Unit 7 :
Servlets and Java Server Pages

- Servlet is a technology which is used to create a web application.

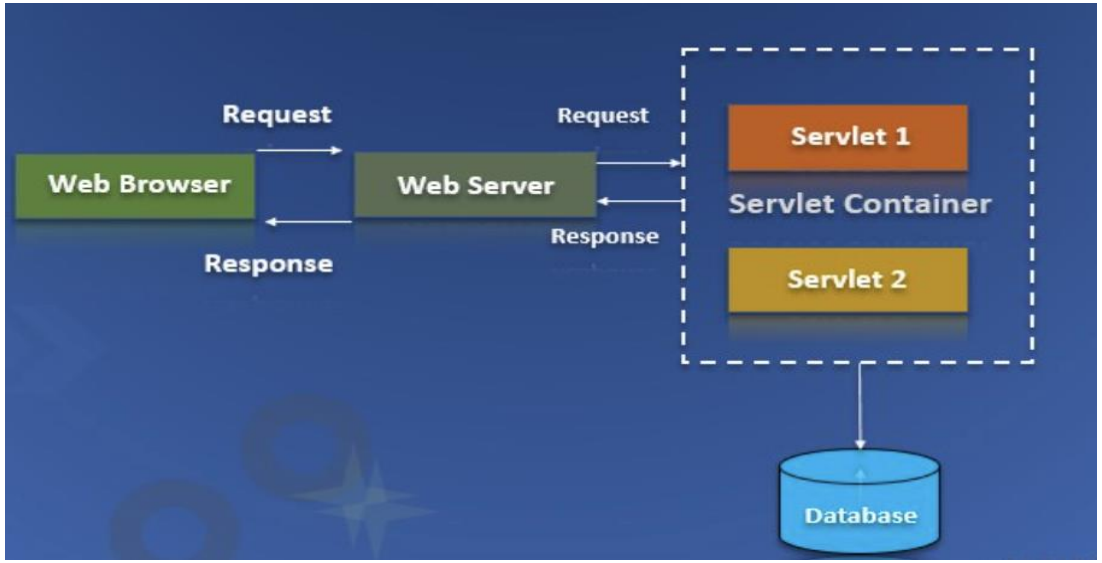
- **Servlet** technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language.

- Java Servlets are programs that run on a Web or Application server and act as a middle layer between a requests coming from a Web browser or other HTTP client and databases or applications on the HTTP server.

Properties of Servlets are as follows:

- Servlets work on the server-side.
- Servlets are capable of handling complex requests obtained from the webserver.

Servlets Architecture



Components of Servlet Architecture :

1. Client

- Web browser acts as a Client. Client or user connected with a web browser.
- The client is responsible for sending requests or HttpRequest to the web server and processing the Web server's responses.

2. Web Server

- Web server controls how web user access hosted files, and it's responsible for processing user request and responses.
- There are two types' web servers

- Static web server
- Dynamic web server.

In a static web server, it sends the file as it is, but in a dynamic web, the server-hosted file is updated before it is sent to the browser.

3. Web Container

- A web container is a component in the webserver it interacts with Java servlets.
- A web container is responsible for managing the lifecycle of servlets, and it also performs the URL mapping task.
- Web container handles the requests of servlets, JSP and other files at the server-side.
- It is also known as Servlet Container and Servlet Engine.

Open-Source Web Server

Open-Source Web Container

Difference between Servlet and CGI

Servlet	**CGI**(Common Gateway Interface)
Servlets are portable and efficient.	CGI is not portable
In Servlets, sharing data is possible.	In CGI, sharing data is not possible.
Servlets can directly communicate with the webserver.	CGI cannot directly communicate with the webserver.
Servlets are less expensive than CGI.	CGI is more expensive than Servlets.
Servlets can handle the cookies.	CGI cannot handle the cookies.

Prerequisites:

- Java Development Kit
- Eclipse IDE for Java EE Developers
- Download and Install Tomcat 9.0

Configure Tomcat Server in eclipse:

- In Eclipse, go to Window -> Preferences -> Server -> Runtime Environments
- Click Add and select Apache Tomcat.
- Browse to the location where you installed Tomcat and click Finish.

Servlet Life Cycle

1. **Load Servlet Class**

- The servlet class is loaded when the first request for the servlet is received by the web container.

2. **Servlet Instance Creation**

- After the servlet class is loaded, web container creates the instance of it. Servlet instance iss created only once in the life cycle.

3. **Call to the init() method**

- Once all the servlet class are instantiated the init() method is invoked for each instantiated servlet. This method initializes the servlet.
- The init() method is called only once during the life cycle of servlet.

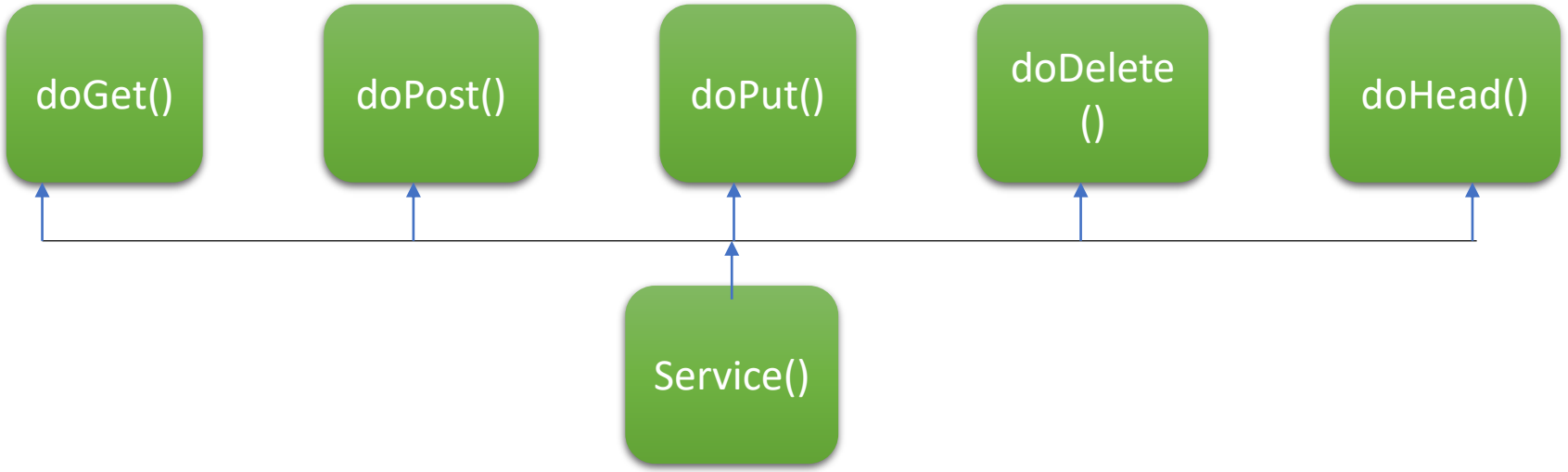
Signature of init() method :

```
public void init(ServletConfig config) throws ServletException {
//Code
}
```

4. Call to the Service method

- The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.
- The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.
- Signature of service() method :

```
public void service(ServletRequest request, ServletResponse response) throws
ServletException, IOException {
//code
}
```

- doGet() and doPost() are most frequently used methods.

doGet() method :

- We use the doGet() method for getting the information from the server.
- The service() method will call doGet() method to allow a servlet to handle a Get request.

