Java Server Pages (JSP)

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.

Features of JSP

- **Coding in JSP is easy**: - As it is just adding JAVA code to HTML/XML.
- **Reduction in the length of Code**: - In JSP we use action tags, custom tags etc.
- **Connection to Database is easier**: - It is easier to connect website to database and allows to read or write data easily to the database.
- **Make Interactive websites**: - In this we can create dynamic web pages which helps user to interact in real time environment.
- **Portable, Powerful, flexible and easy to maintain**: - as these are browser and server independent.
- **No Redeployment and No Re-Compilation**: - It is dynamic, secure and platform independent so no need to re-compilation.
- **Extension to Servlet**: - as it has all features of servlets, implicit objects and custom tags.

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.

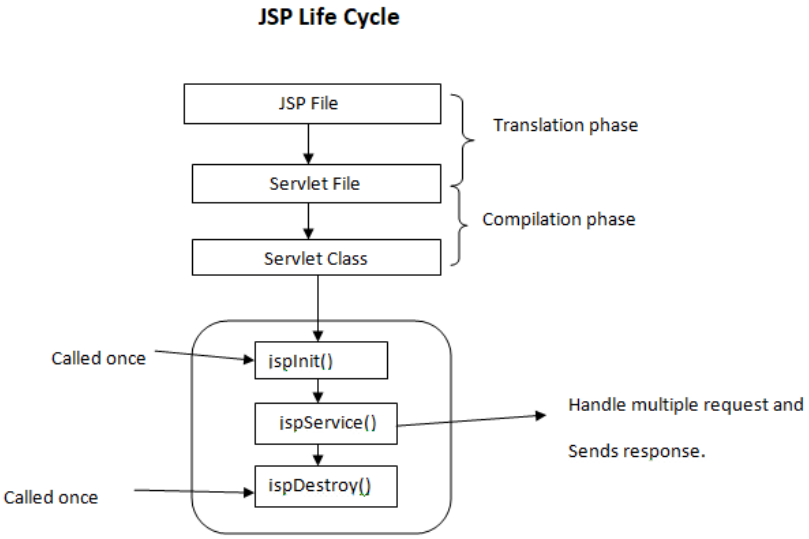
5) JSP has access to entire API of JAVA.

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).

Note: **jspInit()**, **_jspService()** and **jspDestroy()** are the life cycle methods of JSP. We can override jspInit(), jspDestroy() but we can't override _jspService() method.



As depicted in the above diagram, 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. Moreover, all the processes that happen in Servlet are performed on JSP later like initialization, committing response to the browser and destroy.

Translation of JSP page to Servlet: This is the first step of the JSP life cycle. This translation phase deals with the Syntactic correctness of JSP. Here test.jsp file is translated to test.java.
Compilation of JSP page: Here the generated java servlet file (test.java) is compiled to a class file (test.class).
Classloading: Servlet class which has been loaded from the JSP source is now loaded into the container.
Instantiation: Here an instance of the class is generated. The container manages one or more instances by providing responses to requests.
Initialization: jspInit() method is called only once during the life cycle immediately after the generation of Servlet instance from JSP.

Request processing: _jspService() method is used to serve the raised requests by JSP. It takes request and response objects as parameters. This method cannot be overridden.
JSP Cleanup: In order to remove the JSP from the use by the container or to destroy the method for servlets jspDestroy()method is used. This method is called once, if you need to perform any cleanup task like closing open files, releasing database connections jspDestroy() can be overridden.

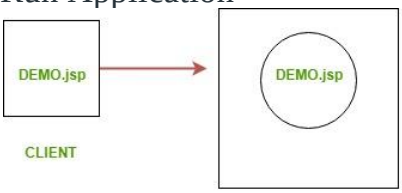
JSP syntax

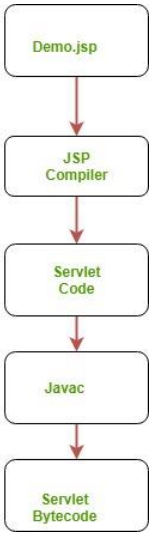
Syntax available in JSP are following

1. **Declaration Tag:** -It is used to declare variables.
   Syntax: - <%! Dec var %>
   Example: - <%! int var=10; %>
2. **Java Scriplets**: - It allows us to add any number of JAVA code, variables and expressions.
   Syntax: - <% java code %>
3. **JSP Expression**: - It evaluates and convert the expression to a string.
   Syntax: - <%= expression %>
   Example: - <% num1 = num1+num2 %>
4. **JAVA Comments**: - It contains the text that is added for information which has to be ignored.
   Syntax: - <% -- JSP Comments %>

