# UNIT-5

**JSP**(Java server page)
The full form of JSP is Java Server Pages. It is a server side technology which is used to create dynamic webpages. It has been specially created to develop Java web applications.
JSP technology can access Java API (Application Program Interface) which makes it easier to create web applications based on Java.
JSP technology works completely opposite to Servlet technology. In Servlet technology we write HTML script inside Java program whereas in JSP technology we write Java program in HTML script. Although a JSP page also generates servlet as output but coding in JSP is much easier than servlet.

## Difference between Servlet And JSP

| JSP | Servlet |
|---|---|
| JSP pages are slower than servlets because it takes time for them to be compiled and converted into servlets. | Servlets are faster than JSP because they are pre-compiled and are executed directly. |
| It provides automatic page compilation so there is no need to deploy the application again and again. Changes made in JSP are live. | Whenever you make any changes in servlet programs, you have to compile and run it again and again. |
| JSP only handles HTTP (Hyper Text Markup Language) requests. | Servlet handles all types of protocol requests. |
| Coding in JSP is very easy. | Servlet programs are very large and their coding is also very difficult. |
| In this you add a Java program to an HTML script. | In this you add HTML script to the Java program. |
| In JSP logic and design are separated. | Logic and design are combined in Servlets. |

## Advantages of JSP

1. **Easy to learn -** JSP is very easy to learn because you use JSP tags just like normal HTML tags. If you can easily create applications in HTML and Java, then you can easily create web applications through JSP too.

2. **Easy to manage -** With the help of JSP, the logic part and designing part are separated. Because you do the designing with the help of HTML and CSS and perform the logic with the help of Java. Therefore, it is very easy to manage a JSP application.

3. **Automatic page compilation -** In JSP you do not have to compile the program again and again, as soon as you make any changes, the changes are automatically applied. JSP provides you automatic page compilation. This saves you from the problem of deploying the application again and again.

4. With JSP you get access to all Java APIs. For example, with JSP you get access to JDBC API through which your application can communicate with the database.

5. **Reliable & Secure –** As I told you earlier, due to the support of Java, JSP is reliable and secure compared to other technologies.

## Working of JSP (Java Server Pages)

4 elements are very important in the working of Java Server Pages:-

1. Web browser
2. Web server
3. JSP engine
4. Servlet engine


The web browser requests a JSP page, the request goes directly to the web server. The web server identifies through the .jsp extension that this is a request for a JSP page that is stored on the web server.

By identifying this page, the web server passes it to the JSP engine for further processing. The JSP engine is installed on the web server. The JSP engine is already enabled with the Apache web server.

The JSP engine converts this JSP page into a servlet. After this, this servlet is passed to the servlet engine. The servlet engine processes this servlet and generates an HTML page as output. This HTML page is passed to the web server. Finally, the web server sends this HTML page to the web browser.