Signature of doGet() :

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
//code
}
```


doPost() method :

- doPost() method is called by the server to allow a servlet to handle Post request.
- We use the doPost() method for sending information to the server like html form data.

Signature of doPost():

```
public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
//code
}
```

5. Call to destroy() method.

- The destroy() method is called only once at the end of the life cycle of a servlet.
- This method gives your servlet a chance to close database connections, halt background threads and perform other such cleanup activities.

Signature of destroy() :

```
public void destroy() {
//code
}
```

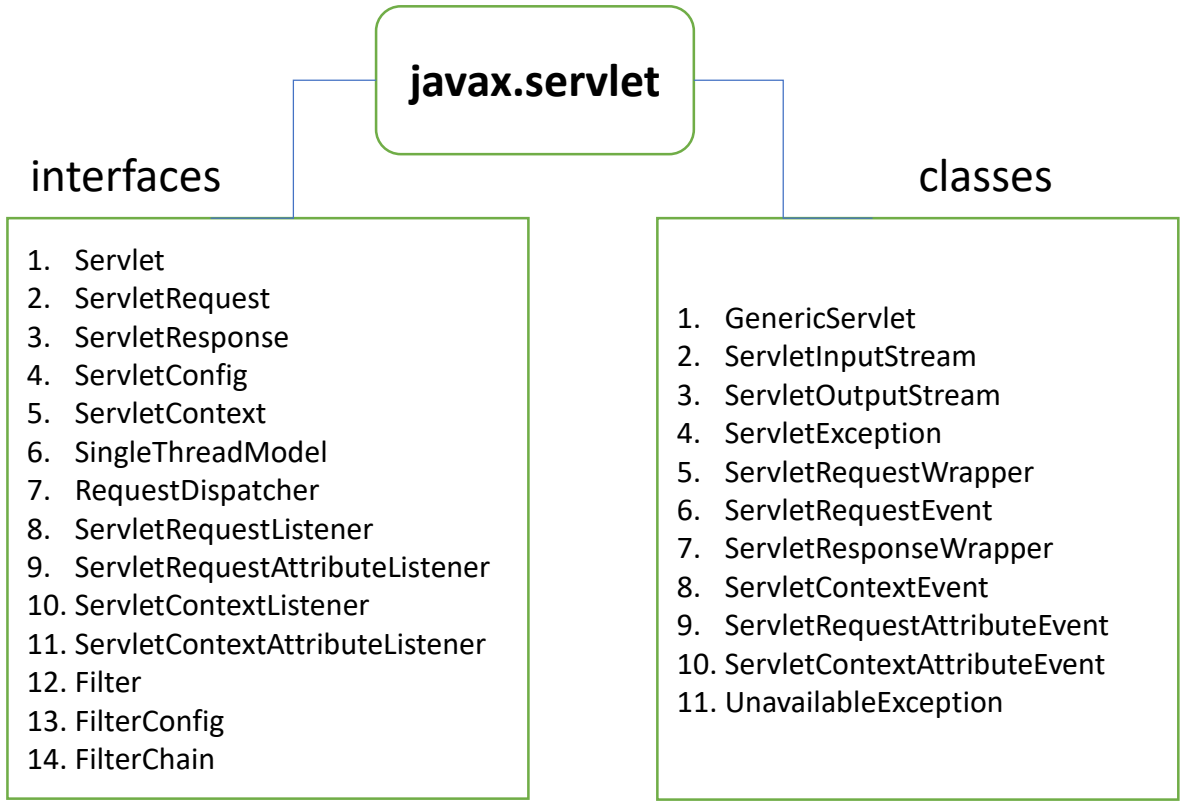
Servlet API

The Servlet API provides interfaces and classes that are required to built servlets. These interfaces and classes are group into the following two packages :

1. javax.servlet
2. javax.servlet.http

1. javax.servlet :

javax.servlet package contains list of interfaces and classes that are used by the servlet or web container. These classes and interface are not specific to any protocol.



Servlet Interface :

- This is the main interface that defines the methods in which all the servlets must implement. To implement this interface, write a generic servlet that extends javax.servlet.GenericServlet or an HTTP servlet that extends javax.servlet.http.HttpServlet.

- If you are creating protocol dependent servlet such as http servlet, then you should extend HttpServlet class else for protocol independent Servlet you extend GenericServlet class.

- In short you have 3 ways to create a servlet:
  1) By extending HttpServlet class
  2) By extending GenericServlet class
  3) By implementing Servlet interface
  Note: However, you should always prefer the first way of creating servlet i.e., by extending HttpServlet class.

Servlet Interface methods

Here is the list of methods available in Servlet interface.

1) **void destroy()**: This method is called by Servlet container at the end of servlet life cycle. Unlike service() method that gets called multiple times during life cycle, this method is called only once by Servlet container during the complete life cycle. Once destroy() method is called the servlet container does not call the service() method for that servlet.

2) **void init(ServletConfig config)**: When Servlet container starts up (that happens when the web server starts up) it loads all the servlets and instantiates them. After this init() method gets called for each instantiated servlet, this method initializes the servlet.

3) **void service(ServletRequest req, ServletResponse res)**: This is the only method that is called multiple times during servlet life cycle. This methods serves the client request, it is called every time the server receives a request.

4) **ServletConfig getServletConfig()**: Returns a ServletConfig object, which contains initialization and startup parameters for this servlet.

5) **java.lang.String getServletInfo()**: Returns information about the servlet, such as author, version, and copyright.

Example :

❖ index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h2>Weclome</h2>
<a href="home">Home</a>
</body>
</html>
```

❖ web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
id="WebApp_ID" version="4.0">
<display-name>ServletnterfaceDemo</display-name>

<servlet>
<servlet-name>Demo1</servlet-name>
<servlet-class>com.texas.servlets.ServletInterfaceEmp</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>Demo1</servlet-name>
<url-pattern>/home</url-pattern>
</servlet-mapping>
<welcome-file-list>
<welcome-file>index.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

❖ ServletInterfaceEmp.java

```java
package com.texas.servlets;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.Servlet;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class ServletInterfaceEmp implements Servlet {

public void init(ServletConfig config) throws ServletException {
}

public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException {
PrintWriter pw = response.getWriter();
pw.println("<html><body><p> From Service method.</p></body></html>");
pw.close();
}

public void destroy() {
}

public ServletConfig getServletConfig() {
return null;
}

public String getServletInfo() {
return null;
}
}
```

ServletConfig :

When the **Web Container** initializes a servlet, it creates a **ServletConfig** object for the servlet. ServletConfig object is used to pass information to a servlet during initialization by getting configuration information from **web.xml**(Deployment Descriptor).

Methods of ServletConfig

1. **String getInitParameter(String name):**
   returns a String value initialized parameter, or NULL if the parameter does not exist.

2. **Enumeration getInitParameterNames():**
   returns the names of the servlet's initialization parameters as an Enumeration of String objects, or an empty Enumeration if the servlet has no initialization parameters.

3. **ServletContext getServletContext()**: returns a reference to the ServletContext

4. **String getServletName()**: returns the name of the servlet instance

Example :

❖ index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<a href="employee">Employee</a>
</body>
</html>
```

❖ web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
id="WebApp_ID" version="4.0">
<display-name>ServletConfigDemo</display-name>

<servlet>
<servlet-name>AA</servlet-name>
<servlet-class>com.texas.servlets.Employee</servlet-class>
<init-param>
<param-name>UserName</param-name>
<param-value>Admin</param-value>
</init-param>
<init-param>
<param-name>UserPassword</param-name>
<param-value>1234567890</param-value>
</init-param>
</servlet>

<servlet-mapping>
<servlet-name>AA</servlet-name>
<url-pattern>/employee</url-pattern>
</servlet-mapping>

<welcome-file-list>
<welcome-file>index.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

❖ Employee.java

```
package com.texas.servlets;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;
import javax.servlet.Servlet;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class Employee implements Servlet {
ServletConfig config = null;

public void destroy() {
}
public ServletConfig getServletConfig() {
return null;
}

public String getServletInfo() {
return null;
}

public void init(ServletConfig config) throws ServletException {
this.config = config;
}

public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException {
PrintWriter pw = response.getWriter();
Enumeration<String> parameterNames = config.getInitParameterNames();
while (parameterNames.hasMoreElements()) {
String paramterName = parameterNames.nextElement();
String UserNameValue = config.getInitParameter(paramterName);
pw.println("<h2> From ServletConfig :  " + paramterName + " , " + UserNameValue);
}
pw.close();
}
}
```

ServletContext :

For every **Web application** a **ServletContext** object is created by the web container. ServletContext object is used to get configuration information from **Deployment Descriptor**(web.xml) which will be available to any servlet or JSPs that are part of the web app.

Method of ServletContext

1. Object getAttribute(String name) : returns the container attribute with the given name, or NULL if there is no attribute by that name.

2. String getInitParameter(String name) : returns parameter value for the specified parameter name, or NULL if the parameter does not exist

3. Enumeration getInitParameterNames() : returns the names of the context's initialization parameters as an Enumeration of String objects

4. void setAttribute(String name,Object obj) : set an object with the given attribute name in the application scope

5. void removeAttribute(String name) : removes the attribute with the specified name from the application context

Example :

❖ index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<a href="employee">Employee</a>
</body>
</html>
```

