unit-~~4~~ 5 JSP BasiCS

what's wrong with servlets, running of first JSP, how JSP works, the JSP servlet core, The JSP API, The generated servlet revisited, implicit objects.

part - II

JSP syntax - directives scripting elements, standard action elements, Comments Converting into XML syntax. The develop JSP beans - Calling your bean from JSP page, making a bean available, brief theory of java bean, Accessing properties using JSP: get property() JSP: setproperty() setting a property value from a request, Java beans code initilization, the sql tool being Example.

part - III

using JSP: custom tags, writing your first custom tag, The role of deployment descriptor, the tag library descriptor. The custom tag syntax, The JSP custom tag API, the life cycle of tag handler.

what's wrong with servlet:

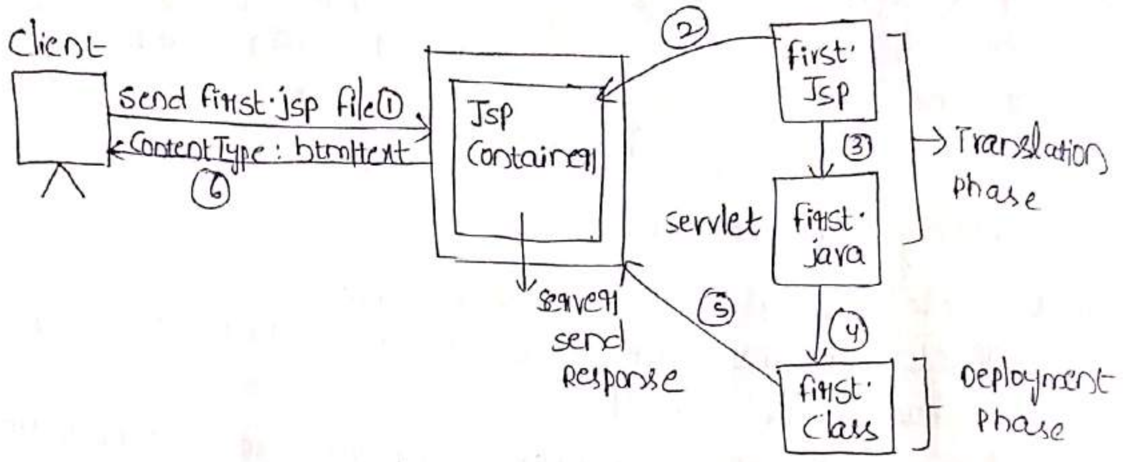
1) servlet Code is almost in java Code.
2) the servlet programmer has a good knowledge in java to develop a servlet.
3) Here the user is manually compiling and generating class file.
4) for any subsequent modifications we have to compile it again.
5) The presentation and logic code specified in java lines.
6) But in JSP's the presentation and logic code is seperated.
7) we have to modify web.xml in servlet

Running your first JSP:.

Eg)
```
<html>
<body>
<%
out.println("Helloworld");
  %>
</body>
</html>
```

1) Save above Code as filename. Jsp
2) Copy your Jsp file into the following path
webapps/your directory/
3) In netbeans IDE we have to Create a web application project.
4) under your project directory you can find source package folder.
5) under source package directory we have default package folder the right click on default package and select Jsp.
6) After creation of Jsp pages all these Jsp pages are located into web Info directory.

How JSP works



1) Create first.jsp and calling using browser.
http://localhost: 8080// your directory.
2) In webserver we have a container ~~or~~ has page compiler
3) The Jsp Container receives Jsp file and convert it into first. Java (The generated servlet code).
4) The generated servlet code is compile and generate class ~~and~~ file and send the same to the server.
5) The server simply send the response to the client.
6) The JSP page compile only once for any subsequent modification it won't compile again. That's only it's performance is so good.

Generated servlet Code:
In netbeans IDE we have to right click on Jsp file then we can find (View servlet option).

* Double click on new servlet option. It will open a generated servlet code. The respective jsp
* In tomcat folder we can find work directory in tomcat class path. After opening work directory we can find the generated servlet code.

Jsp API: It has the following two packages

Javax.servlet.jsp;
Javax.servlet.jsp. ~~[illegible]~~ tag ext;

these two packages are having following Interfaces and classes.