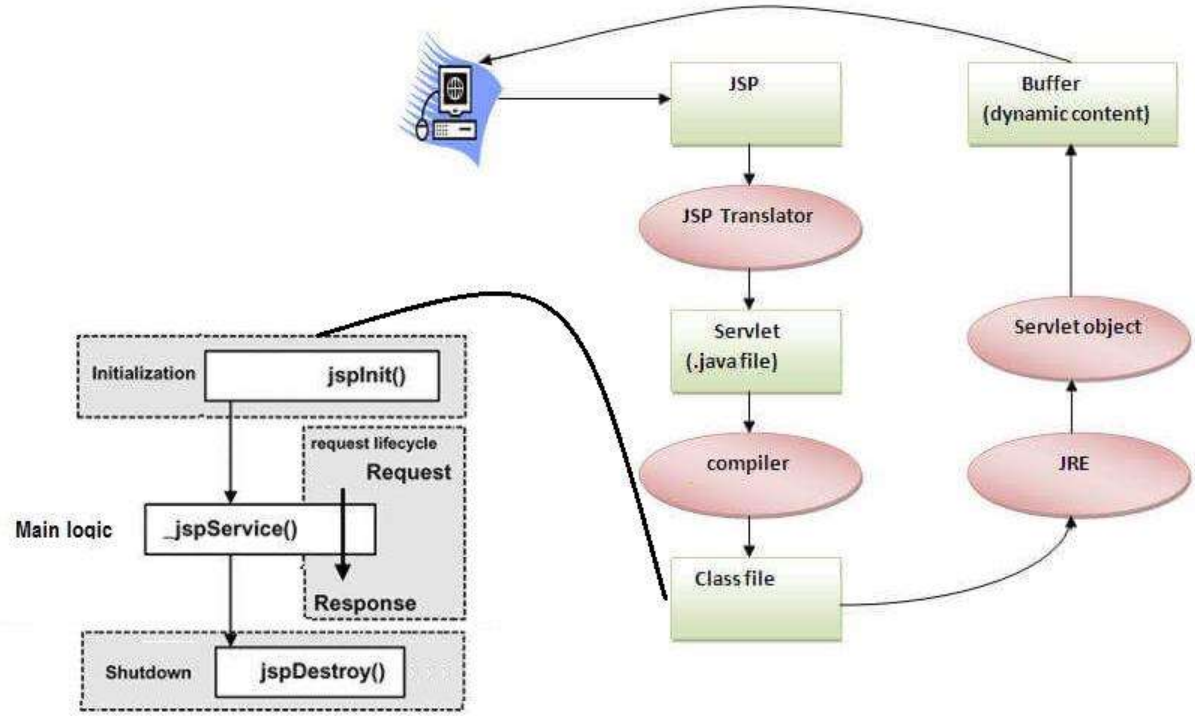
## JSP Life Cycle in Hindi

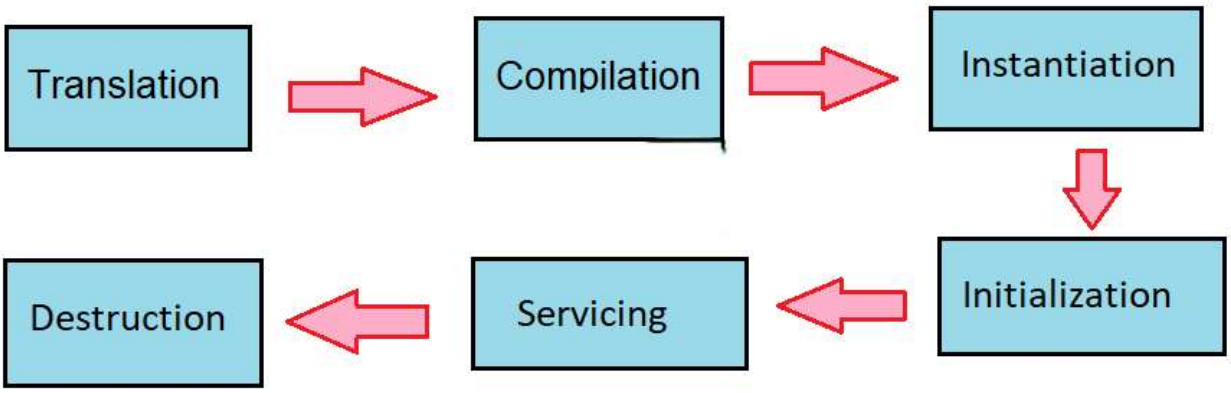
A JSP page is not processed like a normal web page. A JSP page goes through important phases before it is displayed to the client. These phases are called the JSP life cycle.
JSP life cycle is the process from creation of a JSP page to its destruction.

Whenever a client makes a request, the web server identifies it as a request for a JSP page through the .jsp extension of the file. To process the client's request, it is necessary to convert a JSP page into a servlet.
A request for a JSP page is received, this page is sent to the JSP engine. The JSP engine first converts it into a servlet class and then converts it into byte code.
After this the file is loaded into the memory and then jspInit(), jspService() and jspDestroy() methods are called respectively.

JSP page passes through the following 6 phases.

1. Translation
2. Compilation
3. Instantiation
4. Initialization
5. Servicing
6. Destruction






JSP Life Cycle

In each phase some processing is done with the JSP page. Let us now try to know about these phases in detail.

**Translation Phase**

This is the first phase of the JSP life cycle. when a request for a JSP page comes, this request is sent to the JSP engine. The JSP engine first checks the JSP syntax in the file, if the file is correct then it is translated into a servlet class.

Before translating a JSP page, the JSP engine checks whether there have been any changes in the JSP page or not. If there are no changes, the JSP engine skips this phase.

To detect changes, the JSP engine checks whether the servlet class is older than the JSP page. Because if any changes have been made, the JSP page would have been saved with a new date. This lets the JSP engine know that the servlet class is outdated and this page has to be translated back.

**Compilation Phase**

The second phase of the JSP life cycle is compilation. In this phase, the servlet class translated by the translation phase is compiled. After compilation, a .class file is generated. This is a byte code.

**Instantiation Phase**

After the byte code is generated, a JSP page enters the instantiation phase. In this phase, the .class file is loaded into the memory. In simple words, “In this phase, the object (instance) of the .class file is created.”

**Initialization Phase**

After the class is loaded into memory, a JSP page enters the initialization phase. In this phase, the jspInit() method is called by the JSP engine. This method is called only once in the page life cycle.

In this method you write the code that is to be executed before the JSP page is displayed. For example, you can establish a connection to the database before the JSP page is displayed. A web server starts giving response only after this method is called.

