CS -29 MAJOR - 11
Advance Java Programming(J2EE)

Presented by : Dhruvita Savaliya

UNIT 3- JSP

Topics of JSP

- Introduction to JSP and JSP Basics
- JSP vs. Servlet
- JSP Architecture
- Life cycle of JSP
- JSP Elements:
  - Directives Elements (page, include, taglib)
  - Scripting Elements (Declaration, scriptlet, expression)
  - Action Elements (jsp:param, jsp:include, jsp:Forward, jsp:plugin, jsp:useBean, jsp:setAttribute, jsp:getAttribute)
- JSP Implicit Objects (request, response, out, session, application, pagecontext)
- JSP Scope
- Including and Forwarding from JSP Pages
  - include Action
  - forward Action
- Working with Session & Cookie in JSP
- Error Handling and Exception Handling with JSP
- JSP EL (Expression Language), JSP Standard Tag Libraries (JSTL)

Introduction :

- JSP is Java Server Pages (JSP) technology enables you to mix regular, static HTML with dynamically generated content.
- JSP are run in a server-side component known as JSP container which translates them into equivalent Java Servlet.
- Simply write the regular HTML in the normal manner, using familiar Web-page-building tools.
- Then enclose the code for the dynamic parts in special tags, most of which start with <% and end with %>.

➢ **JSP typically comprised of:**

- Static HTML/XML components
- Special JSP tags.
- Code written in the java language called" script less".

- **How is JSP More Advantage than Servlets?**
- JSP simplifies web development by combining the strengths of Java with the flexibility of HTML. Some advantages of JSP over Servlets are listed below:
- JSP code is easier to manage than Servlets as it separates UI and business logic.
- JSP minimizes the amount of code required for web applications.
- Generate content dynamically in response to user interactions.
- It provides access to the complete range of Java APIs for robust application development.
- JSP is suitable for applications with growing user bases.

- **Key Features of JSP**
- It is platform-independent; we can write once, run anywhere.
- It simplifies database interactions for dynamic content.
- It contains predefined objects like request, response, session and application, reducing development time.
- It has built-in mechanisms for exception and error management.
- It supports custom tags and tag libraries.

Comparison of JSP :

1. **JSP vs. Active Server Pages (ASP) :** The advantages of JSP are twofold. First, the dynamic part is written in Java, not Visual Basic or other MS specific language, so it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft Web servers.
2. **JSP vs. Pure Servlets :** It is more convenient to write (and to modify!) regular HTML than to have plenty of println statements that generate the HTML.
3. **JSP vs. Server-Side Includes (SSI) :** SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.
4. **JSP vs. JavaScript :** JavaScript can generate HTML dynamically on the client but can hardly interact with the web server to perform complex tasks like database access and image processing etc.
5. **JSP vs. Static HTML :** Regular HTML, of course, cannot contain dynamic information.

- **Advantages of JSP :**
- **Nobody can borrow the code :**
- The JSP code written and runs and remains on the server. So, issue of copy source code does not arise at all. All of JSP's functionality is handled before the page is sent to browser.
- **Faster loading of pages :**
- With JSP, decision can be made about what user want to see at web server prior to pages being dispatched. So only the content that the user is interested will be dispatched to the user. There is no extra code and extra content.
- **No browser compatibility Issues :**
- JSP pages can run same way in browser. The developer ends up sending standard HTML to a user browser. This largely eliminates scripting issues and cross browser compatibility.
- **JSP support :**
- JSP is supported by number of web server like Apache, Microsoft IIS and PWS, Netscape's Fast Tracks and Enterprise web server and others. Built in support for JSP is available Java Server from Sun Micro system.

- **Compilation :**
- JSP compiled before the web server processes it. This allows the server to handle JSP pages much faster, because in the older technologies such as CGI require the server to load an interpreter and the target script each time the page is requested.
- **JSP elements in HTML/XML pages :**
- JSP page look like HTML /XML page, it holds text marked with a collection of tags. While a regular JSP page is not a valid XML page, there is a variant JSP tag syntax that lets the developer use JSP tags within XML documents.

Disadvantages of JSP :

- **Attractive Java Code**
- Putting java code within web page is really bad design, but JSP makes it tempting to do just that. Avoid this as far as possible. It is done using template using.
- **Java Code required**
- To relatively simple things in JSP can actually demand putting java code in a page.
- **Simple task are hard to code**
- Even including page headers and footers is a bit difficult with JSP.
- **Difficult looping in JSP**
- In regular JSP pages looping is difficult. In advance JSP we can use some custom tags for looping.
- **Occupies a lot of space.**
- JSP consumes extra hard drive and memory space.

Difference Between Servlet & JSP :

- **Servlet** technology is used to create a web application. A **servlet** is a Java class that is used to extend the capabilities of servers that host applications accessed by means of a request-response model. Servlets are mainly used to extend the applications hosted by web services.

- **JSP** is used to create web applications just like **Servlet** technology. A **JSP** is a text document that contains two types of text: static data and dynamic data. The static data can be expressed in any text-based format (like HTML, XML, SVG, and WML), and the dynamic content can be expressed by JSP elements.

- **When to Use Servlet or JSP**
- **Servlet:** When you need to handle business logic, form processing, and request management.
- **JSP:** When you need to build dynamic web pages with UI elements and embed minimal Java logic.

Servlet	JSP
Servlet is a Java code.	JSP is an HTML-based compilation code.
Writing code for servlet is harder than JSP as it is HTML in java.	JSP is easy to code as it is java in HTML.
Servlet plays a controller role in the ,MVC approach.	JSP is the view in the MVC approach for showing output.
Servlet is faster than JSP.	JSP is slower than Servlet because the first step in the JSP lifecycle is the translation of JSP to java code and then compile.
Servlet can accept all protocol requests.	JSP only accepts HTTP requests.
In Servlet, we can override the service() method.	In JSP, we cannot override its service() method.
In Servlet by default session management is not enabled, user have to enable it explicitly.	In JSP session management is automatically enabled.
In Servlet we have to implement everything like business logic and presentation logic in just one servlet file.	In JSP business logic is separated from presentation logic by using JavaBeansclient-side.

Servlet	JSP
Modification in Servlet is a time-consuming compiling task because it includes reloading, recompiling, JavaBeans and restarting the server.	JSP modification is fast, just need to click the refresh button.
There is no method for running JavaScript on the client side in Servlet.	While running the JavaScript at the client side in JSP, client-side validation is used.
Packages are to be imported on the top of the program.	Packages can be imported into the JSP program (i.e, bottom , middleclient-side, or top)
It can handle extensive data processing.	It cannot handle extensive data processing very efficiently.
The facility of writing custom tags is not present.	The facility of writing custom tags is present.
Servlets are hosted and executed on Web Servers.	Before the execution, JSP is compiled in Java Servlets and then it has a similar lifecycle as Servlets.
It does not have inbuilt implicit objects.	In JSP there are inbuilt implicit objects.