1) Jsp page Interface
2) Http Jsp page Interface

Classes:

1) JSP Factory class
2) JSP Engine Info
3) page Context
4) Jsp writer

Exception Classes:

1) JSP Exception
2) JSP Error

Interfaces:

1) Jsp page: It is an interface. it has the following method to initialize Jsp. servlet and to destroy Jsp servlet

```
public void jspInit()
 {
   =
 }
public void jspDestroy()
 {
   =
 }
```

2) Http Jsp page Interface: It is also an interface which contains a service method to handle request and responses.

```
public void _jspservice(ServletRequest req, ServletResponse res)
{
  =
}
```

Classes

1) Jsp Factory Class: It is an abstract class which provides methods for obtaining other objects to Process Jsp page. it has get default factory() which returns a Jsp Factory object.

2) Jsp Engine Info:- It is an also abstract class which can be used to retriew information on the Jsp Container. we have getEngineInfo( ) to return JspEngine Info object.

3) Page Context: It is also an abstract class which is used to provide various objects to process Jsppage it has different methods. Those are

getRequest( )
getResponese( )
getSession( )
getServletContext( )
getServletConfig( )

The method getpageContext( ) is used to return the page Context object.

4) Jsp writer: It is derrived from java.io package it is used to write the response to the Client.

Exception Classes:-

Jsp Exception and Jsp Error: These classes are used to generate an exception.

The generated Servlet Revisited:

we have to revisit the generated Servlet Code. In Servlet page we have all Jsp API's. So we have to use the same("view servlet option") to generate the servlet Code.

Implicit objects:

The Jsp has the following implicit object

1) request → it is a reference of Servlet Request
2) response → It is a reference of servlet Response Interface
3) OUT → it's an object of Jsp writer Class
4) session → it is a referrence of http session
5) Config → it's a reference of Servlet Config
6) Page → it is a reference of http jsp Page

7) Application → it is a reference of servlet context.
8) pageContext → it is a reference of page context.
9) Exception → it is a reference of Jsp Exception and the type is throwable.

For Example,
```
<%
request.getparameter("uname");
%>
<%
out.println("welcome");
%>
```

Jsp syntax:

1) directives: The directives are messages which are sent to Jsp Container to translate Jsp page to servlet page. In Jsp we have three directives

1) page
2) include
3) taglib

1) page directive: page directive has different attributes to send messages to the Jsp container. we have the following page attributes.

1) language   11) Content Type
   Ex:- Java
2) Import
3) extend
4) Info
5) Buffer
6) Autoflush
7) Session
8) Errorpage
9) ISErrorpage
10) ISThreadsafe

In Jsp's the directives are represented as follows
<%@ Directive attribute = "value" %>

1) language: It is a page directive Attribute. it is used to specify the language.
<%@ page language = "java" %>

2) import: It is used to import packages into Jsp page

<%@ page import = "java.io.*"%>

3) Extend: It is used to extend a super class

<%@ page extend="JSPwriter"%>

4) Info: It is used to send an information to the container

<%@ page info ="welcome to jsp"%>

5) Buffer: It is used to maintain a buffer at server side

<%@ page buffer= "1024"%>

6) Autoflush: It is a boolean value if it is true then automatically the buffer will clear

<%@ page Autoflush = "true" %>

7) Session :- It is also a boolean value if the value is true the server will create a session.

<%@ page session= "true"%>

8) Error page: It is used to move the error page wherever an error is occur.

<%@ page errorpage="errorpage.jsp" %>

9) isErrorpage: It is also a boolean value if it is true the server makes sure that it is error page.

<%@ page isErrorpage= "true"%>

10) isThreadsafe: It is also a boolean value by default it is true for all JSP pages. we can also assign false value which make sure that it is not safe

<%@ page isThreadsafe= "false"%>

11) ContentType: it is used to set the content type

<%@ page contentType = "text/html"%>

Include directive: It is used to include the file in a current JSP file. it has the following syntax:

<%@ include file ="relative URL"%>

<%@ include file = "next.html"%>

Taglib directive; It is used to create a custom tags it has the following syntax:

<%@ taglib attribute ="value"%>

where attributes are prefix and class prefix represents a custom tag prefix and the class represents Tag handler class.

