Introduction to JSP

JSP technology is used to create dynamic web applications. JSP pages are easier to maintain than a Servlet. JSP pages are opposite of Servlets as a servlet adds HTML code inside Java code, while JSP adds Java code inside HTML using JSP tags. Everything a Servlet can do, a JSP page can also do it.

JSP enables us to write HTML pages containing tags, inside which we can include powerful Java programs. Using JSP, one can easily separate Presentation and Business logic.

Dis Advantages with Servlet

Servlet is a mixture of java skills and web related HTML skills, because you have to write the business logic in java and for presentation you should write the HTML, so the role-based development is missing in pure servlet. The developer who is writing servlet should know java and HTML both.

The servlet technology requires more steps to develop, servlet require too long time for development.

If your application is built on using only servlet technology, it is very difficult for enhancement and bug fixing.

1. Servlets of web application requires strong knowledge of Java.
2. Servlets are very difficult for Non-Java Programmers to understand and carry out.
3. We know that, Servlet is a picture of both Presentation logic (HTML) and Business Logic (Java). At the time of development of Servlet by using both of the above (Presentation and Business Logic), it may become imbalance, because a Servlet developer can't be good in both Presentation logic and Business logic.
4. Servlet never provides separation between or clarity between Presentation Logic and Business Logic. So that servlets do not give Parallel Development.
5. If we do any changes in a servlet, then we need to do Re-deployment process i.e., Servlets modifications requires redeployment, which is one of the time-consuming processes.
6. If we develop any web application with servlets, then it is mandatory for the web application developer to configure web-application configuration file (Deployment descriptor- web.xml).
7. Servlets do not offer any implicit object [Implicit objects generally provided by containers during the dynamic program execution].
8. Servlets do not contain a facility called Custom Tags Development.
9. Servlets don't provide Global/Implicit exception handling facility.

Advantages of JSP over Servlet

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.

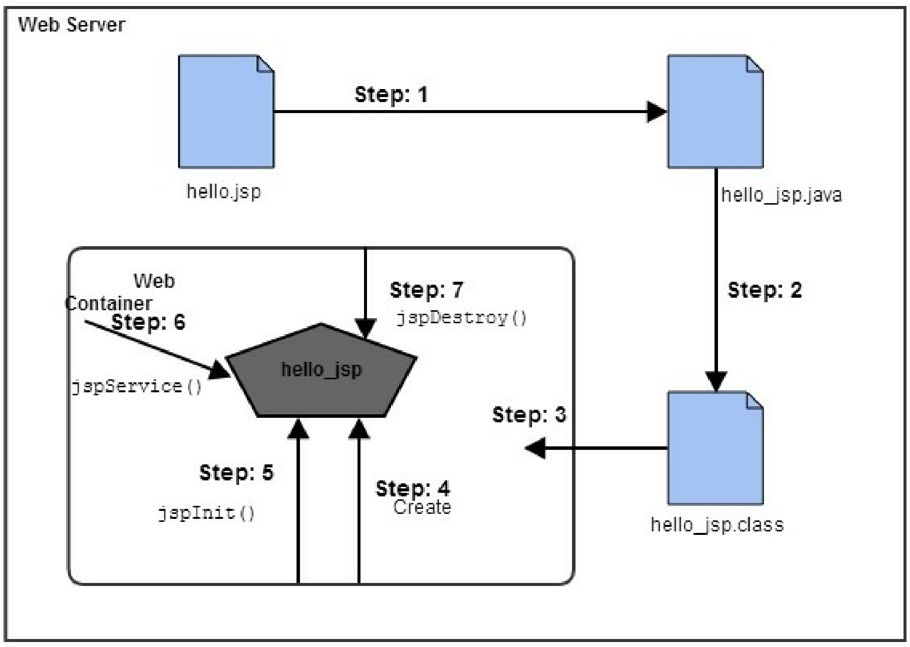
Lifecycle of JSP

A JSP page is converted into Servlet in order to service requests. The translation of a JSP page to a Servlet is called Lifecycle of JSP. JSP Lifecycle is exactly same as the Servlet Lifecycle, with one additional first step, which is, translation of JSP code to Servlet code.