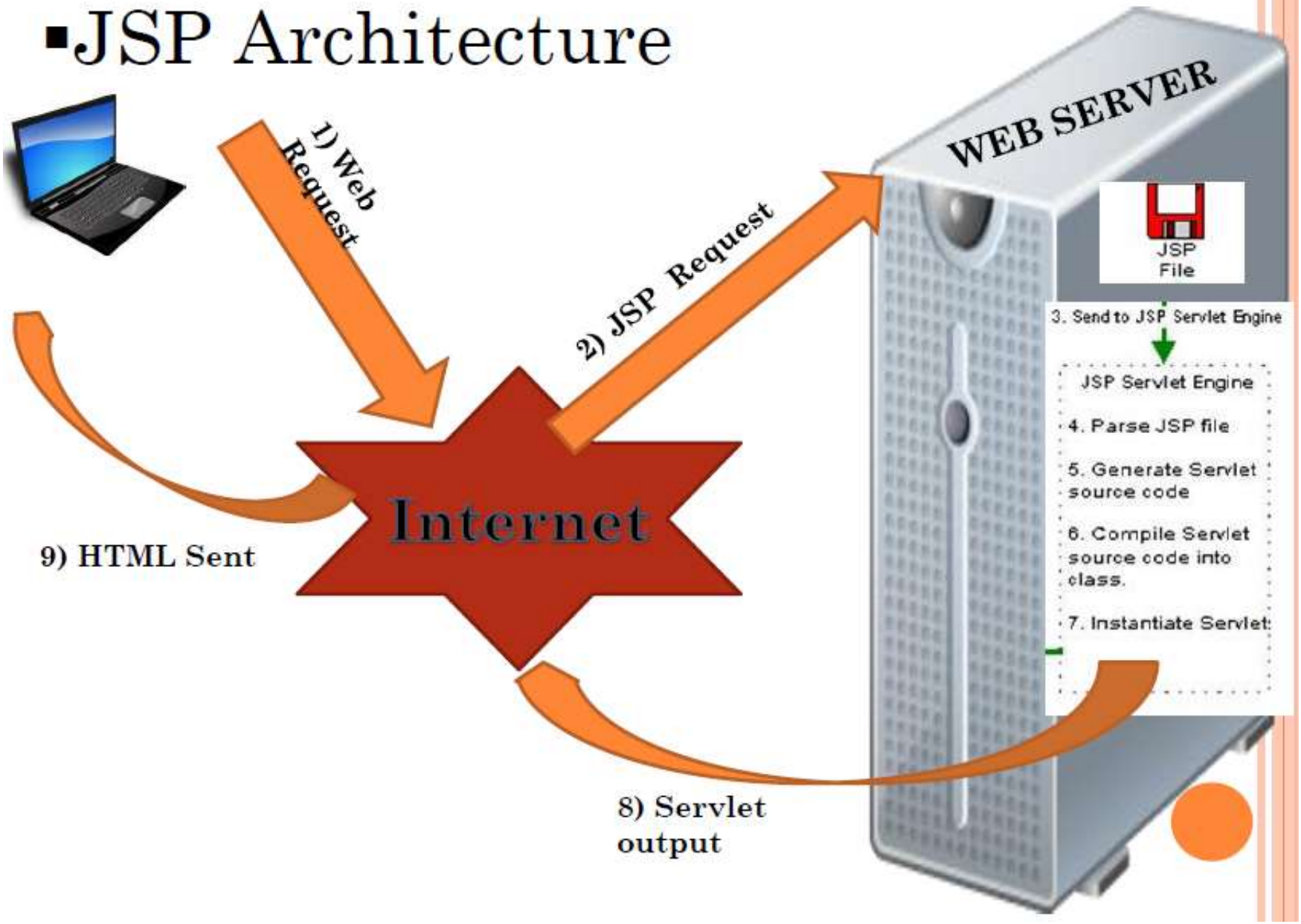
JSP Architecture :

- JSP (JavaServer Pages) architecture is a framework for building web applications in Java.
- It has three main parts: a web container, a JSP engine, and a servlet container.
- The JSP engine processes requests from the user and generates dynamic content using Java code and JSP tags.
- The servlet container manages the lifecycle of servlets and JSP pages.
- JSP are built by SUN Micro systems Servlet technology.
- JSP tag contains java code and its file extension is JSP.

- **Steps for JSP Request:**

1. User requesting a JSP page through internet via web browser.
2. The JSP request is sent to the WebServer.
3. Webserver accepts the requested. Jsp file and passes the JSP file to the JSP Servlet Engine.
4. If the JSP file has been called the first time then the JSP file is parsed other wise servlet is instantiated.
5. The next step is to generate a servlet from the JSP file. In that servlet file, all the HTML code is converted in println statements.
6. Now, The servlet source code file is compiled into a class file (byte code file)
7. The servlet is instantiated by calling the init and service methods of the servlet's life cycle.
8. Now, the generated servlet output is sent via the Internet form webserver to user's web browser.
9. Now in last step, HTML results are displayed on the user's web browser.

JSP Architecture

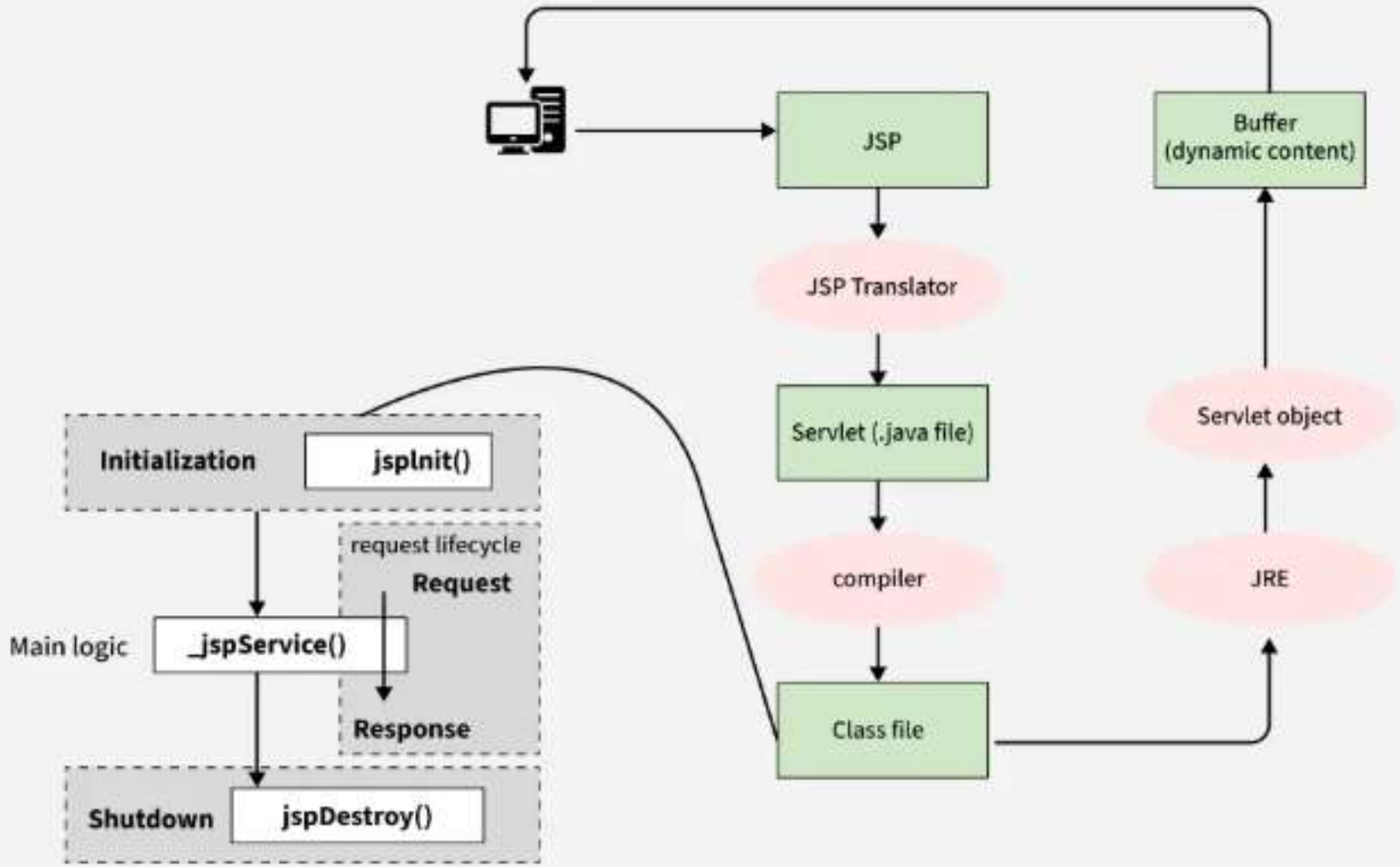


JSP Life Cycle :

- When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page. If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.
- The compilation process involves three steps:

1. Parsing the JSP.
2. Turning the JSP into a servlet.
3. Compiling the servlet.

Life Cycle of JSP



- **Steps of JSP Life Cycle**

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 is invoked by the container)
6. Request processing(_jspService()is invoked by the container)
7. JSP Cleanup (jspDestroy() method is invoked by the container)

- We **can** override jspInit(), jspDestroy() but we **can't** override _jspService() method.

- **1. Translation of JSP to Servlet**

  The JSP file is parsed and converted into a Java servlet source file (test.java).