Ex:- <%@ taglib ·class = "CSE · TagHandler" %>

scripting Elements: The scripting elements are used to write script in JSP page we have the following scripting element.

1) scriplets
2) Declaration
3) expression

1) scriplets: The scriplets are code blocks of JSP page every scriplets starts with <% and ends with %>

Ex:- <%
        out.println("welcome");
        %>

2) Declarations: It is used to declared variables and methods in JSP page. This are always starts with <%! and close with %>

Ex:- <%! int i=4; %>

3) Expressions: It is used to declared an expression in JSP page. it is always starts with <%= and closed with %>

Ex:- <%= i+j ; %>

standard action Elements:

we have the following standard actions in JSP.

1) <jsp: useBean> - it is used to make the bean available to the JSP page.

2) <jsp: set property> - it is used to set the property of bean

3) <jsp: getproperty> - it is used to get the property of bean

4) <jsp: include> - it is ~~als~~ similar to include directive we can include a page in using this action element current JSP page.

Ex:- <jsp: include page = "relative URL" />

5) <jsp: forward> - it is used to stop the execution of current JSP page in the browser will redirect or forward to the specified page

Ex:- <jsp: forward page = "relative url" />

6) <jsp : plugin > → it is used to add plugin software to the browser
7) <jsp: param> → it is used to inside jsp: plugin to define parameters
8) <jsp: fallback> → it is also used in inside jsp:plugin Generally it is used to display an error message.

Comments

Commenting is a good programming practice we can mention two types of comments in JSP page.

a) The comments embedded in web page
b) The comments embedded in jsp page

a) Comments embedded in webpage:

these Comments are embedded in webpage directory

Ex:- <! - - - - - >

```
<%
out.println("<!= Here is a Comment -->");
%>
```

b) Comments embedded in jsp page:

The Comments used in Jsp page is different Compare to the previous. In Jsp page the Comments starts with <%-- and ends with --%>

Ex:- <% - - -

Here is a Comment

- - - %>

Converting into XML Syntax:

In Jsp we have directives scriptlets declarations and expressions and template data. Each and Every one has its own syntaxes. we can covert those syntaxes into XML syntax.

```
<% @ directive attribute = "value" %>
<jsp : directive : directive_name attribute _List />
< % scriplet %>
<jsp : scriplet > script </jsp : scriplet >
<%! Declaration %>
<jsp: Declaration > Declaration </jsp : Declaration>
```

```
<%=expression %>
<jsp:expression> expression </jsp:expression>
```

If you want to write any templet data in jsp we can use the following xml syntax

```
<jsp:text>data </jsp:text>
```

Ex:- 8a) Create a simple jsp page

```
<%@ page language = "java" import = "java.util.*"
            ContentType = "text/html" %>
   <html>
   <head>
  <title> Simple Jsp </title>
   </head>
   <body>
   <%
    out.println("<center> <h1> welcome to jsp </h1></center>
                                                        ");
    %>
  <strong> CurrentTime & Date: </strong> <% = new Date();
                                                      %>
   </body>
   </html>
```

Developing Jsp beans:

A brief theory of java bean:

A bean is a java class always the class name should be declared as public and it won't extend any class and it won't implements any interface. These bean has always a default constructor. If there is no any default construction, the compiler will create a default constructor itself. Every bean has following two methods:

1) public void setPropertyName(type PropertyName)

```
  {
  }
```

The above method is used to set the property of bean for example, if you want to set the property called firstName then the method is as follows:

2) public void setFirstName(string firstName)

```
  {
  this.firstName = firstName;
  }
```

