CHAPTER 4
JSP

1. INTRODUCTION

1.1. Introduction:

JSP (JavaServer Pages) is server side technology to create dynamic java web application. JSP can be thought as an extension to servlet technology because it provides features to easily create user views. JSP Page consists of HTML code and provide option to include java code for dynamic content. Since web applications contain a lot of user screens, JSPs are used a lot in web applications. To bridge the gap between java code and HTML in JSP, it provides additional features such as JSP Tags, Expression Language, Custom tags. This makes it easy to understand and helps a web developer to quickly develop JSP pages.

1.2. Advantages of JSP over Servlet:

- We can generate HTML response from servlets also but the process is cumbersome and error prone, when it comes to writing a complex HTML response, writing in a servlet will be a nightmare. JSP helps in this situation and provide us flexibility to write normal HTML page and include our java code only where it's required.
- JSP provides additional features such as tag libraries, expression language, custom tags that helps in faster development of user views.
- JSP pages are easy to deploy, we just need to replace the modified page in the server and container takes care of the deployment. For servlets, we need to recompile and deploy whole project again.

The Servlet and JSPs compliment each other. We should use Servlet as server side controller and to communicate with model classes whereas JSPs should be used for presentation layer.

1.3. JSP LifeCycle:

JSP's life cycle can be grouped into following phases.

1. JSP Page Translation:

A java servlet file is generated from the JSP source file. This is the first step in its tedious multiple phase life cycle. In the translation phase, the container validates the syntactic correctness of the JSP pages and tag files. The container interprets the standard directives and actions, and the custom actions referencing tag libraries used in the page.

2. JSP Page Compilation:

The generated java servlet file is compiled into a java servlet class.

Note: The translation of a JSP source page into its implementation class can happen at any time between initial deployment of the JSP page into the JSP container and the receipt and processing of a client request for the target JSP page.

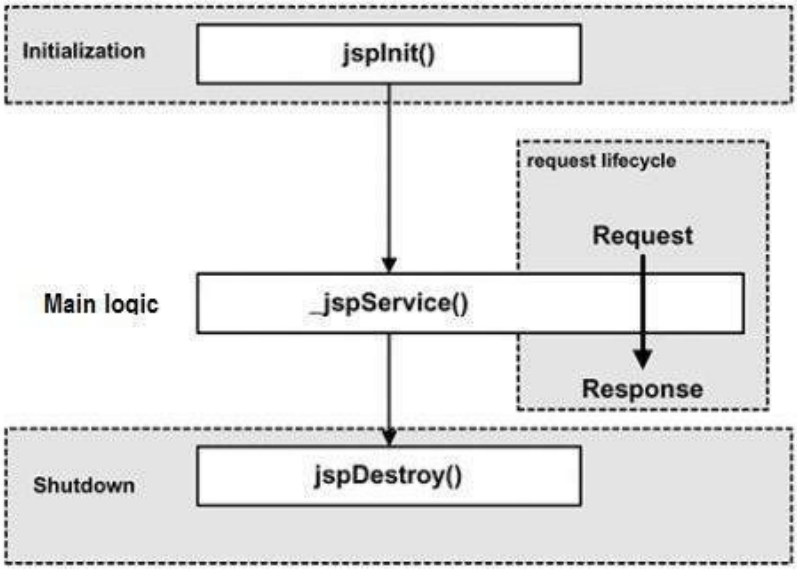


Figure:- JSP LifeCycle

3. Class Loading:
The java servlet class that was compiled from the JSP source is loaded into the container.

4. Execution phase:
In the execution phase the container manages one or more instances of this class in response to requests and other events. The interface JspPage contains jspInit() and jspDestroy(). The JSP specification has provided a special interface HttpJspPage for JSP pages serving HTTP requests and this interface contains _jspService().

5. **Initialization:**
jspInit() method is called immediately after the instance was created. It is called only once during JSP life cycle.