This method can also be overridden by programmers. By overriding it, you can initialize resources related to networks and databases in the program.

The general syntax of this method is given below.

```
public void jspInit()
{
//code to be executed
}
```

**Servicing Phase**
The _jspService() method is called by the JSP engine to respond to the request. This method is generated by the container component of the JSP engine.
The initial underscore (_) in this method means that you cannot override this method. The JSP code written by you goes into this method.
The general syntax of this method is given below.

```
public void _jspService (HttpServlet request, HttpServlet response) throws IOException, ServletException
{
// JSP code that you write with some other code
}
```

**Destruction**
The last phase of the JSP life cycle is destruction. In this phase, the jspDestroy() method is called by the JSP engine. If we want to destroy the JSP page, then this method is called. All network and database resources are freed through this method.
With the call of this method, the life cycle of a JSP page ends and page garbage is collected. You can call this method in any other phase also but before that _jspService() method is definitely called.
The general syntax of this method is given below.

```
public void jspDestroy()
{
// code to be executed
}
```

## Setting up Jsp Environment

Setting up the JSP environment involves preparing your system to create, run, and test JSP-based web applications. Below are the steps to set up and configure the environment:

### 1. Install Java Development Kit (JDK)

JSP is a Java-based technology, so you need the JDK to compile and run Java programs.

- **Download and Install JDK:**
  - Download the latest version of the JDK from the Oracle or OpenJDK website.
  - Install the JDK and configure the JAVA_HOME environment variable.
- **Verify Installation:**

  ```
  java -version
  javac -version
  ```

### 2. Install an Application/Web Server

JSP pages run on a web server that supports Java Servlets and JSP, such as **Apache Tomcat**, **Jetty**, or **WildFly**.

### Install Apache Tomcat (Recommended for Beginners):

1. **Download Apache Tomcat:**
   - Download the latest stable release from the Tomcat website.
2. **Extract the Files:**
   - Extract the downloaded archive to a directory (e.g., C:\tomcat or /usr/local/tomcat).
3. **Set Environment Variables (Optional):**
   - Add CATALINA_HOME (Tomcat installation directory) to your system's environment variables.
4. **Start Tomcat Server:**
   - Navigate to the bin directory and start the server:
     - On Windows: startup.bat
5. **Access Tomcat:**
   - Open a web browser and navigate to http://localhost:8080 to verify the server is running.

## 3. Configure the Development Environment

You can use an Integrated Development Environment (IDE) to make development easier.

*Install an IDE (Eclipse/IntelliJ/NetBeans):*

1. **Download and Install IDE:**
   - Eclipse
   - IntelliJ IDEA
   - NetBeans
2. **Configure the Web Server in IDE:**
   - In Eclipse:
     1. Go to Window > Preferences > Server > Runtime Environments.
     2. Click Add, choose Apache Tomcat, and specify the installation directory.

## 4. Create a JSP Web Application

1. **Set Up Directory Structure:** Create a folder structure for your JSP project:

```
project/
├── WEB-INF/
│   ├── web.xml   # Deployment descriptor
│   ├── classes/  # Compiled Java classes
│   └── lib/      # Dependencies (JAR files)
└── index.jsp     # JSP file
```

2. **Write a JSP File:** Create a simple index.jsp file:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<!DOCTYPE html>
<html>
<head>
   <title>JSP Example</title>
</head>
<body>
   <h1>Welcome to JSP!</h1>
   <%
      out.println("Today's date is: " + new java.util.Date());
   %>
</body>
</html>
```

3. **Configure web.xml:** Add a basic deployment descriptor file in WEB-INF/:

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
         version="3.0">
   <welcome-file-list>
      <welcome-file>index.jsp</welcome-file>
   </welcome-file-list>
</web-app>
```

## 4. Deploy the Application

1. **Copy the Project to Tomcat:**
   - Place your project folder in webapps under the Tomcat installation directory.
2. **Start the Server:**
   - Restart Tomcat if it's already running.
3. **Access the Application:**
   - Open a browser and navigate to http://localhost:8080/project (replace project with your project folder name).

## 5. Testing and Debugging

- **Testing:**
  - Verify the output in the browser matches the JSP content.
- **Debugging:**
  - Check Tomcat logs in the logs folder for errors.

