JAVA SERVER PAGES(JSP)

JavaServer Pages (JSP) is a technology for developing Webpages that supports dynamic content. This helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.

A JavaServer Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands.

Using JSP, you can collect input from users through Webpage forms, present records from a database or another source, and create Webpages dynamically.

JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages, and sharing information between requests, pages etc.

- It is a server side technology.
- It is used for creating web application.
- It is used to create dynamic web content.
- In this JSP tags are used to insert JAVA code into HTML pages.
- It is an advanced version of Servlet Technology. (A **servlet** is a small Java program that runs within a Web server. **Servlets** receive and respond to requests from Web clients, usually across HTTP, the HyperText Transfer Protocol.)

- It is a Web based technology helps us to create dynamic and platform independent web pages.
- In this, Java code can be inserted in HTML/ XML pages or both.
- JSP is first converted into servlet by JSP container before processing the client's request.

JSP pages are more advantageous than Servlet:

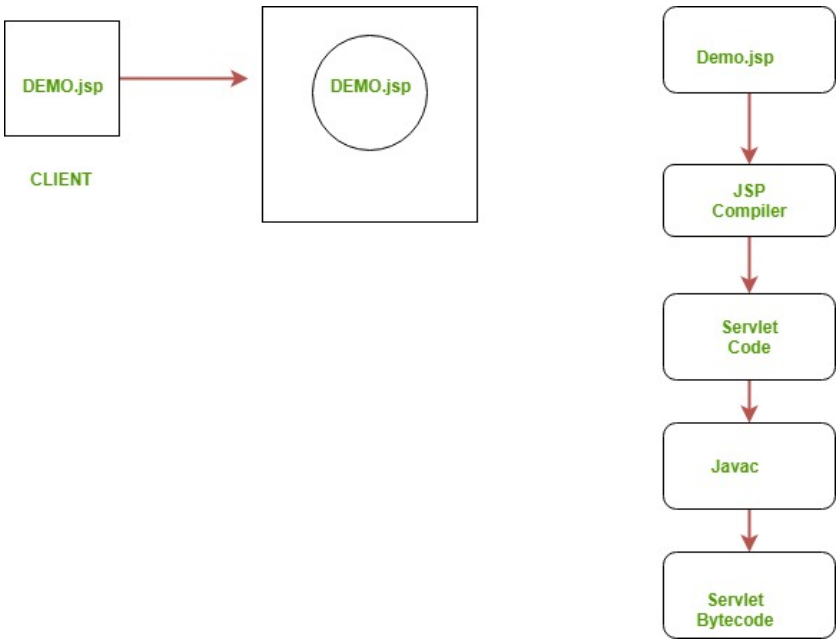
- They are easy to maintain.
- No recompilation or redeployment is required.
- JSP has access to entire API of JAVA .
- JSP are extended version of Servlet.

Features of JSP

- **Coding in JSP is easy** :- As it is just adding JAVA code to HTML/XML.
- **Reduction in the length of Code** :- In JSP we use action tags, custom tags etc.
- **Connection to Database is easier** :-It is easier to connect website to database and allows to read or write data easily to the database.
- **Make Interactive websites** :- In this we can create dynamic web pages which helps user to interact in real time environment.

- **Portable, Powerful, flexible and easy to maintain** :- as these are browser and server independent.
- **No Redeployment and No Re-Compilation** :- It is dynamic, secure and platform independent so no need to re-compilation.
- **Extension to Servlet** :- as it has all features of servlets, implicit objects and custom tags

