JAVA SERVER PAGES (JSP)

JavaServer Pages (JSP) is a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications.

The tags in JSP are enclosed within <% and %>.

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.

Advantages of JSP over CGI

- Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having a separate CGI files.
- JSP are always compiled before it's processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.
- JavaServer Pages are built on top of the Java Servlets API, so like Servlets, JSP also has access to all the powerful Enterprise Java APIs, including JDBC, JNDI, EJB, JAXP etc.
- JSP pages can be used in combination with servlets that handle the business logic, the model supported by Java servlet template engines.
- JSP is an integral part of Java EE, a complete platform for enterprise class applications. This means that JSP can play a part in the simplest applications to the most complex and demanding.

Advantages of JSP over Active Server Pages (ASP):

- The dynamic part is written in Java, not Visual Basic or other MS specific language, so it is more powerful and easier to use.
- It is portable to other operating systems and non-Microsoft Web servers.

Advantages of JSP over Pure Servlets:

- It is an extension to the servlets.
- Easy to maintain than servlets
- No need for recompilation and redeployment (If JSP page is modified, we don't need to recompile and redeploy the project.).
- Less code.

Architecture of JSP:

The web server needs a JSP engine i.e. container to process JSP pages. The **JSP container** is responsible for intercepting requests for JSP pages. A JSP container works with the Web server to provide the runtime environment and other services a JSP needs. It knows how to understand the special elements that are part of JSPs.

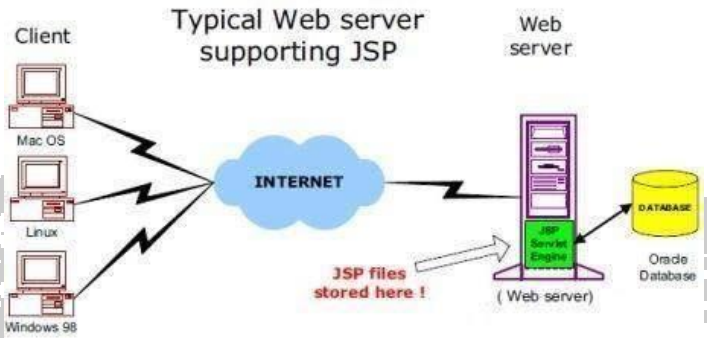


Figure 3.21 JSP-architecture

Processing of JSP

The 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 that 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, which the servlet engine passes to the web server inside an HTTP response.The web server forwards the HTTP response to your browser in terms of static HTML content.Finally web browser handles the dynamically generated HTML page inside the HTTP response exactly as if it were a static page.

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 other scripting languages (such as PHP) and therefore faster.

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

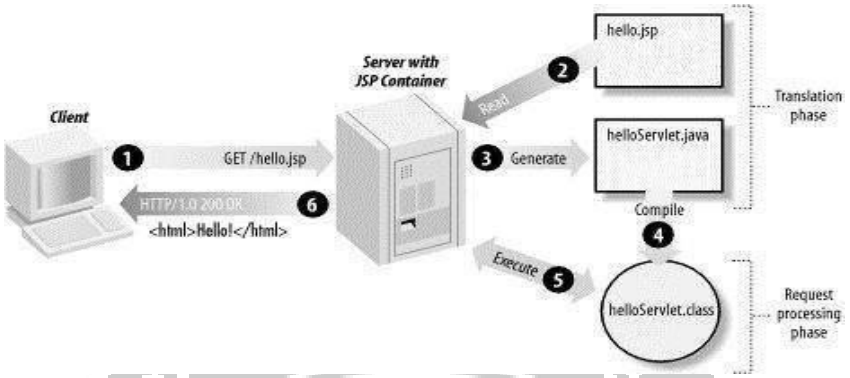


Figure 3.22 Processing of JSP

Difference between JSP and Servlets

JSP	Servlets
JSP is a webpage scripting language that can generate dynamic content.	Servlets are Java programs that are already compiled which also creates dynamic web content.
In MVC, JSP act as a view.	In MVC, servlet act as a controller.
It's easier to code in JSP than in Java Servlets.	More code is needed here.
JSP are generally preferred when there is not much processing of data required.	Servlets are used when there is more processing and manipulation involved.
JSP run slower compared to Servlet as it takes compilation time to convert into Java Servlets.	Servlets run faster compared to JSP.
The advantage of JSP programming over servlets is that we can build custom tags which can directly call Java beans.	There is no such custom tag facility in servlets.
We can achieve functionality of JSP at client side by runningJavaScript at client side.	There are no such methods for servlets.