Process of Execution: Steps for Execution of JSP are following: -

- Create html page from where request will be sent to server eg try.html.
- To handle to request of user next is to create .jsp file Eg. new.jsp
- Create project folder structure.
- Create XML file eg my.xml.
- Create WAR file.
- Start Tomcat
- Run Application





Example of Hello World: We will make one .html file and .jsp file

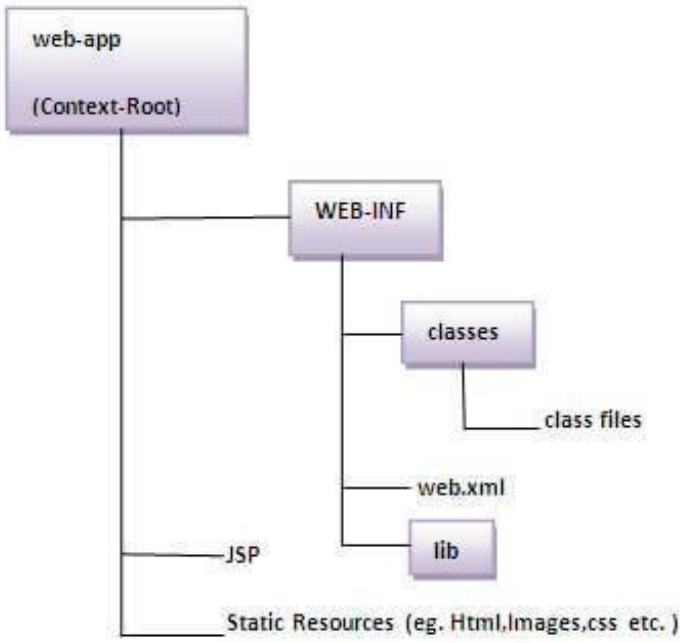
demo.jsp

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Hello World - JSP tutorial</title>
</head>
<body>
<%= "Hello World!" %>
</body>
</html>
```

There is no need of directory structure if you don't have class files or TLD files. For example, put JSP files in a folder directly and deploy that folder. It will be running fine. However, if you are using Bean class, Servlet or TLD file, the directory structure is required.

The Directory structure of JSP

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



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 three different scripting elements:

- **Declaration <%! declarations %>**
- **Scriptlet <% scriplets %>**
- **Expression <%= expression %>**

Declaration

As the name suggests, 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>
```

JSP 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>
```

Expressions

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>
```

There are two types of Scripting elements extra mentioned by some developers.

Comment <%-- Set of comment statements --%> & Directive <%@ directive %>

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>
```

Directives

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 ... %>

Implicit Objects in jsp:

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 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.
- You will learn more about the various concepts of the out Object in subsequent chapters.

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
```

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 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.

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

Example (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>
```

Example (login.jsp):

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

The 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.
- You will learn more about the various concepts of the request and response in subsequent chapters.

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.
- You will learn more about the use of the session in subsequent chapters.

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.

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.

```
<!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):

```
<%@ 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):

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

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

The 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.

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

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.

In later chapters, we will learn more about how to set and use config objects.

Directive Elements:

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

Let us now discuss each of them in detail.

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. However, in general, 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.
8. **info**: This "info" attribute sets the JSP page information, which is later obtained using the getServletInfo() method of the servlet interface.
9. **isThreadSafe**: Both the servlet and JSP are multithreaded. If you want to control JSP page behavior and define the threading model, you can use the "isThreadSafe" attribute of the page directive.
10. **Language**: The language attribute specifies the programming language used in the JSP page. The default value of the language attribute is "Java".
11. **Session**: In JSP, the page directive session attribute specifies whether the current JSP page participates in the current HTTP session.
12. **isELIgnored**: This isELIgnored attribute is used to specify whether the expression language (EL) implied by the JSP page will be ignored.
13. **isScriptingEnabled**: This "isScriptingEnabled" attribute determines if the scripting elements are allowed for use or not.

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>
```

In the above example of JSP code, a JSP header file is being added to the current JSP file using "include directive".