6. _jspService() execution:
This method is called for every request of this JSP during its life cycle. This is where it serves the purpose of creation. Oops! it has to pass through all the above steps to reach this phase. It passes the request and the response objects. _jspService() cannot be overridden.

7. jspDestroy() execution:

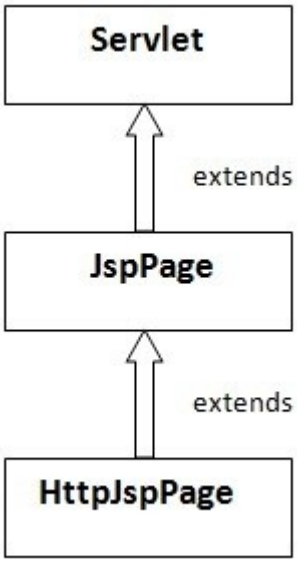
This method is called when this JSP is destroyed. With this call the servlet serves its purpose and submits itself to heaven (garbage collection). This is the end of jsp life cycle. jspInit(), _jspService() and jspDestroy() are called the life cycle methods of the JSP.

1.4. JSP versions:

Sr.No.	Jsp version	Java Plateform	Released Year
1	1.1	J2EE 1.2	Dec 12, 1999
2	1.2	J2EE 1.3	Sept 24, 2001
3	2.0	J2EE 1.4	Nov 11, 2003
4	2.1	Java EE 5	May 11, 2006
5	2.2	Java EE 6	Dec 10, 2009
6	2.3	Java EE 7	June 12, 2013

1.5. JSP API Interfaces and Classes:-

All the core JSP interfaces and classes are defined in javax.servlet.jsp package. Expression Language API interfaces are classes are part of javax.servlet.jsp.el package. JSP Tag Libraries interfaces and classes are defined in javax.servlet.jsp.tagext package.



1. JspPage Interface:

JspPage interface extends Servlet interface and declares jspInit() and jspDestroy() life cycle methods of the JSP pages.

2. HttpJspPage Interface:

HttpJspPage interface describes the interaction that a JSP Page Implementation Class must satisfy when using the HTTP protocol. This interface declares the service method of JSP page for HTTP

protocol as public void_jspService(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException.

3. JspWriter abstract Class:
Similar to PrintWriter in servlets with additional facility of buffering support. This is one of the implicit variables in a JSP page with name "out". This class extends java.io.Writer and container provide their own implementation for this abstract class and use it while translating JSP page to Servlet. We can get it's object using PageContext.getOut() method.
Apache Tomacat concrete class for JspWriter is org.apche.jasper.runtime.JspWriterImpl.

4. JspContext abstract Class:
JspContext serves as the base class for the PageContext class and abstracts all information that is not specific to servlets. The JspContext provides mechanism to obtain the JspWriter for output, mechanism to work with attributes and API to manage the various scoped namespaces.

5. **PageContext abstract Class:**
PageContext extends JspContext to provide useful context information when JSP is used for web applications. A PageContext instance provides access to all the namespaces associated with a JSP page, provides access to several page attributes, as well as a layer above the implementation details. Implicit objects are added to the pageContext automatically.

6. JspFactory abstract Class:
The JspFactory is an abstract class that defines a number of factory methods available to a JSP page at runtime for the purposes of creating instances of various interfaces and classes used to support the JSP implementation.

7. JspEngineInfo abstract Class:
The JspEngineInfo is an abstract class that provides information on the current JSP engine.

8. ErrorData final Class:
Contains information about an error, for error pages.

9. JspException Class:
A generic exception known to the JSP container, similar to ServletException.If JSP pages throw JspException then errorpage mechanism is used to present error information to user.

10. JspTagException Class:
Exception to be used by a Tag Handler to indicate some unrecoverable error.

11. SkipPageException Class:
Exception to indicate the calling page must cease evaluation. Thrown by a simple tag handler to

indicate that the remainder of the page must not be evaluated. This exception should not be thrown manually in a JSP page.

