls its process() method by passing it HttpServletRequest and HttpServletResponse objects.

The RequestProcessor.process() is where most of the request processing takes place. The process() method is implemented using the Template Method design pattern, in which there is a separate method for performing each step of request processing, and all of those methods are called in sequence from the process() method. For example, there are separate methods for finding the ActionForm class associated with the current request, and checking if the current user has one of the required roles to execute action mapping. This gives us tremendous flexibility. The RequestProcessor class in the Struts distribution provides a default implementation for each of the request-processing steps. That means you can override only the methods that interest you, and use default implementations for rest of the methods. For example, by default Struts calls request.isUserInRole() to find out if the user has one of the roles required to execute the current ActionMapping, but if you want to query a database for this, then then all you have to do is override the processRoles() method and return true or false, based whether the user has the required role or not.

First we will see how the process() method is implemented by default, and then I will explain what each method in the default RequestProcessor class does, so that you can decide what parts of request processing you want to change.

```
public void process(HttpServletRequest request,  
                     HttpServletResponse response)  
throws IOException, ServletException {  
    // Wrap multipart requests with a special wrapper  
    request = processMultipart(request);  
    // Identify the path component we will  
    // use to select a mapping  
    String path = processPath(request, response);  
    if (path == null) {  
        return;  
    }  
    if (log.isDebugEnabled()) {  
        log.debug("Processing a " + request.getMethod() +  
                 " for path " + path + "");  
    }  
    // Select a Locale for the current user if requested  
    processLocale(request, response);  
    // Set the content type and no-caching headers  
    // if requested
```

**STRUCTS****DURGASOFT**

```
processContent(request, response);
processNoCache(request, response);
// General purpose preprocessing hook
if (!processPreprocess(request, response)) {
    return;
}
// Identify the mapping for this request
ActionMapping mapping =
    processMapping(request, response, path);
if (mapping == null) {
    return;
}
// Check for any role required to perform this action
if (!processRoles(request, response, mapping)) {
    return;
}
// Process any ActionForm bean related to this request
ActionForm form =
    processActionForm(request, response, mapping);
processPopulate(request, response, form, mapping);
if (!processValidate(request, response, form, mapping)) {
    return;
}
// Process a forward or include specified by this mapping
if (!processForward(request, response, mapping)) {
    return;
}
if (!processInclude(request, response, mapping)) {
    return;
}
// Create or acquire the Action instance to
// process this request
Action action =
    processActionCreate(request, response, mapping);
if (action == null) {
    return;
}
// Call the Action instance itself
```

## STRUTS

DURGASOFT

```
ActionForward forward =  
    processActionPerform(request, response,  
        action, form, mapping);  
  
    // Process the returned ActionForward instance  
    processForwardConfig(request, response, forward);  
}
```

1. **processMultipart():** In this method, Struts will read the request to find out if its contentType is multipart/form-data. If so, it will parse it and wrap it in a wrapper implementing HttpServletRequest. When you are creating an HTML FORM for posting data, the contentType of the request is application/x-www-form-urlencoded by default. But if your form is using FILE-type input to allow the user to upload files, then you have to change the contentType of the form to multipart/form-data. But by doing that, you can no longer read form values submitted by user via the getParameter() method of HttpServletRequest; you have to read the request as an InputStream and parse it to get the values.
2. **processPath():** In this method, Struts will read request URI to determine the path element that should be used for getting the ActionMapping element.
3. **processLocale():** In this method, Struts will get the Locale for the current request and, if configured, it will save it in HttpSession as the value of the org.apache.struts.action.LOCALE attribute. HttpSession would be created as a side effect of this method. If you don't want that to happen, then you can set the locale property to false in ControllerConfig by adding these lines to your *struts-config.xml* file:

4. <controller>
5. <set-property property="locale" value="false"/>
6. </controller>
7. **processContent():** Sets the contentType for the response by calling response.setContentType(). This method first tries to get the contentType as configured in *struts-config.xml*. It will use text/html by default. To override that, use the following:
8. <controller>
9. <set-property property="contentType" value="text/plain"/>
10. </controller>
11. **processNoCache():** Struts will set the following three headers for every response, if configured for no-cache:
- 12.
13. requested in struts config.xml
14. response.setHeader("Pragma", "No-cache");
15. response.setHeader("Cache-Control", "no-cache");
16. response.setDateHeader("Expires", 1);

If you want to set the no-cache header, add these lines to *struts-config.xml*:

```
<controller>  
    <set-property property="noCache" value="true"/>  
</controller>
```

## STRUTS

## DURGA30FT

17. `processPreprocess()`: This is a general purpose, pre-processing hook that can be overridden by subclasses. Its implementation in RequestProcessor does nothing and always returns true. Returning false from this method will abort request processing.
18. `processMapping()`: This will use path information to get an ActionMapping object. The ActionMapping object represents the `<action>` element in your `struts-config.xml` file.
- 19.
20.   `<action path="/newcontact" type="com.sample.NewContactAction"`
21.       `name="newContactForm" scope="request">`
22.           `<forward name="sucess" path="/sucessPage.do"/>`
23.           `<forward name="failure" path="/failurePage.do"/>`
24.   `</action>`

The ActionMapping element contains information like the name of the Action class and ActionForm used in processing this request. It also has information about ActionForwards configured for the current ActionMapping.

25. `processRoles()`: Struts web application security just provides an authorization scheme. What that means is once user is logged into the container, Struts' `processRoles()` method can check if he has one of the required roles for executing a given ActionMapping by calling `request.isUserInRole()`.

- 26.

27.   `<action path="/addUser" roles="administrator"/>`

Say you have `AddUserAction` and you want only the `administrator` to be able to add a new user. What you can do is to add a role attribute with the value

`administrator` in your `AddUserAction` action element. So before executing `AddUserAction`, it will always make sure that the user has the `administrator` role.

28. `processActionForm()`: Every ActionMapping has a ActionForm class associated with it. When Struts is processing an ActionMapping, it will find the name of the associated ActionForm class from the value of the `name` attribute in the `<action>` element.

29.   `<form-bean name="newContactForm"`
30.       `type="org.apache.struts.action.DynaActionForm">`
31.           `<form-property name="firstName"`
32.           `type="java.lang.String"/>`
33.           `<form-property name="lastName"`
34.           `type="java.lang.String"/>`
35.   `</form-bean>`

In our example, it will first check to see if an object of the `org.apache.struts.action.DynaActionForm` class is present in request scope. If so, it will use it; otherwise, it will create a new object and set it in the request scope.

36. `processPopulate()`: In this method, Struts will populate the ActionForm class instance variables with values of matching request parameters.

37. `processValidate()`: Struts will call the `validate()` method of your ActionForm class. If you return `ActionErrors` from the `validate()` method, it will redirect the user to the page indicated by the `input` attribute of the `<action>` element.

## **STRUTS**

## **DURGASOFT**

38. `processForward()` and `processInclude()`: In these functions, Struts will check the value of the forward or include attributes of the `<action>` element and, if found, put the forward or include request in the configured page.

39.

40.       `<action forward="/Login.jsp" path="/loginInput"/>`

41.       `<action include="/Login.jsp" path="/loginInput"/>`

You can guess difference in these functions from their names. `processForward()` ends up calling `RequestDispatcher.forward()`, and `processInclude()` calls `RequestDispatcher.include()`. If you configure both forward and include attributes, it will always call forward, as it is processed first.

42. `processActionCreate()`: This function gets the name of the Action class from the type attribute of the `<action>` element and create and return instances of it. In our case it will create an instance of the `com.sample.NewContactAction` class.

43. `processActionPerform()`: This function calls the `execute()` method of your Action class, which is where you should write your business logic.

44. `processForwardConfig()`: The `execute()`method of your Action class will return an object of type `ActionForward`, indicating which page should be displayed to the user. So Struts will create `RequestDispatcher` for that page and call the `RequestDispatcher.forward()` method.

The above list explains what the default implementation of `RequestProcessor` does at every stage of request processing and the sequence in which various steps are executed. As you can see, `RequestProcessor` is very flexible and it allows you to configure it by setting properties in the `<controller>` element. For example, if your application is going to generate XML content instead of HTML, then you can inform Struts about this by setting a property of the controller element:

### **Creating Your own RequestProcessor**

Above, we saw how the default implementation of `RequestProcessor` works. Now we will present a example of how to customize it by creating our own custom `RequestProcessor`. To demonstrate creating a custom `RequestProcessor`, we will change our sample application to implement these two business requirements:

- We want to create a `ContactImageAction` class that will generate images instead of a regular HTML page.
- Before processing every request, we want to check that user is logged in by checking for `userName` attribute of the session. If that attribute is not found, we will redirect the user to the login page.

We will change our sample application in two steps to implement these business requirements.

1. Create your own `CustomRequestProcessor` class, which will extend the `RequestProcessor` class, like this:
2. `public class CustomRequestProcessor`
3.     `extends RequestProcessor {`
4.     `protected boolean processPreprocess (`
5.       `HttpServletRequest request,`
6.       `HttpServletResponse response) {`
7.       `HttpSession session = request.getSession(false);`
8.       `//If user is trying to access login page`
9.       `// then don't check`
10.      `if( request.getServletPath().equals("/loginInput.do")`
11.       `|| request.getServletPath().equals("/login.do") )`

## STRUTS

## DURGASOFT

```
12.     return true;
13. //Check if userName attribute is there is session.
14. //If so, it means user has allready logged in
15. if( session != null &&
16. session.getAttribute("userName") != null)
17.     return true;
18. else{
19.     try{
20.         //If no redirect user to login Page
21.         request.getRequestDispatcher
22.             ("/Login.jsp").forward(request,response);
23.     }catch(Exception ex){
24.     }
25. }
26. return false;
27. }
28.
29. protected void processContent(HttpServletRequest request,
30.     HttpServletResponse response) {
31. //Check if user is requesting ContactImageAction
32. // if yes then set image/gif as content type
33. if( request.getServletPath().equals("/contactimage.do")){
34.     response.setContentType("image/gif");
35.     return;
36. }
37. super.processContent(request, response);
38. }
39. }
```

In the processPreprocess method of our CustomRequestProcessor class, we are checking for the userName attribute of the session and if it's not found, redirect the user to the login page.

For our requirement of generating images as output from the ContactImageAction class, we have to override the processContent method and first check if the request is for the /contactimage path. If so, we set the contentType to image/gif; otherwise, it's text/html.

40. Add these lines to your *struts-config.xml* file after the *<action-mapping>* element to inform Struts that CustomRequestProcessor should be used as the RequestProcessor class:

```
41. <controller>
42.     <set-property property="processorClass"
43.         value="com.sample.util.CustomRequestProcessor"/>
44. </controller>
```

Please note that overriding processContent() is OK if you have very few Action classes where you want to generate output whose contentType is something other than text/html. If that is not the case, you should create a Struts sub-application for handling requests for image-generating Actions and set image/gif as the .contentType for it.

The Tiles framework uses its own RequestProcessor for decorating output generated by Struts.

```

1 App>>>>>>>>>>Preventing Double Posting by using Tokens>>>>>>>>>>>>
2 -----index.jsp-----
3 <jsp:forward page="beforeRegister.do">
4   <jsp:param name="dispatch" value="savingToken"/>
5 </jsp:forward>
6 -----register.jsp-----
7 <%@taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
8
9 <html:form action="register">
10  UserName:<html:text property ="username"/><br>
11  Password:<html:password property="password"/><br>
12  <html:hidden property="dispatch" value="process1"/>
13  <html:submit value="CheckDetails"/>
14 </html:form>
15 -----error.jsp-----
16 <h1><center>Double Posting Not Allowed</center>
17 <p>
18  <a href="beforeRegister.do?dispatch=savingToken">Form Page</a>
19 </p>
20 -----success.jsp-----
21 <b><h1><center>ResultPage</center></h1><br>
22 Result is : <%=request.getAttribute("msg") %>
23 <p> <a href="beforeRegister.do?dispatch=savingToken">FormPage</a></p>
24 -----web.xml-----
25 <web-app>
26  <servlet>
27    <servlet-name>action</servlet-name>
28    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
29    <init-param>
30      <param-name>config</param-name>
31      <param-value>/WEB-INF/struts-config.xml</param-value>
32    </init-param>
33    <load-on-startup>1</load-on-startup>
34  </servlet>
35
36  <servlet-mapping>
37    <servlet-name>action</servlet-name>
38    <url-pattern>*.do</url-pattern>
39  </servlet-mapping>
40
41  <welcome-file-list>
42    <welcome-file>index.jsp</welcome-file>
43  </welcome-file-list>
44 </web-app>
45 -----struts-config.xml-----
46 <!DOCTYPE struts-config PUBLIC
47   "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
48   "http://struts.apache.org/dtds/struts-config_1_3.dtd">
49 <struts-config>
50  <form-beans>
51    <form-bean name="rf" type="app.RegisterForm"/>
52  </form-beans>
53
54  <action-mappings>
55    <action path="/register" type="app.RegisterAction" name="rf" parameter="dispatch">
56      <forward name="success" path="/success.jsp"/>
57      <forward name="failure" path="/error.jsp"/>
58    </action>
59
60    <action path="/beforeRegister" type="app.RegisterAction" name="rf" parameter="dispatch">
61      <forward name="success" path="/register.jsp"/>

```

```
62 </action>
63 </action-mappings>
64 </struts-config>
65 -----RegisterForm.java-----
66 package app;
67 import org.apache.struts.action.*;
68
69 public class RegisterForm extends ActionForm
70 {
71     private String username;
72     private String password;
73
74     //write getXxx() and setXxx(-) methods
75     public void setUsername(String uname)
76     {
77         username=uname;
78     }
79
80     public void setPassword(String pwd)
81     {
82         password=pwd;
83     }
84
85     public String getUsername()
86     {
87         return username;
88     }
89
90     public String getPassword()
91     {
92         return password;
93     }
94 }
95 -----RegisterAction.java-----
96 package app;
97 import org.apache.struts.actions.*;
98 import org.apache.struts.action.*;
99 import javax.servlet.http.*;
100 public class RegisterAction extends DispatchAction
101 {
102     public ActionForward process1(ActionMapping mapping,
103                                 ActionForm form,HttpServletRequest req,
104                                 HttpServletResponse res)throws Exception
105     {
106         System.out.println("process1(-,-,-) method");
107         System.out.println("Token valid"+isTokenValid(req));
108         if(isTokenValid(req))
109         {
110             System.out.println("Processing B Logic");
111             Thread.sleep(10000);
112             RegisterForm fm=(RegisterForm)form;
113             if(fm.getUsername().equals("raja") && fm.getPassword().equals("hyd"))
114             {
115                 req.setAttribute("msg","validCredentials");
116             }
117             else
118             {
119                 req.setAttribute("msg","InvalidCredentials");
120             }
121             resetToken(req);
122             return mapping.findForward("success");
123     }
124 }
```

## STRUTS

DURGASOFT

```
123    }
124    else
125        return mapping.findForward("failure");
126    }//process1
127
128 public ActionForward savingToken(ActionMapping mapping,
129         ActionForm form,HttpServletRequest req,
130         HttpServletResponse res) throws Exception
131 {
132     System.out.println("savingToken(-,-,-,-) method");
133     //register token for client .
134     saveToken(req);
135     return mapping.findForward("success");
136 } //savingToken
137 }//class
```

```

1 App13(Struts-to-EJB3.x comp Communication)
2 =====
3 -----user.jsp-----
4 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
5
6 <html:html>
7   <html:form action="/wishact">
8     <table border=0 align=center>
9       <tr>
10      <th>Enter UserName </th>
11      <td><html:text property="username"/></td>
12    </tr>
13    <tr>
14      <html:submit value="Register"/>
15    </tr>
16  </table>
17 </html:form>
18 </html:html>
19 -----web.xml-----
20 <?xml version="1.0" encoding="UTF-8"?>
21 <web-app xmlns="http://java.sun.com/xml/ns/j2ee"
22   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
23   version="2.4" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd">
24   <servlet>
25     <servlet-name>action</servlet-name>
26     <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
27     <init-param>
28       <param-name>config</param-name>
29       <param-value>/WEB-INF/struts-config.xml</param-value>
30     </init-param>
31     <load-on-startup>0</load-on-startup>
32   </servlet>
33   <servlet-mapping>
34     <servlet-name>action</servlet-name>
35     <uri-pattern>*.do</uri-pattern>
36   </servlet-mapping>
37   <welcome-file-list>
38     <welcome-file>user.jsp</welcome-file>
39   </welcome-file-list>
40 </web-app>
41 -----struts-config.xml-----
42 <?xml version="1.0" encoding="UTF-8"?>
43 <!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration
44   1.2//EN"
45   "http://struts.apache.org/dtds/struts-config\_1\_2.dtd">
46
47   <struts-config>
48     <form-beans>
49       <form-bean name="rf" type="UserForm"></form-bean>
50     </form-beans>
51     <action-mappings>
52       <action path="/wishact" type="UserAction" name="rf">
53         <forward name="res" path="/result.jsp"></forward>
54       </action>
55     </action-mappings>
56   </struts-config>
57 -----UserForm.java-----
58 import org.apache.struts.action.ActionForm;

```

```
59 public class UserForm extends ActionForm
60 {
61     private String username;
62
63     public String getUsername()
64     {
65         System.out.println("getUsername():UserForm");
66         return username;
67     }
68
69     public void setUsername(String username)
70     {
71         System.out.println("setUsername():UserForm");
72         this.username = username;
73     }
74
75 }
76
77 -----UserAction.java-----
78 import javax.naming.InitialContext;
79 import javax.servlet.http.HttpServletRequest;
80 import javax.servlet.http.HttpServletResponse;
81
82 import org.apache.struts.action.Action;
83 import org.apache.struts.action.ActionForm;
84 import org.apache.struts.action.ActionForward;
85 import org.apache.struts.action.ActionMapping;
86
87 public class UserAction extends Action
88 {
89     Wish wor=null;
90     public UserAction()
91     {
92         try
93         {
94             System.out.println("userAction():UserAction");
95             javax.naming.InitialContext ic=new InitialContext();
96             wor=(Wish)ic.lookup("Wish");
97         }//try
98         catch (Exception e)
99         {
100             e.printStackTrace();
101         }
102
103     }//constructor
104     public ActionForward execute(ActionMapping mapping,ActionForm form,
105                               HttpServletRequest request,
106                               HttpServletResponse response)throws Exception
107     {
108         try
109         {
110             System.out.println("In UserAction:execute()");
111             UserForm rf=(UserForm)form;
112
113             String user=rf.getUsername();
114             String result=wor.getWishMsg(user);
115
116             request.setAttribute("res","result");
117         }//try
118         catch (Exception e)
119         {
```

```
120     e.printStackTrace();
121 }
122
123     return mapping.findForward("res");
124 }
125 }
126
127 -----result.jsp-----
128 <html>
129 <body>
130 <center>
131     <font size=5 color="green">Result Page</font>
132     Wish Message is <%= request.getAttribute("res") %>
133 </center>
134 </body>
135 </html>
136
137 ****
138 EJB 3.X
139 ****
140 -----Wish.java-----
141 //Wish.java(B.interface)(POJI)
142 import javax.ejb.Remote;
143
144 @Remote
145 public interface Wish
146 {
147     public String getWishMsg(String name);
148 }
149
150 -----WishBean.java-----
151 //WishBean.java(Bean class)(POJO)
152
153 import java.util.Calendar;
154
155 import javax.ejb.Stateless;
156
157 @Stateless
158 public class WishBean implements Wish
159 {
160
161     public String getWishMsg(String name)
162     {
163         System.out.println("getWishMsg():WishBean");
164         Calendar cl=Calendar.getInstance();
165         int h=cl.get(Calendar.HOUR_OF_DAY);
166         if(h<12)
167             return "Good Morning"+name;
168         else if(h<16)
169             return "Good Afternoon"+name;
170         else
171             return "Good Evening"+name;
172     }
173
174 }
175 =====
==
```

## STRUTS 2.X

The strut-2 framework is designed for the compilation of the entire development cycle including of building, developing and maintaining the whole application. It is very extensible as each class of the framework is based on an Interface and all the base classes are given an extra application and even you can add your own. The basic platform requirements are Servlet API 2.4, JSP API 2.0 and Java 5.

Some of the general features of the current Apache Strut 2 framework are given below.

**Architecture** – First the web browser request a resource for which the Filter Dispatcher decides the suitable action. Then the Interceptors use the required functions and after that the Action method executes all the functions like storing and retrieving data from a database. Then the result can be seen on the output of the browser in HTML, PDF, images or any other.

**Tags** - Tags in Strut 2 allow creating dynamic web applications with less number of coding. Not only these tags contain output data but also provide style sheet driven markup that in turn helps in creating pages with less code. Here the tags also support validation and localization of coding that in turn offer more utilization. The less number of codes also makes it easy to read and maintain.

**MVC** – The **Model View Controller** in Strut 2 framework acts as a coordinator between application's model and web view. Its Controller and View components can come together with other technology to develop the model. The framework has its library and markup tags to present the data dynamically.

**Configuration** – Provides a deployment descriptor to initialize resources in XML format. The initialization takes place simply by scanning all the classes using Java packages or you can use an application configuration file to control the entire configuration. Its general-purpose defaults allow using struts directly Out of the box.

Configuration files are re-loadable that allows changes without restarting a web container.

### Other Features:

- All framework classes are based on interfaces and core interfaces are independent from HTTP.
- Check boxes do not require any kind of special application for false values.
- Any class can be used as an action class and one can input properties by using any JavaBean directly to the action class.
- Strut 2 actions are Spring friendly and so easy to Spring integration.
- AJAX theme enables to make the application more dynamic.
- Portal and servlet deployment are easy due to automatic portlet support without altering code.
- The request handling in every action makes it easy to customize, when required.

Apache Struts is an open-source framework that is used for developing Java web application. Originally developed by the programmer and author Craig R. McClanahan, this was later taken over by the Apache Software Foundation in 2002. Struts have provided an excellent framework for developing application easily by organizing JSP

and Servlet based on HTML formats and Java code. Strut1 with all standard Java technologies and packages of Jakarta assists to create an extensible development environment. However, with the growing demand of web application, **Strut 1** does not stand firm and needs to be changed with demand. This leads to the creation of Strut2, which is more developer friendly with features like Ajax, rapid development and extensibility.

Struts is a well-organized framework based on **MVC** architecture. In **Model-View-Controller Architecture**, Model stands for the business or database code, View represents the page design

## STRUTS

## DURGASOFT

code and the Controller for navigational code. All these together makes Struts an essential framework for building Java applications. But with the development of new and lightweight MVC based frameworks like Spring, Stripes and Tapestry, it becomes necessary to modify the Struts framework. So, the team of Apache Struts and another J2EE framework, **WebWork** of *OpenSymphony* joined hand together to develop an advanced framework with all possible developing features that will make it developer and user friendly.

**Strut2** contains the combined features of Struts Ti and WebWork 2 projects that advocates higher level application by using the architecture of WebWork2 with features including a plugin framework, a new API, Ajax tags etc. So the Struts communities and the WebWork team brought together several special features in WebWork2 to make it more advance in the Open Source world. Later the name of WebWork2 has changed to Struts2. Hence, Apache Strut 2 is a dynamic, extensible framework for a complete application development from building, deploying and maintaining.

WebWork is a framework for web-application development that has been included in Struts framework 2.0 release. It has some unique concepts and constructs like its compatibility of working within existing Web APIs in Java rather than trying to replace them completely. It has been built specifically taking into account the developer's productivity and code simplicity. Furthermore it is completely context dependent that provides a wrapper around XWork. When working on web applications the web work provides a context that helps web developer in specific implementations.

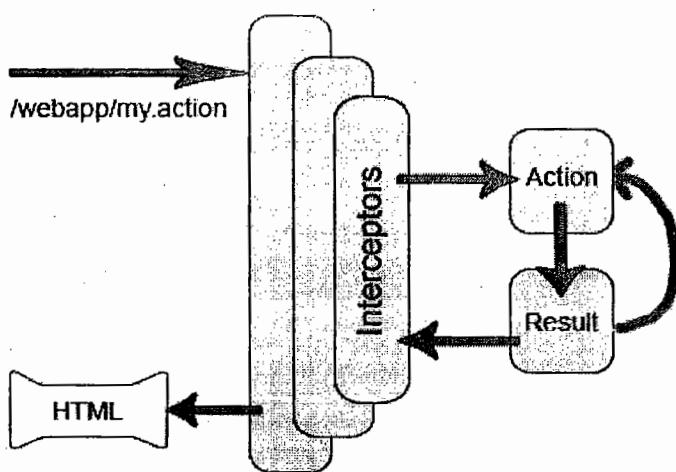
While, XWork provides a mechanism that is used for configuration and factory implementation management. This mechanism is dependencies inject mechanism.

Struts and webwork has joined together to develop the Struts 2 Framework. Struts 2 Framework is very extensible and elegant for the development of enterprise web application of any size. In this section we are going to explain you the architecture of Struts 2 Framework.

### Request Lifecycle in Struts 2 applications

1. **User Sends request:** User sends a request to the server for some resource.
2. **FilterDispatcher determines the appropriate action:** The FilterDispatcher looks at the request and then determines the appropriate Action.
3. **Interceptors are applied:** Interceptors configured for applying the common functionalities such as workflow, validation, file upload etc. are automatically applied to the request.
4. **Execution of Action:** Then the action method is executed to perform the database related operations like storing or retrieving data from the database.
5. **Output rendering:** Then the Result renders the output.
6. **Return of Request:** Then the request returns through the interceptors in the reverse order. The returning request allows us to perform the clean-up or additional processing.
7. **Display the result to user:** Finally the control is returned to the servlet container, which sends the output to the user browser.

Image: Struts 2 high level overview of request processing:



## Struts 2 Architecture

Struts 2 is a very elegant and flexible front controller framework based on many standard technologies like Java Filters, Java Beans, ResourceBundles, XML etc.

For the **Model**, the framework can use any data access technologies like JDBC, EJB, Hibernate etc and for the **View**, the framework can be integrated with JSP, JTL, JSF, Jakarta Velocity Engine, Templates, PDF, XSLT etc.

### Exception Handling:

The Struts 2 Framework allows us to define exception handlers and interceptors.

- **ExceptionHandlers:**

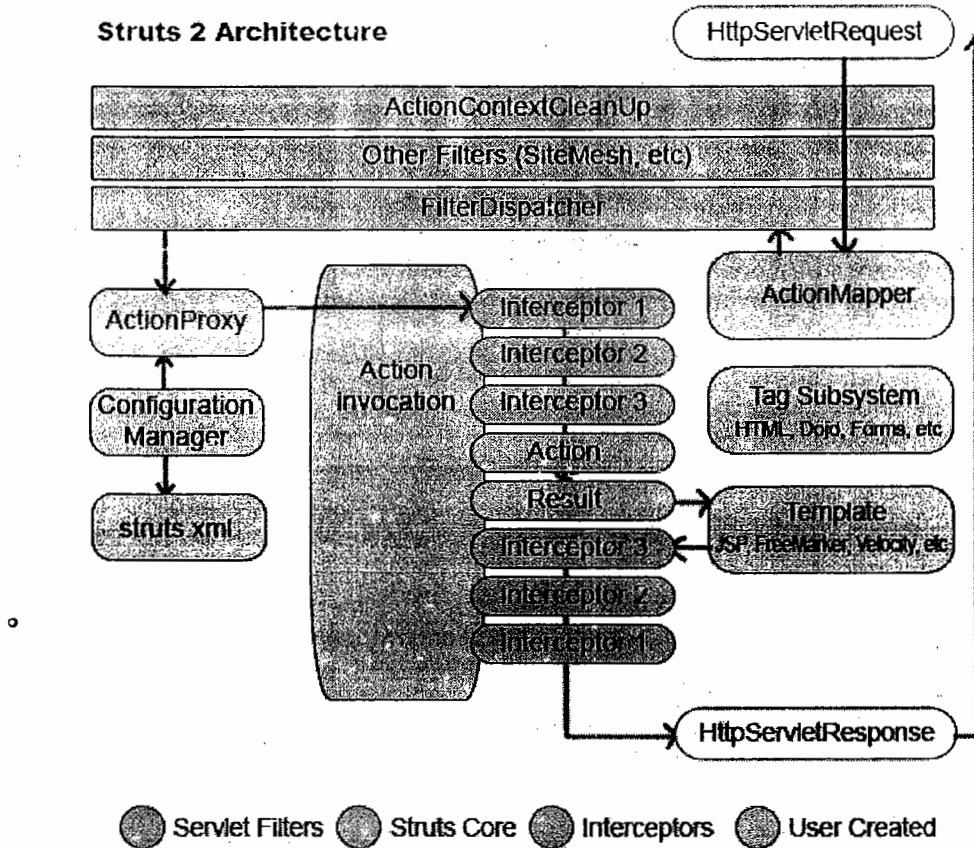
Exception handlers allow us to define the exception handling procedure on global and local basis. Framework catches the exception and then displays the page of our choice with appropriate message and exception details.

- **Interceptors:**

The Interceptors are used to specify the "request-processing lifecycle" for an action. Interceptors are configured to apply the common functionalities like workflow, validation etc.. to the request.

## Struts 2 Architecture

The following diagram depicts the architecture of Struts 2 Framework and also shows the initial request goes to the servlet container such as tomcat, which is then passed through standard filter chain.



The filter chain includes:

- **ActionContextCleanUp filter:** The ActionContextCleanUp filter is optional and it is useful when integration has to be done with other technologies like SiteMash Plug-in.
- **FilterDispatcher:** Next the FilterDispatcher is called, which in turn uses the ActionMapper to determine whether to invoke an Action or not. If the action is required to be invoked, the FilterDispatcher delegates the control to the ActionProxy.
- **ActionProxy:**  
The ActionProxy takes help from Configuration Files manager, which is initialized from the struts.xml. Then the ActionProxy creates an **ActionInvocation**, which implements the command pattern. The ActionInvocation process invokes the Interceptors (if configured) and then invokes the action. The the ActionInvocation looks for proper result. Then the result is executed, which involves the rendering of JSP or templates.

Then the Interceptors are executed again in reverse order. Finally the response returns through the filters configured in web.xml file. If the ActionContextCleanUp filter is configured, the FilterDispatcher does not clean the ThreadLocal ActionContext. If the ActionContextCleanUp filter is not present then the FilterDispatcher will cleanup all the ThreadLocals present.

In this section we have learnt about the Architecture of Struts 2 Framework.

**The new version Struts 2.0 is a combination of the Sturts action framework and Webwork. According to the Struts 2.0.1 release announcement, some key features are:**

## STRUTS

## DURGASOFT

- **Simplified Design** - Programming the abstract classes instead of interfaces is one of design problem of struts1 framework that has been resolved in the struts 2 framework. Most of the Struts 2 classes are based on interfaces and most of its core interfaces are HTTP independent. Struts 2 Action classes are framework independent and are simplified to look as simple POJOs. Framework components are tried to keep loosely coupled.
- **Simplified Actions** - Actions are simple POJOs. Any java class with execute() method can be used as an Action class. Even we don't need to implement interfaces always. Inversion of Control is introduced while developing the action classes. This make the actions to be neutral to the underlying framework .
- **No more ActionForms** - ActionForms feature is no more known to the struts2 framework. Simple JavaBean flavored actions are used to put properties directly. No need to use all String properties.
- **Simplified testability** - Struts 2 Actions are HTTP independent and framework neutral. This enables to test struts applications very easily without resorting to mock objects.
- **Intelligent Defaults** - Most configuration elements have a default value which can be set according to the need. Even there are xml-based default configuration files that can be overridden according to the need.
- **Improved results** - Unlike ActionForwards, Struts 2 Results provide flexibility to create multiple type of outputs and in actual it helps to prepare the response.
- **Better Tag features** - Struts 2 tags enables to add style sheet-driven markup capabilities, so that we can create consistent pages with less code. Struts 2 tags are more capable and result oriented. Struts 2 tag markup can be altered by changing an underlying stylesheet. Individual tag markup can be changed by editing a FreeMarker template. Both JSP and FreeMarker tags are fully supported.
- **Annotations introduced** : Applications in struts 2 can use Java 5 annotations as an alternative to XML and Java properties configuration. Annotations minimize the use of xml.
- **Stateful Checkboxes** - Struts 2 checkboxes do not require special handling for false values.
- **QuickStart** - Many changes can be made on the fly without restarting a web container.
- **customizing controller** - Struts 1 lets to customize the request processor per module, Struts 2 lets to customize the request handling per action, if desired.
- **Easy Spring integration** - Struts 2 Actions are Spring-aware. Just need to add Spring beans!
- **Easy plugins** - Struts 2 extensions can be added by dropping in a JAR. No manual configuration is required!
- **AJAX support** - The AJAX theme gives interactive applications a significant boost. The framework provides a set of tags to help you ajaxify your applications, even on Dojo. The AJAX features include:
  1. AJAX Client Side Validation
  2. Remote form submission support (works with the submit tag as well)
  3. An advanced div template that provides dynamic reloading of partial HTML
  4. An advanced template that provides the ability to load and evaluate JavaScript remotely
  5. An AJAX-only tabbed Panel implementation
  6. A rich pub-sub event model
  7. Interactive auto complete tag

**Differences between Struts 1.X and Struts 2.X :**

In the following section, we are going to compare the various features between the two frameworks. Struts 2.x is very simple as compared to struts 1.x, few of its excellent features are:

**1. Servlet Dependency:**

Actions in Struts1 have dependencies on the servlet API since the **HttpServletRequest** and **HttpServletResponse** objects are passed to the execute method when an Action is invoked but in case of Struts 2, Actions are not container dependent because they are made simple POJOs. In struts 2, the servlet contexts are represented as simple Maps which allows actions to be tested in isolation. Struts 2 Actions can access the original request and response, if required. However, other architectural elements reduce or eliminate the need to access the HttpServletRequest or HttpServletResponse directly.

**2. Action classes**

Programming the abstract classes instead of interfaces is one of design issues of struts1 framework that has been resolved in the struts 2 framework. Struts1 Action classes needs to extend framework dependent abstract base class. But in case of Struts 2 Action class *may* or *may not* implement interfaces to enable optional and custom services. In case of Struts 2 , Actions are not container dependent because they are made simple POJOs. Struts 2 provides a base ActionSupport class to implement commonly used interfaces. Albeit, the Action interface is **not** required. Any POJO object with an execute signature can be used as an Struts 2 Action object.

**3. Validation**

Struts1 and Struts 2 both supports the manual validation via a validate method. Struts1 uses validate method on the ActionForm, or validates through an extension to the Commons Validator. However, Struts 2 supports manual validation via the validate method and the XWork Validation framework. The Xwork Validation Framework supports chaining validation into sub-properties using the validations defined for the properties class type and the validation context.

**4. Threading Model**

In Struts1, Action resources must be thread-safe or synchronized. So Actions are singletons and thread-safe, there should only be one instance of a class to handle all requests for that Action. The singleton strategy places restrictions on what can be done with Struts1 Actions and requires extra care to develop. However in case of Struts 2, Action objects are instantiated for each request, so there are no thread-safety issues. (In practice, servlet containers generate many throw-away objects per request, and one more object does not impose a performance penalty or impact garbage collection.)

**5. Testability**

Testing Struts1 applications are a bit complex. A major hurdle to test Struts1 Actions is that the execute method because it exposes the Servlet API. A third-party extension, Struts TestCase, offers a set of mock object for Struts1. But the Struts 2 Actions can be tested by instantiating the Action, setting properties and invoking methods. Dependency Injection support also makes testing simpler. Actions in struts2 are simple POJOs and are framework independent, hence testability is quite easy in struts2.

**6. Harvesting Input**

Struts1 uses an ActionForm object to capture input. And all ActionForms needs to extend a framework dependent base class. JavaBeans cannot be used as ActionForms, so the developers have to create redundant classes to capture input. However Struts 2 uses Action properties (as input properties independent of underlying framework) that eliminates the need for a second input object, hence reduces redundancy. Additionally in struts2, Action properties can be

## **STRUTS**

**DURGASOFT**

accessed from the web page via the taglibs. Struts 2 also supports the ActionForm pattern, as well as POJO form objects and POJO Actions. Even rich object types, including business or domain objects, can be used as input/output objects.

### **7. Expression Language**

Struts1 integrates with JSTL, so it uses the JSTL-EL. The struts1 EL has basic object graph traversal, but relatively weak collection and indexed property support. Struts 2 can also use JSTL, however it supports a more powerful and flexible expression language called "Object Graph Notation Language" (OGNL).

### **8. Binding values into views**

In the view section, Struts1 uses the standard JSP mechanism to bind objects (processed from the model section) into the page context to access. However Struts 2 uses a "ValueStack" technology so that the taglibs can access values without coupling your view to the object type it is rendering. The ValueStack strategy allows the reuse of views across a range of types which may have the same property name but different property types.

### **9. Type Conversion**

Usually, Struts1 ActionForm properties are all Strings. Struts1 uses Commons-Beanutils for type conversion. These type converters are per-class and not configurable per instance. However Struts 2 uses OGNL for type conversion. The framework includes converters for basic and common object types and primitives.

### **10. Control of Action Execution**

Struts1 supports separate Request Processor (lifecycles) for each module, but all the Actions in a module must share the same lifecycle. However Struts 2 supports creating different lifecycles on a per Action basis via Interceptor Stacks. Custom stacks can be created and used with different Actions as needed.

```

1 -----
2 App1 (Basic Application)
3 -----
4 -----index.jsp-----
5 <%@ taglib uri="/struts-tags" prefix="s" %>
6 <html>
7   <head>
8     <title>Personal Info</title>
9   </head>
10  <body>
11    <h2><s:text name="app.welcome"/></h2>
12
13  <s:form action="displayAction">
14    <s:textfield key="app.username" name="username" />
15    <s:textfield key="app.city" name="city" />
16    <s:submit key="button.cap"/>
17  </s:form>
18
19 </body>
20 <% System.out.println("from Index.jsp" ); %>
21 </html>
22 -----web.xml-----
23 <web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
24   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
25   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
26   http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
27   <display-name>Struts 2 Application</display-name>
28   <filter>
29     <filter-name>struts2</filter-name>
30     <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
31   </filter>
32   <filter-mapping>
33     <filter-name>struts2</filter-name>
34     <url-pattern>*</url-pattern>
35   </filter-mapping>
36
37   <welcome-file-list>
38     <welcome-file>user.jsp</welcome-file>
39   </welcome-file-list>
40 </web-app>
41 -----struts.properties-----
42 struts.custom.i18n.resources=p1.ApplicationResources
43 -----ApplicationResources.properties-----
44 # Resources for parameter 'p1.ApplicationResources'
45 # Project strutsproject
46 app.title=Struts 2 Application
47 app.welcome=Personal Information
48 app.username=User Name
49 app.city=City
50 button.cap=Register
51 -----struts.xml-----
52 <?xml version="1.0" encoding="UTF-8" ?>
53 <!DOCTYPE struts PUBLIC
54   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
55   "http://struts.apache.org/dtds/struts-2.0.dtd ">
56 <struts>
57   <include file="struts-default.xml" />
58   <package name="default" extends="struts-default">
59     <action name="displayAction" class="p1.DisplayAction">

```

```

61      <result name="success">/success.jsp</result>
62      <result name="fail">/fail.jsp</result>
63  </action>
64 </package>
65 </struts>
66 -----DisplayAction.java-----
67 package p1;
68
69 public class DisplayAction {
70     private String username;
71     private String city;
72
73     public DisplayAction()
74     {
75         System.out.println("DisplayAction constructor");
76     }
77
78     public String getCity() {
79         System.out.println("getCity ()");
80         return city;
81     }
82     public void setCity(String city) {
83         System.out.println("setCity ()");
84         this.city = city;
85     }
86
87     public String getUsername() {
88         System.out.println("getUsername ()");
89         return username;
90     }
91
92     public void setUsername(String username) {
93         System.out.println("setUsername ()");
94         this.username = username;
95     }
96
97     public String execute() throws Exception {
98         System.out.println("Execute ()");
99         if("nataraz".equals(username)&& "hyd".equals(city))
100             return "succcess";
101         else
102             return "fail";
103     } //execute()
104 } //class
105 -----success.jsp-----
106 <%@ taglib uri="/struts-tags" prefix="s" %>
107 <html>
108 <head>
109   <title>Personal Info.</title>
110 </head>
111
112 <body>
113   <b> valid Credentials </b>
114   <% System.out.println("from success.jsp" ); %>
115   <s:a href="index.jsp">Back</s:a>
116 </body>
117 </html>
118 -----fail.jsp-----
119 <%@ taglib uri="/struts-tags" prefix="s" %>
120 <html>
121 <head>

```

```

122 <title>Personal Info.</title>
123 </head>
124
125 <body>
126   <b> Invalid Credentials </b>
127   <% System.out.println("from failure.jsp" ); %>
128   <s:a href="index.jsp">Back</s:a>
129 </body>
130 </html>
131 -----
132
133
134
135
136
137
138
139
140 =====
141 App2(Application using Annotations)
142 =====
143 -----user.jsp-----
144 <%@ taglib uri="/struts-tags" prefix="s" %>
145 <html>
146   <head>
147     <title>User Info</title>
148   </head>
149   <body>
150     <h2>User Information</h2>
151
152     <s:form action="show" method="post">
153       <s:textfield key="app.name" name="name" />
154       <s:textfield key="app.age" name="age" />
155       <s:submit key="button.cap"/>
156     </s:form>
157
158   </body>
159 </html:>
160 -----struts.properties-----
161 struts.custom.i18n.resources=nats.ApplicationResources
162 -----ApplicationResources.properties-----
163 # Resources for parameter 'nats.ApplicationResources'
164 # Project strutsproject
165
166 app.title=Struts 2 Application
167 app.name=Enter Name
168 app.age=Enter Age
169 button.cap=send
170
171 my.err.age.msg=provide age between 10 and 30
172 my.err.name.msg=please provide name
173 -----web.xml-----
174 <?xml version="1.0" encoding="UTF-8"?>
175 <web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
176   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
177   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
178   http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
179     <display-name>Struts 2 Application</display-name>
180     <filter>
181       <filter-name>struts2</filter-name>
182       <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>

```

```

182      <init-param>
183          <param-name>actionPackages</param-name>
184          <param-value>nats</param-value>
185      </init-param>
186  </filter>
187
188  <filter-mapping>
189      <filter-name>struts2</filter-name>
190      <url-pattern>/*</url-pattern>
191  </filter-mapping>
192
193  <welcome-file-list>
194      <welcome-file>user.jsp</welcome-file>
195  </welcome-file-list>
196 </web-app>
197 -----ShowAction.java-----
198 package nats;
199 import javax.servlet.http.*;
200 import org.apache.struts2.interceptor.*;
201 import org.apache.struts2.config.Result;
202 import org.apache.struts2.config.Results;
203 import org.apache.struts2.dispatcher.ServletDispatcherResult;
204 import com.opensymphony.xwork2.ActionSupport;
205 import com.opensymphony.xwork2.validator.annotations.*;
206
207 @Results({
208     @Result( name="success", value="/user_success.jsp", type=ServletDispatcherResult.class),
209     @Result( name="input", value="/user.jsp", type=ServletDispatcherResult.class)
210 })
211 public class ShowAction extends ActionSupport implements ServletRequestAware{
212     String name;
213     int age;
214     private HttpServletRequest request;
215
216     public void setServletRequest(HttpServletRequest request){
217         this.request = request;
218     }
219
220     @IntRangeFieldValidator(type = ValidatorType.FIELD,min="10", max="30",
221                             message = "Allowed Age is between 10 and 30.", key="my.err.age.msg")
222     public void setAge(int age) {
223         this.age = age;
224     }
225
226     public int getAge() {
227         return age;
228     }
229
230     @RequiredStringValidator(type = ValidatorType.FIELD,message="name
231 required",key="my.err.name.msg")
232     @EmailValidator(type = ValidatorType.FIELD,message = "invalid email id")
233     @StringLengthFieldValidator(type=ValidatorType.FIELD,
234                                 message="name should contain min of 7 characters and max 10 characters",
235                                 minLength="7", maxLength="10",trim=true)
236     public void setName(String name) {
237         this.name = name;
238     }
239     public String getName() {
240         return name;
241     }

```

```

242 public String execute(){
243     String brname=request.getHeader("user-agent");
244     String msg;
245     if (age>=18)
246     {
247         msg="eligible vote"+brname;
248     }
249     else
250     {
251         msg="not eligible to vote"+brname;
252     }
253     request.setAttribute("data",msg);
254     return "success";
255 }
256 }
257 -----user_success.jsp-----
258 <%@ taglib uri="/struts-tags" prefix="s" %>
259 <html>
260     <head>
261         <title>User Info.</title>
262     </head>
263     <body>
264         <h2>User Information</h2>
265         Your Name: <b><s:property value="name"/></b><br><br>
266         You are <b><s:property value="age"/></b> years old.<br><br>
267         your vote eligibility is <%=request.getAttribute("data") %><br><br>
268         <s:a href="user.jsp">Back</s:a>
269     </body>
270 </html>
271 -----
272
273
274
275
276
277
278
279
280 =====
281 App3(using XWorkValidator )
282 =====
283 -----mystyle.css-----
284 body {
285     font-family: Verdana, Arial, Helvetica, sans-serif; font-size:12px;
286 }
287
288 td {
289     font-family: Verdana, Arial, Helvetica, sans-serif; font-size:12px;
290 }
291
292 .boldred {
293     color: #CB3216;
294     font-size: 10px;
295 }
296 .errorMessage {
297     color: #DB5678;
298     font-size: 10px;
299 }
300 -----login.jsp-----
301 <%@ taglib prefix="s" uri="/struts-tags" %>
302 <html>

```

## STRUTS

DURGASOFT

```
303 <head>
304   <title><s:text name="app.title"/></title>
305   <link rel="stylesheet" href="mystyle.css" type="text/css" />
306 </head>
307 <body>
308   <table align="center" width="300">
309     <tr><td align="center" colspan="2">Login!</td></tr>
310     <tr><td align="center">
311       <s:form action="login" method="post">
312         <s:textfield name="username" key="app.username"/>
313         <s:password name="password" key="app.password"/>
314         <s:submit value="Login"/>
315       </s:form>
316     </td>
317   </tr>
318 </table>
319 </body>
320 </html>
321
322
323
324 -----struts.properties-----
325 struts.custom.i18n.resources=nats.ApplicationResources
326 -----ApplicationResources.properties-----
327 # Resources for parameter 'nats.ApplicationResources'
328 # Project struts2
329 app.username=User Name
330 app.password=Password
331 app.invalid=Invalid User Name or Password.
332 -----web.xml-----
333 <web-app id="WebApp_9" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
334   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
335   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
336   http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
337   <display-name>Struts 2 - Interceptors</display-name>
338   <filter>
339     <filter-name>struts2</filter-name>
340     <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
341   </filter>
342   <filter-mapping>
343     <filter-name>struts2</filter-name>
344     <url-pattern>*</url-pattern>
345   </filter-mapping>
346   <welcome-file-list>
347     <welcome-file>login.jsp</welcome-file>
348   </welcome-file-list>
349 </web-app>
350 -----struts.xml-----
351 <?xml version="1.0" encoding="UTF-8" ?>
352 <!DOCTYPE struts PUBLIC
353 "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
354 "http://struts.apache.org/dtds/struts-2.0.dtd" >
355 <struts>
356   <include file="struts-default.xml"/>
357   <package name="default" extends="struts-default">
358
359     <action name="login" class="nats.LoginAction" >
360       <interceptor-ref name="basicStack"/>
361       <interceptor-ref name="validation"/>
362       <interceptor-ref name="workflow"/>
```

```

363     <result name="success">/login_success.jsp</result>
364     <result name="error">/login_fail.jsp</result>
365     <result name="input">/login.jsp</result>
366   </action>
367 </package>
368 </struts>
369 -----LoginAction-Validation.xml-----
370 <!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
371 "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
372
373 <validators>
374
375   <field name="username">
376     <field-validator type="stringlength">
377       <param name="maxLength">10</param>
378       <param name="minLength">2</param>
379       <param name="trim">true</param>
380       <message>must be a String of a specific greater than 1 less than 10 if specified </message>
381     </field-validator>
382
383   <field-validator type="requiredstring">
384     <message>Enter User Name</message>
385   </field-validator>
386 </field>
387
388 <field name="password">
389   <field-validator type="requiredstring">
390     <message>Enter Password</message>
391   </field-validator>
392 </field>
393
394 </validators>
395 -----LoginAction.java-----
396 package nats;
397
398 import com.opensymphony.xwork2.ActionSupport;
399 public class LoginAction extends ActionSupport {
400
401   private String username;
402   private String password;
403
404   public String getUsername() {
405     return username;
406   }
407   public void setUsername(String username) {
408     System.out.println("setUsername ");
409     this.username = username;
410   }
411
412   public String getPassword() {
413     return password;
414   }
415   public void setPassword(String password) {
416     System.out.println("setPassword");
417     this.password = password;
418   }
419
420   public String execute() throws Exception {
421     System.out.println("Execute method");
422
423

```

## STRUTS

DURGASOFT

```
424     if(username.equals(password))
425         return SUCCESS;
426     else
427         return ERROR;
428
429     }
430
431 }
-----login_success.jsp-----
432 <%@ taglib prefix="s" uri="/struts-tags" %>
434 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
436 <html>
437     <head>
438         <link rel="stylesheet" href="mystyle.css" type="text/css" />
439     </head>
440     <body>
441         <div align="center">
442             <h2>Successfully Logged In!</h2>
443         </body>
444     </html>
445 -----
446 -----login_fail.jsp-----
447 <%@ taglib prefix="s" uri="/struts-tags" %>
448 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
449 <html>
450     <head><title><s:text name="app.title"/></title>
451         <link rel="stylesheet" href="mystyle.css" type="text/css" />
452     </head>
453     <body>
454         <div align="center">
455             <h2>Invalid Credentials</h2>
456         </body>
457     </html>
458
459
460
461
462
463
464
465
466
467 =====
468 App4 (Using tokens mechanism to prevent double posting)
469 =====
470 -----index.jsp-----
471 <%@ taglib uri="/struts-tags" prefix="s" %>
472 <html>
473     <head>
474         <title>Personal Info</title>
475     </head>
476     <body>
477         <h2><s:text name="app.welcome"/></h2>
478         <s:form action="displayAction">
479             <s:token/>
480                 <s:textfield key="app.username" name="username" />
481                 <s:textfield key="app.city" name="city" />
482                 <s:submit key="button.cap"/>
483             </s:form>
484         </body>
```

## STRUTS

DURGASOFT

```
485  <% System.out.println("from Index.jsp" ); %>
486  </html>
487  -----struts.properties-----
488  struts.custom.i18n.resources=p1.ApplicationResources
489  -----ApplicationResources.properties-----
490  # Resources for parameter 'p1.ApplicationResources'
491  # Project strutsproject
492  app.title=Struts 2 Application
493  app.welcome=Personal Information
494  app.username=User Name
495  app.city=City
496  button.cap=send
497  -----struts.xml-----
498  <?xml version="1.0" encoding="UTF-8" ?>
499  <!DOCTYPE struts PUBLIC
500  "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
501  "http://struts.apache.org/dtds/struts-2.0.dtd">
502  <struts>
503  <include file="struts-default.xml" />
504  <package name="default" extends="struts-default">
505
506  <action name="displayAction" class="p1.DisplayAction">
507  <interceptor-ref name="defaultStack"/>
508  <interceptor-ref name="token"/>
509
510  <result name="invalid.token">dblpost.jsp</result>
511  <result name="success">/success.jsp</result>
512  <result name="error">/fail.jsp</result>
513  </action>
514  </package>
515 </struts>
516  -----DisplayAction.java-----
517 package p1;
518
519 public class DisplayAction {
520     private String username;
521     private String city;
522
523     public DisplayAction()
524     {
525         System.out.println("DisplayAction constructor");
526     }
527
528     public String getCity() {
529         System.out.println("getCity ()");
530         return city;
531     }
532     public void setCity(String city) {
533         System.out.println("setCity ()");
534         this.city = city;
535     }
536
537     public String getUsername() {
538         System.out.println("getUsername ()");
539         return username;
540     }
541
542     public void setUsername(String username) {
543         System.out.println("setUsername ()");
544         this.username = username;
545     }
```

```

546
547     public String execute() throws Exception {
548         System.out.println("Execute ()");
549         if("nataraz".equals(username)&& "hyd".equals(city))
550             return "success";
551         else
552             return "fail";
553     }
554 }
555 -----success.jsp-----
556 <%@ taglib uri="/struts-tags" prefix="s" %>
557 <html>
558     <head>
559         <title>Personal Info.</title>
560     </head>
561     <body>
562         <b> valid Credentials </b>
563         <% System.out.println("from success.jsp" ); %>
564         <s:a href="index.jsp">Back</s:a>
565     </body>
566 </html>
567 -----fail.jsp-----
568 <%@ taglib uri="/struts-tags" prefix="s" %>
569 <html>
570     <head>
571         <title>Personal Info.</title>
572     </head>
573     <body>
574         <b> Invalid Credentials </b>
575         <% System.out.println("from failure.jsp" ); %>
576         <s:a href="index.jsp">Back</s:a>
577     </body>
578 </html>
579 -----dblpost.jsp-----
580 <b><font color=red>Double posting is not allowed</font></b>
581 -----
582 =====
583 App5(Example on execAndWait Interceptor)
584 =====
585 -----index.jsp-----
586 <%@ page contentType="text/html;charset=UTF-8" language="java" %>
587 <%@ taglib uri="/struts-tags" prefix="s" %>
588 <html>
589     <head>
590         <title>Personal Info</title>
591     </head>
592     <body>
593         <h2><s:text name="app.welcome"/></h2>
594         <s:form action="displayAction">
595             <s:textfield key="app.username" name="username" />
596             <s:textfield key="app.city" name="city" />
597             <s:submit key="button.cap"/>
598         </s:form>
599     </body>
600
601     <% System.out.println("from Index.jsp" ); %>
602 </html>
603 -----wait.jsp-----
604 <html>
605     <head>
606         <META HTTP-EQUIV="Refresh" CONTENT="4">

```

```

607 </head>
608 <body>
609 <table>
610   <tr>
611     <td width="150">&nbsp;</td>
612     <td bgcolor="#9f1704" height="20">
613       <font color="white"><strong>L o a d i n g . . .</strong></font>
614     </td>
615   </tr>
616   <tr><td>From Wait.jsp</td></tr>
617 </table>
618 </body>
619 </html>
620 -----struts.properties-----
621 struts.custom.i18n.resources=p1.ApplicationResources
622 -----ApplicationResources.properties-----
623 # Resources for parameter 'p1.ApplicationResources'
624 # Project strutsproject
625 app.title=Struts 2 Application
626 app.welcome=Personal Information
627 app.username=User Name
628 app.city=City
629 button.cap=Register
630 -----struts.xml-----
631 <!DOCTYPE struts PUBLIC
632 "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
633 "http://struts.apache.org/dtds/struts-2.0.dtd ">
634 <struts>
635   <include file="struts-default.xml" />
636   <package name="default" extends="struts-default">
637     <action name="displayAction" class="p1.DisplayAction" method="abc">
638       <interceptor-ref name="completeStack"/>
639       <interceptor-ref name="execAndWait">
640         <param name="delay">2000</param>
641         <param name="delaySleepInterval">50</param>
642       </interceptor-ref>
643
644       <result name="success">/success.jsp</result>
645       <result name="failure">/fail.jsp</result>
646       <result name="wait">/wait.jsp</result>
647
648     </action>
649   </package>
650 </struts>
651 -----DisplayAction.java-----
652 package p1;
653
654 public class DisplayAction {
655   private String username;
656   private String city;
657
658   public DisplayAction() {
659     System.out.println("DisplayAction constructor");
660   }
661   public String getCity() {
662     System.out.println("getCity ()");
663     return city;
664   }
665 }
666
667 }

```

```

668 public void setCity(String city) {
669     System.out.println("setCity ()");
670     this.city = city;
671 }
672 public String getUsername() {
673     System.out.println("getUsername ()");
674     return username;
675 }
676
677 public void setUsername(String username) {
678     System.out.println("setUsername ()");
679     this.username = username;
680 }
681
682 public String execute() throws Exception {
683     System.out.println("Execute ()");
684     if("nataraz".equals(username)&& "hyd".equals(city))
685         return "success";
686     else
687         return "fail";
688 }
689 }
-----success.jsp-----
690 <%@ taglib uri="/struts-tags" prefix="s" %>
691 <html>
692 <head>
693 <b> valid Credentials </b>
694 <% System.out.println("from success.jsp" ); %>
695 <s:a href="index.jsp">Back</s:a>
696 </body>
697 </html>
698 -----fail.jsp-----
699 <%@ taglib uri="/struts-tags" prefix="s" %>
700 <html>
701 <head>
702 <b> Invalid Credentials </b>
703 <% System.out.println("from failure.jsp" ); %>
704 <s:a href="index.jsp">Back</s:a>
705 </body>
706 </html>
707 -----=====
708 App6) Application on file uploading
709 =====
710 -----index.jsp-----
711 <%@taglib uri="/struts-tags" prefix="s" %>
712 <s:form action="upl" method="post" enctype="multipart/form-data" >
713     <s:file name="file1" label="Select a file :" />
714     <s:submit />
715 </s:form>
716 -----web.xml-----
717 <web-app>
718 <filter>
719     <filter-name>struts2</filter-name>
720     <filter-class>
721
722
723
724
725
726
727
728

```

```
org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>
729 </filter>
730 <filter-mapping>
731   <filter-name>struts2</filter-name>
732   <url-pattern>/*</url-pattern>
733 </filter-mapping>
734 </web-app>
735 -----struts.xml-----
736 <!DOCTYPE struts PUBLIC
737 "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
738 "http://struts.apache.org/dtds/struts-2.0.dtd">
739 <struts>
740   <include file="struts-default.xml"/>
741   <package name="p1" extends="struts-default">
742
743     <action name="upl" class="p1.FileUploadAction">
744       <interceptor-ref name="completeStack"/>
745       <interceptor-ref name="fileUpload"/>
746
747       <result name="input"/>/index.jsp</result>
748       <result name="success"/>/success.jsp</result>
749     </action>
750
751   </package>
752 </struts>
753 -----FileUploadAction.java-----
754 package p1;
755 import java.io.File;
756 import com.opensymphony.xwork2.ActionSupport;
757 import org.apache.commons.io.*;
758 import javax.servlet.*;
759 import org.apache.struts2.util.*;
760
761 public class FileUploadAction extends ActionSupport implements ServletContextAware{
762   // file1 properties
763   private File file1;
764   private String file1ContentType;
765   private String file1FileName;
766
767   //write getXxx() and setXxx(-) methods for the above properties
768   public File getFile1() {
769     return file1;
770   }
771
772   public void setFile1(File file1) {
773     this.file1 = file1;
774   }
775
776   public String getFile1ContentType() {
777     return file1ContentType;
778   }
779
780   public void setFile1ContentType(String file1ContentType) {
781     this.file1ContentType = file1ContentType;
782   }
783
784   public String getFile1FileName() {
785     return file1FileName;
786   }
787
788   public void setFile1FileName(String file1FileName) {
```

```

789     this.file1FileName = file1FileName;
790 }
791
792 // Interface Injection logic
793 ServletContext sc;
794 public void setServletContext(ServletContext sc)
795 {
796     this.sc=sc;
797 }
798
799 // B.method to complete file uploading
800 public String execute()
801 {
802     try
803     {
804         String dpath=sc.getRealPath("/");
805         File file2=new File(dpath,file1FileName);
806         FileUtils.copyFile(file1,file2);
807         return SUCCESS;
808     }
809     catch(Exception e) { return INPUT; }
810 } //execute
811 } //class
812 -----success.jsp-----
813 <%@taglib uri="/struts-tags" prefix="s"%>
814 <body>
815 You have uploaded the following file.
816 <br>
817 File Name : <s:property value="file1FileName"/> <br>
818 Content Type :<s:property value="file1ContentType"/>
819 </body>
820 -----
821 =====
822 App7 (application to download files)
823 =====
824 -----index.jsp-----
825 <%@ taglib prefix="s" uri="/struts-tags" %>
826 <body>
827     <s:url id="url" action="download2"/>
828     <s:a href="#">Download ZIP file.

