Marwadi
education foundation

Unit – 4
Java Server Pages

Ravikumar R Natarajan
Assistant Professor
Dept. of CE

KNOWLEDGE IS THE CURRENCY
FOR THE 21st CENTURY

Content

- Introduction to JSP
- Comparison with Servlet
- JSP Architecture
- JSP: Life Cycle
- Scripting Elements
- Directives
- Action Tags
- Implicit Object
- Expression Language(EL)
- JSP Standard Tag Libraries(JSTL)
- Custom Tag
- Session Management
- Exception Handling
- CRUD Application

Introduction to JSP

- JSP Stands for **Java Server Pages**.
- 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 logic.**
- JSP is a **server side scripting language**.

Introduction to JSP

Example code showing different types of JSP Tags:

```
<%-- Counter.jsp --%>

<%@ page language="java" %>
<%! int count = 0; %>
<% count++; %>
Welcome! You are visitor number
<%= count %>
```

```
<%-- Comment Tag --%>

<%-- Directive Tag --%>
<%-- Declaration Tag --%>
<%-- Scriptlet Tag --%>

<%-- Expression Tag --%>
```

Problems with Servlet

- Inside the one and only one class in servlet alone, we will perform various task:
  - Acceptance of request
  - Process the request
  - Handling the business logic
  - Generation of response
- For creating a servlet **knowledge of java and html both is needed.**
- While developing an app if the look and feel of program changes we have to **change the entire servlet class.**

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.

Advantage of JSP over Servlet

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.

