**Java Server Pages(JSP).**

Java server pages is a server-side programming technology that enables the creation of dynamic, platform independent method of building web based applications. This course will enable you to learn how to use java server pages to develop web applications in simple and easy ways.

The developers insert java code in html pages by making use of special JSP tags, most of which start with **<% and end with %>**. A server page is a type of java servlet that is designed to fulfill the role of a user interface for a java web application. We have **.jsp** as text files that combine **HTML** code and embedded JSP actions and commands.

**Why Should I use Java Server Pages?**

* Performance is significantly better because JSP allows embedding Dynamic Elements in HTML pages itself instead of having separate Common Gateway Interface files.
* JSP always compiled before they are processed by the server unlike the CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.
* Server 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→(Ensures connection to Databases, like MySql).
* These pages can be used in combination with servelets that handle the business logic, the model supported by the Java servlet template engines.

**Advantages of Java Server Pages (JSP).**

* Since the back end is written in java, it is more powerful and easier to use. It is portable across operating systems and non-Microsoft web servers.
* It is more convenient to write, especially because regular HTML is easier to write than to have many println statements that generate the HTML
* JavaScript can generate HTML dynamically on the client but can hardly interact with the web server to perform complex tasks like Database access and image processing.

**Setting Up the Web Environment.**

**Visit the site to find installation for Windows.**

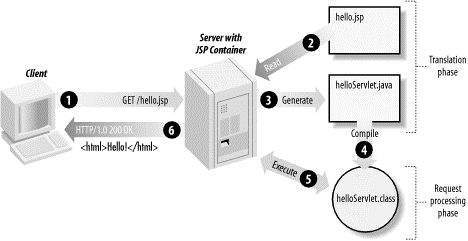
* **https://www.tutorialspoint.com/jsp/jsp\_environment\_setup.htm**

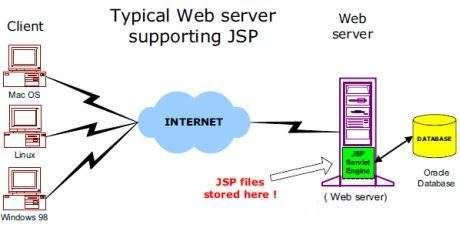
**2. Java Server Page Architecture.**

The web server needs a JSP engine, a container to process JSP pages. The container is responsible for intercepting requests for the pages. A container works with the web server to provide the runtime environment and other services JSP needs.

Just like a normal page, your browser sends a HTTP request to the web server, the web server recognizes that the HTTP request is for a jsp page and forwards it to the jsp engine. The Jsp engine loads the jsp page from the disk and converts it to a servlet content. The conversion is simple in which all the text is converted to println() statements and all the jsp elements are converted to Java Code.

A part of the web server called the servelet engine loads the servlet class and executes it. During execution, the servlet produces an output in HTML format. The output is further passed on to the web server by the servlet engine inside an HTTP response.





**The Execution of Java Server Page.**

Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engines invokes the \_jspsService method in the JSP. The method takes an HttpServletRequest and an HttpServletResponse.

Void \_jspService(HttpServletRequest request, HttpServletResponse response){

//

}

The above method is responsible for generating the response to that request and this method is also responsible for generating responses to all seven HTTP methods, GET, POST, DELETE,PUT, HEAD, OPTIONS,.

Thia is followed by the destruction phase. The jspDestroy() method is the jsp equivalent of the destroy method. Override jspDestroy when you need to perform any cleanup such as releasing database connections or closing any open files.

Public void jspDestroy(){

//Clean up Code..

}

**Java Server Pages – Syntax.**

**a). The 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.

**<% code fragment %>**

The xml equivalent of the above syntax as follows,

<jsp:scriptlet>

code fragment

</jsp:scriptlet>

Any text, HTML tags, or JSP elements you write must be outsides the scriplet.

<html>

<head><title>Hello World</title></head>

<body>

<%

out.println(“Your IP Address is “+request.getRemoteAddr());

%>

</body>

</html>

**Declaration of Variables**

The syntax for jsp declaration, you must declare the variable or method before you can use it in the code later.

<%! declaration; [declaration;]+…%>

XML equivalent of above syntax

<jsp:declaration>

code fragment

</jsp:declaration>

<%! int i=0;%>

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

**JSP Expression**

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

**<html>**

**<head><title>Expressions</title></head>**

**<body>**

**<p>Today’s Date is:<%=(new java.util.Date()).toLocaleString()%></p>**

**</body>**

**</html>**

**Comments→ <%-- This is A Comment –%>**

**JSP Directives.**

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

**<% @ directive attribute=”value” % >**

There are three types of directive tag:-

<[%@page](mailto:%25@page) …%>

→ Defines page-dependent attributes, such as the scripting language, error page and buffering requirements.