Process of Execution



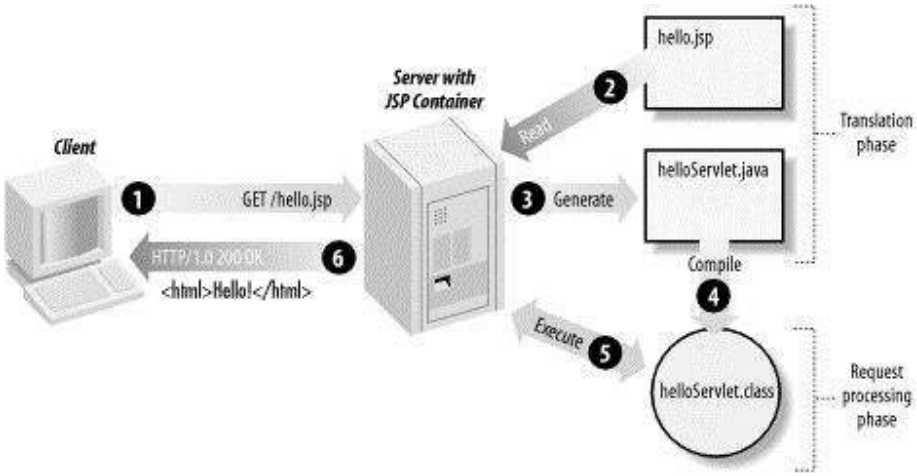
JSP Processing

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 a 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 furthur 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 −



Typically, the JSP engine checks to see whether a servlet for a JSP file already exists and whether the modification date on the JSP is older than the servlet. If the JSP is older than its generated servlet, the JSP container assumes that the JSP hasn't changed and that the generated servlet still matches the JSP's contents. This makes the process more efficient than with the other scripting languages (such as PHP) and therefore faster.

So in a way, a JSP page is really just another way to write a servlet without having to be a Java programming wiz. Except for the translation phase, a JSP page is handled exactly like a regular servlet.

JSP Lifecycle

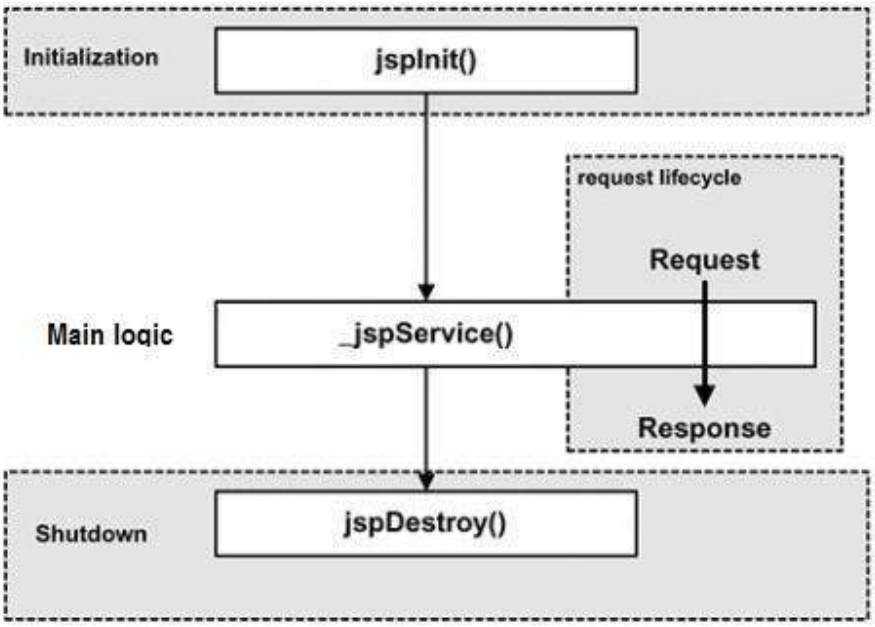
A JSP life cycle is defined as the process from its creation till the destruction. This is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet.

Paths Followed By JSP

The following are the paths followed by a JSP −

- Compilation
- Initialization
- Execution
- Cleanup

The four major phases of a JSP life cycle are very similar to the Servlet Life Cycle. The four phases have been described below −



JSP Compilation

When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page. If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.

The compilation process involves three steps −

- Parsing the JSP.
- Turning the JSP into a servlet.
- Compiling the servlet.

JSP Initialization

When a container loads a JSP it invokes the **jspInit()** method before servicing any requests. If you need to perform JSP-specific initialization, override the **jspInit()** method −

```
public void jspInit(){
  // Initialization code...
}
```

Typically, initialization is performed only once and as with the servlet init method, you generally initialize database connections, open files, and create lookup tables in the jspInit method.

JSP Execution

This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed.

Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the **_jspService()** method in the JSP.

The _jspService() method takes an **HttpServletRequest** and an **HttpServletResponse** as its parameters as follows −

```
void _jspService(HttpServletRequest request, HttpServletResponse response) {
  // Service handling code...
}
```

The **_jspService()** method of a JSP is invoked on request basis. This is responsible for generating the response for that request and this method is also responsible for generating responses to all seven of the HTTP methods, i.e, **GET, POST, DELETE**, etc.

