**JSP ( Java Server Pages )**

It is a server side component.

It contains HTML and Java Programming.

In Jsp we can separate business logic ( java program ) from presentation logic (HTML).

It is a server-side Java Web technology to communicate between a Web client and Web server.

JSP is a Java technology used to develop dynamically generated Web pages using mostly HTML, XML etc.

JSP is **platform-independent** as it is written in Java and is used for developing Web based applications

JSP can make use of entire family of Java Technologies like **JDBC** (Java Database Connectivity), **JMS** (Java Messaging Service) and **EJB** (Enterprise Java Beans) etc.

JSP is a file with extension **.jsp**

JSP is an extension for Servlets.

A JSP page is text document containing **static data** (which never changes like company information) developed using **HTML**, **XML** etc. and **dynamic content** (which changes with each request of the client) developed using JSP elements.

One JSP file can be included in another for better **reusability**.

With JSP, interactive (between Web client and Web server) dynamic Web-site can be created.

JSP is a **higher-level** way (with much abstraction) of writing Servlets.

JSP are **interoperable** with Servlets (data can be exchanged).

Used much in **MVC** architecture where JSP represent a **View page** (and Servlet as a **Controller**).

JSP Is open source file. It can be modified at any time.

## What is JSP container?

Jps execution environment is known as **JSP engine** or **JSP container**. Similarly, a **Servlet container** provides environment for a Servlet to execute.

**In general, a container has the following responsibilities:**

1. Should provide **standard libraries** required for coding.
2. Should provide suitable **environment** for execution.
3. Should call **callback methods** at appropriate times for the smooth execution.
4. Should maintain the **life cycle** of the program

**JSP container responsibilities**:

The JSP container should be capable to translate JSP file to a Servlet file, compile the servlet file, execute the Servlet and send the output of execution to client as response. As internally, a JSP file is converted to Servlet, the JSP container should also come with a Servlet container.

**Servlet container responsibilities**:

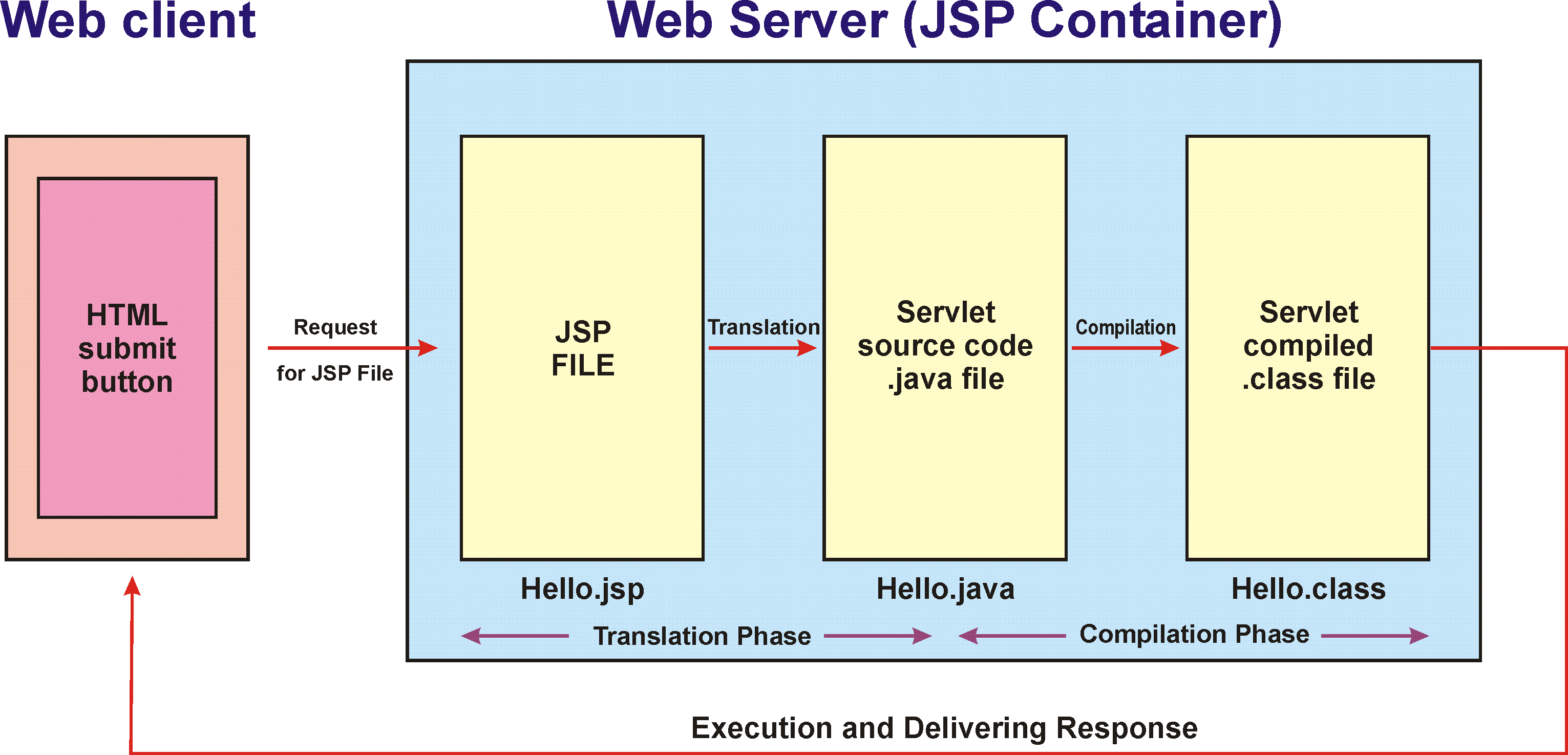
1. Should capable to load the Servlet **.class** file at the request of the client, execute the Servlet.
2. Accees the **database** server, if required.
3. Send **the output of execution to client as response.** When the response is delivered **close the connection** with the client.