❖ web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
id="WebApp_ID" version="4.0">
<display-name>ServletContextDemo</display-name>
<context-param>
<param-name>DBHost</param-name>
<param-value>localhost:8080</param-value>
</context-param>

<context-param>
<param-name>DBUserName</param-name>
<param-value>VDEAdmin</param-value>
</context-param>
<servlet>
<servlet-name>AA</servlet-name>
<servlet-class>com.texas.servlets.Employee</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>AA</servlet-name>
<url-pattern>/employee</url-pattern>
</servlet-mapping>

<welcome-file-list>
<welcome-file>index.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

❖ Employee.java

```
package com.texas.servlets;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;
import javax.servlet.Servlet;
import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class Employee implements Servlet {

ServletContext context = null;

public ServletConfig getServletConfig() {
return null;
}

public String getServletInfo() {
return null;
}

public void init(ServletConfig config) throws ServletException {
context =  config.getServletContext();
}

public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException {
PrintWriter pw = response.getWriter();
Enumeration<String> contextParameterNames = context.getInitParameterNames();
while(contextParameterNames.hasMoreElements()) {
String paramterName = contextParameterNames.nextElement();
String UserNameValue = context.getInitParameter(paramterName);
pw.println("<h2> From ServletContext :  "+paramterName+" , "+UserNameValue);
}
pw.close();
}

public void destroy() {
}
}
```

Difference :

ServletContext Init parameters	ServletConfig Init parameter
Available to all servlets and JSPs that are part of web	Available to only servlet for which the <init-param> was configured
Context Init parameters are initialized within the <web-app> not within a specific <servlet> elements	Initialized within the <servlet> for each specific servlet.
ServletContext object is used to get Context Init parameters	ServletConfig object is used to get Servlet Init parameters
Only one ServletContext object for entire web app	Each servlet has its own ServletConfig object

ServletRequest :
It defines an object that is created by servlet container to pass client request information to a servlet.

ServletResponse :
It defines an object created by servlet container to assist a servlet in sending a response to the client.

RequestDispatcher :
It defines an object that receives the request from client and dispatches it to the resource(such as servlet, JSP, HTML file).

GenericServlet :

- ❖ GenericServlet is an abstract class.
- ❖ This class implements the servlet, ServletConfig and Serializable interface.
- ❖ This class provides the implementation of most of the basic servlet methods.
- ❖ The protocol of this class is independent as it can handle any type of request.

Methods of GenericServlet :

1) **void destroy()**: is invoked only once throughout the life cycle and indicates that servlet is being destroyed.

2) **void init(ServletConfig config)**: is used to initialize the servlet.

3) **void service(ServletRequest req, ServletResponse res)**: provides service for the incoming request. It is invoked at each time when user requests for a servlet.

4) **ServletConfig getServletConfig()**: Returns a ServletConfig object, which contains initialization and startup parameters for this servlet.

5) **java.lang.String getServletInfo()**: Returns information about the servlet, such as author, version, and copyright.

6) **public ServletContext getServletContext() :** Returns the object of ServletContext.

7) **public String getInitParameter(String name) :** Returns the parameter value for the given parameter name.

8) **public Enumeration getInitParameterNames()** : Returns all the parameters defined in the web.xml file.

9) **public String getServletName() :** Returns the name of the servlet object.

Example :

❖ index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<a href="student">Student</a>
</body>
</html>
```

❖ web.xml

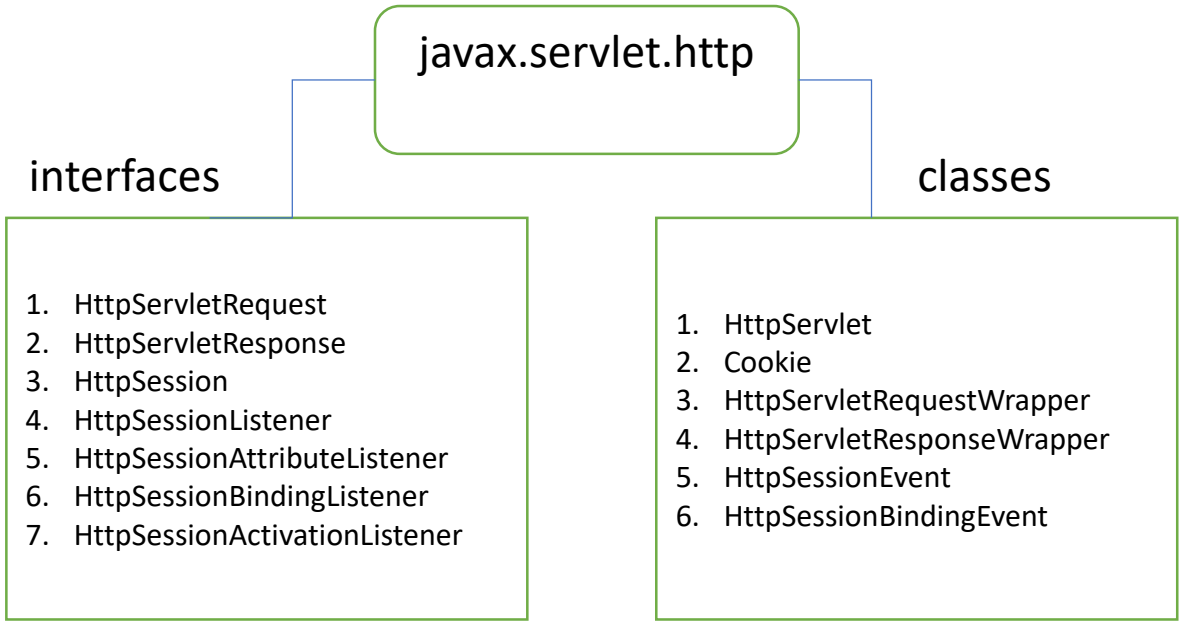
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
id="WebApp_ID" version="4.0">
<display-name>GenericServlet Demo</display-name>
<servlet>
<servlet-name>BB</servlet-name>
<servlet-class>com.texas.servlets.Student</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>BB</servlet-name>
<url-pattern>/student</url-pattern>
</servlet-mapping>
<welcome-file-list>
<welcome-file>index.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

❖ Student.java