```

```
849  </welcome-file-list>
850  </web-app>
851  -----struts.xml-----
852  <!DOCTYPE struts PUBLIC
853  "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
854  "http://struts.apache.org/dtds/struts-2.0.dtd ">
855  <struts>
856  <include file="struts-default.xml" />
857  <package name="default" extends="struts-default">
858  <action name="download2" class="FileDownloadAction">
859  <param name="inputPath">/images/Bookmark.doc</param>
860
861  <result name="success" type="stream">
862  <param name="contentType">application/msword</param>
863  <param name="inputName">InputStream</param>
864  <param name="contentDisposition">filename="Bookmark.doc"</param>
865  <param name="bufferSize">4096</param>
866  </result>
867
868  </action>
869  </package>
870 </struts>
871 -----FileDownloadAction.java-----
872 import java.io.InputStream;
873 import org.apache.struts2.ServletActionContext;
874 import com.opensymphony.xwork2.Action;
875
876 public class FileDownloadAction implements Action {
877
878     private String inputPath;
879     public void setInputPath(String value) {
880         inputPath = value;
881     }
882
883     public InputStream getInputStream() throws Exception {
884         return ServletActionContext.getServletContext().getResourceAsStream(inputPath);
885     }
886
887     public String execute() throws Exception {
888         return SUCCESS;
889     }
890 }
891 -----
892 -----
893
894
895
896
897
898
899
900 =====
901 App8(user-defined Interceptor)
902 =====
903 -----index.jsp-----
904 <%@ taglib prefix="s" uri="/struts-tags" %>
905 <body>
906     <s:url id="url" action="download2"/>
907     <s:a href="#">Download ZIP file.</s:a>
908 </body>
909
```

```

910 -----web.xml-----
911 <web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
912 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
913 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
914 http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
915   <display-name>Struts 2 Application</display-name>
916   <filter>
917     <filter-name>struts2</filter-name>
918     <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
919   </filter>
920   <filter-mapping>
921     <filter-name>struts2</filter-name>
922     <url-pattern>*</url-pattern>
923   </filter-mapping>
924   <welcome-file-list>
925     <welcome-file>index.jsp</welcome-file>
926   </welcome-file-list>
927 </web-app>
928 -----struts.xml-----
929 <!DOCTYPE struts PUBLIC
930 "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
931 "http://struts.apache.org/dtds/struts-2.0.dtd ">
932 <struts>
933   <include file="struts-default.xml" />
934   <package name="default" extends="struts-default">
935     <interceptors>
936       <interceptor name="my_intercept" class="p1.MyInterceptor" />
937     </interceptors>
938
939     <action name="download2" class="FileDownloadAction">
940       <interceptor-ref name="defaultStack"/>
941       <interceptor-ref name="my_intercept"/>
942       <param name="inputPath">/images/struts-gif.zip</param>
943
944       <result name="success" type="stream">
945         <param name="contentType">application/zip</param>
946         <param name="inputName">inputStream</param>
947         <param name="contentDisposition">filename="struts-gif.zip"</param>
948         <param name="bufferSize">4096</param>
949       </result>
950
951       <result name="login">/index.jsp</result>
952     </action>
953   </package>
954 </struts>
955 -----MyInterceptor.java-----
956 package p1;
957 import javax.servlet.http.*;
958 import com.opensymphony.xwork2.*;
959 import com.opensymphony.xwork2.interceptor.*;
960 import org.apache.struts2.*;
961
962 public class MyInterceptor extends AbstractInterceptor
963 {
964   public String intercept (ActionInvocation invocation) throws Exception
965   {
966     System.out.println("MyInterceptor class");
967     final ActionContext context=invocation.getInvocationContext();
968
969     HttpServletRequest request=(HttpServletRequest)

```

```

    context.get(StrutsStatics.HTTP_REQUEST);
970   String brname=request.getHeader("user-agent");
971   int n=brname.indexOf("MSIE");
972   if(n== -1)
973     return "login";
974   else
975     return invocation.invoke();
976 }
977 }
978 }-----FileDownloadAction.java-----
980 import java.io.InputStream;
981 import org.apache.struts2.ServletActionContext;
982 import com.opensymphony.xwork2.Action;
983
984 public class FileDownloadAction implements Action {
985
986   private String inputPath;
987   public void setInputPath(String value) {
988     inputPath = value;
989   }
990
991   public InputStream getInputStream() throws Exception {
992     return ServletActionContext.getServletContext().getResourceAsStream(inputPath);
993   }
994
995   public String execute() throws Exception {
996     return SUCCESS;
997   }
998
999 }
1000 -----App9 (on 118n)-----
1001 =====
1002 =====
1003 =====
1004 -----global.properties-----
1005 #Global messages
1006 global.username = Username
1007 global.password = Password
1008 global.submit = Submit
1009 -----global_de.properties-----
1010 #Global messages
1011 global.username = Benutzername
1012 global.password = Kennwort
1013 global.submit = Einreichen
1014 -----global_fr.properties-----
1015 #Global messages
1016 global.username = Nom d'utilisateur
1017 global.password = Mot de passe
1018 global.submit = Soumettre
1019 -----global_zh_CN.properties-----
1020 #Global messages
1021 global.username = \u7528\u6237\u540d
1022 global.password = \u5bc6\u7801
1023 global.submit=\u63d0\u4ea4
1024 -----login.jsp-----
1025 <%@ page contentType="text/html;charset=UTF-8" %>
1026 <%@ taglib prefix="s" uri="/struts-tags" %>
1027 <html>
1028 <head>
1029 </head>

```

## STRUTS

DURGASOFT

```
1030  
1031 <body>  
1032 <h1>Struts 2 localization example</h1>  
1033  
1034 <s:form action="verify">  
1035   <s:textfield key="global.username" name="uname" />  
1036   <s:password key="global.password" name="pwd"/>  
1037   <s:submit key="global.submit" name="submit" />  
1038 </s:form>  
1039  
1040 <s:url id="localeEN" action="locale" >  
1041   <s:param name="request_locale" >en</s:param>  
1042 </s:url>  
1043  
1044 <s:url id="localezhCN" action="locale" >  
1045   <s:param name="request_locale" >zh_CN</s:param>  
1046 </s:url>  
1047  
1048 <s:url id="localeDE" action="locale" >  
1049   <s:param name="request_locale" >de</s:param>  
1050 </s:url>  
1051  
1052 <s:url id="localeFR" action="locale" >  
1053   <s:param name="request_locale" >fr</s:param>  
1054 </s:url>  
1055  
1056 <s:a href="#">  
1057   href="#">  
1058   href="#">  
1059   href="#">  
1060  
1061 </body>  
1062 </html>  
1063 -----web.xml-----  
1064 <!-- web.xml -->  
1065 <web-app>  
1066   <filter>  
1067     <filter-name> struts2 </filter-name>  
1068     <filter-class> org.apache.struts2.dispatcher.FilterDispatcher</filter-class>  
1069   </filter>  
1070   <filter-mapping>  
1071     <filter-name> struts2 </filter-name>  
1072     <url-pattern>/*</url-pattern>  
1073   </filter-mapping>  
1074 </web-app>  
1075 -----struts.xml-----  
1076 <?xml version="1.0" encoding="UTF-8" ?>  
1077 <!DOCTYPE struts PUBLIC  
1078 "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"  
1079 "http://struts.apache.org/dtds/struts-2.0.dtd">  
1080 <struts>  
1081   <constant name="struts.custom.i18n.resources" value="global" />  
1082   <constant name="struts.devMode" value="true" />  
1083   <include file="struts-default.xml"/>  
1084   <package name="pack1" extends="struts-default">  
1085     <action name="verify" class="LoginAction">  
1086       <result name="success"> /success.jsp </result>  
1087       <result name="error"> /failure.jsp </result>  
1088     </action>  
1089     <action name="locale" class="LocaleAction">  
1090       <result>/login.jsp</result>
```

```
1091     </action>
1092   </package>
1093 </struts>
1094 -----LoginAction.java-----
1095 //LoginAction.java
1096 import com.opensymphony.xwork2.ActionSupport;
1097
1098 public class LoginAction extends ActionSupport
1099 {
1100     private String uname, pwd;
1101
1102     public String getPwd() {
1103         return pwd;
1104     }
1105
1106     public void setPwd(String pwd) {
1107         this.pwd = pwd;
1108     }
1109
1110     public String getUserName() {
1111         return uname;
1112     }
1113
1114     public void setUserName(String uname) {
1115         this.uname = uname;
1116     }
1117     public String execute() {
1118         if(uname.equals("durga") && pwd.equals("struts2"))
1119             return SUCCESS;
1120         else
1121             return ERROR;
1122     }
1123 }
1124 -----
1125 -----success.jsp-----
1126 <h2>
1127 <%@ taglib prefix="s" uri="/struts-tags"%>
1128 U entered Username <s:property value="uname"/> <br>
1129 U entered Password <s:property value="pwd"/> <br>
1130 Login Success
1131 </h2>
1132 -----failure.jsp-----
1133 <%-- failure.jsp --%>
1134 <h2>
1135 <%@ taglib prefix="s" uri="/struts-tags"%>
1136 U entered Username <s:property value="uname"/> <br>
1137 U entered Password <s:property value="pwd"/> <br>
1138 Login Failed
1139 </h2>
1140 -----
1141 =====
1142 App10 >>>>>>Example Application on Struts 2 Tiles >>>>>>
1143 -----index.jsp-----
1144 <%-- index.jsp --%>
1145 <%
1146     response.sendRedirect("welcomeLink.action");
1147 %>
1148 -----welcome.jsp-----
1149 <html>
1150 <head>
1151 </head>
```

```

1152 <body>
1153 Welcome Guest.
1154 </body>
1155 </html>
1156 -----header.jsp-----
1157 <div align="center" style="font-weight:bold">Sathya Technologies</div>
1158 -----footer.jsp-----
1159 <div align="center">&copy; copy rights reserved</div>
1160 -----baseLayout.jsp-----
1161 <%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles" %>
1162 <html>
1163 <head>
1164   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
1165   <title><tiles:insertAttribute name="title" ignore="true" /></title>
1166 </head>
1167 <body>
1168 <table border="1" cellpadding="2" cellspacing="2" align="center">
1169   <tr>
1170     <td height="30" colspan="2">
1171       <tiles:insertAttribute name="header" />
1172     </td>
1173   </tr>
1174   <tr>
1175     <td height="250">
1176       <tiles:insertAttribute name="menu" />
1177     </td>
1178     <td width="350">
1179       <tiles:insertAttribute name="body" />
1180     </td>
1181   </tr>
1182   <tr>
1183     <td height="30" colspan="2">
1184       <tiles:insertAttribute name="footer" />
1185     </td>
1186   </tr>
1187 </table>
1188 </body>
1189 </html>
1190 -----body.jsp-----
1191 <p> sample body content.</p>
1192 -----menu.jsp-----
1193 <%@taglib uri="/struts-tags" prefix="s"%>
1194
1195 <a href="

```

## STRUTS

DURGASOFT

```
1213     STRUTS<br>
1214     SPRING<br>
1215     HIBERNATE<br>
1216     </body>
1217 </html>
1218 -----web.xml-----
1219 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1220   xmlns="http://java.sun.com/xml/ns/javaee"
1221   xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
1222   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
1223   http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
1224   id="WebApp_ID" version="2.5">
1225     <listener>
1226       <listener-class>org.apache.struts2.tiles.StrutsTilesListener</listener-class>
1227     </listener>
1228     <filter>
1229       <filter-name>struts2</filter-name>
1230       <filter-class>
1231         org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
1232       </filter-class>
1233     </filter>
1234     <filter-mapping>
1235       <filter-name>struts2</filter-name>
1236       <url-pattern>/*</url-pattern>
1237     </filter-mapping>
1238
1239     <welcome-file-list>
1240       <welcome-file>index.jsp</welcome-file>
1241     </welcome-file-list>
1242   </web-app>
1243
1244 -----struts.xml-----
1245 <!DOCTYPE struts PUBLIC
1246   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
1247   "http://struts.apache.org/dtds/struts-2.0.dtd">
1248
1249 <struts>
1250   <package name="default" extends="tiles-default">
1251     <action name="*Link" method="{1}" class="LinkAction">
1252       <result name="welcome" type="tiles">welcome</result>
1253       <result name="friends" type="tiles">friends</result>
1254       <result name="office" type="tiles">office</result>
1255     </action>
1256   </package>
1257 </struts>
1258 -----tiles.xml-----
1259 <!DOCTYPE tiles-definitions PUBLIC
1260   "-//Apache Software Foundation//DTD Tiles Configuration 2.0//EN"
1261   "http://tiles.apache.org/dtds/tiles-config_2_0.dtd">
1262
1263 <tiles-definitions>
1264
1265   <definition name="baseLayout" template="/baseLayout.jsp">
1266     <put-attribute name="title" value="Template"/>
1267     <put-attribute name="header" value="/header.jsp"/>
1268     <put-attribute name="menu" value="/menu.jsp"/>
1269     <put-attribute name="body" value="/body.jsp"/>
1270     <put-attribute name="footer" value="/footer.jsp"/>
1271   </definition>
```

```
1272
1273 <definition name="welcome" extends="baseLayout">
1274   <put-attribute name="title" value="Welcome"/>
1275   <put-attribute name="body" value="/welcome.jsp"/>
1276 </definition>
1277
1278 <definition name="friends" extends="baseLayout">
1279   <put-attribute name="title" value="Friends"/>
1280   <put-attribute name="body" value="/friends.jsp"/>
1281 </definition>
1282
1283 <definition name="office" extends="baseLayout">
1284   <put-attribute name="title" value="Office"/>
1285   <put-attribute name="body" value="/office.jsp"/>
1286 </definition>
1287 </tiles-definitions>
1288 -----LinkAction.java-----
1289
1290 import com.opensymphony.xwork2.ActionSupport;
1291
1292 public class LinkAction extends ActionSupport {
1293   public String welcome()
1294   {
1295     return "welcome";
1296   }
1297
1298   public String friends()
1299   {
1300     return "friends";
1301   }
1302
1303   public String office()
1304   {
1305     return "office";
1306   }
1307 }
1308
1309
```

```

1 >>>>>>> programs on javascript>>>>>>>>>
2 =====App1.html=====
3 <HTML>
4 <HEAD>
5 <TITLE>
6 Checking Your Browser Type
7 </TITLE>
8 </HEAD>
9 <BODY>
10 <SCRIPT LANGUAGE = JAVASCRIPT>
11 if(navigator.appName == "Microsoft Internet Explorer")
12 {
13 document.write("<B><CENTER>")
14 document.write("you have Microsoft Internet Explorer" +navigator.appVersion)
15 document.write("</B></CENTER>")
16 }
17 if(navigator.appName == "Netscape")
18 {
19 document.write("<B><CENTER>")
20 document.write("you have Netscape Navigaor"+navigaor.appVrsion)
21 document.write("</B></CENTER>")
22 }
23 </SCRIPT>
24 <CENTER>
25 <H1>
26 Checking You Browser Type
27 </H1>
28 </CENTER>
29 </BODY>
30 </HTML>
31 =====App2.html=====
32 <HTML>
33 <HEAD>
34 <TITLE> Using The history object </TITLE>
35 <SCRIPT LANGUAGE = JavaScript>
36 function goBack()
37 {
38 window.history.back()
39 }
40 function goForward()
41 {
42 window.history.forward()
43 }
44 function goBackTwo()
45 {
46 window.history.go(-2)
47 }
48 function goForwardTwo()
49 {
50 window.history.go(2)
51 }
52 </SCRIPT>
53 </HEAD>
54 <BODY>
55 <CENTER>
56 <H1>Using The history Object</H1>
57 <FORM>
58 <BR>
59 Navigate using the history object!
60 <BR>
61 <BR>

```

```

62      <INPUT TYPE =BUTTON Value = "<Back One Page" onClick ="goBack()">
63      <INPUT TYPE = BUTTON Value ="Forward One Page >" onClick ="goForward()">
64      <BR>
65      <BR>
66      <INPUT TYPE=BUTTON Value ="><< Back Two Pages" onClick ="goBackTwo()">
67      <INPUT TYPE = BUTTON Value ="Forward Two Pages >>" onClick ="goForwardTwo()">
68      </FORM>
69      </CENTER>
70      </BODY>
71      </HTML>
72 =====App3.html=====
73 <HTML>
74   <HEAD>
75     <TITLE> Using Dynamic Styles </TITLE>
76   </HEAD>
77
78   <BODY>
79     <CENTER>
80       <H1>Using Dynamic Styles.</H1>
81       <SPAN onmouseover="this.style.fontSize ='48'">
82         This text gets bigger when you move the mouse over it!
83     </SPAN>
84   </CENTER>
85   </BODY>
86 </HTML>
87 =====App4.html=====
88 <HTML>
89 <HEAD>
90   <TITLE> Changing HTML on the fly</TITLE>
91   <SCRIPT LANGUAGE ="JavaScript">
92     function changeHeader()
93     {
94       HID.innerText = "This is the new header"
95     }
96   </SCRIPT>
97 </HEAD>
98 <BODY>
99   <CENTER>
100    <H1 ID=HID onClick = "changeHeader()">Dynamic HTML</H1>
101    Click the above header to chage it.....
102  </CENTER>
103 </BODY>
104 </HTML>
105 =====App5.html=====
106 <HTML>
107 <TITLE>Changing HTML on the fly</TITLE>
108   <HEAD>
109   <SCRIPT LANGUAGE ="JavaScript">
110     function changeHeader()
111     {
112       Header.innerHTML ="<MARQUEE>This marquee was just created!</MARQUEE>"
113     }
114   </SCRIPT>
115 </HEAD>
116 <BODY>
117   <CENTER>
118    <H1 ID=Header onClick = "changeHeader()">Dynamic HTM</H1>
119    Click the above header to change it.....
120  </CENTER>
121 </BODY>
122 </HTML>

```

## STRUTS

## DURGASOFT

```
123 =====App6.html=====
124 <HTML>
125 <HEAD>
126 <TITLE> Using createElement To Create New Elements</TITLE>
127 <SCRIPT LANGUAGE="JavaScript">
128 function showMore()
129 {
130 var newTextfield, newText;
131
132 newTextfield = document.createElement("INPUT");
133 newTextfield.type ="TEXT";
134 newTextfield.value ="Hello!"
135
136 newText = document.createTextNode("A new textfield: ");
137
138 myId.insertBefore(newText);
139 myId.insertBefore(newTextfield);
140 }
141 </SCRIPT>
142 </HEAD>
143 <BODY>
144 <CENTER>
145 <H1>Using createElement To Create New Elements</H1>
146 </CENTER>
147 <DIV ID=myId>
148 <INPUT TYPE=BUTTON VALUE="Click Me" ONCLICK="showMore()">
149 </DIV>
150 </BODY>
151 </HTML>
152
```

## What's AJAX?

Ajax stands for asynchronous JavaScript and XML. In a nutshell it is the use of the nonstandard XMLHttpRequest object to communicate with server-side scripts. It can send as well receive information in a variety of formats, including XML, HTML, and even text files. Ajax's most appealing characteristic, however, is "its "asynchronous" nature, which means it can do all of this without having to refresh the page. This allows you to update portions of a page based upon user events.

The two features in question are that you can:

- Make requests to the server without reloading the page
- Parse and work with XML document

### Step 1 - say "Please!" or How to Make an HTTP Request

In order to make an HTTP request to the server using "JavaScript", you need an instance of a class that provides you this functionality. Such a class was originally introduced in Internet Explorer as an ActiveX object, called XMLHttpRequest. Then Mozilla, Safari and other browsers followed, implementing an XMLHttpRequest class that supports the methods and properties of Microsoft's original ActiveX object.

```
if (window.XMLHttpRequest) { // Mozilla, Safari, ...
    http_request = new XMLHttpRequest();
} else if (window.ActiveXObject) { // IE
    http_request = new ActiveXObject('Microsoft.XMLHTTP');
}
```

Some versions of some Mozilla browsers won't work properly if the response from the server doesn't have an XML mime-type header. To satisfy this, you can use an extra method call to override the header sent by the server, just in case it's not text/xml.

```
http_request = new XMLHttpRequest();
http_request.overrideMimeType('text/xml');
```

Next, you need to decide what you want to do after you receive the server response to your request. At this stage, you just need to tell the HTTP request object which JavaScript function will do the work of processing the response. This is done by setting the onreadystatechange property of the object to the name of the JavaScript function you plan to use, like this:

```
http_request.onreadystatechange = nameOfTheFunction;
```

Note that there are no brackets after the function name and no parameters passed, because you're simply assigning a reference to the function, rather than actually calling it. Also, instead of giving a function name, you can use the JavaScript technique of defining -functions on the fly (called "anonymous function") and define the actions that will process the response right away, like this:

```
http_request.onreadystatechange = function() {
    // do the thing
};
```

Next, after you've declared what will happen as soon as you receive the response, you need to actually make the request. You need to call the open() and send() methods of the HTTP request class, like this:

```
http_request.open('GET', 'http://www.example.org/some.file', true);
http_request.send(null);
```

## STRUCTS

## DURGASOFT

- The first parameter of the call to open() is the HTTP request method - GET, POST, HEAD or any other method you want to use and that is supported by your server. Keep the method capitalized as per the HTTP standard; otherwise some browsers (like Firefox) might not process the request. For more information on the possible HTTP request methods you can check the W3C specs.
- The second parameter is the URL of the page you're requesting. As a security feature, you cannot call pages on 3rd-party domains. Be sure to use the exact domain name on all of your pages or you will get a 'permission denied' error when you call open(). A common pitfall is accessing your site by domain.tld, but attempting to call pages with www.domain.tld.
- The third parameter sets whether the request is asynchronous. If TRUE, the execution of the JavaScript function will continue while the response of the server has not yet arrived. This is the A in AJAX.

The parameter to the send() method can be any data you want to send to the server if POST-ing the request. The data should be in the form of a query string, like:

name=value&anothername=othervalue&so=on

Note that if you want to POST data, you have to change the MIME type of the request using the following line:

```
http_request.setRequestHeader('ContentType','application/xwwwformurlencoded');
```

Otherwise, the server will discard the POSTed data.

### Step 2 - "There you go!", or Handling the Server Response

Remember that when you were sending the request, you provided the name of a JavaScript function that is designed to handle the response.

```
http_request.onreadystatechange = nameOfTheFunction;
```

Let's see what this function should do. First, the function needs to check for the state of the request. If the state has the value of 4, that means that the full server response is received and it's OK for you to continue processing it.

```
if (http_request.readyState == 4) {  
    // everything is good, the response is received  
} else {  
    // still not ready  
}
```

The full list of the readyState values is as follows:

- 0 (uninitialized)
- 1 (loading)
- 2 (loaded)
- 3 (interactive)
- 4 (complete)

The next thing to check is the status code of the HTTP server response. All the possible codes are listed on the W3C site. For our purposes we are only interested in 200 OK response.

```
if (http_request.status == 200) {  
    // perfect!  
} else {
```

## STRUCTS

DURGASOFT

```
// there was a problem with the request, // for example the response may be a 404  
(Not Found)  
// or 500 (Internal Server Error) response codes  
}
```

Now after you've checked the state of the request and the HTTP status code of the response, it's up to you to do whatever you want with the data the server has sent to you. You have two options to access that data:

- `http_request.responseText` - will return the server's response as a string of text
- `http_request.responseXML` - will return the response as an XML Document object you can traverse using the JavaScript DOM functions

### Step 3 - "All together now!" - A Simple Example

Let's put it all together and do a simple HTTP request. Our JavaScript will request an HTML document, test.html, which contains the text "I'm a test." and then we'll alert() the contents of the test.html file.

```
<script type="text/javascript" language = "javascript">  
Function makeRequest(url) {  
var http_request = false;  
if (window.XMLHttpRequest) { // Mozilla, Safari,...  
http_request = new XMLHttpRequest();  
if (http_request.overrideMimeType) {  
http_request.overrideMimeType('text/xml');  
// See note below about this line  
}  
}  
} else if (window.ActiveXObject) { // IE  
try{...  
http_request = new ActiveXObject("Msxml2.XMLHTTP");  
} catch (e) {  
try {  
http_request = new ActiveXObject("Microsoft.XMLHTTP");  
} catch (e) {}  
}  
}  
  
if (!http_request) {  
alert('Giving up :( Cannot create an XMLHttpRequest');  
return false;  
}  
http_request.onreadystatechange = function() { alertContents(http_request); };  
http_request.open('GET',url,true);  
http_request.send( null);}  
  
function alertContents(http_request) {  
if (http_request.readyState == 4)  
{ if (http_request.status == 200) {  
alert(http_request.responseText);
```

```

) else {
    alert('There was a problem with the request.');
}
}
}
</script>
<span
style="cursor: pointer; text-decoration: underline"
onclick="makeRequest('test.html')>
Make a request
</span>

```

In this example:

- The user clicks the link "Make a request" in the browser;
- This calls the makeRequest() function with a parameter - the name test.html of an HTML file in the same directory;
- The request is made and then (onreadystatechange) the execution is passed to alertContents();
- alertContents() checks if the response was received and it's an OK and then alert()s the contents of the test.html file.

**Note 1:** The line http\_request.overrideMimeType('text/xml'); above will cause JavaScript Console errors in Firefox 1.5 or later, as documented at

<https://bugzilla.mozilla.org/showbugaj2id=111721> if the page called by XMLHttpRequest is not valid XML (eg, if it is plain text). This is actually correct behavior; this article will be revised soon to address this change.

**Note 2:** if you are sending a request to a piece of code that will return XML, rather than to a static XML file, you must set some response headers if your page is to work in Internet Explorer in addition to Mozilla. If you do not set header Content-Type: application/xml, IE will throw a JavaScript error, 'Object Expected', after the line where you try to access an XML element. If you do not set header Cache-Control: no-cache the browser will cache the response and never resubmit the request, making debugging "challenging."

**Note 3:** if the http request variable is used globally, competing functions calling makeRequest() may overwrite each other, causing a race condition. Declaring the http\_request variable local to the function and passing it to the alertContent() function prevents the race condition

**Note 4:** To register the callback function onreadystatechange, you cannot have any arguments. That's why the following code does not work:

```
http_request.onreadystatechange = alertContents(http_request); // (does not work)
```

So, to register the function successfully, you can either pass the arguments indirectly via the anonymous function or use http\_request as a global variable. Examples follow:

```
http_request.onreadystatechange = function() { alertContents(http_request); };
//1 (simultaneous request)

http_request.onreadystatechange = alertContents; //2 (global variable)
```

Method 1 lets you have several requests processed simultaneously, while method 2 is used if http request is a global variable.

**Note 5:** In the event of a communication error (such as the Webserver going down), an exception will be thrown in the onreadystatechange method when attempting to access the .status variable. Make sure that you wrap you if...then statement in a try...catches.

```

functionalertContents(http_request){
try{
if (http_request.readyState == 4) {
if (http_request.status == 200) {
alert(http_request.resp)
} else {
alert('There was a problem with the request.');
}
}
}

catch( e ) {
alert('Caught Exception: ' + e.description);
}
}

```

#### **Step 4 - "The X-Files" or Working with the XML Response**

In the previous example, after the response to the HTTP request was received we used the responseText property of the request object, which contained the contents of the test.html file. Now let's try the responseXML property.

First off, let's create a valid XML document that we'll request later on. The document(test.xml) contains the following-

```

<?xml version = "1.0" ?>
<root>
? I'm a test
</root>

```

In the script we only need to change the request line to:

```

.....
onclick="makeRequest('test.xml')"

.....

```

Then in alertContents(), we need to replace the line alert(http\_request.responseText); with:

```

var xmldoc = http_request.responseXML;
var root_node = xmldoc.getElementsByTagName('root').item(0);
alert(root_node.firstChild.data);

```

This code takes the XMLHttpRequest object given by responseXML and uses DOM methods to access some of the data contained in the XML document

#### Ajax: A New Approach to Web Applications

Take a look at Google Suggest. Watch the way the suggested terms update as you type almost instantly. Now look at Google Maps. Zoom in Use your cursor to grab the map and scroll around a bit. Again, everything happens almost instantly, with no waiting for pages to reload. Google Suggest and Google Maps are two examples of a new approach to web applications that we at Adaptive Path have been calling Ajax. The name is shorthand for Asynchronous JavaScript + XML, and it represents a fundamental shift in what's possible on the Web.

### Defining Ajax

Ajax isn't a technology. It's really several technologies, each flourishing in its own right, coming together in powerful new ways. Ajax incorporates:

- Standards-based presentation using XHTML and CSS;
- Dynamic display and interaction using the Document Object Model
- Data interchange and manipulation using XML and XSLT
- asynchronous data retrieval using XMLHttpRequest
- and JavaScript binding everything together.

The classic web application model works like this: Most user actions in the interface trigger an HTTP request back to a web server. The server does some processing — retrieving data, crunching numbers, talking to various legacy systems — and then returns an HTML page to the client. It's a model adapted from the Web's original use as a hypertext medium, but as fans of the Elements of User Experience know, what makes the Web good for hypertext doesn't necessarily make it good for software applications.

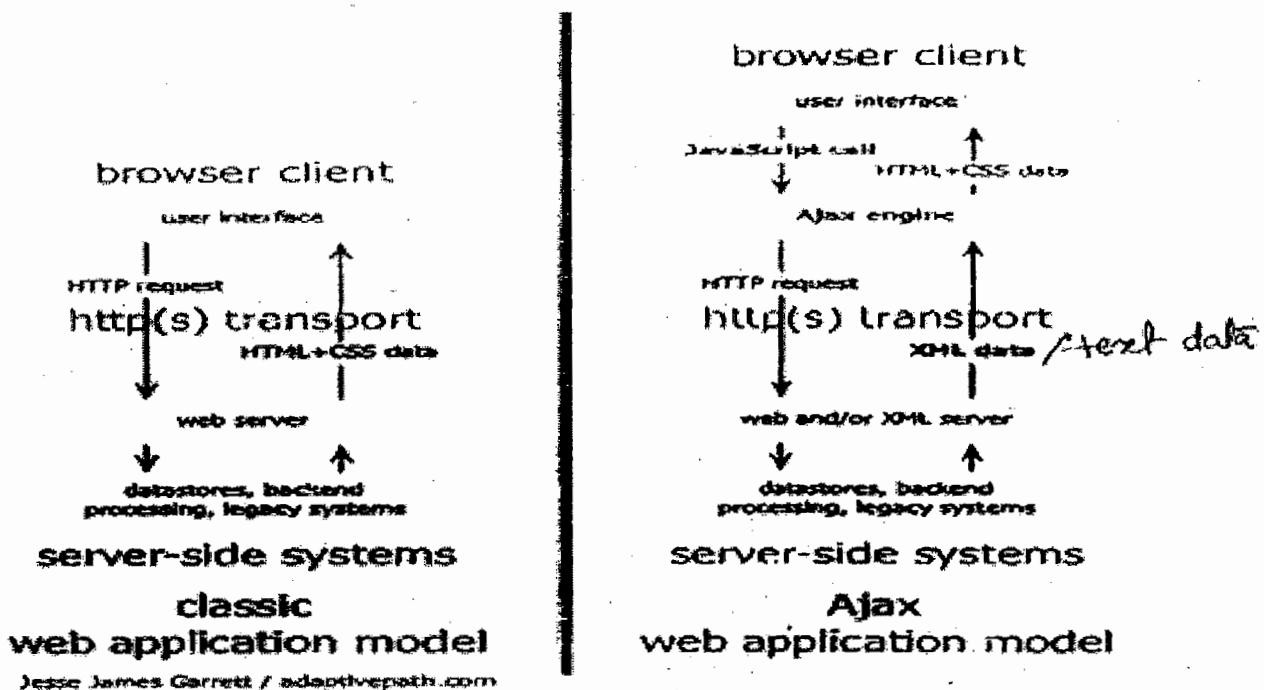


Figure 1: The traditional model for web applications (left) compared to the Ajax model (right).

This approach makes a lot of technical sense, but it doesn't make for a great user experience. While the server is doing its thing, what's the user doing? That's right, waiting. And at every step in a task, the user waits some more.

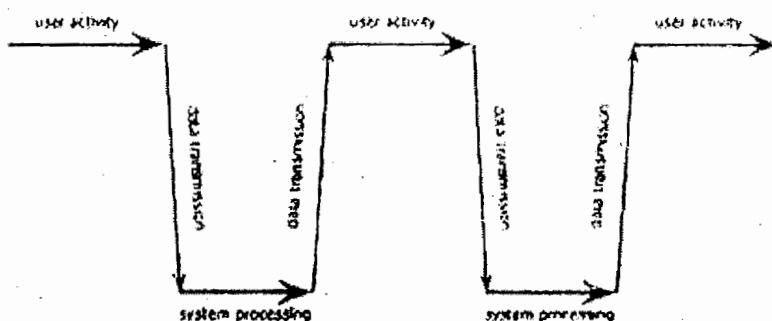
Obviously, if we were designing the Web from scratch for applications, we wouldn't make users around. Once an interface is loaded, why should the user interaction come to a halt every time the application needs something from the server? In fact, why should the user see the application go to the server at all?

### How Ajax is Different

An Ajax application eliminates the start-stop-start-stop nature of interaction on the Web by introducing an intermediary — an Ajax engine — between the user and the server. It seems like adding a layer to the application would make it less response but the opposite is true.

Instead of loading a webpage, at the start of the session, the browser loads an Ajax engine — written in JavaScript and usually tucked away in a hidden frame. This engine is responsible for both rendering the interface the user sees and communicating with the server on the user's behalf. The Ajax engine allows the user's interaction with the application to happen asynchronously — independent of communication with the server. So the user is never staring at a blank browser window and an hourglass icon, waiting around for the server to do something.

classic web application model (synchronous)



Ajax web application model (asynchronous)

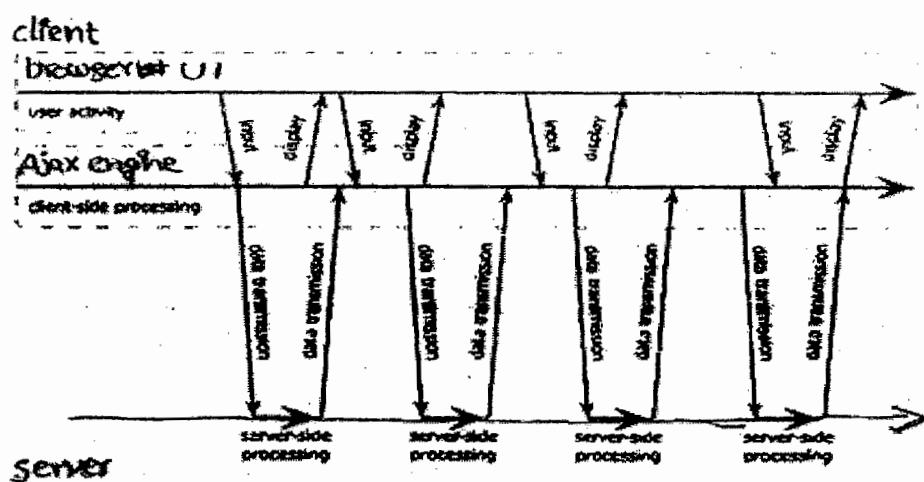


Figure 2: The synchronous interaction pattern of a traditional web application (top) compared with the Asynchronous pattern of an Ajax application (bottom).

Every user action that normally would generate an HTTP request takes the form of a JavaScript call to the Ajax engine instead. Any response to a user action that doesn't require a trip back to the server — such as simple data validation, editing data in memory, and even some navigation — the engine handles on its own. If the engine needs something from the server in order to respond — if it's submitting data for processing, loading additional interface code, or retrieving new data — the engine makes those requests asynchronously, usually using XML, without stalling a user's interaction with the application.

**Who's Using Ajax**

Google is making a huge investment in developing the Ajax approach. All of the major products Google has introduced over the last year — Orkut, Gmail, the latest beta version of GOOGLE GROUPS, Google Suggest, and Google Maps — are Ajax applications. (For more on the technical nuts and bolts of these Ajax implementations, check out these excellent analyses of Gmail, Google Suggest, and Google Maps.) Others are following suit: many of the features that people love in Flickr depend on Ajax, and Amazon's A9.com search engine applies similar techniques.

These projects demonstrate that Ajax is not only technically sound, but also practical or real world applications. This isn't another technology that only works in a laboratory. And Ajax applications can be any size, from the very simple, single function Google Suggest to the very complex and sophisticated Google Maps.

At Adaptive Path, we've been doing our own work with Ajax over the last several months, and we're realizing we've only scratched the surface of the rich interaction and responsiveness that Ajax applications can provide. Ajax is an important development for Web applications, and its importance is only going to grow. And because there are so many developers out there who already know how to use these technologies, we expect to see many more organizations following Google's lead in reaping the competitive advantage Ajax provides.

**Moving Forward**

The biggest challenges in creating Ajax applications are not technical. The core Ajax technologies are mature, stable, and well understood. Instead, the challenges are for the designers of these applications: to forget what we think we know about the limitations of the Web, and begin to imagine a wider, richer range of possibilities.

## Ajax Q&A

**Q: Did Adaptive Path invent Ajax? Did Google? Did Adaptive Path help build Google's Ajax applications?**

**A:** Neither Adaptive Path nor Google invented Ajax. Google's recent products are simply the highest-profile examples of Ajax applications. Adaptive Path was not involved in the development of Google's Ajax applications, but we have been doing Ajax work for some of our other clients.

**Q: Is Adaptive Path selling Ajax components or trade marking the name? Where can I download?**

**A:** Ajax isn't something you can download. It's an approach — a way of thinking about the architecture of web applications using certain technologies. Neither the Ajax name nor the approach are proprietary to Adaptive Path.

**Q: Is Ajax just another name for XMLHttpRequest?**

**A:** No. XMLHttpRequest is only part of the Ajax equation. XMLHttpRequest is the technical component that makes the asynchronous server communication possible; Ajax is our name for the overall approach described in the article, which relies not only on XMLHttpRequest, but on CSS, DOM, and other technologies.

**Q: Why did you feel the need to give this a name?**

**A:** I needed something shorter than "Asynchronous JavaScript+CSS+DOM + XMLHttpRequest" to use when discussing this approach with clients.

**Q: Techniques for asynchronous server communication have been around for years. What makes Ajax a "new" approach?**

**A:** What's new is the prominent use of these techniques in real-world applications to change the fundamental interaction model of the Web. Ajax is taking hold now because these

## **STRUTS**

## **DURGASOFT**

technologies and the industry's understanding of how to deploy them most effectively have taken time to develop.

**Q: Is Ajax a technology platform or is it an architectural style?**

**A:** It's both. Ajax is a set of technologies being used together in a particular way.

**Q: What kinds of applications is Ajax best suited for?**

**A:** We don't know yet. Because this is a relatively new approach, our understanding of where Ajax can best be applied is still in its infancy. Sometimes the traditional web application model is the most appropriate solution to a problem.

**Q: Does this mean Adaptive Path is anti-Flash?**

**A:** Not at all. Macromedia is an Adaptive Path client, and we've long been supporters of Flash technology. As Ajax matures, we expect that sometimes Ajax will be the better solution to a particular problem, and sometimes Flash will be the better solution. We're also interested in exploring ways the technologies can be mixed (as in the case of Flickr, which uses both).

**Q: Does Ajax have significant accessibility or browser compatibility limitations? Do Ajax applications break the back button? Is Ajax compatible with REST? Are there security considerations with Ajax development? Can Ajax applications be made to work for users who have JavaScript turned off?**

**A:** The answer to all of these questions is "maybe". Many developers are already working on ways to address these concerns. We think there is more work to be done to determine all the limitations of Ajax, and we expect the Ajax development community to uncover more issues like these along the way.

**Q: Some of the Google examples you cite don't use XML at all. Do I have to use XML and/or XSLT in an Ajax application?**

**A:** No. XML is the most fully-developed means of getting data in and out of an Ajax client, but there's no reason you couldn't accomplish the same effects using a technology like JavaScript Object Notation or any similar means of structuring data for interchange.

**Q: Are Ajax applications easier to develop than traditional web applications?**

**A:** Not necessarily. Ajax applications inevitably involve running complex JavaScript code on the client. Making that complex code efficient and bug-free is, not a task to be taken lightly, and better development tools and frameworks will be needed to help us meet that challenge.

**Q: Do Ajax applications always deliver a better experience than traditional web applications?**

**A:** Not necessarily. Ajax gives interaction designers more flexibility. However, the more power we have, the more caution we must use in exercising it. We must be careful to use Ajax to enhance the user experience of our applications, not degrade it.

## **Asynchronous JavaScript Technology and XML (Ajax) With the Java Platform**

By Grey Murray , June 9,2005;updated October 2006

Anyone who has used Flickr, GMail, Google Suggest, or Google Maps will realize that a new breed of dynamic web applications is emerging. These applications look and act very similar to traditional desktop applications without relying on plug-ins or browser-specific features. Web applications have traditionally been a set of HTML pages that must be reloaded to change any portion of the content. Technologies such as JavaScript programming language and cascading style sheets (CSS) have matured to the point where they can be used effectively to create very dynamic web applications that will work on all of the major browsers. This article will detail several techniques that you can use today to enable your web applications to be more rich and interactive like desktop applications.

**Introducing Asynchronous JavaScript Technology and XML (Ajax)**

Ajax is not new. These techniques have been available to developers targeting Internet Explorer on the Windows platform for many years. Until recently, the technology was known as web remoting or remote scripting. Web developers have also used a combination of plug-ins, Java applets, and hidden frames to emulate this interaction model for some time. What has changed recently is the inclusion of support for the XMLHttpRequest object in the JavaScript runtimes of the mainstream browsers. The real magic is the result of the JavaScript technology's XMLHttpRequest object. Although this object is not specified in the formal JavaScript technology specification, all of today's mainstream browsers support it. The subtle differences with the JavaScript technology and CSS support among current generation browsers such as Mozilla Firefox, Internet Explorer, and Safari are manageable. JavaScript libraries such as DOJO, Prototype, and the Yahoo User Interface Library have emerged to fill in where the browsers are not as manageable and to provide a standardized programming model. Dojo, for example, is addressing accessibility, internationalization, and advanced graphics across browsers — all of which had been thorns in the side of earlier adopters of Ajax. More updates are sure to occur as the need arises.

What makes Ajax-based clients unique is that the client contains page-specific control embedded as JavaScript technology. The page interacts with the JavaScript technology based events such as the loading of a document, a mouse click, focus changes, or even a timer. Ajax interactions allow for a clear separation of presentation logic from the data. An HTML page can pull in bite-size pieces to be displayed. Ajax will require different server-side architecture to support this interaction model. Traditionally, server-side web applications have focused on generating HTML documents for every client event resulting in a call to the server. The clients would then refresh and re-render the complete HTML page for each response. Rich web applications focus on a client fetching an HTML document that acts as a template or container into which to inject content, based on client events using XML data retrieved from a server-side component.

**Some uses for Ajax interactions are the following:**

**Real-time form data validation:** Form data such as user IDs, serial numbers, postal codes, or even special coupon codes that require server-side validation can be validated in a form before the user submits a form. See Real-time form validations for details.

**Auto completion:** A specific portion of form data such as an email address, name, or city name may be auto completed as the user types.

**Load on demand:** Based on a client event, an HTML page can fetch more data in the background, allowing the browser to load pages more quickly.

**Sophisticated user interface controls and effects:** Controls such as trees, menus, data tables, rich text editors, calendars, and progress bars allow for better user interaction and interaction with HTML pages, generally without requiring the user to reload the page.

**Refreshing data and server push:** HTML pages may poll data from a server for up-to-date data such as scores, stock quotes, weather, or application-specific data. A client may use Ajax techniques to get a set of current data without reloading a full page. Polling is not the most efficient means of ensuring that data on a page is the most current. Emerging techniques such as comet are being developed to provide true server-side push over HTTP by keeping a persistent connection between the client and server. See this blog entry on comment using for more on the development of server push with Java technology.

**Partial submit:** An HTML page can submit form data as needed without requiring a full page refresh.

**Mashups:** An HTML page can obtain data using a server-side proxy or by including an external script to mix external data with your application's or your service's data. For example, you can mix content or data from a third-party application such as Google Maps with your own application.

## STRUTS

## DURGASOFT

Page as an application: Ajax techniques can be made to create single-page applications that look and feel much like a desktop application. See the article on the use of Ajax and portlets for more on how you can use portlet applications today.

Though not all-inclusive, this list shows that Ajax interactions allow web applications to do much more than they have done in the past.

### The Anatomy of an Ajax Interaction

Now that we have discussed what Ajax is and what some higher-level issues are, let's put all the pieces together and look at an Ajax-enabled Java application.

Let's consider an example. A web application contains a static HTML page, or an HTML page generated in JSP technology contains an HTML form that requires server-side logic to validate form data without refreshing the page. A server-side web component (servlet) named ValidateServlet will provide the validation logic. Figure 1 describes the details of the Ajax interaction that will provide the validation logic.

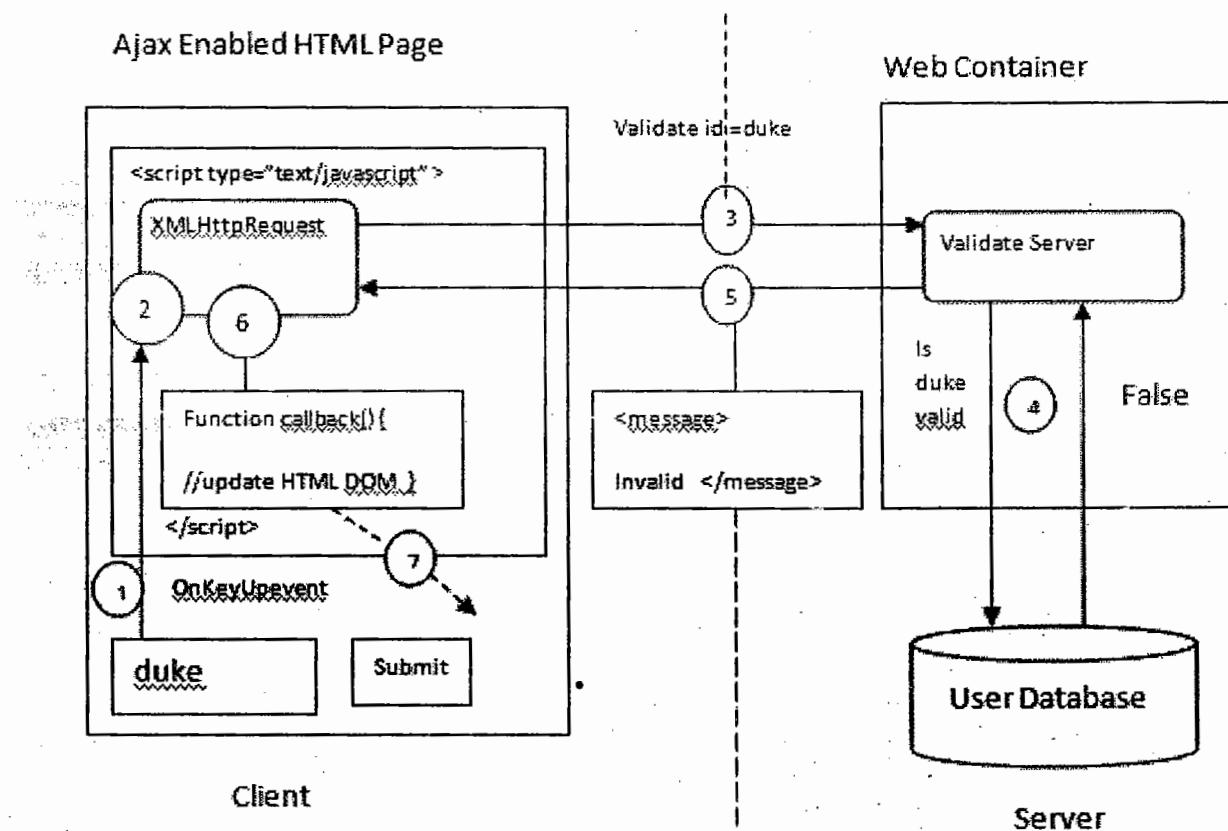


Figure 1: An Ajax Interaction providing Validation Logic.

The following items represent the setups of an Ajax interaction as they appear in Figure 1.

1. A client event occurs.
2. An XMLHttpRequest Object is created and configured
3. The XMLHttpRequest object makes a call
4. The request is processed by the ValidateServlet.

## **STRUSTS**

**DURGASOFT**

5. The ValidateServlet returns an XML document containing the result
6. The XMLHttpRequest Object calls the call back function and processes the result
7. The HTML DOM is updated.

**Now let's look at each step of the Ajax interaction in more detail.**

### **1. A client event occurs.**

JavaScript technology functions are called as the result of an event. In this case, the function validate() may be mapped to a onkeyup event on a link or form component.

```
<input type="text" size = "20" id="userid" name="id" onkeyup="validate();">
```

This form element will call the validate() function each time the user presses a key in the form field.

### **2. A XMLHttpRequest object is created and configured.**

An XMLHttpRequest object is created and configured.

```
var req;  
function validate() {  
    var idField = document.getElementById("userid");  
    var url = "validate?id=" + encodeURIComponent(idField.value);  
    if (typeof XMLHttpRequest != "undefined") {  
        req = new XMLHttpRequest(); } else if (window.ActiveXObject) {  
        req = new ActiveXObject("Microsoft.XMLHTTP");  
        req.open("GET", url, true);  
        req.onreadystatechange = callback;  
        req.send(null); }
```

The validate() function creates an XMLHttpRequest object and calls the open function on the object. The open function requires three arguments: the HTTP method, which is GET or POST; the URL of the server-side component that the object will interact with; and a Boolean indicating whether or not the call will be made asynchronously. The API is XMLHttpRequest. Open (String method, String URL, Boolean asynchronous). If an interaction is set as asynchronous (true) a callback function must be-specified. The callback function for this interaction is set with the statement req.onreadystatechange = callback;. See section 6 for more details.

### **3. The XMLHttpRequest object makes a call.**

When the statement req.send(null); is reached, the call will be made. In the case of an HTTP GET, this content may be null or left blank. When this function is called on the XMLHttpRequest object, the call to the URL that was set during the configuration of the object is called. In the case of this example, the data that is posted () is included as a URL parameter.

Use an HTTP GET when the request is idempotent, meaning that two duplicate requests will return the same results. When using the HTTP GET method, the length of URL, including escaped URL parameters, is limited by some browsers and by server-side web containers. The HTTP POST method should be used when sending data to the server that will affect the server-side application state. An HTML POST requires a Content-Type header to be set on the XMLHttpRequest object by using the following statement:

## **STRUTS**

## **DURGASOFT**

```
req.setRequestHeader("Content-Type", "application/x-www-form-urlencoded"); req.send("id = "+ encodeURIComponent(idTextField.value));
```

When sending form values from JavaScript technology, you should take into consideration the encoding of the field values. JavaScript technology includes an encodeURIComponent() function that should be used to ensure that localized content is encoded properly and that special characters are encoded correctly to be passed in an HTTP request.

### **4. The request is processed by the ValidateServlet.**

A servlet mapped to the URI "validate" checks whether the user ID is in the user database.

A servlet processes an XMLHttpRequest just as it would any other HTTP request. The following example shows a server extracting the id parameter from the request and validating whether the parameter has been taken.

```
public class ValidateServlet extends HttpServlet {  
    private ServletContext context;  
    private HashMap users = new HashMapQ;  
    public void init(ServletConfig config) throws ServletException {  
        super.init(config);  
        this.context = config.getServletContext();  
        users.put("greg","account data");  
        users.put("duke","account data"); }  
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws  
    IOException, ServletException {  
        String targetId = request.getParameter("id");  
        if ((targetId != null) && !users.containsKey(targetId.trim())) {  
            response.setContentType("text/xml"); response.setHeader( "Cache-Control", "no- cache");  
            response.getWriter().write("< message > valid </message>"); } else {  
            response.setContentType("text/xml");  
            response.setHeader( "Cache-Control", "no- cache");  
            response.getWriter().write("< message > invalid </message >");  
        }  
    }  
}
```

In this example, a simple HashMap is used to contain the users. In the case of this example, let us assume that the user typed duke as the ID.

### **5. The ValidateServlet returns an XML document containing the results.**

The user ID duke is present in the list of user IDs in the users HashMap. The ValidateServlet will write an XML document to the response containing a message element with the value of invalid. More complex usecases may require DOM, XSLT, or other APIs to generate the response.

```
response.setContentType("text/xml");  
response.setHeader(" Cache-Control", "no- cache"), response.getWriter().write( "in valid");
```

The developer must be aware of two things. First, the Content-Type must be set to text/xml. Second, the Cache-Control must be set to no-cache. An XMLHttpRequest object will process only requests that are of the Content-Type of only text/xml. and setting Cache-Control to no- cache

## **STRUSTS**

## **DURGASOFT**

will keep browsers from locally caching responses for cases in which duplicate requests for the same URL (including URL parameters) may return different responses.

### **6. The XMLHttpRequest object calls the callback() function and processes the result.**

The XMLHttpRequest object was configured to call the callback() function when there are changes to the readyState of the XMLHttpRequest object. Let us assume the call to the ValidateServlet was made and the readyState is 4, signifying the XMLHttpRequest call is complete. The HTTP status code of 200 signifies a successful HTTP interaction.

```
function callbackQ {  
if (req.readyState == 4) { if (req.status == 200) {  
// update the HTML DOM based on whether or not message is valid  
}}
```

Browsers maintain an object representation of the documents being displayed (referred to as the Document Object Model or DOM). JavaScript technology in an HTML page has access to the DOM, and APIs are available that allow JavaScript technology to modify the DOM after the page has loaded.

Following a successful request, JavaScript technology code may modify the DOM of the HTML page. The object representation of the XML document that was retrieved from the ValidateServlet is available to JavaScript technology code using the req.responseXML, where req is an XMLHttpRequest object. The DOM APIs provide a means for JavaScript technology to navigate the content from that document and use that content to modify the DOM of the HTML page. The string representation of the XML document that was returned may be accessed by calling

req.responseText. Now let's look at how to use the DOM APIs in JavaScript technology by looking the following XML document returned from the ValidateServlet.

```
<message>  
valid  
</message>
```

This example is a simple XML fragment that contains the sender of the <message> element, which is simply the string valid or invalid . A more advanced sample may contain more than one message and valid names that might be presented to the user:

```
function parseMessage(){  
var message=req.responseXML.getElementsByTagName("message")[0];  
value=message.childNodes[0].nodeValue;  
}
```

The parseMessage() function will process an XML document retrieved from the ValidateServlet. This function will call the setMessage() with the value of the message element to update the HTML DOM.

### **7. The HTML DOM is updated.**

JavaScript technology can gain a reference to any element in the HTML DOM using a number of APIs. The recommended way to gain a reference to an element is to call document.getElementById("userIdMessage"), where "userIdMessage" is the ID attribute of an element appearing in the HTML document. With a reference to the element, JavaScript technology may now be used to modify the element's attributes, modify the element's style properties; or add, remove, or modify child elements.

## STRUCTS

DURGASOFT

One common means to change the body content of an element is to set the innerHTML property on the element as in the following example.

```
< script type = "text/javascript">
function setMessage(message) {
var mdiv = document.getElementById("userIdMessage");
if (message == "invalid") { mdiv.innerHTML = "<div style = \"color:red\">Invalid User Id</div>";
} else { mdiv.innerHTML = "<div style=\"color:green\">Valid User Id</div>";
}
</script> <body>
<div id = "userIdMessage">x/div> </body>
```

The portions of the HTML page that were affected are re-rendered immediately following the setting of the innerHTML. If the innerHTML property contains elements such as `<image>` or `<iframe>`, the content specified by those elements is fetched and rendered as well. Ajax applications such as Google Maps use this technique of adding image elements using Ajax calls to dynamically build maps.

The main drawback with this approach is that HTML elements are hardcoded as strings in the JavaScript technology code. Hard coding HTML markup inside JavaScript technology code is not a good practice because it makes the code difficult to read, maintain, and modify. Consider using the JavaScript technology DOM APIs to create or modify HTML elements within JavaScript technology code. Intermixing presentation with JavaScript technology code as strings will make a page difficult to read and edit.

Another means of modifying the HTML DOM is to dynamically create new elements and append them as children to a target element as in the following example.

```
<script type="text/javascript">
function setMessage(message) {
var userMessageElement = document.getElementById("userIdMessage");
var messageText;
if (message == "invalid") {
userMessageElement.style.color = "red";
messageText = "Invalid User Id"; } else {
userMessageElement.style.color = "green";
messageText = "Valid User Id";
}
var messageBody = document.createTextNode(messageText);
// if the messageBody element has been created simple replace it otherwise
// append the new element
if (userMessageElement.childNodes[0]) {
```

## STRUTS

```
userMessageElement.replaceChild(messageBody, userMessageElement.childNodes[0]);  
} else {  
    userMessageElement.appendChild( messageBody);  
}  
</script>  
<body>  
<div id="userIdMessage"></div>  
</body>
```

## DURGASOFT

The code sample shows how JavaScript technology DOM APIs may be used to create an element or alter the element programmatically. The support for JavaScript technology DOM APIs can differ in various browsers, so you must take care when developing applications.

### Final Thoughts

Although many of these benefits are noteworthy, this approach has some challenges as well:

**Complexity:** Server-side developers will need to understand that presentation logic will be required in the HTML client pages as well as in the server-side logic to generate the XML content needed by the client HTML pages. HTML page developers need to have a basic understanding of JavaScript technology to create new Ajax functionality. Other options such as Project jMaki and Project Dynamic Faces provide a way for Java developers to better use Ajax functionality without requiring deep knowledge of JavaScript technology.

**Standardization of the XMLHttpRequest object:** The XMLHttpRequest object is not yet part of the JavaScript technology specification, which means that the behavior may vary depending on the client. It's best to use libraries such as Dojo, which provides fallback solutions for making Ajax-interactions transparently even on older browsers that do not support the XMLHttpRequest Object:

**JavaScript technology implementations:** Ajax interactions depend heavily on JavaScript technology, which has subtle differences depending on the client. See QuirksMode.org for more details on browser-specific differences. Consider using a library such as Dojo, which addresses many of the differences.

**Debugging:** Ajax applications are also difficult to debug because the processing logic is embedded both in the client and on the server. Browser add-ons such as Mozilla Firebug have emerged to make debugging easier. Frameworks such as the Google Tool Kit have emerged to allow for client and server round-trip debugging.

**Securing resources and protecting your data:** You can view client-side JavaScript technology simply by selecting View Source from an Ajax-enabled HTML page. A poorly designed Ajax-based application could open itself up to hackers or plagiarism. When providing Ajax services, you should take care to make sure that those services are made available only to those intended. See Restricting Access to your Ajax Services for more information on protecting your services.

## **STRUTS**

## **DURGASOFT**

We have seen that Ajax interactions can solve many problems. Java technology provides a good base to develop and deploy Ajax-based applications with APIs for tying in HTTP processing, databases, web services, XML processing, and business objects. With a better understanding of this interaction model, today's applications can become more interactive, providing the end user with a better experience.

Using Ajax requires that you use the latest browser versions that support the XMLHttpRequest object needed for Ajax interactions. Using Ajax also requires a great deal of client-side JavaScript technology and CSS. As an application architect or developer, you will need to weigh the needs of having a rich application against browser support, architecture complexity, and developer training. As the Ajax programming model evolves, existing technologies and frameworks will make this transition easier.