Following are the JSP Lifecycle steps:

1. Translation of JSP Page
2. Compilation of JSP Page
3. Class loading (the class loader loads class file)
4. Instantiation (Object of the Generated Servlet is created).
5. Initialization (the container invokes jspInit() method).
6. Request processing (the container invokes _jspService() method).
7. Destroy (the container invokes jspDestroy() method).

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



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.

```
<html>

<body>

<% out.print(2*5); %>

</body>

</html>
```

How to run a simple JSP Page?

Follow the following steps to execute this JSP page:

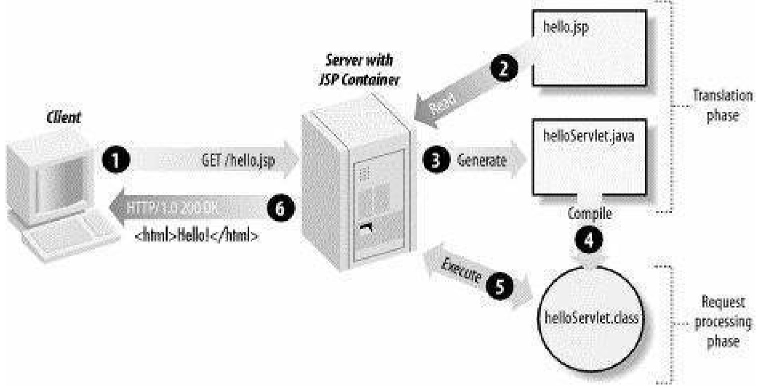
1. Start the server
2. Put the JSP file in a folder and deploy on the server
3. Visit the browser by the URL http://localhost:portno/contextRoot/jspfile, for example, http://localhost:8080/myapplication/index.jsp

JSP Processing

The following steps explain how the web server creates the Webpage using JSP −

- As with a normal page, your browser sends an HTTP request to the web server.
- The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with .jsp instead of .html.
- The JSP engine loads the JSP page from disk and converts it into servlet content. This conversion is very simple in which all template text is converted to println () statements and all JSP elements are converted to Java code. This code implements the corresponding dynamic behavior of the page.
- The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
- A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format. The output is further passed on to the web server by the servlet engine inside an HTTP response.
- The web server forwards the HTTP response to your browser in terms of static HTML content.
- Finally, the web browser handles the dynamically-generated HTML page inside the HTTP response exactly as if it were a static page.

All the above-mentioned steps can be seen in the following diagram



The Anatomy of a JSP Page

A JSP page is made up of template data and/or JSP elements.

- Template data (HTML/Plain Text)
- JSP Elements

JSP Elements

JSP elements are called the JSP tags on JSP page. On the JSP page mainly three groups of JSP elements are used:

- **JSP Scripting Elements**
- **JSP Directive Elements**
- **JSP Standard Action Elements**

JSP Scripting elements

The scripting elements provide the ability to insert java code inside the jsp. There are three types of scripting elements:

- scriptlet tag
- expression tag
- declaration tag

JSP scriptlet tag

A scriptlet tag is used to execute java source code in JSP.

Syntax:

```
<% java source code %>
```

Example

In this example, we are displaying a welcome message.

```
<html>

<body>

<% out.print("welcome to jsp"); %>

</body>

</html>
```

Example of JSP scriptlet tag that prints the user's name

In this example, we have created two files index.html and welcome.jsp. The index.html file gets the username from the user and the welcome.jsp file prints the username with the welcome message.

File: index.html

```
<html>

<body>

<form action="welcome.jsp">

<input type="text" name="uname">
```

```
<input type="submit" value="go"><br/>

</form>

</body>

</html>
```

File: welcome.jsp

```
<html>

<body>

<%

String name=request.getParameter("uname");

out.print("welcome "+name);

%>

</form>

</body>

</html>
```

JSP expression tag

The code placed within **JSP expression tag** is written to the output stream of the response. So you need not write out.print() to write data. It is mainly used to print the values of variable or method.

Syntax

```
<%= statement %>
```

Example

In this example of jsp expression tag, we are simply displaying a welcome message.

```
<html>

<body>

<%= "welcome to jsp" %>

</body>

</html>
```

Note: Do not end your statement with semicolon in case of expression tag.