```java
package com.texas.servlets;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.GenericServlet;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class Student extends GenericServlet {

public void init(ServletConfig config) throws ServletException {
}

public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException {
PrintWriter pw =  response.getWriter();
pw.println("Welcome to GenericServlet!");
pw.close();
}

public void destroy() {
}

}
```

2. javax.servlet.http :

The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.



HttpServletRequest

To provide client HTTP request information for servlets. It extends the ServletRequest interface.

HttpServletResponse

To provide HTTP-specific functionality in sending a response to client. It extends the ServletResponse interface.

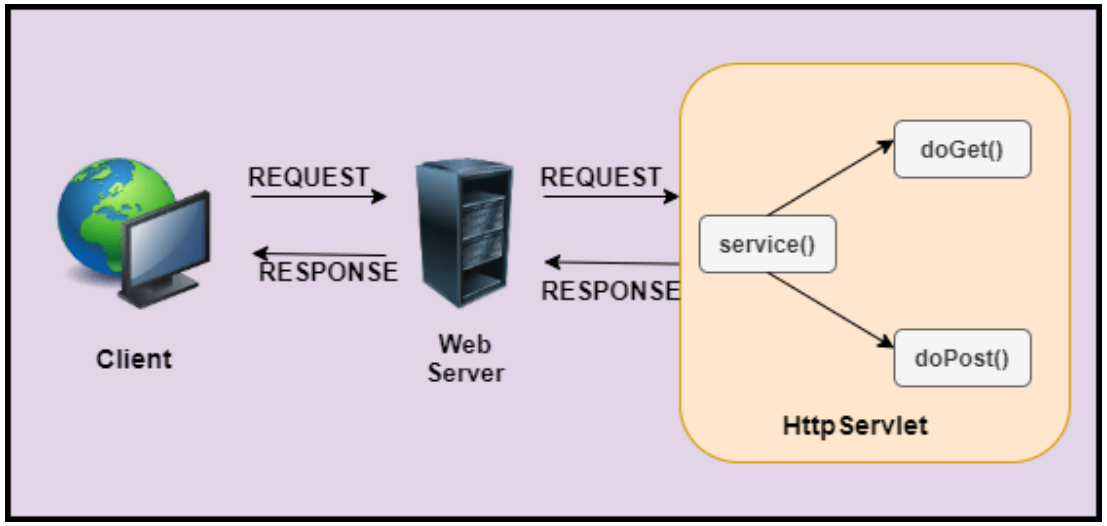
HttpSession

It provides a way to identify a user across web application/web site pages and to store information about that user.

HttpServlet :

- HttpServelt is an abstract class, it comes under package 'javax.servlet.http.HttpServlet' .
- To create a servlet the class must extend the HttpServlet class and override at least one of its methods (doGet, doPost, doDelete, doPut).
- The HttpServlet class extends the GenericServlet class and implements a Serializable interface.
- HttpServlet can process multiple clients requesting from multiple HTML forms hence programmer prefer this servlet.

How HttpServlet Works?



1. The client(web browser) requests the server.

2. These requests can be of any type like- **GET, POST,** etc. Now Web Server will dispatch these requests to the **service()** method for handling the request.

3. Now **service()** will see if it is a **GET** request it will dispatch to **doGet()** method or if it is a **POST** request it will dispatch to **doPost()** and likewise.

Methods of HttpServlet class

1. public void service(ServletRequest req,ServletResponse res) :
- dispatches the request to the protected service method by converting the request and response object into http type.

2. protected void service(HttpServletRequest req, HttpServletResponse res):
- receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.

3. protected void doGet(HttpServletRequest req, HttpServletResponse res)
- handles the GET request. It is invoked by the web container.

4. protected void doPost(HttpServletRequest req, HttpServletResponse res)
- handles the POST request. It is invoked by the web container.

5. protected void doHead(HttpServletRequest req, HttpServletResponse res)
- handles the HEAD request. It is invoked by the web container.

6. protected void doPut(HttpServletRequest req, HttpServletResponse res)
- handles the PUT request. It is invoked by the web container.

7. protected void doTrace(HttpServletRequest req, HttpServletResponse res)
- handles the TRACE request. It is invoked by the web container.

8. protected void doDelete(HttpServletRequest req, HttpServletResponse res)
- handles the DELETE request. It is invoked by the web container.

Example :

❖ httpServletDemo.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>HttpServletDemo</title>
</head>
<body>
<h2>HttpServletDemo</h2>
<a href="user">User</a>
<hr>
<form method="post" action="user">
<label>FirstName : </label>
<input type="text" name="firstName"/><br><br>
<label>LastName : </label>
<input type="text" name="lastName"/><br><br>
<label>Gender : </label>
<input type="radio" name="gender" value="Male"/> Male
<input type="radio" name="gender" value="FeMale"/> FeMale
<input type="radio" name="gender" value="Other"/> Other<br><br>
<input type="submit" value="Register"/>
</form>
</body>
</html>
```


❖ web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
id="WebApp_ID" version="4.0">
<display-name>HttpServletDemo</display-name>
<servlet>
<servlet-name>httpDemo</servlet-name>
<servlet-class>com.texas.servlets.User</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>httpDemo</servlet-name>
<url-pattern>/user</url-pattern>
</servlet-mapping>
<welcome-file-list>
<welcome-file>httpServletDemo.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

❖ User.java

```
package com.texas.servlets;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class User extends HttpServlet {
String Get_Method = "GET";
String POST_Method = "POST";
public void init(ServletConfig config) {
}
public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException {
HttpServletRequest req = (HttpServletRequest) request;
HttpServletResponse res = (HttpServletResponse) response;
service(req, res);
}
protected void service(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
String methodName = request.getMethod();
if (methodName.equalsIgnoreCase(Get_Method)) {
doGet(request, response);
} else if (methodName.equalsIgnoreCase(POST_Method)) {
doPost(request, response);
} else {
}
}

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
PrintWriter pw = response.getWriter();
pw.println("HttpServlet doGet method");
pw.close();
}

public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
String firstName = request.getParameter("firstName");
String lastName = request.getParameter("lastName");
String gender = request.getParameter("gender");
PrintWriter pw = response.getWriter();
pw.println("User Name : " + firstName + " " + lastName);
pw.println("Gender : " + gender);
pw.close();
}
public void destroy() {
}
}
```

Difference :

#	GenericServlet	HttpServlet
1.	GenericServlet defines a generic and protocol-independent servlet.	But HttpServlet defines a HTTP protocol specific servlet.
2.	In GenericServlet, the **service()** method is abstract.	But **service()** method in HttpServlet is non-abstract.
3.	GenericServlet extends Object class and implements Servlet, ServletConfig, and Serializable interfaces.	But HttpServlet extends GenericServlet and implements a Serializable interface.
4.	GenericServlet supports only service() method which does not contain doGet() and doPost() methods.	HttpServlet also supports doGet(), doPost(), doHead(), doPut(), doOptions(), doDelete(), doTrace() methods.
5.	GenericServlet can process multiple clients request from a single form hence programmer doesn't prefer this servlet.	HttpServlet can process multiple clients requesting from multiple HTML forms hence programmer prefer this servlet.
6.	GenericServlet is Stateless and is not commonly used servlet.	But HttpServlet is Stateful and it is most popular servlet because it is commonly used by the developer.

	Servlet	GenericServlet	HttpServlet
What it is?	Interface	Abstract Class	Abstract Class
Package	javax.servlet	javax.servlet	javax.servlet.http
Hierarchy	Top level interface	Implements Servlet interface	Extends GenericServlet
Methods	init(), service(), destroy(), getServletConfig(), getServletInfo()	init(), service(), destroy(), getServletConfig(), getServletInfo(), log(), getInitParameter(), getInitParameterNames(), getServletContext(), getServletName()	doGet(), doPost(), doPut(), doDelete(), doHead(), doOptions(), doTrace(), getLastModified(), service()
Abstract Methods	All methods are abstract.	Only service() method is abstract.	No abstract methods.
When to use?	Use it when you want to develop your own Servlet container.	Use to write protocol independent servlets.	Use to write HTTP-specific servlets.

Reading Servlet Parameters :

- The **ServletRequest** and **HttpServletRequest** interface includes methods that allow you to read the names and values of parameters that are included in a client request.

Methods of ServletRequest interface

Method	Description
public String getParameter(String name)	is used to obtain the value of a parameter by name.
public String[] getParameterValues(String name)	returns an array of String containing all values of given parameter name. It is mainly used to obtain values of a Multi select list box.
java.util.Enumeration getParameterNames()	returns an enumeration of all of the request parameter names.
public int getContentLength()	Returns the size of the request entity data, or -1 if not known.
public String getCharacterEncoding()	Returns the character set encoding for the input of this request.
public String getContentType()	Returns the Internet Media Type of the request entity data, or null if not known.
public ServletInputStream getInputStream() throws IOException	Returns an input stream for reading binary data in the request body.
public abstract String getServerName()	Returns the host name of the server that received the request.
public int getServerPort()	Returns the port number on which this request was received.

