JSP (Java Server Pages)

- **JSP** technology is used to create web application just like Servlet technology.
- It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.
- A JSP page consists of HTML tags and JSP tags.
- The JSP pages are easier to maintain than Servlet because we can separate designing and development.
- It provides some additional features such as Expression Language, Custom Tags, etc.

Advantages of JSP over Servlet

There are many advantages of JSP over the Servlet. They are as follows:

1) Extension to Servlet

JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

2) Easy to maintain

JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.

3) Fast Development: No need to recompile and redeploy

If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

4) Less code than Servlet

In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.

The Lifecycle of a JSP Page

The JSP pages follow these phases:

- Translation of JSP Page
- Compilation of JSP Page
- Classloading (the classloader loads class file)
- Instantiation (Object of the Generated Servlet is created).
- Initialization (the container invokes jspInit() method).
- Request processing (the container invokes _jspService() method).
- Destroy (the container invokes jspDestroy() method).

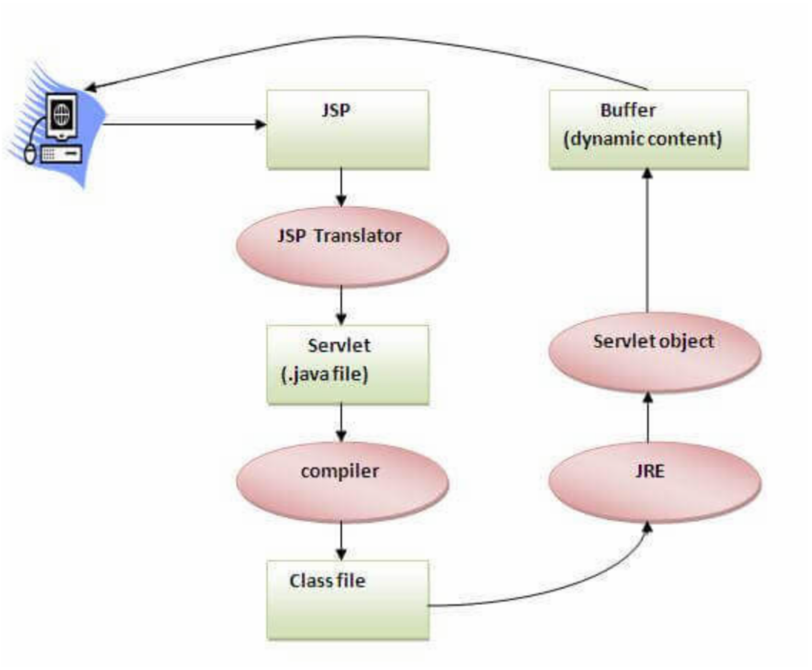
1. **Translation Phase:**
    - When a user navigates to a page ending with a .jsp extension in their web browser, the web browser sends an HTTP request to the webserver.
    - The webserver checks if a compiled version of the JSP page already exists.
    - If the JSP page's compiled version does not exist, the file is sent to the JSP Servlet engine, which converts it into servlet content (with .java extension). This process is known as translation.
    - The web container automatically translates the JSP page into a servlet. So as a developer, you don't have to worry about converting it manually.
2. **Compilation Phase:**
    - In case the JSP page was not compiled previously or at any point in time, then the JSP page is compiled by the JSP engine.
    - In this compilation phase, the Translated .java file of the servlet is compiled into the Java servlet .class file.
3. **Initialization Phase:**

    In this Initialization phase, the container will

    1. Load the equivalent servlet class.
    2. Create instance.
    3. Call the jspInit() method for initializing the servlet instance.

This initialization is done only once with the servlet's init method where database connection, opening files, and creating lookup tables are done.

4. **Execution Phase:**
    - This Execution phase is responsible for all JSP interactions and logic executions for all requests until the JSP gets destroyed.
    - As the requested JSP page is loaded and initiated, the JSP engine has to invoke the _jspService() method.
    - This method is invoked as per the requests, responsible for generating responses for the associated requests, and responsible for all HTTP methods.
5. **Destruction Phase(Clean up Phase)**
    - This destruction phase is invoked when the JSP has to be cleaned up from use by the container. The jspDestroy() method is invoked.
    - You can incorporate and write cleanup codes inside this method for releasing database connections or closing any file.



Summary: JSP page is translated into Servlet by the help of JSP translator. The JSP translator is a part of the web server which is responsible for translating the JSP page into Servlet. After that, Servlet page is compiled by the compiler and gets converted into the class file. All the processes that happen in Servlet are performed on JSP later like initialization, committing response to the browser and destroy.

Creating a simple JSP Page

To create the first JSP page, write some HTML code as given below, and save it by .jsp extension. We have saved this file as index.jsp. Put it in a folder and paste the folder in the web-apps directory in apache tomcat to run the JSP page.

index.jsp

Let's see the simple example of JSP where we are using the scriptlet tag to put Java code in the JSP page.

```
<html>
<body>
<% out.print(2*5); %>
</body>
</html>
```

It will print **10** on the browser.

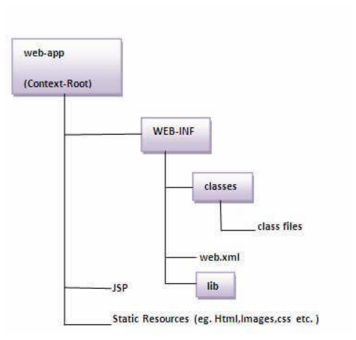
How to run a simple JSP Page?