Example of JSP expression tag that prints current time

To display the current time, we have used the getTime() method of Calendar class. The getTime() is an instance method of Calendar class, so we have called it after getting the instance of Calendar class by the getInstance() method.

index.jsp

```
<html>

<body>

Current Time: <%= java. util.Calendar.getInstance().getTime() %>

</body>

</html>
```

Example of JSP expression tag that prints the user’s name

In this example, we are printing the username using the expression tag. The index.html file gets the username and sends the request to the welcome.jsp file, which displays the username.

File: index.html

```
<html>

<body>

<form action="welcome.jsp">

<input type="text" name="uname"><br/>

<input type="submit" value="go">

</form>

</body>

</html>
```

File: welcome.jsp

```
<html>

<body>

<%= "Welcome "+request.getParameter("uname") %>

</body>

</html>
```

JSP Declaration Tag

The **JSP declaration tag** is used to declare fields and methods.

The code written inside the jsp declaration tag is placed outside the service () method of auto generated servlet.

So, it doesn't get memory at each request.

Syntax

```
<%!  field or method declaration %>
```

Example of JSP declaration tag that declares field

In this example of JSP declaration tag, we are declaring the field and printing the value of the declared field using the jsp expression tag.

index.jsp

```
<html>

<body>

<%! int data=50; %>

<%= "Value of the variable is:"+data %>

</body>

</html>
```

Example of JSP declaration tag that declares method

In this example of JSP declaration tag, we are defining the method which returns the cube of given number and calling this method from the jsp expression tag. But we can also use jsp scriptlet tag to call the declared method.

index.jsp

```
<html>

<body>

<%!
```

```
int cube (int n) {

return n*n*n*;

}

%>

<%= "Cube of 3 is:"+cube (3) %>

</body>

</html>
```

Difference between JSP Scriptlet tag and Declaration tag

Jsp Scriptlet Tag

1. The jsp scriptlet tag can only declare variables not methods.
2. The declaration of scriptlet tag is placed inside the _jspService () method.

Jsp Declaration Tag

1. The jsp declaration tag can declare variables as well as methods.
2. The declaration of jsp declaration tag is placed outside the _jspService () method.

Comments in JSP

JSP comment marks to text or statements that the JSP container should ignore. A JSP comment is useful when you want to hide or "comment out", a part of your JSP page.

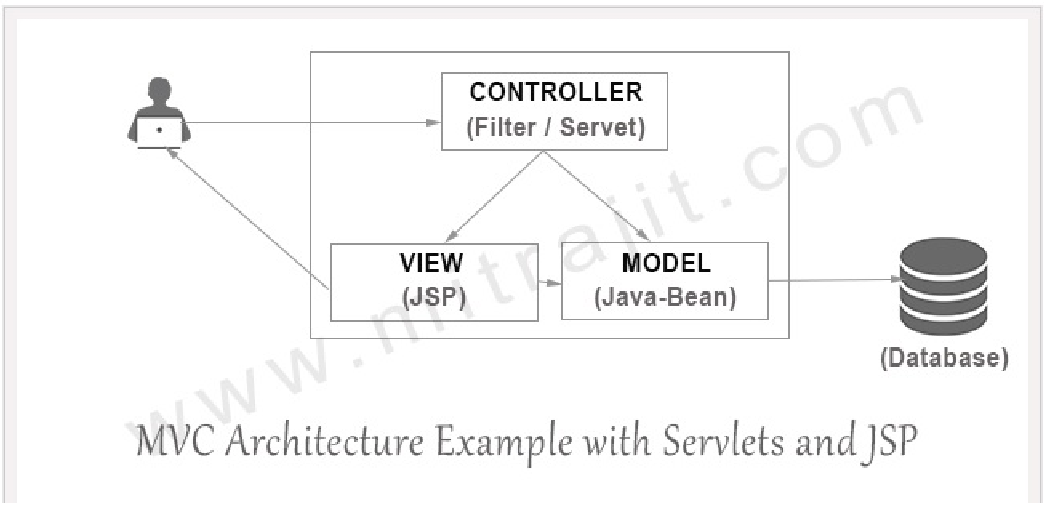
Following is the syntax of the JSP comments −

<% -- This is JSP comment -- %>

MVC Architecture with Servlets and JSP

Model View Controller (MVC) is a software design architectural pattern for developing the web application. MVC pattern separates business logic, presentation, and data. MVC pattern has the following three parts —

1. **Model –** This level is responsible for all kinds of business logic operations. It has the classes which have the connection with the database.
2. **View –** This is the presentation layer which consists of HTML, JSP pages. This layer is responsible for displaying the application's output to the end users.
3. **Controller –** This layer handles the HTTP requests and sends to the appropriate Model layer for data processing, and once the data are processed and sent back to the controller and then displayed on the View layer.



