**Introduction of JSP (JavaServer Pages)**

JavaServer Pages (JSP) is a technology used to create dynamic web applications, similar to Servlet technology. JSP can be considered an extension of Servlet technology, offering additional functionalities such as Expression Language (EL) and JavaServer Pages Standard Tag Library (JSTL). JSP pages are generally easier to maintain than Servlets because they allow for the separation of design and development concerns.

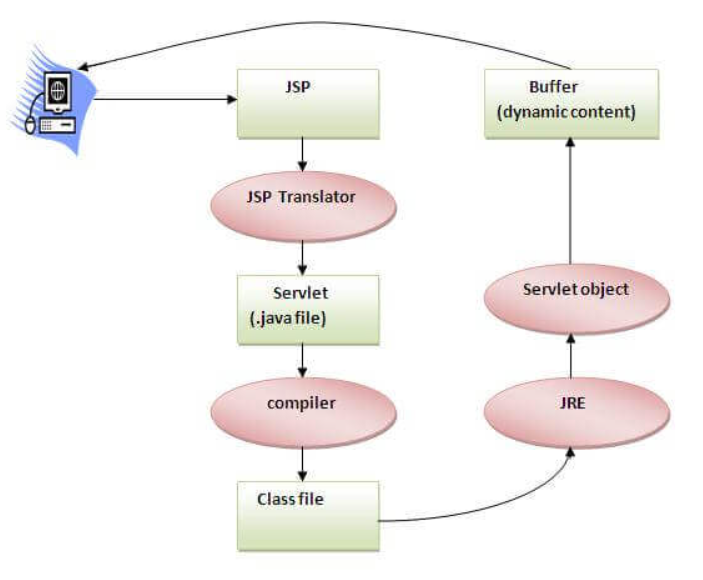
**Advantages of JSP over Servlets**

1. **Extension to Servlet**
   * **Integration**: JSP technology extends Servlet technology, inheriting all its features.
   * **Enhanced Features**: JSP allows the use of implicit objects, predefined tags, expression language (EL), and custom tags, making development more straightforward and feature-rich.
2. **Easy to Maintain**
   * **Separation of Concerns**: JSP allows separation of business logic and presentation logic, making it easier to manage and maintain compared to Servlets, which often mix both concerns.
3. **Fast Development**
   * **Dynamic Changes**: JSP pages can be modified without requiring recompilation or redeployment, unlike Servlets where changes in the look and feel necessitate recompiling the entire Servlet code.
4. **Less Code**
   * **Tag Usage**: JSP reduces code complexity through the use of tags like action tags, JSTL, custom tags, and EL, leading to cleaner and more maintainable code.

**JSP Lifecycle**

The lifecycle of a JSP page includes several phases:

1. **Translation of JSP Page**
   * The JSP page is converted into a Servlet by the JSP translator, which is part of the web server.
2. **Compilation of JSP Page**
   * The generated Servlet code is compiled into a bytecode .class file by the compiler.
3. **Classloading**
   * The compiled Servlet class is loaded into memory by the class loader.
4. **Instantiation**
   * An object of the compiled Servlet is created.
5. **Initialization**
   * The container calls the jspInit() method to initialize the JSP.
6. **Request Processing**
   * The container invokes the \_jspService() method to handle client requests and generate responses.
7. **Destroy**
   * The container calls the jspDestroy() method to clean up resources when the JSP is no longer needed.



**Conversion of JSP to Servlet**

JSP pages are converted into Servlets by the JSP translator. The process involves:

1. **Translation**: The JSP translator converts the JSP into a Servlet.
2. **Compilation**: The Servlet code is compiled into a .class file.
3. **Servlet Lifecycle**: The servlet undergoes the typical Servlet lifecycle phases, including initialization, request handling, and destruction.

**Creating a Simple JSP Page**

To create and run a simple JSP page:

1. **Write JSP Code**: Create a file named index.jsp with the following content:

<html>

<body>

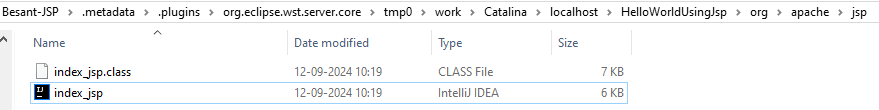
<% out.print(2\*5); %>

</body>

</html>

This code will print 10 on the browser.

