JSP (Java Server Pages)

Java Server Pages (JSP) is a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications.

JSP have access to the entire family of Java APIs, including the JDBC API to access enterprise databases. It provides more functionality than a 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.

This helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.

A JavaServer Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands.

Using JSP, you can collect input from users through Webpage forms, present records from a database or another source, and create Webpages dynamically.

JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages, and sharing information between requests, pages etc.

Advantages of JSP

Following table lists out the other advantages of using JSP over other technologies −

vs. Common Gateway Interface (CGI).

JavaServer Pages often serve the same purpose as programs implemented using the **Common Gateway Interface (CGI)**. But JSP offers several advantages in comparison with the CGI.

- Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having separate CGI files.
- JSP are always compiled before they are processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.
- JavaServer Pages are built on top of the Java Servlets API, so like Servlets, JSP also has access to all the powerful Enterprise Java APIs, including **JDBC, JNDI, EJB, JAXP,** etc.
- JSP pages can be used in combination with servlets that handle the business logic, the model supported by Java servlet template engines.

Finally, JSP is an integral part of Java EE, a complete platform for enterprise class applications. This means that JSP can play a part in the simplest applications to the most complex and demanding.

vs. Active Server Pages (ASP)

The advantages of JSP are twofold. First, the dynamic part is written in Java, not Visual Basic or other MS specific language, so it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft Web servers.

vs. Pure Servlets

It is more convenient to write (and to modify!) regular HTML than to have plenty of println statements that generate the HTML.

JSP	Servlets
Extension to Servlet	Not an extension to servlet
Easy to Maintain	Bit complicated
No need to recompile or redeploy	The code needs to be recompiled
Less code than a servlet	More code compared to JSP

vs. Server-Side Includes (SSI)

SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.

vs. JavaScript

JavaScript can generate HTML dynamically on the client but can hardly interact with the web server to perform complex tasks like database access and image processing etc.

vs. Static HTML

Regular HTML, of course, cannot contain dynamic information.

JSP - Environment Setup

A development environment is where you would develop your JSP programs, test them and finally run them. Setting up JSP development environment involves the following steps −

1. **Setting up Java Development Kit**

This step involves downloading an implementation of the Java Software Development Kit (SDK) and setting up the PATH environment variable appropriately.

Once you download your Java implementation, follow the given instructions to install and configure the setup. Finally set the **PATH and JAVA_HOME** environment variables to refer to the directory that contains **java** and **javac**, typically **java_install_dir/bin** and **java_install_dir** respectively.

set PATH = C:\jdk1.5.0_20\bin;%PATH%

set JAVA_HOME = C:\jdk1.5.0_20

Alternatively, on **Windows NT/2000/XP**, you can also right-click on **My Computer**, select **Properties**, then **Advanced**, followed by **Environment Variables**. Then, you would update the PATH value and press the OK button.

2. Setting up Web Server: Tomcat

A number of Web Servers that support JavaServer Pages and Servlets development are available in the market. Some web servers can be downloaded for free and Tomcat is one of them.

Apache Tomcat is an open source software implementation of the JavaServer Pages and Servlet technologies and can act as a standalone server for testing JSP and Servlets, and can be integrated with the Apache Web Server. Here are the steps to set up Tomcat on your machine −

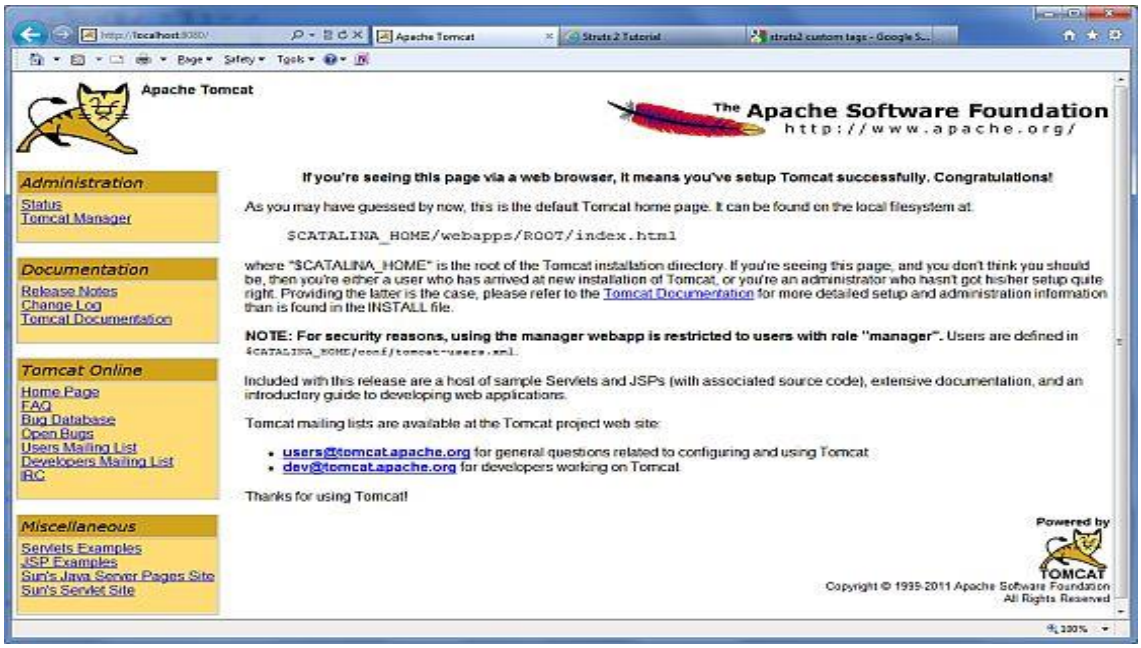
- Download the latest version of Tomcat from https://tomcat.apache.org/.
- Once you downloaded the installation, unpack the binary distribution into a convenient location. Create **CATALINA_HOME** environment variable pointing to these locations.

Tomcat can be started by executing the following commands on the Windows machine −

```
%CATALINA_HOME%\bin\startup.bat

or

C:\apache-tomcat-5.5.29\bin\startup.bat
```

After a successful startup, the default web-applications included with Tomcat will be available by visiting **http://localhost:8080/**.

Upon execution, you will receive the following output −



Tomcat can be stopped by executing the following commands on the Windows machine −

```
%CATALINA_HOME%\bin\shutdown
or

C:\apache-tomcat-5.5.29\bin\shutdown
```

3. Setting up CLASSPATH

Since servlets are not part of the Java Platform, Standard Edition, you must identify the servlet classes to the compiler.

If you are running Windows, you need to put the following lines in your **C:\autoexec.bat** file.

```
set CATALINA = C:\apache-tomcat-5.5.29
set CLASSPATH = %CATALINA%\common\lib\jsp-api.jar;%CLASSPATH%
```

Alternatively, on **Windows NT/2000/XP**, you can also right-click on **My Computer**, select **Properties**, then **Advanced**, then **Environment Variables**. Then, you would update the CLASSPATH value and press the OK button.