JSP Implicit Objects

JSP container makes some Java objects available to the JSP page. No specific declaration or initialization is required within the JSP page. These objects are called implicit objects. List of the JSP implicit objects is included below

Object	Type
Out	JspWriter
Request	HttpServletRequest

Response	HttpServletResponse
Config	ServletConfig
Application	ServletContext
Session	HttpSession
pageContext	PageContext
Page	Object
Exception	Throwable

out implicit object

- Out is one of the implicit objects to write the data to the buffer and send output to the client in response
- Out object allows us to access the servlet's output stream
- Out is object of javax.servlet.jsp.jspWriter class
- While working with servlet, we need printwriter object

Example:

```
<html>

<head>

</head>

<body>

<% int num1=10; int num2=20;

out.println("num1 is " +num1);

out.println("num2 is "+num2);

%>

</body>

</html>
```

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.

index.html

```
<form action="welcome.jsp">

<input type="text" name="uname">

<input type="submit" value="go"><br/>

</form>
```

welcome.jsp

```
<%

String name=request.getParameter("uname");

out.print("welcome "+name);

%>
```

response implicit object

- "Response" is an instance of class which implements HttpServletResponse interface
- Container generates this object and passes to _jspservice() method as parameter
- "Response object" will be created by the container for each request.
- It represents the response that can be given to the client
- The response implicit object is used to content type, add cookie and redirect to response page

index.html

```
<form action="welcome.jsp">

<input type="text" name="uname">

<input type="submit" value="go"><br/>

</form>
```

welcome.jsp

```
<%

response.sendRedirect("http://www.google.com");

%>
```

config implicit object

- "Config" is of the type java.servlet.servletConfig
- It is created by the container for each jsp page
- It is used to get the initialization parameter in web.xml

index.html

```
<form action="welcome">

<input type="text" name="uname">

<input type="submit" value="go"><br/>

</form>
```

web.xml file

```
<web-app>

<servlet>

<servlet-name>s1</servlet-name>

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



<init-param>

<param-name>dname</param-name>

<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>

</init-param>



</servlet>



<servlet-mapping>

<servlet-name>s1</servlet-name>

<url-pattern>/welcome</url-pattern>

</servlet-mapping>



</web-app>
```

welcome.jsp

```
<%

out.print("Welcome "+request.getParameter("uname"));
```

```
String driver=config.getInitParameter("dname");

out.print("driver name is="+driver);

%>
```

application implicit object

- Application object is an instance of javax.servlet.ServletContext and it is used to get the context information and attributes in JSP.
- Application object is created by container one per application, when the application gets deployed.
- ServletContext object contains a set of methods which are used to interact with the servlet container. We can find information about the servlet container.

index.html

```
<form action="welcome">

<input type="text" name="uname">

<input type="submit" value="go"><br/>

</form>
```

web.xml file

```
<web-app>



<servlet>

<servlet-name>s1</servlet-name>

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

</servlet>



<servlet-mapping>

<servlet-name>s1</servlet-name>

<url-pattern>/welcome</url-pattern>

</servlet-mapping>



<context-param>

<param-name>dname</param-name>

<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>

</context-param>



</web-app>
```

welcome.jsp

```
<%

out.print("Welcome "+request.getParameter("uname"));

String driver=application.getInitParameter("dname");

out.print("driver name is="+driver);

%>
```

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.

index.html

```
<html>

<body>

<form action="welcome.jsp">

<input type="text" name="uname">

<input type="submit" value="go"><br/>

</form>

</body>

</html>
```

welcome.jsp

```
<html>

<body>

<%

String name=request.getParameter("uname");

out.print("Welcome "+name);

session.setAttribute("user",name);

<a href="second.jsp">second jsp page</a>

%>

</body>

</html>
```

second.jsp

```
<html>

<body>

<%

String name=(String)session.getAttribute("user");

out.print("Hello "+name);

%>

</body>

</html>
```

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.

index.html

```
<html>

<body>

<form action="welcome.jsp">

<input type="text" name="uname">

<input type="submit" value="go"><br/>

</form>

</body>

</html>
```

welcome.jsp

```
<html>

<body>

<%


String name=request.getParameter("uname");

out.print("Welcome "+name);


pageContext.setAttribute("user", name,PageContext.SESSION_SCOPE);


<a href="second.jsp">second jsp page</a>


%>

</body>

</html>
```

second.jsp

```
<html>

<body>

<%

String name=(String)pageContext.getAttribute("user", PageContext.SESSION_SCOPE);

out.print("Hello "+name);

%>

</body>

</html>
```

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.