## STRUTS

## DURGASOFT

```
1 =====
2 AjaxTest1
3 -----Start.html-----
4 <HTML>
5   <HEAD>
6     <TITLE>Start.html</TITLE>
7     <script>
8
9     var x
10
11    function showHint(str)
12    {
13      if (str.length==0)
14      {
15        document.getElementById("t").innerHTML=""
16        return
17      }
18
19      x=GetXmlHttpObject()
20
21      if (x==null)
22      {
23        alert ("Browser does not support XMLHttpRequest obj")
24        return
25      }
26
27      var url="GetSuggestions.jsp"
28
29      url = url + "?q=" + str
30
31      x.onreadystatechange = stateChanged
32      x.open("GET",url,true)
33      x.send(null)
34    }
35
36    function stateChanged()
37    {
38      if (x.readyState==4 || x.readyState=="complete")
39      {
40        document.getElementById("t").innerHTML=x.responseText
41      }
42    }
43
44    function GetXmlHttpObject()
45    {
46      var objXMLHttp=null
47
48      if (window.XMLHttpRequest)
49      {
50        objXMLHttp=new XMLHttpRequest()
51      }
52      else if (window.ActiveXObject)
53      {
54        objXMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
55      }
56
57      return objXMLHttp
58    }
59
60    </SCRIPT>
61    </HEAD>
```

```

62
63 <BODY>
64 <FORM>
65   First Name:
66   <input type="text" id="txt1" name="put"
67     onkeyup="showHint(this.value)">
68
69   <p>Suggestions:
70     <span id="t"></span></p>
71 </FORM>
72 </BODY>
73 </HTML>
74 -----GetSuggestions.jsp-----
75 <HTML>
76   <HEAD>
77     <TITLE>GetSuggestions.jsp</TITLE>
78   </HEAD>
79   <BODY>
80   <%
81   String a[]={ 
82   {
83     "Anna",
84     "Brittany",
85     "Cinderella",
86     "Diana",
87     "Eva",
88     "Fiona",
89     "Gunda",
90     "Hege",
91     "Inga",
92     "Johanna",
93     "Kitty",
94     "Linda",
95     "Nina",
96     "Ophelia",
97     "Petunia",
98     "Amanda",
99     "Raquel",
100    "Cindy",
101    "Doris",
102    "Eve",
103    "Evita",
104    "Sunniva",
105    "Tove",
106    "Unni",
107    "Durga",
108    "Violet",
109    "Liza",
110    "Elizabeth",
111    "Ellen",
112    "Wenche",
113    "Vicky"
114  };
115  String q = request.getParameter("q");
116  String hint="";
117
118  if(q != null)
119  {
120    for(int i=0;i<a.length;i++)
121    {
122      if(a[i].toUpperCase().startsWith(q.toUpperCase())))

```

## STRUTS

## DURGASOFT

```
123      {
124          hint = hint + " " + a[i];
125      }
126  }
127  }
128 else
129 {
130     hint="";
131 }
132
133 if(hint == null || hint.length() == 0)
134     response.getWriter().write("no suggestion");
135 else
136     response.getWriter().write(hint);
137
138 %>
139 </BODY>
140 </HTML>
141 -----web.xml-----
142 <web-app>
143 </web-app>
144 =====
145 AjaxTest2
146 -----links.html-----
147 <head>
148 <script language="javascript">
149     var xmlhttp1,xmlhttp2;
150
151 function fun1()
152 {
153     if(window.XmlHttpRequest)
154         xmlhttp1=new XmlHttpRequest();
155
156     else if(window.ActiveXObject)
157         xmlhttp1=new ActiveXObject("Msxml2.XMLHTTP");
158
159     xmlhttp1.onreadystatechange=result1;
160     xmlhttp1.open("get","srv1",true);
161     xmlhttp1.send();
162 }
163
164 function fun2()
165 {
166     if(window.XmlHttpRequest)
167         xmlhttp2=new XmlHttpRequest();
168
169     else if(window.ActiveXObject)
170         xmlhttp2=new ActiveXObject("Msxml2.XMLHTTP");
171
172     xmlhttp2.onreadystatechange=result2;
173     xmlhttp2.open("get","srv2",true);
174     xmlhttp2.send();
175 }
176
177 function result1()
178 {
179     if(xmlhttp1.readyState==4)
180     {
181         document.getElementById("id1").style.color="blue"
182         document.getElementById("id1").innerHTML=xmlhttp1.responseText;
183     }
```

```

184    }
185
186    function result2()
187    {
188      if(xmlhttp2.readyState==4)
189      {
190        document.getElementById("id2").style.color="red"
191        document.getElementById("id2").innerHTML+xmlhttp2.responseText;
192      }
193    }
194
195  </script>
196  </head>
197  <body>
198  <b> <a href="javascript:fun1()"> link1 </a> &nbsp;&nbsp;&nbsp;
199  <a href="javascript:fun2()"> link2 </a> </b>
200  <br>
201
202  <h2> <span id="id1"></span> <br><br><br> </h2>
203  <h1> <span id="id2"></span> </h1>
204  </body>
205
206  -----web.xml-----
207 <web-app>
208   <servlet>
209     <servlet-name>dum</servlet-name>
210     <servlet-class>LinksServlet</servlet-class>
211   </servlet>
212   <servlet-mapping>
213     <servlet-name>dum</servlet-name>
214     <url-pattern>/srv1</url-pattern>
215   </servlet-mapping>
216
217   <servlet>
218     <servlet-name>dum1</servlet-name>
219     <servlet-class>LinksServlet1</servlet-class>
220   </servlet>
221   <servlet-mapping>
222     <servlet-name>dum1</servlet-name>
223     <url-pattern>/srv2</url-pattern>
224   </servlet-mapping>
225 </web-app>
226  -----LinksServlet.java-----
227 //LinksServlet.java
228 import javax.servlet.*;
229 import javax.servlet.http.*;
230 import java.io.*;
231 public class LinksServlet extends HttpServlet
232 {
233   public void doGet(HttpServletRequest req,HttpServletResponse res) throws
234   ServletException,IOException
235   {
236     res.setHeader("Cache-Control","no-cache");
237     System.out.println("service() called..");
238
239     String s =req.getParameter("param");
240     PrintWriter pw = res.getWriter();
241
242     try
243     {
244       Thread.sleep(10000);

```

## STRUTS

## DURGASOFT

```
244     }
245     catch(Exception e)
246     {
247         pw.println("Result of LINK1");
248         pw.close();
249     }
250
251 }
-----LinksServlet1.java-----
252 //LinksServlet1.java
253 import javax.servlet.*;
254 import javax.servlet.http.*;
255 import java.io.*;
256 public class LinksServlet1 extends HttpServlet
257 {
258     public void doGet(HttpServletRequest req,HttpServletResponse res) throws
259     ServletException,IOException
260     {
261         res.setHeader("Cache-Control","no-cache");
262         PrintWriter pw = res.getWriter();
263         pw.println("Result of LINK2");
264         pw.close();
265     }
266
267 }
=====AjaxTest3
269 -----one.html-----
270 <!-- one.html -->
271
272 <form name="f1">
273     TypeSome thing:<input type=text name="t1" onKeyUp="fun1()"> <br>
274     Date:<input type=text name="t2" size=40>
275 </form>
276
277 <script language="javascript">
278     var x;
279     function fun1()
280     {
281         x = new ActiveXObject("Msxml2.XMLHTTP");
282         x.onreadystatechange = fun2;
283         x.open("GET","test1.jsp",true);
284         x.send();
285     }
286     function fun2()
287     {
288         if(x.readyState == 4)
289         {
290             document.f1.t2.value=x.responseText;
291         }
292     }
293 }
294 </script>
295 -----test1.jsp-----
296 <%-- test1.jsp --%>
297 <%
298     out.println(new java.util.Date());
299 %>
300 -----web.xml-----
301 <web-app/>
302 =====
303 AjaxTest4
```

```
304 -----Second.html-----
305 <script type="text/javascript" language="javascript">
306
307 function makeRequest(url)
308 {
309     var http_request = false;
310
311     if (window.XMLHttpRequest)
312     {
313         http_request = new XMLHttpRequest();
314
315         if (http_request.overrideMimeType)
316         {
317             http_request.overrideMimeType('text/xml');
318         }
319     }
320     else if (window.ActiveXObject)
321     {
322         try
323         {
324             http_request = new ActiveXObject("Msxml2.XMLHTTP");
325         }
326         catch (e)
327         {
328             try
329             {
330                 http_request = new ActiveXObject("Microsoft.XMLHTTP");
331             }
332             catch (e)
333             {
334             }
335         }
336     }
337
338     if (!http_request)
339     {
340         alert('Giving up :( Cannot create an XMLHTTP instance');
341         return false;
342     }
343
344     http_request.onreadystatechange = function()
345     {
346         alertContents(http_request);
347     };
348
349     http_request.open('GET', url, true);
350     http_request.send();
351 } // makeRequest(url)
352
353 function alertContents(http_request)
354 {
355     try
356     {
357         if (http_request.readyState == 4)
358         {
359             if (http_request.status == 200)
360             {
361                 var xmlDoc = http_request.responseXML;
362                 var r = xmlDoc.getElementsByTagName('root').item(0);
363
364                 alert(r.firstChild.data);
365             }
366         }
367     }
368 }
```

## STRUCTS

```
365     }
366 else
367 {
368     alert('There was a problem with the request.');
369 }
370 }
371 }
372 catch(e)
373 {
374     alert('Caught Exception: ' + e.description);
375 }
376 } //alertContents(http_request)
377
378 </script>
379 <span
380 style="cursor: pointer; text-decoration: underline"
381 onclick="makeRequest('test.xml')"
382     Make a request
383 </span>
384 -----test.xml-----
385 <?xml version="1.0" ?>
386 <root>
387     I'm a test.
388 </root>
389 -----web.xml-----
390 <web-app>
391 </web-app>
392 =====
393 AjaxTest5
394 -----two.html-----
395 <head>
396 <script language="javascript">
397 var xmlhttp
398 function fun1()
399 {
400 var str=document.ttt.sel.value;
401 xmlhttp=new ActiveXObject("Msxml2.XMLHTTP");
402 xmlhttp.onreadystatechange=fun2;
403
404 var url="test2.jsp?q="+str
405 xmlhttp.open("post",url,true);
406 xmlhttp.send();
407 }
408
409 function fun2()
410 {
411 var xmlDoc
412 if(xmlhttp.readyState==4)
413 {
414 xmlDoc=xmlhttp.responseXML.documentElement; //gives Document obj
415
416     document.getElementById("a").innerHTML=xmlDoc.getElementsByTagName("deptno")[0].childNodes[0].nodeValue;
417
418     document.getElementById("b").innerHTML=xmlDoc.getElementsByTagName("dname")[0].childNodes[0].nodeValue;
419
420     document.getElementById("c").innerHTML=xmlDoc.getElementsByTagName("loc")[0].childNodes[0].nodeValue;
421 }
422 }
423 } //if
424 } //fun2
```

## DURGASOFT

## STRUTS

DURGASOFT

```
420 </script>
421 </head>
422 <body>
423 <form name="ttt">
424 <select name="sel" onchange="fun1()">
425 <option value="10"> 10 </option>
426 <option value="20"> 20 </option>
427 <option value="30"> 30 </option>
428 <option value="40"> 40 </option>
429 </select>
430 </form>
431
432 <br>
433 deptno : <span id="a"></span> <br>
434 deptname : <span id="b"></span> <br>
435 loc : <span id="c"> </span> <br>
436 </body>
437 -----test2.jsp-----
438 <!--test2.jsp -->
439 <%@ page import="java.sql.*" %>
440 <%
441 response.setContentType("text/xml");
442
443 String p=request.getParameter("q");
444 int v =Integer.parseInt(p);
445
446 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
447 Connection con=DriverManager.getConnection("jdbc:odbc:oradsn","scott","tiger");
448
449 Statement st = con.createStatement();
450 ResultSet rs=st.executeQuery("select * from dept where deptno="+v);
451
452 if(rs.next())
453 {
454 out.println("<dept>");
455 out.println("<deptno>" +rs.getString(1)+ "</deptno>");
456 out.println("<dname>" +rs.getString(2)+ "</dname>");
457 out.println("<loc>" +rs.getString(3)+ "</loc>");
458 out.println("</dept>");
459 }
460
461 con.close();
462 %>
463 -----web.xml-----
464 <web-app>
465 </web-app>
466 =====
467 AjaxTest6
468 -----example.html-----
469 <!-- example.html -->
470 <head>
471 <script language="javascript">
472 var k;
473 function fun1(form)
474 {
475 k = new ActiveXObject("Msxml2.XMLHTTP");
476
477 var s = document.f1.sel1.value;
478 var url = "example.jsp?p1="+s;
480
```

## STRUSTS

## DURGASOFT

```
481 k.onreadystatechange=fun2;
482
483 k.open("post",url,true);
484 k.send();
485 }
486
487 function fun2()
488 {
489 var z=0;
490 removeall();
491 if(k.readyState==4)
492 {
493 var m = k.responseXML.documentElement;
494 var res=m.getElementsByTagName("ename")[z].childNodes[0].nodeValue;
495 while(res!=null)
496 {
497 addoption(res);
498 z++;
499 res=m.getElementsByTagName("ename")[z].childNodes[0].nodeValue;
500 }
501 }
502 }
503
504 function removeall()
505 {
506 var x1=document.f1.sel2.length;
507 for(i=x1 ; i>0; i--)
508 document.f1.sel2.options[i]=null;
509 }
510
511 function addoption(result)
512 {
513 var opt=document.createElement("OPTION");
514 opt.text=result;
515 opt.value=result;
516 document.f1.sel2.options.add(opt);
517 }
518
519 </script>
520 </head>
521 <body>
522 <form name="f1">
523 Deptno :<select name="sel1" onchange="fun1(this)">
524     <option value="10"> 10 </option>
525     <option value="20"> 20 </option>
526     <option value="30"> 30 </option>
527     <option value="40"> 40 </option>
528 </select> <br><br><br><br><br>
529 Enames : <select name="sel2">
530     <option> ----select ----- </option>
531 </select>
532 </form>
533 </body>
534 -----example.jsp-----
535 <%-- example.jsp --%>
536 <%@ page import="java.sql.*" %>
537 <%
538     response.setContentType("text/xml");
539     String param = request.getParameter("p1");
540     int d = Integer.parseInt(param);
541 }
```

```

542     Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
543     Connection con = DriverManager.getConnection("jdbc:odbc:oradsn","scott","tiger");
544
545     Statement st = con.createStatement();
546     ResultSet rs = st.executeQuery("select ename from emp where deptno="+d);
547
548     out.println("<root>");
549     while(rs.next())
550     {
551         out.println("<ename>" +rs.getString(1)+ "</ename>");
552     }
553     out.println("</root>");
554
555     con.close();
556     %>
557 -----web.xml-----
558 <web-app>
559 </web-app>
560 =====
561 AjaxTest7
562 -----two.html-----
563 <!-- two.html -->
564 <form name="f1">
565 Starting Letter(s) : <input type="text" name="t1" onKeyUp="fun1(this.value)">
566 </form>
567 <br>
568 <br>
569 <span id="id1" > </span>
570 <script language="javascript">
571 var x;
572 function fun1(str)
573 {
574 x = new ActiveXObject("Msxml2.XMLHTTP");
575
576 x.onreadystatechange = fun2;
577
578 var url = "test2.jsp?param1="+str;
579
580 x.open("GET",url,true);
581
582 x.send();
583 }
584 function fun2()
585 {
586
587 if(x.readyState==4)
588 {
589 document.getElementById("id1").innerHTML=x.responseText;
590 }
591
592 }
593 </script>
594 -----test2.jsp-----
595 <%-- test2.jsp --%>
596 <%@ page import="java.sql.*" %>
597 <%
598 String p = request.getParameter("param1"); Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
599 Connection con = DriverManager.getConnection("jdbc:odbc:oradsn","scott","tiger");
600
601 Statement st = con.createStatement();
602 String hint = " ";

```

## STRUSTS

## DURGASOFT

```
603
604     ResultSet rs = st.executeQuery("select ename from emp");
605
606     while(rs.next())
607     {
608         String ename = rs.getString(1);
609         if(ename.startsWith(p))
610             hint = hint+", "+ename;
611     }
612     con.close();
613
614     out.println(hint);
615 %>
616 -----web.xml-----
617 <web-app>
618 </web-app>
619 =====
620 AjaxTest8
621 -----test.html-----
622 <!-- test.html -->
623 <form name="f1">
624     Country : <input type=text name="t1" onkeyup="fun1()">
625 </form>
626 <br>
627 <script language="javascript">
628     var k
629     function fun1()
630     {
631         k = new ActiveXObject("Msxml2.XMLHTTP");
632
633         k.onreadystatechange=fun2;
634
635         var str=document.f1.t1.value
636         var url="test.jsp?q="+str
637
638         k.open("get",url,true);
639         k.send();
640     }
641
642     function fun2()
643     {
644         if(k.readyState==4)
645         {
646             alert("entered2..");
647             document.f1.t1.value=k.responseText;
648         }
649
650     }
651 </script>
652 -----test.jsp-----
653 <%-- test.jsp --%>
654 <%@ page import="java.util.*" %>
655 <%
656 System.out.println("entered into jsp..");
657 ArrayList choices=new ArrayList();
658
659 String input=request.getParameter("q");
660 System.out.println(input);
661
662 Locale[] locales=Locale.getAvailableLocales();
663 System.out.println("locales created..");
```

```

664
665 for(int i=0; i<locales.length; i++)
666 {
667 Locale locale = locales[i];
668 String country=locale.getDisplayCountry();
669
670 if(country.toUpperCase().startsWith(input.toUpperCase()))
671 choices.add(country);
672 }
673
674 Iterator it=choices.iterator();
675 while(it.hasNext())
676 {
677 String str=it.next().toString();
678 out.println(str+",");
679 System.out.println("in while loop");
680 }
681
682 %>
683 -----web.xml-----
684 <web-app>
685 </web-app>
686 =====
687 AjaxTest9
688 -----example4.htm-----
689 <html>
690 <head>
691 <title>Example</title>
692 </head>
693
694 <script>
695 var req;
696 var which;
697 function retrieveURL(url)
698 {
699 if (window.XMLHttpRequest) { // Non-IE browsers
700 req = new XMLHttpRequest();
701
702 req.onreadystatechange = processStateChange;
703 try {
704 req.open("GET", url, true);
705 } catch (e) {
706 alert(e);
707 }
708 req.send();
709 } //if
710 else if (window.ActiveXObject) // IE
711 {
712 req = new ActiveXObject("Microsoft.XMLHTTP");
713
714 if (req) {
715 req.onreadystatechange = processStateChange;
716 req.open("GET", url, true);
717 req.send();
718 }
719
720 } //else
721 //retrieveURL
722
723 function processStateChange() {
724 if (req.readyState == 4) { // Complete

```

```

725     if (req.status == 200) { // OK response
726         document.theForm.theTextarea.value = req.responseText;
727     }/if
728     else {
729         alert("Problem: " + req.statusText);
730     }/else
731     }/if
732 }//processState
733
734 </script>
735
736 <body>
737
738 <h1>Example</h1>
739 Dynamic &lt;textarea&gt;<hr>
740 This example shows how a &lt;textarea&gt; can be loaded on-the-fly with
741 content. Here is one of three poems loaded based on a change in a group of radio buttons.
742 <br><br>
743
744 <form name="theForm">
745
746     <input type="radio" name="poem" onClick="retrieveURL('example4Poem1.do');">Langston
    Hughes - Dreams<br>
747     <input type="radio" name="poem" onClick="retrieveURL('example4Poem2.do');">Walt Whitman -
    When I Heard The Learned Astronomer<br>
748     <input type="radio" name="poem" onClick="retrieveURL('example4Poem3.do');">Robert Frost -
    The Road Not Taken<br>
749
750     <textarea name="theTextarea" cols="80" rows="20">Click one!</textarea>
751 </form>
752 <br>
753
754 </body>
755 </html>
756 -----web.xml-----
757 <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
758 "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
759
760 <web-app>
761     <servlet>
762         <servlet-name>action</servlet-name>
763         <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
764         <init-param>
765             <param-name>config</param-name>
766             <param-value>/WEB-INF/struts-config.xml</param-value>
767         </init-param>
768     </servlet>
769
770     <servlet-mapping>
771         <servlet-name>action</servlet-name>
772         <url-pattern>*.do</url-pattern>
773     </servlet-mapping>
774
775     <welcome-file-list>
776         <welcome-file>index.htm</welcome-file>
777     </welcome-file-list>
778
779 </web-app>
780 -----struts-config.xml-----
781 <!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration
    1.2//EN"

```

```
782 "http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">
783
784 <struts-config>
785   <action-mappings>
786     <action path="/example4Poem1" type="org.apache.struts.actions.ForwardAction"
787       parameter="/poem1.txt" />
788     <action path="/example4Poem2" type="org.apache.struts.actions.ForwardAction"
789       parameter="/poem2.txt" />
790     <action path="/example4Poem3" type="org.apache.struts.actions.ForwardAction"
791       parameter="/poem3.txt" />
792   </action-mappings>
793 </struts-config>
794 -----
795 poem1.txt,poem2.txt,poem.txt files comes here
796 =====
797 -----example.htm-----
798 <html>
799   <head>
800     <title>Example</title>
801   <script>
802     var req;
803     function retrieveURL(url) {
804       if (window.XMLHttpRequest) { // Non-IE browsers
805         req = new XMLHttpRequest();
806         req.onreadystatechange = processStateChange;
807         try {
808           req.open("GET", url, true);
809         } catch (e) {
810           alert(e);
811         }
812         req.send();
813       }
814     else if (window.ActiveXObject) { // IE
815       req = new ActiveXObject("Microsoft.XMLHTTP");
816       if (req) {
817         req.onreadystatechange = processStateChange;
818         req.open("GET", url, true);
819         req.send();
820       }
821     }
822   }
823
824   function processStateChange() {
825     if (req.readyState == 4) { // Complete
826       if (req.status == 200) { // OK response
827         document.getElementById("characters").innerHTML = req.responseText;
828       } else {
829         alert("Problem: " + req.statusText);
830       }
831     }
832   }
833
834 </script>
835   </head>
836   <body onLoad="retrieveURL('characters.do');">
837     <h1>Example</h1>
```

## STRUTS

DURGASOFT

```
840 Dynamic &lt;select&gt; elements.<hr>
841 This example shows how the contents of a &lt;select&gt; can be changed
842 on-the-fly in response to the change in another.
843 <br><br>
844 Select a show, then see how the list of characters changes accordingly:
845 <br><br>
846
847 <form name="dynamicSelect">
848   TV Show: <select name="TVShowSelect"
849     onChange="retrieveURL('characters.do?tvShow=' + this.value);">
850     <option value=""></option>
851     <option value="Babylon5">Babylon 5</option>
852     <option value="StargateSG1">Stargate SG-1</option>
853     <option value="StarTrekTNG">Star Trek: The Next Generation</option>
854   </select>
855
856   <br>
857   Characters: <span id="characters"></span>
858 </form>
859
860 <br>
861 </body>
862 </html>
863 -----web.xml-----
864 <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
865 "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
866
867 <web-app>
868   <servlet>
869     <servlet-name>action</servlet-name>
870     <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
871     <init-param>
872       <param-name>config</param-name>
873       <param-value>/WEB-INF/struts-config.xml</param-value>
874     </init-param>
875   </servlet>
876
877   <servlet-mapping>
878     <servlet-name>action</servlet-name>
879     <url-pattern>*.do</url-pattern>
880   </servlet-mapping>
881
882   <welcome-file-list>
883     <welcome-file>index.htm</welcome-file>
884   </welcome-file-list>
885
886 </web-app>
887 -----struts-config.xml-----
888 <!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration
889 1.2//EN"
890 "http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">
891 <struts-config>
892   <action-mappings>
893     <action path="/characters" type="xhrstruts.GetCharactersAction" />
894   </action-mappings>
895 </struts-config>
896 -----GetCharacterAction.java-----
897 package xhrstruts;
898 import java.io.PrintWriter;
899 import java.util.ArrayList;
```

```
900 import java.util.Iterator;
901 import javax.servlet.http.HttpServletRequest;
902 import javax.servlet.http.HttpServletResponse;
903 import javax.servlet.http.HttpSession;
904 import org.apache.struts.action.Action;
905 import org.apache.struts.action.ActionForm;
906 import org.apache.struts.action.ActionForward;
907 import org.apache.struts.action.ActionMapping;
908
909
910 public class GetCharactersAction extends Action {
911
912
913     public ActionForward execute(ActionMapping mapping,
914                               ActionForm inForm,
915                               HttpServletRequest request,
916                               HttpServletResponse response) throws Exception {
917
918         // Get a list of characters associated with the select TV show
919         String tvShow = (String)request.getParameter("tvShow");
920         if (tvShow == null) {
921             tvShow = "";
922         }
923
924         ArrayList characters = getCharacters(tvShow);
925
926         // And yes, I know creating HTML in an Action is generally very bad form,
927         // but I wanted to keep this example simple.
928         String html = "<select name=\"CharactersSelect1">";
929
930         int i = 0;
931         for (Iterator it = characters.iterator(); it.hasNext();) {
932             String name = (String)it.next();
933             i++;
934             html += "<option value=\"" + i + "\">" + name + "</option>";
935         }
936
937         html += "</select>";
938
939         // Write the HTML to response
940         response.setContentType("text/html");
941         PrintWriter out = response.getWriter();
942         out.println(html);
943         out.flush();
944
945         return null; // Not forwarding to anywhere, response is fully-cooked
946
947     } // End execute()
948
949
950     // This method returns a list of characters for a given TV show. If no TV
951     // show is selected, i.e., initial page view, an empty ArrayList is returned.
952     private ArrayList getCharacters(String tvShow) {
953
954         ArrayList al = new ArrayList();
955
956         if (tvShow.equalsIgnoreCase("StarTrekTNG")) {
957             al.add("Jean Luc Picard");
958             al.add("William T. Riker");
959             al.add("Data");
960             al.add("Deanna Troi");
```

## STRUTS

DURGASOFT

```
961     al.add("Geordi LaForge");
962 }
963
964 if (tvShow.equalsIgnoreCase("Babylon5")) {
965     al.add("John Sheridan");
966     al.add("Delenn");
967     al.add("Londo Mollari");
968     al.add("Stephen Franklin");
969     al.add("Vir Cotto");
970 }
971
972 if (tvShow.equalsIgnoreCase("StargateSG1")) {
973     al.add("Samantha Carter");
974     al.add("Jack O'Neil");
975     al.add("Teal'c");
976     al.add("Daniel Jackson");
977     al.add("Baal");
978 }
979
980 return al;
981
982 } // End getCharacters()
983
984 } // End class
985
986 Struts2 App with Ajax TagLibrary
987 -----index.jsp-----
988 <%@ taglib prefix="ajax" uri="javawebparts/ajaxparts/taglib" %>
989 <html>
990 <body bgcolor=wheat>
991 <center><b>Registration Page</b><br><br>
992 <form name="myForm">
993     UserId: <input type="text" name="userId" size="20" onblur="javascript:void(0);"
994     id="checkUserId" />
995     <ajax:event ajaxRef="myRef/parameters" attachTo="checkUserId" />
996 </form>
997 <br>
998 <br>
999 <span id="id1"></span>
1000 <ajax:enable/>
1001 </body>
1002 </html>
1003 -----web.xml-----
1004 <?xml version="1.0" encoding="UTF-8"?>
1005 <web-app id="WebApp_9" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
1006     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1007     xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
1008     http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
1009     <!-- AjaxParts Taglib parameters -->
1010     <context-param>
1011         <param-name>AjaxPartsTaglibConfig</param-name>
1012         <param-value>/WEB-INF/ajax_config.xml</param-value>
1013     </context-param>
1014     <context-param>
1015         <param-name>AjaxPartsTaglibValidateConfig</param-name>
1016         <param-value>false</param-value>
1017     </context-param>
1018     <!-- End AjaxParts Taglib parameters -->
1019
```

## STRUTS

DURGASOFT

```
1020 <filter>
1021   <filter-name>struts2</filter-name>
1022   <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
1023 </filter>
1024 <filter-mapping>
1025   <filter-name>struts2</filter-name>
1026   <url-pattern>*</url-pattern>
1027 </filter-mapping>
1028
1029 <welcome-file-list>
1030   <welcome-file>index.jsp</welcome-file>
1031 </welcome-file-list>
1032 </web-app>
1033 -----struts.xml-----
1034 <?xml version="1.0" encoding="UTF-8" ?>
1035 <!DOCTYPE struts PUBLIC
1036   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
1037   "http://struts.apache.org/dtds/struts-2.0.dtd">
1038 <struts>
1039   <constant name="struts.devMode" value="true" />
1040   <package name="default" namespace="/" extends="struts-default">
1041     <action name="checkUserId" class="p1.TestAction" method="checkUsername" />
1042   </package>
1043 </struts>
1044 -----ajax_config.xml-----
1045 <ajaxConfig>
1046   <group ajaxRef="myRef" form="myForm">
1047     <element ajaxRef="parameters" method="get">
1048       <event type="onblur">
1049
1050       <requestHandler type="std:FormSender" target="/checkUserId.action">
1051         <parameter>parameters</parameter>
1052       </requestHandler>
1053
1054       <responseHandler type="std:innerHTML">
1055         <parameter>id1</parameter>
1056       </responseHandler>
1057
1058     </event>
1059   </element>
1060 </group>
1061 </ajaxConfig>
1062 -----TestAction.java-----
1063 package p1;
1064 import java.sql.*;
1065 import com.opensymphony.xwork2.ActionSupport;
1066 import java.io.PrintWriter;
1067 import javax.servlet.http.HttpServletRequest;
1068 import javax.servlet.http.HttpServletResponse;
1069 import org.apache.struts2.interceptor.ServletRequestAware;
1070 import org.apache.struts2.interceptor.ServletResponseAware;
1071 public class TestAction extends ActionSupport implements
1072   ServletResponseAware, ServletRequestAware {
1073
1074   private HttpServletRequest request;
1075   private HttpServletResponse response;
1076
1077   public void setServletResponse(final HttpServletResponse res) {
1078     this.response = res;
1079   } // End setServletResponse().
1080 }
```

## STRUCTS

## DURGASOFT

```
1081 public HttpServletResponse getServletResponse() {  
1082     return response;  
1083 } // End getServletResponse().  
1084  
1085 public void setServletRequest(final HttpServletRequest req) {  
1086     this.request = req;  
1087 } // End setServletRequest().  
1088  
1089  
1090 public HttpServletRequest getServletRequest(){  
1091     return request;  
1092 } // End getServletRequest().  
1093  
1094 public String checkUsername() throws Exception {  
1095     String userId=request.getParameter("userId");  
1096     String password=request.getParameter("password");  
1097     System.out.println("UserName="+userId+" Password="+password);  
1098  
1099     validateUserDB(userId);  
1100     return null;  
1101 }  
1102  
1103 public void validateUserDB(String userId)throws Exception  
1104 {  
1105     Class.forName("oracle.jdbc.driver.OracleDriver");  
1106     Connection  
1107         con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");  
1108  
1109     PreparedStatement ps=con.prepareStatement("select count(*) from useraccount where  
1110         userid=?");  
1111     ps.setString(1,userId);  
1112     ResultSet rs=ps.executeQuery();  
1113  
1114     int count=0;  
1115     if(rs.next())  
1116         count=rs.getInt(1);  
1117     PrintWriter out = response.getWriter();  
1118  
1119     if(count>=1)  
1120         out.print("<span style='color:red'>Sorry,UserId is Already taken</span>");  
1121     else  
1122         out.print("<span style='color:green'>Congratulation UserId is Available</span>");  
1123 } // End class.  
1124  
1125  
1126  
1127  
1128  
1129  
1130  
1131  
1132  
1133
```

## JQUERY

JQuery is great library for developing ajax based application. jQuery is great library for the JavaScript programmers, which simplifies the development of web 2.0 applications. You can use jQuery to develop cool web 2.0 applications. jQuery helps the programmers to keep code simple and concise. The jQuery library is designed to keep the things very simple and reusable.

jQuery library simplifies the process of traversal of HTML DOM tree. You can use jQuery to handle events, perform animation, and add the ajax support into your web applications with ease.

### Why jQuery?

You can use simple JavaScript to perform all the functions that jQuery provides. Then why jQuery? The jQuery library is providing many easy to use functions and methods to make rich applications. These functions are very easy to learn and even a designer can learn it fast. Due to these features jQuery is very popular and in high demand among the developers. You can use jQuery in all the web based applications irrespective of the technology.

jQuery is java script and can be used with JSP, Servlets, ASP, PHP, CGI and almost all the web programming languages.

The jQuery code is very simple and easy to learn.

### Here are the features of jQuery

- DOM element selections functions
- DOM traversal and modification
- Events
- CSS manipulation
- Effects and animations
- Ajax
- Extensibility
- Utilities - such as browser version and the each function.
- JavaScript Plugins

### How much time is required to learn jQuery?

You can learn jQuery in a day and master it within 2-3 days. There are so many features available with jQuery and you may spend months to explore these features.

We have spent huge time in exploring the jQuery and developing almost all the examples that can be developed with jQuery. So, explore our huge collection of jQuery tutorials and examples. Please continue reading the tutorials.

### jQuery Features

#### Selection of DOM elements :

The jQuery selector provide us capability to select DOM elements so that we can add functionality to them using methods of jQuery. It is using CSS 3.0 syntax which provide us freedom to select one or more elements. Using CSS , you can select element by id, class and collaborate with events to increase it's functionality.

#### The wrapped set

The selected elements reside inside a object known as wrapped set. It contain all the selected DOM elements, it has array like structure. You can traverse through this like an array and can select elements using index.

**Events**

jQuery provide simplified event handling, You can easily bind and unbind events and for supported browsers it also provide a normalized event model due to this it is very easy to handle events. When any event occurs , it is called under the context of the event that triggered it.

**Extensibility through plug-ins**

The jQuery architecture provide us freedom to extend functionality using plug-ins . The plug-ins are easy to use and easy to clip with your page. You just need to set parameters to use these jQuery plug-ins and also need to include plug-in file. Some the main jQuery plug-ins are :

- 1.XML and XSLT tools
- 2.cookie handling
- 3.datagrids
- 4.drag and drop events.
- 5.modal windows
- 6.dynamic lists
- 7.webservices
- 8.Ajax helpers
- 9.even a jQuery-based Commodore 64 emulator.

**Cross-browser support**

In JavaScript, the DOM implementations for event handling vary considerably between browsers. Where as jQuery providing a normalized event model for all supported browsers that makes it very easy to handle events.

**Ajax support**

AJAX stands for Asynchronous JavaScript and XML . Using AJAX we can connect to database and also can fetch the data from the server's database without refreshing the page. JQuery have very effective AJAX methods library to extend the functionality of AJAX.

**Compatibility with languages**

The jQuery script can be used with nearly all the web languages. Some of Frequently used languages with jQuery are given below:

- 1.PHP
- 2.JSP
- 3.ASP
- 4.Servlet
- 5.CGI

***jQuery Event Methods***

Event methods trigger, or bind a function to an event for all matching elements.

**Trigger example:**

`$("#button").click()` - triggers the click event for a button element.

**Binding example:**

`$("#button").click(function(){ $("#img").hide() })` - binds a function to the click event.

The following table lists all the methods used to handle events.

## STRUCTS

## DURGASOFT

Method	Description
<u>bind()</u>	Add one or more event handlers to matching elements
<u>blur()</u>	Triggers, or binds a function to the blur event of selected elements
<u>change()</u>	Triggers, or binds a function to the change event of selected elements
<u>click()</u>	Triggers, or binds a function to the click event of selected elements
<u>dblclick()</u>	Triggers, or binds a function to the dblclick event of selected elements
<u>delegate()</u>	Add one or more event handlers to current, or future, specified child elements of the matching elements
<u>die()</u>	Remove all event handlers added with the live() function
<u>error()</u>	Triggers, or binds a function to the error event of selected elements
event.currentTarget	The current DOM element within the event bubbling phase
event.data	Contains the optional data passed to jQuery.fn.bind when the current executing handler was bound
<u>event.isDefaultPrevented()</u>	Returns whether event.preventDefault() was called for the event object
event.isImmediatePropagationStopped()	Returns whether event.stopImmediatePropagation() was called for the event object
event.isPropagationStopped()	Returns whether event.stopPropagation() was called for the event object
<u>event.pageX</u>	The mouse position relative to the left edge of the document
<u>event.pageY</u>	The mouse position relative to the top edge of the document

## **STRUCTS**

## **DURGASOFT**

event.preventDefault()

Prevents the default action of the event

event.relatedTarget

The other DOM element involved in the event, if any

event.result

This attribute contains the last value returned by an event handler that was triggered by this event, unless the value was undefined

event.stopImmediatePropagation()

Prevents other event handlers from being called

event.stopPropagation()

Prevents the event from bubbling up the DOM tree, preventing any parent handlers from being notified of the event

event.target

The DOM element that initiated the event

event.timeStamp

This attribute returns the number of milliseconds since January 1, 1970, when the event is triggered

event.type

Describes the nature of the event

event.which

Which key or button was pressed for a key or button event

focus()

Triggers, or binds a function to the focus event of selected elements

focusin()

Binds a function to the focusin event of selected elements

focusout()

Binds a function to the focusout event of selected elements

hover()

Binds one or two functions to the hover event of selected elements

keydown()

Triggers, or binds a function to the keydown event of selected elements

keypress()

Triggers, or binds a function to the keypress event of selected elements

keyup()

Triggers, or binds a function to the keyup event of selected elements

live()

Add one or more event handlers to current, or future, matching elements

## STRUCTS

## DURGASOFT

load()

Triggers, or binds a function to the load event of selected elements

mousedown()

Triggers, or binds a function to the mouse down event of selected elements

mouseenter()

Triggers, or binds a function to the mouse enter event of selected elements

mouseleave()

Triggers, or binds a function to the mouse leave event of selected elements

mousemove()

Triggers, or binds a function to the mouse move event of selected elements

mouseout()

Triggers, or binds a function to the mouse out event of selected elements

mouseover()

Triggers, or binds a function to the mouse over event of selected elements

mouseup()

Triggers, or binds a function to the mouse up event of selected elements

one()

Add one or more event handlers to matching elements. This handler can only be triggered once per element

ready()

Binds a function to the ready event of a document  
(when an HTML document is ready to use)

resize()

Triggers, or binds a function to the resize event of selected elements

scroll()

Triggers, or binds a function to the scroll event of selected elements

select()

Triggers, or binds a function to the select event of selected elements

submit()

Triggers, or binds a function to the submit event of selected elements

toggle()

Binds two or more functions to the toggle between for the click event for selected elements

trigger()

Triggers all events bound to the selected elements

## STRUTS

## DURGASOFT

triggerHandler()

Triggers all functions bound to a specified event for the selected elements

unbind()

Remove an added event handler from selected elements

undelegate()

Remove an event handler to selected elements, now or in the future

unload()

Triggers, or binds a function to the unload event of selected elements

### ***jQuery AJAX Methods***

AJAX is the art of exchanging data with a server, and update parts of a web page - without reloading the whole page.

The following table lists all the jQuery AJAX methods:

<b>Method</b>	<b>Description</b>
<u>\$.ajax()</u>	Performs an AJAX request
<u>ajaxComplete()</u>	Specifies a function to run when the AJAX request completes
<u>ajaxError()</u>	Specifies a function to run when the AJAX request completes with an error
<u>ajaxSend()</u>	Specifies a function to run before the AJAX request is sent
<u>\$.ajaxSetup()</u>	Sets the default values for future AJAX requests
<u>ajaxStart()</u>	Specifies a function to run when the first AJAX request begins
<u>ajaxStop()</u>	Specifies a function to run when all AJAX requests have completed
<u>ajaxSuccess()</u>	Specifies a function to run an AJAX request completes successfully
<u>\$.get()</u>	Loads data from a server using an AJAX HTTP GET request
<u>\$.getJSON()</u>	Loads JSON-encoded data from a server using a HTTP GET request
<u>\$.getScript()</u>	Loads (and executes) a JavaScript from the a server using an AJAX HTTP GET request
<u>load()</u>	Loads data from a server and puts the returned HTML into the selected element
<u>\$.param()</u>	Creates a serialized representation of an array or object (can be used as URL query string for AJAX requests)

<u>\$.post()</u>	Loads data from a server using an AJAX HTTP POST request
<u>serialize()</u>	Encodes a set of form elements as a string for submission
<u>serializeArray()</u>	Encodes a set of form elements as an array of names and values

### jQuery AJAX ajax() Method



#### Example

Change the text of a div element using an AJAX request:

```
$(“button”).click(function(){
    $.ajax({url:”demo_ajax_load.txt”, success:function(result){
        $("div”).html(result);
    }});
});
```

[Try it yourself »](#)

#### Definition and Usage

The ajax() method is used to perform an AJAX (asynchronous HTTP) request.

All jQuery AJAX methods use the ajax() method. This method is mostly used for requests where the other methods cannot be used.

#### Syntax

`$.ajax({name:value, name:value, ... })`

The parameters specifies one or more name/value pairs for the AJAX request.

Possible names/values in the table below:

Name	Value/Description
async	A Boolean value indicating whether the request should be handled asynchronous or not. Default is true
beforeSend(xhr)	A function to run before the request is sent
cache	A Boolean value indicating whether the browser should cache the requested pages. Default is true
complete(xhr,status)	A function to run when the request is finished (after success and error functions).
contentType	The content type used when sending data to the server. Default is: "application/x-www-form-urlencoded"
context	Specifies the "this" value for all AJAX related callback functions

## STRUTS

## DURGASOFT

data	Specifies data to be sent to the server
dataFilter( <i>data,type</i> )	A function used to handle the raw response data of the XMLHttpRequest
dataType	The data type expected of the server response.
error( <i>xhr,status,error</i> )	A function to run if the request fails.
global	A Boolean value specifying whether or not to trigger global AJAX event handles for the request. Default is true
ifModified	A Boolean value specifying whether a request is only successful if the response has changed since the last request. Default is: false.
jsonp	A string overriding the callback function in a jsonp request
jsonpCallback	Specifies a name for the callback function in a jsonp request
password	Specifies a password to be used in an HTTP access authentication request.
processData	A Boolean value specifying whether or not data sent with the request should be transformed into a query string. Default is true
scriptCharset	Specifies the charset for the request
success( <i>result,status,xhr</i> )	A function to be run when the request succeeds
timeout	The local timeout (in milliseconds) for the request
traditional	A Boolean value specifying whether or not to use the traditional style of param serialization
type	Specifies the type of request. (GET or POST)
url	Specifies the URL to send the request to. Default is the current page
username	Specifies a username to be used in an HTTP access authentication request
xhr	A function used for creating the XMLHttpRequest object

```

1 // Example App on JQuery
2 -----index.html-----
3
4 <script language="JavaScript" src="jquery-1.6.2.js"></script>
5
6 <Script language="JavaScript">
7 $(document).ready(f1);//whenever page is loaded f1() will be called
8 var stateName;
9 function f1()
10 {
11   $("#selState").change(function(){
12     stateName=$("#selState").val();
13
14
15   var arrDist=new Array();
16   $.ajax({
17     type:"post",
18     url:"GetDistrict.jsp",
19     data:"state="+stateName,
20     cache: false,
21     success:function(msg)
22     {
23       xmlDoc = $.parseXML(msg);
24       msg = $(xmlDoc);
25       var i=0;
26       var arrDist=new Array();
27       msg.find("district").each(function()
28       {
29         arrDist[i]=$(this).text();
30         i++;
31         alert(arrDist);
32       });//find
33       var temp=<option value=None>-Select-</option>;
34       for(var j=0;j<arrDist.length;j++)
35       {
36         temp+=""+arrDist[j)+"</option>";
37       }/for
38       $("#selDistrict").empty(); //nullify previous data
39       $("#selDistrict").append(temp);
40     }/success
41   });//ajax
42 };//change
43 }/f1
44 </script>
45
46 <form>
47   Select State
48   <select id="selState">
49     <option value="None">-Select-</option>
50     <option value="AP">Andhra Pradesh</option>
51     <option value="Orissa">Orissa</option>
52     <option value="Karnataka">Karnataka</option>
53   </select>
54
55   Select District<select id="selDistrict"></select>
56 </form>
57 -----GetDistrict.jsp-----
58 <%
59 String statename=request.getParameter("state");
60 System.out.println(statename);
61

```

```

62 String data="";
63 if(statename.equals("Karnataka"))
64 {
65     data=<state><district>kolar</district><district>ramanagara</district><district>shimoga</district>
66     </state>";
67 } else if(statename.equals("AP"))
68 {
69     data=<state><district>rangareddy</district><district>medak</district><district>mahbubnagar</di
strict></state>";
70 }
71 else if(statename.equals("Orissa"))
72 {
73     data=<state><district>Rourkela</district><district>Jharsuguda</district><district>Bhubaneswar
    </district></state>";
74 }
75 else
76     data=<state/>;
77 System.out.print(data);
78 out.print(data);
79 %>
80 -----web.xml-----
81 <web-app>
82 =====
83 >>>>>>>App on AutoComplete>>>>>>
84 =====
85 -----AutoComplete.html-----
86 <html>
87 <head>
88 <title>jQuery</title>
89 <link rel="stylesheet" href="script/jquery-ui.css" />
90 <script src="script/jquery-1.8.3.js"></script>
91 <script src="script/jquery-ui.js"></script>
92 <script>
93 $(function() {
94     var nameList= [
95         "Arun",
96         "Anil",
97         "Abhaya",
98         "durga",
99         "Sai",
100        "Sachin",
101        "Kiran",
102        "Reddy",
103        "Rosy",
104        "Jyoti",
105        "Deepak",
106        "Anjaneyulu"
107    ];
108    $("#id1").autocomplete({
109        source: nameList
110    });
111 });
112 </script>
113 </head>
114 <body>
115
116 Enter Your Name<input id="id1" />

```

## **STRUTS**

117 </body>  
118 </html>  
119

**DURGASOFT**

## Structs Interview Questions and Answers (For better answers follow class room notes)

### What is Struts?

Struts is a web page development framework and an open source software that helps developers build web applications quickly and easily. Struts combines Java Servlets, Java Server Pages, custom tags, and message resources into a unified framework. It is a cooperative, synergistic platform, suitable for development teams, independent developers, and everyone between.

### What are the main classes which are used in struts application?

**Action servlet:** it's a back-bone of web application it's a controller class responsible for handling the entire request.

**Action class:** using Action classes all the business logic is developed us call model of the application also.

**Action Form:** it's a java bean which represents our forms and associated with action mapping. And it also maintains the session state its object is automatically populated on the server side with data entered from a form on the client side.

**Action Mapping:** using this class we do the mapping between object and Action.

**ActionForward:** this class in Struts is used to forward the result from controller to destination.

### How is the MVC design pattern used in Struts framework?

In the MVC design pattern, application flow is mediated by a central Controller. The Controller delegates requests to an appropriate handler. The handlers are tied to a Model, and each handler acts as an adapter between the request and the Model. The Model represents, or encapsulates, an application's business logic or state. Control is usually then forwarded back through the Controller to the appropriate View. The forwarding can be determined by consulting a set of mappings, usually loaded from a database or configuration file. This provides a loose coupling between the View and Model, which can make an application significantly easier to create and maintain.

Controller--Servlet controller which supplied by Struts itself; View --- what you can see on the screen, a JSP page and presentation components; Model --- System state and a business logic JavaBeans.

### What are the core classes of Struts?

Action, ActionForm, ActionServlet, ActionMapping, ActionForward are basic classes of Structs.

### What is the design role played by Struts?

The role played by Struts is controller in Model/View/Controller(MVC) style. The View is played by JSP and Model is played by JDBC or generic data source classes. The Struts controller is a set of programmable components that allow developers to define exactly how the application interacts with the user.

### How Struts control data flow?

Struts implements the MVC/Layers pattern through the use of ActionForwards and ActionMappings to keep control-flow decisions out of presentation layer.

### What configuration files are used in Struts?

ApplicationResources.properties  
struts-config.xml

These two files are used to bridge the gap between the Controller and the Model.

**How exceptions are handled in Struts application?**

There are two ways of handling exception in Struts:

**Programmatically handling:** using try {} catch block in code where exception can come and flow of code is also decided by programmer .its a normal java language concept.

**Declarative handling: There are two ways again either we define <global-Exception> tag inside struts config.xml file**

```
<exception  
    key="stockdataBase.error.invalidCurrencyType"  
    path="/AvailbleCurrency.jsp"  
    type="Stock.account.IllegalCurrencyTypeException">  
</exception>
```

Programmatic and Declarative way is some time also asked as followup questions given candidate's response on knowledge on Struts.

**Key:** The key represent the key present in MessageResource.properties file to describe the exception.

**Type:** The class of the exception occurred.

**Path:** The page where the controls are to be followed in case exception occurred.

**What helpers in the form of JSP pages are provided in Struts framework?**

--struts-html.tld  
--struts-bean.tld  
--struts-logic.tld

**Is Struts efficient?**

The Struts is not only thread-safe but thread-dependent(instantiates each Action once and allows other requests to be threaded through the original object).

ActionForm beans minimize subclass code and shorten subclass hierarchies

The Struts tag libraries provide general-purpose functionality

The Struts components are reusable by the application

The Struts localization strategies reduce the need for redundant JSPs

The Struts is designed with an open architecture--subclass available

The Struts is lightweight (5 core packages, 5 tag libraries)

The Struts is open source and well documented (code to be examined easily)

The Struts is model neutral

**How you will enable front-end validation based on the xml in validation.xml?**

The < html:javascript > tag to allow front-end validation based on the xml in validation.xml. For example the code: < html:javascript formName=logonForm dynamicJavascript=true staticJavascript=true /> generates the client side java script for the form logonForm as defined in the validation.xml file. The < html:javascript > when added in the jsp file generates the client site validation script.

**What is ActionServlet?**

The class org.apache.struts.action.ActionServlet is the called the ActionServlet. In the Jakarta Struts Framework this class plays the role of controller. All the requests to the server goes through the controller. Controller is responsible for handling all the requests.

**How you will make available any Message Resources Definitions file to the Struts Framework Environment?**

Message Resources Definitions file are simple .properties files and these files contains the messages that can be used in the struts project. Message Resources Definitions files can be added to the struts-config.xml file through < message-resources /> tag. Example: < message-resources parameter=MessageResources />

**What is Action Class?**

The Action Class is part of the Model and is a wrapper around the business logic. The purpose of Action Class is to translate the HttpServletRequest to the business logic. To use the Action, we need to Subclass and overwrite the execute() method. In the Action Class all the database/business processing are done. It is advisable to perform all the database related stuffs in the Action Class. The ActionServlet (command) passes the parameterized class to Action Form using the execute() method. The return type of the execute method is ActionForward which is used by the Struts Framework to forward the request to the file as per the value of the returned ActionForward object.

**Write code of any Action Class?**

Here is the code of Action Class that returns the ActionForward object.

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

public class TestAction extends Action
{
    public ActionForward execute(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response) throws Exception
    {
        return mapping.findForward("testAction");
    }
}
```

**What is ActionForm?**

An ActionForm is a JavaBean that extends org.apache.struts.action.ActionForm. ActionForm maintains the session state for web application and the ActionForm object is automatically populated on the server side with data entered from a form on the client side.

**What is Struts Validator Framework?**

Struts Framework provides the functionality to validate the form data. It can be used to validate the data on the users browser as well as on the server side. Struts Framework emits the java scripts and it can be used validate the form data on the client browser. Server side validation of form can be accomplished by sub classing your From Bean with DynaValidatorForm class. The Validator framework was developed by David Winterfeldt as third-party add-on to Struts. Now the Validator framework is a part of Jakarta Commons project and it can be used with or without Struts. The Validator framework comes integrated with the Struts Framework and can be used without doing any extra settings.

**How validation is performed in struts application?**

Ans: Another classic Struts interview question it's higher on level than previous interview questions because it's related to important validation concept on web application. In struts validation is performed using validator framework, Validator Framework in Struts consist of two XML configuration files.

1. **validator-rules.xml** file: which contains the default struts pluggable validator definitions. You can add new validation rules by adding an entry in this file. This was the original beauty of struts which makes it highly configurable.

2. **Validation.xml** files which contain details regarding the validation routines that are applied to the different Form Beans.

**Give the Details of XML files used in Validator Framework?**

The Validator Framework uses two XML configuration files validator-rules.xml and validation.xml. The validator-rules.xml defines the standard validation routines, these are reusable and used in validation.xml. to define the form specific validations. The validation.xml defines the validations applied to a form bean. How you will display validation fail errors on jsp page? - The following tag displays all the errors: < html:errors />

**Why do we need Struts?**

Java technologies give developers a serious boost when creating and maintaining applications to meet the demands of today's public Web sites and enterprise intranets. Struts combines Java Servlets, Java ServerPages, custom tags, and message resources into a unified framework. The end result is a cooperative, synergistic platform, suitable for development teams, independent developers, and everyone in between.

**How does Struts work?**

Java Servlets are designed to handle requests made by Web browsers. Java ServerPages are designed to create dynamic Web pages that can turn billboard sites into live applications. Struts uses a special Servlet as a switchboard to route requests from Web browsers to the appropriate ServerPage. This makes Web applications much easier to design, create, and maintain.

**Is Struts compatible with other Java technologies?**

Yes. Struts is committed to supporting industry standards. Struts acts as an integrator of Java technologies so that they can be used in the "real world".

**When would you use the **ForwardAction**?**

The **ForwardAction** class is useful when you're trying to integrate Struts into an existing application that uses Servlets to perform business logic functions. You can use this class to take advantage of the Struts controller and its functionality, without having to rewrite the existing Servlets. Use **ForwardAction** to forward a request to another resource in your application, such as a Servlet that already does business logic processing or even another JSP page. By using this predefined action, you don't have to write your own Action class. You just have to set up the struts-config file properly to use **ForwardAction**.

**When would you use the **IncludeAction**?**

The **IncludeAction** class is useful when you want to integrate Struts into an application that uses Servlets. Use the **IncludeAction** class to include another resource in the response to the request being processed.

**What is the difference between **ForwardAction** and **IncludeAction**?**

The difference is that you need to use the **IncludeAction** only if the action is going to be included by another action or jsp. Use **ForwardAction** to forward a request to another resource in your

application, such as a Servlet that already does business logic processing or even another JSP page.

#### **What are some benefits of using DynaActionForm?**

The biggest advantage is that - we need not create multiple classes to hold form information. We can just declare forms as and when required inside the struts-config.xml file and we are good to go.

#### **Can you think of any drawbacks of the DynaActionForm?**

Of Course. Every coin has two sides. Some drawbacks of using DynaActionForm could be:

1. The DynaActionForm bloats up the Struts config file with the xml based definition. This gets annoying as the Struts Config file grow larger and the config file in itself could become unmanageable
2. The DynaActionForm is not strongly typed as the ActionForm. This means there is no compile time checking for the form fields. Detecting them at runtime is painful and makes you go through redeployment.
3. ActionForm can be cleanly organized in packages as against the flat organization in the Struts Config file.

#### **What is LookupDispatchAction?**

The LookupDispatchAction is a subclass of DispatchAction. It does a reverse lookup on the resource bundle to get the key and then gets the method whose name is associated with the key into the Resource Bundle.

#### **When would you use LookupDispatchAction?**

LookupDispatchAction is useful if the method name in the Action is not driven by its name in the front end, but by the Locale independent key into the resource bundle. This is typically useful when you create something that will be shared across different locales. For ex: The login page of an international banks website would look surprisingly similar in almost all countries. It does the same thing in all countries but what exactly happens after you finish the login depends on the country you are. So, this is the kind of scenario where LookupDispatchAction would come in handy.

#### **What is difference between LookupDispatchAction and DispatchAction?**

The difference between LookupDispatchAction and DispatchAction is that the actual method that gets called in LookupDispatchAction is based on a lookup of a key value instead of specifying the method name directly.

#### **What is SwitchAction?**

The SwitchAction class provides a means to switch from a resource in one module to another resource in a different module. SwitchAction is useful only if you have multiple modules in your Struts application.

#### **What if element has declaration with same name as global forward?**

In this case the global forward is not used. Instead the element's takes precedence.

#### **What is DynaActionForm?**

DynaActionForm is a specialized subclass of ActionForm that allows the creation of form beans with dynamic sets of properties (configured in configuration file), without requiring the developer to create a Java class for each type of form bean. Since the developer need not create actual bean java classes, a lot of developers prefer DynaActionForms.

**What is the Difference between DispatchAction and LookupDispatchAction in Struts Framework?**

Dispatch Action	LookupDispatchAction
It's a parent class of LookupDispatchAction	Subclass of Dispatch Action
DispatchAction provides a mechanism for grouping a set of related functions into a single action, thus eliminating the need to create separate actions for each function.	An abstract <b>Action</b> that dispatches to the subclass mapped executes method. This is useful in cases where an HTML form has multiple submit buttons with the same name. The button name is specified by the parameter property of the corresponding ActionMapping.
If not using Internalization functionality then dispatch action is more useful.	Lookup Dispatch Action is useful when we are using Internalization functionality
DispatchAction selects the method to execute depending on the request parameter value which is configured in the xml file.	<b>LookupDispatchAction</b> looks into the resource bundle file and find out the corresponding key name. We can map this key name to a method name by overriding the getKeyMethodMap() method.
<b>DispatchAction</b> is not useful for I18N	LookupDispatchAction is used for I18N

#### **When do I need "struts.jar" on my classpath?**

When you are compiling an application that uses the Struts classes, you must have the "struts.jar" on the classpath your compiler sees -- it does not have to be on your CLASSPATH environment variable.

Why is that an important distinction? Because if you are using a servlet container on your development machine to test your application, the "struts.jar" must not be on your CLASSPATH environment variable when running the container. (This is because each Web application must also have their own copy of the Struts classes, and the container will become confused if it is on the environment path as well.)

There are several general approaches to this issue:

- \* Use ANT for building your projects -- it can easily assemble classpaths for the compiler. (This is how Struts itself is built, along with Tomcat and most other Java-based projects).
- \* Use an IDE where you can configure the "class path" used for compilation independent of the CLASSPATH environment variable.
- \* Use a shell script that temporarily adds struts.jar to the classpath just for compilation, for example javac -classpath /path/to/struts.jar:\$CLASSPATH \$@

#### **Does Struts include its own unit tests?**

Struts currently has two testing environments, to reflect the fact that some things can be tested statically, and some really need to be done in the environment of a running servlet container. For static unit tests, we use the JUnit framework. The sources for these tests are in the "src/test" hierarchy in the source repository, and are executed via the "test.junit" target in the top-level build.xml file. Such tests are focused on the low-level functionality of individual methods, are particularly suitable for the static methods in the org.apache.struts.util utility classes. In the test

hierarchy, there are also some "mock object" classes (in the org.apache.struts.mock package) so that you can package up things that look like servlet API and Struts API objects to pass in as arguments to such tests.

Another valuable tool is Struts TestCase which provides a useful harness for Action classes that can be used with JUnit or Cactus.

**If the framework doesn't do what I want, can I request that a feature be added?**

First, it's important to remember that Struts is an all-volunteer project. We don't charge anyone anything to use Struts. Committers and other developers work on Struts because they need to use it with their own applications. If others can use it too, that's "icing on the cake". If you submit a patch for a feature that a Committer finds useful, then that Committer may choose to volunteer his or her time to apply the patch. If you just submit an idea without a patch, it is much less likely to be added (since first someone else has to volunteer their time to write the patch).

We are grateful for any patches, and we welcome new ideas, but the best way to see that something gets added to the framework is to do as much of the work as you can, rather than rely on the "kindness of strangers". Worst case, you can apply the patch to your copy of Struts and still use the feature in your own application. (Which is what open source is ~really~ all about.)

**Where can I get help with Struts?**

The Struts package comes complete with a Users Guide to introduce people to the framework and its underlying technologies. Various components also have their own in-depth Developers Guide, to cover more advanced topics. Comprehensive Javadocs are included along with the full source code. For your convenience, these are bundled together as a self-installing application. The struts-documentation.war is the same bundle that is deployed as the Struts Web site. The Strut's mailing list is also very active, and welcomes posts from new users. Before posting a new question, be sure to consult the MAILING LIST ARCHIVE and the very excellent How To Ask Questions The Smart Way by Eric Raymond. Please do be sure to turn off HTML in your email client before posting.

**What's the difference between Struts and Turbine? What's the difference between Struts and Espresso?**

If you are starting from scratch, packages like Turbine and Espresso can be very helpful since they try to provide all of the basic services that your team is likely to need. Such services include things like data persistence and logging.

If you are not starting from scratch, and need to hook up your web application to an existing infrastructure, then "plain vanilla" Struts can be a better choice. The core Struts framework does not presuppose that you are using a given set of data persistence, presentation, or logging tools. Anything goes =:0)

Compared to other offerings, Struts endeavors to be a minimalist framework. We try leverage existing technologies whenever we can and provide only the missing pieces you need to combine disparate technologies into a coherent application. This is great when you want to select your own tools to use with Struts. But, if you prefer a more integrated infrastructure, then packages like Turbine or Espresso (which uses Struts) are perfectly good ways to go.

See also

- \* < <http://www.mail-archive.com/struts-user@jakarta.apache.org/msg03206.html> >
- \* < <http://www.mail-archive.com/general@jakarta.apache.org/msg00495.html> >
- \* < <http://jakarta.apache.org/velocity/ymtd/ymtd.html> >

**Why doesn't the focus feature on the <html:form> tag work in every circumstance?**

Unfortunately, there is some disagreement between the various browsers, and different versions of the same browser, as to how the focus can be set. The <html:form> tag provides a quick and

easy JavaScript that will set the focus on a form for most versions of most browsers. If this feature doesn't work for you, then you should set the focus using your own JavaScript. The focus feature is a convenient "value-add" -- not a core requirement of the tag. If you do come up with a JavaScript that provides the final solution to this project, please post your patch to this Bugzilla ticket.

**What is the use of DispatchAction?**

DispatchAction is an action that comes with Struts 1.1 or later, that lets you combine Struts actions into one class, each with their own method. The org.apache.struts.action.DispatchAction class allows multiple operation to mapped to the different functions in the same Action class. How you will display validation fail errors on jsp page?

Following tag displays all the errors:

```
<html:errors/>
```

How you will enable front-end validation based on the xml in validation.xml?

The <html:javascript> tag to allow front-end validation based on the xml in validation.xml. For example the code: <html:javascript formName=\"logonForm\" dynamicJavascript=\"true\" staticJavascript=\"true\" /> generates the client side java script for the form \"logonForm\" as defined in the validation.xml file. The <html:javascript> when added in the jsp file generates the client site validation script.

Difference between Struts 1.x and Struts2.x.

**Component Differences**

In struts 1.x front controller is ActionServlet

In 2.x front controller is FilterDispatcher

In struts 1.x we have RequestProcessor class

In 2.x we have Interceptors instead RequestProcessor will see about this concept later just remember as of now

In struts 1.x we have multiple tag libraries like, html, logic, bean..etc

In 2.x we do not have multiple libraries, instead we have single library which includes all tags

In struts 1.x the configuration fine name can be [any name].xml and we used to place in web-inf folder

In 2.x the configuration file must be struts.xml only and this must be in classes folder

In struts 1.x we have form beans and Action classes separately

In 2.x form bean, Action classes are combinedly given as Action class only, of course we can take separately if we want

In struts 1.x properties file must be configured in struts-config.xml

But in 2.x we need to configure our resource bundle(s) in struts.properties file

In struts 1.x we have programmatic and declarative validations only

In 2.x we have annotations support too along with programmatic and declarative validations

***Functional Differences***

In struts 1.x declarative validations are done by using validation frame work

In 2.x, declarative validations are done by using xwork2 frame work by webwork the reason being, its support valuations through Annotations

In struts 1.x an Action class is a single ton class, so Action class object is not a thread safe, as a programmer we need to make it as thread safe by applying synchronization

In 2.x an Action class object will be created for each request, so it is by default thread safe, so we no need to take care about safety issues here

In struts 1.x we have only **jsp** as a view technology

In 2.x we have support of multiple view technologies like velocity, Freemarker, jasper reports, jsp bla bla

In struts 1.x Action class is having servlet dependency, because in execute() method accepts req,res parameter right ! so.

In 2.x Action class doesn't have any servlet dependency, because its execute() method doesn't accept any parameters, however we can access all servlet objects with dependency injection

***What are differences between < bean:message > and < bean:write >?***

< bean:message >: is used to retrieve keyed values from resource bundle. It also supports the ability to include parameters that can be substituted for defined placeholders in the retrieved string.

< bean:message key="prompt.customer.firstname" />

< bean:write >: is used to retrieve and print the value of the bean property. < bean:write > has no body.

< bean:write name="customer" property="firstName" />

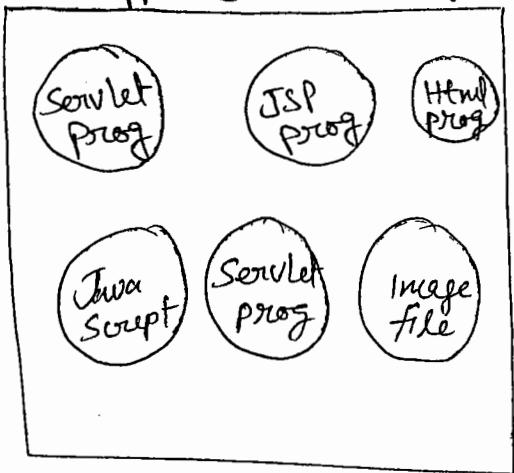
Introduction :-

- Web application is collection of web resource program having the capability to generate web pages.
- The web appl<sup>n</sup> that is hosted on internet network having domain name like www.xyz.com is called website.
- Based on the content they generate in web pages there are two types of web resource programs.
  - Static web resource program generate static web page. (fixed content) like info file, licence page.  
Ex:- Html
  - Dynamic web resource program generate dynamic web page (contain varying content)

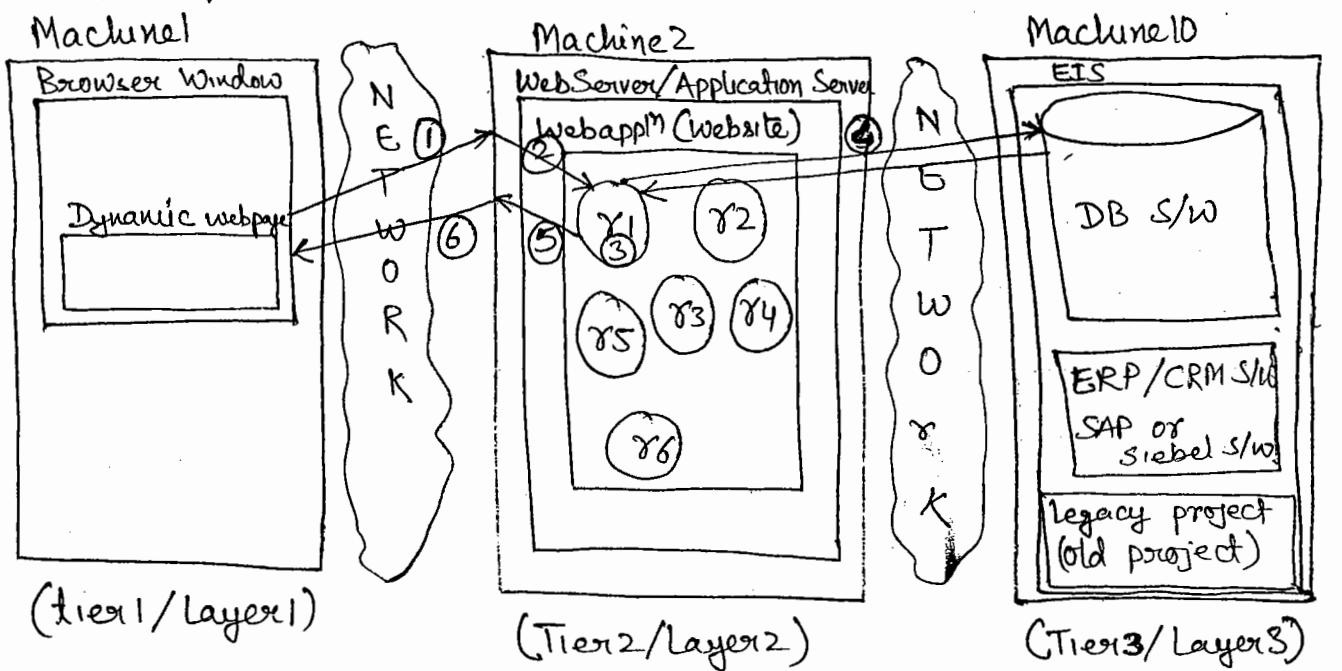
Ex:- Servlet program, jsp program, asp program...

- The content of dynamic web page changes based on the input value of requests or time of request generation.
- Every web application is collection of both static and dynamic web resource programs.
- The web resource program like img file, javascript file cannot generate web pages but they help other web resource program.

## Webappl<sup>n</sup> (Java Webappl<sup>n</sup>)



→ Setup of web application development and execution :-



EIS → Enterprise information System.

web  
r = Resource program

→ We can develop web application either as two-tier appl<sup>n</sup>  
or 3-tier appl<sup>n</sup>  
with backend.

→ When it is developed as a 2-tier application it  
looks like thin client - Fat server appl<sup>n</sup>.

→ Multiple back-end s/w together is called as EIS.  
(Enterprise Information system)

→ SAP is a ERP S/w (Enterprise resource program)

→ Siebel is a CRM S/w (Customer relation management)

→ The project that is developed by using old technologies or outdated version of existing technologies is called legacy project.

Ex:- Corba project, cobol project, java 1.1 project etc...

→ Based on the place where web resource program executes there are two types of web resource program.

I) Server side web resource program

→ Goes to Browser window from Executes in server itself when requested. Ex:- Servlet program, Jsp programs, etc...

II) Client side web resource program.

→ Goes to browser window for execution from web appl'n when requested.

Ex:- html program, javascript code etc...

→ All server side programs are generally Dynamic web resource program.

→ All client side programs are generally static web resources programs.

### Note

Decide whether web resource program is server side program or client side program based on the place where it executes not based on the place where it resides.

→ The web resources program of web application contains the following logics.

26.07.13

- I) Request data Gathering logic
- II) Form validation logic
- III) Service logic / Business logic
- IV) Persistence logic
- V) Presentation logic
- VI) Session management logic
- VII) Middleware service  
and etc...

### ① Request data Gathering logic:

→ The logic that gathers various details from client generated request like form data header values, miscellaneous information by calling various method on request object is called request data gathering

### ② Form validation logic:

The logic that validate the form data by verifying pattern and format of the form data is called form validation logic

Ex → Checking whether username password typed or not

### ③ Service logic / Business logic:

The main logic of application that generates results by gathering inputs is called business logic

Ex → Credit card / debit card processing etc...

#### ④ Persistence logic

→ The logic that manipulates database data is called persistence logic

Ex:- Jdbc code, hibernate code etc...

#### ⑤ Presentation logic

→ The logic that gives user interface to end user to provide inputs and to view results is called presentation logic

#### ⑥ Session management logic

→ The logic that makes web application to remember client data ~~across~~ across the multiple request during a session is called ~~as~~ session management logic

Ex:- Hidden ~~for~~ fields, cookies, HttpSession with cookies, HttpSession with rewriting are session tracking techniques to perform session management.

#### ⑦

→ The additional logic applied on the application to make our application more powerful, re-usable and effective are called middleware services.

Ex:- Transaction management, security, login etc...

### Different models of java web app<sup>n</sup> development

#### Model1

In this all the main web resources prog of web app<sup>n</sup> contains multiple logics. More even if some let prog used as web resource → then jsp prog will not be used as web resource prog and vice-versa

(Model view control)

#### Model2 (MVC)

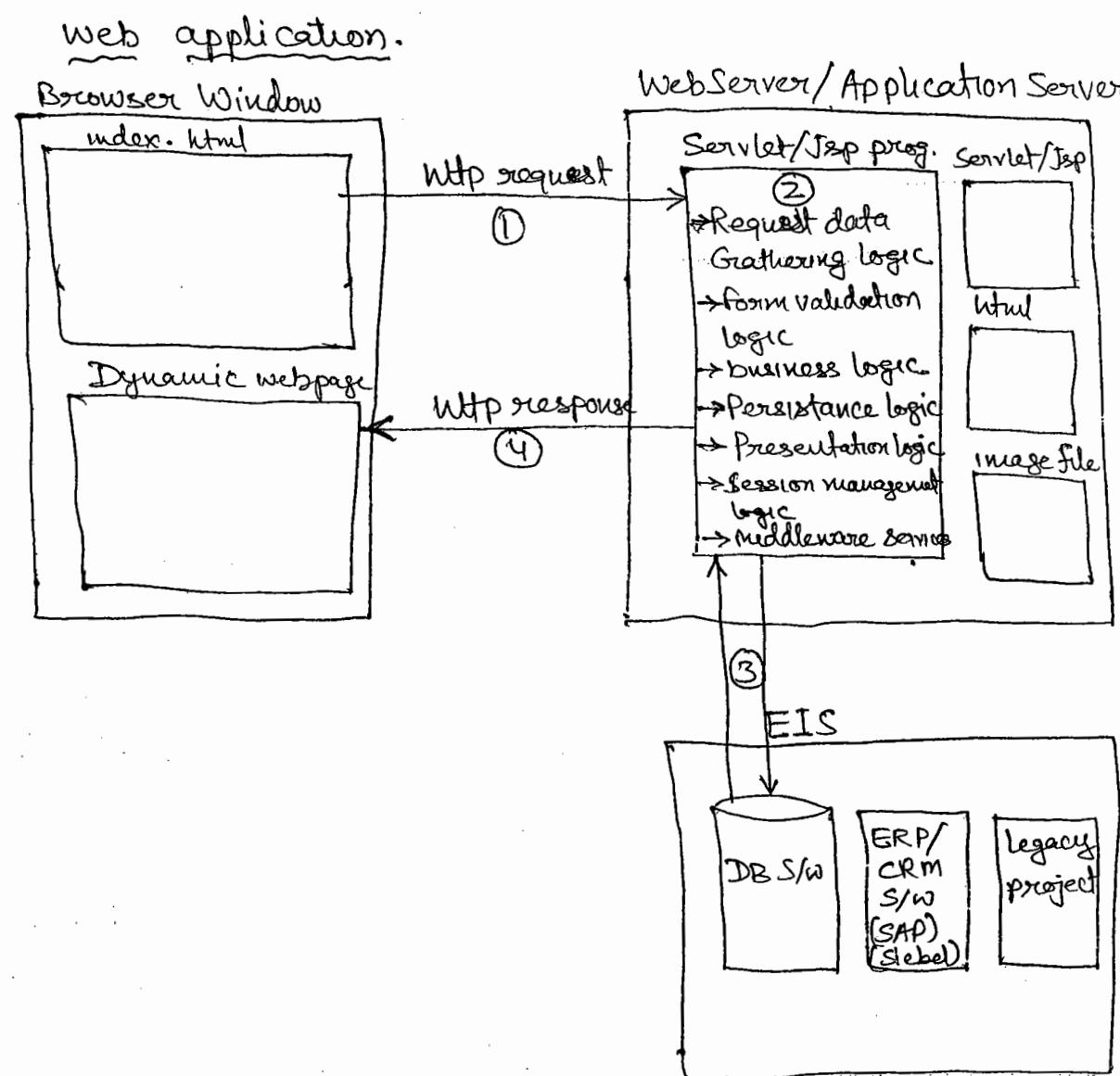
##### MVC1

##### MVC2

In MVC1 and MVC2 architecture we use multiple technologies and multiple layers to develop the web

- Either servlet prog or jsp page prog act as main web resource → prog of web appln.
- Industry prefers to use jsp prog while developing model 1 architecture base web appln.
- This model 1 architecture is page centric that means each web resource prog directly generates one web page.

### Understanding Model 1 architecture to develop java web application.



- In model architecture every main resource prog (either ~~servlet~~ or Jsp) contains multiple logics due to that model architecture is having following advantages and limitation.

### Limitation/ Disadvantages :-

- Since every main web resource program contains ~~a~~ multiple logic so we can say there is no clear separation between logics
- Modification done in one kind of logics may disturb other kind of logics
- Maintenance and enhancement of the project becomes quite complex.
- Parallel development is not possible so productivity is very poor.
- Built in middle ware services are not available in huge scale so certain middleware services must be implemented by the programmer manually. This increases burden on the programmer.

### Advantages :-

- Since the parallel development is not possible, multiple programmers are not required to develop project
- Knowledge on either servlet or Jsp is sufficient to develop the projects
- Doing more work in less time with good accuracy is called productive
- Model is good to develop small scale java websites. (max 10 pages).

## MVC

M → Model layer (Business logic + persistence logic)

↳ Uses EJB, Spring, Sprout with Hibernate, Java Beans and other technologies to develop these logics.

Ex → It is like Accounts officer in company.

V → View layer (Presentation logic)

↳ Uses JSP, HTML, Velocity, freemarker and etc.. to develop these logics

Ex → It is like beautician.

C → Controller layer (Integration logic / connectivity logic)

↳ Uses servlet / servlet filter to develop these logics.

Ex → It is like traffic police, supervisor.

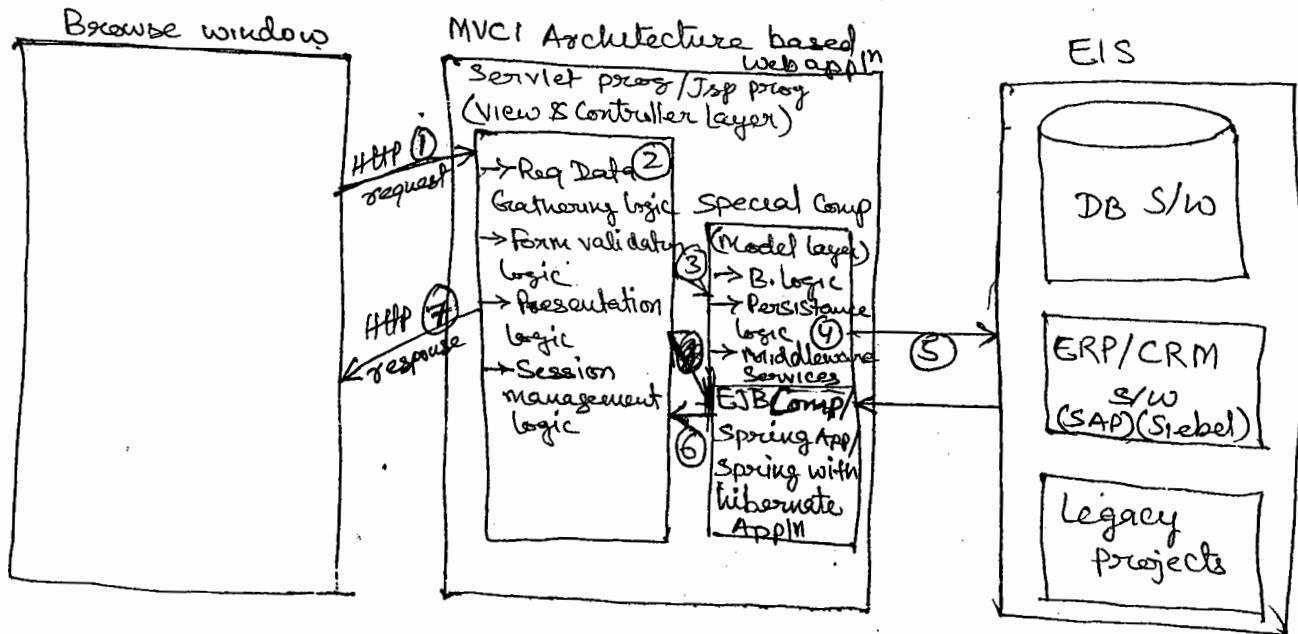
→ The logic that monitors and controls all the operation of the application is called integration logic. This logic takes request from client, passes the request to model layer, gathers the result from model layer resources and passes the results to view layer resources.

→ In MVC1 architecture single resource (file or prog) contains View layer and controller layer logics whereas separate resources will be taken for model layer logics.

→ In MVC2 architecture we take separate resources in View layer, separate resources for controller layer, and separate resources for model layer.



## Understanding MVC1 architecture:-



- In MVC1 architecture we can see a clean separation between logics compare to Model1 architecture because we keep business logic and persistence logic separate Model layer resources.
- To achieve more clean separation logic it is recommended to use MVC2 architecture
- Prefer using MVC1 architecture to develop medium scale web sites (max of 30 pages)

29.07.13

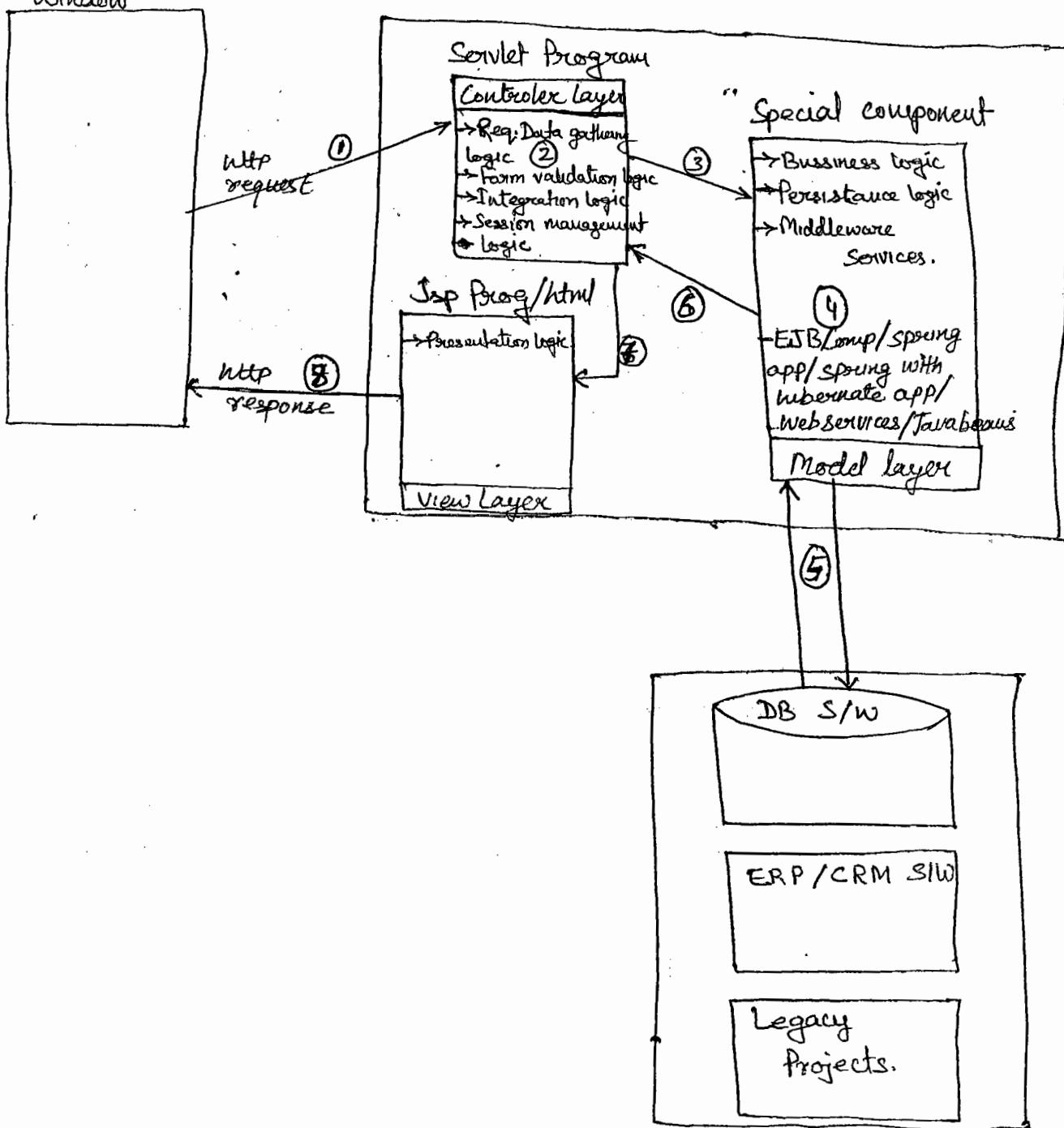
## UNDERSTANDING MVC2 ARCHITECTURE:-

With respect to diagram:-

- ① Browser window gives request to web application.
- ② The controller servlet traps and takes the request.
- ③ Servlet program reads the request data and uses the integration logic to pass the request to model layer Business component.

Browser  
Window

## MVC2 ARCHITECTURE :-



- ④ and ⑤ The model layer component will process the request and interacts with back-end software if necessary.
- ⑥ The results generated by model layer comes back to controller servlet.
- ⑦ Controller servlet uses integration logic to pass the results to view layer resource.

③ The view layer resource formats the result and sends formatted results to browser window as response.

#### Note

- MVC2 arch. is good to develop large scale java based web applications.
- MVC2 arch. is industries defacto standard. To develop large scale java web application.

#### → Advantages of MVC2

- Since, there are multiple layers to develop multiple logics we can say there is a clean separation between logics.
- Modification done in one layer logic does not affect other layer logic. This makes maintenance and enhancement of the project quite easy.
- Parallel development is possible so productivity is good. It is industry defacto standard to develop the web application.
- EJB, Spring technology of model layer gives build-in middleware services this reduces burden on the programmer.

#### Disadvantages :-

- For parallel development multiple programmer are required.
- Knowledge in multiple technology is required.

Q:- How can be achieve parallel development with MVC2 architecture ?

Ans:- The P.L divides the team into two parties

- 1) Web authors → Use servlet/Jsp Tech. to develop View and controller logic
- 2) JEE developers → Use JEE modules (EJB, JMS technology) and other technologies like Spring, hibernate, web services to develop module layer logics.

Since, these two parties can work parallelly we can say the parallel development is possible with MVC2 architecture.

- while developing MVC2 arch. based project we need to follow set of rules they are:-
- I) Every layer is designed to have specific logic so donot place any extra logics in the layers.
- II) All the operations of web application must take place under the control of controller servlet.
- III) There can be multiple resources in view layer, multiple resources in model layer but there must be one controller servlet. in controller layer.
- IV) Model layer resources must not talk with view layer resources directly and vice-versa. These resources should talk through controller servlet.
- V) The two resources of view layer should not talk with each other directly they much ~~attract~~ interact with each other through controller servlet.

30.07.13

#### DESIGN PATTERN PATTERN :-

~~Design~~ Design patterns are set of rules which are given as best solution for reoccurring problems of application development.

MVC2 is given as design-pattern in initial days later it has become industries defacto standard to develop java web application and to create s/w like struts.

Based on this people starts calling MVC2 is architectures to develop web application.

### FRAMEWORK :-

→ JDBC, servlet, JSP, JMS and etc... are ~~core~~ core technology to develop java application. While working with these core technologies we must take care of all the logics of all the layers from scratch level.

Def:-

1. → Framework is a special s/w that is built on core technologies having the capability to generate common logics of the application dynamically based on programmer supplied application specific logic.

2. → Framework is a special s/w that provides abstraction layer on core technologies programming and simplifies application development process.

→ Abstraction layer means hiding the implementation that means every framework internally uses multiple core technologies during execution but it never makes programmer to worry about that this is nothing but getting abstraction layer on core technologies.

→ Struts is a framework that is developed based on servlet, JSP core technologies having the capability to generate integration logic, <sup>of controller layer</sup> dynamically based on the logics supplied in view layer, model layer.

→ If MVC2 architecture based application is developed by using ~~core~~ servlet, JSP technologies then all the logics all the 3 layers must be developed by the programmer manually. If same web app<sup>n</sup> is developed by using Struts the programmer just need to develop view, model layer

logic and struts frameworks dynamically generates the controller layer logic.

→ There are three types of frameworks

i) Web framework S/w.

→ Given to develop mvc2 architecture based web app<sup>n</sup> by getting abstraction layer on servlet, jsp technologies.

Abstraction layer → hiding core technology.

Ex:- Struts → from Apache. (1)

JSF → from Sun Ms. (1)

Spring mvc → from interface 21. (3)

webwork → from OpenSymphony.

ADF → from Oracle ~~APP~~ Corp. (4) (Application development P/w)  
---

ii) ORM framework S/w

→ Provides abstraction layer on jdbc core technologies and allows us to develop o-r-mapping persistency logic using objs as DB S/w independent persistency logic.

Ex:-  
hibernate → from SoftTree (Redhat) ①

Toplink → from Oracle Corp ④

OJB → from Apache

iBatis → from Apache ②

OpenJPA → from Sun Ms ③

JDO → from Adobe

---

### 3) Java-JEE framework (Application F/W) :-

→ Provides abstraction layer on multiple core technologies (Java-JEE) and allows to develop different kinds of applications and different kinds of logics.

Ex: Spring → from Interface21.

~~SPRING~~

#### STRUTS

- Type: web framework to develop MVC2 architecture java web applications.
- Version: 1.3.x (Compatible with jdk 1.4+) 2.3.x (Compatible with Jdk 1.5+)
- Vendor: Apache foundation
- Open source framework S/w.
- Creator: Mr. craig
- To download: Download as zip file from www.apache.org website
- To install: Extract zip file to a folder
- For Docs and FAQs: www.apache.org
- Online Tutorial: www.roseindia.net, www.tutorialpoint.com
- Reference Books: Jakarta struts from O'reilly (from struts1.x)  
: Black book of struts.

→ To install struts 1.x s/w extract struts-1.3.8-all.zip file

→ In struts based web appl<sup>n</sup> the predefined servlet of Struts api will be taken as the controller servlet of web application.

→ Struts s/w is given based on MVC2 architecture so it can be used only to develop MVC2 architecture based java web applications.

- To develop small and medium scale java web appl'n use servlet, jsp as direct core technologies.
- To develop MVC2 architecture based large scale java web application use struts framework.
- <struts1.x-home>\App folder gives war files representing ~~ex~~ example application.
- <struts1.x-home>\lib folder gives Struts1.x libraries(jars)
- <struts1.x-home>\docs folder gives Documentation
- <struts1.x-home>\src folder gives source code of Struts S/W.

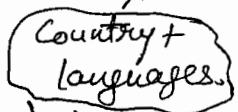
### Struts1.x S/W gives :-

- Base Struts framework
- Documentation
- Two plugins (Validator plugin, Tiles plugin)
  - ↑ To design web page with layout
  - ↓ for form validation
  - ↓ for layout management
- JSP tag libraries having Jsp tags
- Source code
- Example application etc...

- plugin is a patch S/W or SW application that can enhance the functionality of existing S/W or SW appl'n.
- Jsp tag library is a library that contains set of readily available jsp tags.
- <struts1.x-home>\lib\struts-core-1.3.8.jar file represents Struts1.x API. And this jar files having multiple dependent jar files...

STRUTS not getting enough amount of time = not sufficient time 31.07.13

Defn

- Struts is an open source java based web framework given by apache foundation to develop MVC2 architecture based java web application.
- Features of Struts :-
- Allows us to develop MVC2 architecture based web applications.
- Simplifies large scale web application development by generating the integration layer of controller layer dynamically.
- Gives validator plugin having form validation logic to perform form validations.
- Gives Tiles plugin for better layout management.
- Allows to have multiple models.
- Gives 5 no. of JSP tag libraries having JSP tag to develop JSP program as Java code less JSP program.
- Gives built-in support for file uploading and file downloading.
- Allows most of the operations as declarative operation through XML entries.
- Gives facilities to implement MVC2 rules...
- Supports internationalization (I18n).
  - Make our application run in content in different locales  
is nothing but enable I18n.  

  - Make web application's enhancement and maintenance quite easy.
  - Supports parallel development and gives good productivity.

→ When java web application uses third-party API (other than Jdk api, servlet api, JSP api) then in its web resource program then the third-party API related main jar should be added to classpath and main and dependent classpath should be added to WEB-INF\lib folder. Here, the .jar files added to classpath will be used to recognize third-party API ~~do~~ during the compilation of web resource program. The jar files added to WEB-INF\lib folder will be used by servlet container to recognize and use third party API during the execution of web resource prog.

#### Example Scenario :-

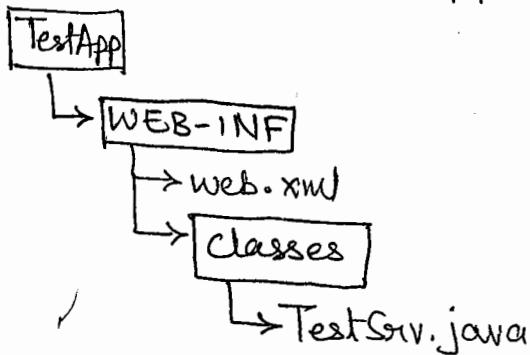
##### First.jar (main jar file)

```
//Test.java
public class Test
{
    public void m1()
    {
        Demo d1 = new Demo();
        d1.m2()
    }
}
```

##### Second.jar (Dependent jar file to First.jar)

```
// Demo.java
public class Demo
{
    public void m2()
    {
        -- -
        -- -
    }
}
```

## Example web application :-



### 1) TestServ.java

```
public class TestServ extends HttpServlet  
{  
    public void service(,--) throws SE,IOE  
    {  
        ---  
        ---  
        Test t1 = new Test()  
        t1.m1();  
    }  
}
```

#### ① Compilation

TestApp\WEB-INF\classes > javac TestServ.java (X) error:  
~~Test~~ cannot find symbols Test, HttpServlet and etc...

Solution: add first.jar file where (Test class) is available to classpath.  
also add servlet-api.jar file to classpath.

#### ②

TestApp\WEB-INF\classes > javac TestServ.java (success)  
→ Gives TestServ.class file

⇒ Deploy TestApp web application in Tomcat server.

① ⇒ Give request to TestServ program of TestApp  
web application  
<http://localhost:2013/TestApp/testurl> (X)

→ java.lang.NoClassDefFoundException : Test

Solution:- Add First.jar file to WEB-INF\lib folder of TestApp.

- ② → Give request to TestSrv program of TestApp web application

http://localhost:2013/TestApp/testurl (X) error.

problem:- java.lang.OutOfMemoryError: NoClassDefFoundException : Demo

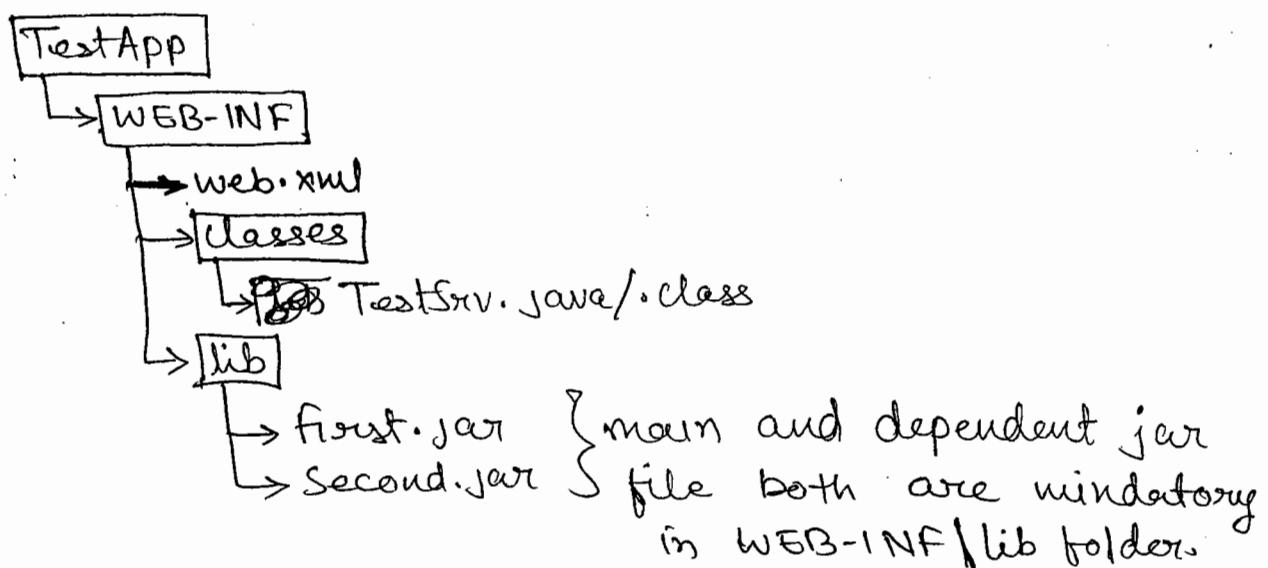
Solution:- Add Second.jar file to WEB-INF\lib folder of TestApp.

- ③ Give request to TestSrv program of TestApp web application

http://localhost:2013/TestApp/testurl (success)

→ Output follows here.

### Final Deployment Structure :-



→ Developing struts app<sup>n</sup> is nothing but developing normal java web app<sup>n</sup> by having struts api utilization in the web resource programs of java web app<sup>n</sup> for this the struts api related main jar file

Struts-core-1.3.8.jar file should be added to classpath and struts api related main and dependent jar files (struts-core-1.3.8.jar file and its dependent jar files) must be added to WEB-INF \ lib folder

01.08.13

### Resources of the Struts1.x Application :-

1. Jsp programs (View Layer)
2. ActionServlet (Controller Layer)
3. web.xml
4. FormBean/ActionForm (Controller Layer)
5. Action class (Controller Layer / Model Layer)
6. Action forwards (Controller Layer)
7. Struts configuration file ~~and~~ (Controller layer)  
and etc...

#### 1. Jsp Programs:-

→ These are view layer resources having presentation logic. These resources provide user interface to end user to supply inputs and to view results. It is recommended to develop these Jsp program as java codeless Jsp program by using following tags.

- Jsp tag (excluding scriptlet, expression, declaration tags)
- ~~JSTL~~ JSTL tags
- ~~Struts~~ Struts supplied jsp tags
- Third party supplied jsp tags
- custom jsp tags.

#### 2. Action Servlet

It is built-in servlet of Struts API acting as controller servlet of Struts based web appl'. The integration logic of

this Action Servlet will be generated dynamically based on configuration done in struts configuration file.

- Action servlet acts as entry and exit point of struts application so it is called as front controller of web application.
- Action Servlet is responsible to take request from client, to pass response request to model layer resources, to gather the result from model layer resources and to pass the results to view layer resources. Action Servlet class is given extending from HttpServlet class.

### 3. web.xml

- It is deployment descriptor file of web application. The web.xml file of struts appln contains Action Servlet configuration, jsp tag libraries configuration, welcome file configuration, error pages configuration and etc...
- Even though Action Servlet is built-in servlet its configuration in web.xml file is mandatory. To make underlying servlet container recognizing and using Action Servlet.
- To make ActionServlet as front controller it must be configured in web.xml file either with directory match url pattern or extension ~~or~~ match url pattern.

### 4. Form Bean / Action Form :-

It is java Bean class extending from org.apache.struts.action. ActionForm class having capability to hold form data and to validate form data.

Actionform is a Abstract class without abstract method.

- The java class with getter and setter method is called JavaBeans.
- Action servlet writes the received form data to FormBean class object so programmer can validate form data in FormBean class through form validation logic.

Q:- What is need of FormBean in struts application?

Ans:- According MVC2 arch. the controller servlet is responsible to validate form data. In Struts application ActionServlet is controller servlet and it is built-in servlet. So programmers cannot place form validation logic in ActionServlet. To overcome this problem ActionServlet writes received form data to the programmer supplied FormBean class. So, Programmer can place form validation logic in FormBean class to validate form data.

## 5. Action class :-

- The ordinary java class that extends from org.apache.struts.action.Action is called Struts Action class. Every request coming to Struts application will try to execute one or other Action class through ActionServlet.
- Struts Action class acts as model layer resource when business logic is directly placed in Action class. And it acts as controller layer resource when logic to communicate with other model layer resources like EJB components and spring and etc is placed in Action class.

### Note

In small scale Struts appln Action class will be used as model layer Resource. In large scale Struts applications

the Action class will be used as controller layer resource

- In MVC2 arch. based appl'n controller ~~is~~ <sup>Servlet</sup> is responsible to talk with model layer resources. but in Struts application the controller servlet is built-in ActionServlet so programmer cannot place logic to interact with model layer resources.
- To overcome this problem ActionServlet passes every request to one or other Action class, So. programmer can place the logic of interacting with model layer in Action class.

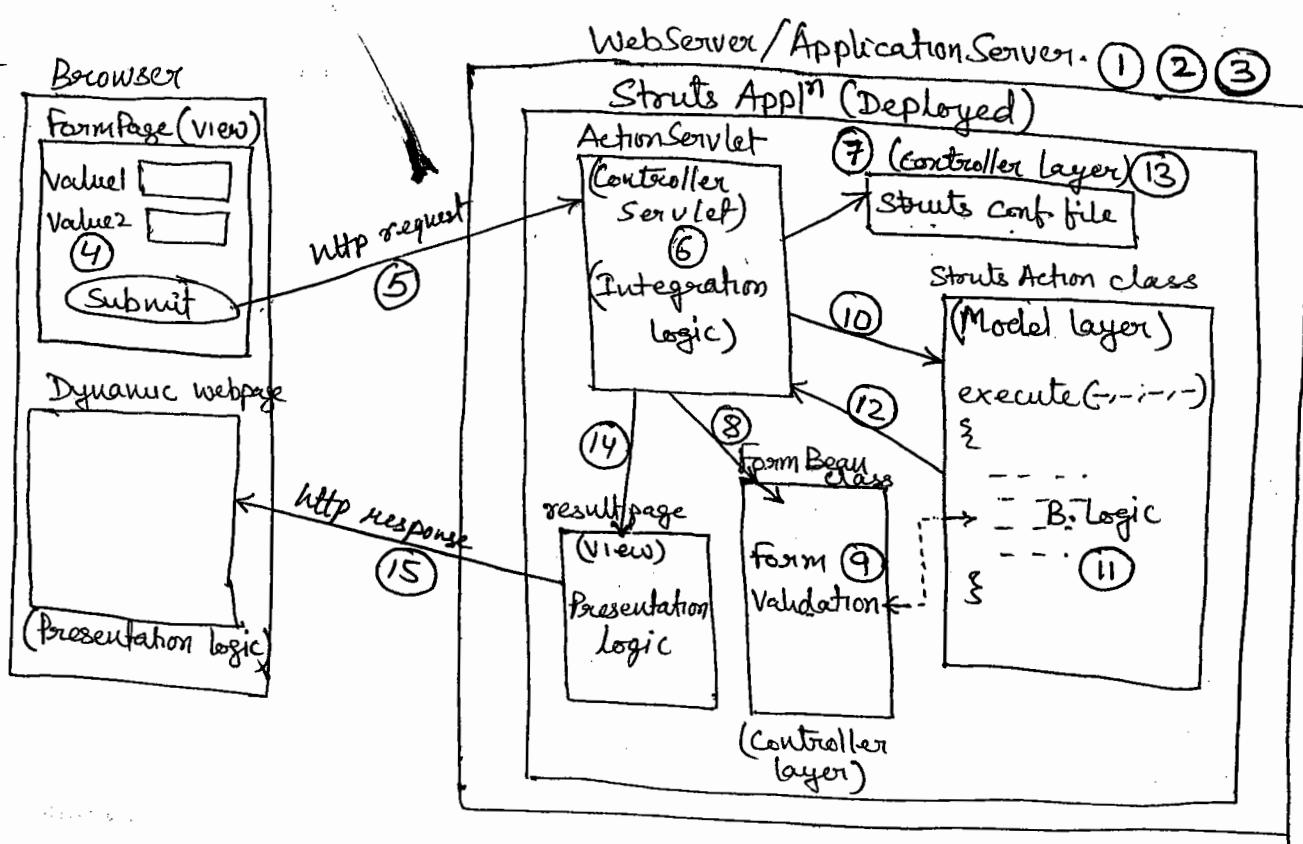
#### 6. ActionForwards :-

- These are XML entries in struts config file specifying the result pages of struts Action classes by using <forward> tags

#### 7. struts configuration file

- Any <file name>.xml can be taken as struts configuration file if no file name is specified struts-config.xml file of WEB-INF folder will acts as default configuration file name. This file is heart of struts appl'n because ActionServlet generate integration logic dynamically by using the instruction of this file.
- We configure struts resources in this file like FormBeans Action classes, properties files, plugins, ActionForward and etc...
- Underlying server managed servlet container reads web.xml file content.

→ ActionServlet of Struts App reads struts configuration file content.



### Note :-

- In above diagram Struts action class is taken as model layer resource by directly placing business logic in it.
- Every FormBean class is visible in Action class to make Action class reading the form data of form page.

With respect to above diagram :-

1. Programmer deploys Struts application in web server or application server.
2. Because of load-on-startup the servlet container creates object of ActionServlet either during server startup or during deployment of Struts appln.
3. ActionServlet verifies Struts config. file entries and also reads and stores its data.
4. End user launches the form page on browser window.

- ⑤⑥ ActionServlet traps and takes the form page generated request as front controller.
- ⑦ ActionServlet uses struts conf. file entries to decide the Form Bean and Action class that are required to process the request.
- ⑧⑨ ActionServlet writes the received form data to FormBean class obj and FormBean class executes form validation logic to validate the form data.
- ⑩ ActionServlet creates or locate struts Action class obj
- ⑪ ActionServlet calls execute() method on struts Action Class obj. This method uses the business logic to process the request.
- ⑫ execute() method return action forward object back to ActionServlet
- ⑬ ActionServlet uses struts conf. file entries (Action Forwards) to decide the result page of Action class.
- ⑭ ActionServlet forwards the result to result page.
- ⑮ Result page formats the result by using presentation logic. and sends output to browser window as dynamic web page.

### UNDERSTANDING FRONT CONTROLLER :-

→ Three types of url patterns:-

1. ExactMatch

→ url pattern must begin with "/" symbol and should not contain "\*" symbol.

Ex:- <url-pattern>/test</url-pattern>

request urls from Browser windows :-

http://localhost:2020/TestApp/test1 (valid)

http://localhost:2020/TestApp/test1.do (invalid)

http://localhost:2020/TestApp/xyz (invalid)

http://localhost:2020/TestApp/abc1 (invalid)

other example of Exact match url patterns :-

Ex1:- <url-pattern>/abc.c</url-pattern>

<url-pattern>/xyz.123</url-pattern>

<url-pattern>/x/test1</url-pattern>

2. Directory Match url-pattern :-

→ url pattern must begin with "/" symbol and must end with "\*" symbol.

Ex:- <url-pattern>/x/y/\*</url-pattern>

example request from Browser windows :-

http://localhost:2020/TestApp/x/y/abc (valid)

http://localhost:2020/TestApp/y/x/123 (invalid)

http://localhost:2020/TestApp/x (invalid)

http://localhost:2020/TestApp/x/y (valid)

http://localhost:2020/TestApp/x/y/123.c (valid)

http://localhost:2020/TestApp/a/b/x/y (invalid)

other example of directory match url-pattern :-

Ex1:- <url-patterns>/a/\*</url-pattern>

<url-pattern>/test1/test2/\*</url-pattern>

<url-pattern>/a/b.x/\*</url-pattern>

3. Extension match url pattern :-

→ url pattern must begin with "\*" symbol having extension word/letter.

Syntax:- #.<extension>

Ex:- \* <url-pattern> \*.do </url-pattern>

example url request from browser window.

Http://localhost:2020/TestApp/scyz.do (valid)

Http://localhost:2020/TestApp/xyz/z.do (valid)

Http://localhost:2020/TestApp/x/z.do/abc.do (valid)

Http://localhost:2020/TestApp/xc/j.x (invalid)

Http://localhost:2020/TestApp/my.scyz (invalid)

Http://localhost:2020/ff.x (invalid)

other example of extension match url pattern

Ex1:- <url-pattern> \*.abc </url-pattern>

Ex2:- <url-pattern> \*.durga </url-pattern>

Ex3:- <url-pattern> \*.doggy </url-pattern>

Ex4:- <url-pattern> \*.abc.xyz </url-pattern>

| <url-pattern> /test.c </url-pattern>

→ It is exact match url-pattern.

<url-pattern> /x/y/\*.do </url-pattern>

→ Invalid url-pattern formation.

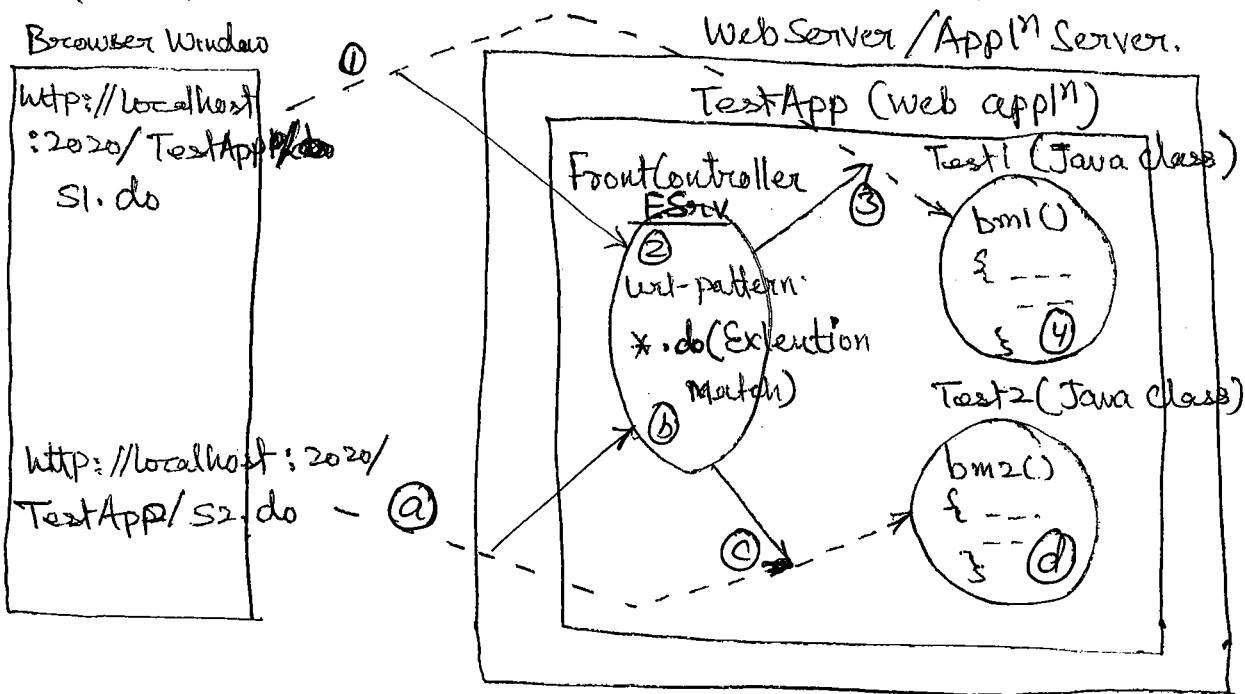
→ The Front

→ The special web resource program of web appl<sup>n</sup> that acts as entry and exit point of web application is called front controller. Java classes of web appl<sup>n</sup> cannot take http request directly so we make front controller (servlet, JSP prog) trapping http request and passing them to java classes.

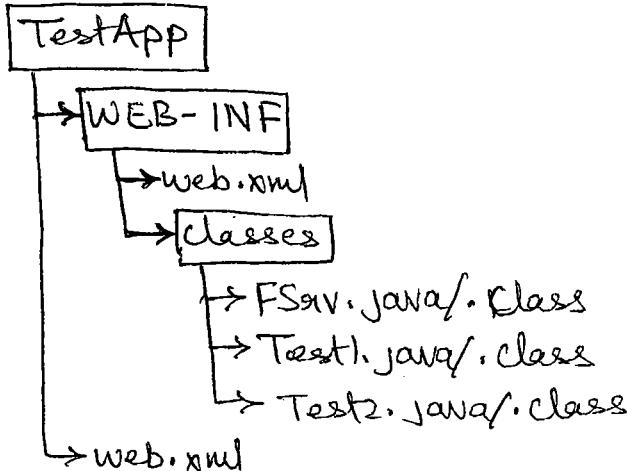
→ Servlet, JSP program becomes front controller only when it is configured in web.xml file either with extension match

or directory match web-pattern.

Example App<sup>M</sup> to develop user-defined frontController.



→ In the above diagram Test1, Test2 java classes are getting request from browser window through Front Controller servlet



// Test1.java

```
public class Test1
{
    public String bmi()
    {
        return "Test1 : Good morning from bmi()";
    }
}
```

### //Test2.java

```
public class Test2
{
    public String bm2()
    {
        return "Test2 : Good evening from bm2()";
    }
}
```

### //Fserv.java (FrontController Servlet)

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class Fserv extends HttpServlet
{
    public void doGet(-,-) throws ServletException, IOException
    {
        // General setting
        PrintWriter pw = res.getWriter();
        res.setContentType("text/html");

        // Delegate the request to java class
        String spath = req.getServletPath(); // gives the last
                                            // word of request url
                                            // like /s1.do, /s2.do etc...
        if(spath.equals("/s1.do"))
        {
            Test1 t1 = new Test1();
            pw.println("Result is" + t1.m1());
        }
        else
        {
            Test2 t2 = new Test2();
        }
    }
}
```

```

pw.println("Result is " + t2.bm2C());
}

-- // Close stream

pw.close();
}//doGet(-,-)

public void doPost(-,-) throws SE, IOException
{
    doGet(req,res);
}//doPost(-,-)

}//class

```

### web.xml :-

```

<web-app>
    <servlet>
        <servlet-name> S1 <servlet-name>
        <servlet-class> FServlet </servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name> S1 <servlet-name>
        <url-pattern> *.do </url-pattern>
    </servlet-mapping>
</web-app>

```

→ Deploy the above appln in Tomcat Server.

### Request url :-

Http://localhost:2013/TestApp/S1.do

Http://localhost:2013/TestApp/S2.do

→ Struts S/W is created by using multiple technology and by implementing multiple design patterns. that means every Struts appln contains implicit design patterns. ~~some built-in~~

⇒ Some Important Built-in Design-patterns of Struts Application development.

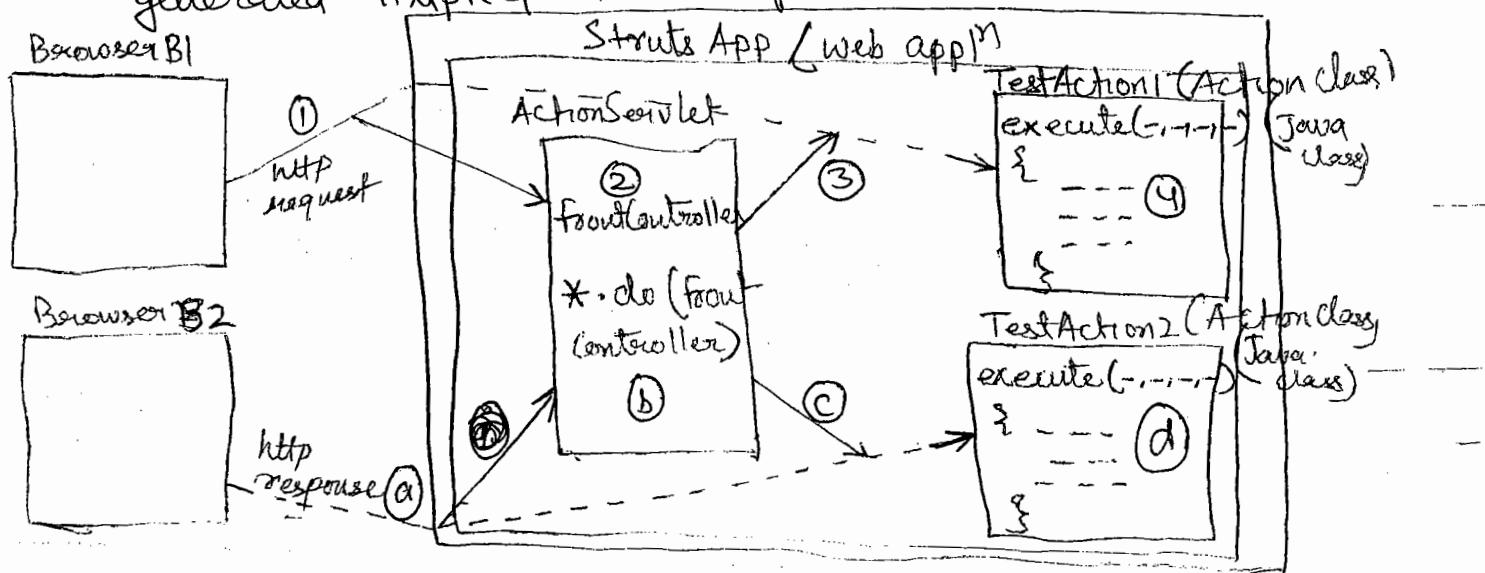
1. FrontController (ActionServlet)
2. AbstractController (RequestProcessor (the helper class to ActionServlet))
3. D.T.O class / V.O class (Form Bean)  
(inverse of controller)
4. IOC / Dependency Injection (ActionServlet while writing)
5. MVC2 (The whole struts App.)

→ In struts app<sup>m</sup> Action Servlet is front controller.

Q: Why ActionServlet is given as frontController?

A: All the request coming to struts app<sup>m</sup> want to go to different classes to process the request by executing business logic but these ActionClasses are Java classes and they cannot take client generated HttpServletRequest directly. To

To over come this problem they have given ActionServlet as FrontController which takes client generated HttpServletRequest and passes them to ActionClasses.



→ In struts app<sup>n</sup> we cannot change flow of execution from request arrival to response generation because the controller servlet in struts app<sup>n</sup> is pre-defined servlet class called ActionServlet but the resources which participate in that flow of execution like form FormBean, Action Class, Result pages and etc... are in programmatic control.

→ You should configure ActionServlet in web.xml of struts app<sup>n</sup> either with ~~extension~~ extension match or directory match url pattern to make it as frontController.

→ We generally configure ActionServlet in web.xml file as shown below.

```
<web-app>
  <servlet>
    <servlet-name> action </servlet-name> logical name
    <servlet-class> org.apache.struts.action.ActionServlet <servlet-class>
      <init-param> package Name
        <param-name> config </param-name> // Fixer init param having
          <param-value>/WEB-INF /struts->.config.xml </param-value> the value and location
          </init-param> of Struts
          </param-value> </param-param> configuration
          file.
    <load-on-startup>2 </load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name> action </servlet-name> logical name
    <url-pattern> *.do </url-pattern> extension match url-pattern to make ActionServlet
      </servlet-mapping> as frontController
  </web-app>
```

→ To gather non-technical inputs from user (Client-side) use request parameters (Form data), To gather technical

input from programmers (server side) use init, context parameters.

- The name and location of Struts configuration file required for ActionServlet should be passed to it as technical information so ActionServlet is expecting it from programmer as "config" init parameter's value as shown in the above ActionServlet configuration done in web.xml file

#### <load-on-startup>

- To minimize response time of 1st request and to equalize the response time other than 1st request and also to keep ActionServlet object ready before arrival of 1st request. We need to make servlet container creating the object of ActionServlet either during server startup or during the deployment of Struts appM by enabling <load-on-startup> on it.
- Enabling <load-on-startup> on ActionServlet is optional when form pages are designed by using traditional HTML tag. And Enabling <load-on-startup> on ActionServlet is mandatory when same form pages are designed by using the Struts supplied JSP tags.

Servlet/JSP Java Web appM (Classic Java Web appM) :-

	<u>&lt;load-on-startup&gt;</u>	<u>&lt;load-on-startup&gt;</u>	<u>&lt;load-on-startup&gt;</u>
Srv1	10 (IV)	10 (III)	10
Srv2	5 (II)	0 (I)	10
Srv3	0 (I)	6 (II)	10
Srv4	9 (III)	-1 (Ignore <load-s>)	10

} // Server decide the priority order.

- When multiple servlet program is enabled with <load-on-startup> then the servlet container decides the order of creating those servlet class objects based on their <load-on-startup> priority values.
- High value represents low priority and low value represents high priority

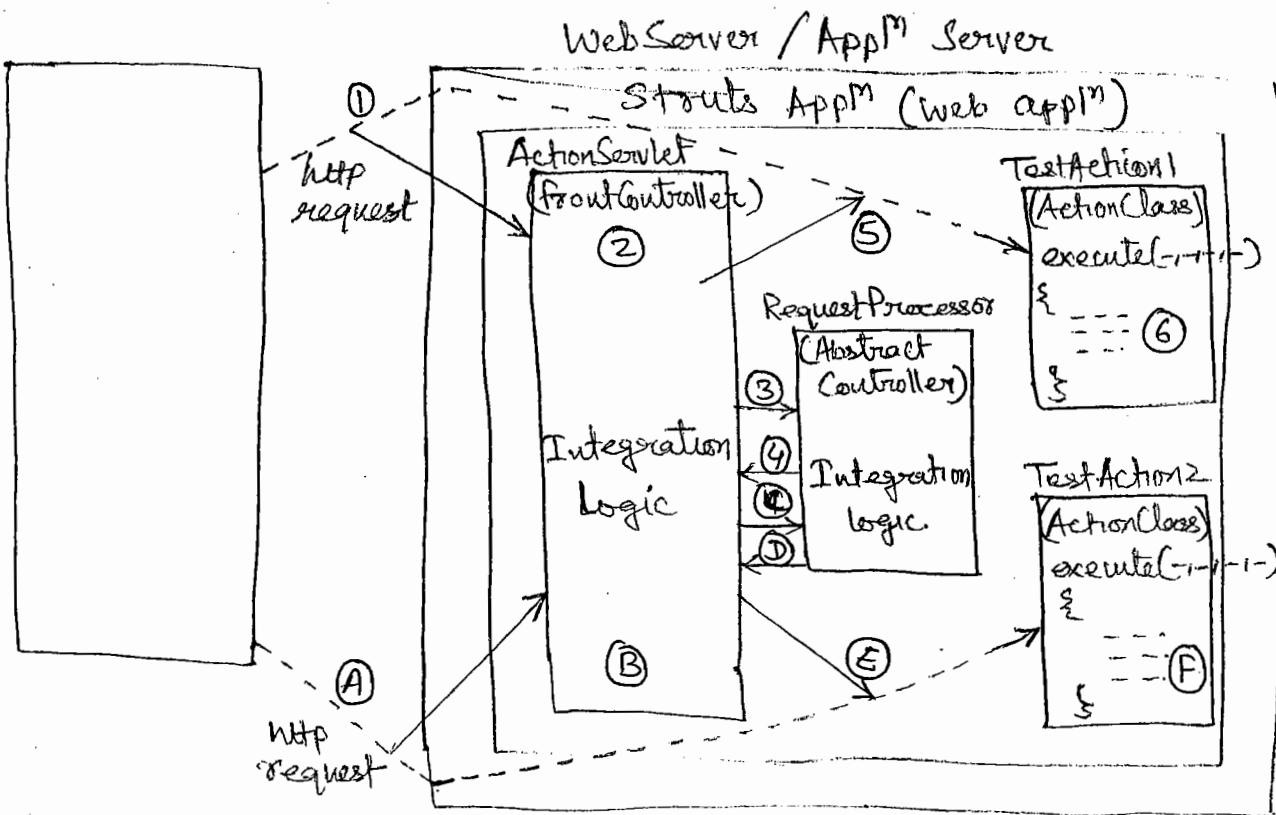
The negative value equated as <load-on-startup> priority values.

#### Note

- In Tomcat 5.x server "0" indicates least priority.  
~~classic java web app~~
- Since ActionServlet is only one servlet of Struts app<sup>M</sup> you can use any "tve" number or ~~also~~ "0" as <load-on-startup> priority value.

### ABSTRACT CONTROLLER :-

The helper java class for frontController which is capable of customizing the behaviour of frontController is called Abstract controller. In struts ~~pre~~ PredefinedRequestProcessor class is helper class for the frontController ActionServlet So to customize the behaviour of ActionServlet, either we give instruction to request processor class or we can place our class as AbstractController for ActionServlet instead of ~~front~~ RequestProcessor class.



→ In struts appln creating or locating FormBean class obj, creating or locating ActionClass obj, reading and verifying Struts configuration file and etc... operations will be done by RequestProcessor class on behalf of ActionServlet. So, this RequestProcessor class is called as Abstract controller.

(Data Transfer object) D.T.O class/V.O class :-

06.08.13

→ While transferring huge amount of data/values from source layer to destination layer instead of transferring them one by one for multiple time it is recommended to combine them into single object and send that object from source layer to destination layer. This reduces round trips between source layer and destination layer. Here the class of that single object is called VO class or D.T.O class.

- In struts appl<sup>n</sup> the FormBean class is called as VO class or D.T.O class because the huge amount of form data belonging to the form page of view layer goes to the action class of model/controller layer as single FormBean class object.
- The parameter of execute(--,--) method exposes FormBean class obj to Action class.

### Inversion of Control (IOC) / Dependency Injection :-

- Dependency lookup :
- Resource of the appl<sup>n</sup> spends time to search and gather its dependent value.
- Here resource pools the dependent values from other resources.
- If student gets cource material only after demanding for it is called dependency lookup.
- The way dataSource obj is gathered from JNDI registry is called dependency lookup.

### Dependency Injection (IOC)

- Underlying server or container or framework or runtime environment dynamically assigns values (dependent values) to resource.
- Here underlying server or framework or etc... pushes the value to resource
- Ex-1 :- If material is assign to student the moment he register for cource is called Dependency ~~not~~ Injection.

- The way JVM execute constructor to initialize data to the obj when object is created is called dependency injection.
- The way ActionServlet writes received form data to FormBean class object dynamically by creating or locating that obj is called dependency injection.

### MVC2

- Developing web appl<sup>n</sup> having multiple layers is called developing appl<sup>n</sup> based on MVC2 architecture. In Struts appl<sup>n</sup> → HTML/Jsp program acts as View layer resource.
- ActionServlet acts as controller Servlet ~~Action~~ (frontcontroller)
- struts config file, FormBean, RequestProcessor classes will act as controller <sup>layer</sup> resources.
- Action class acts as model layer resource, if you place business logic in it. Action class acts as controller layer resource if you place logic to interact with other model layer resources.

#### UNDERSTANDING struts config file:

- any <filename>.xml can be taken as struts file name if no file name is specified as "config" init param value of web.xml file during conf. of ActionServlet, the struts-configuration file of WEB-INF folder will be taken as default struts cfg file.
- ActionServlet/ Request processor uses SAX XML parser to read and verify struts cfg file.

We provide the following cfgs in Struts config file.

- ↳ FormBean config
- ↳ action class config
- ↳ action forwarding config
- ↳ action mapping config
- ↳ properties files
- ↳ plugins ... etc...

- The process of linking FormBean with Action class Actionforward based result pages with Action class Defining Action class ~~initialization~~ configuration is called Action mapping
- The java class/JavaBean class that extends from org.apache.struts.action.Actionform class is called FormBean class. Every FormBean class must be configured in Struts configuration file having logical name using ~~FormBean~~ tag <form-Beans> tag.

Ex:-      // RegisterForm.java

```
public class RegisterForm extends ActionForm  
{  
    private String username; // FormBean property to  
    private String password; // hold to form data)  
    // write setXXX() and getXXX() methods.  
    -- -- --  
}
```

- FormBean class must be taken as public class to make it visible to ActionServlet

Form Components Name of form page and ~~formBean~~ formBean properties name need not to match but Form Components name and "xxx" words of setXXX(), setXXX() method must match.

In struts cfg file :-

```
<form-beans>
  <form-bean logical-name="tf" type="Registerform"/>
</form-beans>
```

formBean logical name  
formBean class name

### Note

- Every FormBean is identified by its logical name. This logical name becomes obj name when ActionServlet creates the form bean class obj.
- The java class that extends from org.apache.struts.action.Action class is called Struts Action class. Every struts class must be cfg in struts cfg file having "action path" by using `<action>`

Example :- RegisterAction.java

```
public class RegisterAction extends Action
{
    public ActionForward execute(..., ...) throws Exception
    {
        ...
    }
}
```

In struts cfg file :-

```
<struts-config>
  <form-beans> *1
    <form-bean name="rf" type="Registerform"/> *2
  </form-beans>
  <action-mapping> *3
    <action path="/reg" type="RegisterAction" name="rf"> *4
      <forward name="result" path="/Success.jsp"/> *5
    </action>
  </action-mapping>
</struts-config>
```

Annotations:

- \*1 → "rf" is form bean logical name.
- \*2 → RegisterForm is FormBean class name.
- \*3 → "/reg" is action path of Action class (Every Action is identified with its action path).
- \*4 → "RegisterAction" is Action class name.
- \*5 → "rf" is formBean logical name (must match with \*1). It is required to link formBean with Action class.
- \*6 → action for "result" is action forward cfg logical name.
- \*7 → "success.jsp" is the result page of Action class.