JSP Cleanup

The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.

The **jspDestroy()** method is the JSP equivalent of the destroy method for servlets. Override jspDestroy when you need to perform any cleanup, such as releasing database connections or closing open files.

The jspDestroy() method has the following form −

```
public void jspDestroy() {
  // Your cleanup code goes here.
}
```

Example of Hello World
We will make one .html file and .jsp file

demo.jsp

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Hello World - JSP tutorial</title>
</head>
<body>
   <%= "Hello World!" %>
</body>
</html>
```

Advantages of using JSP

- It does not require advanced knowledge of JAVA
- It is capable of handling exceptions
- Easy to use and learn
- It can tags which are easy to use and understand
- Implicit objects are there which reduces the length of code
- It is suitable for both JAVA and non JAVA programmer

Disadvantages of using JSP

- Difficult to debug for errors.
- First time access leads to wastage of time
- It’s output is HTML which lacks features.

JSP syntax

Syntax available in JSP :

1. **Declaration Tag** :-It is used to declare variables.

Syntax:- <%! declaration; [declaration;]+ ... %>

Example:- <%! int var=10; %>

<%! int i = 0; %>

<%! int a, b, c; %>

<%! Circle a = new Circle(2.0); %>

2. **Java Scriplets** :- It allows us to add any number of JAVA code, variables and expressions.

A scriptlet can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language. Any text, HTML tags, or JSP elements you write must be outside the scriptlet.

Syntax:- <% java code %>

Example :

```
<html>
   <head><title>Hello World</title></head>

   <body>
      Hello World!<br/>
      <%
         out.println("Your IP address is " + request.getRemoteAddr());
      %>
   </body>
</html>
```

3. **JSP Expression** :- It evaluates and convert the expression to a string.

A JSP expression element contains a scripting language expression that is evaluated, converted to a String, and inserted where the expression appears in the JSP file.

Because the value of an expression is converted to a String, you can use an expression within a line of text, whether or not it is tagged with HTML, in a JSP file.

The expression element can contain any expression that is valid according to the Java Language Specification but you cannot use a semicolon to end an expression.

Syntax:- <%= expression %>

Example:- **1.** <%= num1 = num1+num2 %>

2.

