

# STRUTS 1.X

BY

**Mr. NATARAJ**

**Sun Certified**

## INTRODUCTION

- A web application is a collection of static and dynamic web resource programs or client side and server side web resource programs.

- Static web resource programs generate static web-pages.

Ex: Html programs

- Dynamic web resource programs generate dynamic web-pages.

Ex: Servlet and Jsp Programs

- Client side web-resource programs come to browser window from web-application of web server for execution.

Ex: Html programs, Java Scripts, Ajax programs.

- Server side programs executes in the server.

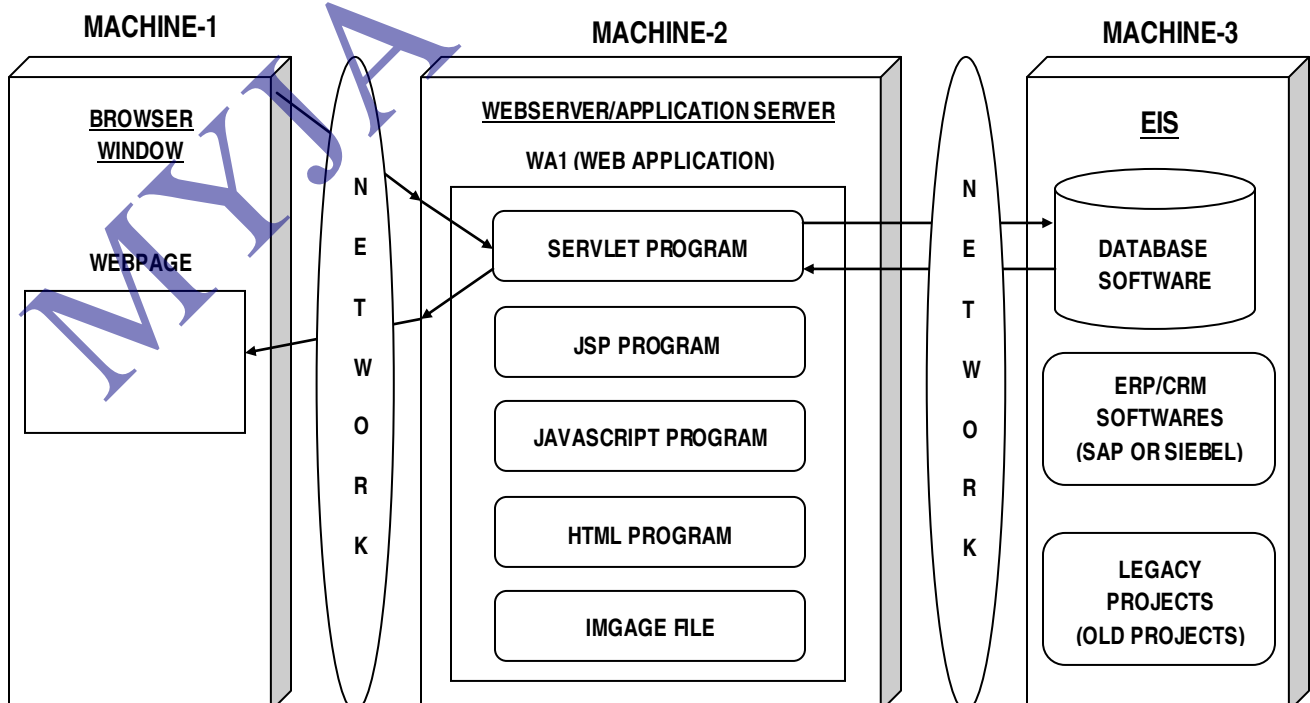
Ex: Servlet programs, Jsp programs

- Beside whether web-resource program is client side program and server side program based on the place where it executes. Not based on the place where it resides.

- Multiple backend softwares together are called as EIS software.

- A project that is developed by using old technologies is called as legacy project.

Ex: Fotron project, JDK 1.0 project, C project and etc.



## The typical web-application contains the following logics

- ✓ Request Parameter Gathering Logic
  - ✓ Form Validation Logic
  - ✓ Business/Request Processing/Service Logic
  - ✓ Session Management Logic
  - ✓ Presentation Logic
  - ✓ Persistence Logic
  - ✓ Middleware Services Configuration/Implementation Logic and etc.
- The logic that can read all the details from Http request like header values, Request parameter values and miscellaneous details is called as request data gathering logic.
  - The logic that verifies the pattern and format of the data is called as form validation logic. It can be done at client side or server side.  
Ex: Checking whether email id is having '.', '@' symbols.
  - The main logic of application that generates result by using input values and by performing calculations is called as business logic.  
Ex: Credit card or debit card processing and etc.
  - The logic that remembers client/user data across the multiple requests during a session and makes web-application as state-full web application is called as Session management or Session tracking logic. We use hidden forms, cookies, URL rewriting and extra techniques to this purpose.
  - The logic that generates user interface for end users and formats results generated by business logic and also can send web-pages to browser window is called as presentation logic.
  - The logic that interacts with backend softwares like Database softwares and manipulates the data by performing CRUD operations is called as persistence logic.  
Ex: JDBC code, Hibernate code.
  - The additional logic/services that configurable on web-applications to make our web applications running smoothly in all the situations are called as middleware services.
  - Middleware services are not minimum logics of the application development. They are additional, optional and configurable logics of the application.  
Ex: Security service, Transaction Management, JDBC connection pooling and etc.

## **An Example Program (Sample Code) of web application**

### **Request Data Gathering Logic**

- ✓ Read m1, m2, m3, sno and same details of student from form page

### **For Validation Logic**

- ✓ Validate whether required field typed or not.
- ✓ Validate whether marks are coming as numeric values or not.

### **Business Logic**

- ✓  $Total = m1 + m2 + m3$   
 $Avg = total / 300f$
- ✓ Generate rank for student.

### **Persistence Logic**

- ✓ Write student details to database table as record

### **Presentation Logic**

- ✓ Display student details and results on the browser window as html table content.

### **Middleware Services**

- ✓ Apply security services on application.
- ✓ Use JDBC connection pool support to interact with database software.

## **Different approaches of developing java web application**

- Model-1
- Model-2

### **Model-1**

In this approach, we use either only servlet programs or only jsp programs as server side web resource programs in web application.

### **Model-2**

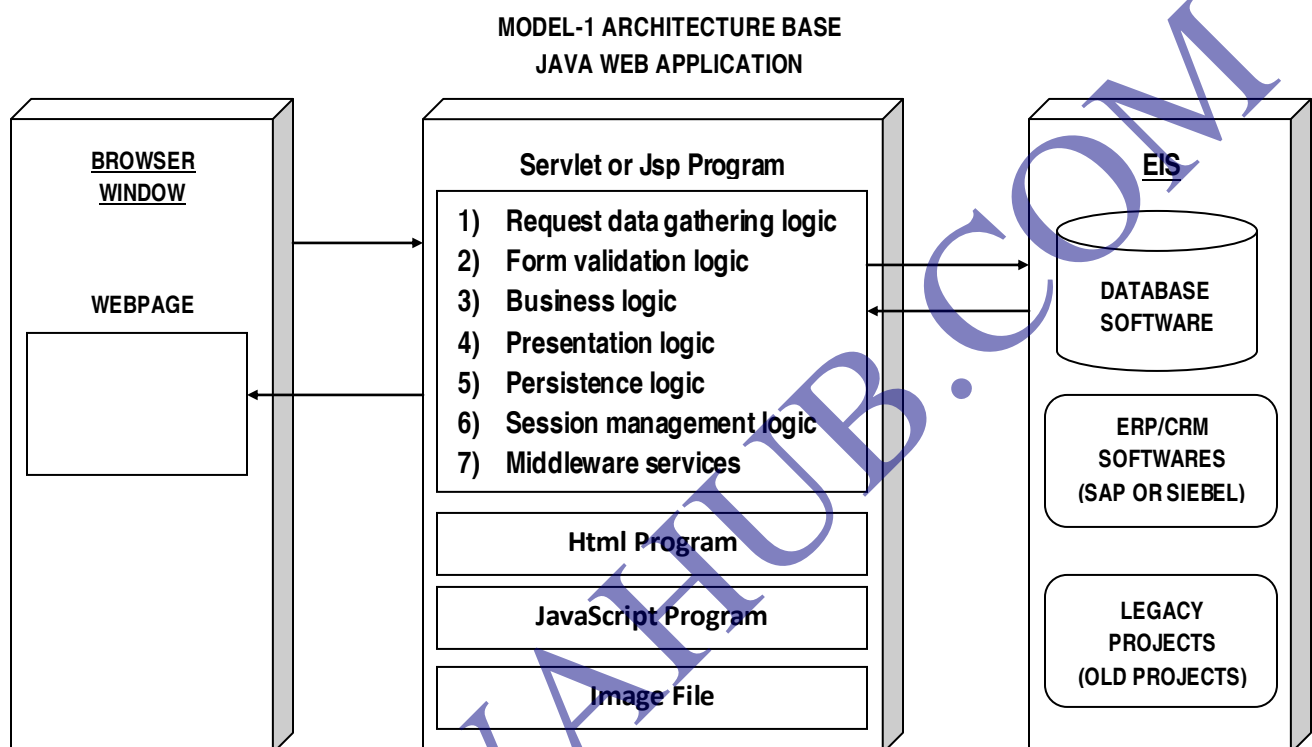
This model is again divided into two sub models. They are

- MVC-1
- MVC-2

MVC represents Model-View-Controller approach.

- In MVC-1, MVC-2 models multiple technologies support will be taken to develop web applications.
- In MVC-1, MVC-2 models multiple layers will be there in web-application development.
- Each layer in the application represents one logical partition of the application having logics.

### Model-1 architecture/Model-1 approach



- ✓ In model-1 architecture based web application if servlet programs are use jsp programs will not be used and vice versa.
- ✓ In every server side web resource program of model-1 architecture based web application like servlet program or jsp program, multiple logics will be placed.

### Drawbacks of Model-1 Architecture

- ☒ In sever side program multiple logics will be mixed up. So we can say there are no clean separations of logics.
- ☒ Modification done in one logic affects other logics. Due to this enhancement and maintenance of the project is complex.
- ☒ Parallel development is not possible so productivity is very poor.
- ☒ Some middleware services must be implemented by the programmer manually along with regular logics. This improves burden on the programmer.

## Advantages

- ☑ Since parallel development is not possible since multiple programmers are not required.
- ☑ Knowledge on multiple technologies is not required to develop the web application.

## Note

- Use Model-1 architecture only to develop small scale web-applications which contains single digit count web-pages.
- To solve all the problems of Model-1 architecture use Model-2 architecture. (MVC-1 or MVC-2)
- Now-a-days the software industry prefers to develop MVC-2 architecture base software projects by using multiple technologies support.

M:	Model Layer	:(Business Logic + Persistence Logic) :Like Accounts officer :Use java class, java bean, EJB, spring, Spring with hibernate, web services and etc.
V:	View Layer	:(Presentation Logic) :Like Beautician :Use JSPs, HTML, Free markers, velocity and etc.
C:	Controller Layer	:(Integration Logic/Controller Logic) :Like Traffic police :Use servlets, servlet filters and etc.

The integration logic of controller layer keeps track of all the operations of web-application execution. Also monitors and controls the flow of execution in the web-application.

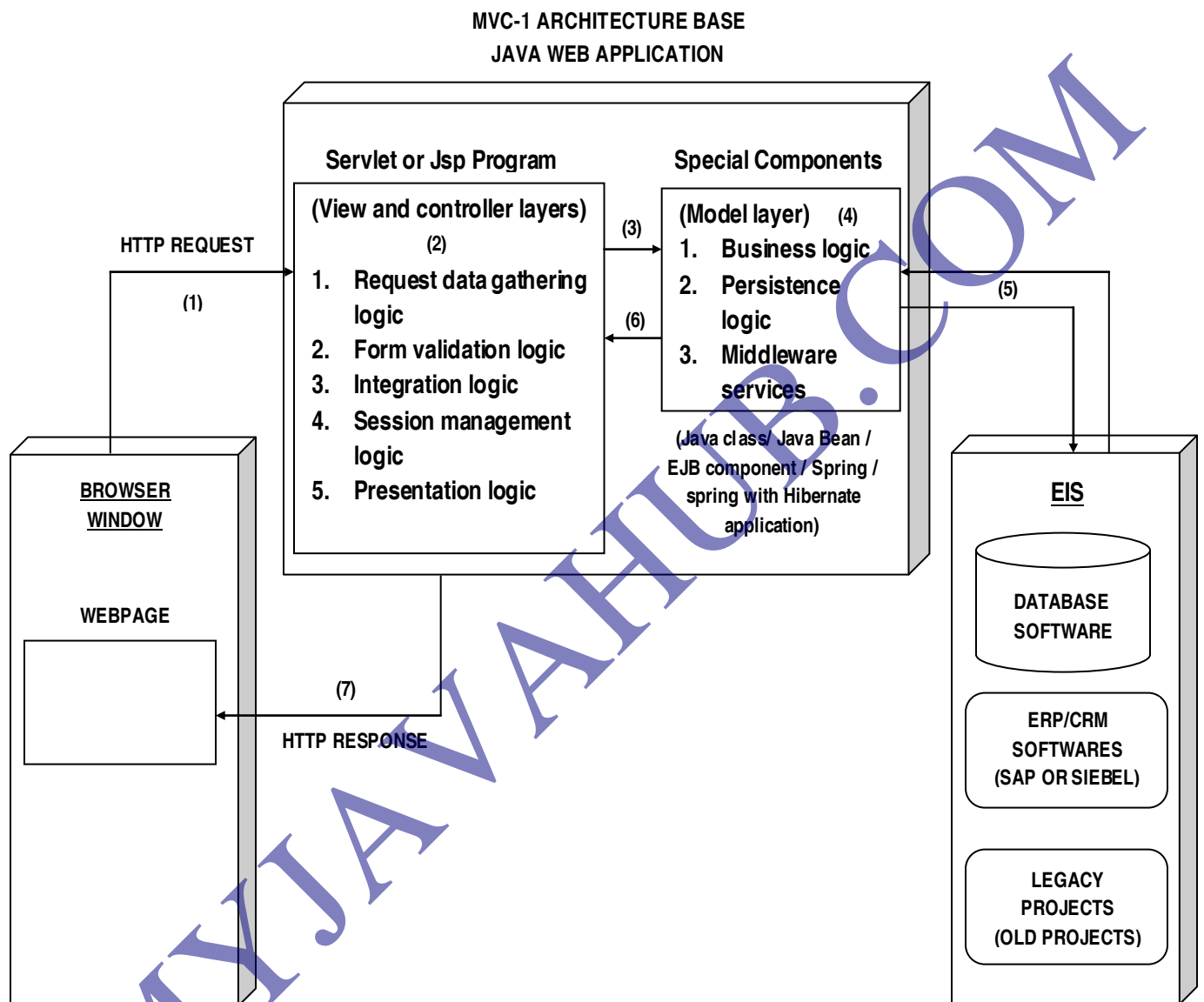
In MVC-1 → Single resource will be taken acting as view layer and controller layer.

In MVC-2 → 3 different resources will be taken in 3 different layers.

In MVC-1 → Same resource program will act as view layer and controller layer.

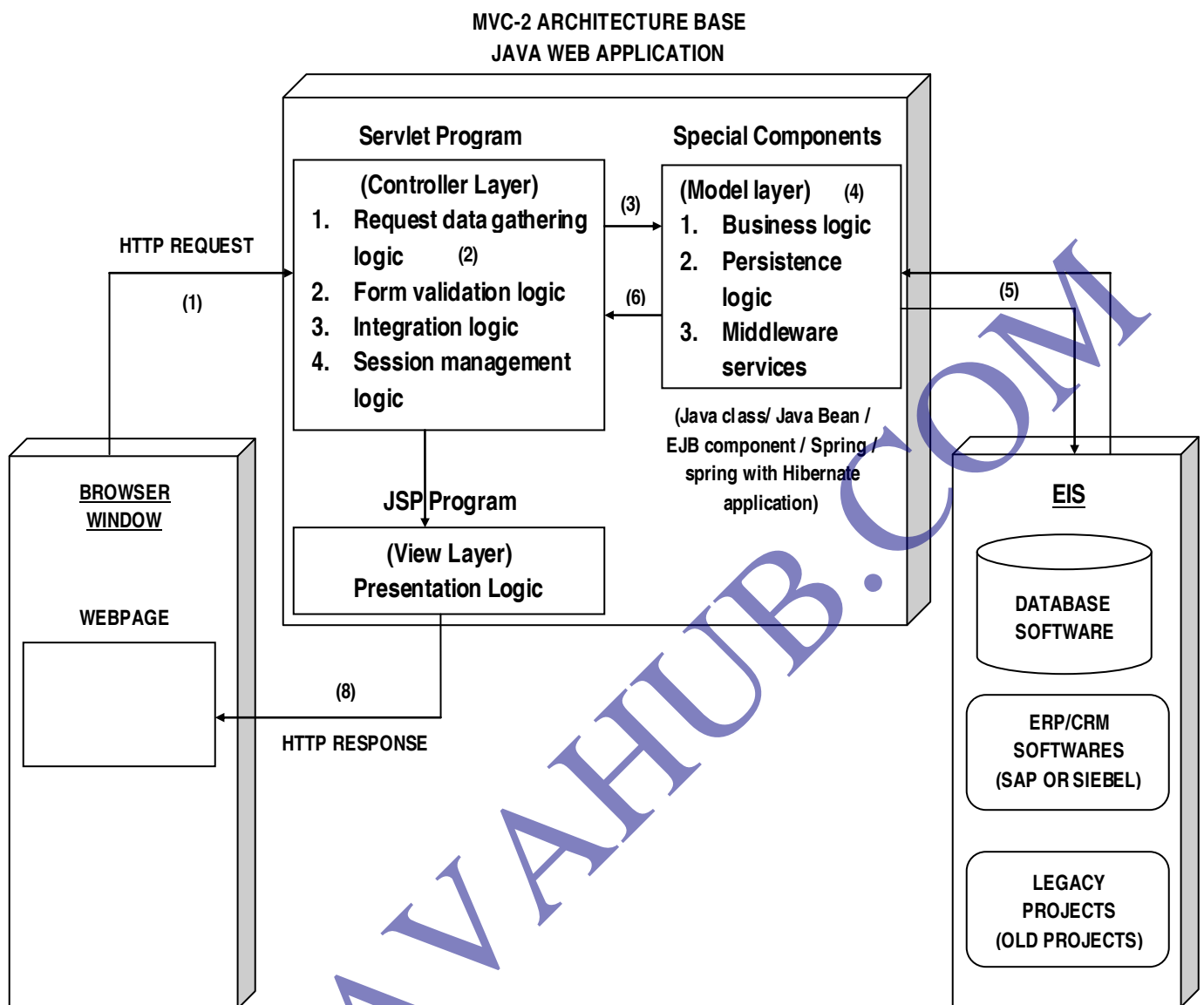
In MVC-2 → In every layer, one separate resource program will be there.

## MVC-1 Architecture/ MVC-1 Approach



In MVC-1 architecture, the model layer is separated from web-resource program that represent both view layer and controller layer. Due to this it gives better and clean separation of logics when compared to Model-1 architecture. But this separation of logics suitable only for medium scale applications, which contains 20-30 web-pages and not suitable for large scale web applications which contains more web pages and which gets more requests then use MVC-2 architecture.

## MVC-2 Architecture/MVC-2 Approach



- In MVC-2 architecture, multiple layers will be there representing multiple logics and giving clean separation of logics.
- In each layer separate programs will be there to develop the logic.
- Design pattern is a set of rules, which come as best solution for re-occurring problems of application development.
- Initially MVC-2 is given as design pattern having set of rules and as best solution to solve the problems of Model-1 and MVC-1 architecture based web-application development. Later these rules have become industry standard to develop large scale web-applications and to create softwares like struts.
- Then onwards people started calling MVC-2 architecture to develop the web-application.



### With respect to the above MVC-2 architecture diagram

(1)	: Browser window gives request to web-application.
(2)	: The controller servlet program traps and takes the request, reads the request data and performs validations.
(3)	: Controller servlet uses integration logic to communicate with model layer resource.
(4 & 5)	: The model layer resource processes the request and also interacts with backend software by using persistence logic.
(6)	: The business logic generated results comes to the controller servlet program.
(7)	: The servlet program uses integration logic to send the results to view layer JSP program.
(8)	: The servlet program uses presentation logic and sends formatted result to browser window as web-page having HttpServletResponse

### Advantages of MVC-2 Architecture

- ☑ There is clear separation of logics because multiple layers are there in web-application development.
- ☑ Modification done in logics of one layer will not affect the logics of another layer.
- ☑ Maintenance and enhancement become easy.
- ☑ Parallel development is possible. So productivity is good.

### Drawbacks of MVC-2 Architecture

- ☒ For parallel development multiple and more programmers are required.
- ☒ Knowledge on multiple technologies is required.

### How parallel development is possible

The project leader divides the team into two parties

- 1) Web-Authors : Takes care of presentation logic and integration logic by using servlet and jsp technologies.
- 2) JEE Developer : Takes care of business logic and persistence logic ( model layer) by using EJB, spring, hibernate and etc technologies.

### Note

Since these two parties can work in parallel, we can say parallel development is possible.

EJB, Spring technologies will give built-in middleware services. This reduces burden on the programmers.

## MVC-2 Architecture Principles

Alone working with multiple technologies and layers the MVC-2 architecture based web-application should follow the following MVC-2 principles.

1. Every layer is designed to maintain specific logic. So, add only those logics in each layer.
2. Every operation in the web-application execution must takes place under the control of controller servlet program.
3. They can be multiple resources in the view layer (JSP), they can be multiple resources in the model layer. But there must be only one servlet program acting as controller.
4. Two JSPs of view layer must not communicate each other directly. They must communicate each other through controller servlet program.
5. View layer resources should not talk with model layer resources directly. They must talk with each other through controller servlet program.

**FAQ) What happens if i changed the roles of technologies that are suggested to use in MVC-2 architecture based web-application development.**

Ans) Yes, we can change their roles but it is not recommended to change their roles. If servlet program is taken as view layer resource

Modification done in source code of servlet program will be affected only after recompilation of servlet program, reload of the web-application where as the modification done in JSP program will be affected without re-compilation and reloading. Since the presentation logic of website changes JSP programs in view layer.

If JSP program is taken as controller of controller layer

Keeping java code in JSP program is against industry standard. Because it kills the readability of JSP program when JSP program is taken as controller it has to communicate with the model layer EJB component or spring application hibernate application for this we need to add java code in JSP program because built-in JSP available for this process. So it is recommended to take servlet program as controller where placing java code is allowed.

If EJB component/Spring Application/Hibernate Application is taken as controller or view layer resources.

The view layer and controller layer resources must be web components like servlets, JSP programs having capability to handle http request and http response. Since all above said components/applications cannot handle http request and http response. They cannot be there in view and controller layers.

If separate servlet program/Jsp program is taken in the model layer

The business logic placed in servlet program/Jsp program becomes specific to that web-application, allows only http clients and doesn't provide implicit middleware services support. To solve all these problems use EJB or spring or spring with hibernate based application in model layer to develop business logic and persistence logic.

### **Framework software**

Framework software is a special software that is developed based on core technologies having capability generate the common logics of application development dynamically and makes application developer to just concentrate on application specific logic.

Framework software provides abstraction layer on core technologies towards application development and makes the application development process by concentrating application specific logics of the application.

### **Struts**

Struts is a web framework software to develop MVC-2 architecture based java web-application having capability to generate the integration logic of controller layer dynamically based on the logics given by the programmer view model layers.

Struts provides abstraction layer on servlet, JSP core technologies and allows to develop MVC-2 architecture based web-application easily and quickly.

**Developing MVC-2 architecture based web-application by directly using core servlet, Jsp technologies. (No web framework software is used)**

- Implement all logics, all layers manually.
- Implement MVC-2 principles explicitly.
- Take care flow of execution manually.

**Developing MVC-2 architecture based web-application by using web framework software like struts.**

- Integration logic of controller servlet comes dynamically.
- Programmer just needs to concentrate only on view, model layer logics.
- Most of the MVC-2 principles will be implemented automatically.
- The web framework software itself takes care of flow of execution.

### **Note**

All web framework softwares are given based on MVC-2 architecture and they allow us to develop only MVC-2 architecture based web-applications.

## Web Framework Softwares

Struts	From Apache Foundation
JSF	From Sun Micro Systems
Webwork / Xwork	From Open Symphony Company
Spring web MVC	From Interface21

### Note

All web framework softwares use servlet/jsp technologies as underlying core technologies

**ORM Framework softwares to develop O-R Mapping persistence logic which is database independent logic**

Hibernate	From Soft tree
OJB	From Apache
TopLink	From Oracle Corporation
iBatis	From Apache
JDO	From Adobe

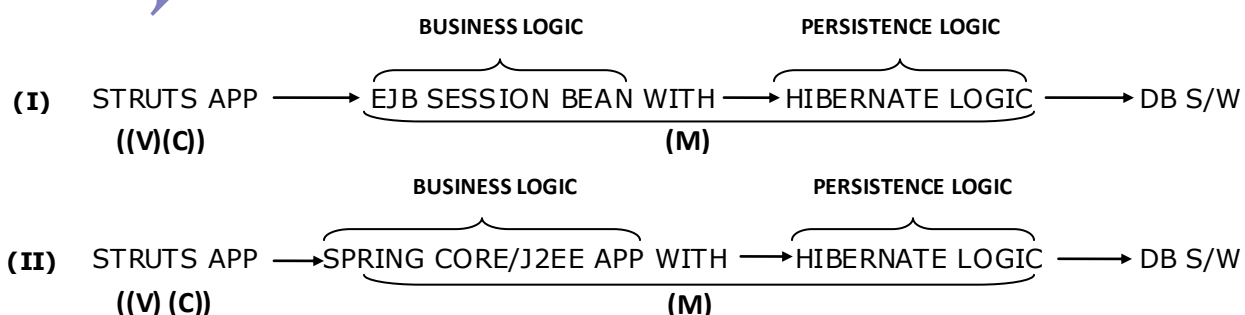
### Note

All ORM framework softwares internally use JDBC as underlying core technology.

**Java/Jee framework softwares (to all kinds of java/jee application including web-application and O-R Mapping persistence logic) in Framework software.**

Spring framework software from Interface21

Spring framework software uses multiple Jsp module, Jee module technologies as underlying core technologies.



## Struts Framework Software

Type	: Java based Web Software
Version	: 1.x (compatible with JDK 1.4 +) 2.x (compatible with JDK 1.5 +)
Vendor	: Apache Foundation
Software type	: Open source software
Creator	: Craig mc-Challon
To download software	: Download as zip file from <a href="http://www.apache.org">www.apache.org</a> website and Extract the zip file to install the software.
For help	: <a href="http://www.forum.apache.org">www.forum.apache.org</a> website
For good online tutorial	: <a href="http://www.roseindia.net">www.roseindia.net</a> website
For good articles	: <a href="http://www.onjava.com">www.onjava.com</a> , <a href="http://www.javabeat.net">www.javabeat.net</a> , <a href="http://www.precisejava.com">www.precisejava.com</a>
Reference books	: For Struts 1.x-Jakarta struts from O' reilly press For Struts 2.x-Blackbook of struts

### Struts installation gives the following resources

- Struts 1.x software gives us basic framework software
- Set of Jsp tag libraries (5 Nos)
- Source code of struts software
- Docs on struts API and jsp tag libraries
- Additional plug-ins like validator plug-in and tiles plug-in
- Jar files representing struts API and dependent APIs
- A JSP Tag library is that contains set of readily available JSP tags. The Jsp technology supplied built-in tags are not sufficient to make Jsp program as java code less. Jsp programs. To solve this problem struts software supplies 5 number of Jsp tag libraries to make the Jsp programs of struts application a java code less Jsp programs.
- A plug-in is a patch or small software/ software application which can enhance the functionalities of existing software or software application. Plug-in comes as jar file.
- Struts home\apps\struts-cookbook-1.3.8.war file contains various example scenarios that can be developed by using struts 1.x.
- Struts home\lib\struts-core-1.3.8.jar file represents the whole struts API and for this jar file multiple dependent jar files are there.

- If classes of first.jar file are using the classes of second.jar file directly or indirectly. Then second.jar file is called as depended jar file to first.jar file.
- In struts application the controller servlet is pre defined http servlet based servlet program whose name is ActionServlet. This ActionServlet integration logic will be generated dynamically by struts framework software based on the resource and logics given in view, model layers.
- Developing the struts application is nothing but developing the normal web-application by adding the struts API support in the web-resource programs of that java web-application. For this, the struts API related jar files must be placed in classpath and must be also placed in WEB-INF/lib folder of that struts application.
- Using struts framework software, we can develop only MVC-2 architecture based web-applications. We cannot develop other architecture based web-applications.

### **Resources of struts 1.x application development**

1. JSP programs (view layer resources)
2. Action Servlet (Built-in controller servlet program) (controller layer resource)
3. web.xml (Deployment descriptor file of web-application)
4. FormBean class (Java class) (controller layer resource)
5. Action Class (Java class) (controller layer resource)
6. Action Forwards (xml entries) (controller layer resource)
7. Struts Configuration file (xml file) (controller layer resource)

### **JSP Programs**

These programs contain presentation logic to generate user interface for end users and to format the result given by model layer resource. It is always recommended to develop this Jsp programs as java code less Jsp programs.

For this use the following Jsp Tags

- a) Jsp tags of built-in Jsp technology
- b) Jsp tags of struts Jsp tag library
- c) Jsp tags of JSTL
- d) Custom Jsp tags

### **Action Servlet**

It is a built-in controller servlet program of every struts application. The integration logic in this, servlet program will be generated dynamically based on the rules and guidelines given in struts configuration file. This integration logic controls and decides the flow of execution in struts application.

**web.xml file**

- In the xml file ActionServlet configuration will be there with special url-pattern like <directory-match>, <extension-match>
- Even though ActionServlet is built-in servlet its configuration in web.xml is mandatory.
- The web.xml file also contains Jsp tag libraries configuration.

**FormBean class**

- It is a JavaBean class having capability to store the form data of the form page and also control form validation logic to validate the form data.
- The JavaBean class that extends “org.apache.struts.action.ActionForm” class is called as FormBean class.
- ActionForm is abstract class with no abstract methods.
- A JavaBean is Java class that contains getter and setter methods.

**FAQ) What is need of FormBean class in struts application?**

Ans: According to MVC-2 architecture ActionServlet is responsible to read form data and to validate the form data. In struts application ActionServlet acts as controller servlet. Since it is built-in servlet, programmer cannot place his choice of form validation logic in that servlet program.

To solve this problem, ActionServlet writes the received form data to a java class supplied by the programmer and allows to the programmer to validate form data while writing form validation logic. That java class is nothing but FormBean Class.

**Action Class**

- It is a java class which is extended from “org.apache.struts.action.Action” class is called as struts Action class.
- It can acts as model layer resource by having the logic to communicate with other model layer resource like EJB component, spring application, spring with hibernate application and etc.

**FAQ) What is the need of struts Action Class in struts application?**

Ans: In small scale struts application, struts Action class contains business logic directly and acts as model layer resource. Generally in large scale struts application EJB component or spring application will be chosen as model layer resource. According to MVC-2 architecture, the controller servlet called Action Servlet to communicate with these model layer resources. But programmer cannot add these logics in ActionServlet because ActionServlet is built-in servlet. To overcome this problem, he place that logic in struts Action Class and he make struts Action class as controller layer resource. Every request coming to Action servlet will also goes to struts Action class.



## Action Forwards (xml.file)

These are xml entries in struts configuration file pointing to the result Jsp programs or other Jsp programs of view layer using which the results given by struts Action class will be formatted by executing the presentation logic.

## Struts.cfg.xml

Any <filename>.xml can act as struts configuration file. This file is heart of the struts application and based on this file based configuration the integration logic of ActionServlet will be generated dynamically.

This file contains the following configurations

- ✓ FormBean class configuration
- ✓ Struts Action class configuration
- ✓ Action Forwards configuration
- ✓ Some plug-ins configuration and etc.

## Note

Making the underlying software recognizing the resources by providing details of the resource is called as resource configuration.

- Action servlet looks to take struts-config.xml of WEB-INF folder as default struts configuration file
- If any other name is taken for struts configuration file, that must be specified explicitly in web.xml file during the configuration of ActionServlet.
- The underlying server reads the web.xml file entries, ActionServlet reads struts configuration entries. So we can't take single xml file having the resources configuration of both "web.xml" and "struts-cfg.xml" file.

## Note

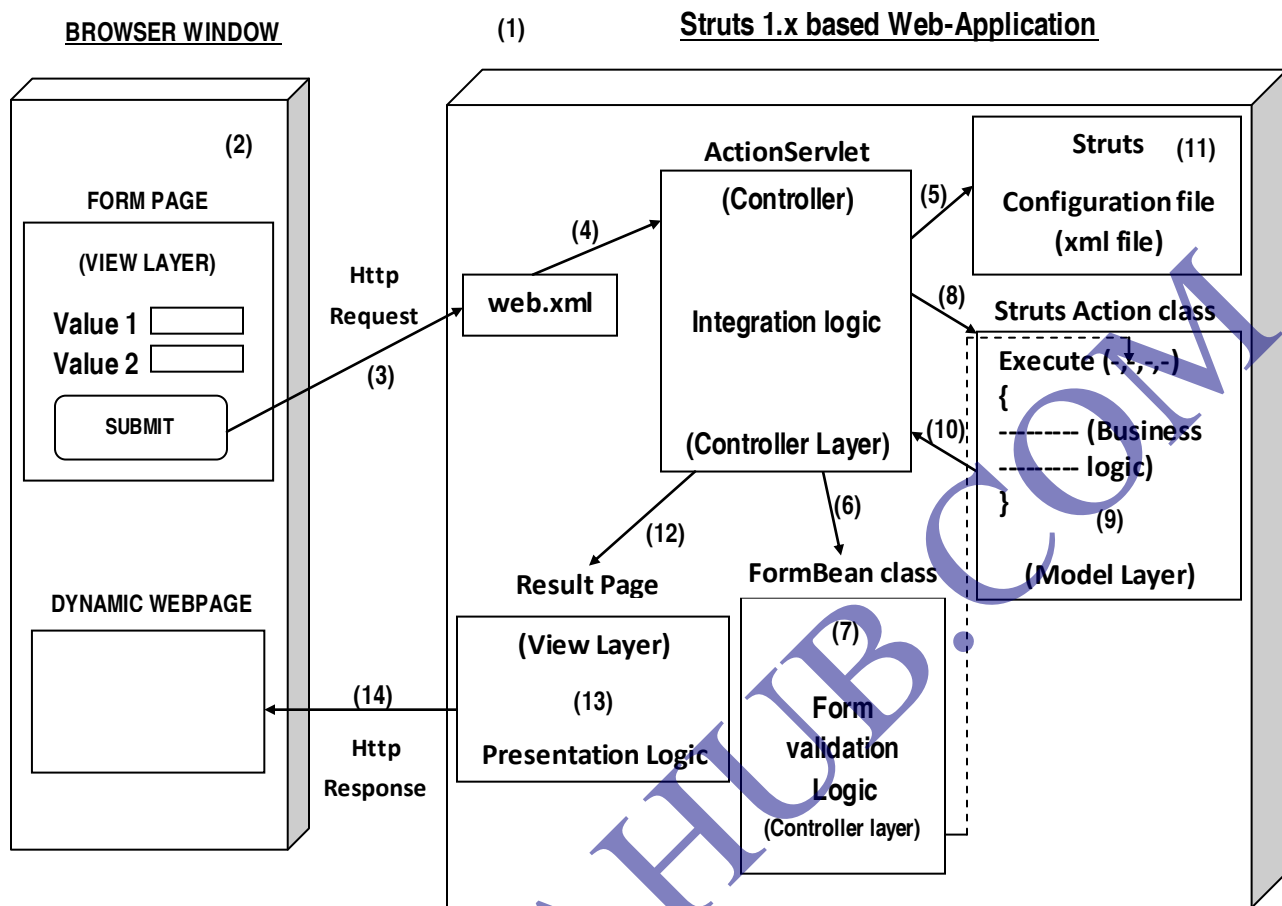
In the above given list of resources of struts application, except ActionServlet the remaining all resources must be developed by programmer explicitly (manually).



**FAQ) Can you explain struts 1.x architecture.**

**(OR)**

**Can you explain flow of execution in struts 1.x web-application?**



In the above diagram, struts Action class is taken as model layer resource by keeping business logic directly.

With respect to above diagram

(1)	Programmer deploys struts application in web server as web-application.
(2)	The end user launches form page of struts application in a browser window.
(3)	End user submits the form page by filling up the form.
(4)	ActionServlet traps and takes the form page generated request based on its configuration done in web.xml file.
(5)	ActionServlet uses the struts configuration file entries to decide the FormBean and Action class using which the form page generated request should be processed.
(6)	ActionServlet creates or locates the struts Action class object and writes the form data to it.
(7)	The form validation logic of FormBean class validates the form data of form page.

(8)	ActionServlet creates or locates the struts Action class object.
(9)	ActionServlet calls execute method of struts Action class and process the request by executing business logic.
(10)	Execute method returns the controller and result to ActionServlet.
(11)	ActionServlet uses the ActionForward configuration of struts configuration file to decide the result page of struts Action class.
(12)	ActionServlet transfers the control to result page having result given by Action class.
(13)	The presentation logic of result page formats the result.
(14)	The formatted result goes to browser window from result page as http response in the form of dynamic web-page.

### Note

Every FormBean will be linked with one struts Action class. So the FormBean class object is visible in the Action class as the parameter of execute method.

In the creation of struts software multiple technologies are used. To use these technologies more effectively they have implemented some design patterns. These design patterns are called implicit design patterns of struts software.

### Some important implicit design patterns are

- Singleton Java class (ActionServlet)
- Front Controller (ActionServlet)
- MVC-2 (The whole struts Application)
- V.O Class/D.T.O Class (FormBean class)
- I.O.C (Inversion Of Control)/Dependency Injection (DI)
- Abstract Controller (RequestProcessor)

All these design patterns will be there as implicit design patterns in struts application.

### Singleton Java class

The Java class that allows creating only one object per JVM is called as "Singleton java class". Instead of creating multiple objects for a java class having same data, it is recommended to create only one object for that class and use it for multiple times. This avoids the memory wastage.

Most of the JDBC Driver classes are given as singleton java classes because only one object will be created for JDBC Driver class even though it is used in multiple applications that run in a single JVM.

In struts application the ActionServlet is given as Singleton java class that means the servlet container creates only one object for ActionServlet even though huge number of requests are given to struts application.

Every servlet program is a “Single Instance Multiple Threads” Component. That means if multiple requests are given to servlet program, only one object of servlet class will be created and multiple threads will be started on that object representing multiple requests.

**FAQ) ActionServlet is a built-in servlet class and every servlet is a single instance multiple threads component by default. Then what is the need of giving ActionServlet as singleton java class by struts software creators.**

Ans: There are some servers in the market who creates multiple objects for our servlet, when huge numbers of requests (300+) are given to that servlet class simultaneously or concurrently. This is violation of servlet specification which says servlet component should be single instance multiple threads component.

When struts application is deployed in the above said servers having the above scenario. In order to see only one object for ActionServlet irrespective of all conditions. Struts software creators have made ActionServlet as Singleton java class

### **Different types of Url-Patterns**

According to Sun Microsystems, there are 3 types of url-patterns which can be assigned to servlet program.

1. Exact Match
2. Directory Match
3. Extension Match

#### **Exact Match**

The exact match url-pattern must begin with ‘ / ’ symbol and should not contain ‘ \* ’ symbol.

Ex:   <url-pattern>/test1</url-pattern>  
       <url-pattern>/test1.xyz</url-pattern>  
       <url-pattern>/test1/test2</url-pattern>  
       <url-pattern>/test1/abc.do</url-pattern>

#### **Directory Match**

This url pattern must begin with ‘ / ’ symbol and must end with ‘ \* ’ symbol.

Ex:   <url-pattern>/x/y/\*</url-pattern>  
       <url-pattern>/x/\*</url-pattern>  
       <url-pattern>/test1/test2/\*</url-pattern>  
       <url-pattern>/test1/abc.do/\*</url-pattern>

## Extension Match

This url-pattern must begin with ' \* ' symbol and must end with extension letter or extension word.

Ex: `<url-pattern>*.abc</url-pattern>`  
`<url-pattern>*.do</url-pattern>`  
`<url-pattern>*.c</url-pattern>`

## Note

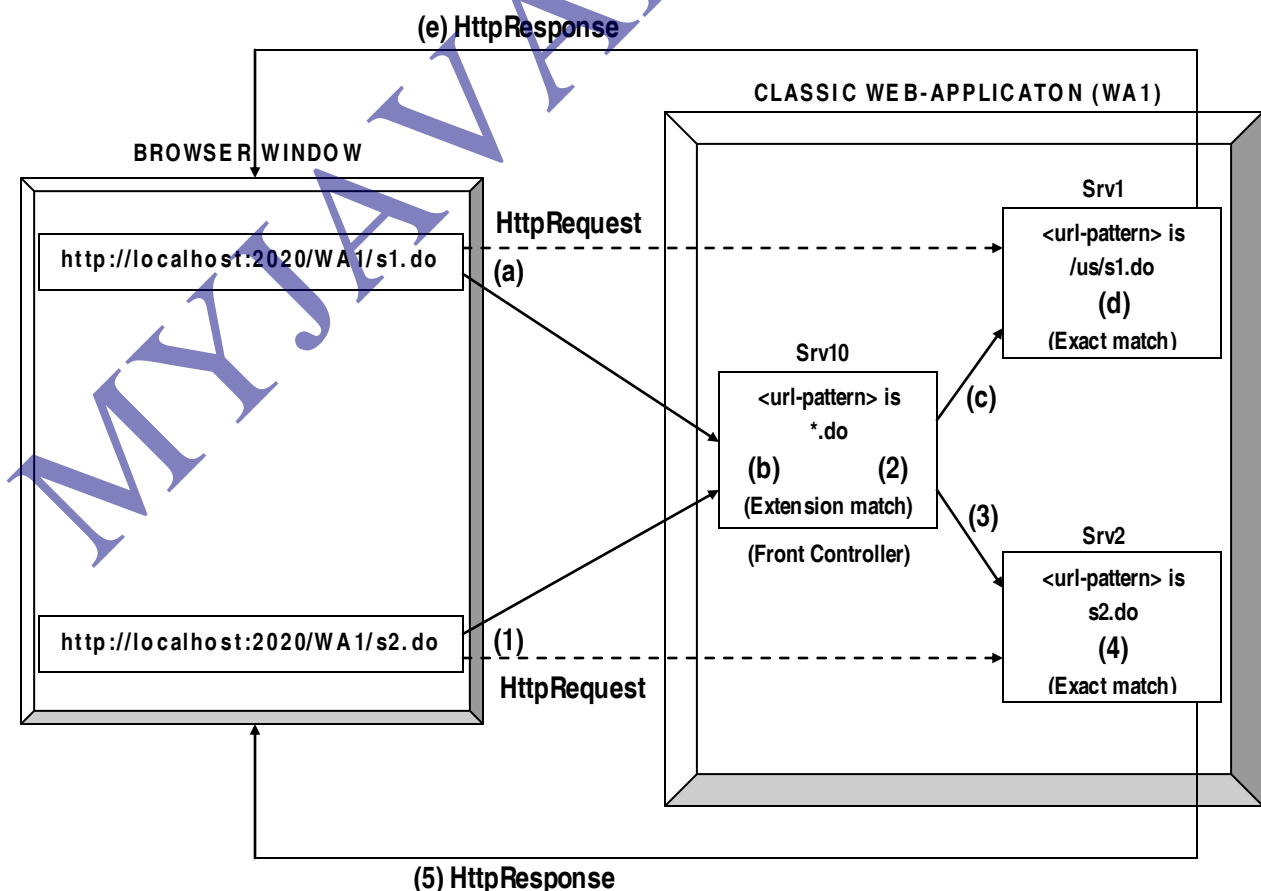
You cannot frame url-pattern for servlet program without following those 3 styles of above url-patterns.

Ex: `<url-pattern>{/x/y/*}.do</url-pattern>`  
 Invalid url-pattern formation

## Front Controller

- ✓ A special web-resource program in web-application that is capable of trapping and taking request coming to other web-resource programs of web-application is called as Front Controller.
- ✓ The Front Controller contains the common and global pre-requisite process like authentication logic, authorization logic and etc.
- ✓ The servlet program or Jsp program becomes front controller only when it contains extension Match or Directory Match url-pattern.

## Request Trapping Process In Classic Web-Application (Non-Struts Application)



- ✓ Front controller is just capable of trapping and taking web resource program of web-application that means it cannot trap the response of web resource program of the web-application where as servlet filter program is capable of trapping and taking both the request and response of web-resource programs of a web-application.
- ✓ In struts application ActionServlet is Front Controller servlet program, so the programmer must configure ActionServlet program in web.xml file by having either "Extension Match" or "Directory Match" url-pattern.

### FAQ) Why an ActionServlet should be taken as Front Controller in our struts application.

Ans: The java class of web-application cannot take http request directly.

In struts application, Struts Action class is not servlet program and it is an ordinary Java class. So, struts Action class cannot take http request directly from the clients but every request given by struts application must execute the struts Action class once. So, we make ActionServlet to trap the request coming to struts Action classes. In this process, the struts ActionServlet must be taken as Front Controller.

### Note

As Front Controller, it traps the request coming to struts application and as controller of MVC-2 architecture based struts application it also traps the results of Action Classes.

The struts application developer generally configures the pre-defined ActionServlet program in web.xml file as shown below.

### web.xml

```
<web-app>
```

```
<servlet>
```

```
<servlet-name>action</servlet-name>
```

```
<servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
```

```
<!--Config init param containing the name and location of struts-cfg file-->
```

```
<init-param>
```

```
<param-name>config</param-name>
```

```
<param-value>WEB-INF/MyCfg.xml</param-value>
```

```
</init-param>
```

The name and location of struts configuration file

```
<!--Enables pre-instantiation and pre-initialization on ActionServlet-->
```

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

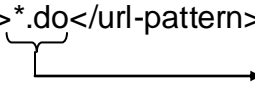
```
</servlet>
```

Load on startup priority value

```

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

```



Extension Match url-pattern to configure  
ActionServlet as Front Controller.

### Note

Any extension word or letter can be there in the url-pattern of ActionServlet.

Ex: '\*.c', '\*.sathya', '\*.abc' are allowed

- ✓ "config" is the pre-defined init parameter of ActionServlet expecting, the technical input value called the name and location of struts configuration file. If that is not specified, the ActionServlet takes, "struts-config.xml" file of WEB-INF folder as default struts configuration file.
- ✓ To supply non-technical input values to servlet program from browser window by end-user, use request parameters. (Form page form data like name, address and etc).
- ✓ To supply technical input values to servlet program from web.xml file by programmer use servlet init parameters.
- ✓ Since the name and location of struts configuration file is technical input value expected by ActionServlet, we must supply them as "config" init parameter values.
- ✓ When <load-on-startup> is enabled on ActionServlet the servlet container creates and initializes our ActionServlet class object either during server startup or during the deployment of struts Application. This process minimizes response time of first request and equalizes the response time of first request with other than first request coming to ActionServlet.
- ✓ When multiple servlet programs of web-application (classic web-application) are enabled with <load-on-startup> in which order there objects will be created during server startup or during the deployment of the web-application will be decided based on <load-on-startup> priority value. High value indicates low priority, Low value indicates high priority. Negative value ignores the enabled <load-on-startup>

### WA1 (as classic web-application) <load-on-startup> (<L-O-S>) values priorities

Web-Resource Program	<load-on-startup> Priority value	Priority order
Servlet1	1	II
Servlet2	10	III
Servlet3	0	I
Servlet4	-1	Ignores <L-O-S>

- ❖ Since ActionServlet is the only one servlet in struts based web-application. So we can use any positive number or zero can be taken as priority value while enabling <load-on-startup> on ActionServlet.

### **D.T.O or V.O (Data Transfer Object or Value Object)**

While transferring multiple values from one layer to another layer, instead of transferring them one-by-one for multiple times. It is recommended to combine all these values into a single java class object and send that object from source layer to destination layer. This reduces the roundtrips between layers. In this process the class of that single object is called as V.O class/D.T.O class.

In struts application the multiple values of form page that comes to ActionServlet (controller layer) will go to struts Action class as single object data of FormBean class. So, FormBean class in struts application is called as V.O class or D.T.O class.

FormBean class object is visible in struts Action class as the parameter of execute (-,-,-) method.

### **Dependency Lookup**

If resource spends time to search and gather its dependent values, then it is called as Dependency Lookup.

#### Real Life Example

If the student gets his course material after demanding for it is called Dependency Lookup.

#### Technical Example

The way we get JDBC DataSource object from JNDI registry software is called as Dependency Lookup.

#### **Note**

In Dependency Lookup, the resource pulls the dependency values.

### **Dependency Injection (DI) / Inversion of Control (IOC)**

If the underlying container software or framework software or server software or special resource is dynamically assigning the required dependent values to the resource, then it is called as Dependency Injection.

#### Real Life Example

If an institute dynamically assigns course material (dependent value) to students immediately after registration is called as Dependency Injection.

#### Technical Example

The way ActionServlet dynamically assigns the form data of form page to FormBean class is called as Dependency Injection.

### **MVC-2**

Developing web-application with multiple layers and by following set of principle comes under MVC-2 design pattern.

In struts application, we can see multiple layers and MVC-2 principles implementation.

### **Abstract Controller**

- The helper class for Front Controller servlet that minimizes burden of Front Controller servlet and allows customizing the behavior of Front controller servlet from outside is called as Abstract Controller.



- In struts application ActionServlet is Front Controller servlet program. It is internally uses a pre-defined helper java class called “org.apache.struts.action.RequestProcessor” class to perform the integration logic related operations.
- We can also use this class to customize the behavior of ActionServlet. So, this RequestProcessor class is technically called as Abstract Controller class.

### Key Points

- ❖ Managing the whole life cycle of ActionServlet is the responsibility of Servlet container.
- ❖ Managing the whole life cycle of FormBean class and Struts Action class is the responsibility of ActionServlet.
- ❖ Managing the life cycle of certain resource is nothing but taking care of all operations related to that resource from object birth to object death.
- ❖ Servlet container creates and manages ActionServlet class object. Similarly ActionServlet is responsible to create and manage objects of FormBean, Action classes.

### Naming Conventions in Struts Application

If “x” is file name

X	:Jsp file name
XForm	:Form Bean class name
XAction	:Struts Action class name
Xresult	:Result jsp file name

Example

Register.jsp	:Form page name
RegisterForm	:Form Bean class name
RegisterAction	:Struts Action class name
RegisterResult.jsp	:Result jsp file name

### Note

Naming conventions are suggestions to follow but not the rules.

### Understanding Struts Configuration file Configurations.

- The java class that extends from pre-defined ActionForm class is called as Form Bean class.
- Every Form Bean class must be configured in Struts Configuration file having logical name.

### RegisterForm.java (Form Bean class)

```
public class RegisterForm extends org.apache.struts.action.ActionForm
{
```

```
    // form bean properties.
```

```
    String uname; }
```

```
    String pwd; }
```

```
    // write getXxx() and setXxx(-) methods
```

```
    public void setUname(String unname)
```

```
    {
```

```
        this.uname=unname;
```

These Form Bean property names and count must match with the form component (like textbox/radio button etc) names and count of form page.



```

    }
    public void setPwd(String pwd)
    {
        this.pwd=pwd;
    }
    public string getUname()
    {
        return uname;
    }
    public string getPwd()
    {
        return pwd;
    }
}

```

### Note

This Form Bean class related Form page (jsp/html) contains two form components (like text boxes) having names uname, pwd.

### In Struts Cfg file

```

<form-beans>
    <form-bean name="rf" type="RegisterForm"/>
</form-beans>

```

Java class name that is taken as form bean class

Form bean logical name

**Note:** Every form bean class of struts application will be identified with its logical name, because this logical name becomes the object name when ActionServlet creates object for form bean class.

### Action Class

- The java class that extends from "org.apache.struts.action.Action" class is called as Struts Action class.
- Every Struts Action class must be configured in struts configuration file having "action path".
- The process of linking Form Bean with Action class linking result page with Action class through ActionForward configurations is technically called as Action Mapping operation.
- Every Struts Action class will be identified through its action path.

### RegisterAction.java (Struts Action Class)

```

public class RegisterAction extends org.apache.struts.action.Action
{
    {
        public ActionForward execute(-,-,-);
        -----;
        -----;
        //Returns an ActionForward class object
    }
}

```

## In Struts Configuration file

```

<struts-config>
(1) {
    <form-beans>
        <form-bean name= "rf" type= "RegisterForm"/>
    </form-beans>
(2) {
    <action-mappings>
        <action path= "/reg" type= "RegisterAction" name= "rf">
            <forward name= "success" path= "/result.jsp"/>
        </action>
    </action-mappings>
}
</struts-config>

```

Diagram annotations: (3) points to "rf", (4) points to "RegisterForm", (5) points to "/reg", (6) points to "RegisterAction", (7) points to "rf", (8) points to "success", (9) points to "/result.jsp", and (10) points to the closing tag of the forward element.

(1) Form bean class configuration.

(2) Struts Action class configuration.

(3) name= "rf" represents form bean logical name.

(4) type= "RegisterForm" represents fully qualified java class that acting as form bean class.

(5) Path= "/reg" represents action path of Struts Action class. The '/' symbol in front of the path name is mandatory.

(6) type= "RegisterAction" represents fully qualified java class that acts as Struts Action class.

(7) name= "rf" represents Form bean logical name, here it is used to link form bean with struts Action class.

(8) name= "success" represents ActionForward logic name.

(9) path= "/result.jsp" represents result page of Struts Action class.

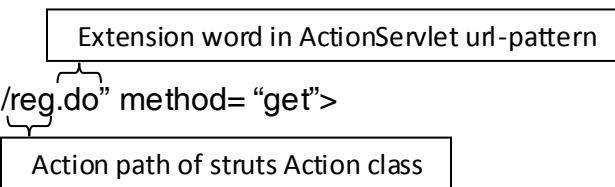
(10) result page config (ActionForward config)

## Key Points

- A Form bean is identified through its logical name.
- A Struts Action class is identified to its action path.
- The result page of ActionForward configuration will be identified through its logical name.
- To make Form page generated request to be trapped by ActionServlet, the extension word or letter of ActionServlet url-pattern (like ".do" or "\*.do") must be there in the Action url of form page (url kept in the Action attribute of <form> tag).

After trapping the form page generated request the ActionServlet send that request to an appropriate Action class.

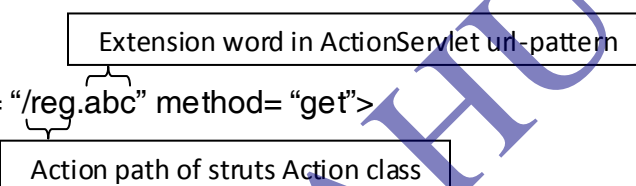
**A typical form page that targets request to the above RegisterAction class through ActionServlet looks like this.**


  
 form action= "/reg.do" method= "get">

```
username:<input type= "text" name= "uname"><br/>
password:<input type= "password" name= "pwd">>br/>
<input type= "submit" value= "send">
</form>
```

In the above situation the ActionServlet (AS) url-pattern must be taken as "\*.do".

In the below situation the ActionServlet (AS) url-pattern must be taken as "\*.abc".


  
 form action= "/reg.abc" method= "get">

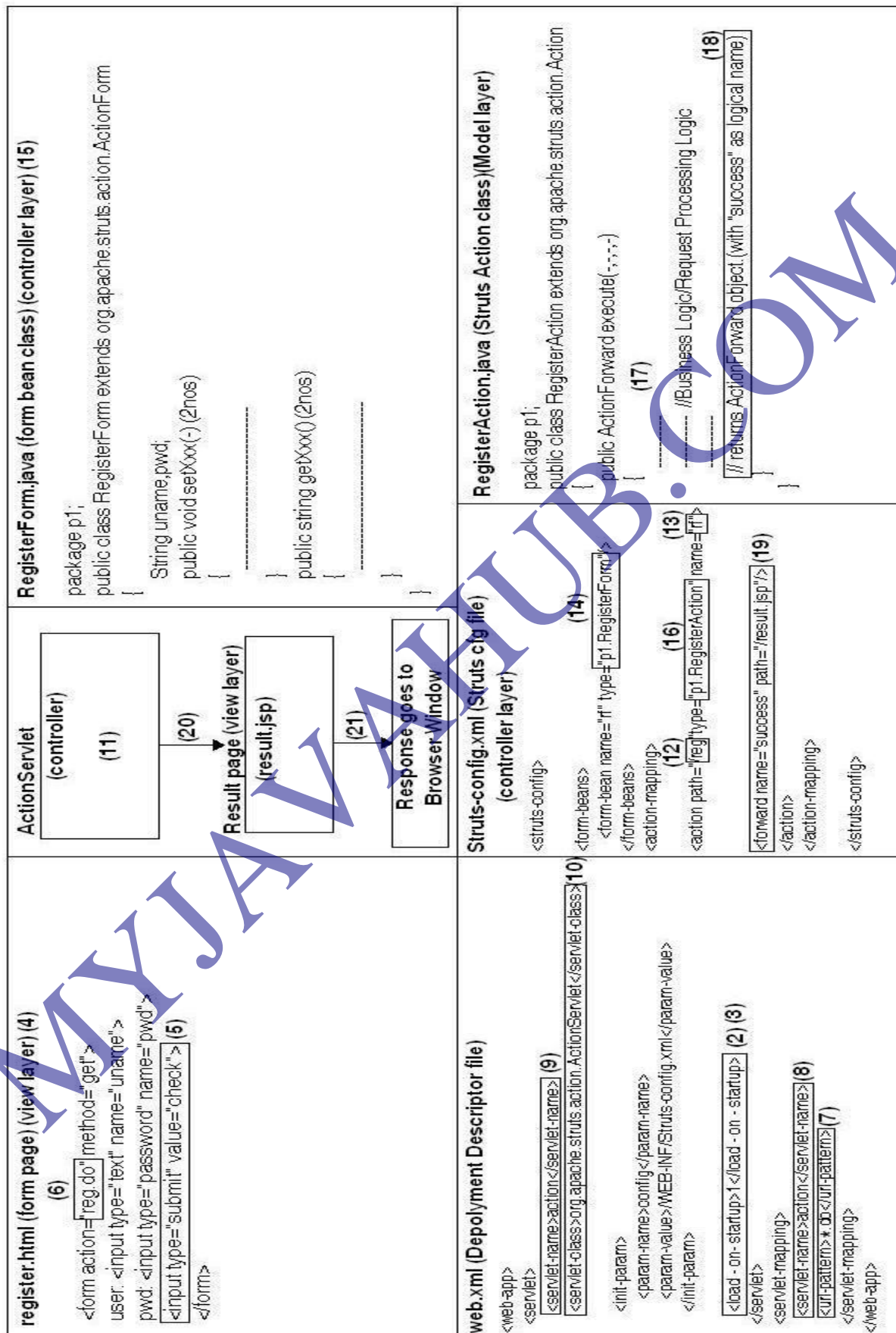
```
username:<input type= "text" name= "uname"><br/>
password:<input type= "password" name= "pwd">>br/>
<input type= "submit" value= "send">
</form>
```

### Note

To make form bean, Action classes visible and instantiatable by ActionServlet they must be taken as public classes.