1. **Save and Deploy**: Save the file with a .jsp extension. Place it in the webapps directory of your Apache Tomcat server.
2. **Start Server**: Ensure that your Apache Tomcat server is running.
3. **Access JSP Page**: Open a browser and navigate to http://localhost:portnumber/contextRoot/index.jsp.
4. For example, <http://localhost:8888/myapplication/index.jsp>.

Note: This Location you can check how JSP page converted into Servlet.

**Directory Structure for JSP**

The directory structure for JSP pages is generally similar to that of Servlets. Key points include:

* **Placement**: JSP files can be placed outside the WEB-INF folder or within any directory structure of the web application.
* **No Structure Needed**: For simple JSPs without additional components like Beans, Servlets, or TLD files, directory structure requirements are minimal.
* **When Structure is Needed**: Directory structure becomes important when using Beans, Servlets, or TLD files, which should be organized appropriately.

**Summary**

* **JSP vs Servlet**: JSP is an extension of Servlets, offering additional features and a more manageable approach to web application development.
* **Lifecycle**: JSPs go through a lifecycle of translation, compilation, classloading, instantiation, initialization, request processing, and destruction.
* **Simple JSP Creation**: Writing, saving, deploying, and accessing JSP pages is straightforward.
* **Directory Structure**: JSPs can be deployed with minimal structure, but a proper directory layout is essential for more complex applications.

### ****JSP Tag Explanation in details****

### ****1. Scriptlet Tag (****<% %>****)****

The **Scriptlet tag** in JSP is used to write Java code inside JSP pages. Anything you write inside a scriptlet tag is inserted directly into the service() method of the JSP’s servlet when the JSP page is converted into a servlet.

#### **Syntax**:

<%

// Java code here

%>

#### **Example**:

<%

int count = 10;

out.println("Count is: " + count);

%>

#### **Explanation**:

* The Java code is placed inside <% %>.
* The out implicit object is used to send output to the client.

### ****2. Expression Tag (****<%= %>****)****

The Expression tag is used to output the value of a Java expression to the client (browser). It is mainly used to display variable values or results of expressions.

#### **Syntax**:

<%= expression %>

#### **Example**:

<%

int length = 20;

int width = 10;

%>

<p>The area of the rectangle is: <%= length \* width %></p>

#### **Explanation**:

* The expression inside <%= %> is evaluated, and the result is inserted into the response stream.

### ****3. Declaration Tag (****<%! %>****)****

The Declaration tag allows you to declare variables and methods in a JSP page that are shared by all requests. The declarations are translated into instance variables or methods of the servlet class that the JSP is compiled into.

#### **Syntax**:

<%!

// Variable or method declaration

%>

#### **Example**:

<%!

int count = 0;

public int getCount() {

return ++count;

}

%>

<p>Current count is: <%= getCount() %></p>

#### **Explanation**:

* You can declare methods or variables using <%! %>.
* These declarations are shared across all requests to the JSP page.

### ****4. Directive Tags****

Directive tags provide instructions to the JSP container. They do not produce output directly but control how the JSP page is processed. There are three types of directives:

#### **1. Page Directive**

Used to define page-specific attributes, such as error handling, import statements, etc.

#### **Syntax**:

<%@ page attribute="value" %>

#### **Example**:

<%@ page import="java.util.Date" %>

#### **Common attributes**:

* import: Imports Java classes.
* contentType: Defines the content type of the response (e.g., text/html).
* errorPage: Defines an error page.
* isErrorPage: Indicates whether the page is an error page.

#### **2. Include Directive**

Used to include a file at translation time (when the JSP is compiled).

#### **Syntax**:

<%@ include file="relativeURL" %>

#### **Example**:

<%@ include file="header.jsp" %>

#### **3. Taglib Directive**

Used to define and use custom tags in a JSP page.

#### **Syntax**:

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

#### **Example**:

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

### ****5. Action Tags****

JSP Action tags are used to control the flow of a JSP page or interact with JavaBeans components and other server resources.

#### **1.** jsp:include

Includes another JSP or static resource at runtime (request time).

#### **Syntax**:

<jsp:include page="relativeURL" />

#### **Example**:

<jsp:include page="footer.jsp" />

#### **2.** jsp:forward

Forwards the request to another resource (JSP, Servlet, etc.).

#### **Syntax**:

<jsp:forward page="relativeURL" />

#### **Example**:

<jsp:forward page="nextPage.jsp" />

#### **3.** jsp:useBean

Instantiates or references a JavaBean.

