**UNIT IV: Introduction to Servlets & JSP:**

* Discuss Life cycle of Servlet and Limitations of servlets
* Illustrate Components of a JSP Page,
* Directives,
* Demonstrate Expressions,
* Scriptlets,
* Declarations,
* Implicit objects,
* Database Access and
* Session tracking

**Discuss Life cycle of Servlet and Limitations of servlets**

* **Servlet** technology is used to create a web application (resides at server side and generates a dynamic web page).
* **Servlet** technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language.
* Servlet is an API that provides many interfaces and classes including documentation.
* Servlet is an interface that must be implemented for creating any Servlet.
* Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
* Servlet is a web component that is deployed on the server to create a dynamic web page.



A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter, etc. and other elements such as HTML, CSS, and JavaScript. The web components typically execute in Web Server and respond to the HTTP request.

**Life Cycle of a Servlet (Servlet Life Cycle):**

The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:

* Servlet class is loaded.
* Servlet instance is created.
* init method is invoked.
* service method is invoked.
* destroy method is invoked.



As displayed in the above diagram, there are three states of a servlet: new, ready and end. The servlet is in new state if servlet instance is created. After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks. When the web container invokes the destroy() method, it shifts to the end state.

**1) Servlet class is loaded**

The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

**2) Servlet instance is created**

The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

**3) init method is invoked**

The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below:

public void init(ServletConfig config) throws ServletException

**4) service method is invoked**

The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:

public void service(ServletRequest request, ServletResponse response)

throws ServletException, IOException

**5) destroy method is invoked**

The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:

public void destroy()

**Limitations of servlet:**

1.Time-consuming: Modification in the Servlet file is time-consuming due to reloading, recompiling, and restarting the server.

2.Hefty: Writing HTML code in Servlet programming is challenging, and it makes Servlet looks bulky.

**Illustrate Components of a JSP Page**

JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.

A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags, etc.

**Advantages of JSP over Servlet**

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

**1) Extension to Servlet**

JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

**2) Easy to maintain**

JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.

**3) Fast Development: No need to recompile and redeploy**

If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

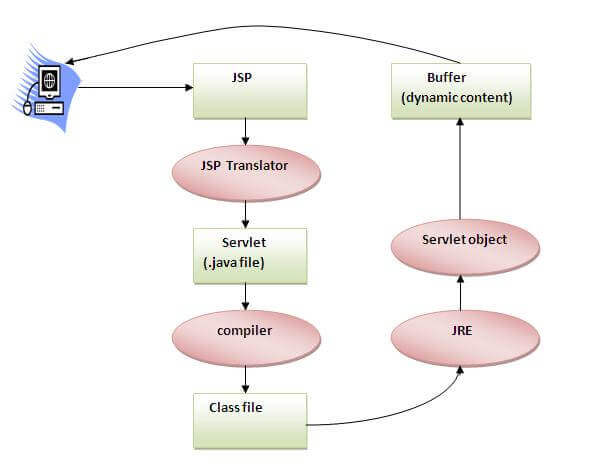
**4) Less code than Servlet**

In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.

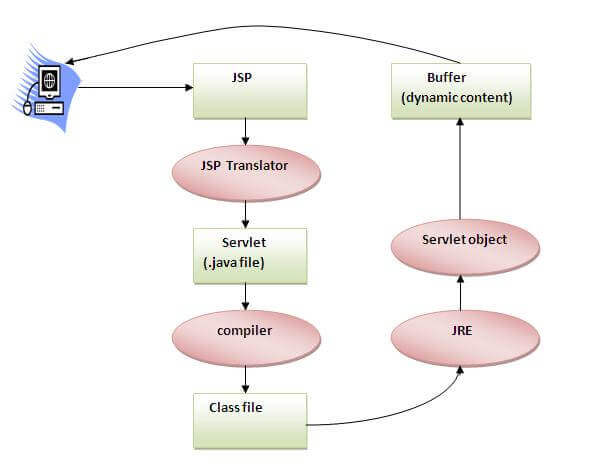
**The Lifecycle of a JSP Page**

The JSP pages follow these phases:

* Translation of JSP Page
* Compilation of JSP Page
* Classloading (the classloader loads class file)
* Instantiation (Object of the Generated Servlet is created).
* Initialization ( the container invokes jspInit() method).
* Request processing ( the container invokes \_jspService() method).
* Destroy ( the container invokes jspDestroy() method).
* jspInit(), \_jspService() and jspDestroy() are the life cycle methods of JSP.