Understanding flow of execution in struts application by using the sample code of the struts application resources.

(1)



**According to the above figure.**

1)	With respect to above sample code programmer deploys the struts application in the web server. In this process the web server reads and verifies the web.xml file entries.
2)	Because of load-on-startup the servlet container creates ActionServlet object either during server startup or during the deployment of the web-application.
3)	ActionServlet gets the name and location of struts configuration file and also reads, verifies the struts configuration file entries (ActionServlet gets name and location of struts configuration file from the config init parameter of ActionServlet configuration done in web.xml file.
4)	The end-user launches the form page on the browser window (register.html)
5)	End-user submits the form page having form data.
6)	The form page generates request url based on Action url having "reg.do" in it.
7,8,9,10)	Since ActionServlet url pattern is "*.do" and form page generated request url is having "do" (from reg.do) so the ActionServlet traps and takes the form page generated request.
11)	ActionServlet uses struts configuration file entries to decide the form bean and struts Action class using which the request can be processed. (form page generated request)
12)	ActionServlet uses "reg" of request url to search for a struts Action class configuration in struts configuration file whose Action path is "/reg".
13)	ActionServlet uses name attribute value of <action> to get form bean name (name="rf")
14)	ActionServlet uses the above form bean logical name to known form bean class name and to create or locate that form bean class object. (RequestFrom)
15)	ActionServlet calls setXxx(-) of form bean class to write the form data of form page to form bean class object.
16)	ActionServlet gathers struts Action class name (RegisterAction) and creates or locates that Action class object.
17)	ActionServlet calls execute(-,-,-) method of struts Action class.
18)	execute(-,-,-) method returns ActionForward object to ActionServlet with logical name (success)
19)	ActionServlet uses the "success" logical name based ActionForward configuration and gets the result page of Action class. (i.e Result.jsp)
20)	ActionServlet forwards the control to the result page (result.jsp)
21)	The presentation logic of result page formats the result and the formatted result to browser window as response in the form of web page.

## Understanding the JSP Tag Libraries

- JSP Tag Library is a library that contains set of JSP tags.
- Each JSP Tag Library contains one '.tld' file having description and configuration of JSP tags like tag name, attribute name, possible values of attributes, Tag Handler class name and etc.
- The java class that defined the functionalities of JSP technology by extending from "javax.servlet.jsp.tagext.TagSupport" class is called as JSP Tag Handler class.

### Procedure to develop and use custom JSP Tag libraries.

#### Step-1

Design your JSP Tag Library

Sathya JSP Tag Library

<ABC>

<XYZ>

#### Step-1

Develop Tag Handler classes by defining the functionalities of JSP Tags and save them in WEB\_INF/classes folder of web application.

ABCTag.java

```
public class ABCTag extends TagSupport
{
    -----;
    -----;
}
```

XYZTag.java

```
public class XYZTag extends TagSupport
{
    -----;
    -----;
}
```

#### Step-3

Prepare 'tld' file having the configurations of JSP tags and other tag handler classes

WEB\_INF/sathya.tld (xml content)

<ABC>-----→ABCTag.class

<XYZ>-----→XYZTag.class

Other configurations regarding tags will also be there.

**Step-4**

Configure JSP Tag Library in the web.xml file of web application having "taglib uri".

**Note**

Every JSP Tag Library that is configured in web.xml file will be identified with its "taglib uri".

In web.xml

```
<web-app>

  <taglib>

    <taglib-uri>demo</taglib-uri>

    <taglib-location>/WEB_INF/sathya.tld</taglib-location>

  </taglib>

</web-app>
```

**Step-5**

Use the JSP tags of custom tag library in the JSP programs of web-application.

Test.jsp

```
<%@taglib uri="demo" prefix="st"%>

<st:ABC/>

<st:XYZ/>
```

Output of ABCTag.class execution goes to browser window as the output of <st:ABC/> tag.

Steps 4 & 5 indicates utilization of tag libraries.

In the above code (a) to (h) indicates the flow of execution related to <st:ABC>

- Every tag of JSP tag library must be utilized along with prefix. Prefix is very useful to differentiate tags when the tag libraries utilized in the same JSP program are having same name.
- Struts software gives 5 number of pre-defined JSP Tag Libraries to help the programmer to develop the JSP programs of struts application as Java code less JSP programs. So the struts software supplies pre-defined tag handler classes, tld files.
- While working with pre-defined JSP Tag Libraries we need to gather and arrange tld files and the jar files that represents tag handler classes.

Tag Library Name	tld file	Recommended prefix	Jar files having Tag Handler classes and tld files.
Html tag library	Struts-html.tld	Html	struts-taglib-1.3.8.jar
Bean tag library	Struts-bean.tld	Bean	struts-taglib-1.3.8.jar
Logic tag library	Struts-logic.tld	Logic	struts-taglib-1.3.8.jar
Nested tag library	Struts-nested.tld	Nested	struts-taglib-1.3.8.jar
Tiles tag library	Struts-tiles.tld	tiles	struts-tiles-1.3.8.jar

- The struts supplied JSP Tag libraries must be used only in struts application that means they cannot be used in regular java web applications.

### Produce to use struts supplied Jsp Tag Library (Html Tag Library) in our Struts Application.

#### Step-1

Gather information about the JSP tags of Html Tag Library through documentation.

#### Note

Refer struts home\docs\struts-taglib\index.html file

#### Step-2

Gather 'tld' file of html tag library (Struts-html.tld) by extracting Struts-tag lib-1.3.8.jar file and keep that file in WEB-INF folder of struts Application.

#### Step-3

Keep the html tag lib related jar file (Struts-taglib-1.3.8.jar) in WEB-INF/lib folder of struts application.

#### Step-4

Configure html tag library in web.xml file having taglib uri.

#### In web.xml

```
<web-app>
-----
-----
<taglib>
  <taglib-uri>demo1</taglib-uri>
  <taglib-location>/WEB-INF/Struts-html.tld</taglib-location>
</taglib>
</web-app>
```



**Step-5**

Use the tags of html-taglibrary in the JSP programs of struts application.

**In register.jsp**

```
<%taglib uri= "demo1" prefix= "ht" %>
```

```
<ht:form action= "reg.do" method= "get">
```

optional

```
User name:<ht:text property= "uname"/><br>
```

```
Password:<ht:password property= "pwd"/><br>
```

```
<ht:submit value= "Check Details"/>
```

```
</ht:form>
```

The above form page is developed by using the struts supplied JSP tags.

**FAQ) What is the advantage of working with Struts supplied Html Tag Libraries when compared to traditional Html Tags**

Traditional Tags	Struts Supplied Html Tag Library
1) Extension word in form page action url must be placed manually. <pre>&lt;form action= "reg.do"/&gt;</pre> <p style="text-align: center;">----- mandatory</p> <p style="text-align: center;">-----</p> <pre>&lt;/form&gt;</pre>	1) Placing extension word in the action url of form page is optional. <pre>&lt;html:form action= "reg.do"&gt;</pre> <p style="text-align: center;">----- Optional</p> <p style="text-align: center;">-----</p> <pre>&lt;/html:form&gt;</pre>
2) The MVC-2 rules implementation is not possible.	2) The MVC-2 rules implementation is possible.
3) Cannot be used to generate dynamic content.	3) Can be used to generate dynamic Location.

- The process of making jsp programs as Java codeless jsp programs, by taking the support of various types of tags comes under implementing "view helper" design pattern.
- To make the jsp programs of web-application as (Struts Application) Java codeless JSP programs, use following tags.

- 1) Built-in JSP Tags of JSP Technologies. (except scriplets, Declaration, Expression tags)
- 2) Struts supplied JSP Tag Libraries based JSP tags.
- 3) JSTL tags.
- 4) Custom JSP Tag Library Tags.

**Prototypes of two execute methods**

1. Public ActionForward execute (ActionMapping mapping, ActionForm form, ServletRequest req, ServletResponse res) throws Exception.
2. Public ActionForward execute (ActionMapping mapping, ActionForm form, HttpServletRequest req, HttpServletResponse res) throws Exception

Second form of execute(-,-,-) method is recommended to use.

- ✓ Compared to simple ServletRequest, ServletResponse objects the HttpServletRequest, HttpServletResponse object can gather more and all the details from HttpRequest and can work with all the details of response.
- ✓ Mapping object is useful to gather various details of current action class using object the current Action class using this object. The current Action class can communicate with other Action classes. This object can also be used to generate ActionForward object pointing to the result page.
- ✓ Form object represents the FormBean class object that is linked with Action class. This object help the programmer to read form page form data and to use that data as input values in the business logic of execute(-,-,-) method.
- ✓ 'req' object is useful to gather various details from request coming to current struts Action class like header values, miscellaneous information and etc.
- ✓ 'res' object is useful for the programmer to set HttpServletResponse details like content type and other header values. These details help the programmer to generate the web-page more dynamically.

### Note

The above execute(-,-,-) method gets all the four parameters from ActionServlet.

execute(-,-,-) method always returns one ActionForward class object pointing to the result processing destination of Action class (result page). In this process, it makes ActionServlet to use the ActionForward configurations of struts configuration file internally.

### Software setup of first struts application development

- JDK 1.6 (JSE 6.0)(Java 6)(mustang)
- Tomcat 6.0 (compatible with JDK 1.6)
- Struts 1.3.8
- ❖ When java web-application uses struts API in its web-resource program, then the struts API related main jar file (Struts-core-1.3.8.jar) file must be placed in classpath and the struts API related main and dependent jar files (1+9 jar files) must be placed in WEB-INF/lib folder of struts application.
- ❖ In the above statement, jar files added in the classpath will be used by java compiler to recognize struts API during the compilation of FormBean, Action class programs.
- ❖ Similarly, the jar files added in WEB-INF/lib folder will be used by servlet API during the execution of form bean, Action class programs.

### Resources of our first struts application

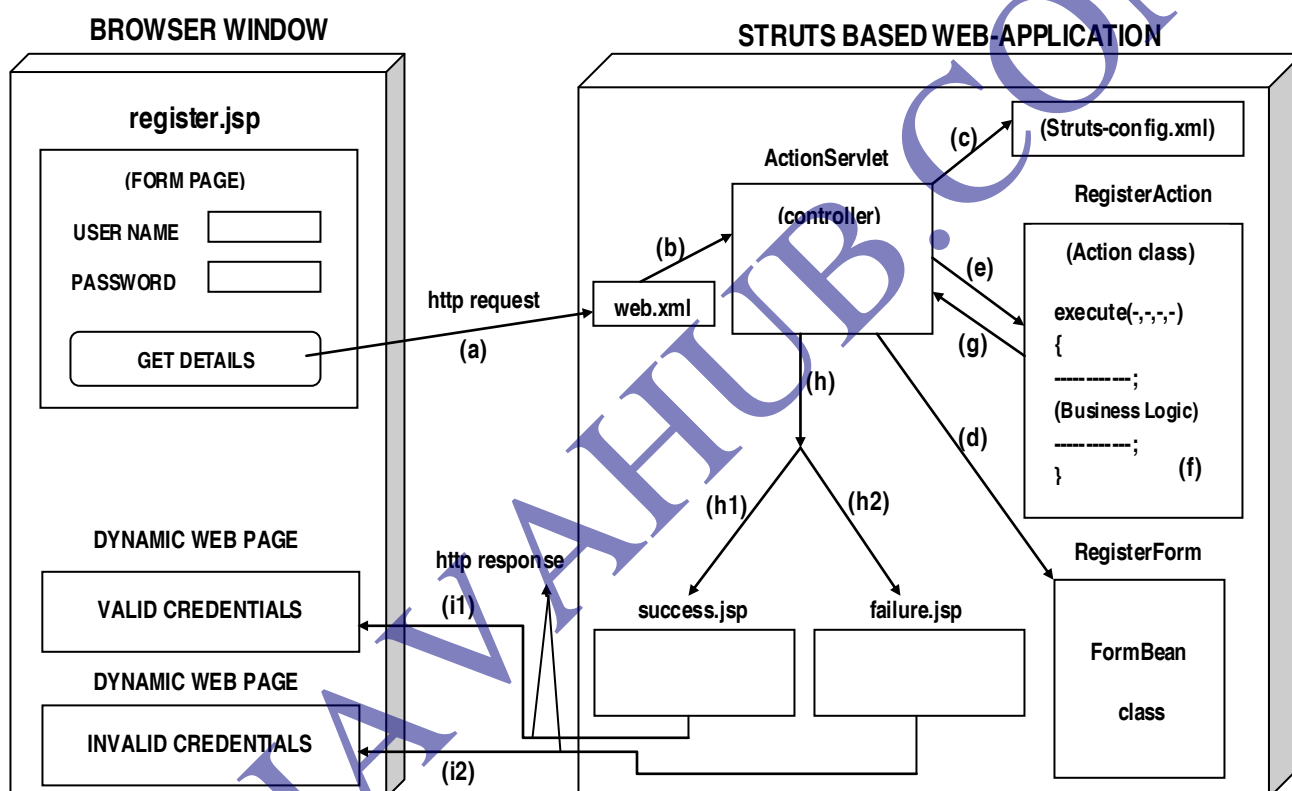
- ✓ Register.jsp (form page)(View layer)
- ✓ RegisteForm.java (FormBean class) (Controller layer)
- ✓ RegisterAction.java (Action class) (Model layer)
- ✓ Struts-config.xml (struts configuration file) (Controller)

- ✓ Web.xml (Deployment Descriptor file)(View layer)
- ✓ success.jsp
- ✓ failure.jsp

When xml files are developed based on DTD or Schema rules, then the tag names and attribute names in that file are pre-defined. Otherwise the tag names and attribute names are user defined.

Generally all the xml files of software technology based applications will be developed based on DTD or Schema rules given by those technologies. The tags and attributes in web.xml, struts-config.xml files are pre-defined because they will be developed based on DTD, Schema rules.

### Flow of execution of the struts first application



### Q) Is it compulsory to enable <load-on-startup> on ActionServlet in our Struts Application?

A) If the form page or first page of the struts application is developed by using traditional Html tags then enabling <load-on-startup> on ActionServlet is optional.

If the same form page is designed by using struts supplied JSP Tags then enabling <load-on-startup> on ActionServlet is mandatory operation because, the struts supplied JSP Tags internally uses framework software support and this framework software will be activated only with instantiation of ActionServlet.

**Note:** Don't create the FormBean class object manually in struts Action class, because it acts as empty object and doesn't contains the form data of form page. Always use the ActionServlet supplied FormBean class object (ie: second parameter of execute(-,-,-) method because it contains form data of form page.

## Procedure to develop and deploy the above diagram related struts application

### Step-1

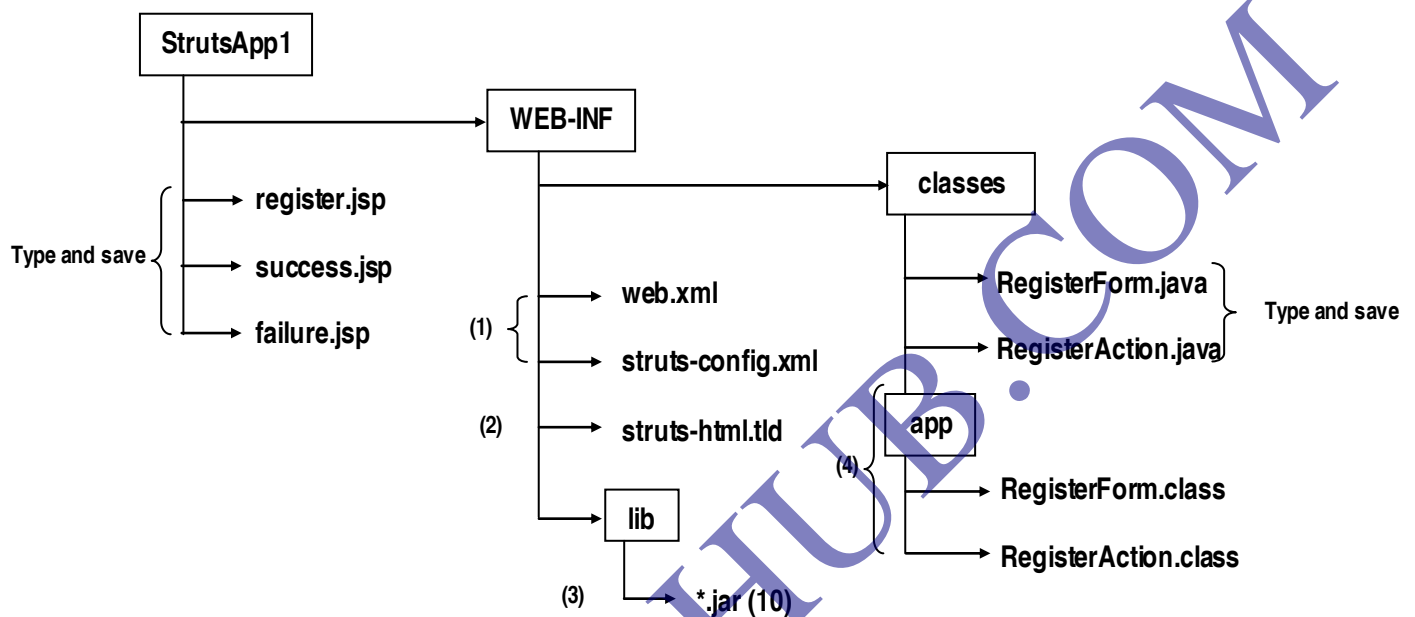
Extract struts home\apps\struts-blank-1.3.8.war file to temp folder.

### Step-2

Extract struts home\lib\Struts-tablib-1.3.8.jar file to temp folder.

### Step-3

Create the deployment directory structure (staging diagram) as follows



(1) Gather from step-1 extraction and modify according to your requirement.

(2) Gather from step-2 extraction.

(3) Gather from step-1 extraction.

(4) Comes after step-5.

### Note

The 10 jar files kept in WEB-INF\lib folder are main and dependent jar files of Struts api, Struts supplied Jsp Tag libraries.

### Step-4

Add Tomcat home\lib\servlet-api.jar file, Struts home\lib\struts-core-1.3.8.jar file to classpath environment variable.

### Note

The above two jar files are added to the classpath to make Javac Compiler recognizing servlet-api, struts api.

### Step-5

Compile FormBean, Action class source files by using javac -d as shown below

Cmd:/Tomcat 6.0/webapps/StrutsApp1/WEB-INF/classes>javac -d . \*.java

## Note

Steps 1-5 indicates the development process of Struts Application

## Step-6

Start Tomcat Server

## Step-7

Deploy the Struts Application.

## Step-8

Test the Struts Application.

## Key points

- ✓ The modifications done in the html, jsp, web.xml of Struts Application will be recognized by the underlying server automatically.
- ✓ The modifications done in struts configuration file will be reflected after reloading the web application.
- ✓ The modifications done in '.java' resources of struts applications will be reflected after recompilation of source files and reloading of the web-application.

**Q) Why the process of enabling <load-on-startup> on ActionServlet is mandatory when Form Page is designed by using struts supplied Jsp-tags?**

**Ans)** When struts supplied Jsp tags based Form page is launched, it internally tries to gather the default data of related FormBean class object and shows them as initial values of Form components (like text boxes in form page). For this before execution of these Jsp tags itself, ActionServlet object should be ready to activate framework software and to create FormBean class object. So enabling <load-on-startup> on ActionServlet is mandatory in the above situation.

The traditional html tags based form page will not try to gather FormBean class object default data when form page is launched. So enabling <load-on-startup> on ActionServlet is optional process in this situation.

## Struts Basic Application

register.jsp (form page) (view layer)

```

1  <%@ taglib uri="/demo" prefix="html" %>
2
3  <html:form action = "register">
4
5  UserName :<html:text property = "username"/><br>
6  PassWord :<html:password property = "password"/><br>
7      <html:submit value = "Register"/>
8  </html:form>

```

Refer line no: 34 of web.xml file

Logical names of form components

Caption of submit button

ActionPath of targetAction class

**web.xml (Deployment Descriptor file)**

```

1 <web-app>
2
3 <!-- Action Servlet Configuration -->
4 <servlet>
5   <servlet-name>action</servlet-name>
6   <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
7
8   <init-param>
9     <param-name>config</param-name>
10    <param-value>/WEB-INF/struts-config.xml</param-value>
11  </init-param>
12
13  <load-on-startup>1</load-on-startup>
14 </servlet>
15
16 <!-- Action Servlet Mapping -->
17 <servlet-mapping>
18   <servlet-name>action</servlet-name>
19   <url-pattern>*.do</url-pattern>
20 </servlet-mapping>
21
22 <!-- The Usual Welcome File List -->
23 <welcome-file-list>
24   <welcome-file>register.jsp</welcome-file>
25 </welcome-file-list>
26
27 <!-- Struts Tag Library Descriptors -->
28 <taglib>
29   <taglib-uri>/demo</taglib-uri>
30   <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
31 </taglib>
32
33 </web-app>

```

Specifies the name and location of the Struts Configuration file

Mandatory because the form page is designed by using Struts supplied Jsp tags

Extension match url pattern to make ActionServlet as Front Controller

Struts supplied Html tag libraries configuration

**struts-config.xml ( struts configuration file) (controller layer)**

```

1 <!DOCTYPE struts-config PUBLIC
2   "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
3   "http://jakarta.apache.org/struts/dtds/struts-config_1_3.dtd">
4
5 <struts-config>
6
7   <form-beans>
8     <form-bean name="rf" type="app.RegisterForm"/>
9   </form-beans>
10
11 </struts-config>

```

Indicates that the programmer is developing this struts configuration file against the DTD file (struts-config\_1\_3.dtd) rules

FormBean logical name

FormBean class configuration

```

16
17 <action-mappings>
18
19   <action name="rf" path="/register" type="app.RegisterAction">
20       <forward name="ok" path="/success.jsp"/>
21       <forward name="fail" path="/failure.jsp"/>
22   </action>
23
24 </action-mappings>
25
26 </struts-config>

```

Refer line no: 12 of struts-config.xml file

Action path of Action class

ActionForward configuration pointing to result pages

Struts Action class configuration

### RegisterForm.java (FormBean class) (controller layer)

```

1 package app;
2 import org.apache.struts.action.*;
3
4 public class RegisterForm extends ActionForm {
5
6     String username;
7     String password;
8
9     public void setUsername(String username) {
10         this.username = username;
11     }
12     public void setPassword(String password) {
13         this.password = password;
14     }
15
16     public String getUsername() {
17         return username;
18     }
19     public String getPassword() {
20         return password;
21     }
22 }

```

Madatory

FormBean properties, these names must match with the form page component names. (Refer line nos: 5,6 of register.jsp)

ActionServlet internally calls these methods to write form data to FormBean class object.

Programmer calls these methods in struts Action class to read form data from FormBean class object.

### RegisterAction.java (Struts Action class) (model layer)

```

1 package app;
2 import org.apache.struts.action.*;
3 import javax.servlet.http.*;
4
5 public class RegisterAction extends Action {
6
7     public ActionForward execute(ActionMapping mapping, ActionFormform,
8     HttpServletRequest request, HttpServletResponse response)
9     throws Exception

```

Madatory

```

10  {
11      RegisterForm rf = (RegisterForm)form;
12
13      String user = rf.getUsername();
14      String pass = rf.getPassword();
15  // Business logic
16      if(user.equals("sathya") && pass.equals("java"))
17          return mapping.findForward("ok");//Refer line no: 22 of struts-config.xml
18      else
19          return mapping.findForward("fail");//Refer line no: 23 of struts-config.xml
20  }//execute
21  }//class

```

Type casting

Reading form data from FormBean class object

Mapping.findForward ( ) return ActionForward class object pointing to the result page of ActionServlet based on its logical name specified in the struts configuration file.

Mapping.findForward ( ) return Action class object having logical name pointing to the result page of Action class.

### success.jsp (result page) (view layer)

```

1  <html>
2      <body>
3          <center>
4              <font size = 5 color = green>Login Successfull</font>
5          </center>
6      </body>
7  </html>

```

### failure.jsp (result page) (view layer)

```

1  <html>
2      <body>
3          <center>
4              <font size = 5 color = red>Login Failure</font><br>
5              <a href = "register.jsp">Try Again</a>
6          </center>
7      </body>
8  </html>

```

This hyperlink is against MVC-2 principles as failure.jsp is directly talking with register.jsp

### Scopes of FormBean class object

We can keep FormBean class object of struts application in two scopes

#### Request Scope

ActionServlet creates 1 separate FormBean class object for every request generated by form page or when form page is launched each side.

#### Session Scope (Default Scope)

ActionServlet creates one FormBean class object for all the requests generated by form page from a browser window.



### In struts-config.xml

```
<action path= "/register" type= "app.RegisterAction" name= "rf"
scope= "request (or) session">
-----
-----
</action>
```

Scope of the FormBean class object

#### Note

There are no scopes for struts Action class object. ActionServlet either creates (when not available) or locates (when available) struts Action class object.

When form page is launched, if we want to observe the previous submission related form data as initial values, gives session scoped FormBean. Otherwise use request scoped FormBean (Default scope is session scope)

#### Flow of execution of struts application which is discussed in the above.

- a) Programmer deploys StrutsApp1 web-application in Tomcat web server.
- b) Because of <load-on-startup> the servlet container creates ActionServlet object either during server startup or during the deployment of the web-application. (refer line no:15 of web.xml)
- c) ActionServlet reads and verifies struts configuration file (struts-config.xml) entries and activates struts framework software internally.
- d) End user launches form page (register.jsp) on browser window by following url <http://localhost:2020/StrutsApp1/register.jsp>
- e) The Jsp tags of form page (register.jsp) uses given action url (action="register"), gathers the FormBean class of form page based on FormBean logical name "rf" in struts configuration file (struts-config.xml) (refer line-3 of register.jsp, lines-20, 12 of struts-config.xml)
  - Makes ActionServlet to create FormBean class object (session scoped)
  - Makes ActionServlet to call getXxx() methods on FormBean class object.
  - Keeps the default data of FormBean class object as initial values of form components (text boxes and etc).
- f) End user fills up the form page and submits the form page. (refer line-7 of register.jsp)
- g) Form page generate request url as shown below <http://localhost:2020/StrutsApp1/register.do> (refer line-3 of register.jsp)
- h) ActionServlet traps and takes the request (because request url is having ".do" word and url-pattern of ActionServlet id "\*.do") (refer 19-24 & 4-16 of web.xml)
- i) ActionServlet locates FormBean class object of formpage (from session scope) and writes formdata to that object by calling setXxx(-) methods. (refer lines-20, 12 of struts-config.xml and 10-17 of FormBean class (RegisterForm))
- j) ActionServlet calls execute (-,-,-) method of struts Action class (RegisterAcion) to process the request. (refer lines-1-20 of RegisterAction class)

OK

FAIL

<b>k)</b>	Returns ActionForward object. (refer line-17 of RegisterAction)	<b>k)</b>	Returns ActionForward object. (refer line-19 of Register form)
<b>l)</b>	ActionServlet uses ActionForward configurations to get result page. (refer line-22 of struts-config.xml)	<b>l)</b>	ActionServlet uses ActionForward configurations to get result page. (refer line-23 of struts-config.xml)
<b>m)</b>	ActionServlet sends the control to result page (success.jsp)	<b>m)</b>	ActionServlet sends the control to result page (failure.jsp)
<b>n)</b>	End.	<b>n)</b>	(Hyper link) control goes to form page (register.jsp)
		<b>o)</b>	End.

### Note

The FormBean class and Action class of the struts application must be taken as public classes in order to make them visible to ActionServlet.

SAX (Simple Api for Xml parser), DOM (Document Object Model), DOM4J (Document Object Model for Java), JDOM and etc are XML parsers which can read and process XML document. Most of the web servers, ActionServlet uses SAX parser to read and process the web.xml, struts-config.xml files respectively

### NetBeans IDE

Type	: IDE software for java application development
Version	: 6.7.1 (compatible with Jdk 1.5&1.6)
Vendor	: Sun Micro Systems
Source type	: Open Source
Server supplied along with it	: Glassfish 2.x Application server
Download link ( for software)	: <a href="http://www.netbeans.org">www.netbeans.org</a> website
Download link ( for help/doc)	: <a href="http://www.netbeans.org">www.netbeans.org</a> website

**Note:** Glassfish server default port number to access web-applications is 8081

For struts application to database software communication add JDBC code in the Action class of struts application.

**Procedure to develop first struts application by using NetBeans IDE. But make that application interacting with database software to verify whether given username and password are correct or not.**

### Step-1

Create database table having username and password columns with name "userlist".

## Query of the application

select count(\*) from userlist where uname= "raja" and pwd= "rani";

if user exists it returns 1 else returns 0.

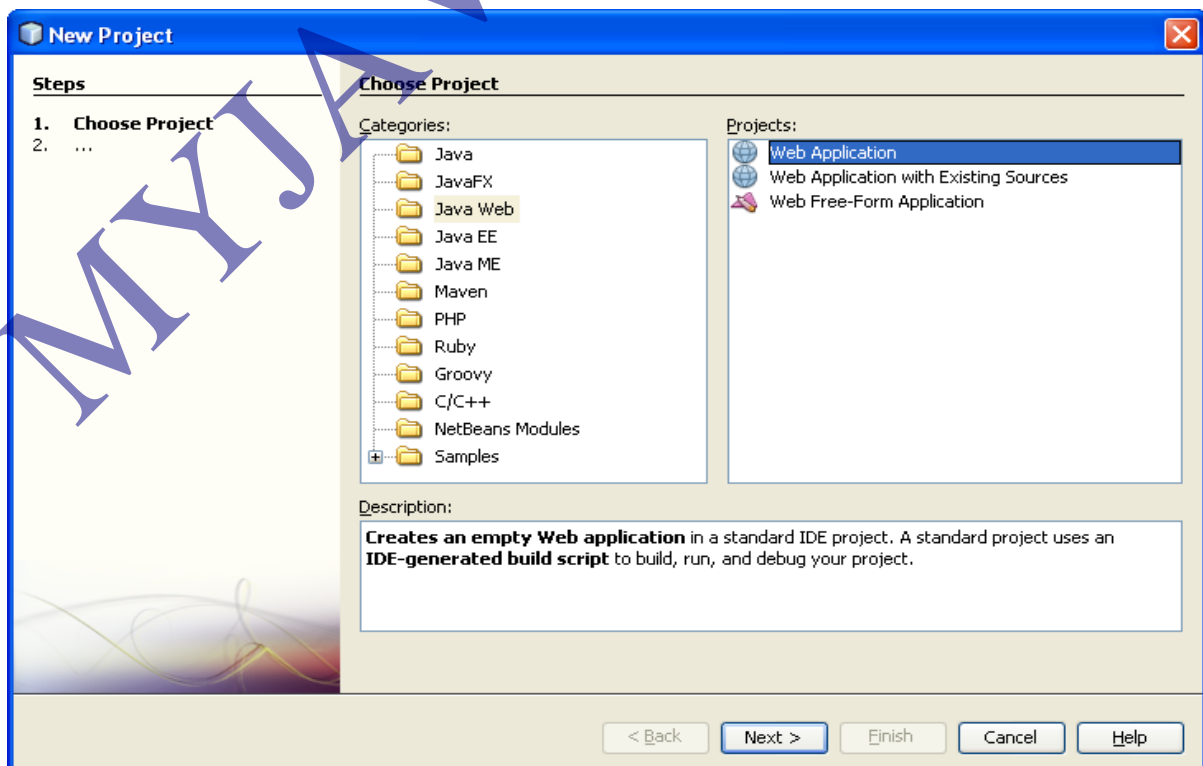
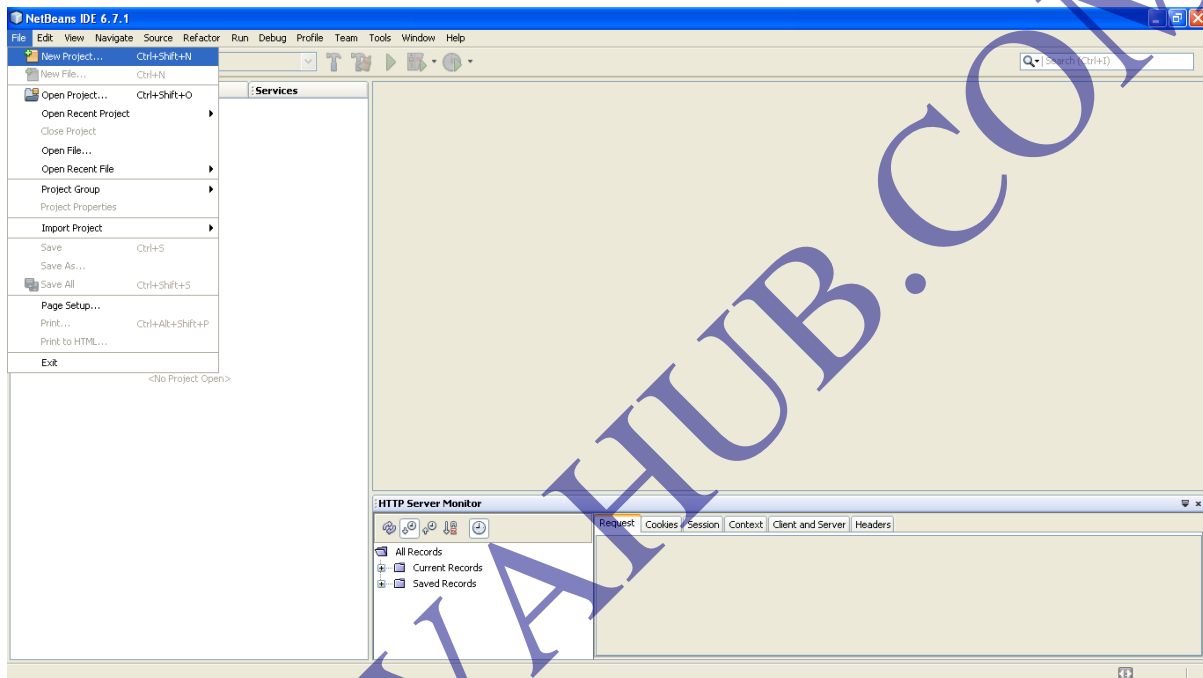
## Step-2

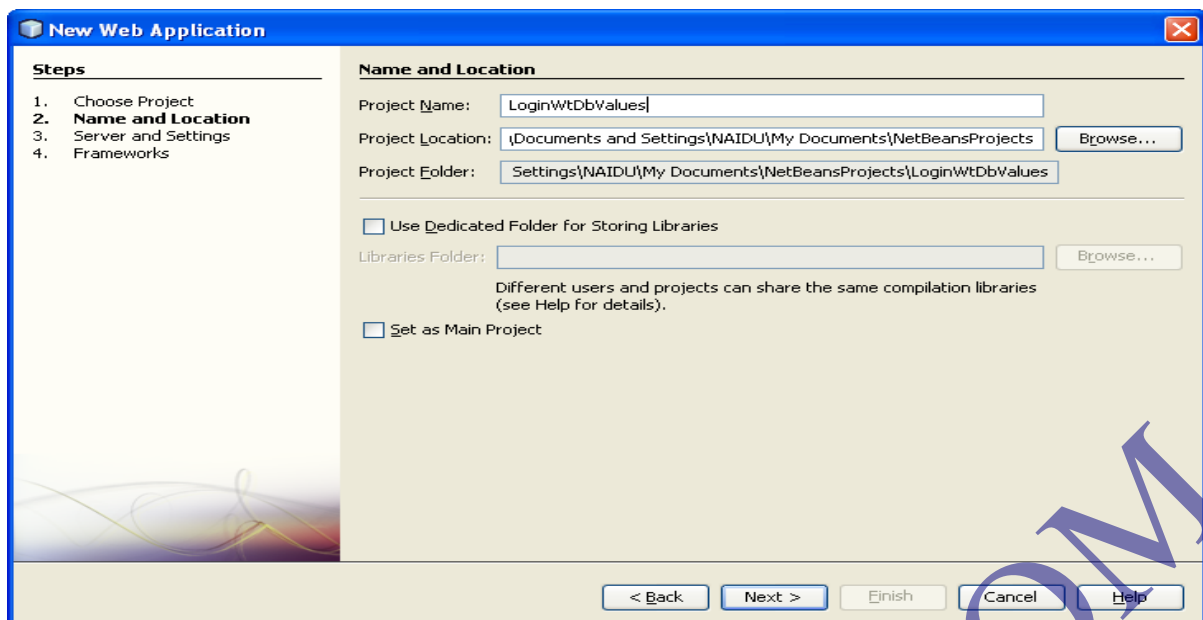
Launch NetBeans IDE, create web project in it adding struts capabilities.

File menu → New project → Java web → web-application → next → Project name LoginWtDbVal

→ next → Server Glassfish 2.1 → next → Select struts 1.3.8 → check Add struts TLDs → finish.

Delete index.jsp, welcome struts.jsp files.





**New Web Application**

**Steps**

1. Choose Project
2. **Name and Location**
3. Server and Settings
4. Frameworks

**Name and Location**

Project Name:

Project Location:

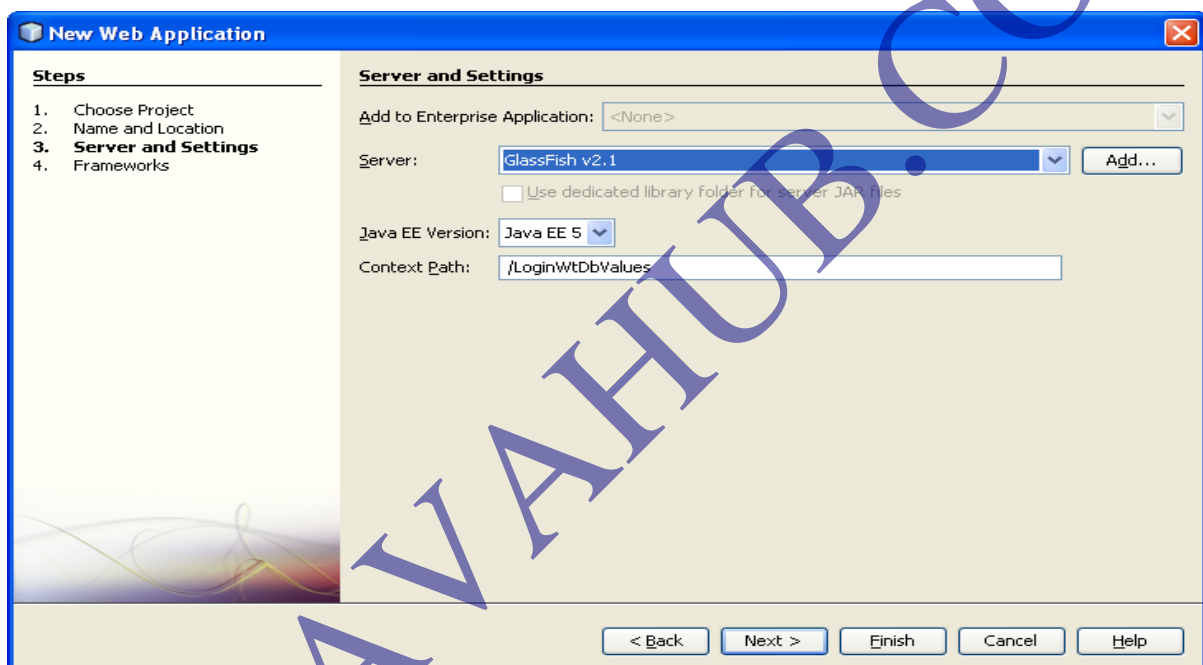
Project Folder:

☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

☐ Set as Main Project



**New Web Application**

**Steps**

1. Choose Project
2. Name and Location
3. **Server and Settings**
4. Frameworks

**Server and Settings**

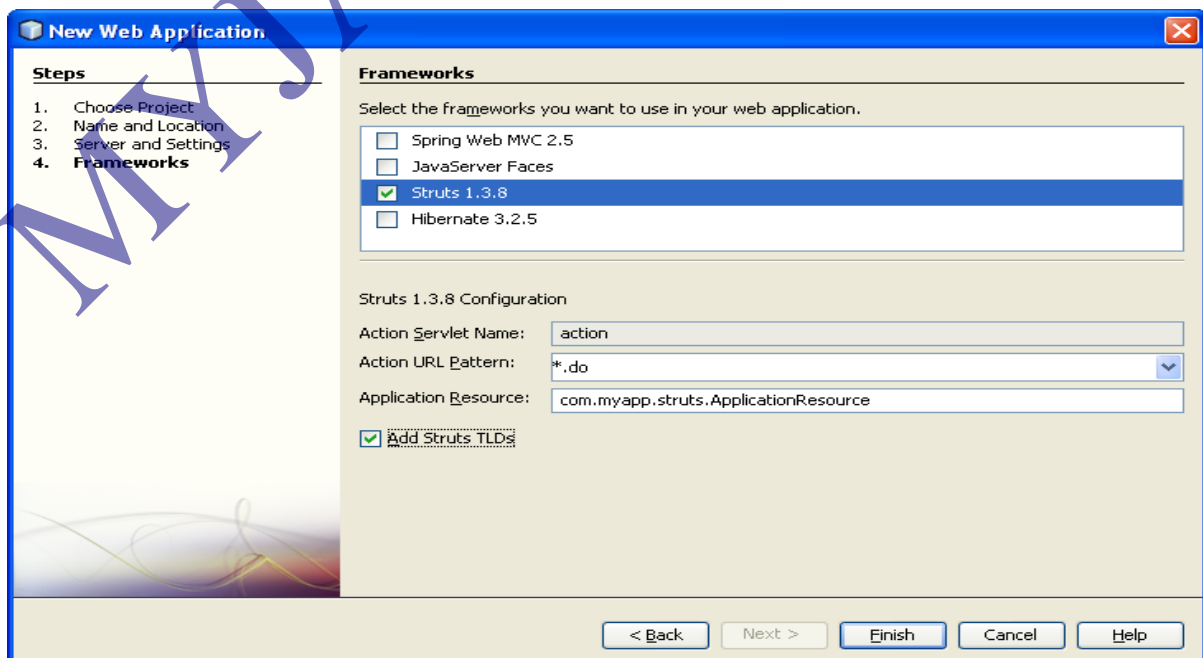
Add to Enterprise Application:

Server:

☐ Use dedicated library folder for server JAR files

Java EE Version:

Context Path:



**New Web Application**

**Steps**

1. Choose Project
2. Name and Location
3. Server and Settings
4. **Frameworks**

**Frameworks**

Select the frameworks you want to use in your web application.

☐ Spring Web MVC 2.5

☐ JavaServer Faces

☒ Struts 1.3.8

☐ Hibernate 3.2.5

Struts 1.3.8 Configuration

Action Servlet Name:

Action URL Pattern:

Application Resource:

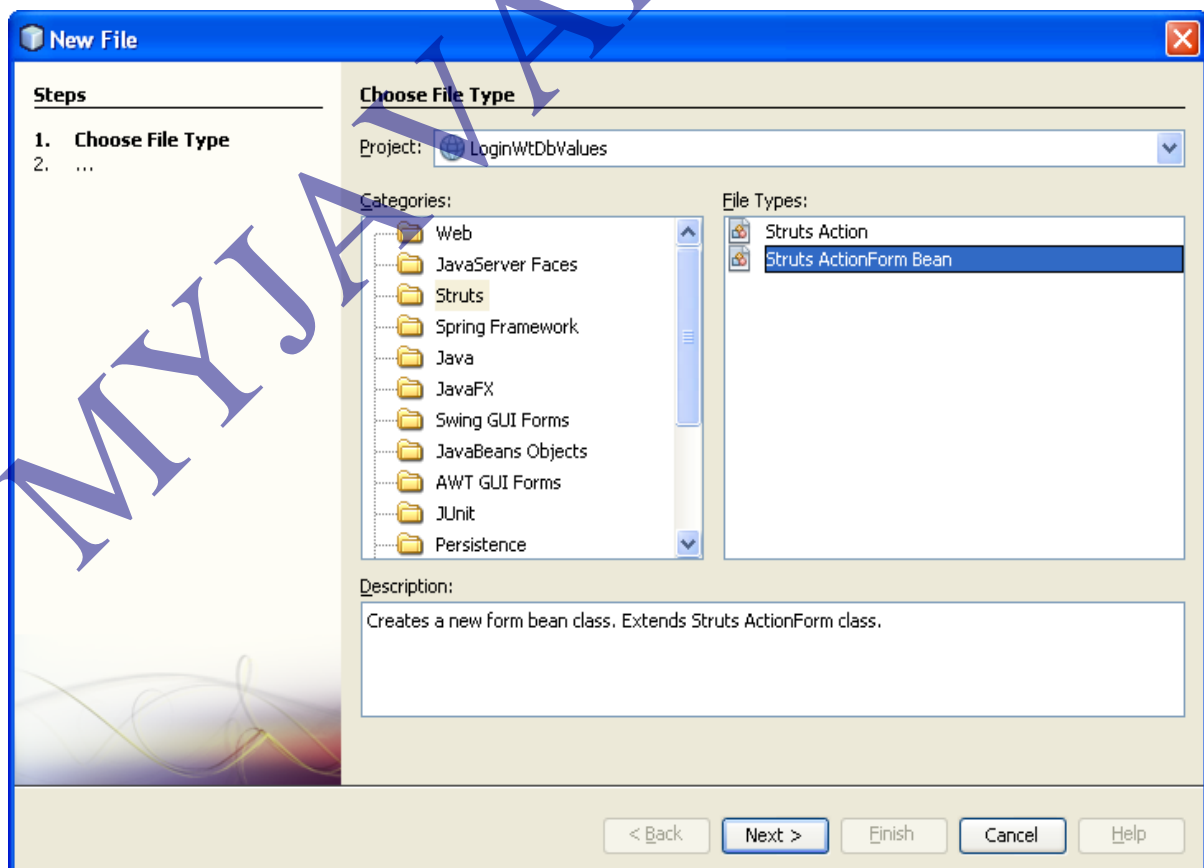
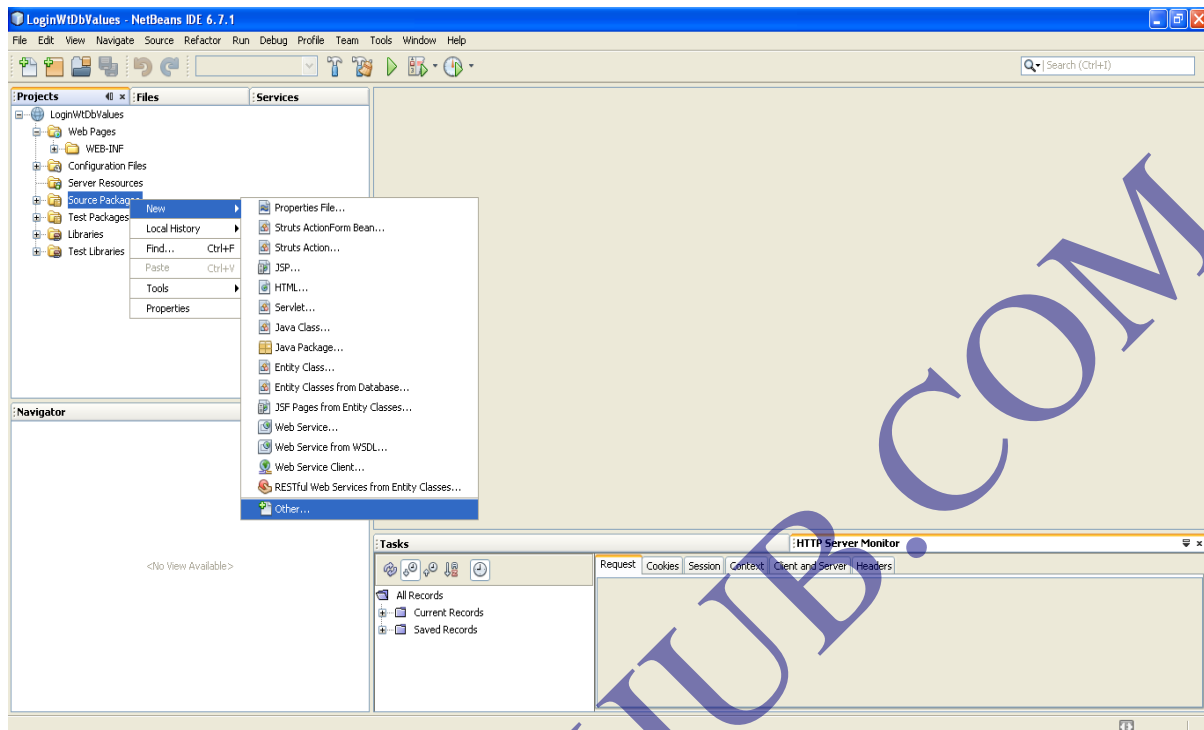
☒ Add Struts TLDs

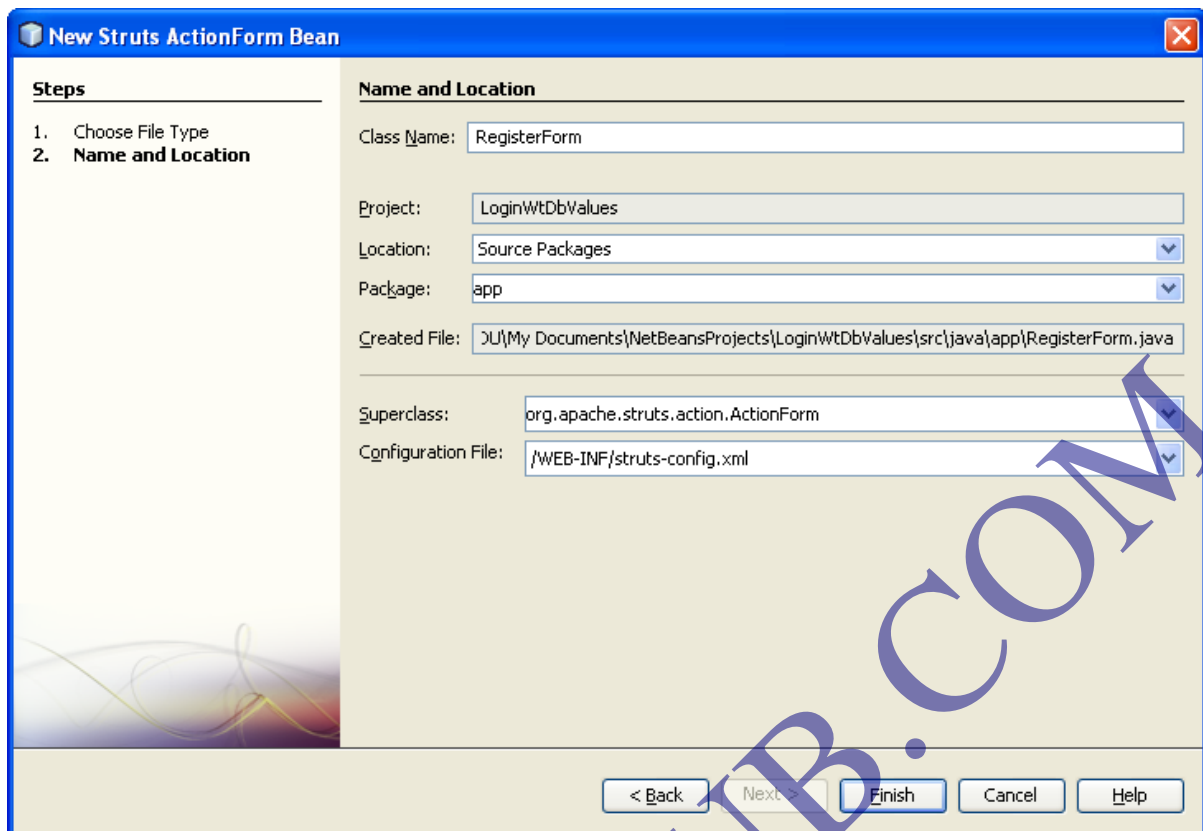
### Step-3

Add FormBean class to the project.

Right click on the source packages folder of the project → new → others → Struts →

Struts ActionForm Bean → next → Class name `RegisterForm` → Package name `app` → Finish.

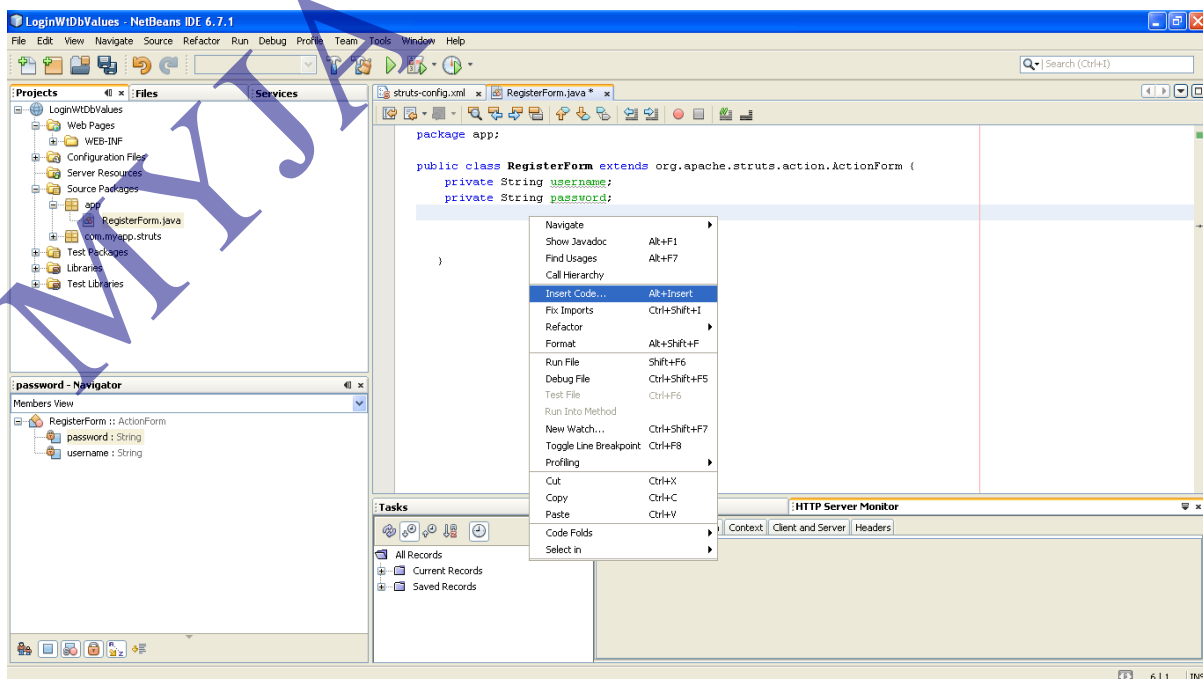


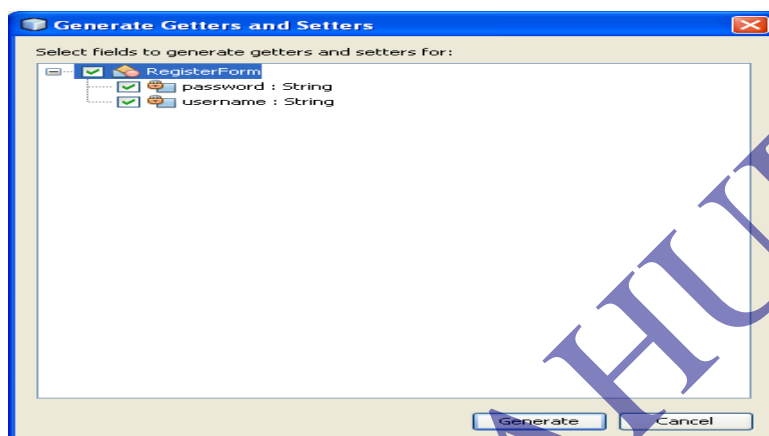
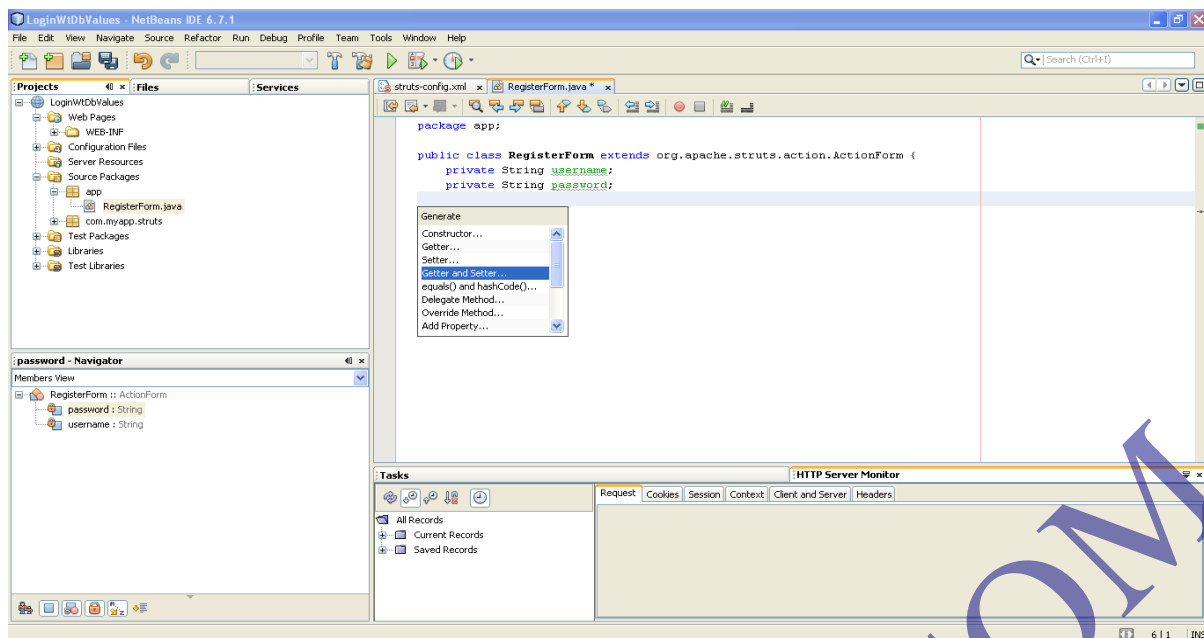


Add same code of first application's "RegisterForm.java" which is discussed in previous discussions.

### Note

To generate getXxx() and setXxx(-) methods, type FormBean properties and select FormBean property names → Right click → insert code → select getXxx(), setXxx(-) → Generate.