# JSP implicit objects

JSP implicit objects are Java objects that are created by the web container and can be used in all JSP pages.
JSP implicit objects are created in the translation phase (translation from JSP to servlet).
JSP implicit objects can be called directly without explicitly declaring and initializing them.
JSP implicit objects are also called pre-defined variables.
The advantage of JSP over service is that JSP has implicit objects through which we can do programming easily.
There are 9 types of JSP implicit objects in it:-
1:- out

2:- request

3:- response

4:- session

5:- application

6:- config

7:- PageContext

8:- page

9:- exception

**1:- out implicit object:-**

The out implicit object is used to write content to the buffer and this content is sent to the client as output.
Through out object we can access the output stream of the servlet.
out is an instance of the javax.servlet.jspWriter object.
While working with servlet we need jspWriter object.
The JspWriter object has the same methods as the java.io.PrintWriter class, but jspwriter also has some additional methods that handle buffering.
**For example: –** In this example the division of two numbers is displayed.
**File name:-exout.jsp**

```
<body>
<%
out.println(“simple program of out implicit object<br>”);
int a= 2400, b= 60;

out.println(“division of ” + a+ ” and ” + b+ ” is ” + a/b+ “<br>”);

%>

<body>
```

**2:- request implicit object:-**

The request implicit object is an instance of the Javax.servlet.http.HttpServletRequest object.
The request object is created by the container for each request.
This object is used by the developer to extract the data received from the client such as: username, password, parameter, header information, remote address, server name, server port, and request attributes etc.
It uses getParameter() to access the request parameter.
**For example:-**
In this example, we will receive information from the user on the Request.html page and will display the same information on the ExRequest.jsp page through the request implicit object.
**File name:- request.html**

```
<body>
<h2>"simple program of request implicit object"</h2>
<form action="exrequest.jsp">
Enter User Name: <input type="text" name="uname" > <br>
Enter Password: <input type="text" name="pass" > <br>
<input type="submit" value="login">
</form>
</body>
```

**file name:- exrequest.jsp**

```
<body>
<%
String username=request.getParameter("uname");
String password=request.getParameter("pass");
out.print("Name: "+username+" Password: "+password);
%>
</body>
```

**3:- response implicit object:-**

The response implicit object represents the response to be returned to the client.
The response object is an instance of the javax.servlet.http.HttpServletResponse object.
The response object is created by the container for each request.
The response object is used to send cookies to the client, set the response content type, and redirect the response to another page.
**For example:-**
**file name:- exresponse.jsp**

```
<html>
<head>
<title>simple program of response jsp implicit objects</title>
</head>
<body>
<form action=”hello.jsp”>
<input type=”text” name=”uname”>
<input type=”submit” value=”go”><br/>
</form>
</body>
</html>
```

**file name:- hello.jsp**

```
<%
response.sendRedirect(“https://ehindistudy.com”);
%>
```

**4:- session implicit object:-**

The session object is an instance of the javax.servlet.http.HttpSession object. It is used to set, get, and remove attributes and to retrieve session information.
**5:- application implicit object:-**

The application object is an instance of javax.servlet.ServletContext and is used to get context information and attributes in JSP.
The application object represents the JSP page throughout its lifecycle. This object is created when the JSP page is initialized and is removed only when the JSP page is removed.
The application object is a global object and is used by all JSPs in the project. There can be only one application object in the entire project.
**6:- config implicit object:-**

The config object is an instance of javax.servlet.ServletConfig. It is a servlet configuration object and is mostly used to get configuration information such as: servlet context, server name, configuration parameters, etc.

It is created by a container for each JSP page.

**7:- PageContext object:-**

The PageContext implicit object is an instance of the javax.servlet.jsp.PageContext object. The pageContext object represents the entire JSP page.
PageContext implicit object provides methods through which we can access other implicit objects. It has more than 40 methods.
The PageContext object has methods like getPage, getRequest, getResponse etc.
The PageContext object is used to get, set and remove attributes from a particular scope.
There are four types of scope:-
1:- PAGE_SCOPE
2:- REQUEST_SCOPE
3:- SESSION_SCOPE
4:- APPLICATION_SCOPE

**8:- page implicit object:-**

The page implicit object is an instance of the java.lang.object class.
It represents the current JSP page i.e. it is a reference to the current servlet instance.
It can also be said that it is an object that represents the entire JSP page. It is used very rarely in JSP, almost negligible.
JSP implicit objects

**9:- exception implicit object:-**

The exception implicit object is an instance of the java.lang.Throwable class.
The exception object is used for error handling. Through which the error message is displayed.
This object is available only for JSP pages that have isErrorPage set to true.
It has two important methods:-
1:- getMessage()- It is used to print the error message for exception.
2:- printStackTrace(output):- It is used to print the execution stack.

# Custom Tags in JSP

Custom tags are user-defined action tags that can be used within Java Server Pages. A tag handler is associated with each tag to implement the operations. Therefore, it separates the business logic from JSP and helps avoid the use of scriptlet tags. The scriptlet tag embeds java code inside the JSP page itself rendering the page difficult to understand therefore, it is better to avoid the use of scriptlet.

## Creating custom tags

**Syntax**

**empty custom tag(without body)**

<prefix : suffix attribute = “value”/>

**Non-empty custom tag**

<prefix : suffix attribute = “value”>body</prefix : suffix>

Three components are required to develop custom tags .