As depicted in the above diagram, JSP page is translated into Servlet by the help of JSP translator. The JSP translator is a part of the web server which is responsible for translating the JSP page into Servlet. After that, Servlet page is compiled by the compiler and gets converted into the class file. Moreover, all the processes that happen in Servlet are performed on JSP later like initialization, committing response to the browser and destroy.



As depicted in the above diagram, JSP page is translated into Servlet by the help of JSP translator. The JSP translator is a part of the web server which is responsible for translating the JSP page into Servlet. After that, Servlet page is compiled by the compiler and gets converted into the class file. Moreover, all the processes that happen in Servlet are performed on JSP later like initialization, committing response to the browser and destroy.

**Creating a simple JSP Page**

To create the first JSP page, write some HTML code as given below, and save it by .jsp extension. We have saved this file as index.jsp. Put it in a folder and paste the folder in the web-apps directory in apache tomcat to run the JSP page.

index.jsp

Let's see the simple example of JSP where we are using the scriptlet tag to put Java code in the JSP page. We will learn scriptlet tag later.

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

It will print 10 on the browser.

**Components of a JSP Page**

**1.Directives**

A directives tag always appears at the top of your JSP file. It is global definition sent to the JSP engine. Directives contain special processing instructions for the web container. You can import packages, define error handling pages or the session information of the JSP page. Directives are defined by using <%@ and %> tags.

**Syntax -** <%@ directive attribute="value" %>

**2.Expressions**

Expressions in JSPs is used to output any data on the generated page. These data are automatically converted to string and printed on the output stream. It is an instruction to the web container for executing the code with in the expression and replace it with the resultant output content. For writing expression in JSP, you can use <%= and %> tags.

**Syntax -** <%= expression %>

**3.Scriptlet**

In this tag we can insert any amount of valid java code and these codes are placed in \_jspService method by the JSP engine. Scriptlets can be used anywhere in the page. Scriptlets are defined by using <% and %> tags.

**Syntax -**<% Scriptlets%>

**4.Declaration**

This tag is used for defining the functions and variables to be used in the JSP. This element of JSPs contains the java variables and methods which you can call in expression block of JSP page. Declarations are defined by using <%! and %> tags. Whatever you declare within these tags will be visible to the rest of the page.

**Syntax -**<%! declaration(s) %>

**5.Comments :**

Comments help in understanding what is actually code doing. JSPs provides two types of comments for putting comment in your page. First type of comment is for output comment which is appeared in the output stream on the browser. It is written by using the <!-- and --> tags.

<!-- comment text -->

Second type of comment is not delivered to the browser. It is written by using the <%-- and --%> tags.

**Syntax -**

<%-- comment text --%>

**JSP directives**

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

There are three types of directives:

* page directive
* include directive
* taglib directive

Syntax of JSP Directive

<%@ directive attribute="value" %>

**JSP page directive**

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

Syntax of JSP page directive

<%@ page attribute="value" %>

**Attributes of JSP page directive**

import, contentType, extends, info, buffer, language, autoFlush, session

PageEncoding, errorPage, isErrorPage

**1)import**

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

Example of import attribute

<html>

<body>

<%@ page import="java.util.Date" %>

Today is: <%= new Date() %>

</body>

</html>

2)contentType

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

Example of contentType attribute

<html>

<body>

<%@ page contentType=application/msword %>

Today is: <%= new java.util.Date() %>

</body>

</html>

3)extends

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

4)info

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

Example of info attribute

<html>

<body>

<%@ page info="composed by abc" %>

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.

Example of errorPage attribute

//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.

Note: The exception object can only be used in the error page.

Example of isErrorPage attribute

//myerrorpage.jsp

<html>

<body>

<%@ page isErrorPage="true" %>

 Sorry an exception occured!<br/>

The exception is: <%= exception %>

</body>

</html>

**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. The include directive includes the original content of the included resource at page translation time (the jsp page is translated only once so it will be better to include static resource).