#### **Syntax**:

<jsp:useBean id="beanInstance" class="package.BeanClass" />

#### **Example**:

<jsp:useBean id="user" class="com.example.User" />

<jsp:setProperty name="user" property="username" value="JohnDoe" />

<jsp:getProperty name="user" property="username" />

#### **Explanation**:

* jsp:useBean: Creates an instance of User bean.
* jsp:setProperty: Sets the property of the bean.
* jsp:getProperty: Retrieves the property of the bean.

#### **4.** jsp:setProperty

Used to set the value of a JavaBean property.

#### **Syntax**:

<jsp:setProperty name="beanName" property="propertyName" value="value" />

#### **Example**:

<jsp:setProperty name="user" property="username" value="John" />

#### **5.** jsp:getProperty

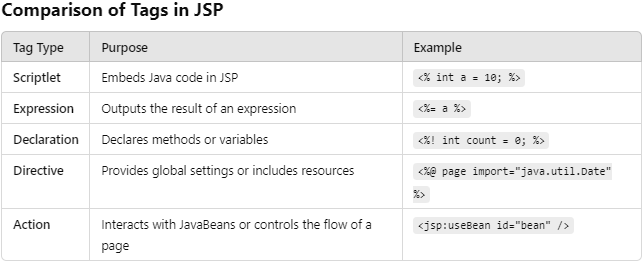
Used to get the value of a JavaBean property.

#### **Syntax**:

<jsp:getProperty name="beanName" property="propertyName" />

#### **Example**:

<jsp:getProperty name="user" property="username" />



**JSP Tag Practical notes in details**

### 1. ****Scriptlet Tag (****<% %>****)****

**Description**: The scriptlet tag allows you to write Java code inside JSP pages. The code within the tag is executed when the page is requested.

#### **Practical Example**:

* **index.jsp**:

<html>

<head>

<title>Scriptlet Example</title>

</head>

<body>

<h2>Scriptlet Tag Example</h2>

<%

// Declaring a Java variable

String name = "Amarjeet Kumar Singh";

int count = 5;

// Outputting the variables in HTML

out.println("<p>Hello, " + name + "</p>");

out.println("<p>Count: " + count + "</p>");

// Iterating using a loop

for(int i = 1; i <= count; i++) {

out.println("<p>Iteration " + i + "</p>");

}

%>

</body>

</html>

**Explanation**:

* Inside <% %>, you can write any Java code like variable declarations, loops, and conditional statements.
* The output is directly inserted into the HTML response using the out object, which is automatically available in JSP.

### 2. ****Expression Tag (****<%= %>****)****

**Description**: The **expression tag** is mainly used to **print the values of a variable or method**. It evaluates a Java expression and converts the result into a string, which is then inserted directly into the output (HTML page).

#### **Practical Example**:

* **index.jsp**:

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

<html>

<head>

<title>Expression Tag Example</title>

</head>

<body>

<h2>Expression Tag Example</h2>

<%

// Declare a Java variable

String message = "Welcome to JSP!";

int length = message.length();

// Declare a method to get current year

int getCurrentYear() {

return 2024;

}

%>

<!-- Printing the value of the variable using Expression Tag -->

<p>Message: <%= message %></p>

<!-- Printing the length of the string -->

<p>Length of the message: <%= length %></p>

<!-- Printing the result of the method using Expression Tag -->

<p>Current Year: <%= getCurrentYear() %></p>

</body>

</html>

**Explanation**:

* **Main Use**: The **expression tag** is mainly used to **print the values** of a variable (message) or method (getCurrentYear()).
* The **message** is declared and then output using <%= message %>.
* The **length** of the message is calculated and printed with <%= length %>.
* The **getCurrentYear()** method is defined, and its return value is displayed using <%= getCurrentYear() %>.

### Key Points:

1. **No out.println() Needed**: The expression tag automatically converts the result of the expression into a string and appends it to the response.
2. **Simple & Concise**: It simplifies the process of outputting dynamic content on a JSP page without writing explicit print statements.
3. **Evaluates Java Code**: The Java code inside the tag is evaluated and the result is placed in the HTML output.

### 3. ****Declaration Tag (****<%! %>****)****

**Description**: The declaration tag is used to declare variables and methods at the class level in JSP. These declarations are shared across multiple requests.

#### **Practical Example**:

* **index.jsp**:

<html>

<head>

<title>Declaration Tag Example</title>

</head>

<body>

<h2>Declaration Tag Example</h2>

<p>Current Count: <%= getCurrentCount() %></p>

<%

out.println("<p>Current Message: " + currentMessage + "</p>");

%>

</body>

</html>

<%!