1. Tag Handler
2. TLD(Tag Library Descriptor) file
3. Taglib directive in jsp file

## Create the Tag Handler Class

The tag handler class contains the logic to execute when the custom tag is invoked. It typically extends the `TagSupport` or `SimpleTagSupport` class.

It provides the following methods :

- int doEndTag()
- int doStartTag()
- Tag getParent()
- void release()
- void setPageContext(PageContext pc)
- void setParent(Tag t)

Example:

```
import javax.servlet.jsp.tagext.TagSupport;
import javax.servlet.jsp.JspException;
import java.io.IOException;

public class HelloTag extends TagSupport {
    @Override
    public int doStartTag() throws JspException {
        try {
            pageContext.getOut().write("Hello, Custom Tag!");
        } catch (IOException e) {
            throw new JspException("Error: " + e.getMessage());
        }
        return SKIP_BODY; // Skip the body content of the tag
```

```
    }
}
```

## 2. TLD(Tag Library Descriptor)

It is a file saved with a .tld extension that contains a set of related tags mapped to their respective tag handlers along with their description such as the name of the tag, attributes of the tag, etc.

<Tag> consists of the following sub-elements.

- <body-content>: describes the type of body content: empty, JSP, scriptless, tagdependent
- <name>: used to provide a unique name to the tag.(suffix)
- <tagclass>: used to provide tag handler class for the tag
- <attribute>: used to provide attributes for the tag.

The TLD file defines the custom tag and its attributes. It is usually placed in the `WEB-INF` directory.

**Example:** hello.tld

```
<?xml version="1.0" encoding="UTF-8" ?>
<taglib xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                        http://java.sun.com/xml/ns/javaee/web-jsptaglibrary_2_1.xsd"
        version="2.1">
    <tlib-version>1.0</tlib-version>
    <short-name>HelloTag</short-name>
    <uri>/WEB-INF/hello.tld</uri>

    <tag>
        <name>hello</name>
        <tag-class>HelloTag</tag-class>
        <body-content>empty</body-content>
    </tag>
</taglib>
```

## 3. Taglib Directive

Taglib directive is used to access a particular tag library within a JSP. It specifies the URI of the tag library and its prefix.

**Syntax:**
<%@taglib uri=” ” prefix=” “%>

**Example:** example.jsp.

```
<%@ taglib uri="/WEB-INF/hello.tld" prefix="custom" %>
<html>
<head>
   <title>Custom Tag Example</title>
</head>
<body>
   <custom:hello />
</body>
</html>
```

# JSTL (JSP Standard Tag Library)

The JSP Standard Tag Library (JSTL) represents a set of tags to simplify the JSP development. The **JSP Standard Tag Library (JSTL)** is a collection of custom tags that are part of the Java EE standard. JSTL simplifies common tasks in JSP, such as iteration, conditionals, internationalization, and database access, allowing developers to write less scriptlet code.

## Advantage of JSTL

1. **Fast Development** JSTL provides many tags that simplify the JSP.
2. **Code Reusability** We can use the JSTL tags on various pages.
3. **No need to use scriptlet tag** It avoids the use of scriptlet tag.

JSTL Tags

There JSTL mainly provides five 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**. |
| Function 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**. |
| Formatting tags | The Formatting tags provide support for message formatting, number and date formatting, etc. The URL for the Formatting tags is **http://java.sun.com/jsp/jstl/fmt** and prefix is **fmt**. |
| XML tags | The XML tags provide flow control, transformation, etc. The URL for the XML tags is **http://java.sun.com/jsp/jstl/xml** and prefix is **x**. |
| 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**. |

For creating JSTL application, you need to load the jstl.jar file.

Download the jstl.jar file

Download the jstl1.2.jar file

**JSTL Core Tags**

The JSTL core tag provides variable support, URL management, flow control etc. The syntax used for including JSTL core library in your JSP is:

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

JSTL Core Tags List

| Tags | Description |
|---|---|
| c:out | It display the result of an expression, similar to the way <%=...%> tag work. |

| | |
|---|---|
| c:import | It Retrives relative or an absolute URL and display the contents to either a String in 'var',a Reader in 'varReader' or the page. |
| c:set | It sets the result of an expression under evaluation in a 'scope' variable. |
| c:remove | It is used for removing the specified scoped variable from a particular scope. |
| c:catch | It is used for Catches any Throwable exceptions that occurs in the body. |
| c:if | It is conditional tag used for testing the condition and display the body content only if the expression evaluates is true. |
| c:choose, c:when, c:otherwise | It is the simple conditional tag that includes its body content if the evaluated condition is true. |
| c:forEach | It is the basic iteration tag. It repeats the nested body content for fixed number of times or over collection. |
| c:forTokens | It iterates over tokens which is separated by the supplied delimeters. |
| c:param | It adds a parameter in a containing 'import' tag's URL. |
| c:redirect | It redirects the browser to a new URL and supports the context-relative URLs. |
| c:url | It creates a URL with optional query parameters. |

