

Java Server Pages (JSP)

Java Server Pages (JSP) is a technology for developing web pages that include dynamic content. Unlike a plain HTML page, which contains static content that always remains the same, a JSP page can change its content based on any number of variable items, including the identity of the user, the user's browser type, information provided by the user, and selections made by the user. JSP allows us to directly embed pure java code with HTML, JSP runs under the supervision of an environment called web container.

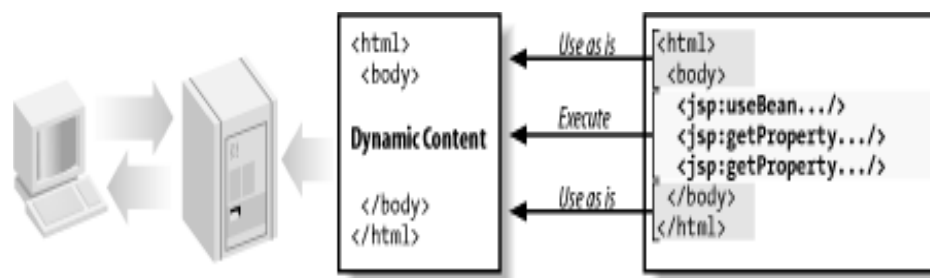


Figure 1: Generating dynamic content with JSP elements

A JSP page contains standard markup language elements, such as HTML tags, just like a regular web page. However, a JSP page also contains special JSP elements that allow the server to insert dynamic content in the page. JSP elements can be used for a variety of purposes, such as retrieving information from a database or registering user preferences. When a user asks for a JSP page, the server executes the JSP elements, merges the results with the static parts of the page, and sends the dynamically composed page back to the browser, as illustrated in Figure.

Problems with Servlet

In many Java servlet-based applications, processing the request and generating the response are both handled by a single servlet class.

An example servlet looks like this:

```
public class OrderServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
    }
}
```

```

if (isOrderInfoValid(request))
{
    saveOrderInfo(request);
    out.println("<html>");
    out.println(" <head>");
    out.println(" <title>Order Confirmation</title>");
    out.println(" </head>");
    out.println(" <body>");
    out.println(" <h1>Order Confirmation</h1>");
    renderOrderInfo(request);
    out.println(" </body>");
    out.println("</html>");
}
}}

```

The point is that the servlet contains request processing and business logic (implemented by methods such as `isOrderInfoValid()` and `saveOrderInfo()`) and also generates the response HTML code, embedded directly in the servlet code using `println()` calls. A more structured servlet application isolates different pieces of the processing in various reusable utility classes, and may also use a separate class library for generating the actual HTML elements in the response. But even so, the pure servlet-based approach still has a few problems:

- ✓ Java Servlet Technology is an extremely powerful technology. However when it is used to generate large, complex HTML code, it becomes a bit cumbersome.
- ✓ Each time the Servlet code is modified, it needs to be recompiled and the web server also needs to restart. The JSP engine takes care of all these issues automatically. Whenever a JSP code is modified, the JSP engine identifies it and translates it into a new Servlet. The Servlet code is then compiled, loaded and instantiated automatically.
- ✓ In most Servlets, a small piece of code is written to handle application logic and large code is written using several `pw.println()` statements. Since the code for application logic and formatting are closely tied, it is difficult to separate and reuse a portion of the code when a different logic or output format is required.

JSP Processing

The following steps explain how the web server creates the web page using JSP:

STEP 1: Client (Browser) sends an HTTP request to the Web Server like tomcat server.

STEP 2: Web Server recognizes that, the HTTP request is for JSP page and forwards it to a JSP engine. This is done by using URL or JSP page which ends with `.jsp` instead of `.html`.