1.6. JSP Scripting Elements:

We can insert the java code within jsp using following tags

1. JSP Declaration tag:

JSP Declarations are used to declare member methods and variables of servlet class. JSP Declarations starts with `<%!` and ends with `%>`.

e.g. <%! int a=10, b=20,result;%>

2. JSP Scriptlet tag:

Scriptlet tags are the easiest way to put java code in a JSP page. A scriptlet tag starts with `<%` and ends with `%>`.

e.g. <% result=a+b;%>

Any code written inside the scriptlet tags go into the `_jspService()` method.

3. JSP Expression tag:

Since most of the times we print dynamic data in JSP page using *out.print()* method, there is a shortcut to do this through JSP Expressions. JSP Expression starts with <%= and ends with %>.

e.g. if we use scriptlet tag then we have to write <% out.print(result); %>

It can be written using JSP Expression as <%=result %>

Notice that anything between <%= %> is sent as parameter to out.print() method. Also notice that scriptlets can contain multiple java statements and always ends with semicolon (;) but expression doesn't end with semicolon.

1.7. JSP Comments:

Since JSP is built on top of HTML, we can write comments in JSP file like html comments.

HTML comment : <-- This is HTML Comment -->

JSP comment : <%-- This is JSP Comment--%> . This comment is suitable for developers to provide code level comments because these are not sent in the client response.

1.8. JSP Directives:

JSP Directives are used to give special instructions to the container while JSP page is getting translated to servlet source code:

JSP directives starts with <%@ and ends with %>

For example, in above JSP Example, I am using *page* directive to to instruct container JSP translator to import the Date class.

2. JSP Implicit Objects

2.1. out Object:

JSP out implicit object is instance of javax.servlet.jsp.JspWriter implementation and it's used to output content to be sent in client response. This is one of the most used JSP implicit object and thats why we have JSP Expression to easily invoke out.print() method.
In case of Servlets we need to write PrintWriter out=response.getWriter().
But in JSP, you do not need to write this code. Because in jsp for print any result on browser we use expression tag and this tag are auto converted to out.print() statement and insert the out.print() into _jspService() of the servlet class by the container.

Example of out implicit object:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!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=UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
out.print("Welcome");
%>
</body>
</html>
```

2.2. request Object:

JSP request implicit object is instance of javax.servlet.http.HttpServletRequest implementation and it's one of the argument of JSP service method. We can use request object to get the request parameters, cookies, request attributes, session, header information and other details about client request.

Example of JSP request implicit object:
Jsp file 1: Input.jsp

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

```
<!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=UTF-8">
<title>Insert title here</title>
</head>
<body>
<form action="Output.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

Jsp file 2: Output.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!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=UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
</body>
</html>
```

2.3. response Object:

JSP response implicit object is instance of javax.servlet.http.HttpServletResponse implementation and comes as argument of service method. We can response object to set content type, character encoding, header information in response, adding cookies to response and redirecting the request to other resource.

Example of response implicit object:

Jsp file: Redirect.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!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=UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
response.sendRedirect("http://www.google.com");
%>
</body>
</html>
```

2.4. config Object:

JSP config implicit object is instance of javax.servlet.ServletConfig implementation and used to get the JSP init params configured in deployment descriptor. config object is an implicit object of type ServletConfig and it is created by the container, whenever servlet object is created. This object can be used to get initialization parameter for a particular JSP page. config object is created by the web container for every jsp page. It means if a web application has three jsp pages then three config object are created.

In jsp, it is not mandatory to configure in web.xml. If we configure a jsp in web.xml then the logical name and init parameter given in web.xml file are stored into config object by the container. config object is accessible, only if the request is given to the jsp by using its url pattern, but not with name of the jsp. config object is accessible, only if the request is given to the jsp by using its url pattern, but not with name of the jsp.