Follow the following steps to execute this JSP page:

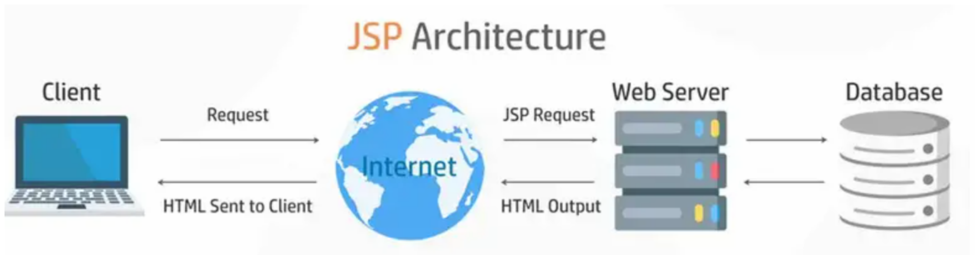
- Start the server
- Put the JSP file in a folder and deploy on the server
- Visit the browser by the URL http://localhost:portno/contextRoot/jspfile, for example, http://localhost:8888/myapplication/index.jsp

JSP Architecture

The directory structure of JSP page is same as Servlet. We contain the JSP page outside the WEB-INF folder or in any directory.



- JSP architecture is a 3-tier architecture that separates a web application's presentation, logic, and data layers.
- The presentation layer, or client side, is responsible for displaying the user interface and handling user interaction.
- The logic layer, or server-side, is responsible for processing user requests and handling business logic.
- The data layer is responsible for storing and retrieving data from a database or other storage system.
- This separation of concerns allows for better maintainability and scalability of the application.



What is a web Container?

- A JSP-based web application requires a JSP engine, also known as a web container, to process and execute the JSP pages.
- The web container is a web server component that manages the execution of web programs such as servlets, JSPs, and ASPs.
- When a client sends a request for a JSP page, the web container intercepts it and directs it to the JSP engine.
- The JSP engine then converts the JSP page into a servlet class, compiles it, and creates an instance of the class.
- The service method of the servlet class is called, which generates the dynamic content for the JSP page.

The web container also manages the lifecycle of the JSP pages and servlets, handling tasks such as instantiating, initializing and destroying them. Additionally, it provides security, connection pooling, and session management services to the JSP-based web application.

Component of JSP Architecture:

JSP architecture is a web application development model that defines the structure and organization of a JSP-based web application. It consists of the following components:

- **JSP pages:** These are the main building blocks of a JSP application. They contain a combination of HTML, XML, and JSP elements (such as scriptlets, expressions, and directives) that generate dynamic content.
- **Servlets:** JSP pages are converted into servlets by the JSP engine. Servlets are Java classes that handle HTTP requests and generate dynamic content.
- **JSP engine (web container):** This web server component is responsible for processing JSP pages. It converts JSP pages into servlets, compiles them, and runs them in the Java Virtual Machine (JVM).
- **JavaBeans:** These are reusable Java classes that encapsulate business logic and data. They are used to store and retrieve information from a database or other data sources.

- **JSTL (JavaServer Pages Standard Tag Library):** This is a set of predefined tags that can be used in JSP pages to perform common tasks such as iterating over collections, conditional statements, and internationalization.
- **Custom Tag Libraries:** JSP allows the creation of custom tags that can be used on JSP pages. These reusable Java classes encapsulate complex logic and can generate dynamic content cleanly and consistently.

JSP Architecture Flow

JSP architecture flow refers to the sequence of steps a JSP-based web application goes through to process and execute JSP pages. The general flow of a JSP architecture can be described as follows:

1. A client (such as a web browser) sends a request for a JSP page to a web server.
2. The web server forwards the request to the JSP engine responsible for processing JSP pages.
3. The JSP engine checks if the requested JSP page has been compiled into a servlet. If not, it compiles the JSP page into a servlet class. This is done by parsing the JSP page and converting its elements (such as scriptlets, expressions, and directives) into Java code.
4. The JSP engine then compiles the servlet class, which creates a Java class file that can be executed by the Java Virtual Machine (JVM).
5. The JSP engine then creates an instance of the servlet class and calls the service() method, which generates the dynamic content for the JSP page. Within the service() method, the JSP engine generates the HTML code for the response by combining the static template in the JSP page with the dynamic content generated by the Java code.
6. The JSP engine sends the generated HTML code back to the web server, which then sends it back to the client as a response.
7. The JSP engine also maintains a cache of the compiled servlet classes so subsequent requests for the same JSP page can be handled more efficiently.

Comparison between JSP & Servlet

- **Servlet** technology is used to create a web application.
- A **servlet** is a Java class that is used to extend the capabilities of servers that host applications accessed by means of a request-response model.
- Servlets are mainly used to extend the applications hosted by web services.
- **JSP** is used to create web applications just like **Servlet** technology.
- A **JSP** is a text document that contains two types of text: static data and dynamic data.
- The static data can be expressed in any text-based format (like HTML, XML, SVG, and WML), and the dynamic content can be expressed by JSP elements. Difference between Servlet and JSP.