Example :

❖ ReadParameter.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>HttpServletDemo</title>
</head>
<body>
<h2>HttpServletDemo</h2>
<hr>
<form method="post" action="user">
<label>FirstName : </label>
<input type="text" name="firstName"/><br><br>
<label>LastName : </label>
<input type="text" name="lastName"/><br><br>
<label>Gender : </label>
<input type="radio" name="gender" value="Male"/> Male
<input type="radio" name="gender" value="FeMale"/> FeMale
<input type="radio" name="gender" value="Other"/> Other<br><br>
<input type="submit" value="Register"/>
</form>
</body>
</html>
```


❖ web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
id="WebApp_ID" version="4.0">
<display-name>ReadParameter</display-name>
<servlet>
<servlet-name>httpDemo</servlet-name>
<servlet-class>com.texas.servlets.User</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>httpDemo</servlet-name>
<url-pattern>/user</url-pattern>
</servlet-mapping>
<welcome-file-list>
<welcome-file>ReadParameter.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

❖ User.java

```
package com.texas.servlets;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class User extends HttpServlet {

public void init(ServletConfig config) {
}

public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
String firstName = request.getParameter("firstName");
String lastName = request.getParameter("lastName");
String gender = request.getParameter("gender");
PrintWriter pw = response.getWriter();
pw.println("User Name : " + firstName + " " + lastName);
pw.println("Gender : " + gender);
pw.close();
}
public void destroy() {
}
}
```

RequestDispatcher :

- The **RequestDispatcher** interface defines an object that receives the request from client and dispatches it to the resource(such as servlet, JSP, HTML file).
- A RequestDispatcher object can be used to forward a request to the resource or to include the resource in a response. The resource can be dynamic or static.

Create an Object of RequestDispatcher :

RequestDispatcher rd = request.getRequestDispatcher("String path");

Description:

- request is the HttpServletRequest type object.

- path is a string specifying the pathname to the resource. If it is relative, it must be relative to the current servlet.

This interface has following two methods:

1. **forward :**

Syntax:

void forward(ServletRequest request, ServletResponse response) throws ServletException,IOException

Description:

- ***Modifier and Type:-*** *void*
- *This method is used to forward a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.*

- The method get called before the response has been sent to the client. If the response is already sent then the method will throws an IllegalStateException.
- The parameter request(HttpServletRequest type) and response(HttpServletResponse type) are the same objects as were passed to the calling servlet's service method.
- This method sets the dispatcher type of the given request to DispatcherType.FORWARD.

2. include :

Syntax:

void include(ServletRequest request, ServletResponse response) throws ServletException,IOException

Description:

- **Modifier and Type**:- void
- This method is used to include the response of resource(for which the request passed servlet, JSP page, HTML file) in the current servlet response.
- The parameter request(HttpServletRequest type) and response(HttpServletResponse type) are the same objects as were passed to the calling servlet's service method.
- This method sets the dispatcher type of the given request to DispatcherType.INCLUDE.

Example :

❖ login.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>login</title>
</head>
<body>
<form method="post" action="login">
<label>UserName : </label>
<input type="text" name="txtUserName"/><br><br>
<label>Password : </label>
<input type="password" name="txtPassword"/> <br><br>
<input type="submit" value="Login"/>
</form>
</body>
</html>
```

❖ web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd" id="WebApp_ID" version="4.0">
  <display-name>RequestDispatcherDemo</display-name>
  <servlet>
  <servlet-name>Demo1</servlet-name>
  <servlet-class>com.texas.servlets.Home</servlet-class>
  </servlet>

  <servlet-mapping>
  <servlet-name>Demo1</servlet-name>
  <url-pattern>/home</url-pattern>
  </servlet-mapping>

  <servlet>
  <servlet-name>Demo2</servlet-name>
  <servlet-class>com.texas.servlets.Login</servlet-class>
  </servlet>

  <servlet-mapping>
  <servlet-name>Demo2</servlet-name>
  <url-pattern>/login</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
    <welcome-file>login.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

❖ Login.java

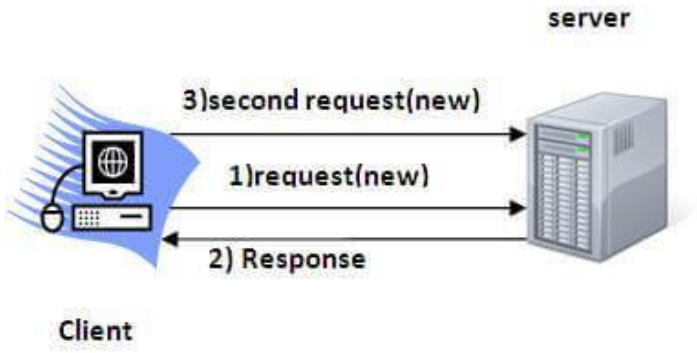
```java
package com.texas.servlets;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class Login extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

String txtUserName  = request.getParameter("txtUserName");
String txtPassword  = request.getParameter("txtPassword");
PrintWriter pw = response.getWriter();

if(txtUserName.equals("") || txtUserName ==null || txtPassword.equals("") || txtPassword==null) {
pw.println("Please enter both credentials.");
RequestDispatcher rd = request.getRequestDispatcher("login.jsp");
rd.include(request, response);
}

else {

if(txtUserName.equals("Admin") && txtPassword.equals("1234")) {
pw.println("Success!");
RequestDispatcher rd = request.getRequestDispatcher("home");
rd.forward(request, response);
}

else {
pw.println("Not Authorized!");
RequestDispatcher rd = request.getRequestDispatcher("login.jsp");
rd.include(request, response);
}
}
}
}
```

❖ Home.java

```java
package com.texas.servlets;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class Home extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
String txtUserName  = request.getParameter("txtUserName");
String txtPassword  = request.getParameter("txtPassword");
PrintWriter pw = response.getWriter();
pw.println("UserName : "+txtUserName);
pw.close();
}
}
```

Session Management/ Session Tracking :

- **Session Tracking** is a way to maintain state (data) of an user. It is also known as **session management** in servlet.
- Http protocol is a stateless, so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So, we need to maintain the state of an user to recognize to particular user.
- HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below:



- In order to maintain an identity of the client for a period of time. we have four types of session management/ tracking techniques :

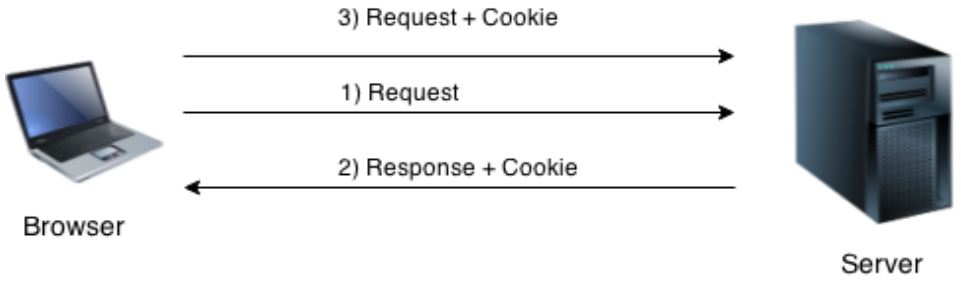
1. Cookies
2. HttpSession
3. Hidden Form field
4. URL rewritten

1. Cookies :

- Cookies are small piece of information that are sent in response from the web server to the client.
- Cookies are the simplest technique used for storing client state.
- Cookies are stored on client computer. They have a lifespan and are destroyed by the client browser at the end of that lifespan.

How Cookie works

- By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. 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.



Types of Cookie :

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

1. Non-persistent cookie :
It is **valid for single session** only. It is removed each time when user closes the browser.

2. Persistent cookie :
It is **valid for multiple session** . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

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.
- ❖ Only textual information can be set in Cookie object.