Example of config implicit object:

Jsp file:InitParameter.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!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=UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
String name=config.getInitParameter("name");
out.print("Institute name is = "+name);
%>
</body>
```

```
</html>
```

web.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" version="3.0">
  <display-name>jsp_practice</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
  <servlet>
<servlet-name>Home</servlet-name>
<jsp-file>/config.jsp</jsp-file>
  <init-param>
<param-name>name</param-name>
<param-value>Coder technologies</param-value>
</init-param>
</servlet>
<servlet-mapping>
<servlet-name>Home</servlet-name>
<url-pattern>/config.jsp</url-pattern>
</servlet-mapping>
  </web-app>
```

2.5. application Object:

JSP application implicit object is instance of javax.servlet.ServletContext implementation and it's used to get the context information and attributes in JSP. We can use it to get the RequestDispatcher object in JSP to forward the request to another resource or to include the response from another resource in the JSP.

Example of Application implicit object:

Jsp file 1: Context.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!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=UTF-8">
<title>Insert title here</title>
</head>
<body>
<form action="Application.jsp">
<input type="submit" value="click to check context value"><br/>
</form>
</body>
</html>
```

Jsp file 2: Application.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!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=UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
String info=application.getInitParameter("info");
out.print("Program info = "+info); %>
</body>
</html>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" version="3.0">
  <display-name>jsp_practice</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>

<servlet>
<servlet-name>One</servlet-name>
```

```
<jsp-file>/Application.jsp</jsp-file>
</servlet>

<servlet-mapping>
<servlet-name>One</servlet-name>
<url-pattern>/Application.jsp</url-pattern>
</servlet-mapping>

<context-param>
<param-name>info</param-name>
<param-value>Example of application object</param-value>
</context-param>

</web-app>
```

2.6. session Object:

JSP session implicit object is instance ofjavax.servlet.http.HttpSession implementation. Whenever we request a JSP page, container automatically creates a session for the JSP in the service method. Since session management is heavy process, so if we don't want session to be created for JSP, we can use page directive to not create the session for JSP using <%@ page session="false" %>. This is very helpful when our login page or the index page is a JSP page and we don't need any user session there

Example of Session implicit object:

Input.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!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=UTF-8">
<title>Insert title here</title>
</head>
<body>
<form action="session1.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

session1.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!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=UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
String name=request.getParameter("uname");
out.print("Welcome "+name);
session.setAttribute("user",name);
 %>
 <a href="session2.jsp">second jsp page</a>
</body>
</html>
```

session2.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!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=UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
String name=(String)session.getAttribute("user");
out.print("Hello "+name);
%>
</body>
</html>
```

2.7. pageContext Object:

JSP pageContext implicit object is instance of javax.servlet.jsp.PageContext abstract class implementation. We can use pageContext to get and set attributes with different scopes and to forward request to other resources. pageContext object also hold reference to other implicit object. The pageContext object can be used to set,get or remove attribute from one of the following scopes.

- page
- request

- session
- application

Example of pageContext implicit object

Input.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!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=UTF-8">
<title>Insert title here</title>
</head>
<body>
<%

String name="Coder technologies";
pageContext.setAttribute("user",name,PageContext.SESSION_SCOPE);
pageContext.setAttribute("user",name,PageContext.PAGE_SCOPE);
pageContext.setAttribute("user",name,PageContext.REQUEST_SCOPE);
pageContext.setAttribute("user",name,PageContext.APPLICATION_SCOPE);

String sname=(String)pageContext.getAttribute("user",PageContext.SESSION_SCOPE);
String pname=(String)pageContext.getAttribute("user",PageContext.PAGE_SCOPE);
String rname=(String)pageContext.getAttribute("user",PageContext.REQUEST_SCOPE);
String aname=(String)pageContext.getAttribute("user",PageContext.APPLICATION_SCOPE);

%>
<b>Value in Session scope   <%=sname%></b><br>
<b>Value in Request scope   <%=rname%></b><br>
<b>Value in Page scope   <%=pname%></b><br>
<b>Value in Application scope   <%=aname%></b><br>