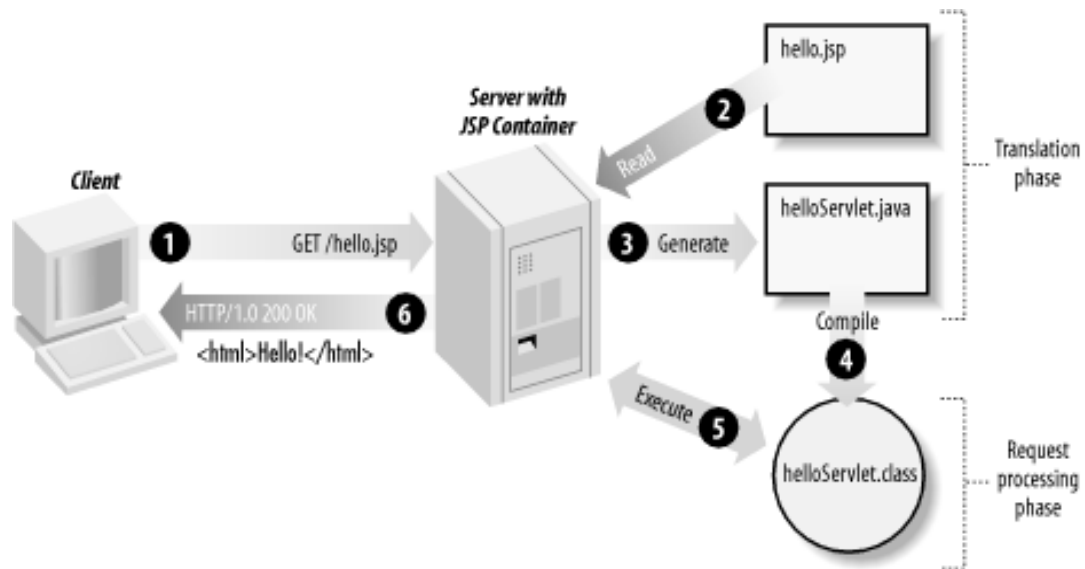


Figure 2: JSP page translation and processing phases

STEP 3: JSP engine loads the JSP page from the disk and convert it into Servlet. Converting the JSP page to a Servlet and compiling the Servlet form the **translation phase**. The JSP container initiates the translation phase for a page automatically when it receives the first request for the page. Since the translation phase takes a bit of time, the first user to request a JSP page notices a slight delay. The translation phase can also be initiated explicitly; this is referred to as pre-compilation of a JSP page. Pre-compiling a JSP page is a way to avoid hitting the first user with this delay.

STEP 4: The JSP container is also responsible for invoking the JSP page implementation class (the generated Servlet) to process each request and generate the response. This is called the **request processing phase**. The two phases are illustrated in Figure 2. (JSP engine compiles the Servlet into a class file and forwards the original request to Servlet container.)

STEP 5: Web Server call the Servlet engine, which loads the Servlet class and execute it. During the execution, the Servlet produces an output in HTML format, which Servlet engine passes to the web server inside an HTTP response.

STEP 6: The Web Server forwards the HTTP response to client browser in terms of static HTML Content.

STEP 7: Finally web Browser handles the dynamically generated HTML page inside the HTTP response exactly as a static page.

Typically, the JSP engine checks to see whether a Servlet for a JSP file already exist and whether the modification date on the JSP is older than the Servlet. If the JSP is older than its generated Servlet, the JSP engine assumes that the JSP hasn't changed and that generated Servlet still matches the JSP's Content. This makes the process more efficient than with other scripting language and therefore faster.

JSP Application using MVC Architecture

MVC was first described by Xerox in a number of papers published in the late 1980s. The key point of using MVC is to separate logic into three distinct units: the Model, the View, and the Controller. In a server application, we commonly classify the parts of the application as business logic, presentation, and request processing.

- ✓ **Business logic** is the term used for the manipulation of an application's data, such as customer, product, and order information.
- ✓ **Presentation** refers to how the application data is displayed to the user, for example, position, font, and size.
- ✓ And finally, **request processing** is what ties the business logic and presentation parts together. In MVC terms, the Model corresponds to business logic and data, the View to the presentation, and the Controller to the request processing.