Advantage of Include directive

Code Reusability

Syntax of include directive

<%@ include file="resourceName" %>

Example of include directive

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

<html>

<body>

<%@ include file="header.html" %>

Today is: <%= java.util.Calendar.getInstance().getTime() %>

</body>

</html>

**JSP 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 JSP Taglib directive

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

**Implicit object**

There are 9 jsp implicit objects. These objects are created by the web container that are available to all the jsp pages.The available implicit objects are out, request, config, session, application etc.

|  |  |
| --- | --- |
| Object | Type |
| out | JspWriter |
| request | HttpServletRequest |
| response | HttpServletResponse |
| config | ServletConfig |
| application | ServletContext |
| session | HttpSession |
| pageContext | PageContext |
| page | Object |
| exception | Throwable |

**Example of out implicit object**

In this example we are simply displaying date and time.

index.jsp

<html>

<body>

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

</body>

</html>

**JSP request implicit object**

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

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

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

**Example of JSP request implicit object**

**index.html**

<form action="welcome.jsp">

<input type="text" name="uname">

<input type="submit" value="go"><br/>

</form>

**welcome.jsp**

<%

String name=request.getParameter("uname");

out.print("welcome "+name);

%>

**JSP response implicit object**

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

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

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

**Example of response implicit object**

**index.html**

<form action="welcome.jsp">

<input type="text" name="uname">

<input type="submit" value="go"><br/>

</form>

**welcome.jsp**

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

**JSP config implicit object**

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

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

**Example of config implicit object:**

index.html

<form action="welcome">

<input type="text" name="uname">

<input type="submit" value="go"><br/>

</form>

**web.xml file**

<web-app>

<servlet>

<servlet-name>abcl</servlet-name>

<jsp-file>/welcome.jsp</jsp-file>

<init-param>

<param-name>dname</param-name>

<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>abc</servlet-name>

<url-pattern>/welcome</url-pattern>

</servlet-mapping>

</web-app>

**welcome.jsp**

<%

out.print("Welcome "+request.getParameter("uname"));

String driver=config.getInitParameter("dname");

out.print("driver name is="+driver);

%>

**JSP application implicit object**

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

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

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

Example of application implicit object:

index.html

<form action="welcome">

<input type="text" name="uname">

<input type="submit" value="go"><br/>

</form>

**web.xml file**

<web-app>

<servlet>

<servlet-name>abc</servlet-name>

<jsp-file>/welcome.jsp</jsp-file>

</servlet>

<servlet-mapping>

<servlet-name>abc</servlet-name>

<url-pattern>/welcome</url-pattern>

</servlet-mapping>

<context-param>

<param-name>dname</param-name>

<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>

</context-param>

</web-app>

**welcome.jsp**

<%

out.print("Welcome "+request.getParameter("uname"));

String driver=application.getInitParameter("dname");

out.print("driver name is="+driver);

%>

**session implicit object**

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

**Example of session implicit object**

**index.html**

<html>

<body>

<form action="welcome.jsp">

<input type="text" name="uname">

<input type="submit" value="go"><br/>

</form>

</body>

</html>

**welcome.jsp**

<html>

<body>

<%

String name=request.getParameter("uname");

out.print("Welcome "+name);

session.setAttribute("user",name);

<a href="second.jsp">second jsp page</a>

%>

</body>

</html>

**second.jsp**

<html>

<body>

<%

String name=(String)session.getAttribute("user");

out.print("Hello "+name);

%>

</body>

</html>

**pageContext implicit object**

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

* page
* request
* session
* application

In JSP, page scope is the default scope.

**Example of pageContext implicit object**

**index.html**

<html>

<body>

<form action="welcome.jsp">

<input type="text" name="uname">

<input type="submit" value="go"><br/>

</form>

</body>

</html>

**welcome.jsp**

<html>

<body>

<%

String name=request.getParameter("uname");

out.print("Welcome "+name);

pageContext.setAttribute("user",name,PageContext.SESSION\_SCOPE);

<a href="second.jsp">second jsp page</a>

%>

</body>

</html>

**second.jsp**

<html>

<body>

<%

String name=(String)pageContext.getAttribute("user",PageContext.SESSION\_SCOPE);

out.print("Hello "+name);

%>

</body>

</html>

**page implicit object:**

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

Object page=this;

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

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

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

<% this.log("message"); %>

**exception implicit object**

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

**Example of exception implicit object:**

**error.jsp**

<%@ page isErrorPage="true" %>

<html>

<body>

Sorry following exception occured:<%= exception %>

</body>

</html>

**Database Access**

To access a database from a JSP (JavaServer Pages) application, you typically use Java Database Connectivity (JDBC) to establish a connection to the database, execute SQL queries, and retrieve data. Here's a step-by-step guide on how to access a database from a JSP page:

1.Set up your database:

Ensure you have a database system installed and running, and you have the necessary credentials (username and password) to access it. Common database systems include MySQL, PostgreSQL, Oracle, and SQLite.