Cookie class :
- ❖ **javax.servlet.http.Cookie** class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

Constructor of Cookie class
1. **Cookie**() : *constructs a cookie.*
2. **Cookie**(String name, String value) : *constructs a cookie with a specified name and value.*

Useful Methods of Cookie class

1. public void setMaxAge(int expiry) : *Sets the maximum age of the cookie in seconds.*
2. public String getName() : *Returns the name of the cookie. The name cannot be changed after creation.*
3. public String getValue() : *Returns the value of the cookie.*
4. public void setName(String name) : *changes the name of the cookie.*
5. public void setValue(String value) : *changes the value of the cookie.*
6. public void addCookie(Cookie ck) : *used to add cookie.*
7. public Cookie[] getCookies() : *used to return all the cookies*

Example :

❖ login.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Login</title>
</head>
<body>
<form method="post" action="login">
<label>UserName : </label> <input type="text" name="userName" /><br><br>
 <label>Password : </label> <input type="password" name="userPassword" /><br><br>
<input type="submit" value="Login" />
</form>
</body>
</html>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
id="WebApp_ID" version="4.0">
<display-name>CookieInServlet</display-name>
<servlet>
<servlet-name>Demo1</servlet-name>
<servlet-class>com.texas.servlets.Login</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>Demo1</servlet-name>
<url-pattern>/login</url-pattern>
</servlet-mapping>
<servlet>
<servlet-name>Demo2</servlet-name>
<servlet-class>com.texas.servlets.LogOut</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>Demo2</servlet-name>
<url-pattern>/logout</url-pattern>
</servlet-mapping>
<welcome-file-list>
<welcome-file>login.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

❖ Login.java

```java
package com.texas.servlets;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class Login extends HttpServlet {

public void init(ServletConfig config) {
}

public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException{
String userName = request.getParameter("userName");
String userPassword = request.getParameter("userPassword");
PrintWriter pw =  response.getWriter();

if(userName.equals("")|| userName==null || userPassword.equals("") || userPassword==null) {
pw.println("Please enter both username and password.");
RequestDispatcher rd = request.getRequestDispatcher("login.jsp");
rd.include(request, response);
}else {

if(userName.equals("Admin") && userPassword.equals("1234")) {
Cookie cookie = new Cookie("userDetails", userName);
cookie.setMaxAge(60*1000);
response.addCookie(cookie);
response.sendRedirect("home.jsp");
}else {
pw.println("User is not aauthorized");
RequestDispatcher rd = request.getRequestDispatcher("login.jsp");
rd.include(request, response);
}
}
pw.close();
}

public void destroy() {
}
}
```

❖ home.jsp

```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Home</title>
</head>
<body>
<%
String cookieUservalue = null;
Cookie[] cookies =  request.getCookies();
if(cookies!=null) {
for(Cookie cookie : cookies) {

if(cookie.getName().equals("userDetails")) {
cookieUservalue = cookie.getValue();
}
}
}
if(cookieUservalue==null) {
response.sendRedirect("login.jsp");
}
%>
<hr>
Home || <a href="logout">LogOut</a>
<hr>
<p> User Value using Cookie : <%=cookieUservalue%></p>
</body>
</html>
```

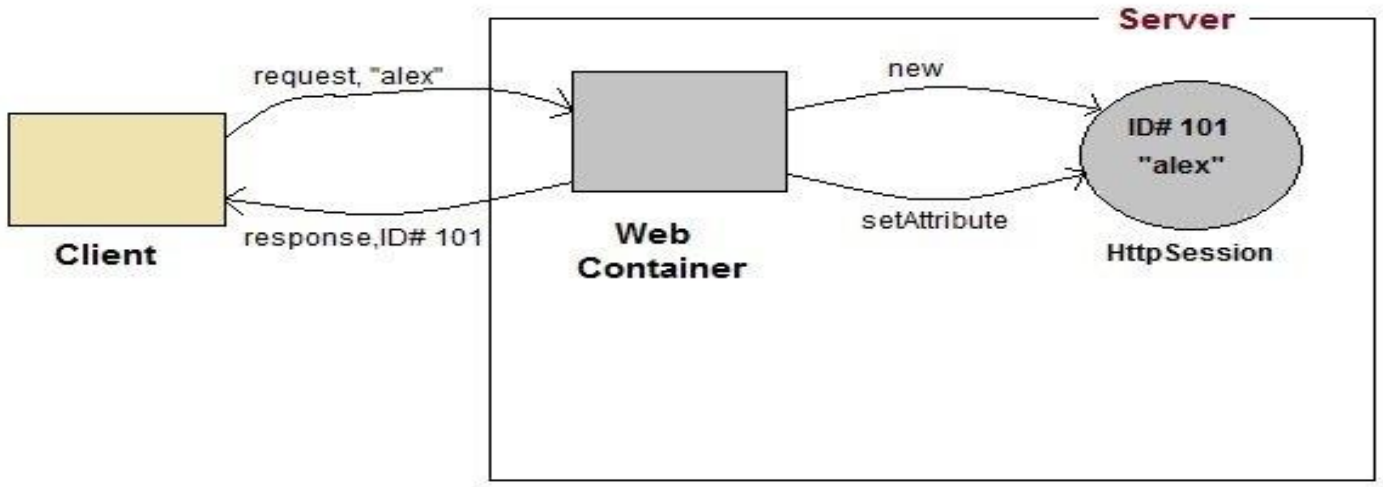
❖ LogOut.java

```
package com.texas.servlets;

import java.io.IOException;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class LogOut extends HttpServlet {
public void init(ServletConfig config) {
}
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException{
Cookie[] cookies = request.getCookies();
Cookie cookieUser = null;
String uservalue = null;
if(cookies!=null) {
for(Cookie cookie : cookies) {
if(cookie.getName().equals("userDetails")) {
cookieUser = cookie;
uservalue=  cookie.getValue();
break;
}
}
}
if(cookieUser!=null || uservalue!=null) {
cookieUser.setMaxAge(0);
response.addCookie(cookieUser);
}
response.sendRedirect("login.jsp");
}

public void destroy() {
}
}
```

HttpSession

- **HttpSession** object is used to store entire session with a specific client. We can store, retrieve and remove attribute from **HttpSession** object.
- Any servlet can have access to **HttpSession** object throughout the getSession() method of. the **HttpServletRequest** object.

How HttpSession works



1. On client's first request, the **Web Container** generates a unique session ID and gives it back to the client with response. This is a temporary session created by web container.

2. The client sends back the session ID with each request. Making it easier for the web container to identify where the request is coming from.

3. The **Web Container** uses this ID, finds the matching session with the ID and associates the session with the request.

Get the HttpSession object :

The HttpServletRequest interface provides two methods to get the object of HttpSession:

1. **public HttpSession getSession():**

   Returns the current session associated with this request, or if the request does not have a session, creates one.

2. **public HttpSession getSession(boolean create):**

   Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

Methods of HttpSession :

1. public void setAttribute(String name, Object value):

Binds the object with a name and stores the name/value pair as an attribute of the HttpSession object. If an attribute already exists, then this method replaces the existing attributes.

2. public Object getAttribute(String name):

Returns the String object specified in the parameter, from the session object. If no object is found for the specified attribute, then the getAttribute() method returns null.

3. public Enumeration getAttributeNames():

Returns an Enumeration that contains the name of all the objects that are bound as attributes to the session object.

4. public void removeAttribute(String name):

Removes the given attribute from session.

5. setMaxInactiveInterval(int interval):

Sets the session inactivity time in seconds. This is the time in seconds that specifies how long a sessions remains active since last request received from client.