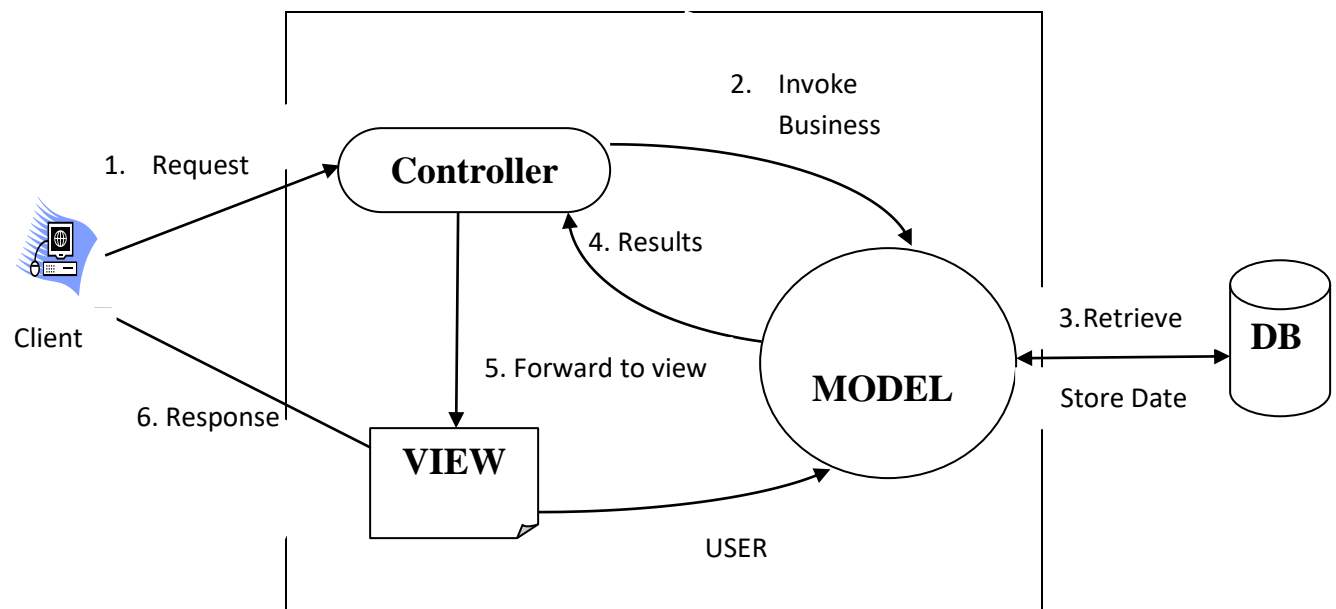


Figure 3: MVC Architecture (Model, View and Control)

Model: It is the business logic of application, responsible for performing the actual work conducted by the application. Hence, we can say that this unit deals with the modeling of real-world problems. It does not know anything about view or controller.

View: This is presentation logic of the application, responsible mainly for rendering the information of application. It may have little or no programming logic at all.

Controller: This is request, processing logic of the application, mainly responsible for coupling both the model and view together. We can think of it as a traffic controller directing request to the corresponding resources and forwarding appropriate response to the user.

JSP pages are used as both the Controller and the View, and JavaBeans components are used as the Model.

Why use this design with JSP?

An application data structure and logic (the Model) is typically the most stable part of an application, while the presentation of that data (the View) changes fairly often. Some of the reasons to use this architecture:

- ✓ Need to change look and feel of web pages periodically either to attract the user to come back to the page again and again (Presentation). New presentation interfaces can be developed without touching the business logic.
- ✓ Need to give access to different subsets of data to different users. (Controlling)
- ✓ Need to support different types of devices such as cell phones and PDA's. So that data can be displayed in their corresponding format.
- ✓ Need to support different languages.

Anatomy of JSP Page

A JSP page is simply a regular web page with JSP elements for generating the parts that differ for each request, as shown in the Figure 4. JSP Page basically consists of two parts.