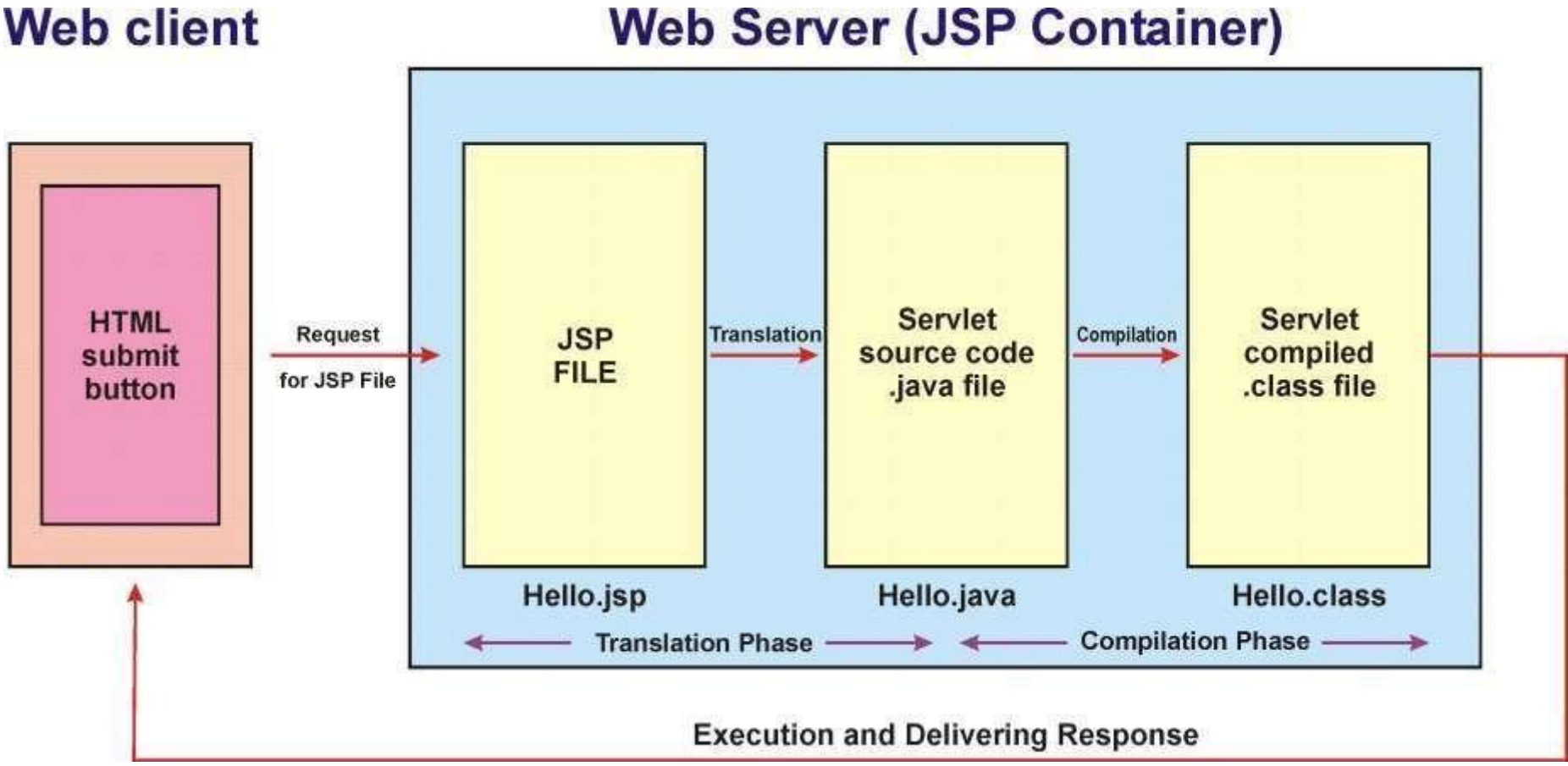
4) Less code than Servlet

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 VS SERVLET

Sr. No.	JSP	SERVLET
1	**JSP is a scripting language** which generates dynamic content	**Servlet are java program** that can be compiled to generate dynamic content
2	**JSP runs slower** than servlet	**Servlet runs faster** than JSP
3	In MVC **JSP acts as view.**	In MVC **Servlet acts as controller.**
4	JSP **can build custom tags**	**No facility** for creating custom tags
5	It is easier to code a JSP	The servlet are basically complex java programs

JSP Architecture



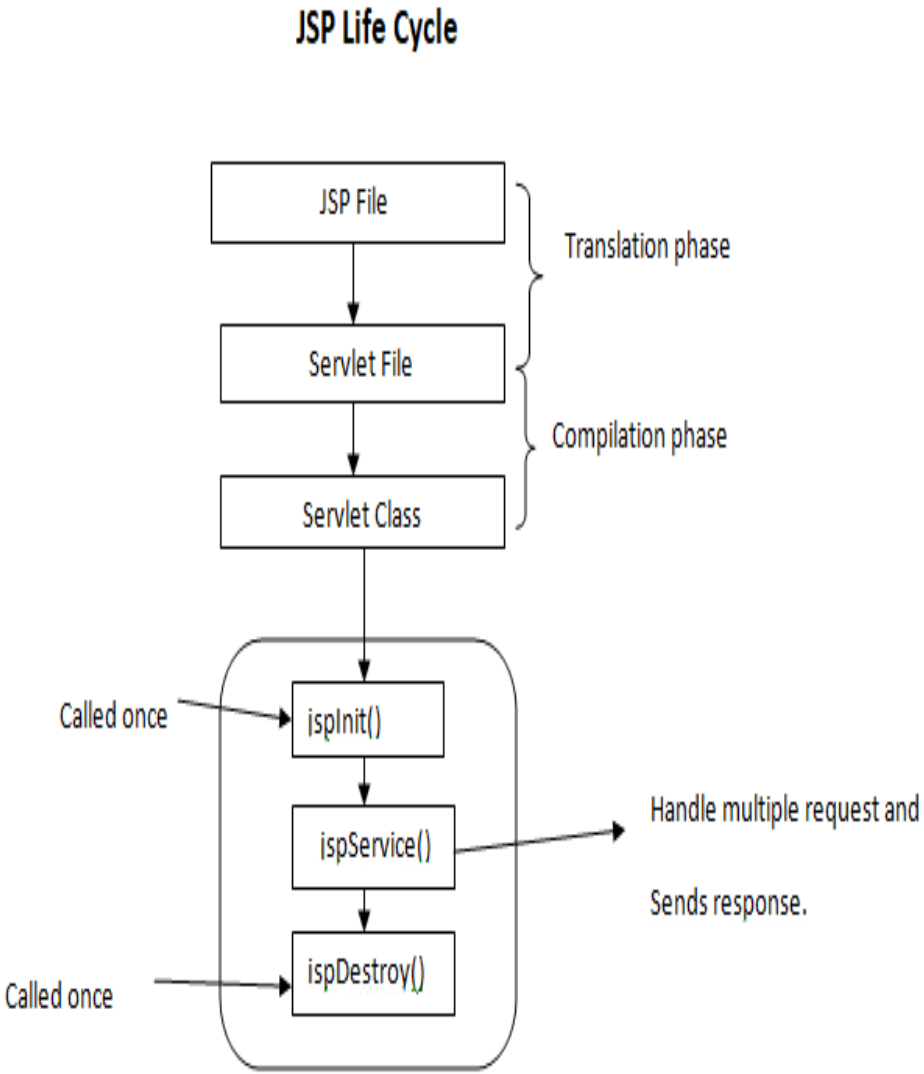
JSP Architecture

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**.

JSP Architecture

1. Translation of JSP page to Servlet
2. Compilation of JSP page(Compilation of JSP into test.java)
3. Classloading (test.java to test.class)
4. Instantiation(Object of the generated Servlet is created)
5. Initialization(jspInit() method invoked by the container)
6. Request processing(jspService()is invoked by the container)
7. JSP Cleanup (jspDestroy() method is invoked by the container)



First JSP Program

Write a program to print hello using JSP.

Step 1: In NetBeans Start new Project. Give name “FirstJsp”
Step 2: Right click on web page folder add new JSP file. Name it as MyJsp
Step 3: Add below line in <body> tag

```
<h1>Hello From JSP</h1>
```

Step 4: Run File

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:

1. page directive
2. include directive
3. taglib directive

Syntax of JSP Directive

<%@ directive attribute="value" %>

1.PAGE DIRECTIVE

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

1. import,
2. contentType,
3. extends,
4. info,
5. buffer,
6. isELIgnored
7. language,
8. errorPage,
9. isErrorPage
10. IsThreadSafe,

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>
```

contentType

The contentType attribute **defines the MIME (Multipurpose Internet Mail Extensions) type of the HTTP response.**

The default value is “text/html”.