Action Class Configuration

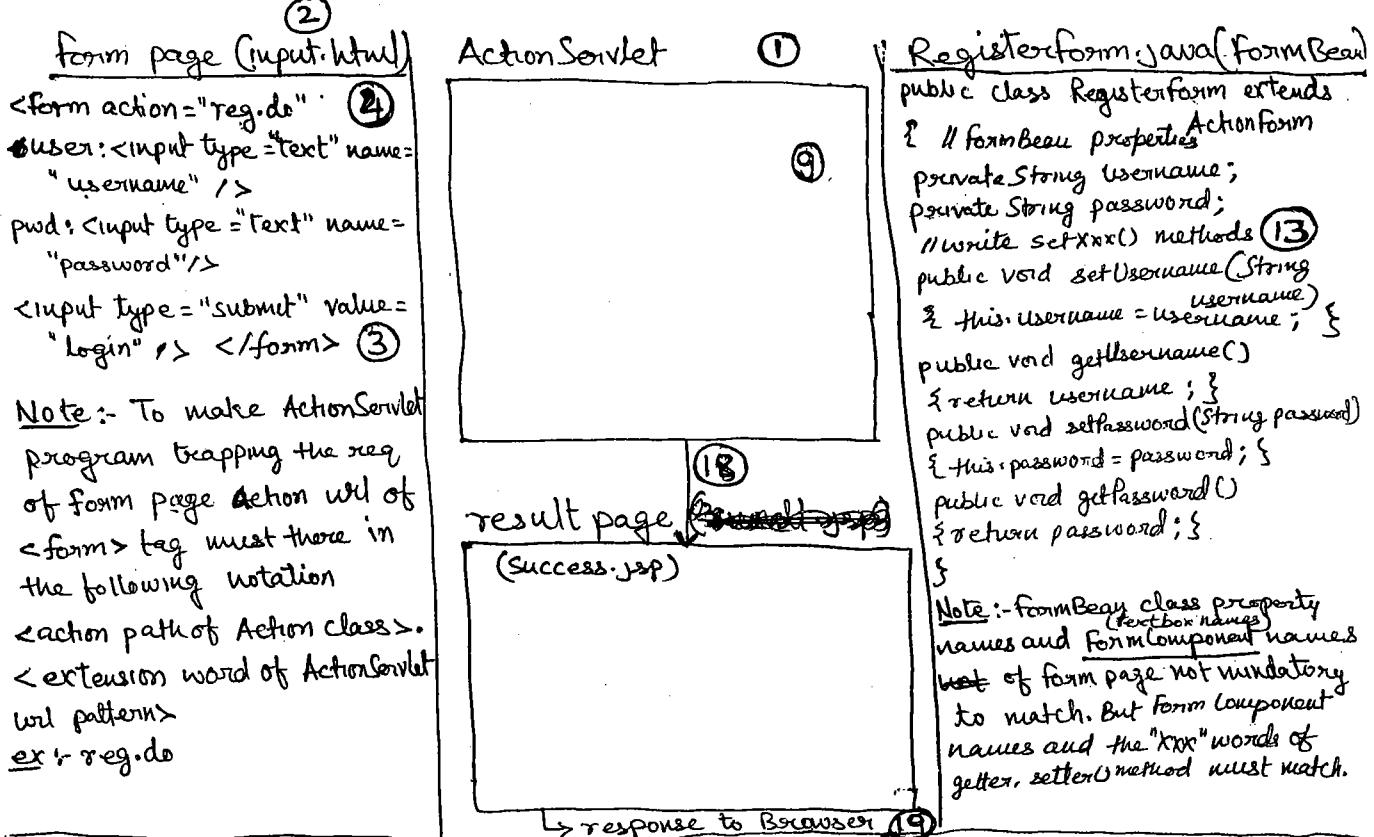
→ Struts Action class must be taken as public class

in order to make it visible to Action Servlet.



08.08.13

# UNDERSTANDING THE CODE BASED FLOW OF EXECUTION OF STRUTS APPLICATION



Web.xml

```
<web-app>
  <servlet>
    <s-m> action <s-n> ⑦
    <s-c> org.eaction.ActionServlet
    <s-c> o.a.s.action.ActionServlet <s-c>
  </servlet>
  <servlet-mapping>
    <s-n> action <s-n> ⑥
    <url-pattern> *.do <url-pattern>
  </servlet-mapping>
</web-app>
```

Note :- If form page is designed by using traditional HTML tags then enabling load on Start-up on ActionServlet is optional.

→ If Struts Conf. filename is struts-conf.xml belonging to WEB-INF folder then its conf. in web.xml during the ActionServlet conf. is optional.

```
<struts-config>
<form-beans>
<form-bean name="rf" type="RegisterForm"/>
<action-mappings>
<action path="treg" type="RegisterAction">
  name="rf"><!-- 11 -->
<forward name="result" path="/success.jsp"/>
</action-mapping> <!-- 17 -->
</struts-config>
```

```
RegisterAction.java (Action Class)
public class RegisterAction extends Action
{
    public ActionForward execute(-----)
        throws Exception
    {
        ----- (15) (B: Logic)
        returns ActionForward obj (with
        logical name result); (16)
    }
}
```

With respect to previous code :-

- ① → Programmer deploys web app<sup>M</sup> in web Server (or) App<sup>M</sup> Server  
(Do observe that <load-on-startup> is not enable on ActionServlet).
- ② → End user lunches the form page input.html on browser window
- ③ ④ → End user ~~sends~~ submit the form page in this process  
based on ~~Action~~ <sup>(reg.do)</sup> action url of form page request will be  
generated to struts app<sup>M</sup>.
- ⑤ ⑥ ⑦ ⑧ → Based on web.xml configurations ActionServlet gets the  
request.
- ⑨ → Since it is 1st request to ActionServlet and <load-on-startup>  
is not enabled on ActionServlet the ~~Action~~ Servlet container  
creates ActionServlet object. In this process ActionServlet  
reads and verifies the struts-config. file entries and also  
traps form page generated request.
- ⑩ → ActionServlet uses the "reg" word of form page ~~&~~ request url  
(reg.do) and locates RegisterAction class configuration in  
Struts-config. file based on it ~~Action~~ path="/reg".
- ⑪ → ActionServlet uses name="rf" ~~attribute~~ of <action>  
tag to get Registerform as the formBean class linked  
with RegisterAction class.
- ⑫ → ActionServlet creates or locates RegisterForm class object  
(FormBean class obj)
- ⑬ → ActionServlet calls ~~set~~ setXXX() method on FormBean class  
obj to write the received form data of form page to  
FormBean class properties.
- ⑭ → ActionServlet creates or locates RegisterAction class object  
(Action class).

- (15) ActionServlet calls execute(----) method on RegisterAction class object
- (16) ActionServlet call execute(----) method process the request by using business logic and returns ActionForward object with logical name "result" to ActionServlet
- (17) ActionServlet uses the received logical name "result" and gets "success.jsp" as result page of RegisterAction class through ActionForward configuration.
- (18) ActionServlet passes the control to result page "success.jsp".
- (19) The result page "success.jsp" formats the results and sends formatted results to Browser window as response.

→ To make form page, hyperlink generated request going to certain Action class through ActionServlet the action url of form page and href url of hyperlink must be in the following notation:-

<action path of action class>. <extension word of ActionServlet's url pattern>

Ex:- If action path "/reg" and ActionServlet url pattern is \*.do then

g,1) form page      action path  
extension  
<form action = "reg.do">  
</form>

In Hyperlink

<a href = "reg.do" > go </a>



09.08.13

- web.xml is one per web app<sup>m</sup>
- struts.config. file is one per each struts module (we can take multiple modules in a struts application). By default every struts app<sup>m</sup> contains one default module.
- In one struts app<sup>m</sup> we can take multiple form pages, FormBean class, Action classes and result pages.
- **UNDERSTANDING JSP TAG LIBRARIES:**
- Jsp tag library is a library that contains set of readily available ~~set~~ jsp tag.
- Every jsp tag library contains one tld file having various details about like tag names, tag handler class names, attribute details and etc...
- The java class that extends from javax.servlet.jsp.tagext.TagSupport (c) having the functionality of jsp tag is called tag handler class.

- There are three types of jsp tag libraries:-
  - I) Pre-defined/Built-in jsp tag libraries (Given by Jsp Tech.)
  - II) Custom/user-defined jsp tag libraries (Developed by programmers)
  - III) Third-party jsp tag libraries (custom-tag libraries given by third-party vendors)  
third-party like Apache, Oracle corp and etc...

- Procedure to develop custom jsp tag libraries:-

Step-1

- Design custom jsp tag library
  - write jsp Tag libraries
    - <ABC>
    - <XYZ>

### Step-2

Develop JSP tag handler classes for the above JSP tags.

#### WEB-INF\classes\ABCTag.java

```
public class ABCTag extends TagSupport  
{  
    public int doStartTag() // executes for open <ABC>  
    {  
        ---.  
        ---.  
        ---.  
    }  
    public int doEndTag() // executes for </ABC> tag.  
    {  
        ---.  
        ---.  
    }
```

#### WEB-INF\classes\XYZTag.java

```
public class XYZTag extends TagSupport  
{  
    public int doStartTag() // executes for open <xyz>  
    {  
        --- (g)  
        ---.  
    }  
    public int doEndTag() // executes for </xyz> tag  
    {  
        --- (g)  
        ---.  
    }
```

### Step-3

Develop tld file for Durga JSP tag library.

## WEB-INF\dwrga.tld (it is xml file)

<ABC> → ABCTag.class  
<XYZ> → XYZTag.class  
    (F)

→ Step-1 to Step-3 talks about the development of JSP tag libraries.

## Step-4

Configure JSP tag library in web.xml file of Web application

### web.xml

```
<web-app>
  <taglib> (D)
    <taglib-uri> demo </taglib-uri>
    <taglib-location>/WEB-INF/dwrga.tld </taglib-location>
  <taglib> (E)
    <web-app>

  <taglib>
```

→ Every JSP tag library configured in web.xml file will be identified with its taglib uri

## Step-5

Use the JSP tags of JSP tag library in the JSP programs of Web application.

Test.jsp      (C) taglib-uri (refer web.xml)  
<%@taglib uri="demo" prefix="st"%>  
                        ↑ programmer choice word  
<st:ABC/>  
<st:XYZ/> (A)  
    (B)

→ If tag name of multiple JSP tag libraries are having same name then they can be differentiated based on

there prefixes. These prefixes are user-defined and programmer choice.

### Note :-

Step-4 and Step-5 talk about utilization of JSP tag libraries.

(H) → <stl:xyz> output goes to browser window as response

### Note

In the above code (A) to (H) indicates the flow of EXP.-  
ecution.

→ The third-party JSP taglibraries supplied TLD files,  
tag handler classes in the form of jar files.

→ Struts supplies 5 no. of JSP tag library. as third  
party JSP tag libraries, the details are :-

Jsp tag library name.	TLD file name.	Recommended prefix
I) html	struts-html.tld	html
II) bean	struts-bean.tld	bean
III) nested	struts-nested.tld	nested
IV) logic	struts-logic.tld	logic
V) tiles	struts-tiles.tld	tiles

→ html, bean, nested, logic, JSP tag libraries related  
tag handler classes, TLD file are available in  
<Struts-home>\lib\struts-taglib-1.3.8.jar file

→ tiles JSP tag libraries related tag handler classes,  
TLD file is available in <Struts-home>\lib\struts-  
~~taglib-1.3.8~~-tiles-1.3.8.jar files.

### Note

- Struts supplied "html" tag library contains Jsp tags that are alternative. tho of the traditional html tag.
- Struts supplies "bean" tag library gives tags to work with FormBeans and to work with different types of attribute.
- Struts supplied "logic, nested" tag library contain tags to perform conditional and iterationel operations.
- Struts supplied tiles tag library gives Jsp tag to work with tiles plugin related presentations.
- Struts supplied jsp tags can be used only in struts based web applications.
- Procedure to use struts supplied html tag libraries Jsp tags in our struts Appl.

#### Step-1

- Gather info about various jsp tags of html tag library.

⇒ <struts-home>\docs\struts-taglib\index.html

#### Step-2

- Gather struts-html.tld file and place in WEB-INF folder of struts appl<sup>n</sup>.

#### Step-3

- place struts-taglib-1.3.8.jar file and its dependent jar files in WEB-INF\lib folder of struts appl<sup>n</sup>.

#### Step-4

- Configure this html tag library in web.xml file.

### Web.xml

```

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

```

### Step-5

→ Use the jsp tags html tag library in the jsp programs of struts appn.

### Test.jsp

```

<%@ taglib uri = "demo" prefix = "ht" %>
<ht:form action = "reg.do">
  username : <ht:text property = "uname" /> <br>
  password : <ht:password property = "pwd" /> <br>
</ht:form>

```

↑ prefix      ↑ tag

↓ form component  
names

10.08.13

What is the difference between traditional html tag and struts supplied html tags.

#### Traditional html tag

- Given by W3C (world wide web consortium)
- The form page default request methodology is get "GET"
- Does not support to implement MVC2 rules
- Does not generate extension word dynamically in form page action and

#### Struts supplied tag library html tag

- Given by apache foundation (struts).
- The form page default request methodology is "POST"
- supports to implement MVC2 rule.
- generates the extension word dynamically in form page action and

~~Form page~~

→ Form page from starts appl' using traditional Wind tags

```
<form action="reg.do"> //Default Request methodology is  
-----  
-----  
-----  
methodology  
</form>  
"GET"
```

⇒ Form page for struts appl'n using struts supplied Jsp tags

```
<html : form action = "reg.do" > // default request methodology  
|-----  
|----- optional  
|-----  
</html : form>
```

⇒ The two overloaded form of execute(-,-,-,-) method in org.apache.struts.action.Action class.

form! public ActionForward execute(ActionMapping mapping,  
ActionForm form,  
ServletRequest request,  
ServletResponse response) throws Exception

Form 2 :-

```
public ActionForward execute(ActionMapping mapping,
                             ActionForm form,
                             HttpServletRequest request,
                             HttpServletResponse response) throws
                             Exception.
```

→ The form2 is recommended to use...while developing our state Action action class we need override one of the above execute(----) methods.

## Action Mapping mapping

→ It is the object of org.apache.struts.action.ActionMapping class. This object holds the information that the ~~request~~ controller, requestProcessor class known about mapping of a particular request to a particular instance of Action class.

Using this object we can get current Action class details and other Action class details that as config. in struts config. file. Using this object we can make current Action class interacting with other Action class. and we can also create and return ActionForward object pointing to the result pages of Action class.

## ActionForm form :-

ActionForm a = new ~~ActionForm~~

→ It is ActionForm class reference variable pointing to the FormBean class object that is linked with Action class. This is very useful to read form data of form page in the execute(----) method of Struts Action class.

## Servlet Request request :-

→ Represents the current request given to Action class through ActionServlet this object comes through Action Servlet. Using this object we can gather additional details about client and client generated request like browser s/w name languages, mime type supported by browser and etc... (through requestHeaders)

This object is also useful to create request ~~Header~~<sup>Attributes</sup> and to send data from Action class to other resources.

This object is even useful to create other utility object like HttpSession object. Prefer reading form data of form page through FormBean class object because there is a possibility of correcting form data in FormBean class if necessary. Since this data correction is not possible don't prefer using request object to read form data.

### ServletResponse response :-

(rarely)

- It is less used object parameter of execute(-,-,-,-) method
- This object is useful to make Action class sending response to browser window directly, but this process (sending response to browser window) is against industry standard and violation of MVC2 rules.

When Action class wants to send response to browser window directly the "response" object is used as shown below

PrintWriter pw = ~~response~~.getWriter

```
response.setContentType("Text/html");
pw.println("<b><i> Directly from Action Class</i></b>");
```

→ Programmers are not responsible to create the above four objects, ActionServlet creates them and uses them as arguments while calling in execute(-,-,-,-) method call.

→ execute(-,-,-,-) of Action class should return ActionForward class object to ActionServlet having logical name.

ActionServlet uses this ActionForward object to locate and get result page of Action class through struts configuration file entries.

In execute(---,--) method we can call ~~mapping~~ mapping.findForward("<logical name>") to get and return ActionForward obj.

### Naming Conventions:-

FormPage → X.jsp

formBean → XForm

ActionClass → XAction

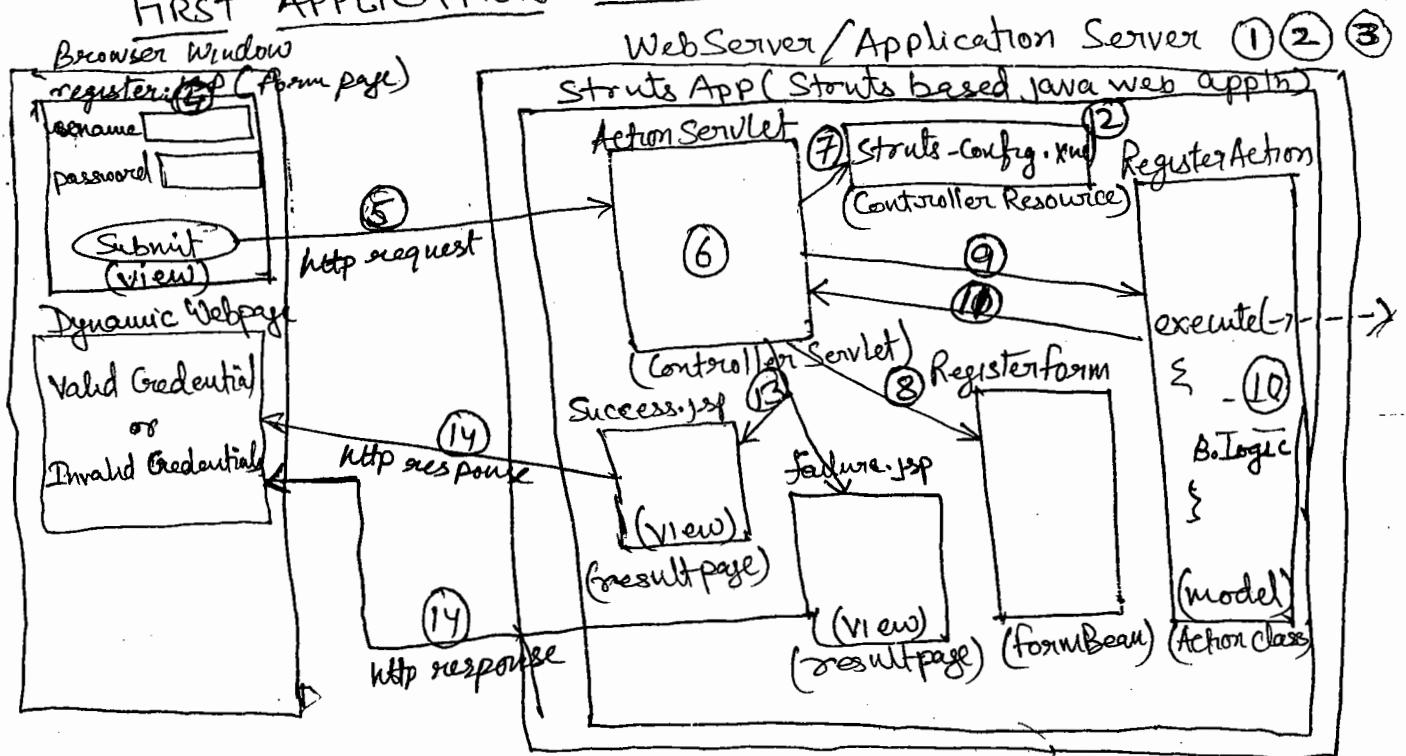
Ex:-

FormPage → register.jsp

formBean → Registerform

Action class → RegisterAction

### FIRST APPLICATION DEVELOPMENT



→ For the source code of above appl'n refer Appl'n-1 of page Nos - 8 and 9.

### ActionForm form

Here form is ActionForm class reference variable pointing to our FormBean class obj. So we cannot call the direct methods of ~~Action~~ ~~to~~ our FormBean class on this "form" reference variable.

To make that happening type cast this reference variable to our FormBean class.

Ex:

```
ActionForm form = new app.Registerform();
form.getUsername(); (Invalid)
form.getPassword(); (Invalid)
```

```
app.Registerform form = (app.Registerform) form;
form.getUsername(); (Valid)
form.getPassword(); (Valid)
```

→ Procedure to develop, deploy, Test 1st struts appl'n that is placed given in page No-8 and 9 of course Booklet.

#### Step-1

→ Extract <Struts-home>\apps\struts-blank-1.3.8.war file to a zip file by using winzip or winRAR tool.

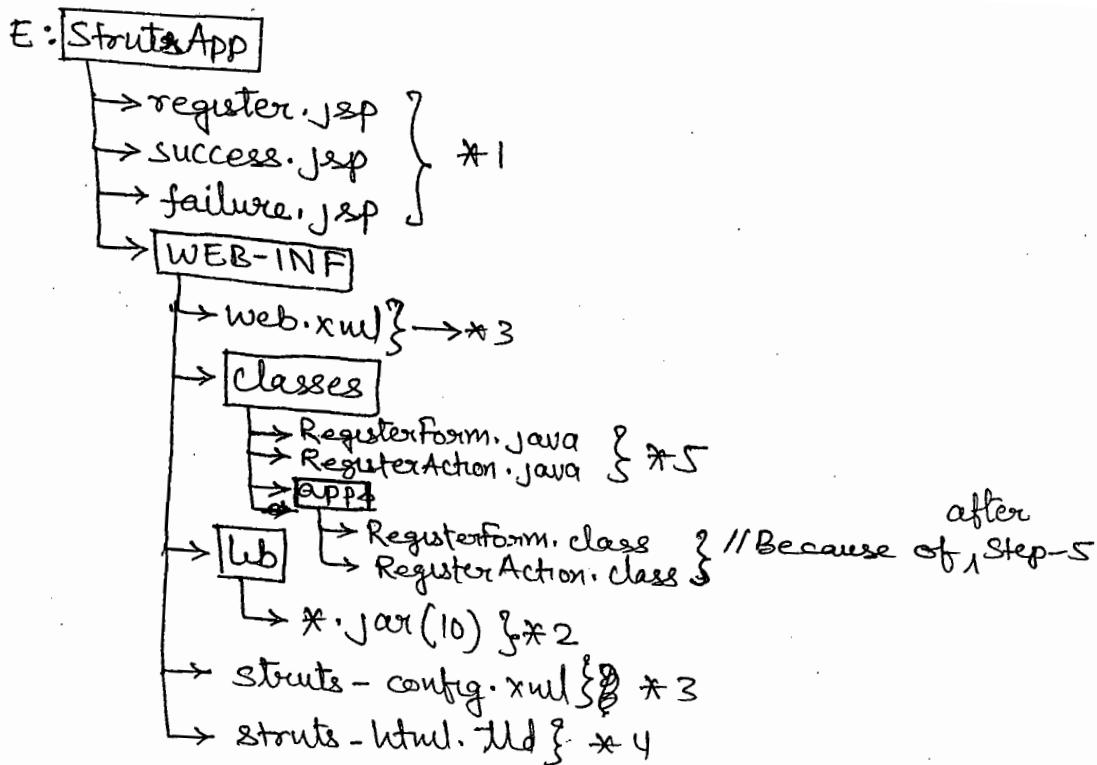
#### Step-2

Arrange struts-html.tld file ~~is~~ by ~~so~~ searching in struts.lx S/w.

#### Step-3

Create the deployment directory of structure of struts Appl'n,

~~parent~~ ~~parent~~ <sup>aggregating</sup>



\*1 → Type and save JSP prog

\*2 → Gather from ~~WEB-INF~~ folder of step1 extraction

\*3 → develop them by taking step1 extraction content as reference content.

\*4 → Gather through step-2 process

\*5 → Type and save java sources.

#### Step-4

Add <tomcat-home>\lib\ servlet-api.jar, <struts-home>\lib\ struts-core-1.3.8.jar files to classpath environment variable...

#### Step-5

Compile the java sources of app1.

>javac -d . \*.java <

#### Step-6

Start tomcat server and deploy the above web app1  
(StrutsApp) in tomcat server (webapp)

⇒ Copy e:\strutsApp folder to tomcat\_home\webapps folder.

### Step-7

Test the appln.

open browser window → typ this url  
<http://localhost:2013/StrutsApp/register.jsp> ↴

### Note

- Donot create formBean obj explicitly in Action class because it doesnot carry the form data of form page. Always use ActionServlet created and supplied formBean class object, because it holds form data of form page for this use the 2nd parameter of execute(-, -, -, -) method in Action class
- 13.08.13
- In the above application ActionServlet class will be recognized from struts-core-1.3.8.jar file that is placed in WEB-INF\lib folder.
- While working with Tomcat7 server the jsp tag ~~library~~ library in web.xml must be configured as shown below.

In web.xml :-

```
<jsp-config>
  <taglibs>
    <taglib>
      <taglib-uri>demo</taglib-uri>
      <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
    </taglib>
  </taglibs>
</jsp-config>
```

- If struts-configuration file name is struts-config.xml file of WEB-INF folder then there is no need of configuring that file name in ~~WEB~~ web.xml file as the ~~id~~ <init-parameter> of ActionServlet but the same configuration is mandatory if file name or file location is changed.
- Enabling <load-on-startup> on ActionServlet is mandatory when form page is designed by using struts supplied <jsp> tags but it is optional when the same form page is designed by using traditional <html> tags.
- When struts supplied <jsp> tag based form page is launched on the browser window ActionServlet creates FormBean class object, calls getter methods on that obj. to populate the data of FormBean properties as initial values to form components. For this ActionServlet object should be available before launching form page. Either during server start-up or during the deployment of web-application. by enabling <load-on-startup>.
- When traditional <html> tag based form page is launched no operation will takes place on FormBean class and form components don't show any initial values so, there is no need of keeping ActionServlet object before launching form page this makes enabling <load-on-startup> on ActionServlet as optional process.
- To make FormBean class and ActionClass visible and accessible to ActionServlet for initiation and initialization we must take them as "public" classes.

- If there are not constructors in java class (explicitly placed) then the java compiler generates one 0-param constructor as default constructor.
- ActionServlet uses 0-param constructors while creating FormBean, Action class object So we must place 0-param constructor directly or indirectly <sup>(\*)1</sup> <sup>(\*)2</sup> while working within FormBean, Action classes.
- \* 1 → explicitly placed 0-param constructor
- \* 2 → The compiler generated default 0-param constructor.

→ We can make ActionServlet keeping FormBean class objects in two scope

1. Session (default scope) (ActionServlet keeps FormBean obj as session attribute value)
2. request scope. (ActionServlet keeps FormBean obj as request attribute value).

#### Request Scope :-

ActionServlet creates FormBean class object for every request coming to form page and generated by form page. Here the form page component cannot show previous request related data.

#### Session Scope :-

ActionServlet creates FormBean class obj. on one per browser window basis. Here the form page components can show previous request related data.

#### Example code :-

```
<form-beans>
  <form-bean name="rf" type="app.RegisterForm"/>
</form-beans>
```

```
<action-mapping>
<action-path="/register" type="app.RegisterAction" name="rf" scope="session/request">
    </action>
</action-mappings>
```

Q → Why scope attribute is given in ~~<action>~~ tag to specify formBean scope.

Aus: There is a provision to make multiple Action classes using single FormBean. In this process to control FormBean class object scope with respect to each Action class the scope attribute is given in ~~<action>~~ tag.

Ex:-

```
<action-mappings>
<action-path="/reg1" type="obj1Action1" name="rf"
    scope="request">
    </action>
<action-path="/reg2" type="obj2Action2" name="rf"
    scope="session">
    </action>
</action-mappings>
```

(It is not action scope)

(It is the ~~action~~ scope of formBean class obj)

Note:-

There is no provision to specify the scope for Action class obj. that means Action class obj has global scope within web app. (Creates only one obj for

Action class and uses it across the multiple request.

### UNDERSTANDING FLOW OF EXECUTION OF STURTS

#### 1st APPLICATION OF PAGE NO- 8 and 9 OF BOOKLET

Form (formBean scope = "session") FORMBEANS

- (A) Programmer deploys "StrutsApp" web application. in Webserver Tomcat.
- (B) Because of <Load-on-startup> the servlet container creates the object of ActionServlet either during Server startup or during the development of StrutsApp.
- (C) ActionServlet reads and verifies the entries of struts-config.xml file. And also store them in module Config obj.

#### Note

Servlet container picks up ActionServlet class from struts-  
~~core~~-1.3.8.jar file that is placed in WEB-INF\lib  
\struts-core-1.3.8.jar file

- (D) End user launches form page on to the browser window by using following request url.

<http://localhost:2013/StrutsApp/register.jsp>

- (E) Based on Action url of form page (~~register~~)  
<sup>refer line no-6</sup> (action = "register") the ActionServlet locates RegisterAction class config in struts config file whose action path is <sup>refer line no-45</sup> "/register" → based on <sup>refer line no-47</sup> name = "of" attribute of <action> tag the <sup>refer line no-41</sup> form bean (RegisterForm) that is linked with Action class (RegisterAction) will be identified.

(5)

refer line no = 41  
(Registerform)

- F ActionServlet creates FormBean class obj and keeps in "Session" scope, ActionServlet calls getXXX() methods on FormBean class object to read data from formbean form bean properties and to write that data to the form components (textboxes) of form page as initial values.

- G EndUser fills up the form page and submit the req.  
refer line no - 9
- H Based on the action url kept in <action> tag, the following request url will be generated to send request to Struts App

http://localhost:2013/StrutsApp/register.do (refer line no 6).

- I Since the ActionServlet url pattern is \*.do, the Action Servlet traps and takes the request. (line no = 13 to 26)

- J ActionServlet uses the "register" word of register register.do to locate RegisterAction class in struts-config.xml entries whose action path is "/register".

- K ActionServlet uses name = "rf" attribute of <action> tag and locates Registerform class obj from "session" scope (refer line no - 47 and 41)

- L ActionServlet calls setXXX(-) methods on the located FormBean class object and writes the received form data of form page to form bean properties.

- M ActionServlet creates the Action class object (RegisterAct) (RegisterAction). (refer line no = 46)

- (N) ActionServlet calls execute(-,-,-,-) on the above RegisterAction class object. (refer-88 to 101)
- (O) execute(-,-,-,-) reads form data from the FormBean class object. (refer line no-92 to 95) and (72 to 79)
- (P) execute(-,-,-,-) returns Action forward ~~at~~ object having logical name "success" or "failure" (refer line no-98 or 100)
- (Q) ActionServlet uses the received ActionForward object logical name and locates result page of RegisterAction class based on action forwards config done in Struts config. file. (refer line no=48 or 49)
- (R) ActionServlet uses ~~sd.~~ forward(-,-) internally and forwards the control to either Success.jsp / failure.jsp. (refer line no - 103 to 112 (or) 113 to 122)  
Success.jsp              failure.jsp
- (S) The result page "success.jsp" / "failure.jsp" formats the result and sends the formatted result to browser window as final response.

- There are two approaches to work with Jsp Taglibrary..
- I) By using user-defined taglib wiz  
(As discussed in first StrutsApp to work with the struts supplied "html" Jsp tag library)
  - II) By using fixed jsp taglib wiz. (recommended)  
(We can gather these fixed taglib wiz from tld files of Jsp tag libraries)

→ The fixed taglib was struts supplied 5 jsp taglibraries.

~~html tag lib~~

### Tag libraries

① html

### Fixed taglib wri

<http://struts.apache.org/tags-html>

② bean

<http://struts.apache.org/tags-bean>

③ logic

<http://struts.apache.org/tags-logic>

④ nested

<http://struts.apache.org/tags-nested>.

⑤ tiles

<http://struts.apache.org/tags-tiles>.

→ We can gather these wri from <wri> tag of the respective tld files.

⇒ Procedure to make 1st Struts app<sup>m</sup> using the struts supplied html tag library based on approach 2.

#### Step-1

Keep 1st struts app<sup>m</sup> ready.

#### Step-2

Keep the html tag library related main and dependent jar files. (Struts-~~taglib~~-1.3.8.jar and its dependent jar files in WEB-INF\lib folder)

#### Step-3

Delete struts-html.tld file from WEB-INF folder.

#### Step-4

Delete jsp tag library configuration from Web.xml

file. (open tag to close tag lib)

<taglib> ----- </taglib>

#### Step-5

Gather fixed taglib uri of html tag library from <uri> tag of struts-html.tld file.

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

#### Step-6

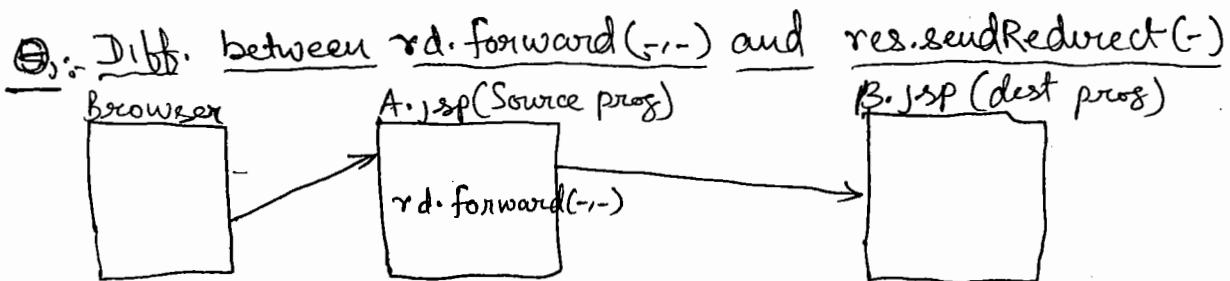
Gives the above gathered fixed taglib uri in the jsp program of struts application (register.jsp)

#### register.jsp

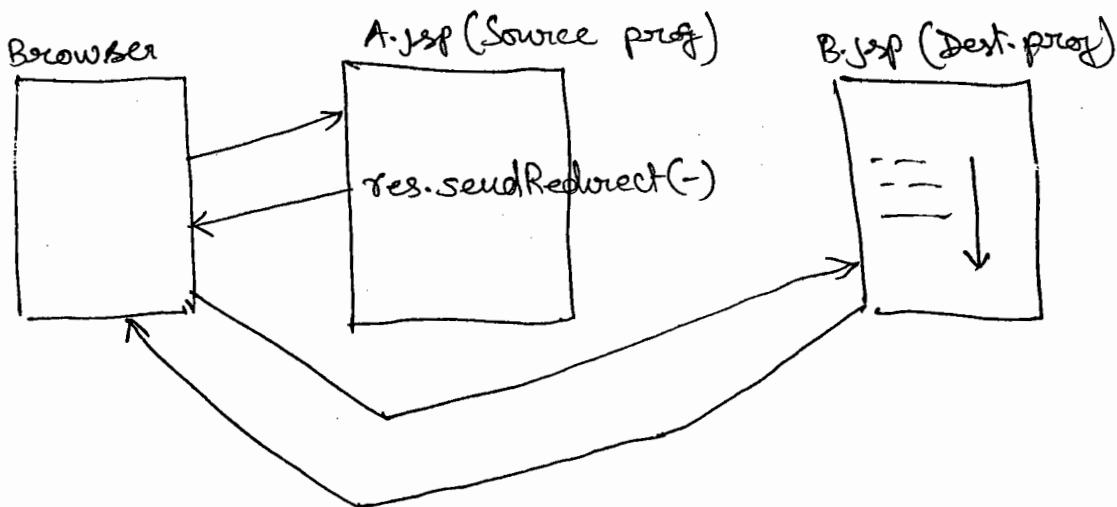
```
<%@ taglib uri="http://struts.apache.org/tags-html"
   prefix="html" %>
<%@system %> <html : form action="register">
  -- - -
  - - -
<html : submit value="register"/>
</html : form>
```

#### Note:-

Always prefer using approach-2 to use jsp tag libraries in our web application.



Here source prog. directly communicates with dest prog, and both source and dest prog uses same request and response objects



Here source prog interacts with dest. prog by having one network round trip ~~back~~ with browser window. Here source and dest. prog do not use same request and response object.

15.08.13

In struts-config file:

```

<action path="/register" type="app.RegisterAction" name="of">
<forward name="success" path="/success.jsp" redirect="true"/>
<forward name="failure" path="/failure.jsp" redirect="false"/>
</action>

```

redirect = "true"

→ ActionServlet uses `response.sendRedirect(-)` method to pass the control to result page. i.e. ActionServlet and Result page will not use same request and response objects. ActionServlet and Action class always use same request and response objects. i.e. In the above situation Action class and its Result page <sup>will not use</sup> same request, response objects when `redirect = "true"`.

## redirect = "false"

ActionServlet uses rd.forward(-,-) to pass the control to Result page so ActionServlet and Result page, Action class and Result page use same request and response objects. The default value of "redirect" attribute ~~is~~ is ~~true~~ false.

Q: How to pass data from Action class to its Result pages?

Ans: 1) Using "request" attributes

→ When Action class and its result pages use same request, response objs.

2) Using "session" Attributes

→ When Action class and its result pages get requests from same browser window (client)

3) Using "Application/ServletContext" attributes.

→ When Action class and its result pages are not getting request from same browser window and not using same request and response objects.

## Example

### Step-1

→ Keep 1st appl" ready.

### Step-2

→ Configure Result pages for Action class as shown below.

### In struts-config.xml

```
<action path="/register" type="app.RegisterAction" name="rf">
<forward name="success" path="/success.jsp" redirect="true"/>
<forward name="failure" path="/failure.jsp" redirect="false"/>
</action>
```

### Step-3

→ Creates attributes in ~~RegisterAction~~ class execute(-,-,-,-) as shown below.

```
public ActionForward execute(-,-,-,-)
{
    System.out.println("In RegisterAction : execute()");
    RegisterForm rf = (RegisterForm) form;
    String user = rf.getUsername();
    String pass = rf.getPassword();
    //request attributes
    request.setAttribute("attr1", "val1");
    //Session attributes
    HttpSession ses = req.getSession();
    ses.setAttribute("attr2", "val2");
    //ServletContext/ Application Attribute
    ServletContext sc = ses.getServletContext();
    sc.setAttribute("attr3", 100);
    System.out.println("In B. logic of RegisterAction");
    if (user.equals("durga") && pass.equals("soft"))
        return mapping.findForward("success");
    else
        return mapping.findForward("failure");
```

### Step-4

→ Read and display the above attribute value in Result pages (Success.jsp / failure.jsp)

#### // Success.jsp / failure.jsp

```
<body>
<center>
<%= sop(" From success.jsp"); %>
<font size=5 color=green> login successful </font>
</center>
<br>request attribute attr1 value = <%= request.getAttribute("attr1")%>
<br>session attribute attr2 value = <%= session.getAttribute("attr2") %>
<br>application attribute attr3 value = <%= application.getAttribute("attr3")%>
</body>
```

### Conclusion

→ Industries does not keep redirect = true while configuring Result pages for Action class. So real world programmer generally use request attributes a lot to send data from Action class to Result pages.

→ Writing java code in jsp as shown above is not industries standard. you should always develop jsp program as java codeless jsp programs.

→ In struts appl" we can use struts supplied jsp tags to make jsp programs as java codeless jsp programs.

→ <bean-write> tag can be used to read ~~diff~~ attribute value of different scope

<logic: notEmpty > can be used to check whether attribute is available or not in certain scope.

→ <bean-write> tag, <logic-not empty> tags can be used with or without scope attribute. When they are used with scope attribute then they will deal only specified scope attributes. When they are used without scope attributes then they will search for attributes in multiple scopes by using pageContext.findAttribute() method.

→ While displaying numeric value we should use <bean-write> tag with format="" attribute.

→ re-writing the above given ~~the~~ java code of success.jsp / failure.jsp by using <bean-write>, <logic-notempty> tags.

### In success.jsp / failure.jsp

```
<%@ taglib uri="http://struts.apache.org/tags-bean"
   prefix="bean"%>
<%@ taglib uri="http://Struts.apache.org/tags-logic"
   prefix="logic"%>
-----  

<logic:notEmpty name="attr1" scope="request">
<br>request attribute attr1 value:<bean:write name="attr1"
   scope="request">
</logic:notEmpty>
<logic:notEmpty name="attr2" scope="session">
<br>request session attribute attr2 value:<bean:write name=
```

```

"attr2" scope="session" />
</logic:NotEmpty>
<logic:NotEmpty name="attr3" scope="application">
<br><application attribute attr3 value=<bean-write name=
"attr3" scope="application" format="" />
</logic:NotEmpty>
</body>
</html>

```

It is like "if" condition.  
 It is like expression tag calling tag get attr.  
 required for only while reading numeric data.

### Note

Scope attribute is optional in above tags.

→ We can also use <bean-write> tag to display FormBean property values. for this we must use "property" attribute of <bean-write> tag.

Example      In sucess.jsp / failure.jsp      FormBean logical name  
 <br> <br>  
 The given form data is <bean-write name="rf" property = "username" /> &  
 <bean-write name="rf" property = "password" /> .  
 FormBean property name.

→ Developing the jsp program as java code less jsp programs is nothing but implementing "view helper design pattern".

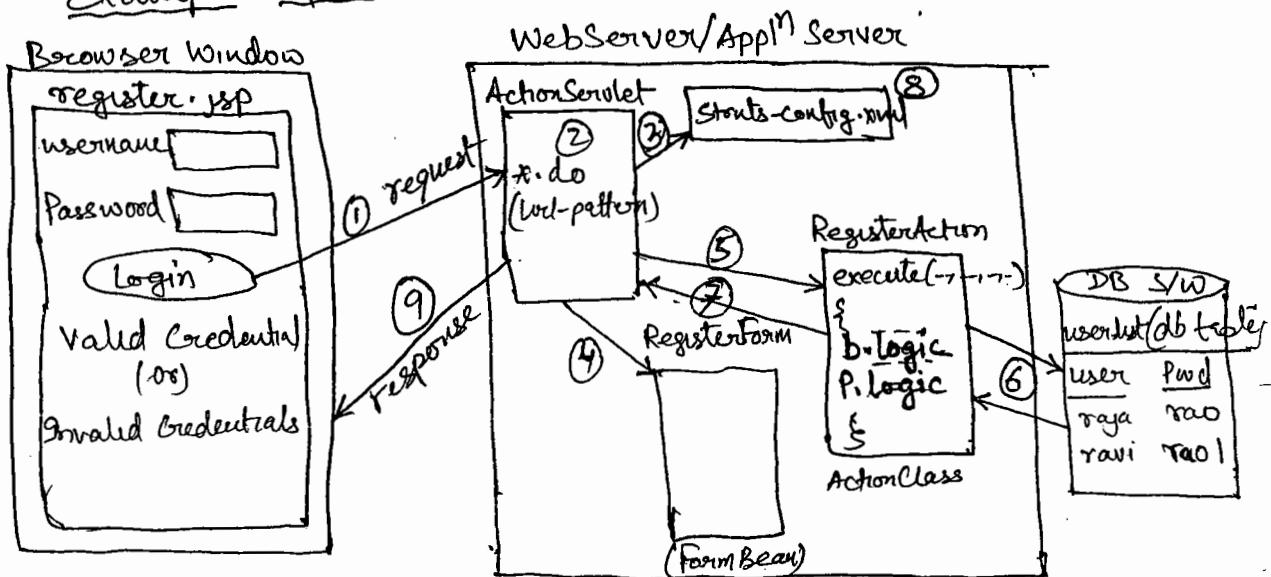
16.08.13

- While creating different scopes of attribute in the execute(-,-,-,-) method of Action class there is no need of worrying about scope of ~~Form~~ formBean.
- To display Action class result in Form page we must configure the Form page also as the result page of Action class.
- For struts appl<sup>n</sup> to db s/w communication place persistant logic directly in Action class. When Action class is taken as model layer resource.
  - ↳ like jdbc code

### NetBeans

- type: IDE s/w for java env...
- Version: 6.7.1 (Compatable with Jdk1.5+)
- Vendor: Sun Ms (Oracle corp)
- Open Source IDE
- gives Glassfish 2.0 as Built-in server
- for help : www.netbeans.org
- for docs: www.netbeans.org

### Example Application : StrutsApp2



→ The application is using form page also as the result page.

DB table of the application

```
sql> Create table userlist(uname varchar2(15), pwd  
varchr2(15));
```

```
sql> insert into userlist values('raja','rani');
```

```
sql> insert into userlist values('king', 'kingdom');
```

sql > select count(\*) from userlist where uname = 'raja'  
and pwd = 'rao';  
→ wrong password gives Count(\*)  
0

O mean invalid credentials.

```
Sql> select count(*) from userlist where uname='raja'  
      and pwd='raju';
```

gives  $\frac{\text{count}(*)}{1}$  because both Username & password are correct

I means Valid credentials.

→ Procedure to develop the above Struts app<sup>n</sup> by using NetBeans IDE.

### Step-1

Create web project in NetBeans IDE having name  
StrutsApp2

file menu → new project → java web → web application →  
next → ~~select~~ StrutsApp2 → next → server = Glassfish 2.1 →  
next → Select struts 1.3.8 → finish app<sup>1)</sup> resource = app<sup>2)</sup> resource

→ select add struts tiles → finish → delete index.jsp  
, WelcomeStruts.jsp

### Step-2

Add FormBean class to the project.

Right click on source packages Folder → new → other → Struts → Struts Actionform Bean → next → classname = RegisterForm → package = App → finish → add FormBean properties (username, password) and generate getter and setter methods on them.

refer RegisterForm.java of first application.

To generate getter and setter method in IDE select properties → Right click → insert code → getter and setter → -----

### Step-3

Add formpage to the web pages of the project

#### Register.jsp

Same as 1st appln.

<form> ----- </form>

### Step-4

Add Action class to project

Right click on source packages Folder → new → other → Struts → Struts Action → next → class name = RegisterAction → package = app → action path = /register → next → Action form Bean name = RegisterForm , scope = session → finish

### Step-5

Configure the form page also as the result page of Action class.

#### In struts-config.xml

```
<action name="Registerform" path="/register" scope="session" type="app.RegisterAction">
  <forward name="result" path="/register.jsp" />
</action>
```

↳ Formpage as Resultpage

### Step-6

Add ojdbc14.jar file to the libraries of the project.

Right click on library folder → add jar → browse and select ojdbc14.jar file

### Step-7

Write following code in the execute(-,-,-,-) method of Action class.

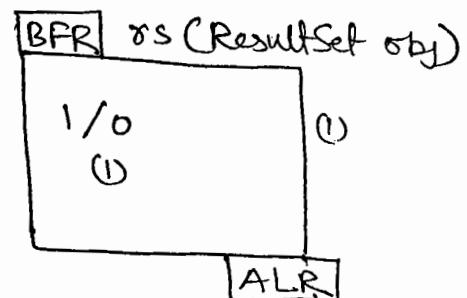
```
// Read form data
Registerform fm = (Registerform) form;
String un = fm.getUsername();
String pass = fm.getPassword();

// Write JDBC code
Class.forName("Oracle.jdbc.driver.OracleDriver");
Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl", "scott", "tiger");
Statement st = con.createStatement();
```

```

//Select count(*) from userlist where uname='raya' and
    pwd = '8ao';
String qry="Select count(*) from userlist where uname='"+un+
    " and pwd = "+pass+"'";
ResultSet rs = st.executeQuery(qry);
//process the ResultSet Obj
int cut=0;
if(rs.next())
{
    cut=rs.getInt(1);
}
//if
if(cut==0)
{
    request.setAttribute("res", "Invalid Credentials");
}
else
{
    request.setAttribute("res", "Valid Credentials");
}
return mapping.findForward("result");
}//execute(--),
}//class

```



### Step - 8

Add following code in Register.jsp to read and display the results.

In Register.jsp

taglib=bean  
taglib=logic

```
<logic:NotEmpty name="res">  
<bean:write name="res"/>  
</logic:NotEmpty>
```

### Step-9

Run the project

Right click on project → Run

Q: What is sql injection problem and how to resolve the problem?

Ans: Allowing endUser(Hacker) to supply special sql instruction along with the inputs and destroying query behaviour is called sql injection problem. Jdbc Simple Statement obj based sql query execution rises the sql injection problem because it makes database s/w to compile the given sql query along with input values so enduser(Hacker) can send sql instruction along with input values.

In the above application we can feel sql injection as shown below.

username = raja' --  
password = Hyderabad  
                ↳ wrong password  
remember     

Valid Credential

We can solve sql injection problem by using of jdbc PreparedStatement object because it sets input values to query after compilation of query in database s/w. So the special sql instructions (like --) that are given with input values will not be recognised as

SQL instructions and they cannot change query behaviour

Improve JDBC code in StrutsApp2 in execute() method of Action class.

```
PreparedStatement ps = con.prepareStatement("Select  
Count(*) from userlist where username=? and  
pwd=?");
```

```
ps.setString(1, un);
```

```
ps.setString(2, pass);
```

```
ResultSet rs = ps.executeQuery();
```

### Properties File

17.08.13

→ The text file that maintains entries in the form of key = value pairs is called as properties file / resource bundle file / Application resource file.

#### myfile.properties

# demo → # demo property file

for comment  
in property  
file

name = raja

age = 30

address = hyd.

### Properties file in StrutsApp

→ must be placed in WEB-INF\classes folder and its sub-folder.

→ Every property file must be configured in struts.xml file using <message-resources> tag.

→ We can place multiple properties file in a struts

application.

### Uses of Properties file

- To maintain presentation labels of view layer resources.
- To maintain form validation error messages.
- While implementing I18n concepts in Struts application.
- To maintain JDBC properties (driver class name, url, db user name, db password, and etc) to make the JDBC code of Struts Action class as flexible as code to modify and etc...

Adding properties file support in Struts application to maintain presentation logic levels.

#### Step-1

→ Keep the 1st Struts appln ready.

#### Step-2

myfile.properties (WEB-INF/classes folder)

# properties file

my.header = Welcome to Struts

my.footer = ©copy; All rights Reserved.

my.user = Enter Username

my.pwd = Enter password

(key) → my.cap = Login

Note : (key) → (value)

① In Struts env... the properties file must have .properties as the extension ("myfile.properties" must write in double quote commas)

②

#### Step-3

→ Configure the above properties file in struts-config file  
- using <message-resource> tags

In Struts-Config.xml  
(After </action-mapping>)

<message-resources parameter="myfile" />

#### Step-4

→ Read and display the messages of properties file in jsp program by using <bean-message> tag

#### In register.jsp

<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html"%>

<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean"%>

<center><b><bean:message key="my.header"/></center>  
<br>

<html:form action="register">

<bean:message key="my.user"/><html:text property="username"/><br>

<bean:message key="my.pwd"/><html:text property="password"/><br>

<html:submit>

<bean:message key="my.cap"/>

</html:submit>

</html:form>

<center><b><bean:message key="my.footer"/></center>  
</b><br>

#### Note

1. levels kept in properties file gives the flexibility of modification and becomes ready to use in multiple resources.

→ To make one message working as multiple presentation levels we can design message with argument. each message can have max of - five argument  $\{0\}$ ,  $\{1\}$  ...,  $\{4\}$ . To supply values to these arguments we can use ~~arg $\{0\}$~~ , ~~... arg $\{4\}$~~  arg $0$ , ..., arg $4$  attribute of <bean-message> tag

Example (with respect to above scenario)

In myfile.properties

~~msg mylabel~~ my.lbl=Login $\{0\}$

In register.jsp

```
<bean-message key="my.lbl" arg $0$ ="username"/><html:text  
property="username"/><br>  
<bean-message key="my.lbl" arg $0$ ="password"/><html:text  
property="password"/><br>
```

Note

- 1 → Modification done in struts-config file and properties file will be reflected after reloading of the web application.
- 2 → Modification done in web.xml, jsp prog, Html prog will be reflected directly.
- 3 → Modification done in java source file will be reflected only after recompilation of .java file and reloading of the web application.
- For better performance and for faster retrieval of msgs, we should not place more than 500 msgs in one properties file if needed prefer taking multiple properties

file while configuring these multiple properties file in struts-config file we must provide logical name to each properties file by using the attribute called key

### Example

#### Step-1

→ Keep 1st appln ready

#### Step-2

→ Create multiple properties files

//myfile.properties (WEB-INF\classes folder)

# properties file1

my.lbl = Login \$0{

my.cap = Login

//myfile.properties (WEB-INF\classes\app)

# properties file2

my.header = welcome to struts

my.footer = &copy; All rights Reserved.

#### Step-3

→ Configure the above two properties file in struts-config.xml file having logical names.

#### In struts-config.xml

<message-resources parameter="myfile" key="f1" />

<message-resources parameter="app myfile" key="f2" />

#### Step-4

→ Use the messages of both properties file in register.jsp by identifying properties file with "bundle" attribute.

## In register.jsp

```
<center><b><bean: message key="my.footer" bundle="f2" />
</center></b> <br>
<html: form action="register">
<bean: message key="my.lbl" arg0="username" bundle="f1" />
<html: text property="username" /><br>
<bean: message key="my.lbl" arg0="password" bundle="f1" />
<html: password property="password" /><br>
<html: submit>
<bean: message key=
```

→ To add new properties file to the struts app<sup>M</sup> of NetBeans  
IDE ~~IDE~~

Right click on project → New → Other → Other → properties file  
→ next ----

→ This properties file must be configure explicitly in  
struts-config.xml file.

→ We can also use properties file to make the  
jdbc code of Action class as flexible code to  
modify by supplying driver class name, url, dbuser  
name, password from properties file.

DBA changes db password at regular intervals for security reason in that situation it is good send password to ~~the~~ application from properties file.

### Example

#### Step-1

→ keep StrutsApp2 ready. (In NetBeans IDE appl") (refer Aug 16th discussion)

#### Step-2

→ Write jdbc properties in application Resources.properties file of project

##### # jdbc properties

```
my.driver = "oracle.jdbc.driver.OracleDriver"  
my.url = jdbc:oracle:thin:@localhost:1521:orcl  
my.dbuser = scott  
my.dbpwd = tiger
```

#### Step-3

Write following code in the execute(-,-,-) method of Action class. to read and use jdbc property from properties file

```
execute(-,-,-)
```

```
{
```

```
//read form data
```

```
---
```

```
// get Access properties file
```

```
MessageResources ms = this.getResources(request);
```

```
// read jdbc properties from properties file
```

```
String s1 = ms.getMessage("my.driver");
```

↳ key in the properties file

```

String s2 = ms.getMessage("my.wd");
String s3 = ms.getMessage("my.dbuser");
String s4 = ms.getMessage("my.dbpwd");
// write jdbc code
Class.forName(s1);
Connection con = DriverManager.getConnection(s2, s3, s4);
-----
} // execute(----)

```

19.08.13

- The java class that allows us to create only one object per jvm is called Singleton java class.
- If underlying container or server or programmer is creating only one object even though that class allows us to create multiple object then that ~~class~~ class is not called as Singleton java class.
- ActionServlet is not a ~~singleton~~ singleton java class, in fact no servlet class is singleton java class.
- Working with different types of Form Components.

<u>Form Component</u>	<u>Form bean</u>	<u>Property type</u>
TextBox	→ String or simple type(int, float, long...)	
Password	→ String or simple type(int, float, long..)	
TextArea	→ String	
RadioButton (grouped)	→ String	
CheckBox	→ String	
checkboxes (grouped)	→ String[]	
SelectBox/ComboBox (allows to select 1 item at a time)	→ String	

- listBox (allows to select multiple item at a time) → string []
  - File Uploading → Formfile (org.apache.struts.upload pkg)
- We can group multiple radio buttons or, multiple checkboxes into single unit by giving same name to them. When radio buttons are grouped we can select only one radio button at a time from that group. When checkboxes are grouped we can select multiple checkboxes from that group.
- It is always recommended to design form page as the content of html table.
- Procedure to develop Struts app<sup>n</sup> that deals with different types of Form Components by using NetBeans IDE

### Step-1

- Create web project (Struts App<sup>3</sup>) and add struts capabilities to project.
  - \* file menu → new project → java web → web app<sup>n</sup> → project name = StrutsApp3 → next → Server = Glassfish 2.2 → next → select Struts 1.3.8, select add struts tld → finish.
  - delete index.jsp, welcomestruts.jsp file .

### Step-2

- Add Form page to Struts app<sup>n</sup>.
  - Right click on web pages folder → new → Jsp → Jsp file Name = Input → finish.

### Input.jsp

→ Prefer Page No-10 and 11 of booklet

iPath = action path of InputAction class

### Step-3

→ Add FormBean class to the project having logical name

\* Right click on source packages Folder → new → Structs Action Form Bean → class name = InputForm , package = P1 → finish → declare the following FormBean properties that generate getter and setter method on them.

```
private String name;  
private int age;  
private String address;  
private String ms;  
private String gender;  
public String quby;  
public String[] cars;
```

→ InputForm.java refer page No-11 and 12 of booklet

### Step-4

→ Add Action class to the project

\* Right click on source packages → new → Structs Action...

→ class name = InputAction, package = P1 , Action path = iPath → next → finish

### Step-5

→ Configure Result.jsp as the result page for the above ~~Input~~ InputAction class.

→ refer struts-config.xml of page No-11

### Step - 6

→ Write following logic in the execute(-,-,-,-) method of InputAction class.

\* refer InputAction.java of page-13

```
request.setAttribute("name", name);  
          ↑           ↑  
Attribute   Attribute  
name        value.
```

→ add result.jsp to the web pages folder having logic to read and display request.Attribute values created in ActionClass.

\* Right click on web pages folder → New → JSP → ~~result~~ filename = result → finish

### Result.jsp

→ refer page No-14 of Booklet.

### Step - 7

→ Run the project

406 to 411 → (A) → The equivalent java code

for the above code is :-

```
<%! if(pageContext.getAttribute("hb") != null)  
{  
    for(int i=0; i <= pageContext.getAttribute("hb").length; i++)  
    { %>  
        <%= pageContext.getAttribute("hb") %>  
    } %>
```

In material

### Result.jsp

375 - name = name // refer line 338

401 - ~~Collect~~ name = "cscs"

408 - / name = "hb"

402 - id = item // acts as counter variable holding ~~one~~ one element of given array or collection in each iteration

414 - request Upplacing

java code in jsp is not industry standard so replace with bean writer tag

359 - // refer line 399

399 - csc // refer line 389

## UNDERSTANDING FORMBEAN LIFE-CYCLE

20.08.13

- EndUser fills up the form page and submits the request.
- The Controller Servlet (Action Servlet) traps and takes the request
- Action Servlet creates/locates Form Bean class object.
- If created, the Form Bean class obj will be placed in the specified scope.
- Action Servlet calls reset() method on the Form Bean class object.
- Action Servlet calls setXXX(-) on Form Bean class object and writes the received form data of form page to form bean properties.
- Action Servlet calls validate(-,-) method on Form Bean class object and this validate(-,-) method returns ActionErrors class object back to Action Servlet.

→ If ActionErrors object is empty (No form validation)

Focuses

ActionServlet creates/locates Action class obj and calls execute(-,-,-) methods on that object.

If ActionErrors obj is not Empty  
(If form Validation & errors are there)

ActionServlet forwards the control to input jsp program to display form validation error msg.

⇒ Special behaviour of CheckBox and ListBox

- Except checkBox, listBox Form components the remaining all form components generate form request parameters irrespective of whether they are selected or not filled up or not.
- The setter methods of Form Bean class executes only when there related form components generate request parameters.

⇒ UNDERSTANDING THE PROBLEM OF CHECKBOXES AND LISTBOXES WHILE WORKING WITH "SESSION" SCOPE FORMBEAN IN STRUTS APPLICATIONS:-

→ When formBean scope = "session" the ActionServlet uses same FormBean class object across the multiple request on one per Browser window basis.

→ In this process there is a possibility of showing selected state data in the form Bean properties of checkbox, listbox even though they are not selected. as shown in the below diagram.

The diagram shows two JSP pages, R1 and R2. R1 has fields Name (Raja), Marital Status (Married checked), and Courses (Java, .Net, Oracle). R2 has fields Name (Ravi) and Courses (Java, .Net, Oracle). Both pages have a 'Submit' button. A circled 'R1' is above R1, and a circled 'S1' is above R2. A circled 'R2' is above R2, and a circled 'S2' is above R2.

Session scope - RegisterForm class object

(R4) Name = raja | ravi  
ms = Married | married  
crs[] = Java, .Net, Oracle... Wrong data even though checkbox and listbox are not selected req2 to ~~req1~~

R1 → R4 = Request1 related flow

S1 → S4 = Request2 related flow

RegisterForm.java (FormBean) CScope = "session")

public class RegisterForm extends ActionForm

```
{
    String name;
    String ms;
    String[] crs;
```

(R3)

```
public void setXXX()
{
    ...
}
public xxx getXXX()
{
    return XXX;
}
```

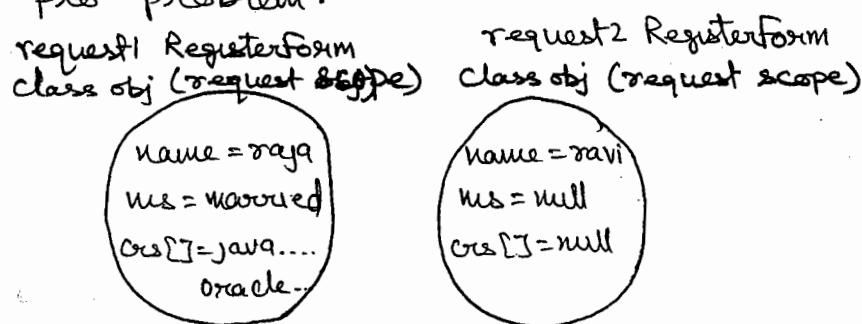
(only setter method setXXX(-))

~~step Setter method (setXXX(-))~~

There are two solution for the above problem:-

### Solution -1

- Change the FormBean scope to request that means for every request a separate FormBean class object will be created and there is no possibility of holding previous request data in FormBean properties.
- Solution 1 is not recommended because it is not direct solution to the problem. It is something like escaping from ~~the~~ problem.



### Solution -2

~~inside~~ → Override reset(-,-) method in FormBean class and assign not selected state values to the properties of checkbox, listbox components.

→ In this solution 2 we can keep Form bean in "session" scope itself.