- ✓ **HTML/XML Markups**
Large percentage of your JSP page, in many cases, consists of static HTML/XML elements.
- ✓ **JSP Constructs (or) JSP Elements**

The diagram illustrates the components of a JSP page. It shows a sequence of code blocks, each with a bracket on the right indicating its category:

- `<%@ page language="java" contentType="text/html" %>` is labeled as a *JSP element*.
- `<html>` and `<body bgcolor="white">` are grouped as *template text*.
- `<jsp:useBean id="userInfo" class="com.ora.jsp.beans.userinfo.UserInfoBean">`, `<jsp:setProperty name="userInfo" property="*" />`, and `</jsp:useBean>` are grouped as *JSP element*.
- `The following information was saved:` and `` are grouped as *template text*.
- `User Name:` is grouped as *template text*.
- `<jsp:getProperty name="userInfo" property="userName" />` is labeled as a *JSP element*.
- `Email Address:` is grouped as *template text*.
- `<jsp:getProperty name="userInfo" property="emailAddr" />` is labeled as a *JSP element*.
- ``, `</body>`, and `</html>` are grouped as *template text*.

Figure 4: Template text and JSP Construct (or) JSP elements

Everything in the page that isn't a JSP element is called **template text**. Template text can be any text: HTML, XML, or even plain text. Since HTML is by far the most common web page language in use today, but keep in mind that JSP has no dependency on HTML; it can be used with any markup language. Template text is always passed straight through to the browser.

When a JSP page request is processed, the template text and dynamic content generated by the JSP elements are merged, and the result is sent as the response to the browser.

JSP Elements

JSP is all about generating dynamic content: content that differs based on user input, time of day, the state of an external system, or any other runtime conditions. JSP provides you with lots of tools for generating this content.

There are three types of JSP elements you can use: directive, action, and scripting. A new construct added in JSP 2.0 is an Expression Language (EL) expression; let's call this a forth element type, even though it's a bit different than the other three.

a) Scripting elements

Scripting elements, shown in Table-1, allow you to add small pieces of code (typically Java code) in a JSP page, such as an if statement to generate different HTML depending on a certain condition. Like actions, they are also executed when the page is requested. You should use scripting elements with extreme care: if you embed too much code in your JSP pages, you will end up with the same kind of maintenance problems as with Servlets embedding HTML.

	Element	Equivalent XML Tag	Description
Scriptlet	<code><% %></code>	<code><jsp:Scriptlet>.... </jsp:Scriptlet></code>	Scriptlet, used to embed scripting code
Declaration	<code><%!%></code>	<code><jsp:declaration> </jsp:declaration></code>	Declaration, used to declare instance variables and methods in the JSP page implementation class
Expression	<code><% =... %></code>	<code><jsp:expression> </jsp:declaration></code>	Expression, used to embed scripting code expressions when the result shall be added to the response; also used as request-time action attribute values.

Table 1: Scripting Elements

a) Scriptlets:

You can insert an arbitrary piece of java code using the JSP scriptlet construct in JSP Page. This code will be inserted in Servlets `_jspService()` method. Scriptlets are useful if you want to insert complex code which would otherwise be difficult using expressions. A scriptlet contain any number of variables, class declarations, expressions.

Syntax:

```
<%  
    //Embed your java code here  
%>
```

Example: Display Current Date and Time

```
<%  
    Java.util.Date dt= new java.util.Date();  
    String current  = dt.toString();  
    Out.println("Current Date is: " +current);  
%>
```

The Equivalent Servlet code for Display Current Date and Time JSP page

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Helloworldservlet extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException,
                                                                    IOException
    {
        res.setContentType("text/html");
        PrintWriter pw = res.getWriter();
        Date d=new Date( );
        String da = d.toString( );
        pw.println("<HTML><BODY>");
        pw.println( "Current Date is: " +da );
        pw.println("</BODY></HTML>");
        pw.close( );
    }
}
```

JSP also has the following XML equivalent for scriptlet

```
<jsp:scriptlet>
    //Embed your java code here
</jsp:scriptlet>
```

b) Expression:

It is used to insert usually a small piece of data in JSP page, without using the `pw.println()` statement. It is faster, easier and clearer way to display the values of variables/parameters/expressions in JSP page.

Syntax:

```
<%= Expression %>
```

The expression is embedded within the pair `<%=` and `%>` . Note that no semicolon is used at the end of expression. The expression is evaluated and converted to string and inserted in the place of expression using `out.println()`.

Example:


