JSP Basics

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 Tag etc.

Advantage of JSP over Servlet

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

1) Extension to Servlet

JSP technology is the extension to servlet technology. We can use all the features of 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.

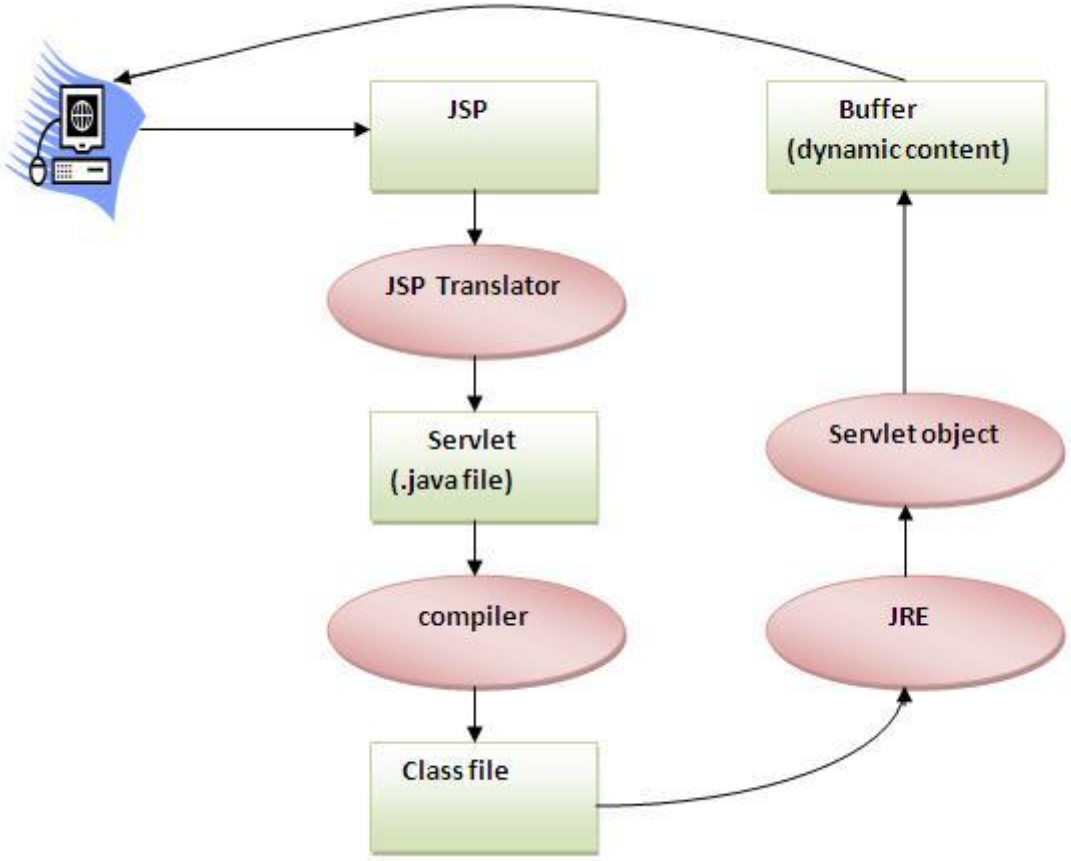
4) Less code than Servlet

In JSP, we can use a lot of tags such as action tags, jstl, custom tags etc. that reduces the code. Moreover, we can use EL, implicit objects etc.

Life cycle of a JSP Page

The JSP pages follows these phases:

- Translation of JSP Page
- Compilation of JSP Page
- Classloading (class file is loaded by the classloader)
- Instantiation (Object of the Generated Servlet is created).
- Initialization (jspInit() method is invoked by the container).
- Reqeust processing (_jspService() method is invoked by the container).
- Destroy (jspDestroy() method is invoked by the container).



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 webserver that is responsible to translate the JSP page into servlet. Afterthat Servlet page is compiled by the compiler and gets converted into the class file. Moreover, all the processes that happens in servlet is 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 save 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, here we are using the scriptlet tag to put java code in the JSP page. We will learn scriptlet tag later.

```
<html>
<body>
<% out.print(“welcome to JSP”); %>
</body>
</html>
```

It will print **welcome to JSP** on the browser.

How to run a simple JSP Page?