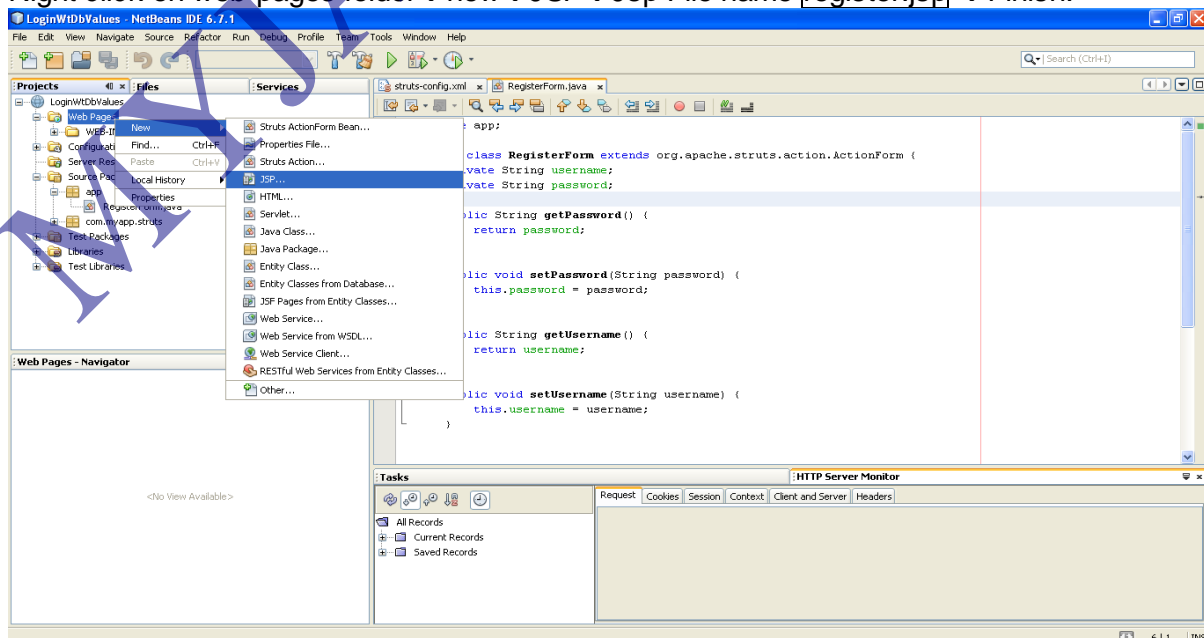


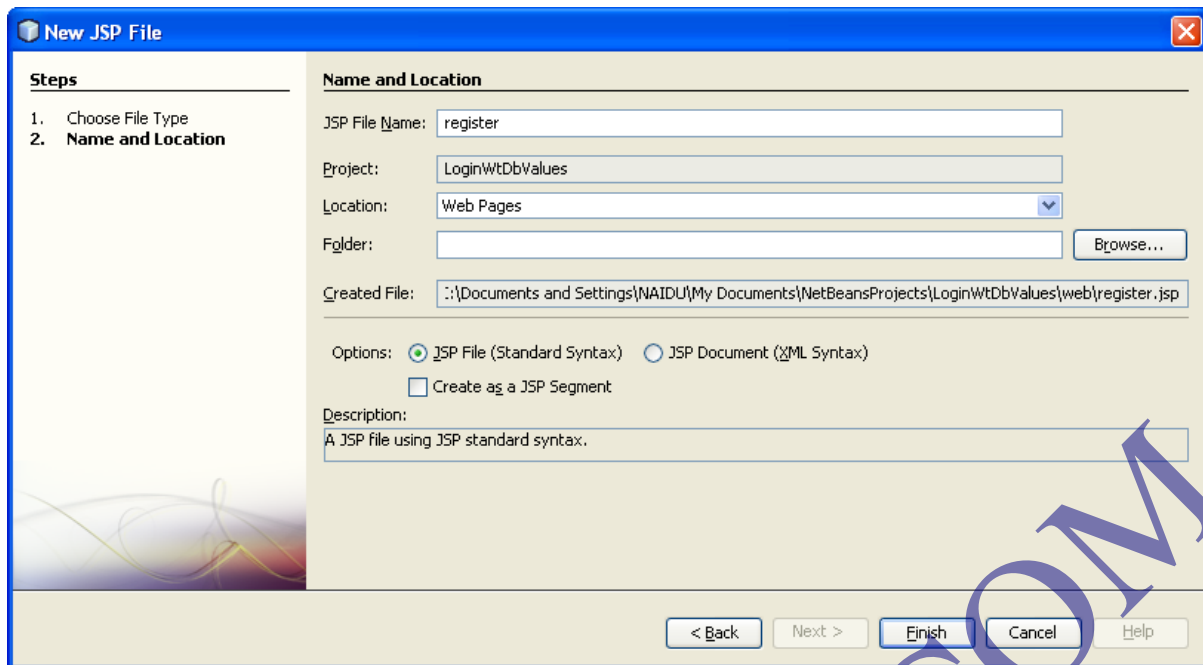
(The above operation updates struts configuration file (struts-config.xml) having FormBean configuration)

#### Step-4

Add Form page to struts project.

Right click on web-pages folder → new → JSP → Jsp File name register.jsp → Finish.





Add same code of first application's "register.jsp" which is discussed in previous discussions.

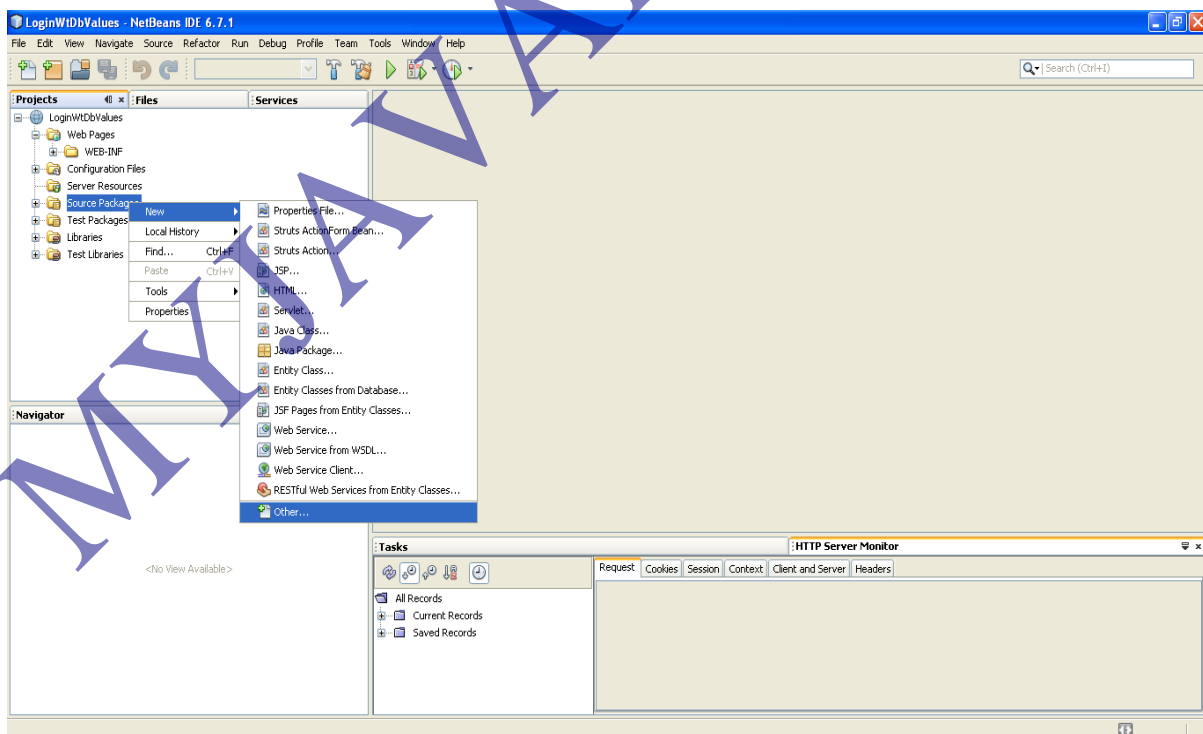
### Step-5

Add Action class to the project.

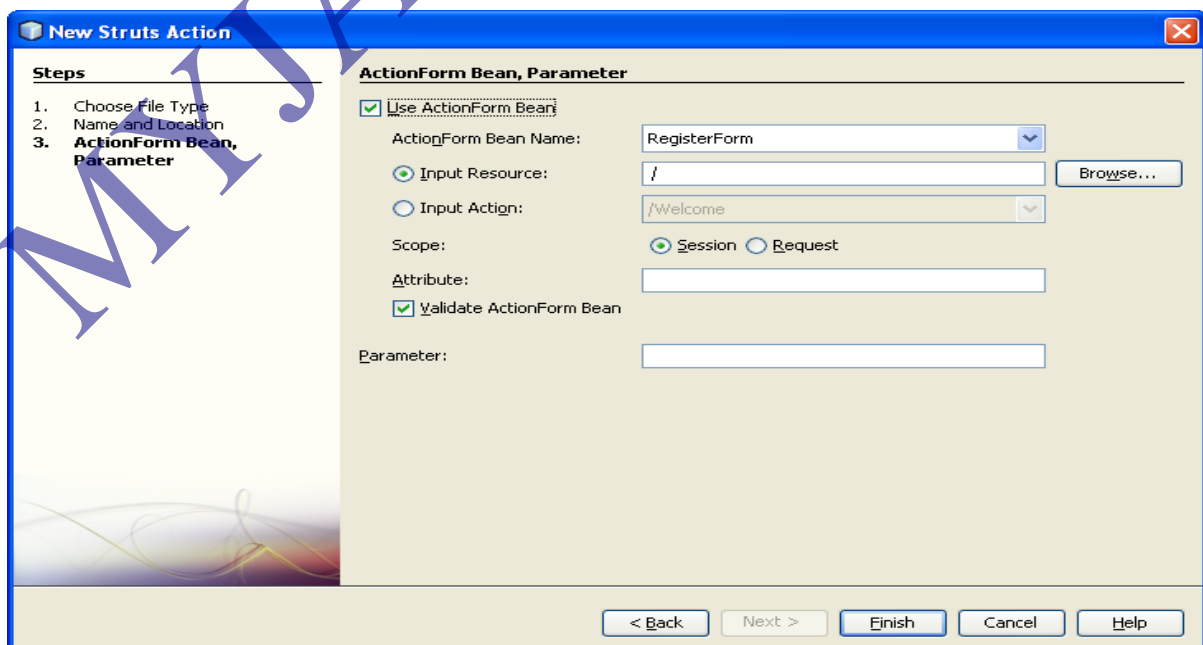
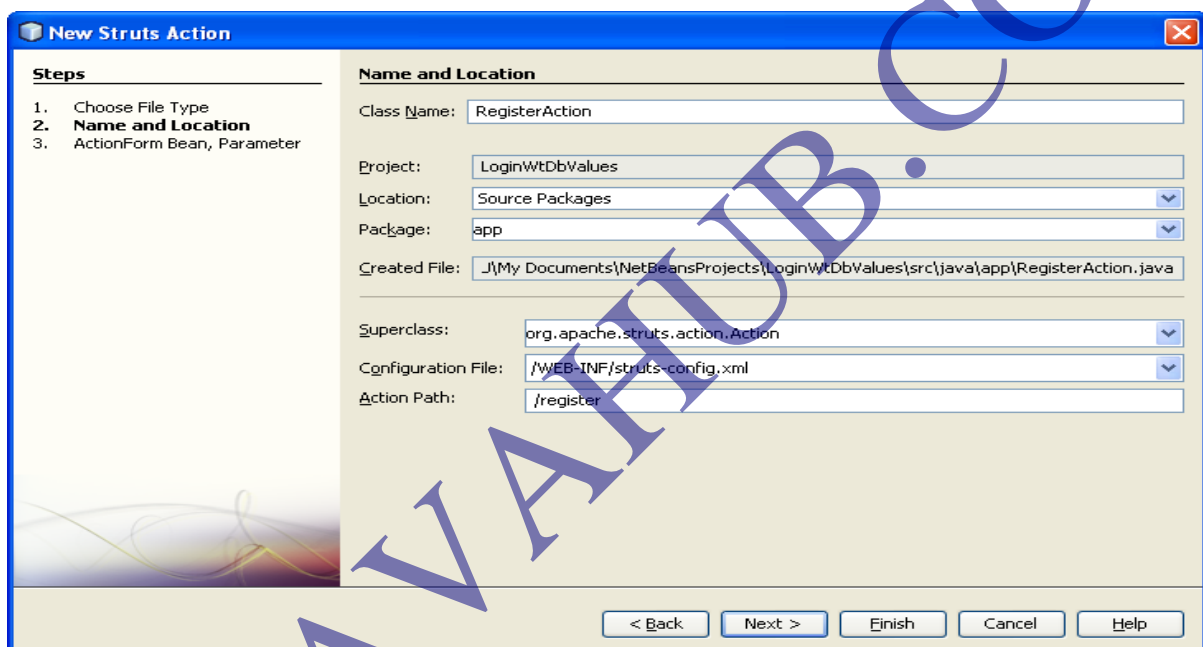
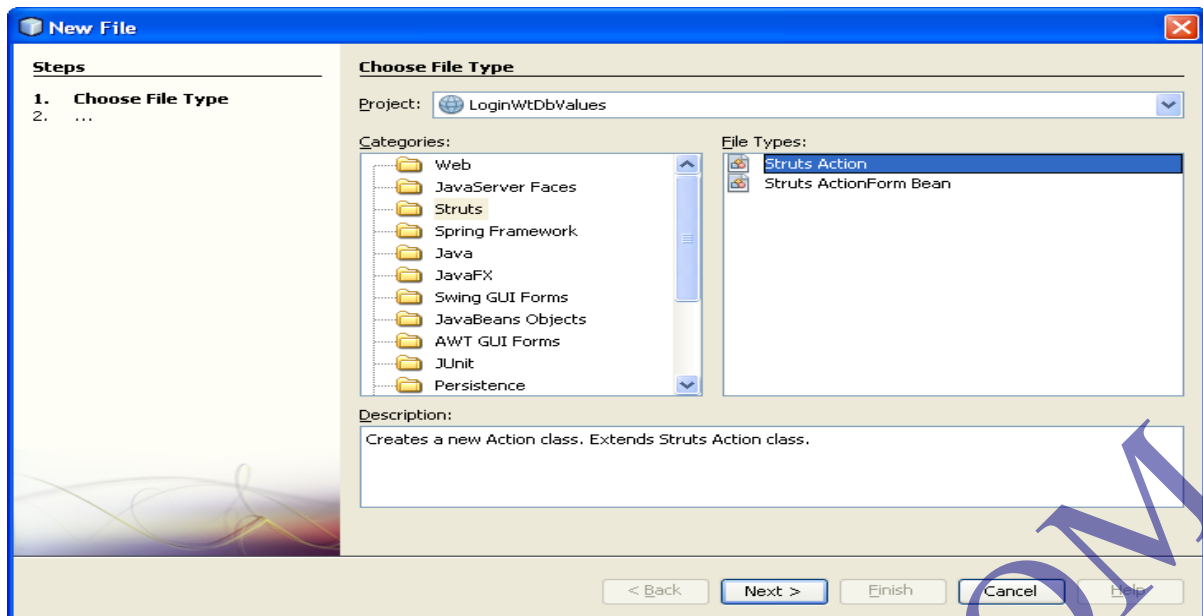
Right click on source packages folder → new → others → Struts → Struts Action → next →

Class name `RegisterAction` → Package name `app` → Action path `/register` → next →

Check scope as session → Finish.







## Step-6

Add two result pages as ActionForward configurations in struts configuration file under <action> tag that configure the above “RegisterAction” class as follows

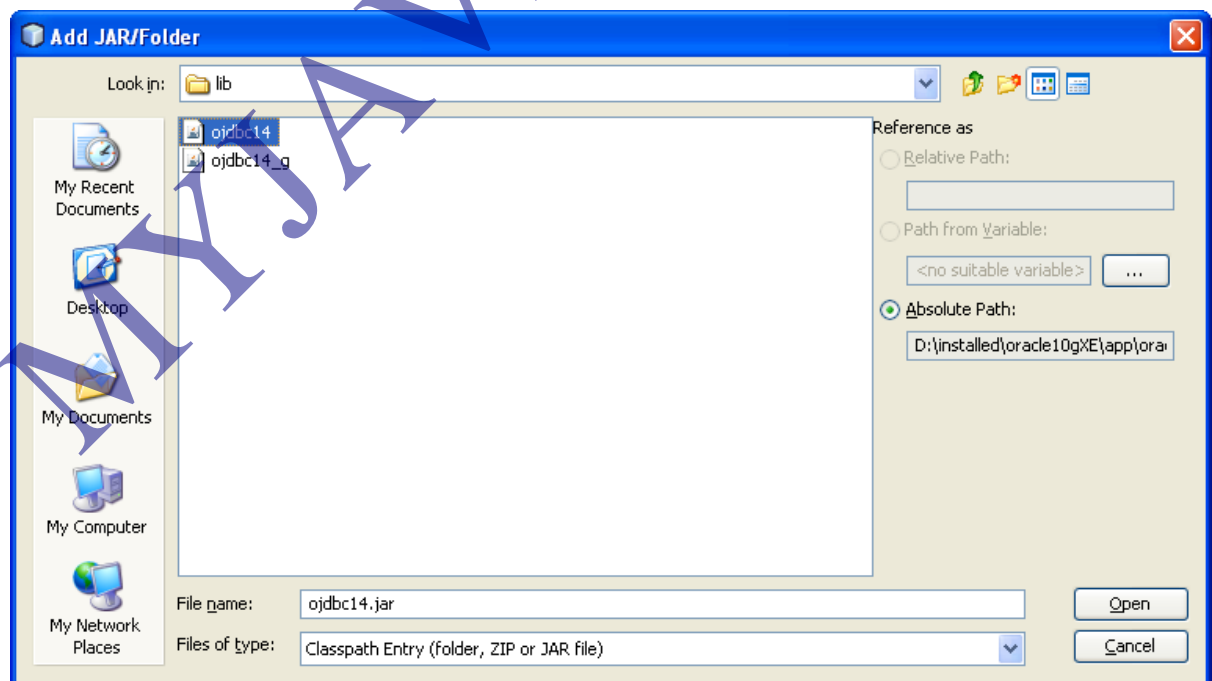
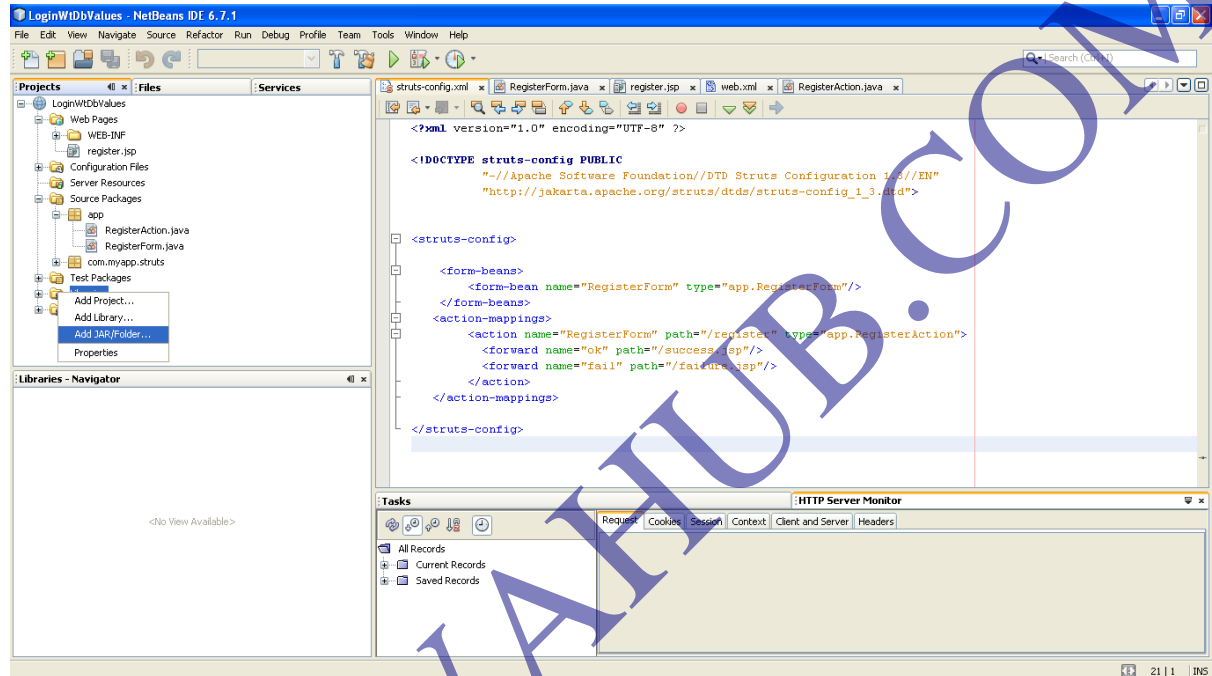
```
<forward name= "ok" path= "/success.jsp"/>
```

```
<forward name= "fail" path= "/failure.jsp"/>
```

## Step-7

Add ojdbc14.jar file to the libraries of the project to work with oracle thin driver.

Right click on libraries folder→Add jar→Browse and select ojdbc14.jar



**Step-8**

Write following code in Struts Action class.

```
public class RegisterAction extends org.apache.struts.action.Action {
    Connection con;
    PreparedStatement ps;
    public RegisterAction()
    {
        try
        {
            Class.forName("oracle.jdbc.driver.OracleDriver");

            con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","scott","tiger");
            ps=con.prepareStatement("select count(*) from userlist where uname=? and pwd=?");
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
    @Override
    public ActionForward execute(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response)
        throws Exception {
        RegisterForm rf = (RegisterForm)form;
        String user = rf.getUsername();
        String pass = rf.getPassword();
        ps.setString(1,user);
        ps.setString(2,pass);
        ResultSet rs=ps.executeQuery();
        int cnt=0;
        if(rs.next())
            cnt=rs.getInt(1);
        if(cnt!=0)
            return mapping.findForward("ok");
        else
            return mapping.findForward("fail");
    }
}
```

To interact with database software for multiple times create JDBC connection object only once, use it for multiple times. Don't create multiple JDBC connection objects.

**Step-9**

Add success.jsp, failure.jsp files to the project in web pages folder.

**Step-10**

Run the project.

Full source code for the struts application which is explained in the above procedure.

#### RegisterForm.java (FormBean class)

```
package app;
public class RegisterForm extends org.apache.struts.action.ActionForm {
    private String username;
    private String password;
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
}
```

#### RegisterAction.java (Action class)

```
package app;
import java.sql.*;
import javax.servlet.http.*;
import org.apache.struts.action.*;
public class RegisterAction extends org.apache.struts.action.Action {
    Connection con;
    PreparedStatement ps;
    public RegisterAction()
    {
        try
        {
            Class.forName("oracle.jdbc.driver.OracleDriver");

            con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","scott","tiger");
            ps=con.prepareStatement("select count(*) from userlist where uname=? and pwd=?");
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
    @Override
    public ActionForward execute(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response)
        throws Exception {
        RegisterForm rf = (RegisterForm)form;
        String user = rf.getUsername();
```

```

String pass = rf.getPassword();
ps.setString(1,user);
ps.setString(2,pass);
ResultSet rs=ps.executeQuery();
int cnt=0;
if(rs.next())
    cnt=rs.getInt(1);
if(cnt!=0)
    return mapping.findForward("ok");
else
    return mapping.findForward("fail");
}
}

```

### register.jsp

```

<%@ taglib uri="demo" prefix="html" %>
<html:form action = "register">
    <table border = 0 align = center>
        <tr>
            <th>UserName
            <td><html:text property = "username"/>
        </tr>
        <tr>
            <th>PassWord
            <td><html:password property = "password"/>
        </tr>
        <tr>
            <td colspan = 2 align = center>
                <html:submit value = "Register"/>
            </td>
        </tr>
    </table>
</html:form>

```

### struts-config.xml

```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts
Configuration 1.3//EN" "http://jakarta.apache.org/struts/dtds/struts-config_1_3.dtd">
<struts-config>
    <form-beans>
        <form-bean name="RegisterForm" type="app.RegisterForm"/>
    </form-beans>
    <action-mappings>
        <action name="RegisterForm" path="/register" type="app.RegisterAction">
            <forward name="ok" path="/success.jsp"/>
            <forward name="fail" path="/failure.jsp"/>
        </action>
    </action-mappings>
</struts-config>

```

**web.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <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>2</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>register.jsp</welcome-file>
  </welcome-file-list>
  <jsp-config>
    <taglib>
      <taglib-uri>demo</taglib-uri>
      <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
    </taglib>
  </jsp-config>
</web-app>
```

**success.jsp**

```
<%@taglib uri="demo" prefix="html" %>
<html:html>
  <center>
    <font size = 5 color = green>Login Successful</font>
  </center>
</html:html>
```

**failure.jsp**

```
<%@taglib uri="demo" prefix="html" %>
<html:html>
  <center>
    <font size = 5 color = red>Login Failure</font>
  </center>
</html:html>
```

There are two approaches to work with Jsp tag libraries in our java web-application or struts application

### Approach-1

By using user-defined taglib uri.

### Approach-2

By using pre-defined and fixed taglib uri.

### Procedure to work with Approach-1 (By using user-defined taglib uri)

#### Step-1

Keep required tld file in WEB-INF folder of web application.

#### Step-2

Configure tag library in web.xml file having user defined taglib uri.

#### Step-3

Keep jar files of tag libraries in WEB-INF/lib folder

#### Step-4

Use the Jsp tags of tag library in Jsp programs of web-application by specifying that user-defined tag-lib uri.

#### Note

While developing our first struts application, we have used the above approach-1 to work with struts supplied Jsp tag libraries.

#### Example

##### Jsp form page

```
<%@ taglib uri="demo" prefix="html" %>
  <html:form action = "register">
    -----
    -----
  </html:form>
```

##### web.xml

```
<web-app>
-----
-----
  <taglib>
    <taglib-uri>demo</taglib-uri>
    <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
  </taglib>
</web-app>
```

Every Jsp tag library contains one taglib-uri (fixed and pre-defined) in its tld file. If you can work with these taglib uri there is no necessity of working with user-defined taglib uri(s).

## The pre-defined taglib uris of struts supplied jsp tag library

Jsp Tag Library	Pre-Defined Lib Uri
html tag library	http://struts.apache.org/tags-html (collected from <uri> tag of struts-html.dld)
bean tag library	http://struts.apache.org/tags-bean (collected from <uri> tag of struts-bean.dld)
logic tag library	http://struts.apache.org/tags-logic (collected from <uri> tag of struts-logic.dld)
nested tag library	http://struts.apache.org/tags-nested (collected from <uri> tag of struts-nested.dld)
tiles tag library	http://struts.apache.org/tags-tiles (collected from <uri> tag of struts-tiles.dld)

### Procedure to work with Approach-2 (By using pre-defined taglib uri)

#### Step-1

Place Jsp taglib related jar files in WEB-INF/lib folder

#### Step-2

Use Jsp tags of tag library in the Jsp program by specifying the pre-defined taglib uri (collect this taglib-uri from the related tld file)

#### Example

##### Jsp form page

```
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
    <html:form action = "register">
        -----
        -----
    </html:form>
```

#### Note

- ✓ Approach-2 is recommended.
- ✓ There is no need of adding the any tld files in WEB-INF folder.
- ✓ Configuration of <taglib-uri> in web.xml file is not required.
- Generally ActionServlet and Struts Action class uses same request and response objects and also ActionServlet and result page uses same request and response objects. So we can say Struts Action class and its result page will also use same request and response object.
 

```
<forward name= "ok" path= "success.jsp" redirect= "true/false">
```
- redirect= "true" indicates ActionServlet internally uses Response.sendRedirect (-) to communicate with result jsp page. Due to this ActionServlet and its result page, Action class and its result page will not use same request and response objects.
- redirect= "false" indicates that ActionServlet internally uses rd.forward() to communicate with result page. Due to this ActionServlet and its result page, Action class and its result page will be use same request and response objects.
- "false" is the default value of redirect attribute.



### Q) How to pass result values or data from Action class to result page of Struts Application.

**Ans:** If Action class and its result page utilizes same request and response objects (redirect=false) then use request attributes otherwise use session attributes (when redirect=true).

#### Example

With respect to first application follow the below steps

#### Step-1

Create attributes in the execute(-,-,-) method of Action class (Registration)

```
int a=10;

int res1=a*a;

int res2=a*a*a;

//keep result in attributes

req.setAttribute("result", new Integer(res1)); //request attribute

HttpSession ses=req.getSession();

ses.setAttribute("result2", new Integer(res2)); //session attribute
```

#### Step-2

Observe how result pages are configured for struts Action class in struts configuration file.

In struts-config.xml file

```
<action path= "/register" type= "app.RegisterAction" name= "rf">

  <forward name= "ok" path= "/success.jsp" redirect= "true"/>

  <forward name= "failure" path= "/failure.jsp" redirect= "false"/>

</action>
```

#### Step-3

Read attribute values in result.jsp pages success.jsp/failure.jsp

In success.jsp/failure.jsp

```
Result 1 value is (req attribute) <%=request.getAttribute("result1");

Result 2 value is (ses.attribute) <%=request.getAttribute("result2");
```

#### Note

Failure.jsp can read both request and session attributes where as success.jsp can read only session attribute values and gives null when it reads request attribute value.

- Attribute is logical name that holds java object as an value and can carry data from one web-resource program to another web-resource program like action class to result page.
- To write above code in success.jsp/failure.jsp to make those Jsp programs as Java codeless Jsp programs we can work with the struts supplied bean, logic tag library tags.
- The <bean:write> can be used to read any scope attribute value where as <logic:notEmpty> tag can be used to check whether given attribute value is null/empty value or not.
- We can write code in success.jsp/failure.jsp as shown below to make them as Java code less Jsp program.

### Success.Jsp/Failure.Jsp:

```
<%@tagliburl="http://struts.apache.org/tags-bean" prefix="bean"%>
<%@tagliburl="http://struts.apache.org/tags-logic" prefix="logic"%>
<logic:notEmpty name="results"scope="request">
    Result1 is (req after): <bean:write name="result1" scope="request" format=""/> <br>
</logic:notEmpty>
<logic:notEmpty name="result2" scope="session">
    Result2 is if(ses):<bean:write name="result2" scope="session" format=" "/>
</logic:notEmpty>
```

In <logic:notEmpty>, <bean:write> the scope option is optional. Because these tags can verify for the given attributes in multiple scopes.(request, session and application scopes):<bean:write name="result1">

</logic:notEmpty>

### The equivalent java code for the above code

```
if(pageContext.findAttribute("result1")!=null)
{
    Result1 is (req attribute):
    out.println(pageContext.findAttribute("result");
}
```

When scope attribute is not specifier the <logic:notEmpty> & <bean:write> tag internally use pageContext.findAttribute(-) to verify the attribute in multiple scopes.

We can use <bean:write> tag to read form data from any scoped form bean class object.

### In success.Jsp/failure.Jsp:

```
<b> the given form data is </b> <br>
<bean:write name="rf" property="username"/><br>
<bean:write name="rf" property="password"/><br>
```

**Q) How to display result generated by struts action class in the form page itself.**

**Ans:** Make the form page of struts action class as the result page struts action class as shown below with respect to first application.

**Step-1**

Configure register.jsp as the result page of RegisterAction class in struts-config.xml.

```
<action mapping>
  <action path="/register" type="appRegisterAction" name="rf">
    <forward name="myres" path="/register.jsp"/>
  </action>
</action mapping>
```

**Step-2**

Write the following code in RegisterAction class to deal with single result page.

```
public ActionForward execute(ActionMapping mapping, ActionForm form)
{
    -----
    -----
    if(user.equals("sathya") && pass.equals("tech"))
    {
        req.setAttribute("resultmsg", "validCredentials");
    }
    else
    {
        req.setAttribute("resultmsg", "invalidCredentials");
    }
    return mapping.findForward("myres");
} //execute.
```

**Step-3**

Write following code in register.jsp after </html.form> tag:

```
<logic:notEmpty name="resultmsg">
  Result is:<bean:write name="resultmsg"/>
</logic:notEmpty>
```

We can make Action servlet of struts app location as front controller either with directory match or extension match url pattern.

**Procedure to configure Action servlet as front controller having directory match url pattern****Step-1**

Specify url pattern for Action servlet configuration in web.xml file as shown below.

**In web.xml**

```
<url-pattern>/x/y/*</url-pattern>
```

**Step-2**

Take following action url in the form page in (register.jsp)

```
<html:form action="/x/y/register" method="get">
.....
.....
</html:form>
```

**Step-3**

Configure registration class having the following Action path.

**In struts-config.xml file**

```
<action path="/x/y/register" type="app.RegisterAction" name="rf" ....>
.....
.....
</action>
```

**Q) Find out number of requests coming to struts application?**  
(or)

**Q) Can you develop your own servlet as the replace for ActionServlet?**

**Ans:** Yes, the right place to place this counter logic is Action servlet, but the action servlet is pre-defined servlet so we develop user-defined ActionServlet extending from pre-defined ActionServlet class as shown below. Develop your own ActionServlet as shown below.

**//MyAsSrv.java(same in WEB-INF/class folder)**

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import org.apache.struts.action.*;
public class MyAsSrv extends ActionServlet
{
    int cnt;
    public void do get(HttpServletRequest req, HttpServletRequest res) throw
    {
        cnt ++;
        ServletContent sc=get ServletContent();
        sc.setAttribute("counter",new.Integer(cnt));
        //call doGet(-,-) of pre-defined ActionServlet
        super.doGet(req,res);
    }
}
```

**Step-2**

Configure the above servlet in web.xml file as front controller by having extension match or directory match url pattern.

```
<web-app>
<servlet>
<servlet-name> myaction</servlet-name>
<servlet-class> MyAsSrv</servlet-class>
<load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>myaction</servlet-name>
<url-path> *do </url-path>
</servlet-mapping>
</web-app>
```

**Step-3**

Display counter value in any JSP page in register.jsp

```
<logic:notEmpty name= "counter">
No.of visits: <bean:write name= "counter"/>
</logic:notEmpty>
```

**Step-4**

Execute the application in normal manner

**Key points**

- ✓ In traditional HTML tags form page the default request method is "Get".
- ✓ In Struts supplied JSP tags based form page the default request method is "Post".

**Working with properties file**

The text file that maintains entries in the form of (key=value) pairs is called as properties file

**Example****myfile.properties**

```
#text properties file (comment)
My.name=raja1
My.age=30
```

The standard principle of software industry is don't hard code (type) any value in your java resources that are changeable in the future. To make them as flexible to modify collect them from properties file.

In struts application, properties file is useful for the following

- a) To gather presentation logic labels from outside the jsp programs and to make them re-usable in multiple jsp programs.
- b) To make jsp code or any code of struts Action Class as flexible code to modify and place driver class name, url, username & password details of JDBC code in properties file.
- c) To maintain form validation error messages.

**Procedure to add properties file to struts application having presentation logic labels of the form page JSP program****Step-1**

Add properties file in 'WEB-INF/classes' folder of struts App as shown below.

**myfile.properties** → Mandatory extension

```
# maintains presentation login related takes
my.title =<center> <b>welcome</b></center>
my.uname=username
my.pwd=password
my.btn.cap=checkdetails.
```

**Step-2**

Configure properties file in struts configuration file.

**In struts-config.xml (After </action-mapping>)**

```
<message-resources parameter="myfile"/>
```

**Step-3**

Read presentation logic labels from properties file being from JSP program as shown below.

**register.jsp**

```
<bean:message key="my.title"/><br>
<html:form action="register">
<bean:message key="my.uname"/><html:text property="username">
<br>
<bean:message key="my.pwd"/><html:password property="password">
<br>
<html:submit>
<bean:message key="my.btn.cap"/>
</html:form>
```

In format server the modification done in properties will be affected only after the reloading application.

Work with multiple properties files if there is need of keeping more than 500 messages in the properties file of struts application.

**Procedure to work with multiple properties files in struts applications****Step-1**

Prepare multiple properties files as shown below.

**myfile.properties (available in WEB-INF/classes folder)**

```
#maintains presentation logic related lables
my.title =<center><b>welcome</b></center>
my.uname=username
```

**myfile1.properties (available in 'app' package of WEB-INF/class folder)**

```
my.pwd=give password
my.btn.cap=verify details
```

**Step-2**

Configure these properties files in struts-config file having logical names.

**In struts-config.xml**

```
<message-resources key="f1" parameter="myfile"/>
<message-resources key="f2" parameter="myfile1"/>
```

**Step-3**

Use messages of properties files in jsp program by identifying the properties files through its logical name

**In register.jsp**

```

<html:html>
  <bean:message key="my.title" bundle="f1"/><br>
  <html:form action="register">
    <bean:message key="my.uname" bundle="f1"/>
    <html:text property="username:/><br>
    <bean:message="my.btn.cap" bundle="f2"/>
    <html:password property="password"/><br>
    <html:submit>
    <bean:message key="my.btn.cap" bundle="f2"/>
    </html:submit>
  </html:form>
</html:html>

```

**Note**

- Don't keep more than 500 lines in one property file for better performance
- Properties file is also known as resource bundle file or application resource file.

The Netbean IDE related struts project comes with one default properties file called "ApplicationResource.properties", having automatic configuration in struts configuration file.

**To add another properties file to this project follow the procedure.**

Expand project → right click on source package → new → other → other → properties file → next → filename → Finish.

**Note:** The explicit properties file must be configured in struts configuration file explicitly.

**Procedure to make JDBC code of struts action class as flexible code to modify by placing JDBC details like driver classname, url, DB username, pwd in properties file:**

**With respect to first application of Netbeans IDE****Step-1**

Add properties file to source packages and configure that properties file in struts configure file explicitly.

**In servlet-config file**

```
<message-resources key="f1" parameter="myfile"/>
```

**Step-2**

Add the following JDBC details to the properties file.

**myfile.properties**

```

#jdbc details
my.jdbc.driver=oracle.jdbc.driver.OracleDriver
my.jdbc.url=jdbc.oracle:thin:@localhost:1521:XE
my.dbuser=scott
my.dbpwd=tiger

```

**Step-3**

Write following code in the execute (-,-,-) of registration class to create Jdbc connection object.

**//Get Access to specific properties file**

```
MessageResources msgs=this.getResources(request, "f1");
```

**//Read data from properties file**

```
String dname=msgs.getMessage("my.jdbc.driver");
```

```
String url=msgs.getMessage("my.jdbc.url");
```

```
String uname=msgs.getMessage("my.dbuser");
```

```
String pwd=msgs.getMessage("my.dbpwd");
```

**//Write jdbc code**

```
Class.forName(dname);
```

```
Connection con=DriverManager.getConnection(url,uname,pwd);
```

```
PreparedStatement ps=con.prepareStatement("select count (*) from userlist where uname=? and pwd=?");
```

Based on the component type in the form page we need to choose the property type in the FormBean class.

- For text box, password box, Radio button, Combo box, Single Check box text area component take simple string type property in form bean class.
- For list box/ multiple checkboxes having same name components take string [] type property in form bean class.

**Example application to understand different form components behavior.****Resources**

- ✓ MyForm.jsp (form page)
- ✓ MyForm.java (form bean class)
- ✓ MyFormAction.java (action class)
- ✓ Result.jsp (result page)
- ✓ Web.xml
- ✓ servlet-config.xml

**myform.jsp**

```
<%@taglib url="http://struts.apache.org/tags-html" prefix="html"%>
<html:form action="mypath" method="get">
Name:<html:text property="name"/><br>
Age:<html:password property="age"/><br>
Gender: <html:radio property="g1" value="m"/>male
       <html:radio property="g2" value="f"/>female <br>
Maritalstatus <html:checkbox property="ms"/> married <br>
Qualification: <html:select property="qlfy">
  <html:option value="engg">be/btech</html:option>
  <html:option value="pg">mca/ms </html:option>
  <html:option value="arts">b.a</html:option>
</html:select><br>
Courses <html:select property="crs" multiple="yes">
  <html:option value="java"> JAVA package </html:option>
  <html:option value=".net"> .NET package </html:option>
  <html:option value="db"> Database package</html:option>
</html:select>
<html:submit value="send"/>
</html:form>
```



**MyForm.java**

```

public class MyForm extends org.apache.struts.action.ActionForm
{
    //form bean properties
    String name;
    String age;
    String gt;
    String ms;
    String qlfy;
    String crs[];
    //Generate getter & setter methods.
    -----
    -----
}

```

**MyFormAction.java**

```

public class MyFormAction extends org.apache.action.ActionForm
{
    public ActionForward execute(-,-,-)
    {
        MyForm fm=(MyForm)form;
        String name=fm.getName();
        String age=fm.getAge();
        String gen=fm.getGt();
        String ms=fm.getMs();
        String qlfy=fm.getqlfy();
        String crs[]=fm.getcrs();
        System.out.println ("name=" +name+ " .....");
        System.out.println ("crs["]=");
        for(int i=0; i<crs.length; ++i)
        {
            System.out.println (crs[i] + " .....");
        }
        return mapping.findForward("res");
    }
}

```

**result.jsp**

<b> observe server console to know the form data </b>

**Note**

- Configure result.jsp as result page to MyFormAction class having logical name "res"  
<forward name="res" path="/result.jsp"/>
- Except checkbox, listbox, the remaining form components of form page send default values (atleast empty string) as request parameter values when form page is submitted even though they are not selected / filled up. (This will not happen in case of checkbox and list box)
- When checkbox and listbox items are not selected in the form page their related set Xxx(-) in FormBean class will not be executed. For other components the related setXxx(-) will be executed when they are selected / filled up or not.

## Problem Scenario

In session scoped FormBean, the FormBean properties of checkbox, list box may show selected values even though they are not selected by end user while generating other than first request from browser window.

## Example Scenario code

myform.jsp (form page)

Name  (1a)

Marital Status ☒ married

Name  (2a)

Marital Status ☐ married

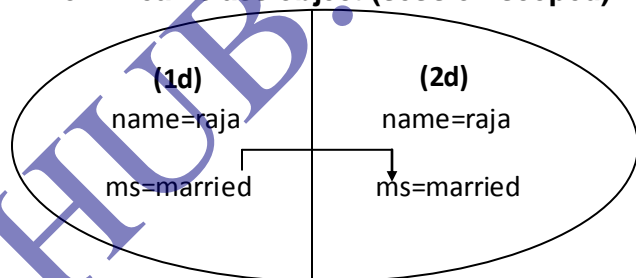
Case-1

Case-2

MyForm.java(formbeanclass) (1b) (11b)

```
public class Myform extends ActionForm
{
    String name;
    String ms;
    (1C) (11C)
    public void setXxx(-) → 2nos
    public String getXxx() → 2nos
}
```

FormBean class object (session scoped)



(1a-1d) → Shows 1<sup>st</sup> request flow, (2a-2d) → Shows 2<sup>nd</sup> request flow.

- For second request setMs(-) method will not executed because the checkbox is not selected in that request.
- Both 1<sup>st</sup> & 2<sup>nd</sup> requests are utilizing single FormBean class object because the FormBean scope is session.
- The problem with above code is the “ms” property of FormBean object shows value married (selected state) even though it is not selected in the second request that means the ms property is holding first request state data for second request when checkbox is not selected.

To solve the above problem there are two solutions.

### Solution-1

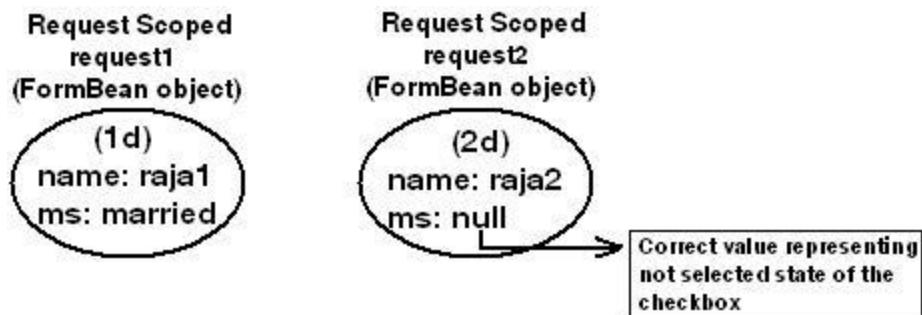
Change the FormBeanScope to “request” from ‘session’.

### Solution-2

Work with reset () method even though FormBean Scope is ‘session’ scope.

**Solution-1**

Solves the problem because one new form bean object will be created for every request so it cannot hold previous request data in form bean properties for current request.

**Solution-2**

The reset (-,-) method of form bean class execute & for every request generated by form page, before executing the setXxx(-) methods of FormBean Class. This method is very useful for the programmer to specify the not selected state values for checkbox, listbox related form bean properties while working with session scoped form bean.

This behavior solves the problem of above scenario as show below.

**myform.jsp (form page)**

Name  (1a)

Marital Status ☒ married

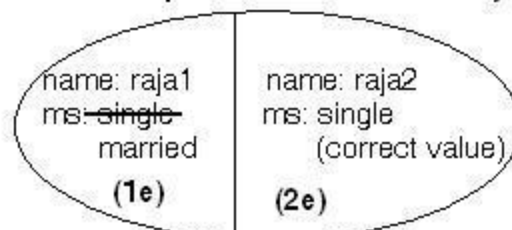
**Case-1**

Name  (2a)

Marital Status ☐ married

**Case-2****MyForm.java(FormBean Class)**

```
public class MyForm extends ActionForm
{
    String name;
    String ms;
    public void reset (-,-)
    {
        ms="single"; (1c) (2c)
    }
    public void set xxx(-) → 2nos
    public String getxxx(-) → 2nos } (1d) (2d)
}
```

**Session Scoped FormBean class object**

Solution-2 is quite recommended to solve the problem of above given scenario reset (-,-) is available in pre-defined ActionForm class having the following two overloaded form.

- 1) public void reset (ActionMapping mapping, ServletRequest req)
- 2) public void reset (ActionMapping mapping, HttpServletRequest req)
- 3) Form-2 is recommended house.

When reset(-,-) method is overridden in our form bean class to place not selected state values for list box, the code should be written as shown below.

```
public class MyForm extends ActionForm
{
    String ms;    //check box property
    String crs[]; //list box property
    public void reset (-,-)
    {
        Ms="single";
        crs=new string [1];
        crs[0]="no course is selected";
    }
    public void setxxx(-) {.....}
    public string getxxx(-) {.....}
}
```

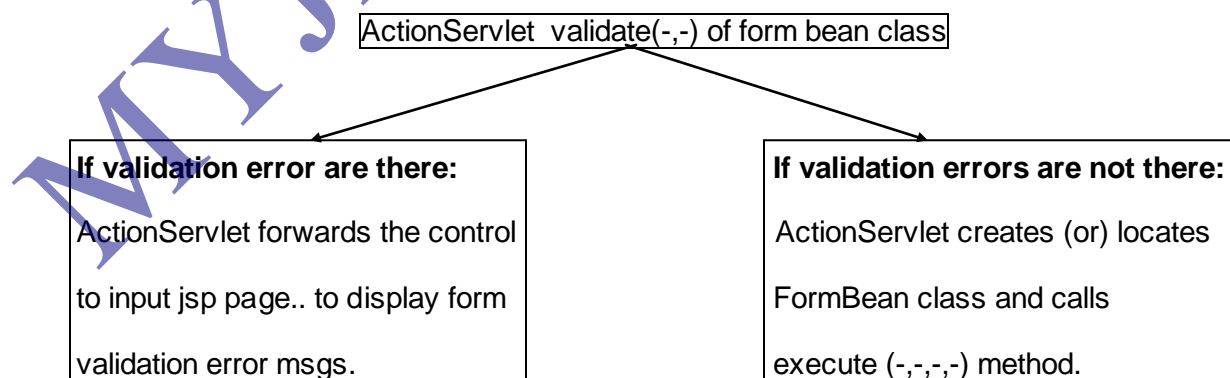
### Understanding Life Cycle of FormBean:

- Enduser generates request from Formpage by using submit button.
- ActionServlet creates/locates the form page related FormBean class objects.

#### Note

If created formbean class obj will be placed in the specified scope.

- ActionServlet calls reset(-,-) of FormBean class.
- ActionServlet calls setxxx() of formbean class to write the received form components data to formbean class properties.
- ActionServlet calls validate(-,-) of form bean class.



The JSP page that is capable of displaying form validation error messages is called as input JSP program. Generally it is recommended to take the form jsp program as input jsp program.

## Form Validations

The process of verifying pattern and formate of the form data before it is getting used as input value in Business logic is called as Form validation and such logic is called as Form Validation logic.

### Example

- ✓ Checking weather required fields are typed or not.
- ✓ Checking weather email id is having "@" and "." Symbols.

### Q) What is the difference between Form validation logic and Business logic?

A) Form validation logic verifies pattern and format of the form data where as business logic generates results by taking form data as input values.

### Example

- ✓ Checking whether username and passwords are typed (or) not comes under Form validation logic.
- ✓ Checking whether given username and passwords are correct (or) not against database table comes under business logic.

#### Form validations in classic web application (Non Struts Application)

Client side	Server side
Place java script code or VB script code (java script is recommended)	Write java code before business logic in servlet, jsp programs.

- Prefer working with client side form validations. Because it executes in browser window and avoids unnecessary network roundtrips between browser window and web-server towards form validations.
- Since there is a chance of disabling client side script code execution (like java script) through settings by end user, the developers are preferring to write both client side and server side form validation logic but they executes only when client side form validation logic is disabled through browser settings.
- Without validating form data if it is used in business logic, the business logic may generate wrong result.
- The scripting language java script is no where related with programming language java.
- When validator plugin is used for client side form validations, it will dynamically generate java script code. When validate plugin use for server side form validation it will dynamically generate java code.
- Programmatic form validation means the programmer should write the form validation logic manually.
- Declarative form validation means programmer should work with readily available built-in form validation logics.
- Struts will allow to mix-up both programmatic and declarative form validation logic to satisfy the application requirement towards form validations.

- In struts application, plugin will be activated to perform its task only when that plugin is configured in struts configuration file. Validator plugin is purely given to perform form validations in struts applications in declarative mode.
- In java class when no constructors are placed explicitly the java compiler automatically generates zero-argument constructor as default constructor. Otherwise this default constructor will not be generated.

**Q) Can I place only parameterized constructors in our FormBean class and Action class of struts Application?**

A) Not Possible. Reason is ActionServlet uses zero-argument constructor while creating objects for FormBean, Action classes. So the programmer must make sure that these FormBean, Action classes are getting zero-argument constructor, explicitly (or) implicitly.



**The prototypes of validate (-,-) methods**

- 1) public ActionErrors validate(ActionMapping mapping, HttpServletRequest req)
- 2) public ActionErrors validate(ActionMapping mapping, ServletRequest req)

**Note:** Form-1 is recommended.

To write server-side programmatic form validation logic for form page. Keep your form validation logic in the form of java statement by over-riding validate(-,-) method in FormBean class.

When ActionServlet calls validate(-,-) method. The method returns ActionErrors class object (internally a Map data structure). If this object size is greater zero (when validations errors are there) then, ActionServlet transfers the control to input Jsp to display form validation error messages. If ActionErrors class object size is zero (No validation errors) then ActionServlet calls execute(-,-,-) method of Action class.

ActionErrors class represents all form validation errors generated by validate(-,-) method. In that each ActionError (ActionMessage class object) represent one form validation error with error message.

**Note:** ActionError class is not there from struts 1.3

**Procedure to perform programmatic server side form validations on first struts application**

**Step-1**

Configure properties file in struts configuration file by keeping that file in WEB-INF/classes folder.

**In struts-config.xml**

```
<message-resources parameter= "myfile"/>
```

**Step-2**

Write following form validation error messages in properties file (myfile.properties)

**myfile.properties**

```
#form validation error messages for required, alphabet
my.un.req.err=User name is required
my.pwd.req.err=Password is mandatory
my.un.alphabet.err=User name must begin with an alphabet
my.pwd.alphabet.err=Password must begin with an alphabet
```

**Step-3**

Write the following form validation logic(Java code based) in the validate(-,-) method of RegisterForm class.

**In RegisterForm.java**

```
public ActionErrors validate(ActionMapping mapping,HttpServletRequest request)
{
    ActionErrors errs=new ActionErrors();
    //form validation logic (server side)
    System.out.println("RegisterForm:validate(-,-)");
    if(username==null || username.equals("") || username.length()==0)
        errs.add("unerr",new ActionMessage("my.un.req.err");
    else
    {
        //check weather first character is alphabet or not
        char fchar=username.charAt(0);
        If(!Character.isUpperCase(fchar)&&Character.isLowerCase(fchar))
        {errs.add("unerr",new ActionMessage("my.un.alphabet.err");
        }
        if(password==null || password.equals("") || password.length()==0)
        errs.add("pwderr",new ActionMessage("my.pwd.req.err");
        else
        {
            //check weather first character is alphabet or not
            char fchar=password.charAt(0);
            If(!Character.isUpperCase(fchar)&&Character.isLowerCase(fchar))
            {
                errs.add("pwderr",new ActionMessage("my.pwd.alphabet.err");
            }
        }
        System.out.println("err object size is :"+errs.size());
        return errs;
    }
}
```

Key in properties file representing form validation errors messages

Logical name of validation error

**Note:** Recompile FormBean class

**Step-4**

Configure input Jsp for Action class to display the generated Form validation error messages. It is recommended to take the form jsp program as input Jsp program.

**In struts-config.xml**

```
<action input= "/register.jsp" path= "/register" type= "app.RegisterAction" name= "rf">
.....
.....
</action>
```

form page as input page

**Step-5**

Add <html:errors/> tag in input Jsp program to decide the position of displaying form validation error messages.

Add <html:errors/> tag in register.jsp





**In myfile.properties**

```
errors.header=<font color= "red" size=2>
errors.footer=</font>
```

**Step-3**

Use <html:errors> tag with property attribute as shown below.

**In register.jsp**

```
<html:form action= "register" method= "get">
Username: <html:text property= "username"/><html:errors property= "username"/><br>
Password: <html:password property= "password"/><html:errors property= "password"/><br>
<html:submit value= "Check Details">
</html:form>
```

Logical names given for form validation errors  
in the validate (-,-) method of FormBean class.

**Note**

Perform all these operations with respect to that struts application that deals with server-side form validations.

- validate= "true/false" attribute kept in action tag can pass instructions to ActionServlet to call/ignore validate(-,-) method of FormBean class during the life cycle of FormBean class.
- This gives control to programmer to enable or disable server-side validations temporarily.
- The default value of this attribute is "true".

**Sample code in struts-config.xml**

```
<action input="/register.jsp" path="/register" type="app.RegisterAction" name="rf"
validate= "true/false">
-----
-----
</action>
```

**Client side programmatic form validations****Note**

For the above operation, write JavaScript code in the form page against "onSubmit" event as shown below.

- In form page, we generally perform form based form validations. That means we need to activate JavaScript code for form validations when submit button is clicked.
- Against "onSubmit" event, always call the form validation logics based JavaScript function along with "return" statement.

**In resgister.jsp**

```
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
<html:html>
<head>
```

```

<script language= "JavaScript">
function myValidate(frm)
{
//read form data
var unval=frm.username.value;
var pwdval=frm.password.value;
//client side form validation logic (programmatic)
If(unval== " ")
{
    alert("User name required");
    frm.username.focus();
    return false;
}
else
{
    var fchar=unval.charAt(0);
    if(!isNaN(fchar))
    {
        alert("first character of username must be an alphabet");
        frm.username.focus();
        frm.username.value= " ";
        return false;
    }
    //if
    //else
    If(pwdval== " ")
    {
        alert("Password is required")
        frm.password.focus();
        retrun false;
    }
    else
    {
        var fchar=pwdval.charAt(0);
        if(!isNaN(fchar))
        {
            alert("first character of password must be alphabet");
            frm.password.focus();
            frm.password.value= " ";
            return false;
        }
        // if
        // else
        return true;
    }
}
}
//myValidate(-)
</script>
</head>
<html:form action= "/register" method= "get" onSubmit= "return myValidate(this)">
Username:<html:text property= "username"/>
<html:errors property= "unerr"/><br>
Password:<html:password property= "password"/>
<html:errors property= "pwderr"/><br>
<html:submit value= "CheckDetails"/>
</html:form>
</html:html>

```

Mandatory

JavaScript function call against  
onSubmit event

return myValidate(this)

Represents current form related object.

- ❖ onSubmit= "return myValidate(this)">
- ❖ In the above statement the return keyword takes the return value of myValidate(-) JavaScript function call.(true/false) and gives to browser software. If this browser

software gets true indicating no form validation errors are there, then request goes to server otherwise, browser window blocks/stops request going to servlet.

- ❖ To guaranty perform form validations even though the client side script code execution is disabled. It is recommended to place both client side and server side form validation logics in struts applications.
- ❖ To disable script code execution in Internet Explorer the procedure is Tools menu -> Internet Options -> Security tab -> Custom -> Scripting -> Action Scripting -> Disable
- ❖ To disable script code in Netscape Navigator Edit menu -> Preferences -> Advanced -> Scripts & Plugins -> EnableJavaScript -> Deselect navigation.
- ❖ To hide script code based form validation logic from end user separate java script code for the form page as shown below.

**Note:** Place the below validation is file along with register.jsp

#### Validation.js

```
function myValidate (frm)
{
    .....
    .....
    .....
}
```

#### In register.jsp

```
.....
.....
<head>
<script language ="JavaScript" src="Validation.js">
</script>
</head>
```

#### Working With ValidatorPlugin:

- ❖ Plugin is a patch software or software application that can enhance functionalities and facilities of existing software or software application.
- ❖ In struts environment validator plugin is given to perform declarative mode form validations at client side and server side, because this validator plugin supplies set of pre-defined form validation logics.
- ❖ Every struts plugin contains one plugin class implementing "org.apache.struts.action.Plugin" Interface.
- ❖ Validator plug-in related plug-in class is "org.apache.struts.validator.ValidatorPlugin" class.
- ❖ Even though validator plugin, tiles plugin are built-in plugins of struts software to activate and utilize them in struts application, their plugin classes must configured in struts configuration file as show below.

### In Struts – Config.xml

```
<struts – config>
.....
.....
<plugin-in> className = "org.apache.struts.validator.ValidatorPlugin"
<set-property="pathnames" value = "/WEB-INF/Validator-rules.xml,/WEB-INF/validation.xml"/>
</plug-in>
</struts.config>
```

- ❖ To work with ValidatorPlugIn supplied pre-defined form validation logics (14+ rules), programmer must develop this FormBean class by extending from "org.apache.struts.validator.ValidatorForm class" (It is the sub-class of pre-defined ActionForm class)
- ❖ When ActionServlet calls Validate(-,-) of our FormBean class, the programmatic form validations takes place.
- ❖ When ActionServlet calls validate(-,-) of super class (pre-defined ValidatorForm class), then the ValidatorPlugIn based declarative form validations will take place.

#### Q) When ValidatorPlugIn will be recognized and will be activated in our struts application.

**Ans:** When ActionServlet reads struts configuration file, the ValidatorPlugIn will be configuration file, the ValidatorPlugIn will be activated. Generally this takes place after instantiation of ActionServlet class either during server startup or during the deployment of struts application.

This ValidatorPlugIn will be activated for form validations when ActionServlet calls Validate() of pre-defined "Validator Form" class through our FormBean class object directly or Indirectly.

- ❖ ValidatorPlugIn supplied the following pre-defined validation rules having both java, JavaScript based form validation logics.

They are: required, minlength, maxlength, creditcard, email, date, range and etc. (14+)

- ❖ This Validator – rules.xml contains configuration details about ValidatorPlugIn supplied pre-defined validator rules. Those details are
  - a) Validator rule names.
  - b) Methods and java class that contains java code based form validation logics.
  - c) Keys in properties file representing form validation error messages.
  - d) The '.Js' file names containing java script based from validation logics for validator rules.

**Note:** This file can be extracted from: struts\_home\lib\ struts\_core-1.3.8.jar file extraction.

- ❖ Since validator rules and their validation logics are pre-defined, the ValidatorPlugIn has given pre-defined error messages having fixed keys like:

errors.required = {0} is required.

errors.minlength = {0} cannot be less than {1} characters.

errors.maxlength = {0} cannot be less than {1} characters.

