JSP

**JSP** technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.

A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags, etc.

Advantages of JSP over Servlet

There are many advantages of JSP over the Servlet. They are as follows:

1) Extension to Servlet

JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

2) Easy to maintain

JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.

3) Fast Development: No need to recompile and redeploy

If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

4) Less code than Servlet

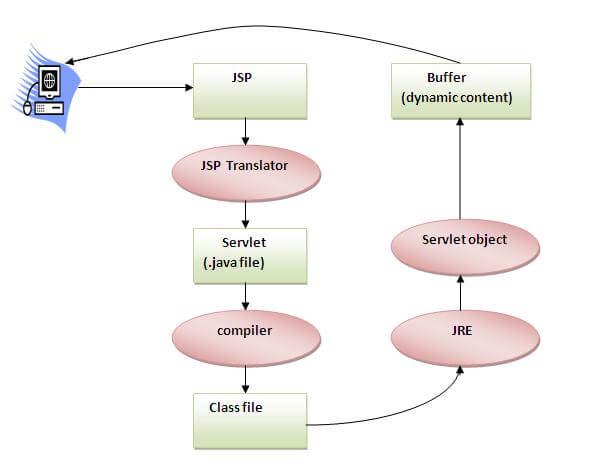
In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.

The Lifecycle of a JSP Page

The JSP pages follow these phases:

* Translation of JSP Page
* Compilation of JSP Page
* Classloading (the classloader loads class file)
* Instantiation (Object of the Generated Servlet is created).
* Initialization ( the container invokes jspInit() method).
* Request processing ( the container invokes \_jspService() method).
* Destroy ( the container invokes jspDestroy() method).

Note: jspInit(), \_jspService() and jspDestroy() are the life cycle methods of JSP.



As depicted in the above diagram, JSP page is translated into Servlet by the help of JSP translator. The JSP translator is a part of the web server which is responsible for translating the JSP page into Servlet. After that, Servlet page is compiled by the compiler and gets converted into the class file. Moreover, all the processes that happen in Servlet are performed on JSP later like initialization, committing response to the browser and destroy.

Creating a simple JSP Page

To create the first JSP page, write some HTML code as given below, and save it by .jsp extension. We have saved this file as index.jsp. Put it in a folder and paste the folder in the web-apps directory in apache tomcat to run the JSP page.

**index.jsp**

Let's see the simple example of JSP where we are using the scriptlet tag to put Java code in the JSP page. We will learn scriptlet tag later.

1. <html>
2. <body>
3. <% out.print(2\*5); %>
4. </body>
5. </html>

It will print **10** on the browser.

How to run a simple JSP Page?

Follow the following steps to execute this JSP page:

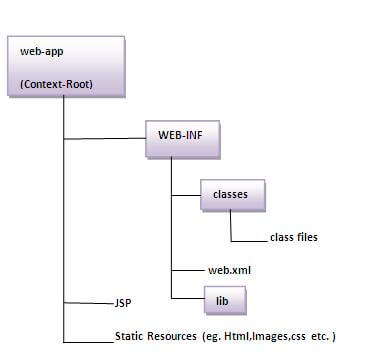
* Start the server
* Put the JSP file in a folder and deploy on the server
* Visit the browser by the URL http://localhost:portno/contextRoot/jspfile, for example, http://localhost:8888/myapplication/index.jsp

Do I need to follow the directory structure to run a simple JSP?

No, there is no need of directory structure if you don't have class files or TLD files. For example, put JSP files in a folder directly and deploy that folder. It will be running fine. However, if you are using Bean class, Servlet or TLD file, the directory structure is required.

The Directory structure of JSP

The directory structure of JSP page is same as Servlet. We contain the JSP page outside the WEB-INF folder or in any directory.



