JSP

Introduction to JSP:

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 technology is used to create web application just like Servlet technology.

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 Tag etc

Java Server Pages (JSP) is a technology for developing Web pages that supports dynamic content. 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 Java Server 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 Web pages 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.

Advantage of JSP over Servlet

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

1) Extension to Servlet

JSP technology is the extension to servlet technology. We can use all the features of 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.

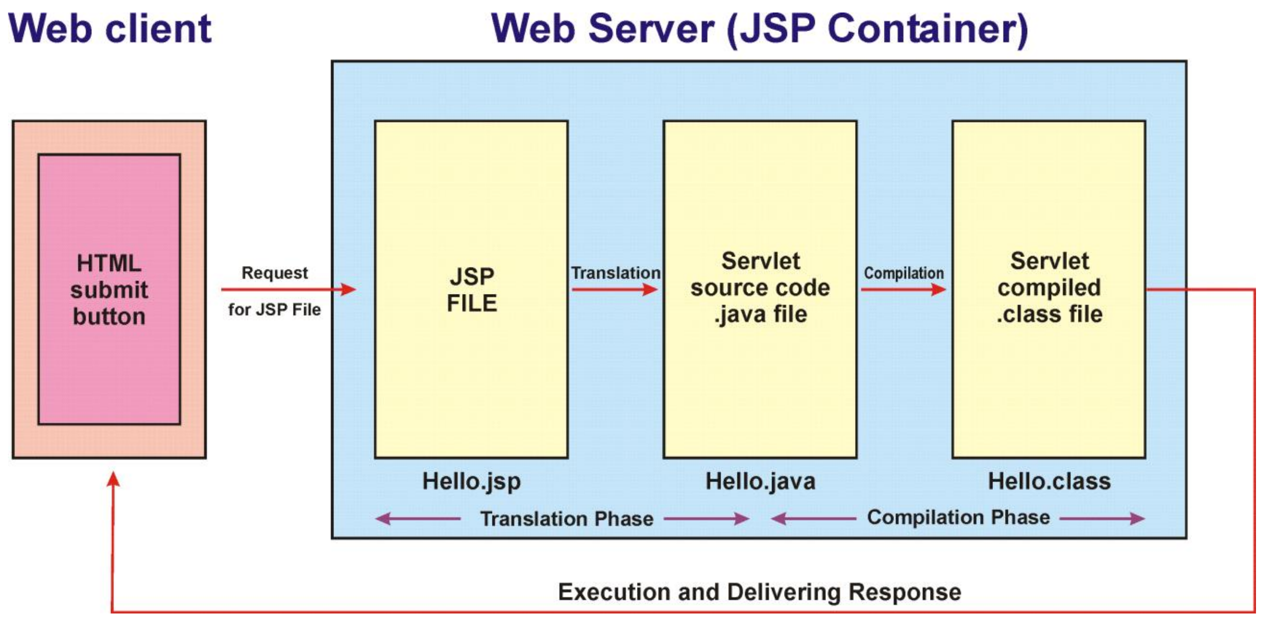
3) Fast Development: No need to recompile and redeploy

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 a lot of tags such as action tags, jstl, custom tags etc. that reduces the code. Moreover, we can use EL, implicit objects etc.

JSP Architecture:



Following steps includes the JSP Architecture as shown in the above figure.

1. Web client sends request to Web server for a **JSP page** (extension **.jsp**).
2. As per the request, the Web server (here after called as **JSP container**) loads the page.
3. JSP container converts (or translates) the JSP file into Servlet source code file (with extension **.java**). This is known as **translation**.
4. The translated **.java** file of Servlet is compiled that results to Servlet **.class** file. This is known as **compilation**.
5. The **.class** file of Servlet is executed and output of execution is sent to the client as **response**.

The JSP container converts the JSP file into a Servlet source file with extension **.java**. It is simply as equivalent you write a Servlet file. That is, instead of you write a Servlet file, container writes (as per the source code of JSP file). That means, a **JSP file is converted into a Servlet internally**. This is known as **translation** and **translation phase**.