  This step checks for syntax correctness.

  ```
  // The JSP is converted to a servlet class.
  public class TestServlet extends HttpServlet {
          // The generated servlet code
  }
  ```

- **2. Compilation of JSP page**

  The generated test.java file is compiled into a class file (test.class).

  This step converts the servlet code into bytecode.

  // JSP is automatically converted into a servlet, such as **public class TestServlet extends** HttpServlet { *// Generated servlet code here*

  ```
  }
  ```

- **3. Classloading**

  The container dynamically loads the compiled class.

- **4. Instantiation :**

The container creates an instance of the generated servlet class.

This instance handles multiple requests unless explicitly configured otherwise.

```
// The container creates an instance automatically.
TestServlet servlet = new TestServlet();
```

5. Initialization (jspInit()) :

This method is called only once when the JSP is first loaded.

It is used for initializing resources like database connections or configurations.

```
public void jspInit() {       // Initialization code, like setting up resources.
        System.out.println("JSP Initialized.");
}
```

- **6. Request Processing (_jspService()) :**

This method is called for every request.

It cannot be overridden because it is auto-generated and declared as final.

It receives HttpServletRequest and HttpServletResponse objects to handle the request.

```
public void _jspService(HttpServletRequest request, HttpServletResponse response) { // Code that handles the request, like generating HTML output.
        response.getWriter().write("<html><body>Hello,
                                        World!</body></html>");
}
```

7. JSP Cleanup (jspDestroy())

- This method is called once before removing the JSP from service.
- It is used for releasing resources, such as closing database connections or cleaning up open files.

```
public void jspDestroy() {
// Clean up resources like closing database connections.
System.out.println("JSP Destroyed.");
}
```

- This Lifecycle ensures that JSP pages are efficiently compiled, managed and cleaned up by the server container.

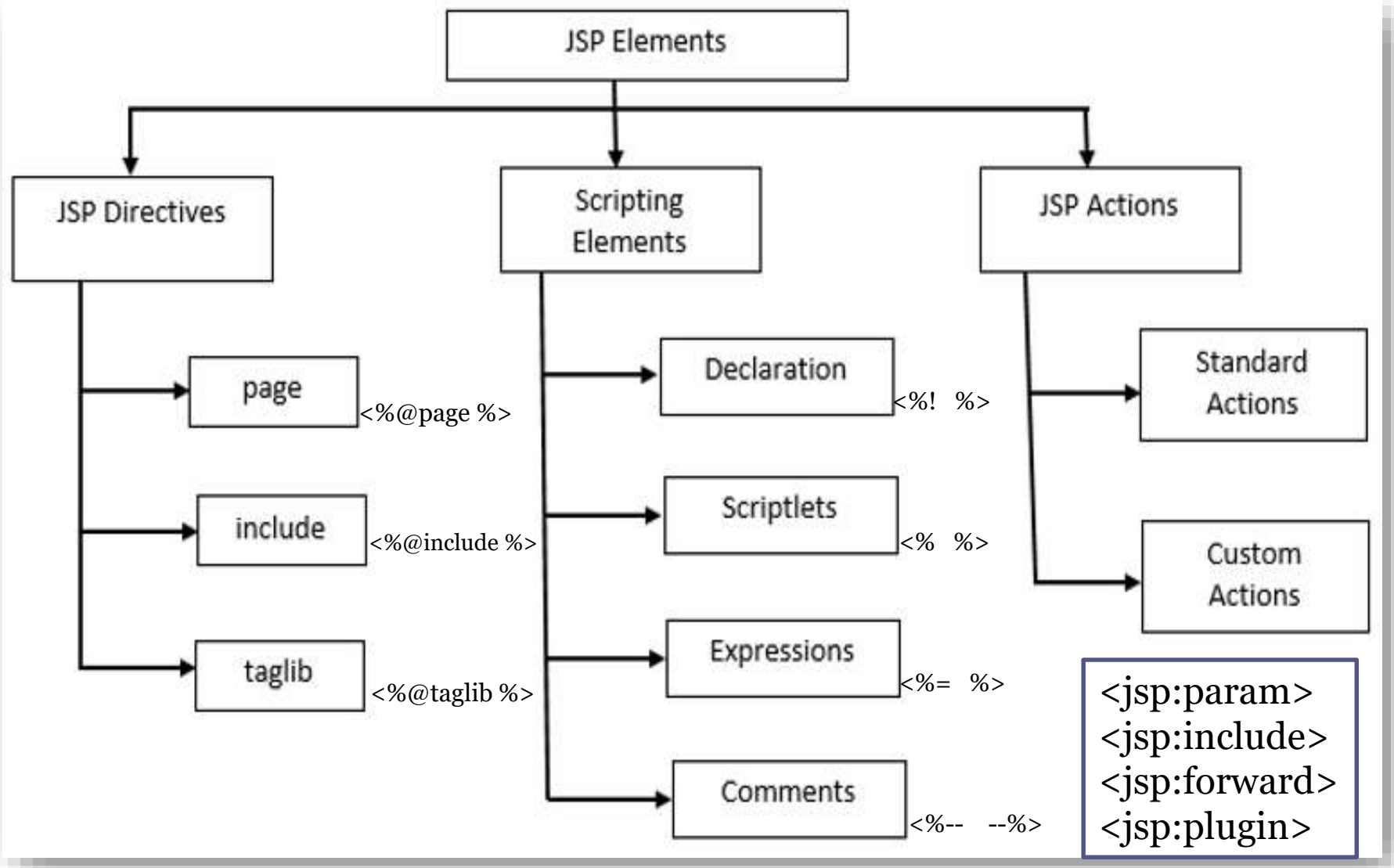
JSP Elements :

- JSP Element have a special identity to JSP compiler because it starts and ends with special kind of tags.
- Template (HTML) code is not compiled by the JSP compiler and also not recognized by the JSP container.
- It is known as **Component of JSPpages.**
- There are 3 basic JSP elements.

    1.Directive Element

    2.Scripting Element

    3.Action Element


JSP Elements
JSP Directives
Scripting Elements
JSP Actions
page
<%@page %>
include
<%@include %>
taglib
<%@taglib %>
Declaration
<%! %>
Scriptlets
<% %>
Expressions
<%= %>
Comments
<%-- --%>
Standard Actions
Custom Actions
<jsp:param>
<jsp:include>
<jsp:forward>
<jsp:plugin>

- 1.**Directive Element:**
- The role of the directives is to pass information to the JSP container.
- Following is the general syntax of Directive element.
- Syntax :-<%@ directive attribute= “value” %>
- There are three types of directives

  1.1.page directive

  1.2.include directive

  1.3.taglib directive

- **1.1. Page Directive:**
  Page directive is used to specify attributes for the whole JSP page.
- **Syntax:<%@ page attibute= “value” %>**
  Attributes of JSP page directive are as under :

1. extends=“class_name”
2. import=“import_list_package”
3. contentType=“content_info”
4. session=“true|false”
5. buffer=“auto flush”
6. autoFlush=“true|false”
7. IsThreadSafe=“true|false”
8. info=“Information”
9. errorPage=“error_file”
10. language=“scripting language”
11. IsErrorPage“true|false”
12. pageEncoding

- **Import Page Directive :**
- 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.
- Example of import attribute:

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

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

```
<BODY>
    <H2> The contentType Attribute </H2>
    <%@ page contentType="text/plain" %>
    This should be rendered in plain text,
    <B> not </B> as HTML.
</BODY>
```

- **errorPage Page Directive :**
  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.
  100/100=1 then 100/0=display error page
- **Example:**

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

- **1.2.Include Directive:**
- The include directive is used to include the contents of any resource it may be jsp file, html file or text file.
- **Advantageof Include directive Code Reusability.**
- **Syntax :-<%@ include file="relative path" %>**
- **Example:**
- **File1 header.html**

```
<body>
    <h1>header file </h1>
</body>
```

- **File 2 JSP file :**

```
<html>
    <body>
        <%@ include file="header.html" %>
        <h1>JSP FILE CODE </h1>
    </body>
</html>
```