# The JSP API

The JSP API consists of two packages:

1. javax.servlet.jsp
2. javax.servlet.jsp.tagext

## javax.servlet.jsp package

The javax.servlet.jsp package has two interfaces and classes.The two interfaces are as follows:

1. JspPage
2. HttpJspPage

The classes are as follows:

* JspWriter
* PageContext
* JspFactory
* JspEngineInfo
* JspException
* JspError

## The JspPage interface

According to the JSP specification, all the generated servlet classes must implement the JspPage interface. It extends the Servlet interface. It provides two life cycle methods.

### Methods of JspPage interface

1. **public void jspInit():** It is invoked only once during the life cycle of the JSP when JSP page is requested firstly. It is used to perform initialization. It is same as the init() method of Servlet interface.
2. **public void jspDestroy():** It is invoked only once during the life cycle of the JSP before the JSP page is destroyed. It can be used to perform some clean up operation.

## The HttpJspPage interface

The HttpJspPage interface provides the one life cycle method of JSP. It extends the JspPage interface.

### Method of HttpJspPage interface:

1. **public void \_jspService():** It is invoked each time when request for the JSP page comes to the container. It is used to process the request. The underscore \_ signifies that you cannot override this method.

# JSP Implicit Objects

There are **9 jsp implicit objects**. These objects are *created by the web container* that are available to all the jsp pages.

The available implicit objects are out, request, config, session, application etc.

A list of the 9 implicit objects is given below:

|  |  |
| --- | --- |
| **Object** | **Type** |
| Out | JspWriter |
| Request | HttpServletRequest |
| Response | HttpServletResponse |
| Config | ServletConfig |
| Application | ServletContext |
| Session | HttpSession |
| pageContext | PageContext |
| Page | Object |
| Exception | Throwable |

### 1) JSP out implicit object

For writing any data to the buffer, JSP provides an implicit object named out. It is the object of JspWriter. In case of servlet you need to write:

1. PrintWriter out=response.getWriter();

But in JSP, you don't need to write this code.

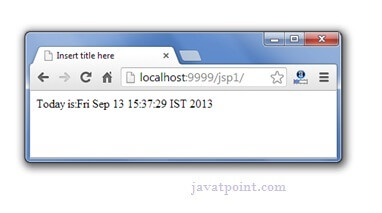
### Example of out implicit object

In this example we are simply displaying date and time.

### index.jsp

1. <html>
2. <body>
3. <% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
4. </body>
5. </html>

#### Output



JSP request implicit object

The **JSP request** is an implicit object of type HttpServletRequest i.e. created for each jsp request by the web container. It can be used to get request information such as parameter, header information, remote address, server name, server port, content type, character encoding etc.

It can also be used to set, get and remove attributes from the jsp request scope.

Let's see the simple example of request implicit object where we are printing the name of the user with welcome message.

Example of JSP request implicit object

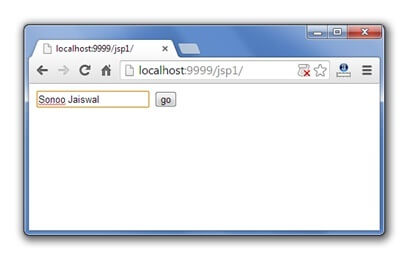
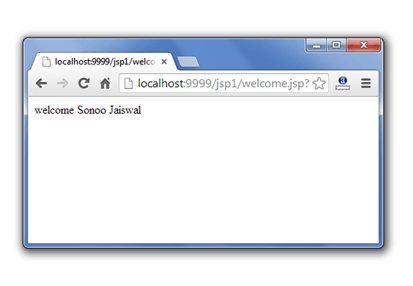
**index.html**

1. **<form** action="welcome.jsp"**>**
2. **<input** type="text" name="uname"**>**
3. **<input** type="submit" value="go"**><br/>**
4. **</form>**

**welcome.jsp**

1. <%
2. String name=request.getParameter("uname");
3. out.print("welcome "+name);
4. %>

Output

3) JSP response implicit object

In JSP, response is an implicit object of type HttpServletResponse. The instance of HttpServletResponse is created by the web container for each jsp request.