```
<html>
   <head><title>A Comment Test</title></head>

   <body>
      <p>Today's date: <%= (new java.util.Date()).toLocaleString()%></p>
   </body>
</html>
```

4. **JAVA Comments** :- It contains the text that is added for information which has to be ignored.

Syntax:- <% -- JSP Comments %> (A JSP comment. Ignored by the JSP engine.)

<!-- comment --> (An HTML comment. Ignored by the browser.)

JSP Directives

A JSP directive affects the overall structure of the servlet class. It usually has the following form –

<%@ directive attribute="value" %>

There are three types of directive tag –

S.No.	Directive & Description
1	**<%@ page ... %>** Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.
2	**<%@ include ... %>** Includes a file during the translation phase.

3	**<%@ taglib ... %>** Declares a tag library, containing custom actions, used in the page

JSP - The page Directive

The **page** directive is used to provide instructions to the container. These instructions pertain to the current JSP page. You may code page directives anywhere in your JSP page. By convention, page directives are coded at the top of the JSP page.

Following is the basic syntax of the page directive −

<%@ page attribute = "value" %>

You can write the XML equivalent of the above syntax as follows −

<jsp:directive.page attribute = "value" />

Attributes

Following table lists out the attributes associated with the page directive −

S.No.	Attribute & Purpose
1	**Buffer :** Specifies a buffering model for the output stream.
2	**autoFlush :** Controls the behavior of the servlet output buffer.
3	**contentType :** Defines the character encoding scheme.
4	**errorPage :** Defines the URL of another JSP that reports on Java unchecked runtime exceptions.
5	**isErrorPage :** Indicates if this JSP page is a URL specified by another JSP page's errorPage attribute.
6	**Extends :** Specifies a superclass that the generated servlet must extend.
7	**Import :** Specifies a list of packages or classes for use in the JSP as the Java import statement does for Java classes.
8	**Info :** Defines a string that can be accessed with the servlet's **getServletInfo()** method.

9	**isThreadSafe :** Defines the threading model for the generated servlet.
10	**Language :** Defines the programming language used in the JSP page.
11	**Session :**Specifies whether or not the JSP page participates in HTTP sessions
12	**isELIgnored :** Specifies whether or not the EL expression within the JSP page will be ignored.
13	**isScriptingEnabled :** Determines if the scripting elements are allowed for use.

The buffer Attribute : The **buffer** attribute specifies the buffering characteristics for the server output response object. You may code a value of "**none**" to specify no buffering so that the servlet output is immediately directed to the response object or you may code a maximum buffer size in kilobytes, which directs the servlet to write to the buffer before writing to the response object.

To direct the servlet to write the output directly to the response output object, use the following

<%@ page buffer = "none" %>

Use the following to direct the servlet to write the output to a buffer of size not less than 8 kilobytes −

<%@ page buffer = "8kb" %>

The autoFlush Attribute : The **autoFlush** attribute specifies whether buffered output should be flushed automatically when the buffer is filled, or whether an exception should be raised to indicate the buffer overflow.

A value of **true (default)** indicates automatic buffer flushing and a value of false throws an exception.

The following directive causes the servlet to throw an exception when the servlet's output buffer is full −

<%@ page autoFlush = "false" %>

This directive causes the servlet to flush the output buffer when full −

<%@ page autoFlush = "true" %>

Usually, the buffer and the autoFlush attributes are coded on a single page directive as follows −

<%@ page buffer = "16kb" autoflush = "true" %>

The contentType Attribute : The contentType attribute sets the character encoding for the JSP page and for the generated response page. The default content type is **text/html**, which is the standard content type for HTML pages.

If you want to write out XML from your JSP, use the following page directive −

<%@ page contentType = "text/xml" %>

The following statement directs the browser to render the generated page as HTML −

<%@ page contentType = "text/html" %>

The following directive sets the content type as a Microsoft Word document −

<%@ page contentType = "application/msword" %>

You can also specify the character encoding for the response. For example, if you wanted to specify that the resulting page that is returned to the browser uses **ISO Latin 1**, you can use the following page directive −

<%@ page contentType = "text/html:charset=ISO-8859-1" %>

The errorPage Attribute : The **errorPage** attribute tells the JSP engine which page to display if there is an error while the current page runs. The value of the errorPage attribute is a relative URL. The following directive displays MyErrorPage.jsp when all uncaught exceptions are thrown −

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

The isErrorPage Attribute : The **isErrorPage** attribute indicates that the current JSP can be used as the error page for another JSP. The value of isErrorPage is either true or false. The default value of the isErrorPage attribute is false. For example, the **handleError.jsp** sets the isErrorPage option to true because it is supposed to handle errors −

<%@ page isErrorPage = "true" %>

The extends Attribute : The **extends** attribute specifies a superclass that the generated servlet must extend. For example, the following directive directs the JSP translator to generate the servlet such that the servlet extends ***somePackage.SomeClass*** −

<%@ page extends = "somePackage.SomeClass" %>

The import Attribute : The **import** attribute serves the same function as, and behaves like, the Java import statement. The value for the import option is the name of the package you want to import. To import **java.sql.***, use the following page directive −

<%@ page import = "java.sql.*" %>

To import multiple packages, you can specify them separated by comma as follows −

<%@ page import = "java.sql.*,java.util.*" %>

By default, a container automatically imports **java.lang.*, javax.servlet.*, javax.servlet.jsp.*,** and **javax.servlet.http.***.

The info Attribute : The **info** attribute lets you provide a description of the JSP. The following is a coding example −

<%@ page info = "This JSP Page Written By ZARA" %>

The isThreadSafe Attribute : The **isThreadSafe** option marks a page as being thread-safe. By default, all JSPs are considered thread-safe. If you set the isThreadSafe option to false, the JSP engine makes sure that only one thread at a time is executing your JSP. The following page directive sets the **isThreadSafe** option to false −

<%@ page isThreadSafe = "false" %>

The language Attribute : The **language** attribute indicates the programming language used in scripting the JSP page.For example, because you usually use Java as the scripting language, your language option looks like this −

<%@ page language = "java" %>

The session Attribute : The **session** attribute indicates whether or not the JSP page uses HTTP sessions. A value of true means that the JSP page has access to a builtin **session** object and a value of false means that the JSP page cannot access the builtin session object.

Following directive allows the JSP page to use any of the builtin object session methods such as **session.getCreationTime()** or **session.getLastAccessTime()** −

<%@ page session = "true" %>

The isELIgnored Attribute : The isELIgnored attribute gives you the ability to disable the evaluation of Expression Language (EL) expressions which has been introduced in JSP 2.0.

The default value of the attribute is true, meaning that expressions, **${...}**, are evaluated as dictated by the JSP specification. If the attribute is set to false, then expressions are not evaluated but rather treated as static text.

Following directive sets an expression not to be evaluated −

<%@ page isELIgnored = "false" %>

The isScriptingEnabled Attribute : The **isScriptingEnabled** attribute determines if the scripting elements are allowed for use. The **default value (true)** enables scriptlets, expressions, and declarations. If the attribute's value is set to false, a translation-time error will be raised if the JSP uses any scriptlets, expressions (non-EL), or declarations. The attribute’s value can be set to false if you want to restrict the usage of scriptlets, expressions (non-EL), or declarations −

<%@ page isScriptingEnabled = "false" %>

The include Directive : The **include** directive is used to include a file during the translation phase. 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. The general usage form of this directive is as follows −

<%@ include file = "relative url" >

The filename in the include directive is actually a relative URL. If you just specify a filename with no associated path, the JSP compiler assumes that the file is in the same directory as your JSP.

You can write the XML equivalent of the above syntax as follows −

<jsp:directive.include file = "relative url" />

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

The general usage form of this directive is as follows −

<%@ include file = "relative url" >

The filename in the include directive is actually a relative URL. If you just specify a filename with no associated path, the JSP compiler assumes that the file is in the same directory as your JSP.

You can write the XML equivalent of the above syntax as follows −

<jsp:directive.include file = "relative url" />

Example

A good example of the **include** directive is including a common header and footer with multiple pages of content.

Let us define following three files **(a) header.jsp, (b)footer.jsp,** and **(c)main.jsp** as follows −

Following is the content of **header.jsp** −