Scanned by CamScanner

3) Public type getpropertyName()
```
{
-
}
```
It is used to get the property ~~fa~~ value from the bean.
```
public String getFirstName()
{
return . FirstName;
}
```
Calling your first Jsp bean:-

we can call Jsp bean using following steps

Step1:- Create the bean class using the following code
```
Package bean;
public Class myBean
{
public. int doubleIt (int i)
{
return 2*i;
}
}
```
Step2: Save the above Code as MyBean. java and Compile it paste the Class file into Classes Folder which is in webInfo.
```
webapps
    ↳ your directory
      ↳ web Info
          ↳ Classes
              ↳ bean
                  ↳ myBean. class
```
Step3: Create a Jsp page as Follows.
```
<%@ page language=" java" ContentType ="text/html" %>
<jsp: useBean id= "cse" Class ="bean. myBean"/>
<html>
<body>
<%
```

```
int i=4;
int j = cse.doubleIt(i);
out.println("<h1> 2* 4 = "+j+"</h1>");
%>
</body>
</html>
```

save above code as filename.jsp and call jsp page using browser in your browser the JSP page displays 2*4=8

Making a bean available: we can make a bean to be available for the available JSP page using following syntaxes

```
<jsp:useBean attribute ="value"/>
<jsp: useBean attribute="value">
    inilization code
</jsp : useBean>
```

where attributes are as follows

1) id → defines a unique identifier for a bean

2) class → It defines the fully qualified name for bean[to specify the package name also]

3) type → It specify the type of java bean the class name itself is a type.

4) scope → It defines the accessibility & life time of a bean. we can take any one of the following values for the scope a) page b) session c) request d) application

5) beanName → this attribute specify the name of the bean.

using jsp:setproperty & jsp: getproperty

jsp: setproperty:- This action element is used to set the properties for bean from the Jsp page. It has the following syntax

```
<jsp:setproperty name="Bean name" property="property name"
                                    value ="property value"/>
```

jsp:getproperty:- This action element is used to get the property from the bean. It has following syntax

```
<jsp:getproperty name="Bean name" property="property name"/>
```

Ex:- write a jsp program to set and get the property from the bean. (week 8C. jsp)

```
    <html>
    <head>
    <title> Accessing property </title>
     </head>
    <body>
    <jsp: useBean id="person" class="Bean.personBean">
    <jsp: setproperty name="person" property ="FirstName"
                                         value ="cse"/>
  <jsp: setproperty name="person" property="LastName"
                                         value ="IT"/>
  <Center>  </jsp:useBean>
  <b> FirstName:
        <jsp: getproperty name="person" property="FirstName"/>
  <b> Lastname:
        <jsp: getproperty name="person" property="LastName"/>
   </center>
   </body>
   </html>
```

personBean.java

```
  Package Bean;
  Public class personBean
  {
   Private string FirstName =null;
   Private string lastName =null;
   Public void setFirstName(string FirstName)
   {
    this.FirstName = FirstName;
   }
```

Scanned by CamScanner

```
Public String getFirstName()
{
 return firstName;
}
public void setLastName(String LastName)
{
this.LastName =LastName;
}
public ~~vo~~String getLastName()
{
 return lastName;
}
}
```

Setting a property value from a request:

we can set the property values using jsp:setproperty action element. if there is no request we can set the property values as follows

```
<jsp: setproperty name="Beanname" property="property_name"
Ex:-                              value = "property_value"/>

<jsp:setproperty name="person" property="age" value="29"/>
```

we can set the properties from the request (from the submitted form) using the same jsp: setproperty action element as follows, but instead of specifying the property values, we can get the property values from the form parameter names. The parameter names are specified by using param attribute.

```
<jsp: set property name="Bean name" property="property.nam
                                    Param="ParameterName"/>

<input type="text" name="myage"/>
```

form
age 29
Submit

```
<jsp: setproperty name="person" property="age"
                                param="myage"/>
```

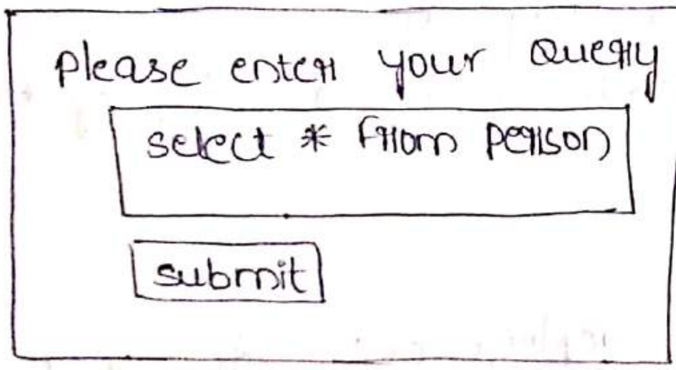
JavaBean Code initialization:

we can initialize the JavaBean Code by using Jsp: useBean action element this action element has the Bean Classname and the Bean IDE.