The difference between Servlet and JSP is as follows:

Servlet	**JSP**
Servlet is a java code.	JSP is a HTML-based compilation code.
Writing code for servlet is harder than JSP as it is HTML in java.	JSP is easy to code as it is java in HTML.
Servlet plays a controller role in the ,MVC approach.	JSP is the view in the MVC approach for showing output.
Servlet is faster than JSP.	JSP is slower than Servlet because the first step in the JSP lifecycle is the translation of JSP to java code and then compile.
Servlet can accept all protocol requests.	JSP only accepts HTTP requests.
In Servlet, we can override the service() method.	In JSP, we cannot override its service() method.
In Servlet by default session management is not enabled, user have to enable it explicitly.	In JSP session management is automatically enabled.
In Servlet we have to implement everything	In JSP business logic is separated from

like business logic and presentation logic in just one servlet file.	presentation logic by using JavaBeansclient-side.
Modification in Servlet is a time-consuming compiling task because it includes reloading, recompiling, JavaBeans and restarting the server.	JSP modification is fast, just need to click the refresh button.
It does not have inbuilt implicit objects.	In JSP there are inbuilt implicit objects.
There is no method for running JavaScript on the client side in Servlet.	While running the JavaScript at the client side in JSP, client-side validation is used.
Packages are to be imported on the top of the program.	Packages can be imported into the JSP program (i.e, bottom , middleclient-side, or top)
It can handle extensive data processing.	It cannot handle extensive data processing very efficiently.
The facility of writing custom tags is not present.	The facility of writing custom tags is present.
Servlets are hosted and executed on Web Servers.	Before the execution, JSP is compiled in Java Servlets and then it has a similar lifecycle as Servlets.

Basic Structure of JSP Code file

- A JSP page is a normal web page with JSP elements for generating the parts of the web page that differ for each request.
- A simple JSP web page that contains the JSP elements and template text.
- Everything on the page that is not a JSP element is called Template text. Template text can be any text, i.e., HTML, XML, WML or even plain text.JSP has no dependency on HTML, it can be used with any markup language.

- Template text is always passed straight to the browser.

JSP Elements

There are three types of JSP elements present:

1. Directive
2. Action
3. Scripting

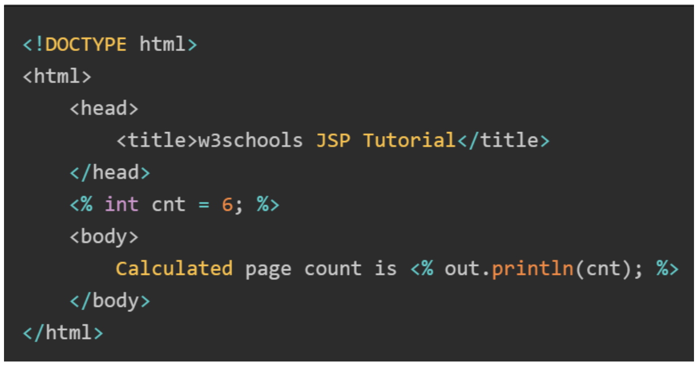
Java Scriplet

- Java provides various scripting elements that allow you to insert Java code from your JSP code into the servlet.
- Scripting elements have different components that are allowed by JSP.

Scripting Elements in JSP

Scripting elements in JSP must be written within the <% %> tags. The JSP engine will process any code you write within the pair of the <% and %> tags, and any other texts within the JSP page will be treated as HTML code or plain text while translating the JSP page.

Example



```
<!DOCTYPE html>
<html>
    <head>
        <title>w3schools JSP Tutorial</title>
    </head>
    <% int cnt = 6; %>
    <body>
        Calculated page count is <% out.println(cnt); %>
    </body>
</html>
```

Here are five different scripting elements:

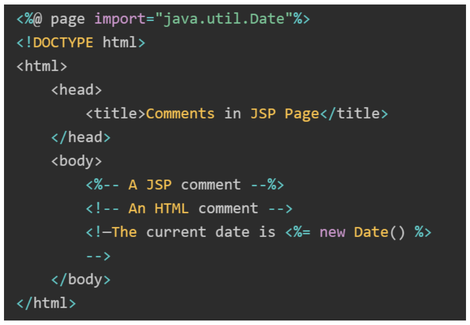
- Comment <%-- Set of comment statements --%>
- Directive <%@ directive %>
- Declaration <%! declarations %>

- Scriptlet <% scriplets %>
- Expression <%= expression %>

Comment:

Comments are marked as text or statements that are ignored by the JSP container. They are useful when you want to write some useful information or logic to remember in the future.

Example



```
<%@ page import="java.util.Date"%>
<!DOCTYPE html>
<html>
    <head>
        <title>Comments in JSP Page</title>
    </head>
    <body>
        <%-- A JSP comment --%>
        <!-- An HTML comment -->
        <!—The current date is <%= new Date() %>
        -->
    </body>
</html>
```

Directive:

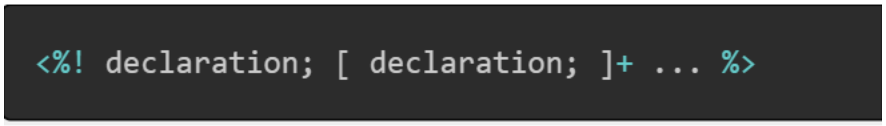
These tags are used to provide specific instructions to the web container when the page is translated. It has three subcategories:

- Page: <%@ page ... %>
- Include: <%@ include ... %>
- Taglib: <%@ taglib ... %>

Declaration

It is used to declare methods and variables you will use in your Java code within a JSP file. According to the rules of JSP, any variable must be declared before it can be used.

Syntax



```
<%! declaration; [ declaration; ]+ ... %>
```

Example

```
<!DOCTYPE html>
<html>
    <body>
        <%! int variable_value=62; %>
        <%= " Your integer data is :"+ variable_value %>
    </body>
</html>
```

Java Scriptlet Tag

The scriptlet tag allows writing Java code statements within the JSP page. This tag is responsible for implementing the functionality of _jspService() by scripting the java code.