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>
```

Exception Handling in JSP

Exceptions can be defined as an object that is thrown during a program run. Exception handling is the practice of handling errors at runtime. Exceptions can occur at any time in a web application. Therefore, the web developer must handle exceptions to be on the safe side and make the user work flawlessly on the web.

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.

Types of Exceptions in JSP

- **Checked Exception**
- **Runtime Exception**
- **Errors Exception**

Checked Exceptions

"Checked exceptions" are a type of exception that is usually a user fault or a problem that cannot be foreseen by the programmer. This type of exception is checked at compile-time. Here is a list of some common "Checked exceptions" that occur in the programming domain:

- **IO Exception**: This is an example of a checked exception where some exceptions may occur while reading and writing a file. If the file format is not supported, then the IO exception will occur.
- **FileNotFoundException**: This is an example of a checked exception where a file in which data needs to be written is not found on that particular path on the disk.
- **SQLException**: This is also an example of a checked exception where the file is associated with a SQL database. The exception will be if there is a problem or concern with the connectivity of the SQL database.

Runtime Exceptions

"Runtime exceptions" can be defined as exceptions evaded by the programmer. They are left unnoticed at compilation time. Here is the list of some of the common examples that occurred in the programming domain:

- **ArrayIndexOutOfBoundsException**: This is a runtime exception; This occurs when the array size goes beyond the elements or index value set.
- **NullPointer Exception**: This is also an example of a runtime exception raised when a variable or object is empty or null, and users or programmers try to access it.
- **ArithmeticException**: This is an example of a runtime exception when a mathematical operation is not allowed under normal circumstances. A common example of such a scenario is to divide the number by 0.

Error Exception

"Error exception" arises solely because of the user or programmer. Here you might encounter error due to stack overflows. Here is the list of some of the common examples that occurred in the programming domain:

- **Error**: This is a subclass of throwable that indicates serious problems the applications cannot catch.
- **Internal Error**: This error occurs when a fault occurs in the Java Virtual Machine (JVM).
- **Instantiation error**: This error occurs when you try to instantiate an object, ultimately failing to do that.

ArithmeticException Example

Example (HTML file):

```
<!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):

```
<%@ 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):

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

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

JSP Action Tags

There are many JSP action tags or elements. Each JSP action tag is used to perform some specific tasks.

The action tags are used to control the flow between pages and to use Java Bean. The Jsp action tags are given below.

JSP Action Tags	Description
jsp:forward	forwards the request and response to another resource.
jsp:include	includes another resource.
jsp:useBean	creates or locates bean object.
jsp:setProperty	sets the value of property in bean object.
jsp:getProperty	prints the value of property of the bean.
jsp:plugin	embeds another component such as applet.
jsp:param	sets the parameter value. It is used in forward and include mostly.
jsp:fallback	can be used to print the message if plugin is working. It is used in jsp:plugin.

The jsp:useBean, jsp:setProperty and jsp:getProperty tags are used for bean development. So we will see these tags in bean developement.

jsp:forward action tag: The jsp:forward action tag is used to forward the request to another resource it may be jsp, html or another resource.

Syntax of jsp:forward action tag without parameter

1. <jsp:forward page="relativeURL | <%= expression %>" />

Syntax of jsp:forward action tag with parameter

1. <jsp:forward page="relativeURL | <%= expression %>">
2. <jsp:param name="parametername" value="parametervalue | <%=expression%>" />
3. </jsp:forward>

Example of jsp:forward action tag without parameter

In this example, we are simply forwarding the request to the printdate.jsp file.

index.jsp

```
<html>
<body>
<h2>this is index page</h2>

<jsp:forward page="printdate.jsp" />
</body>
</html>
```

printdate.jsp

```
<html>
<body>
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
</body>
</html>
```

Example of jsp:forward action tag with parameter

In this example, we are forwarding the request to the printdate.jsp file with parameter and printdate.jsp file prints the parameter value with date and time.

index.jsp

```
<html>
<body>
<h2>this is index page</h2>

<jsp:forward page="printdate.jsp" >
<jsp:param name="name" value="google.com" />
</jsp:forward>
</body>
</html>
```

printdate.jsp

```
<html>
<body>

<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
<%= request.getParameter("name") %>

</body>
</html>
```