The JSP container software compiles the Servlet source code (**.java** file) into a Servlet **.class** file. This is known as JSP compilation.

The **.class** file of Servlet is executed by the JSP container and the output of execution is sent to client as response. If the same JSP file is not used again, the **.class** file is deleted. If used again and often, the .class file is **retained** by the container for further usage without the need of second time translation and compilation.

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 save 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

1. <html>
2. <body>
3. <% out.print(2*5); %>
4. </body>
5. </html>

Steps

1.Open Eclipse, Click on **New → Dynamic Web Project**

2.Give a name to your project and click on OK

3.You will see a new project created in Project Explorer

4.To create a new JSP file right click on Web Content directory, **New → JSP file**

5.Give a name to your JSP file and click Finish.

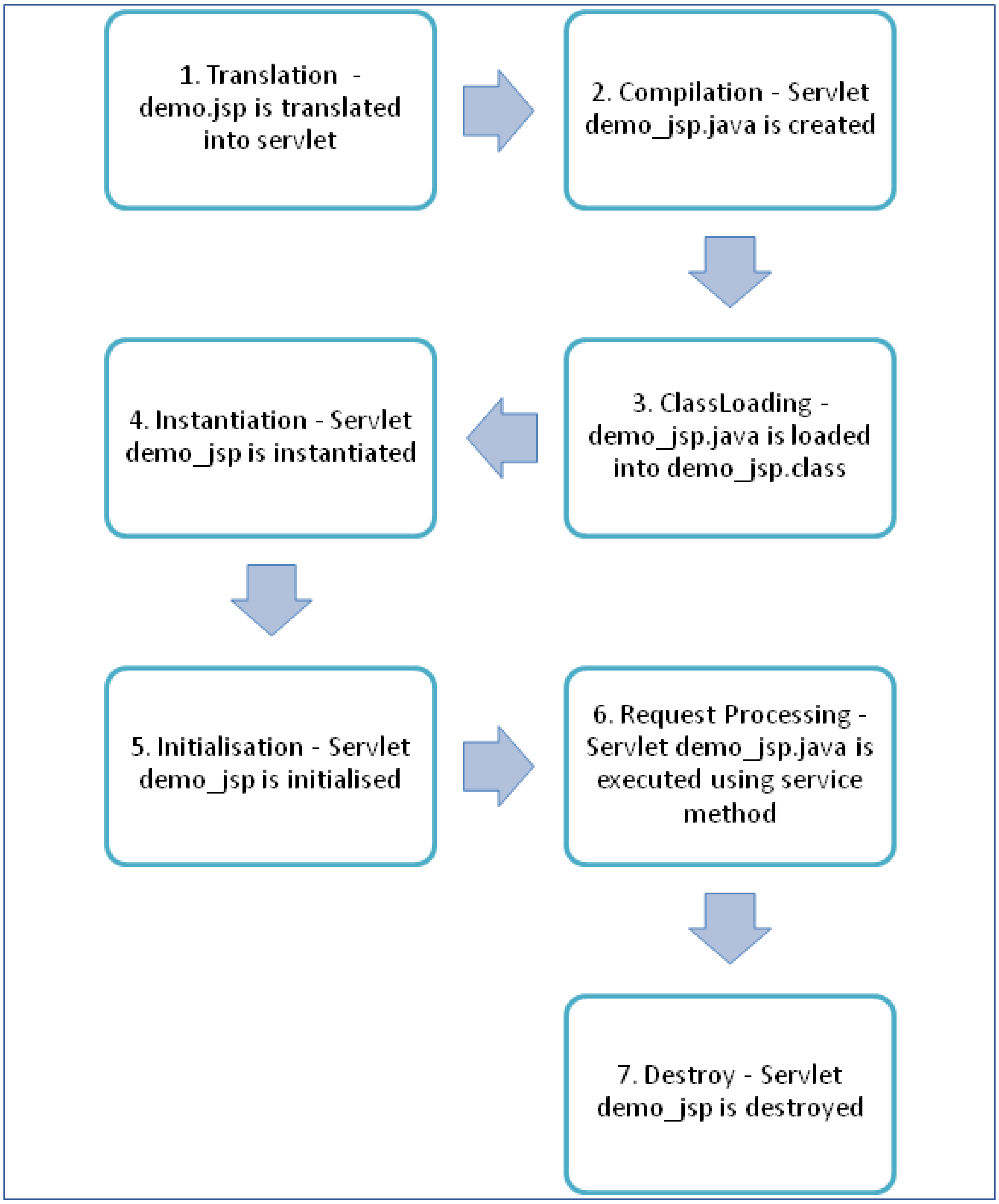
6.Write something in your JSP file. The complete HTML and the JSP code, goes inside the `<body>` tag, just like HTML pages.