**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).

## JSP Architecture



First time Client Request to JSP 🡪 Hello.jsp

🡪Jsp Translator

🡪Hello.java

🡪 java Compiler

🡪Hello.class

🡪JRE

🡪 Servlet instance

🡪 output of servlet sent to client.

Developing first example to display “Hello World”

Hello.jsp:

<body>

 Hello Every one see, this is my first JSP Program

<br> without any JSP tags printing  <b> Hello world </b>

</body>

Copy Hello.jsp into root folder>

Create war file and deploy on tomcat server.

Send request :

<http://localhost:8080/test/Hello.jsp>

test 🡪 is a .war file name.

**JSP Scriptlet tag (Scripting elements):**

In JSP, java code can be written inside the jsp page using the scriptlet tag.

## 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:

<%

Here we can write java source code

%>

Hello.jsp can be written as below also:

<html>

<body>

<%

out.print("welcome to jsp");

%>

</body>

</html>

## JSP Implicit objects and Scope

There are total **9 JSP implicit objects**

|  |  |  |
| --- | --- | --- |
| **Object** | **Description** | **Scope** |
| Out | javax.servlet.jsp.JspWriter  Represents the JspWriter object to the output stream | Page |
| Request | javax.servlet.http.HttpServletRequest  Represents the request objects that triggered the request | Request |
| response | javax.servlet.http.HttpServletResponse  Reprsents the response object that triggered the response | Response |
| Session | javax.servlet.http.HttpSession  Represents session object, if any, during the HTTP request | Session |
| Config | javax.servlet.ServletConfig  Represents ServletConfig for the JSP page | Page |
| Application | javax.servlet.ServletContext   |  |  | | --- | --- | | Represents the servlet context returned from a call to getServletConfig.getContext |  | | Application |
| Exception | java.lang.Throwable  Represents the uncaught error | Page |
| Page | javax.servlet.jsp.HttpJspPage  Represents this object for the instance of the page | Page |
| pageContext | javax.servlet.jsp.PageContext  Represents the page context for the JSP | Page |

The above objects are used in all the examples of JSP.