JSTL Function Tags

The JSTL function provides a number of standard functions, most of these functions are common string manipulation functions. The syntax used for including JSTL function library in your JSP is:

1. <%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>

| JSTL Functions | Description |
|---|---|
| fn:contains() | It is used to test if an input string containing the specified substring in a program. |
| fn:containsIgnoreCase() | It is used to test if an input string contains the specified substring as a case insensitive way. |
| fn:endsWith() | It is used to test if an input string ends with the specified suffix. |
| fn:escapeXml() | It escapes the characters that would be interpreted as XML markup. |
| fn:indexOf() | It returns an index within a string of first occurrence of a specified substring. |
| fn:trim() | It removes the blank spaces from both the ends of a string. |
| fn:startsWith() | It is used for checking whether the given string is started with a particular string value. |
| fn:split() | It splits the string into an array of substrings. |

| | |
|---|---|
| fn:toLowerCase() | It converts all the characters of a string to lower case. |
| fn:toUpperCase() | It converts all the characters of a string to upper case. |
| fn:substring() | It returns the subset of a string according to the given start and end position. |
| fn:substringAfter() | It returns the subset of string after a specific substring. |
| fn:substringBefore() | It returns the subset of string before a specific substring. |
| fn:length() | It returns the number of characters inside a string, or the number of items in a collection. |
| fn:replace() | It replaces all the occurrence of a string with another string sequence. |

## JSTL Formatting tags

The formatting tags provide support for message formatting, number and date formatting etc. The url for the formatting tags is **http://java.sun.com/jsp/jstl/fmt** and prefix is **fmt.**

The JSTL formatting tags are used for internationalized web sites to display and format text, the time, the date and numbers. The syntax used for including JSTL formatting library in your JSP is:

1. <%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>

| Formatting Tags | Descriptions |
|---|---|

| | |
|---|---|
| fmt:parseNumber | It is used to Parses the string representation of a currency, percentage or number. |
| fmt:timeZone | It specifies a parsing action nested in its body or the time zone for any time formatting. |
| fmt:formatNumber | It is used to format the numerical value with specific format or precision. |
| fmt:parseDate | It parses the string representation of a time and date. |
| fmt:bundle | It is used for creating the ResourceBundle objects which will be used by their tag body. |
| fmt:setTimeZone | It stores the time zone inside a time zone configuration variable. |
| fmt:setBundle | It loads the resource bundle and stores it in a bundle configuration variable or the named scoped variable. |
| fmt:message | It display an internationalized message. |
| fmt:formatDate | It formats the time and/or date using the supplied pattern and styles. |

## JSTL XML tags

The JSTL XML tags are used for providing a JSP-centric way of manipulating and creating XML documents.

The xml tags provide flow control, transformation etc. The url for the xml tags is **http://java.sun.com/jsp/jstl/xml** and prefix is x. The JSTL XML tag library has custom tags used for interacting with XML data. The syntax used for including JSTL XML tags library in your JSP is:

1. <%@ taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x" %>
Before you proceed further with the examples, you need to copy the two XML and XPath related libraries into the <Tomcat Installation Directory>\lib:

**Xalan.jar**: Download this jar file from the link:

1. http://xml.apache.org/xalan-j/index.html
**XercesImpl.jar**: Download this jar file from the link:

1. http://www.apache.org/dist/xerces/j/

| XML Tags | Descriptions |
|---|---|
| x:out | Similar to <%= ... > tag, but for XPath expressions. |
| x:parse | It is used for parse the XML data specified either in the tag body or an attribute. |
| x:set | It is used to sets a variable to the value of an XPath expression. |
| x:choose | It is a conditional tag that establish a context for mutually exclusive conditional operations. |
| x:when | It is a subtag ofthat will include its body if the condition evaluated be 'true'. |
| x:otherwise | It is subtag ofthat followstags and runs only if all the prior conditions evaluated be 'false'. |

| | |
|---|---|
| x:if | It is used for evaluating the test XPath expression and if it is true, it will processes its body content. |
| x:transform | It is used in a XML document for providing the XSL(Extensible Stylesheet Language) transformation. |
| x:param | It is used along with the transform tag for setting the parameter in the XSLT style sheet. |

JSTL 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**.