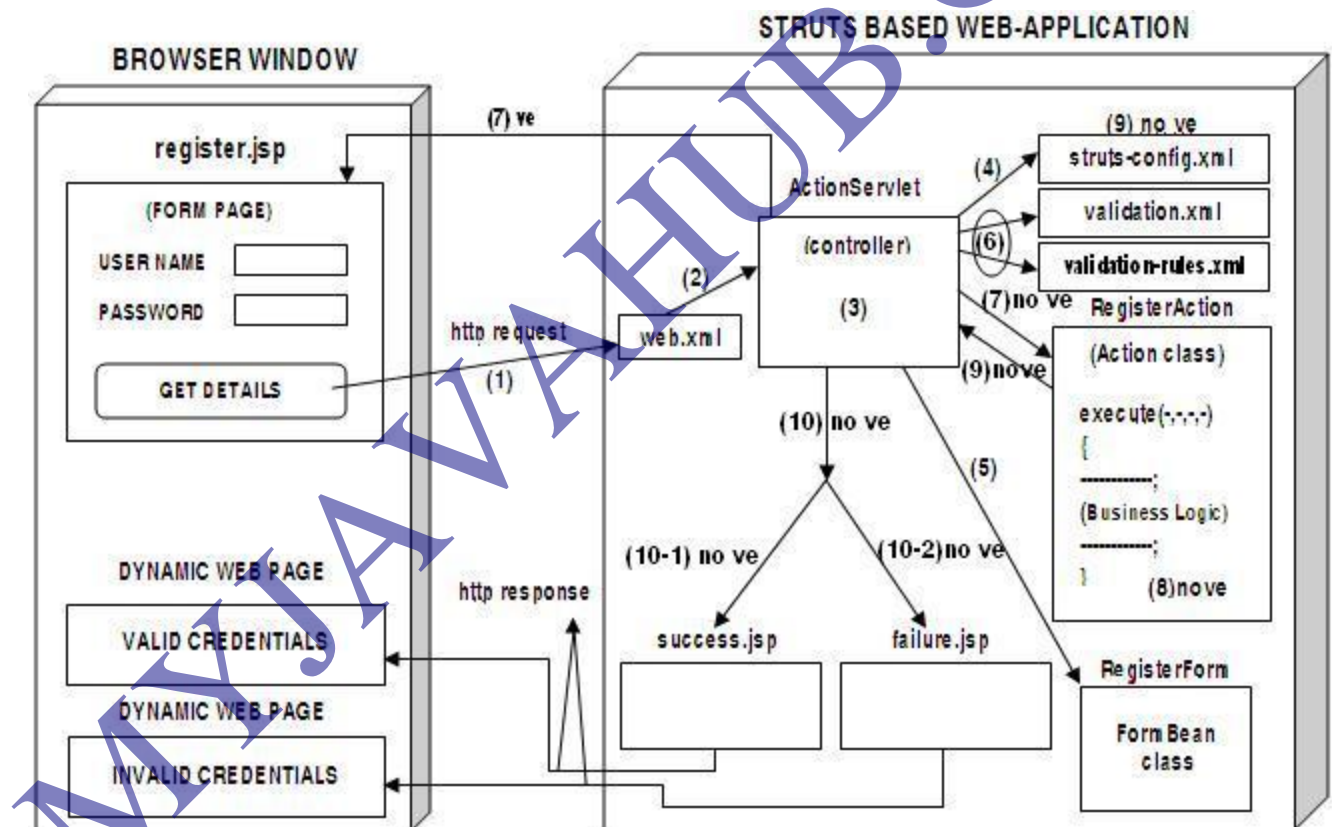
.....

.....

14 + messges.

**Note:** The above 14+ messages are given with arguments to form complete error messages for different form components, when validator rules are applied on different FormBean properties.

- ❖ This validation.xml file contains programmer supplied entries to apply validator rules on form bean properties. This file also supplies argument values of form validation error messages.
- ❖ The activated ValidatorPlugIn takes the support of Validation.xml file instructions to apply specific validator rules on form bean properties and to validate the form data.
- ❖ When ValidatorPlugIn is used for server side validations, the java code of validator rules will be activated. Similarly javascript code will be activated when validator plugin is configured for client side validation.
- ❖ ValidatorPlugIn comes along with struts software in the form of two jar files representing java code.
  - 1) Struts – core – 1.3.8 jar
  - 2) Commons – validator – 1.3.1. jar



**Procedure to perform declarative mode ValidatorPlugIn based server-side form validations on struts application with respect to first struts application (Struts Applications1).**

#### Step-1

Configure validatorPlugIn struts configuration file.

**In struts-config.xml**

After </action-mapping> add the following code must be placed

```
<plug-in className="org.apache.struts.validator.ValidatorPlugin">
<set-property property="pathnames"
            value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
</plug-in>
```

**Step-2**

Keep validator-rules.xml in WEB-INF folder by gathering that file from struts-core-1.3.8.jar file extraction.

**Step-3**

Add properties file to WEB-INF/classes folder and configure that file in struts configuration file.

**In struts-config.xml**

Before <plugin>tag

```
<message-resources parameter="myfile"/>
```

**Step-4**

Add default error messages related the validator rules of ValidatorPlugin in properties file.

**Note:** Collect these messages from the comments of validator-rules.xml

**In myfile.properties**

```
errors.required = {0} is required (14 error messages)
```

```
.....
```

```
.....
```

```
errors.email={0} is an invalid e-mail address.
```

**Step-5**

Make the form bean class of struts application extending from "org.apache.struts.validator.ValidatorForm" class.

**Note**

The regular pre-defined ActionForm class based form bean class cannot work with validator plugin supplied validator rules.

- ❖ Make sure that in your FormBean class validate(-,-) method is not defined explicitly. This is required to perform to make ActionServlet calling validate(-,-) method of our FormBean class super class (That is pre-defined validator form class.)
- ❖ validate (-,-) method of pre-defined ActionForm class cannot activate ValidatorPlugin where as the validate(-,-) method of pre-defined ValidatorForm class can do this work.

**Step-6**

Add validation.xml file in WEB-INF folder to apply the required Validator rule on FormBean class properties.

**In validation.xml**

```

<form-validation>
  <formset>
    <form name="rf">
      <field property="username" depends="required">
        <arg.key="my.un"/>
      </field>
      <field property="password" depends="required">
        <arg.key="my.pwd"/>
      </field>
    </form>
  </formset>
</form-validation>

```

**Note:** In the above file username, password FormBean properties are configured with required validator rule.

**Step-7**

Add user defined messages in properties file as specified in validation.xml file to supply argument values of error messages.

**In myfile.properties:**

```

# user-defined messages to supply {n} values of above error messages
my.un=Login Username
my.pwd=Login Password
supply {0} value of "required" rule related error messages.

```

**Step-8**

Configure input page for Action class.

**In struts-config.xml**

```

<action input="/register.jsp" path="/register" type="app.RegisterAction" name="rf">
  .....
  .....
</action>

```

**Step-9**

Add <html:errors> tag in input jsp program.

**In register.jsp**

```

<html:errors/>

```

**Step-10**

Develop the remaining resources of struts application in regular manner.

**Note:** Test the application in normal manner.

### Q) How ValidatorPlugIn performs form validations in the above steps based example application.

**Ans:** After trapping request given by form page, the ActionServlet reads the form data -> ActionServlet creates or locates FormBean class object based on struts configuration file entries-> ActionServlet calls reset (-,-), set Xxx(-) methods to populate Formdata to FormBean class properties-> ActionServlet calls validate (-,-) of our FormBean class since that method is not available, super class validate(-,-) will execute (ValidationForm class) -> This validate (-,-) activates the ValidatorPlugIn and reads the instructions from validation.xml file -> After executing Form validation logic, this validate method returns Action Error class object if this ActionErrors class object's size is zero, ActionServlet sends the control to execute(-,-) of Action class. Otherwise ActionServlet sends the control to input jsp program to display FromValidation error messages.

For the above steps based complete example application refer the following application.

### Struts application with form validation

#### register.jsp

```
<%@taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
<%@taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
<%@taglib uri="http://struts.apache.org/tags-logic" prefix="logic" %>
<% System.out.println("register.jsp");%>
<bean:message key="my.Title"/>
<html:form action="register" method="GET">
  <bean:message key="my.un"/><html:text property="username"/>
  <html:errors property="username"/><br>
  <bean:message key="my.pwd"/><html:text property="password"/>
  <html:errors property="password"/><br>
  <html:submit>
    <bean:message key="my.btn.cap"/>
  </html:submit>
</html:form>
<logic:notEmpty name="msg" scope="request">
  The Result is: <bean:write name="msg" scope="request"/>
</logic:notEmpty>
```

#### web.xml

```
<web-app>
  <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>2</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
</web-app>
```



**struts-config.xml**

```

<!DOCTYPE struts-config PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
    "http://jakarta.apache.org/struts/dtds/struts-config_1_3.dtd">
<struts-config>
    <form-beans>
        <form-bean name="rf" type="app.RegisterForm"/>
    </form-beans>
<action-mappings>
    <action input="/register.jsp" name="rf" path="/register" type="app.RegisterAction">
        <forward name="result" path="/register.jsp"/>
    </action>
</action-mappings>
<message-resources parameter="myfile"/>
<!--===== Validator plugin ===== -->
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
    <set-property property="pathnames"
        value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
</plug-in>
</struts-config>

```

**validation.xml**

```

<!DOCTYPE form-validation PUBLIC
    "-//Apache Software Foundation//DTD Commons Validator Rules Configuration 1.1.3//EN"
    "http://jakarta.apache.org/commons/dtds/validator_1_1_3.dtd">
<form-validation>
    <formset>
        <form name="rf">
            <field property="username" depends="required">
                <arg key="my.un"/>
            </field>
            <field property="password" depends="required">
                <arg key="my.pwd"/>
            </field>
        </form>
    </formset>
</form-validation>

```

**RegisterForm.java**

```

package app;
import org.apache.struts.validator.ValidatorForm;
public class RegisterForm extends ValidatorForm
{
    //FormBean properties
    String username="xyz",password;
    public RegisterForm()
    {
        System.out.println("0-arg constructor: RegisterForm"+this.hashCode());
    }
    public String getPassword()

```

```

    {
        System.out.println("getPassword: RegisterForm");
        return password;
    }
    public void setPassword(String password)
    {
        System.out.println("setUsername: RegisterForm");
        this.password = password;
    }
    public String getUsername()
    {
        System.out.println("getUsername: RegisterForm");
        return username;
    }
    public void setUsername(String username)
    {
        System.out.println("setPassword: RegisterForm");
        this.username = username;
    }
}

```

### RegisterAction.java

```

package app;
import javax.servlet.http.*;
import org.apache.struts.action.*;
public class RegisterAction extends Action
{
    public RegisterAction()
    {
        System.out.println("0-arg constructor: RegisterAction");
    }
    public ActionForward execute(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response)
        throws Exception
    {
        System.out.println("Business logic in execute(-,-,-):RegisterAction"+this.hashCode());
        RegisterForm fm=(RegisterForm)form;
        //read form data
        String uname=fm.getUsername();
        String pwd=fm.getPassword();
        System.out.println("Control is leaving execute(-,-,-) of RegisterAction");
        //Write Business logic
        if(uname.equals("sathya") && pwd.equals("java"))
        {
            request.setAttribute("msg", "Valid Credentials");
            return mapping.findForward("result");
        }
        else
        {
            request.setAttribute("msg", "Invalid Credentials");
            return mapping.findForward("result");
        }
    }
} //execute
} //class

```

**myfile.properties**

```
# -- standard errors --
errors.header=<font color=red>
errors.footer=</font>
# -- validator --
errors.invalid={0} is invalid.
errors.maxLength={0} can not be greater than {1} characters.
errors.minLength={0} can not be less than {1} characters.
errors.range={0} is not in the range {1} through {2}.
errors.required={0} is required.
errors.byte={0} must be an byte.
errors.date={0} is not a date.
errors.double={0} must be an double.
errors.float={0} must be an float.
errors.integer={0} must be an integer.
errors.long={0} must be an long.
errors.short={0} must be an short.
errors.creditcard={0} is not a valid credit card number.
errors.email={0} is an invalid e-mail address.
# -- other --
my.un=Enter Username
my.pwd=Enter Password
my.un1=Login Username
my.btn.cap=Login
my.Title=<b><center>Welcome to Sathya</center></b>
```

- ❖ We can use some old pre-defined keys based error messages in properties file, like errors.header, errors.footer, errors.pre-fix and errors.suffix to performed by validator plugin.
- ❖ While working with ValidatorPlugin, the validator plugin generated validation errors, automatically takes the form bean class property names as logical names. So, while displaying Form validation errors with respect to form component positions of form page, we must specify FormBean property names in <html:errors> tag as shown below.

**In register.jsp**

```
<html:form action="register" method="get"> username:<html:text property="username"/>
    <html:errors property="username"/><br>password:<html:password property="password"/>
    <html:errors property="password"/><br>
    <html:submit value="checkdetails"/>
</html:form>
```

- ❖ FormBean class property names as logical names of validator plugin generated validation errors.

**Performing Client-side form validation through ValidatorPlugin on struts application:****Step-1**

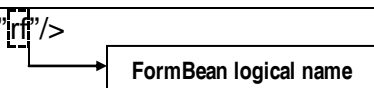
Make sure that all the 10 steps are completed on struts application that are related to ValidatorPlugin based server side declarative form validation.

## Step-2

Add `<html:javascript/>` in the form page that is also acting as input JSP page.

### In register.jsp

`<html:javascript formName="rf"/>`



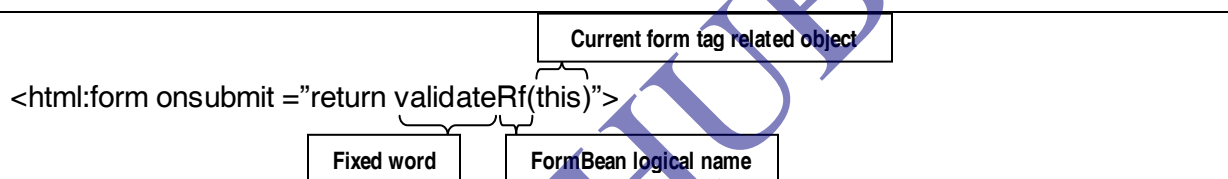
### Note:

- The above statement adds the ValidatorPlugIn generation Javascript code to the Formpage or insert page.
- In this process the validatorPlugIn gives javascript function to perform ClientSide Form Validations having this notation in the function name.
- Validate `<FormBean logical name>` (-) if form bean logical name is "rf" then that javascript function will be validate rf(-)

## Step-3

Call the above generated special Javascript function against onSubmit event in the form page current form tag selected object.

`<html:form onsubmit="return validateRf(this)">`



### Complete register.jsp(with respect to above steps)

```
<%@taglib uri="http://struts.apache.org/tags-html" prefix="html"%>
<html:form action="register" on submit return validateRf(this)>
<html:javascript formName="rf"/>
Username:<html:text property="username"/><html:errors property="username"/><br>
Password:<html:text property="password"/><html:errors property="password"/><br>
<html:submit value="CheckDetails"/>
</html:form>
```

- ❖ ValidatorPlugIn automatically enables serverside form validations when client side Formvalidations are not performed, because the enduser has disabled script code execution (like javascript code) through browser window settings.
- ❖ The 'validate' attribute of `<action>` tag in struts configuration file which can be there with true/false value can enable/disable only serverside programmatic and declarative form validations. But, it cannot enable or declarative form validations. Because the script code executes at client side. It is well before the execution of `validate(-,-)` in the Form Bean class.
- ❖ If argument of validator rule related error message is just there to complete the error message then pass that argument value from properties file.
- ❖ If argument of error message is not just there to complete the error message and when it is also required as input value of that validator rule related form validation logic. Then pass that argument value from 'validation.xml' file itself.

## Example

Minlength rule error message is errors.minlength={0} cannot be less than {1} characters when that min length rule is applied on username formbean properties for five number of minimum characters. Then the error messages will be "login username cannot be less than 5 characters".

- ❖ Here {0} value is just required to complete the error message. So, pass it from properties file.
- ❖ Here {1} value (5) is required as input value of form validation logic (minlength rule) and also required to complete the error message. So, pass it from "validation.xml" file itself.

**Applying required, minlength validator rules on username formbean property in "validation.xml" file:**

### In Validation.xml

```
<field property="username" depends="required, minlength">
  <arg 0 key="my.un"/>
  <arg 1 name="minlength" key="{var:minlength}" resource="false"/>
</field>
```

(1) (2) (3) (4)

```
<var>
  <var-name> minlength </var-name>
  <var-value> 4 </var-value>
</var>
```

(5)

- 1) my.un is the key in the properties file supplying {0} value for required, minlength rules related error messages to just complete the error messages.
- 2) minlength is the validator rule name.
- 3) "var:minlength" where minlength is the variable name that contain arg 1/{1} of minlength rule in "validation.xml" file itself.
- 4) Resource="false" indicates that this argument value ({1} value of minlength rule) is not collected from properties file (resource bundle).
- 5) Supplies minlength variable value as {1} value of minlength validator rule.

### Note

Generally {0} value of validator rule error messages will be supplied from properties file and remaining argument values like {1}, {2} will be supplied from validation.xml file.

- ❖ When arg<n> tag is taken (<arg0>, <arg1>, <arg2>) without specifying validator rule name. Then, it supplies argument value for multiple validator rules that are configured on formbean property.

```
<field property="username" depends="required, minlength">
  <arg0 key="myun"/>
  .....
  .....
</field>
```

Supplies {0} value of both required, minlength validator rule messages.

- ❖ When `arg<n>` tag is taken by specifying validator rule names then, that tag supplies argument value of that specified validator rule related error message.

```
<field property="username" depends="required minlength">
```

```
<arg 0 name="required" key="my.un"/>
```

This `<arg0>` tag supplies only {0} value of required rule error message.

```
<arg 0 name="minlength" key="my.un1"/>
```

This `<arg0>` tag supplies only {0} value of minlength rule error message.

```
.....  
</field>
```

- ❖ We can take all `arg<n>` tags without argument number (As `<arg>`) in validation.xml from struts 1.3 version.
- ❖ While supplying {1} value for minlength, maxlength rules, the variable name taken in validation.xml file must have same name of validator rule name.

Applying required, minlength, maxlength validator rules on password form bean property in validation.xml file based on above given statement.

#### In validation.xml

```
<field property="password" depends="required, maxlength, minlength">
```

```
<arg name="required" key="my.pwd"/> {0} value of required.
```

```
<arg name="maxlength" key="my.pwd1"/> {0} value of maxlength.
```

```
<arg name="minlength" key="my.pwd2"/> {0} value of minlength.
```

```
<arg name="maxlength" key="{var:maxlength}" resource="false"/>
```

```
<var>
```

```
<var-name>maxlength</var-name>
```

```
<var-value>10</var-value>
```

```
</var>
```

{1} value of maxlength rule.

```
<arg name="min length" key="{var:minlength}" resource="false"/>
```

```
<var>
```

```
<var-name>minlength</var-name>
```

```
<var-value>50</var-value>
```

```
</var>
```

{1} value of minlength rule.

```
</field>
```

- ❖ The ValidatorPlugIn supplied pre-defined validator rules (14+) are not at all sufficient in real world project development. For this there is a provision of executing these validator rules along with the programmer supplied custom/programmatic form validation logics. For this, there are three approaches to follow.

#### Approach-1

Mask rule

#### Approach-2

Call Super validate(-,-) from your validate (-,-) of formbean class.

### Approach-3

Add custom validator rule in validator plugin

**Note:** we can apply all the three approaches at a time in single struts-application to mixup our form validation logics with the validator plugin supplied validator rules.

#### Mask rule

- ❖ Mask is the pre-defined validator rule of ValidatorPlugIn that allows the programmer to specify user-defined form validation logic in the form of regular expression.
- ❖ The ValidatorPlugIn generates, the necessary java code, javascript code that is required for form validation based on the given regular expression.
- ❖ The default error message of mask rule is errors invalid but, programmer can take his own choice error message for this mask rule with the support of <msgs> tag.
- ❖ Regular expression allows the programmer, to define the complex logics in a simple manner.
- ❖ Every regular expression will starts with 'V' symbol and ends with '\$' symbol.

**Applying Required, Mask validator rules on the formbean properties.**

**In validator.xml**

```
<field property="username" depends="required.mask">
```

```
<arg0 key="my.un"/>
```

Supplies {0} value of mask, validator rule error messages.

```
<msg name="mask" key="my.mask.msg"/>
```

Defines user-defined error message for mask rule.

```
<var>
  <var-name>mask</var-name>
  <var-value>^[0-9a-zA-Z]{5}$</var-value>
</var>
```

```
</field>
```

Regular expression that allows exactly 5 number of alphanumerics

- ❖ Regular expression based form validation logic for mask rule.

**Note:** Add the following message in properties file.

**In myfile.properties**

```
my.mask.msg={0} must contain exactly 5 alphanumeric character
```

**Regular expression for mask rule to allow data in AbCd...notation.**

```
<var>
  <var-name>mask</var-name>
  <var-value>^[A-Z][a-z])*$</var-value>
</var>
```

- ❖ Mask rule can perform both server side and client side validations. That means it can generate both java code and java script code based on the given regular expression code. For related information of regular expression Regular Expression text file of STV folder DVD.

### Limitation with Mask rule

- ❖ Mask rule and other pre-defined validator rules of validator plugin can deal with only one form bean property data at a time to perform form validations, that means these rules cannot be utilized for the following form validations.
  - a) Type pressured, Retype password must match.
  - b) Username and password must have same length.
  - c) Username and password must begin with same character.

All these are dealing with more than one FormBean data.

- ❖ For above requirements and to write any kind of custom form validation logic use approach-2 (call super validate method from validate(-,-) method of your formbean class.

Example to write both client side and server side custom form validation logic by mixing them with the validate plugin generated client side and server side form validation logics.

### Step-1

Complete all the formalities to work with validator plugin based server side form validation (10 steps of 9<sup>th</sup> July).

### Step-2

Keep validate (-,-) in our FormBean class, calling super. Validate(-,-) inside that method definition.

### Note:

Make sure that your FormBean class is extending from pre-defined validator Form class.

### In Register Form java

```
public ActionErrors validate(-,-)
{
    // call super.validate(-,-) to get validation errors.
    // given by ValidatorPlugIn
    ActionErrors errs=super.validate(mapping,req);
    // write custom form validation logic and add validation errors
    if(username.length ()!=password.length()) // username and password must have same
    length.
    {
        errs.add("lenerr",new ActionMessage("my.unpwd.len"));
    }
    System.out.println("size errors objects "+errs.size());
    //return ActionErrors object to ActionServlet
    return errs;
} // validate(-,-)
}
```



**Step-3**

Add the following additional error message in properties file.

**In my file properties**

My.unpwd.len=Username and password must have same length.

**Step-4**

Write following code in Formpage / input page calling ValidatorPlugIn generated javascript from user-defined javascript function. (To mixup custom client side form validation logic with the ValidatorPlugIn generated form validation logic).

**In register.jsp**

```
<html>
<head>
<script language="JavaScript">
function myValidate(frm)
{
    //call ValidatorPlugIn generated javascript function
    var result = validateRf(frm);
    if(result == false)
    {
        return false;
    } //if
    else
    {
        // read form data for writing custom form validation logic
        var unval = frm.username.value;
        var pwdval=frm.password.value;
        If (unval.length!=pwdval.length)
        {
            alert("username and password must have same length");
            frm.username.focus();
            return false;
        } //if
        return true;
    } //else
} //myvalidate(-,-)
</script>
</head>

<html:javascript formName="rf"/>
<html:form action="register" on submit="return myvalidate (this)">
.....
.....
</html:form>
<html:error/>
```

In order to work with ValidatorPlugIn related server side declarative form validation our formbean class must be extended from validator form class. To perform ValidatorPlugIn related clientside declarative form validations our form bean class can take ActionForm or Validator Form as super class.

- ❖ The validation logic of Mask rule is specific to that FormBean property for which it is configured.
- ❖ Form validation logic written in approach-2 is specific to one form page / FormBean class, to make user defined form validation logic visible for multiple FormBean classes / Form pages of struts application add custom validator rule in validator plugin.

### Approach-3

#### Procedure to add custom validator rule in ValidatorPlugIn

#### Note:

This custom validator rule can work with only one form data at a time while writing the formvalidation logic.

#### Step-1

Define error message in properties file related to you custom validator rule.

#### In my file properties

my.errors.customrule={0} beign with an alphabet.

#### Step-2

Develep java class having the code of custom validation logic in WEB-INF/classes folder as shown below.

#### **MyCustRule.java (Develop with reference to pre-defined field check class)**

```

import org.apache.commons.validator.*;
import org.apache.struts.validator.*;
import org.apache.commons.validator.util.*;
import org.apache.struts.action.*;
import javax.servlet.http.*;

public class MyCustRule
{
    public Static boolean validate FirstChar(Object bean, ValidatorAction va, Fiedl field)
    {
        System.out.println("validate FirstChar(-,-) MyCustRule var");
        String value=ValidatorUtils.getValue AsString(bean,field, getproperty());
        //read first char
        Char fchar=value.charAt(0);
        //check wheather first char is alphabet or not
        If (Character.is UpperCase (fchar) && !(Character.isLowerCase(fchar))
        {
            errors.add(field.get Key()).Resources.getActionMessage (Validator,request,va, field));
            return false;
        }
        else
        {
            return true
        }
    } // method
}

```

**Step-3**

Add following jar file in the class part and compile the above MyCustRule.java file.

- 1) Servlet-api.jar
- 2) Struts-core-1.3.8.jar
- 3) Commons-validator-1.3.1.jar

**Step-4**

Configure this custom validator rule in validator-rules.xml file.

**In WEB-INF/validator-rules.xml file**

Under<global>

```
<validator name="alphabetrule" classname="MyCustRule" method="validateFirstChar"
method params="-,-,-,-,-"
msg="my.errors.custrule"/>
```

**Note:**

In the above validator rule configuration no java script file name is configured. That means the above custom rule is developed purely for server side form validation.

**Note:**

The step-1-4 talks about custom validator rule development.

**Step-5**

Apply the above custom validator rule (alphabet rule) in validation.xml file on any form bean property.

**In validation.xml**

```
<field property="username" depends="required, alphabet rule">
<arg0 key="my.un"/>
</field>
<field property="password" depends="required, alphabet rule">
<arg0 key="my.pwd"/>
</field>
```

**Assignment**

For the above custom Validator rule alphabet rule add java script code based client side form validation login.

- ❖ You can apply all the three approaches at a time on single struts application to mix up our form validation logics with validator plugin.

For example application that shows above statements refer the following application.

**Struts Application with all the 3 approaches of adding custom form validation logics****register.jsp**

```

<%@taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
<%@taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
<%@taglib uri="http://struts.apache.org/tags-logic" prefix="logic" %>
<% System.out.println("register.jsp");%>
<bean:message key="my.Title"/>
<html:form action="register" method="GET">
    <bean:message key="my.un"/><html:text property="username"/>
    <html:errors property="username"/><br>
    <bean:message key="my.pwd"/><html:text property="password"/>
    <html:errors property="password"/><br>
    <html:submit>
        <bean:message key="my.btn.cap"/>
    </html:submit>
</html:form>
<logic:notEmpty name="msg" scope="request">
    The Result is: <bean:write name="msg"/>
</logic:notEmpty>

```

**web.xml**

```

<web-app>

<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>2</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
</servlet-mapping>
<welcome-file-list>
    <welcome-file>register.jsp</welcome-file>
</welcome-file-list>

</web-app>

```

**struts-config.xml**

```

<!DOCTYPE struts-config PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
    "http://jakarta.apache.org/struts/dtds/struts-config_1_3.dtd">
<struts-config>
    <form-beans>
        <form-bean name="rf" type="app.RegisterForm"/>
    </form-beans>
</struts-config>

```

```

</form-beans>
<action-mappings>
  <action input="/register.jsp" name="rf" path="/register" type="app.RegisterAction">
    <forward name="result" path="/register.jsp"/>
  </action>
</action-mappings>
<message-resources parameter="com/myapp/struts/ApplicationResource"/>
<!-- =====Validator plugin ===== -->
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
  <set-property
    property="pathnames"
    value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
</plug-in>
</struts-config>

```

### validation.xml

```

<!DOCTYPE form-validation PUBLIC
  "-//Apache Software Foundation//DTD Commons Validator Rules Configuration 1.1.3//EN"
  "http://jakarta.apache.org/commons/dtds/validator_1_1_3.dtd">
<form-validation>
  <formset>
    <form name="rf">
      <field property="username" depends="required,alphanumeric">
        <arg key="my.un"/>
      </field>
      <field property="password" depends="required,mask">
        <arg key="my.pwd"/>
        <msg name="mask" key="my.err.mask.pwd"/>
        <var>
          <var-name>mask</var-name>
          <var-value>^[0-9a-zA-Z]*$</var-value>
        </var>
      </field>
    </form>
  </formset>
</form-validation>

```

Diagram illustrating the validation rules configuration:

### validator-rules.xml

```

<!DOCTYPE form-validation PUBLIC
  "-//Apache Software Foundation//DTD Commons Validator Rules Configuration 1.3.0//EN"
  "http://jakarta.apache.org/commons/dtds/validator_1_3_0.dtd">
<form-validation>
  <global>

```

```
<validator name="alphanumeric"
  classname="MyCustomrValidation"
  method="myFormLogic"
  methodParams="java.lang.Object,
    org.apache.commons.validator.ValidatorAction,
    org.apache.commons.validator.Field,
    org.apache.struts.action.ActionMessages,
    org.apache.commons.validator.Validator,
    javax.servlet.http.HttpServletRequest"
  msg="my.err.un.fchar"/>

<validator name="required"
  classname="org.apache.struts.validator.FieldChecks"
  method="validateRequired"
  methodParams="java.lang.Object,
    org.apache.commons.validator.ValidatorAction,
    org.apache.commons.validator.Field,
    org.apache.struts.action.ActionMessages,
    org.apache.commons.validator.Validator,
    javax.servlet.http.HttpServletRequest"
  msg="errors.required"/>

<validator name="requiredif"
  classname="org.apache.struts.validator.FieldChecks"
  method="validateRequiredIf"
  methodParams="java.lang.Object,
    org.apache.commons.validator.ValidatorAction,
    org.apache.commons.validator.Field,
    org.apache.struts.action.ActionMessages,
    org.apache.commons.validator.Validator,
    javax.servlet.http.HttpServletRequest"
  msg="errors.required"/>

<validator name="validwhen"
  msg="errors.required"
  classname="org.apache.struts.validator.validwhen.ValidWhen"
  method="validateValidWhen"
  methodParams="java.lang.Object,
    org.apache.commons.validator.ValidatorAction,
    org.apache.commons.validator.Field,
    org.apache.struts.action.ActionMessages,
    org.apache.commons.validator.Validator,
    javax.servlet.http.HttpServletRequest"/>

<validator name="minlength"
  classname="org.apache.struts.validator.FieldChecks"
  method="validateMinLength"
  methodParams="java.lang.Object,
    org.apache.commons.validator.ValidatorAction,
    org.apache.commons.validator.Field,
    org.apache.struts.action.ActionMessages,
```

```
        org.apache.commons.validator.Validator,  
        javax.servlet.http.HttpServletRequest"  
depends=""  
msg="errors.minlength"  
jsFunction="org.apache.commons.validator.javascript.validateMinLength"/>  
  
<validator name="maxlength"  
    classname="org.apache.struts.validator.FieldChecks"  
    method="validateMaxLength"  
    methodParams="java.lang.Object,  
        org.apache.commons.validator.ValidatorAction,  
        org.apache.commons.validator.Field,  
        org.apache.struts.action.ActionMessages,  
        org.apache.commons.validator.Validator,  
        javax.servlet.http.HttpServletRequest"  
    depends=""  
    msg="errors.maxlength"  
    jsFunction="org.apache.commons.validator.javascript.validateMaxLength"/>  
  
<validator name="mask"  
    classname="org.apache.struts.validator.FieldChecks"  
    method="validateMask"  
    methodParams="java.lang.Object,  
        org.apache.commons.validator.ValidatorAction,  
        org.apache.commons.validator.Field,  
        org.apache.struts.action.ActionMessages,  
        org.apache.commons.validator.Validator,  
        javax.servlet.http.HttpServletRequest"  
    depends=""  
    msg="errors.invalid"/>  
  
<validator name="byte"  
    classname="org.apache.struts.validator.FieldChecks"  
    method="validateByte"  
    methodParams="java.lang.Object,  
        org.apache.commons.validator.ValidatorAction,  
        org.apache.commons.validator.Field,  
        org.apache.struts.action.ActionMessages,  
        org.apache.commons.validator.Validator,  
        javax.servlet.http.HttpServletRequest"  
    depends=""  
    msg="errors.byte"  
    jsFunctionName="ByteValidations"/>  
  
<validator name="short"  
    classname="org.apache.struts.validator.FieldChecks"  
    method="validateShort"  
    methodParams="java.lang.Object,  
        org.apache.commons.validator.ValidatorAction,  
        org.apache.commons.validator.Field,  
        org.apache.struts.action.ActionMessages,
```

```
        org.apache.commons.validator.Validator,  
        javax.servlet.http.HttpServletRequest"  
        depends=""  
        msg="errors.short"  
        jsFunctionName="ShortValidations"/>  
  
<validator name="integer"  
    classname="org.apache.struts.validator.FieldChecks"  
    method="validateInteger"  
    methodParams="java.lang.Object,  
        org.apache.commons.validator.ValidatorAction,  
        org.apache.commons.validator.Field,  
        org.apache.struts.action.ActionMessages,  
        org.apache.commons.validator.Validator,  
        javax.servlet.http.HttpServletRequest"  
    depends=""  
    msg="errors.integer"  
    jsFunctionName="IntegerValidations"/>  
  
<validator name="long"  
    classname="org.apache.struts.validator.FieldChecks"  
    method="validateLong"  
    methodParams="java.lang.Object,  
        org.apache.commons.validator.ValidatorAction,  
        org.apache.commons.validator.Field,  
        org.apache.struts.action.ActionMessages,  
        org.apache.commons.validator.Validator,  
        javax.servlet.http.HttpServletRequest"  
    depends=""  
    msg="errors.long"/>  
  
<validator name="float"  
    classname="org.apache.struts.validator.FieldChecks"  
    method="validateFloat"  
    methodParams="java.lang.Object,  
        org.apache.commons.validator.ValidatorAction,  
        org.apache.commons.validator.Field,  
        org.apache.struts.action.ActionMessages,  
        org.apache.commons.validator.Validator,  
        javax.servlet.http.HttpServletRequest"  
    depends=""  
    msg="errors.float"  
    jsFunctionName="FloatValidations"/>  
  
<validator name="double"  
    classname="org.apache.struts.validator.FieldChecks"  
    method="validateDouble"  
    methodParams="java.lang.Object,  
        org.apache.commons.validator.ValidatorAction,  
        org.apache.commons.validator.Field,  
        org.apache.struts.action.ActionMessages,
```



```
        org.apache.commons.validator.Validator,  
        javax.servlet.http.HttpServletRequest"  
        depends=""  
        msg="errors.double"/>  
  
<validator name="byteLocale"  
    classname="org.apache.struts.validator.FieldChecks"  
    method="validateByteLocale"  
    methodParams="java.lang.Object,  
        org.apache.commons.validator.ValidatorAction,  
        org.apache.commons.validator.Field,  
        org.apache.struts.action.ActionMessages,  
        org.apache.commons.validator.Validator,  
        javax.servlet.http.HttpServletRequest"  
    depends=""  
    msg="errors.byte"/>  
  
<validator name="shortLocale"  
    classname="org.apache.struts.validator.FieldChecks"  
    method="validateShortLocale"  
    methodParams="java.lang.Object,  
        org.apache.commons.validator.ValidatorAction,  
        org.apache.commons.validator.Field,  
        org.apache.struts.action.ActionMessages,  
        org.apache.commons.validator.Validator,  
        javax.servlet.http.HttpServletRequest"  
    depends=""  
    msg="errors.short"/>  
  
<validator name="integerLocale"  
    classname="org.apache.struts.validator.FieldChecks"  
    method="validateIntegerLocale"  
    methodParams="java.lang.Object,  
        org.apache.commons.validator.ValidatorAction,  
        org.apache.commons.validator.Field,  
        org.apache.struts.action.ActionMessages,  
        org.apache.commons.validator.Validator,  
        javax.servlet.http.HttpServletRequest"  
    depends=""  
    msg="errors.integer"/>  
  
<validator name="longLocale"  
    classname="org.apache.struts.validator.FieldChecks"  
    method="validateLongLocale"  
    methodParams="java.lang.Object,  
        org.apache.commons.validator.ValidatorAction,  
        org.apache.commons.validator.Field,  
        org.apache.struts.action.ActionMessages,  
        org.apache.commons.validator.Validator,  
        javax.servlet.http.HttpServletRequest"  
    depends=""
```

```
msg="errors.long"/>

<validator name="floatLocale"
  classname="org.apache.struts.validator.FieldChecks"
  method="validateFloatLocale"
  methodParams="java.lang.Object,
    org.apache.commons.validator.ValidatorAction,
    org.apache.commons.validator.Field,
    org.apache.struts.action.ActionMessages,
    org.apache.commons.validator.Validator,
    javax.servlet.http.HttpServletRequest"
  depends=""
  msg="errors.float"/>

<validator name="doubleLocale"
  classname="org.apache.struts.validator.FieldChecks"
  method="validateDoubleLocale"
  methodParams="java.lang.Object,
    org.apache.commons.validator.ValidatorAction,
    org.apache.commons.validator.Field,
    org.apache.struts.action.ActionMessages,
    org.apache.commons.validator.Validator,
    javax.servlet.http.HttpServletRequest"
  depends=""
  msg="errors.double"/>

<validator name="date"
  classname="org.apache.struts.validator.FieldChecks"
  method="validateDate"
  methodParams="java.lang.Object,
    org.apache.commons.validator.ValidatorAction,
    org.apache.commons.validator.Field,
    org.apache.struts.action.ActionMessages,
    org.apache.commons.validator.Validator,
    javax.servlet.http.HttpServletRequest"
  depends=""
  msg="errors.date"
  jsFunctionName="DateValidations"/>

<validator name="intRange"
  classname="org.apache.struts.validator.FieldChecks"
  method="validateIntRange"
  methodParams="java.lang.Object,
    org.apache.commons.validator.ValidatorAction,
    org.apache.commons.validator.Field,
    org.apache.struts.action.ActionMessages,
    org.apache.commons.validator.Validator,
    javax.servlet.http.HttpServletRequest"
  depends="integer"
  msg="errors.range"/>
```

```
<validator name="longRange"
  classname="org.apache.struts.validator.FieldChecks"
  method="validateLongRange"
  methodParams="java.lang.Object,
    org.apache.commons.validator.ValidatorAction,
    org.apache.commons.validator.Field,
    org.apache.struts.action.ActionMessages,
    org.apache.commons.validator.Validator,
    javax.servlet.http.HttpServletRequest"
  depends="long"
  msg="errors.range"/>
```

```
<validator name="floatRange"
  classname="org.apache.struts.validator.FieldChecks"
  method="validateFloatRange"
  methodParams="java.lang.Object,
    org.apache.commons.validator.ValidatorAction,
    org.apache.commons.validator.Field,
    org.apache.struts.action.ActionMessages,
    org.apache.commons.validator.Validator,
    javax.servlet.http.HttpServletRequest"
  depends="float"
  msg="errors.range"/>
```

```
<validator name="doubleRange"
  classname="org.apache.struts.validator.FieldChecks"
  method="validateDoubleRange"
  methodParams="java.lang.Object,
    org.apache.commons.validator.ValidatorAction,
    org.apache.commons.validator.Field,
    org.apache.struts.action.ActionMessages,
    org.apache.commons.validator.Validator,
    javax.servlet.http.HttpServletRequest"
  depends="double"
  msg="errors.range"/>
```

```
<validator name="creditCard"
  classname="org.apache.struts.validator.FieldChecks"
  method="validateCreditCard"
  methodParams="java.lang.Object,
    org.apache.commons.validator.ValidatorAction,
    org.apache.commons.validator.Field,
    org.apache.struts.action.ActionMessages,
    org.apache.commons.validator.Validator,
    javax.servlet.http.HttpServletRequest"
  depends=""
  msg="errors.creditcard"/>
```

```
<validator name="email"
  classname="org.apache.struts.validator.FieldChecks"
  method="validateEmail"
```

```

methodParams="java.lang.Object,
    org.apache.commons.validator.ValidatorAction,
    org.apache.commons.validator.Field,
    org.apache.struts.action.ActionMessages,
    org.apache.commons.validator.Validator,
    javax.servlet.http.HttpServletRequest"
depends=""
msg="errors.email"/>

<validator name="url"
    classname="org.apache.struts.validator.FieldChecks"
    method="validateUrl"
    methodParams="java.lang.Object,
        org.apache.commons.validator.ValidatorAction,
        org.apache.commons.validator.Field,
        org.apache.struts.action.ActionMessages,
        org.apache.commons.validator.Validator,
        javax.servlet.http.HttpServletRequest"
    depends=""
    msg="errors.url"/>

<!--
    This simply allows struts to include the validateUtilities into a page, it should
    not be used as a validation rule.
-->
<validator name="includeJavaScriptUtilities"
    classname=""
    method=""
    methodParams=""
    depends=""
    msg=""
    jsFunction="org.apache.commons.validator.javascript.validateUtilities"/>

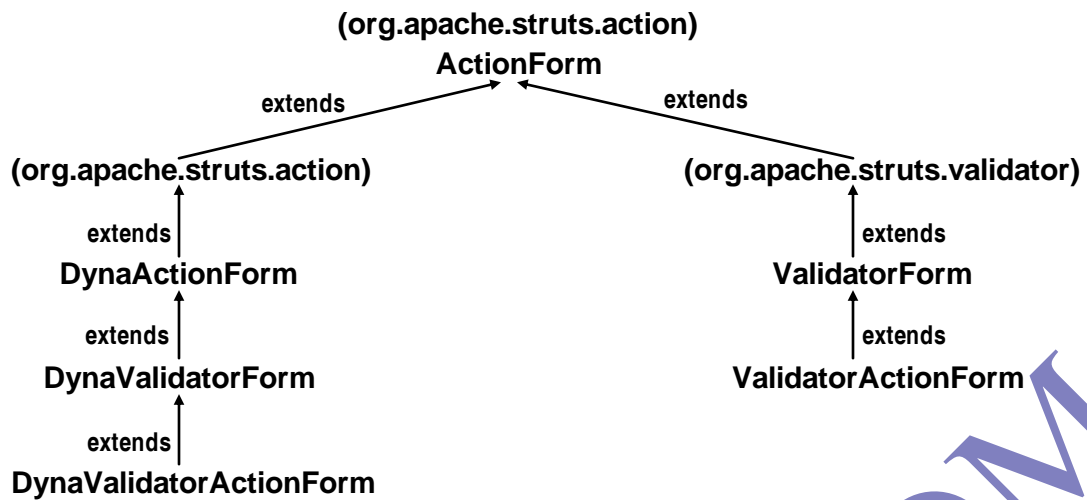
</global>
</form-validation>

```

**Summary of all the 3 approaches that are to mix up our form validation logics with ValidatorPlugin validator rule.**

	<b>Serverside validation</b>	<b>Client side validation</b>	<b>Visibility</b>	<b>ability to work with form data</b>
<b>Approach-1 (mask)</b>	With regular expressions	With automatically generated JavaScript	Specific to on FormBean property	At a time it can use one FormBean property data can be used
<b>Approach-2</b>	Explicit with java code of validate (-,-)	With explicit JavaScript code	Specific to one FormBean class	At a time one or more FormBean properties data can be used.
<b>Approach-3</b>	With explicit java code placed in user defined java class	With explicit java script code placed in '.js' file	One struts application level	Same as approach-1

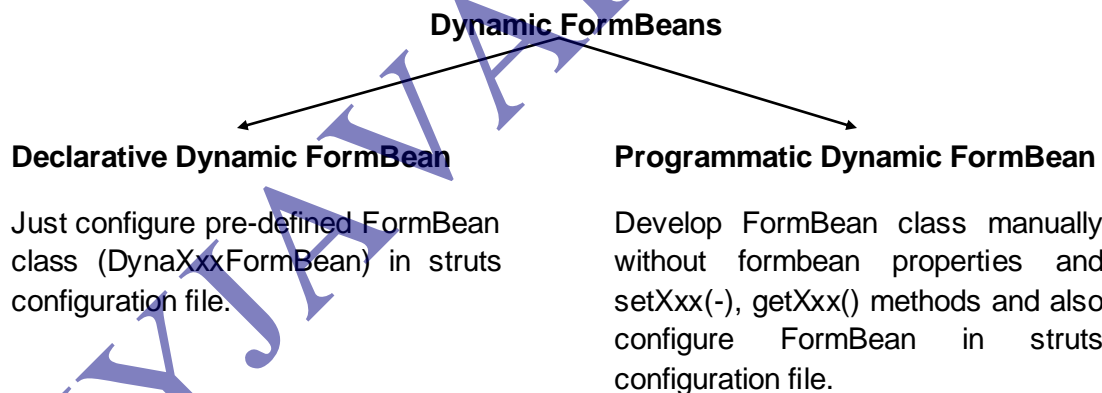
## Understanding the hierarchy related to predefined formbean classes



The FormBean class whose properties will be generated by struts Formwork software is called as Dynamic FormBean.

To work with dynamic form beans programmers must deal with DynaXxxForm classes (like DynaActionForm, DynaValidatorForm and etc.)

- ❖ ActionForm, DynaAction Form classes cannot work with ValidatorPlugIn.
- ❖ ValidatorForm, DynaValidatorForm classes can work with ValidatorPlugIn to perform FormValidations.
- ❖ The programmer can work with dynamic form beans in two modes.



## Working with declarative dynamic formbean in ValidatorPlugIn related struts application

### Step-1

Make sure that, no manually added FormBean class in there in ValidatorPlugIn related strut, application. (Delete Register Form.java, .class file.)

### Step-2

Configure the declarative dynamic formbean as shown below.

### Step-3

Replace already available formbean code with following code in struts configuration file.

```

<form-beans>
  <form-bean name="rf" type="org.apache.struts.validator.DynaValidatorForm">
    <form-property name="username" type="java.lang.String" initial="ravi"/>
    <form-property name="password" type="java.lang.string"/>
  </formbean>
</formbean>

```

Declarative Dynamic FormBean configuration

**Note:** FormBean properties must match with the form component names of formpage.

#### Step-4

Write following code in the execute (-,-,-) method of struts ActionClass to read form bean data from the above declarative dynamic form bean class object.

#### In execute(-,-,-) of Register Action Class

```

execute(-,-,-)
{
  -----
  // type casting
  DynaValidatorForm fm=(DynaValidatorForm) form;
  //read form data from FromBean class object
  String user=(string) fm.get("username");
  String pass=(string) fm.get("password");
  //write bussiness logic
  If(user.equals("sathya") && pass.equals("tech"))
  return mapping.find Forward ("success");
  else
  return mapping.find Forward ("failure");
}

```

#### Note:

Since the above dynamic form bean class name is "DynaValidatorForm" class perform all the above steps only after completing the basic 10 steps of "working with validator plugin".

#### Limitation with declarative DynamicFormBean

- ❖ Allows working with all pre-defined validator rules of ValidatorPlugIn and allows working with approach 1 and approach-3 to mixup custom form validator logics with ValidatorPlugIn related form validations. But does not allow to work with Approach-2 to perform the same.
- ❖ To overcome above problem to work with programmatic dynamic formbean. Which allows the programmer to develop the formbean class manually but FormBean properties, getXxx(), setXxx() will be generated by Framework Software dynamically.

## Procedure to work with programmatic dynamic frombean in validator plugin related struts application

### Step-1

Complete all the 10 steps related to ValidatorPlugIn based form validations.

### Step-2

Develop programmatic dynamic formbean as shown below extending from dynaxxx Form classes and over-ride validate(-,-) method showing approach-2

### Step-3

Compile the above FormBean class `javac -d.*.java`

### Note:

To compile to the above FormBean class, make sure that the following jar files are there in classpath environment variable.

- ➔ Servlet-api.jar (old)
- ➔ Struts-core-1.3.8.jar (old)
- ➔ Commons-validator-1.3.1.jar (old)
- ➔ Commons-bean utils-1.2.0.jar (new)

### Step-4

Make sure that the above given 4 jar file are also available in WEB-INF/lib folder in already added regular 10 jar file.

### Step-5

Configure the above form bean class in struts configuration file as shown below.

#### In struts-config.xml file

```
<form-beans>
<form-bean name="rf" type="app.RegisterForm">
<form-property name="username" type="java.lang.string" initial="ravi"/>
<form-property name="password" type="java.lang.string"/>
</form-bean>
</form-beans>
```

### Step-6

Write following code in the execute (-,-,-) of Action class to read formdata from formbean class object.

#### In execute (-,-,-) of RegisterAction class

```
//type captions
RegisterForm fm=(RegisterForm) form;
//read form data from FormBean class object
String user=(String)fm.get("username");
String pass=(String)fm.get("password");
//sub-config
.....
.....
```

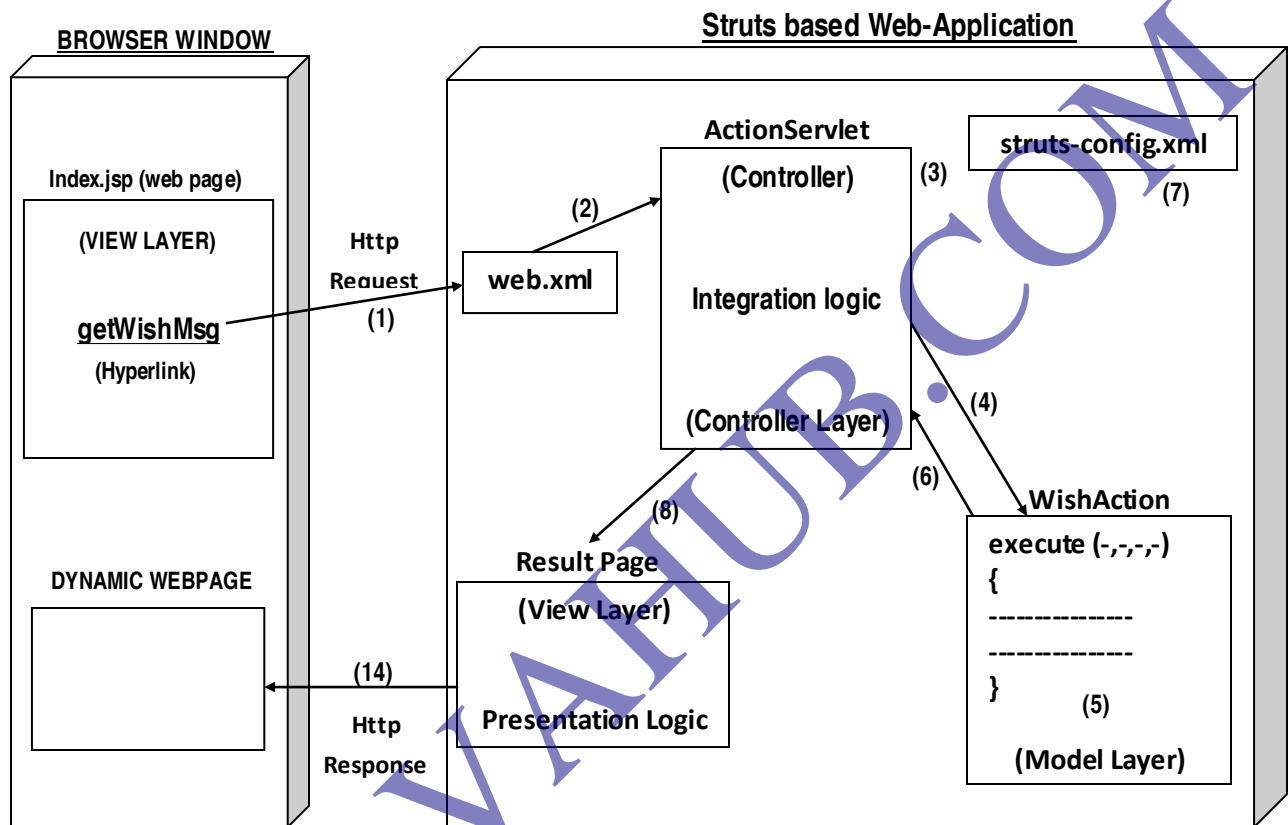
Dynamic FormBean are good when compared to working with manually developed formbean classes But it is recommended to work with programmatic dynamic formbeans in order to develop the application with all modes of Form validations.

**Q) Can I develop struts application without FormBean class.**

**Ans:** yes.

If struts application is getting request without request parameters (form data) then form bean is not required.

To process hyperlink generated request in struts Action class there is no need of FormBean class. To process form page generated request in struts Action class by performing Form validations then FormBean class is required.



**Q) What is the difference between Eclipse and My Eclipse IDEs ?**

Eclipse	My Eclipse
An open source IDE	Commercial IDE
Provides Basid environment to develop JSE module application	Allows to develop JSE, JEE, and other framework software application.
Doesn't provide built-in plugging to work with advanced technologies.	Provides built in plugins to work with advanced technology.
Suitable for small scale companies	Suitable for largeScale companies
Both IDEs allow to add user-defined plugins	EasyEclipse, Eclipse Galileo are other flavours of My Eclipse which are given based on Eclipse IDE.



## How to add plugins to work with advanced technologies in eclipse IDE

### Step-1

Download Eclipse Plugins for that advanced technologies in the form of jar files from internet.

### Step-2

Create project in eclipse IDE and observe the availability of plugins folder in that folder.

### Step-3

Add the above plugin related jar files to the plugin folder of projects.

### Step-4

Relaunch Eclipse IDE only to activate the plugins

### MyEclipse IDE

Type	: IDE for Java.
Version	: 8.x (compatible with JDK 1.6)
Vendor	: Eclipse org.
Software type	: Commercial software.
To download software	: <a href="http://www.myeclipseide.com">www.myeclipseide.com</a>
For help	: <a href="http://www.myeclipse.com">www.myeclipse.com</a>

### Note

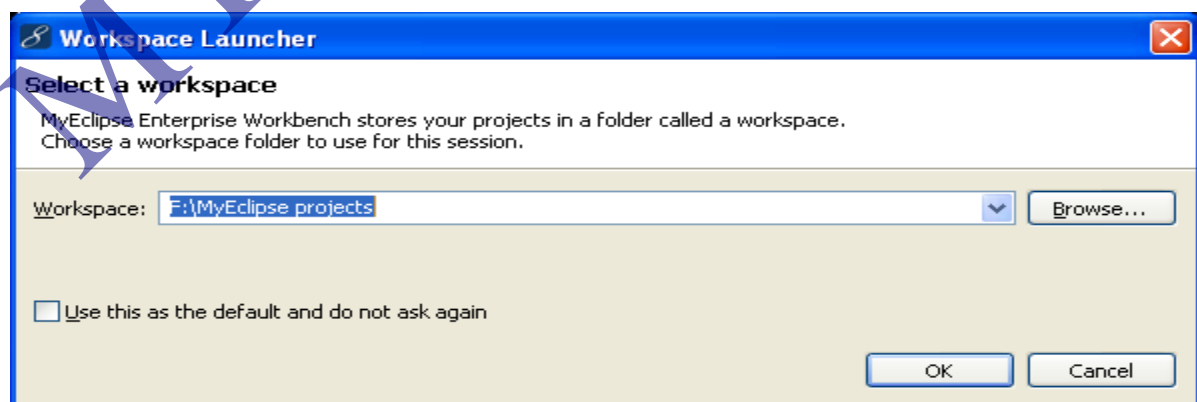
Does not give Built-in errors, but allows the programme to configure and external serves software.

### Procedure to develop the above diagram based struts application by using MyEclipse

### Step-1

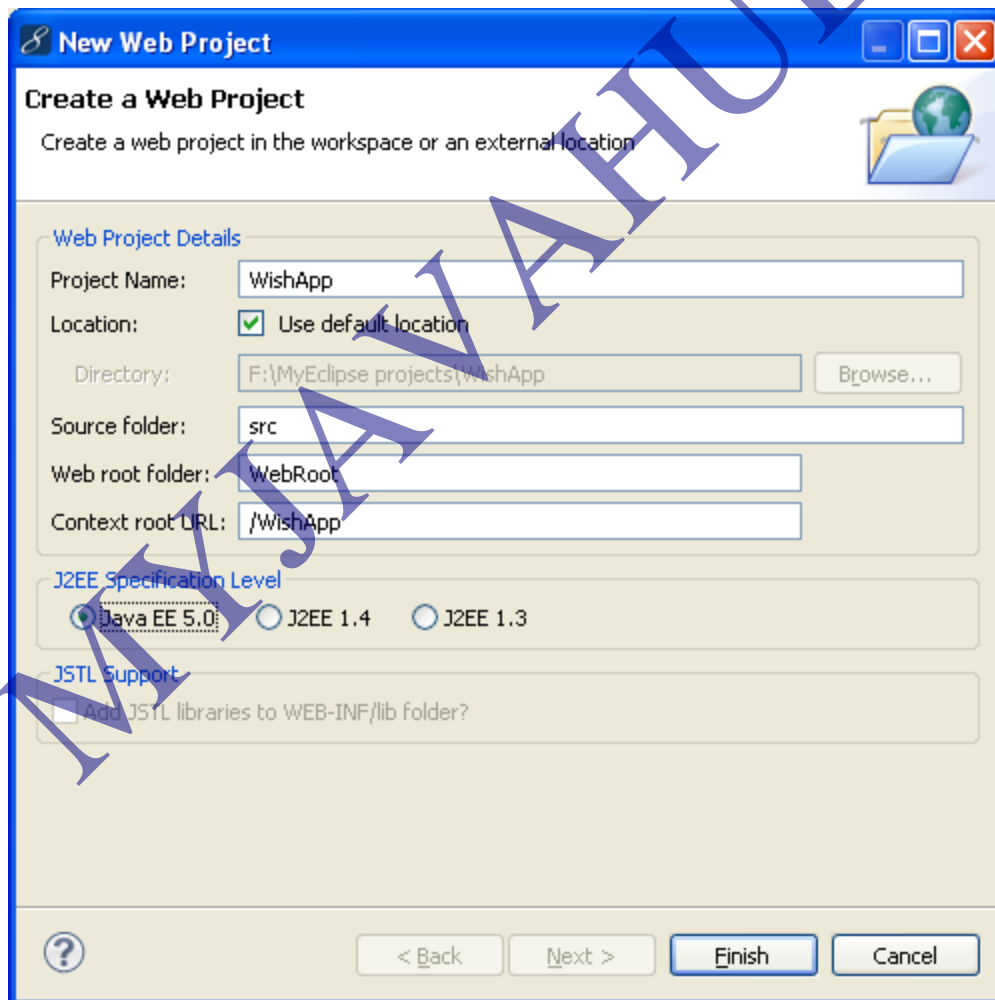
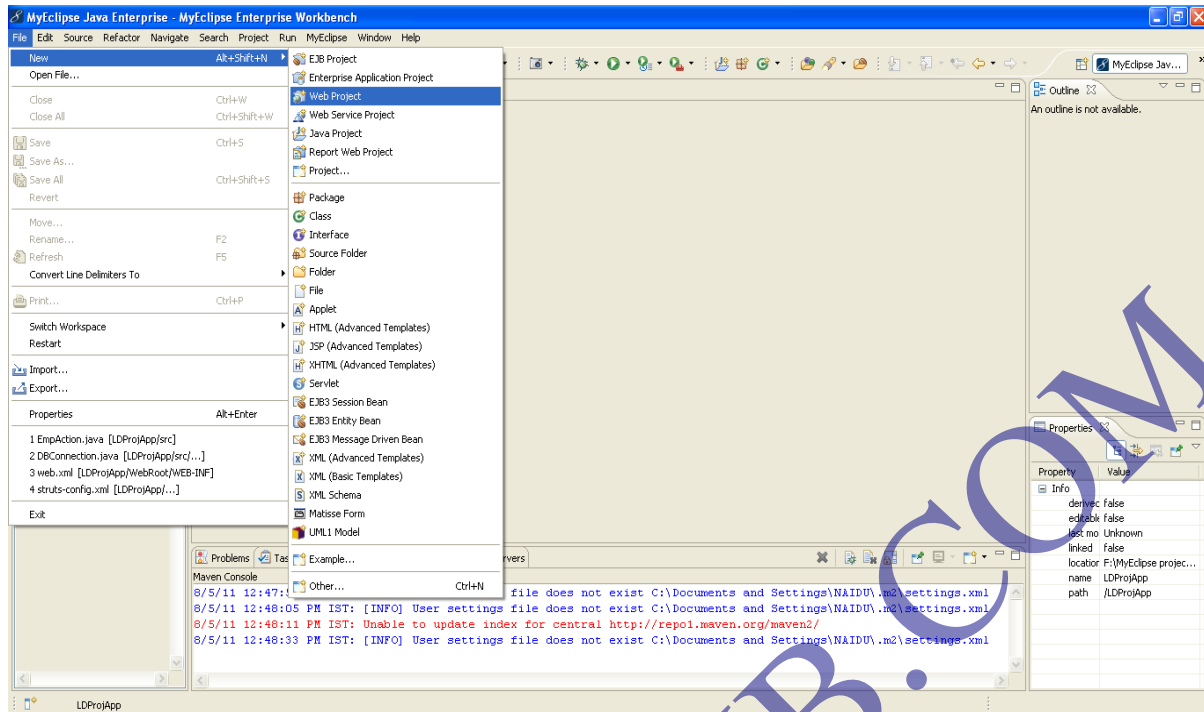
Launch MyEclipse IDE by choosing workspace folder.