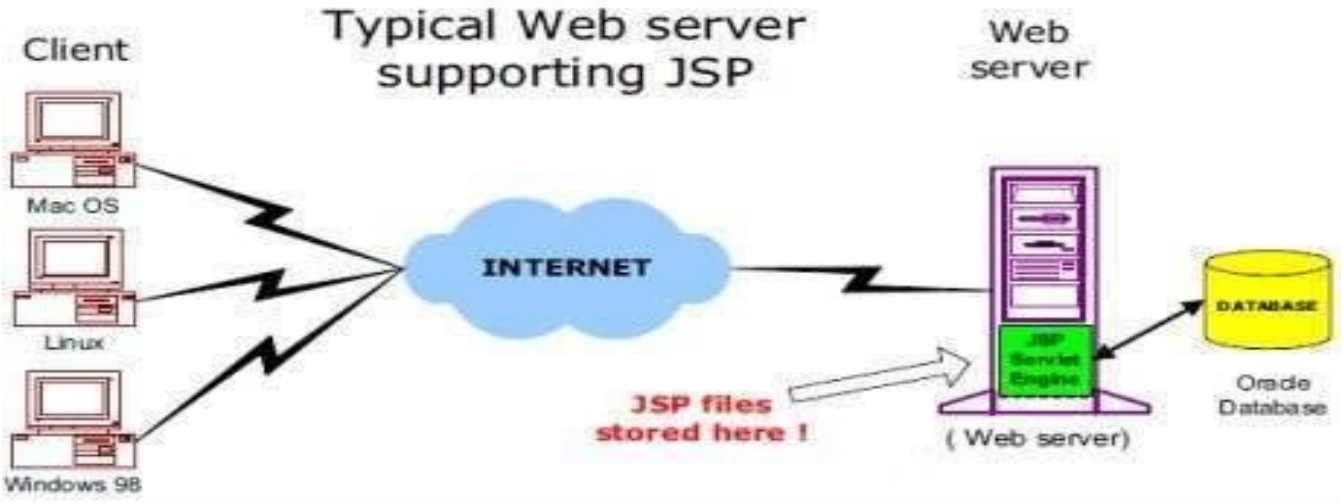
JSP - Architecture

The web server needs a JSP engine, i.e, a container to process JSP pages. The JSP container is responsible for intercepting requests for JSP pages.

A JSP container works with the Web server to provide the runtime environment and other services a JSP needs. It knows how to understand the special elements that are part of JSPs.

Following diagram shows the position of JSP container and JSP files in a Web application.

JSP Processing

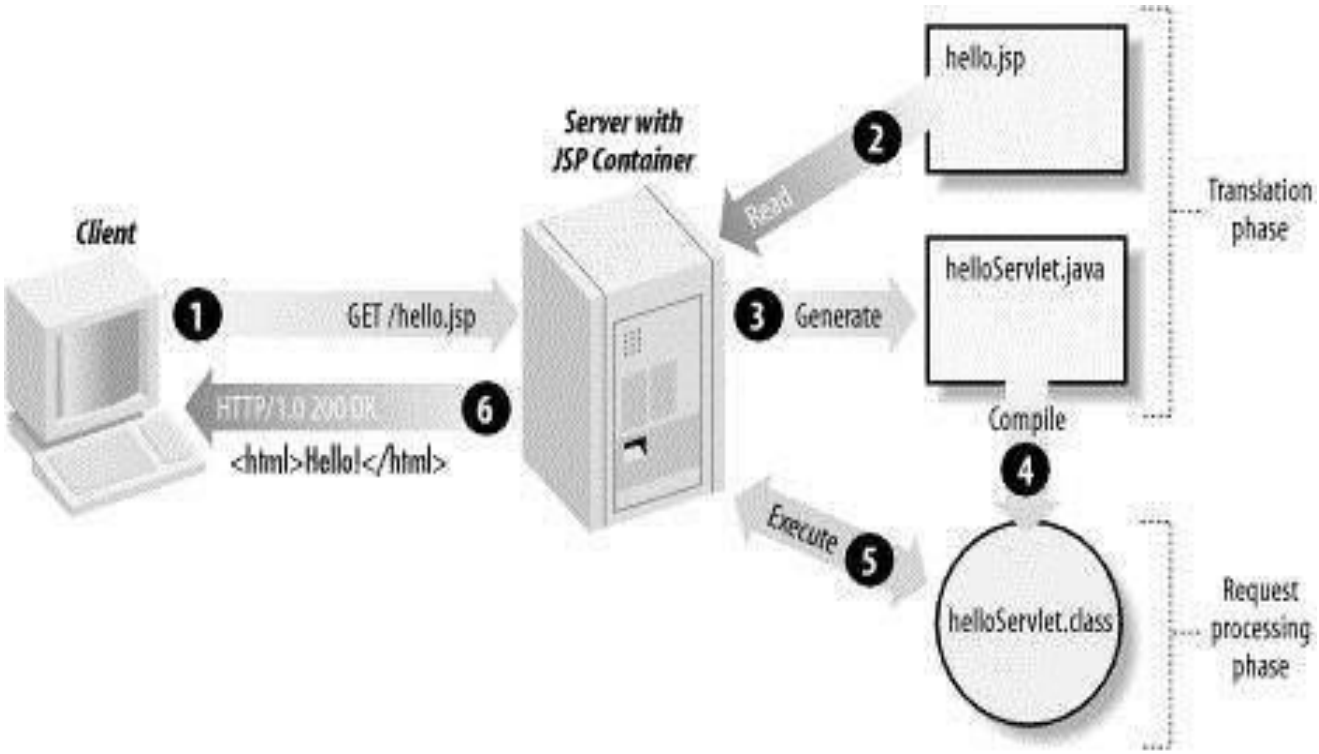


The following steps explain how the web server creates the Webpage using JSP −

- As with a normal page, your browser sends an HTTP request to the web server.

- The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with **.jsp** instead of **.html**.
- The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to println( ) statements and all JSP elements are converted to Java code. This code implements the corresponding dynamic behavior of the page.
- The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
- A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format. The output is furthur passed on to the web server by the servlet engine inside an HTTP response.
- The web server forwards the HTTP response to your browser in terms of static HTML content.
- Finally, the web browser handles the dynamically-generated HTML page inside the HTTP response exactly as if it were a static page.

All the above mentioned steps can be seen in the following diagram −



Typically, the JSP engine checks to see whether a servlet for a JSP file already exists and whether the modification date on the JSP is older than the servlet. If the JSP is older than its generated servlet, the JSP container assumes that the JSP hasn't changed and that the generated servlet still matches the JSP's contents. This makes the process more efficient than with the other scripting languages (such as PHP) and therefore faster.

So in a way, a JSP page is really just another way to write a servlet without having to be a Java programming wiz. Except for the translation phase, a JSP page is handled exactly like a regular servlet.