**1. out implicit object**

It is an implicit object of **javax.servlet.jsp.JspWriter**, as subclass of java.io.Writer. Programmer uses **out** object to send response to client.

**2. request implicit object**

This is an implicit object of **javax.servlet.http.HttpServletRequest** interface.  
This object is used by Programmer to extract (from JSP) the data send by client like user name and password and also header information, request query parameters, retrieving cookies from client etc

**3. response implicit object**

This is an implicit object of **javax.servlet.http.HttpServletResponse** interface. This is used to send response to client (via **out** object). Used for setting content type, sending cookies to client, redirect operations etc.

**4. session implicit object**

It is an implicit object of **javax.servlet.http.HttpSession** interface. It represents the communication with the client. The session object is used to store a single client’s **state**

**5. config implicit object**

This config represents an object of **javax.servlet.ServletConfig** interface. Used to retrieve configuration or **initialization particulars** of a jSP.

**6. application implicit object**

It is an implicit object of **javax.servlet.ServletContext** interface. This represents the global object wherein all the JSPs of an application can store and retrieve values. Using this object, all JSPs can communicate in between.

**7. exception implicit object**

It is an instance of class **java.lang.Throwable**. Used to generate error pages.

**8. page implicit object**

It is an instance of class **java.lang.Object**. This object represents the current JSP page. This objects takes lot of memory and thereby used rarely. It represents or used as a reference to the generated Servlet. Similar to this keyword

**9. pageContext implicit object**

It is an instance of **javax.servlet.jsp.PageContext** abstract class. It gets the all the page context information. Used to forward request to other sources with get and set methods. It can used as a reference to all other implicit objects.

**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.

<html>

<body>

<%="welcome to jsp"%>

</body>

</html>

**Another example:**

<html>

<body>

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

</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.

**Syntax :**

<%!  field or method declaration %>

**Test.jsp:**

<html>

<body>

<%!**int**data =20; %>

<%

out.println("Value of the variable is:"+data);

%>

</body>

</html>

Example of JSP **request** tag that prints the user name

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);

%>

</body>

</html>

**Using if statement:**

<html>

<body>

<%!**int** a=10;

**int** b=20;

%>

<%

**if**(a>b)

{

%>

<h1> A is greater </h1>

<%

}

**else**

{

%>

<h1> B is greater </h1>

<%

}

%>

</body>

</html>

Using for loop:

<html>

<body>

<%

**for**(**int** i = 0; i < 10; i++)

{

out.println("Iteration:" + i + "<br>");

}

%>

</body>

Example for implicit object out JSP uses to send data to client.

<html>

<body>

<%!**int** a=10;

**int** b=20;

%>

<%

out.println("Multiplication:"+a\*b);

%>

</body>

</html>

Using **response** object:

**UserName.html:**

<body>

<formaction=*"UsingResponse.jsp"*method=*"post"*>

Enter Your Name <inputtype=*"text"*name=*"name"*>

<inputtype=*"submit"*value=*"SEND"*>

</form>

</body>

UsingResponse.jsp:

<body>

<%

String str = request.getParameter("*name*");

**if**(str.equals("MNRAO"))

{

response.sendRedirect("Valid.html");

}

**else**

{

response.sendRedirect("Invalid.html");

}

%>

</body>

**Valid.html:**

<body text=*"green"*>

<h2>Your name is Valid</h2>

</body>

**Invalid.html:**

<body text=*"red"*>

<h2>Your name is Invalid</h2>

</body>

Using session object:

<body>

<%@page session=*"true"*%>

<% // setting some values to session object

session.setAttribute("mnrao1", "10");

session.setAttribute("mnrao2", "20");

session.setAttribute("mnrao3", "30");

%>

<%

out.println("Value of mnrao1: ");

out.println(session.getAttribute("mnrao1")); // prints 5

// to retrieve all attribute values at a time

java.util.Enumeration <String> e = session.getAttributeNames();

**while**(e.hasMoreElements())

{

String key = e.nextElement();

String value = (String) session.getAttribute(key);

out.println("<br>Key: " + key + " and value: " + value);

}

%>

</body>

Where implicit object exception JSP can be used?