It can be used to add or manipulate response such as redirect response to another resource, send error etc.

Let's see the example of response implicit object where we are redirecting the response to the Google.

Example of response implicit object

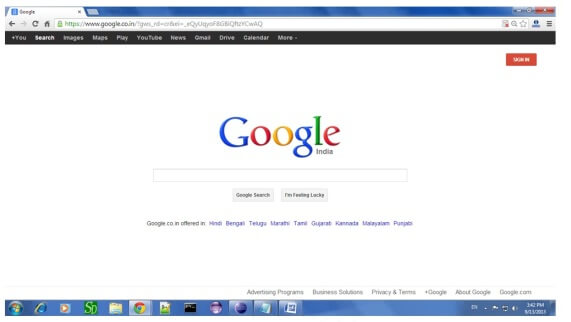
**index.html**

1. **<form** action="welcome.jsp"**>**
2. **<input** type="text" name="uname"**>**
3. **<input** type="submit" value="go"**><br/>**
4. **</form>**

**welcome.jsp**

1. **<**%
2. response.sendRedirect("http://www.google.com");
3. %**>**

Output



4) JSP config implicit object

In JSP, config is an implicit object of type *ServletConfig*. This object can be used to get initialization parameter for a particular JSP page. The config object is created by the web container for each jsp page.

Generally, it is used to get initialization parameter from the web.xml file.

Example of config implicit object:

**index.html**

1. **<form** action="welcome"**>**
2. **<input** type="text" name="uname"**>**
3. **<input** type="submit" value="go"**><br/>**
4. **</form>**

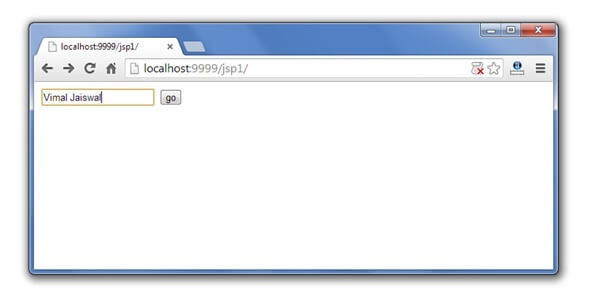
**web.xml file**

1. **<web-app>**
3. **<servlet>**
4. **<servlet-name>**sonoojaiswal**</servlet-name>**
5. **<jsp-file>**/welcome.jsp**</jsp-file>**
7. **<init-param>**
8. **<param-name>**dname**</param-name>**
9. **<param-value>**sun.jdbc.odbc.JdbcOdbcDriver**</param-value>**
10. **</init-param>**
12. **</servlet>**
14. **<servlet-mapping>**
15. **<servlet-name>**sonoojaiswal**</servlet-name>**
16. **<url-pattern>**/welcome**</url-pattern>**
17. **</servlet-mapping>**
19. **</web-app>**

**welcome.jsp**

1. **<**%
2. out.print("Welcome "+request.getParameter("uname"));
4. String driver=config.getInitParameter("dname");
5. out.print("driver name is="+driver);
6. %**>**

Output



5) JSP application implicit object

In JSP, application is an implicit object of type *ServletContext*.

The instance of ServletContext is created only once by the web container when application or project is deployed on the server.

This object can be used to get initialization parameter from configuaration file (web.xml). It can also be used to get, set or remove attribute from the application scope.

This initialization parameter can be used by all jsp pages.