<a href="pageContext.jsp">One jsp page</a>

</body>
</html>
```

pageContext.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!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=UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
String sname=(String)pageContext.getAttribute("user",PageContext.SESSION_SCOPE);
String pname=(String)pageContext.getAttribute("user",PageContext.PAGE_SCOPE);
String rname=(String)pageContext.getAttribute("user",PageContext.REQUEST_SCOPE);
String aname=(String)pageContext.getAttribute("user",PageContext.APPLICATION_SCOPE);
%>
<b>Value in Session scope  <%=sname%></b><br>
<b>Value in Request scope   <%=rname%></b><br>
<b>Value in Page scope   <%=pname%></b><br>
<b>Value in Application scope   <%=aname%></b><br>
</body>
</html>
```

2.8. page Object:

JSP page implicit object is instance of java.lang.Object class and represents the current JSP page. page object provide reference to the generated servlet class. This object is very rarely used.
When a jsp is translated to an internal Servlet, we can find the following statement in the service() method of servlet. 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"); %>

2.9. exception Object:

JSP exception implicit object is instance of java.lang.Throwable class and used to provide exception details in JSP error pages. We can't use this object in normal JSP pages and it's available only in JSP error pages.

Example:
This program is to show use of the <%=exception> object. In this 3 files are created. One file is used to take input from the user for the calculation which is **"calculation.jsp".** And another file is used to perform the operation which is **"op.jsp".** Third file is Exception file which is

“exception.jsp”. For this 2 changes are required in jsp page directive.

- In every file at page directive isErrorPage=”true”.
- And set errorPage=”exception.jsp”.

Input.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" isErrorPage="true"
  errorPage="exception.jsp"  pageEncoding="UTF-8"%>
<!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=UTF-8">
<title>Insert title here</title>
</head>
<body>
<form action="operation.jsp">
<table>
<tr><td>Enter first Number:</td><td><input type="text" name="firsrnum"></td></tr>
<tr><td>Enter second Number:</td><td><input type="text" name="secondnum"></td></tr>
<tr><td><input type="submit" value="Divide"></td></tr>
</table><br><br>
Result:    <input type="text" value="<%=session.getAttribute("result")%>">
</form>
</body>
</html>
```

operation.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" isErrorPage="true"
errorPage="exception.jsp" pageEncoding="UTF-8"%>
<!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=UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
String num1=request.getParameter("firsrnum");
String num2=request.getParameter("secondnum");
int v1= Integer.parseInt(num1);
int v2= Integer.parseInt(num2);
```

```
int res= v1/v2;
session.setAttribute("result", res);
response.sendRedirect("calculation.jsp");
%>
</body>
</html>
```