Syntax

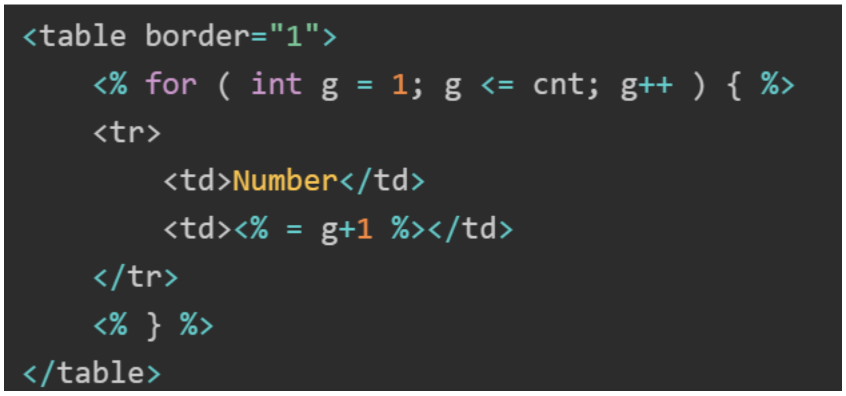
```
<% JAVA CODE %>
```

Example

```
<!DOCTYPE html>
<html>
    <head>
        <title>Scriptlet Tag</title>
    </head>
    <% int count = 2; %>
    <body>
        Calculated page count <% out.println(cnt); %>
    </body>
</html>
```

Another code snippet that shows how to write and mix scriptlet tags with HTML:

Example

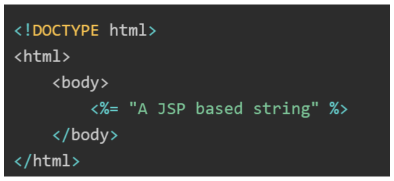


```
<table border="1">
    <% for ( int g = 1; g <= cnt; g++ ) { %>
    <tr>
        <td>Number</td>
        <td><% = g+1 %></td>
    </tr>
    <% } %>
</table>
```

Expression

Expressions elements are responsible for containing scripting language expression, which gets evaluated and converted to Strings by the JSP engine and is meant to the output stream of the response. Hence, you are not required to write out.print() for writing your data. This is mostly used for printing the values of variables or methods in the form of output.

Example



```
<!DOCTYPE html>
<html>
    <body>
        <%= "A JSP based string" %>
    </body>
</html>
```

JSP Directives

- Directives supply directions and messages to a JSP container.
- The directives provide global information about the entire page of JSP.
- Hence, they are an essential part of the JSP code. These special instructions are used for translating JSP to servlet code.
- In JSP's life cycle phase, the code must be converted to a servlet that deals with the translation phase.
- They provide commands or directions to the container on how to deal with and manage certain JSP processing portions.
- Directives can contain several attributes that are separated by a comma and act as key-value pairs.
- In JSP, directives are described with a pair of <%@ .... %> tags.

The syntax of Directives looks like:

```
<%@ directive attribute="" %>
```

There are 3 types of directives:

1. Page directive
2. Include directive
3. Taglib directive

1. Page Directive

- The page directive is used for defining attributes that can be applied to a complete JSP page.
- You may place your code for Page Directives anywhere within your JSP page.
- Page directives are implied at the top of your JSP page.

The basic syntax of the page directive is:

```
<%@ page attribute = "attribute_value" %>
```

The XML equivalent for the above derivation is:

```
<jsp:directive.page attribute = "attribute_value" />
```

The attributes used by the Page directives are:

1. **buffer:** Buffer attribute sets the buffer size in KB to control the JSP page's output.
2. **contentType:** The ContentType attribute defines the document's MIME (Multipurpose Internet Mail Extension) in the HTTP response header.
3. **autoFlush:** The autofill attribute controls the behavior of the servlet output buffer. It monitors the buffer output and specifies whether the filled buffer output should be flushed automatically or an exception should be raised to indicate buffer overflow.
4. **errorPage:** Defining the "ErrorPage" attribute is the correct way to handle JSP errors. If an exception occurs on the current page, it will be redirected to the error page.

5. **extends:** extends attribute used for specifying a superclass that tells whether the generated servlet has to extend or not.
6. **import:** The import attribute is used to specify a list of packages or classes used in JSP code, just as Java's import statement does in a Java code.
7. **isErrorPage:** This "isErrorPage" attribute of the Page directive is used to specify that the current page can be displayed as an error page.

2. Include Directive

- The JSP "include directive" is used to include one file in another JSP file.
- This includes HTML, JSP, text, and other files. This directive is also used to create templates according to the developer's requirement and breaks the pages in the header, footer, and sidebar.