JSP - Lifecycle

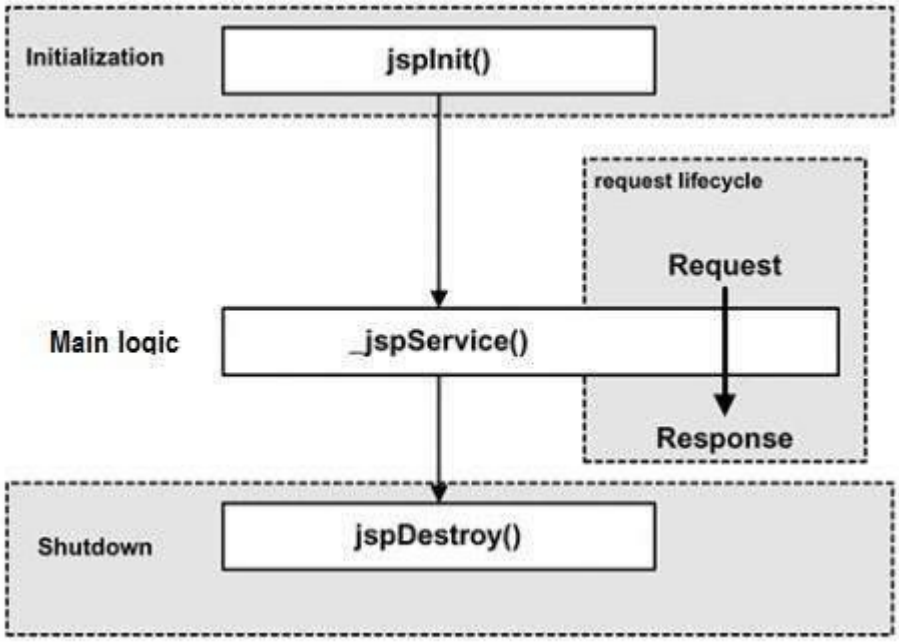
A JSP life cycle is defined as the process from its creation till the destruction. This is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet.

Paths Followed By JSP

The following are the paths followed by a JSP −

- Compilation
- Initialization
- Execution
- Cleanup

The four major phases of a JSP life cycle are very similar to the Servlet Life Cycle. The four phases have been described below −



JSP Compilation

When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page. If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.

The compilation process involves three steps −

- Parsing the JSP.
- Turning the JSP into a servlet.
- Compiling the servlet.

JSP Initialization

When a container loads a JSP it invokes the **jspInit()** method before servicing any requests. If you need to perform JSP-specific initialization, override the **jspInit()** method −

```
public void jspInit(){
  // Initialization code...
}
```

Typically, initialization is performed only once and as with the servlet init method, you generally initialize database connections, open files, and create lookup tables in the jspInit method.

JSP Execution

This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed.

Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the **_jspService()** method in the JSP.

The _jspService() method takes an **HttpServletRequest** and an **HttpServletResponse** as its parameters as follows −

```
void _jspService(HttpServletRequest request, HttpServletResponse response) {
  // Service handling code...
}
```

The **_jspService()** method of a JSP is invoked on request basis. This is responsible for generating the response for that request and this method is also responsible for generating responses to all seven of the HTTP methods, i.e, **GET, POST, DELETE**, etc.

JSP Cleanup

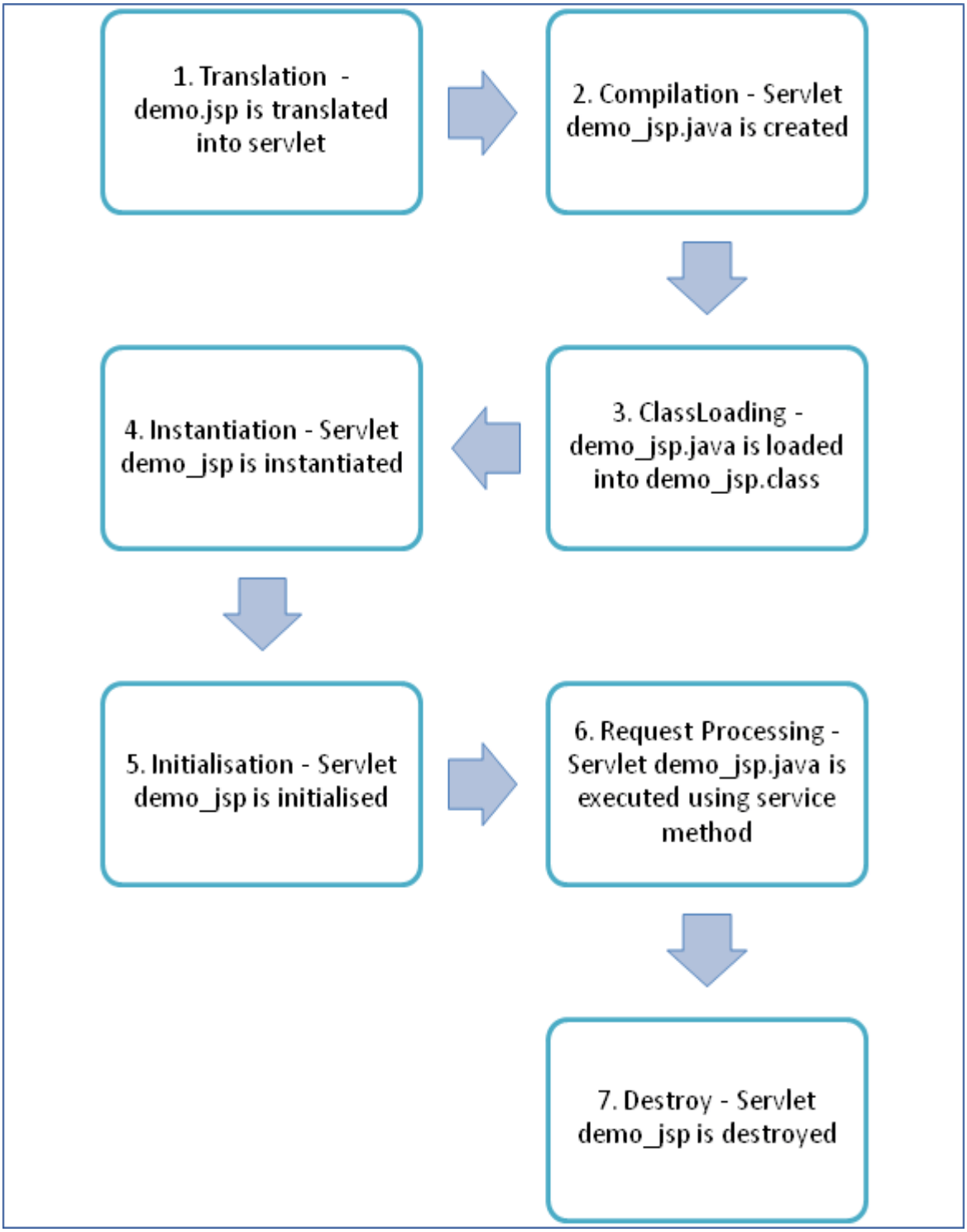
The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.