Example of application implicit object:

**index.html**

1. **<form** action="welcome"**>**
2. **<input** type="text" name="uname"**>**
3. **<input** type="submit" value="go"**><br/>**
4. **</form>**

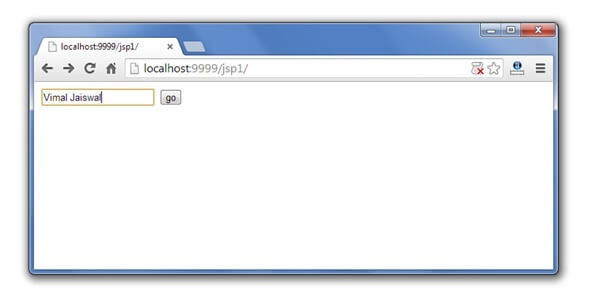
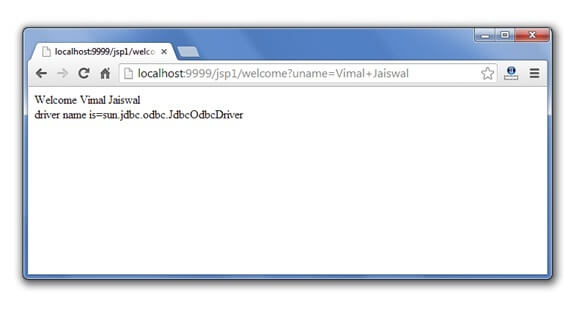
**web.xml file**

1. **<web-app>**
3. **<servlet>**
4. **<servlet-name>**sonoojaiswal**</servlet-name>**
5. **<jsp-file>**/welcome.jsp**</jsp-file>**
6. **</servlet>**
8. **<servlet-mapping>**
9. **<servlet-name>**sonoojaiswal**</servlet-name>**
10. **<url-pattern>**/welcome**</url-pattern>**
11. **</servlet-mapping>**
13. **<context-param>**
14. **<param-name>**dname**</param-name>**
15. **<param-value>**sun.jdbc.odbc.JdbcOdbcDriver**</param-value>**
16. **</context-param>**
18. **</web-app>**

**welcome.jsp**

1. **<**%
3. out.print("Welcome "+request.getParameter("uname"));
5. String driver=application.getInitParameter("dname");
6. out.print("driver name is="+driver);
8. %**>**

Output

6) session implicit object

|  |
| --- |
| In JSP, session is an implicit object of type HttpSession.The Java developer can use this object to set,get or remove attribute or to get session information. |

Example of session implicit object

**index.html**

1. <html>
2. <body>
3. <form action="welcome.jsp">
4. <input type="text" name="uname">
5. <input type="submit" value="go"><br/>
6. </form>
7. </body>
8. </html>

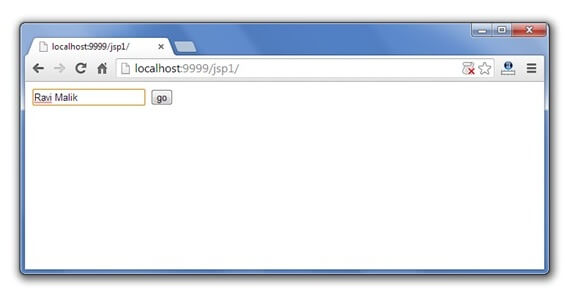
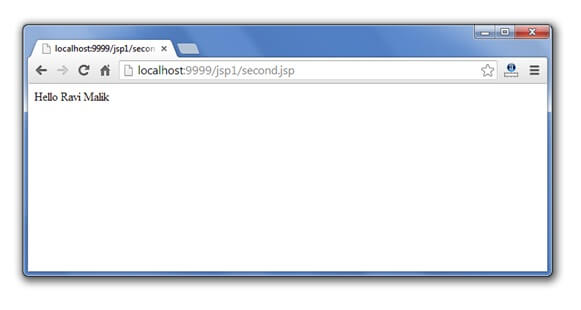
**welcome.jsp**

1. <html>
2. <body>
3. <%
5. String name=request.getParameter("uname");
6. out.print("Welcome "+name);
8. session.setAttribute("user",name);
10. <a href="second.jsp">second jsp page</a>
12. %>
13. </body>
14. </html>