Lifecycle of JSP

A JSP life cycle can be defined as the entire process from its creation till the destruction which is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet.The following are the phases followed by a JSP: Compilation, Initialization, Execution and Cleanup

- **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 and Compiling the servlet.

- **JSP Initialization:**

  When a container loads a JSP it invokes the jspInit() method before servicing any requests.In case of 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. The jspInit()initialize database connections, open files, and create lookup tables.

- **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.

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

  The _jspService() method of a JSP is invoked once per a request and 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 ie. 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.Override jspDestroy to perform any cleanup, such as releasing database connections or closing open files.

```
public void jspDestroy()
{ //cleanup code }
```

JSP Syntax

A JSP document contains the following tags: scriplet, expressions and declaration tags.

- **Scriptlet:**

  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 must be outside the scriptlet.

1. <% code fragment %> OR

2. <jsp:scriptlet> code fragment </jsp:scriptlet> (XML format)

Scriplet

```
<html><head><title>Hello  </title></head>
<body>
Hello <br/>
<%out.println("The IP address is " + request.getRemoteAddr());
%>
</body></html>
```

Declaration

This part declares one or more variables or methods that used in Java code later in the JSP file.

1. <%! declaration; [declaration;]+ ... %> OR

2. <jsp:declaration> code fragment </jsp:declaration> (XML format)

Example: <%! int i = 0; %> ;<%! int a, b, c; %> ;<%! Circle a = new Circle(2.0); %>

- **JSP Expression:**

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, the expression could be used 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 semicolon to end an expression is invalid.

1. <%= expression %> OR

2. <jsp:expression> expression</jsp:expression> (XML format)

Expression tags

Index.html

```
<html><body>
<form action="welcome.jsp">
<input type="text" name="uname"><br/>
```

```
<input type="submit" value="go">
</form></body></html>
```

Hai.jsp

```
<html>
<body>
<%= "Hai"+request.getParameter("uname") %>
</form></body></html>
```

In this example, the username is displayed using the expression tag. The index.html file gets the username and sends the request to the hai.jsp file, which displays the username.

- **JSP Comments:**
  - JSP comment marks text or statements that the JSP container should ignore.

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

Declarations and Comments:

```
<html><head><title>A Comment Test</title></head>
<body><h2> Comments</h2><p>
  Today's date: <%= (new java.util.Date()).toLocaleString()%>
<%-- Displays today's date --%>
</p></body></html>
```

A Comment Test
Today's date: 01-JAN-2015 21:24:25

JSP Directives:

A JSP directive affects the overall structure of the servlet class.

<%@ directive attribute="value" %>

Directive	Description
<%@ page ... %>	Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.
<%@ include ... %>	Includes a file during the translation phase.
<%@ taglib ... %>	Declares a tag library, containing custom actions, used in the page.

1. **The page Directive:**

   The page directive is used to provide instructions to the container that pertain to the current JSP page. Page directives can be used anywhere in the JSP page. By convention, page directives are coded at the top of the JSP page.

   1. <%@ page attribute="value" %> OR

   2. <jsp:directive.page attribute="value" /> (XML format)

2. **The include Directive:**

   The include directive is used to includes 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.

   1. <%@ include file="relative url" > OR

   2. <jsp:directive.include file="relative url" /> (XML Format)

3. **The taglib Directive:**

   The JavaServer Pages API allows 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 a means for identifying the custom tags in a JSP page.

   1. <%@ taglib uri="uri" prefix="prefixOfTag" > OR

   2. <jsp:directive.taglib uri="uri" prefix="prefixOfTag" /> (XML Format)

   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.

JSP Actions:

JSP actions use constructs in XML syntax to control the behavior of the servlet engine.

<jsp:action_name attribute="value" />

Syntax	Purpose
jsp:include	Includes a file at the time the page is requested
jsp:useBean	Finds or instantiates a JavaBean
jsp:setProperty	Sets the property of a JavaBean

jsp:getProperty	Inserts the property of a JavaBean into the output
jsp:forward	Forwards the requester to a new page
jsp:plugin	Generates browser-specific code that makes an OBJECT or EMBED tag for the Java plugin
jsp:element	Defines XML elements dynamically.
jsp:attribute	Defines dynamically defined XML element's attribute.
jsp:body	Defines dynamically defined XML element's body.
jsp:text	Use to write template text in JSP pages and documents.

Attributes:

There are two attributes that are common to all Action elements:

- **Id attribute:** The id attribute uniquely identifies the Action element, and allows the action to be referenced inside the JSP page. If the Action creates an instance of an object the id value can be used to reference it through the implicit object PageContext
- **Scope attribute:** This attribute identifies the lifecycle of the Action element. The id attribute and the scope attribute are directly related, as the scope attribute determines the lifespan of the object associated with the id. The scope attribute has four possible values: (a) page, (b)request, (c)session, and (d) application.