Follow the following steps to execute this JSP page:

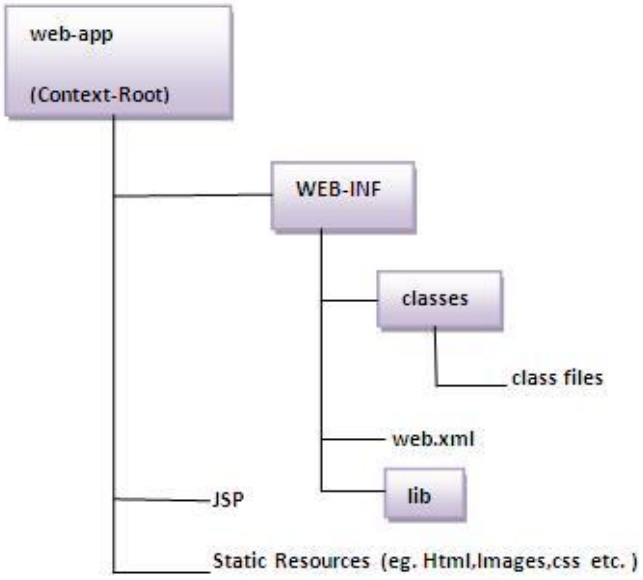
- Start the server
- put the jsp file in a folder and deploy on the server
- visit the browser by the url http://localhost:8080/folder/programname.jsp

Do I need to follow 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.But if you are using bean class, Servlet or tld file then directory structure is required.

Directory structure of JSP

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



JSP Scriptlet tag (Scripting elements)

In JSP, java code can be written inside the jsp page using the scriptlet tag. Let's see what the scripting elements are first.

JSP Scripting elements

The scripting elements provides 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 is as follows:

<% java source code %>

Example of JSP scriptlet tag

In this example, we are displaying a welcome message.

index.html

```
<html>
<body>
<% out.print("welcome to jsp"); %>
</body>
</html>
```

Example of JSP scriptlet tag that prints the user name and password

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

index.html

```
<html>
<body>
<form action="welcome.jsp">
Enter Username: <input type="text" name="uname"> <br>
Enter Password: <input type="password" name="pwd"> <br>
<input type="submit" >
<input type="reset" >
</form>
</body>
</html>
```

welcome.jsp

```
<%
String name=request.getParameter("uname");
String pass=request.getParameter("pwd");
out.print("welcome "+name);
```

```
out.println("password" + pass);

%>
```

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 of JSP expression tag

<%= statement %**>**

In expression tag statement should not end with semicolon.

Example of JSP expression tag

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

```
<html>

<body>

<%= "welcome to jsp" %>

</body>

</html>
```

Example of JSP expression tag that prints the user name and password

index.html

```
<html>

<body>

<form action="welcome.jsp">

Enter Username: <input type="text" name="uname"> <br>

Enter Password: <input type="password" name="pwd"> <br>

<input type="submit" >

<input type="reset" >

</form>

</body>

</html>
```

welcome.jsp

```
<%

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

String pass=request.getParameter("pwd");

%>
```

<%= "welcome" + name %>

<%= "password" + pass %>

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

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 of JSP declaration tag

The syntax of the declaration tag is as follows:

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

Difference between JSP Scriptlet tag and Declaration tag

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=10; %>

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

Jsp Scriptlet Tag	Jsp Declaration Tag
The jsp scriptlet tag can only declare variables not methods.	The jsp declaration tag can declare variables as well as methods.
The declaration of scriptlet tag is placed inside the _jspService() method.	The declaration of jsp declaration tag is placed outside the _jspService() method.

</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

```
<%!

int cube(int n)

{

return n*n*n*;

}

%>

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

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:

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

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

2) 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

```
<html>
<body>
<form action="welcome.jsp">
Enter Username: <input type="text" name="uname"> <br>
Enter Password: <input type="password" name="pwd"> <br>
<input type="submit" >
<input type="reset" >
</form>
</body>
</html>
```

welcome.jsp

```
<%
String name=request.getParameter("uname");
String pass=request.getParameter("pwd");
```

```
out.print("welcome "+name);

out.println("password" + pass);

%>
```

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

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

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

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

</form>
```

welcome.jsp

```
<%

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

%>
```