```
<%= "WELCOME TO VRSE" %>
```

```
Addition of 3 and 2 = <%=3+2 %>
```

JSP also has the following XML equivalent for scriptlet

```
<jsp:expression>  
    //Embed your java expressions here  
</jsp:expression>
```

c) Declarations

JSP Declarations are used to declare one or more variables, methods or inner classes, which can be used in later JSP Pages.

Syntax:

```
<%! Declarations are here %>
```

Variable Declaration:

```
<%! int a =0; %>
```

```
<%! int x, y, z; %>
```

```
<%! double PI=22/.0; %>
```

Example: Number of users visited my webpage

```
<html>  
  
<head> <title> Users Count </title> </head>  
    <body>  
        <%! int users_visited=0; %>  
        Welcome to My Web page <br>  
        <%  
            users_visited++;  
            out.println("Number of Users visited My web page: " +users_visited);  
        %>  
    </body>  
</html>
```

Method Declaration:

Here is a method Add, to perform addition between two integer values.

```
<%!
```

```

public int Add(int first_number, int second_number)
{
    return first_number+second_number;
}
%>

```

Variables, methods and classes declared in the declaration section are available in scriptlets. So, the following scriptlet is valid for this declaration

```

<% out.println("Addition of 2 and 3 is: " +Add(2,3)); %>

```

JSP also has the following XML equivalent for scriptlet

```

<jsp:declaration>
    //Embed your java Declarations here
</jsp:declaration>

```

Conditional Processing:

Servlets and template text can be merged, to do some designated applications. Consider the simple example to find the given number is even or odd in the following example

a) **<% - - Find the given number is even or odd - -%>**

```

<html>

<head> <title> Even or Odd Number</title> </head>
<body>
    <%! int number=5; %>
    <%
        if (number % 2 == 0 ){ %>
        <p> <h2> Given Number is Even </h2>      </p>
        <%
            } else
            {
        %>
        <p> <h3> Given Number is Odd </h3>      </p>
        <%
            }
    %>

```

%>

This will produce the following result:

b) <% - - Factorial of 1 to N numbers - -%>

i) **Factorial.html**

```
<html>
<head>    <title> Factorial Table </title>    </head>
<body>
    <form method="post" action="http://localhost:8080/CSE2/Factorial.jsp" >
        <table align="center">
            <tr>
                <td colspan='3' align='center'> <h2> Factorial Table </h2> </td>
            </tr>
            <tr>
                <td> Number : </td> <td> <input type="text" name="last" /> </td>

                <td> <input type="submit" value="Generate" /> </td>
            </tr>
        </table>
    </form>
</body>
</html>
```

ii) **Factorial.jsp**

```
<html>
<head>    <title> Factorial upto Given Number </title>    </head>

<body>
    <h2> Factorial Up to 1 to N </h2>

    <%
        int input = Integer.parseInt(request.getParameter("last"));
        long fact;
    %>
    <table border='1'>
```

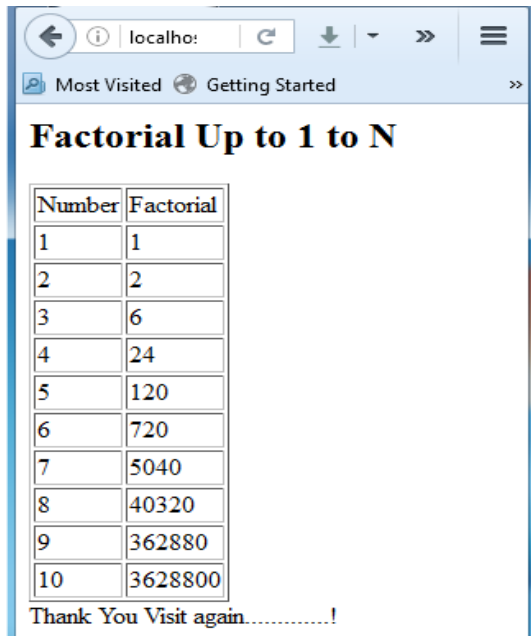
```