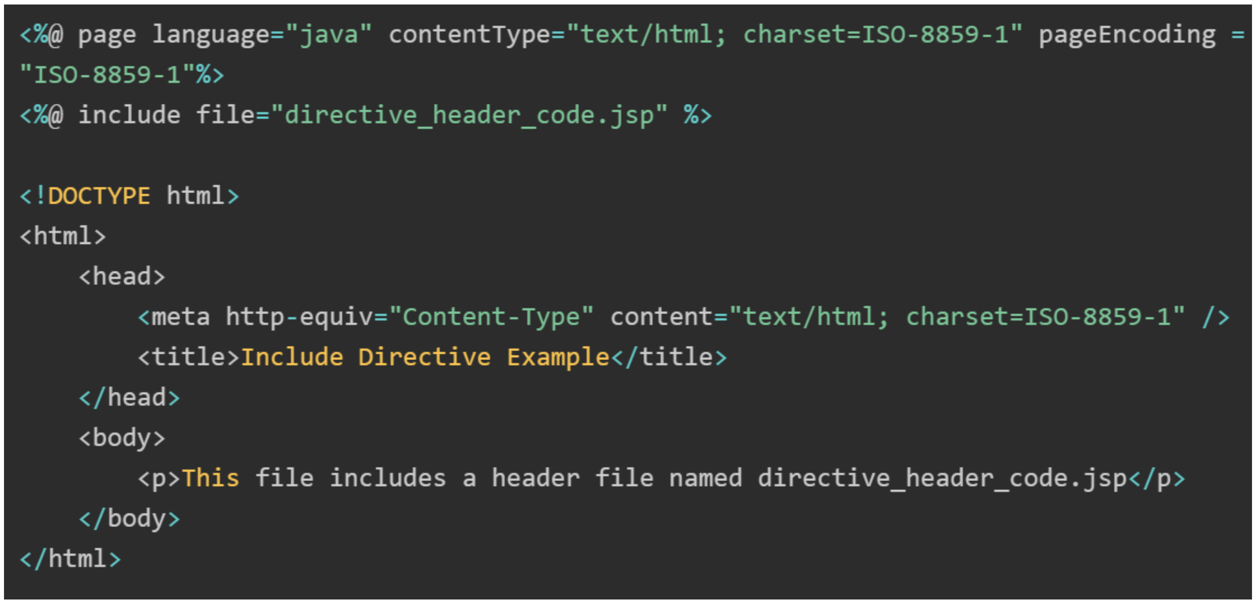
To use this Include Directive, you have to write it like:

```
<%@ include file = "relative url" >
```

The XML equivalent of the above way of representation is:

```
<jsp:directive.include file = "relative url" />
```

Example



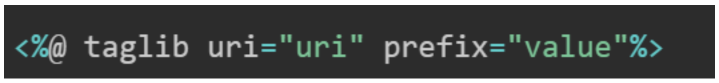
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding =
"ISO-8859-1"%>
<%@ include file="directive_header_code.jsp" %>

<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
        <title>Include Directive Example</title>
    </head>
    <body>
        <p>This file includes a header file named directive_header_code.jsp</p>
    </body>
</html>
```

3. Taglib Directive

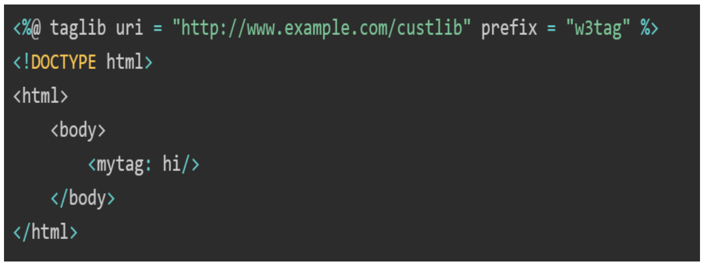
- The JSP taglib directive is implemented to define a tag library with "taglib" as its prefix.
- Custom tag sections of JSP use taglib. JSP's taglibdirective is used as standard tag libraries.

To implement taglib, you have to write it like this:



```
<%@ taglib uri="uri" prefix="value"%>
```

Example



```
<%@ taglib uri = "http://www.example.com/custlib" prefix = "w3tag" %>
<!DOCTYPE html>
<html>
    <body>
        <mytag: hi/>
    </body>
</html>
```

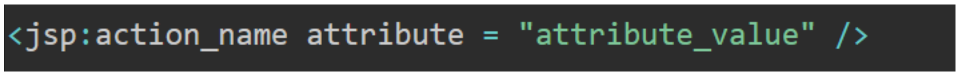
JSP Action

- It is necessary to control the servlet engine's behavior, which can be controlled dynamically by inserting the file by reusing the JavaBeans components or redirecting or forwarding the user to another page., i.e., by forwarding the request to another resource, which will possibly be a JSP, HTML, or other resources.

Actions Tags

- The JSP specification provides a standard tag called Action tag used within JSP code and is used to remove or eliminate Scriptlet code from your JSP code as JSP Scriptlets are obsolete and are not considered nowadays.
- There are many JSP action tags or elements, and each of them has its own uses and characteristics. Each JSP action tag is implemented to perform some precise tasks.

The action tag is also implemented to streamline flow between pages and to employ a Java Bean. As it coincides with the XML standard, the syntax for the action element is:
Syntax:



```
<jsp:action_name attribute = "attribute_value" />
```

Here is the list of JSP Actions:

1. **jsp:forward:** is used for forwarding the request and response to other resources.
2. **jsp:include:** is used for including another resource.
3. **jsp:body:** is used for defining dynamically-defined body of XML element.
4. **jsp:useBean:** is used for creating or locating bean objects.
5. **jsp:setProperty:** is used for setting the value of property in the bean object.
6. **jsp:getProperty:** is used for printing the value of the property of the bean.
7. **jsp:element:** is used for defining XML elements dynamically.
8. **jsp:plugin:** is used for embedding other components (applets).

Basic Attributes of Action Tags
Two basic attributes are commonly used for all action tags. These are:

1. **id:** The id attribute defines unique action elements and allows actions to be referenced within the JSP page. When the action creates an object's instance, the id attribute is used to refer to it.
2. **Scope:** The scope attribute is used for identifying an action's life cycle. It correlates with the id attribute because the scope attribute is used to establish that particular object's lifespan associated with the ID.

Syntax

This "include action" allows you to include another resource in the page being generated.