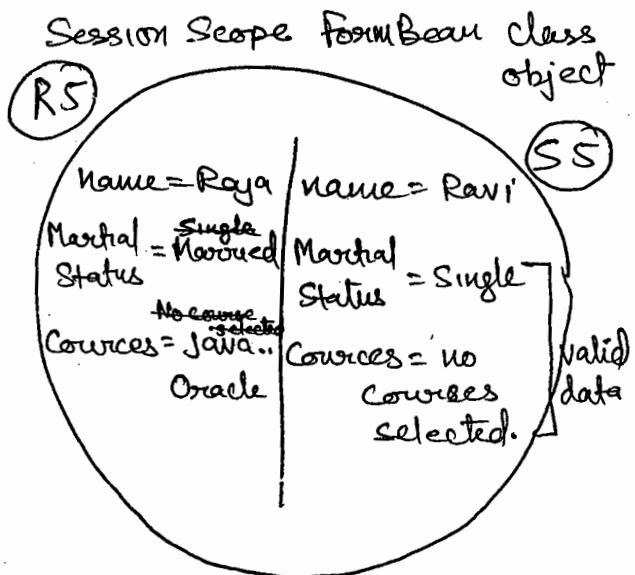
→ reset(-,-) method executes for every request before executing setXXX(-) method. in Form Bean class.

→ The prototypes of the overloaded forms of reset(-,-) methods are:-

- a) public void reset(ActionMapping mapping, ServletRequest req)
- b) public void reset(ActionMapping mapping, HttpServletRequest req)

Form(b) is recommended to use.

<b>R1</b>	<b>S1</b>
Name <input type="text" value="Raya"/>	<input type="text" value="Ravi"/>
Marital Status <input checked="" type="checkbox"/> married	<input type="checkbox"/> married
Courses <input checked="" type="checkbox"/> Java <input type="checkbox"/> .Net <input type="checkbox"/> Oracle	<input type="checkbox"/> Java <input type="checkbox"/> .Net <input type="checkbox"/> Oracle
<b>Submit</b>	<b>Submit</b>



**R2**      **S2**

Registerform.java (Form Bean) (Scope = "session")

public class Registerform extends Actionform

{  
    String name;  
    String ms;  
    String[] crs;

public void reset(ActionMapping mapping, HttpServletRequest req)

{

    ms = "Single";  
    crs = new String[1];  
    crs[0] = "no courses selected";

{  
    public void setXXX()  
{

    ---      **(S4)** → only one setter method execute (setName);  
    ---

}  
    ---  
    ---  
    public XXX getXXX()  
{  
    ---  
    ---

**R1 → R5 = Request1 related flow**

**S1 → SS = Request2 related flow.**

- The only use of reset(-,-) method in struts is placing not selected state values for listbox, checkbox form Bean properties while working with "session" scope form Bean as shown above.
- For example app<sup>1</sup>" of working with different form component and working with reset(-,-) method prefer App<sup>1</sup>"2 of the page No-10 to 14 .

## FORM VALIDATIONS:-

21.08.13

- Verifying the pattern and format of form data is called Form Validation and the logic used for it is called form Validation logic.

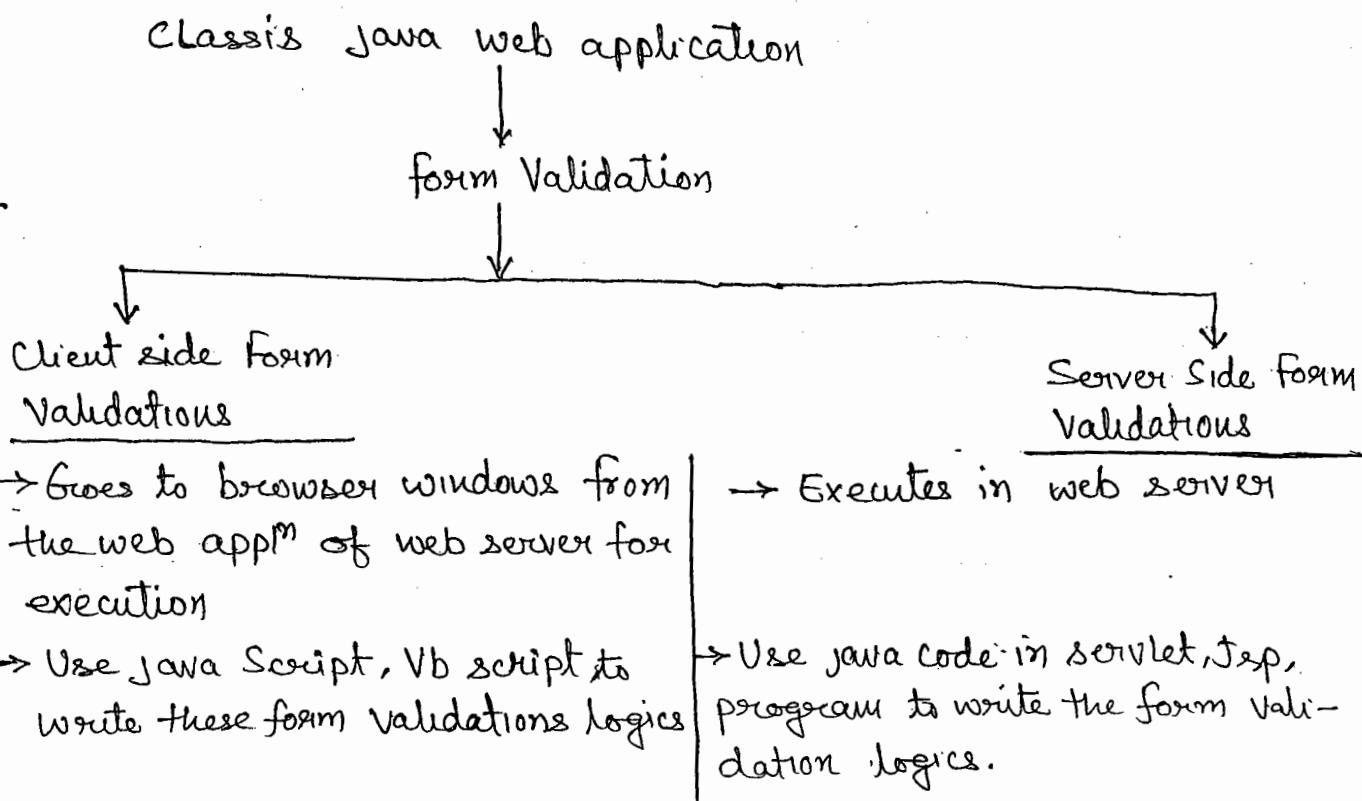
Q:- What is the difference between form Validation logic and Business logic.?

- Form validation logic verifies the pattern and format of form data. ~~check~~  
Eg:- checking whether username and passwords are typed or not  
checking emails having ".","@" symbols or not  
checking password and re-type password are matching or not.
- Business logic uses form data as the input data and generates the results.  
Eg:- inserting form data into db table as record.

- Checking whether email id is valid or not by talking with DB SW.
- Performing  $c = a + b$  operation to generate results.

### Note

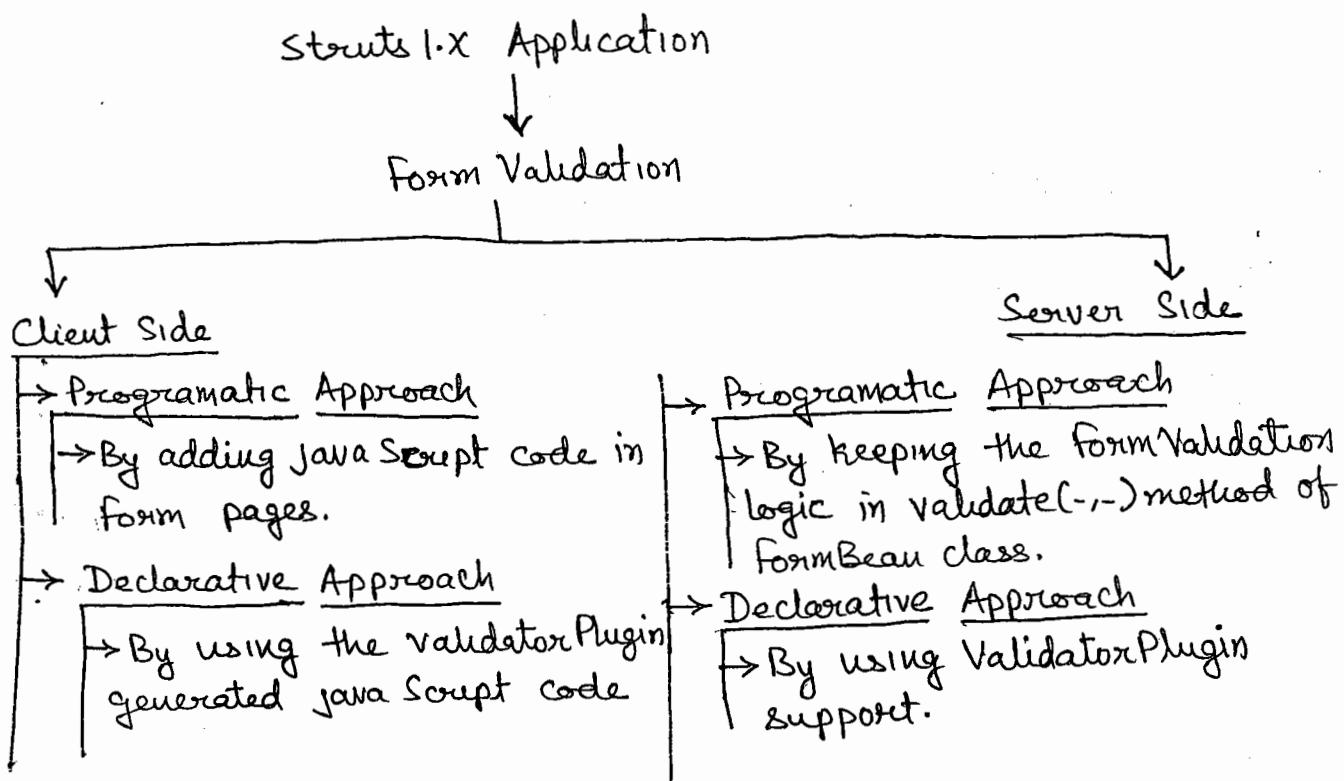
It is always recommended to use the validated form data as the input values of Business logic.



- Goes to browser window from the web app<sup>m</sup> of web server for execution
- Use Java Script, Vb script to write these form validation logic
- Decide whether ~~form~~ the form validation logic is server side or client side based on the place where that logic executes not based on the place where that logic resides
- Only server side form validations ~~will~~ increases network round trips between browser window and web application.
- Only client side form validation reduces the above network round trips but there is a possibility of disabling script code execution through browser setting by user.

→ Writing both client side and server side and executing them every time kills the performance of web app<sup>n</sup>.

→ To overcome the above problems the industries prefers the following steps to perform form validation. The s/w industries writes both client side and server side form validation but they execute server side validation only when the client side form validations are disable.



→ Writing logic manually is called Programmatic Approach.  
→ Working with readily available build-in logic and configuring them with app<sup>n</sup> is called declarative approach.  
    ↑  
    using XML entries

→ Validator plugin gives 14+ validator rules by having both java code and java Script code.  
→ When validator plugin is used for server side form validations then java ~~Script~~ code will be generated dynamically.  
→ When validator plugin is used for client side form

validations then the java script code will be generated dynamically.

- For server side programmatic form validations we need the form validation logic in the validate(-,-) method of form bean class.
- The prototypes of validate(-,-) methods are
  1. public ActionErrors validate(ActionMapping mapping, ServletRequest req)
  2. public ActionErrors validate(ActionMapping mapping, HttpServletRequest req)

Note :-

2nd form is recommended to use.

- If this validate(-) methods returns ActionErrors obj with element (size>0) to ActionServlet indicating there are form validation errors then ActionServlet forwards the control to input page to display form validation error msgs otherwise (If ActionErrors object is empty (size=0)) ActionServlet calls execute(-,-,-,-) method of Action class to process the request.

- The Jsp program that is capable of displaying form validation error msgs is called input page... any Jsp program can be taken as input page. It is recommended to take form page as the input page of the web app<sup>n</sup>

Q:- Example app<sup>n</sup> to perform server side programmatic Form Validation in struts app<sup>n</sup> ?

Step-1

Keep 1st struts app<sup>n</sup> ready.

## Step-2

Configure properties file to stout app<sup>n</sup> by keeping form validation error msgs in that properties file.

### WEB-INF\classes\myfile.properties

un.required = Username is mandatory { //Form Validation  
pwd.required = Password is mandatory } error message.

### In struts-config.xml

<message-resources parameter="myfile" />

## Step-3

Override validate() method in form bean class having server side programmatic form validation logic

### In RegisterForm.java

```
public ActionErrors validate(ActionMapping mapping, HttpServletRequest
                           req)
{
    ActionErrors errs = new ActionErrors();
    //write server side programmatic form validation
    //required logics on username.
    if(username == null || username.equals("") ||
       username.length() == 0)
    {
        errs.add("username", new ActionMessage("un.required"));
        //logical name
        //Recommended to take FormBean property name
    }
    // required logics on password
    if(password == null || password.equals("") ||
       password.length() == 0)
    {
        errs.add("password", new ActionMessage("pwd.required"));
        //key in the properties file
    }
    System.out.println("errs obj size = " + errs.size());
    return errs;
} // Validate(-,-)
```

ActionErrors obj	
Keys	Values
String username	ActionMessage obj username is mandatory
String Password	ActionMessage obj Password is mandatory

### Step - 4

→ Configure form page as the input page for Action class to display form validation error msgs.

#### In struts-config.xml

```
<action-mappings>
<action input="/register.jsp" path="/register" type="app.RegisterAction" name="rf">
    <forward name="success" path="/success.jsp" redirect="true"/>
    <forward name="failure" path="/failure.jsp" redirect="true"/>
</action>
</action-mappings>
```

### Step - 5

→ Add `<html:errors>` tag in input page (`register.jsp`) to specify the location of displaying form validation error msgs.

#### In register.jsp

```
<html:errors /> // at the end of html code.
```

### Step - 6

→ Test the application.

Q: What is the difference between `ActionErrors` obj and `ActionMessage` / `ActionError` obj.

Aus `ActionErrors` object represents all form validation errors of form page. Each `ActionError` / `ActionMessage` object represent one form validation error msgs.

From struts 1.3 `ActionError` class is removed so use `ActionMessage` class instead of it.

#### Note

→ Each element of `ActionErrors` obj represents one form validation error ~~message~~ including the ~~error~~ message.

22.08.13

→ To display the Form validation errors generated by the above appl" by applying styles we need to place the following special key based messages. in myfile.properties file.

In myfile.properties

fixedErrors. header = <font color="red" size="2"><u>  
key } errors. footer = </font> </u>  
{ errors. prefix = <li>  
errors. suffix = </li>  
myfile = <b> Welcome to Struts </b>

→ To display Form validation error messages beside the form component of form page use <html:errors> tag along with property attributes.

In register.jsp

```
<html:form action="register">
username: <html:text property="username"/> <html:errors
property="username"/> <br/>
          ↴(*) [the logical names of form Validation errors given
Password: <html:password property="password"/> <html:errors
property="password"/> <br/>
          ↴(*) [the logical names of form Validation errors given in
<html:submit value="login"/> Validate method in FormBean class
          these messages generally matches
</form>           with FormBean properties name.]
```

→ When we develop multiple form validation logic based on single FormBean property data it is recommended to give same logical name for multiple form Validation errors of that FormBean properties. So that we can use that logical name while specifying the location of displaying Form Validation error messages.

(w.r.t previous App<sup>n</sup>) In validate method of RegisterForm class.

```
// required logic on username
if (username == null || username.equals(" ") || username.length() == 0)
{
    errors.add("username", new ActionMessage("un.required"));
    {
        Logical name
    }
    else
{
    // Check whether username's first character is alphabet
    or not
    char fchar = username.charAt(0);
    if (!Character.isUpperCase(fchar) && !Character.isLowerCase(fchar));
    errors.add("username", new ActionMessage("un.fchar"));
    {
        Logical name.
    }
}
```

### In myfile.properties

un.fchar = Username must begin with alphabet

⇒ Writing client side programmatic Form Validations by adding JavaScript to form page.

#### Step-1

→ Keep 1st app<sup>n</sup> ready. with previous class enhancement  
(Server Side Validation)

#### Step-2

→ Write ~~following~~ code in the form page (Register.jsp)  
JavaScript

Note :- While writing this code also take care of sending flag  
to server side to enable server side form validation only  
when client side form validation are not done

#### Step-3

→ enhance the code of Validate(--) method of FormBean  
class to receive flag send by form page indicating

Client side form validation status and to enable/disable  
Form Validation (Server side)

- For above step based complete example appl<sup>n</sup> that contains programmatic client side and server side form validation and enables server side form validation only when client side form validations are not done. Prefer the application given in page No-15 to 18.

→ To disable script code execution in Internet Explorer.

Tools menu → Internet option → security → custom level → Scripting → ~~Do~~ Active Scripting → disable → ok → ok.

### In NetScape

~~Tools~~ → Edit menu → advanced → Scripts & plug-ins → deselect Navigator



## Plugins

23.08.13

- Plugin is a patch s/w or s/w application which can enhance the functionality of existing s/w or s/w applications.
- Struts gives two build-in plugins

### 1) Validator plugin

- To perform form validation declaratively.

### 2) Tiles plugin

- To arrange web page content having logical partition.
- Every struts plugin will be identify with java class that implements org.apache.struts.action.plugin(I). To use struts plugin in struts appln, the plugin related class must be cfg in struts cfg file using ~~<plug-in>~~ <plug-in> tag and plugin related jar files must be added to WEB-INF\lib folder.

- Validator plugin gives 14+ validator rules having both java code, java script code to perform both client side and server side form validations ~~logic~~ declaratively.

- To use Validator plugin in struts appln its plugin class must be cfg in struts configuration ~~file~~ file as shown below.

In struts.cfg file      Class name →

```
<plug-in className="org.apache.struts.validator.ValidatorPlugin">
<set property property="pathnames" fixed property
      value="/WEB-INF/validator-rules.xml, /WEB-INF/
            validation.xml" />
</plug-in>
```

- Validator plugin supplied 14 rules related java code location, java script location and default error msgs info. is available in validator-rule.xml file. It is predefined file and we gather it struts-core-1.3.8.jar file extractions.
- Programmers uses validation.xml file to apply validator rules of validator plugin on formBean properties to validate the form data declaratively.
- Validator-rules.xml, validation.xml file names are not fixed...
- While developing struts app we gather validator-rules.xml from struts S/W installation, but we develop validation.xml file manually.
- The important validator plugin or ~~required~~

required	byte	email
mask	int	creditcard
minlength	short	etc...
maxlength	long	
range	float	

Since these rules are predefined some default error msgs are given

~~errors.required = {0}~~ is required

~~errorsmaxlength = {0}~~ can<sup>not</sup> be greater than {1} character  
~~errors.email = {0}~~ is an invalid Email ID.

---

---

etc...

Note :-

Default error msgs are given with arguments to use them

for multiple form components related error msg.

- ActionServlet activates the cfg validator plugin either during server startup or during the deployment of struts appn, because ActionServlet reads struts-cfg file entries at that time because of <load-on-startup> that is enabled on ActionServlet
- This activated validator plugin will perform form validations based on the configuration done in validation.xml file when validate method of predefined ValidatorForm class execute  
↳ sub class of ActionForm.

for this we need to do the following two operations in struts appn to work with validator plugin.

- ▷ ① Make your FormBean class extending from org.apache.struts.validator.ValidatorForm
- ▷ ② Call super.validate() method from the validate() method of our FormBean class or don't override validate() method in our FormBean class.
- The validate() method of predefined ActionForm class cannot perform the validator plugin based form validation So while working with validator plugin our FormBean class must extends from predefined ~~for~~ ValidatorForm class.
- When ActionServlet calls our validate() method of FormBean class programmatic form validation takes place. if same ActionServlet calls validate() method of ValidatorForm class (directly or indirectly) then the declarative form validation will takes place by using validator plugin.

⇒ Procedure to perform declarative server side form validation by using required rule of validator plugin

### Step-1

→ Keep 1st appl<sup>n</sup> ready.

### Step-2

#### Note:-

Make sure that commons-validator-1.3.1.jar file is available in WEB-INF\lib folder along with other jar files.

### Step-2

Configure Validator plugin in struts configuration file

struts-config.xml (after <message-resources> tag)

after </action-mapping> tag.

```
<plug-in className="org.apache.struts.ValidatorPlugin">
<set-property property="pathnames"
  value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"
  />
</plug-in>
</struts-config>
```

#### Note

⇒ Gather above code from

<Struts-home>\app\struts-blank-1.3.8.jar file extraction related struts-config.xml file.

⇒ Gather validator-rules.xml file from.

<Struts-home>\lib\struts-core-1.3.8.jar file extraction and place that file in WEB-INF folder.

### Step-3

→ Create properties file in WEB-INF/classes folder having default error messages gathered from the XML comments of Validator-rules.xml file. (line no- 42 to 57)

Ex: errors.required = {0} is required.

---

---

### Step-4

→ Configure the above properties file in struts-config file.  
 (In struts-config.xml) ~~In~~ after </action-mapping>  
<message-resources parameter="my file" />

### Step-5

→ Make sure that formBean class is extending from org.apache.struts.validator.ValidatorForm, and also ensure no validate(-) method is overridden in that class.

In Registerform.java

```
import package app;
import org.apache.struts.action.*;
import org.apache.struts.validator.*;
public class Registerform extends ValidatorForm
  ↳ Mandatory
{
```

--- // Same as 1st app!

{

24.08.13

### Step-6

Add validation.xml file to the WEB-INF folder of struts app<sup>n</sup> having configuration to apply required validator rule on username, password properties.

#### validation.xml

<DOCTYPE --- &gt;validator\_1\_3\_0.dtd">

```

<form-validator>
  <formset>           logical name of formBean class to clear if
    <form name="ff">   the formBean class to validate
      <field property="username" depends="required"> plugin
        arg position="0" key="my.un"/>
        </field> supplies {0} value for key in the properties file
        "required" rule default error msg.
      <field property="password" depends="required">
        arg position="0" key="my.pass"/> // required rule config on password
        </field> formBean Property
      </form>
    </formset>
  </form-validator>

```

### Note :-

Develop the above validation.xml by using struts-home \apps\struts-blank-1.3-war file related example appM content.

### Step - 7

Configure input page for RegisterAction class to display form Validation error msg.

In struts-efg.xml

```

<action-mapping> form page acting as input page (it is page capable of displaying error msg)
  <action input="register.jsp"> path="/register" type="app.."
    RegisterAction name="rf" >
    -----
    -----
    </action>
  </action-mapping>

```

### Step-8

→ Add `<html:errors>` in input page (`register.jsp`) to specify the location of displaying form validation error msg.

#### In register.jsp

```
<html:form action = "register">  
username = <html:text property = "username"/>  
    <html:errors property = "username"/>  
password = <html:text property = "password"/> logical name of form validation error.  
    <html:errors property = "password"/>  
<html:submit value = "register"/>  
</html:form>
```

### Step-9

→ Test the application.

#### Note

- The validate method of Validator Form class uses formBean property as logical names while adding Form Validation errors to ActionErrors class object.
- For the above steps based complete example app<sup>n</sup> refer the app<sup>n</sup> given in page No - 19 to 22

#### → Flow of execution of above application :-

Validator plugin will be activated either during server startup or during the deployment of web app<sup>n</sup> along with ActionServlet pre initiation → end user launches form page and sends the request to struts app<sup>n</sup> → ActionServlet traps and takes the request and populates Form data to the formBean class object → ActionServlet calls validate() method on FormBean

class object, since it is not available the super class (validateForm) validate() method executes. → This validate() method uses the activated validator plugin and performs server side declarative form validations on FormBean properties based on config. done in validation.xml, validators-rules.xml and also returns ActionErrors object to ActionServlet. If this ActionError object size is zero(0) control goes to Action class execute(...,...) method otherwise control goes to input page (register.jsp).

→ Procedure to perform client side declarative form validation by using validator plugin.

### Step-1

→ Keep StoutsApp-DV app ready (refer page 19 to 22)

## Step-2

→ Add `<html><head><script>` tag in form page. to get validator plugin generated javascript code.

### Step-3

call the validator plugin generated javascript function against "onsubmit" event whose name is ~~validate~~ validate<formBeanLogicalName>(form)

In register.jsp

```
<html:form action = "register" onsubmit = "return validateRF(this)">
<html:javascript formName = "rf" />
-- username: --.                                ↳ formBean logical name
-- --.
</html:form>
```

### Step-4

→ Test the app'n

### Note

- The above app<sup>M</sup> is not disabling server side ~~form~~ validation
- When client side form validation is done, for that we need to do some additional activities.

### Step-1

→ Keep StrutsApp-Dv app<sup>M</sup> ready with both server side and client side form validations.

### Step-2

→ Add userdefined javascript function in form page calling the validator plugin generated validateRf(form) function and sending flag to server that client side form validators are done.

#### In register.jsp

```
<html:html>
<head>
<script language = "JavaScript" >
function myValidate(form) (form)
{
    // set "yes" to vflag indicating client form validations are
    // done
    form.vflag.value = "yes" ;
    // call validationplugin generated function
    var flag = validateRf(form) ;
    return flag ;
}
</script> </head>
```

```

<html:form action="register" onsubmit = "return myValidate(this)">
<html:javascript formName="xf">
    username:<html:text property="username"/><html:errors -->
    password:<html:text property="password"/><html:errors prop -->
<html:hidden property="vflag" value="no"/>
<html:submit value="register"/>
</html:form>
</html:html>

```

### Step-3

→ override validate() method in FormBean class and call super.validate(-,-) method only when client side form validations are not done. (vflag = "no")

#### In RegisterForm.java

```

public ActionErrors validate(ActionMapping mapping,
                            HttpServletRequest req)
{
    //get vflag req param value
    String vstatus = req.getParameter("vflag");
    ActionErrors errs = null;
    if(vstatus.equals("no")) //enable server side form validate
        only when client side validation
        errs = super.validate(mapping,req); are not done
    else
        errs = new ActionErrors();
    return errs;
}
//validate(-,-)

```

26.08.13

→ While working with validator rules of validator plugins supply  $\{0\}$  value of error messages from properties file and supply  $\{1\}, \{2\}$  - value from validation.xml file itself.  $\{0\}$  of error messages are there to just complete error messages whereas  $\{1\}, \{2\}, \dots$  of error messages are given to complete the error msgs. and to provide input values to the form validation logic of validator rules.

Ex:- Default error msg :-

errors.minLength =  $\{0\}$  can not be less than  $\{1\}$  characters.

errors.maxLength =  $\{0\}$  can not be greater than  $\{1\}$  characters.

$\{0\}$  Sample error msgs:-  $\{1\}$   
username can not be less than 4 characters. (minlength error msg)  
password can not be greater than 8 characters. (maxlength error msg)

→  $\{0\}$  values (username, password) are just required to complete error msgs. So pass them from properties file.

→  $\{1\}$  values {4/8} are required to complete error msgs and also as input values of form validation logic, so pass them from validation.xml itself.

→ While working with minlength, maxlength validator rules the  $\{1\}$  value must be supplied from validation.xml file using variable, this variables name must match with minlength, maxlength validator rules name.

→ If you write <arg> tag with resource = "false" attributes that indicates argument value of error msg should not be collected from property file and it should be collected from a variable.

of validator.xml whose name is specified in the same <arg> tag.

→ When multiple multiple validator rule is applied on formBean property and if <arg> tag is taken without specifying validator rule name then that <arg> tag supplied value becomes \${n} value of multiple validator rules related error msgs. otherwise the <arg> tag supplied value becomes specific to one validator rule related error msg.

→ Updated validation.xml of StoutApp-DV having required, minlength, maxlength validator rules:-

~~<form-validation>~~ validation.xml

~~<form-set>~~

<form-validator>

<formset>

<form name="rf"> *validator rule names.*

<field property="username" depends="required, minlength">

<arg position="0" key="my.un"/>

↳ supplies \${0} value of required, minlength rule error msg.

<arg position="1" name="minlength" key="\${var:minlength}" resource="false"/>

↳ variable name

indicates that this arg

value should be collected from // supplies \${1} value

<var-name> minlength </var-name> of ~~the~~ minlength

<var-value> 4 </var-value> *validator rule.*

</var> // defining variable with a value

</fields>

<field property="password" depends="required, maxlength">

<arg position="0" name="required" key="my.pass">

↳ supplies \${0} value of "required" rule error msg.

```

<arg position="0" name="maxLength" key="my.pass"/>
    ↳ supplies ${0} value of "maxLength" rule error msg.
<arg - position = "1" name = "maxLength" key = "${var:maxLength}" />
    resource = "false" /> // supplies ${1} value of
                                ↑
                                "maxLength" validator rule. (variable
                                name)
    { <Var>
        <var-name> maxLength </var-name> // supplying value to
        <var-value> 10 </var-value> the variable
    } </Var>
</field>

</form>
</formset>
</form-validation>

```

- The ActionForm class based formBean is not capable of performing validator plugin based server side form validation but can be used to perform validator plugin based client side form validations.
- ~~Validator based~~ ValidatorForm based FormBean class can be used to perform both server side and client side form validator plugins.
- While evaluating performance of Action class with or without form validation we get the need of disabling server side form validation (both programmatic, declarative) temporarily for this we need to place validate = "false" in <action> tag of struts-config.xml. This validate = "false" makes ActionServlet for ~~not~~ ignoring validate(-,-) method of FormBean life-cycle method calls.

```
<action validate="false" input="/register.jsp"
      path="/register"
      type="app.RegisterAction" name="rf">
  - - -
  - - -
</action>
```

### Note

- validate = "false" can disable server side form validation but not client side form validation.
- Any jsp program can be taken as ~~as~~ input page for Action class in struts app<sup>n</sup> but it is recommended to take the form page as input page to display form validation errors w.r.t. with respect to form component position.
- A Good struts app<sup>n</sup> developer will mix both programmatic (user-defined) and validator plugin's declarative form validation logics while performing form validation on struts app<sup>n</sup>. For this there are three approaches :-

Approach 1 Use mask validator rule of Validator plugin

Approach 2 Call super.validate(-,-) method from our validate(-,-) method of Form Bean class and call Validator plugin generated java script function from ~~the~~ our java script function of form page

Approach 3 Add custom validator rule in validator plugin.

## Approach-1

27.08.13

### Mask validator rule

- Mask is a pre-defined validator rule of validator plugins that allows regular expression based custom form validation logic.
- Mask rule uses errors.invalid key based msg as default error msg.  
 $\text{errors.invalid} = \{\emptyset\}$  is invalid.
- The regular expression given to mask validator rule can perform server side and client side form validations. If you are not happy with default error msg of any validator rule like mask then we can configure custom form validation error msg with or without arguments using <msg> tag

### Applying Mask validator rule

#### Step-1

keep StrutsApp-DV appln ready.

#### Step-2

Add custom form validation error msg for mask validator rule in properties file.

in myfile.properties

my.mask.un =  $\{\emptyset\}$  must contain only Alphabets

#### Step-3

Configure Mask validator rule on username property in validation.xml file as shown below

## In validation.xml

```
<field property = "username" depends = "required,mask">  
<arg position = "0" key = "my.un"/> // supplies ${0} value  
<msg name = "mask" key = "my.mask.un"/> of required, mask validator  
x1 { <var> defines custom validation error msg for "mask" validator  
<var-name> Mask </var-name> rule  
<var-value> ^[A-Za-z]* $ </var-value>  
</var> defines regular expression for "mask" validator rule  
</field> regular expression based form Validation logic to allow only alphabets.
```

→ Regular Expression to allow one or more alpha numerics

```
<var-value> ^[A-Za-z0-9]+ $ </var-value>
```

→ Regular expression to allow minimum 5 lower case alphabets

```
<var-value> ^[a-z]{5,} $ </var-value>  
↑ Start of regular expression      ↓ end of Regular Expression.
```

→ Regular expression to allow exactly 10 alpha-numerics.

```
<var-value> ^[A-Za-z0-9]{10} $ </var-value>
```

→ Regular expression to allow minimum 5 and maximum 10 alphabet.

```
<var-value> ^[A-Za-z]{5,10} $ </var-value>
```

→ Regular expression to allow either 0 alphabets or 1 alphabet.

```
<var-value> ^[a-zA-Z]? $ </var-value>
```

## → Limitations with "Mask" validator rule (Approach-1)

1. Habituating with regular expression takes time.
2. When it is applied on password component client side validation java script based form validation does not take place
3. Does not allow to use multiple FormBean properties data while writing regular expression based form validation logic

Q: What is regular expression?

A: It is a technique to write shorter version code for large scale and complex content

## Approach-2

Calling super.validate(-,-) from our validate method of FormBean class and calling validator plugin generated java script function from our javascript function of form page.

→ This approach-2 allows to use multiple FormBean property data while developing form validation logic like username, password ~~have~~ must have same length, or type password and retype password must match.

## Example

### Step-1

- Keep StrutsApp-DV app<sup>n</sup> ready.

### Step-2

- Call super.validate(-,-) method from the validate(-) method of FormBean class.

To check whether username and password having same length or not

In registerform.java

```
public ActionErrors validate(ActionMapping mapping, HttpServletRequest req) {
```

```
    String vstatus = req.getParameter("vflag");
```

```
    ActionErrors errors = null;
```

```
    if (vstatus.equals("no"))
```

```
    {  
        // enable server side form validation only when client  
        // side form validation are not done
```

```
        // perform server side declarative form validations
```

```
        errors = super.validate(mapping, req);
```

```
        if (errors.size() == 0)
```

```
        {  
            // when form validation errors are not there
```

```
            // write approach2 based server side programmatic form  
            // validation logic
```

```
// to check whether username and passwords are having  
// same length or not
```

```
if (username.length() != password.length())
```

```
{  
    errors.add("password", new ActionMessage("my.error.uspwlen"));
```

```
} // if
```

```
{ // if
```

```
else
```

```
    errors = new ActionErrors();
```

```
    return errors;
```

```
} // validate(--)
```

### Step-3

→ Add the following additional msg in the properties file  
my.unpwdlen = user name and password must have same length

### Step-4

28.08.13

→ In form page call the validator plugin generated JavaScript function in our userdefine JavaScript function that contains programmatic form validation logic.

#### In register.jsp

```
<html>
<head>
<script language="JavaScript">
function myValidate(form)
{
    //set yes to vflag indicating client form validation are done.
    form.vflag.value="yes";
    //call validation plugin generated function
    var flag = validateRF(form);
    if(flag==true) //when Declarative form validation error are not there
    {
        //write approach2 based client side programmatic form validation logic to check whether user name and password having same length or not
        if(form.username.value.length != form.password.value.length)
        {
            alert("username and pwd must have same length");
        }
    }
}
```

```
    form.password.focus();
    flag = false;
}

{
    return flag
}

</script>
</head>
<html:form action="register" onsubmit="return
myValidate(this)">
<html:javascript forName="rf"/>
    - - -
    - - -
<html:submit value="register" />
```

<html:form>  
<html:html>

(Approach 1)

→ Mask rule form validation logic is specific to that  
form bean property on which validator rule is applied

(Approach 2)

→ Form validation logic is specific to current formBean  
class/ current form page.

→ To give more visibility to custom form validation logic  
add custom validator rule to Validator plugin and  
use that rule ~~for~~ for multiple form bean properties  
of multiple FormBean classes.

→ Procedure to add custom validator rule to Validator plugin.

#### Step-1

→ Keep StrutsApp-DV ready.

#### Step-2

→ Define default error msg for custom validator rule.

In myfile.properties :-

my.errorone.myrule = \$0\$ must have same first and last characters.

#### Step-3

→ Develop Java class having custom validator rule related server side form validation logic.

MyCustRule.java (WEB-INF\classes folder)

refer MyCustRule.java of page no-26 and 27

~~Develop~~ Step-4

Add JavaScript file to Struts app<sup>n</sup> having client side form validation logic

ValidateMyRule.js (WEB-INF/classes folder)

refer page no-27 and 28.

#### Step-5

Configure the above custom validator rule in validator-rule.xml file using validator tag under global tag.

~~global~~

### In validator-rules.xml

→ refer page no-25 and 26.

#### Note:

The above steps completes custom rule development.

### Step-6

→ Configure the above custom validator rule in validation.xml

### In validation.xml

```
<form name="rf">
<field property="username" depends="required,mask,myrule">
-->
-->
```

```
</fields>
```

```
<field property="password" depends="required,maxlength,myrule">
-->
-->
```

```
</fields>
```

### Step-7

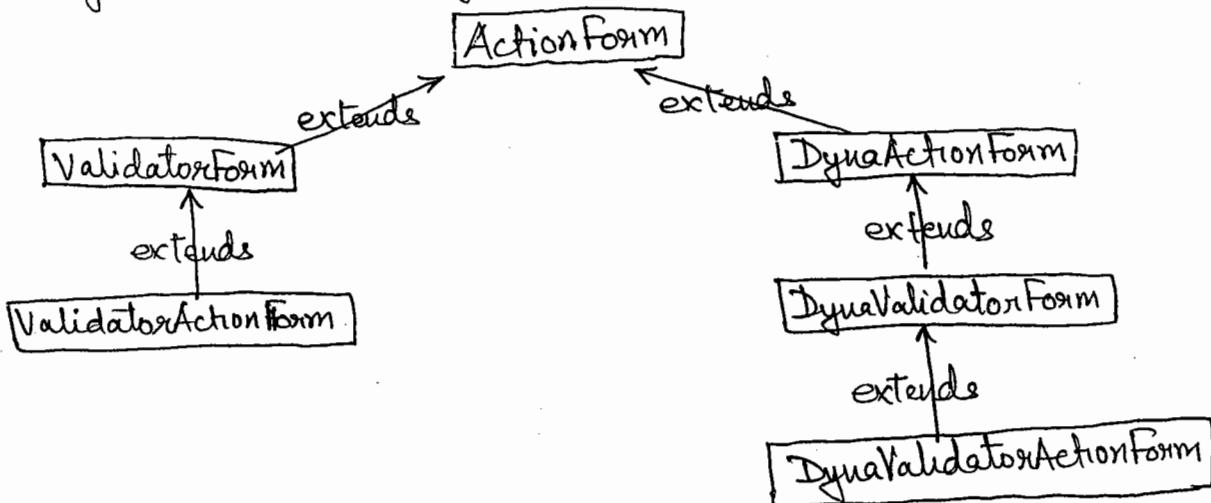
→ Test the appl<sup>n</sup>

29.08.13

→ For example appl<sup>n</sup> that uses all the three approaches to mixup programmatic form validations with validator plugin supplied validator rules refer appl<sup>n</sup> 4 of page Nos- 23 to 28.

## Dynamic Form Beans / Dyna Form Bean :-

- The form bean that gathers form bean properties dynamically from struts-config file and avoids the process of writing getter and setter methods is called Dyna Form Bean.
- To develop these Form bean the FormBean classes must extends from DynaXXX Form classes like DynaActionForm, DynaValidatorForm, DynaValidatorActionForm etc...



### To Two types of Dynamic Form Beans

#### 1. Declarative Dynamic FormBean:

- This FormBean total cfg will be done in struts-cfg file... i.e there is no need of developing separate java file for FormBean class.
- This ~~not~~ FormBean not recommended to use, because it does not give provision to programmer to place validate(-,-), reset(-,-) methods.
- Prefer this FormBean only when you are ready to use validator plugin declarative form validations and no checkboxes and listboxes issues are there.

#### 2. Programmatic Dynamic

- This FormBean contains formBean.java file having java class extending from DynaXXX<sup>form</sup> classes. So we can place `reset(-,-)`, `validate(-,-)` methods.
- This FormBean allows programmatic and declarative form validation
- In this FormBean there is no need of placing FormBean properties and `getXXX()`, `setXXX(-)` methods

issues are these.

- The DynaActionForm based dyna form bean does not allow us to work with validator plugin form validations.
- The DynaValidatorForm based dynaFormBean allows us to work with Validator plugin based Form validations.
- Example app<sup>m</sup> on Declarative Dynamic Form Bean (DynaActionForm based).

#### Step-1

- keep 1st app<sup>m</sup> ready.

#### Step-2

- Delete RegisterForm.java, RegisterForm.class files.

#### Step-3

- Configure DynaActionForm as Declarative Dynamic FormBean in struts-config file.

#### In struts-config.xml

```
<form-beans>
<form-bean name="rf" type="org.apache.struts.action.DynaActionForm">
<form-property name="username" type="java.lang.String" initial="raya"/>
<form-property name="password" type="java.lang.String"/>
</form-bean>
</form-beans>
```

↑  
FormBean property must match with  
form component name.

#### Step-4

- Read form data from FormBean class obj in execute() method as shown below.

#### In RegisterAction.java

```
DynaActionForm fm = (DynaActionForm) form;
```

```
String user = (String) fm.get("username");
String pass = (String) fm.get("password");
    ^ method of DynaActionForm class.
```

### Step-5

- Run the appl<sup>n</sup>
- Example appl<sup>n</sup> on Programmatic Dynamic FormBean.  
(DynaValidatorForm based)

#### Step-1

~~keep strutsapp~~

→ Keep strutsApp-DV appl<sup>n</sup> ready  
(or)

→ Appl<sup>n</sup> 4 of page - 23 to 28 of booklet.

#### Step-2

→ Make formBean class extending from DynaValidatorForm class, delete FormBean properties, getter, setter methods and use get(-) method to read form data. ~~from form~~.

→ Note: ~~to~~ add struts-home\lib\commons-beanutil-1.7.0.jar file to classpath variable.  
→ refer the validate(-,-) method of appl<sup>n</sup> 4 of page No- 25.

→ Also refer Total RegisterForm.java of page No-24 and 25.

#### Step-3

Configure the above formBean class as shown below in Struts-config file.

##### in struts-config file

```
<form-bean name="rf" type="app.Registerform">
<form-property name="username" type="java.lang.String"
    initial="raya"/>
<form-property name="password" type="java.lang.String"/>
</form-bean>
```

#### Step-4

→ Read FormBean data in Action class execute(-,-,-) method as shown below.

#### In RegisterAction.java

```
Registerform rf = (Registerform) form;  
String user = rf.get("username");  
String pass = rf.get("password");
```

#### Step-5

→ Run the appM.

#### Note

→ We can see the above steps based implementation  
→ Add "common-beanutil.jar" to classpath variable.

30.08.13

For example appM on validator plugin ~~and~~ validators-rule declarative dynamic FormBean and for Database connectivity refer appM4 page- 29 to 33

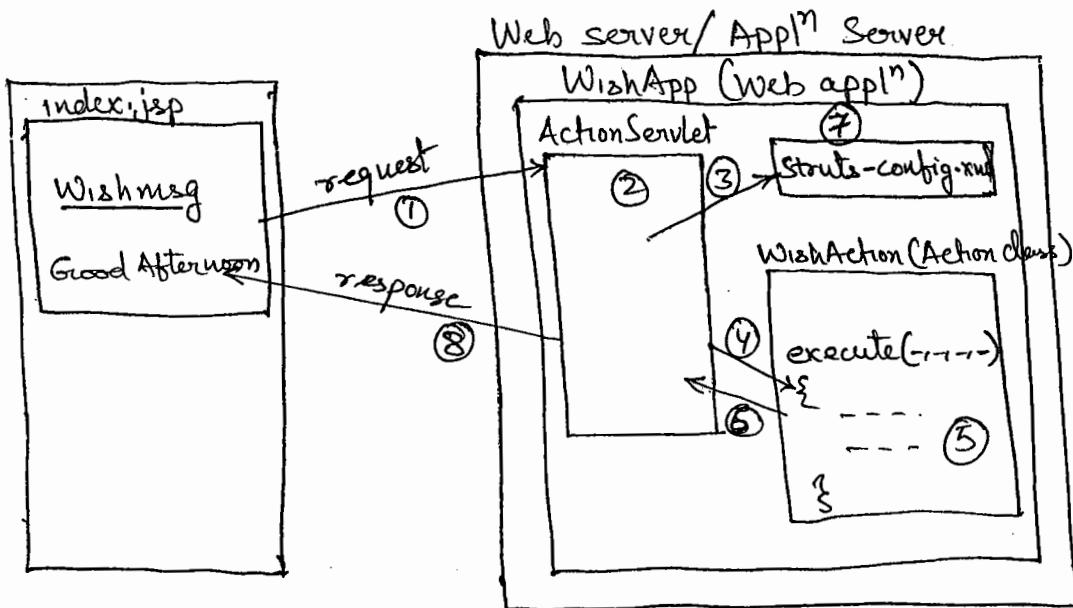
→ If given date(String) values there in yyyy-MM-dd pattern, then it can be converted directly to java.sql.Date class obj by using valueOf(-) java.sql.Date class.

```
String d1 = "2012-10-20"; // yyyy-MM-dd  
java.sql.Date &d1 = java.sql.Date.valueOf(d1);
```

Q5: Develop struts appM without any kind of FormBean by taking request from hyperlink.

31.08.13

## Struts app<sup>n</sup> without FormBean Support :-



→ Hyperlink by using struts supplied jsp tags:

<html:link action = "reg.do" > Go </html:link >

↙ (sends req to action class whose action path is /reg)

<html:link href = "reg.do" > Go </html:link > (same as above)

Note :- In the above app<sup>n</sup> there is no form page sending request so working with FormBean is not required.

### Eclipse

Type : IDE for the Java Env...

Version : Eclipse Kepler

Open source SW

Vendor : Eclipse org

Does not give built-in plug-ins... allows to add maven plugin externally.

To download : [www.eclipse.org](http://www.eclipse.org)

eclipse-jee-kepler-R-win32.zip

⇒ Procedure to develop the above diagram based app<sup>n</sup> by using eclipse kepler IDE.

#### Step-1

→ Launch eclipse IDE by choosing its workspace folder.

The folder where project will be saved.

#### Step-2

→ Create new web project in IDE.

File menu → New → Other → Web → Dynamic Web project → Project name = WishApp → next → next →  Generate web.xml deployment directory → finish.

#### Step-3

Configure ActionServlet in web.xml file.

#### web.xml

//same as 1st app<sup>n</sup>.

#### Step-4

→ Add struts related jar files ~~from~~ to lib folder of the project (available in web content)

Right click on lib folder → build path → configure build path  
→ add external jars → Browse and select the 1st app<sup>n</sup> related jar files.

#### Step-5

→ Add index.jsp to web content folder.

Right click on web content → new → jsp file → file name = index.jsp → next →  use JSP Template → finish.

### index.jsp

```
<%@ taglib uri = "http://struts.apache.org/tags-html" prefix =  
    "html"%>  
  
<html:link action = "wpath"> Wish msg </html:link> <br> <br>  
<html:link href = "wpath.do" > WishMsg </html:link> <br> <br>
```

### Step - 6

→ Add Action class to java resources folder.

Right click on java resources → new → class → name = WishAction  
superclass org.apache.struts.action.Action → finish .

Write the following code in the execute(-----) method  
of Action class.

~~post~~

```
public class WishAction extends Action  
{  
    public ActionForward execute(ActionMapping mapping,  
        ActionForm form, HttpServletRequest req, HttpServletResponse res)  
        throws Exception.  
  
    {  
        // Generate WishMsg  
  
        Calendar cl = Calendar.getInstance();  
        int h = cl.get(Calendar.HOUR_OF_DAY);  
  
        if (h <= 12)  
            request.setAttribute("msg", "Good morning");  
        else if (h <= 16)  
            request.setAttribute("msg", "Good Afternoon");  
        else  
            request.setAttribute("msg", "Good Evening");  
        return mapping.findForward("success");  
    }  
}
```

{  
}

### Step-7

Add struts.cfg file to WEB-INF folder of web content folder

Right click on ~~the~~ WEB-INF Folder → new → other → XML  
→ XML file → file name = Struts-~~the~~ cfg.xml → next →  
next →  use XML Template → finish

### struts-cfg.xml

```
<!DOCTYPE ---->
<struts-config>
  <action-mappings>
    <action path="/wpath" type="WishAction">
      <forward name="success" path="/index.jsp"/>
    </action>
  </action-mappings>
</struts-config>
```

### Step-8

Add servlet-api.jar file to build path of the project

Right click on project → build path → config build path →  
add external jars → browse and select servlet-api.jar

### Step-9

Add following entries in index.jsp

<br><br> In index.jsp

~~Result is:~~ <bean: taglib ----- "bean" %>  
                  <taglib----- "logic" %>

```
<br> <br>
<logic:unless name="msg">
    Result is: <bean:write name="msg"/>
</logic:unless>
```

### Step - 10

Configure Tomcat Server with eclipse IDE

window menu → preferences → Servers → runtime env... → add  
select apache 6.0 → Next → Tomcat installation directory  
D:\tomcat6.0 Browse and select → finish → ok.

### Step - 11

→ Deploy the project in Tomcat server

Right click on project → run as → run on server →  
choose Tomcat6.0 → finish.

→ Add struts related (10) jar files to webContent → WEB-INF → lib folder

### Step - 12

Test the app

open browser window type this url.

http://localhost:2013/WishApp/index.jsp.

### Builtin Action Classes :-

ForwardAction

IncludeAction

LocaleAction

DispatchAction

LookupDispatchAction

MappingDispatchAction

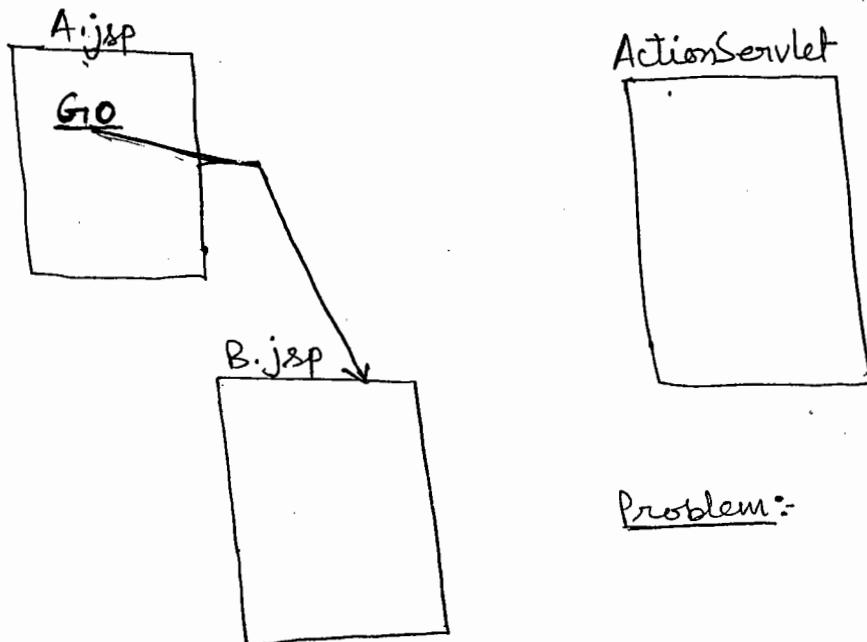
SwitchAction

DownloadAction etc...

Available in org.apache.struts.actions package of  
<struts1.x-home>\lib\struts-extras-1.3.8.jar.  
Note: All the classes are sub classes of org.apache.struts.action.Action class.

## Underus

→ Understanding the need of ForwardAction class.



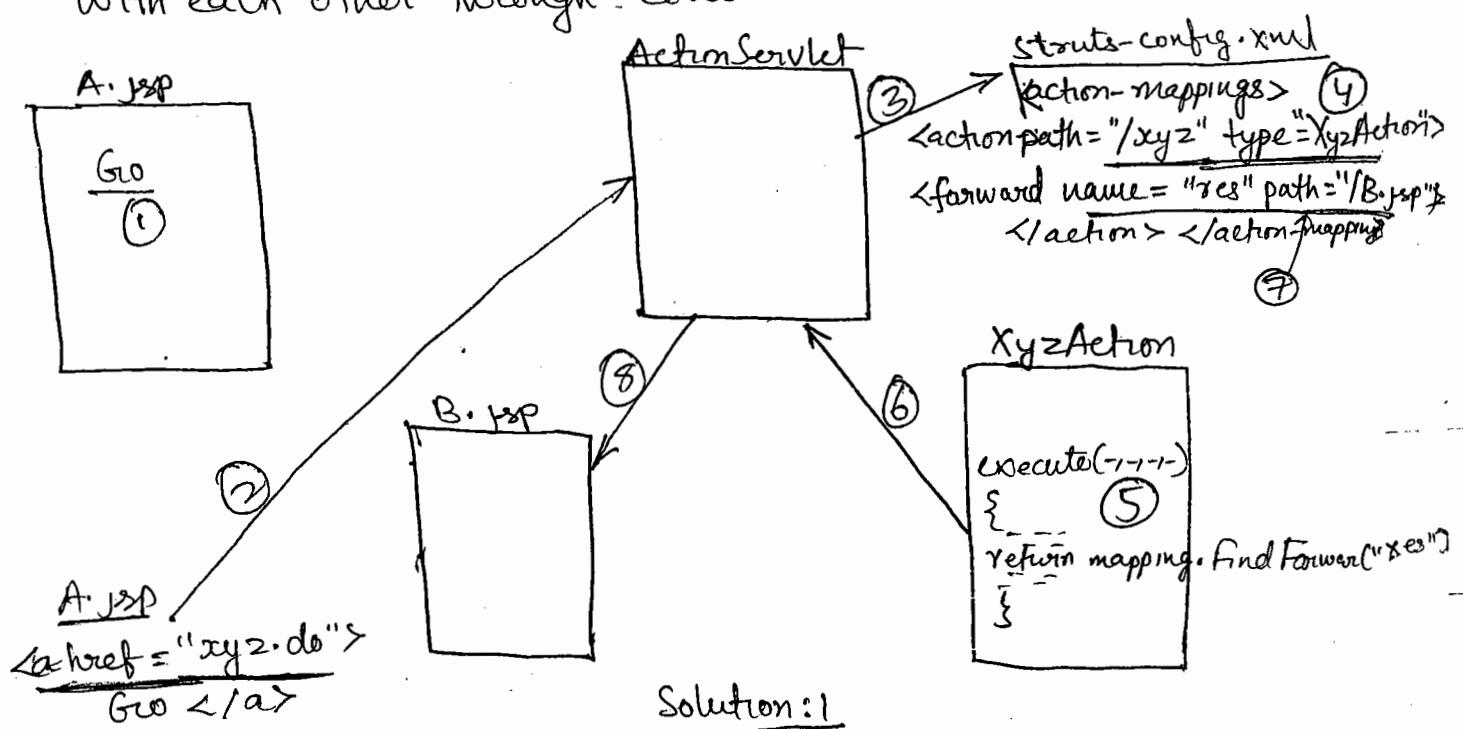
Problem:-

Here A.jsp prog. taking with B.jsp prog directly without interacting within controller ActionServlet. So we can say MVC2 rule are violated.

A.jsp

<a href = "B.jsp" > Go </a>

MVC2 rule is the JSP page of web app<sup>n</sup> must interact with each other through controller servlet.



Solution:1

In the above solution-1 A.jsp is interacting with B.jsp through the controller ActionServlet. but taking the support of one Additional dummy Action class ~~for that~~ (~~XyzAction~~) for that navigation which is not at all good practice.

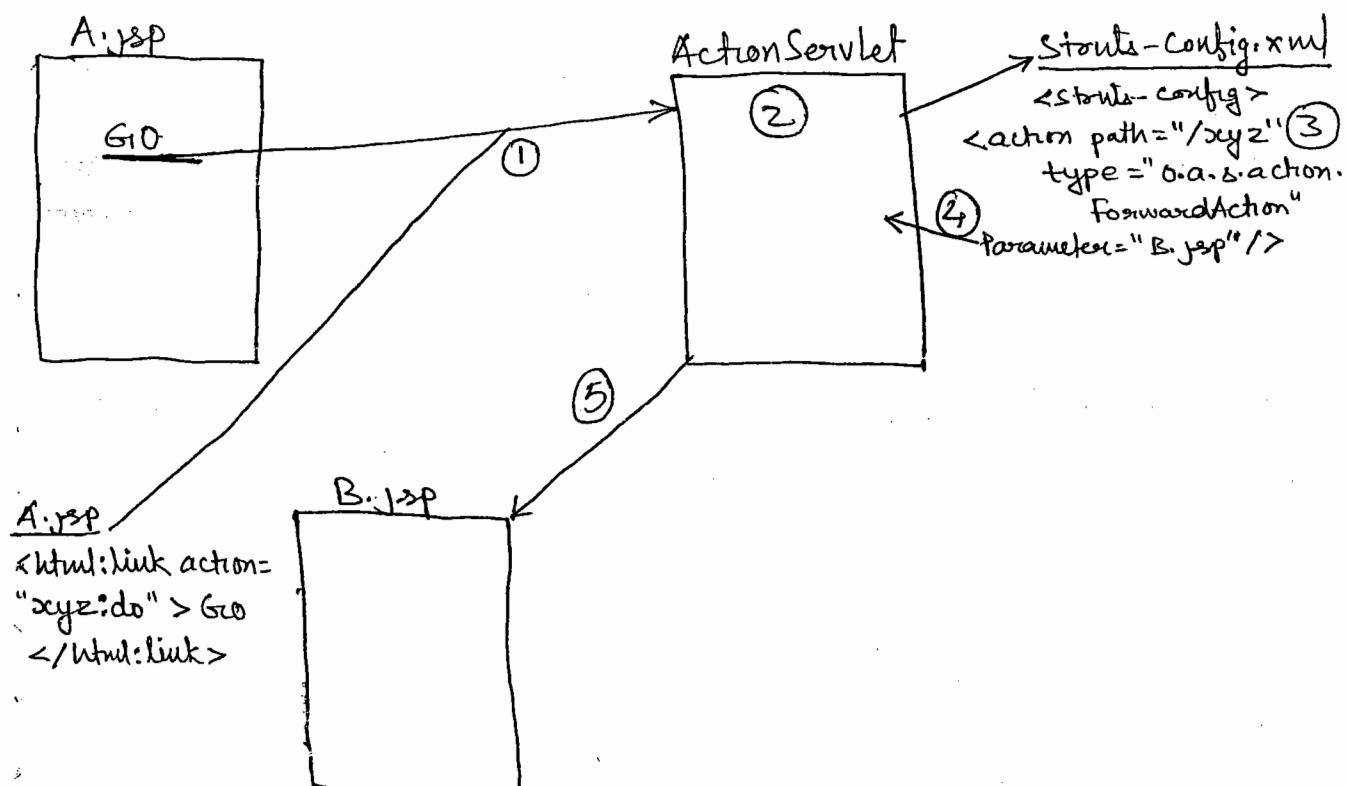
### Note

To overcome the limitation of solution-1 (working with dummy action class) use the build-in Action class called ForwardAction.

### Solution-2

02.09.13 ~~02.09.12~~

Using ForwardAction class (Built-in Action class)



w.r. to above diagram.

- ActionServlet traps and takes the request given by A.jsp
- ActionServlet passes the request to forward Action class that is configure in Struts-config file.

④ forwardAction class returns ActionForward object back to ActionServlet having B.jsp.

⑤ ActionServlet passes the control to B.jsp.

### Example App<sup>11</sup> on ForwardAction class utilization

#### Step-1

→ StrutsApp5 app<sup>11</sup> ready.

Refer page No-29 to 33 of booklet. (refer Aug 30 discussion)

#### Step-2

Add struts-extras-1.3.8.jar file to WEB-INF\lib folder  
↑ All built-in classes are available here.

#### Step-3

→ Configure forwardAction built-in class in struts-config file pointing to Register.jsp

#### In struts.cfg.xml

<action path="/xyz" type="org.apache.struts.actions.ForwardAction" parameter="/Register.jsp"/>  
    ↑ supplies inputs to Action class.

#### Step-4

Prepare hyperlink in failure.jsp as shown below.

#### In failure.jsp

<%@ taglib uri="----" %>  
<center> ~~<~~ <H2>

The following errors were encountered

<b></b> <html:errors/>

<br><br>

<html:link action="xyz" > try Again </html:link>  
</he> </center>

Q: What is difference between ActionForward and ForwardAction?

Ans: ActionForward object is useful to locate the result page of Action class.

Where as ~~ForwardClass is used~~ forwardAction class is useful to form navigator between two resources of Struts appM through controller servlet, Action servlet by ~~satisfy~~ satisfying MVC2 rule.

→ We use ActionForwards to cfg result pages of struts Action class.

There are two types of ActionForwards:-

1. Local Forwards :-

→ specific to each Action class

2. Global Forwards

→ Common for multiple Action classes of struts App.

In struts-config file

<Struts-config>

<form-beans>

- - - - -

</form-beans> <global-forwards>

~~<actions-mappings>~~

<forward name="res1" path="result.jsp" /> // global forward

</global-forwards>

cfg so can be used in XyzAction  
'XyzAction2 classes.