7.To run your project, right click on **Project**, select **Run As → Run on Server**

8.To start the server, Choose existing server name and click on finish

JSP LifeCycle

JSP Life Cycle is defined as translation of JSP Page into servlet as a JSP Page needs to be converted into servlet first in order to process the service requests. The Life Cycle starts with the creation of JSP and ends with the disintegration of that.


1. Translation - demo.jsp is translated into servlet
2. Compilation - Servlet demo_jsp.java is created
3. ClassLoading - demo_jsp.java is loaded into demo_jsp.class
4. Instantiation - Servlet demo_jsp is instantiated
5. Initialisation - Servlet demo_jsp is initialised
6. Request Processing - Servlet demo_jsp.java is executed using service method
7. Destroy - Servlet demo_jsp is destroyed

Following steps explain the JSP life cycle:

1. Translation of JSP page
2. Compilation of JSP page(Compilation of JSP page into _jsp.class)
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)

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

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.

JSP Implicit Objects

There are **9 jsp implicit objects**. These objects are *created by the web container* that are available to all the jsp pages.

1) JSP out implicit object

For writing any data to the buffer, JSP provides an implicit object named out. It is the object of JspWriter. In case of servlet you need to write:

```
PrintWriter out=response.getWriter();
```

But in JSP, you don't need to write this code.

Example of out implicit object

In this example we are simply displaying date and time.

index.jsp

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

2.JSP request implicit object

The **JSP request** is an implicit object of type HttpServletRequest i.e. created for each jsp request by the web container. It can be used to get request information such as parameter, header information, remote address, server name, server port, content type, character encoding etc.

It can also be used to set, get and remove attributes from the jsp request scope.

Let's see the simple example of request implicit object where we are printing the name of the user with welcome message.

Example of JSP request implicit object

index.html

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

```
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
```

3) JSP response implicit object

In JSP, response is an implicit object of type HttpServletResponse. The instance of HttpServletResponse is created by the web container for each jsp request.

It can be used to add or manipulate response such as redirect response to another resource, send error etc.

Let's see the example of response implicit object where we are redirecting the response to the Google.

Example

index.html

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

```
<%
response.sendRedirect("http://www.google.com");
%>
```

4) JSP config implicit object

In JSP, config is an implicit object of type *ServletConfig*. This object can be used to get initialization parameter for a particular JSP page. The config object is created by the web container for each jsp page.

Generally, it is used to get initialization parameter from the web.xml file.

Example :

index.html

```
<form action="welcome">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

web.xml file

```
<web-app>
<servlet>
<servlet-name>sonoojaiswal</servlet-name>
<jsp-file>/welcome.jsp</jsp-file>
<init-param>
<param-name>dname</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</init-param>
</servlet>
<servlet-mapping>
<servlet-name>sonoojaiswal</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>
</web-app>
```

welcome.jsp

```
<%
out.print("Welcome "+request.getParameter("uname"));
String driver=config.getInitParameter("dname");
out.print("driver name is="+driver);
%>
```

5) JSP application implicit object

In JSP, application is an implicit object of type *ServletContext*.

The instance of ServletContext is created only once by the web container when application or project is deployed on the server.

This object can be used to get initialization parameter from configuaration file (web.xml). It can also be used to get, set or remove attribute from the application scope.

This initialization parameter can be used by all jsp pages.

Example

index.html

```
<form action="welcome">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

web.xml file

```
<web-app>
<servlet>
<servlet-name>sonoojaiswal</servlet-name>
<jsp-file>/welcome.jsp</jsp-file>
</servlet>
<servlet-mapping>
<servlet-name>sonoojaiswal</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>
<context-param>
<param-name>dname</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</context-param>
</web-app>
```

welcome.jsp

```
<%
out.print("Welcome "+request.getParameter("uname"));
String driver=application.getInitParameter("dname");
out.print("driver name is="+driver);
%>
```

6) session implicit object

In JSP, session is an implicit object of type HttpSession.The Java developer can use this object to set,get or remove attribute or to get session information.