```
<jsp:include page = "Page URL"  flush = "Boolean Value" />
```

Example

```
<!DOCTYPE html>
<html>
    <head>
        <title>JSP include Action</title>
    </head>
    <body>
        <h2>JSP page: with include</h2>
        <jsp:include page="header.jsp" flush="false" />
    </body>
</html>
```

1. Jsp: forward Action

This "forward action" terminates the current page and allows you to forward the request to other resources.

Syntax:

```
<jsp:forward page = "URL of another static, JSP, OR Servlet page" />
```

Example

```
<!DOCTYPE html>
<html>
    <head>
        <title>Example for Forward Action</title>
    </head>
    <body>
        <h2>JSP page: Demo forward</h2>
        <jsp:forward page="forward_eg.jsp" />
    </body>
</html>
```

It is to be noted that this JSP file and forward_eg.jsp file should have to reside in the same directory to make it work.

2. Jsp: param Action

This "param action" is used for setting the parameter for (forward or include) value.

Syntax

```
<jsp: param name = "param_name" value = "value_of_parameter" />
```

Example

```
<!DOCTYPE html>
<html>
    <head>
        <title>Example of param Action</title>
    </head>
    <body>
        <h2>JSP page: Param with forward</h2>
        <jsp:forward page="forward_eg.jsp">
            <jsp:param name="date" value="06-09-2019" />
            <jsp:param name="time" value="10:30PM" />
            <jsp:param name="data" value="GKR" />
        </jsp:forward>
    </body>
</html>
```

Expression Tag

- Expression tags are one of the most useful scripting elements of JSP.
- Expression tags use special tags to print the different java expressions and output directly on the client-side.
- You can also use this for displaying information on your client browser.
- The expression tag is used to evaluate Java's expression within the JSP, which then converts the result into a string and forwards the result back to the client browser via the response object.
- Essentially, it is implemented for printing the result to the client (browser).
- An expression tag can hold any java language expression that can be employed as an argument to the out.print() method.

Syntax

```
<%= expression %>
```

Example

This is how the JSP container sees this when it encounters the above expression:

```
<%= (6*4) %>
```

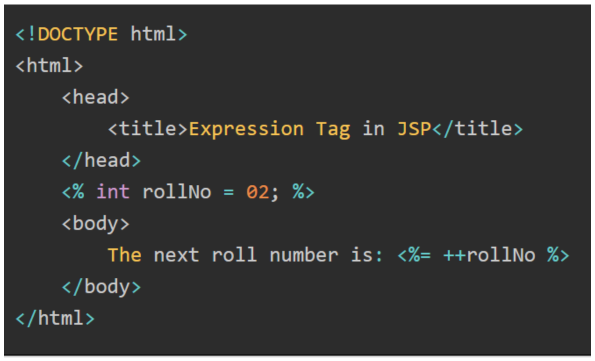
It turns out to be:

```
out.print((6*4));
```

It is to be noted that you will never terminate an expression using semicolon within Expression Tag. Like this:

```
<%= (6*4); %>
```

Program:

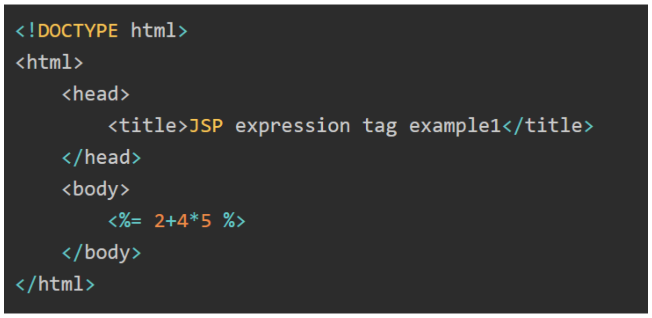


```
<!DOCTYPE html>
<html>
    <head>
        <title>Expression Tag in JSP</title>
    </head>
    <% int rollNo = 02; %>
    <body>
        The next roll number is: <%= ++rollNo %>
    </body>
</html>
```

Expression of Value

In this case, you have to pass the expression of values within the expression tag.

Example

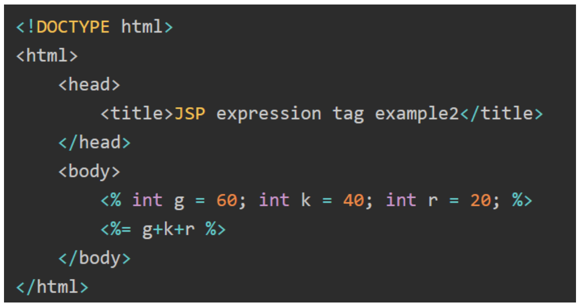


```
<!DOCTYPE html>
<html>
    <head>
        <title>JSP expression tag example1</title>
    </head>
    <body>
        <%= 2+4*5 %>
    </body>
</html>
```

Expression of Variables

In this case, you have to initialize a few variables, which can then be passed as the expression of variables within the expression tag in order to evaluate the result.

Example