**second.jsp**

1. <html>
2. <body>
3. <%
5. String name=(String)session.getAttribute("user");
6. out.print("Hello "+name);
8. %>
9. </body>
10. </html>

Output

7) pageContext implicit object

|  |
| --- |
| In JSP, pageContext is an implicit object of type PageContext class.The pageContext object can be used to set,get or remove attribute from one of the following scopes:   * page * request * session * application |
| In JSP, page scope is the default scope. |

Example of pageContext implicit object

**index.html**

1. <html>
2. <body>
3. <form action="welcome.jsp">
4. <input type="text" name="uname">
5. <input type="submit" value="go"><br/>
6. </form>
7. </body>
8. </html>

**welcome.jsp**

1. <html>
2. <body>
3. <%
5. String name=request.getParameter("uname");
6. out.print("Welcome "+name);
8. pageContext.setAttribute("user",name,PageContext.SESSION\_SCOPE);
10. <a href="second.jsp">second jsp page</a>
12. %>
13. </body>
14. </html>

**second.jsp**

1. <html>
2. <body>
3. <%
5. String name=(String)pageContext.getAttribute("user",PageContext.SESSION\_SCOPE);
6. out.print("Hello "+name);
8. %>
9. </body>
10. </html>

Output

# jsp pageContext implicit object output 1 jsp pageContext implicit object output 2 jsp pageContext implicit object output 3

# 8) page implicit object:

|  |
| --- |
| In JSP, page is an implicit object of type Object class.This object is assigned to the reference of auto generated servlet class. It is written as: |
| Object page=this; |
| For using this object it must be cast to Servlet type.For example: |
| <% (HttpServlet)page.log("message"); %> |
| Since, it is of type Object it is less used because you can use this object directly in jsp.For example: |
| <% this.log("message"); %> |

# 

9) exception implicit object

|  |
| --- |
| In JSP, exception is an implicit object of type java.lang.Throwable class. This object can be used to print the exception. But it can only be used in error pages.It is better to learn it after page directive. Let's see a simple example: |

Example of exception implicit object:

**error.jsp**

1. <%@ page isErrorPage="true" %>
2. <html>
3. <body>
5. Sorry following exception occured:<%= exception %>
7. </body>
8. </html>

# JSP directives

The **jsp directives** are messages that tells the web container how to translate a JSP page into the corresponding servlet.

There are three types of directives:

* page directive
* include directive
* taglib directive

### Syntax of JSP Directive

1. <%@ directive attribute="value" %>

### JSP page directive

The page directive defines attributes that apply to an entire JSP page.

### Syntax of JSP page directive

1. <%@ page attribute="value" %>

### Attributes of JSP page directive

* import
* contentType
* extends
* info
* buffer
* language
* isELIgnored
* isThreadSafe
* autoFlush
* session
* pageEncoding
* errorPage
* isErrorPage

### 1)import

|  |
| --- |
| The import attribute is used to import class,interface or all the members of a package.It is similar to import keyword in java class or interface. |

### Example of import attribute

1. <html>
2. <body>
4. <%@ page **import**="java.util.Date" %>
5. Today is: <%= **new** Date() %>
7. </body>
8. </html>

### 2)contentType

The contentType attribute defines the MIME(Multipurpose Internet Mail Extension) type of the HTTP response.The default value is "text/html;charset=ISO-8859-1".

### Example of contentType attribute

1. <html>
2. <body>
4. <%@ page contentType=application/msword %>
5. Today is: <%= **new** java.util.Date() %>
7. </body>
8. </html>

### 3)extends

The extends attribute defines the parent class that will be inherited by the generated servlet.It is rarely used.

### 4)info

This attribute simply sets the information of the JSP page which is retrieved later by using getServletInfo() method of Servlet interface.