```
<html>
<body>

<%@ page contentType=text/html %>

</body>
</html>
```

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>
```

Example

Prog 2: Use import, contentType and buffer JSP PAGE Directive

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
   <head>
      <title>Import,contentType and buffer tag Example</title>
   </head>
   <body>
      <%@page buffer="16kb" %>
      <%@ page import="java.util.Date" %>
         Today is: <%= new Date() %>
   </body>
</html>
```

isThreadSafe

Marwadi education foundation

Servlet and JSP both supports multithread.

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 serialise** the multiple requests, i.e. it will wait until the JSP finishes responding to a request before passing another request to it.

Its syntax is:

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

info

This attribute simply **sets the information** of the JSP page.

```
<html>
<body>

<%@ page info="Composed by RK KEYNOTES" %>
Today is: <%= new java.util.Date() %>

</body>
</html>
```

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.

index.jsp

```
<html>
    <body>

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

    </body>
</html>
```

isErrorPage

The isErrorPage attribute is used **to declare that the current page is the error page.**

myerrorpage.jsp

```
<html>
<body>

<%@ page isErrorPage="true" %>

 Sorry an exception occured!<br/>

The exception is: <%= exception %>

</body>
</html>
```

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.

Syntax:

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

language

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

Extends

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

2. JSP Include Directive

The include directive is **used to include the contents** of any resource it may be **jsp file, html file or text file**.

Syntax of include directive

```
<%@ include file="resourceName" %>
```

Example

In this example, we are **including the content of the index.html** file in a include_action.jsp page.

```
<html>
        <body>
                <%@ include file="index.html" %>
        </body>
</html>
```

3. JSP Taglib directive

This directive allows user to use **custom tags** in JSP. The custom tags are those that are created by user.

Syntax JSP Taglib directive:

```
<%@ taglib uri="uriofthetaglibrary" prefix="prefixoftaglibrary" %>
```

JSP SCRIPTING ELEMENTS

There are three types of scripting elements:

1. scriptlet tag
2. expression tag
3. declaration tag

1. scriptlet tag

In JSP, **java code can be written** inside the jsp page using the scriptlet tag.

Syntax:

```
<%
        ……..……
    ……. …….
%>
```

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);
        %>

  </body>
  </html>
```

2. JSP expression tag

The code placed within JSP expression **tag is written to the output stream** of the response.

Syntax of JSP expression tag:

```
<%= statement %>
```

Example


```
<html>
        <body>
        Expression: <%= java.util.Calendar.getInstance().getTime() %>
        </body>
</html>
```

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>

        <%= "Welcome "+request.getParameter("uname") %>

   </body>
   </html>
```

3. JSP Declaration Tag

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

The syntax of the declaration tag is as follows:

```
<%!  field or method declaration %>
```

Example: index.jsp

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

JSP Action Tag

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

The JSP action tags are given below:

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

FORWARD ACTION AND PARAM TAG:

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

Syntax of jsp:forward action tag:

```
<jsp:forward page="relativeURL">
<jsp:param name="parametername" value="parametervalue" />
</jsp:forward>
```

OR

```
<jsp:forward page="relativeURL“/>
```

Example

index.jsp

```
<%@page contentType="text/html"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
        <title>JSP Page</title>
  </head>
  <body>
    <h1>This page contains forward</h1>
    <jsp:forward page="second.html"/>
  </body>
</html>
```

second.html

```
<html>
  <head>
    <title>TODO supply a title</title>
</head>

 <body>
    <h1>You are in a forwarded page</h1>
  </body>
</html>
```

INCLUDE ACTION

Using the <jsp:include> action we can **include different files in current jsp page.**

Example: (files = index.html , newjsp.jsp, includejsp.jsp)

index.html

```
<!DOCTYPE html>
<html>
   <head>
      <title>TODO supply a title</title>
   </head>
   <body>
      <h3>This is a html page</h3>
   </body>
</html>
```

Example

Example: (files = index.html , newjsp.jsp, includejsp.jsp)

newjsp.jsp

```
<%@page contentType="text/html"  pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
   <head> <title>JSP Page</title> </head>
   <body>
      <%--returns hour--%>

      <br> Time is <%= new java.util.Date().getHours() %>
      <%--returns min--%>
      : <%= new java.util.Date().getMinutes()%>
      <%--returns sec--%>
      : <%= new java.util.Date().getSeconds()%>

   </body>
</html>
```

Example

Example: (files = index.html , newjsp.jsp, includejsp.jsp)

includejsp.jsp

```
<%@page contentType="text/html"  pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
   <head>
    <title>JSP Page</title>
   </head>
   <body>
      <h1>This page contains include action</h1>

      <jsp:include page="index.html"/>
      <jsp:include page="newjsp.jsp"/>

   </body>
</html>
```

jsp:useBean action tag

The jsp:useBean action tag is used **to locate or instantiate a bean class.**

If bean object of the Bean class is already created, it doesn't create the bean depending on the scope.

But if object of bean is not created, it instantiates the bean.