Ex - <jsp: useBean id = "person" class = "Bean. personBean">
 Bean is initializing
 </jsp: useBean>

SQL Tool Bean Example:

SQL Tool Bean has a jsp page and a Bean class in Jsp page we can create a text area. in text area we can enter a Query. The same query is submi-tted to the bean from the Jsp page. In Beanclass we have a data base activity steps. All these activities will crosses the submitted query and the result will be displayed in our web browser.



Please enter your Query

select * from person

submit

Name	email	phone
cse	cse@gmail.com	123456

• custom tags:

SQLTOOL BEAN

Login.html:

```
<HTML>
<HEAD>
<TITLE>Login Page</TITLE>
</HEAD>
<BODY>
<CENTER>
<FORM METHOD=POST ACTION=SQLTool.jsp>
<TABLE>
<TR>
 <TD>User Name:</TD>
 <TD><INPUT TYPE=TEXT NAME=userName></TD>
</TR>
<TR>
 <TD>Password:</TD>
 <TD><INPUT TYPE=PASSWORD NAME=password></TD>
</TR>
<TR>
 <TD><INPUT TYPE=RESET></TD>
 <TD><INPUT TYPE=SUBMIT VALUE="Login"></TD>
</TR>
</TABLE>
</FORM>
</CENTER>
</BODY>
</HTML>
```

SQLTool.jsp:

```
<jsp:useBean id="theBean" class="Bean.SQLToolBean">
<%
 try {
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
 }
 catch (Exception e) {
 out.println(e.toString());
 }
%>
</jsp:useBean>
<jsp:setProperty name="theBean" property="userName"/>
<jsp:setProperty name="theBean" property="password"/>
<jsp:setProperty name="theBean" property="connectionUrl"
value="jdbc:odbc:JavaWeb"/>
<jsp:setProperty name="theBean" property="sql"/>
<HTML>
<HEAD>
<TITLE>SQL Tool</TITLE>
</HEAD>
```

```
<BODY>
<BR><H2>SQL Tool</H2>
<BR>Please type your SQL statement in the following box.
<BR>
<BR><FORM METHOD=POST>
<INPUT TYPE=HIDDEN NAME=userName VALUE="<jsp:getProperty
name="theBean"
property="userName"/>">
<INPUT TYPE=HIDDEN NAME=password VALUE="<jsp:getProperty
name="theBean"
property="password"/>">
<TEXTAREA NAME=sql COLS=80 ROWS=8>
<jsp:getProperty name="theBean" property="sql"/>
</TEXTAREA>
<BR>
<INPUT TYPE=SUBMIT>
</FORM>
<BR>
<HR>
<BR>
<%= theBean.getResult() %>
</BODY>
</HTML>
```
SQLToolBean.java:
```
import java.sql.*;
import Bean.StringUtil;
public class SQLToolBean {
 private String sql = "";
 private String userName = "";
 private String password = "";
 private String connectionUrl;
 public String getSql() {
 return StringUtil.encodeHtmlTag(sql);
 }
 public void setSql(String sql) {
 if (sql!=null)
 this.sql = sql;
 }
 public void setUserName(String userName) {
 if (userName!=null)
 this.userName = userName;
 }
 public String getUserName() {
 return StringUtil.encodeHtmlTag(userName);
 }
 public void setPassword(String password) {
 if (password!=null)
 this.password = password;
 }
 public String getPassword() {
 return StringUtil.encodeHtmlTag(password);
 }
 public void setConnectionUrl(String url) {
 connectionUrl = url;
 }
public String getResult() {
 if (sql==null || sql.equals(""))
 return "";
 StringBuffer result = new StringBuffer(1024);
 try {
 Connection con = DriverManager.getConnection(connectionUrl, userName,
password);
 Statement s = con.createStatement();
 if (sql.toUpperCase().startsWith("SELECT")) {
 result.append("<TABLE BORDER=1>");
 ResultSet rs = s.executeQuery(sql);
 ResultSetMetaData rsmd = rs.getMetaData();
 // Write table headings
 int columnCount = rsmd.getColumnCount();
 result.append("<TR>");
 for (int i=1; i<=columnCount; i++) {
 result.append("<TD><B>" + rsmd.getColumnName(i) + "</B></TD>\n");
 }
 result.append("</TR>");
 while (rs.next()) {
 result.append("<TR>");
 for (int i=1; i<=columnCount; i++) {
 result.append("<TD>" + StringUtil.encodeHtmlTag(rs.getString(i)) +
"</TD>" );
 }
 result.append("</TR>");
 }
 rs.close();
 result.append("</TABLE>");
 }
 else {
 int i = s.executeUpdate(sql);
 result.append("Record(s) affected: " + i);
 }
 s.close();
 con.close();
 result.append("</TABLE>");
 }
 catch (SQLException e) {
 result.append("<B>Error</B>");
 result.append("<BR>");
 result.append(e.toString());
 }
 catch (Exception e) {
 result.append("<B>Error</B>");
 result.append("<BR>");
 result.append(e.toString());
 }
 return result.toString();
 }
}
```