```
<%!
  int pageCount = 0;
  void addCount() {
    pageCount++;
  }
%>

<% addCount(); %>

<html>
  <head>
```

```
    <title>The include Directive Example</title>
  </head>

  <body>
    <center>
      <h2>The include Directive Example</h2>
      <p>This site has been visited <%= pageCount %> times.</p>
    </center>
    <br/><br/>
```

Following is the content of **footer.jsp** −

```
    <br/><br/>
    <center>
      <p>Copyright © 2010</p>
    </center>
  </body>
</html>
```

Finally here is the content of **main.jsp** −

```
<%@ include file = "header.jsp" %>
<center>
  <p>Thanks for visiting my page.</p>
</center>
<%@ include file = "footer.jsp" %>
```

Let us now keep all these files in the root directory and try to access **main.jsp**. You will receive the following output −

The include Directive Example

This site has been visited 1 times.
Thanks for visiting my page.

Refresh **main.jsp** and you will find that the page hit counter keeps increasing.

You can design your webpages based on your creative instincts; it is recommended you keep the dynamic parts of your website in separate files and then include them in the main file. This makes it easy when you need to change a part of your webpage.

The taglib Directive

The JavaServer Pages API allow you to define custom JSP tags that look like HTML or XML tags and a tag library is a set of user-defined tags that implement custom behavior.

The **taglib** directive declares that your JSP page uses a set of custom tags, identifies the location of the library, and provides means for identifying the custom tags in your JSP page.

The taglib directive follows the syntax given below −

<%@ taglib uri="uri" prefix = "prefixOfTag" >

Here, the **uri** attribute value resolves to a location the container understands and the **prefix** attribute informs a container what bits of markup are custom actions.

You can write the XML equivalent of the above syntax as follows −

<jsp:directive.taglib uri = "uri" prefix = "prefixOfTag" />

The JavaServer Pages API allow you to define custom JSP tags that look like HTML or XML tags and a tag library is a set of user-defined tags that implement custom behavior.

The **taglib** directive declares that your JSP page uses a set of custom tags, identifies the location of the library, and provides means for identifying the custom tags in your JSP page.

The taglib directive follows the syntax given below −

<%@ taglib uri = "uri" prefix = "prefixOfTag" >

Where, the **uri** attribute value resolves to a location the container understands and the **prefix** attribute informs a container what bits of markup are custom actions.

You can write the XML equivalent of the above syntax as follows −

<jsp:directive.taglib uri = "uri" prefix = "prefixOfTag" />

When you use a custom tag, it is typically of the form **<prefix:tagname>**. The prefix is the same as the prefix you specify in the taglib directive, and the tagname is the name of a tag implemented in the tag library.

Example

For example, suppose the **custlib** tag library contains a tag called **hello**. If you wanted to use the hello tag with a prefix of **mytag**, your tag would be **<mytag:hello>** and it will be used in your JSP file as follows −