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

  <%@ taglib uri="URI" prefix="pr" %>

  <pr:h1>

- **2.1.scriptlet tag:**
- A scriptlet tag is used to execute java source code in JSP.
- You can Declare a variable, use looping, statements use if-else, use switch-case etc in scriptlet tag.
- **Syntax:**

```
<%java source code%>
<html>
   <body>
      <%
         int a=8;
    out.print(“java code here”+a);
    out.print "Welcome"+request.getParameter(“txt_nm"));
      %>
   </body>
</html>
```

- **2.2.Expression tag:**
- The code placed within **JSP expression tag is *written to the output stream of the response.***
- So you need not write out.print() to write data. It is mainly used to print the values of variable or method.
- **Syntax:**
  <%=statement%>

Example :

```
<html>
   <body>
      <%= “Expression here ” %>
      <%= "Welcome "+request.getParameter(“txt_name")
   %>
   </body>
</html>
```

- **2.3.Declaration Tag:**
- The **JSP declaration tag is used *to declare fields and methods*.**
- The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet.
- So it doesn't get memory at each request.
- **Syntax:**
  <%! field or method declaration %>
- **Example :**

```
<html>
<body>
    <%!
      int method()
      {
              return 10+20;
      }
    %>
    <%= method() %>
</body>
</html>
```

- **Difference between JSP Scriptlet tag and Declaration tag.**

Jsp Scriplet Tag	Jsp Declaration Tag
The jsp scriptlet tag can only declare variables not methods.	The jsp declaration tag can declare variables as well as methods.
The declaration of scriptlet tag is placed inside the _jspService() method.	The declaration of jsp declaration tag is placed outside the _jspService() method.

Action Elements :

- Action element are high level jsp element which are used to create ,modify and use other objects.
- Syntax of action element's tags in jsp tags is just like XML syntax:
- There are basic 4 types of action Element:

1. <jsp:param>
2. <jsp:include>
3. <jsp:forward>
4. <jsp:plugin>
5. <jsp:useBean>
6. <jsp:setAttribute>
7. <jsp:getAttribute>

- **3.1.<jsp:param>**
- The <jsp:param> action is used with other tags to provide additional information in the form of name value pairs.
- This action is used in conjunction with jsp:include, jsp:forwardand jsp:pluginactions.
- **Syntax :**

  <jsp:paramname=”parameter_name” value=”parameter_value” />
- **Example:**

  <jsp:paramname=”font_size” value=”20” />

- **3.2.<jsp:include> :**
- The jsp: include action tag is used to include the content of another resource it may be jsp, html or servlet.
- The jsp include action tag includes the resource at **request time** (runtime) so it is better for dynamic pages because there might be changes in future.

- Advantage of include action tag:
- **Code reusability:** We can use a page many times such as including header and footer pages in all pages. So it saves a lot of time.
- **Syntax:**
- <jsp:include page="relativeURL|<%=expression%>"/>

- **Example :**
- **first_file.jsp**

```
<jsp:include page="second_file.jsp"></jsp:include>
<body>
    <h1>hello from first</h1>
</body>
```

- **second_file.jsp**

```
<body>
    <h1>second</h1>
</body>
```

JSP include directive	JSP include action
includes resource at translation time.	includes resource at request time.
better for static pages.	better for dynamic pages.
includes the original content in the generated servlet.	calls the include method.

- **3.3.<jsp:forward>:**
- With the forward action, the current page stops processing the request and forwards the request to another page.
- Execution never returns to the calling (current) page.
- **Syntax :**
- <jsp:forwardpage= “file_name” />
- **First_page.jsp**

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

second_page.jsp

```
<%= "Hello second page" %>
```

- **Forward one page to other page with parameter First.jsp**

```
<jsp:forward page="second_page.jsp">
        <jsp:param name=“par_1" value="<%= 10+20 %>">
        </jsp:param>
 </jsp:forward>
```

second_page.jsp

```
<%= request.getParameter(" par_1 ") %>
```

- **3.4.<jsp:plugin>:**

The jsp:plugin action tag is used to embed applet in the jsp file. The jsp:plugin action tag downloads plugin at client side to execute an applet or bean.

- **Syntax:**

```
<jsp:plugin type="applet|bean“ code="nameOfClassFile“
codebase="directoryNameOfClassFile“> </jsp:plugin>
 <jsp:plugin type="applet" code="NewApplet.class"
    codebase=".">
          <jsp:fallback>
             <h2>sorry unable to open</h2>
          </jsp:fallback>
       </jsp:plugin>
```

Right click on source package→new→other→java→applet

NewApplet.java

```
import java.awt.*;
import javax.swing.JApplet;
public class NewApplet extends JApplet {
public void paint(Graphics g)
   {
      setBackground(Color.red);
      setForeground(Color.black);
      g.drawString("tybca", 100, 100);
   }
}
```

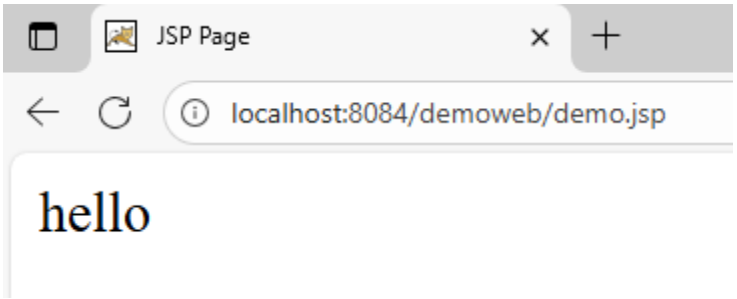
- **3.5 <jsp:useBean> :**
- JavaServer Pages provides the mechanism to interact with the Java objects called JavaBeans.
- The **useBean action tag** is used to instantiate and access JavaBeans within the JSP pages.
- This allows the developers to encapsulate the data and business logic in the Java classes and it can easily integrate them into the JSP pages.
- **useBean action tag in JSP**
- The **<jsp: useBean>** event tag in JSP can be used to create or retrieve a JavaBean instance and provides a way to use Java objects in JSP pages without the need for explicit Java code This tag is consumed Automatic handling of JavaBeans modeling and management, so that developers Java in JSP applications It is easier to work with objects.
- <jsp:useBean id = "name" class = "package.class" />
- Once a bean class is loaded, you can use **jsp:setProperty** and **jsp:getProperty** actions to modify and retrieve the bean properties.

- useBean with UDF create **jsp** file:

```
<jsp:useBean class="demopackage.myClass" id="obj"
   scope="application"></jsp:useBean>
      <%=obj.fun()%>
```

- Create package with name demopackage→create javaClass with name : **myClass.java**

```
package demopackage;
public class myClass {
    private String a;
    public String fun()
    {
       return "hello";
    }
}
```

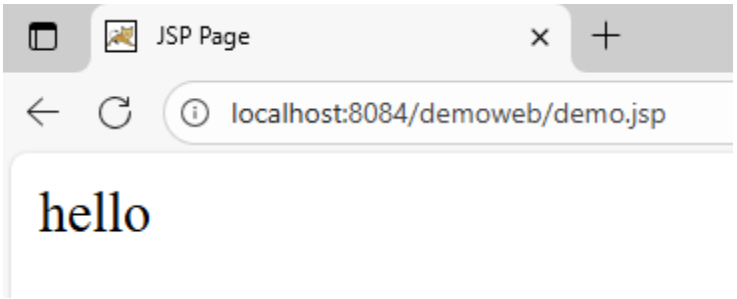


- **useBean with private Property jsp file :**

```
<jsp:useBean class="demopackage.bean_cls" id="obj"
   scope="application"> </jsp:useBean>
<jsp:setProperty name="obj" property="a" value="hello">
   </jsp:setProperty>
<jsp:getProperty name="obj" property="a"> </jsp:getProperty>
```