**Note:** The folder where projects created in MyEclipse IDE will be saved is called as workspace folder.



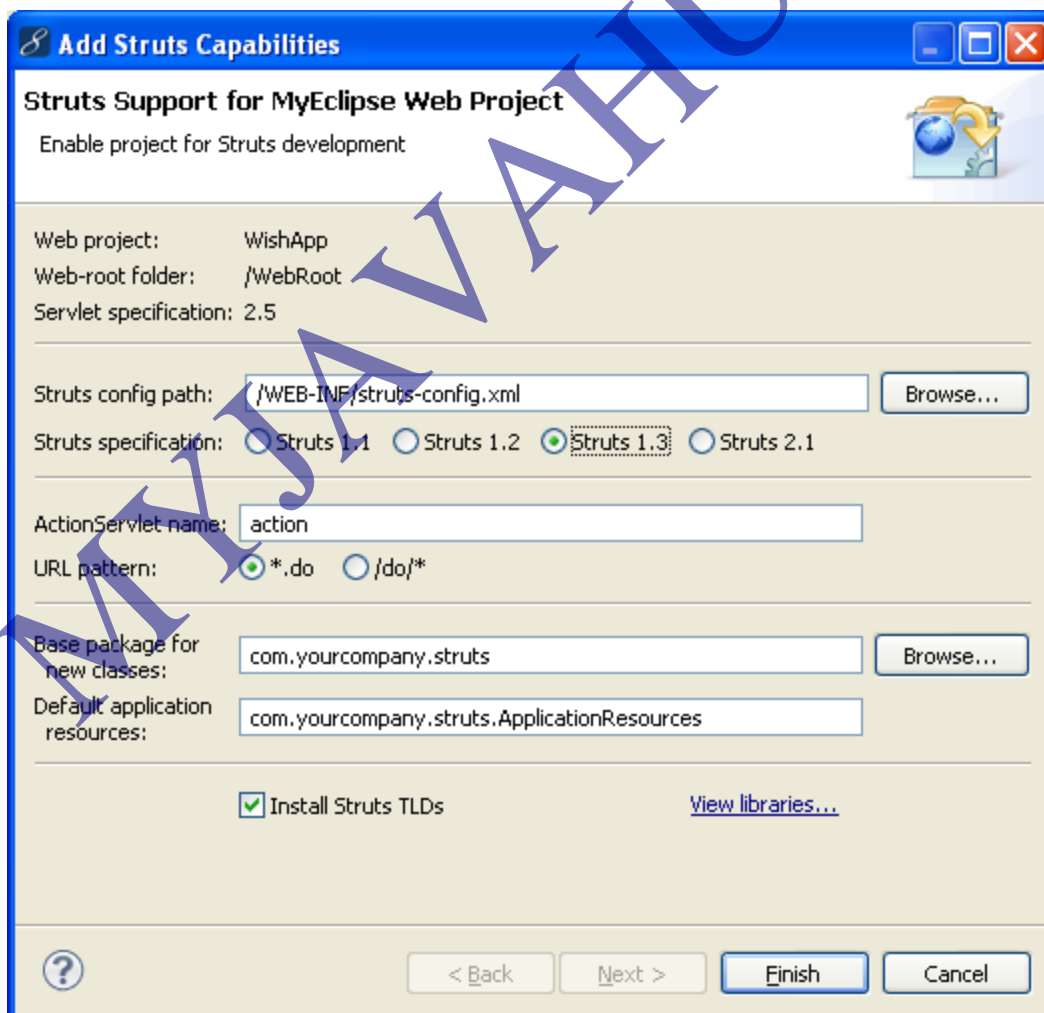
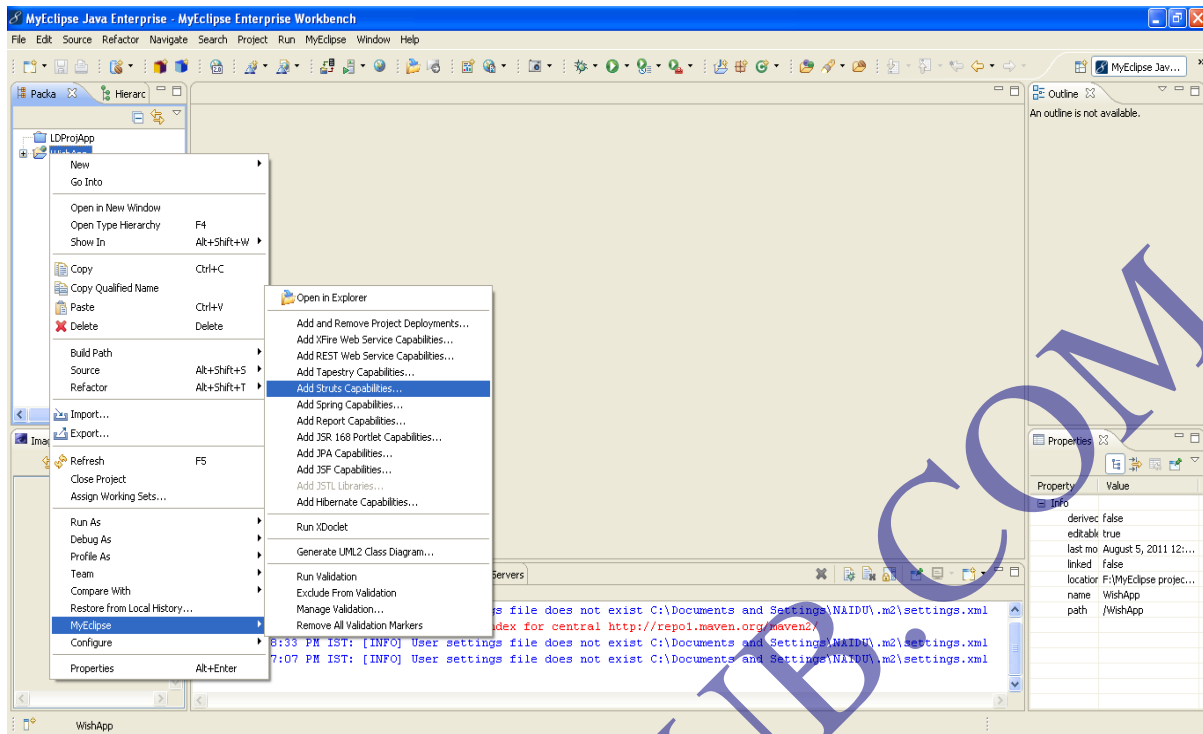
## Step-2

### Create web project



### Step-3

Add struts capabilities to the projects Right click on project -> myEclipse -> All Struts Capabilities -> select struts 1.3 -> finish.



## Step-4

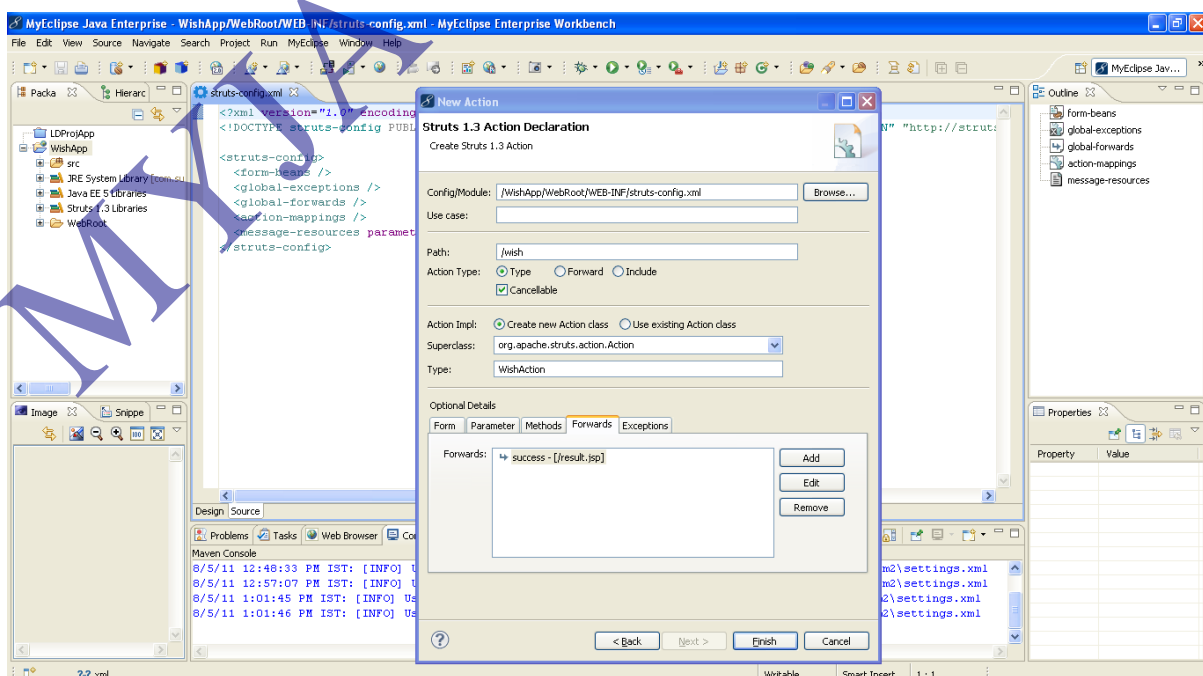
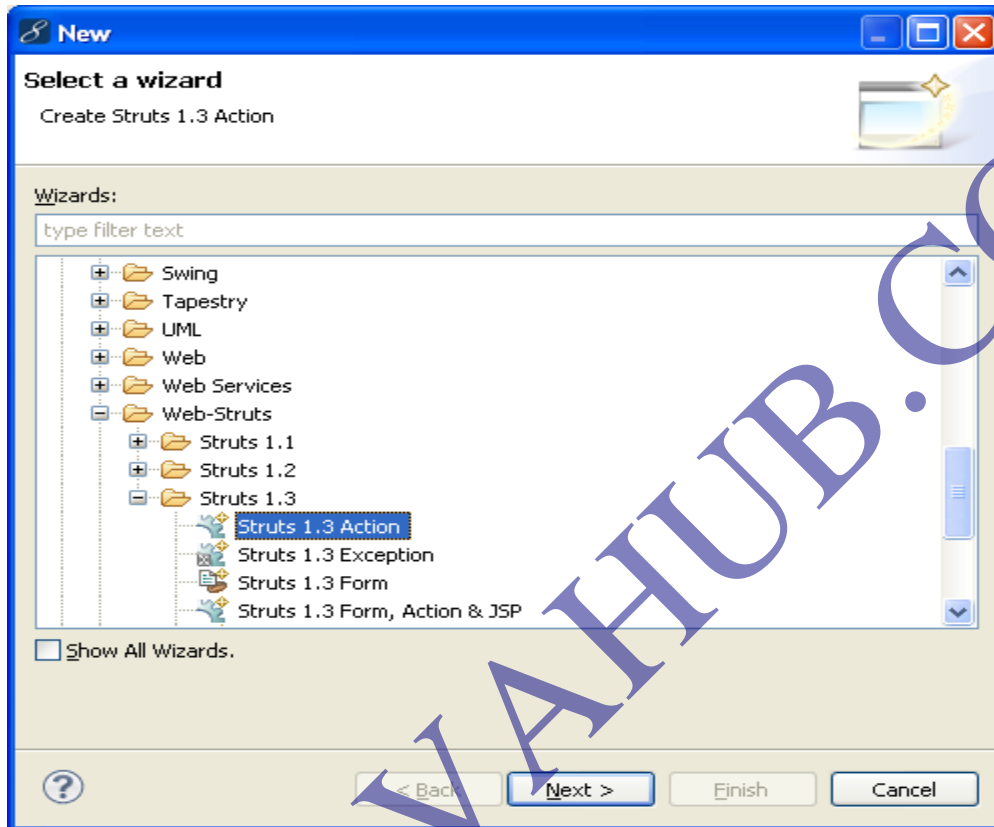
Write following code in the index.jsp of web root folder. Index.jsp

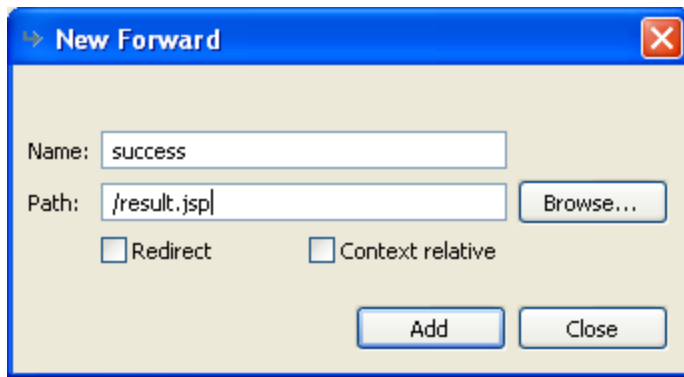
```
<html:link action="with"> getWithMsg</html:link>
```

## Step-5

Add struts Action class to project. Right click on src folder -> new -> other -> MyEclipse -> webstruts -> struts 1/3 -> struts 1.3 Action -> next -> path -> type: wishAction -> next -> forward tab -> name: success, path: result.jsp -> Add -> close -> finish.

Or press Ctrl+N





### Step-6

Add following code in the execute(-,-,-) of with Action class

In execute (-,-,-)

// set current date and time.

Calendar cal=Calendar.getInstance();

//get current hour of the date

Int h =cl.get(Calendar.Hour\_Of\_Day);

String ms=null;

If (h<=12)

msg="Good Morning";

else if (h<=16)

msg="Good Afternoon";

else if(h<=20)

msg="Good Evening";

else

msg="Good Night";

// Keep generated msgs in request attribute request.setAttribute("withmsgs",msg)

return mapping find Forward ("firstclass");

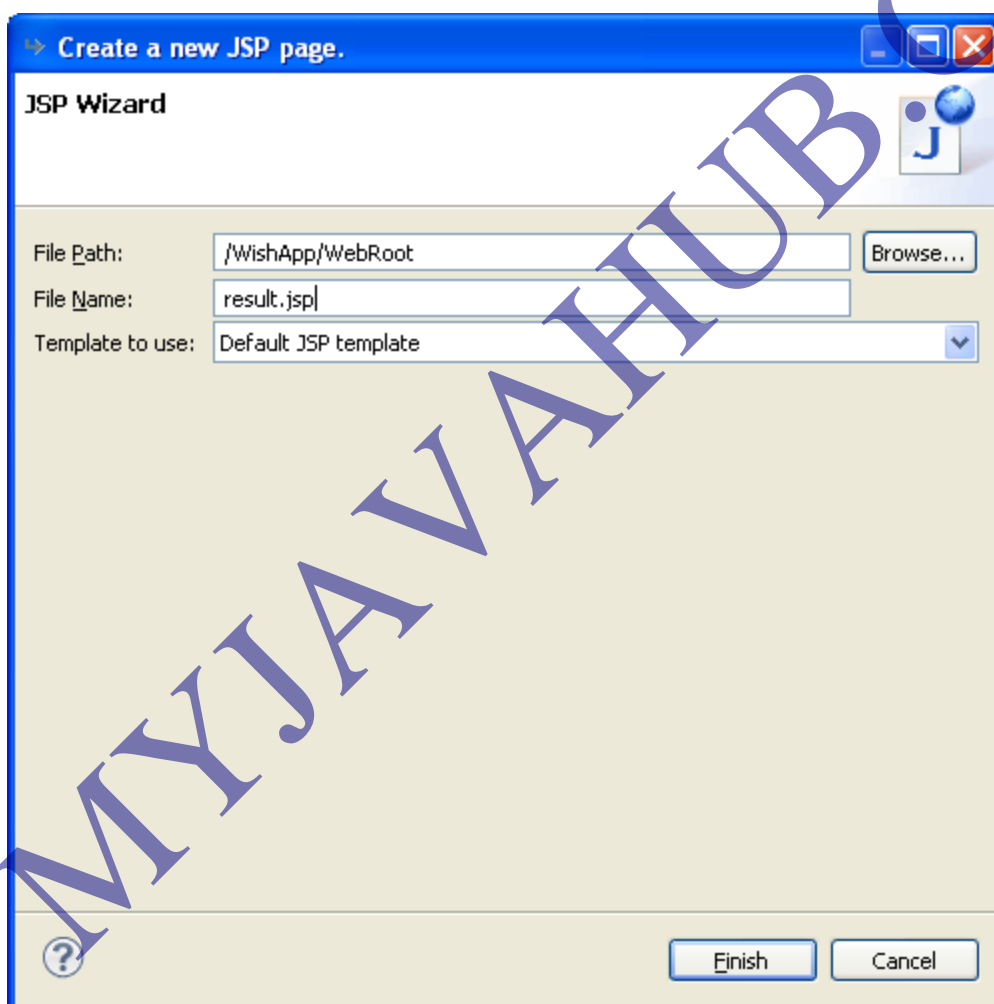
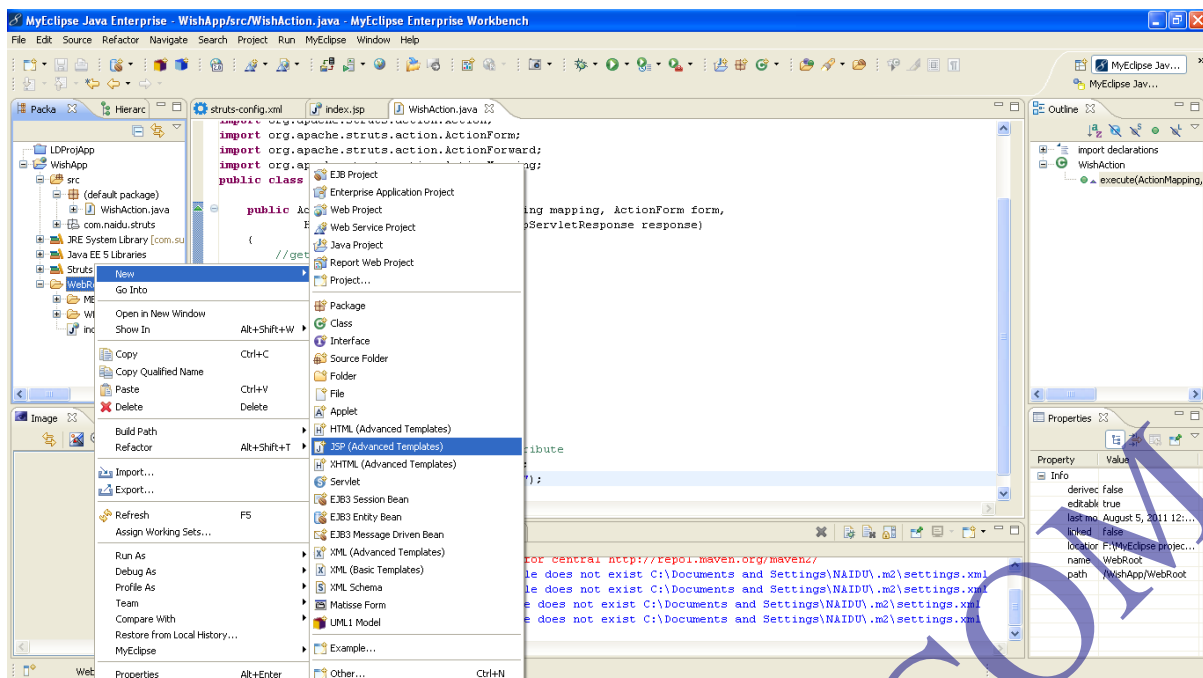
}

}

### Step-7

Add result.jsp page to the web-root folder of the project.

Rightclick on web-root folder -> new -> JSP -> File name result.jsp -> finish



### Request.jsp

<b> The wish msg is </b>

<bean:write name="wish msg" scope="request"/>

</logic:notEmpty>

## Configure Tomcat 6.0 server with MyEclipse IDE

Window menu-> preferences -> Myeclipse -> servers -> Tomcat 6 -> configure tomcat 6 -> enable.

Browse Tomcat home directory -> Apply -> ok

Start the Tomcat server from MyEclipse IDE

Servers icon in the tool bar -> Tomcat 6 -> start.

### Step-8

Deploy project in MyEclipse IDE. Go to deploy icon in the tool bar

Select deploy icon -> choose project -> Add -> server(tomcat 6) -> Deploy type.

Packaged Archive -> finish -> ok

### Step-9

Test the application -> open browser window (Globe symbol, in the tool bar) type the following url

<http://localhost:2020/strutsapp/index.jsp>

**Complete source code for the application developed above, refer the following code.**

#### Index.jsp

```
<%@taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
<html:link action="wish">getWishMsg</html:link>
```

#### web.xml

```
<web-app>

  <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>0</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

</web-app>
```

**struts-config.xml**

```

<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts
Configuration 1.3//EN" "http://struts.apache.org/dtds/struts-config_1_3.dtd">
<struts-config>

<action-mappings>
  <action path="/wish" type="WishAction" cancellable="true">
    <forward name="success" path="/result.jsp" />
  </action>
</action-mappings>

</struts-config>

```

**WishAction.java**

```

import java.util.Calendar;
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 WishAction extends Action {

    public ActionForward execute(ActionMapping mapping, ActionForm form,
                                HttpServletRequest request, HttpServletResponse response)
    {
        //get current date and time
        Calendar cl=Calendar.getInstance();
        //get current hour of the day.
        int h=cl.get(Calendar.HOUR_OF_DAY);
        String msg=null;
        if(h<=12)
            msg="Good Morning";
        else if(h<=16)
            msg="Good Afternoon";
        else if(h<=20)
            msg="Good Evening";
        else
            msg="Good Night";
        //keep generated msg is request attribute
        request.setAttribute("WishMsg",msg);
        return mapping.findForward("success");
    }
}

```

**result.jsp**

```

<%@page pageEncoding="UTF-8"%>
<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
<%@ taglib uri="http://struts.apache.org/tags-logic" prefix="logic" %>
<b>The wish message is</b>
<logic:notEmpty>
<bean:write name="WishMsg" scope="request"></bean:write>
</logic:notEmpty>

```



**BROWSER WINDOW**

**Register.jsp**

**ID**  → (required, integer rule)

**NAME**  → (required, mask rule)

**ADDRESS**  → (required, minlength rule)

**AGE**  → (required, intrange rule)

**DATE OF BIRTH**  → (required, date rule)

**EMAIL**  → (required, email rule)

**REGISTER**

Develop an application to validate above form data by using ValidatorPlugIn and insert the validated values to database table as a record.

(Take the support of Dynamic formbean)

#### Note

The above application is demanding to use only ValidatorPlugIn supplied validator rules. So use declarative dynamic formbean.

The complete source code for the above diagram based application is as follows.

#### Register.jsp (View Layer)

```

1 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html"%>
2 <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean"%>
3 <html:form action="/insert">
4 <center><h1><u>Registration Screen</u></h1>
5 <h2>
6 <table border="0" width="100" align="center">
7 <tr>
8 <td><bean:message key="registration.id"/></td>
9 <td><html:text property="id"/></td>
10 </tr>
11 <tr>
12 <td><bean:message key="registration.name"/></td>
13 <td><html:text property="name"/></td>
14 </tr>
15 <tr>
16 <td><bean:message key="registration.address"/></td>
17 <td><html:text property="address"/></td>
18 </tr>
19 <tr>
20 <td><bean:message key="registration.doj"/></td>
21 <td><html:text property="doj"/></td>
22 </tr>
23 <tr>

```

Action path of RegisterAction class (refer

Key in properties file referring presentation logic label (refer-

```

24 <td><bean:message key="registration.age"/></td>
25 <td><html:text property="age"/></td>
26 </tr>
27 <tr>
28 <td><bean:message key="registration.email"/></td>
29 <td><html:text property="email"/></td>
30 </tr>
31 </table><br><br>
32 <html:submit><bean:message key="btn.cap"/></html:submit>
33 </h2></center>
34 </html:form>

```

Refer line-234

### web.xml (Deployment Descriptor File)

```

35 <web-app>
36 <servlet>
37 <servlet-name>action</servlet-name>
38 <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
39 <init-param>
40 <param-name>config</param-name>
41 <param-value>/WEB-INF/struts-config.xml</param-value>
42 </init-param>
43 <load-on-startup>2</load-on-startup>
44 </servlet>
45 <servlet-mapping>
46 <servlet-name>action</servlet-name>
47 <url-pattern>*.do</url-pattern>
48 </servlet-mapping>
49 <welcome-file-list>
50 <welcome-file>Register.jsp</welcome-file>
51 </welcome-file-list>
52 </web-app>

```

### struts-config.xml (Struts Configuration File)

```

53 <!DOCTYPE struts-config PUBLIC
54 "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
55 "http://struts.apache.org/dtds/struts-config_1_3.dtd">
56 <struts-config>
57 <form-beans>
58 <form-bean name="bean" type="org.apache.struts.validator.DynaValidatorForm">
59 <form-property name="id" type="java.lang.String"/>
60 <form-property name="name" type="java.lang.String"/>
61 <form-property name="address" type="java.lang.String" initial="hyd"/>
62 <form-property name="email" type="java.lang.String"/>
63 <form-property name="doj" type="java.lang.String"/>
64 <form-property name="age" type="java.lang.String"/>
65 </form-bean>
66 </form-beans>
67 <action-mappings>
68 <action input="/Failure.jsp" path="/insert" type="RegisterAction" name="bean"
69 validate="true">
70 <forward name="ok" path="/Success.html"/>
71 <forward name="fail" path="/Failure.html"/>
72 </action>
73 </action-mappings>

```

Logical name of FormBean

Declarative Dynamic FormBean supporting ValidatorPlugIn based form validations.

```

74 <message-resources parameter="ApplicationResources"/>
75 <plug-in className="org.apache.struts.validator.ValidatorPlugIn">
76   <set-property property="pathnames" value="/WEB-INF/validator-rules.xml,
77     /WEB-INF/validation.xml"/>
78 </plug-in>
79 </struts-config>

```

ValidatorPlugIn  
Configuration.

### validation.xml

```

80 <!DOCTYPE form-validation PUBLIC
81 "-//Apache Software Foundation//DTD Commons Validator Rules Configuration 1.3.0//EN"
82 "http://jakarta.apache.org/commons/dtds/validator_1_3_0.dtd">
83 <form-validation>
84   <formset>
85     <form name="bean">
86       <field property="id" depends="required,integer">
87         <arg0 key="my.error.id"/>
88       </field>
89       <field property="name" depends="required,mask">
90         <msg name="mask" key="my.error.mask.name"/>
91         <arg0 key="my.error.name"/>
92         <var>
93           <var-name>mask</var-name>
94           <var-value>^[a-zA-Z]*$</var-value>
95         </var>
96       </field>
97       <field property="address" depends="required,minlength">
98         <arg0 key="my.error.address"/>
99         <arg1 name="minlength" key="${var:minlength}" resource="false">
100           <var>
101             <var-name>minlength</var-name>
102             <var-value>10</var-value>
103           </var>
104         </field>
105         <field property="email" depends="required,email">
106           <arg0 key="my.error.email"/>
107         </field>
108         <field property="age" depends="required,integer,intRange">
109           <arg0 key="my.error.age"/>
110           <arg1 name="intRange" key="${var:min}" resource="false">
111             <arg2 name="intRange" key="${var:max}" resource="false">
112               <var>
113                 <var-name>min</var-name>
114                 <var-value>20</var-value>
115               </var>
116               <var>
117                 <var-name>max</var-name>
118                 <var-value>50</var-value>
119               </var>
120             </field>
121             <field property="doj" depends="required,date">
122               <arg0 key="my.error.doj"/>
123             </field>
124

```

Logical name of FormBean. (Refer line-58)

Key in the properties file supplying {0} value for required, integer rules

Error msg configuration for mask rule  
(Refer line-

Regular Expressions based form  
validation logic for mask rule which  
accepts only alphabets.

Supplies {0} value for required, minlength rules.

Supplies {0} value for minlength validator rule.

Key in properties file supplying {0} value of required, email rules

Supplies {1} {2}  
values of range rule.

Supplies {1} min value of range rule.

Supplies {2} max value of range rule.

Supplies {0} value for required, date rules.

```

125     <var>
126         <var-name>datePattern</var-name>
127         <var-value>yyyy-MM-dd</var-value>
128     </var>
129 </field>
130 </form>
131 </formset>
132 </form-validation>

```

Fixed variable for date rule to specifies the expected date pattern.

### RegisterAction.java (Struts Action class)

```

133 import org.apache.struts.action.*;
134 import javax.servlet.http.*;
135 import java.sql.*;
136 import org.apache.struts.validator.DynaValidatorForm;
137 public class RegisterAction extends Action
138 {
139     public ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest
        req, HttpServletResponse res) throws Exception
140     {
141         Connection con=null;
142         PreparedStatement ps=null;
143         String status=null;
144         try
145         {
146             System.out.println("Action class excute() is called");
147             DynaValidatorForm b=(DynaValidatorForm)form; // Type casting
148             Class.forName("oracle.jdbc.driver.OracleDriver");
149             con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","scott","tiger");
150             ps=con.prepareStatement("insert into struts_student values(?,?,?,?,?,?)");
151             ps.setInt(1,Integer.parseInt((String)b.get("id")));
152             ps.setString(2,(String)b.get("name"));
153             ps.setString(3,(String)b.get("address"));
154             ps.setString(4,(String)b.get("email"));
155             ps.setDate(5,java.sql.Date.valueOf((String)b.get("doj")));
156             ps.setInt(6,new Integer((String)b.get("age")).intValue());
157             status=null;
158             if(ps.executeUpdate()==0)
159                 status="fail";
160             else
161                 status="ok";
162         }
163         catch(ClassNotFoundException ce)
164         {
165             ce.printStackTrace();
166         }
167         catch(SQLException se)
168         {
169             se.printStackTrace();
170         }
171         finally
172         {
173             try
174             {
175                 if(ps!=null)
176                     ps.close();
177             }
178             catch(SQLException se)

```

```

179 {
180     se.printStackTrace();
181 }
182 try
183 {
184     if(con!=null)
185         con.close();
186 }
187 catch(SQLException se)
188 {
189     se.printStackTrace();
190 }
191 }
192 return mapping.findForward(status);
193 }//execute()
194 }//class

```

### Success.html

```

195 <HTML>
196 <BODY>
197 <center><br><br>
198 <h1>Registration successful !!!</h1>
199 </BODY>
200 </HTML>

```

### Failure.html

```

201 <HTML>
202 <BODY>
203 <center><br><br>
204 <h1>Registration Failure !!!</h1>
205 </BODY>
206 </HTML>

```

### ApplicationResources.properties

```

207 # Struts Validator Error Messages
208 errors.header=<h1>List of error(s)</h1><h2><hr><b>
209 errors.footer=</b></h2><hr>
210 errors.required={0} is required.
211 errors.minlength={0} can not be less than {1} characters.
212 errors.maxlength={0} can not be greater than {1} characters.
213 errors.invalid={0} is invalid.
214 errors.byte={0} must be a byte.
215 errors.short={0} must be a short.
216 errors.integer={0} must be an integer.
217 errors.long={0} must be a long.
218 errors.float={0} must be a float.
219 errors.double={0} must be a double.
220 errors.date={0} is not a date.
221 errors.range={0} is not in the range {1} through {2}.
222 errors.creditcard={0} is an invalid credit card number.
223 errors.email={0} is an invalid e-mail address.
224
225 my.error.id=<br><b>I D</b>
226 my.error.name=<br><b>Name</b>
227 my.error.address=<br><b>Address</b>
228 my.error.email=<br><b>Email</b>
229 my.error.age=<br><b>Age</b>
230 my.error.doj=<br><b>Date of Joining</b>

```

Validator rules related error messages

User defined messages to supply argument values for validator rules.

231

232

```
233 my.error.mask.name=<br>Please provide only alphabets for <b>Name</b>
```

Message related to mask validator rule

```
234 btn.cap=Register Details
```

```
235 registration.id=Student ID
```

```
236 registration.name=First Name
```

```
237 registration.email=E-Mail Address
```

```
238 registration.address=Residential Address
```

```
239 registration.doj=Date of Joining (YYYY-MM-DD)
```

```
240 registration.age=Age
```

User defined messages representing presentation logic. (labels of fomepage)

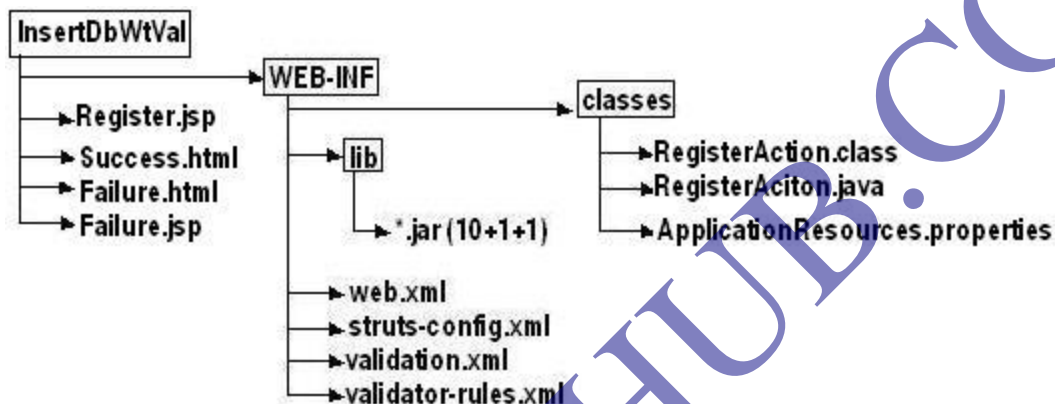
### Failure.jsp

```
241 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html"%>
```

```
242 <b><center>The Following Form validation error(s) are generated</center></b><br>
```

```
243 <html:errors/>
```

### Depolyment Directory Structure



**Jar files in classpath** → servlet-api.jar, struts-core-1.3.8.jar

**Jar files in WEB-INF\lib folder** → 10-regular struts jar files, 1-ojdbc14.jar

When more form components are there in the form page, it is recommended to the form page as the content of html table rows and columns. This process makes the alignment of form page easy.

Once data value is given to JDBC driver as "java.sql.Date" class object. It is the responsibility of JDBC driver to insert that data value in the date datatype column having that date pattern. That is supported by underlying database software.

### Converting string date value to 'java.sql.Date' class object

#### When given string date value is there in yyyy-MM-dd pattern

```
String s1="1987-07-25"; //yyyy-MM-dd
```

```
java.sql.Date sqd1=java.sql.Date.valueOf(s1);
```

#### When given string date value is not there in yyyy-MM-dd pattern

```
String s1="22-11-2003"; //dd-MM-yyyy
```

```
//convert to java.util.Date class object.
```

```
SimpleDateFormat sdf1=new SimpleDateFormat("dd-MM-yyyy");
```

```
java.util.Date ud1=sdf1.parse(s1);
```

```
//convert to java.util.Date to java.sql.Date object.
```

```
Long ms=ud1.getTime();
```

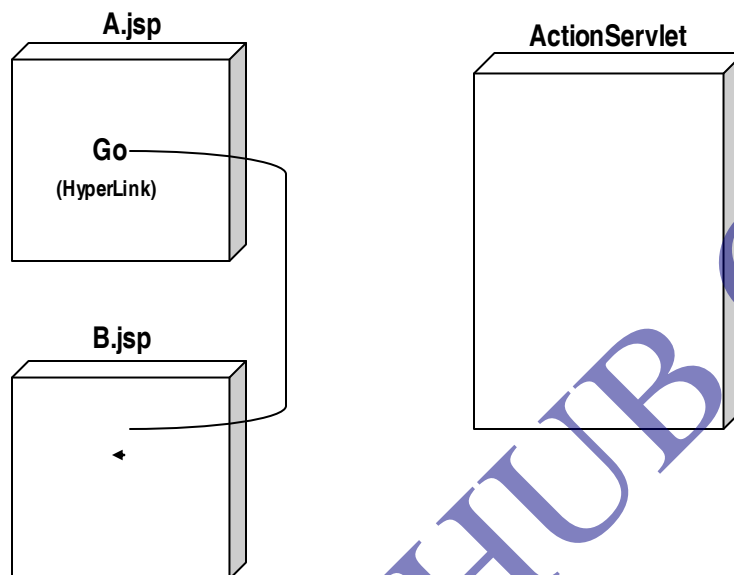
```
java.sql.Date sqld1=new java.sql.Date(ms);
```

Struts supplies set of built-in Action classes to develop our action classes or to use them directly in our struts applications. They are

1. ForwardAction
2. IncludeAction
3. SwitchAction
4. DispatchAction
5. LookupDispatchAction
6. MappingDispatchAction
7. DownloadAction and etc.

Available in “**org.apache.struts.actions**” package which is available in struts\_home\lib\struts-extras-1.3.8.jar

### Problem scenario



### Code in A.jsp

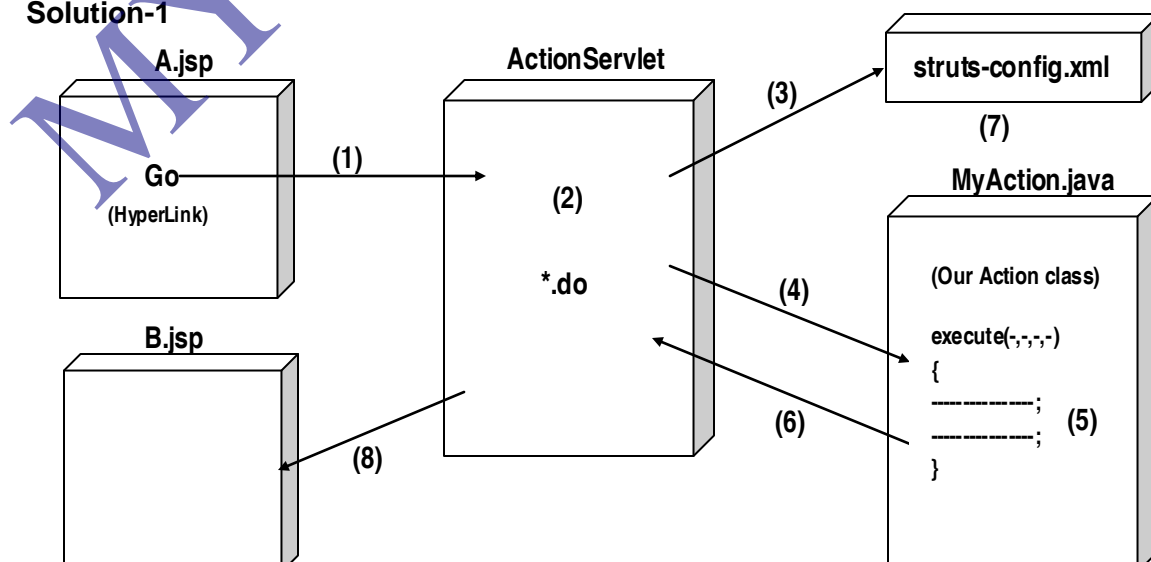
```
<a href= "B.jsp">Go</a>
```

(or)

```
<html:link href= "B.jsp">Go</html:link>
```

In the above diagram the 'A.jsp' is directly interacting with 'B.jsp' (without having controller servlet support) which is against MVC-2 principles.

### Solution-1



**In A.jsp**

```
<a href= "xyz.do">Go</a>
```

(or)

```
<html:link href= "xyz.do">Go</html:link>
```

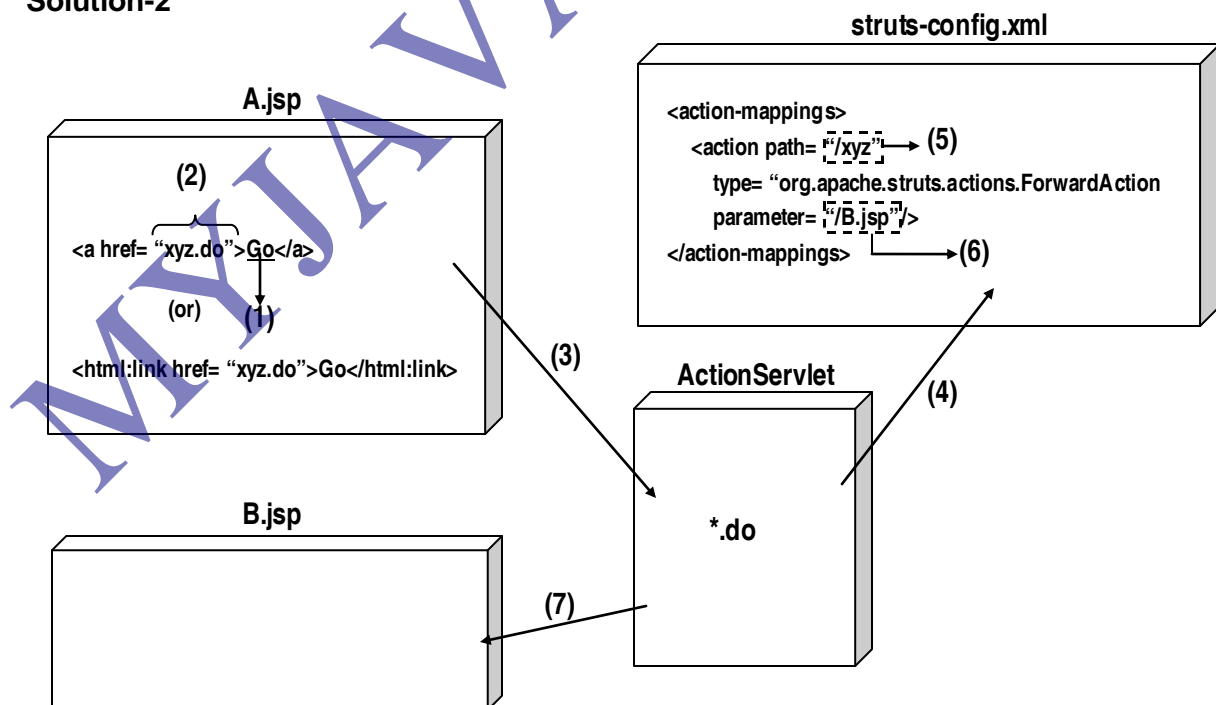
**In struts configuration file**

```
<action path= "/xyz" type= "MyAction">
  <forward name= "ok" path= "B.jsp"/>
</action>
```

**In MyAction.java**

```
public class MyAction extends Action
{
  public ActionForward execute (-,-,-)
  {
    return mapping.findForward("ok");
  }
}
```

- ✓ The above solution-1 transfers the control to B.jsp from A.jsp through the controller servlet called ActionServlet.
- ✓ But a dummy user-defined ActionClass must be taken and must be configured in this whole process.
- ✓ To overcome the above limitation with solution-1 take the support of ForwardAction built-in class as shown below.

**Solution-2**

**Note:** By working with all the above given built-in Action classes like ForwardAction, place "struts-extras-1.3.8.jar" file in WEB-INF/lib folder.



**With respect to the above sample code in the flow**

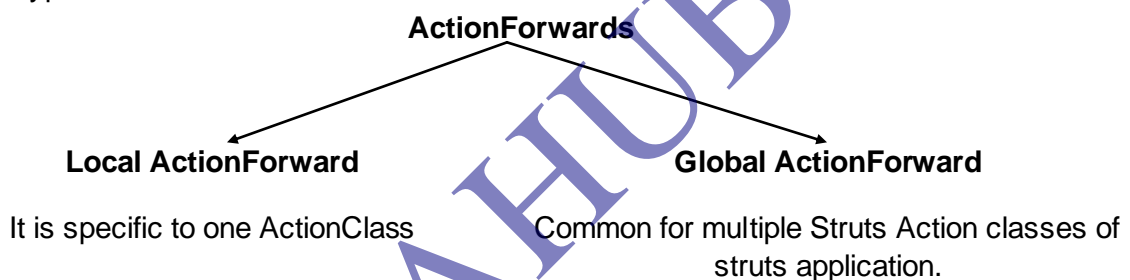
- 1) End user clicks on the hyperlink of 'A.jsp'.
- 2) From href attribute request url will be gathered.
- 3) ActionServlet traps and takes the request.
- 4) ActionServlet uses struts configuration file entries to locate the Action class.
- 5) ActionServlet looks for an Action class configuration whose Action path is "\xyz".
- 6) Since that Action class is ForwardAction (built-in) class, the parameter attribute value 'B.jsp' goes to ActionServlet as the data of ActionForward class object.
- 7) ActionServlet transfers the control to the destination page 'B.jsp'

**Q) What is the difference between ActionForward and ForwardAction.**

**Ans:** ForwardAction is built-in Action class which is given to transfer control between web-resource programs of struts application under the control of the controller servlet called "ActionServlet".

ActionForward class represents ActionForward configurations of struts configuration file pointing to the result pages of Struts Action class.

ActionForwards are there pointing to the result pages of struts Action classes. There are two types of ActionForwards.

**In struts-config.xml**

```

<struts-config>
  <form-beans>
    -----
  </form-beans>
  <global-forwards>
    <forward name="success" path="/one.jsp"/>
    <forward name="failure" path="/two.jsp"/>
  </global-forwards>
  <action-mappings>
    <action path="/xyz1" type="MyAction1">
      <forward name="success" path="/three.jsp"/>
      <forward name="failure" path="/four.jsp"/>
    </action>
    <action path="/xyz2" type="MyAction2">
      <forward name="success" path="/five.jsp"/>
      <forward name="failure" path="/six.jsp"/>
    </action>
  </action-mappings>
</struts-config>
  
```

Global forwards pointing to result pages

Local forwards specific to MyAction1 Action class.

Local forwards specific to MyAction2 Action class.



### Improvisation-1 (I-4)

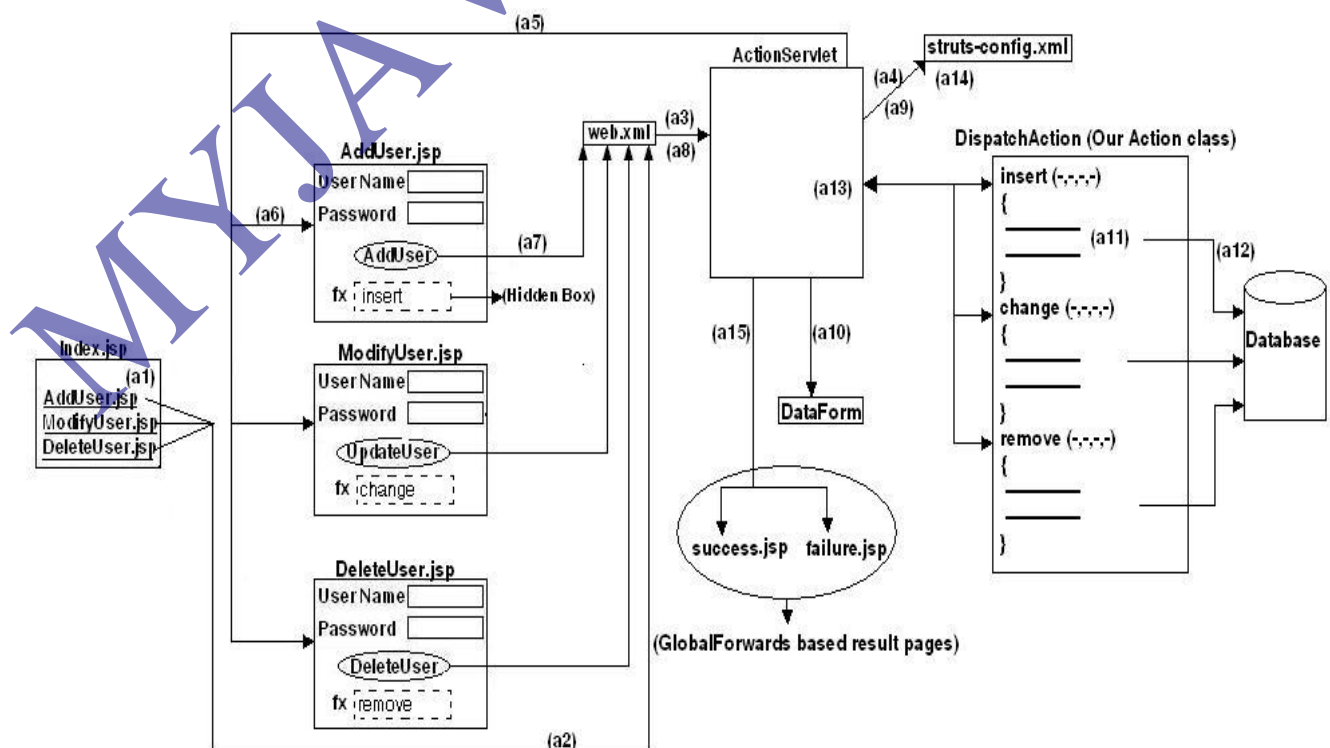
Since all the three Action classes are dealing with similar logics, it is recommended to combine these three Action classes into single Action class by taking the support of DispatchAction class.

The Action class whose type is DispatchAction class can group, multiple related Action classes into single Action class and can maintain the multiple logics of these multiple Action classes in multiple userdefined methods which contains same signature of execute(-,-,-) method as shown below sample code.

```
public class OurAction extends DispatchAction
{
    public ActionForward method1(-,-,-)
    {
        -----;
        -----; //Logic 1
    }
    public ActionForward method2(-,-,-)
    {
        -----;
        -----; //Logic 2
    }
    public ActionForward method3(-,-,-)
    {
        -----;
        -----; //Logic 3
    }
}
```

In the above class the method1,method2,method3 are user-defined methods, but their signature must be same as execute(-,-,-) method.

### Mini Project Design after modifying with the above mentioned improvisations.



**With respect to the above diagram.**

(a1)	: End user clicks “AddUser” hyperlink of Index.jsp
(a2),(a3)	: Based on the configuration done in “web.xml” file, the ActionServlet traps and takes the request.
(a4)	: ActionServlet uses ForwardAction class configuration done in “struts-config.xml” file to decide the target page.
(a5),(a6)	: In this process, ActionServlet chooses AddUser.jsp as target page and launches that page as form page on the browser window.
(a7)	: Enduser fills up the form and clicks “AddUser” button.
(a8),(a9)	: Based on the configuration done in “web.xml” file, the ActionServlet traps and takes the request along with the hidden form request parameter.
(a10)	: Reads the form data form FormBean
(a11),(a12)	: According to the configuration done in struts-config.xml file, the insert (-,-,-) method will be executed and the data is stored in the database.
(a13),(a14)	: control comes to ActionServlet and again checks the struts-config.xml file for choosing the result page.
(a15)	: According to the result given by the insert (-,-,-) method and configurations done in struts-config.xml file, corresponding result page will be launched on the browser window.

**Note**

When multiple form pages are targeting to execute multiple user-defined methods of DispatchAction class through ActionServlet, then form page should send its target method name as additional request parameter value along with request submitted by form page. Then name of this additional request parameter value will be chosen by programmer during DispatchAction class configuration done in “struts-config.xml” file. (as these value of parameter attribute of <action> tag)

**In struts-config.xml**

```
<action-mappings>
```

Our DispatchAction class

```
<action path= “/xyz” name= “df” type= “OurAction” parameter= “fx”/>
```

```
</action-mappings>
```

Additional request parameter name that should hold method name of DispatchAction class as value.

- Generally programmers prefer working with hidden boxes to send method names as additional request parameter values from the form pages. So, the names of the hidden boxes should be taken as the additional request parameter name of DispatchAction class configuration. (parameter attribute value of <action> tag. (like ‘fx’)) as shown in the above given diagram.
- Don’t ever place execute (-,-,-) method in your DispatchAction class. Because that will not make other user-defined methods to execute. Always give chance to execute pre-defined execute (-,-,-) method of pre-defined DispatchAction class (super class of our DispatchAction class). Because this method is capable of calling on user-defined method of sub-class. (our DispatchAction class) based on method name that comes as additional request parameter value.

For the above diagram based source code (Mini project part-1 refer the following application)

### Index.jsp (Welcome page)

```

1  <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
2  <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
3  <html:html>
4  <h2><center>Please click on a link below, to perform your desired activity.<br><br>
5      <html:link action="/add">Add User</html:link><br>
6      <html:link action="/modify">Modify User</html:link><br>
7      <html:link action="/delete">Delete User</html:link>
8  </center></h2>
9  </html:html>

```

Refer line-47

Refer line-48

Refer line-49

### web.xml (Deployment Descriptor file)

```

10 <?xml version="1.0" encoding="ISO-8859-1"?>
11 <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
12 "http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">
13 <web-app>
14 <servlet>
15 <servlet-name>Dummy</servlet-name>
16 <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
17 <init-param>
18 <param-name>config</param-name>
19 <param-value>/WEB-INF/struts-config.xml</param-value>
20 </init-param>
21 <load-on-startup>1</load-on-startup>
22 </servlet>
23 <servlet-mapping>
24 <servlet-name>Dummy</servlet-name>
25 <url-pattern>*.do</url-pattern>
26 </servlet-mapping>
27 <welcome-file-list>
28 <welcome-file>Index.jsp</welcome-file>
29 </welcome-file-list>
30 </web-app>

```

### struts-config.xml (struts configuration file)

```

31 <?xml version="1.0" encoding="UTF-8"?>
32 <!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration
33 1.1//EN" "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
34 <struts-config>
35 <form-beans>
36 <form-bean name="bean" type="org.apache.struts.action.DynaActionForm">
37 <form-property name="id" type="java.lang.String"/>
38 <form-property name="pass" type="java.lang.String"/>
39 <form-property name="function" type="java.lang.String"/>
40 </form-bean>
41 </form-beans>

```

Declarative Dynamic FormBean for all the three form pages

```

41 <global-forwards>
42   <forward name="success" path="/success.jsp"/>
43   <forward name="failure" path="/failure.jsp"/>
44 </global-forwards>
45 <action-mappings>
46   <action path="/controller1" name="bean" type="OurAction" parameter="function"
      scope="request"/>
47   <action path="/add" type="org.apache.struts.actions.ForwardAction"
      parameter="/AddUser.jsp"/>
48   <action path="/modify" type="org.apache.struts.actions.ForwardAction"
      parameter="/ModifyUser.jsp"/>
49   <action path="/delete" type="org.apache.struts.actions.ForwardAction"
      parameter="/DeleteUser.jsp"/>
50   <action path="/home" type="org.apache.struts.actions.ForwardAction"
      parameter="/Index.jsp"/>
51 </action-mappings>
52 <message-resources parameter="ApplicationResources"/>
53 </struts-config>

```

Global ActionForwards pointing to result pages

Additional Request parameter name

ForwardAction classes configurations towards hyperlinks.

### AddUser.jsp (Form page-1)

```

54 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
55 <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
56 <html:html>
57   <html:form action="/controller1">
58     <center><br><br>
59     <table>
60       <caption><font size = 6><u>Adding User</font></h2></u></caption>
61       <tr>
62         <td><bean:message key="form.id"/></td>
63         <td><html:text property="id"/></td>
64       </tr>
65       <tr>
66         <td><bean:message key="form.pass"/></td>
67         <td><html:password property="pass"/></td>
68       </tr>
69     </table>
70     <br>
71     <html:submit value="AddUser"/>
72     <html:hidden property="function" value="insert"/>
73   </center>
74 </html:form>
75 </html:html>

```

Action path of DispatchAction class (OurAction). (Refer line-46)

Refer line-274

Refer line-275

Target method name in our DispatchAction class.

Additional Request parameter name. (Refer line-46)

### ModifyUser.jsp (Form page -2)

```

76 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
77 <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
78 <html:html>
79   <html:form action="/controller1">
80     <center><br><br>
81     <table>
82       <caption><font size = 6><u>Modifying User</font></h2></u></caption>

```

```

83     <tr>
84         <td><bean:message key="form.id"/></td>
85         <td><html:text property="id"/></td>
86     </tr>
87     <tr>
88         <td><bean:message key="form1.pass"/></td>
89         <td><html:password property="pass"/></td>
90     </tr>
91 </table>
92 <br><br>
93 <html:submit value="UpdateUser"/>
94 <html:hidden property="function" value="change"/>
95 </center>
96 </html:form>
97 </html:html>

```

Target method name in our DispatchAction class.

Additional Request parameter name. (Refer line-46)

### DeleteUser.jsp (Form page-3)

```

98 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
99 <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
100 <html:html>
101     <html:form action="/controller1">
102         <center><br><br>
103         <table>
104             <caption><font size = 6><u>Deleting User</font></h2></u></caption>
105             <tr>
106                 <td><bean:message key="form.id"/></td>
107                 <td><html:text property="id"/></td>
108             </tr>
109         </table>
110         <br><br>
111         <html:submit value="DeleteUser"/>
112         <html:hidden property="function" value="remove"/>
113     </center>
114 </html:form>
115 </html:html>

```

Target method name in our DispatchAction class.