Example of session implicit object

index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

welcome.jsp

```
<html>
<body>
<%
String name=request.getParameter("uname");
out.print("Welcome "+name);
session.setAttribute("user",name);
<a href="second.jsp">second jsp page</a>
%>
</body>
</html>
```

second.jsp

```
<html>
<body>
<%
String name=(String)session.getAttribute("user");
out.print("Hello "+name);
%>
</body>
</html>
```

pageContext implicit object

In JSP, pageContext is an implicit object of type PageContext class.The pageContext object can be used to set,get or remove attribute from one of the following scopes:

- page
- request
- session
- application

In JSP, page scope is the default scope.

Example

index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

welcome.jsp

```
<html>
<body>
<%
String name=request.getParameter("uname");
out.print("Welcome "+name);
pageContext.setAttribute("user",name,PageContext.SESSION_SCOPE);
<a href="second.jsp">second jsp page</a>
%>
</body>
</html>
```

second.jsp

```
<html>
<body>
<%
String name=(String)pageContext.getAttribute("user",PageContext.SESSION_SCOPE);
```

```
out.print("Hello "+name);
%>
</body>
</html>
```

8) page implicit object:

In JSP, page is an implicit object of type Object class.This object is assigned to the reference of auto generated servlet class. It is written as: Object page=this;

For using this object it must be cast to Servlet type.

For example:<% (HttpServlet)page.log("message"); %>

Since, it is of type Object it is less used because you can use this object directly in jsp.

For example:<% this.log("message"); %>

9) exception implicit object

In JSP, exception is an implicit object of type java.lang.Throwable class. This object can be used to print the exception. But it can only be used in error pages.It is better to learn it after page directive.

Example

error.jsp

```
<%@ page isErrorPage="true" %>
<html>
<body>
Sorry following exception occured:<%= exception %>
</body>
</html>
```

JSP Scripting elements

The scripting elements provides the ability to insert java code inside the jsp. There are three types of scripting elements:

- scriptlet tag
- expression tag
- declaration tag

1.Scriptlet tag

A scriptlet tag is used to execute java source code in JSP. Syntax : <% java source code %>

Example

```
<html>
<body>
<% out.print("welcome to jsp"); %>
</body>
</html>
```

2.Expression tag

The code placed within JSP expression tag is *written to the output stream of the response*. So you need not write out.print() to write data. It is mainly used to print the values of variable or method. Expression tag statement never ends with semicolon.

Syntax :<%= statement %>

Example

```
<html>
<body>
<%= "welcome to jsp" %>
</body>
</html>
```

3.Declaration Tag

The **JSP declaration tag** is used *to declare fields and methods*.

The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet. So it doesn't get memory at each request.

syntax :<%! field or method declaration %>

```
<html>
<body>
<%! int data=50; %>
<%= "Value of the variable is:"+data %>
</body>
</html>
```

JSP directives

The **jsp directives** are messages that tells the web container how to translate a JSP page into the corresponding servlet.

There are three types of directives:

- page directive
- include directive
- taglib directive

Syntax of JSP Directive:<%@ directive attribute="value" %>

1.Page directive

The page directive defines attributes that apply to an entire JSP page.