### Example of info attribute

1. <html>
2. <body>
4. <%@ page info="vij" %>
5. Today is: <%= **new** java.util.Date() %>
7. </body>
8. </html>

The web container will create a method getServletInfo() in the resulting servlet.For example:

1. **public** String getServletInfo() {
2. **return** "composed by vij";
3. }

### 5)buffer

The buffer attribute sets the buffer size in kilobytes to handle output generated by the JSP page.The default size of the buffer is 8Kb.

### Example of buffer attribute

1. <html>
2. <body>
4. <%@ page buffer="16kb" %>
5. Today is: <%= **new** java.util.Date() %>
7. </body>
8. </html>

### 6)language

The language attribute specifies the scripting language used in the JSP page. The default value is "java".

### 7)isELIgnored

|  |
| --- |
| We can ignore the Expression Language (EL) in jsp by the isELIgnored attribute. By default its value is false i.e. Expression Language is enabled by default. We see Expression Language later. |

1. <%@ page isELIgnored="true" %>//Now EL will be ignored

### 8)isThreadSafe

|  |
| --- |
| Servlet and JSP both are multithreaded.If you want to control this behaviour of JSP page, you can use isThreadSafe attribute of page directive.The value of isThreadSafe value is true.If you make it false, the web container will serialize the multiple requests, i.e. it will wait until the JSP finishes responding to a request before passing another request to it.If you make the value of isThreadSafe attribute like: |

<%@ page isThreadSafe="false" %>

The web container in such a case, will generate the servlet as:

1. **public** **class** SimplePage\_jsp **extends** HttpJspBase
2. **implements** SingleThreadModel{
3. .......
4. }

### 9)errorPage

The errorPage attribute is used to define the error page, if exception occurs in the current page, it will be redirected to the error page.

### Example of errorPage attribute

1. //index.jsp
2. <html>
3. <body>
5. <%@ page errorPage="myerrorpage.jsp" %>
7. <%= 100/0 %>
9. </body>
10. </html>

### 10)isErrorPage

The isErrorPage attribute is used to declare that the current page is the error page.

#### Note: The exception object can only be used in the error page.

### Example of isErrorPage attribute

1. //myerrorpage.jsp
2. <html>
3. <body>
5. <%@ page isErrorPage="true" %>
7. Sorry an exception occured!<br/>
8. The exception is: <%= exception %>
10. </body>
11. </html>

# Jsp Include Directive

The include directive is used to include the contents of any resource it may be jsp file, html file or text file. The include directive includes the original content of the included resource at page translation time (the jsp page is translated only once so it will be better to include static resource).

### Advantage of Include directive

Code Reusability

### Syntax of include directive

1. <%@ include file="resourceName" %>

### Example of include directive

In this example, we are including the content of the header.html file. To run this example you must create an header.html file.

1. <html>
2. <body>
4. <%@ include file="header.html" %>
6. Today is: <%= java.util.Calendar.getInstance().getTime() %>
8. </body>
9. </html>

# JSP Taglib directive

1. [JSP Taglib directive](https://www.javatpoint.com/jsp-taglib-directive)
2. [Example of JSP Taglib directive](https://www.javatpoint.com/jsp-taglib-directive#taglibex)

The JSP taglib directive is used to define a tag library that defines many tags. We use the TLD (Tag Library Descriptor) file to define the tags. In the custom tag section we will use this tag so it will be better to learn it in custom tag.

#### Syntax JSP Taglib directive

1. <%@ taglib uri="uriofthetaglibrary" prefix="prefixoftaglibrary" %>

### Example of JSP Taglib directive

In this example, we are using our tag named currentDate. To use this tag we must specify the taglib directive so the container may get information about the tag.

1. <html>
2. <body>
4. <%@ taglib uri="http://www.javatpoint.com/tags" prefix="mytag" %>
6. <mytag:currentDate/>
8. </body>
9. </html>

# MVC in JSP

**MVC** stands for Model View and Controller. It is a **design pattern** that separates the business logic, presentation logic and data.