The SQL tag library allows the tag to interact with RDBMSs (Relational Databases) such as Microsoft SQL Server, mySQL, or Oracle. The syntax used for including JSTL SQL tags library in your JSP is:

1. <%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>

JSTL SQL Tags List

| SQL Tags | Descriptions |
|---|---|
| sql:setDataSource | It is used for creating a simple data source suitable only for prototyping. |
| sql:query | It is used for executing the SQL query defined in its sql attribute or the body. |
| sql:update | It is used for executing the SQL update defined in its sql attribute or in the tag body. |

| | |
|---|---|
| sql:param | It is used for sets the parameter in an SQL statement to the specified value. |
| sql:dateParam | It is used for sets the parameter in an SQL statement to a specified java.util.Date value. |
| sql:transaction | It is used to provide the nested database action with a common connection. |

## Processing Input and Output

In JavaServer Pages, Form processing is the fundamental aspect of web development, and it can allow users to interact with websites by submitting data. It can handle form submissions involving extracting the data from the HTTP requests and generating the appropriate responses.
Form Processing in JSP
Form processing in the JavaServer Pages involves receiving data submitted through the HTML forms extracting this data from the request object and generating the response based on the received data.
*Key Terminologies:*

- **JavaServer Pages (JSP)**: This is a technology that can help software developers create dynamic web pages in HTML, XML, and other document types. It can allow Java code to be inserted into HTML pages and that code executed on the server before the pages are served on client's websites.
- **Form processing**: This refers to the handling of data transmitted via HTML forms on web pages. It can insert customer form data into HTML requests, and process this data, generating responses.
- **HTTP Methods:**
    - **GET:** HTTP GET requests are used to request data from the specified object and when a form is submitted using the GET method and the form data is loaded into the query parameters of the URL.
    - **POST**: This can be an HTTP POST request if the data to be processed is sent to the specified object and when the form is submitted using the POST method and the form data is sent to the body of the HTTP request.
- **Request Object**: A request object in a JSP can represent an HTTP request made by a client and provide methods for retrieving information sent by the client such as form parameters, headers, and cookies

- **Response Object**: The response object in the JSP can represent an HTTP response to be sent back to the client, and provide a response header, and methods for setting cookies and delivering content to the client browser.
- **Parameter**: It is the values sent to the server as part of an HTTP request and in form processing the parameters typically can represent the form field values submitted by the users.

In JavaServer Pages (JSP), form processing involves handling data submitted through HTML forms. This can be done using various HTTP methods like GET and POST. Here are examples of processing forms using both GET and POST methods.

**JSP Form Processing Using GET Method**

The GET method is one of the HTTP methods used to send form data to a web server. When this happens, the form is submitted using the GET method and the form data is inserted into the query parameters of the URL.

**Step-by-Step implementation of GET Method**

**Step 1**: Create the dynamic web project using eclipse and its project named as **jsp-GET-demo**.

***Note: If you beginner click here to know the process of creating the dynamic web project.***

**Step 2**: After creating a JSP form named **index.jsp** and this form includes input fields to store user data and the form attributes can be set on the JSP page to customize the form to be submitted and set the method attribute enter "GET".

HTML

```
<%@ page language="java" contentType="text/html; charset=UTF-8"

    pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

    <title>GET Form Processing</title>

    <style>
```

```
/* CSS styles for better visibility */

body {

    font-family: Arial, sans-serif;

    background-color: #f4f4f4;

}


.container {

    width: 50%;

    margin: 50px auto;

    padding: 20px;

    border: 1px solid #ccc;

    border-radius: 5px;

    background-color: #fff;

    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}



h2 {

    color: #333;
```

```
    text-align: center;
}

form {
    margin-top: 20px;
}

label {
    display: block;
    margin-bottom: 5px;
    font-weight: bold;
}

input[type="text"]
input[type="email"]
input[type="submit"] {
    width: 100%;
```

```
        padding: 10px;
        margin-bottom: 10px;
        border: 1px solid #ccc;
        border-radius: 5px;
        box-sizing: border-box; /* Ensure padding and border are included in width */
    }

    input[type="submit"] {
        background-color: #007bff;
        color: #fff;
        cursor: pointer;
    }

    input[type="submit"]:hover {
        background-color: #0056b3;
    }
  </style>
</head>
```

```
<body>

    <div class="container">

        <h2>Submit Form (GET Method)</h2>

        <form action="processGet.jsp" method="GET">

            <label for="name">Name:</label>

            <input type="text" id="name" name="name" required><br>

            <label for="email">Email:</label>

            <input type="email" id="email" name="email" required><br>

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

        </form>

    </div>

</body>

</html>
```

**Step 3**: Create a new JSP page and name it **processGet.jsp** which processes the submission process and retrieves the form data from the request object using the **getParameter()** method on this page. This method can name the form field as its parameter and return the value entered by the user.
HTML

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
```

```
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>GET Form Processing</title>
    <link rel="stylesheet" type="text/css" href="WEB-INF/css/style.css">
    <style>
        /* Additional CSS for better visibility */
        body {
            font-family: Arial, sans-serif;
            background-color: #f4f4f4;
        }