Syntax :<%@ page attribute="value" %>

Attributes of JSP page directive

- import
- contentType
- extends
- info
- buffer
- language
- isELIgnored
- isThreadSafe
- autoFlush
- session
- pageEncoding
- errorPage
- isErrorPage

1)import

The import attribute is used to import class,interface or all the members of a package.It is similar to import keyword in java class or interface.

```
<html>
<body>
<%@ page import="java.util.Date" %>
Today is: <%= new Date() %>
</body>
</html>
```

2)contentType

The contentType attribute defines the MIME(Multipurpose Internet Mail Extension) type of the HTTP response.The default value is "text/html;charset=ISO-8859-1".

```
<html>
<body>
<%@ page contentType=application/msword %>
Today is: <%= new java.util.Date() %>
</body>
</html>
```

3)extends

The extends attribute defines the parent class that will be inherited by the generated servlet. It is rarely used.

4)info

This attribute simply sets the information of the JSP page which is retrieved later by using getServletInfo() method of Servlet interface.

```
<html>
<body>
<%@ page info="composed by ABC" %>
Today is: <%= new java.util.Date() %>
</body>
</html>
```

The web container will create a method getServletInfo() in the resulting servlet.For example:

```
public String getServletInfo() {
  return "composed by ABC";
}
```

5)buffer

The buffer attribute sets the buffer size in kilobytes to handle output generated by the JSP page.The default size of the buffer is 8Kb.

```
<html>
<body>
<%@ page buffer="16kb" %>
Today is: <%= new java.util.Date() %>
</body>
</html>
```

6)language

The language attribute specifies the scripting language used in the JSP page. The default value is "java".

7)isELIgnored

We can ignore the Expression Language (EL) in jsp by the isELIgnored attribute. By default its value is false i.e. Expression Language is enabled by default. We see Expression Language later.

```
<%@ page isELIgnored="true" %>//Now EL will be ignored
```

8)isThreadSafe

Servlet and JSP both are multithreaded.If you want to control this behaviour of JSP page, you can use isThreadSafe attribute of page directive.The value of isThreadSafe value is true.If you make it false, the web container will serialize the multiple requests, i.e. it will wait until the JSP finishes responding to a request before passing another request to it.If you make the value of isThreadSafe attribute like:

<%@ page isThreadSafe="false" %>

The web container in such a case, will generate the servlet as:

```
public class SimplePage_jsp extends HttpJspBase
  implements SingleThreadModel{
.......
}
```

9)errorPage

The errorPage attribute is used to define the error page, if exception occurs in the current page, it will be redirected to the error page.

```
<html>
<body>
<%@ page errorPage="myerrorpage.jsp" %>
 <%= 100/0 %>
</body>
</html>
```

10)isErrorPage

The isErrorPage attribute is used to declare that the current page is the error page. The exception object can only be used in the error page.

```
<html>
<body>
<%@ page isErrorPage="true" %>
 Sorry an exception occured!<br/>
The exception is: <%= exception %>
</body>
</html>
```

2. Include Directive

The include directive is used to include the contents of any resource it may be jsp file, html file or text file. The include directive includes the original content of the included resource at page translation time (the jsp page is translated only once so it will be better to include static resource).

Advantage of Include directive

Code Reusability

Syntax of include directive:<%@ include file="resourceName" %>

In this example, we are including the content of the header.html file. To run this example you must create an header.html file.

```
<html>
<body>
<%@ include file="header.html" %>
Today is: <%= java.util.Calendar.getInstance().getTime() %>
</body>
</html>
```

Note: The include directive includes the original content, so the actual page size grows at runtime.

3.Taglib directive

The JSP taglib directive is used to define a tag library that defines many tags. We use the TLD (Tag Library Descriptor) file to define the tags. In the custom tag section we will use this tag so it will be better to learn it in custom tag.

Syntax :<%@ taglib uri="uriofthetaglibrary" prefix="prefixoftaglibrary" %>

In this example, we are using our tag named currentDate. To use this tag we must specify the taglib directive so the container may get information about the tag.

```
<html>
<body>
<%@ taglib uri="http://www.javatpoint.com/tags" prefix="mytag" %>
<mytag:currentDate/>
</body>
</html>
```

JSTL (JSP Standard Tag Library)

The JSP Standard Tag Library (JSTL) represents a set of tags to simplify the JSP development.