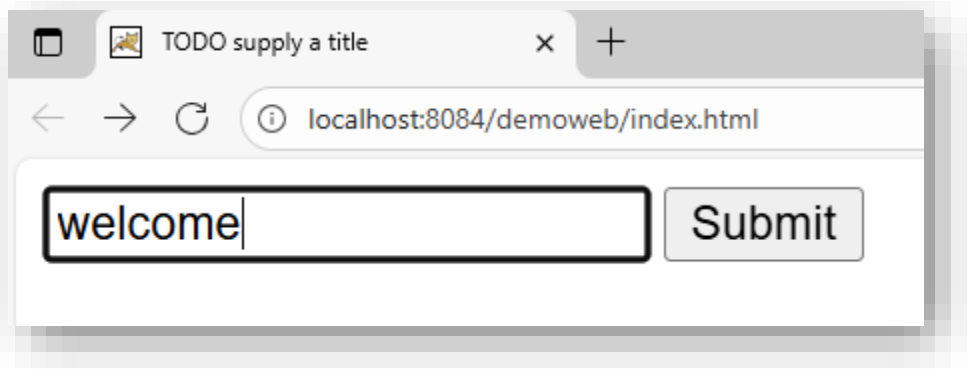
- **Bean_cls.java file (javaClass file)**

```
package demopackage;
public class bean_cls {
   private String a;
   public String getA() {
      return a;
   }
   public void setA(String a) {
      this.a = a;
   }
}
```



- **useBean with dynamic value with private property :**
- **Index.html file**

```
<form action="demo.jsp">
    <input type="text" name="a">
    <input type="submit">
</form>
```

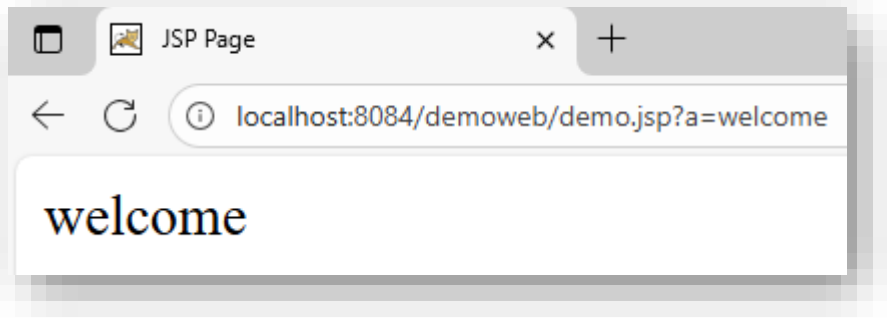


- **File.jsp**

```
<jsp:useBean class="demopackage.bean_cls" id="obj" scope="application"></jsp:useBean>
<jsp:setProperty name="obj" property="a" value="hello"> </jsp:setProperty>
<jsp:getProperty name="obj" property="a"> </jsp:getProperty>
```

- **Bean_cls.java file (javaClass file)**

```
package demopackage;
public class bean_cls {
   private String a;
   public String getA() {
      return a;
   }
   public void setA(String a) {
      this.a = a;
   }
}
```



Note:textbox name must be same as property name

Implicit Objects :

- There are some implicit object which container will provide to use directly in JSP these are called Implicit Object.
- There are 7 implicit Object in JSP Implicit Object

1. out
2. request
3. response
4. session
5. application
6. config
7. exception

- Content for 2 3 marks :
- **Out:** An instance of javax.servlet.jsp.JspWriter, used to send content to the client's browser. It acts as the output stream for the JSP page.
- **Request:** An instance of javax.servlet.http.HttpServletRequest, representing the HTTP request from the client. It provides access to request parameters, headers, cookies, and other client-side information.
- **response:** An instance of javax.servlet.http.HttpServletResponse, representing the HTTP response sent back to the client. It allows setting response headers, status codes, and redirecting the client.
- **Session:** An instance of javax.servlet.http.HttpSession, used to maintain user-specific data across multiple requests within a single session.
- **Application:** An instance of javax.servlet.ServletContext, representing the web application context. It provides access to application-wide resources and attributes.
- **Config:** An instance of javax.servlet.ServletConfig, providing access to initialization parameters for the specific JSP (which is translated into a Servlet).
- **pageContext:** An instance of javax.servlet.jsp.PageContext, providing access to all other implicit objects and managing the scope of attributes (page, request, session, application).
- **page:** A direct reference to the instance of the Servlet generated from the JSP page. This object is rarely used directly by developers.
- **Exception:** An instance of java.lang.Throwable, available only in error pages (where isErrorPage="true" in the page directive). It provides information about the exception that occurred.

1.Out :

- For writing any data to the buffer, JSP provides an implicit object named out.
- Itis the object of **JspWriter**.
- In case of servlet you need to write PrintWriter interface:
- PrintWriter out=response.getWriter();
- **Example:**

```
<html>
<body>
<% out.print(“Hello student”) %>
</body>
</html>
```

2.Request :

- The request object is an instance of java.servlet.http.**HttpServletRequest** and it is one of the argument of service method.
- It uses getParameter() to access the request parameter.
- **Example:**

```
<html>
<body>
<%
  String name=request.getParameter(“txtnm”);
  out.print(“welcome: ”+name) ;
%>
</body>
</html>
```

3.Response :

- Response" is an instance of class which implements **HttpServletResponse** interface.
- The response implicit object is used to content type, add cookie and redirect to response page
- **Example:**

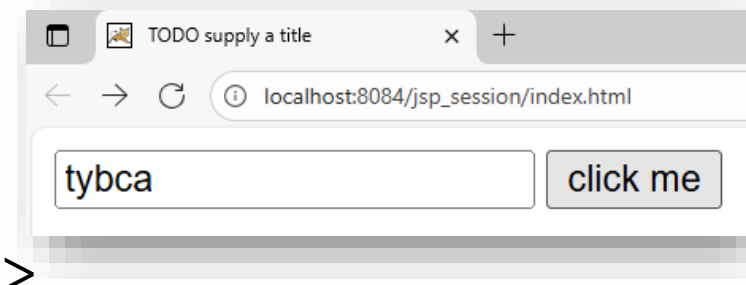
```
<html>
<body>
<%
        response.setContentType("text/html");
%>
</body>
</html>
```

4.Session :

- Session object is used to get, set and remove attributes to session scope and also used to get session information.
- Session is An instance of javax.servlet.http.**HttpSession** interface.
- **Example :**

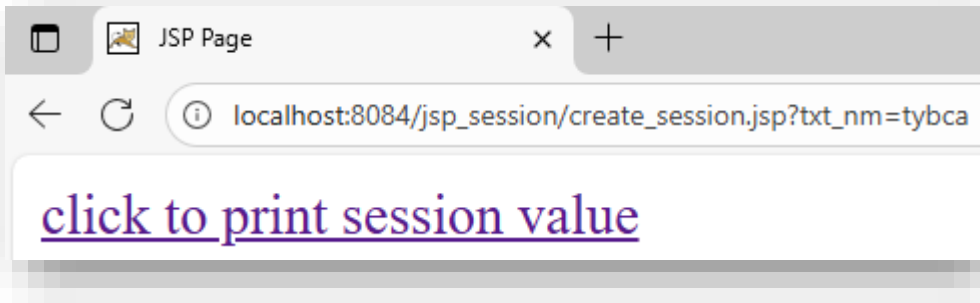
1. index.html

```
<form action="create_session.jsp">
   <input type="text" name="txt_nm">
   <input type="submit" value="click me">
</form>
```



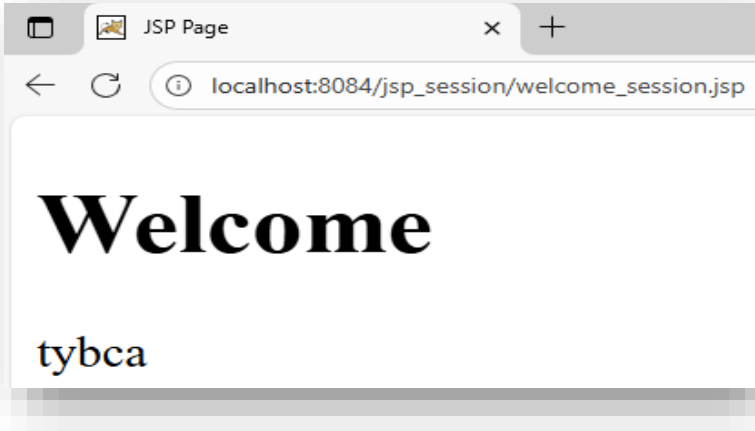
2. create_session.jsp

```
<body>
    <%
      session.setAttribute("s_nm", request.getParameter("txt_nm"));
    %>
    <a href="welcome_session.jsp">click to print session data    </a>
  </body>
```