StringUtil.java:

```
public class StringUtil
 {
public static String encodeHtmlTag(String tag) {
 if (tag==null)
 return null;
 int length = tag.length();
 StringBuffer encodedTag = new StringBuffer(2 * length);
 for (int i=0; i<length; i++) {
 char c = tag.charAt(i);
 if (c=='<')
 encodedTag.append("<");
 else if (c=='>')
 encodedTag.append(">");
 else if (c=='&')
 encodedTag.append("&amp;");
 else if (c=='"')
 encodedTag.append("&quot;"); //when trying to output text as tag's
 // value as in values="???".
 else if (c==' ')
 encodedTag.append("&nbsp;");
 else
 encodedTag.append(c);
 }
 return encodedTag.toString();
}
}
```

using jsp custom tags:

using Jsp Custom tags we can seperate presentation code and implementation code. In presentation code we have only tags. In Implementation code we have java code. In Jsp we can use only Custom tags. There is no Java code simply we can seperate a java code.

writing your first Custom tag:-

step1:- first create tld file (tag library descriptor file.tld) and save it inside web info directory

step2: create a jsp page which contains a custom tag.

step3: Create a tag handler class which is used to handle the custom tag

step4: Change the deployment descriptor (XMl file) just we have to add the tag link directory uri and location.

Step5: Restart the tomcat

step6: open the browser and call our jsp page.

Role of deployment descriptor:

Generally the deployment descriptor is a XML file when we are using custom Tags in XML File. we have to add the following lines to the XML descriptor the following lines contains the tag uri

```
<web-app>
<display-name> template </display-name>
<taglib>
 <tag-uri>/mytld</tag-uri>
 <tag - location> web-inf /mytld.tld </tag_~~name~~ location>
 </taglib>
</web-app>
```

Instead of changing deployment descriptor we can also mention the tag uri and tag_location in the Jsp page itself.

we can mention the uri location, by using the tag-lib directive

Ex:- <%@ taglib uri="web-inf/ mytld.tld" prefix="myprefix"%>

The taglibrary descriptor:

we can create taglibrary descriptor (~~a~~ tld) by using tag lib directive. It has the following elements. we can save every tld file as .tld.

Ex:- mytld.tld

Every tld file has a root node or root back which is <taglib>. under this ~~<taglib>~~ we have sub nodes or subtags. Those are as follows.

a) tlib. version → It represents the taglibrary version.

b) Jsp version → It represents the Jsp version number.
c) short name → it represents the short name of tld.
d) Info → It represents the information of tld.
e) Tag → It represents the custom tag name, the custom handler name class.
f) uri → It represents the resource identity of a tld file.

<Tag>:- The tag is used to create a custom tag it is always used inside a tld file. this tag also represents the location of tag handler class. It has the following sub elements.

a) name → it represents the name of the custom tag
b) tag class → it represents the tag handler class.
c) body content → it represents the body content of custom tag. If there is no any content we can make it as empty.
d) info → it represents the information of custom tags.
e) attribute → it represents the attributes of the custom tag.