Additional Request parameter name. (Refer line-46)

### OurAction.java (Our DispatchAction Class)

```

116 import java.io.*;
117 import org.apache.struts.action.*;
118 import org.apache.struts.actions.*;
119 import java.sql.*;
120 import javax.naming.*;
121 import javax.sql.*;
122 import javax.servlet.http.*;
123 public class OurAction extends DispatchAction → Mandatory
124 {
125     private Connection getConnection() → User-defined method having logic to
126     {                                     create JDBC connection object.
127     Connection con = null;
128     try
129     {
130         Class.forName("oracle.jdbc.OracleDriver");
131         con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE", "Scott", "Tiger");
132     }
133     catch(Exception e)
134     {
135     }

```



```

136 return con;
137 }
138 private void releaseResources(Connection con, PreparedStatement pst)
139 {
140 if(con != null)
141 {
142 try
143 {
144 con.close();
145 }
146 catch(Exception e)
147 {
148 }
149 }
150 if(pst != null)
151 {
152 try
153 {
154 pst.close();
155 }
156 catch(Exception e)
157 {
158 }
159 }
160 }
161 public ActionForward insert(ActionMapping mapping, ActionForm f, HttpServletRequest
    request, HttpServletResponse response)
162 {
163 Connection con = null;
164 PreparedStatement pst = null;
165 try
166 {
167 DynaActionForm form=(DynaActionForm)f; //Typcasting
168 con = this.getConnection(request);
169 pst=con.prepareStatement("INSERT INTO STRUTS_DISP_ACTION_TEST VALUES(?, ?)");
170 pst.setString(1,(String)form.get("id"));
171 pst.setString(2,(String)form.get("pass"));
172 int i = pst.executeUpdate();
173 request.setAttribute("Action", "Insert ");
174 if(i != 0)
175 {
176 return mapping.findForward("success");
177 }
178 else
179 {
180 return mapping.findForward("failure");
181 }
182 } // try
183 catch(Exception e)
184 {
185 return mapping.findForward("failure");
186 }
187 finally
188 {
189 releaseResources(con,pst);
190 }
191 } // addUser()

```

Returns the JDBC connection object.

User-defined helper method to close JDBC connection object.

Representing the insert operation of the current method.



```

192 public ActionForward remove(ActionMapping mapping,ActionForm f,HttpServletRequest
    request,HttpServletResponse response)
193 {
194 Connection con = null;
195 PreparedStatement pst = null;
196 try
197 {
198 DynaActionForm form=(DynaActionForm)f;
199 con=getConnection(request);
200 pst=con.prepareStatement("DELETE FROM STRUTS_DISP_ACTION_TEST WHERE ID=?");
201 pst.setString(1,(String)form.get("id"));
202 int i=pst.executeUpdate();
203 request.setAttribute("Action", "Deletion");
204 if(i != 0)
205 {
206 return mapping.findForward("success");
207 }
208 else
209 {
210 return mapping.findForward("failure");
211 }
212 }
213 catch(Exception e)
214 {
215 return mapping.findForward("failure");
216 }
217 finally
218 {
219 releaseResources(con, pst);
220 }
221 } // deleteUser()
222 public ActionForward change(ActionMapping mapping,ActionForm f,HttpServletRequest
    request,HttpServletResponse response)
223 {
224 Connection con = null;
225 PreparedStatement pst = null;
226 try
227 {
228 DynaActionForm form=(DynaActionForm)f;
229 con = this.getConnection(request);
230 pst=con.prepareStatement("UPDATE STRUTS_DISP_ACTION_TEST SET PASS=? WHERE ID=?");
231 pst.setString(1,(String)form.get("pass"));
232 pst.setString(2,(String)form.get("id"));
233 int i = pst.executeUpdate();
234 request.setAttribute("Action", "Updation");
235 if(i != 0)
236 return mapping.findForward("success");
237 else
238 return mapping.findForward("failure");
239 }
240 catch(Exception e)
241 {
242 return mapping.findForward("failure");
243 }
244 finally
245 {
246 releaseResources(con,pst);
247 }

```

```

248 } // updateUser()
249 } // OurAction class.

```

### success.jsp (Result page-1)

```

250 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
251 <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
252 <html>
253 <body>
254 <div align="center">
255 <font color="#0000FF" size="5"><em><strong>
256 <%= (String)request.getAttribute("Action")%>Operation Successful <br>
257 and values are <%= (String)request.getParameter("id")%>
258 and <%= (String)request.getParameter("pass")%>
259 </strong></em></font>
260 </div>
261 <center>
262 <br><br>
263 <html:link action="/home">HOME</html:link><br>
264 </center>
265 </body>
266 </html>

```

Use request attribute value given by Action class methods, indicating the operation that is performed.

Refer line-50

### failure.jsp (Result page-2)

```

267 <html>
268 <body>
269 <p align="center"><strong><font size="5"><em>
270 Operation failed !!! </em></font></strong>
271 </p>
272 </body>
273 </html>

```

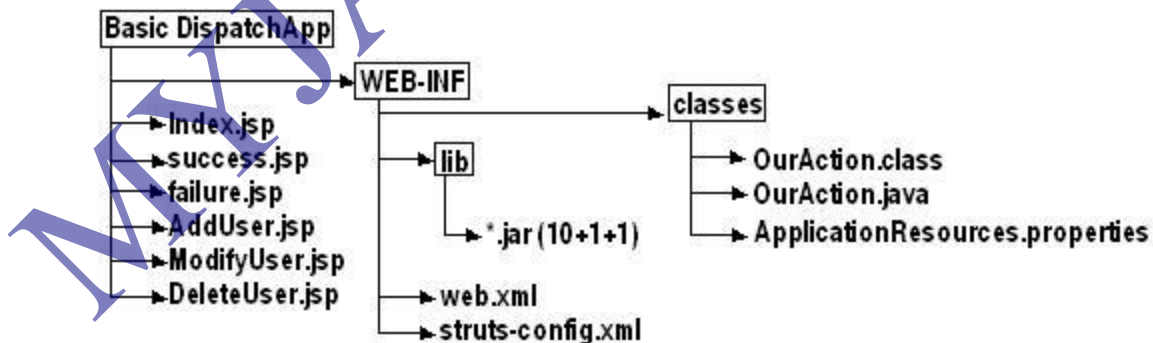
### ApplicationResources.properties (Properties file)

```

274 form.id=<b><u>User Name</u></b>
275 form.pass=<b><u>Password</u></b>
276 form1.pass=<b><u>New Password</u></b>

```

### Deployment Directory Structure



### Jar files required in classpath

→servlet-api.jar  
→struts-core-1.3.8.jar  
→struts-extras-1.3.8.jar

### Jar files required in WEB-INF/lib folder

→struts.jar  
→ojdbc14.jar  
→struts-extras-1.3.8.jar

### Flow of execution of above mini project part-1

**Step-1**

Programmer deploys struts application in web server or application server.

**Step-2**

Because of <load-on-startup> enables on ActionServlet, the servlet container completes instantiation and initialization operations on ActionServlet. (Refer line-21)

**Note**

Initialization means ActionServlet reads and verifies the entries of struts configuration file.

**Step-3**

End user gives first request to struts application. In this process welcome file executes automatically. (Refer line-28 and lines 1-9)

**Step-4**

End user clicks on AddUser hyperlink. (Refer line-5)

**Step-5**

ActionServlet traps and takes the request. (Refer lines 23-26 and 14-22)

**Step-6**

ActionServlet uses ForwardAction class configuration in struts configuration file to forward the request to AddUser.jsp. (Refer line-47)

**Step-7**

The AddUser.jsp form page will be launched on the browser window.

**Step-8**

ActionServlet creates FormBean class object and keeps that in request scope. (Refer lines 35-39 & 46)

**Step-9**

End user fills up the form page and submits the form page (AddUser.jsp) (Refer line-71)

**Step-10**

Generates the request url having 'controller1.do' and function=insert and etc.(Refer lines 57 & 72)

**Step-11**

ActionServlet traps and takes the request. (Refer lines 23-26 and 14-22)

**Step-12**

ActionServlet creates FormBean class object in request scope and write the received form data of AddUser.jsp to it. (Refer lines 46 & 35-39)

**Step-13**

ActionServlet creates OurAction class (DispatchAction) object and call execute(-,-,-) method on it. (46)

**Step-14**

Since execute(-,-,-) method is not there in 'OurAction' class, super class (pre-defined DispatchAction class) execute(-,-,-) method will executes.

**Step-15**

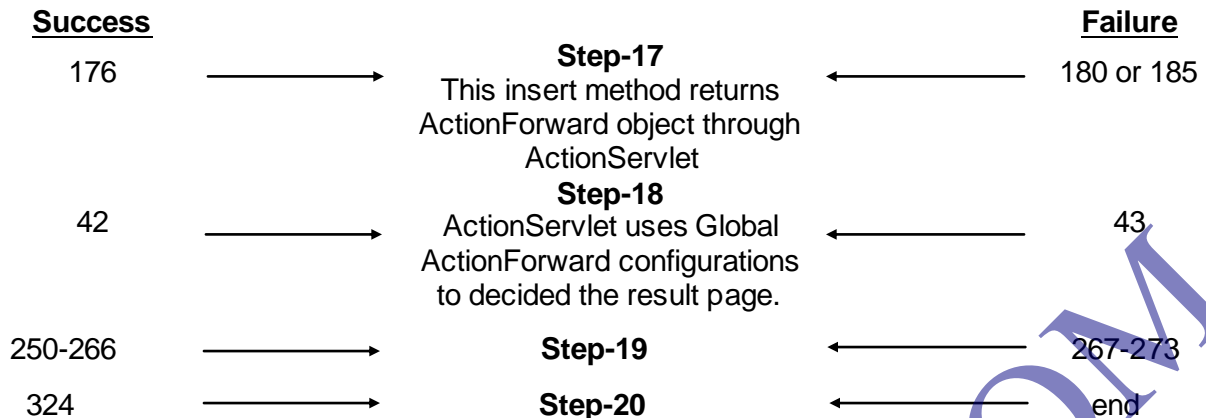
This execute(-,-,-) method reads the additional request param value. i.e. , function= "insert" and calls insert method.

**Step-16**

Insert(-,-,-) method of OurAction class will executes. (Refer lines 161-191)

**Note**

When super class method is executing based on sub-class object, the method that is called from the definition of super class method will be verified first in sub class. If not available then it will be verified in super class.



(End user clicks on the  
Hyperlink of success.jsp page)

**Step-21**

ActionServlet traps and takes the request. (Refer lines 23-26 and 14-22)

**Step-22**

ActionServlet uses ForwardAction class configuration and get index.jsp as target page. (refer line -50)

**Step-23**

ActionServlet launches index.jsp on browser window. (1-9)

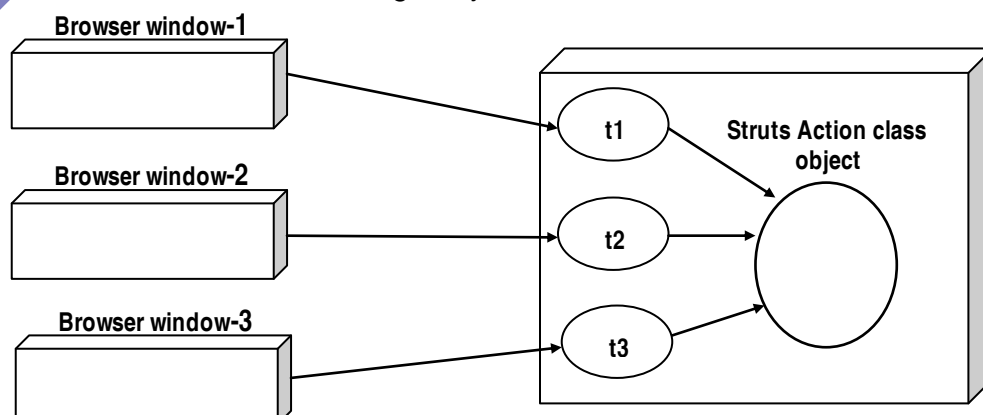
**Note**

The pre-defined DispatchAction class is an abstract class containing no abstract methods.

**Q) Can I place the regular execute(-,-,-) method in our DispatchAction class?**

**Ans:** No, because ActionServlet always calls this method for any request coming to our DispatchAction class and other user-defined business methods which contain same signature of execute(-,-,-) method will not be executed.

- If multiple threads are allowed to manipulate single object data concurrently or simultaneously then that object is called as “non-thread safe object”. Because these threads may corrupt object data.
- When multiple threads are started on single object but only one thread is allowed on to the object at a time to manipulate the data then that object is called as “thread safe object”. To make object as “thread safe”, work with synchronization concept.
- Struts Action class is not thread safe by default. Because, it allows multiple threads, concurrently or parallel on the single object of struts Action class.



- t1, t2, t3 are threads started on single Action class object representing the multiple requests that are started.
- When multiple requests are given to single struts Action class then only one object of struts Action class will be created and multiple threads will be started on that object representing multiple requests as shown in the above diagram.
- To make struts Action class as thread safe
  - a) Work with synchronized execute(-,-,-) method.
  - b) Use synchronized blocks inside the execute(-,-,-) method.
- Approach (b) is recommended.

#### Working with synchronized execute(-,-,-)

```
public class MyAction extends Action
{
    Public synchronized ActionForward execute(-,-,-)
    {
        -----; //write Bussiness logic or Request Processing logic
        -----;
    }
}
```

#### Using synchronized blocks inside the execute(-,-,-) method

```
public class MyAction extends Action
{
    public synchronized ActionForward execute(-,-,-)
    {
        Synchronized(con obj)
        {
            -----; //connection object related statements
            -----;
        }
        -----; //normal statements
        -----;
        Synchronized(req obj)
        {
            -----; //request object related statements
            -----;
        }
    }
}
```

#### Problem

When FormBean type is ValidatorForm or DynaValidatorForm, we need to configure validator rules on FormBean properties in validation.xml file based on FormBean logical name. This gives following problem.

With respect to Mini project part-1, when FormBean type is ValidatorForm or DynaValidatorForm and multiple form pages are using single FormBean class, then configuring different validator rules for each form page is impossible.

#### AddUser.jsp

id→required

pass→required

**ModifyUser.jsp**

id→required

pass→required

**DeleteUser.jsp**

Id→required

Performing this kind of validator rules configuration with above said setup is impossible when FormBean type is "ValidatorForm" or "DynaValidatorForm" classes.

**Solution-1****Note**

ValidatorActionForm, DynaValidatorActionForm type FormBean classes allows the programmer to configure validator rules on each form page based on the Action class used by the form page.

**Step-1**

Take multiple form pages working with multiple form pages and single FormBean.  
(Action classes are one per form page)

**Step-2**

Take FormBean type as "ValidatorActionForm" or "DynaValidatorActionForm"

**Step-3**

In validation.xml, configure separate validation rules for each form page based on the Action path of Action class used by form page as shown below.

**Sample code based on above steps with respect to mini project part-1****struts-config.xml**

```
<struts-config>
  <form-beans>
    <form-bean name="bean" type="org.apache.struts.validator.DynaValidatorActionForm">
      <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" name="rf" type="MyAction1"/>
    <action path="/xyz2" name="rf" type="MyAction2"/>
    <action path="/xyz3" name="rf" type="MyAction3"/>
  </action-mappings>
</struts-config>
```

**Form pages****AddUser.jsp**

```
<html:form action= "xyz1">
  Id: <html:text property= "id"/><br>
  Passord: <html:text property= "pass"/><br>
  <html:submit value="AddUser"/>
</html:form>
```

Action path of Action class used by AddUser.jsp

**ModifyUser.jsp**

```
<html:form action= "xyz2">
  Id: <html:text property= "id"/><br>
  Passord: <html:text property= "pass"/><br>
  <html:submit value="ModifyUser"/>
</html:form>
```

Action path of Action class used by ModifyUser.jsp

**DeleteUser.jsp**

```
<html:form action= "xyz3">
  Id: <html:text property= "id"/><br>
  <html:submit value="DeleteUser"/>
</html:form>
```

Action path of Action class used by DeleteUser.jsp

**validation.xml**

```
<form-validation>
```

```
  <formset>
```

```
    <form name="/xyz1">
```

```
      <field name= "id" depends= "required">
```

```
      _____
```

```
      _____
```

```
    </field>
```

```
    <field name= "pass" depends= "required">
```

```
    _____
```

```
    _____
```

```
  </field>
```

```
  </form>
```

Validation rules configuration for AddUser.jsp

```
    <form name="/xyz2">
```

```
      <field name= "id" depends= "required,minlength">
```

```
      _____
```

```
      _____
```

```
    </field>
```

```
    <field name= "pass" depends= "required,maxlength">
```

```
    _____
```

```
    _____
```

```
  </field>
```

```
  </form>
```

Validation rules configuration for ModifyUser.jsp

```
    <form name="/xyz3">
```

```
      <field name= "id" depends= "required">
```

```
      _____
```

```
      _____
```

```
    </field>
```

```
  </form>
```

Validation rules configuration for DeleteUser.jsp

```
</form-validation>
```

**Note**

The above sample code setup is using single FormBean class and multiple ActionForms for multiple form pages.

**Solution-2**

This talks about differentiating validator rules for each form page when multiple form pages are using single Action class.

**Step-1**

Make multiple form pages working with a single FormBean and single Action class.

**Step-2**

Take FormBean type as "ValidatorActionForm" or "DynaValidatorActionForm".

**Step-3**

Configure single Action class for multiple times having multiple Action paths.

**Step-4**

Make each form page is using one Action path of the multiple Action paths specified for Action class.

**Step-5**

In "validation.xml" configure separate validator rules for each form page based on the Action path of Action class used by each form page.

## Sample code based on above steps with respect to mini project part-1

### struts-config.xml

```
<struts-config>
  <form-beans>
    <form-bean name="bean" type="org.apache.struts.validator.DynaValidatorActionForm">
      <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" name="rf" type="MyAction"/>
    <action path="/xyz2" name="rf" type="MyAction"/>
    <action path="/xyz3" name="rf" type="MyAction"/>
  </action-mappings>
</struts-config>
```

Single Action class is configured for multiple times having multiple Action paths

### Form pages

#### AddUser.jsp

```
<html:form action="/xyz1">
  Id: <html:text property="id"/><br>
  Passord: <html:text property="pass"/><br>
  <html:submit value="AddUser"/>
</html:form>
```

Action path of Action class used by AddUser.jsp

#### ModifyUser.jsp

```
<html:form action="/xyz2">
  Id: <html:text property="id"/><br>
  Passord: <html:text property="pass"/><br>
  <html:submit value="ModifyUser"/>
</html:form>
```

Action path of Action class used by ModifyUser.jsp

#### DeleteUser.jsp

```
<html:form action="/xyz3">
  Id: <html:text property="id"/><br>
  <html:submit value="DeleteUser"/>
</html:form>
```

Action path of Action class used by DeleteUser.jsp

### validation.xml

```
<form-validation>
  <formset>
    <form name="/xyz1">
      <field name="id" depends="required">
        -----
      </field>
      <field name="pass" depends="required">
        -----
      </field>
    </form>
  </formset>
</form-validation>
```

Validation rules configuration for AddUser.jsp



```

<form name="/xyz2">
  <field name="id" depends="required,minlength">
    _____
  </field>
  <field name="pass" depends="required,maxlength">
    _____
  </field>
</form>

```

Validation rules configuration for  
ModifyUser.jsp

```

<form name="/xyz3">
  <field name="id" depends="required">
    _____
  </field>
</form>

```

Validation rules configuration for DeleteUser.jsp

</form-validation>

For solution-2 based mini project having form validations refer the following application.

### Index.jsp (Welcome page)

```

277 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
278 <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
279 <html:html>
280 <h2><center>Please click on a link below, to perform your desired activity.<br><br>
281   <html:link action="/add">Add User</html:link><br>
282   <html:link action="/modify">Modify User</html:link><br>
283   <html:link action="/delete">Delete User</html:link>
284 </center></h2>
285 </html:html>

```

Refer line-47

Refer line-48

Refer line-49

### web.xml (Deployment Descriptor file)

```

286 <?xml version="1.0" encoding="ISO-8859-1"?>
287 <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
288 "http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">
289 <web-app>
290 <servlet>
291 <servlet-name>Dummy</servlet-name>
292 <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
293 <init-param>
294 <param-name>config</param-name>
295 <param-value>/WEB-INF/struts-config.xml</param-value>
296 </init-param>
297 <load-on-startup>1</load-on-startup>
298 </servlet>
299 <servlet-mapping>
300 <servlet-name>Dummy</servlet-name>
301 <url-pattern>*.do</url-pattern>
302 </servlet-mapping>
303 <welcome-file-list>
304 <welcome-file>Index.jsp</welcome-file>

```

```
305 </welcome-file-list>
306 </web-app>
```

### struts-config.xml (struts configuration file)

```
307 <?xml version="1.0" encoding="UTF-8"?>
308 <!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration
1.1//EN" "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
309 <struts-config>
310 <form-beans>
311 <form-bean name="bean" type="org.apache.struts.validator.DynaValidatorActionForm">
312 <form-property name="id" type="java.lang.String"/>
313 <form-property name="pass" type="java.lang.String"/>
314 <form-property name="function" type="java.lang.String"/>
315 </form-bean>
316 </form-beans>
317 <global-forwards>
318 <forward name="success" path="/success.jsp"/>
319 <forward name="failure" path="/failure.jsp"/>
320 </global-forwards>
321 <action-mappings>
322 <action path="/controller1" name="bean" type="OurAction" parameter="function"
323 validate="true" input="/error.jsp" scope="request"/>
324 <action path="/add" type="org.apache.struts.actions.ForwardAction"
325 parameter="/AddUser.jsp"/>
326 <action path="/modify" type="org.apache.struts.actions.ForwardAction"
327 parameter="/ModifyUser.jsp"/>
328 <action path="/delete" type="org.apache.struts.actions.ForwardAction"
329 parameter="/DeleteUser.jsp"/>
330 <action path="/home" type="org.apache.struts.actions.ForwardAction"
331 parameter="/Index.jsp"/>
332 </action-mappings>
333 <message-resources parameter="ApplicationResources"/>
334 </struts-config>
```

To configure validator rules, based on action path of Struts Action class.

Global ActionForwards pointing to result pages

Single Action class is configured for two times having two different action paths.

ForwardAction classes configurations towards hyperlinks.

### AddUser.jsp (Form page-1)

```
331 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
332 <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
333 <html:html>
334 <html:form action="/controller1">
335 <center><br><br>
336 <table>
337 <caption><font size = 6><u>Adding User</font></h2></u></caption>
338 <tr>
339 <td><bean:message key="form.id"/></td>
```

Action path of DispatchAction class (OurAction). (Refer line-46)

Refer line-274

```

340     <td><html:text property="id"/></td>
341 </tr>
342 <tr>
343     <td><bean:message key="form.pass"/></td>
344     <td><html:password property="pass"/></td>
345 </tr>
346 </table>
347 <br>
348 <html:submit value="AddUser"/>
349 <html:hidden property="function" value="insert"/>
350 </center>
351 </html:form>
352 </html:html>

```

Refer line-275

Target method name in our DispatchAction class.

Additional Request parameter name. (Refer line-46)

### ModifyUser.jsp (Form page -2)

```

353 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
354 <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
355 <html:html>
356 <html:form action="/controller1">
357 <center><br><br>
358 <table>
359 <caption><font size = 6><u>Modifying User</font></h2></u></caption>
360 <tr>
361 <td><bean:message key="form.id"/></td>
362 <td><html:text property="id"/></td>
363 </tr>
364 <tr>
365 <td><bean:message key="form1.pass"/></td>
366 <td><html:password property="pass"/></td>
367 </tr>
368 </table>
369 <br><br>
370 <html:submit value="UpdateUser"/>
371 <html:hidden property="function" value="change"/>
372 </center>
373 </html:form>
374 </html:html>

```

Target method name in our DispatchAction class.

Additional Request parameter name. (Refer line-46)

### DeleteUser.jsp (Form page-3)

```

375 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
376 <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
377 <html:html>
378 <html:form action="/controller2">
379 <center><br><br>
380 <table>
381 <caption><font size = 6><u>Deleting User</font></h2></u></caption>
382 <tr>
383 <td><bean:message key="form.id"/></td>
384 <td><html:text property="id"/></td>
385 </tr>
386 </table>
387 <br><br>
388 <html:submit value="DeleteUser"/>
389 <html:hidden property="function" value="remove"/>
390 </center>
391 </html:form>
392 </html:html>

```

Target method name in our DispatchAction class.

Additional Request parameter name. (Refer line-46)

## OurAction.java (Our DispatchAction Class)

```

393 import java.io.*;
394 import org.apache.struts.action.*;
395 import org.apache.struts.actions.*;
396 import java.sql.*;
397 import javax.naming.*;
398 import javax.sql.*;
399 import javax.servlet.http.*;
400 import org.apache.struts.validator.*;
401 public class OurAction extends DispatchAction → Mandatory
402 {
403     private Connection getConnection() → User-defined method having logic to
404     {                                     create JDBC connection object.
405         Connection con = null;
406         try
407         {
408             Class.forName("oracle.jdbc.OracleDriver");
409             con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE", "Scott", "Tiger");
410         }
411         catch(Exception e)
412         {
413         }
414         return con; → Returns the JDBC connection object.
415     }
416     private void releaseResources(Connection con, PreparedStatement pst)
417     {
418         if(con != null)
419         {
420             try
421             {
422                 con.close();
423             }
424             catch(Exception e)
425             {
426             }
427             if(pst != null)
428             {
429                 try
430                 {
431                     pst.close();
432                 }
433                 catch(Exception e)
434                 {
435                 }
436             }
437         }
438     }
439     public ActionForward insert(ActionMapping mapping, ActionForm f, HttpServletRequest
440     request, HttpServletResponse response)
441     {
442         Connection con = null;
443         PreparedStatement pst = null;
444         try
445         {
446             DynaValidatorForm form=(DynaValidatorForm)f; //Typcasting
447             con = this.getConnection(request);
448             pst=con.prepareStatement("INSERT INTO STRUTS_DISP_ACTION_TEST VALUES(?, ?)");
449             pst.setString(1,(String)form.get("id"));

```

```

447 pst.setString(2,(String)form.get("pass"));
448 int i = pst.executeUpdate();
449 request.setAttribute("Action", "Insert");
450 if(i != 0)
451 {
452 return mapping.findForward("success");
453 }
454 else
455 {
456 return mapping.findForward("failure");
457 }
458 } // try
459 catch(Exception e)
460 {
461 return mapping.findForward("failure");
462 }
463 finally
464 {
465 releaseResources(con,pst);
466 }} // addUser()
467 public ActionForward remove(ActionMapping mapping,ActionForm f,HttpServletRequest
    request,HttpServletResponse response)
468 {
469 Connection con = null;
470 PreparedStatement pst = null;
471 try
472 {
473 DynaValidatorForm form=( DynaValidatorForm)f;
474 con=getConnection(request);
475 pst=con.prepareStatement("DELETE FROM STRUTS_DISP_ACTION_TEST WHERE ID=?");
476 pst.setString(1,(String)form.get("id"));
477 int i=pst.executeUpdate();
478 request.setAttribute("Action", "Deletion");
479 if(i != 0)
480 {
481 return mapping.findForward("success");
482 }
483 else
484 {
485 return mapping.findForward("failure");
486 }}
487 catch(Exception e)
488 {
489 return mapping.findForward("failure");
490 }
491 finally
492 {
493 releaseResources(con, pst);
494 }} // deleteUser()
495 public ActionForward change(ActionMapping mapping,ActionForm f,HttpServletRequest
    request,HttpServletResponse response)
496 {
497 Connection con = null;
498 PreparedStatement pst = null;
499 try
500 {
501 DynaValidatorForm form=( DynaValidatorForm)f;

```

Representing the insert operation of the current method.

```

502 con = this.getConnection(request);
503 pst=con.prepareStatement("UPDATE STRUTS_DISP_ACTION_TEST SET PASS=? WHERE ID=?");
504 pst.setString(1,(String)form.get("pass"));
505 pst.setString(2,(String)form.get("id"));
506 int i = pst.executeUpdate();
507 request.setAttribute("Action", "Updation");
508 if(i != 0)
509 return mapping.findForward("success");
510 else
511 return mapping.findForward("failure");
512 }
513 catch(Exception e)
514 {
515 return mapping.findForward("failure");
516 }
517 finally
518 {
519 releaseResources(con,pst);
520 }
521 } // updateUser()
522 } // OurAction class.

```

### success.jsp (Result page-1)

```

523 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
524 <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
525 <html>
526 <body>
527 <div align="center">
528 <font color="#0000FF" size="5"><em><strong>
529 <%= (String)request.getAttribute("Action")%>Operation Successful <br>
530 and values are <%= (String)request.getParameter("id")%>
531 and <%= (String)request.getParameter("pass")%>
532 </strong></em></font>
533 </div>
534 <center>
535 <br><br>
536 <html:link action="/home">HOME</html:link><br>
537 </center>
538 </body>
539 </html>

```

Use request attribute value given by Action class methods, indicating the operation that is performed.

Refer line-50

### failure.jsp (Result page-2)

```

540 <html>
541 <body>
542 <p align="center"><strong><font size="5"><em>
543 Operation failed !!! </em></font></strong>
544 </p>
545 </body>
546 </html>

```

### ApplicationResources.properties (Properties file)

```

547 errors.required={0} is required.<br>
548 errors.minlength={0} cannot be less than {1} characters.<br>
549 errors.maxlength={0} cannot be greater than {1} characters.<br>
550 errors.invalid={0} is invalid.It can take only alphenumerics.<br>
551 errors.byte={0} must be a byte.<br>
552 errors.short={0} must be a short.<br>
553 errors.integer={0} must be an integer.<br>

```

```

554 errors.long={0} must be a long.<br>
555 errors.float={0} must be a float.<br>
556 errors.double={0} must be a double.<br>
557 errors.date={0} is not a date.<br>
558 errors.range={0} is not in the range {1} through {2}.<br>
559 errors.creditcard={0} is an invalid credit card number.<br>
560 errors.email={0} is an invalid e-mail address.<br>
561 bean.id=<b><u>User Name</u></b>
562 bean.pass=<b><u>Password</u></b>
563 form.id=<b><u>User Name</u></b>
564 form.pass=<b><u>Password</u></b>
565 form1.pass=<b><u>New Password</u></b>

```

## validator-rules

```

566 <!DOCTYPE form-validation PUBLIC
567 "-//Apache Software Foundation//DTD Commons Validator Rules Configuration 1.0/EN"
568 "http://jakarta.apache.org/commons/dtds/validator_1_0.dtd">
569 <form-validation>
570 <global>
571 <validator name="required"
572 classname="org.apache.struts.validator.FieldChecks" method="validateRequired"
573 methodParams="java.lang.Object, org.apache.commons.validator.ValidatorAction,
574 org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors,
575 javax.servlet.http.HttpServletRequest" msg="errors.required">
576 <javascript><![CDATA[
577 function validateRequired(form) {
578 var bValid = true;
579 var focusField = null;
580 var i = 0;
581 var fields = new Array();
582 oRequired = new required();
583 for (x in oRequired) {
584 var field = form[oRequired[x][0]];
585 if (field.type == 'text' || field.type == 'textarea' || field.type == 'file' || field.type == 'select-one' ||
586 field.type == 'radio' || field.type == 'password') {
587 var value = "";
588 // get field's value
589 if (field.type == "select-one") {
590 var si = field.selectedIndex;
591 if (si >= 0) {
592 value = field.options[si].value;
593 }
594 } else {
595 value = field.value;

```

```

596 }
597 if (value == "") {
598 if (i == 0) {
599 focusField = field;
600 }
601 fields[i++] = oRequired[x][1];
602 bValid = false;
603 }
604 }
605 }
606 if (fields.length > 0) {
607 focusField.focus();
608 alert(fields.join("\n"));
609 }
610 return bValid;
611 }]]>
612 </javascript>
613 </validator>
614 <validator name="minlength"
615 classname="org.apache.struts.validator.FieldChecks" method="validateMinLength"
616 methodParams="java.lang.Object, org.apache.commons.validator.ValidatorAction,
617 org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors,
618 javax.servlet.http.HttpServletRequest" depends=" "msg="errors.minlength">
619 <javascript><![CDATA[
620 function validateMinLength(form) {
621 var bValid = true;
622 var focusField = null;
623 var i = 0;
624 var fields = new Array();
625 oMinLength = new minlength();
626 for (x in oMinLength) {
627 if (form[oMinLength[x][0]].type == 'text' ||
628 form[oMinLength[x][0]].type == 'textarea') {
629 var iMin = parseInt(oMinLength[x][2]("minlength"));
630 if (form[oMinLength[x][0]].value.length < iMin) {
631 if (i == 0) {
632 focusField = form[oMinLength[x][0]];
633 }
634 fields[i++] = oMinLength[x][1];
635 bValid = false;

```



```

636 }
637 }}
638 if (fields.length > 0) {
639   focusField.focus();
640   alert(fields.join("\n"));
641 }
642 return bValid;
643 }]]>
644 </javascript>
645 </validator>
646 <validator name="maxlength"
647   classname="org.apache.struts.validator.FieldChecks" method="validateMaxLength"
648   methodParams="java.lang.Object, org.apache.commons.validator.ValidatorAction,
649   org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors,
650   javax.servlet.http.HttpServletRequest" depends=" "msg="errors.maxlength">
651 <javascript><![CDATA[
652   function validateMaxLength(form) {
653     var bValid = true;
654     var focusField = null;
655     var i = 0;
656     var fields = new Array();
657     oMaxLength = new maxlength();
658     for (x in oMaxLength) {
659       if (form[oMaxLength[x][0]].type == 'text' || form[oMaxLength[x][0]].type == 'textarea') {
660         var iMax = parseInt(oMaxLength[x][2]("maxlength"));
661         if (form[oMaxLength[x][0]].value.length > iMax) {
662           if (i == 0) {
663             focusField = form[oMaxLength[x][0]];
664           }
665           fields[i++] = oMaxLength[x][1];
666           bValid = false;
667         }
668       }}
669       if (fields.length > 0) {
670         focusField.focus();
671         alert(fields.join("\n"));
672       }
673       return bValid;
674     }]]>
675 </javascript>

```

```

676 </validator>
677 <validator name="mask" classname="org.apache.struts.validator.FieldChecks"
678 method="validateMask" methodParams="java.lang.Object,
679 org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field,
680 org.apache.struts.action.ActionErrors, javax.servlet.http.HttpServletRequest" depends=""
681 msg="errors.invalid">
682 <javascript><![CDATA[
683 function validateMask(form) {
684 var bValid = true;
685 var focusField = null;
686 var i = 0;
687 var fields = new Array();
688 oMasked = new mask();
689 for (x in oMasked) {
690 if ((form[oMasked[x][0]].type == 'text' || form[oMasked[x][0]].type == 'textarea'
691 || form[oMasked[x][0]].type == 'password') && (form[oMasked[x][0]].value.length > 0))
692 {
693 if (!matchPattern(form[oMasked[x][0]].value, oMasked[x][2]("mask"))) {
694 if (i == 0){
695 focusField = form[oMasked[x][0]];
696 }
697 fields[i++] = oMasked[x][1];
698 bValid = false;
699 }
700 }}
701 if (fields.length > 0) {
702 focusField.focus();
703 alert(fields.join("\n"));
704 }
705 return bValid;
706 }
707 function matchPattern(value, mask)
708 {
709 var bMatched = mask.exec(value);
710 if (!bMatched) {
711 return false;
712 }
713 return true;
714 }]}>
715 </javascript>

```

```

716 </validator>
717 -----
718 -----

```

#### validation.xml

```

719 <?xml version="1.0" encoding="ISO-8859-1" ?>
720 <!DOCTYPE form-validation PUBLIC
721 "-//Apache Software Foundation//DTD Commons Validator Rules Configuration 1.0/EN"
722 "http://jakarta.apache.org/commons/dtds/validator_1_0.dtd">
723 <form-validation>
724 <formset>
725 <form name="/controller1">
726 <field property="id" depends="required, maxlength">
727 <arg0 key="bean.id"/>
728 <arg1 name="maxlength" key="${var:maxlength}" resource="false"/>
729 <var>
730 <var-name>maxlength</var-name>
731 <var-value>15</var-value>
732 </var>
733 </field>
734 <field property="pass" depends="required, minlength, maxlength, mask">
735 <arg0 key="bean.pass"/>
736 <arg1 name="minlength" key="${var:minlength}" resource="false"/>
737 <arg1 name="maxlength" key="${var:maxlength}" resource="false"/>
738 <var>
739 <var-name>minlength</var-name>
740 <var-value>5</var-value>
741 </var>
742 <var>
743 <var-name>maxlength</var-name>
744 <var-value>15</var-value>
745 </var>
746 <var>
747 <var-name>mask</var-name>
748 <var-value>^[0-9a-zA-Z]*$</var-value>
749 </var>
750 </field>
751 </form>
752 <form name="/controller2">
753 <field property="id" depends="required, maxlength">
754 <arg0 key="bean.id"/>
755 <arg1 name="maxlength" key="${var:maxlength}" resource="false"/>
756 <var>
757 <var-name>maxlength</var-name>
758 <var-value>15</var-value>
759 </var>
760 </field>
761 </form>
762 </formset>
763 </form-validation>

```

Validator rules  
configuration  
for  
AddUser.jsp,  
ModifyUser.jsp

Validator rules  
configuration  
for  
DeleteUser.jsp

#### error.jsp (input page)

```

764 <%@ taglib uri="/tags/struts-html" prefix="html" %>
765 <html:html>
766 <center><br>
767 <html:errors />
768 <br><br>

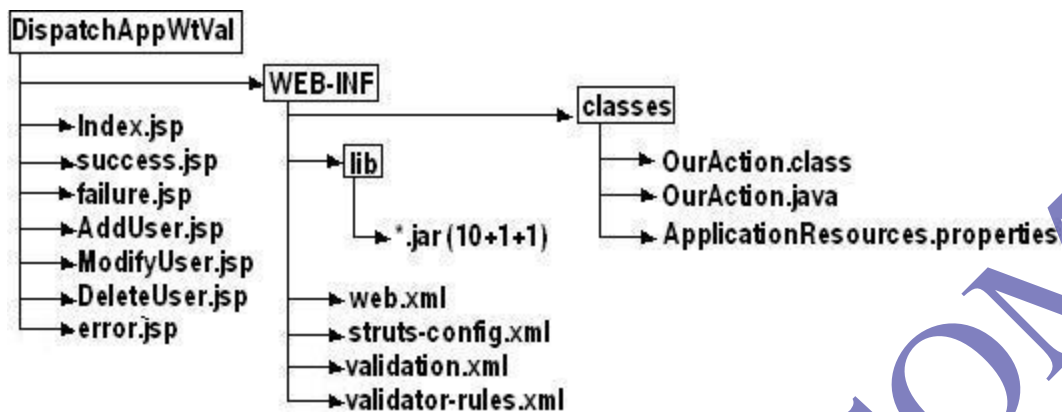
```

```

769 <html:link action="/home">HOME</html:link><br>
770 </center>
771 </html:html>

```

### Deployment Directory Structure



### Q) Can you explain different types of FormBeans and their behaviors?

**Ans:** ActionForm, DynaActionForm can be used only for programmatic form validations but they cannot be used for ValidatorPlugIn based form validations.

ValidatorForm, DynaValidatorForm can be used for both programmatic and ValidatorPlugIn based declarative form validations. These two classes allow to configure validator rules on FormBean properties through FormBean logical name.

ValidatorActionForm, DynaValidatorActionForm allows to work with both programmatic and ValidatorPlugIn based declarative form validations but the validator rules on the FormBean properties must be configured through the Action path of Action class.

### Note

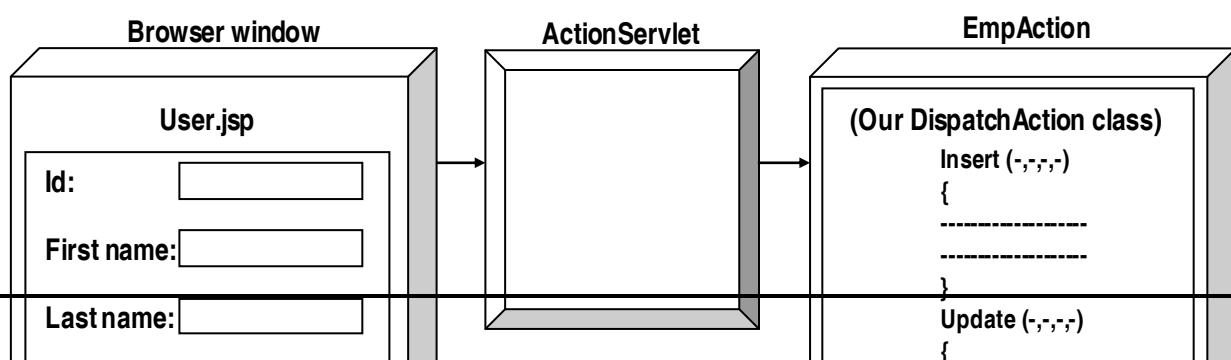
All DynaXxxForm classes are capable of generating formbeans dynamically.

### Making multiple buttons of form page targeting to execute multiple methods of DispatchAction class.

There are 3 approaches to handle the above situation.

#### Approach-1

Take submit button captions as the method names of DispatchAction class.



**Note**

- 1) When submit button is taken without logical name related to submit button no request parameter will be formed when form is submitted.

**Example**

```
<html:form action= "register" method= "get">
-----
-----
<html:submit value= "CheckDetails"/>
</html:form>
```

- 2) The submit button caption "CheckDetails" will go to server as request parameter value (s1=CheckDetails).

```
<html:form action= "register" method= "get">
-----
-----
<html:submit property= "s1" value= "CheckDetails"/>
</html:form>
```

**Implementation steps for approach-1****Step-1**

Take same logical name for multiple submit buttons with different captions as shown below.

```
<html:form action= "emp" method= "get">
-----
-----
<html:submit property= "s1" value= "insert"/>
<html:submit property= "s1" value= "update"/>
<html:submit property= "s1" value= "remove"/>
</html:form>
```

**Step-2**

Develop your own DispatchAction class having submit buttons' captions as user-defined method names.

**Step-3**

Configure your DispatchAction class in struts configuration file by having the logical name of submit button (s1) as additional request parameter name.

**In struts-config.xml**

```
<action path= "\emp" type= "EmpAction" name= "rf" parameter= "s1">
-----
-----
</action>
```

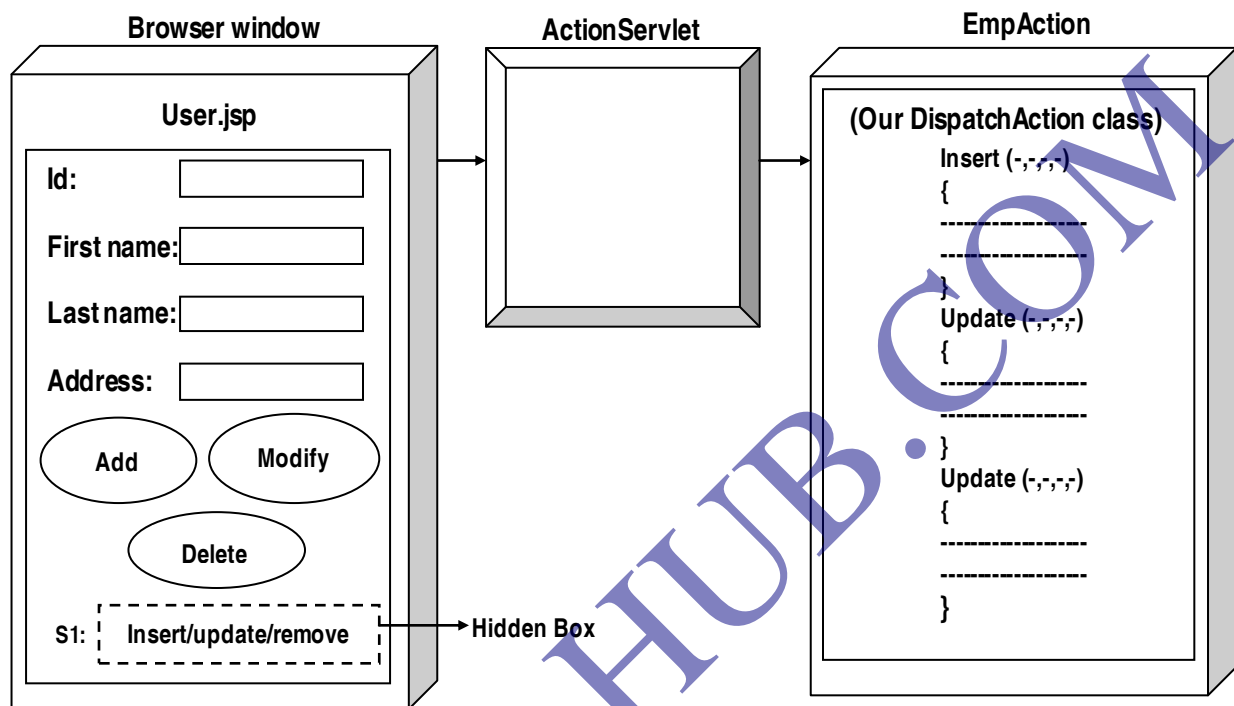
**Note**

Don't forget to configure "s1" logical name of submit buttons as FormBean property in FormBean class.

### Limitation

Submit button captions are tightly coupled with our DispatchAction class method name. it kills the flexibility of modifying submit button captions freely.

### Approach-2 (By using JavaScript and hidden forms)



### Implementation steps

#### Step-1

Develop DispatchAction class of your own choice user-defined method names.

#### Step-2

Take form page having multiple submit buttons with your own choice captions.

#### Step-3

Take hidden box having name. use JavaScript support to place respective DispatchAction class method name in the hidden box based on the submit button that is clicked.

#### Example

If "add" submit button is clicked the hidden box should have the insert word as value.

#### Step-4

Configure our DispatchAction class in struts configuration file by taking hidden box logical name "s1" as additional request parameter name as follows.

#### In struts-config.xml

```
<action path= "/emp" type= "EmpAction" name= "rf" parameter= "s1">
-----
-----
</action>
```

## Advantage

- ☑ Submit button captions are loosely coupled with the DispatchAction class method names. So, they are very flexible to change freely.

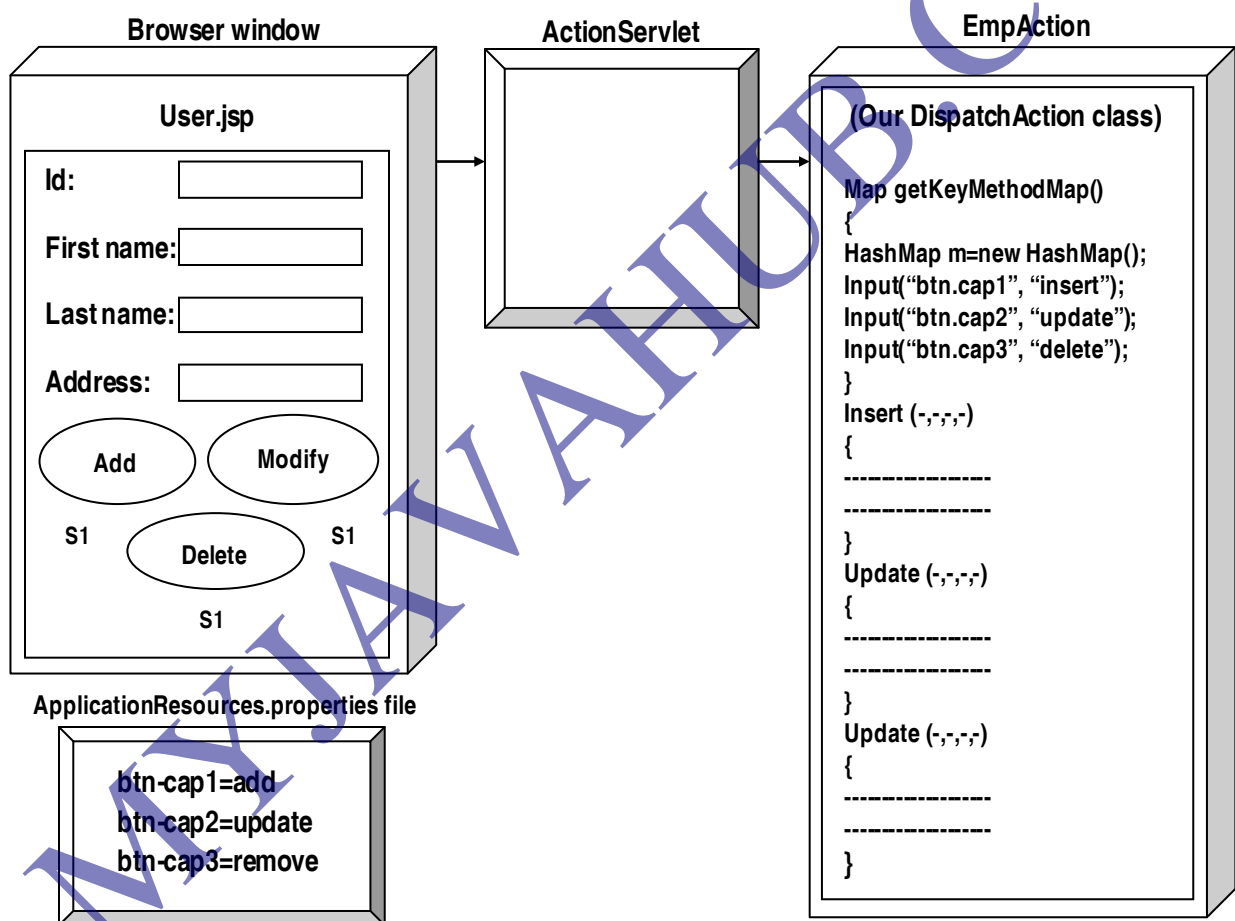
## Disadvantage

- ☒ If JavaScript execution is disabled through browser settings, this technique will not work.

## Approach-3 (With LookupDispatchAction class)

LookupDispatchAction class is an abstract class and sub class for DispatchAction class. This class contains one abstract method called “protected Map getKeyMethodMap()”.

- Bringing value from property file based on the key is called as “Lookup Operation”.
- Bringing key from property file based on the value is called as “Reverse Lookup Operation”.
- The Reverse LookupDispatchAction action class runs on the principle called Reverse Lookup Operation.



## In struts-config.xml

```
<action path="/xml" name="rf" type="OurAction" parameter="s1">
```

</action>

## Implementation steps

### Step-1

Prepare properties file having submit buttons captions.

### Step-2

Take multiple submit buttons in the form page having same logical name and different captions. Gather these captions from properties file.

### Step-3

Develop Struts Action class extending from LookupDispatchAction class

### Step-4

Take your own choice method names in the Action class having no relation with the submit button captions.

### Step-5

Implement getKeyMethodMap() returning Map DataStructure having keys and values. These keys must be keys in the properties file which holds submit button caption. The values must be method names of OurAction class(LookupDispatchAction)

### Step-6

Configure our LookupDispatchAction class by specifying the logical name of multiple submit buttons as additional request parameter name.

## Flow of execution with respect to “Add” button

- I. End-user clicks on “Add” submit button in the form page.
- II. ActionServlet traps and takes the request having s1=add as RequestParameter value along with other request parameters.
- III. ActionServlet calls execute(-,-,-) method on our LookupDispatchAction class.
- IV. This super class execute(-,-,-) method
  - a) Reads the additional request parameter (s1) value (add).
  - b) Calls getKeyMethodMap() of our LookupDispatchAction class and gets Map object.
  - c) Uses the value “add” and gets key (btn.cap1) by performing reverse lookup operation on properties file.
  - d) Gets “btn.cap1” related value (insert) from the above received Map data structure (refer (b)).
  - e) Calls insert(-,-,-) method on our LookupDispatchAction class object to process the request related to “Add” submit button.

## Advantage

- ☑ No need of working with JavaScript and submit button captions are not tied with LookupDispatchAction class method names.

**For example application on LookupDispatchAction refer the following application.**

## User.jsp (form page)

1   <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean"%>



```

2  <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html"%>
3  <html>
4    <head>
5      <title>JSP for EmpForm form</title>
6    </head>
7    <body>
8      <html:form action="/emp">
9        userid : <html:text property="userid"/><html:errors property="userid"/><br/>
10       firstname : <html:text property="firstname"/><html:errors property="firstname"/><br/>
11       lastname : <html:text property="lastname"/><html:errors property="lastname"/><br/>
12       address : <html:text property="address"/><html:errors property="address"/><br/>
13       <html:submit property="function">
14         <bean:message key="btn.cap1"/>
15       </html:submit>
16       <html:submit property="function">
17         <bean:message key="btn.cap2"/>
18       </html:submit>
19       <html:submit property="function">
20         <bean:message key="btn.cap3"/>
21       </html:submit>
22       <html:cancel/>
23     </html:form>
24     <%if(request.getAttribute("operation")!=null)>
25     {
26     %>
27     <%=request.getAttribute("operation") %> is success
28     <%} %>
29   </body>
30 </html>

```

Multiple submit buttons having same logical name and with different captions gathered from properties file.

#### web.xml (Deployment Descriptor File)

```

31 <?xml version="1.0" encoding="UTF-8"?>
32 <web-app xmlns="http://java.sun.com/xml/ns/javaee"
33   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.4"
34   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
35     http://java.sun.com/xml/ns/javaee/web-app_2_4.xsd">
36   <servlet>
37     <servlet-name>action</servlet-name>
38     <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
39     <init-param>
40       <param-name>config</param-name>
41       <param-value>/WEB-INF/struts-config.xml</param-value>
42     </init-param>
43     <load-on-startup>0</load-on-startup>
44   </servlet>
45   <servlet-mapping>
46     <servlet-name>action</servlet-name>
47     <url-pattern>*.do</url-pattern>
48   </servlet-mapping>
49   <welcome-file-list>
50     <welcome-file>User.jsp</welcome-file>
51   </welcome-file-list>
52 </web-app>

```

#### struts-config.xml (Struts configuration file)

```

50 <?xml version="1.0" encoding="UTF-8"?>
51 <!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration
52   1.2//EN" "http://struts.apache.org/dtds/struts-config_1_2.dtd">

```

```

52 <struts-config>
53 <form-beans>
54 <form-bean name="ef" type="EmpForm" />
55 </form-beans>
56 <action-mappings>
57 <action attribute="ef" input="/User.jsp" name="ef" parameter="function" path="/emp"
58 scope="request" type="EmpAction">
59 <forward name="failure" path="/failure.jsp" />
60 <forward name="success" path="/User.jsp" />
61 </action>
62 </action-mappings>
63 <message-resources parameter="struts.ApplicationResources" />
64 </struts-config>

```

Additional RequestParameter name

Action path of LookupDispatchAction class.

### EmpForm.java (FormBean class)

```

65 import org.apache.struts.action.ActionForm;
66 public class EmpForm extends ActionForm
67 {
68 private String address;
69 private String userid;
70 private String lastname;
71 private String firstname;
72 public String getAddress()
73 {
74 return address;
75 }
76 public void setAddress(String address)
77 {
78 this.address = address;
79 }
80 public String getUserid()
81 {
82 return userid;
83 }
84 public void setUserid(String userid)
85 {
86 this.userid = userid;
87 }
88 public String getLastname()
89 {
90 return lastname;
91 }
92 public void setLastname(String lastname)
93 {
94 this.lastname = lastname;
95 }
96 public String getFirstname()
97 {
98 return firstname;
99 }
100 public void setFirstname(String firstname)
101 {
102 this.firstname = firstname;
103 }
104 }

```

### EmpAction.java (Action class)

```

105 import db.DBConnection;
106 import java.io.PrintStream;
107 import java.sql.Connection;

```

```

108 import java.sql.PreparedStatement;
109 import java.util.HashMap;
110 import java.util.Map;
111 import javax.servlet.http.HttpServletRequest;
112 import javax.servlet.http.HttpServletResponse;
113 import org.apache.struts.action.*;
114 import org.apache.struts.actions.LookupDispatchAction;
115 public class EmpAction extends LookupDispatchAction
116 {
117     public EmpAction()
118     {
119     }
120     protected Map getKeyMethodMap()
121     {
122         System.out.println("getKeyMethodMap():EmpAction");
123         Map map = new HashMap();
124         map.put("btn.cap1", "insert");
125         map.put("btn.cap2", "update");
126         map.put("btn.cap3", "delete");
127         return map;
128     }
129     public ActionForward insert(ActionMapping mapping, ActionForm form,
130     HttpServletRequest request, HttpServletResponse response)
131     {
132         String output;
133         Connection con;
134         PreparedStatement ps;
135         DBConnection dbcon;
136         output = null;
137         con = null;
138         ps = null;
139         dbcon = null;
140         try
141         {
142             System.out.println("insert():EmpAction");
143             EmpForm ef = (EmpForm)form;
144             dbcon = new DBConnection();
145             con = dbcon.getConnection();
146             ps = con.prepareStatement("insert into employee_info values(?,?,?,?)");
147             ps.setString(1, ef.getUserid());
148             ps.setString(2, ef.getFirstname());
149             ps.setString(3, ef.getLastname());
150             ps.setString(4, ef.getAddress());
151             int result = ps.executeUpdate();
152             System.out.println(result);
153             if(result != 0)
154                 output = "success";
155             else
156                 output = "failure";
157             request.setAttribute("operation", "insert operation");
158         }
159         catch(Exception e)
160         {
161             e.printStackTrace();
162         }
163         dbcon.releaseResourc(ps, con);
164         Exception exception;
165         exception;

```

```

166 dbcon.releaseResourc(ps, con);
167 throw exception;
168 dbcon.releaseResourc(ps, con);
169 return mapping.findForward(output);
170 }
171 public ActionForward update(ActionMapping mapping, ActionForm form,
172 HttpServletRequest request, HttpServletResponse response)
173 {
174 String output;
175 Connection con;
176 PreparedStatement ps;
177 DBConnection dbcon;
178 output = null;
179 con = null;
180 ps = null;
181 dbcon = null;
182 try
183 {
184 System.out.println("insert():EmpAction");
185 EmpForm ef = (EmpForm)form;
186 dbcon = new DBConnection();
187 con = dbcon.getConnection();
188 ps = con.prepareStatement("update employee_info set firstname=?,lastname=?,address=?
189 where empid=?");
190 ps.setString(1, ef.getUserid());
191 ps.setString(1, ef.getFirstname());
192 ps.setString(3, ef.getLastname());
193 ps.setString(4, ef.getAddress());
194 int result = ps.executeUpdate();
195 if(result != 0)
196 output = "success";
197 else
198 output = "failure";
199 request.setAttribute("operation", "update operation");
200 }
201 catch(Exception e)
202 {
203 e.printStackTrace();
204 }
205 dbcon.releaseResourc(ps, con);
206 Exception exception;
207 exception;
208 dbcon.releaseResourc(ps, con);
209 throw exception;
210 dbcon.releaseResourc(ps, con);
211 return mapping.findForward(output);
212 }
213 public ActionForward delete(ActionMapping mapping, ActionForm form,
214 HttpServletRequest request, HttpServletResponse response)
215 {
216 String output;
217 Connection con;
218 PreparedStatement ps;
219 DBConnection dbcon;
220 output = null;
221 con = null;
222 ps = null;
223 dbcon = null;

```

```

224 try
225 {
226 System.out.println("insert():EmpAction");
227 EmpForm ef = (EmpForm)form;
228 dbcon = new DBConnection();
229 con = dbcon.getConnection();
230 ps = con.prepareStatement("delete from employee_info where empid=?");
231 ps.setString(1, ef.getUserid());
232 int result = ps.executeUpdate();
233 if(result != 0)
234 output = "success";
235 else
236 output = "failure";
237 request.setAttribute("operation", "Delete operation");
238 }
239 catch(Exception e)
240 {
241 e.printStackTrace();
242 }
243 dbcon.releaseResourc(ps, con);
244 Exception exception;
245 exception;
246 dbcon.releaseResourc(ps, con);
247 throw exception;
248 dbcon.releaseResourc(ps, con);
249 return mapping.findForward(output);
250 }
251 }

```

**DBConnection.java (Helper java class for Action class having JDBC code)**

```

252 package db;
253 import java.io.PrintStream;
254 import java.sql.*;
255 public class DBConnection
256 {
257 public Connection getConnection()
258 {
259 try
260 {
261 Connection con;
262 System.out.println("getConnection(): DBConnection");
263 Class.forName("oracle.jdbc.driver.OracleDriver");
264 con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","scott","tiger");
265 return con;
266 }
267 Catch(Exception e)
268 {
269 return null;
270 }
271 }
272 public void releaseResourc(PreparedStatement ps, Connection con)
273 {
274 System.out.println("releaseResourc():DBConnection");
275 try
276 {
277 ps.close();
278 }
279 catch(Exception e)
280 {
281 e.printStackTrace();
282 }

```

```

283 try
284 {
285 con.close();
286 }
287 catch(Exception e)
288 {
289 e.printStackTrace();
290 }
291 }
292 }

```

failure.jsp (Result page)

```
293 <%=request.getAttribute("operation")%> is failed
```

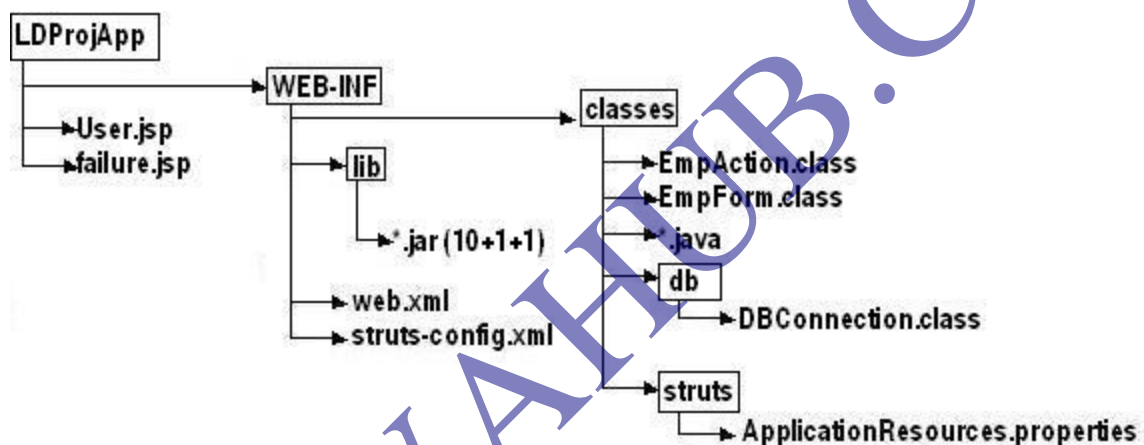
ApplicationResources.properties (Properties file having submit button captions)

```

294 btn.cap1=add
295 btn.cap2=Modify
296 btn.cap3=Remove

```

### Deployment Directory Structure



### JBoss

Type: Application server

Version: 5.x compatible with JDK 1.6

Vendor: Apache Foundation (Acquired by Red Hat)

Default Port No:8080 (changeable through server.xml)

Address to create domains the default domains are default, web, standard, minimal, all.