exception.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" isErrorPage="true"
    pageEncoding="UTF-8"%>
<!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=UTF-8">
<title>Insert title here</title>
</head>
<body>
Got this Exception: <%= exception %> <br><br>
<b>Please enter correct value</b>
</body></html>
```

3. JSP DIRECTIVE

3.1. Introduction:

JSP Directives are used to give special instruction to container for translation of JSP to Servlet code. JSP Directives are placed between <%@ %>.

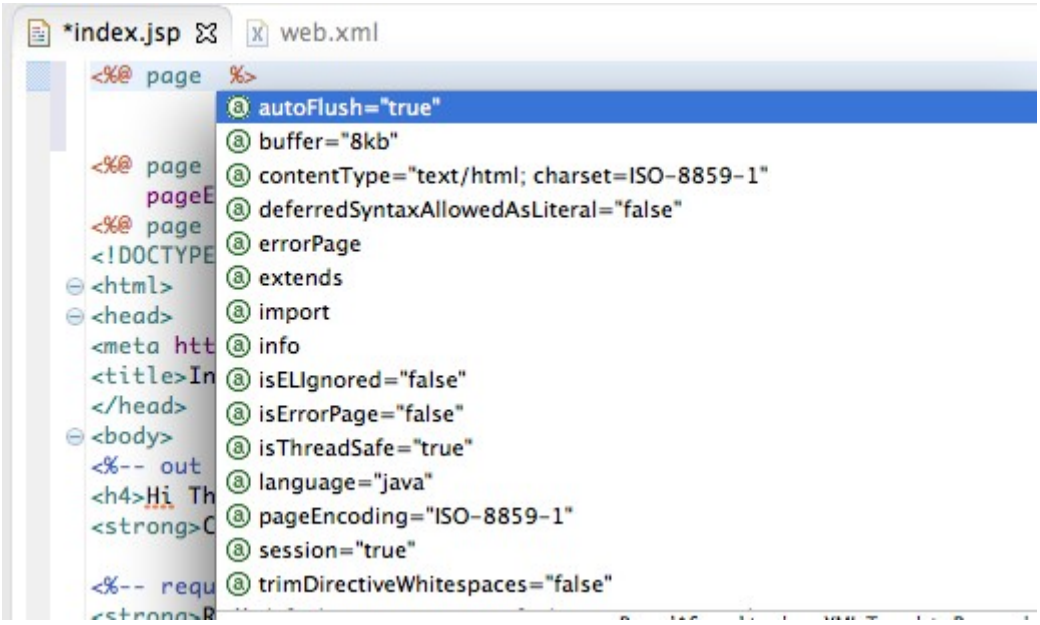
JSP provides three directives for us to use;

- page directive
- include directive
- taglib directive

Every directive has a set of attributes to provide specific type of instructions. So usually a JSP directive will look like <%@ directive attribute="value" %>.

3.2. JSP page directive:

page directive provide attributes that gets applied to the entire JSP page. page directive has a lot of attributes that we will look now. We can define multiple attributes in a single page directive or we can have multiple page directives in a single JSP page.



1. import attribute:

This is one of the most used page directive attribute. It's used to instruct container to import other java classes, interfaces, enums etc. while generating servlet code. This is similar to import

statements in java classes, interfaces.
An example of import page directive usage is:

```
<%@ page import="java.util.Date,java.util.List,java.io.*" %>
```

2. contentType attribute:
This attribute is used to set the content type and character set of the response. The default value of contentType attribute is”text/html; charset=ISO-8859-1″.

We can use it like below.

```
<%@ page contentType="text/html; charset=US-ASCII" %>
```

3. pageEncoding attribute: We can set response encoding type with this page directive attribute, its default value is “ISO-8859-1”.

```
<%@ page pageEncoding="US-ASCII" %>
```

4. extends attribute: This attribute is used to define the super class of the generated servlet code. This is very rarely used and we can use it if we have extended HttpServlet and overridden some of it’s implementations.

For example:

```
<%@ page extends="org.apache.jasper.runtime.HttpJspBase" %>
```

5. info attribute: We can use this attribute to set the servlet description and we can retrieve it using Servlet interface getServletInfo() method.

For example:

```
<%@ page info="Home Page JSP" %>
```

6. buffer attribute: We know that JspWriter has buffering capabilities, we can use this attribute to set the buffer size in KB to handle output generated by JSP page. Default value of buffer attribute is 8kb. We can define 16 KB buffer size as;

```
<%@ page buffer="16kb" %>
```

7. language attribute: language attribute is added to specify the scripting language used in JSP page. It’s default value is “java” and this is the only value it can have. May be in future, JSPs provide support to include other scripting languages like C++ or PHP too.

```
<%@ page language="java" %>
```

8. isELIgnored attribute: We can ignore the Expression Language (EL) in JSP using this page directive attribute. Its datatype is **Java Enum** and default value is *false*, so EL is enabled by default.

We can instruct container to ignore EL using below directive;

```
<%@ page isELIgnored="true" %>
```

9. isThreadSafe attribute: We can use this attribute to implement SingleThreadModel interface in generated servlet. Its an Enum with default value as true. If we set it's value to false, the generated servlet will implement SingleThreadModel and eventually we will loose all the benefits of servlet **multi-threading** features. You should never set it's value to false.