Custom tag syntax:-

```
<%@ taglib uri=" uri of tld" prefix=" prefix of custom
                                              tag"%>
        <prefix: customtag name/>
```

The above two syntaxes are used inside Jsp page in order to use a custom tag from the Jsp page. where prefix is used to seperate the custom tags.

The Jsp custom Tag API:- It has the following interfaces and classes. These interfaces and classes are used to create a tag handler class. if you want to use these interfaces and classes, we have to import two packages.

```
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
```

Interfaces:

1) Tag interface → It is a default interface to create custom tag handler. It has the following methods.

1) doStartTag( ) → it is used to start custom tag
2) doEndTag( ) → it is used to stop custom tag process
3) getParent ( ) → it is used to custom tag parent
4) setParent ( ) → it is used to set parent for custom tag by default. The parent of each custom tag is Tag object
5) setPageContent ( ) → It is used to set the object of pageContent which refers the Jsp page
6) release ( ) → It is used to release all objects.

2) Iteration Tag Interface → It is used to perform iteration using custom Tag. It is always extends the tag interface.

3) The bodyTag Interface → it is used to set the body content of a custom tag. it has the following two methods.

setBodyContent ( ) → it is used to set the object of body content class. This class is always refers the body content of custom tag.

doInitBody( ) → It is used to initialize the body content of a custom tag.

using above interfaces we can implement a tag handler but it is not possible to use these interfaces in each tag handler, because the handler should be implement all methods of interfaces. These may increase more complexity of Taghandler class. These problem is overcome by the Taghandler classes.

Classes:

1) SimpleTag support class → It is a default class which is extended by a tag handler class. It has doTag( ) to process the custom tag.

2) The body Content class → This class is used to process the content of custom tags.

3) BodyTagSupport Class → This Class is always implements the BodyTaginterface.

Life cycle of Tag Handler:

Every Tag handler class will follow the following life cycle methods.

$$\begin{tikzcd}
\text{setpageContext()} \arrow[d] \\
\text{setparent()} \arrow[d] \\
\text{getparent()} \arrow[d] \\
\text{dostartTag()} \arrow[d] \\
\text{doEndTag()} \arrow[d] \\
\text{release()}
\end{tikzcd}$$

step1: The tag handler class will set the page context of a JSP page by using setpageContext(). It has the following syntax.

public void setpageContext(pageContext pc)

step2: It will set the parent tag for the custom tag. The default parent of custom tag is Tag object. It has the following syntax.

public void setparent(Tag t)

step3:- If you want to retrieve the parent of handler class we have to use ~~differ~~ getparent method. it has the following syntax:

public void getparent()

step4: Now we can start the process of custom tag using dostartTag(). it has the following syntax.

public int doEndTag() throws JspException

step5: we can stop the process of custom tag by following doEndTag(). it has the following syntax.

public ~~void~~ int doendTag() throws JSPException

step6:- Release() is used to release the allocated object. it has the following syntax:

Public void Release ( )

write a jsp program to create a Custom tag.

mytld.tld

```
<taglib>
 <tlib_version> 1.2.3 </tlib_version>
 <jsp-version> 1.2 </jsp-version>
 <short-name>mytld </short-name>.
 <info> This is tld File </info>
 <tag>
     <name> myHandler </name>
     <tag-class> Bean.TagHandler. </tag-class>
 </tag>
 </taglib>
```

week 8b.jsp:-

```
<%@ Page ContentType ="text/html" %>
<%@ taglib ~~rag~~ Prefix ="myprefix" uri="/web-Inf/mytld/
                                          mytld.tld"%>
<html>
<head>
<title> custom tag</title>
</head>
<body>
   <myprefix: myHandler/>
</body>
</html>
```

TagHandler.java

```
 package Bean;
 import javax.servlet.jsp.*;
 import javax.servlet.jsp.tagext.*;
 public class TagHandler extends simpleTagsupport.
 {
  public void doTag() throws IOException, JspException
  {
  JspWriter out = getJspContent().get out();
  out.println("<center> <h1> welcome to custom tag
                                         </h1> </center>");
 }
}
```