// Class-level variable

String currentMessage = "Hello, JSP!";

// Method that returns the current count

public int getCurrentCount() {

return 10;

}

%>

**Explanation**:

* The variable currentMessage and the method getCurrentCount() are declared inside <%! %>, making them accessible throughout the JSP page.
* These declarations exist as class members of the generated servlet and can be reused across requests.

### 4. ****Directive Tags****

**Description**: Directives provide global information about the JSP page, like importing classes, including other files, and error handling.

#### **Page Directive**:

Used to define attributes for the page, like importing Java classes.

* **index.jsp**:

<%@ page import="java.util.Date" %>

<html>

<head>

<title>Page Directive Example</title>

</head>

<body>

<h2>Page Directive Example</h2>

<p>Current Date and Time: <%= new Date() %></p>

</body>

</html>

**Explanation**:

* The import attribute imports the java.util.Date class, allowing you to use it in the JSP page.
* This example prints the current date and time using the Date class.

#### **Include Directive**:

Used to include another file at compile time.

* **header.jsp**:

<h1>Header Section</h1>

<hr/>

* **index.jsp**:

<%@ include file="header.jsp" %>

<html>

<head>

<title>Include Directive Example</title>

</head>

<body>

<h2>Body Section</h2>

<p>This is the body of the page.</p>

</body>

</html>

**Explanation**:

* The header.jsp file is included at the top of the main index.jsp page.
* The included content becomes part of the JSP file at translation time (when the JSP is converted into a servlet).

### 5. ****Action Tags****

**Description**: Action tags are used to control the flow of the application and interact with JavaBeans and other resources.

#### **jsp**

#### :

Dynamically includes a resource (JSP or static file) at request time.

* **header.jsp**:

<h1>Header Section</h1>

<hr/>

* **index.jsp**:

<html>

<head>

<title>jsp:include Example</title>

</head>

<body>

<jsp:include page="header.jsp" />

<h2>Body Section</h2>

<p>This is the body of the page.</p>

</body>

</html>

**Explanation**:

* The jsp:include tag includes the header.jsp file when the page is requested.
* The included file can be a JSP, HTML, or other resources.

#### **jsp**

#### :

Forwards the request to another resource (JSP or servlet).

* **index.jsp**:

<jsp:forward page="welcome.jsp" />

* **welcome.jsp**:

<html>

<head>

<title>Forwarded Page</title>

</head>

<body>

<h2>Welcome Page</h2>

<p>You have been forwarded to this page.</p>

</body>

</html>

**Explanation**:

* When the user accesses index.jsp, they are immediately forwarded to welcome.jsp.
* The jsp:forward tag transfers the control to another JSP or servlet.

#### **jsp**

#### , **jsp**

#### , and **jsp**

#### :

These tags are used to interact with JavaBeans on the JSP page.

* **User.java (JavaBean)**:

package com.example;

public class User {

private String username;

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

}

* **index.jsp**:

<jsp:useBean id="user" class="com.example.User" />

<jsp:setProperty name="user" property="username" value="John Doe" />

<html>

<head>

<title>jsp:useBean Example</title>

</head>

<body>

<h2>Welcome, <jsp:getProperty name="user" property="username" />!</h2>

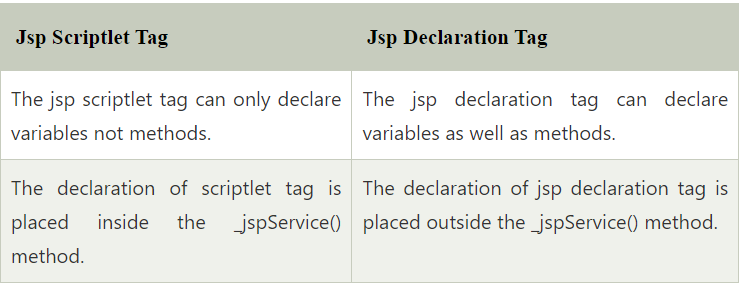
</body>

</html>

**Explanation**:

* The jsp:useBean tag creates an instance of the User bean.
* The jsp:setProperty tag sets the username property of the bean to "John Doe."
* The jsp:getProperty tag retrieves and displays the username property.

**Difference between Scriptlet tag vs Declaration tag**



**JSP Implicit Objects:**

**Implicit objects** in JSP are pre-defined objects that the JSP container creates and makes available to developers for use within the JSP pages without needing to instantiate them. These objects provide access to various functionalities such as handling requests, responses, sessions, and more.