**Controller** acts as an interface between View and Model. Controller intercepts all the incoming requests.

**Model** represents the state of the application i.e. data. It can also have business logic.

**View** represents the presentaion i.e. UI(User Interface).

#### Advantage of MVC (Model 2) Architecture

1. Navigation Control is centralized
2. Easy to maintain the large application



#### If you new to MVC, please visit [Model1 vs Model2](https://www.javatpoint.com/model-1-and-model-2-mvc-architecture) first.

### MVC Example in JSP

In this example, we are using servlet as a controller, jsp as a view component, Java Bean class as a model.

In this example, we have created 5 pages:

* **index.jsp** a page that gets input from the user.
* **ControllerServlet.java** a servlet that acts as a controller.
* **login-success.jsp** and **login-error.jsp** files acts as view components.
* **web.xml** file for mapping the servlet.

*File: index.jsp*

1. <form action="ControllerServlet" method="post">
2. Name:<input type="text" name="name"><br>
3. Password:<input type="password" name="password"><br>
4. <input type="submit" value="login">
5. </form>

*File: ControllerServlet*

1. **package** com.javatpoint;
2. **import** java.io.IOException;
3. **import** java.io.PrintWriter;
4. **import** javax.servlet.RequestDispatcher;
5. **import** javax.servlet.ServletException;
6. **import** javax.servlet.http.HttpServlet;
7. **import** javax.servlet.http.HttpServletRequest;
8. **import** javax.servlet.http.HttpServletResponse;
9. **public** **class** ControllerServlet **extends** HttpServlet {
10. **protected** **void** doPost(HttpServletRequest request, HttpServletResponse response)
11. **throws** ServletException, IOException {
12. response.setContentType("text/html");
13. PrintWriter out=response.getWriter();
15. String name=request.getParameter("name");
16. String password=request.getParameter("password");
18. LoginBean bean=**new** LoginBean();
19. bean.setName(name);
20. bean.setPassword(password);
21. request.setAttribute("bean",bean);
23. **boolean** status=bean.validate();
25. **if**(status){
26. RequestDispatcher rd=request.getRequestDispatcher("login-success.jsp");
27. rd.forward(request, response);
28. }
29. **else**{
30. RequestDispatcher rd=request.getRequestDispatcher("login-error.jsp");
31. rd.forward(request, response);
32. }
34. }
36. @Override
37. **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp)
38. **throws** ServletException, IOException {
39. doPost(req, resp);
40. }
41. }

*File: LoginBean.java*

1. **package** com.javatpoint;
2. **public** **class** LoginBean {
3. **private** String name,password;
5. **public** String getName() {
6. **return** name;
7. }
8. **public** **void** setName(String name) {
9. **this**.name = name;
10. }
11. **public** String getPassword() {
12. **return** password;
13. }
14. **public** **void** setPassword(String password) {
15. **this**.password = password;
16. }
17. **public** **boolean** validate(){
18. **if**(password.equals("admin")){
19. **return** **true**;
20. }
21. **else**{
22. **return** **false**;
23. }
24. }
25. }

*File: login-success.jsp*

1. <%@page **import**="com.javatpoint.LoginBean"%>
3. <p>You are successfully logged in!</p>
4. <%
5. LoginBean bean=(LoginBean)request.getAttribute("bean");
6. out.print("Welcome, "+bean.getName());
7. %>

*File: login-error.jsp*

1. <p>Sorry! username or password error</p>
2. <%@ include file="index.jsp" %>

*File: web.xml*

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<web-app** xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"
4. xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"
5. id="WebApp\_ID" version="3.0"**>**
7. **<servlet>**
8. **<servlet-name>**s1**</servlet-name>**
9. **<servlet-class>**com.javatpoint.ControllerServlet**</servlet-class>**
10. **</servlet>**
11. **<servlet-mapping>**
12. **<servlet-name>**s1**</servlet-name>**
13. **<url-pattern>**/ControllerServlet**</url-pattern>**
14. **</servlet-mapping>**
15. **</web-app>**