```
<jsp:useBean id= "instanceName" scope= "page | request | session | appli
cation"   class= "packageName.className" " >
</jsp:useBean>
```

jsp:setProperty and jsp:getProperty action tags

Marwadi education foundation

The setProperty and getProperty action tags are used for developing web application with Java Bean.

Example (Files : index.html , process.jsp, user.java)

index.html

```
<form action="process.jsp" method="post">
Name:<input type="text" name="name"><br>
Password:<input type="password" name="password"><br>
Email:<input type="text" name="email"><br>
<input type="submit" value="register">
</form>
```

process.jsp

```
<jsp:useBean id="u" class="org.sssit.User"></jsp:useBean>
<jsp:setProperty property="*" name="u"/>

Record:<br>
<jsp:getProperty property="name" name="u"/><br>
<jsp:getProperty property="password" name="u"/><br>
<jsp:getProperty property="email" name="u" /><br>
```

Example

User.java

```
package org.sssit;

public class User{
private String name,password,email;

    public void setName(String name){
        this.name=name;
    }
    public String getName(){
        return name;
    }

    public void setPassword(String Password){
        this.password=password;
    }
    public String getPassword(){
        return password;
    }

    public void setEmail(String Email){
        this.email=email;
    }
    public String getEmail(){
        return email;
    }

}
```

JSP implicit Objects

The implicit objects are **pre-defined variable** used to access request and application data. JSP implicit objects are:

Object	Class/Interface	Meaning
out	JspWriter	It provides method related to I/O
request	HttpServletRequest	It provides method for accessing information made by current request
response	HttpServletResponse	It provides method for sending information
config	ServletConfig	For creating config object
application	ServletContext	For creating context object
session	HttpSession	This variable is used for accessing current client session
exception	Throwable	Used for handling Errors

JSP Session Objects

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 files required → index.html, welcome.jsp , second.jsp

index.html

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

JSP Session Example

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>
```

JSP Session Example

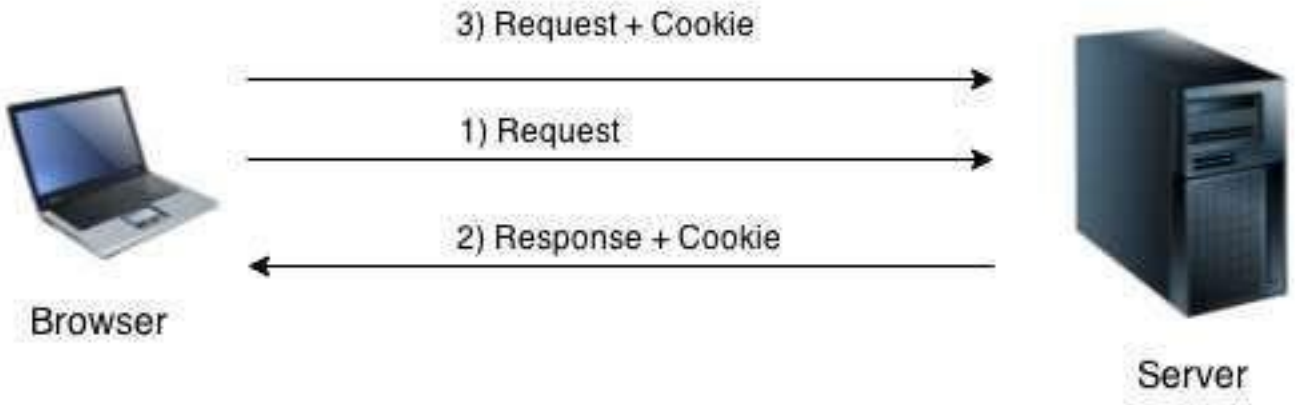
second.jsp

```
<html>
<body>
<%

String name=(String)session.getAttribute("user");
out.print("Hello "+name);

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

JSP Cookie Handling

By default, each request is considered as a new request. In cookies technique, we add cookie with response from the Server. So cookie is stored in the cache of the browser.

After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.



JSP Cookie Handling

Marwadi education foundation

Cookies are small text file that are stored in the client computer. These are basically used to keep track of user who browse the web.

The info stored in the cookie are generally name, age , id, city and so on.
The servlet container sends a set of cookies to the web browser. The browser stores the cookies on the local machine and makes use of this information next time when the browser is browsing the web.

Cookies are usually set in HTTP header.

Advantage of Cookies:

- ✓ Simplest technique of maintaining the state.
- ✓ Cookies are maintained at client side.

Disadvantage of Cookies:

- ✓ It will not work if cookie is disabled from the browser.

Example: to create, delete and read cookies

Files needed → index.html, createcookie.jsp, readcookie.jsp, deletecookie.jsp

index.html