The **exception object** is used to print the exception message at runtime in JSP file, or to say, exception object represents errors and exceptions. Let us create a small scenario.

The exception object cannot be used always and everywhere in the JSP code. It is specially used in combination with [**"errorPage" and "isErrorPage"**](http://way2java.com/jsp/jsp-exception-handling-errorpage-iserrorpage-exception/)**.**

**What are "errorPage" and "isErrorPage" in JSP?**

"**errorPage**" in JSP can generate error at runtime. When generated it can be written to another JSP file which accepts by declaring "**isErrorpage=true**".

**What is JSP directive?**

1. JSP directive **directs** or instructs the container how it should behave with a few aspects of JSP code processing.
2. Directives dictate the **processing** of the whole JSP file. As per the directive, the container processes the JSP file.
3. JSP directive provides **special processing information** to the container.

**Syntax of a directive:**

**<%@ directiveName attributeName="value assigned" %>**

There are three types of directives:

* page directive
* include directive
* taglib directive

**JSP page directive**

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

**Syntax:**

<%@ 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:**  <html>  <body>  <%@page import=*"java.util.Date"*%>  Today is: <%=**new** Date() %>  </body>  </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".  <html>  <body>  <%@page contentType=*application/msword*%>  Today is: <%=**new** java.util.Date() %>  </body>  </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.  <html>  <body>  <%@page info=*"composed by MNRAO"* %>  Today is: <%=**new** java.util.Date() %>  </body>  </html>  The web container will create a method getServletInfo() in the resulting servlet.  public String getServletInfo() {  return "composed by MNRAO";  }  **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.  <html>  <body>  <%@page buffer=*"16kb"*%>  Today is: <%=**new** java.util.Date() %>  </body>  </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.  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:  public class SimplePage\_jsp extends HttpJspBase    implements SingleThreadModel{  .......  }  figure  **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.  **10)isErrorPage**  The isErrorPage attribute is used to declare that the current page is the error page. | |

InputNumbers.html:

<body>

<h2>Find Division value</h2>

<form action=*"division.jsp"*method=*"post"*><b>

Enter First Number <inputtype=*"text"*name=*"num1"*><br>

Enter Second Number <inputtype=*"text"*name=*"num2"*><br>

<input type=*"submit"*value=*"Division"*></b>

</form>

</body>

Division.jsp:

<body>

<%@page errorPage=*"receive.jsp"*%>

<%

**int** n1 = Integer.parseInt(request.getParameter("num1"));

**int** n2 = Integer.parseInt(request.getParameter("num2"));

%>

<h3>Your first number is <%= n1 %> and second number is <%= n2 %>. <br>

Division Value is <%= n1/n2 %></h3>.

</body>

Receive.jsp:

<body>

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

<h3><font color=*"red"*>

Sorry, Result cannot be printed. <br>

Cause of problem: <%= exception %></font></h3>

</body>

**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 is Code Reusability**

<html>

<body>

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

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

</body>

</html>

Include directive:

<%@include file=*"resourcename"* %>

include.jsp:

<html>

<body>

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

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

</body>

</html>

Header.html:

<!DOCTYPEhtml>

<html>

<h1>this is header file</h1>

<body>

</body>

</html>

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.

JSP Taglib directive syntax:

<%@taglib uri=*"uri of the taglibrary"*prefix=*"prefix of taglibrary"*%>

<html>

<body>

<%@taglib uri=*"tags"* prefix=*"mytag"*%>

<mytag:currentDate/>

</body>

</html>

Here tagsis folder inside root folder.

###### Another Example on include directive JSP:

The application consists of 2 JSP files, 1 HTML file and one client (HTML) program (to enter the name of employee). The application is meant to give an **Experience certificate** to an employee. For any employee, **header** (company name) and **footer** (company information) are common and never changes. Only body changes where employee name and years of experience comes.