Download software as Zip file from [www.jboss.org](http://www.jboss.org) website.

To install software extract zip file to a folder.

### Procedure to configure JBoss server in MyEclipse

Window menu-> preferences-> MyEclipse-> servers-> JBoss-> JBoss 5x -> enable.

JBoss home directory

- ❖ The Exception handling in web application will give two benefits.

- i) Avoid abnormal termination in the execution of web application when exceptions are raised.
- ii) To Display non-technical layman messages based web pages for end user when exceptions are raised.

### Exceptions Handling in FormBean class:

- 1) Local exception handling (specific to each JSP program) (use errorpages is Errorpage attributes of <%page%>)
- 2) Global exception handling (use <error-page> of web.xml common for all the JSP programs of struts applications)

**Note:** If both configure with respect to single JSP progress the setting done in local exception handling will effective.

### Programmatic Exception Handling in struts Action Class

```
public class MyAction extends Action
{
    public Actionforward execute(-,-,-)
    {
        try
        {
            .....
            .....(code that may raise exceptions.)
            return mapping.find Forward("success| failure");
        }
        catch(Exception e)
        {
            return mapping find Forward ("err");
        }
    }
}
```

In struts-config.xml:

```
<action name="rf" type="MyAction" path="xyz">
    <forward name="success" path="/success-Jsp"/>
    <forward name="failure" path="/failure.jsp"/>
    <forward name="err" path="/error.jsp"/>
</action>
```

error.jsp

```
<b><i> <center> Internal problem <b></i></center>
```

### Declarative Exception Handling on StrutsAction class

- 1) Local Exception Handling & specific to one struts Action class.
- 2) Global Exception Handling (common for all the Action classes of struts Application)

**Note:** If with are configured settings done in local exception handling will be effected.

### Sample Code

#### Action class

```
public class MyAction extends Action
{
```

```

public Actionforward execute (-,-,-) throws Exception
{
    .....
    .....
}
return mapping find Forward("Success | Failure");
}

```

**In struts.xml: (for local declarative exception handling on struts action class)**

```

<struts-config>
  <form-beans>
    .....
  </form-beans>
  <action-mapping>
    <action path="xyz" name="rf" type="MyAction">
      <exception type="java.lang.Exception" path="myerr.jsp" key="my.exp.msg">
        <forward name="success" path="|success.jsp">
        <forward name="failure" path="|failure.jsp">
      </action>
    </action-mapping>
  </struts-config>

```

**In properties file**

```
My.exp.msg=<b><i> Internal problem </i></b>
```

**In myErr.jsp**

```
<html:errors1>
```

- ❖ The <exception> in the above struts configuration files performs local Exception handling by launching MyErr.jsp page having the message collected from "My.exp.ms" key of properties file when Exception raised in MyAction class.

**In s-c.xml (for global declarative exception handling of struts application)**

```

<struts-config>
  <form-beans>
    .....
  </form-beans>
  <global-exceptions>
    <exceptions type="java.lang.Exception" path="|Myerr.jsp" key="My.exp.msg">
    </global-exceptions>
  <global-forwards>
    .....
  </global-forwards>
  <action-mappings>
    <action-path="xyz" name="rf" type="MyAction">
      .....(local forward configuration)
    </action>
    <action path="xyz1" name="rf" type="MyAction1">
      .....
      ..... (local forward configurations)
    </action>
  </action-mappings>

```



```
</struts-config>
```

### Programmatic Exception Handling with Error JSP configuration on FormBean class

```
public class Registerform extends ActionForm
{
    String username, password;
    //write setxxx(-), getxxx(-) methods
    .....
    .....
    public ActionError validate (Action Mapping mapping, Http Servlet Request rq)
    {
        Action Errors errs;
        try
        {
            errs=new Actionerrors();
            .....
            .....(form validation logic code that may generate Exceptins)
        }
        catch (Exception e)
        {
            errs.add("exp err", new Action Message ("My.exp.msg"));
            return errs;
        }
    } //method
} //class
```

#### In properties file

```
My.exp.msg=<b><i> Internal problem </i></b>
```

#### In s-c.xml

```
<struts-config>
<form=beans>
<form-bean name="rf" type="app.RegisterForm"/>
</form-beans>
<action-mapping>
<action path="xyz name="rf" input="/register.jsp" type="RegisterAction">
.....
.....
</action>
</action-mapping>
</struts-config>
```

#### In register.jsp page

```
.....
.....
<html:errors>
```

#### Local ExceptionHandling in JSP program (not related to struts)

```
Work with error page is Errorpage attribute of <%@page%>
```

#### Main JSP program (register.jsp)

```
<%@page errorpage="error.jsp"%>
.....
```

```

.....
<%class.forName("abc");%>
<html:errors/>

```

### error.jsp (error page)

```

<%@page is Errorpage="true"%>
<b><i> internal problem </i></b>

```

### Note

When exception is raised in register.jsp cont comes to error.jsp program.

### Global exception handling in JSP program

#### register.jsp

```

.....
.....
<%class.forName("abc");%> — code that may raise exception.
<html:errors/>

```

#### failure.jsp

```

<%class.forName("abc");%> — code that may raise exception.
.....
.....

```

### error.jsp (error page)

```

<b><i> Internal problem </i></b>

```

#### web.xml

```

<web-app>
.....
.....
.....
<error-page>
<exception-type> java.lang.Exception </exception-type>
<location> /error.jsp <location>
</error-page>
</web-app>

```

In the above web.xml code handles all the exceptions raised in all the JSP programs of web-application.

All ways develop web application by keeping non-technical end users in mind. In that process the above said Exception Handling is quite important process.

### Internationalization (I18N)

Making our application / web application specific to one "locale" is called as "localization".

Making our web application ready to work with multiple "locales" is called as Internationalization (I18N)

Locale means country + language

**Examples:** en-US, fr-FR, de-DE, fr-CA, hi-IN

- ❖ While executing Application (or) web application that deals with Internationalization we must install certain special language packs.

- ❖ The I18N enabled application renders the presentation logic label in the language that is chosen by end user.
- ❖ To enable I18N we need to work with multiple properties files and each properties file should have one local specific presentation logic labels.
- ❖ The I18N enabled application can do business with global customers and clients belonging to different “locales”
- ❖ In most of the situation we can use the google translator tool to get other languages script based words on the given English script. If that is not possible work with UNICODE character set this also not possible take the support of some third party tools which converts English script to certain target language related light weight image and make that image as web page content.
- ❖ In standalone and regular classic web applications we need to use “java.util.ResourceBundle” class for I18N effect.
- ❖ Struts gives built in support to work with I18N concepts.

### **The multiple properties files for I18N based Web Applications:**

#### **App.properties (based file)**

Label 1= hello  
Label 2= how r u  
Label 3= login

#### **App-fr.properties (France language related)**

Label 1 = helio  
Label 2 = bind ding // use google translator  
Label 3 = loging

#### **App-tc.properties (Telugu language related)**

Label 1 = bagunara  
Label 2 = ela unaru  
Label 3 = \u4567\u6784\u5056\u4534 (pravesinchu)

- ❖ When End user supplies locale (mainly language) the application looks to take the locale specific properties file. If not available then application uses the base properties file to gather presentation logic label.
- ❖ Generally we keep English text based presentation logic labels in our base properties file.
- ❖ Struts 1.x and 2.x gives built in support to enable I18N on struts based webapplication.

### **Procedure to enable I18N on struts 1.x application**

**Step-1:**

Prepare multiple properties file having base file and locale specific files having presentation logic labels.

**Step-2:**

Configure only base properties file in struts configuration file.

**Step-3:**

Use <bean:message> tag to read the presentation logic labels from the activated properties file.

**Step-4:**

Deploy the struts application in webserver in regular manner.

**Step-5:**

Choose accept-language header through browser setting and send request to that struts application.

If accept-language header value is "fr" then it looks use app-fr.properties file if not available then it looks to use application properties file (base file)

**For example application on I18N refers the following application.**

**Customer.jsp (form page)**

```

1  <%@taglib uri="http://struts.apache.org/tags-html" prefix="html"%>
2  <%@taglib uri="http://struts.apache.org/tags-html" prefix="html"%>
3  <html:form action="Customer">
4  <bean:message key="label1"/><html:text property="cno"/><br>
5  <bean:message key="label2"/><html:text property="cname"/><br>
6  <bean:message key="label3"/><html:text property="bill"/><br>
7  <html:submit value="Submit"/>
8  </html:form>

```

In Customer.jsp the labels of text component will be gathered from one activated locale specific properties file.

**web.xml (Deployment Descriptor file)**

```

9  <?xml version="1.0" encoding="UTF-8"?>
10 <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3/EN"
11 "http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">
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 <servlet-mapping>
23 <servlet-name>action</servlet-name>
24 <url-pattern>*.do</url-pattern>

```

```

25 </servlet-mapping>

26 <welcome-file-list>
27   <welcome-file>Customer.jsp</welcome-file>
28 </welcome-file-list>
29 </web-app>

```

### struts-config.xml

```

30 <?xml version="1.0" encoding="UTF-8"?>
31 <!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration
    1.2//EN" "http://struts.apache.org/dtds/struts-config_1_2.dtd">
32 <struts-config>
33   <form-beans>
34     <form-bean name="CustForm" type="CustFormBean"></form-bean>
35   </form-beans>
36   <action-mappings>
37     <action path="/Customer" name="CustForm" scope="request" type="CustFormBean">
38       <forward name="success" path="/Success.jsp"></forward>
39     </action>
40   </action-mappings>
41   <message-resources parameter="App"/>
42 </struts-config>

```

Properties file configuration (Base file)

### CustFormBean.java (FormBean class)

```

43 import org.apache.struts.action.ActionForm;
44 public class CustFormBean extends ActionForm
45 {
46   public CustFormBean()
47   {
48     System.out.println("construtor is called:CustFormBean()");
49   }
50   private String cno,cname,bill;
51   public String getCno()
52   {
53     System.out.println("getCno():CustFormBean");
54     return cno;
55   }
56   public void setCno(String cno)
57   {
58     System.out.println("setCno(-):CustFormBean");
59     this.cno=cno;
60   }
61   public String getCname()
62   {
63     System.out.println("getCname():CustFormBean");
64     return cname;
65   }
66   public void setCname(String cname)
67   {
68     System.out.println("setCname(-):CustFormBean");
69     this.cname=cname;
70   }
71   public String getBill()
72   {
73     System.out.println("getBill():CustFormBean");
74     return bill;
75   }

```

```

76 public void setBill(String bill)
77 {
78     System.out.println("setBill(-):CustFormBean");
79     this.bill=bill;
80 }
81 }

```

### CustomerAction.java (Action Class)

```

82 import javax.servlet.http.*;
83 import org.apache.struts.action.*;
84 public class CustomerAction extends Action
85 {
86     public CustomerAction()
87     {
88         System.out.println("CustomerAction():CustomerAction");
89     }
90     public ActionForward execute(ActionMapping mapping,ActionForm form,HttpServletRequest
request,HttpServletResponse response)throws Exception
91     {
92         CustFormBean cfb=new CustFormBean();
93         String cno=cfb.getCno();
94         String cname=cfb.getCname();
95         String bill=cfb.getBill();
96         HttpSession session=request.getSession(true);
97         session.setAttribute("cno",cno);
98         session.setAttribute("cname",cname);
99         session.setAttribute("bill",bill);
100         return mapping.findForward("success");
101     }
102 }

```

Reading data from FormBean class object.

FormBean class is kept in session attribute  
resend to result page.

### Success.jsp (Result page)

```

103 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
104 <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
105 <html:html>
106 <center>
107 <bean:message key="label1"/>:<bean:write name="CustForm" property="cno"
scope="session"/><br>
108 <bean:message key="label2"/>:<bean:write name="CustForm" property="cname"
scope="session"/><br>
109 <bean:message key="label3"/>:<bean:write name="CustForm" property="bill"
scope="session"/><br>
110 </center>
111 </html:html>

```

### App.properties (Base file) (English language)

```

112 label1=Customer No
113 label2=Customer Name
114 label3=Customer Bill

```

### App\_de.properties (German language)

```

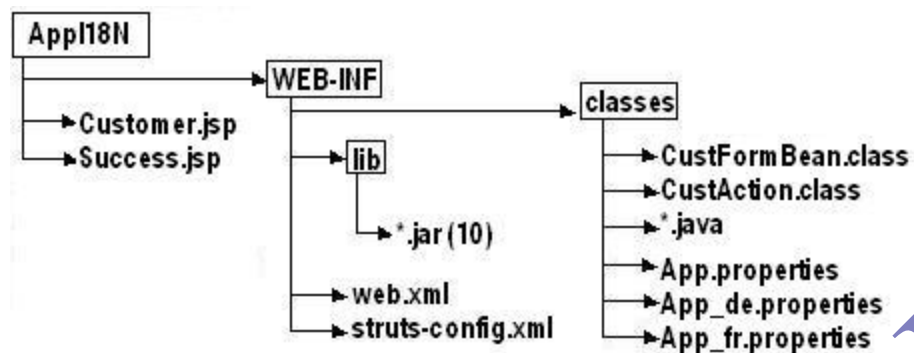
115 label1=JCustomer No
116 label2=JCustomer Name
117 label3=JCustomer Bill

```

### App\_fr.properties (French language)

118 label1=Customero Noo  
 119 label2=Customero Nameo  
 120 label3=Customero Billo

## Deployment Directory Structure



By just taking multiple locale specific properties files along with base file we can make any struts application working with I18N concept.

**To choose language and to set that value as accept-language request header for browser window generated request procedure in internet explorer.**

Tools menu -> Internet options -> languages -> add -> .....

## Procedure in Netscape

Edit menu -> preferences -> navigator -> languages -> add -> .....

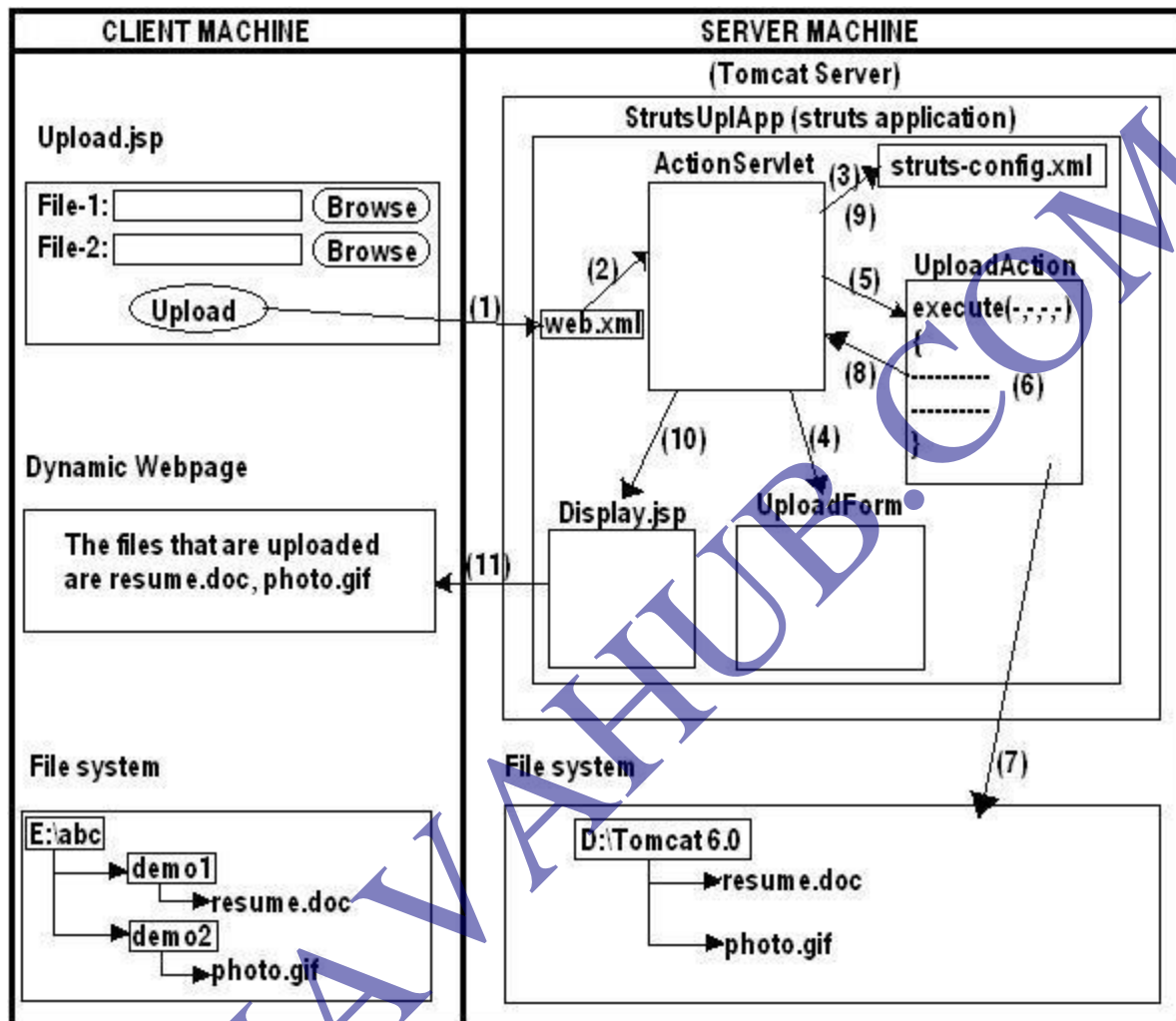
- ❖ I18N that is enabled on the application just deals with presentation logic of the application and it is no way responsible dealing with business logic and persistence logic of the application.
- ❖ Instead of using browser settings we can write java script code in the welcome page of struts application to set enduser chosen language as accept-language header value and to submit the request to struts application for this we need to call various methods on java script form object like submit of some other methods.
- ❖ The admin console applications that are given in glass fish, websphere server are developed as I18N enabled webapplications.
- ❖ Websphere admin console application is struts based webapplications.
- ❖ Glass fish admin console application is JSF based application.
- ❖ In web application development and execution process the process of selecting file from client machine file system and sending that file to server machine file system is called as file up loading process and reverse is called as file downloading process.
- ❖ While working with matrimony websites, jobportal website these file uploading and downloading operations are quite common.
- ❖ All the files and directory of a particular computer together is called as file systems.
- ❖ To select file form client machine file system the formpage uses exception control to prepare that use <html:file> tag is shown below.

## In Formpage

Select file <html:file properties ="f1"/>

In servlet.jsp environment we generally prefer working with third party api called java zoom api for file upload operations where as struts gives built in support to perform file upload operation.

To represent the uploaded file the programmer must take "org.apache.struts.upload.FormFile" type form bean property in FormBean class.



In the above diagram step-7 indicates struts Action class completes file uploading process by saving uploaded file on server in active file system.

By default struts application saves the uploaded files in the home directory of underlying web server.

To Specify different location to save the files that are uploaded use <controller> tag in struts configuration file as shown below.

**In struts-config.xml**

```
<controller.temp dir ="e:\\upload\\store">
```

**Procedure to develop struts page file uploading web application**

**Step-1**



Take form page having request method "post" and also having file uploading components.

### Step-2

Develop FormBean class having formbean properties of type org.apache.struts.upload Form file.

### Step-3

Develop I/O streams based logic in the execute (-,-,-) of struts Action class to complete file uploading operations.

### Step-4

Develop the remaining resources of struts application in a regular manner.

- ❖ Buffer is a temporary memory that holds data temporarily.
- ❖ The process of storing data in a buffer and using that data while transferring content from source resource to destination resource is called as buffering.

For above diagram based struts application (File Uploading) refer the following application.

#### upload.jsp (Form page)

```

1 <%@ page language="java" %>
2 <%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
3 <html:form action="upload.do" enctype="multipart/form-data">
4   Enter File1<html:file property="file1" /><br /><br />
5   Enter File2<html:file property="file2" /><br /><br />
6   <html:submit />
7 </html:form>

```

Refer line-62

Indicates that this form page can upload multiple types of files like text documents, audio, video and etc files.

File uploading components.

#### display.jsp (Result page)

```

8 <%@ page language="java" %>
9 <b>The File name1</b>:<%= request.getAttribute("fileName1") %> <br />
10 <b>The File name2</b>:<%= request.getAttribute("fileName2") %> <br />
11 <hr />

```

Reading the request attribute values from struts Action class to display the names of the uploaded

#### web.xml (Deployment Descriptor file)

```

12 <?xml version="1.0" encoding="ISO-8859-1"?>
13 <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2/EN"
14 "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
15 <web-app>
16 <!-- Action Servlet Configuration -->
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>debug</param-name>
26     <param-value>2</param-value>
27   </init-param>
28   <init-param>

```

```

29     <param-name>detail</param-name>
30     <param-value>2</param-value>
31 </init-param>
32 <init-param>
33     <param-name>validate</param-name>
34     <param-value>true</param-value>
35 </init-param>
36 <load-on-startup>1</load-on-startup>
37 </servlet>
38 <!-- Action Servlet Mapping -->
39 <servlet-mapping>
40     <servlet-name>action</servlet-name>
41     <url-pattern>*.do</url-pattern>
42 </servlet-mapping>
43 <!-- The Welcome File List -->
44 <welcome-file-list>
45     <welcome-file>upload.jsp</welcome-file>
46 </welcome-file-list>
47 <taglib>
48     <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
49     <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
50 </taglib>
51 </web-app>

```

#### struts-config.xml

```

52 <?xml version="1.0" encoding="ISO-8859-1" ?>
53 <!DOCTYPE struts-config PUBLIC
54     "-//Apache Software Foundation//DTD Struts Configuration 1.0//EN"
55     "http://jakarta.apache.org/struts/dtds/struts-config_1_0.dtd">
56 <struts-config>
57     <form-beans>
58         <form-bean name="uploadForm" type="UploadForm"/>
59     </form-beans>
60     <action-mappings>
61         <!-- Upload Action -->
62         <action path="/upload" type="UploadAction" name="uploadForm">
63             <forward name="display" path="/display.jsp"/>
64         </action>
65         <controller tempDir="D:\installed\Apache Software Foundation\Tomcat
66             5.5\webapps\StrutsUPLApp\"/>
67     </action-mappings>
68 </struts-config>

```

#### UploadForm.java

```

68 import org.apache.struts.upload.FormFile;
69 import org.apache.struts.action.ActionForm;
70 public class UploadForm extends ActionForm
71 {
72     protected FormFile file1,file2;
73     public FormFile getFile1()
74     {
75         return file1;
76     }
77     public void setFile1(FormFile file1)
78     {
79         this.file1 = file1;
80     }
81     public FormFile getFile2()

```

Special FormBean properties to hold the files that are uploaded.

```

82 {
83     return file2;
84 }
85 public void setFile2(FormFile file2)
86 {
87     this.file2 = file2;
88 }
89 }

```

### UploadAction.java

```

90 import java.io.InputStream;
91 import java.io.IOException;
92 import java.io.OutputStream;
93 import java.io.FileOutputStream;
94 import java.io.ByteArrayOutputStream;
95 import java.io.FileNotFoundException;
96 import javax.servlet.http.HttpServletRequest;
97 import javax.servlet.http.HttpServletResponse;
98 import org.apache.struts.upload.FormFile;
99 import org.apache.struts.action.Action;
100 import org.apache.struts.action.ActionForm;
101 import org.apache.struts.action.ActionMapping;
102 import org.apache.struts.action.ActionForward;
103 import org.apache.struts.action.ForwardingActionForward;

104 public class UploadAction extends Action
105 {
106     public ActionForward perform(ActionMapping mapping, ActionForm form,
107     HttpServletRequest request, HttpServletResponse response)
108     {
109         try
110         {
111             UploadForm uf = (UploadForm) form;
112
113             FormFile file1 = uf.getFile1();
114             FormFile file2 = uf.getFile2();
115
116             InputStream stream1 = file1.getInputStream();
117             InputStream stream2 = file2.getInputStream();
118
119             String fname1 = file1.getFileName();
120             String fname2 = file2.getFileName();
121
122             OutputStream bos1 = new FileOutputStream(fname1);
123             int bytesRead = 0;
124             byte[] buffer = new byte[1024];
125             while ((bytesRead = stream1.read(buffer, 0, 1024)) != -1)
126             {
127                 bos1.write(buffer, 0, bytesRead);
128             }
129             bos1.close();
130
131             OutputStream bos2 = new FileOutputStream(fname2);
132             bytesRead = 0;
133
134             while ((bytesRead = stream2.read(buffer, 0, 1024)) != -1)
135             {
136                 bos2.write(buffer, 0, bytesRead);
137             }
138             bos2.close();
139
140             return mapping.findForward("success");
141         }
142         catch (Exception e)
143         {
144             return mapping.findForward("error");
145         }
146     }
147 }

```

Deprecated method of execute(-,-,-) method.

Reading the uploaded files from FormBean class object in the form to FormFile class objects.

Gives InputStream objects representing the files that are uploaded.

Names of uploaded files.

Logic to move uploaded file content to server machine file system by using streams and by using buffering concepts.

```

137 while ((bytesRead = stream2.read(buffer, 0, 1024)) != -1)
138 {
139     bos2.write(buffer, 0, bytesRead);
140 }
141 bos2.close();
142 request.setAttribute("fileName1", fname1);
143 request.setAttribute("fileName2", fname2);
144 return mapping.findForward("display");
145 }

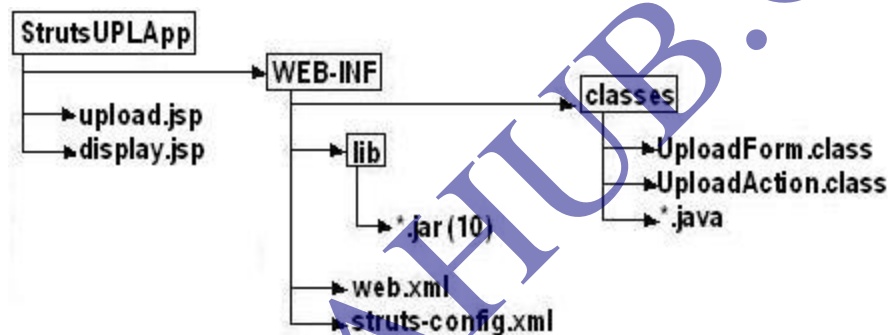
146 catch (FileNotFoundException fnfe)
147 {
148     return mapping.findForward("display");
149 }
150 catch (IOException ioe)
151 {
152     return mapping.findForward("display");
153 }
154 }
155 }

```

Completes file uploading for file2.

Request attributes holding the names of the uploaded files.

### Deployment Directory Structure



### Jar files required in the WEB-INF/lib folder.

- ✓ struts.jar
- ✓ struts-legacy.jar
- ✓ jakarta-oro.jar
- ✓ commons-beanutils.jar
- ✓ commons-collections.jar
- ✓ commons-digester.jar
- ✓ commons-fileupload.jar
- ✓ commons-lang.jar
- ✓ commons-logging.jar
- ✓ commons-validator.jar

Collect the above listed jar files from struts1.1 software extraction folder.

### Jar files required in the classpath

- ✓ servlet-api.jar
- ✓ struts-core-1.3.8.jar

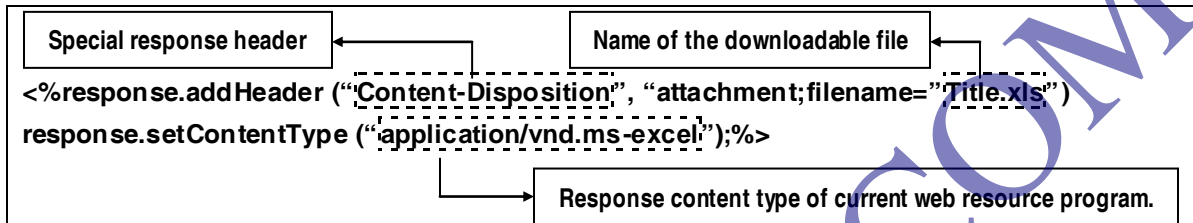
### File download in struts application

### Approach-1

- ✓ making the response of web application as downloadable file
- ✓ Here use special response header called content-disposition

### Approach-2

- ✓ making the resource of the web-application as downloadable file
- ✓ Here use the pre-defined downloaded Action class
- ❖ Write the following 2 lines of code in any java web resource program of regular web-application or struts application to make response of those web resource programs as downloadable file in servlet program /JSP program/struts Action class.



- ❖ The pre-defined download action class is an abstract class containing one abstract method called `getStreamInfo(-,-,-)`.
- ❖ This class also contains one inner interface, two inner classes.
- ❖ The Inner interface is `StreamInfo`
- ❖ The Inner classes are
  - `ResourceStreamInfo`
  - `FileStreamInfo`.

These two inner classes internally implements the inner interface called `StreamInfo`.

### Prototype of `getStreamInfo()` method

protected abstract `DownloadAction.StreamInfo getStreamInfo(-,-,-)`

When this method is implemented in our struts Action class that extends from `DownloadAction` class, it must return object of a class that implements `StreamInfo` interface pointing to the file that has to be downloaded (should be either `ResourceStreamInfo` class object or `FileStreamInfo` class object)

### Example application on downloading resource of the web-application as downloadable file (approach-2)

MyJsp.jsp

```
1 <%@taglib uri="http://struts.apache.org/tags-html" prefix="html"%>
```

```
2 <html:link action="/dpath">Download Word Doc</html:link>
```

Action path of struts Action class. (Refer line : 40)

abc.doc

File with any content

web.xml (Deployment Descriptor file)

```

3  <?xml version="1.0" encoding="UTF-8"?>
4  <web-app xmlns=http://java.sun.com/xml/ns/j2ee
5  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.4"
6  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web
7  app_2_4.xsd">
8    <!-- Action Servlet Configuration -->
9    <servlet>
10      <servlet-name>action</servlet-name>
11      <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
12      <init-param>
13        <param-name>config</param-name>
14        <param-value>/WEB-INF/struts-config.xml</param-value>
15      </init-param>
16      <init-param>
17        <param-name>debug</param-name>
18        <param-value>3</param-value>
19      </init-param>
20      <init-param>
21        <param-name>detail</param-name>
22        <param-value>3</param-value>
23      </init-param>
24      <load-on-startup>0</load-on-startup>
25    </servlet>
26    <!-- Action Servlet Mapping -->
27    <servlet-mapping>
28      <servlet-name>action</servlet-name>
29      <url-pattern>*.do</url-pattern>
30    </servlet-mapping>
31    <!-- The Welcome File List -->
32    <welcome-file-list>
33      <welcome-file>MyJsp.jsp</welcome-file>
34    </welcome-file-list>
35  </web-app>

```

#### struts-config.xml

```

36 <?xml version="1.0" encoding="UTF-8"?>
37 <!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration
38 1.2//EN" "http://struts.apache.org/dtds/struts-config_1_2.dtd">
39 <struts-config>
40   <action-mappings>
41     <action path="/dpath" type="MyAction" parameter="/abc.doc" />
42   </action-mappings>
43 </struts-config>

```

#### MyAction.java

```

43 import java.io.*;
44 import javax.servlet.*;
45 import javax.servlet.http.*;
46 import org.apache.struts.action.*;
47 import org.apache.struts.actions.*;
48 public class MyAction extends DownloadAction
49 {
50   protected StreamInfo getStreamInfo(ActionMapping mapping, ActionForm
51   form,HttpServletRequest request,HttpServletResponse response)throws Exception
52   {
53     // get the File Name from parameter attribute
54     String fileName=mapping.getParameter();
55     // Set the content disposition

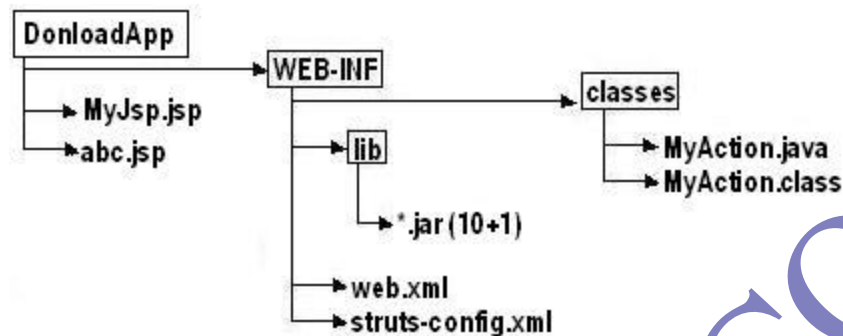
```

```

55 response.setHeader("Content-disposition","attachment;filename="+fileName);
56 // set the content type
57 String contentType="application/msword";
58 ServletContext sc=servlet.getServletContext();
59 return new ResourceStreamInfo(contentType,sc,fileName);
60 }
61 }

```

### Deployment Directory Structure



#### Jar files in the classpath

- ✓ Servlet-api.jar
- ✓ Struts-core-1.3.8.jar
- ✓ Struts-extras-1.3.8.jar

#### Jar files in WEB-INF/sub folder

- ✓ 10 → regular struts jar file
- ✓ 1 → struts-extras-1.3.8.jar

- ❖ We cannot send ResultSet object over the network because it is not a serializable object. Java object becomes serializable only when class of that object implements java.io.Serializable interface.
- ❖ To solve the problem of ResultSet object use one of these two solutions.

#### Solution 1

Stop working with ResultSet object and start working with RowSet object.

**Note:** All Rowset objects are serializable objects by default.

**Limitations:** very few JDBC drivers support RowSet.

#### Solution 2

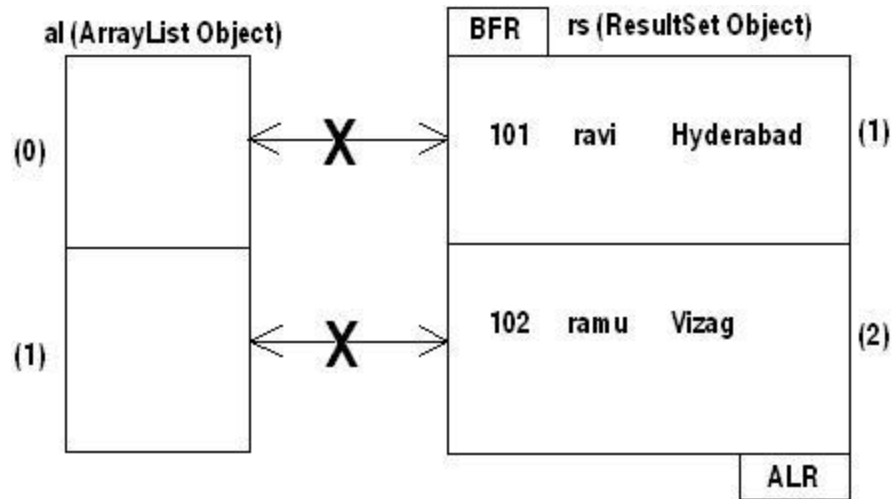
Copy ResultSet object to collection Framework data structure and send that data structure over the network.

**Note:**

All collection Framework data Structures are serializable objects (like Array List) by default.

Solution-2 is recommended.

- ❖ Since each element of Array List allows only one object and each record of ResultSet object may contain multiple primitives, objects. So, we cannot move each record of ResultSet object directly to each element of ArrayList object. (or to any other collection Framework datastructure)



- ❖ To solve the above implementation problem store each record values to a user defines java class object and add that object to ArrayList elements as shown below. In this process this user defined java class is called as DTO class or VO class.

### Sample Code

#### V.O / D.T.O class

```
public class StudentBean implements Seriliazable
{
    int sno;
    String sname;
    String sadd;
    // write getxxx() and setxxx() methods
    public void setXxx(-)
    {
        .....
        .....
    }
}
```

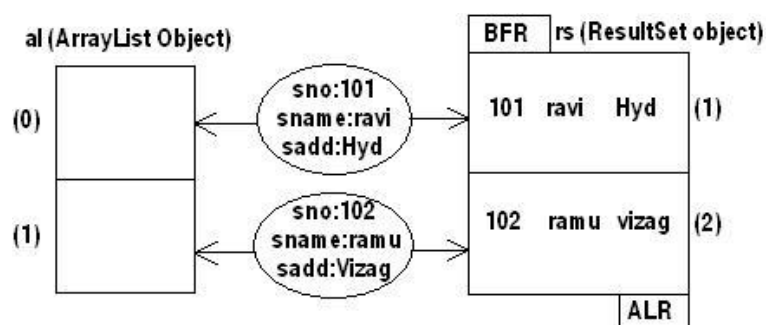
#### Logic to copy ResultSet object data to ArrayList object through D.T.O / V.O class

```
Result Set rs=st.execute Query ("select * from student");
```

```
Array List al = new ArrayList();
```

```
while (rs.next())
```

```
{
    st.setNo(rs.getInt(1));
    st.setSname(rs.getString(2));
    st.setSadd(rs.getString(3));
    al.add(st);
}
```





**Q) Can you tell me how exactly you have taken the support of collection framework data structure in your projects?**

1. To send ResultSet Object data over the network it transfers its data to collection framework data structure like (ArrayList).
2. To make JDBC code as flexible code, we gather JDBC details from properties file. In that project, java.util.properties class will be utilized.
3. To maintain JNDI properties we need Map data structures.
4. To maintain mail properties in Javamail program. We use Java.util.properties class.
5. To maintain multiple values of request in a session as single session attribute value, to store that data in collection framework data structure and we add that data as session Attribute value.

**Q) Where exactly you have used JavaBean in your projects.**

1. As Model layer resource to maintain business logic and persistence logic in MVC-1, MVC-2 architecture based projects.
  2. As DTO or VO class while transferring JDBC ResultSet object data to collection framework data structure.
  3. As FormBean class in struts applications.
  4. As persistent class in hibernate applications.
  5. As SpringBean in spring applications.
- ❖ While developing huge large scale struts application instead of making all programmers dealing with single module and single struts configuration file which may give naming classes.
  - ❖ It is recommended to create multiple logical modules in the struts application and work with multiple struts configuration files on one per module basis. This avoids naming classes on struts configuration file.
  - ❖ By default every struts application contains an default module and also allows us to create multiple explicit modules with names. If two explicit modules are created in struts application means there are one default module (nameless) and two Named module (module with names)

**In web.xml file**

```
<web-app>
<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>
<inti-param>
<!--mod1 module related struts configuration file configuration-->
<p-n> config/mod1</p-n>
<p-v>/WEB-INF/struts-config-mod1.xml</p-v>
</init-param>
<!--mod2 module related struts configuration file configuration-->
<init-param>
<p-n>config /mod2</p-n>
<p-v> /WEB-INF/struts-config-mod2.xml</p-v>
</init-param>
```

```

<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
<load-a-startup> 1 <load-or-startup>
</web-app>

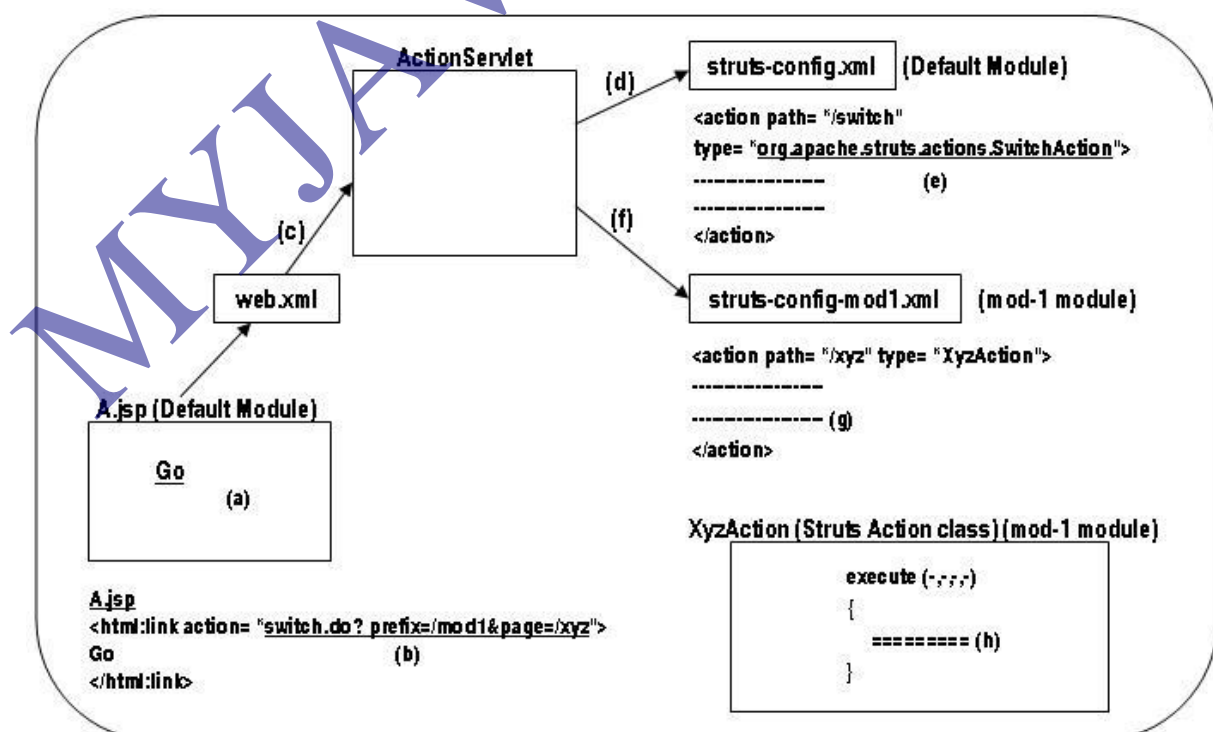
```

- ❖ Any name can be given to any module.
- ❖ Any file name.xml can be configured as struts configuration file for any module.
- ❖ Each module is not separate struts application it is a logical partition in the struts application.
- ❖ In each module created in the struts application that module related FormBean classes, Action Classes, Form pages, result pages will be maintain.
- ❖ To get the communication between various web resource programs of two different modules of a struts application we need 'SwitchAction' class support.

### To following possibilities are there for communication

1. Named Module resource – Default Module resource.
  2. Named Module resource – Named Module resource.
  3. Default Module resource – Name Module resource.
- ❖ While working in SwitchAction class, it must be configured in source module related struts configuration file and the request coming to the 'SwitchAction' class through ActionServlet must have two request parameters in the request.
    1. Prefix -> holding name of the destination module.
    2. Page -> holding identification name of destination web resource program in destination module.

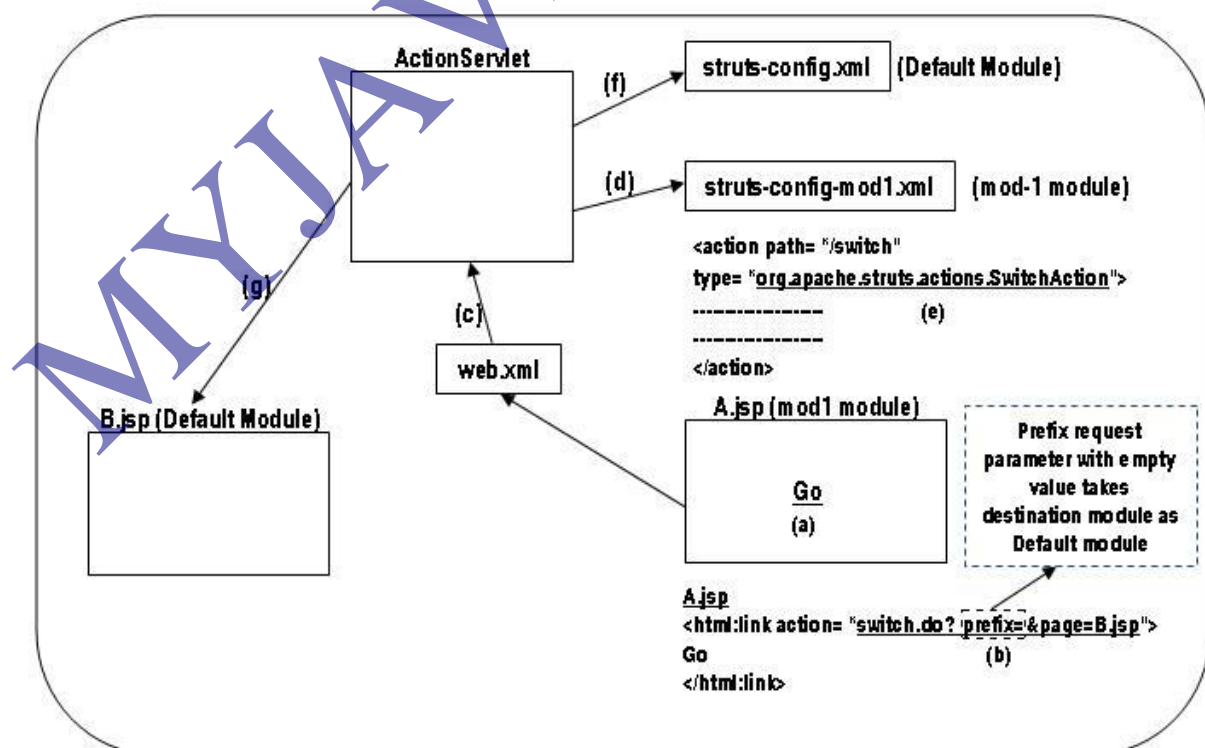
### Transferring control from default module resource to named module resource (mod-1)



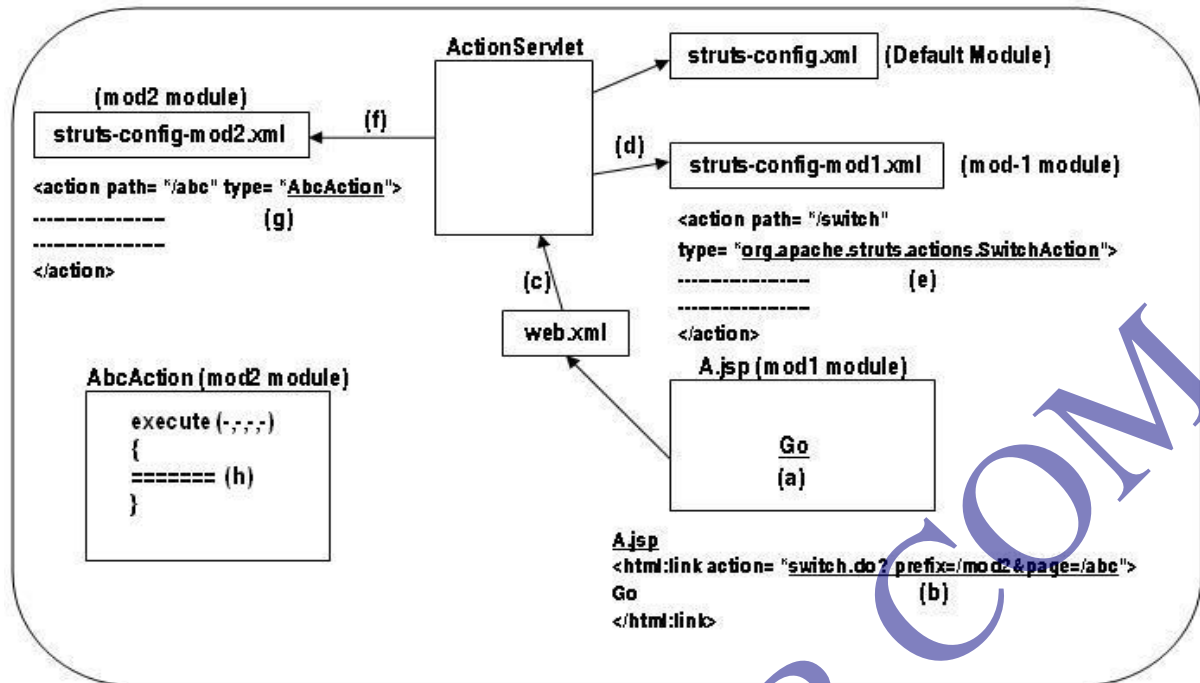
### With respect to above diagram

- End user clicks on the hyperlink of A.JSP belongs to default module.
- Based on the URL placed in action attribute of `<html:link>`, the request URL will be generated.
- ActionServlet traps and takes the request based on its configuration done in struts configuration file.
- ActionServlet uses default module related struts configuration file entries. Because the request has come from default module.
- ActionServlet request to the SwitchAction class configured in default module related struts configuration file. This SwitchAction reads 'prefix', 'page' request parameter values.
- SwitchAction makes ActionServlet to locate mod1 module related struts configuration file i.e, s-(-mod1.xml) based on 'prefix' request parameter value (mod1).
- ActionServlet looks for an Actionclass configure in s-c-mod1.xml having action path "/xyz" based on 'page' request parameter value (i.e , xyz) gets (xyz Action Class configure).
- ActionServlet calls the execute method of xyz Action class (mod1- module)

### Transferring control between resource of named module (mod1) and the resource of default module



Transferring control between resource of one named module and another named module.



Example application on multiple modules based struts application work with multiple modules refer the following application.

In this application 'index.jsp' (Default module) wants to navigate the control to 'Student.jsp' (Student module) and 'Faculty.jsp' (Faculty module) by using SwitchAction class through hyperlinks.

index.jsp (Belongs to default module)

```

1 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
2 <html:html>
3 <html:link action="/switch?prefix=/Student&page=/Student.jsp">Student Search</html:link>
4 <html:link action="/switch?prefix=/Faculty&page=/Faculty.jsp">Faculty Search</html:link><br>
5 </html:html>

```

Annotations for the code above:

- Line 3: `refer line 15 of struts-config.xml` points to the `switch` action definition.
- Line 3: `Module name` points to `/Student` and `Resource name` points to `/Student.jsp`.

web.xml (Deployment Descriptor File)

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
3 "http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">
4
5 <web-app>
6 <servlet>
7 <servlet-name>action</servlet-name>
8 <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
9
10 <init-param>
11 <param-name>config</param-name>
12 <param-value>/WEB-INF/struts-config.xml</param-value>
13 </init-param>
14
15 <init-param>
16 <param-name>config/Student</param-name>
17 <param-value>/WEB-INF/struts-config-student.xml</param-value>
18 </init-param>
19

```

Annotations for the code above:

- Lines 10-13: **Default module related struts-config.xml file configuration.**
- Lines 15-18: **Student module related struts-config.xml file configuration.**

```

20 <init-param>
21 <param-name>config/Faculty</param-name>
22 <param-value>/WEB-INF/struts-config-faculty.xml</param-value>
23 </init-param>
24 <load-on-startup>1</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>index.jsp</welcome-file>
32 </welcome-file-list>
33 </web-app>

```

Faculty module related  
struts-config.xml file  
configuration.

### struts-config.xml (Default module configuration file)

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration
  1.3//EN" "http://jakarta.apache.org/struts/dtds/struts-config_1_3.dtd">
3 <struts-config>
4 <action-mappings>
5 <action path="/switch" type="org.apache.struts.actions.SwitchAction"/>
6 </action-mappings>
7 </struts-config>

```

In source module related struts configuration file  
SwitchAction is configured.

### struts-config-student.xml (Student module configuration file)

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration
  1.3//EN" "http://jakarta.apache.org/struts/dtds/struts-config_1_3.dtd">
3 <struts-config>
4 <form-beans>
5 <form-bean name="StudentForm" type="StudentFormBean"/>
6 </form-beans>
7 <action-mappings>
8 <action name="studentform" path="/student" type="StudentAction">
9 <forward name="succ" path="/Student.jsp"/>
10 <forward name="fail" path="/fail.jsp"/>
11 </action>
12 </action-mappings>
13 </struts-config>

```

### struts-config-faculty (Faculty module configuration file)

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration
  1.3//EN" "http://jakarta.apache.org/struts/dtds/struts-config_1_3.dtd">
3 <struts-config>
4 <form-beans>
5 <form-bean name="FacultyForm" type="FacultyFormBean"/>
6 </form-beans>
7 <action-mappings>
8 <action name="FacultyForm" path="/faculty" type="FacultyAction">
9 <forward name="succ" path="/Faculty.jsp"/>
10 <forward name="fail" path="/Fail.jsp"/>
11 </action>
12 </action-mappings>
13 </struts-config>

```

### Student.jsp (Form page of student module)

```

1  <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
2  <%@ page import="java.util.*"%>
3  <%@ taglib uri="http://struts.apache.org/tags-logic" prefix="logic" %>
4  <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
5  <center><font size=6>Search Criteria of Student</font></center>
6  <html:form action="student">
7  <table>
8  <tr>
9  <td>StudentID</td>
10 <td><html:text property="id" /></td>
11 <td><html:submit value="Send" /></td>
12 </tr>
13 </table>
14 </html:form>
15 <table border=1>
16 <%
17 ArrayList as=(ArrayList)session.getAttribute("res");
18 %>
19 <logic:notEmpty name="res" scope="session">
20 <tr>
21 <td>ID</td>
22 <td>Name</td>
23 <td>Course</td>
24 <td>Qualification</td>
25 </tr>
26 <logic:iterate id="id1" collection="<%= as %>">
27 <tr>
28 <td><bean:write name="id1" property="id" /></td>
29 <td><bean:write name="id1" property="name" /></td>
30 <td><bean:write name="id1" property="course" /></td>
31 <td><bean:write name="id1" property="qualification" /></td>
32 </tr>
33 </logic:iterate>
34 </logic:notEmpty>
35 </table>

```

action path of StudentAction class  
refer line:8 of struts-config-student.xml

Refer line:17

Points to 1 element of 'as' at a time.

Logic to display ArrayList  
object data as html table  
content.

Loop iterating through  
elements of ArrayList  
object.

Calls getXxx() methods  
of properties on  
student class object.