```
<html>
  <head>
    <title>TODO supply a title</title>
  </head>
  <body>

    <form action="createCookie.jsp" method="post">
        USERNAME:<input type="text" name="name"><br>
        CITY:<input type="text" name="city"><br>
        <input type="submit" value="submit">
    </form>

 </body>
</html>
```

Cookie in JSP

createCookie.jsp

```
<body>
    <%
        String name = request.getParameter("name");
        String city = request.getParameter("city");

        Cookie nameCookie = new Cookie("name", name);  //creating cookie object
        Cookie cityCookie = new Cookie("city", city);      //creating cookie object

        response.addCookie(nameCookie);   //adding cookie in the response
        response.addCookie(cityCookie);       //adding cookie in the response

        nameCookie.setMaxAge(60*60*24);       //changing the maximum age
        cityCookie.setMaxAge(60*60*24);       //changing the maximum age
    %>
    <a href="readCookie.jsp" >Click here</a> to read the cookies
</body>
```

Example

readCookie.jsp

```
<body>

 <h3>Reading the cookie</h3>
     <%
     Cookie [] cookies = request.getCookies();          //return all the cookies from the browser
     for(int i =0;i<cookies.length;i++)
    {                                                   //printing name and value of cookie
        out.println("Cookie_name"+cookies[i].getName()+"<br>");
        out.println("Cookie_value"+cookies[i].getValue()+"<br>");
     }
        %>
        <a href="deletecookies.jsp">Click here</a>To delete the cookie..!!
  </body>
```

Example

deleteCookie.jsp

```
<body>

    <%
    Cookie nameCookie = new Cookie("name", "");
    nameCookie.setMaxAge(0);
    nameCookie.setValue("");
    response.addCookie(nameCookie);

    Cookie cityCookie = new Cookie("city", "");
    cityCookie.setMaxAge(0);
    cityCookie.setValue("");
    response.addCookie(cityCookie);
    %>

    <a href="readCookie.jsp">Click here</a> to check the deletion..!!
</body>
```

JSP Expression Language

The **Expression Language (EL)** simplifies the accessibility of data stored in the **Java Bean component**, and other objects like **request, session, application** etc.

There are many implicit objects, operators and reserve words in EL.

Syntax:

```
${ expression }
```

JSP Expression Language

Implicit Objects	Usage
pageScope	it maps the given attribute name with the value set in the page scope
requestScope	it maps the given attribute name with the value set in the request scope
sessionScope	**it maps the given attribute name with the value set in the session scope**
applicationScope	it maps the given attribute name with the value set in the application scope
param	**it maps the request parameter to the single value**
paramValues	it maps the request parameter to an array of values
header	it maps the request header name to the single value
headerValues	it maps the request header name to an array of values
cookie	**it maps the given cookie name to the cookie value**
initParam	it maps the initialization parameter
pageContext	it provides access to many objects request, session etc.

EL Param Example

In this example, we have created two files index.jsp and process.jsp. The index.jsp file gets input from the user and sends the request to the process.jsp which in turn prints the name of the user using EL.

index.jsp

```
<form action="process.jsp">
Enter Name:<input type="text" name="name" />
<br/><br/>
<input type="submit" value="go"/>
</form>
```

process.jsp

```
Welcome, ${ param.name }
```

EL sessionScope example

In this example, we printing the data stored in the session scope using EL. For this purpose, we have used sessionScope object.

index.jsp

```
<h3>welcome to index page</h3>
<%
session.setAttribute("user", "RK Keynotes!!");
%>

<a href="process.jsp">visit</a>
```

process.jsp

```
Value is ${ sessionScope.user }
```

EL cookie example

index.jsp

```
<h1>First JSP</h1>
<%
Cookie ck=new Cookie("name","RK Keynotes!");
response.addCookie(ck);
%>
<a href="process.jsp">click</a>
```

process.jsp

```
Hello, ${cookie.name.value }
```

JSP DATABASE ACCESS

Marwadi education foundation

index.html

STUDENT FORM:

ROLL No: 2

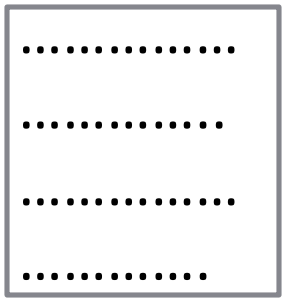
FIRST NAME: AMIT

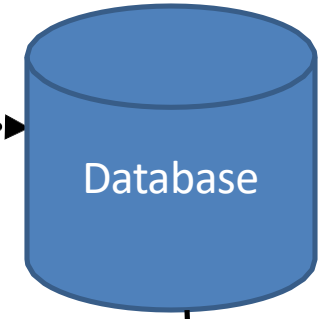
LAST NAME: AYALANI

ADDRESS: UJJAIN

REGISTER

newjsp.jsp





Web Browser

ROLL NO	FIRST NAME	LAST NAME	ADDRESS
1	Saurabh	Shrivastava	R-67
2	AMIT	AYALANI	UJJAIN

Example : JSP-JDBC

Files needed → index.html, newjsp.jsp

index.html