Example :

❖ login.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Login</title>
</head>
<body>
<form method="post" action="login">
<label>UserName : </label> <input type="text" name="userName" /><br><br>
 <label>Password : </label> <input type="password" name="userPassword" /><br><br>
<input type="submit" value="Login" />
</form>
</body>
</html>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
id="WebApp_ID" version="4.0">
<display-name>CookieInServlet</display-name>
<servlet>
<servlet-name>Demo1</servlet-name>
<servlet-class>com.texas.servlets.Login</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>Demo1</servlet-name>
<url-pattern>/login</url-pattern>
</servlet-mapping>
<servlet>
<servlet-name>Demo2</servlet-name>
<servlet-class>com.texas.servlets.LogOut</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>Demo2</servlet-name>
<url-pattern>/logout</url-pattern>
</servlet-mapping>
<welcome-file-list>
<welcome-file>login.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

❖ Login.java

```java
package com.texas.servlets;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class Login extends HttpServlet {
public void init(ServletConfig config) {
}
public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
String userName = request.getParameter("userName");
String userPassword = request.getParameter("userPassword");
PrintWriter pw = response.getWriter();
if (userName.equals("") || userName == null || userPassword.equals("") || userPassword == null) {
pw.println("Please enter both username and password.");
RequestDispatcher rd = request.getRequestDispatcher("login.jsp");
rd.include(request, response);
} else {
if (userName.equals("Admin") && userPassword.equals("1234")) {
HttpSession httpSession = request.getSession();
httpSession.setAttribute("userName", userName);
httpSession.setAttribute("Password", userPassword);
response.sendRedirect("home.jsp");
} else {
pw.println("User is not aauthorized");
RequestDispatcher rd = request.getRequestDispatcher("login.jsp");
rd.include(request, response);
}
}
pw.close();
}
public void destroy() {
}
}
```

❖ home.jsp

```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Home</title>
</head>
<body>
<%
String userName = null;
String Password = null;
if(session.getAttribute("userName")!=null && session.getAttribute("Password")!=null){
userName = session.getAttribute("userName").toString();
Password = session.getAttribute("Password").toString();
}else{
response.sendRedirect("login.jsp");
}
%>
<hr>
Home || <a href="logout">LogOut</a>
<hr>
<p> User Value using httpSession : <%=userName%></p>
</body>
</html>
```

❖ LogOut.java

```java
package com.texas.servlets;

import java.io.IOException;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class LogOut extends HttpServlet {

public void init(ServletConfig config) {
}

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException{
HttpSession httpSession =  request.getSession(false);
String username = null;
String userpassword = null;
if(httpSession.getAttribute("userName")!=null && httpSession.getAttribute("Password")!=null) {
username = httpSession.getAttribute("userName").toString();
userpassword = httpSession.getAttribute("Password").toString();
}
if(username!=null && username!=null) {
httpSession.setAttribute("userName", null);
httpSession.setAttribute("Password", null);
httpSession.invalidate();
}
response.sendRedirect("login.jsp");
}

public void destroy() {
}
}
```

JSP

- Java Server pages is a web page development technology that support dynamic content.
- This allow programmers to use specific JSP tags to insert Java code into HTML pages.
- It is an advanced version of Servlet Technology.
- JSP is first converted into servlet by JSP container before processing the client's request.

Features of JSP

1. **Coding in JSP is easy** :- As it is just adding JAVA code to HTML.
2. **Reduction in the length of Code** :- In JSP we use action tags, custom tags etc.
3. **Connection to Database is easier** :-It is easier to connect website to database and allows to read or write data easily to the database.
4. **Make Interactive websites** :- In this we can create dynamic web pages which helps user to interact in real time environment.
5. **Portable, Powerful, flexible and easy to maintain** :- as these are browser and server independent.
6. **Extension to Servlet** :- as it has all features of servlets, implicit objects and custom tags.

Implicit Objects

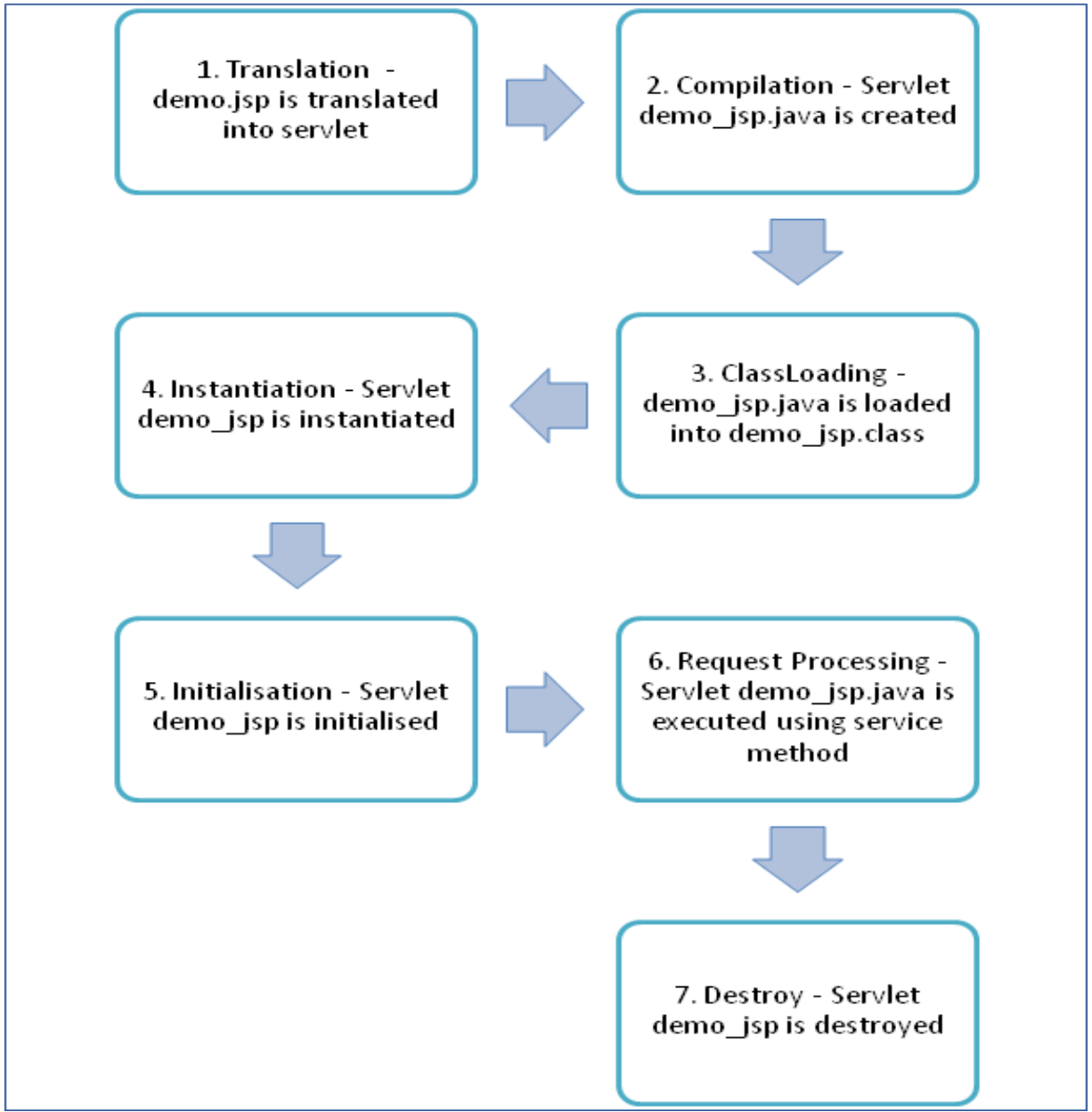
- There are **9 jsp implicit objects**. These objects are *created by the web container* that are available to all the jsp pages.

A list of the 9 implicit objects is given below:

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

JSP Life Cycle :

- A JSP life cycle is defined as the process from its creation till the destruction.
- This is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet.



1. Translation of the JSP Page:

- A Java servlet file is generated from a JSP source file. This is the first step of JSP life cycle.
- In translation phase, container validates the syntactic correctness of JSP page and tag files.
- The JSP container interprets the standard directives and actions, and the custom actions referencing tag libraries (they are all part of JSP page and will be discussed in the later section) used in this JSP page.
- In the above pictorial description, demo.jsp is translated to demo_jsp.java in the first step

2. Compilation of the JSP Page :

- The generated java servlet file is compiled into java servlet class.
- The translation of java source page to its implementation class can happen at any time between the deployment of JSP page into the container and processing of the JSP page.
- In the above pictorial description demo_jsp.java is compiled to a class file demo_jsp.class

3. Classloading

- Servlet class that has been loaded from JSP source is now loaded into the container

4. Instantiation

- In this step the object i.e. the instance of the class is generated.
- The container manages one or more instances of this class in the response to requests and other events.
- Typically, a JSP container is built using a servlet container. A JSP container is an extension of servlet container as both the container support JSP and servlet.
- A JSPPage interface which is provided by container provides init() and destroy () methods.