This object is an actual reference to the instance of the page. It can be thought of as an object that represents the entire JSP page.

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"); %>

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.

JSP directives

- Directives control the processing of an entire JSP page. It gives directions to the server regarding processing of a page.
- Directive Tag gives special instruction to Web Container at the time of page translation.
- JSP directives are the messages to JSP container. They provide global information about an entire JSP page.

There are three types of directives:

1. Page directive
2. Include directive
3. Taglib directive

Syntax:

<%@ directive_name [attribute name= “value” attribute name= “value” ....] %>

Page directive

- It provides attributes that get applied to entire JSP page.
- It defines page dependent attributes, such as scripting language, error page, and buffering requirements.
- It is used to provide instructions to a container that pertains to current JSP page.

Syntax:

<%@ page attribute="value" %>

Attributes of JSP page directive

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

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:

```
<html>

<body>



<%@ page import="java.util.Date" %>

Today is: <%= new Date() %>



</body>

</html>
```

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:

```
<html>

<body>



<%@ page contentType=”text/html “%>

Today is: <%= new java.util.Date() %>

</body>

</html>
```

Info

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

```
<html>

<body>



<%@ page info="composed by CSE Students" %>

Today is: <%= new java.util.Date() %>



</body>

</html>
```

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

```
public String getServletInfo() {

  return "composed by CSE Students";

}
```

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).

This directive tells the container to merge the content of other external files with the current JSP during the translation phase. You may code the include directives anywhere in your JSP page.

Syntax

```
<%@ include file="resourceName" %>
```

Example

```
<html>

<body>



<%@ include file="header.html" %>



Today is: <%= java.util.Calendar.getInstance().getTime() %>



</body>

</html>
```

header.html

```
<html>

<body>



<h3> Today date and time is:</h3>

</body>

</html>
```

Taglib directive

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.

Syntax

```
<%@ taglib uri="uriofthetaglibrary" prefix="prefixoftaglibrary" %>
```

Example

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.

```
<html>

<body>



<%@ taglib uri="C/mydocuments/sample.tld" prefix="mytag" %>



<mytag:currentDate/>



</body>

</html>
```

Exception Handling in JSP

The exception is normally an object that is thrown at runtime. Exception Handling is the process to handle the runtime errors.

Example:

In this case, you must define and create a page to handle the exceptions, as in the error.jsp page. The pages where may occur exception, define the errorPage attribute of page directive, as in the process.jsp page.

There are 3 files:

- index.jsp for input values
- process.jsp for dividing the two numbers and displaying the result
- error.jsp for handling the exception

index.jsp

```
<form action="process.jsp">

No1:<input type="text" name="n1" /><br/><br/>

No2:<input type="text" name="n2" /><br/><br/>

<input type="submit" value="divide"/>

</form>
```

process.jsp

```
<%@ page errorPage="error.jsp" %>

<%



String num1=request.getParameter("n1");

String num2=request.getParameter("n2");



int a=Integer.parseInt(num1);

int b=Integer.parseInt(num2);

int c=a/b;

out.print("division of numbers is: "+c);



%>
```

error.jsp

```
<%@ page isErrorPage="true" %>

<h3>Sorry an exception occured!</h3>

Exception is: <%= exception %>
```

JSP Action Tags

The action tags are used to control the flow between pages and to use Java Bean. JSP provides a bunch of standard action tags that we can use for specific tasks such as working with java bean objects, including other resources, forward the request to another resource etc. The Jsp action tags are given below.