        <tr> <td> Number </td> <td> Factorial </td> </tr>
    <%
        for(int i=1;i<=input;i++)
        {
            fact =1;
            for(int j=1; j <= i; j++)
            {
                fact = fact * j;
            }
        }
    %>
    <tr> <td> <%=i %></td> <td> <%=fact%> </td> </tr>
    <%
    }
    %>
</table>
    Thank You Visit again.....!
</body>
</html>

```

OUTPUT:





A screenshot of a web browser window. The address bar shows 'localho:'. The page title is 'Factorial Up to 1 to N'. Below the title is a table with two columns: 'Number' and 'Factorial'. The table contains rows for numbers 1 through 10 and their corresponding factorials. At the bottom of the page, there is a text message: 'Thank You Visit again.....!'.

Number	Factorial
1	1
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320
9	362880
10	3628800

Thank You Visit again.....!

Directives

Refer your running Note

Action elements

Refer your running Note

Implicit JSP Scripting Objects

Scripting elements can use predefined variables that the container assigns as references to implicit objects (Table -2) to access request and application data. These objects are instances of classes defined by the servlet and JSP specifications.

VARIABLE NAME	JAVA TYPE
request	javax.servlet.http.HttpServletRequest
response	javax.servlet.http.HttpServletResponse
out	javax.servlet.jsp.JspWriter
pageContext	javax.servlet.jsp.PageContext
config	javax.servlet.ServletConfig
application	javax.servlet.ServletContext
session	javax.servlet.http.HttpSession
exception	java.lang.Throwable

Table-2: Implicit Objects

request

The request variable contains a reference to an instance of a class that implements an interface named **javax.servlet.http.HttpServletRequest**. It provides methods for accessing all the information that's available about the current request, such as request parameters, attributes, headers, and cookies.

response

The response variable contains a reference to an object representing the current response message. It's an instance of a class that implements the **javax.servlet.http.HttpServletResponse** interface, with methods for setting headers and the status code, and adding cookies. It also provides methods related to session tracking.

out

The out object is an instance of **javax.servlet.jsp.JspWriter**. You can use the **print()** and **println()** methods provided by this object to add text to the response message body. In most cases, however, you will just use template text and JSP action elements instead of explicitly printing to the out object.

pageContext

The `pageContext` variable contains a reference to an instance of the class named **`javax.servlet.jsp.PageContext`**. It provides methods for accessing references to all the other objects and attributes for holding data that is shared between components in the same page. Attribute values for this object represent the page scope;

application

The `application` variable contains a reference to the instance of a class that implements the **`javax.servlet.ServletContext`** interface that represents the application. This object holds references to other objects that more than one user may require access to, such as a database connection pool shared by all application users. It also contains `log()` methods you can use to write messages to the container's log file.

session

The `session` variable allows you to access the client's session data, managed by the server. It's assigned a reference to an instance of a class that implements the **`javax.servlet.http.HttpSession`** interface, which provides access to session data as well as information about the session, such as when it was created and when a request for the session was last received.

exception

`exception` is an instance of **`java.lang.Throwable`**. The exception object is available only in error pages and contains information about a runtime error.

config

It is `javax.servlet.ServletConfig` type object refers to the configuration of the underlying Servlet. It is used to retrieve initial parameters, Servlet name etc. This object has page scope.

JSP Page Showing the Current Date and Time (date.jsp)

```
<% @ page language="java" contentType="text/html" %>
```

```
<html>
```

```
<body bgcolor="blue">
```

```
<jsp:useBean id="clock" class="java.util.Date" />
```

The current time at the server is:

```
<ul>
```

```
<li>Date: <jsp:getProperty name="clock" property="date" />
```

```
<li>Month: <jsp:getProperty name="clock" property="month" />
```

```
<li>Year: <jsp:getProperty name="clock" property="year" />
```

```
<li>Hours: <jsp:getProperty name="clock" property="hours" />
```

```
<li>Minutes: <jsp:getProperty name="clock" property="minutes" />
```

```
</ul>
```

```
</body>
```

```
</html>
```