The <jsp:include> Action:

Let us define following two files (a)date.jps and (b) main.jsp as follows:

date.jsp file:

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

main.jsp file:

```
<html><head><title>The include Action Example</title>
</head>
<body><center><h2>The include action Example</h2>
```

```
<jsp:include page="date.jsp" flush="true" />
</center></body></html>
```

```
The include action Example
Today's date: 12-Sep-2010 14:54:22
```

JSP Implicit Objects:

JSP Implicit Objects are the Java objects that the JSP Container makes available to developers in each page and developer can call them directly without being explicitly declared.JSP supports nine automatically defined variables, which are also called implicit objects.

Objects	Description
request	This is the HttpServletRequest object associated with the request.
response	This is the HttpServletResponse object associated with the response to the client.
out	This is the PrintWriter object used to send output to the client.
session	This is the HttpSession object associated with the request.
application	This is the ServletContext object associated with application context.
config	This is the ServletConfig object associated with the page.
pageContext	This encapsulates use of server-specific features like higher performance JspWriters.
page	This is simply a synonym for this, and is used to call the methods defined by the translated servlet class.
Exception	The Exception object allows the exception data to be accessed by designated JSP.

1. out. print(dataType dt)- Print a data type value
2. config.getServletName()-gets the name of the servlet using config object.
3. pageContext.removeAttribute("attrName", PAGE_SCOPE) -removes the attribute from page scope.

Control-Flow Statements

JSP provides full power of Java to be embedded in the web application. All the APIs and building blocks of Java can be used in JSP programming including decision making statements, loops etc.

Decision-Making Statements

The if...else block starts out like an ordinary Scriptlet, but the Scriptlet is closed at each line with HTML text included between Scriptlet tags.

If.else

```
<%! int day = 3; %><html>
<head><title>IF...ELSE   Example</title></head>
<body><% if (day == 1 | day == 7)
{ %><p> Today is weekend</p>
<% }
else { %><p> Today is not weekend</p>
<% } %></body></html>
```

Today is not weekend

Switch:

```
<%! int day = 3; %>
<html><head><title>SWITCH...CASE    Example</title></head>
<body>
<% switch(day) {
case  0:out.println("It\'s  Sunday.");  break;
case 1:out.println("It\'s  Monday.");  break;
case  2:out.println("It\'s  Tuesday."); break;
case 3:out.println("It\'s Wednesday."); break;
case 4:out.println("It\'s Thursday."); break;
case 5:out.println("It\'s Friday."); break;
default:out.println("It's Saturday.");}
%>
</body></html>
```

It's Wednesday.

Loop Statements:

All three basic types of looping blocks in Java can be used in JSP: for, while,and do…while blocks

For loop

```
<%! int fontSize; %>
<html>
<head><title>FOR LOOP Example</title></head>
<body><%for ( fontSize = 1; fontSize <= 3; fontSize++){ %>
<font color="green" size="<%= fontSize %>">
   JSP </font><br /><%}%></body></html>
```

```
JSP
JSP
JSP
```

JSP Operators:JSP supports all the logical and arithmetic operators supported by Java.

JSP Literals: The JSP expression language defines the following literals:Boolean: true and false, Integer: as in Java, Floating point: as in Java, String: with single and double quotes; "is escaped as \", ' is escaped as \', and \ is escaped as \\ and null.

JSP Standard Tag Library (JSTL)

The JSP Standard Tag Library (JSTL) represents a set of tags to simplify the JSP development.

Advantage of JSTL

- Fast developement: JSTL provides many tags that simplifies the JSP.
- Code reusability: We can use the JSTL tags in various pages.
- It avoids the use of scriptlet tag.

There JSTL mainly provides 5 types of tags:

Tag Name	Description
core tags	The JSTL core tag provide variable support, URL management, flow control etc. The url for the core tag is http://java.sun.com/jsp/jstl/core. The prefix of core tag is c.

sql tags	The JSTL sql tags provide SQL support. The url for the sql tags is http://java.sun.com/jsp/jstl/sql and prefix is sql.
xml tags	The xml sql tags provide flow control, transformation etc. The url for the xml tags is http://java.sun.com/jsp/jstl/xml and prefix is x.
internationalization tags	The internationalization tags provide support for message formatting, number and date formatting etc. The url for the internationalization tags is http://java.sun.com/jsp/jstl/fmt and prefix is fmt.
functions tags	The functions tags provide support for string manipulation and string length. The url for the functions tags is http://java.sun.com/jsp/jstl/functions and prefix is fn.