```
<!DOCTYPE html>
<html>
    <head>
        <title>JSP expression tag example2</title>
    </head>
    <body>
        <% int g = 60; int k = 40; int r = 20; %>
        <%= g+k+r %>
    </body>
</html>
```

Difference Between Scriptlet Tag and Expression Tag

1. In Scriptlet tag, it evaluates your Java expression but does not print or show your result in conjunction with the HTML created. The declared variables have a local scope only and hence can't take access from another place in the .jsp. In contrast, the Expression Tag has the capability to evaluate the Java expression. It is used for inserting the result in the form of a string in conjunction with the HTML in the JSP.
2. You don't have to write the out.println in the expression tag to print the expression based output because these are changed into out.print() statement (as shown above the process of transformation of expression), which gets inserted by the container in the _jspService(-, -) of the servlet class.

JSP Implicit Object

- JSP implicit objects are essential components used in this regard. In this chapter, you will learn how to deal with implicit objects and implement them.
- The JSP engine produces these objects during the translation phase (i.e., at the time of translation from JSP to Servlet).
- They are being formed within the service method so that JSP developers can use them directly in Scriptlet without declaration and initialization.

There are a total of nine implicit objects supported by the JSP. These are:

1. **out:** javax.servlet.jsp.JspWriter
2. **request:** javax.servlet.http.HttpServletRequest
3. **response:** javax.servlet.http.HttpServletResponse

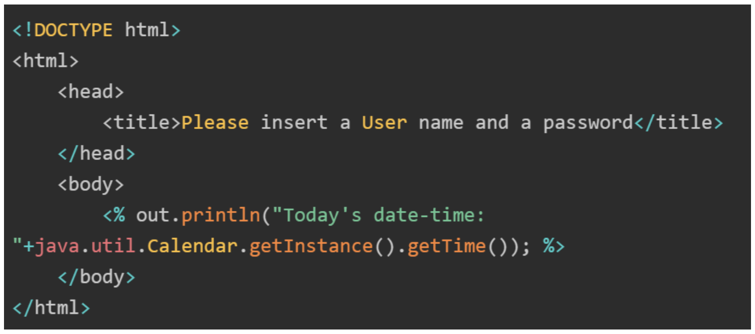
4. **session:** javax.servlet.http.HttpSession
5. **application:** javax.servlet.ServletContext
6. **exception:** javax.servlet.jsp.JspException
7. **page:** java.lang.Object
8. **pageContext:** javax.servlet.jsp.PageContext
9. **config:** javax.servlet.ServletConfig

The brief information about the implicit objects are given below:

1. The out Implicit Object

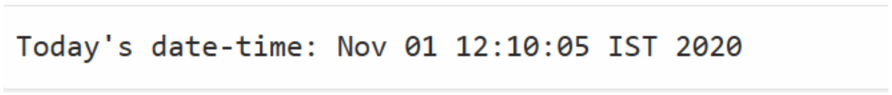
- An out object is an implicit object for writing data to the buffer and sending output as a response to the client's browser.
- The out implicit object is an instance of a javax.servlet.jsp.jspWriter class.

Example (HTML file):



```
<!DOCTYPE html>
<html>
    <head>
        <title>Please insert a User name and a password</title>
    </head>
    <body>
        <% out.println("Today's date-time:
"+java.util.Calendar.getInstance().getTime()); %>
    </body>
</html>
```

Output



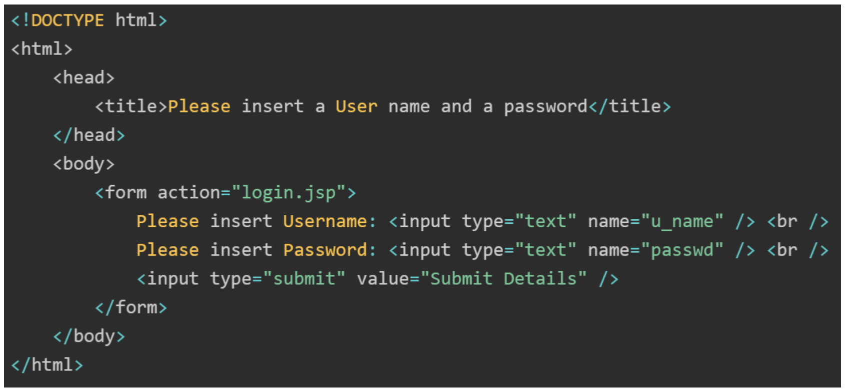
```
Today's date-time: Nov 01 12:10:05 IST 2020
```

2. The request implicit object

- A request object is an implicit object that is used to request an implicit object, which is to receive data on a JSP page, which has been submitted by the user on the previous JSP/HTML page.
- The request implicit object used in Java is an instance of a javax.servlet.http.HttpServletRequest interface where a client requests a page every time the JSP engine has to create a new object for characterizing that request.
- The container creates it for every request.
- It is used to request information such as parameters, header information, server names, cookies, and HTTP methods.
- It uses the getParameter() method to access the request parameter.

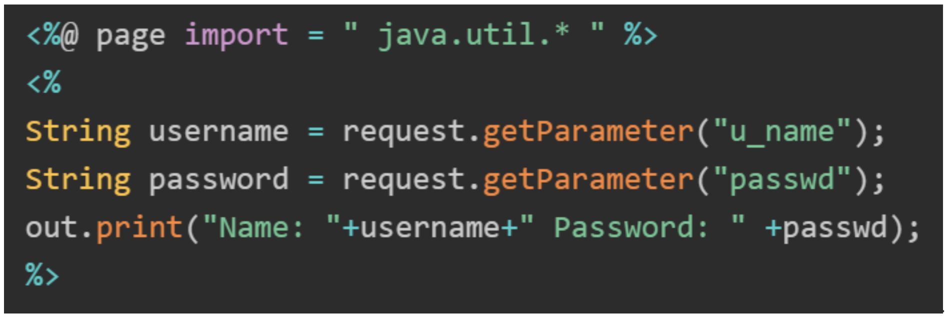
Example: An example of a JSP request implicit object where a user submits login information, and another JSP page receives it for processing:

(HTML file):