3. welcome_session.jsp

```
<body>
    <h1>Welcome</h1>
    <%=session.getAttribute("s_nm") %>
</body>
```

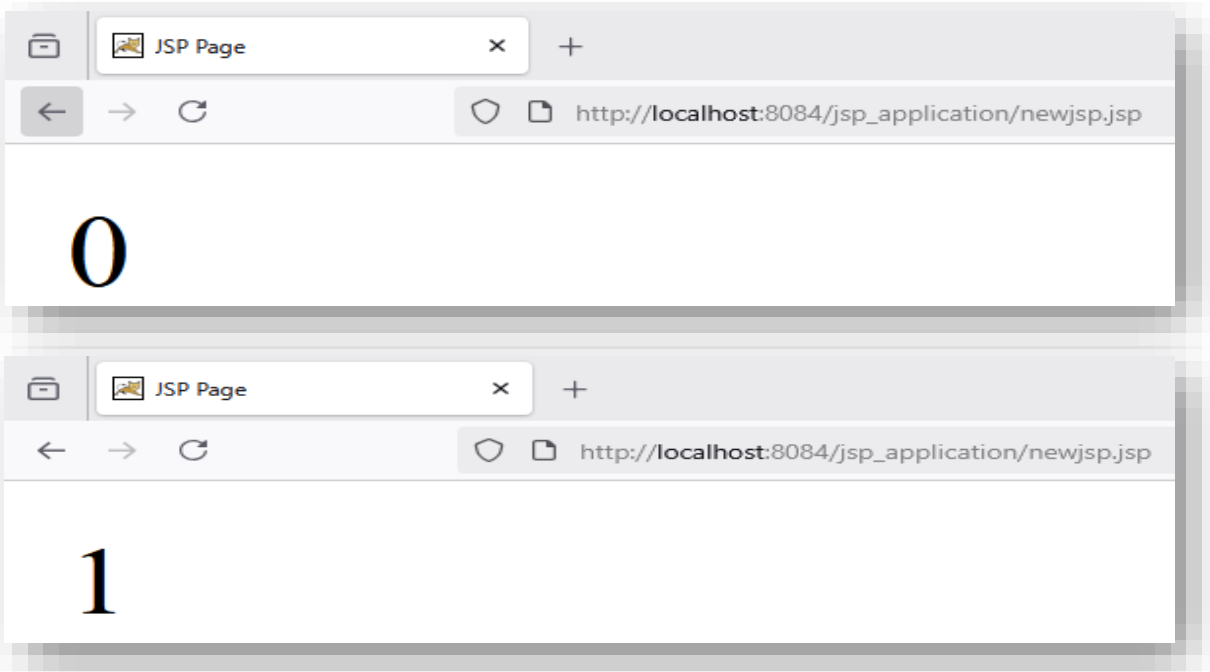


5. Application :

- The application implicit object in JSP represents the javax.servlet.**ServletContext** object, which provides a way to share data across all users and all requests within a web application.
- Its scope is the entire web application, meaning any data stored in the application object is accessible from any JSP page or servlet within the same web application throughout its lifecycle.

Jsp file :

```
<body>
    <%
      Integer a=(Integer)application.getAttribute("app");
      if(a==null)
      {
        a=0;
      }
      else
      {
        a++;
      }
      application.setAttribute("app",a);
      out.print(a);
    %>
  </body>
```



6. Exception :

- The exception is normally an object that is thrown at runtime.
- Exception Handling is the process to handle the runtime errors.
- There may occur exception any time in your web application.
- So handling exceptions is a safer side for the web developer. In JSP, there are two ways to perform exception handling. (page number 72-73)

1. By **errorPage** and **isErrorPage** attributes of page directive.
2. By **<error-page>** element in web.xml file.

- **Example**

```
<body>
<%
    try {
        int result = 10 / 0;
        out.println("Result: " + result);
    } catch (Exception e) {
        out.println("Error : "+e);
    }
%>
<body>
```

7. Config :

- The config implicit object in JSP is an instance of javax.servlet.ServletConfig.
- It provides access to initialization parameters defined for a specific JSP page (which is treated as a servlet by the web container) within the web.xml deployment descriptor.
- It also allows retrieving the servlet name.

- **Html file :**

```
<a href="newjsp">click me</a>
```

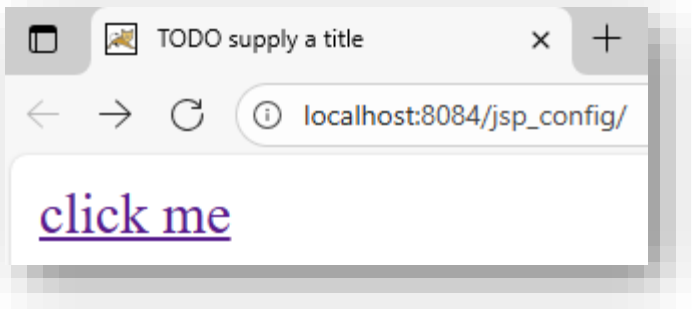
- **Jsp file :**

```
<body>
    <%
    // Get the servlet name
    String servletName = config.getServletName();
    out.println("<h1>Servlet Name: " + servletName + "</h1>");
    // Get initialization parameters
    String greeting = config.getInitParameter("greetingMessage");
    String version = config.getInitParameter("version");
    out.println("<p>Greeting Message: " + greeting + "</p>");
    out.println("<p>Version: " + version + "</p>");
  %>
</body>
```

Web.xml file

```
<servlet>
    <servlet-name>myJspServlet</servlet-name>
    <jsp-file>/newjsp.jsp</jsp-file>
    <init-param>
      <param-name>greetingMessage</param-name>
      <param-value>Hello from JSP Config!</param-value>
    </init-param>
    <init-param>
      <param-name>version</param-name>
      <param-value>1.0</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>myJspServlet</servlet-name>
    <url-pattern>/newjsp</url-pattern>
  </servlet-mapping>
```

Output :


TODO supply a title
localhost:8084/jsp_config/
click me


JSP Page
localhost:8084/jsp_config/newjsp
Servlet Name: myJspServlet
Greeting Message: Hello from JSP Config!
Version: 1.0

Jsp scope :

- The availability of a JSP object for use from a particular place of the application is defined as the scope of that JSP object.
- Every object created in a JSP page will have a scope.
- Object scope in JSP is divide into four parts and they are page, request, session and application.

1. page scope
2. Request scope
3. Session Scope
4. Application Scope

- **1.Page Scope:**
- Page Scope makes variable available to the developer for the current page only.
- Once the current page is closed by user or forwarded internally by application or redirected by application, than the variables having page scope will not be accessible on next page.
- **2.Request Scope:**
- A JSP object created using the request scope can be accessed from any pages that serves that request.
- More than one page can serve a single request.
- The JSP object will be bound to the request object.
- Implicit object request has the request scope.
- **3.session:**
- session scope means, the JSP object is accessible from pages that belong to the same session from where it was created.
- The JSP object that is created using the session scope is bound to the session object. Implicit object session has the session scope.
- **4.application:**
- A JSP object created using the „application“ scope can be accessed from any pages across the application.
- The JSP object is bound to the application object. Implicit object application has the application scope.

- Including and Forwarding from JSP Pages
  - include Action
  - forward Action

- **Include : there are two thing in JSP include using include directive and using include action.**
- **1. using include directive (<%@include>) :**
- The include directive is used to include the contents of any resource it may be jsp file, html file or text file.
- **Advantageof Include directive Code Reusability.**
- **Syntax :-<%@ include file="relative path" %>**
- **Example:**
- **File1 header.html**

```
<body>
    <h1>header file </h1>
</body>
```

- **File 2 JSP file :**

```
<html>
    <body>
        <%@ include file="header.html" %>
        <h1>JSP FILE CODE </h1>
    </body>
</html>
```

- **2. using action element <jsp:include> :**
- The jsp: include action tag is used to include the content of another resource it may be jsp, html or servlet.
- The jsp include action tag includes the resource at **request time** (runtime) so it is better for dynamic pages because there might be changes in future.