```
<%@ taglib uri = "http://www.example.com/custlib" prefix = "mytag" %>

<html>
  <body>
    <mytag:hello/>
  </body>
</html>
```

We can call another piece of code using **<mytag:hello>**.

JSP Actions

JSP actions use **constructs** in XML syntax to control the behavior of the servlet engine. You can dynamically insert a file, reuse JavaBeans components, forward the user to another page, or generate HTML for the Java plugin.

There is only one syntax for the Action element, as it conforms to the XML standard −

<jsp:action_name attribute="value" />

Action elements are basically predefined functions. Following table lists out the available JSP Actions −

S.No.	Syntax & Purpose
1	**jsp:include :** Includes a file at the time the page is requested.
2	**jsp:useBean :** Finds or instantiates a JavaBean.
3	**jsp:setProperty :** Sets the property of a JavaBean.
4	**jsp:getProperty :** Inserts the property of a JavaBean into the output.
5	**jsp:forward :** Forwards the requester to a new page.
6	**jsp:plugin :** Generates browser-specific code that makes an OBJECT or EMBED tag for the Java plugin.
7	**jsp:element :** Defines XML elements dynamically.
8	**jsp:attribute :** Defines dynamically-defined XML element's attribute.
9	**jsp:body :** Defines dynamically-defined XML element's body.
10	**jsp:text :** Used to write template text in JSP pages and documents.

Control Statements

Example 1. If else

```
<html>  <head>  <title>Tutorial and Example</title> </head>
<body>
<%! int num=8; %>
<% if(num%2==0)
{
   out.println("Number is even");
}
else
{
   out.println("Number is odd");
}
%>
</body>
</html>
```

Example 2: For loop

```
<html> <head>  <title>Tutorial and Example</title> </head>
<body>
<%
for(int i=0;i<5;i++)
{
   for(int j=1;j<=i+1;j++)
   {
   out.print(j);
}
   out.print("<br>");
}
%>
</body>
</html>
```

Example 3: While loop

```
<html> <head> <title>Tutorial and Example</title> </head>
<body>
<%
int i=0;
while(i<4)
{
   i++;
   out.print(i+"<br>");
}
%>
</body>
</html>
```

Example 4: Switch

```
<html>  <head>  <title>Tutorial and Example</title> </head>
<body>
<%! int weekday=4; %>
<%
switch(weekday)
{
case 1:
   out.print("Monday");
   break;
case 2:
   out.print("Tuesday");
   break;
case 3:
   out.print("Wednesday");
   break;
case 4:
   out.print("Thursday");
   break;
case 5:
   out.print("Friday");
   break;
case 6:
```

```
    out.print("Saturday");
    break;
default:
    out.print("Sunday");
    break;
}
%>
</body>  </html>
```

```
<html>
  <head>
    <title> Read Form Data</title>
  </head>

  <body>
    <h1> Read Form Data</h1>
    <ul>
      <li><p><b>First Name:</b>
        <%= request.getParameter("first_name")%>
      </p></li>
      <li><p><b>Last  Name:</b>
        <%= request.getParameter("last_name")%>
      </p></li>
    </ul>

  </body>
</html>
```

http://localhost:8080/main.jsp?first_name=yourname&last_name=yoursirname

```
<html>
  <body>

     <form action = "main.jsp" method = "POST">
        First Name: <input type = "text" name = "first_name">
        <br />
        Last Name: <input type = "text" name = "last_name" />

  <input type = "checkbox" name = "B.Sc(hons.) CS" checked = "checked" /> Computer Science
        <input type = "checkbox" name = "B.Sc(prog.) MS"  /> Mathematical science
        <input type = "checkbox" name = "B.A(Prog)" checked = "checked" /> BA(prog)
        <input type = "submit" value = "Select Subject" />

        <input type = "submit" value = "Submit" />
     </form>

  </body>
</html>
```