The **jspDestroy()** method is the JSP equivalent of the destroy method for servlets. Override jspDestroy when you need to perform any cleanup, such as releasing database connections or closing open files.

The jspDestroy() method has the following form −

```
public void jspDestroy() {
  // Your cleanup code goes here.
}
```

JSP Lifecycle is depicted in the below diagram.



Following steps explain the JSP life cycle:

1. Translation of JSP page
2. Compilation of JSP page(Compilation of JSP page into _jsp.java)
3. Classloading (_jsp.java is converted to class file _jsp.class)
4. Instantiation(Object of generated servlet is created)
5. Initialisation(_jspinit() method is invoked by container)

6. Request Processing(_jspservice() method is invoked by the container)
7. Destroy (_jspDestroy() method invoked by the container)

Let us have more detailed summary on the above points:

1. **Translation of the JSP Page:**

A Java servlet file is generated from a JSP source file. This is the first step of JSP life cycle. In translation phase, container validates the syntactic correctness of JSP page and tag files.

- The JSP container interprets the standard directives and actions, and the custom actions referencing tag libraries (they are all part of JSP page and will be discussed in the later section) used in this JSP page.
- In the above pictorial description, demo.jsp is translated to demo_jsp.java in the first step
- Let's take an example of "demo.jsp" as shown below:

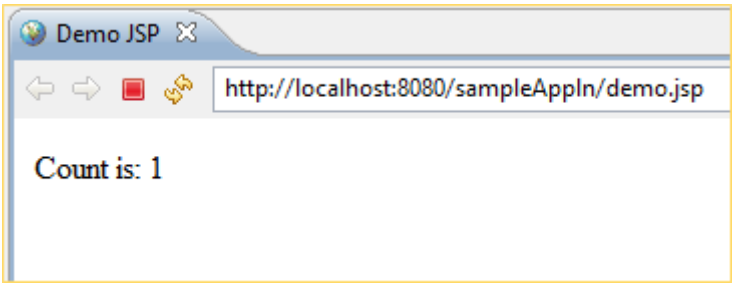
Demo.jsp

<html>
<head>
<title>Demo JSP</title>
</head>
<%
int demovar=0;%>
<body>
Count is:
<% Out.println(++demovar); %>
<body>
</html>

Demo JSP Page is converted into demo_jsp servlet in the below code.

```
1  Public class demp_jsp extends HttpServlet{
2     Public void _jspservice(HttpServletRequest request, HttpServletResponse response)
3           Throws IOException, ServletException
4       {
5  PrintWriter out = response.getWriter();
6  response.setContentType("text/html");
7  out.write("<html><body>");
8  int demovar=0;
9  out.write("Count is:");
10 out.print(demovar++);
11 out.write("</body></html>");
12 }
13 }
14
```

Output:



- Here you can see that in the screenshot the Output is 1 because demovar is initialized to 0 and then incremented to 0+1=1

In the above example,

- demo.jsp, is a JSP where one variable is initialized and incremented. This JSP is converted to the servlet (demo_jsp.class) wherein the JSP engine loads the JSP Page and converts to servlet content.
- When the conversion happens all template text is converted to println() statements and all JSP elements are converted to Java code.

This is how a simple JSP page is translated into a servlet class.

2. **Compilation of the JSP Page**

- The generated java servlet file is compiled into java servlet class
- The translation of java source page to its implementation class can happen at any time between the deployment of JSP page into the container and processing of the JSP page.
- In the above pictorial description demo_jsp.java is compiled to a class file demo_jsp.class

3. **Classloading**

- Servlet class that has been loaded from JSP source is now loaded into the container

4. **Instantiation**

- In this step the object i.e. the instance of the class is generated.
- The container manages one or more instances of this class in the response to requests and other events. Typically, a JSP container is built using a servlet container. A JSP container is an extension of servlet container as both the container support JSP and servlet.
- A JSPPage interface which is provided by container provides init() and destroy () methods.
- There is an interface HttpJSPPage which serves HTTP requests, and it also contains the service method.

5. **Initialization**

   public void jspInit()
   {
   //initializing the code
   }

- _jspinit() method will initiate the servlet instance which was generated from JSP and will be invoked by the container in this phase.
- Once the instance gets created, init method will be invoked immediately after that
- It is only called once during a JSP life cycle, the method for initialization is declared as shown above

6. **Request processing**

   void _jspservice(HttpServletRequest request HttpServletResponse response)
   {
   //handling all request and responses
   }

- _jspservice() method is invoked by the container for all the requests raised by the JSP page during its life cycle
- For this phase, it has to go through all the above phases and then only service method can be invoked.
- It passes request and response objects
- This method cannot be overridden
- The method is shown above: It is responsible for generating of all HTTP methods i.eGET, POST, etc.

7. **Destroy**

   public void _jspdestroy()
   {
   //all clean up code
   }

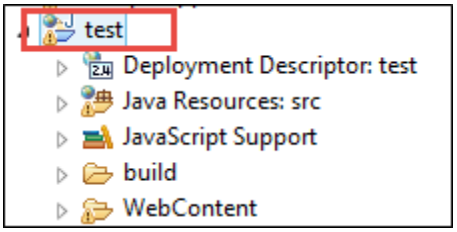
- _jspdestroy() method is also invoked by the container
- This method is called when container decides it no longer needs the servlet instance to service requests.
- When the call to destroy method is made then, the servlet is ready for a garbage collection
- This is the end of the life cycle.
- We can override jspdestroy() method when we perform any cleanup such as releasing database connections or closing open files.

How to run simple JSP Page

- JSP can be run on web servers or application servers.

- Here we will be using a webserver, and we can deploy it on the server enclosing it in a war application.
- We can create JSP in an application.

This is an application which has following directory structure, and the application has to be build.