- Advantage of include action tag:
- **Code reusability:** We can use a page many times such as including header and footer pages in all pages. So it saves a lot of time.
- **Syntax:**
- <jsp:include page="relativeURL|<%=expression%>"/>

- **Example :**
- **first_file.jsp**

```
<jsp:include page="second_file.jsp"></jsp:include>
<body>
    <h1>hello from first</h1>
</body>
```

- **second_file.jsp**

```
<body>
    <h1>second</h1>
</body>
```

JSP include directive	JSP include action
includes resource at translation time.	includes resource at request time.
better for static pages.	better for dynamic pages.
includes the original content in the generated servlet.	calls the include method.

- **Forward using action element <jsp:forward>:**
- With the forward action, the current page stops processing the request and forwards the request to another page.
- Execution never returns to the calling (current) page.
- **Syntax :**
- <jsp:forwardpage= “file_name” />
- **First_page.jsp**

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

second_page.jsp

```
<%= "Hello second page" %>
```

- **Forward one page to other page with parameter First.jsp**

```
<jsp:forward page="second_page.jsp">
        <jsp:param name=“par_1" value="<%= 10+20 %>">
        </jsp:param>
 </jsp:forward>
```

second_page.jsp

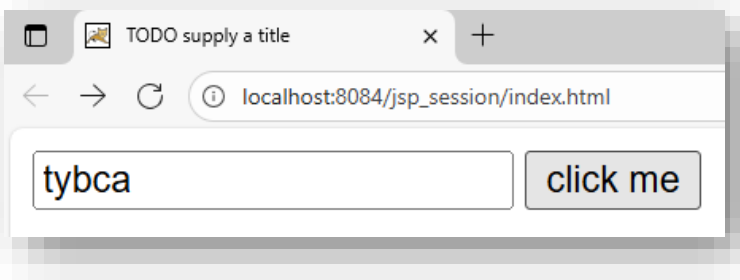
```
<%= request.getParameter(" par_1 ") %>
```

Working with Session & Cookie in JSP :

- **1. Working with Session :**
- Session is a implicit object of JSP. It used to get, set and remove attributes to session scope and also used to get session information.
- Session is An instance of javax.servlet.http.**HttpSession** interface.
- **Example :**

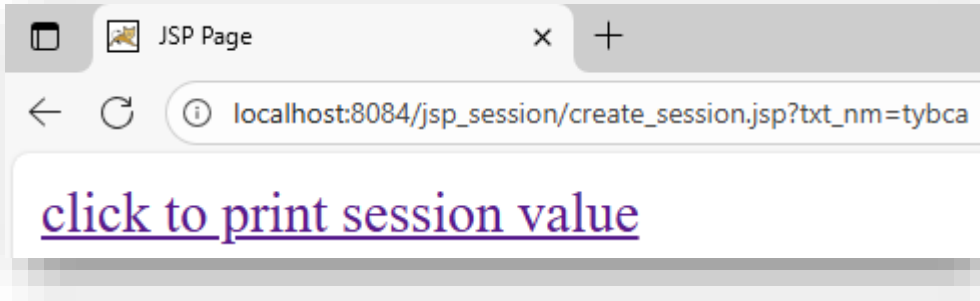
1. index.html

```
<form action="create_session.jsp">
   <input type="text" name="txt_nm">
   <input type="submit" value="click me">
</form>
```



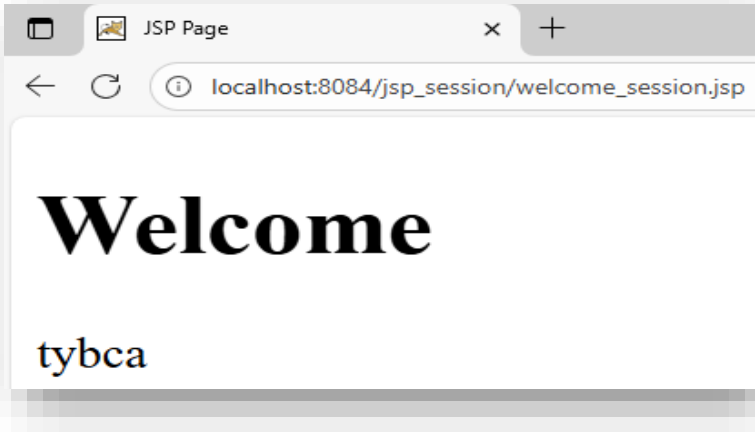
2. create_session.jsp

```
<body>
    <%
      session.setAttribute("s_nm", request.getParameter("txt_nm"));
    %>
    <a href="welcome_session.jsp">click to print session data    </a>
  </body>
```



3. welcome_session.jsp

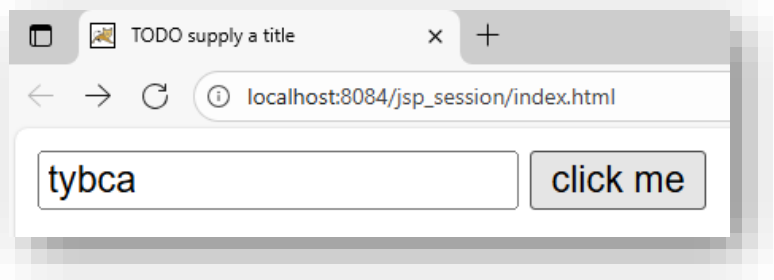
```
<body>
    <h1>Welcome</h1>
    <%=session.getAttribute("s_nm") %>
</body>
```



- **2. Working with Cookie :**
- Cookies are the text files which are stored on the client machine.
- They are used to track the information for various purposes.
- It supports HTTP cookies using servlettechnologyThe cookies are set in the HTTP Header.
- If the browser is configured to store cookies, it will keep information until expiry date.

Index.html :

```
<form action="first.jsp">
   <input type="text" name="txt1">
   <input type="submit">
</form>
```



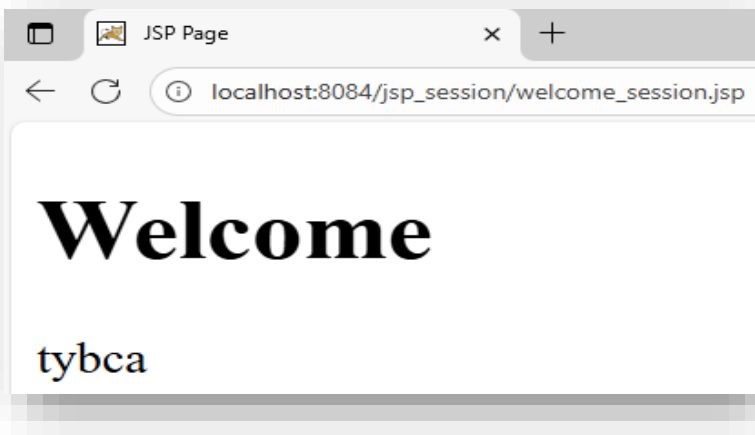
First.jsp :

```
<%
        Cookie ck=new Cookie("ck_first",request.getParameter("txt1"));
        ck.setMaxAge(20);
        response.addCookie(ck);
    %>
    <a href="second.jsp">click to print cookie value in other page</a>
```

click to print cookie value in other page

Second.jsp :

```
<%
        Cookie p[]=request.getCookies();
        out.print(p[0].getValue());
        for(Cookie ck:p)
        {
           out.print(ck.getValue());
        }
    %>
```



Error Handling and Exception Handling with JSP :

- The exception is normally an object that is thrown at runtime.
- Exception Handling is the process to handle the runtime errors.
- There may occur exception any time in your web application.
- So handling exceptions is a safer side for the web developer. In JSP, there are two ways to perform exception handling

1. By **errorPage** and **isErrorPage** attributes of page directive.
2. By **<error-page>** element in web.xml file.
3. By using **Exception** Class

- **Example 1** of Exception handling using page directive :
- **First.jsp file :**

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

- **myerrorpage.jsp**

```
<%@ page isErrorPage="true" %>
<h1> Any number is not cannot divided by zero </h1>
```

- **Example 2 of try catch block : jsp file**

```
<body>
   <%
      try {
         int result = 10 / 0;
         out.println("Result: " + result);
      } catch (Exception e) {
         out.println("Error : "+e);
      }
   %>
<body>
```

- **Example 3 Exception handling using deployment** descriptor**<error-page>** element in web.xml file.