<[%@include](mailto:%25@include) …%>

→ Includes a file during the translation phase.

<[%@taglib](mailto:%25@taglib)…. %>

→ Declares a tag library, containing custom actions, used in the page.

**JSP Actions.**

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

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:forward→ Forwards the Requester to a new page.**
5. **Jsp:plugin→ Generates browser-specific code that makes an OBJECT for the java plugin.**
6. **Jsp:element→ Defines XML elements dynamically.**
7. **Jsp:body→ Deines dynamically-defines XML element’s body.**
8. **Jsp:text→ Used to write template text in JSP pages and documents.**

**JSP Implicit Objects.**

JSP supports nine automatically defined variables, which are also caleed implicit objects. Example:

**Request→ This is the HttpServletRequest object associated with the request.**

**Response→ This is the HttoServletResponse object with the response to the client.**

**Session→ This is the HttpSession object associated with the request.**

**Application→ This is the servletContext associated with the application context.**

**PageContext→ This encapsulates use of server-specific features like higher performance JspWriters.**

**Exception→ Objects allows the exception of data to be accessed by designated JSP.**

**The <jsp:include> Action,**

The action lets you insert files into the page being generated.

**<jsp:include page =”relative url” flush = “true”/>**

**Page→ The Relative URL of the page to be included.**

**Flush→ The boolean attribute that determines whether the included resource has its buffer flushed before it is included.**

**Example:**

**Define two jsp files a). date.jsp and main.jsp**

**Date.jsp**

**<p>**Today’s Date:<%=(new java.util.Date()).toLocaleString()%></p>

**Main.jsp**

<html>

<head>

<title> The Include Action</title>

</head>

<body>

<center>

<h2> The Include Action Example</h2>

<jsp:include page =”date.jsp” flush =”true”/>

</center>

</body>

</html>

**Defining a Test Bean:.→ What is a Java Bean??**

**package action;**

**public class TestBean {**

**private String message = “ No Message Specified”**

**public String getMessage(){**

**return message;**

**}**

**public void setMessage(String message){**

**this.message=message;**

**}**

**}**

**<html>**

**<head>**

**<title>Java Beans</title>**

**</head>**

**<body>**

**<center>**

**<jsp:useBean id =”test” class= “action.TestBean”/>**

**<jsp:setProperty name = “test” property= “message” value = “Trial JSF” />**

**<jsp:getProperty name = “test” property = “message”/>**

**</center>**

**</body>**

**</html>**

**Implicit Objects.**

These are java objects that the JSP container makes available for developers in each page and the developer can call them directly without being explicitly declared. → **Predefined Variables.**

**See Above implicit objects,**

**a). The Request Object.**

The request object is an instance of the **javax.servlet.http.HttpServletRequest** object. Each time a client requests a page in the jsp engine creates a new object to represent that request.

**b). The out Object.**

The out implicit object is an instance of **javax.servlet.jsp.JspWriter** object and is used to send content in a response.

**c). The Session Object.**

It is an instance of **javax.servlet.http.HttpSession** and behaves exactly the same way as objects under the java servlet.

**Java Server Pages – Client Request.**

We learn on what happens when a browser requests for a web page, → The browser sends a lot of information to the web server. The info cannot be read directly since it travels as a part of the header of HTTP request.

**HTTP Header Request Example.**

Following is the example which uses **getHeaderNames() method** of the **HtpServletRequest** to read the HTTP header information. This method returns an enumeration that contains the header information associated with the current HTTP request.

**<%@ page import = “java.io,\*, java.util.\*” %>**

**<html>**

**<head>**

**<title> HTTP Header Request Example</title>**

**</head>**

**<body>**

**<center>**

**<h2> Header Request Example</h2>**

**<table width = “100%” border = “1” align = “center”>**

**<tr bgcolor = “#949494”>**

**<th>Header Name</th>**

**<th> Header Value(s)</th>**

**</tr>**

**<%**

**Enumeration headerNames = request.getHeaderNames();**

**while(headerNames.hasMoreElements()){**

**String paranName = (String)headerNames.nextElement();**

**out.print(“<tr><td>”+paranName+ “</td>\n”);**

**String paramValue = request.getHeader(paramName);**

**out.print(“<td”+paramValue+ “</td></tr>\n”);**

**}**

**%>**

**</table>**

**</center>**

**</body>**

**</html>**

**Client → Response.**

The response typically consists of a status line, some response headers, a blank line and the document.

**Allow→** This header specifies the request methods (**GET, POST, etc.)** that the server supports.

**Cache-Control→** The header specifies the circumstances in which the response document can be safely cached. It can have values **public, private or non-cache** etc.