JSP Action	Description
jsp:include	To include a resource at runtime, can be HTML, JSP or any other file
jsp:useBean	To get the java bean object from given scope or to create a new object of java bean.
jsp:getProperty	To get the property of a java bean, used with jsp:useBean action.
jsp:setProperty	To set the property of a java bean object, used with jsp:useBean action.
jsp:forward	To forward the request to another resource.
jsp:plugin	To generate the browser-specific code that makes an OBJECT or EMBED tag for the Java plugin.
jsp:param	sets the parameter value. It is used in forward and include mostly.

jsp:forward action tag

The jsp:forward action tag is used to forward the request to another resource it may be jsp, html or another resource.

Syntax

```
<jsp:forward page="relativeURL | <%= expression %>"/>

(or)

<jsp:forward page="relativeURL | <%= expression %>">

<jsp:param name="parametername" value="parametervalue | <%=expression%>" />

</jsp:forward>
```

first.jsp

```
<html>

<head>

<title>Demo of JSP Param Action Tag</title>

</head>

<body>

<h3>JSP page: Demo Param along with forward</h3>

<jsp:forward page="second.jsp">

<jsp:param name ="date" value="20-05-2012" />

<jsp:param name ="time" value="10:15AM" />

<jsp:param name ="data" value="ABC" />

</jsp:forward>

</body>

</html>
```

Second.jsp

```
Date:<%= request.getParameter("date") %>

Time:<%= request.getParameter("time") %>

My Data:<%= request.getParameter("data") %>
```

<jsp:include> action tag

The jsp:include action tag is used to include the content of another resource it may be jsp, html or servlet.

The jsp include action tag includes the resource at request time so it is better for dynamic pages because there might be changes in future.

The jsp:include tag can be used to include static as well as dynamic pages.

Syntax

```
<jsp:include page="relativeURL | <%= expression %>"/>

(or)

<jsp:include page="relativeURL | <%= expression %>">

<jsp:param name="parametername" value="parametervalue | <%=expression%>" />

</jsp:include>
```

Example

index.jsp

```
<h2>this is index page</h2>
```

<jsp:include page="printdate.jsp" />

<h2>end section of index page</h2>

printdate.jsp

<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>

<jsp:useBean> action tag

The jsp:useBean action tag is used to locate or instantiate a bean class.

Syntax

<jsp:useBean id= "instanceName" scope= "page | request | session | application"

class= "packageName.className" />

Attributes and Usage of jsp:useBean action tag

1. **id:** is used to identify the bean in the specified scope.
2. **scope:** represents the scope of the bean. It may be page, request, session or application. The default scope is page.

- **page:** specifies that you can use this bean within the JSP page. The default scope is page.
- **request:** specifies that you can use this bean from any JSP page that processes the same request. It has wider scope than page.
- **session:** specifies that you can use this bean from any JSP page in the same session whether processes the same request or not. It has wider scope than request.
- **application:** specifies that you can use this bean from any JSP page in the same application. It has wider scope than session.

1. **class:** instantiates the specified bean class (i.e. creates an object of the bean class) but it must have no-arg or no constructor and must not be abstract.

<jsp:setProperty> Action Tag

This action tag is used to set the property of a Bean, while using this action tag, you may need to specify the Bean's unique name (it is nothing but the id value of useBean action tag).

Syntax:

<jsp:setProperty name="unique_name_to_identify_bean" property="property_name"

value=" property_value "/>

<jsp:getProperty> Action Tag

It is used to retrieve or fetch the value of Bean's property.

Syntax

<jsp:getProperty name="unique_name_to_identify_bean" property="property_name" />

Example of <jsp:useBean>, <jsp:setProperty> & <jsp:getProperty>:

Once Bean class is instantiated using above statement, you have to use **jsp:setProperty** and **jsp:getProperty** actions to use the bean's parameters.

EmployeeBeanTest.jsp

<html>

<head>

<title>JSP Page to show use of useBean action</title>

</head>

<body>

<h1>Demo: Action</h1>

<jsp:useBean id="student" class="javabeansample.StudentBean"/>

<jsp:setProperty name="student" property="name" value="Raju"/>

```
<jsp:setProperty name="student" property="rollno" value="501"/>

<h1>

Name:<jsp:getProperty name="student" property="name"/><br>

Rollno:<jsp:getProperty name="student" property="rollno"/><br>

</h1>

</body>

</html>
```