5. Initialization :

```
public void jspInit() {
//initializing the code
}
```

- _jspinit() method will initiate the servlet instance which was generated from JSP and will be invoked by the container in this phase.
- Once the instance gets created, init method will be invoked immediately after that It is only called once during a JSP life cycle, the method for initialization is declared as shown above

6. Request processing :

```
void _jspservice(HttpServletRequest request HttpServletResponse response) {

//handling all request and responses

}
```

- _jspservice() method is invoked by the container for all the requests raised by the JSP page during its life cycle.
- For this phase, it has to go through all the above phases and then only service method can be invoked.
- It passes request and response objects
- This method cannot be overridden
- The method is shown above: It is responsible for generating of all HTTP methods i.eGET, POST, etc.

7. Destroy :

```
public void _jspdestroy() {
  //all clean up code

}
```

- _jspdestroy() method is also invoked by the container
- This method is called when container decides it no longer needs the servlet instance to service requests.
- When the call to destroy method is made then, the servlet is ready for a garbage collection
- This is the end of the life cycle.
- We can override jspdestroy() method when we perform any cleanup such as releasing database connections or closing open files.

- **JSP Scripting elements / JSP Tags**

- JSP Scripting element are written inside <% %> tags.
- These code inside <% %> tags are processed by the JSP engine during translation of the JSP page.
- Any other text in the JSP page is considered as HTML code or plain text.

- **There are five different types of scripting elements :**

Scripting Element	Example
Comment	<%-- comment --%>
Directive	<%@ directive %>
Declaration	<%! declarations %>
Scriptlet	<% scriplets %>
Expression	<%= expression %>

1. JSP Comment :

- JSP Comment is used when you are creating a JSP page and want to put in comments about what you are doing.
- These comments are not included in servlet source code during translation phase, nor they appear in the HTTP response.

Syntax :

<%-- comment --%>

2. Scriptlet Tag :

- Scriptlet Tag allows you to write java code inside JSP page.

Syntax :

<% scriplets %>

3. Declaration Tag :

- Declarations in JSP declare java language statements.
- They can declare classes/instances/inner classes/variables/methods.

Syntax:

<%! declarations %>

4. Expression Tag :

- Expression Tag is used to print out java language expression that is put between the tags.
- An expression tag can hold any java language expression that can be used as an argument to the **out.print()** method.

Syntax:

<%= expression %>

5. Directive Tag :

- Directive Tag gives special instruction to Web Container at the time of page translation. Directive tags are of three types: page, include and taglib.

Directive	Description
<%@ page ... %>	defines page dependent properties such as language, session, errorPage etc.
<%@ include ... %>	defines file to be included.
<%@ taglib ... %>	declares tag library used in the page

❖ Example : index.jsp

```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>JSP Tags</title>
</head>
<body>
<p>JSP Tags Demo</p>
<hr>
<p>JSP Comments</p>
<%-- JSP Comments Example --%>

<p>JSP Scriptlet tags</p>
<%
int a = 10;
int b = 20;
int c = a+b;
out.println("Sum is : "+c);
%>

<p>Declaration Tag</p>
<%!
public int multi(int a , int b){
return a*b;
}
%>
<% out.print(multi(10,40)); %>

<p> Expression Tag</p>
<%=multi(10,4) %>

<p>directive Tag</p>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" %>
<%@ include file="Header.jsp" %>

</body>
</html>
```

❖ web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd" id="WebApp_ID" version="4.0">
  <display-name>JSPTags</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
     </welcome-file-list>
</web-app>
```

❖ Header.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

Home || Profile

</body>
</html>
```

Advantages of JSP

- It is very much convenient to modify the regular HTML. We can write the servlet code into the JSP.
- JSP can also include the database connections into it. It can contain all type of java objects.
- It is very easy to maintain
- Performance and scalability of JSP are very good because JSP allows embedding of dynamic elements in HTML pages.
- As it is built on Java technology, hence it is platform independent and not depending on any operating systems.
- We can also make use of exception handling of java into JSP.
- Also, it includes the feature of multithreading of java into it.
- It is easy for developers to show as well as process the information.
- It enables to separate presentation layer with the business logic layer in the web application.

Disadvantages of JSP

- It is hard to trace JSP pages error because JSP pages are translated to servlet.
- As JSP output is HTML, it is not rich in features.
- JSP pages require more disk space and time to hold JSP pages as they are compiled on the server.

Difference between Servlet and JSP

Servlet	JSP
Servlets are faster as compared to JSP, as they have a short response time.	JSP is slower than Servlets, as the first step in the JSP lifecycle is the conversion of JSP to Java code and then the compilation of the code.
Servlets are Java-based codes.	JSP are HTML-based codes.
Servlets are harder to code, as here, the HTML codes are written in Java.	JSPs are easier to code, as here Java is coded in HTML.
In an MVC architecture, Servlets act as the controllers.	In MVC architectures, the JSPs act as a view to present the output to the users.
The service() function can be overridden in Servlets.	The service() function cannot be overridden in JSPs.
The Servlets are capable of accepting all types of protocol requests.	The JSPs are confined to accept only the HTTP requests.
Modification in Servlets is a time-consuming and challenging task, as here, one will have to reload, recompile, and then restart the servers.	Modification is easy and faster in JSPs as we just need to refresh the pages.
Servlets require us to implement the business logic and presentation logic in the same servlet file.	JSPs give us the flexibility to separate the business logic from the presentation logic using javaBeans.
Servlets can handle extensive data processing.	JSPs cannot handle data processing functions efficiently.
Servlets do not provide the facility of writing custom tags.	JSPs can provide the facility of building the JSP tags easily, which can directly call javaBeans.
In Servlets, we do not have implicit objects.	In JSPs, we have support for implicit objects.
Servlets are hosted and executed on Web Servers.	JSP is compiled in Java Servlets before their execution. After that, it has a similar lifecycle as Servlets.
We need to import all the packages at the top of the Servlets.	In JSPs, we can import packages anywhere in the file.

Java Web Framworks

- Frameworks are tools with pre-written code, act as a template, which can be reused to create an application by simply filling with your code as needed which enables developers to program their application with no overhead of creating each line of code again and again from scratch.

Why do we need framework?

- Virtually all web applications have a common set of basic requirements such as user management e.g. secure user login, password recovery, group management and access authorization. A web application framework usually includes all these functionalities, refined through hundreds of production deployments, freeing developers to focus on the needs of their specific application.

- In high traffic web application like social sites, registration sites etc. Web framework provide excellent support for developing application having a good traffic handling capacity by applying pooling techniques.

- WAFs store important data in a relational database, and they interact with users via a web-based user interface.

Few popular Framework:

1. Spring
2. Hibernate
3. Google Web Toolkit (GWT)
4. JavaServer Faces (JSF)

1. Spring :

It is a light-weighted, powerful Java application development framework. Its other modules are **Spring Security, Spring MVC, Spring Batch, Spring ORM, Spring Boot** and **Spring Cloud** etc.

Advantages :

1. Loose coupling
2. Lightweight
3. Fast Development
4. Powerful abstraction
5. Easy to test

2. Hibernate :

Hibernate is ORM (Object-Relation Mapping) framework that allows us establish communication between Java programming language and the RDBMS.

Advantages :

1. Portability, productivity, maintainability.
2. Open-source framework.
3. It avoids repetitive code from the JDBC API.

Disadvantages of using Framework:

- To be able to use the framework at its best, it often requires significant education and experience.
- If a bug or a security risk in the framework is found it will be in all applications using the framework. Some framework are very stiff and do not give the developer enough flexibility needed for some applications.
- Building from scratch often gives a feeling of more productive which can make the developer more peaceful and less feeling of being stuck and thereby more creative and less bored.