2.Add the JDBC driver:

Download the JDBC driver for your database and add it to your project's classpath. Different databases have different JDBC drivers, so make sure you use the appropriate one.

3.Create a database connection:

In your JSP page or in a Java class that you can call from your JSP, establish a connection to the database using the JDBC driver. Here's an example of how to establish a connection to a MySQL database:

java

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class DatabaseUtil {

private static Connection connection = null;

public static Connection getConnection() {

if (connection != null)

return connection;

else {

try {

String driver = "com.mysql.jdbc.Driver";

String url = "jdbc:mysql://localhost:3306/your\_database";

String username = "your\_username";

String password = "your\_password";

Class.forName(driver);

connection = DriverManager.getConnection(url, username, password);

} catch (ClassNotFoundException | SQLException e) {

e.printStackTrace();

}

return connection;

}

}

}

**Replace your\_database, your\_username, and your\_password with your database information.**

**Execute SQL queries:**

You can now use the Connection object obtained in the previous step to execute SQL queries. Here's an example of querying a database and retrieving data in a JSP page:

**jsp**

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

<%@ page import="java.sql.Connection, java.sql.PreparedStatement, java.sql.ResultSet" %>

<%@ page import="your.package.DatabaseUtil" %>

<html>

<head>

<title>Database Access from JSP</title>

</head>

<body>

<h1>Employee List</h1>

<table border="1">

<tr>

<th>ID</th>

<th>Name</th>

<th>Email</th>

</tr>

<%

Connection connection = DatabaseUtil.getConnection();

String sql = "SELECT id, name, email FROM employees";

PreparedStatement statement = connection.prepareStatement(sql);

ResultSet resultSet = statement.executeQuery();

while (resultSet.next()) {

%>

<tr>

<td><%= resultSet.getInt("id") %></td>

<td><%= resultSet.getString("name") %></td>

<td><%= resultSet.getString("email") %></td>

</tr>

<%

}

resultSet.close();

statement.close();

connection.close();

%>

</table>

</body>

</html>

This JSP page connects to the database, executes a SELECT query, and displays the results in an HTML table.

**Session tracking**

Session tracking in JSP allows you to maintain state information about a user across multiple HTTP requests. It's a fundamental feature for building web applications that need to remember user-specific data or maintain a user's session throughout their interactions with your site. JSP provides several ways to implement session tracking:

**1. \*\*Using HttpSession:\*\***

HttpSession is a built-in object in JSP that allows you to store and retrieve session-specific data. Here's how to use it:

To create or obtain a session in JSP:

HttpSession session = request.getSession();

To set session attributes:

session.setAttribute("username", "john\_doe");

To get session attributes:

String username = (String) session.getAttribute("username");

To invalidate a session (usually when the user logs out):

session.invalidate();

**2. \*\*Using Cookies:\*\***

Cookies are small pieces of data that can be stored on the client-side and sent with each HTTP request. You can use cookies to maintain session information by storing a unique identifier on the client-side and associating it with the session data stored on the server. JSP provides implicit objects for working with cookies:

To set a cookie:

Cookie cookie = new Cookie("username", "john\_doe");

response.addCookie(cookie);

To get cookies:

Cookie[] cookies = request.getCookies();

if (cookies != null) {

for (Cookie cookie : cookies) {

if (cookie.getName().equals("username")) {

String username = cookie.getValue();

}

}

}

**3. \*\*Using URL Rewriting:\*\***

URL rewriting involves appending session information as URL parameters. You can encode session data into URLs to track sessions. This is done automatically for you in JSP when you use the `<c:url>` tag:

<c:url var="urlWithSession" value="/somepage.jsp">

<c:param name="username" value="john\_doe" />

</c:url>

<a href="${urlWithSession}">Link with Session</a>

This will generate a URL like `/somepage.jsp?username=john\_doe`, including the session information.

**4. \*\*Using Hidden Form Fields:\*\***

You can also store session data as hidden form fields in HTML forms. This way, data is sent back to the server when the user submits the form. However, this approach is less secure because users can manipulate hidden fields.

<form action="processForm.jsp" method="post">

<input type="hidden" name="username" value="john\_doe" />

<!-- Other form fields -->

<input type="submit" value="Submit" />

</form>

Choose the session tracking mechanism that best fits your application's requirements and security considerations. HttpSession is the most commonly used method for session tracking in JSP because it provides a straightforward and secure way to manage session data.