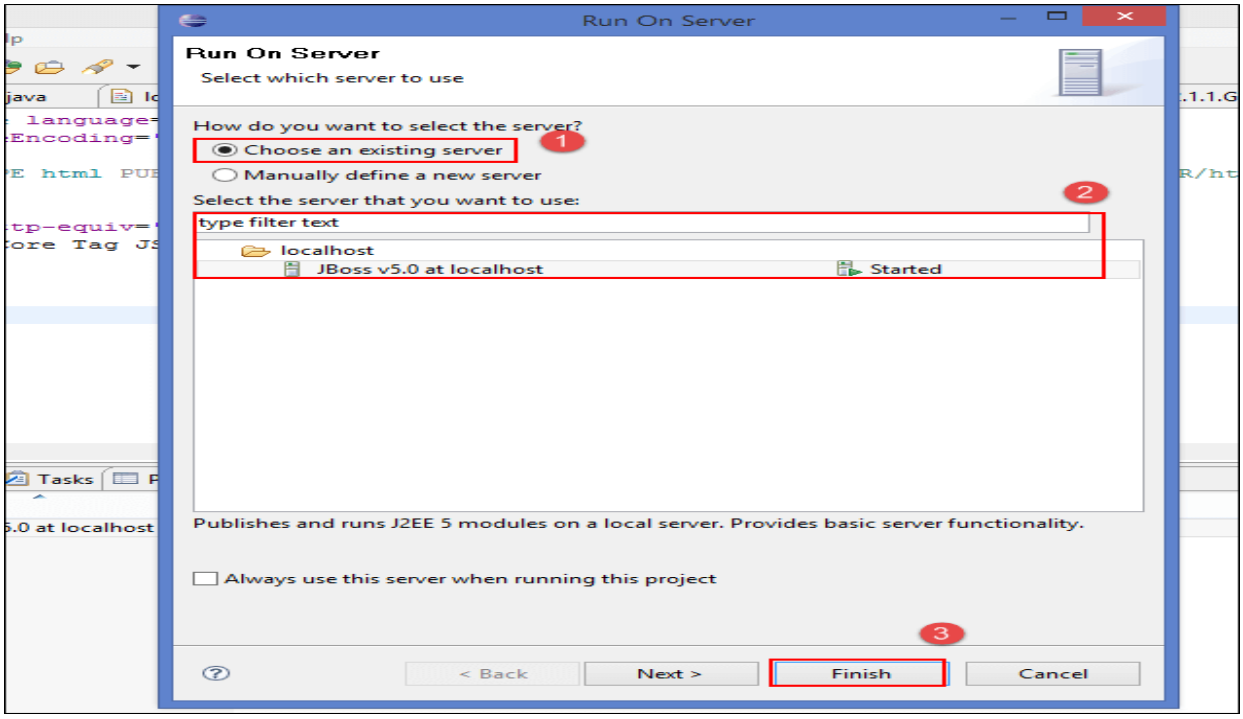


After the application is built then, the application has to be run on the server.

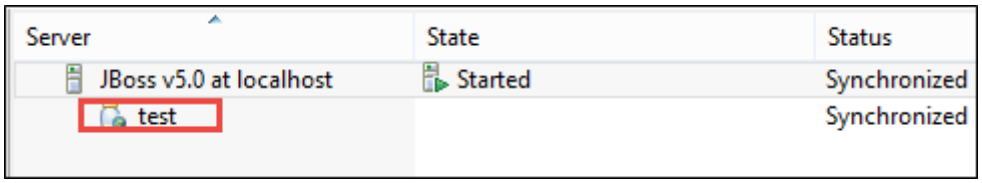
To run JSP on the webserver, right click on the project of the IDE (eclipse used in this case) and there are many options. Select the option of run on the server. It is shown in the screenshot below;

From the diagram, following points are explained:

1. There are two options either to choose a server or manually add the server to this application. In this case, we have already added JBoss server to the application hence, we select the existing server.
2. Once we select the server the server option is shown in point 2 which server we want to select. There can be multiple servers configured on this application. We can select one server from all those options
3. Once that option is selected click on finish button and application will run on that server.



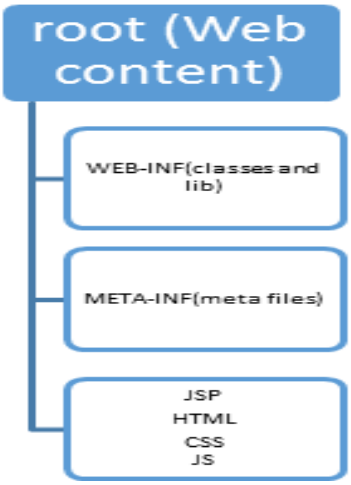
In the below screenshots, you can notice that our JSP program gets executed, and the test application is deployed in JBoss server marked in the red box.



Directory Structure of JSP

In directory structure, there is a root folder which has folder WEB-INF, which has all configuration files and library files.

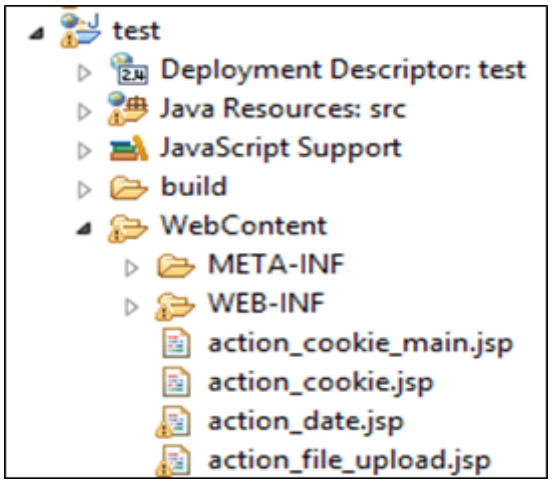
JSP files are outside WEB-INF folder



Directory structure of JSP

Example:

In this example there is test application which has folder structure has following:



Elements of JSP (JSP TAGS)

The elements of JSP have been described below −

The Scriptlet

A scriptlet can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language.

Following is the syntax of Scriptlet −

<% code fragment %>

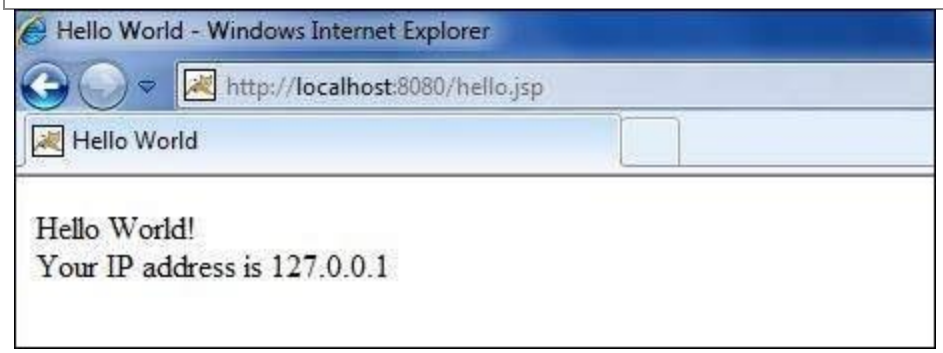
You can write the XML equivalent of the above syntax as follows −

```
<jsp:scriptlet>
  code fragment
</jsp:scriptlet>
```

Any text, HTML tags, or JSP elements you write must be outside the scriptlet. Following is the simple and first example for JSP –

Example 1:-

```
<html>
  <head><title>Hello World</title></head>

  <body>
    Hello World!<br/>
    <%
      out.println("Your IP address is " + request.getRemoteAddr());
    %>
  </body>
</html>
```