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

```
<form action="welcome">

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

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

</form>
```

web.xml file

```
<web-app>
```

```
  <servlet>
      <servlet-name>myservlet</servlet-name>
      <jsp-file>/welcome.jsp</jsp-file>
     <init-param>
          <param-name>student</param-name>
          <param-value>Rajesh</param-value>
     </init-param>
</servlet>
<servlet-mapping>
     <servlet-name>myservlet</servlet-name>
     <url-pattern>/welcome</url-pattern>
</servlet-mapping>
</web-app>
```

welcome.jsp

```
<%
String name = request.getParameter("uname");
out.print("Welcome "+ name);
String std=config.getInitParameter("student");
out.print("student name is="+std);
%>
```

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

```
<form action="welcome">
<input type="text" name="uname">
<input type="submit">
</form>
```

web.xml file

```
<web-app>
    <servlet>
        <servlet-name>myservlet</servlet-name>
        <jsp-file>/welcome.jsp</jsp-file>
    </servlet>
    <servlet-mapping>
        <servlet-name>myservlet</servlet-name>
        <url-pattern>/welcome</url-pattern>
    </servlet-mapping>
    <context-param>
        <param-name>student</param-name>
        <param-value>Rahul</param-value>
    </context-param>
</web-app>
```

welcome.jsp

```
<%
String name = request.getParameter("uname");
out.print("Welcome "+ name);
String std=config.getInitParameter("student");
out.print("student name is="+std);
%>
```

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.

Syntax:

session.setAttribute(“name”,”value”);

session.getAttribute(“name”);

session.removeAttribute(“name”);

Example:

Index.html

```
<html>
<body>
<form action="login.jsp">
Enter User Name:<input type="text" name="uname"> <br>
```

```
Enter Password: <input type="password" name="pwd"><br>
<input type="submit">
<input type="reset">
</form>
</body>
</html>
```

Links.html

```
<a href="index.html">Login</a>|
<a href="profile.jsp">Profle</a>|
<a href="logout.jsp">Logout</a>
<hr>
```

Login.jsp

```
<jsp:include page="links.html" />
</br>
<%
   String name=request.getParameter("uname");
   String pass=request.getParameter("pwd");

   String un = "lbrce";
   String ps = "lbrce123";

   if(name.equals(un) && pass.equals(ps))
   {
     session.setAttribute("user",name);
     out.println("welcome:" + name);
  }
  else
  {
      out.println("invalid user name & password");
%>
      <jsp:include page="index.html" />
<%
   }
%>
```

Profile.jsp

```
<jsp:include page="links.html" />
<br>
<%

  String uname = (String)session.getAttribute("user");
  if(uname != null)
  {
       out.println("welcome" + uname);
       out.println(" this is your profile");
  }
  else
 {
      out.println("please login first");
%>
      <jsp:include page="index.html" />
```

```
<%
  }
%>
```

Logout.jsp

```
<jsp:include page="links.html" />

<br>

<%

  session.removeAttribute("user");

  out.println("logout successfully");

%>
```

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.

Syntax:

page scope: (default scope)

pageContext.setAttribute(“name”,”value”, PageContext.PAGE_SCOPE);

pageContext.getAttribute(“name”);

PageContext.removeAttribute(“name”);

session scope:

pageContext.setAttribute(“name”,”value”, PageContext.SESSION_SCOPE);

pageContext.getAttribute(“name”);

PageContext.removeAttribute(“name”);

request scope:

pageContext.setAttribute(“name”,”value”, PageContext.REQUEST_SCOPE);

pageContext.getAttribute(“name”);

PageContext.removeAttribute(“name”);

application scope:

pageContext.setAttribute(“name”,”value”, PageContext.APPLICATION_SCOPE);

pageContext.getAttribute(“name”);

PageContext.removeAttribute(“name”);

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:**

Index.html

```
<form name="myform" action="welcome.jsp">
Enter number1: <input type="text" name="num1"><br>
Enter your number2: <input type="text" name="num2"><br>
<input type="submit">
<input type="reset">
</form>
```

Welcome.jsp

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

<%

int x = Integer.parseInt(request.getParameter("num1"));

int y = Integer.parseInt(request.getParameter("num2"));

int div = x/y;

out.println("the division is:" + div);

%>
```

Error.jsp

```
<%@ page isErrorPage="true" %>
The exception is: <%= exception %>
```

JSP program for Static Login Validation

Index.html