```
<h1>STUDENT FORM:</h1>
    <form action="newjsp.jsp" method="GET">
    <table>
      <tr>
        <td>ROLL No:</td><td><input type="text" name="rollno" ></td>
      </tr>
      <tr>
        <td>FIRST NAME:</td><td><input type="text" name="fname" ></td>
      </tr>
      <tr>
        <td>LAST NAME:</td><td><input type="text" name="lname" ></td>
      </tr>
      <tr>
        <td>ADDRESS:</td><td><input type="text" name="addr" ></td>
      </tr>
      <tr>
        <td></td><td><input type="submit" value="REGISTER" ></td>
      </tr>
          </table>
    </form>
```

Example

newjsp_1.jsp

```
<%@page language="java" import="java.sql.*" %>
<%@page import="java.io.*" %>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html>
   <body>
      <table>
         <tr>
            <td><b>ROLL NO</b></td>
            <td><b>FIRST NAME</b></td>
            <td><b>LAST NAME</b></td>
            <td><b>ADDRESS</b></td>
         </tr>
```

Example

<%

```
//read request parameter
String rollno = request.getParameter("rollno");
String fname = request.getParameter("fname");
String lname = request.getParameter("lname");
String addr = request.getParameter("addr");

   //initialize variable of driver

String driverClassName="com.mysql.jdbc.Driver";
String url="jdbc:mysql://localhost:3306/mu";
String user="root";
String pwd="";

 //register jdbc driver or load driver
Class.forName(driverClassName).newInstance();
```

Example


```
//Open connection
        Connection con=DriverManager.getConnection(url,user,pwd);

 //making a prepared statement
        PreparedStatement ps = con.prepareStatement("insert into student
values(?,?,?,?)");

        ps.setString(1,  rollno);
        ps.setString(2, fname);
        ps.setString(3, lname);
        ps.setString(4, addr);
        int p = ps.executeUpdate();
        PreparedStatement st = con.prepareStatement("select * from
student");
```

Example

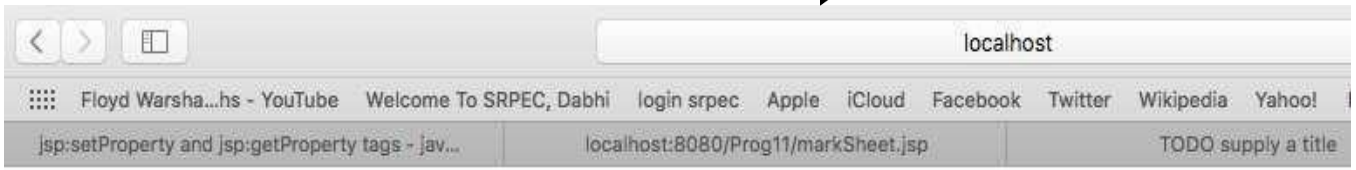
```
ResultSet rs= st.executeQuery();
        while(rs.next()){
        %>
<tr>
            <td><%= rs.getString("rollno") %></td>
            <td><%= rs.getString("fname") %></td>
            <td><%= rs.getString("lname") %></td>
            <td><%= rs.getString("addr") %></td>
        </tr>
        <%
        }
            out.print("</table>");
            rs.close();
            ps.close();
            st.close();
            con.close();
        %>
  </table>   </body> </html>
```

Exercise

Write a JSP to display semester mark sheet. Give necessary file to deploy it.

STUDENT MARKSHEET

Enter Seat No: 111 Submit



MARK SHEET

SEAT NO	NAME	SUBJECT 1	SUBJECT 2	SUBJECT 3	STATUS
111	AAA	40	50	60	pass

JSP Standard Tag Library (JSTL)

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

Advantage of JSTL:

Fast Developement: JSTL provides many tags that simplifies the JSP.

Code Reusability: We can use the JSTL tags in various pages.

No need to use scriptlet tag: It avoids the use of scriptlet tag.

JSTL Tags

Tag Name	Description
Core tags	The JSTL core tag provide variable support, URL management, flow control, etc. The URL for the core tag is **http://java.sun.com/jsp/jstl/core**. The prefix of core tag is **c**.
Function tags	The functions tags provide support for string manipulation and string length. The URL for the functions tags is **http://java.sun.com/jsp/jstl/functions** and prefix is **fn**.
Formatting tags	The Formatting tags provide support for message formatting, number and date formatting, etc. The URL for the Formatting tags is **http://java.sun.com/jsp/jstl/fmt** and prefix is **fmt**.
XML tags	The XML tags provide flow control, transformation, etc. The URL for the XML tags is **http://java.sun.com/jsp/jstl/xml** and prefix is **x**.
SQL tags	The JSTL SQL tags provide SQL support. The URL for the SQL tags is **http://java.sun.com /jsp/jstl/sql** and prefix is **sql**.

JSTL Core Tags

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.
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:forTokens	It iterates over tokens which is separated by the supplied delimeters.
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.

JSTL Core <c:out> Tag

The <c:out> tag **is similar to JSP expression tag.** It will display the result of an expression, similar to the way **< %=...% >** work.

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

JSTL Core <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 JavaBean.