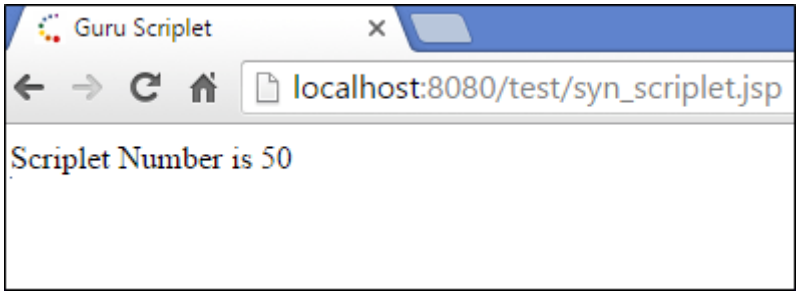
Example 2:

In this example, we are taking Scriptlet tags which enclose java code.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>JSP Scriplet</title>
</head>
<body>
<% int num1=10;
  int num2=40;
  int num3 = num1+num2;
  out.println("Scriplet Number is " +num3);
%>
</body>
</html>
```

When you execute the code, you get the following output:



Output:

The output for the Scriptlet Number is 50 which is addition of num1 and num2.

JSP Declarations

A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file.

Following is the syntax for JSP Declarations −

<%! Declaration ; %>

You can write the XML equivalent of the above syntax as follows −

```
<jsp:declaration>
  code fragment
</jsp:declaration>
```

Example 1:-

Following is an example for JSP Declarations −

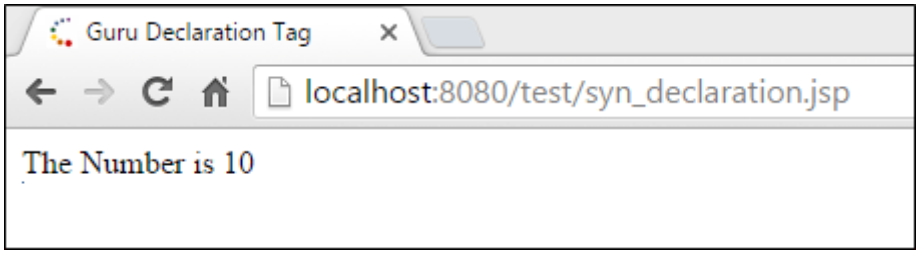
```
<%! int i = 0; %>
<%! int a, b, c; %>
<%! Circle a = new Circle(2.0); %>
```

Example2:

In this example, we are going to use the declaration tags

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>JSP Declaration Tag</title>
</head>
<body>
<%! int count =10; %>
<% out.println("The Number is " +count); %>
</body>
</html>
```

When you execute the above code you get the following output:



Output:

The variable which is declared in the declaration tag is printed as output.

JSP Expression

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

Because the value of an expression is converted to a String, you can use an expression within a line of text, whether or not it is tagged with HTML, in a JSP file.

The expression element can contain any expression that is valid according to the Java Language Specification but you cannot use a semicolon to end an expression.

Following is the syntax of JSP Expression −

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

You can write the XML equivalent of the above syntax as follows −

```
<jsp:expression>
  expression
</jsp:expression>
```

Following example shows a JSP Expression –

Example 1:-

```
<html>
```

```
  <head><title>A Comment Test</title></head>

  <body>
    <p>Today's date: <%= (new java.util.Date()).toLocaleString()%></p>
  </body>
</html>
```

The above code will generate the following result −

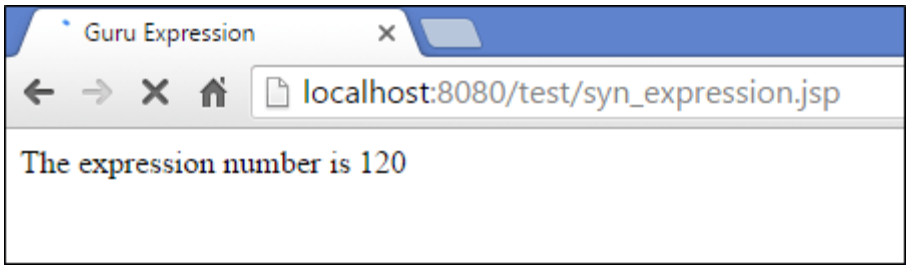
Today's date: 11-Sep-2010 21:24:25

Example 2:

In this example, we are using expression tag

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>JSP Expression</title>
</head>
<body>
<% out.println("The expression number is "); %>
<% int num1=10; int num2=10; int num3 = 20; %>
<%= num1*num2+num3 %>
</body>
</html>
```

When you execute the above code, you get the following output:



Output:

The expression number is 120 where we are multiplying two numbers num1 and num2 and adding that number with the third number.

JSP Comments

JSP comment marks text or statements that the JSP container should ignore. A JSP comment is useful when you want to hide or "comment out", a part of your JSP page.

Following is the syntax of the JSP comments −

<%-- This is JSP comment --%>

Following example shows the JSP Comments –

Example 1:-

```
<html>
  <head><title>A Comment Test</title></head>

  <body>
    <h2>A Test of Comments</h2>
    <%-- This comment will not be visible in the page source --%>
  </body>
</html>
```

The above code will generate the following result −

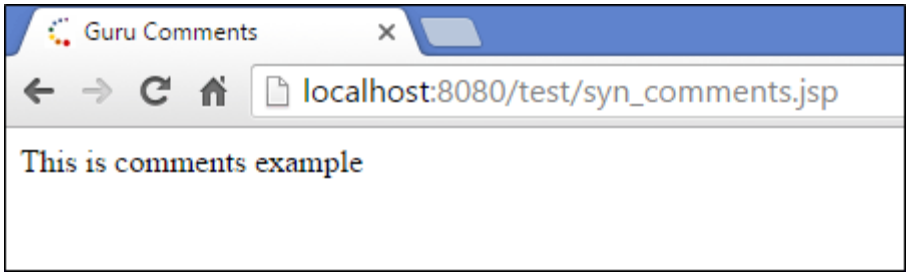
A Test of Comments

Example 2:

In this example, we are using JSP comments

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>JSP Comments</title>
</head>
<body>
<%-- JSP Comments section --%>
<% out.println("This is comments example"); %>

</body>
</html>
```

When you execute the above code you get the following output:



Creating a simple JSP Page

- A JSP page has an HTML body incorporated with Java code into it
- We are creating a simple JSP page which includes declarations, scriplets, expressions, comments tags in it.