### Alternate code for the lines 15-35 of Student.jsp

```

1  <%ArrayList al=(ArrayList)session.getAttribute("res");
2  If(al!=null)
3  {
4  <table>
5  <tr>
6  <th>id</th>
7  <th>name</th>
8  <th>course</th>
9  <th>qualification</th>
10 <tr>
11 <% for(int i=0;i<al.size();++i)
12 {
13 Student st=(Student)al.get(i);%>
14 <tr>
15 <td><%=st.getId()%></td>
16 <td><%=st.getName()%></td>
17 <td><%=st.getCourse()%></td>
18 <td><%=st.getQualification()%></td>
19 </tr>
20 <%}%>
21 </table>
22 <%}%>

```

**Faculty.jsp (Form page of faculty module)**

```

1  <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
2  <%@ page import="java.util.*"%>
3  <%@ taglib uri="http://struts.apache.org/tags-logic" prefix="logic" %>
4  <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
5  <center><font size=6>Search Criteria of Faculty</font></center>
6  <html:form action="faculty">
7  <table>
8  <tr>
9  <td>FacultyID</td>
10 <td><html:text property="id" /></td>
11 <td><html:submit value="Send" /></td>
12 </tr>
13 </table>
14 </html:form>
15 <table border=1>
16 <%
17 ArrayList as=(ArrayList)session.getAttribute("resf");
18 %>
19 <logic:notEmpty name="resf" scope="session">
20 <tr>
21 <td>ID</td>
22 <td>Name</td>
23 <td>Subject</td>
24 <td>Qualifacation</td>
25 </tr>
26 <logic:iterate id="id1" collection="<%= as%>">
27 <tr>
28 <td><bean:write name="id1" property="id" /></td>
29 <td><bean:write name="id1" property="name" /></td>
30 <td><bean:write name="id1" property="subject" /></td>
31 <td><bean:write name="id1" property="qualification" /></td>
32 </tr>
33 </logic:iterate>
34 </logic:notEmpty>
35 </table>

```

**FacultyFormBean.java (Faculty module)**

```

1  import javax.servlet.http.HttpServletRequest;
2  import org.apache.struts.action.ActionError;
3  import org.apache.struts.action.ActionErrors;
4  import org.apache.struts.action.ActionForm;
5  import org.apache.struts.action.ActionMapping;
6  public class FacultyFormBean extends org.apache.struts.action.ActionForm
7  {
8  public FacultyFormBean()
9  {
10 }
11 private String id;
12 public void setid(String id)
13 {
14 this.id=id;
15 }
16 public String getid()
17 {
18 return id;
19 }
20 }

```



**StudentFormBean.java (Student module)**

```

1  import javax.servlet.http.HttpServletRequest;
2  import org.apache.struts.action.ActionError;
3  import org.apache.struts.action.ActionErrors;
4  import org.apache.struts.action.ActionForm;
5  import org.apache.struts.action.ActionMapping;
6  public class StudentFormBean extends org.apache.struts.action.ActionForm
7  {
8      public StudentFormBean()
9      {
10     }
11     private String id;
12     public void setid(String id)
13     {
14         this.id=id;
15     }
16     public String getid()
17     {
18         return id;
19     }
20 }

```

**Student.java (D.T.O class of Student module)**

```

1  package bean;
2  public class Student
3  {
4      public Student()
5      {
6      }
7      String id,name,course,qualification;
8      public void setid(String id)
9      {
10         this.id=id;
11     }
12     public String getid()
13     {
14         return this.id;
15     }
16     public void setname(String name)
17     {
18         this.name=name;
19     }
20     public String getname()
21     {
22         return this.name;
23     }
24     public void setcourse(String course)
25     {
26         this.course=course;
27     }
28     public String getcourse()
29     {
30         return this.course;
31     }
32     public void setqualification(String qualification)
33     {
34         this.qualification=qualification;
35     }
36     public String getqualification()
37     {
38         return this.qualification;
39     }
40 }

```



**Faculty.java (D.T.O class of Faculty module)**

```

1  package bean;
2  public class Faculty
3  {
4      public Faculty()
5      {
6          String id,name,subject,qualification;
7          public void setid(String id)
8          {
9              this.id=id;
10         }
11         public String getid()
12         {
13             return this.id;
14         }
15         public void setname(String name)
16         {
17             this.name=name;
18         }
19         public String getname()
20         {
21             return this.name;
22         }
23         public void setsubject(String subject)
24         {
25             this.subject=subject;
26         }
27         public String getsubject()
28         {
29             return this.subject;
30         }
31         public void setqualification(String qualification)
32         {
33             this.qualification=qualification;
34         }
35         public String getqualification()
36         {
37             return this.qualification;
38         }
39     }

```

**StudentAction.java (Action class of Student module)**

```

1  import java.io.*;
2  import javax.servlet.RequestDispatcher;
3  import javax.servlet.ServletException;
4  import javax.servlet.http.HttpServletRequest;
5  import javax.servlet.http.HttpServletResponse;
6  import javax.servlet.http.HttpSession;
7  import org.apache.struts.action.Action;
8  import org.apache.struts.action.ActionError;
9  import org.apache.struts.action.ActionErrors;
10 import org.apache.struts.action.ActionForm;
11 import org.apache.struts.action.ActionForward;
12 import org.apache.struts.action.ActionMapping;
13 import org.apache.struts.action.ActionServlet;
14 import org.apache.struts.action.DynaActionForm;

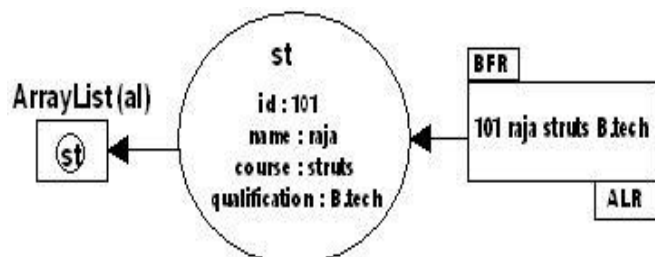
```

```

15 import org.apache.struts.util.MessageResources;
16 import java.sql.*;
17 import java.util.ArrayList;
18
19 public class StudentAction extends org.apache.struts.action.Action
20 {
21     private static final String FORWARD_fail = "fail";
22     private static final String FORWARD_succ = "succ";
23     public ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest
        request, HttpServletResponse response) throws Exception
24     {
25         try
26         {
27             ArrayList al=new ArrayList();
28             StudentFormBean sfb=(StudentFormBean)form;
29             String id=sfb.getId();
30             Class.forName("oracle.jdbc.driver.OracleDriver");
31             Connection
                con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","scott","tiger");
32             PreparedStatement ps=con.prepareStatement("select * from students where id = ?");
33             ps.setString(1,id);
34             ResultSet rs=ps.executeQuery();
35             int count=0;
36             while(rs.next())
37             {
38                 count++;
39                 bean.Student st=new bean.Student();
40                 st.setid(rs.getString(1));
41                 st.setname(rs.getString(2));
42                 st.setcourse(rs.getString(3));
43                 st.setqualification(rs.getString(4));
44                 al.add(st);
45             }
46             if(count==0)
47             return mapping.findForward("fail");
48             HttpSession session=request.getSession(true);
49             session.setAttribute("res",al);
50         }catch(Exception e)
51         {
52             e.printStackTrace();
53         }
54         return mapping.findForward("succ");
55     }
56 }

```

Logic to transfer ResultSet Object data to ArrayList with the support of D.T.O class.



Keeping the result in the session attribute

**FacultyAction.java (Action class of Faculty module)**

```

1  import java.io.*;
2  import javax.servlet.RequestDispatcher;
3  import javax.servlet.ServletException;
4  import javax.servlet.http.HttpServletRequest;
5  import javax.servlet.http.HttpServletResponse;
6  import javax.servlet.http.HttpSession;
7  import org.apache.struts.action.Action;
8  import org.apache.struts.action.ActionError;
9  import org.apache.struts.action.ActionErrors;
10 import org.apache.struts.action.ActionForm;
11 import org.apache.struts.action.ActionForward;
12 import org.apache.struts.action.ActionMapping;
13 import org.apache.struts.action.ActionServlet;
14 import org.apache.struts.action.DynaActionForm;
15 import org.apache.struts.util.MessageResources;
16 import java.sql.*;
17 import java.util.*;
18 public class FacultyAction extends org.apache.struts.action.Action
19 {
20     public FacultyAction()
21     {
22     }
23     public ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest
24     request, HttpServletResponse response) throws Exception
25     {
26     {
27         ArrayList al=new ArrayList();
28         FacultyFormBean sfb=(FacultyFormBean)form;
29         String id=sfb.getId();
30         Class.forName("oracle.jdbc.driver.OracleDriver");
31         Connection con
32         =DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","scott","tiger");
33         PreparedStatement ps=con.prepareStatement("select * from faculty where fid = ?");
34         ps.setString(1,id);
35         ResultSet rs=ps.executeQuery();
36         int count=0;
37         while(rs.next())
38         {
39             count=1;
40             bean.Faculty st=new bean.Faculty();
41             st.setid(rs.getString(1));
42             st.setname(rs.getString(2));
43             st.setsubject(rs.getString(3));
44             st.setqualification(rs.getString(4));
45             al.add(st);
46         }
47         if(count==0)
48             return mapping.findForward("fail");
49         HttpSession session=request.getSession(true);
50         session.setAttribute("resf",al);
51     }catch(Exception e){
52         e.printStackTrace();}
53     return mapping.findForward("succ");
54 }

```

**Fail.jsp (Result page of Faculty module)**

```

1  <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
2  <center><font size=6>No Data Found</font></center>
3  <%session.removeAttribute("resf");%>

```

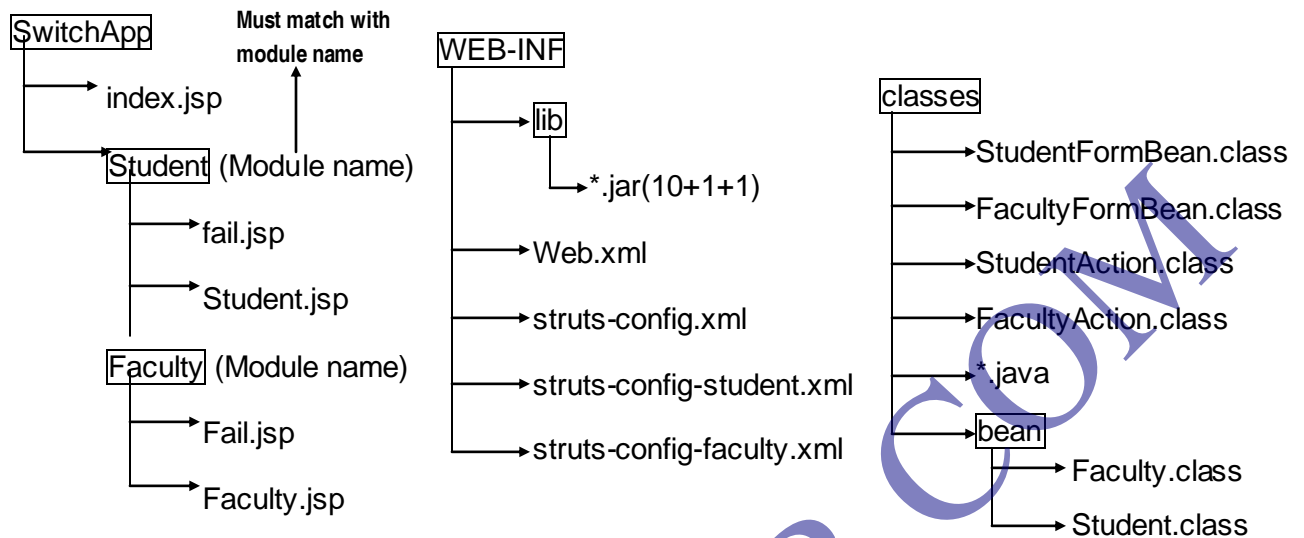
## fail.jsp (Result page of Student module)

```

1 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
2 <center><font size=6>No Data Found</font></center>
3 <%session.removeAttribute("res");%>

```

## Deployment Directory Structure



## Jar files in classpath

struts-core-1.3.8.jar

servlet-api.jar

## Jar files in WEB-INF/lib folder

10→struts.jar file

1→ojdbc14.jar file

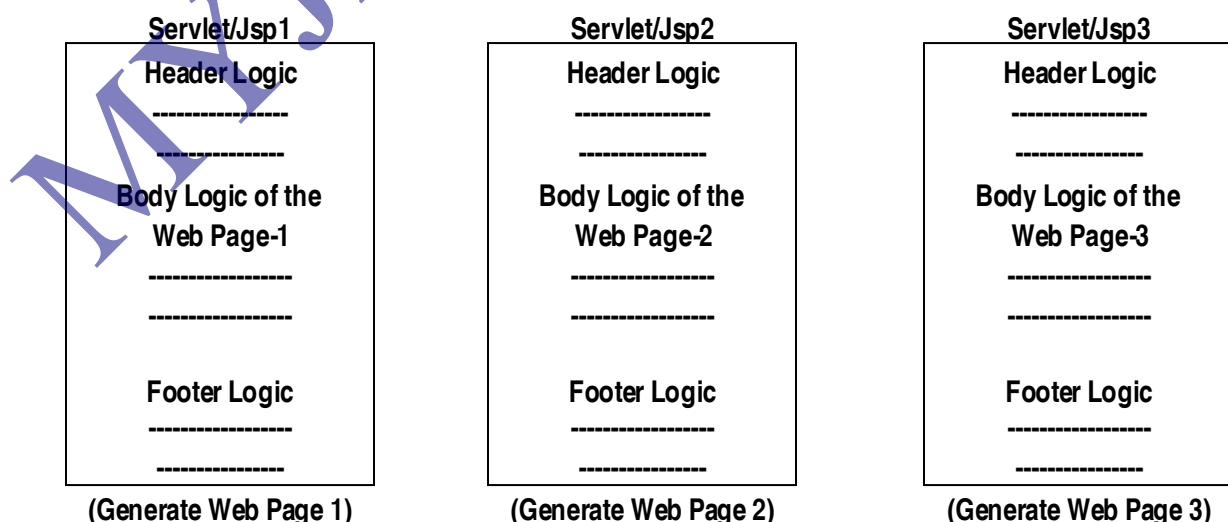
1→struts-extras-1.3.8.jar file

## Note

Irrespective of network availability always move ResultSet object data from one layer to another layer by copying the data to collection framework data structure.

## Understanding composite view Design Pattern:

### Problem:

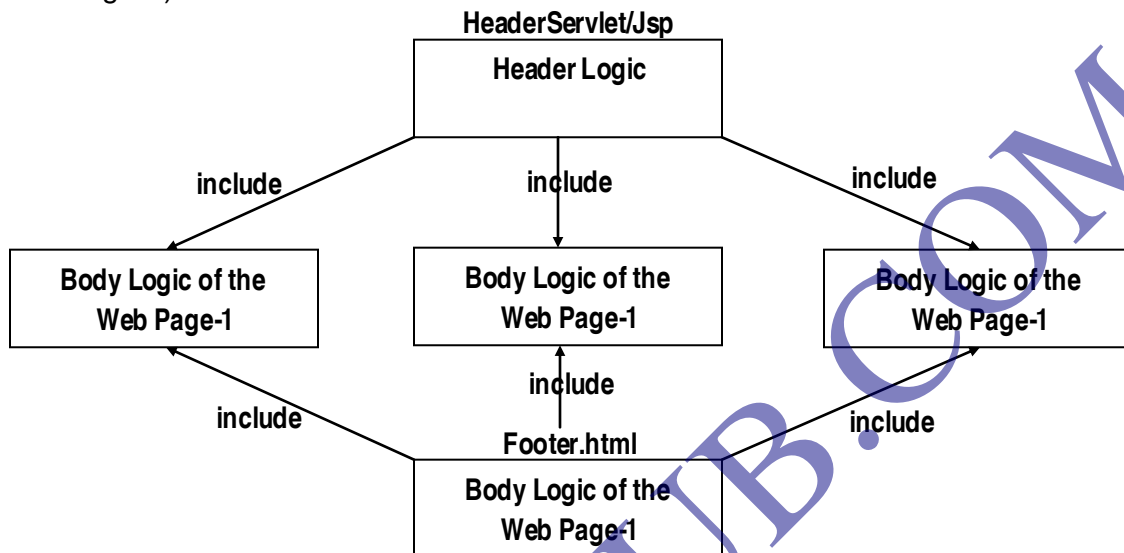


In the above diagram each web resource page having header content, body content and footer content.

- Generates all web pages of websites contains same header and footer contents.
- According to that in the above diagram same header and footer logics are return in every web resource program that means they are not taken as reusable logics.

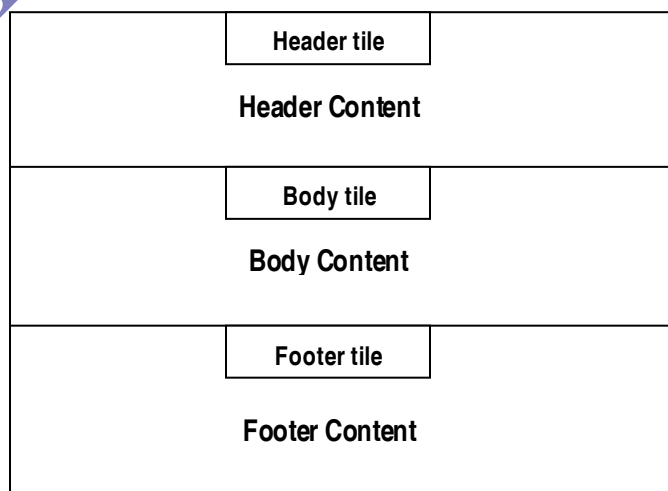
For this work with composite view design pattern as shown below.

**Solution:** work with composite-view design pattern (makes header and footer logics are reusable logics.)



**Note:** The above solution is violating MVC-2 architecture.

- The above diagram talks about classic web application development. Where MVC2 principle is violated because two JSP programs are two servlet programs of web application are interacting with each other directly.
- In struts environment Tiles plugin / Tiles framework is given to implement **view design pattern** by satisfying MVC2 principles and also allows the programmer to design the web pages by having layout page towards appearance of webpage will reflect to all the web pages of web application.
- While working with Tiles framework first layout page will be created having designing of the webpage. Based on that layout page the remaining pages will be constructed having content.
- A Tile is a logical partition (or) portion of a web page representing the content.



- Struts 1.x software gives struts\_home\lib\struts-tiles-1.3.8.jar file representing tiles plugin. This jar file contain struts-tiles.tld file representing tiles JSP tag library.

### Html tag library

Gives jsp tags as alternate tags for traditional html tags. Useful for form page designing.

### Bean tag library

Useful to with attributes, form bean properties and etc useful to read messages from properties file.

### Logic tag library

Gives tags for conditional operation.

### Nested tag library

Extension to logic tag library to nested condition based operations.

### Tiles tag library

Useful to design files of tiles in layout page while working with tiles plugin (or) tiles framework.

- Tiles framework deals with only presentation logic of struts application.
- Tiles framework supplies one plugin class. (org.apache.struts.tiles.TilesPlugin)

### Procedure to work with Tiles Framework in our struts Application:

**Step-1:** Configure the PlugIn class of tiles concept in struts configure file.

In s-c.xml:

```
<plug-in className="org.apache.struts.tiles.TilesPlugin">
  <set-property property="definitions-config" value="|WEB-INF | tiles -dets.xml"|>
  <set-property property="Module Aware" value="true"|>
</plug-in>
```

**Note:** Any <FileName>.xml can be taken as Tiledefinition configuration file.

**Step-2:** Design layout page of the application having tiles.

```
<%@ taglib url="http://struts.apache.org/tags-tiles" prefix="tiles"%>
<html>
  <body>
    <tiles:insert attribute="header"/>→ tile 1 (header tile)
    <tiles:insert attribute="body"/>→ tile 2 (body tile)
    <tiles:insert attribute="footer"/>→ tile 3 (footer tile)
  </body>
</html>
```

**Step-3:** Count number of web pages that are going to come in your struts application.

**Step-4:** Develop the definition configuration file having multiple tile definition on one per webpage basis.

**Note:** Every Tile definition will be developed based on layout page having logical name.

**tiles-defs.xml**

```

<!DOCTYPE tiles-definitions PUBLIC
    "-//Apache Software Foundation//DTD Tiles Configuration//EN"
    "http://jakarta.apache.org/struts/dtds/tiles-config_1_1.dtd">
<tiles-definitions>
<definition name="MyTiles" path="/pages/MyTilesHome.jsp">
    <put name="header" value="/pages/BaseHeader.jsp" />
    <put name="left-content" value="/pages/LeftNavigation.jsp"/>
    <put name="main-content" value="/pages/welcome.jsp"/>
    <put name="footer" value="/pages/BaseFooter.jsp" />
</definition>
<definition name="tile.main.success" extends="MyTiles">
    <put name="main-content" value="/pages/success.jsp" />
</definition>

<definition name="tile.main.failure" extends="MyTiles">
    <put name="main-content" value="/pages/failure.jsp" />
</definition>
<definition name="insertTile" extends="MyTiles">
    <put name="main-content" value="/pages/insert.jsp" />
</definition>
<definition name="deleteTile" extends="MyTiles">
    <put name="main-content" value="/pages/delete.jsp" />
</definition>
<definition name="updateTile" extends="MyTiles">
    <put name="main-content" value="/pages/update.jsp" />
</definition>
</tiles-definitions>

```

It is recommended to write tile definition based on Inheritance as show below.

```

<?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>
    <definition name= "aDef" path= "/baseDef.jsp">
        <put name= "body" value= "/aBody.jsp"/>
    </definition>
    <definition name= "bDef" path= "/baseDef.jsp">
        <put name= "body" value= "/bBody.jsp"/>
    </definition>
    <definition name= "cDef" path= "/baseDef.jsp">
        <put name= "body" value= "/cBody.jsp"/>
    </definition>
</component-definitions>

```

**Step-5:** Develop struts Action class returning ActionForward object having logical name from the execute (-,-,-) method.

```

public class MyAction 1/2/...../n extends Action
{
    public ActionForward execute(-,-,-)
    {
        .....
        .....
        .....
        If(cond)
            return mapping.find Forward ("Success");
        elseif (cond)
            return mapping.find Forward ("Failure");
    }
}

```

Configure above Struts Action class in struts configure file along with ActionForward configurations specifying tile definition names as shown below.

In s-c.xml:

```

<action-mappings>
    <action path="/xyz" type="MyAction" name="rf">
        <forward name="success" path="adef"/>
        <forward name="failure" path="bdef"/>
        <forward name="home" path="cdef"/>
    </action>
</action-mappings>

```

### Different Layouts of Layout page

#### Classic Layout

Header Tile
Main-Content Tile
Footer Tile

#### Two Column Layout

Header Tile	
Left Content Type	Main-Content Tile
Footer Tile	

#### Three Column Layout

Header Tile		
Left Content Type	Main-Content Tile	Right Content Type
Footer Tile		



**Note:** The layout page of tiles framework based will be designed based on the above set layouts.

- The java class that separates persistence logic from other logics of the application to make persistence logic as flexible logic to modify is called as D.A.O (Data Access Object).
- The java class with full of table code (or) Hibernate code performing all the persistence operations of the application is called as D.A.O
- In struts application every Action class can have its own D.A.O class are multiple Action classes can use single D.A.O classes.

**For example application on tiles framework having utilization of D.A.O form refer the following application.**

**index.jsp (welcome page) (generates implicit request)**

```
1 <%
2 response.sendRedirect("mytiles.do");
3 %>
```

**web.xml (Deployment Descriptor file)**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">
4 <web-app>
5 <!-- Action Servlet Configuration -->
6 <servlet>
7 <servlet-name>action</servlet-name>
8 <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
9 <init-param>
10 <param-name>config</param-name>
11 <param-value>/WEB-INF/struts-config.xml</param-value>
12 </init-param>
13 <load-on-startup>1</load-on-startup>
14 </servlet>

15 <!-- Action Servlet Mapping -->
16 <servlet-mapping>
17 <servlet-name>action</servlet-name>
18 <url-pattern>*.do</url-pattern>
19 </servlet-mapping>

20 <!-- The Usual Welcome File List -->
21 <welcome-file-list>
22 <welcome-file>index.jsp</welcome-file>
23 </welcome-file-list>
24 </web-app>
```

**struts-config.xml (struts configuration file)**

```
25 <?xml version="1.0" encoding="UTF-8"?>
26 <!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration
1.1//EN" "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
27 <struts-config>
28 <form-beans>
```

```

29 <form-bean name="dataForm" type="org.apache.struts.action.DynaActionForm">
30 <form-property name="eid" type="java.lang.String"/>
31 <form-property name="name" type="java.lang.String"/>
32 <form-property name="address" type="java.lang.String"/>
33 </form-bean>
34 <form-bean name="delForm" type="org.apache.struts.action.DynaActionForm">
35 <form-property name="eid" type="java.lang.String"/>
36 </form-bean>
37 </form-beans>
38 <action-mappings>
39 <action path="/mytiles" forward="MyTiles"/>
40 <action path="/insert" forward="insertTile"/>
41 <action path="/update" forward="updateTile" />
42 <action path="/delete" forward="deleteTile"/>
43 <action path="/updatedata" name="dataForm" type="updateAction" scope="session">
44 <forward name="success" path="tile.main.success" redirect="false"/>
45 <forward name="failure" path="tile.main.failure" redirect="false"/>
46 </action>
47 <action path="/insertdata" name="dataForm" type="insertAction" scope="session">
48 <forward name="success" path="tile.main.success" redirect="false"/>
49 <forward name="failure" path="tile.main.failure" redirect="false"/>
50 </action>
51 <action path="/deletedata" name="delForm" type="delAction" scope="session">
52 <forward name="success" path="tile.main.success" redirect="false"/>
53 <forward name="failure" path="tile.main.failure" redirect="false"/>
54 </action>
55 </action-mappings>
56 <plug-in className="org.apache.struts.tiles.TilesPlugin">
57 <set-property property="definitions-config" value="/WEB-INF/tiles-defs.xml"/>
58 <set-property property="moduleAware" value="true"/>
59 </plug-in>
60 </struts-config>

```

action path

Tile definition name in tile definition configuration file.refer line : 344

Tile definition logical name. refer lines : 141-146

Tile definition logical name. refer lines : 147-155

#### welcome.jsp

```

61 <b>
62 <center>
63 Hello welcome to Tiles Example
64 </center>
65 </b>

```

#### insert.jsp (main-content tile)

```

66 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
67 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html"%>
68 <html:form action="insertdata">
69 <table>
70 <tr><td> Employee Id </td><td><html:text property="eid"/></td></tr>
71 <tr><td> Employee Name </td><td><html:text property="name"/> </td></tr>

```

```

72 <tr><td> Employee Address </td><td><html:textarea property="address"/></td></tr>
73 <tr> <td colspan=2 align="center"><html:submit/><html:reset/> </td></tr>
74 </table>
75 </html:form>

```

#### update.jsp (main-content tile)

```

76 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
77 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
78 <html:form action="updatedata">
79 <table>
80 <tr><td> Employee Id </td><td><html:text property="eid"/></td></tr>
81 <tr><td> Employee Name </td><td><html:text property="name" /></td></tr>
82 <tr><td> Employee Address </td><td><html:textarea property="address"/> </td></tr>
83 <tr> <td colspan=2 align="center"><html:submit/><html:reset/> </td></tr>
84 </table>
85 </html:form>

```

#### delete.jsp (main-content tile)

```

86 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
87 <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html"%>
88 <html:form action="deletedata">
89 <table>
90 <tr><td> Employee Id </td><td><html:text property="eid" /></td></tr>
91 <tr> <td colspan=2 align="center"><html:submit/><html:reset/> </td></tr>
92 </table>
93 </html:form>

```

#### MyTilesHome.jsp (Layout page) (Two-column layout)

```

94 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
95 <%@ taglib uri="http://struts.apache.org/tags-tiles" prefix="tiles" %>
96 <html>
97 <head>
98 <title>
99     Login Screen
100 </title>
101 </head>
102 <body>
103 <table width="100%" cellpadding="4" cellspacing="0">
104 <tr>
105 <td align="center" colspan="2">
106 <tiles:insert attribute="header" /> → Tile-1 (Header tile)
107 </td>
108 </tr>
109 <!-- Content -->
110 <tr>
111 <!-- start content -->
112 <td width="20%">
113 <tiles:insert attribute="left-content" /> → Tile-2 (Left-content tile)
114 </td>
115 <td width="80%">
116 <tiles:insert attribute="main-content" /> → Tile-3 (Main-content tile)
117 </td>
118 <!-- end content -->
119 </tr>
120 <tr>
121 <td colspan="2">
122 <tiles:insert attribute="footer" /> → Tile-4 (Footer tile)

```

```

123     </td>
124 </tr>
125 </table>
126 </body>
127 </html>

```

#### BaseHeader.jsp (Header tile content)

```

128 <div align="center" style="color:#000033; background-color: #CCD9F9; font-size: 25px;
    font-weight: 900;text-align: center; border-width: 1px; border-style: solid;height:50px">
129 Tiles Sample Application<br>
130 <font style="color:#000033; background-color: #CCD9F9; font-size: 11px; font-weight: 100;
    text-align: center;">
131 Developed in Struts Exadel Studio
132 </font>
133 </div>

```

#### BaseFooter.jsp (Footer tile content)

```

134 <div align="center" style="color:#000033; background-color: #CCD9F9; font-size: 11px;
    font-weight: 900;text-align: center; border-width: 1px; border-style: solid;height:30px">
135 All Rights Are Reserved By Sathya Technologies @ 2004<br>
136 <font style="color:#000033; background-color: #CCD9F9; font-size: 11px; font-weight: 100;
137 text-align: center;">
138 Developed in Struts Exadel Studio
139 </font>
140 </div>

```

#### success.jsp (result page (main-content) tile content)

```

141 <table width="100%">
142 <tr><td align="center"></td></tr>
143 <tr><td align="center"></td></tr><br><br><b>
144 <%= (String)request.getAttribute("Action")%>
145 Operation has Successfully Completed</b></td></tr>
146 </table>

```

#### failure.jsp (result page (main-content) tile content)

```

147 <table width="100%">
148 <tr>
149 <td align="center"><b>
150 <font color="red">Warning</font>
151 <%= (String)request.getAttribute("Action")%>
152 <b>Sorry The Operation You Have Done is Failed</b>
153 </td>
154 </tr>
155 </table>

```

#### DataBaseBean.java

```

156 package bean;
157 import java.sql.*;
158 import java.io.*;
159 public class DatabaseBean
160 {
161     Connection con;
162     Statement st;
163     PreparedStatement prest;
164     int i;
165     public void setConnection()
166     {
167         try
168         {

```

```

169 Class.forName("oracle.jdbc.driver.OracleDriver");
170 con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","scott","tiger");
171 System.out.println("Connection Successfully Created "+con);
172 }
173 catch(Exception e)
174 {
175     con=null;
176     e.printStackTrace();
177     System.out.println("Connection Not Established");
178 }
179 }
180 public int insertData(int eid,String name,String add)
181 {
182     System.out.println(eid+": "+name+": "+add);
183     try
184     {
185         System.out.println("Connection "+con);
186         st=con.createStatement();
187         String qry="insert into StrutsEmployee values("+eid+",\'"+name+"\',"+\'"+add+"\')";
188         System.out.println(qry);
189         i=st.executeUpdate(qry);
190         System.out.println("Data Inserted Successfully");
191         st.close();
192         return 1;
193     }
194     catch(Exception e)
195     {
196         e.printStackTrace();
197     }
198     System.out.println("The Value of i "+i);
199     return 0;
200 }
201 public int updateData(int eid,String name,String add)
202 {
203     try
204     {
205         st=con.createStatement();
206         String query="update StrutsEmployee set name=\'"+name+"\',address=\'"+add+"\' where
eid="+eid;
207         System.out.println(query);
208         i=st.executeUpdate(query);
209         System.out.println("Data Updated Successfully");
210         st.close();
211         if(i==0)
212             return 0;
213         else
214             return 1;
215     }
216     catch(Exception e)
217     {
218         e.printStackTrace();
219     }
220     System.out.println("Update Qry : The Value of i "+i);
221     return 0;
222 }
223 public int deleteData(int eid)
224 {
225     try

```

```

226 {
227     st=con.createStatement();
228     String query="delete from StrutsEmployee where eid="+eid;
229     System.out.println(query);
230     i=st.executeUpdate(query);
231     System.out.println("Data Deleted Successfully");
232     st.close();
233     if(i==0)
234         return 0;
235     else
236         return 1;
237 }
238 catch(Exception e)
239 {
240     e.printStackTrace();
241 }
242 System.out.println("del Qry : The Value of i "+i);
243 return 0;
244 }
245 public void closeConnection()
246 {
247     try
248     {
249         con.close();
250         System.out.println("Connection Closed Successfully");
251     } catch(Exception e)
252     {
253         e.printStackTrace();
254     }
255 }
256 }

```

### insertAction.java

```

257 import javax.servlet.http.*;
258 import org.apache.struts.action.*;
259 import org.apache.struts.action.DynaActionForm;
260 import bean.DatabaseBean;
261 public class insertAction extends Action
262 {
263     DatabaseBean dbean;
264     public ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest
        request, HttpServletResponse response)
265     {
266         dbean = new DatabaseBean(); //DAO Class instantiation
267         dbean.setConnection(); //refer line 165
268         DynaActionForm myForm = (DynaActionForm)form;
269         System.out.println(myForm);
270         String eid=(String)myForm.get("eid");
271         int empid=Integer.parseInt(eid);
272         String name=(String)myForm.get("name");
273         String add=(String)myForm.get("address");
274         System.out.println(empid+name+add);
275         System.out.println("Result is Checking");
276         int result=dbean.insertData(empid,name,add);
277         System.out.println("Result is "+result);
278         dbean.closeConnection();
279         request.setAttribute("Action","Insert");

```

Reading form data from FormBean class object.

Method of DAO class inserting record.  
Refer lines :180-200

```

280 if (result==1)
281 return mapping.findForward("success");
282 else
283 return mapping.findForward("failure");
284 }
285 }

```

#### updateAction.java

```

286 import javax.servlet.http.*;
287 import org.apache.struts.action.*;
288 import org.apache.struts.action.DynaActionForm;
289 import bean.DatabaseBean;
290 public class updateAction extends Action
291 {
292 DatabaseBean dbean;
293 public ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest
    request, HttpServletResponse response)
294 {
295 dbean = new DatabaseBean();
296 dbean.setConnection();
297 DynaActionForm myForm = (DynaActionForm)form;
298 System.out.println(myForm);
299 String eid=(String)myForm.get("eid");
300 int empid=Integer.parseInt(eid);
301 String name=(String)myForm.get("name");
302 String add=(String)myForm.get("address");
303 System.out.println(eid+name+add);
304 int result=dbean.updateData(empid,name,add);
305 System.out.println("Result is "+result);
306 dbean.closeConnection();
307 request.setAttribute("Action","update");
308 if (result==1)
309 return mapping.findForward("success");
310 else
311 return mapping.findForward("failure");
312 }
313 }

```

#### deleteAction.java

```

314 import javax.servlet.http.*;
315 import org.apache.struts.action.*;
316 import org.apache.struts.action.DynaActionForm;
317 import bean.DatabaseBean;
318 public class delAction extends Action
319 {
320 DatabaseBean dbean;
321 public ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest
    request, HttpServletResponse response)
322 {
323 dbean = new DatabaseBean();
324 dbean.setConnection();
325 DynaActionForm myForm = (DynaActionForm)form;
326 System.out.println(myForm);
327 String eid=(String)myForm.get("eid");
328 int empid=Integer.parseInt(eid);
329 System.out.println(eid);
330 int result=dbean.deleteData(empid);
331 System.out.println(empid);
332 dbean.closeConnection();
333 request.setAttribute("Action","Delete");
334 if (result==1)

```



```

335 return mapping.findForward("success");
336 else
337 return mapping.findForward("failure");
338 }
339 }

```

#### tiles-defs.xml (tile definition cfg file (based on tiles inheritance))

```

340 <!DOCTYPE tiles-definitions PUBLIC
341 "-//Apache Software Foundation//DTD Tiles Configuration//EN"
342 "http://jakarta.apache.org/struts/dtds/tiles-config_1_1.dtd">
343 <tiles-definitions>
344   <definition name="MyTiles" path="/pages/MyTilesHome.jsp">
345     <put name="header" value="/pages/BaseHeader.jsp" />
346     <put name="left-content" value="/pages/LeftNavigation.jsp" />
347     <put name="main-content" value="/pages/welcome.jsp" />
348     <put name="footer" value="/pages/BaseFooter.jsp" />
349   </definition>
350   <definition name="tile.main.success" extends="MyTiles">
351     <put name="main-content" value="/pages/success.jsp" />
352   </definition>
353   <definition name="tile.main.failure" extends="MyTiles">
354     <put name="main-content" value="/pages/failure.jsp" />
355   </definition>
356   <definition name="insertTile" extends="MyTiles">
357     <put name="main-content" value="/pages/insert.jsp" />
358   </definition>
359   <definition name="deleteTile" extends="MyTiles">
360     <put name="main-content" value="/pages/delete.jsp" />
361   </definition>
362   <definition name="updateTile" extends="MyTiles">
363     <put name="main-content" value="/pages/update.jsp" />
364   </definition>
365 </tiles-definitions>

```

Diagram illustrating the tile definitions in `tiles-defs.xml`:

- Tile definition name**: Points to the `name` attribute in the `<definition>` tag.
- Name of the Layout page**: Points to the `path` attribute in the `<definition>` tag.
- Tile definition for webpage -1 (also acts as base definition)**: Points to the `<definition name="MyTiles">` block.
- Tile definition for 5th web page**: Points to the `<definition name="tile.main.success">` block.
- Tile definition for 6th web page**: Points to the `<definition name="tile.main.failure">` block.
- Tile definition for 2nd web page**: Points to the `<definition name="insertTile">` block.
- Tile definition for 4th web page**: Points to the `<definition name="deleteTile">` block.
- Tile definition for 3rd web page**: Points to the `<definition name="updateTile">` block.

#### LeftNavigation.jsp

```

366 <!DOCTYPE html PUBLIC "-//W3C/DTD XHTML 1.0 Transitional//EN"
367 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
368 <%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
369 <table width="100%" style="color:#000033; background-color: #CCD9F9; font-size: 15px; font-
370 weight: 900; text-align: center; border-width: 1px; border-style: solid;border-color:#000033">
371   <tr><td align="center" height=30></td></tr>
372   <tr><td align="center" height=30><html:link action="insert" >insert</html:link></td>
373   <tr>
374     <td align="center" height=30><html:link action="update" >update<br></html:link></td>
375   <tr>
376     <td align="center" height=30><html:link action="delete" >delete</html:link></td>
377   <tr>
378   <tr><td align="center" height=30></td></tr>
379 </table>
380 </table>

```

Diagram illustrating the navigation links in `LeftNavigation.jsp`:

- Refer line : 40**: Points to the `<html:link action="insert">` tag.
- Refer line : 41**: Points to the `<html:link action="update">` tag.
- Refer line : 42**: Points to the `<html:link action="delete">` tag.

**Flow of Execution of the Tiles application source code which is discussed above.**

- Programmer deploys tiles application in web server



- b. Because of load on startup servlet container creates ActionServlet object during struts Application deployment (or) during server startup.
- c. ActionServlet reads struts configure file and recognizes tiles PlugIn.
- d. Enduser gives request to tiles Application. <http://localhost:2020/tile> application.
- e. The default welcome file index.JSP will executes.
- f. Index.jsp gives internal request to tiles application (refer line: 2)
- g. ActionServlet to applications and takes the request (internal) (refer lines: 16-19 & 6-14)
- h. ActionServlet looks for Action tag in struts configure file whose Action path is /Mytiles (refer line :39)
- i. ActionServlet Forwards control to tiles-defs.xml file pointing the tile definition whose name is "MyTiles" (refer lines :344 to 349)
- j. ActionServlet launches MyTilesHome.jsp (layoutpage) having the tile values
  - Header -> BaseHeader.jsp
  - Footer -> BaseFooter.jsp
  - Main-content -> welcome.jsp
  - Left-content -> LeftNavigation.jsp
- k. Enduser clicks on insert hyperlink of left content file.
- l. ActionServlet traps and takes the request (refer lines: 16-19 & 6-14)
- m. ActionServlet looks for <action> in struts-config.xml whose action path is "/insert". (refer line : 40)
- n. ActionServlet Forwards the control to tiles-defs.xml file whose logical name is "insert Tile" (refer lines : 356-358).
- o. ActionServlet displays Mytiles Home.jsp (layout page) having the following Tile values.
  - Header -> BaseHeader.jsp
  - Footer -> BaseFooter.jsp
  - Main-content -> insert.jsp
  - Left-content -> leftNavigation.jsp
- p. In the above process the form page insert.jsp becomes main-content file value so its form bean class object will be created. (refer lines 66-75 and 29-33)
- q. Enduser submits the insert.jsp related form page.
- r. Generated request url contains (insertdata.do) and sends request to strutsapp (ref 114)
- s. ActionServlet traps and takes the request (ref 26-29 & 16-24)

- t. ActionServlet looks for action class configure in struts-config.xml whose action path is /insertion (refer line : 47)
- u. ActionServlet locates form bean class object and writes formdata.( refer lines 29-33)
- v. ActionServlet calls execute (-,-,-) of insert Action class.this execute (-,-,-) takes insertData(-,-,-) method of Database Bean class (DAO) (refer lines:264-284 & 180-200)

Success		Failure
281	(w)	283
48	(x)	49
350-352	(y)	353-355

Z. ActionServlet displays the layout page MyTilesHome.jsp having the following tile values.

Header -> Baseheader.jsp

Footer -> Basefooter.jsp

Left-content -> leftNavigation.jsp

Main-content -> success.jsp | failure.jsp

### **DataBase table required for the above application.**

#### StrutsEmployee

eid – number

name – varchar2(20)

address – varchar2(20)

sql>Create table strutsemployee (eid, number, name varchar2(20),address varchar2(20));

sql>Commit;

- The helper java class for ActionServlet that helps ActionServlet while performing Request-processing operation is called as Abstract Controller (refer initial classes of Abstract controller)
- The predefined Request processor class Acts as abstract controller for ActionServlet. This class contains 16+ process xxx(-,-) methods to perform Request processing operation on behalf of ActionServlet for the request given to struts Application.

These processXxx(-,-) methods are

processValidate (-,-)

process Preeprocess (-,-)

process ActionPerform (-,-)

.....

.....

And e.t.c

### **Understanding Template method DesignPattern**

**Task 1 →** x();

```

y();
z();
m1();
m2();

```

( Need to remember order and sequence of methods to call sequence of methods to call in order to complete the task)

### Solution: Provide Template method

```

public void mymethod()
{
    x();
    y();
    z();
    m1();
    m2();
}

```

### Task-1 → mymethod();

Here we just need to call only one method to complete the task-1 and there is no need of remembering order and sequence to call the methods.

- ❖ The process (-,-) method of predefined request dispatcher is template method because it is called 16 process xxx(-,-) methods in a sequence to complete request processing operation. For the request that is trapped by ActionServlet.
- ❖ Instead of calling 16 processxxx(-,-) of requestprocess class in a sequence to perform Request processing ActionServlet just calls only one process method of RequestProcessor class to complete Request Processing (refer source code of ActionServlet.java and Request Processor.java)
- ❖ ActionServlet creates RequestProcessor class object on one per module basis. For blueprint of process method in Request processor class refer page nos 70 & 71 of Booklet.
- ❖ To provide Additional instructions to processxxx (-,-) of custom (or) predefined Request processor class to change their default operation we take the support of controller tag in struts configure file as shown below.

### To change the default Response content type text |html:

```

<controller>
    <set-property property="contentType" value="text/xml">
</controller>

```

### To make the response generated by Struts Application storing in buffer of Browser Window:

```

<controller>
    <set-property property="no cache" value="+ve">
</controller>

```

- Process preProcess() of predefined Request processor class is an empty method containing no logic and returning true always.

- In developing to custom RequestProcessor class programmer generally override this method to place his choice of Request Processing logic.
- In order to get control on ActionServlet related Request Processing operation we can develop our own custom RequestProcessor class as shown below.

### Procedure to develop our own RequestProcessor class.

#### Step-1

Develop a java class extending from predefined RequestProcessor class and override your choice processXxx(-,-) methods.

**Note:** Our RequestProcessor class is configurable on per module basis.

**MyRp.java (save in WEB-INF/classes folder of switch application which is discussed in SwitchAction class discussions)**

```
// MyRp.java
//Blocks the request going to StudentAction of Student module
//when request comes from browser window other than Internet Explorer.
import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.struts.action.*;
public class MyRp extends RequestProcessor
{
    public boolean processprocess(HttpServletRequest req, HttpServletResponse res)
    {
        System.out.println("Myrp: processPreprocess (-,-));
        try
        {
            String bname=req.getHeader("user-agent");
            If(bname.indexOf("MSIE") == -1) //if request is coming from other than iexplorer.
            {
                RequestDispatcher rd= req.getRequestDispatcher("err.jsp");
                rd.forward(req,res);
                return false;
            }
            else // if req is coming IE then continue request processing.
            {
                return false;
            }
        } //try
        catch (Exception e)
        {
            return false;
        }
    } //method
} //class
```

#### Step-2

Add err.jsp in student folder of SwitchApp application.

**In err.jsp**

```
<b><font color=red> Hey !!! Give request from IE Browser window for student module
</font> </b>
```

### Step-3

Configure the above java class as Abstract controller class for student module of SwitchApp application through that module related struts configuration file called "struts-config-student.xml".

#### In struts-config-student.xml

After </action-mapping> tag

```
<controller>
    <set-property property="processorClass" value="Myrp"/>
</controller>
```

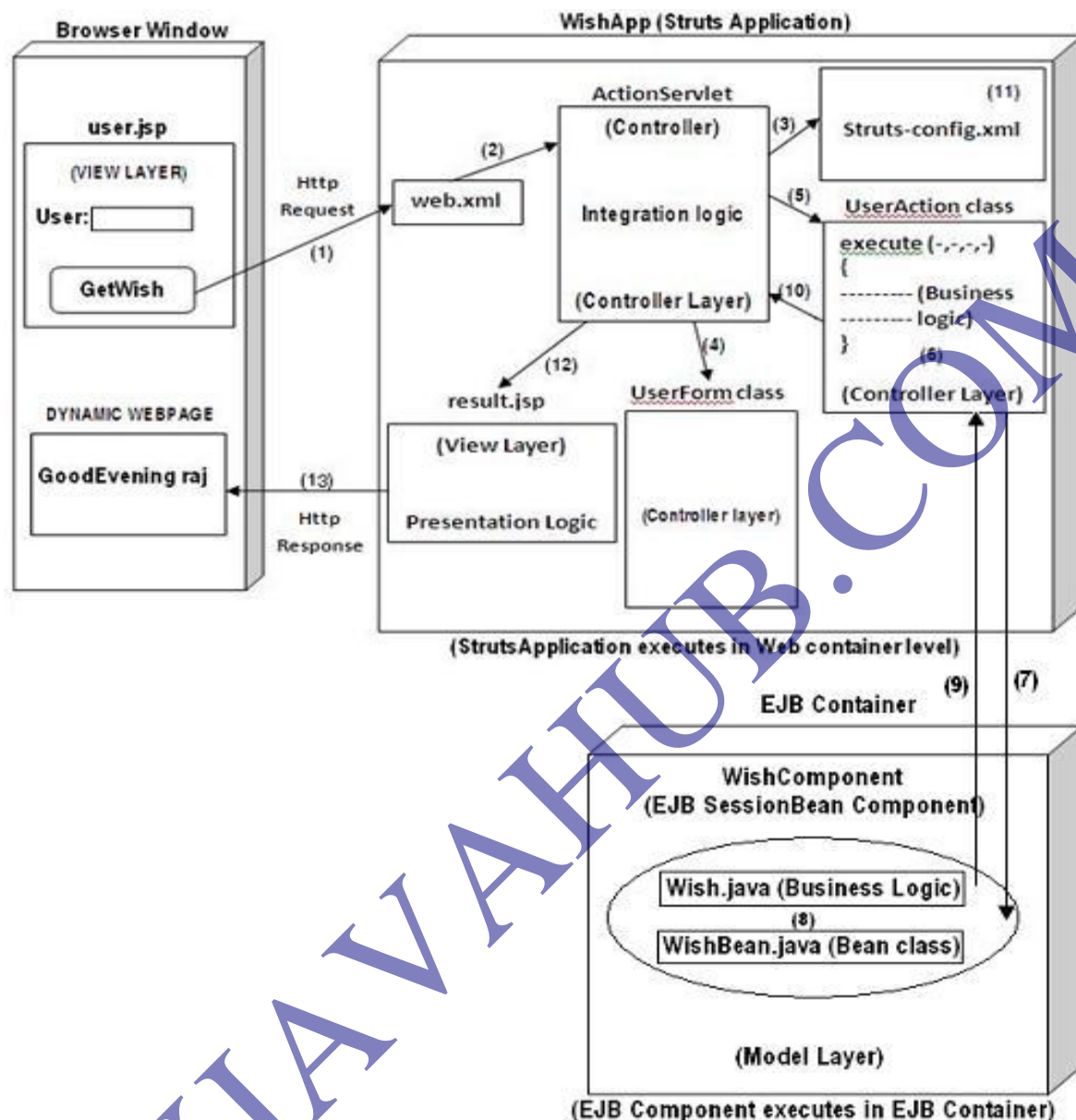
- RequestProcessor class object is one per module in struts application. So we can configured custom Request processor class on per module basis.
- For another example on custom RequestProcessor class for session related operations for changing Default response content type refer example code given in 74.

Don't place the model layer business logic and persistence logic directly in struts application while developing large scale projects. The reasons are.

1. Business logic and persistence logic becomes specific to one struts application. That means logics cannot be accessed from other applications.
  2. Only those clients who can generate http request can access business logic and persistence logic of struts Action class.
  3. Middleware services must be implemented manually in struts Action class. This improves burden on the programmer. The middleware services are like transaction management, Security, logging and etc.
- To overcome all these problems, keep business logics and persistence logic in separate model layer resources, like EJB component (session bean components) or spring apps or spring with hibernate apps.
  - In all the above 3 technologies, the middleware services are given as implicit middleware services.
  - When B.L and P.L is placed in other layer resources, then struts Action class needs to communicate with them. In that process, struts Action class acts as controller layer resource.
  - In struts-EJB communicate, the struts Action class acts as remote client to EJB component Struts App -> EJB component -> DB s/w

- Strutapp -> contains presentation logic and Integration logic EJB component -> contains Bussiness logic and persistence logic.

### Example application a struts EJB communication:



- Java annotations are java statements (begins with '@' symbol) which are alternate for the xml files based MetaData operations and resources configurations. They can be applied class oblique interface level, method level and field level (member variable)

#### Syntax:

@<annotation-name> (param1=var1, param2 =val2,.....)

For above diagram based example application refer the following application.

- When EJB component is deployed in Glassfish server as jar file, it will be identified with its business interface name as, identification name / JNDI name.
- To provide global visibility to any object, we can keep that object reference in JNDI registry s/w having nickname or alias name.
- Client applications need wrapper object of EJB 3.x components to call its business methods.

- When EJB 3.x component is deployed, the component related wrapper object reference will be placed in JNDI registry, by having business interface name as nickname / alias name.

**Complete source code for the above diagram based Struts-to-EJB Communication.**

### 1) Struts Application source code

#### user.jsp

```

1  <%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
2  <html:html>
3      <html:form action = "/wishact">
4          <table border = 0 align = center>
5              <tr>
6                  <th>Enter user Name</th>
7                  <td><html:text property = "username"/>
8              </tr>
9              <html:submit value="send"/>
10         </tr>
11     </table>
12 </html:form>
13 </html:html>

```

#### web.xml

```

14 <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
15 "http://java.sun.com/dtd/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         <load-on-startup>2</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>user.jsp</welcome-file>
32     </welcome-file-list>
33 </web-app>

```

#### struts-config.xml

```

34 <!DOCTYPE struts-config PUBLIC
35 "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
36 "http://struts.apache.org/dtds/struts-config_1_3.dtd">
37 <struts-config>
38     <form-beans><form-bean name="rf" type="UserForm"/></form-beans>
39     <action-mappings>
40         <action path="/wishact" type="UserAction" name="rf">
41             <forward name="res" path="/result.jsp"/>
42         </action>
43     </action-mappings>
44 </struts-config>

```

#### UserForm.java

```

45 import org.apache.struts.action.*;
46 import org.apache.struts.validator.*;
47 public class UserForm extends ActionForm
48 {
49     private String username ;
50     public void setUsername(String username)
51     {
52         this.username = username;
53     }
54     public String getUsername()
55     {
56         return username;
57     }
58 }

```

#### UserAction.java

```

59 import org.apache.struts.action.*;
60 import javax.servlet.http.*;
61 import java.util.*;
62 import javax.naming.*;
63 public class UserAction extends Action
64 {
65     Wish wor=null;
66     public UserAction()
67     {
68         try
69         {
70             InitialContext ic=new InitialContext();
71             wor=(Wish)ic.lookup("Wish");
72         }
73         catch(Exception e)
74         { e.printStackTrace();}
75     }
76     public ActionForward execute(ActionMapping mapping,ActionForm form,HttpServletRequest
77     request,HttpServletResponse response)throws Exception
78     {
79         try
80         {
81             System.out.println("In UserAction:execute() ");
82             UserForm rf = (UserForm)form;
83             String user = rf.getUsername();
84             String result=wor.getWishMsg(user); //calling business method of EJB Component
85             request.setAttribute("result",result); //keeping the result in the request attribute to send to result page
86         }
87         catch(Exception e)
88         { e.printStackTrace();}
89         return mapping.findForward("res");
90     }

```

Lookup code to get wrapper object reference from JNDI registry.

Reading form data from FormBean class object.

#### result.jsp

```

91 <html>
92 <body><center>
93 <font size = 5 color = green>Result Page</font> Wish message is
94 <%=request.getAttribute("result") %>
95 </center></body>
96 </html>

```

Displaying the result gathered from request attribute.

#### 2) EJB 3.x Component Source code



**Wish.java (Business Interface) (POJI)**

```

1  import javax.ejb.Remote;
2  @Remote
3  public interface Wish
4  {
5      public String getWishMsg(String uname); //Declaration of Business method.
6  }

```

**WishBean.java (Bean class) (POJO)**

```

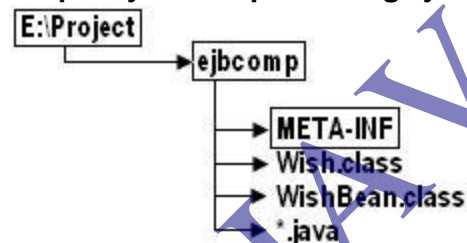
7  import javax.ejb.*;
8  import java.util.*;
9  @Stateless
10 public class WishBean implements Wish
11 {
12     public String getWishMsg(String name)
13     {
14         System.out.println("getWishMsg(-):WishBean");
15         Calendar cl=Calendar.getInstance();
16         int h=cl.get(Calendar.HOUR_OF_DAY);
17         if(h<12)
18             return "Good morning:"+name;
19         else if(h<16)
20             return "Good Afternoon :"+name;
21         else
22             return "Good evening:"+name;
23     } //method
24 }

```

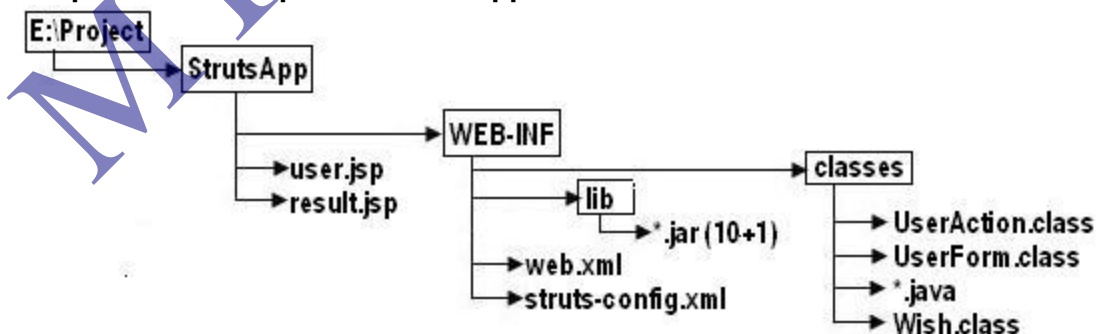
Business method implementation.

**Procedure for deploying the above application**

To compile the \*.java files add 'javaee.jar' (glassfish) or weblogic (weblogic) to classpath.  
 Javaee.jar -> available in glassfish home\appserver\lib folder.

**Prepare jar file representing ejb component**

E:\project\ejbcomp>.jar cf comp.jar

**Prepare war file represents web application**

➤ Get wish.class file from ejbcomp

E:\project\strutsapp>.jar cf strutsapp.war

## Deployment methodologies

### Approach-1

Deploy strutsApp.war, component.jar files separately in Glassfish server.

To do this, copy the comp.jar, strutsApp.war files in

E:\SunMicroSystems\AppServer\Domains\Domain1\AntiDeploy folder

### To Test this App the request url is

http://localhost:2020/strutsApp/user.jsp (changes for system to system)

### Approach-2 (May not work in all versions of tomcat)

- Deploy strutsApp.war in tomcat home\webapp folder.
- Deploy component.jar in Glassfish server as show above approach.

### Test the application with the following url

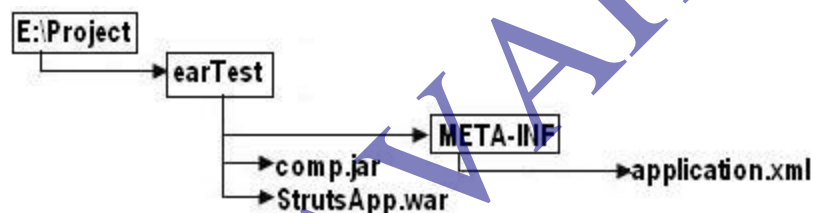
http://localhost:1987/strutsApp/usre.jsp (changes for system to system)

**Note:** To make tomcat webserver aware of EJB 3.x API, copy the "javaee.jar" file of Glassfish server (Glassfish home\appserver\lib) to tomcat\_home\lib folder also copy "appserver-rt.jar", "appserver-ee.jar", "appserv-admin.jar" file in the same folder.

### Approach-3

Prepare ear(jar+war) file and deploy the ear file in glassfish server.

### To prepare ear file



E:\project\earTest>jar cf final.ear.

- 'Application.xml' is the deployment description file of EAR based application.

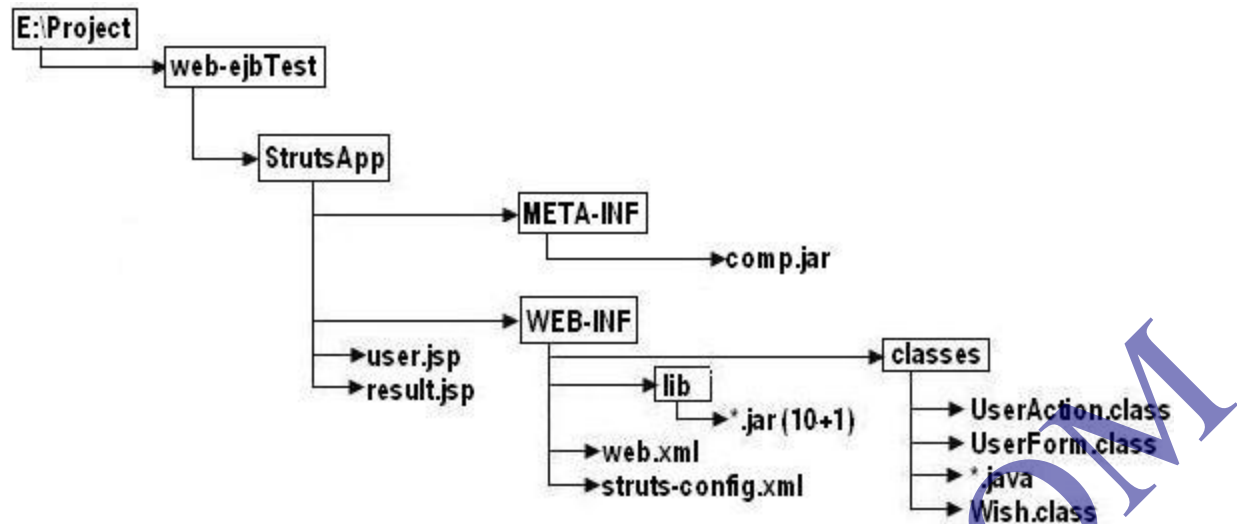
### application.xml

```

<!DOCTYPE application PUBLIC "-//Sun Microsystems, Inc.//DTD J2EE Application 1.2/EN"
'http://java.sun.com/j2ee/dtds/application_1_2.dtd'>
<application>
  <display-name>EARFile</display-name>
  <module>
    <web>
      <web-uri>StrutsApp.war</web-uri>
      <context-root>MyWeb</context-root>
    </web>
  </module>
  <module>
    <ejb>comp.jar</ejb>
  </module>
</application>
  
```

#### Approach-4

Deploy web Application + EJB component as war file.



#### Prepare war file representing web-application

E:\projcet\web-ejbTest\StrutsApp> jar cf strutsApp1.war

Deploy the above war file in Glassfish server.

#### Request url to test the application

http://localhost:8080/strutsApp1/user.jsp

The best approach is approach-3 to deploy struts and ejb combination based projects.

#### Netbeans IDE

File → new → Java EE → Enterprise Application → next → project name MyEarProg → server name Glassfish

- While developing struts and EJB integration project through netbeans, MyEclipse IDE software. Create your project in IDE software as Enterprise Application project. This project gives two sub modules one for web-application development and another one for EJB component development.

#### Limitations of struts 1.x

- Struts Application resources are struts API dependent.
- Struts API give more abstract classes and less interfaces. This is bad designing of API (Because 1 Java class cannot extend from multiple abstract classes).
- Working with check boxes and list boxes in session scoped FormBean is quiet complex process.
- Debugging problems from struts 1.x application is quiet complex process.
- In most of the situations we feel FormBean class is unnecessary because the data of form page can be gathered from request object of execute method parameter.
- No built-in support to work with Ajax.
- No built-in support to integrate with spring & Hibernate concepts.
- Working with validator PlugIn is quiet complex process.
- More servlet API dependency is there while working with FormBean, Action classes.