  Here, there is no abc.html file found after click on link so it will execute error_page.jsp file

```
<body>
    <a href="abc.html">Click here</a>
</body>
```

error_page.jsp

```
<h1> Any number is not cannot divided by zero </h1>
```

→right click on WEB-INF → new → other → web → standard devlopment descriptor(we.xml)

Web.xml

```
<error-page>
    <error-code>404</error-code>
    <location>/error_page.jsp</location>
</error-page>
```

JSP EL :

- Expression Language (EL) was introduced in JSP 2.0 as a simpler way to access and manage data in JSP pages.
- It provides an easy syntax to retrieve values from commonly used objects like request, session, application and JavaBeans.
- Normally, to get data from these objects, we need to write extra code. For example:
- 1. To get data from a request, we write.

  request.getParameter("name")
- 2. To get data from a session, we write.

  session.getAttribute("user")
- This makes the code longer and harder to read. But with Expression Language, we can do the same in a much simpler way:

  ${param.name}

  ${sessionScope.user}

- **Why Use Expression Language (EL)?**
- Expression Language makes the data access work much easier.
- The data stored in the objects can be directly retrieved with the help of the Expression Language
- The data is stored in the JavaBeans Component.
- It allows to access a bean by following a short syntax, It can also include literals.
- **Common Usage Patterns**
- **1. Variable Access:**
- Used to access the value of a variable

  ${variableName}

  ${xxxScope.variableName}
- **2. Property Access:** Used to access the value of a property

  ${object.property}
- **3. Arithmetic and Logical Operators**: Performs arithmetic or logical operations on numbers

  ${num1>num2}

  ${num1+num2}
- **4. Conditional Expressions:** Evaluate a condition and return one of two values.

  ${condition ? value1 : value2 }
- **5. Collection Iterator :** Iterates over a collection and prints each item

  <c:forEach var="item" items = "${collection}" > ${item} </c:forEach >
- **6. Accessing Request Parameters**:

  ${param. parametersName}

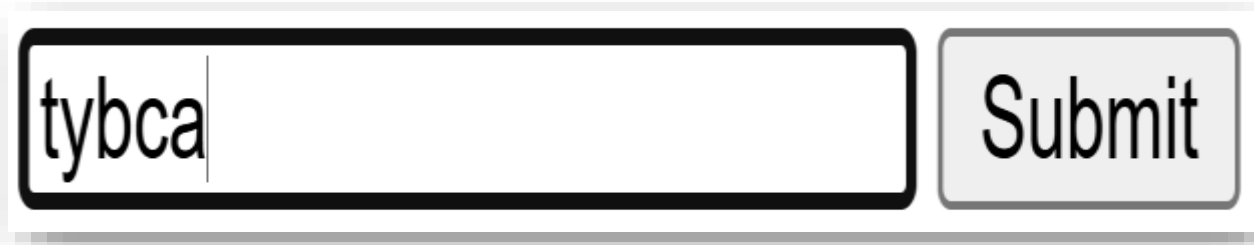
- Some implicit objects in Expression Language are:
- **requestScope:** contains attributes that are specific to the current HTTP request.
- **param**: provides access to request parameters sent to the server.
- **paramValues**: provides access to request parameter values as an array.
- **sessionScope**: Contains attributes that are specific to the current user session.
- **pageContent**: Provides access to various objects and methods related to the JSP page content.
- **pageScope**: Contains attributes that are specific to the current JSP Page.
- **applicationScope**: Contains attributes that are specific to the entire web application, etc.
- **Expression Language also works for**
- Arithmetic Operators (+, - , * , / , %)
- Logical Operators (&&, ||, !)
- Relational Operators (==, !=, >, <, <=, >=)
- Conditional Operator (?)
- Empty Operator (empty, used to check if a value is empty, such as null or an empty collection/string)

- list of reserved words in EL that should not be used as variable names or identifiers within EL expressions:
- ge
- ne
- lt
- le
- gt
- eq
- true
- false
- null
- and
- or
- div
- not
- instanceof
- mod
- empty
- **Example :**
  {10 **gt** 20}
  ${10 < 20}

false true

- Index.html

```
<form action="first.jsp" method="post">
   <input type="text" name="txt1">
   <input type="submit">
</form>
```



- first.jsp

```
<body>
        ${param.txt1}
        ${10+20}
</body>
```

JSP Standard Tag Library (JSTL) :

- JSTL(JSP Standard Tag Library) is a collection of custom tags that provide common functionalities like flow control, database operations, etc. JSTL tags can be embedded in Java Server Pages just like other HTML tags. It is convenient for front-end developers to work with HTML-like tags for including logic in webpages rather than writing Java code in scripts.
- JSTL tags are grouped into five major categories:
- Core Tags
- Formatting Tags
- SQL Tags
- XML Tags
- Function Tags
- **Must Visit Following link for all JSTL tags :**
- https://www.tutorialspoint.com/jsp/jsp_standard_tag_library.htm

How to use JSTL :

Step 1 : add jstl 1.2.jar file in your lib folder.

Step 2 : add uri using taglib in JSP file use prefix in every JSTL tags like <c:set> </c:set>

first.Jsp file :

```
<%@taglib  uri = "http://java.sun.com/jsp/jstl/core" prefix = "c"%>
<body>
<p>How to create variable and print variable in JSTL :</p>
<c:set var="b" value="10"></c:set>
      <c:out value="${b}"></c:out>
<c:out value="${param.num}"></c:out>
<p>how to check condition</p>
      <c:if test="${b>5}">
         <h1>hello sem 5</h1>
      </c:if>
</body>
```

choose like is used to check multiple conditions **when** is like if condition **otherwise** is like else or default.

first.Jsp file :

```
<%@taglib  uri = "http://java.sun.com/jsp/jstl/core" prefix = "c"%>
<body>
<c:set var="b" value="10"></c:set>
      <c:choose>
         <c:when test="${b>0}">
               positive
         </c:when>
         <c:when test="${b<0}">
            negative
         </c:when>
         <c:otherwise>
                  zero
         </c:otherwise>
      </c:choose>
</c:choose>
</body>
```

first.Jsp file :

```
<%@taglib  uri = "http://java.sun.com/jsp/jstl/core" prefix =
   "c"%>
<body>
<!-- foreach -->
      <c:forEach var="i" begin="1" end="10">
            ${i}
      </c:forEach>
<!-- forEach with Collection of array -->
<%
        int[] arr={10,12,15,14,9};
        session.setAttribute("marks",arr);
        %>

      <c:forEach var="m" items="${sessionScope.marks}">
        ${m}
      </c:forEach>
</body>
```

JSP with Database :

```
<%@ page import="java.sql.*" %>
<html>
<head><title>Student
   List</title></head>
<body>
   <h1>Student Information</h1>
   <table border="1">
      <tr>
         <th>Roll No</th>
         <th>Name</th>
         <th>Marks</th>
      </tr>
      <%
         try {

   Class.forName("com.mysql.cj.jdbc.Drive
   r");
            Connection conn =
   DriverManager.getConnection("jdbc:my
   sql://localhost:3306/your_database_na
   me", "root", " ");
            Statement stmt =
   conn.createStatement();
            ResultSet rs =
   stmt.executeQuery("SELECT *FROM
   student");
            while (rs.next()) {
      %>
         <tr>
            <td><%= rs.getInt("roll_no")
   %></td>
            <td><%= rs.getString("name")
   %></td>
            <td><%= rs.getInt("marks")
   %></td>
         </tr>
      <%
            }
         } catch (Exception e) {
           out.print("error : "+e);
         }
      %>
   </table>
</body>
</html>
```

Step 1 : add JDBC jar file in lib folder

- SVG : Scalable Vector Graphics
- SSI : Server-Side Includes
- ASP : Active Server Page
- WML : Wireless Markup Language
- EL : Expression Language
- JSTL : Java Server Page Standard Library
- https://repo1.maven.org/maven2/javax/servlet/jstl/1.2/