```
<form name="myform" action="welcome.jsp">
Enter your Name: <input type="text" name="uname"><br>
Enter your Password: <input type="password" name="pwd"><br>
<input type="submit">
<input type="reset">
</form>
```

Welcome.jsp

```
<%
String un = request.getParameter("uname");
String ps = request.getParameter("pwd");

String user = "lbrce";
String pass = "lbrce123";

if(un.equals(user) && ps.equals(pass))
{
  out.println(“welcome” + user);
}
else
{
  out.println(“invalid username & password”);
}

%>
```

JSP program to insert data from HTML form to oracle database

Index.html

```
<form name="myform" action="register.jsp">
enter username <input type="text" name="uname"> <br>
```

```
eter password <input type="password" name="pwd"> <br>
enter your mail<input type="mail" name="mail"><br>
choose your gender:<input type ="radio" name="gender" value="male">male
<input type ="radio" name="gender" value="female">female<br>
<input type="submit" >
<input type="reset">
</form>
```

Register.jsp

```
<%@ page import="java.sql.*" %>
<%
String un=request.getParameter("uname");
String ps=request.getParameter("pwd");
String email=request.getParameter("mail");
String gen=request.getParameter("gender");
try{
   oracle.jdbc.driver.OracleDriver d=new oracle.jdbc.driver.OracleDriver();
DriverManager.registerDriver(d);
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","cse","cse");
String qry="insert into student values(?,?,?,?)";
PreparedStatement pst=con.prepareStatement(qry);
pst.setString(1,un);
pst.setString(2,ps);
pst.setString(3,email);
pst.setString(4,gen);
int i=pst.executeUpdate();
out.println(i+"Row's inserted successully");
}
catch(Exception e)
{
 e.printStackTrace();
}
%>
```

JSP Program to display data from oracle database to web browser.

Display.jsp

```
<%@page import="java.sql.*"%>
<%
try{
  Class.forName("oracle.jdbc.driver.OracleDriver");
  Connection con=DriverManager.getConnection(("jdbc:oracle:thin:@localhost:1521:xe","cse","cse");
  Statement stmt=con.createStatement();
  String query="select * from student";
  ResultSet rs=stmt.executeQuery(query);
%>
<html>
<body bgcolor="pink">
<center>
```

```
<h1>Welcome to Student Details</h1>
<table border="5" bgcolor="white">
<th>
<tr>
<td colspan="5" align="center">Student Details</td></tr>
</th>
<tr>
<td>Student id</td>
<td>Name</td>
<td>Address</td>
<td>DOB</td>
</tr>
<%
while(rs.next())
{
%>
<tr>
<td><%=rs.getString(1)%></td>
<td><%=rs.getString(2)%></td>
<td><%=rs.getString(3)%></td>
<td><%=rs.getString(4)%></td>
</tr>
<%
}
%>
</table>
<br>
<br>
}
catch(Exception e)
{
  out.println(e.toString());
}
%>
```

JSP program to print multiplication table for the given table number with limit from HTML form

Index.html

```
<form name="myform" action="welcome.jsp">
Enter table number: <input type="text" name="table"><br>
Enter limit value: <input type="text" name="limit"><br>
<input type="submit">
<input type="reset">
</form>
```

Welcome.jsp

```
<%
int table = Integer.parseInt(request.getParameter("table"));
int limit = Integer.parseInt(request.getParameter("limit"));
out.println("<table border=1>");
for(int i=1; i<=limit; i++)
```

```
{
  out.println("<tr>");
  out.println("<td>"+table+"</td>");
  out.println("<td>*</td>");
  out.println("<td>"+i+"</td>");
  out.println("<td>=</td>");
  out.println("<td>"+table*i+"</td>");
  out.println("</tr>");
}
out.println("</table>");
%>
```

JSP directive elements

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

<%@ directive attribute="value" %>

1) JSP page directive

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

Syntax of JSP page directive

<%@ 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 of import attribute

```
<html>
    <body>

    <%@ page import="java.util.Date" %>
    Today is: <%= new Date() %>

    </body>
</html>
```

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

```
    //index.jsp
    <html>
    <body>

    <%@ page errorPage="myerrorpage.jsp" %>

    <%= 100/0 %>

    </body>
    </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

```
//myerrorpage.jsp
<html>
<body>

<%@ page isErrorPage="true" %>

Sorry an exception occured!<br/>
        The exception is: <%= exception %>

        </body>
</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

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

JSP 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. 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" %>
```

JSP Action Tags/ Elements

There are many JSP action tags or elements. Each JSP action tag is used to perform some specific tasks.