```
<%@ 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>
```

JSTL Core <c:remove> Tag

It is used for removing the specified variable from a particular 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>
```

JSTL Core <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.

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

JSTL Core <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.

```
<%@ 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} </p>
</c:if>

</body>
</html>
```

JSTL Core <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 & <when > 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.

Example

```
<%@ 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 Core <c:forEach> Tag

The <c:for each > is **an iteration tag u**sed for repeating the nested body content for fixed number of times or over the collection.

```
<%@ 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>
```

Output:

Item 1

Item 2

Item 3

JSTL Core <c:redirect> Tag

The < c:redirect > tag **redirects the browser to a new URL**.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
        <html>
        <head>
        <title>Core Tag Example</title>
        </head>
        <body>
          <c:set var="url" value="0" scope="request"/>
          <c:if test="${url<1}">
        <c:redirect url="http://www.marwadieducation.edu.in"/>
          </c:if>
          <c:if test="${url>1}">
            <c:redirect url="http://facebook.com"/>
          </c:if>
        </body>
        </html>
```

.

JSTL FUNCTION TAG LIBRARY

The JSTL function provides a number of standard functions, most of these functions are **common string manipulation** functions.

The syntax used for including JSTL function library in your JSP is:

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

JSTL Functions	Description
fn:contains()	It is used to test if an input string containing the specified substring in a program.
fn:containsIgnoreCase()	It is used to test if an input string contains the specified substring as a case insensitive way.
fn:endsWith()	It is used to test if an input string ends with the specified suffix.
fn:indexOf()	It returns an index within a string of first occurrence of a specified substring.
fn:trim()	It removes the blank spaces from both the ends of a string.
fn:startsWith()	It is used for checking whether the given string is started with a particular string value.
fn:split()	It splits the string into an array of substrings.
fn:toLowerCase()	It converts all the characters of a string to lower case.
fn:toUpperCase()	It converts all the characters of a string to upper case.
fn:substring()	It returns the subset of a string according to the given start and end position.
fn:substringAfter()	It returns the subset of string after a specific substring.
fn:substringBefore()	It returns the subset of string before a specific substring.
fn:length()	It returns the number of characters inside a string, or the number of items in a collection.
fn:replace()	It replaces all the occurrence of a string with another string sequence.

Example

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

    <%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix=
"fn" %>
<html>
<head>
<title>Using JSTL Function </title>
</head>
<body>
  <c:set var="str1" value="I LOVE INDIA"/>
  <c:set var="str2" value="${fn:toLowerCase(str1)}"/>
  <p><b>Original :</b>${str1}</p>
  <p><b>Conversion :</b>${str2}</p>
</body>
</html>
```

OUT PUT:

Original : I LOVE INDIA
Conversion : i love india

SQL TAG LIBRARY

The JSTL sql tags provide SQL support. The url for the sql tags is **http://java.sun.com/jsp/jstl/sql** and prefix is **sql**.

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

SQL Tags	Descriptions
sql:setDataSource	It is used for creating a simple data source suitable only for prototyping.
sql:query	It is used for executing the SQL query defined in its sql attribute or the body.
sql:update	It is used for executing the SQL update defined in its sql attribute or in the tag body.
sql:param	It is used for sets the parameter in an SQL statement to the specified value.
sql:dateParam	It is used for sets the parameter in an SQL statement to a specified java.util.Date value.
sql:transaction	It is used to provide the nested database action with a common connection.

Example

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

      <html>
      <head>
      <title>sql:setDataSource Tag</title>
      </head>
      <body>

      <sql:setDataSource var="db" driver="com.mysql.jdbc.Driver"
      url="jdbc:mysql://localhost/test"
          user="root"  password="1234"/>

      <sql:query dataSource="${db}" var="rs">
      SELECT * from Students;
      </sql:query>

      <table border="2" width="100%">
      <tr>
      <th>Student ID</th>
      <th>First Name</th>
      <th>Last Name</th>
      <th>Age</th>
      </tr>
```

Example

```
<c:forEach var="table" items="${rs.rows}">
    <tr>
    <td><c:out value="${table.id}"/></td>
    <td><c:out value="${table.First_Name}"/></td>
    <td><c:out value="${table.Last_Name}"/></td>
    <td><c:out value="${table.Age}"/></td>
    </tr>

    </c:forEach>
    </table>
    </body>
    </html>
```

JSP XML tags

The **XML tags are useful for creating and manipulating the XML documents through the JSP.**

The **url** for the xml tags is **http://java.sun.com/jsp/jstl/xml** and **prefix is x.**

The syntax used for including JSTL XML tags library in your JSP is:

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

XML Tags	Descriptions
x:out	Similar to <%= ... > tag.
x:parse	It is used for parse the XML data specified either in the tag body or an attribute.
x:set	It is used to sets a variable to the value of an XPath expression.
x:choose	It is a conditional tag that establish a context for mutually exclusive conditional operations.
x:when	It is a subtag of that will include its body if the condition evaluated be 'true'.
x:otherwise	It is subtag of that follows tags and runs only if all the prior conditions evaluated be 'false'.
x:if	It is used for evaluating the test XPath expression and if it is true, it will processes its body content.
x:transform	It is used in a XML document for providing the XSL(Extensible Stylesheet Language) transformation.
x:param	It is used along with the transform tag for setting the parameter in the XSLT style sheet.