**1. Experience.html:** This takes employee name from client and sends to Certificate.jsp.

**2. Certificate.jsp:** This includes the contents of Header.html and Footer.jsp.

**3. Header.html:** This gives company name and address as title to a certificate.

**4. Footer.jsp:** This gives the footer of date and company Web site name etc.

**experience.html**

<body>

<h2> Request for issue of Experience Certificate </h2>

<form method=*"get"* action=*"certificate.jsp"*>

<b> Enter Employee Name <input type=*"text"* name=*"*empName*"*></b>

<inputtype=*"submit"*>

</form>

</body>

*certificate.jsp*

<body>

<%@includefile=*"header.html"*%><hr>

<b>This is to certifiy that <%= request.getParameter("empName") %> is working with us for the last 3 years and his conduct and behaviour are good. He can discharge his duties in the stipulated time and can lead a team of developers. Presently, his status in the company is "Team Lead".</b>

<hr><%@include file=*"footer.jsp"*%><hr>

</body>

header.html:

<body>

<h2>VRNDA Solutions </h2>

<h3>Secunderabad<br>AndhraPradesh</h3>

</body>

# footer.jsp:

<%@pageimport=*"java.util.Date"*%>

<body>

<%=**new** Date() %><br>

Visit <ahref=*""*>www.vrnda.com </a>

</body>

**JSP Action Tags:**

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:element | used with XML elements |
| jsp:body | used with XML body element |
| jsp:text | to write text for pages used as template |
| jsp:attribute | used with attribute of an XML element |

**include action JSP Example**

*It is similar to include directive of syntax* ***<%@ include file="somefile" %>*** *. But they differ in how container treats or execute them.*

**SeparateState.jsp**

<body>

Today hot news are <br><br>

<jsp:include page=*"telangana.jsp"* flush=*"true"*/>

<br><br>Have a nice day

</body>

**Telangana.jsp:**

**<body>**

**The long ambition of Telangana people is fulfilled. <br>**

**A separate state of Telangana is created. <br>**

**Thanks for efforts of KCR.**

**<br> This is late news given on <%=new java.util.Date() %>.**

**</body>**

## Difference include directive vs include action JSP

Basically both are used to include HTML **static page** contents or a **dynamic JSP page** contents in another JSP file.

**The difference comes mainly in how the included file contents are placed in including file**. It is very important to know when to use what, else performance will be lost.

**1. First let us see how include directive (also known as "file include" as file attribute is used) works:**

**certificate.jsp**

<body>

<%@include file=*"footer.jsp"*%>

</body>

In the above code, **Certificate.jsp** is called as **including file** and **Footer.jsp** is known as **included file**. While the **Certificate.jsp** file is **translated** to Servlet source file (of .java extension), the Footer.jsp file contents are **pasted** as it is in Certificate.jsp where **<%@ inlcude .. %>** tag comes. Then, the Certificate.jsp is translated to Servlet source file, compiled to a .class file and then executed. For both JSP files, only **one Servlet file is created**.

What is the disadvantage here. If the code calls the statement **<%@ include file="Footer.jsp" %>** in two places in different parts of **Certificate.jsp** code, in both places the **Footer.jsp** content is pasted. It is okay, but see here what happens. By the time execution comes to the second statement, if the **Footer.jsp** contents are changed, the second statement also gives the old data only but not changed data of Footer.jsp because the content is pasted only once in two places. So, it is not suitable for included files that are very dynamic in nature and changing very often like hit count files, share value files etc. "**include directive**" is best suitable for static files (like advertisement banners, company headers and footers etc.) that do not change the data frequently. But by **performance it is faster**.