StudentBean.java

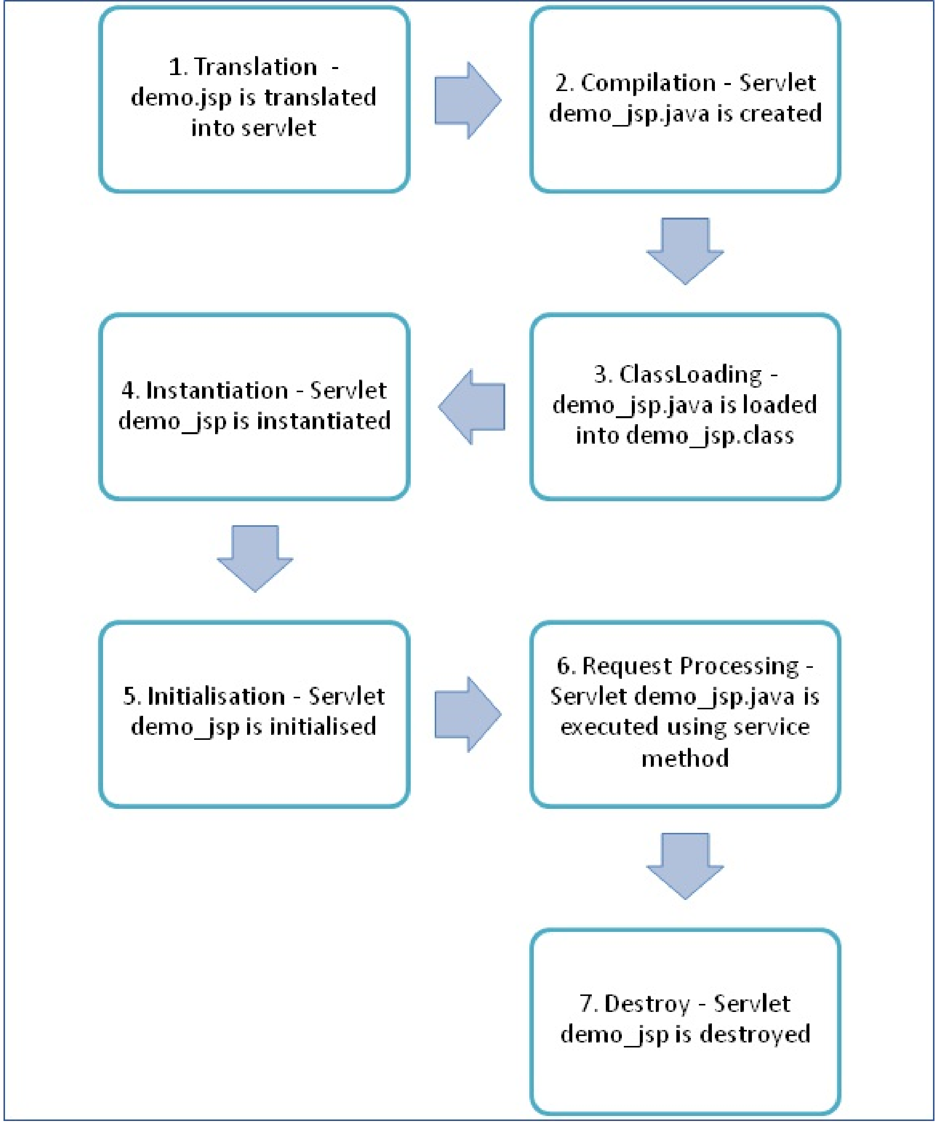
```
package javabeansample;

public class StudentBean {

public StudentBean() {

}

private String name;

private int rollno;

public void setName(String name)

{

this.name=name;

}

public String getName()

{

return name;

}

public void setRollno(int rollno)

{

this.rollno=rollno;

}

public int getRollno()

{

return rollno;

}

}
```

Compilation process of JSP Page:

Compilation process of JSP page involves three steps:

- Parsing of JSP
- Turning JSP into servlet
- Compiling the servlet

JSP Lifecycle is depicted in the below diagram.


1. Translation - demo.jsp is translated into servlet
2. Compilation - Servlet demo_jsp.java is created
3. ClassLoading - demo_jsp.java is loaded into demo_jsp.class
4. Instantiation - Servlet demo_jsp is instantiated
5. Initialisation - Servlet demo_jsp is initialised
6. Request Processing - Servlet demo_jsp.java is executed using service method
7. Destroy - Servlet demo_jsp is destroyed