JSTL Core Tags

The JSTL core tags mainly provides 4 types of tags:

- miscellaneous tags: catch and out.
- url management tags: import, redirect and url.
- variable support tags: remove and set.
- flow control tags: forEach**,** forTokens, if and choose.

Syntax for core tags

s%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

Tag	Description
c:catch	It handles the exception and doesn't propagate the exception to error page. The exception object thrown at runtime is stored in a variable named var.
c:out	It is just like JSP expression tag but it is used for expression. It renders data to the page.
c:import	It is just like jsp include but it can include the content of any resource either within server or outside the server.
c:forEach	It repeats the nested body content for fixed number of times or over collection.
c:if	It tests the condition.
c:redirect	It redirects the request to the given url.

HTML forms with JSP tags

The browser uses two methods to pass some information to web server: GET Method and POST Method. The data entered in the forms reach the server through these methods.

GET method:

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ?character as follows:

Example: http://www.abc.com/hello?key1=value1&key2=value2

The GET method is the default method to pass information from browser to web server and it produces a long string that appears in the browser's Location:box. Never use the GET method if password or other sensitive information is passed to the server.

The GET method has size limitation: only 1024 characters can be in a request string.This information is passed using QUERY_STRING header and will be accessible through QUERY_STRING environment variable which can be handled using getQueryString() and getParameter() methods of request object.

POST method:

A more reliable method of passing information to a backend program is the POST method.This method packages the information in exactly the same way as GET methods, but instead of sending it as a text string after a ?in the URL it sends it as a separate message.

This message comes to the backend program in the form of the standard input which can be parsed and use for processing.JSP handles this type of requests using getParameter() method to read simple parameters and getInputStream() method to read binary data stream coming from the client.

Reading Form Data using JSP

JSP handles form data parsing automatically using the following methods depending on the situation:

Methods	Description
getParameter()	to get the value of a form parameter.
getParameterValues()	Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
getParameterNames()	Call this method to get a complete list of all parameters in the current request.
getInputStream()	Call this method to read binary data stream coming from the client.

Get():

Form.html

```
<html><body><form action="main.jsp" method="GET">
First Name: <input type="text" name="first_name">
<br />
Last Name: <input type="text" name="last_name" />
<input type="submit" value="Submit" />
</form></body></html>
```

Main.jsp

```
<html><head><title>Using GET Method to Read Form Data</title>
</head><body><center>
<h1>Using GET Method to 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>
```

First Name: SONA
Last Name: RAJ Submit

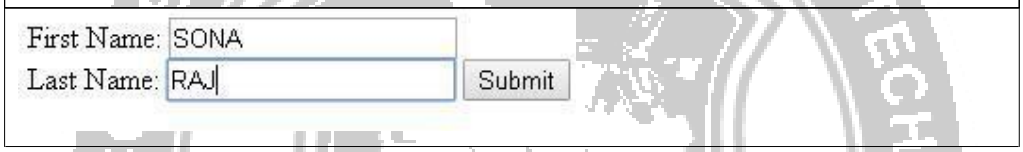
POST Method Using Form:

Form.html

```
<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="submit" value="Submit" />
</form></body></html>
```

Main.jsp

```
<html><head>
<title>Using GET and POST Method to Read Form Data</title></head>
<body><center>
<h1>Using GET Method to 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>
```



Reading All Form Parameters:

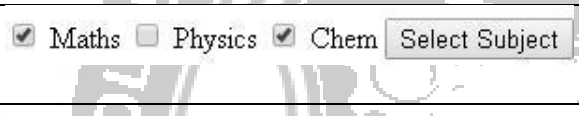
Form.html

```
<html><body>
<form action="main.jsp" method="POST" target="_blank">
<input type="checkbox" name="maths" checked="checked" /> Maths
<input type="checkbox" name="physics" /> Physics
<input type="checkbox" name="chemistry" checked="checked" /> Chem
<input type="submit" value="Select Subject" />
</form></body></html>
```

Main.jsp

```
<%@ page import="java.io.*,java.util.*" %>
<html><head><title>HTTP Header Request Example</title>
</head>
<body><center><h2>HTTP Header Request Example</h2>
<table width="100%" border="1" align="center">
```

```
<tr  bgcolor="#949494">
<th>Param  Name</th><th>Param  Value(s)</th></tr>
<%
  Enumeration paramNames = request.getParameterNames();
  while(paramNames.hasMoreElements()) {
    StringparamName=(String)paramNames.nextElement();
    out.print("<tr><td>" + paramName + "</td>\n");
    String paramValue = request.getHeader(paramName);
    out.println("<td> " + paramValue + "</td></tr>\n");
  }%></table></center></body></html>
```



Param Name	Param Value(s)
maths	on
chemistry	on