1. **Now let us see how include action (also known as "dynamic include" or "page include" as page attribute is used) works:**

**Certificate.jsp**

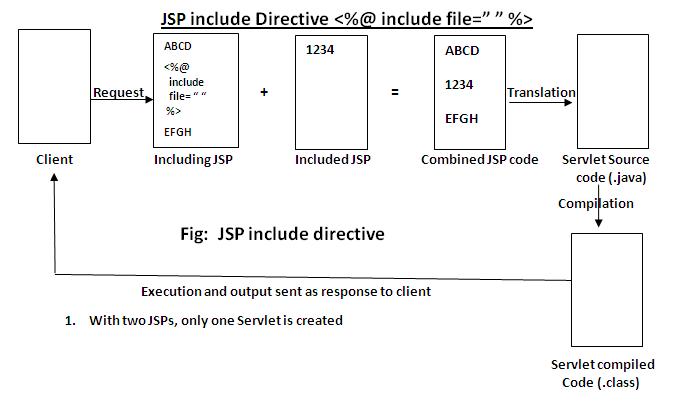
<body>

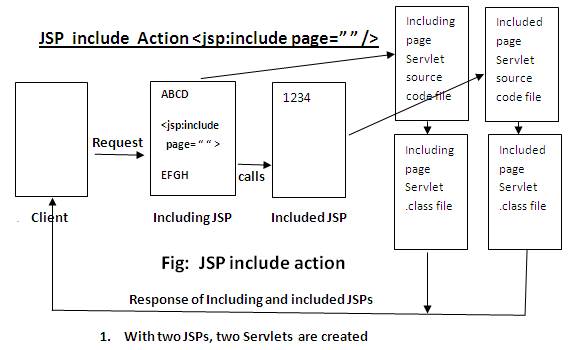
<jsp:include page=*"Footer.jsp"*/>

</body>

It is another style of including the contents of one JSP file in another. Here, **two Servlet source files**, one for each JSP, are created and compiled. **Two Servlet .class files are generated**. At the time of execution, the output of **Footer.jsp** is put in the output of **Certificate.jsp** and total output goes to client as response.

Now think of which is advantageous. It depends on the application needs. If the included file (Footer.jsp) changes very often, it is the best style to use. If the **Footer.jsp** changes, by the time control comes to the second statement, it is again compiled, executed and latest output is pasted in **Certificate.jsp**.





|  |  |  |
| --- | --- | --- |
| Property | include directive | include action |
| Code syntax | <%@ include file="Footer.jsp" %> | <jsp:include page="Footer.jsp" /> |
| Output binding | Static binding | Dynamic binding |
| include Names | Known as file include | Known as page include |
| Nature of files | Preferred when included page does not change frequently | Preferred when included page changes frequently |
| No. of Servlets created | Only one Servlet is created for multiple JSPs involved | Multiple Servlets are created, one for each JSP involved |
| Nature of Import | It is static import | It is dynamic import |
| Included file | Loaded at compile time | Loaded at runtime |
| Changes visible in included file | Not visible until next compilation | Visible in the next request |
| Attribute used | file attribute is used | page attribute is used |
| Execution action | Execution done at translation time | Execution done at request time |
| Performance | Faster | Comparatively slower |

## Forward Action JSP Example jsp:forward

The JSP **forward action** is similar to **JSP includ**e but with a small difference. In include directive, output of both JSP files goes to client. But in forward action, client gets **only one JSP file** content. **Client gets included file content only but not including file**.

The JSP forward action sends client request to another JSP file while terminating the current file (actually which is called by client).

**Example:**<jsp:forward page="Marks.jsp" flush="true" />

## **name.jsp**

<body>

Hello

<jsp:forward page=*"password.jsp"*/>

MNRAO

## </body>

## **Password.jsp:**

<body>

java

## </body>

## Here final output is only:

## Java

## forward action with Parameter passing Example JSP

We have seen earlier simple [<jsp:forward> action](http://way2java.com/jsp/jsp-forward-action-jspforward-with-example-screenshot/). Now let us see the same forward action with an extra feature of **parameters passing**. With **jsp:forward** action, parameters can also be passed to including page. Included page reads the parameters from the including page. start reading forward action with Parameter JSP.