**HTTP Header Response Example.→ ← ←**

**/\* Simulate the Digital Clock \*/**

**<%@ page import = “java.io.\*,java.util.\*” %>**

**<html>**

**<head>**

**<title> Auto Refresh Header Example</title>**

**</head>**

**<body>**

**<center>**

**<h2> Auto Refresh Header Example</h2>**

**<%**

**response.setIntHeader(“Refresh”, 5);**

**Calendar calendar = new GregorianCalendar();**

**String am\_pm;**

**int hour = calendar.get(Calendar.HOUR);**

**int minute = calendar.get(Calendar.MINUTE);**

**int second = calendar.get(Calendar.SECOND);**

**if(calendar.get(Calendar.AM\_PM)==0){**

**am\_pm = “AM”;**

**}else{**

**am\_pm = “PM”;**

**String CT = hour + “:” + minute + “:” +second + “ ” +am\_pm;**

**out.println(“Current Time is :” +CT + “\n”);**

**%>**

</center>

</body>

</html>

**Java Server Pages → Handling Forms**

**The GET and POST 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.

[**http://www.test.com/hello?key1=value1&key2=value2**](http://www.test.com/hello?key1=value1&key2=value2)

The GET method is the default method to pass information from the browser to the web server and it produces a long string that appears in your browser’s **Location:box.** The GET method has a size limitation of **1024** characters in a request String.

It handles this type of request using the **getParameter()** method to read simple parameters and **getInputStream()** method to read binary data stream coming from the client.

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

* **GetParameter() →** You call request.getParameter() method to get the value of a form parameter.
* **GetParameterValues() →** Call this method if the parameter appears more than once and returns multiple values, like checkbox.
* **GetParameterNames() →** Call this method if you want a completer list of all parameters in the current request.
* **GetInputStream()→** Call this method to read binary data stream from the client.

**<html>**

**<head>**

**<title>Using the GET method to Read Form-Data</title>**

**</head>**

**<body>**

**<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>**

**Passing CheckBox Data To JSP Program.**

**<html>**

**<body>**

**<form action= “ main.jsp” method= “post” target= “\_blank”>**

**<input type = “checkbox” name= “maths” checked = “Checked”/>Maths.**

**<input type= “checkbox” name = “Physics” checked = “checked”/> Physics**

**<input type = “checkbox” name = “Chemistry” checked = “checked”/> Che**

**<input type = “submit” value = “Select Subject”/>**

**</form>**

**</body>**

**</html>**

**JAVA SERVER PAGES :: FILTERS.**

**Why ?**

* To intercept requests from a client before they access a resource at the back-end.
* To manipulate responses from server before they are sent back to the client.

**Types of Filters.**

* **Authentication Filters.**
* **Data Compression Filters.**
* **Encryption Filters.**
* **Filters that trigger resource access events.**
* **Image Conversion Filters. → ←**
* **Logging and Auditing Filters.**
* **MIME-TYPE Chain Filters.**
* **Tokenizing Filters.**
* **XSL/T Filters that transform XML Content.**

Filters are deployed in the deployment descriptor file web.xml and then map to either sevlet or jsp names or url patterns in your deployment descriptor

**Servlet Filter Methods.**

**Public void doFilter(ServletRequest, ServletResponse, FilterChain)→** This method is called by the container each time a request/response pair is passed through the chain due to a client request for a resource at the end of the chain.

**Public void init (FilterConfig filterconfig) →** This method is called by the web container to indicate to a filter that is being placed into service.

**Public void destroy() →** This method is called by the web container to indicate to a filter that is being taken out of service.

**Example:-**

import java.io\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.util.\*;

public class LogFilter implements Filter{

public void init(FilterConfig config ) throws ServletException{

String testParam = config.getInitParameter(“test-param”);

System.out.println(“Test Param: ” + testParam);

}

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {

String ipAdress = request.getRemoteAddr();

System.out.println(“IP” + ipAddress+ “, Time” + new Date().toString();

chain.doFilter(request, response);

public void destroy(){

}

}

**Session Tracking.**

HTTP is a ‘stateless’ protocol which means that each time a client retrieves a webpage, the client opens a separate connection to the web server and the server automatically does not keep any record of previous client request.

A web server can send a hidden html form field along with a unique id as follows ; -

**<input type = “hidden” name= “sesssionid” value = “12345”>**

The entry means that, when the form is submitted, the specified name and value are automatically included in the **GET** and **POST** data. Each time the web browser sends the request back, the session\_id can be used to keep the track of different web browsers.

**Creating an File Upload Form.**

The form method attribute should be set to **POST** .

The form enctype should be set to multipart/form-data.

Form action attribute should be set to a JSP file which would handle uploading the file to the back end server. Like for example **uploadfile.jsp**

**use→ <input type = “file” name = “file” size = “50”/>**