Formatting

Formatting Tags	Descriptions
fmt:parseNumber	It is used to Parses the string representation of a currency, percentage or number.
fmt:timeZone	It specifies a parsing action nested in its body or the time zone for any time formatting.
fmt:formatNumber	It is used to format the numerical value with specific format or precision.
fmt:parseDate	It parses the string representation of a time and date.
fmt:bundle	It is used for creating the ResourceBundle objects which will be used by their tag body.
fmt:setTimeZone	It stores the time zone inside a time zone configuration variable.
fmt:setBundle	It loads the resource bundle and stores it in a bundle configuration variable or the named scoped variable.
fmt:message	It display an internationalized message.
fmt:formatDate	It formats the time and/or date using the supplied pattern and styles.

JSP Custom Tags

Tags are made available within a JSP page *via* the **`taglib`** directive:

```
<%@ taglib uri="uri" prefix="prefix" %>
```

Directive's **`uri`** attribute references the TLD (established via WAR file's **`web.xml`**)

Directive's **`prefix`** attribute provides a local namespace for the TLD's tags

Steps for implementing, using & deploying custom tags

The steps you follow in order to implement and deploy custom tags are relatively straight-forward.

First, you write tag handlers. Under JSP 2.0 architecture, tag handlers are Java classes.

Second, you write so called tag library descriptor, TLD file, in short.

Third, you package a set of tag handlers and TLD file into what is called tag library in either unpacked or packed form.

Then, you write JSP pages that use these tags. Then you deploy the tag library along with JSP pages.
Now let's talk about 3 things here by using examples - tag handler, TLD file, and JSP pages.

JSP Custom Tag Example: Create "Hello" Tag

To write a custom tag you can simply extend SimpleTagSupport **class** and override the **doTag()** method, where you can place your code to generate content for the tag.

Consider you want to define a custom tag named <ex:Hello> and you want to use it in the following fashion without a body:

<ex:Hello />

To create a custom JSP tag, you must first create a Java class that acts as a tag handler.

JSP Custom Tag Example: Create "Hello" Tag

So let us create **HelloTag** class as follows:

```
package myTags;

import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;  import java.io.*;

public class HelloTag extends SimpleTagSupport  {

 public void doTag() throws JspException, IOException {
  JspWriter out = getJspContext().getOut();
  out.println("Hello Custom Tag!");
 }
}
```

JSP Custom Tag Example: Create "Hello" Tag

Above code has simple coding where doTag() method takes the current JspContext object using getJspContext() method and uses it to send "Hello Custom Tag!" to the current JspWriter object.

Let us compile above class and copy it in a directory available in environment variable CLASSPATH. Here we have copied it to ROOT\WEB-INF\classes\myTags

Note : You need to include the following to compile:

TOMCAT_HOME/lib/jsp-api.jar

to your CLASSPATH

Finally create following tag library file: <Tomcat-Installation-Directory> webapps\ROOT\WEB-INF\custom.tld.

JSP Custom Tag Example: Create "Hello" Tag


```
<taglib>
 <tlib-version>1.0</tlib-version>
 <jsp-version>2.0</jsp-version>
 <short-name>Example TLD</short-name>
 <tag>
  <name>Hello</name>
  <tag-class>myTags.HelloTag</tag-class>
  <body-content>empty</body-content>
 </tag>
</taglib>
```

JSP Custom Tag Example: Create "Hello" Tag

Now it's time to use above defined custom tag **Hello** in our JSP program(Hello.jsp) as follows: (Note : Hello.jsp should also be saved in WEB-INF folder)

```
<%@ taglib prefix="ex"  uri="WEB-INF/custom.tld"%>
<html>
 <head>
  <title>A sample custom tag</title>
 </head>
 <body>
  <ex:Hello/>
 </body>
</html>
```

Start Tomcat server and Try to call above JSP in a browser using:
http://localhost:8080/Hello.jsp

This should produce following result:

Hello Custom Tag!

Summary

- Introduction to JSP
- Comparison with Servlet
- JSP Architecture
- JSP: Life Cycle
- Scripting Elements
- Directives
- Action Tags
- Implicit Object
- Expression Language(EL)
- JSP Standard Tag Libraries(JSTL)
- Custom Tag
- Session Management
- Exception Handling
- CRUD Application

Up Next??

- Introduction to Hibernate
- Exploring Architecture of Hibernate
- Object Relation Mapping(ORM) with Hibernate
- Hibernate Annotation
- Hibernate Query Language (HQL)
- CRUD Operation using Hibernate API

END OF UNIT - 4