<action-mappings>

<action path="xyz1" type="XyzAction1" >

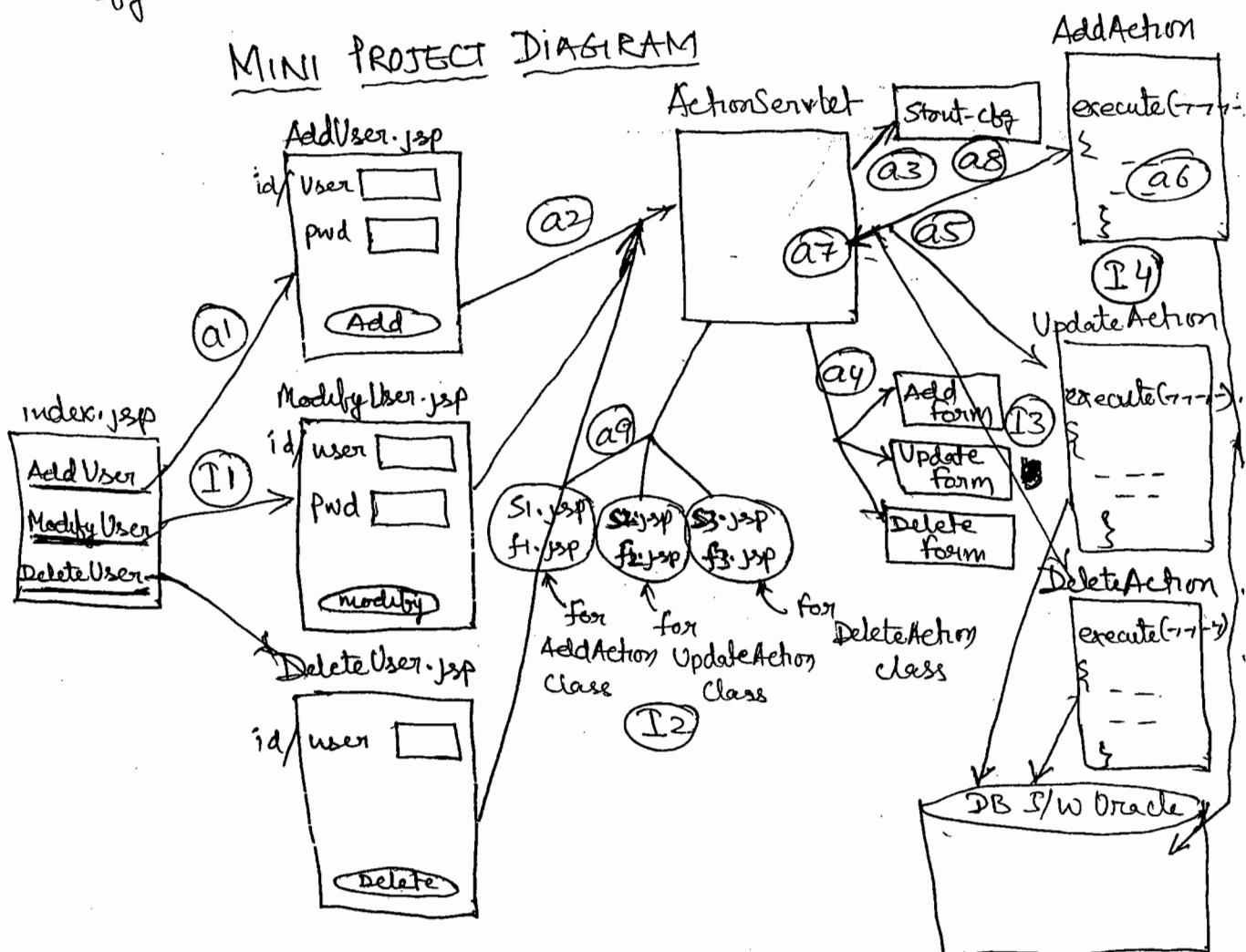
```

<forward name="res2" path="/result2.jsp" /> // local
</action> Forward cfg specific to xyzAction1 class
<action path="/xyz2" type="XyzAction2">
    <forward name="res3" path="/result3.jsp" />
</action> //local forward cfg specific to xyzAction2
</action-mappings>
</struts-config>

```

Q: If we cfg two different jsp as local and global forward cfg respectively having same logical name can you tell me what will happen?

Aus: The result page that is cfg through local forward cfg will be executed.



We can develop Struts app<sup>n</sup> by having multiple form pages, Action class, FormBean and multiple form pages as shown above.

### Note

The above app<sup>n</sup> will work with given setup but there is a possibility of improvising the Struts app<sup>n</sup> in four areas.

- (1) → index.jsp is talking with other jsp programs directly by violating MVC2 rules. For this use forwardAction to resolve the problem.
- (2) → Separate local Action forward based result pages are configured for every Action class. It is recommended to configure common result pages for multiple Action class through global-forwards.
- (3) → Three form pages are dealing with same form data so we can use single FormBean class instead of multiple FormBean classes.
- (4) → ~~There are~~ There are three related individual Action classes we can combine them into single Action class by taking the Action class as DispatchAction class.

### The DispatchAction based Action class :-

→ Allows us to combine multiple Action classes into single Action class by giving the possibility of multiple user defined method as business method.

Ex:- public class OurAction extends org.apache.struts.actions.DispatchAction

```
{    public ActionForward method1(...,...,...)
```

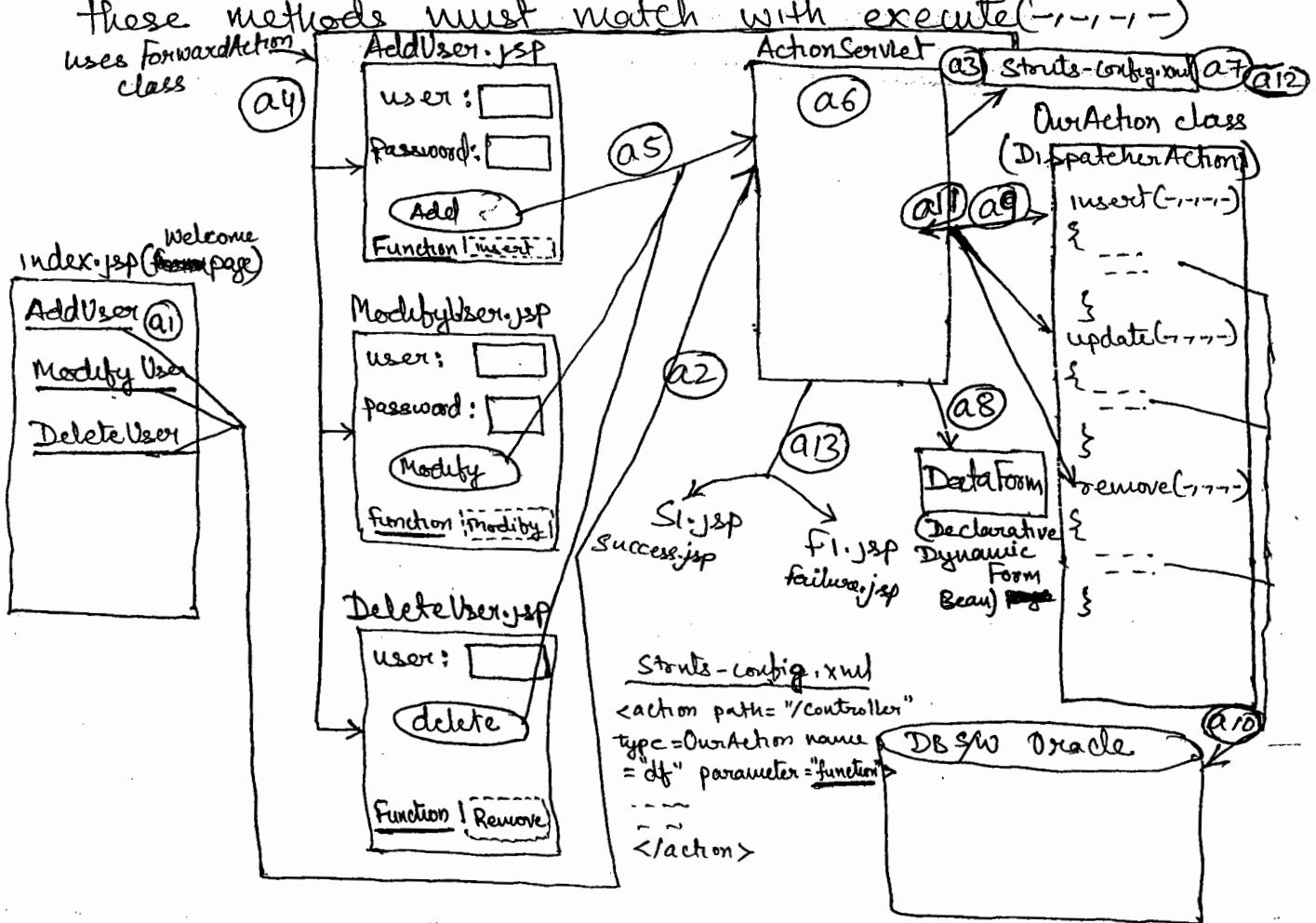
```

{
    --- // B. logic1
}
public ActionForward method2(---,---)
{
    --- // B. logic2
}
public ActionForward method3(---,---)
{
    --- // Business method logic3
}
}

//class

```

→ Here method1, method2, method3 are user-defined methods, But the signature return type modifiers of these methods must match with execute(---,---)



## Observations with respect to above diagram:

- ① index.jsp is talking with other JSP programs through controller servlet by taking the support of `ForwardAction` class.
- ② For three form pages one `formBean` is taken as declarative dynamic `formBean`.
- ③ Global Forward based result pages are taken instead of local Forward based result pages.
- ④ Single Action classes is taken having multiple user-defined methods to process the request of multiple form pages.
  - Each request coming to dispatch action class must carry on additional request parameter value specifying the user-defined method of `DispatchAction` class that has to be executed to process the request.
  - While configuring `DispatchAction` class in our struts configuration file we must specify `parameter` attribute having addition request parameter name that can carry above said method name.
  - When multiple form pages are generating request to Action class of type `DispatchAction` to execute multiple methods of that Action class, <sup>then</sup> each for page must send one method name of dispatch action class as additional request parameter values. The name of additional request parameter will be specify by programmer as parameter attribute value in `struts-config` file while configuring that `DispatchAction` class.

## The six type of FormBeans:

05.09.13

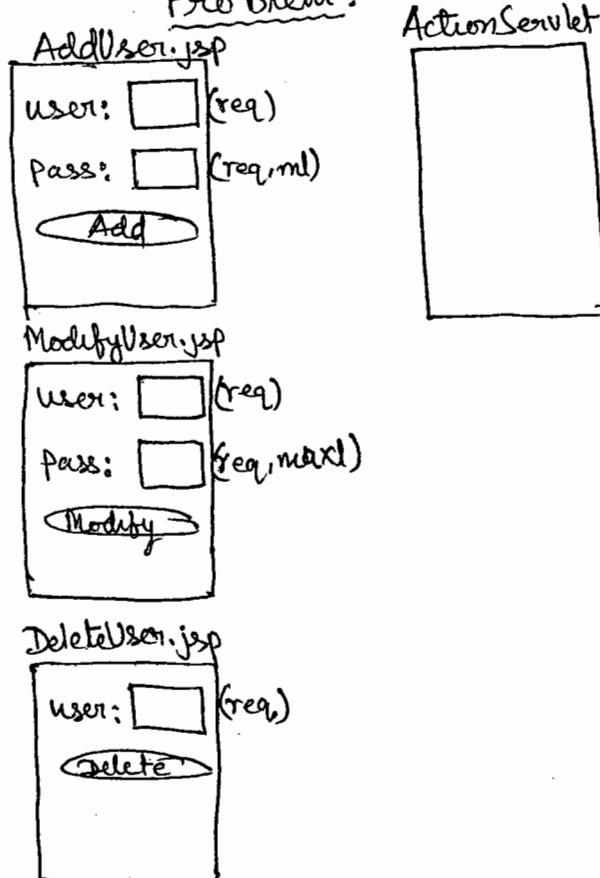
- ① ActionForm, DynaActionForm  $\Rightarrow$  Supports only programmatic form validations (no support for ValidatorPlugin form validations)
- ② ValidatorForm, DynaValidatorForm  $\Rightarrow$  Supports both programmatic and Declarative Form validation (Allows to cfg validator rules Based on Formbean logical name)
- ③ ValidatorActionForm, DynaValidatorAction  $\Rightarrow$  Supports both programmatic and Declarative form validation (Allows to cfg Validator rule based on action path of struts action class).



Note :-

DynaXXXForm classes are given to generate form beans dynamically.

Problem :-



DynaValidatorForm  
(based FormBean)

OurAction (Dispatch Action)  
public AF add(----)  
{  
---  
---  
}  
Public AF update(----)  
{  
---  
---  
}  
Public AF delete(----)  
{  
---  
---  
}

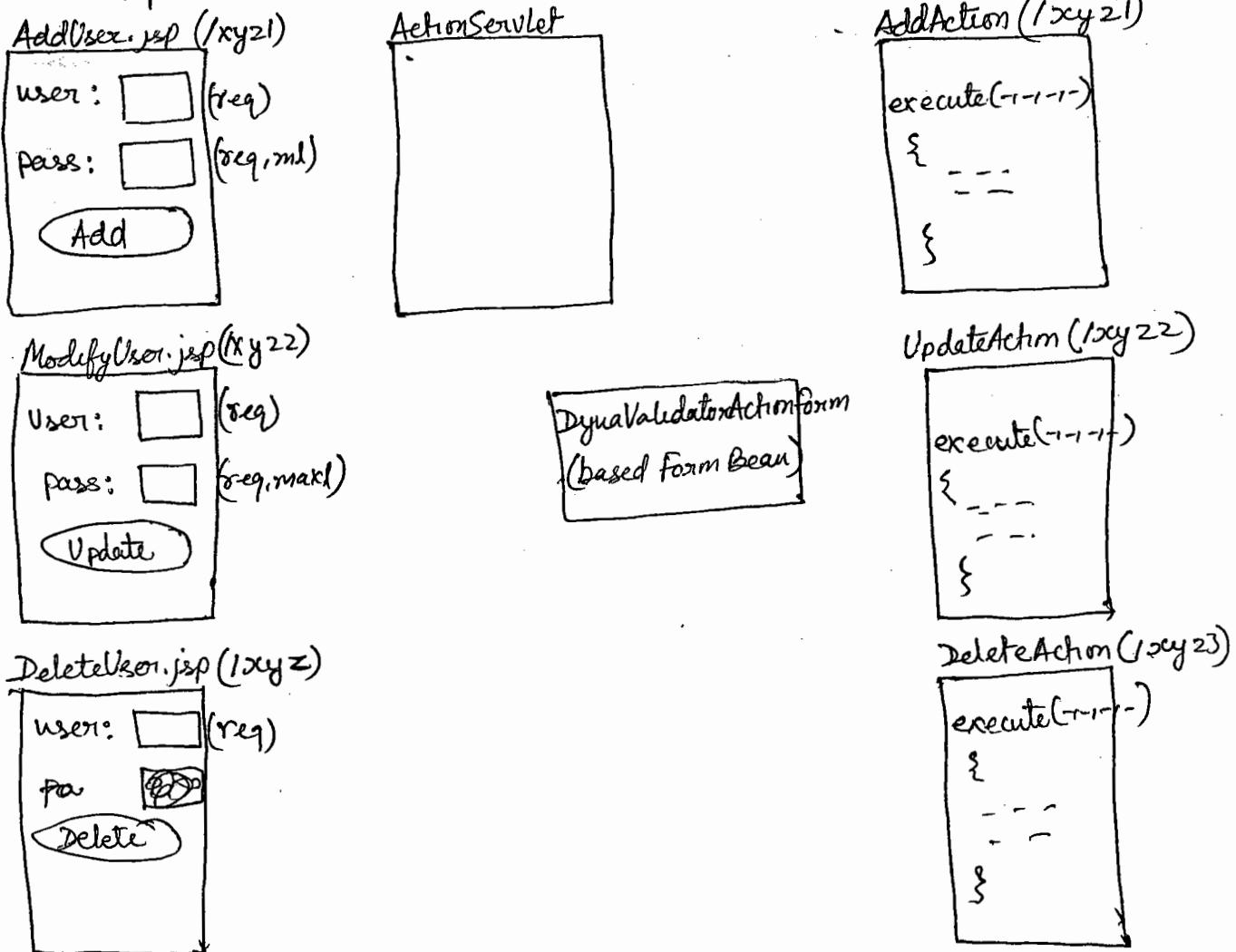
→ Understanding the problem of above diagram:-

ValidatorForm, dynaValidatorForm allows to configure validator rule in validation.xml based on formBean logical name.

In the above diagram multiple form pages are using single FormBean of type dynaValidatorForm due to this we cannot write separate Validator rules for each form page. because there is one FormBean having one logical name and they allow only one set of validator rules on FormBean properties

Solution-1 :-

Using ValidatorActionForm or dynaValidatorActionForm with multiple action class support.



### Step-1

Take multiple form pages pointing to single formBean and multiple Action classes on one per form page basis.

### Step-2

Take formBean type as validatorActionForm or DynaValidatorActionForm which allows to configure validator rule based on action path of action classes.

### Step-3

Configure validator rules in validation.xml separately for each form page based on Action path of action class used by each form page.

### Sample code of Solution-1

#### AddUser.jsp

```
<html:form action="xyz">  
--  
</html:form>
```

#### ModifyUser.jsp

```
<html:form action="xyz2">  
--  
</html:form>
```

#### DeleteUser.jsp

```
<html:form action="xyz3">  
--  
</html:form>
```

## Struts - config.xml

```
<struts-config>
  <form-beans>
    <form-bean name="bean" type="org.apache.struts.validator.ValidatorActionForm">
      <form-property name="id" type="java.lang.String"/>
      <form-property name="pass" type="java.lang.String"/>
    </form-bean>
  </form-beans>
  <action-mappings>
    <action path="/xyz1" type="AddAction" name="bean">
      -- -
    </action>
    <action path="/xyz2" type="UpdateAction" name="bean">
      -- -
    </action>
    <action path="/xyz3" type="DeleteAction" name="bean">
      -- -
    </action>
  </action-mappings>
</struts-config>
```

## Validator.xml

```
<form-validation>
  <formset> action path used by AddUser.jsp
    <form name="xyz1"> AddAction
      <field name="id" depends="required"> validator rule conf for
        <field> AddUser.jsp.
        <field name="pass" depends="required, minlength">
          -- -
        </field>
    </form>
  </formset>
</form-validation>
```

```

</fields>
</forms>
<form name = "xyz22">
<field name = "id" depends = "required" >
    -->
</field>
</form>
<!-- Validator rule conf for ModifyUser.jsp -->
<field name = "id" depends = "required, maxLength" >
    -->
</field>
</form>
<!-- Action path used by DeleteUser.jsp -->
<form name = "xyz33">
<field name = "id" depends = "required" >
    -->
</field>
</form>
</formset>

```

→ The above solution-1 is not suitable if multiple form pages are pointing to single formBean and single Action class.

### Solution-2

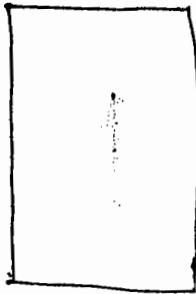
→ Using ValidatorActionForm, DynaValidatorActionForm with the support of single Action class.

AddUser.jsp (xyz1) ActionServlet

user:

pass:

**add**



ModifyUser.jsp (xyz2)

User:

pass:

**Modify**

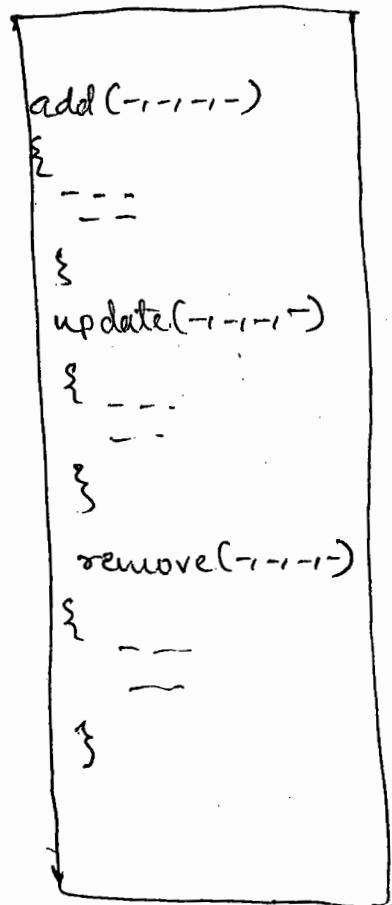
DynaValidatorActionForm  
(based FormBean class)

DeleteUser.jsp (xyz3)

User:

**Delete**

(xyz1, xyz2, xyz3)  
OurAction (DispatchAction)



#### Step-1

→ Take multiple form pages pointing to single FormBean and single Action class.

#### Step-2

→ Take FormBean type as validation Action Form or Dyna Validator Action Form.

#### Step-3

Configure single Action class for multiple times with different action paths

#### Step-4

Make multiple Form pages using multiple action paths of that single Action class on one per form page basis

→ In validation.xml configure separate validator rules for each 'form page' based on 'action path used by each form page.'

### Sample code of Solution-2

form page → Same as solution-1 form pages.

Struts-config.xml →

<struts-config>

<form-beans>

<form-bean name="bean" type="org...."

--- //same as solution-1

--

</form-beans>

<action-mappings>

<action path="/xyz1" type="OurAction" name="bean" parameter="function">

--- // Single struts action class is

--- conf. by having multiple action paths.

</action>

<action path="/xyz2" type="OurAction" name="bean" parameter="function">

---

--

</action>

<action path="/xyz3" type="OurAction" name="bean" parameter="function">

---

</action>

</action-mappings>

</struts-config>

### Validation.xml

// Same as solution-1

### Note :-

We cannot cfg single formBean for multiple times with different logical names but we can cfg single Action class for multiple times with different Action paths.

06.09.13

⇒ For improvised mini project by enabling form validations refer app16 of page No-51 to 58.

⇒ Handling multiple submit button based form page.

### Note :-

When Form page submit button is taken with name the caption of submit button goes to server as req parameter value. Otherwise submit button caption

does not go to server as request parameter value.

### Ex:-

① <html:form action="reg">

- - -  
- - -

<html:submit property="s1" value="login">

</html:form>

→ Sends "s1=login" as ~~req~~ request parameter name and values.

② <html:form action="reg">

- - -  
- - -

<html:submit value="login">

<html:form>

→ does not send "s1=login" as request parameter values. name and values.

⇒ In struts we can handle multiple submit buttons based form page in the following ways:

① Approach-1

Using Dispatch Action and no JavaScript

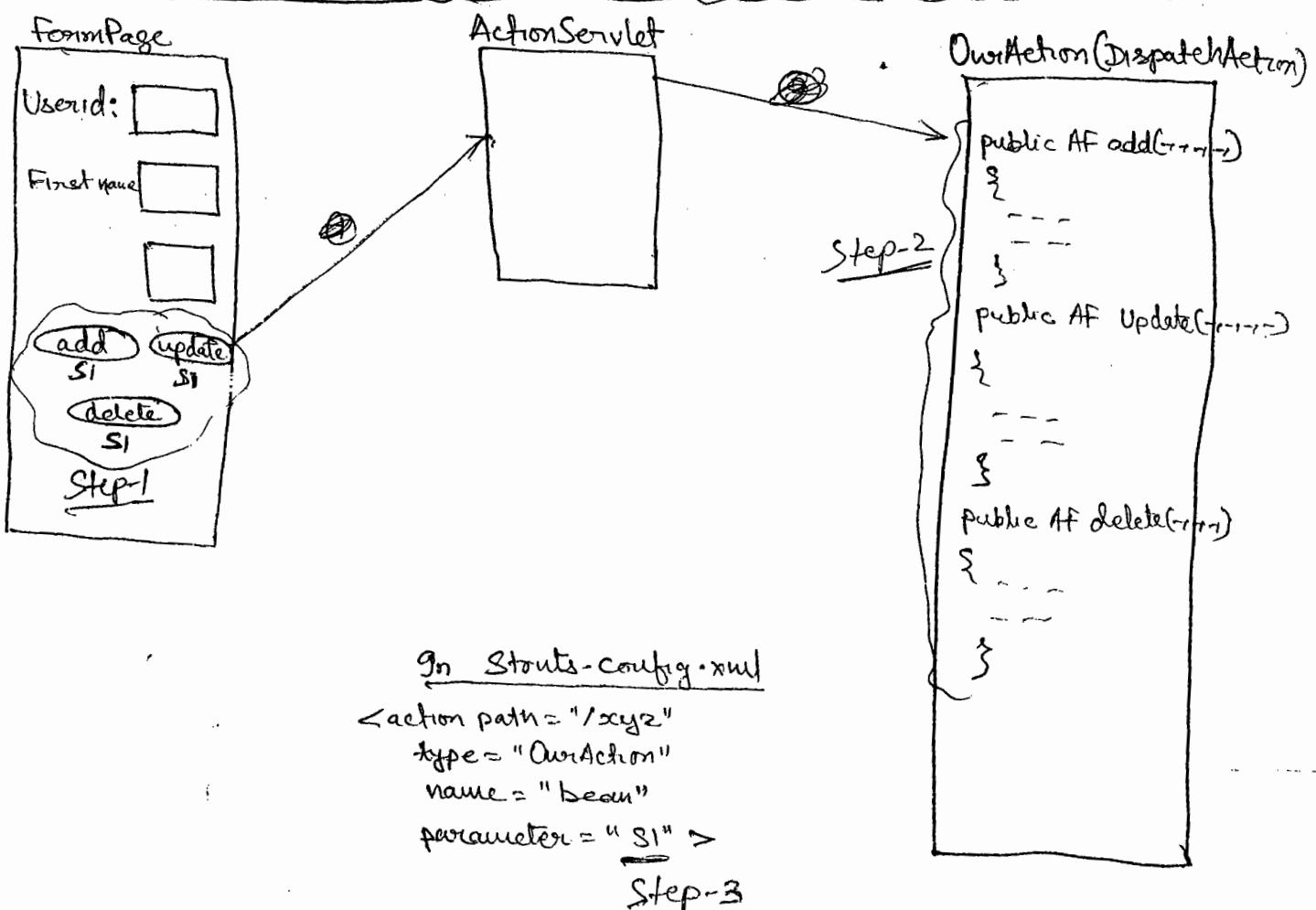
② Approach-2

Using DispatchAction and JavaScript

③ Approach-3

Using LookDispatchAction

① Approach-1 (DispatchAction and no JavaScript)



### Step-1

Take form page having multiple submit button with same name and different captions.

### Step-2

Develop OurDispatchAction class having submit button captions as method names.

### Step-3

Configure ~~our~~OurDispatchAction class having <sup>the</sup> names given to submit button as additional request parameter name.

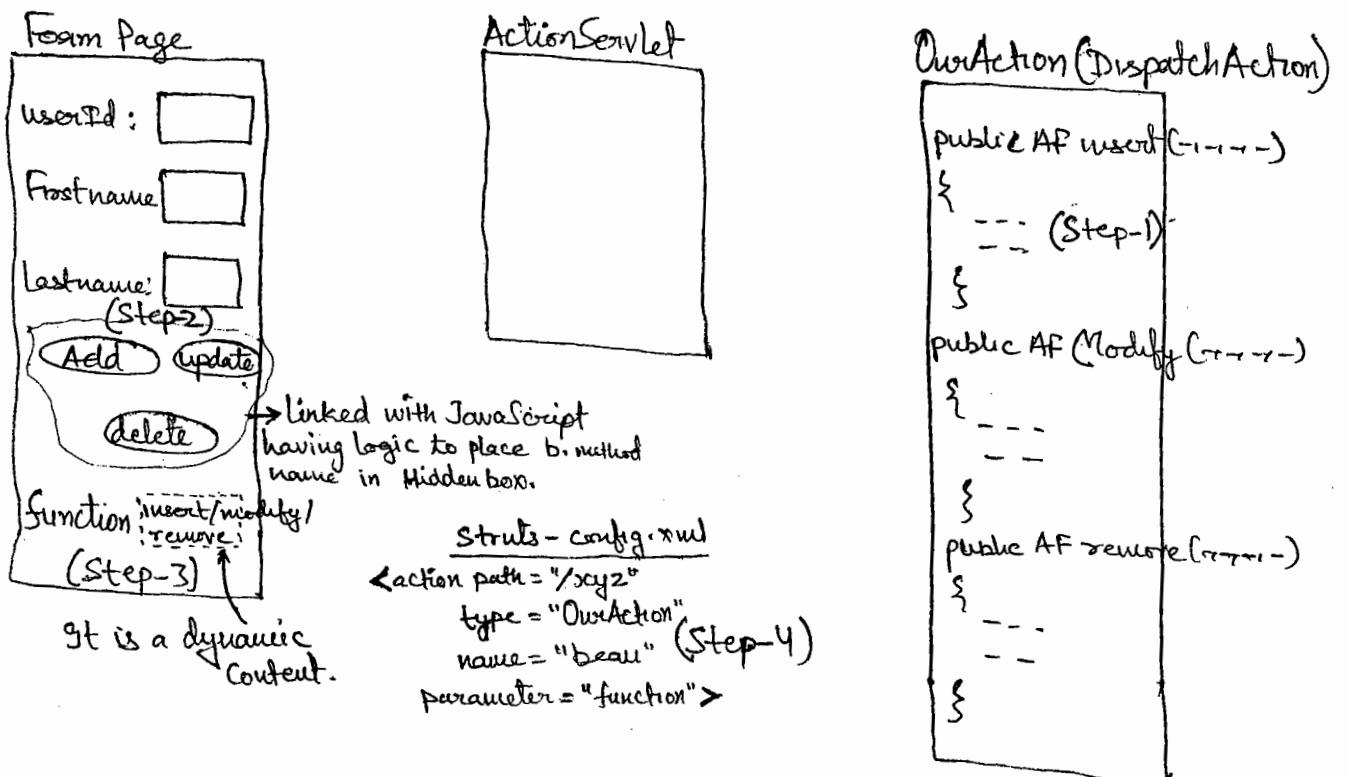
### Flow of execution

Form page add button is clicked → ActionServlet takes request having "s1 = add" as addition request parameter value  
→ ActionServlet calls execute(-,-,-) method on our DispatchAction class object → since not available the superclass execute(-,-,-) method will execute →  
the super class execute(-,-,-) method reads "s1 = add" as addition req parameter values and calls add(-,-,-)  
method of OurDispatchAction class. ~~(OurAction)~~ (OurAction).

### Limitation of Approach-1

→ Business methods name and submit button captions are tightly coupled that means submit buttons are not flexible to modify.

### Approach-2



### Step-1

Develop Our DispatchAction class having programmer choice method names as business method names.

### Step-2

Design Form page having multiple submit buttons and programmer choice submit button captions.

### Step-3

Write JavaScript based logic to place our DispatchAction class method name as Hidden box value based on the submit button that is clicked

### Step-4

Configure dispatch Action class (OurAction) having Hidden box name as additional request parameter ~~new~~ value.

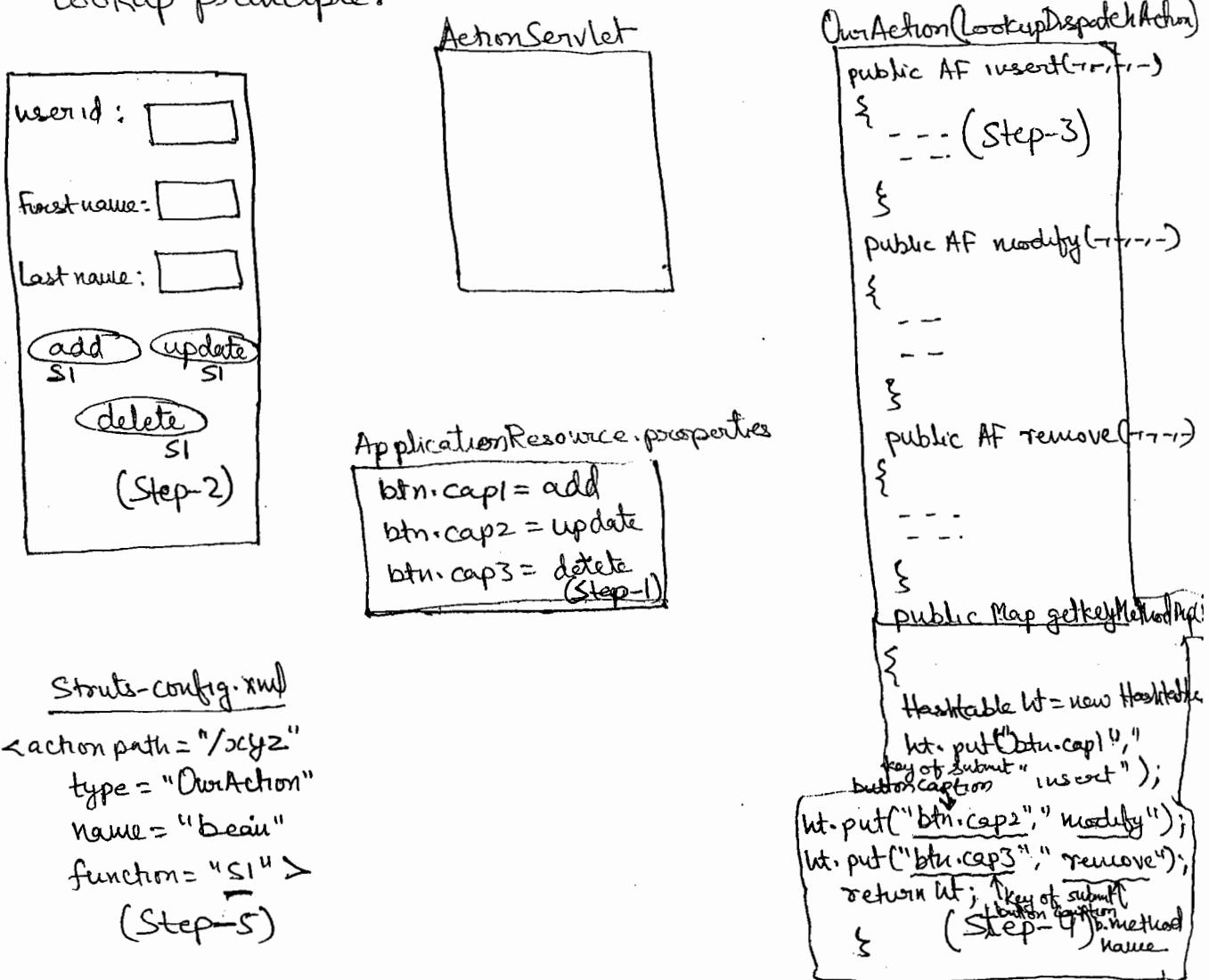
### Limitation

→ Here the submit button caption are not tightly coupled with business method names but this execution fails if JavaScript

disabled through browser settings.

### Approach-3 (using LookupDispatchAction)

- LookupDispatchAction class is sub-class of DispatchAction class having one abstract method getKeyMethodMap(). DispatchAction class is an abstract class containing no abstract method. LookupDispatchAction class is an abstract class containing one abstract method (getKeyMethodMap())
- Getting value from properties file based on key is called Lookup operation, getting key from properties files based on values is called reverse Lookup operation.
- LookupDispatchAction class runs based on reverse lookup principle.



→ Step-1

Prepare properties file having submit button captions.

Step-2

Add multiple submit button to form page having same name and different captions collected from the properties file.

Step-3

Develop our LookupDispatchAction class having user-defined method as business method names.

Step-4

Implement getKeyMethodMap() method having logic to return Map object. This Map object must contain the keys of submit button caption as keys and business method names as values.

Step-5

Configure our LookupDispatchAction having the name given to submit buttons as the addition ~~post~~ request parameter name.

~~Step 6~~ Flow of Execution :-

→ ActionServlet takes "add" submit button request having "SI = add" as additional request parameter value → ActionServlet calls execute(-,-,-) methods on our LookupDispatchAction class object → since execute(-,-,-) method is not available the super. class execute(-,-,-) method will execute → this super class execute(-,-,-) method performs the following operations.

① Calls getKeyMethodMap() method and gets Map object.

- b) Reads additional request parameter \$1 value and gets "add" value.
- c) Gets btn.cap1 from properties file by submitting "add" value. through reverse Lookup operations.
- d) The received btn.cap1 will be used to get insert() method name from Map data structure. of (a) step.
- e) Calls insert() method of our LookupDispatchAction class.

→ for example appln on LookupDispatchAction refer appln7 of page No-67 to 72.

#### Advantages of Approach-3

→ In approach-3 no need of working with javascript and submit button captions are to lightly coupled with business method names.

10.09.13

MyEclipse = Eclipse + Built-in plugins

(To work with advanced Technologies)

Type : IDE S/w Java Env...

Version: 10.x (Compatable with jdk1.5+)

Commercial IDE S/w

Vendor: Eclipse org

To download &/w: [www.myeclipseide.com](http://www.myeclipseide.com)

for help and FAQs = [www.myeclipseide.com](http://www.myeclipseide.com)

Procedure to develop lookup DispatchAction appM  
(appM7) of pages 66 - 67 to 72 of booklet by using  
MyEclipse 10.8 IDE.

### Step-1

Launch myEclipse IDE by choosing work space folder.

### Step-2

Create web project having name LDProj

File menu → new → web project → Project name =  
LDProj → finish.

### Step-3

Add Struts compatible to the project

right click on project → myEclipse → add struts capabilities → select Struts 1.3, base package for new class = PI → finish.

### Step-4

→ Addojdbc14.jar file to the build-path of the project.

Right click on project → Build path → add external archives → Browse and selectojdbc14.jar.

### Step-5

Create form page, formBean, ActionClass

Right click on project → new → other → myEclipse → web struts → Struts 1.3 form, Action & JSP → next → Name = EF, form type = EasyForm, super class = org.apache.struts.action.ActionForm → Form properties tab → add

Name = userid → add , firstname → add , lastname → add ,  
address → add → method tab → deselect all the  
checkboxes → jsp tab → select create jsp form? →  
New jsp path = /user.jsp → next  $\xrightarrow{\text{action class}} \text{path} = /emp$  ,  
superclass = org.... LookupDispatchAction , type = EmpActn  
Name = ef , Attribute = ef → parameter tab →  
parameter = function → method tab → ~~do~~ select  
public ~~do~~ ActionForward execute (Http--) method →  
forward tab → add → Name = success , path = /user.jsp  
→ add → Name = failure , path = /failure.jsp →  
finish .

Step-

→ Add following entries in ApplicationResource.properties.

btn.cap1 = Add  
btn.cap2 = update  
btn.cap3 = delete .

Add multiple submit btn in form page having same  
name and different caption

### Step-8

Develop DB class DBConnector class to helper class to Action class.

Right click on src folder → New → Class → package Con. durga. comanager → Name = DBConnection

#### DBConnection.java

→ refer page no- 71 and 72 of booklet

### Step-9

Complete the code of ActionClass (EmpAction)

Refer EmpAction of page-69

### Step-10

Add failure.jsp to web root folder of the project.

Right click on project → New → JSP → Name = failure.jsp .

refer page No- 72 of the booklet

### Step-11

Read the request attribute value also in user.jsp

### Step-12

Configure Tomcat server with myEclipse IDE.  
window menu → preference → myEclipse → servers →  
Tomcat → configure Tomcat 6.X → enable → browse  
and select <Tomcat\_home> directory (installation  
folder of Tomcat) → apply → ok

### Step-13

Make sure that Employee\_info table is there in  
Oracle db s/w.

~~Create table~~

### Step-14

Run the project

Right click on project → run as → myEclipse Server  
app/m → Tomcat 6.X → ok

### Step-15

Test the app/m

<http://localhost:2020/LDProj/user.jsp>

↗ Procedure to configure glassfish 2.X server with  
myEclipse IDE

### Step-1

window menu → preference → myeclipse → servers →  
glassfish → glassfish 2.X → enable → home directory  
of Glassfish (C:\sun\appServer)

→ Procedure to ~~st~~ configure web logic server with myEclipse IDE  
window menu → preference → my Eclipse → servers →  
web logic → enable → browse and select BEA  
<home\_directory> (c:\oracle\middleware),  
Administrator username = durgasoft, administration  
password = durgasoft, execution domain root =  
c:\oracle\middleware\user-projects\domains\durga →  
apply → ok

→ Procedure to cfg of jBoss 5.1 default domain  
server with myEclipse IDE

window menu → preference → myeclipse → servers  
→ JBoss → JBoss 5.1 → enable → browse and select  
~~hom~~ directory of jBoss.

16.09.13

## FILE UPLOADING AND DOWNLOADING

- The process of selecting file from client machine file system and sending that file to server machine file system is called file uploading and reverse is called file downloading.
- While developing metainfony applications, job portal applications, file/video sharing app<sup>this</sup>. File ~~app~~

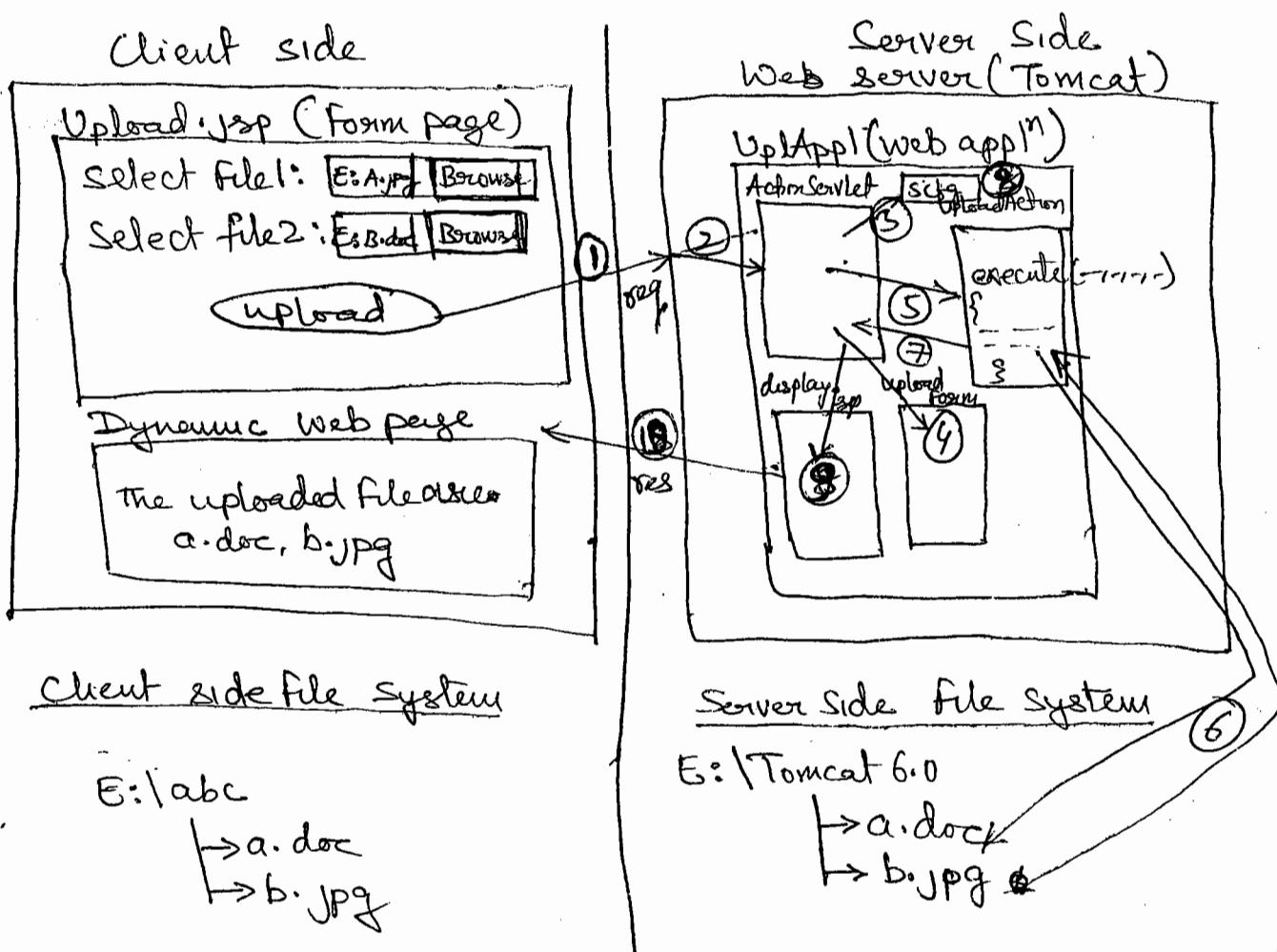
download and uploading ~~are~~ operations are required

→ Struts gives built in supports for file uploading and file downloading.

→ By default the uploaded file will be save to the installation folder of underlying tomcat server.

→ To specify new location we must write initial logic in struts Action class.

→ ~~client side and server side.~~ ~~initial~~



→ Procedure to develop the struts app<sup>m</sup> (performing file uploading)

#### Step-1

Design Form page having File uploading components with the support of ~~HTML~~ <html:file> tag.

#### Step-2

Design Form Bean Class having @ org.apache.struts.upload.FormFile type FormBean properties.

#### Step-3

Write Stream based logic in execute(----) of action class to receive the uploaded files and to complete the file uploading.

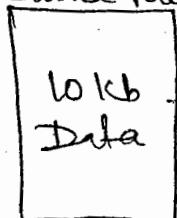
For example app<sup>m</sup> on struts based file uploading refer app<sup>m</sup>/I of page No-88 of booklet.

#### → Buffer/cache

→ It is a temporary memory that holds data for temporary time of period.

While transferring huge amount of data from source file to destination file if the buffer support is taken we can reduce the number of read and write operation on source, destination file respectively.

Source file

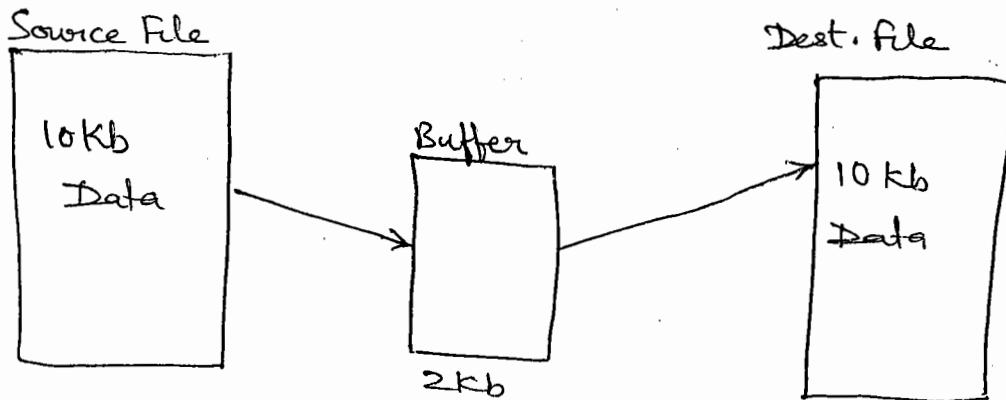


Dest. file



→ We need to perform read operation on source file and write

operations on Dest. file for  $10 \times 1024$  times.



Here we need to perform read operations for 5 times to ~~read~~ source file ~~write~~ and write operation for 5 times on Dest. file respectively.

→ In servlet/JSP env... we must use some third party like Java Zoom API to complete file uploading, but struts gives built-in support for file uploading.

### FILE DOWNLOADING :-

#### Resource Downloading

→ The files placed in web appln like audio file and video file and etc will be downloaded when resource downloading is enable.

→ In struts env... we need to use DownloadAction class support for it

#### Response Downloading

→ In struts env... we need to use ~~Download Action class~~ support for it.

→ Here the output generated by web resource progs like JSP, servlet and action classes will be ~~be~~ downloaded as files

- For this we need to use special response header called "Content-disposition"

- Content-disposition response header gives instruction to browser window to render the receive content as the file to download
- Place the following two lines of code in any web resource program to make its output as downloadable file

```
<% response.addHeader("Content-Disposition", "attachment; filename = Title1.xls"); // downloaded  
response.setContentType("application/ms-excel")%>  
                                ↑ response header  
                                ↑ file default name  
                                ↓ MIME type
```

→ For above steps based appln refer appln 10 of page-85 to 87

→ Working with multiple modules having multiple struts cfg files.

→ By default every struts-app contains one module as default module. Module is a logical partition of web app. While developing large scale web app and if we cfg all Action class and FormBean with one struts-config file then there is a possibility of getting naming clashes. To overcome these problems create multiple modules in struts app, and work with multiple struts-config file to avoid those naming clashes (like formBean logical name and action path etc...)

→ If we create 3 modules in struts app, that struts app contains total 4 modules

\* One default module  
Three named modules.

→ Each modules can have one struts cfg file having that module related formBeans, action classes cfg...

→ We must cfg the struts cfg files of multiple modules in web.xml using config Init Parameter as shown below.

### In web.xml

```

< servlet >
< servlet-name > action </ servlet-name >
< servlet-class > org.apache.struts.action.ActionServlet
</ servlet-class >
```



```

    {<init-param> // Default module struts-config file
      <param-name> config </param-name> configuration
      <param-value>/WEB-INF/struts-config.xml </param-value>
    }</init-param>

    {<init-param> // Model module struts-config file configuration
      <param-name> config/mod1<module name> </param-name>
      <param-value>/WEB-INF/struts-config-student.xml
      </param-value>
    }</init-param>

    {<init-param> // Faculty module struts-config file configuration
      <param-name> config/faculty<module name> </param-name>
      <param-value>/WEB-INF/struts-config/faculty.xml </param-value>
    }</init-param>
  
```

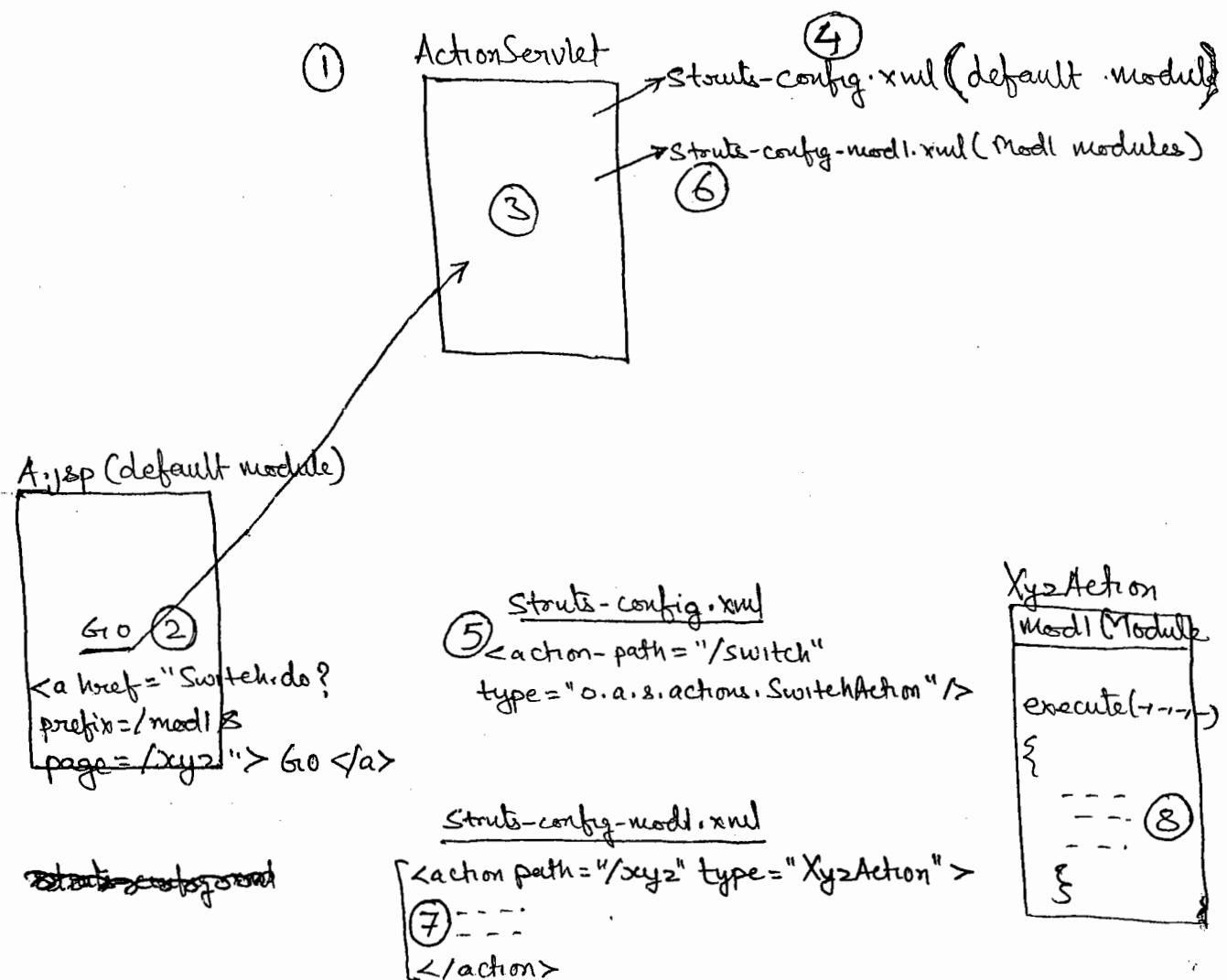
### ~~Note~~ :-

- Module name need not to reflect in struts-config file name.
- Navigation between Modules :-  
(or)  
Communication/interaction between Modules :-
- To make resources of different modules interacting with each other we have to take support of built-in action class called **SwitchAction**
- \* **SwitchAction** (`org.apache.struts.action.SwitchAction`)
- While giving request to this SwitchAction class the request must have two additional request parameters.
  1. **Prefix** : Contains the target resource name.
  2. **Page** : Contains the target resource name or its identity name.

The following Navigation are possible:-

1. Default module to Named Module
2. Named Module to Default Module
3. Named Module to Named Module.

→ Navigation between Default module resource to Named Module Resource :-



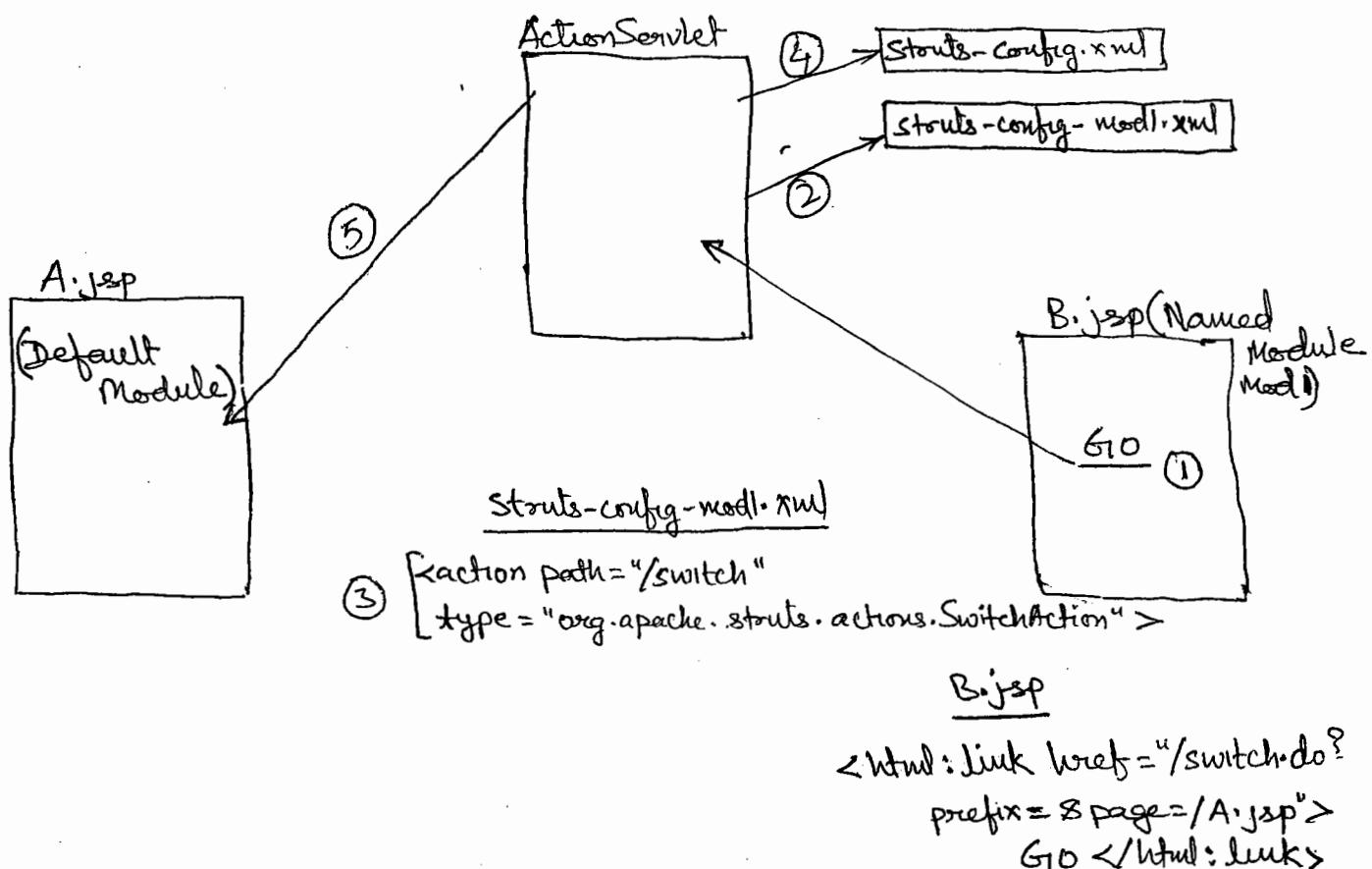
### Note

→ We should always configure SwitchAction class in source module related struts-config file.

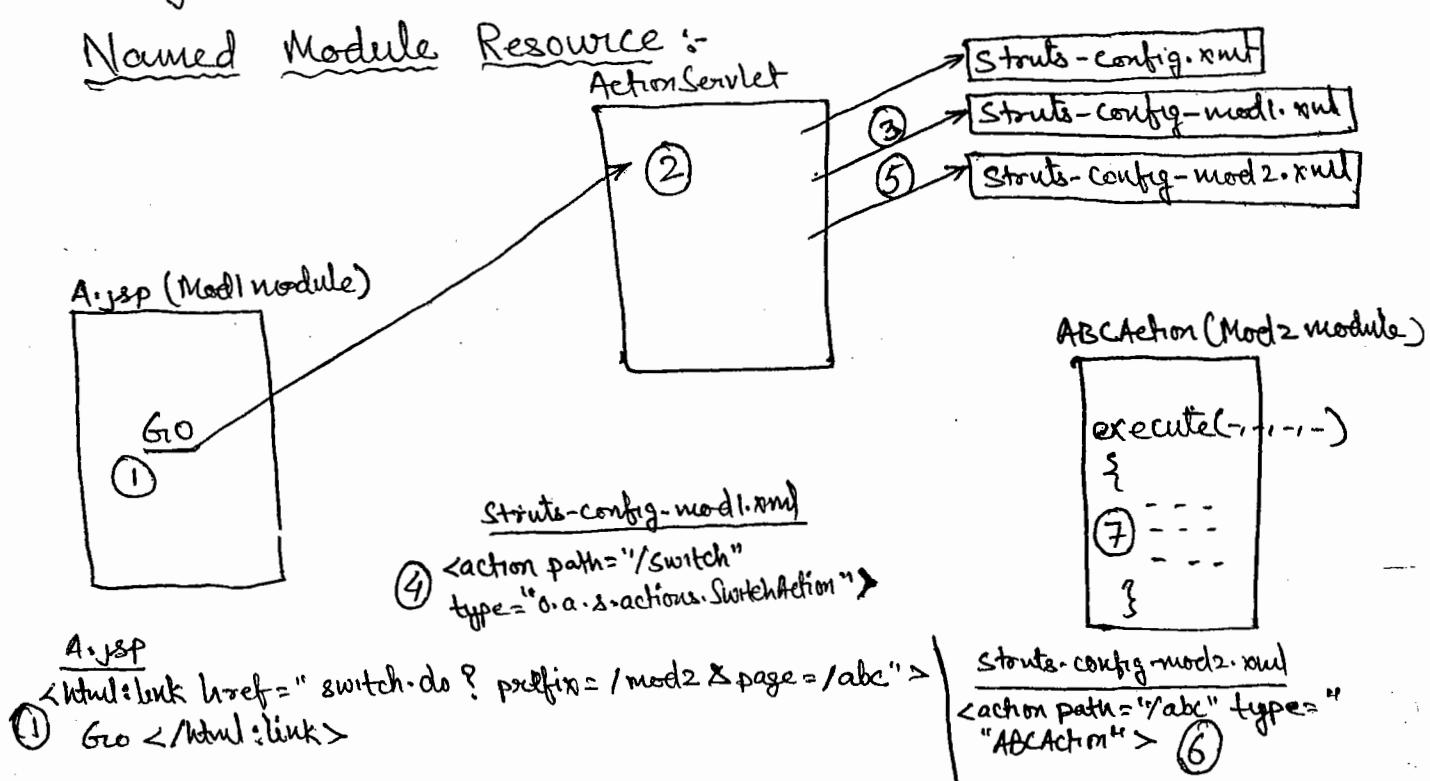
With respect to the diagram:-

- ① Programmer deploys struts-app<sup>TM</sup> in web server (or) Application server. In this process web.xml will be read and ActionServlet become aware of multiple modules and multiple module related struts-config files.
- ② End user generate request from G10 hyperlink of default module A.jsp.
- ③ ActionServlet traps and takes the request passes into default module ~~set~~ "struts-config.xml" file.  
④
- ⑤ Since request wd having "switch.do" and action path of SwitchAction is "/switch" So the SwitchAction class takes the request and reads page, prefix additional request parameter values.
- ⑥ Based on prefix="/mod1" value SwitchAction class makes ActionServlet to use the content of mod1 modules "struts-config-mod1.xml" file.
- ⑦ Based on page="/xyz" request param value ActionServlet locates xyzAction class configuration in "struts-config-mod1.xml" file. object of
- ⑧ ActionServlet creates and locates, XYZAction class and calls execute(-,-,-,-) method on it.

## → Named Module to default Module Navigation :-



## → Navigation between Named Module resource to another



## MINI PROJECT DISCUSSIONS:-

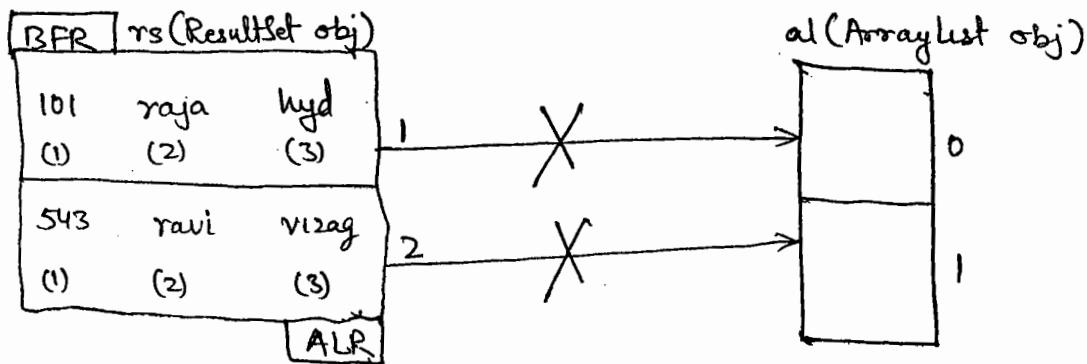
- We cannot send the ResultSet over the network ~~be~~ because it is not serializable object.
- To solve this problem use
  1. Rowsets instead of ResultSets
    - All Rowsets are serializable object
    - Very few jdbc driver supports Rowsets, so working with Rowsets is not industry standard.
  2. Copy the records of ResultSets to Collection framework Data structure and send that Data structure over the network.
- All Collections of java are serializable objects.