Advantage of JSTL

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

JSTL Core Tags

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

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

JSTL Core Tags List

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

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

JSTL Core Tags

1<c:out> Tag

The <c:out> tag is similar to JSP expression tag, but it can only be used with expression. It will display the result of an expression, similar to the way < %=...% > work.

The < c:out > tag automatically escape the XML tags. Hence they aren't evaluated as actual tags.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Tag Example</title>
</head>
<body>
<c:out value="${'Welcome to javaTpoint'}"/>
</body>
</html>
```

2<c:import> Tag

The <c:import> is similar to jsp 'include', with an additional feature of including the content of any resource either within server or outside the server.

This tag provides all the functionality of the <include > action and it also allows the inclusion of absolute URLs.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Tag Example</title>
</head>
<body>
<c:import var="data" url="http://www.javatpoint.com"/>
<c:out value="${data}"/>
</body>
</html>
```

3.<c:set> Tag

It is used to set the result of an expression evaluated in a 'scope'. The <c:set> tag is helpful because it evaluates the expression and use the result to set a value of java.util.Map or JavaBean. This tag is similar to jsp:setProperty action tag.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
<c:set var="Income" scope="session" value="${4000*4}"/>
<c:out value="${Income}"/>
</body>
</html>
```

4. <c:remove> Tag

It is used for removing the specified variable from a particular scope. This action is not particularly helpful, but it can be used for ensuring that a JSP can also clean up any scope resources.The <c:remove > tag removes the variable from either a first scope or a specified scope.

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

```
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
<c:set var="income" scope="session" value="${4000*4}"/>
<p>Before Remove Value is: <c:out value="${income}"/></p>
<c:remove var="income"/>
<p>After Remove Value is: <c:out value="${income}"/></p>
</body>
</html>
```

5.<c:catch> Tag

It is used for Catches any Throwable exceptions that occurs in the body and optionally exposes it. In general it is used for error handling and to deal more easily with the problem occur in program.The < c:catch > tag catches any exceptions that occurs in a program body.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
<c:catch var ="catchtheException">
   <% int x = 2/0;%>
</c:catch>
<c:if test = "${catchtheException != null}">
   <p>The type of exception is : ${catchtheException} <br />
   There is an exception: ${catchtheException.message}</p>
</c:if>
</body>
</html>
```

JSTL Conditional Tags

1.<c:if> tag

The < c:if > tag is used for testing the condition and it display the body content, if the expression evaluated is true.It is a simple conditional tag which is used for evaluating the body content, if the supplied condition is true.

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

```
<head>
<title>Core Tag Example</title>
</head>
<body>
<c:set var="income" scope="session" value="${4000*4}"/>
<c:if test="${income > 8000}">
  <p>My income is: <c:out value="${income}"/><p>
</c:if>
</body>
</html>
```

2 <c:choose>, <c:when>, <c:otherwise> Tag

The < c:choose > tag is a conditional tag that establish a context for mutually exclusive conditional operations. It works like a Java **switch** statement in which we choose between a numbers of alternatives.

The <c:when > is subtag of <choose > that will include its body if the condition evaluated be 'true'.

The < c:otherwise > is also subtag of < choose > it follows &l;twhen > tags and runs only if all the prior condition evaluated is 'false'.

The c:when and c:otherwise works like if-else statement. But it must be placed inside c:choose tag.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
<c:set var="income" scope="session" value="${4000*4}"/>
<p>Your income is : <c:out value="${income}"/></p>
<c:choose>
   <c:when test="${income <= 1000}">
      Income is not good.
   </c:when>
   <c:when test="${income > 10000}">
       Income is very good.
   </c:when>
   <c:otherwise>
      Income is undetermined...
   </c:otherwise>
</c:choose>
</body>  </html>
```

JSTL Looping <c:forEach> Tag

The <c:for each > is an iteration tag used for repeating the nested body content for fixed number of times or over the collection.These tag used as a good alternative for embedding a Java **while, do-while, or for** loop via a scriptlet. The < c:for each > tag is most commonly used tag because it iterates over a collection of object.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
<c:forEach var="j" begin="1" end="3">
   Item <c:out value="${j}"/><p>
</c:forEach>
</body>
</html>
```