The action tags are used to control the flow between pages and to use Java Bean. The Jsp action tags are given below.

JSP Action Tags	Description
jsp:forward	forwards the request and response to another resource.
jsp:include	includes another resource.

jsp:useBean	creates or locates bean object.
jsp:setProperty	sets the value of property in bean object.
jsp:getProperty	prints the value of property of the bean.
jsp:plugin	embeds another components such as applet.
jsp:param	sets the parameter value. It is used in forward and include mostly.
jsp:fallback	can be used to print the message if plugin is working. It is used in jsp:plugin.

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 of jsp:forward action tag without parameter

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

Syntax of jsp:forward action tag with parameter

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

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="location of the page"/>

Advantage of jsp:include action tag

Code reusability : We can use a page many times such as including header and footer pages in all pages. So it saves a lot of time.

Difference between jsp include directive and include action

JSP include directive	JSP include action
includes resource at translation time.	includes resource at request time.
better for static pages.	better for dynamic pages.
includes the original content in the generated servlet.	calls the include method.

jsp:useBean action tag

The jsp:useBean action tag is used to locate or instantiate a bean class. If bean object of the Bean class is already created, it doesn't create the bean depending on the scope. But if object of bean is not created, it instantiates the bean.

Syntax of jsp:useBean action tag

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

class= "packageName.className" beanName="packageName.className”>

</jsp:useBean>

jsp:setProperty and jsp:getProperty action tags

The setProperty and getProperty action tags are used for developing web application with Java Bean. In web devlopment, bean class is mostly used because it is a reusable software component that represents data.

The jsp:setProperty action tag sets a property value or values in a bean using the setter method.

Syntax of jsp:setProperty action tag

<jsp:setProperty ="instanceOfBean" property="propertyName" value="value of property”/>

jsp:getProperty action tag

The jsp:getProperty action tag returns the value of the property.

Syntax of jsp:getProperty action tag

<jsp:getProperty name="instanceOfBean" property="propertyName" />

Example:

Employee.java

```
package mypack;
public class Employee implements java.io.Serializable
{
   private int id;
   private String name;

   public Employee()
   {
   }
   public void setId(int id)
   {
        this.id=id;
   }
   public int getId()
   {
         return id;
    }
    public void setName(String name)
```

```
    {
        this.name=name;
     }
    public String getName()
    {
        return name;
     }

}
```

index.jsp

```
<jsp:useBean id="emp" class="mypack.Employee"/>

<jsp:setProperty name="emp" property="id" value="30" />
<jsp:setProperty name="emp" property="name" value="Venky" />

<h3> Employee Details</h3>

<h3> Emp id: <jsp:getProperty name="emp" property="id"/> </h3>
<h3> Emp name: <jsp:getProperty name="emp" property="name" /></h3>
```

jsp:plugin action element

embeds another components such as applet.

syntax:

```
<jsp:plugin type="applet    code="packagename.applet.class"      codebase="."/>
```

Example:

MyApplet.java

```
import java.applet.*;

import java.awt.*;

public class MyApplet extends Applet

{

  public void paint(Graphics g)

  {

     g.drawString(“hello world”,150,150);

 }

}
```

welcome.jsp

```
<jsp:plugin type="applet    code="MyApplet.class"      codebase="."/>
```

jsp:fallback action tag/element

can be used to print the message if plugin is not working. It is used in jsp:plugin.

syntax:

```
<jsp:plugin type="applet    code="packagename.applet.class"      codebase=".">
<jsp:fallback>your alternate message </jsp:fallback>
</jsp:plugin>
```

Example:

MyApplet.java

```
import java.applet.*;
import java.awt.*;
public class MyApplet extends Applet
{
  public void paint(Graphics g)
  { g.drawString("hello world",150,150);
  }
}
```

welcome.jsp

```
<jsp:plugin type="applet    code="MyApplet.class"      codebase=".">
<jsp:fallback>your alternate message </jsp:fallback>
</jsp:plugin>
```