        .container {
            width: 50%;
            margin: 50px auto;
            padding: 20px;
```

```
        border: 1px solid #ccc;
        border-radius: 5px;
        background-color: #fff;
        box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    }

    h2, h3 {
        color: #333;
        text-align: center;
    }

    p {
        text-align: center;
    }
  </style>
</head>
<body>
```

```
    <div class="container">

        <h2>GET Form Processing</h2>

        <%-- Retrieving parameters from the request --%>

        <% String name = request.getParameter("name"); %>

        <% String email = request.getParameter("email"); %>



        <h3>Hello, <%= name %></h3>

        <p>Your email is: <%= email %></p>

    </div>

</body>

</html>
```

**JSP Form Processing Using Post Method**
Below is the step-by-step implementation of POST method.
**Step-by-Step implementation of POST Method**
**Step 1**: Create the dynamic web project using eclipse and its project named as **jsp-POST-demo**.
**Step 2**: Create a JSP form named index.jsp and this form includes input fields to store user data and form attributes that can be stored on a JSP page to configure the form to be submitted and set the method attribute to "POST ".

```
<%@ page language="java" contentType="text/html; charset=UTF-8"

    pageEncoding="UTF-8"%>
```

```
<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

    <title>Form Processing</title>

    <link rel="stylesheet" type="text/css" href="WEB-INF/css/style.css">

    <style>

        /* Additional CSS for better visibility */

        body {

            font-family: Arial, sans-serif;

            background-color: #f4f4f4;

        }



        .container {

            width: 50%;

            margin: 50px auto;

            padding: 20px;

            border: 1px solid #ccc;
```

```
    border-radius: 5px;
    background-color: #fff;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

h2 {
    color: #333;
    text-align: center;
}

form {
    margin-top: 20px;
}

label {
    display: block;
    margin-bottom: 5px;
```

```
    font-weight: bold;
}

input[type="text"],
input[type="email"],
input[type="checkbox"],
input[type="radio"],
input[type="submit"] {
    width: 100%;
    padding: 10px;
    margin-bottom: 10px;
    border: 1px solid #ccc;
    border-radius: 5px;
    box-sizing: border-box; /* Ensure padding and border are included in width */
}

input[type="submit"] {
    background-color: #007bff;
```

```
            color: #fff;
            cursor: pointer;
        }

        input[type="submit"]:hover {
            background-color: #0056b3;
        }
    </style>
</head>
<body>
    <div class="container">
        <h2>Submit Form (POST Method)</h2>
        <form action="processPost.jsp" method="POST">
            <label for="name">Name:</label>
            <input type="text" id="name" name="name" required><br>
            <label for="email">Email:</label>
            <input type="email" id="email" name="email" required><br>
            <label for="subscribe">Subscribe:</label>
```

```
        <input type="checkbox" id="subscribe" name="subscribe"><br>

        <label for="gender">Gender:</label>

        <input type="radio" id="male" name="gender" value="male"> Male

        <input type="radio" id="female" name="gender" value="female"> Female<br>

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

    </form>

  </div>

</body>

</html>
```

**Step 3**: Create the new JSP page and it named as **processPost.jsp** to handles the form submission and in this page can retrieve the form data from the request object using **getParameter()** method. This method can take the name of the form field as its the parameter and returns the value entered by user.
HTML

```
<%@ page language="java" contentType="text/html; charset=UTF-8"

  pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>

<head>
```

```
<meta charset="UTF-8">
<title>POST Form Processing</title>
<link rel="stylesheet" type="text/css" href="WEB-INF/css/style.css">
<style>
    /* Additional CSS for better visibility */
    body {
        font-family: Arial, sans-serif;
        background-color: #f4f4f4;
    }

    .container {
        width: 50%;
        margin: 50px auto;
        padding: 20px;
        border: 1px solid #ccc;
        border-radius: 5px;
        background-color: #fff;
        box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
```

```
        }

        h2, h3 {
            color: #333;
            text-align: center;
        }

        p {
            text-align: center;
        }
    </style>
</head>
<body>
    <div class="container">
        <h2>POST Form Processing</h2>
        <%-- Retrieving parameters from the request --%>
        <% String name = request.getParameter("name"); %>
```

```
        <% String email = request.getParameter("email"); %>

        <% String subscribe = request.getParameter("subscribe"); %>

        <% String gender = request.getParameter("gender"); %>



        <h3>Hello, <%= name %></h3>

        <p>Your email is: <%= email %></p>

        <p>Subscribed: <%= (subscribe != null) ? "Yes" : "No" %></p>

        <p>Gender: <%= gender %></p>

    </div>

</body>

</html>
```