###### Example on forward action with Parameter JSP

EmpParam.jsp

<body>

<jsp:forwardpage=*"mnrao.jsp"*>

<jsp:paramname=*"empName"*value=*"Nageswar Rao"*/>

<jsp:paramname=*"empOffice"*value=*"VRNDA"*/>

<jsp:paramname=*"place"*value=*"Hyderabad"*/>

</jsp:forward>

</body>

Mnrao.jsp:

<body>

<b> Employee Name: <%= request.getParameter("empName") %><br>

Employee Office: <%= request.getParameter("empOffice") %><br>

Place of work: <%= request.getParameter("place") %></b>

</body>

Include action vs forward action JSP

It is the same [differences between include and forward in RequestDispatcher](http://way2java.com/servlets/difference-between-include-and-forward-in-requestdispatcher/)

|  |  |  |
| --- | --- | --- |
| **Property** | **include action – jsp:include** | **forward action – jsp:forward** |
| Occurs | At translation time | At request time |
| Performance | Faster | Comparatively slower |
| No. of Servlets created | Only one | Multiple |
| Response Data sent | Includes both including and included JSPs | Goes only that of included JSP |
| Used when | When content of included file DOES NOT CHANGE frequently like advertisement banners | When content of included file CHANGES freuently like share price |
| Transfer of control | Execution control shifts temporarily from including file to included file. When included file is executed, the control transfers back to including file. | Shifts permanently from including file to included file. After the execution of included file, the control goes to client directly but not returned to including file. |
| Response to client | Goes from the same JSP which client requested. | Goes from a different JSP |
| Merge of response | Both responses of including and included files merged and sent client | No merging happens. Only included response goes to client |
| Extra activity | When control returned to including file, extra activity like including another JSP file can be done. | As control never returns, no extra activity can be done in including file. |

## Read Client Form Data in JSP

## There are 3 Ways:

**request** is an object of **HttpServletRequest** and is one of the [9 implicit objects](http://way2java.com/jsp/jsp-made-simple-jsp-list-of-9-implicit-objects-and-scope/), JSP supports. **request** object is used to retrieve client’s system (like IP address) information, client’s browser (header) information and the data entered by the client in the <form> fields.

To retrieve what the client entered in the <form> fields, request object comes with 3 methods **getParameter(String)**, **getParameterNames()** and **getParameterValues(String)**.

Let us see what **Java API** says about these methods as defined in **javax.servlet.ServletRequest** interface and inherited by **HttpServletRequest** interface.

* **public java.lang.String getParameter(java.lang.String name):** Returns the value of a request parameter as a String, or null if the parameter does not exist. Request parameters are extra information sent with the request.
* **public java.util.Enumeration getParameterNames():**  
  Returns an Enumeration of String objects containing the names of the parameters contained in this request. If the request has no parameters, the method returns an empty Enumeration.
* **public java.lang.String[] getParameterValues(java.lang.String name):**  
  Returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist. If the parameter has a single value, the array has a length of 1.

##### Examples on Read Client Form Data JSP

**1. Using getParameter(String)**

This method is used to read only one form field data as a string. Example is available at [First JSP Example User Name and Password Login Validation](http://way2java.com/jsp/first-jsp-example-user-name-and-password-login-validation/). Anyhow this method usage is shown in the next 2nd way.

**2. Using getParameterNames()**

This method is used to read all form fields data at a time as an Enumeration object.

As usual there exists minimum two programs running one on client-side and the other on server-side.