```
<%@ page isThreadSafe="false" %>
```

10. errorPage attribute: This attribute is used to set the error page for the JSP, if the JSP throws exception, the request is redirected to the error handler defined in this attribute. It's datatype is URI.

For example:

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

11. isErrorPage attribute: This attribute is used to declare that current JSP page is an error page. It's of type Enum and default value is false. If we are creating an error handler JSP page for our application, we have to use this attribute to let container know that it's an error page. JSP implicit attribute exception is available only to the error page JSPs.

For example:

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

12. autoFlush attribute: autoFlush attribute is to control the buffer output. Its default value is true and output is flushed automatically when buffer is full. If we set it to false, the buffer will not be flushed automatically and if it's full, we will get exception for buffer overflow. We can use this attribute when we want to make sure that JSP response is sent in full or none.

For example:

```
<%@ page autoFlush="false" %>
```

13. session attribute: By default JSP page creates a session but sometimes we don't need session in JSP page. We can use this attribute to indicate compiler to not create session by default. It's default value is true and session is created. To disable the session creation, we can use it like below.

```
<%@ page session ="false" %>
```

14. trimDirectiveWhitespaces attribute: This attribute was added in JSP 2.1 and used to strip out extra white spaces from JSP page output. Its default value is false. It helps in reducing the generated code size, notice the generate servlet code keeping this attribute value as true and false. You will notice no out.write("\n") when it's true.

```
<%@ page trimDirectiveWhitespaces ="true" %>
```

3.3. JSP include directive:

JSP include directive is used to include the contents of another file to the current JSP page. The included file can be HTML, JSP, text files etc. Include directive is very helpful in creating templates for user views and break the pages into header, footer, sidebar sections.

We can include any resource in the JSP page like below.

```
<%@ include file="test.html" %>
```

The file attribute value should be the relative URI of the resource from the current JSP page.

3.4. JSP taglib directive:

JSP taglib directive is used to define a tag library with prefix that we can use in JSP, we will look into more details in JSP Custom Tags tutorial.

We can define JSP tag libraries in like below;

```
<%@ taglib uri="/WEB-INF/c.tld" prefix="c"%>
```

4. JSP Standard Action Tags

4.1. Introduction:

JSP provides a bunch of standard action tags that we can use for specific tasks such as working with java bean objects, including other resource, forward the request to other resource etc.

The list of standard JSP action elements are given below.

JSP Action	Description
jsp:include	To include a resource at runtime, can be HTML, JSP or any other file
jsp:useBean	To get the java bean object from given scope or to create a new object of java bean.
jsp:getProperty	To get the property of a java bean, used with jsp:useBean action.
jsp:setProperty	To set the property of a java bean object, used with jsp:useBean action.
jsp:forward	To forward the request to another resource.
jsp:text	To write template text in JSP page.
jsp:element	To define the XML elements dynamically.
jsp:attribute	To define the dynamically generated XML element attributes
jsp:body	To define the dynamically generated XML element body
jsp:plugin	To generate the browser-specific code that makes an OBJECT or EMBED tag for the Java plugin.

4.2. JSP Bean Actions:

We can use jsp:useBean like below in JSP pages to get the bean object.

```
<jsp:useBean id="myBeanAttribute" class="com.coder" scope="request" />
```

In above example, JSP container will first try to find the myBeanAttribute attribute in the request scope but if it's not existing then it will create the instance of MyBean and then assign it to the myBeanAttribute id variable in JSP and sets it as an attribute to the request scope. Once the bean is defined in JSP, we can get it's properties using jsp:getProperty action like below.