Example:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>JSP Example</title>
</head>
<body>

<%-- This is a JSP example with scriplets, comments , expressions --%>
<% out.println("This is JSP Example"); %>
<% out.println("The number is "); %>
<%! int num12 = 12; int num32 = 12; %>
<%= num12*num32 %>
Today's date: <%= (new java.util.Date()).toLocaleString()%>
</body>
</html>
```

JSP Directives

- JSP directives are the messages to JSP container. They provide global information about an entire JSP page.
- JSP directives are used to give special instruction to a container for translation of JSP to servlet code.
- In JSP life cycle phase, JSP has to be converted to a servlet which is the translation phase.
- They give instructions to the container on how to handle certain aspects of JSP processing
- Directives can have many attributes by comma separated as key-value pairs.
- In JSP, directive is described in <%@ %> tags.

Syntax of Directive:

<%@ directive attribute="" %>

There are three types of directives:

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

JSP Page directive

Syntax of Page directive:

<%@ page…%>

- It provides attributes that get applied to entire JSP page.
- It defines page dependent attributes, such as scripting language, error page, and buffering requirements.
- It is used to provide instructions to a container that pertains to current JSP page.

Following are its list of attributes associated with page directive:

1. Language
2. contentType
3. pageEncoding
4. Import
5. errorPage
6. IsErrorPage
7. info
8. session
9. Extends
10. isThreadSafe
11. autoflush
12. buffer
13. isELIgonored

More details about each attribute

1. **language**: It defines the programming language (underlying language) being used in the page.

   Syntax of language:

   <%@ page language="value" %>
   Here value is the programming language (underlying language)

2. **contentType**:

- It defines the character encoding scheme i.e. it is used to set the content type and the character set of the response
- The default type of contentType is “text/html; charset=ISO-8859-1”.

Syntax of the contentType:

<%@ page contentType="value" %>

3. **PageEncoding:**

The “pageEncoding” attribute defines the character encoding for JSP page.

The default is specified as “ISO-8859-1” if any other is not specified.

Syntax of pageEncoding:

<%@ page pageEncoding="vaue" %>
Here value specifies the charset value for JSP

4. **Import**: This attribute is most used attribute in page directive attributes.It is used to tell the container to import other java classes, interfaces, enums, etc. while generating servlet code.It is similar to import statements in java classes, interfaces.

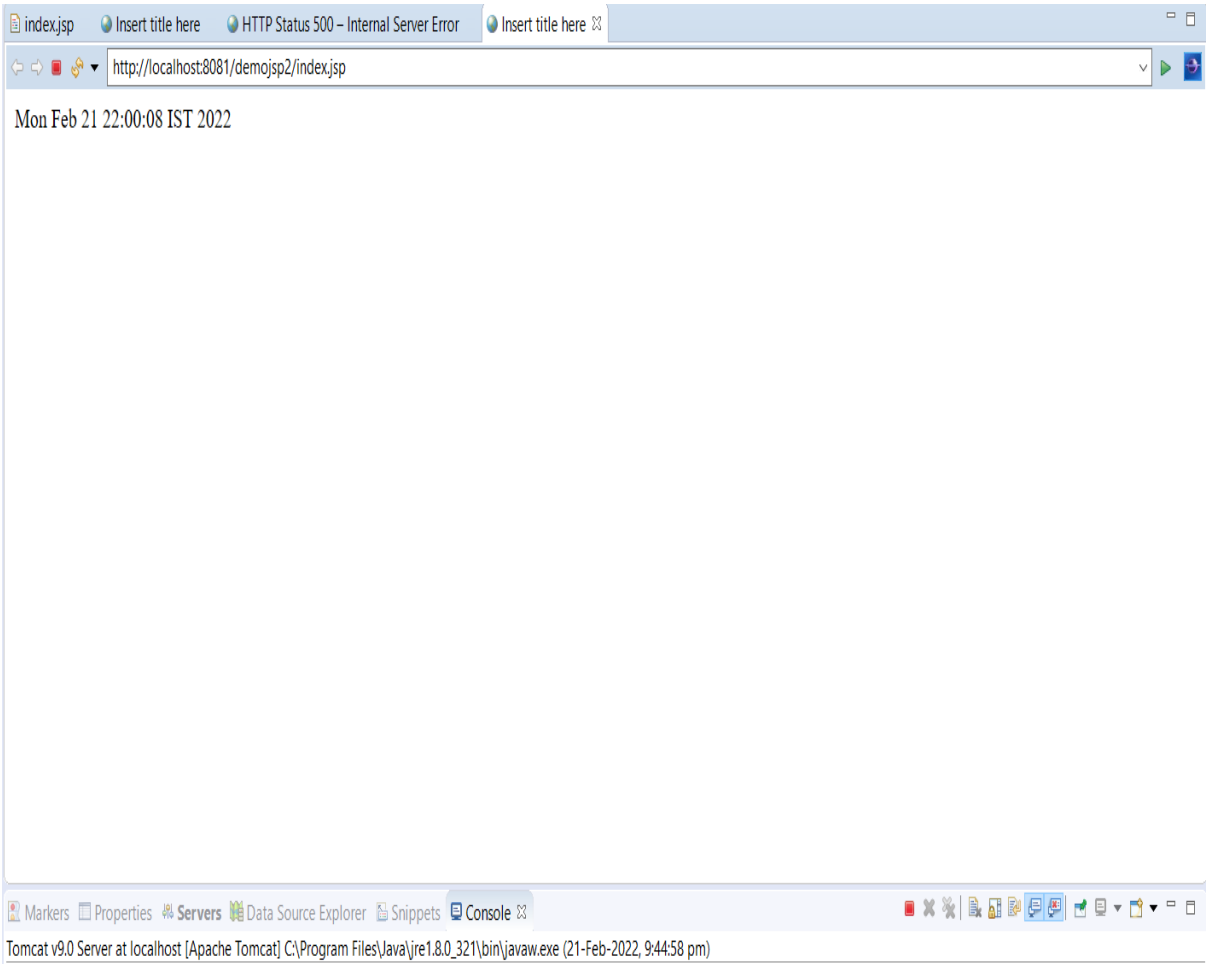
Syntax of import:

<%@ page import="value" %>
Here value indicates the classes which have to be imported.

Example:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<%@ page import="java.util.Date"%>
<%

out.println(new Date().toString());

%>
</body>
</html>
```

OUTPUT:-



5. **isErrorPage:**

- It indicates that JSP Page that has an errorPage will be checked in another JSP page
- Any JSP file declared with “isErrorPage” attribute is then capable to receive exceptions from other JSP pages which have error pages.
- Exceptions are available to these pages only.
- The default value is false.