```
<!DOCTYPE html>
<html>
    <head>
        <title>Please insert a User name and a password</title>
    </head>
    <body>
        <form action="login.jsp">
            Please insert Username: <input type="text" name="u_name" /> <br />
            Please insert Password: <input type="text" name="passwd" /> <br />
            <input type="submit" value="Submit Details" />
        </form>
    </body>
</html>
```

(login.jsp)



```
<%@ page import = " java.util.* " %>
<%
String username = request.getParameter("u_name");
String password = request.getParameter("passwd");
out.print("Name: "+username+" Password: " +passwd);
%>
```

3. Response Implicit Object

- A response object is an implicit object implemented to modify or deal with the reply sent to the client (i.e., browser) after processing the request, such as redirect responding to another resource or an error sent to a client.
- The response implicit object is an instance of a javax.servlet.http.HttpServletResponse interface.
- The container creates it for every request.

4. The session Implicit Object

- A session object is the most commonly used implicit object implemented to store user data to make it available on other JSP pages until the user's session is active.
- The session implicit object is an instance of a javax.servlet.http.HttpSession interface.
- This session object has different session methods to manage data within the session scope.

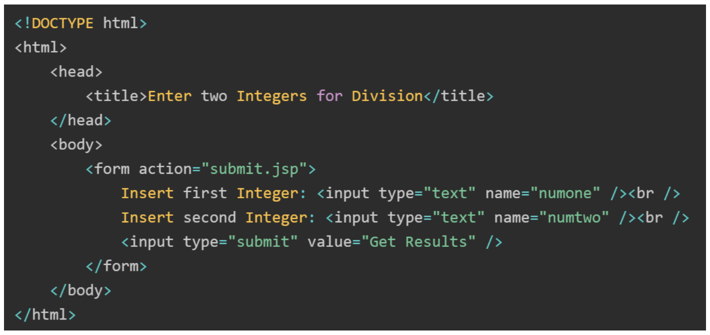
5. The application Implicit Object

An application object is another implicit object implemented to initialize application-wide parameters and maintain functional data throughout the JSP application.

6. The exception Implicit Object

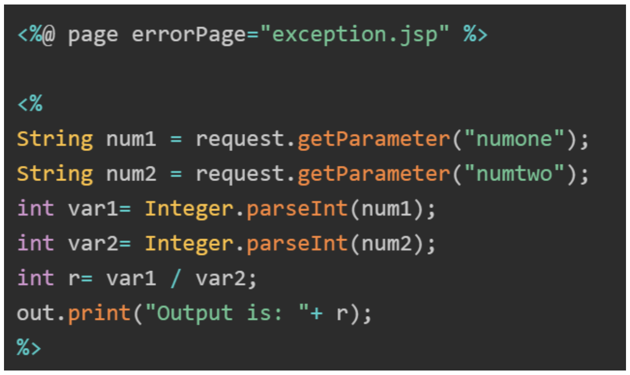
- An exception implicit object is implemented to handle exceptions to display error messages.
- The exception implicit object is an instance of the java.lang.Throwable class.
- It is only available for JSP pages, with the isErrorPage value set as "True". This means Exception objects can only be used in error pages.

Example(HTML file)



```html
<!DOCTYPE html>
<html>
    <head>
        <title>Enter two Integers for Division</title>
    </head>
    <body>
        <form action="submit.jsp">
            Insert first Integer: <input type="text" name="numone" /><br />
            Insert second Integer: <input type="text" name="numtwo" /><br />
            <input type="submit" value="Get Results" />
        </form>
    </body>
</html>
```

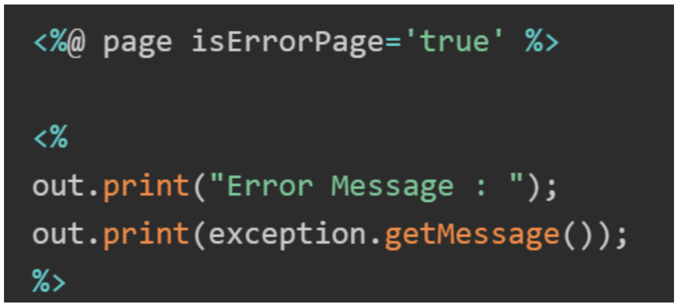
Example(Submit.jsp)



```jsp
<%@ page errorPage="exception.jsp" %>

<%
String num1 = request.getParameter("numone");
String num2 = request.getParameter("numtwo");
int var1= Integer.parseInt(num1);
int var2= Integer.parseInt(num2);
int r= var1 / var2;
out.print("Output is: "+ r);
%>
```

Example(exception.jsp)



```jsp
<%@ page isErrorPage='true' %>

<%
out.print("Error Message : ");
out.print(exception.getMessage());
%>
```

7. Page Implicit Object

- A page object is an implicit object that is referenced to the current instance of the servlet.
- You can use it instead. Covering it specifically is hardly ever used and not a valuable implicit object while building a JSP application.

```jsp
<% String pageName = page.toString();
out.println("The current page is: " +pageName);%>
```

8. The config Implicit Object

- A config object is a configuration object of a servlet that is mainly used to access and receive configuration information such as servlet context, servlet name, configuration parameters, etc.
- It uses various methods used to fetch configuration information.