There are **9 implicit objects** in JSP:

1. **request** - (HttpServletRequest)
2. **response** - (HttpServletResponse)
3. **session** - (HttpSession)
4. **application** - (ServletContext)
5. **out** - (JspWriter)
6. **config** - (ServletConfig)
7. **pageContext** - (PageContext)
8. **page** - (Object)
9. **exception** - (Throwable)

### 1. request ****Object**** (HttpServletRequest)

**index.html**:

<!DOCTYPE html>

<html>

<head>

<title>Request Object Example</title>

</head>

<body>

<form action="requestExample.jsp" method="POST">

<label for="uname">Enter your name:</label>

<input type="text" name="uname" id="uname">

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

</form>

</body>

</html>

**requestExample.jsp**:

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

<html>

<head>

<title>Request Object Example</title>

</head>

<body>

<%

// Getting data from the request

String userName = request.getParameter("uname");

out.print("Hello, " + userName + "!<br/>");

// Fetching additional request information

out.print("Request Method: " + request.getMethod() + "<br/>");

out.print("Client IP Address: " + request.getRemoteAddr() + "<br/>");

%>

</body>

</html>

### 2. response ****Object**** (HttpServletResponse)

**responseExample.jsp**:

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

<html>

<head>

<title>Response Object Example</title>

</head>

<body>

<%

// Redirecting the response to another page

response.sendRedirect("https://www.example.com");

%>

</body>

</html>

### 3. session ****Object**** (HttpSession)

**sessionExample.jsp**:

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

<html>

<head>

<title>Session Object Example</title>

</head>

<body>

<%

// Storing a value in the session

session.setAttribute("userName", "Amarjeet Kumar Singh");

// Retrieving the session attribute

String user = (String) session.getAttribute("userName");

out.print("Session User: " + user + "<br/>");

// Invalidating the session (optional)

// session.invalidate();

%>

</body>

</html>

### 4. application ****Object**** (ServletContext)

**applicationExample.jsp**:

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

<html>

<head>

<title>Application Object Example</title>

</head>

<body>

<%

// Storing a value in the application scope

application.setAttribute("appName", "My Web Application");

// Retrieving the application attribute

String appName = (String) application.getAttribute("appName");

out.print("Application Name: " + appName);

%>

</body>

</html>

### 5. out ****Object**** (JspWriter)

**outExample.jsp**:

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

<html>

<head>

<title>Out Object Example</title>

</head>

<body>

<%

// Using out to print dynamic content

out.print("Current time: " + new java.util.Date() + "<br/>");

// You can use out.println() for line breaks

out.println("This is printed using out.println() method.<br/>");

%>

</body>

</html>

### 6. config ****Object**** (ServletConfig)

**configExample.jsp**:

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

<html>

<head>

<title>Config Object Example</title>

</head>

<body>

<%

// Getting servlet config information

String servletName = config.getServletName();

out.print("Servlet Name: " + servletName + "<br/>");

// You can also fetch initialization parameters using config.getInitParameter("paramName")

%>

</body>

</html>

### 7. pageContext ****Object**** (PageContext)

**pageContextExample.jsp**:

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

<html>

<head>

<title>PageContext Object Example</title>

</head>

<body>

<%

// Setting an attribute in the page context

pageContext.setAttribute("pageVar", "This is a page context attribute");

// Retrieving the page context attribute

String pageVar = (String) pageContext.getAttribute("pageVar");

out.print(pageVar + "<br/>");

// Forwarding the request to another page (optional)

// pageContext.forward("anotherPage.jsp");

%>

</body>

</html>

### 8. page ****Object**** (Object)

**pageExample.jsp**:

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

<html>

<head>

<title>Page Object Example</title>

</head>

<body>

<%

// Printing the page object (equivalent to 'this')

out.print("This is the current JSP page instance: " + page.toString());

%>

</body>

</html>

### 9. exception ****Object**** (Throwable)

**errorPage.jsp**:

<%@ page isErrorPage="true" %>

<html>

<head>

<title>Error Page</title>

</head>

<body>

<h2>Error Details</h2>

<%

// Displaying exception details

if (exception != null) {

out.print("Error: " + exception.getMessage() + "<br/>");

}

%>

</body>

</html>

To trigger the error page, set the errorPage attribute in the main JSP:<%@ page errorPage="errorPage.jsp" %>