**1st Program (running on client): AllFields.html**

<body>

<H3> Reading all Text box values at a time </H3>

<formmethod=*"get"*action=*"ReadAll.jsp"*>

Enter First Name <inputtype=*"text"*name=*"t1"*><br>

Enter Middle Name <inputtype=*"text"*name=*"t2"*><br>

Enter Last Name <inputtype=*"text"*name=*"t3"*><br>

Enter Experience <inputtype=*"text"*name=*"t4"*><br>

Enter Hobby <inputtype=*"text"*name=*"t5"*><br>

<inputtype=*"submit"*value=*"Print All"*>

</form></body>

**2nd Program (running on server): ReadAll.jsp**

<body>

<%

String str1=request.getParameter("t3");

out.println("Reading only one t3 field value : " + str1 + "<br><br>");

// to retrieve all fields values at a time

out.println("Reading all text boxes values at a time:<br>");

Enumeration e = request.getParameterNames();

out.println("<bodybgcolor=*pink*>");

while(e.hasMoreElements())

{

String name=(String) e.nextElement();

out.println( name +" : ");

String value=request.getParameter(name);

out.println(value.toUpperCase() +"<br>");

}

%>

</body>

**3. Using getParameterValues(String)**

**getParameterValues(String)** is used to read multiple form field data as a string array where fields have some common name. .

Observe the following client HTML file,

**user.html**

<body>

<formmethod=*"get"*action=*"readall.jsp"*>

Enter Name <inputtype=*"text"*name=*"t1"*><br>

Enter ID <inputtype=*"text"*name=*"t2"*><br>

Enter H.No. <inputtype=*"text"*name=*"t3"*><br>

Enter Street <inputtype=*"text"*name=*"t3"*><br>

Enter Area <inputtype=*"text"*name=*"t3"*><br>

Enter City <inputtype=*"text"*name=*"t3"*><br>

Enter PIN <inputtype=*"text"*name=*"t3"*><br>

<inputtype=*"submit"*value=*"PRINT ALL"*><br>

</form>

</body>

readall.jsp:

<body>

<%

String address[] = request.getParameterValues("t3");

StringBuffer temp = **new** StringBuffer();

**for**(String str : address)

{

temp.append(str + ", ");

}

out.println(temp);

%>

</body>

# 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



### 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

<formaction=*"ControllerServlet"*method=*"post"*>

Name:<inputtype=*"text"*name=*"name"*><br>

Password:<inputtype=*"password"*name=*"password"*><br>

<inputtype=*"submit"*value=*"login"*>

</form>

**ControllerServlet.java:**

package com.nrit.mvc.test;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** javax.servlet.RequestDispatcher;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**publicclass**ControllerServlet**extends** HttpServlet {

**protectedvoid** doPost(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter out=response.getWriter();

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

String password=request.getParameter("password");

LoginBean bean=**new** LoginBean();

bean.setName(name);

bean.setPassword(password);

request.setAttribute("bean",bean);

**Boolean** status=bean.validate();

**if**(status){

RequestDispatcher rd=request.getRequestDispatcher("login-success.jsp");

rd.forward(request, response);

}

**else**{

RequestDispatcher rd=request.getRequestDispatcher("login-error.jsp");

rd.forward(request, response);

}

}

@Override

**protectedvoid** doGet(HttpServletRequest req, HttpServletResponse resp)

**throws** ServletException, IOException {

doPost(req, resp);

}

}

**LoginBean.java**

package com.nrit.mvc.test;

**publicclass** LoginBean {

**private** String name,password;

**public** String getName() {

**return**name;

}

**publicvoid** setName(String name) {

**this**.name = name;

}

**public** String getPassword() {

**return**password;

}

**publicvoid** setPassword(String password) {

**this**.password = password;

}

**publicboolean** validate(){

**if**(password.equals("admin")){

**returntrue**;

}

**else**{

**returnfalse**;

}

}

}

File: login-success.jsp

<%@pageimport=*"com.nrit.mvc.test.LoginBean"*%>

<p>You are successfully logged in!</p>

<%

LoginBeanbean=(LoginBean)request.getAttribute("bean");

out.print("Welcome, "+bean.getName());

%>

File: login-error.jsp:

<p>Sorry! username or password error</p>

<%@includefile=*"index.jsp"*%>

File: web.xml

<web-app>

<servlet>

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

<servlet-class>ControllerServlet</servlet-class>

</servlet>

<servlet-mapping>

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

<url-pattern>/ControllerServlet</url-pattern>

</servlet-mapping>

</web-app>