```
<jsp:getProperty name="myBeanAttribute" property="count" />
```

Example on <jsp:useBean>:

Pojo for user details: Userdetails.java

```
public class Userdetails {
	private String username;
	private String password;
	private int age;
	private String address;
	public String getUsername() {
		return username;
	}
	public void setUsername(String username) {
		this.username = username;
	}
	public String getPassword() {
		return password;
	}
	public void setPassword(String password) {
		this.password = password;
	}
	public int getAge() {
		return age;
	}
	public void setAge(int age) {
		this.age = age;
	}
	public String getAddress() {
		return address;
	}
	public void setAddress(String address) {
		this.address = address;
	}

	@Override
	public String toString() {
		return "Userdetails [username=" + username + ", password=" + password
				+ ", age=" + age + ", address=" + address + "]";
	}
}
```

Input.jsp

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

<!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=UTF-8">
<title>Insert title here</title>
</head>
```

```
<body>
<form action="userbeanb1.jsp" method="post">
<table>
<tr><td>User Name:</td><td> <input type="text" name="username"></td>
<tr><td>User Password:</td><td> <input type="password" name="password"></td>
<tr><td>User Age:</td><td> <input type="text" name="age"></td>
<tr><td><input type="submit" value="Register"></td>
<td><input type="reset" value="Clear"></td></tr>
</table></form> </body></html>
```

userbeanb1.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!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=UTF-8">
<title>Insert title here</title>
</head>
<body>
<jsp:useBean id="userinfo" class="test.Userdetails"></jsp:useBean>
<jsp:setProperty property="*" name="userinfo"/>
You have entered below details:<br>
<jsp:getProperty property="username" name="userinfo"/><br>
<jsp:getProperty property="password" name="userinfo"/><br>
<jsp:getProperty property="age" name="userinfo" /><br>
</body>
</html>
```

In above program

```
<jsp:useBean id="userinfo" class="test..Userdetails"></jsp:useBean>
```

This line defines useBean id=”userinfo” and the file source is defined by class attribute class="*servlet.Userdetails*".

Proprty is set by:

```
<jsp:setProperty property="*" name="userinfo"/>
```

By using <jsp:setProperty> we can set property to the id which is defined in the useBean tag. If all request parameter names matches with the java bean property the we can simply put “*”.

It will be used by using:

```
<jsp:getProperty property="username" name="userinfo"/><br>
```

1. JSP Include Action:

We can use jsp:include action to include another resource in the JSP page, earlier we saw how we can do it using **JSP include Directive**.
Syntax of jsp:include action is:

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

The difference between JSP include directive and include action is that in include directive the content to other resource is added to the generated servlet code at the time of translation whereas with include action it happens at runtime.
We can pass parameters to the included resource using jsp:param action like below.

```
<jsp:include page="header.jsp">
<jsp:param name="myParam" value="myParam value" />
</jsp:include>
```

We can get the param value in the included file using **JSP Expression Language**.

2. JSP Forward Action:
We can use jsp:forward action tag to forward the request to another resource to handle it. It's syntax is like below.

```
<jsp:forward page="login.jsp" />
```

Note: You may find output of <jsp:include> and <jsp:forward> is same but if you have some data in file then you can see diffrence in this two tags.
Simply
<jsp:include> : will include another file in our current file
<jsp:forward>: will forward the current request to the forwarding page

ASSIGNMENT

1) Write jsp program to display current date.
2) Write jsp program to create multiplication method.
3) Write a jsp program which shows the use of scriptlet and expression.
4) Write a jsp program which will another page and also pass parameter to included page.
5) Define Book name, author and publication in web.xml file for one jsp page and retrive that information into same jsp file.
6) Make a JSP page that is accessible only to employees and executives. Display the username of the person accessing the page.

7)Make a JSP page that randomly selects a background color for each request. Just choose at

random among a small set of predefined colors. Be sure you do not use the JSP-Styles.css style sheet, since it overrides the colors given by <BODY BGCOLOR="...">.

7. Make a JSP page that lets the user supply a request parameter indicating the back-ground color. If no parameter is supplied, a background color should be selected at random.

8.Make a JSP page that lets the user supplya request parameter indicating the back-ground color. If no parameter is supplied, the most recently used background color (from a previous request by any user) should be used.

9)Make a JSP page that is accessible only to employees and executives. Display the username of the person accessing the page.

10) Make a variation of the above page that also keeps track of the username of the previous person to access the page. Show the current username to everyone; show the name of the previous user only to executives. The simplest way to simulate two different users is to use two different browsers.