### Note:-

- ⇒ If you want to perform only read operations on Collection data structure then prefer using ~~synchronized~~ non-synchronized data structure for better performance  
Ex:- ArrayList, HashMap etc...
- ⇒ If you are looking to perform both read and write operation on data structure then prefer using synchronized data structure. For thread safety.

### Ex:- Vector and Hashtable

- To send Collection data structure over the network the objects added to the elements of Collection data structure must be taken as serializable objects.
- Understanding the problem related to copying ResultSet obj to records to ArrayList Elements.
- Each element of ArrayList allows only one object at a time but each record of ResultSet may contain multiple objects.



So, we cannot copy each record of ResultSet to each element of ArrayList.

→ To solve the above problem copy each record to one user-defined java class obj and add those objects to ArrayList elements as shown below. Here the class of that user-defined object is generally JavaBean and it is technically called as DTO (Data Transfer Object) class.

### Sample Code

#### Student Bean.java

```
public class StudentBean implements java.io.Serializable
{
    private int no;
    private String name, address;
    // write setXXX(-) and getXXX(-) methods.
    ----
}
```

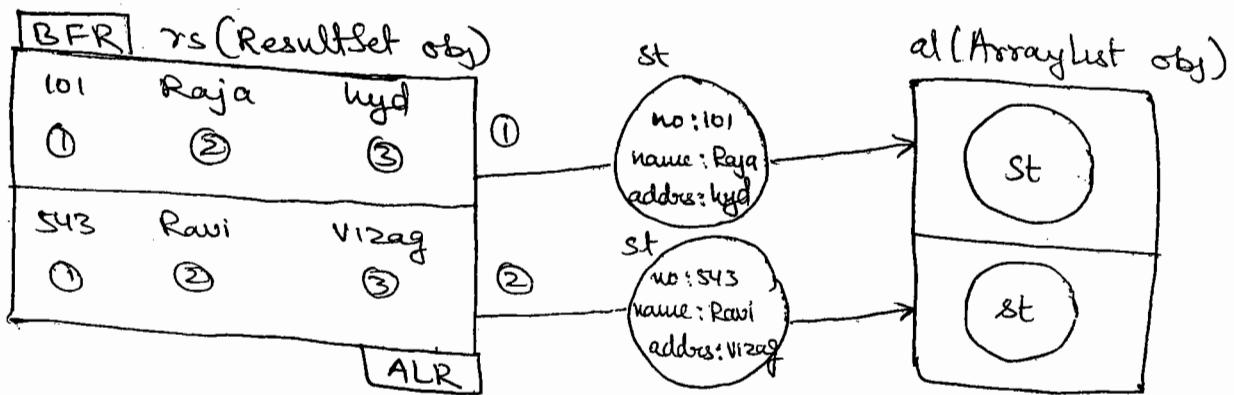
→ Logic to copy ResultSet obj records to ArrayList elements

```
ResultSet rs = st.executeQuery("select * from student");
ArrayList al = new ArrayList();
while (rs.next()) {
}
```

```

StudentBean st = new StudentBean();
st.setNo(rs.getInt(1));
st.setName(rs.getString(2));
st.setAddress(rs.getString(3));
al.add(st);
} // end while

```



→ For multi-module based mini projects that uses ~~switch~~ SwitchAction for Navigation prefer applying of page Nos - 77 to 84.

→ Problem

→ JSP program cannot use the Java class that is placed directly in WEB-INF\classes folder

Solution1

Place Java class along with JSP program (but this class cannot be used in servlet programs, Action classes of web application).

Solution2

Keep Java class in the package of WEB-INF\classes folder.

(This class can be used in all servlet program, JSP

Note: programs, Java classes and Action classes of struts 'appn')

Solution2 is recommended to use in real time envr...

## → Exception Handling in Struts

15.09.13

→ The main intention of Exception Handling is render non-technical guiding msgs to end-user when exception are raised to the web appln. because end-user's cannot understand technical words of exception messages. In struts env... we need to perform exception handling on three resources.

- ① On struts Action classes
- ② On Struts FormBean classes
- ③ On ~~struts~~ JSP programs...

→ Exception Handling on struts Action Classes:-

### 1. Programmatic approach.

→ Using try-catch block.

### 2. Declarative approach

→ Using XML entries of struts cfg file

#### a) Local Exception Handling

→ Specific to each Action class

#### b) Global exception Handling

→ Common to multiple action class.

- ① Programmatic approach or of exception handling or on Struts Action class.

#### TestAction.java

public class TestAction extends Action

```

    {
        public ActionForward execute(----)
    {
        try
        {
            -- // if exception rise in B. logic. control goes
            -- to myError.jsp.
        }
        catch (Exception e)
        {
            RequestDispatcher rd = request.getRequestDispatcher
                ("myError.jsp");
            rd.forward (request, response);
        } // catch
    } // class

```

### MyError.jsp

<font color="red" size=3> Internal problem </font>

② Declarative approach. of Exception handling on struts Action class.

#### Example

##### Step-1

→ keep action classes execute(----) methods ready with "throws" statement.

##### Step-2

→ Configure declarative exception handling for Action classes by using <exception> tag as shown below.

##### in struts cfg file

<Struts-config>

```

<form-beans>
-----
</form-beans>
<global-exceptions> Handles all exceptions of all
<exception type="java.lang.Exception" path="/error.jsp" key="my.global.msg" /> Action classes
</global-exception>
<action-mappings>
<action path="/xyz1" type="XyzAction1" >
<exception type="java.lang.Exception" path="/error1.jsp" />
    Handles all key="my.local.msg" exceptions of XyzAction1 class
    <forward name="success" path="success.jsp"/>
</action>
</action-mappings>
<message-resource parameter="App" />
</struts-config>
<action path="/xyz2" type="XyzAction2" >
<exception type="java.lang.Exception" path="/error2.jsp" />
    Handles all key="my.local2.msg" exceptions of XyzAction2 class
    <forward name="success" path="success.jsp"/>
</action>
</action-mappings>
<message-resources parameter="App" />
</struts-config>

```

### Step-3

Write msgs in properties file.

#### App.properties

my.local1.msg = Problem in XYZAction1 class

my.local2.msg = Problem in XYZAction2 class

my.global.msg = Problem in Struts app!.

#### Note

If we cfg both global and local exception handling cfg on one action class then the configurations done local exception handling will take place.

### Step-4

Implement error.jsp as shown below

① error1.jsp    ② error2.jsp    ③ error3.jsp

<html:errors/>    <html:errors/>    <html:errors/>

→ When declarative exception handling is cfg then control goes to JSP program at path attributes and displays the error msg that is collected from properties file based on the keys that is specify when exception is used.

16.09.13

## 1 Exception Handling in formBean class:

- It allows <sup>only</sup> programmatic form validation
- ~~display form validation error msg on input page.~~
- Displays Exception related error msg in input page

### Step-1

keep any struts appl<sup>n</sup> ready.

### Step-2

Place logic in FormBean class validate(-,-) as shown below.

```
Step public class RegisterForm extends ValidatorForm
{
    -- //FormBean properties declaration
    -- and setXXX(-), getXXX(-) methods
}

public ActionErrors validate(ActionForm form)
{
    try
    {
        ActionErrors error = new ActionErrors();
        -- //form validation logic
        --
        return error;
    }
    catch (Exception e)
}
```

```
ActionErrors error2 = new ActionErrors();
error2.add("expmsg", new ActionMessage("my.exp.fb.msg"));
return error2;
}

} // Validate(-,-)

} // class.
```

### Step-3

Keeping msg in properties file.

Myfile.properties

my.exp.fb.msg = problem in form bean class

### Step-4

→ add <html:errors> tag in input page  
In input page (register.jsp)

<html:errors property="expmsg" />

## Exception Handling in JSP program :-

### Step-1

1. Local Exception Handling / Error page Cfg.

→ Specific to each jsp program

2. Global Exception Handling / Error page Cfg.

→ Common for multiple jsp programs of web application.

### ① Local Exception Handling / Error Page Cfg. :-

Main.jsp

<%@ page errorPage="err.jsp" %>

```
--- // Code that may raise exception  
---  
---
```

### err.jsp (Error page)

```
<%@page isErrorPage = "true" %>  
<b><i> Some internal problem </b> </i>
```

### Global Exception Handling / Error page Cfg :-

#### Main.jsp

```
--- // Code that may raise exception  
---
```

#### Main.jsp

```
--- // Code that may raise exception  
---
```

#### err.jsp

```
<b> Some internal problem </b>
```

#### web.xml

```
<Web-app>
```

```
  <error-page>
```

```
    <exception-type>java.lang.Exception</exception-type>
```

```
    <location>/err.jsp</location>
```

```
  </error-page>
```

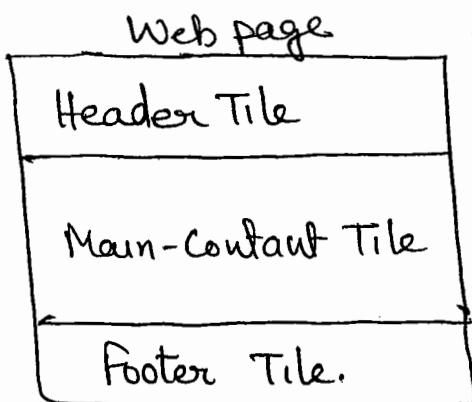
```
</Web-app>
```

#### Note :-

→ If you config both global and local exception handling  
② exception on jsp program the setting done in  
local exception handling cfg will take place.

## TILES FRAMEWORK / TILES PLUGINS :-

- Tile is a logic portion of a web-page. So far we generated webpages as the output generated by single web resource program. (content)
- Tiles framework or plugin renders each web page having multiple Tiles based on layout page. The values of these multiple Tiles will be ~~rendered~~ through multiple web resource program.



- In Tiles Framework environment all web pages of web site can be rendered based on the common layout page. So it gives layout control that means we can change appearance of every web page by just doing modification in layout page.

Ex:-

To increase Header Tile height modify the code in layout page and there is no need of doing modification in every web page.

Procedure to work with Tiles plugin:

### Step-1

- Cfg the Tiles plugin related class in struts cfg file  
In struts-config.xml (Tiles plugin cfg)

```
<plug-in className="org.apache.struts.tiles.TilesPlugin">
<set-property property="definitions->fixed name" value="/WEB-INF/
tiles-defs.xml"/> File definition cfg file
<set-property property="moduleAware" value="true"/>
</plug-in>
```

↑  
fixed name      ↑  
                  recognizes the  
                  modules

## Note

Any `<file_name>.xml` can be taken as Tile ~~or~~ definition Cfg file.

- Design layout page specifying the tiles.  
(refer layout.jsp of page no-105 & 106)

## Step-2

- Analyse and get number of web pages. in web appln.

### Step-3

Sug. - Write definition in tile definition cfg file on one/web page basis. Specifying Tile values of each page with respect to layout page.

- Refer Tiles-def.xml of page No-106 and 107.

### Step-4

- cfg Tile definition as result pages for Action class.

### In struts-config.xml

```
<action path="/xyz" type="XyzAction">
    <forward name="success1" path="aDef" />
    <forward name="success2" path="bDef" />
    <forward name="success3" path="cDef" />
</action>
```

↑ file definition name

### Step-5

→ Develop Action class using the above specify action-forward logical name

### In TestAction.java

```
public class TestAction extends Action
{
    public ActionForward execute(----) throws Exception
    {
        // B. logic
        if (cond1)
        {
            return mapping.findForward ("success1");
        }
        else if
        {
            return mapping.findForward ("success2");
        }
        else
        {
            return mapping.findForward ("success3");
        }
    }
}
```

## Note

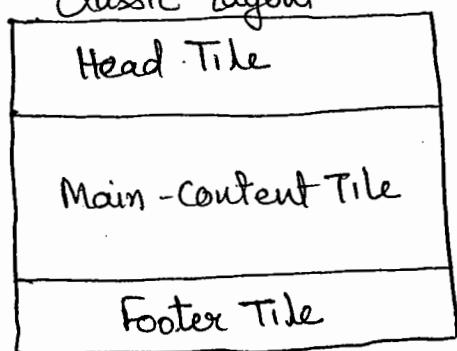
We can keep definitions of Tile definition cfg file participating in inheritance to get the benefit of extensibility as shown in page No-108 and 109.

16.09.13

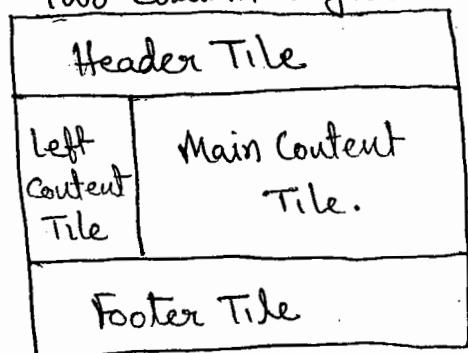
Tiles plugin or framework is given to develop web pages of Struts appln based on layout page having the advantage of layout control and uniform appearance.

→ Popular Layouts to design Layout pages :-

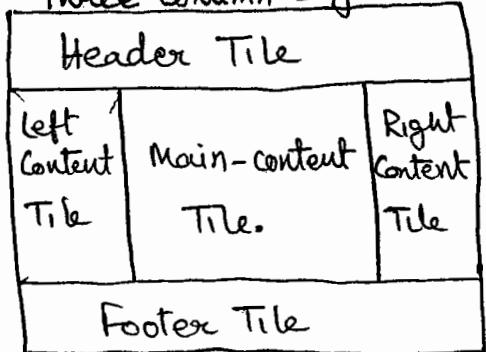
① Classic layout



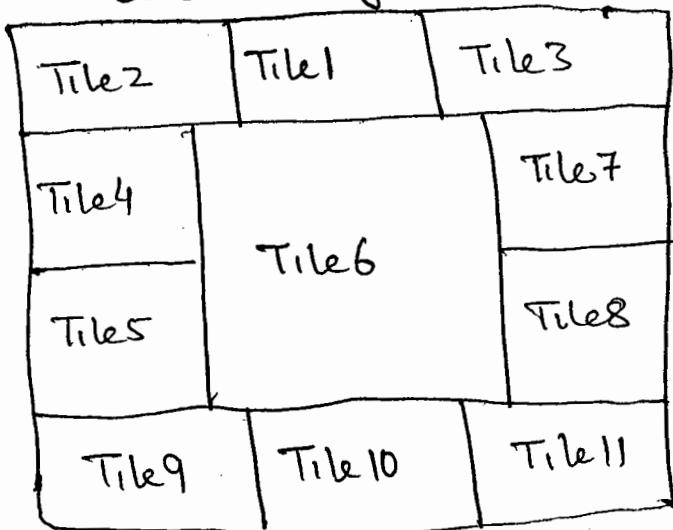
② Two Column layout



③ Three Column layout



④ Circular layout



## DAO Design Pattern

- The Java class that separates persistence logic from the other logic of the appl<sup>n</sup> and makes those logics as flexible logic to modify and reusable logics. is called DAO.
- Each DAO class contains logic to establish the connection to db s/w
- logic to close connection
- logic to perform persistence operations (insert, update, delete, select operations)
- DAO classes persistence logics can be developed by using persistence technologies like hibernate, jdbc, JPA and etc.
- In struts env... each Action class can have ~~a~~ separate DAO class or All Action classes can use single DAO class.
- For example appl<sup>n</sup> on Tiles framework having the implementation Of DAO design pattern refer appl<sup>n</sup>/3 of page No- ~~111~~ to 118
- While working with Tile framework to get 1st page through Tile defination we must make welcome page generating implicit request. This implicit request uses one or other Tile defination and generates <sup>1st</sup> web page-
- Flow of execution of Tiles appl<sup>n</sup> that is there in material as appl<sup>n</sup>/3 from page No-111 to 118.
- ⇒ Programmer deploys TilesApp web appl<sup>n</sup> in web server.

B> Because of <load-on-startup> ActionServlet will be pre-instantiated and ActionServlet reads and verify the struts cfg file entries... and Tile definition Cfg file entries.

C> programmer gives request to TilesApp web appln due to this the welcome page index.jsp executes automatically.

//<http://localhost:2020/TilesApp> (refer line-33 to 35)

D> index.jsp sends implicit request (refer line-5)

E> ActionServlet traps and takes the request. (refer line-16 to 31)

F> ActionServlet passes the request to <action> tag whose action path is /mytiles (refer line-197) This <action> tag forward the control to "basedef" tile definition of tiles-defs.xml file.

G> The tile definition "basedef" of tiles-defs.xml executes (refer line -232 to 237)

H> The layout page MyTilesHome.jsp executes to generate webpage having the following tile values. (refer line-95 to 112)

header → BaseHeader.jsp

footer → BaseFooter.jsp

left-content → LeftNavigation.jsp

main-content → Welcome.jsp.

I> End user clicks on insert hyperlink of LeftNavigation.jsp of left-content. (refer line- 134)

J> ActionServlet traps and takes the request. (refer line- 16-31)

K> ActionServlet passes the request to that <action> tag whose action path is ~~is~~ "/insert" (refer line-198)

- L> ActionServlet forwards the request to the Tile definition whose name is insert Tile (refer line-198 and 247 to 249)
- M> The layout page MyTilesHome.jsp executes having the following tile values  
Header : BaseHeader.jsp  
Footer : BaseFooter.jsp  
Main-content : Insert.jsp  
Left-content : LeftNavigation.jsp

#### Note

Since Insert.jsp is launched, the Insert.jsp related FormBean class object will be created (refer line-185 to 189)

- N> End user fills up the form page and submits the request. (refer line-78 and 66)

- O> ActionServlet traps and takes this request (refer line-16 to 31)
- P> ActionServlet writes the received form data by locating FormBean class obj. (refer line-185 to 189)
- Q> ActionServlet creates/locates InsertAction class obj and calls execute(-,-,-,-) method on it. (refer line - 207 to 210 ~~382~~)



## Note

This execute(----) method uses multiple methods of DAO class to complete record insertion process. (Set connection(), insertData(), close connection())

↳ 361

↳ prefer 272

↳ 289

R> execute(----) method returns ActionForward obj having success/failure (refer 403/405)

## Success

→ 208 → uses struts cfg file  
actionforward cfg.

→ 209 → uses struts cfg file  
actionforward cfg.

## failure

S> ActionServlet passes the control to tile definition  
(tile.main-success / tile.main-failure) (239-241) (243-245)

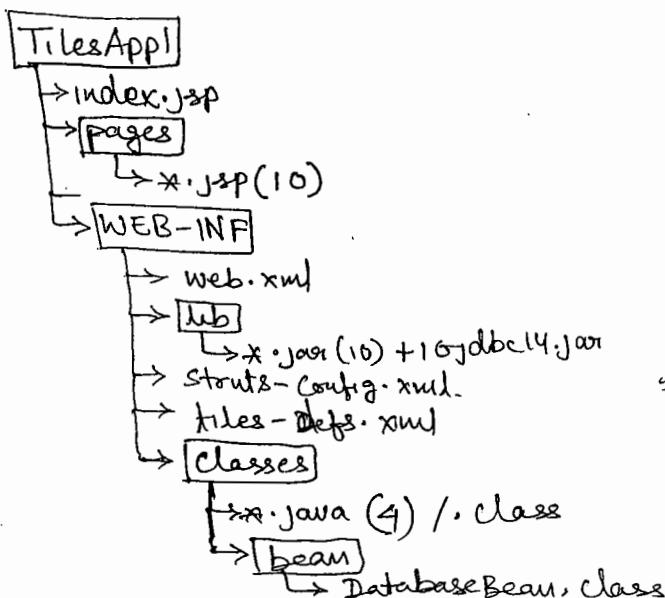
T> ActionServlet launches layout page (MyTileHome.jsp) having the following file values.

header → BaseHeader.jsp

footer → BaseFooter.jsp

left-content → leftNavigation.jsp

Main-content → success.jsp / failure.jsp



## Jar files in web-INF/lib folder

- Regular struts jar file + 10.jdbc14.jar

## Jar files in classpath

Servlet-api.jar, Struts-core-1.3.8.jar

## DB Table

StrutsEmployee (Table name)

→ EID number(4)

→ Name Varchar(20)

→ Address Varchar(20)

## Request Processor class :-

(OR)

Abstract class (~~Helper class~~)

Abstract controller (Helper class of ActionServlet)

- The helper java class of front controller servlet (ActionServlet) that reduces burden on frontController and allows to change behaviour of frontController is called Abstract controller
- In struts environment ActionServlet is frontController and RequestProcessor class is Abstract controller.

Problem :-

task1 → x();  
y();  
z();

We need to remember multiple method and their sequence of invocation to complete task.

Solution :-

```
public void xyz() (Complete method)
{
    x();
    y();
    z();
}
```

task1 → xyz();

- In solution we need to call only one method to complete the task because that one method internally calls multiple other methods in a sequence.
- Actually ActionServlet should call 16+ processXXX() methods to perform 16+ operation on each request. For this it has to call 16+ processXXX() methods of RequestProcessor class in a sequence. but it just call only one method of

RequestProcessor class that is process() method because this process(-) method is template method calling all 16+ processXXX(-) methods in a sequence.

→ ~~processPreprocess(-)~~ method of RequestProcessor class is an empty method returning null. This method is very useful for programmer to place custom logic while developing user-defined ~~request processor~~ RequestProcessor class by extending from pre-defined RequestProcessor class.

→ ActionServlet creates object for RequestProcessor class on one per module basis.

→ If you want to enhance the behaviour of existing RequestProcessor class we can develop custom RequestProcessor class by extending it from pre-defined RequestProcessor class.

→ We can give instruction to ActionServlet from struts-fig file using controller tag.

For information about various processXXX(-) and <controller> tag refer page - 119 to 124.

18.09.13

→ Procedure to develop custom request processor for model module of SwitchAction appl<sup>n</sup>. (appl<sup>n</sup>s of booklet) to allow request going to Action classes only when they are given after 9 am before 5 pm.

Step-1

Keep SwitchApp appl<sup>n</sup> ready (appl<sup>n</sup>s of booklet)

Step-2

Develop customRequestProcessor class by extending

it from pre-defined RequestProcessor class and  
override user choice ~~the~~ processXXX() methods.  
(Specially processPreprocess(, -) method)

MyProcessor.java (In WEB-INF\classes folder of SwitchApp)

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

public class MyRequestProcessor extends RequestProcessor {
    public MyRequestProcessor() {
        System.out.println("MyRequestProcessor: o-param constructor");
    }

    public boolean processProcessor(HttpServletRequest req, HttpServletResponse res) {
        System.out.println("MyRequestProcessor: processProcessor(, -)");
        // get current hour of the day
        Calendar cl = Calendar.getInstance();
        int h = cl.get(Calendar.HOUR_OF_DAY); // 24 hr format
        if(h == 9 || h >= 9 & h <= 17)
            return true; // continue the flow of execution
        else
            try {
                RequestDispatcher rd = req.getRequestDispatcher("/timeout.jsp");
                rd.forward(req, res);
            }
            catch (Exception e) {}
        return false; // aborts the execution
    }
}
```

```
    }  
    &lt;/processPreprocess(-1-)&gt;
```

### Step-3

Configure the above Custom RequestProcessor class in Model module specific struts configuration file using <controller> tag.

#### In struts-config-student.xml

after </action-mappings> tag.

```
<controller>  
<set-property property="processorClass" value="MyReq  
uestProcessor"/>  
</controller>
```

### Step-4

→ Add timeout.jsp to the root folder of web app<sup>n</sup>  
(SwitchApp!)

#### timeout.jsp

```
<b><font color=red>U must give request between  
9 am - 5 pm </font></b>
```

→ For one more scenario on custom RequestProcessor class development refer page No- 123 and 124.

## UNDERSTANDING AND SOLVING DOUBLE POSTING PROBLEM:

- While working with web application if form page submit button is clicked multiple times concurrently or if refresh button is clicked on the response page of form submission then the form page received same request will be processed for multiple times where these actually expected to process only for one time this is called "double posting problem".
- In struts "double posting problem" can be solved using tokens. These tokens are specific to each user/browser window. (will be managed through session)
- To prevent double posting problem by using tokens we can use the following methods of Action class.  
org.apache.struts.action.Action Class.

### ① ~~Save~~ 1. saveToken(-,-)

- saves a ~~to~~ new token in the user's/client's ~~to~~ current session
- Creates new HttpSession object if necessary.

### 2. resetToken(-,-)

- Resets the saved token from user's/ client's session.

### 3. isTokenValid(-,-)

- returns true if valid token is stored in the user's/client's current session.

→ returns false <sup>in</sup> the following situations:-

- a) if No session associated with this request.
- b) if No transaction token saved in the session.
- c) if No transaction token ~~not~~ included as a request

parameter

d) The included transaction token value does not match ~~the~~ the transaction token in the user's session.

→ For example app<sup>n</sup> on Tokens to prevent double posting problem refer App<sup>m</sup> given in page No-125 to 127.

20.09.13

→ Make Form page generating

request to Action class having valid Token. If request is generated without Token  
Make Action class processing request otherwise Action class rejects request generated by the Form page.

→ To make Form page generating request with valid Token make sure that the Form page is touched only after calling ~~saveToken~~ <sup>to</sup> saveToken() method.



### Struts And EJB Integration :-

→ Keeping Business logic and persistence logic in Struts Action class is not recommended process because of the following limitations

a) logic becomes specific to one web app<sup>m</sup>. (Kills the reusability)

b) Clients must use http protocol environment to access the logic

c) Struts does not supply middleware services. So they must be implemented by programmer manually.

- To overcome above problems keep business logic and persistence logic in EJB component (or) sprung appln (or) Spring with JBoss (or) webservices and make Action class interacting with them.
- When Struts Action class contains Business logic then it is called Model layer resource.
- ⇒ When ~~business~~ Struts Action class interacts with other Model layer resources then it is called controller layer resources.
- In Struts and EJB integration the Struts Action class ~~contains logic~~ acts as client to EJB component.
- Every EJB 3.x components contains two resources
  - a) Business Interface (declaration of B. methods)
  - b) Bean class (Implementation of B. methods)

### 3 types EJB Components :-

#### 1. Session Beans

- a) Stateless Session Bean (SLSB)
- b) Stateful Session Bean (SFSB)

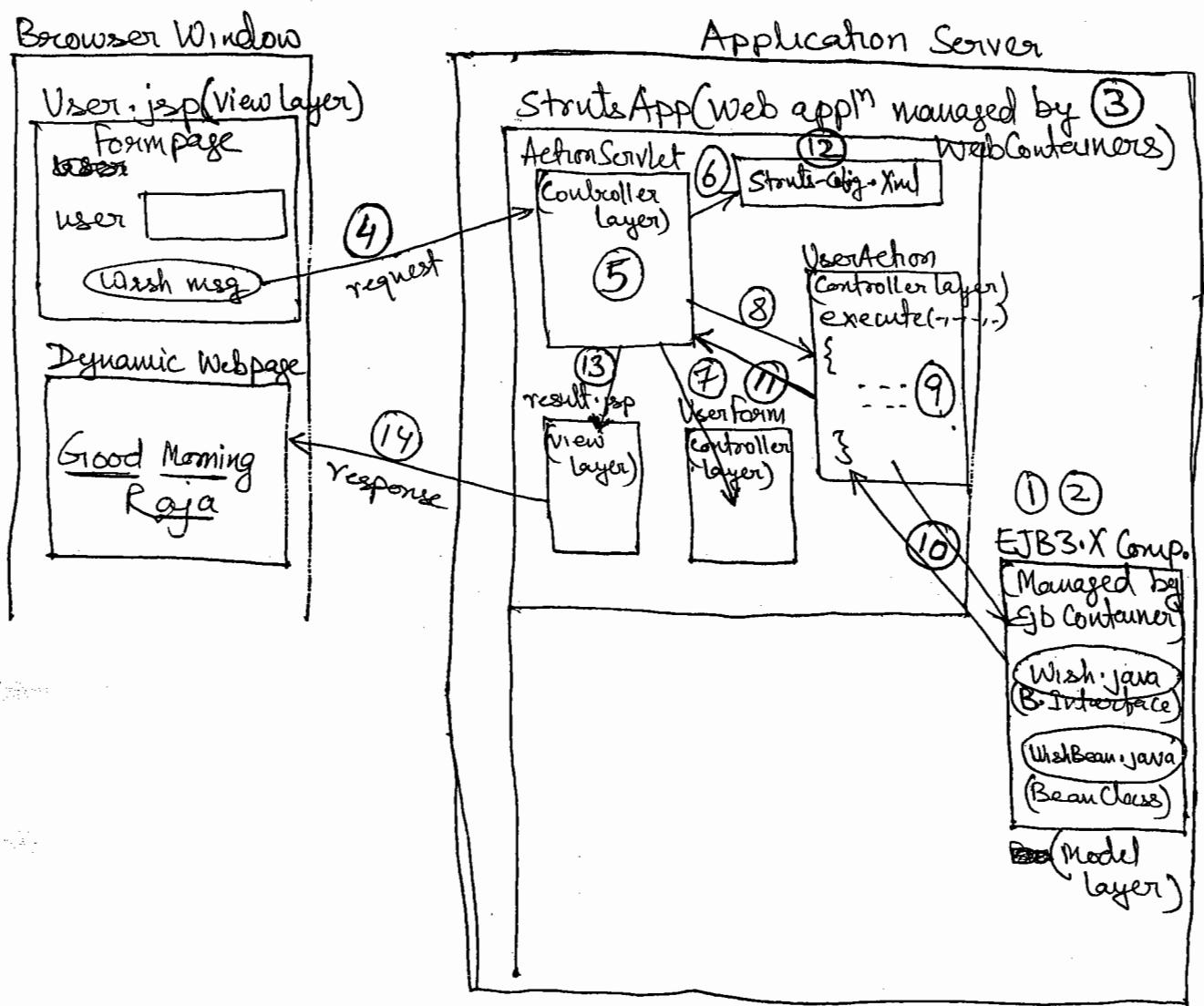
#### 2. Entity Beans

- a) Bean managed persistence entity Beans (BMP EJB)
- b) Container managed persistence entity beans (CMPEB)

#### 3. MessageDrivenBean (MDB)

- EJB 3.x programming is annotations based programming.

## Example appl<sup>n</sup> on Struts and EJB Integration



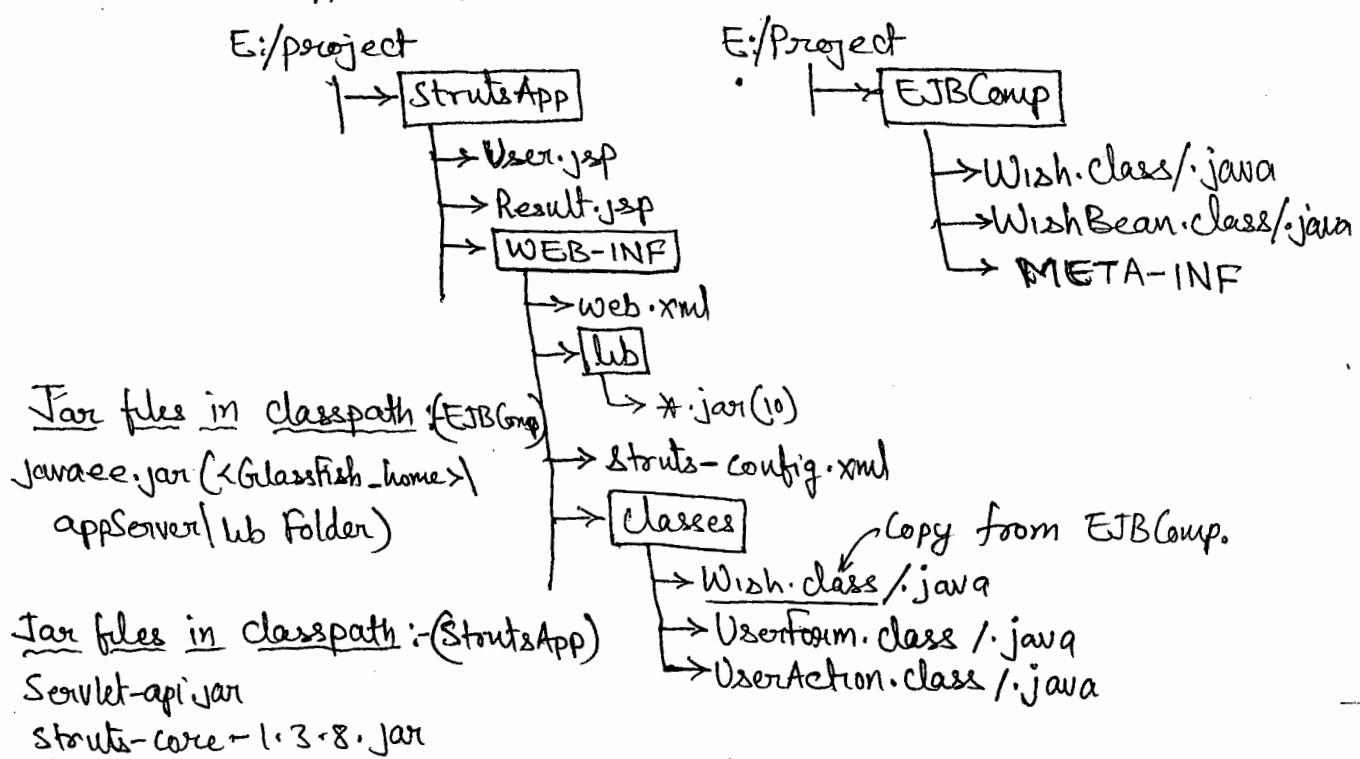
→ With respect to the diagram:-

- (1) Programmer deploys EJB 3.x Component in Web App<sup>n</sup> Server (Glassfish)
- (2) The EJB containers generate dynamic classes and registers wrapper object references in Jndi registry having Business interface name as jndi names.
- (3) Programmer deployed struts app<sup>n</sup> in App<sup>n</sup> server, in this process all deployment related or server start-up related formalities will be completed.

- ⑨ The execute(-,-,-,-) method of Action class gets wrapper ~~object~~ object reference from Jndi registry.
- ⑩ execute(-,-,-,-) methods calls Business method of EJB component by using wrapper object reference and gets results from business method.

21.09.13

- For example appl<sup>n</sup> for struts and EJB integration refer appl<sup>n</sup>13 of page No- 128 to 130.
- In order to interact with Jndi registry of a server being from that server Jndi properties are not required to create initial context object. but Jndi properties are required to interact with same registry from outside the server.
- Procedure to deploye and execute Struts and EJB integration example appl<sup>n</sup> (appl<sup>n</sup>13)



→ Prepare jar file representing ejb component :-

E:\project\ejbComp>jar cf Compl.jar .

→ Prepare war file representing struts application :-

E:\project\strutsApp>jar cf StrutsApp.war .

Approach1 for Deployment :-

→ Deploy StrutsApp.war and Compl.jar files separately Glassfish 2.X server.

→ Copy StrutsApp.war and Compl.jar files to <Glassfish-home> \ Appserver\domains\domain1\autodeploy folder.

→ Start Default Domain server of Glassfish  
start → progs → SunME → Application server → start default server.

→ request url : ~~http://localhost:8080/StrutsApp/~~ user.jsp

Approach2 for Deployment

→ Create ear file having both war file and jar files

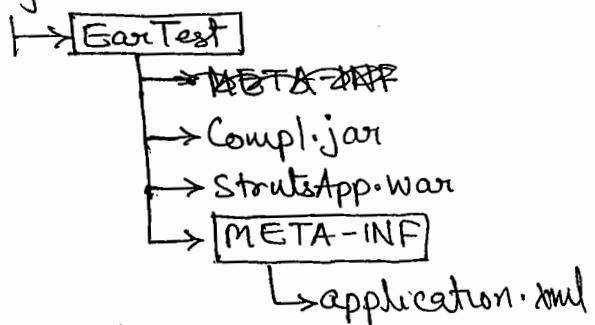
ear file = jar + jar + ...

ear file = war + war + ...

ear file = jar + war + ...

Creating ear file

E:\project



E:\project\EarTest>jar cf FinalApp.ear .

application.xml

```
<DOCTYPE ----- ----- /application_1-3.dtd">

<application>
  <display-name> EARfile </display-name>
  <module>
    <web-uri> StrutsApp.war </web-uri>
    <context-root> /MyWeb </context-root>
    </web>
  </modules>
  <module>
    <ejb> Compt.jar </ejb>
  </module>
</application>
```

→ application.xml dd (deployment descriptor) file of ear file contains configuration details about both war and jar files that are placed into ear file.

#### Deployee ear file

Copy FinalApp.ear file to <Glassfish-home>\AppServer\domains\domain1\autodeploy folder.

Request url : http://localhost:8080/MyWeb/user.jsp

→ In NetBeans IDE we can create project as java ee enterprise project that give one ejb and one web module to develop the above struts ejb integration example application.

~~Struts2.X = Struts1.X + Webwork F/W.~~

Struts2.X = Struts1.X + Webwork framework

22.09.13

## STRUTS 2.X

Struts2.X = Struts1.X + Webwork framework

Limitation of Struts1.X :

- Debugging is very much difficult because of XML domination.
- Gives bad API having more abstract classes.
- Doesn't support annotations and Java5 features.
- Working with Validator plugin is very complex.
- Doesn't give rich UI Component
- No built-in support for Ajax, Spring integration.
- Stateless checkboxes should be handled separately.
- Doesn't allow multiple technologies in view layer.

→ For struts2.X features refer page No-134 and 135.

→ Unit Testing is complex and customRequestProcessor class is ~~changeable~~ configurable per module basis (not per Action class basis) in Struts1.X.

Q: What is the difference between Struts1.X and Struts2.X?

Sol: Refer page No-136 and 137.

## Struts 2.0.11

### Struts 2.X

type: java based web framework

purpose: To develop MVC2 architecture based web application.

version: 2.3 X (Compatible with jdk 1.5+)

Designed based on: Servlet api 2.5, JSP 2.0 Struts1.X and web work.

Vendor: Apache Foundation

Open ~~so~~ source s/w

To download s/w: download zip file (Struts-2.0.11-all.zip)

(or) Struts-2.3.4-all.zip)

from www.apache.org (website).

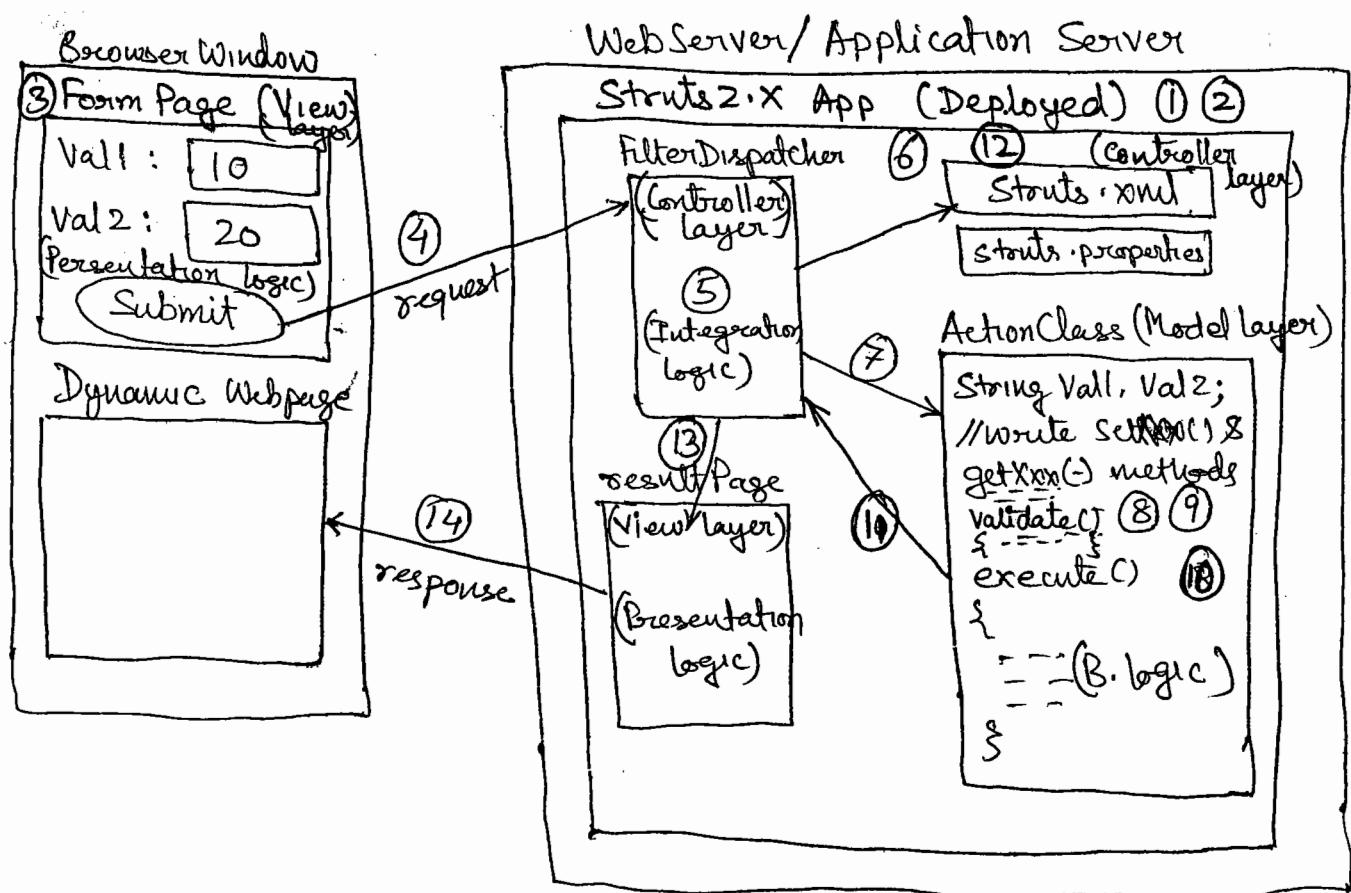
→ <Struts2-home>\lib\struts2-core-2.0.11.jar, xwork-2.0.11.jar  
 jar represents Struts2.X api's      Version

→ Terminology matching between Struts1.X and Struts2.X.

Feature	Struts1.X	Struts2.X
1. Controller	ActionServlet (Servlet)	FilterDispatcher StrutsPrepareAndExecuteFilter (Servlet filter)
2. Form Beans	Form Beans	No Form Beans
3. Action Class	Action class (Non-pojo class)	Action class (pojo class)
4. Struts- <del>cfg</del> .file	struts cfg file any <filename>.xml	struts cfg file struts.xml
5. Result page cfg	ActionForwards	results cfg
6. Action Mapping	ActionMapping	Packaging (No way related with java pkgs)
7. Abstract Controller	Request processor	16+ interceptors

Q: Why struts 2.x is taking ServletFilter program as controller?

- Aus:- 1. Controller should be pre-initiated either during server startup (or) during the deployment of web application. to make itself ready and available from the beginning. To achieve above pre-initiation of servlet we must enable load-on-startup on servlet program. Servlet Filter is having the natural behaviour of getting pre-initiated without enabling any load-on-startup.
2. Controller must trap and take all the request, response of web appln. To bring above behaviour for servlet program it must be cfg in web.xml either with extension match url pattern or with directory match url-pattern. Servletfilter is having natural behaviour of trapping request and responses of web application.



With respect to diagram :-

- ① Programmer deploys struts2.x applications in server
- ② The controllerfilterDispatcher will be pre-initiated automatically either during deployment or during server startup. In this process FilterDispatcher <sup>& Struts.properties</sup> reads and verifies struts.xml file entries.
- ③ End user touches formpage on browser window (related ActionClass object will not be created).
- ④ After filling the form page end user generates the request.
- ⑤ FilterDispatcher traps and takes the request.
- ⑥ FilterDispatcher used struts.xml entries to decide the ActionClass for processing the request.
- ⑦ FilterDispatcher creates Action class object.
- ⑧ FilterDispatcher calls setXXX() methods on Action class object to write received form data to Action class properties.
- ⑨ FilterDispatcher calls validate() method to perform Form validation
- ⑩ FilterDispatcher calls execute() method to process request by executing B. logic
- ⑪ execute() method returns result stream back to FilterDispatcher
- ⑫ FilterDispatcher uses struts.xml entries to decide the result page of Action class.

- (13) FilterDispatcher passes control to result page
- (14) Result page formats the result and sends response to browser window as dynamic web page.

### Note

→ While performing Step- 6 to 13 filterDispatcher uses various predefined interceptors (16+) to decide Action class flow.

→ Struts2.x package is a logical unit where set of Action classes interceptors, interceptor-stack, results can be configured.

→ Interceptors stack is a place where set of related interceptors are placed.

→ Struts2.x s/w gives struts-default.xml file having package called struts-default. This package contains 11 result configuration, 30 interceptor cfg, 30 interceptor stack cfg, and also specify default-stack as default interceptors ~~stack~~ of Action class if no interceptors are cfg ~~specified~~ explicitly.

Some resultType

chain, dispatcher, stream and etc...

Some interceptors

cookie, alias, file uploading, timer, token etc...

Some interceptors stack

basicStack, validationWorkflowStack, defaultStack etc..

- Struts-default.xml file is available in <struts2-home>\lib\struts2-core-<version>.jar.
  - The defaultStack, interceptorStack contains 16+ interceptors to decide the flow of execution related to ActionClass.
  - To make ActionClass of Struts2.x using all interceptors of defaultStack.
- ① include struts-default.xml file to struts.xml file.
  - ② Create package extending from struts-default package
  - ③ Configure Action class without interceptors cfg.
  - ④

#### Example:- Struts.xml

```

<!DOCTYPE ----- /Struts-2.0.dtd">
<struts>
  <include file = "struts-default.xml"/>                                ↗ mandatory
                                                               ↗ package name in
                                                               ↗ our package name
                                                               ↗ Struts-default.xml
  <package name = "pack1" extends = "struts-default">                    ↗ action logical name
                                                               ↗ not related to java pkg,
                                                               ↗ Action class name
  <action name = "display" class = "pl.DisplayAction">
    <result page = "success"> @/success.jsp</result>
    <result page = "fail">/fail.jsp</result>
  </action>                                                               ↗ results cfg
  </package>
</struts>

```

- In struts1.x Action class is identified with its action path.
- In struts2.x Action class is identified with its logical name.
- To make servletFilter as controller of web app" it must be configured in web.xml file with "/\*" url-pattern. This indicates we must cfg struts2.x supplied filter in web.xml

24.09.13

file with "/\*" wild-pattern.

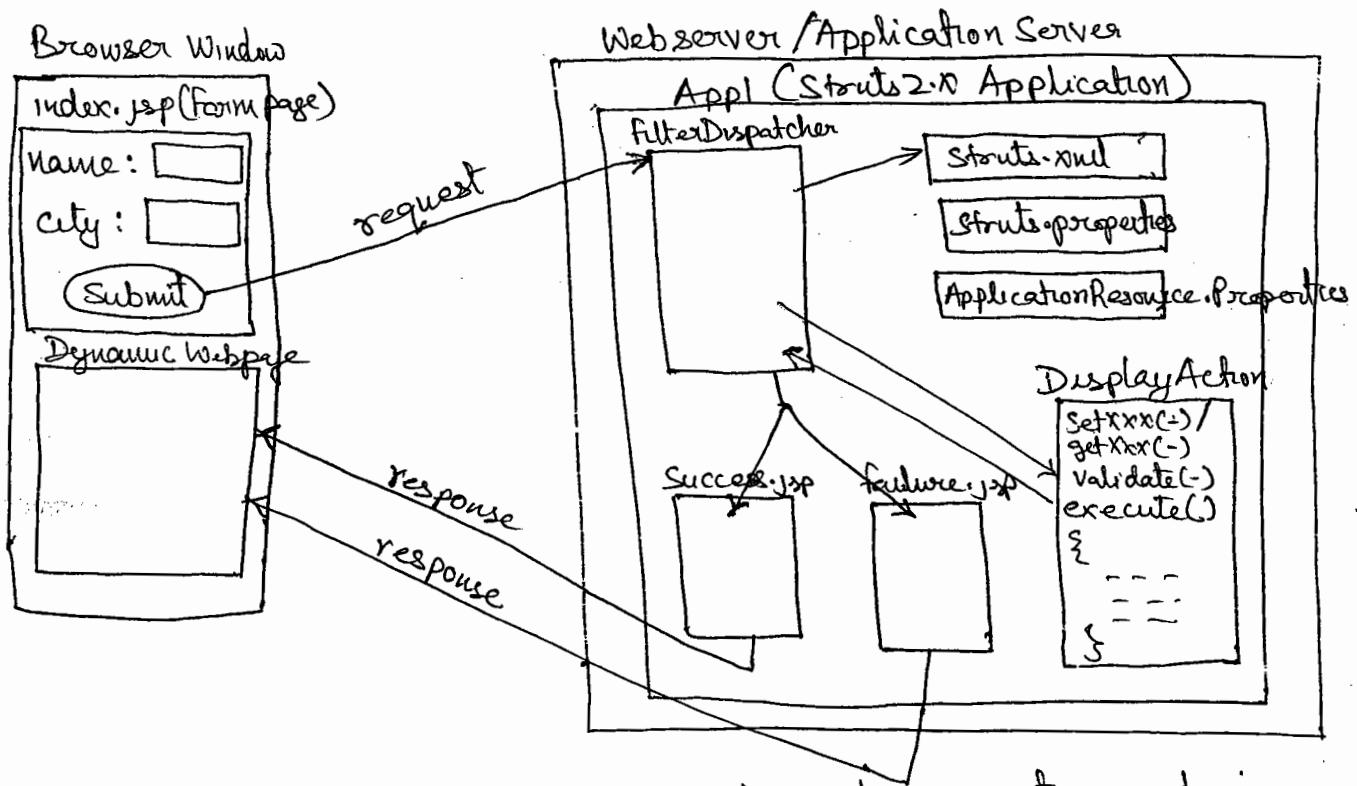
The Filters are :-

FilterDispatcher (or) PrepareAndExecuteFilter.

→ Struts.properties is fixed file which can be used to provide additional instruction to struts2.x framework.

like - cfg of property file, I18n setting, mode of struts & w utilization (development mode / production mode)

First Struts2.X Application Development :-



FilterDispatcher reads struts.xml, struts.properties entries together when it is pre-instantiated.

⇒ For above diagram based app<sup>n</sup> prefer app<sup>0</sup>1 of the page Nos - 138 to 140.

→ While cfg struts2.x action class if no business method name is cfg explicitly then its looks to ~~execute~~ <sup>use</sup> execute() method as default business method.

→ Since Action class is Pojo class here. So there is no need of adding jar files in classpath.

Jarfiles in WEB-INF/lib folder.

Commons-logging-1.1.jar, freemarker-2.3.8.jar, struts2-core-2.0.6.jar, ognl-2.6.11.jar, XWork-2.0.1.jar.

### Request workflow

http://localhost:2020/App1/index.jsp

→ Flow of execution for appM1 of the page nos-138 to 140

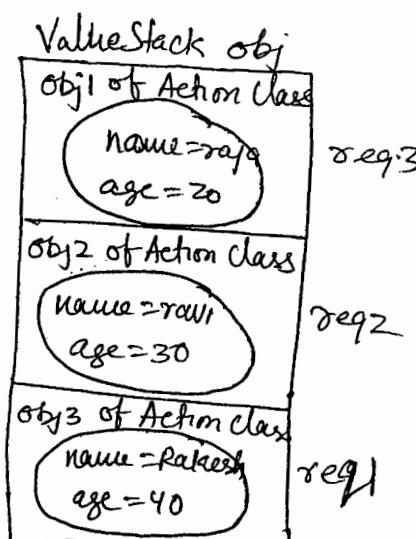
- (A) (B) Same as ① ② Struts2 flow diagram refer previous class discussion.
- (C) (D) Same as ③ ④ of steps of struts2 flow diagram.  
(ref 4, 10, 11, 13 line of App1 page-138)
- (E) FilterDispatcher (FD) traps and takes the request (27-30 and 32-35)
- (F) FilterDispatcher uses struts.xml entries to decide the action class (refer 60)
- (G) FilterDispatcher creates Action class object for current request
- (H) FD calls setXXX() methods on Action class objects and writes form data to Action class properties. (refer 82, 92)
- (I) FD calls execute() on Action class objects (refer 97-103)
- (J) execute() method process the request through business logics and generate result string (Success/fail) and this result string goes to filterDispatcher (refer line 998/100)
- (K) FD uses result page cfg to decide the result page of

action class. (refer 61 or 62)

Q) FilterDispatcher passes the control to result page (success.jsp / fail.jsp) (refer 105 to 107) or (refer 118 to 130)

Q) Why Struts 2.x is creating multiple objects for action class when multiple request are given?

Ans: In struts2.x ValueStack obj is introduced to maintain all request data each request data will be there in action class object. To place multiple request data in the form of multiple ~~action class~~ objects of a action class. in valuestack object. The struts2.x env... is creating multiple objects for action class for multiple request.



Here, Every Action class of Struts2 maintains separate ValueStack obj. This ValueStack obj visible throughout struts2.x app<sup>n</sup>.

In struts Note To pass user-defined business method instead of execute() method.

In struts.xml

```
<action name="display" class="P1.DisplayAction">
```

    </action>

uses execute() method as business method

```
    <action name="display" class="P1.DisplayAction" method="m1()">
```

    </action>

uses m1() method as business method.

25.09.13

## To Pass data to Result page from Action Class:-

We can pass data to Result page from Action class using following two approaches:-

### Approach-1

- Using request, response session, application attributes
- For this we need to inject request, session, application objs through interface injection by implementing XxxAware classes on Action class.

### Approach-2

By using ValueStack obj.

- Approach-2 is recommended because it allows keep action as pojo class.

### Approach-1 Implementation:-

#### List of XxxInterface:-

- (I) ServletRequestAware → To inject HttpServletRequest obj
- (I) ServletResponseAware → To inject HttpServletResponse obj
- ApplicationAware(I) → To inject ServletContext obj

#### Example on Approach-1

Step-1 > keep application-1 ready (refer page-138)

Step-2 > Develop the action class as shown below.

#### DisplayAction.java

```
package P1;  
import org.apache.struts2.util.ServletContextAware;  
import org.apache.struts2.interceptor.ServletRequestAware;  
import javax.servlet.*;
```

```

import javax.servlet.http.*;
public class DisplayAction implements ServletContextAware,
    { ServletRequestAware // for interface injection.
        private String username;
        private String city;
        // -- // getter & setter methods.
        Set HttpServletRequest request;
        public void setServletRequest(HttpServletRequest request)
        {
            this.request = request; method of ServletRequestAware(I)
        }
        ServletContext context;
        public void setServletContext(ServletContext context)
        {
            this.context = context; method of ServletContextAware(I)
        }
        public String execute() throws Exception
        {
            request.setAttribute("attr1", "val1");
            context.setAttribute("attr2", "val2");
            // b. logic
            if(this "Anut".equals(username) && "Patna".equals(city))
                return "success";
            else
                return "fail";
        }
    }

```

This approach-1 makes Action class as non-pojo class

Step-3 Add following three jar files to classpath.

1. struts2-core-2.0.11.jar, ~~xwork~~-2.0.4.jar, servlet-api.jar

Step-4 Read attribute values from success.jsp/fail.jsp  
 Attr1(req) value <% =request.getAttribute("attr1") %> <br> in success.jsp  
 Attr2(ServletContext) value <% =application.getAttribute("attr2") %> <br>

## Approach-2 implementation (Recommended)

Step-1 > keep appln-1 ready

Step-2 > Write following additional code in Action class to store the results in the instance variable of Action class.

### DisplayAction.java

// after setter, getter method

```
private String res1, res2;
```

```
public String getRes1()
```

```
{ return res1; }
```

```
public String execute() throws Exception
```

```
{
```

```
    res1 = "val1";
```

```
    res2 = "val2";
```

// B. Logic

```
if("Ankit".equals(username) && "Pune".equals(city))
```

```
    return "success";
```

```
else
```

```
    return "fail";
```

```
} // execute()
```

```
} // Class.
```

Step-3 > Use <s:property> tag in success.jsp/fail.jsp to read values from ValueStack obj

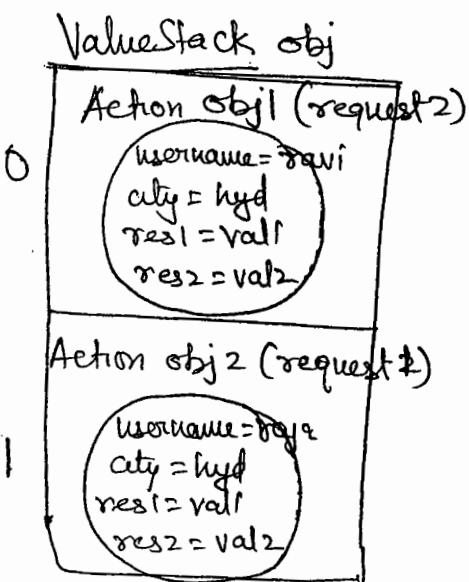
### In success.jsp

Result values are : <s:property value="res1"/>,

```
- <s:property value="res2"/> - <br> <br>
```

form Data is : <s:property value="username"/>,

```
<s:property value="city"/>
```



<s:property> tag always reads values from the top of stack by default.

→ Struts2.X gives only one jsp tag library, its taglib will be / struts-tags.

This jsp tag library contains 3 types of tags

### 1. Generic Tags

- Control tags
- Data tags

### 2. UI Tags

- form tags
- Non-form tags
- Ajax tags (Uses DOJO toolkit internally)

tld file = struts-tags.tld.

⇒ Procedure to convert appln1 from struts2.0 to struts2.3

Step-1 > Make sure that org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter is config in web.xml file instead of filterDispatcher.

Step2 > Removing existing jar files from lib folder and add follow jar files (II)

Collect these 11 jar files from  
<Struts2.3-home>\apps\struts2-blank-2.3.4.war file  
extraction.

26.09.13

### Annotation in Struts2.x

→ Annotation in Struts2.x are given for resource cfg and for form validations

@Result, @Results are given to cfg result pages for Action class.

### The Annotations for Form Validation:-

@EmailValidator

@RequiredStringValidator

@StringLengthFieldValidator

@IntRangeFieldValidator

@DateRangeFieldValidator

and etc...

All these annotations are available in

com.opensymphony.xwork2.validator.annotations

pkg.

### FormValidations in Struts2.x

#### 1. Client Side

→ Programmatic Approach

    → Using javascript code  
        manually

→ Declarative Approach

    → Using Xwork Validator

#### 2. Server Side

Programmatic

Using java code placed  
in Validate() method  
of Action class.

Declarative

→ Using Xwork  
Validator based on  
Xml cfg or Annotation  
cfg.

→ To get more control on flow of execution of Action class struts2.x allows us to develop custom interceptor. For this we must take a class extending from com.opensymphony.xwork2.interceptor.AbstractInterceptor

→ Procedure to develop custom Interceptor for an Action class.

#### Step-1

Take a java class extending from abstract interceptor class

refer MyInterceptor.java of page No-153 and 154

#### Step-2

cfg interceptor in struts.xml file having logical name.  
(935-937) of page-153.

#### Step-3

link the above interceptor with action class along with ~~default stack~~. DefaultStack.

refer. 940 - 941 page -153

→ Develop remaining resources in regular manner.

~~Note~~

→ For the above based steps complete example app<sup>n</sup>  
refer app<sup>n</sup>8. of page No-152 to 153

#### Note

In struts2.x interceptors are configurable per Action class basis (Not per module basis)

## FILE UPLOADING:-

30.09.13

- Struts 2.x gives interceptors "fileupload" to perform file uploading.
- We need to use this "fileupload" interceptor along with FileUtil class.

For example appln on File uploading refer appln of page nos - 149 ~~to~~ 151

## FILE DOWNLOADING

→ result-type is use for file downloading.

for result-type:

Struts-core-2.0.11.jar → struts-default.xml → line 75 to 84

→ Struts 2.x gives some built-in result-types.

i) dispatcher (default) → uses std. forward (-) to pass the control to destination page.

ii) freemarker → to use freemarker technology in view layer

iii) Velocity → to use velocity technology in view layer.

iv) Stream → sends stream to response (useful in file download operation) and etc...

→ The default result-type for any <result> tag is dispatcher

→ for example appln on file downloading using the result-type "stream" refer appln of page nos - 151 and 152

→ While working with Annotation we can avoid ~~struts~~ struts.xml file from web app<sup>M</sup>. In this process we must take Action class name as XAction if action id of form page contains ~~X~~. More ever the package name of Action class must be cfg in XML file as Filter init() parameter.

→ For Annotation based struts2.x example app<sup>M</sup> prefer application2 of page No-140 and 142)

→ As the value of one Annotation parameter another annotation can be cfg.

→ If Annotation parameter name is value we can set value to that parameter without specifying parameter name.

#### Note

→ In struts1.x Validator rule contains default error msg with arguments. In struts2.x ~~there~~ there are no default error msg.

→ So every error msg should be cfg by programmer manually.

→ While performing declarative XML, Annotation based formValidations our Action class must extends from ActionSupport class because this ActionSupport class implements multiple interfaces and also contains validate() method having logic to use xwork validator to perform formValidations.

→ In struts2.x to perform xwork validator based client side declarative form validations we need to keep the server side declarative form validation err... as it is and we need to place validate="true" attribute in ~~<~~s:form....> tag.

→ This JavaScript generated error msg will be rendered directly to form page (No alert dialogbox)

In user.jsp

```
<s:form action="show" method="post" validate="true">
    <!--
    -->
</s:form>
```

Note

for this we must take form page as error page.

⇒ If other than form page is taken as input page/error page to display the form validation error msg use

<s:fielderror> tag

Ex:                    error.jsp (when it taken as error page)

```
<%@taglib uri="/struts-tags" prefix="s" %>
<s:fielderror />
```