Syntax of isErrorPage:

<%@ page isErrorPage="true/false"%>
Example:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    isErrorPage="true"%>
```

6. **errorPage:**

This attribute is used to set the error page for the JSP page if JSP throws an exception and then it redirects to the exception page.

Syntax of errorPage:

<%@ page errorPage="value" %>
Here value represents the error JSP page value

Example:

```
<%@ page language="java" contentType="text/html;" pageEncoding="ISO-8859-1"
   errorPage="errorHandler.jsp"%>
```

EXAMPLE BASED ON ERROR HANDLING IN JSP

1. **Index.jsp**

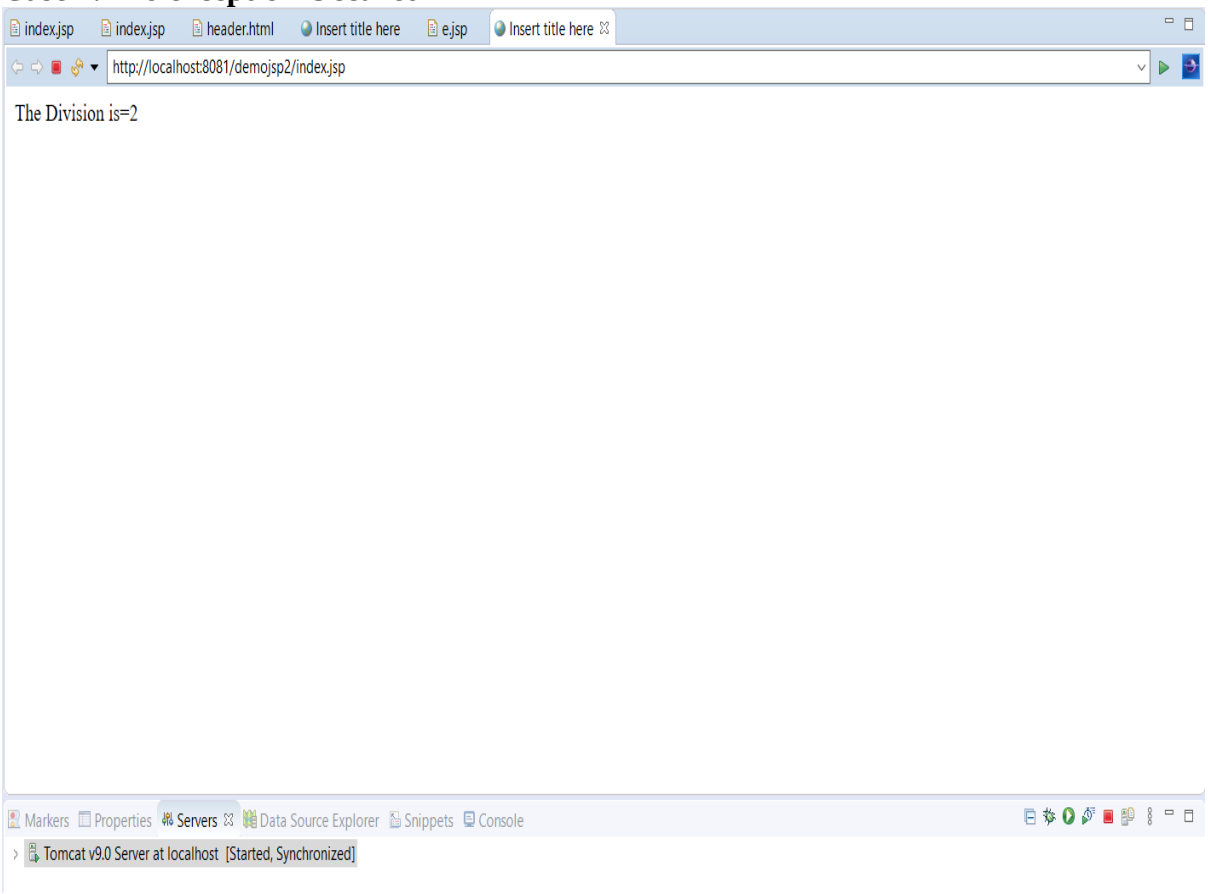
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<%@ page errorPage="e.jsp" %>
<%
int x=10;
int y=5;
out.println("The Division is="+(x/y));

%>
</body>
</html>
```

2. **e.jsp**

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<%@ page isErrorPage="true" %>
<h1>DIVISION BY ZERO IS NOT ALLOWED</h1>
<%=exception%>
</body>
</html>
```

OUTPUT:-

Case 1:- No exception Occured



Case 2:- Exception is Occurred and Handled

7. info

- It defines a string which can be accessed by getServletInfo() method.

- This attribute is used to set the servlet description.

Syntax of info:

<%@ page info="Hello Students" %>
Here, the value represents the servlet information.

8. **Session**

- JSP page creates session by default.
- Sometimes we don't need a session to be created in JSP, and hence, we can set this attribute to false in that case. The default value of the session attribute is true, and the session is created.

  When it is set to false, then we can indicate the compiler to not create the session by default.

Syntax of session:

<%@ page session="true/false"%>
Here in this case session attribute can be set to true or false

9. **Extends**: This attribute is used to extend (inherit) the class like JAVA does

Syntax of extends:

<%@ page extends="super.java" %>
Here the value represents class from which it has to be inherited.

Example:

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

<%@ page extends="demotest.DemoClass.java" %>
```

10. **isThreadSafe:**

- It defines the threading model for the generated servlet.
- It indicates the level of thread safety implemented in the page.
- Its default value is true so simultaneous
- We can use this attribute to implement SingleThreadModel interface in generated servlet.
- If we set it to false, then it will implement SingleThreadModel and can access any shared objects and can yield inconsistency.

Syntax of isThreadSafe:

<% @ page isThreadSafe="true/false" %>
Here true or false represents if synchronization is there then set as true and set it as false.

11. AutoFlush:

This attribute specifies that the buffered output should be flushed automatically or not and default value of that attribute is true.

If the value is set to false the buffer will not be flushed automatically and if its full, we will get an exception.

When the buffer is none then the false is illegitimate, and there is no buffering, so it will be flushed automatically.

Syntax of autoFlush:

<% @ page autoFlush="true/false" %>
Here true/false represents whether buffering has to be done or not

Example:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   autoFlush="false"%>
```

12. Buffer:

- Using this attribute the output response object may be buffered.
- We can define the size of buffering to be done using this attribute and default size is 8KB.
- It directs the servlet to write the buffer before writing to the response object.

Syntax of buffer:

<%@ page buffer="value" %>
Here the value represents the size of the buffer which has to be defined. If there is no buffer, then we can write as none, and if we don't mention any value then the default is 8KB

Example:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   buffer="16KB"%>
```

Explanation of the code:

In the above code, buffer size is mentioned as 16KB wherein the buffer would be of that size

13. isELIgnored:

- IsELIgnored is a flag attribute where we have to decide whether to ignore EL tags or not.
- Its datatype is java enum, and the default value is false hence EL is enabled by default.

Syntax of isELIgnored:

<%@ page isELIgnored="true/false" %>
Here, true/false represents the value of EL whether it should be ignored or not.

Example:

```
<%@ page language="java" contentType="text/html;" pageEncoding="ISO-8859-1"
    isELIgnored="true"%>
```

JSP Include directive

- JSP “include directive”(codeline 8) is used to include one file to the another file
- This included file can be HTML, JSP, text files, etc.
- It is also useful in creating templates with the user views and break the pages into header&footer and sidebar actions.
- It includes file during translation phase

Syntax of include directive:

<%@ include….%>
Example:

1. **index.jsp (Main file)**

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<%@ include file="header.html" %>
<h1>Welcome</h1>
</body>
</html>
```

2. **header.html**

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h6>Demonstration of Include Directive</h6>
```

</body>
</html>

OUTPUT:-