Chapter-10 Java Server (JSP) / Servlet Technology

JSP/Servlet Technology Overview:

Prerequisites

1. *I assume, you have little knowledge of how web applications work over HTTP, what is web server and what is web browsers.*

2. *It will be great if you have some knowledge of web application development using any programming language.*

JSP

- ❖ It stands for **Java Server Pages**.
- ❖ It is a server side technology.
- ❖ It is used for creating web application.
- ❖ It is used to create dynamic web content.
- ❖ In this JSP tags are used to insert JAVA code into HTML pages.
- ❖ It is an advanced version of Servlet Technology.
- ❖ It is a Web based technology helps us to create dynamic and platform independent web pages.
- ❖ In this, Java code can be inserted in HTML/ XML pages or both.
- ❖ JSP is first converted into servlet by JSP container before processing the client's request.

Why JSP pages are more advantageous than Servlet?

- ❖ They are easy to maintain.
- ❖ No recompilation or redeployment is required.
- ❖ JSP has access to entire API of JAVA.
- ❖ JSP are extended version of Servlet.

Features of JSP

- ❖ **Coding in JSP is easy**: - As it is just adding JAVA code to HTML/XML.
- ❖ **Reduction in the length of Code**: - In JSP we use action tags, custom tags etc.
- ❖ **Connection to Database is easier**:-It is easier to connect website to database and allows to read or write data easily to the database.
- ❖ **Make Interactive websites**: - In this we can create dynamic web pages which helps user to interact in real time environment.

❖ **Portable, Powerful, flexible and easy to maintain**: - as these are browser and server independent.
❖ **No Redeployment and No Re-Compilation**:- It is dynamic, secure and platform independent so no need to re-compilation.
❖ **Extension to Servlet** :- as it has all features of servlets, implicit objects and custom tags

JSP syntax

Syntax available in JSP are following
1. *Declaration Tag***:-**It is used to declare variables.

> **Syntax:-**
> <%! Dec var %>
>
> **Example:-**
> <%! int var=10; %>

2. *Java Scriplets***: -** It allows us to add any number of JAVA code, variables and expressions.
> **Syntax:-**
> <% java code %>

3. *JSP Expression***: -** It evaluates and convert the expression to a string.

> **Syntax:-**
> <%= expression %>
> **Example:-**
> <% num1 = num1+num2 %>

4. *JAVA Comments: -* It contains the text that is added for information which has to be ignored.

> **Syntax:-**
> <% -- JSP Comments %>

How to Execute JSP?

Steps for Execution of JSP are following:-

1. Create html page from where request will be sent to server e.g. test.html.
2. To handle to request of user next is to create .jsp file E.g. demo.jsp
3. Create project folder structure.
4. Create XML file e.g. mydemo.xml.
5. Create WAR file.
6. Start Tomcat
7. Run Application

Example of Hello World

Make one **.html** file and **.jsp** file

demo.jsp

```html
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Hello World Lets Learn JSP</title>
</head>
<body>
    <%= "Hello World!" %>
</body>
</html>
```

Advantages of JSP

❖ It does not require advanced knowledge of JAVA
❖ It is capable of handling exceptions
❖ Easy to use and learn
❖ It can tags which are easy to use and understand
❖ Implicit objects are there which reduces the length of code
❖ It is suitable for both JAVA and non-JAVA programmer

Disadvantages of JSP

- ❖ Difficult to debug for errors.
- ❖ First time access leads to wastage of time
- ❖ Its output is HTML which lacks features.

Life cycle of JSP and Writing JSP Pages

A Java Server Page life cycle is defined as the process started with its creation which later translated to a servlet and afterward servlet lifecycle comes into play. This is how the process goes on until its destruction.
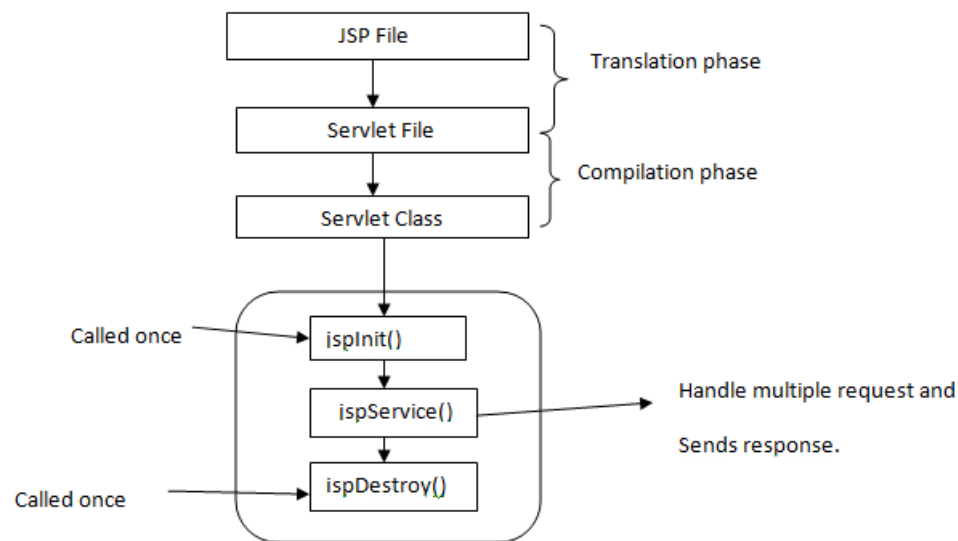
Figure: JSP Life Cycle

Following **steps are involved** in JSP life cycle:

- Translation of JSP page to Servlet
- Compilation of JSP page(Compilation of JSP into test.java)
- Class loading (test.java to test.class)
- Instantiation(Object of the generated Servlet is created)
- Initialization(jspInit() method is invoked by the container)
- Request processing(_jspService()is invoked by the container)
- JSP Cleanup (jspDestroy() method is invoked by the container)

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

Explanation of JSP Life Cycle Step Involved

Translation of JSP page to Servlet:

This is the first step of JSP life cycle. This translation phase deals with Syntactic correctness of JSP. Here test.jsp file is translated to test.java.

Compilation of JSP page:

Here the generated java servlet file (test.java) is compiled to a class file (test.class).

Class loading:

Servlet class which has been loaded from JSP source is now loaded into container.

Instantiation:

Here instance of the class is generated. The container manages one or more instance by providing response to requests.

Initialization:

jspInit() method is called only once during the life cycle immediately after the generation of Servlet instance from JSP.

Request processing:

_jspService() method is used to serve the raised requests by JSP. It takes request and response object as parameters. This method cannot be overridden.

JSP Cleanup:

In order to remove the JSP from use by the container or to destroy method for servlets jspDestroy()method is used. This method is called once, if you need to perform any cleanup task like closing open files, releasing database connections jspDestroy() can be overridden.

Develop the first JSP Page that Gives "Hello World" to the web browser.

Assume that you have installed NetBeans IDE and Apache Tomcat Server on your PC

Step-1 Lunch the NetBeans IDE

Step-2 Create new web application by **File→New Project→Java Web→Web Application** then click **next**

Step-3 Enter the name of project: Hello World and Location where you want to store project file Eg. D/JavaJSP

Step-4 Choose the server so that you can execute the JSP by clicking on server→ add then select TomCat

Step-5 Don't use any framework then just click **Finish** Project Hello World is created then

Step-6 NetBeans generates the project skeleton for you with a JSP page: **index.jsp**

Step-7 Replace the code in the **index.jsp** file by the following code snippet
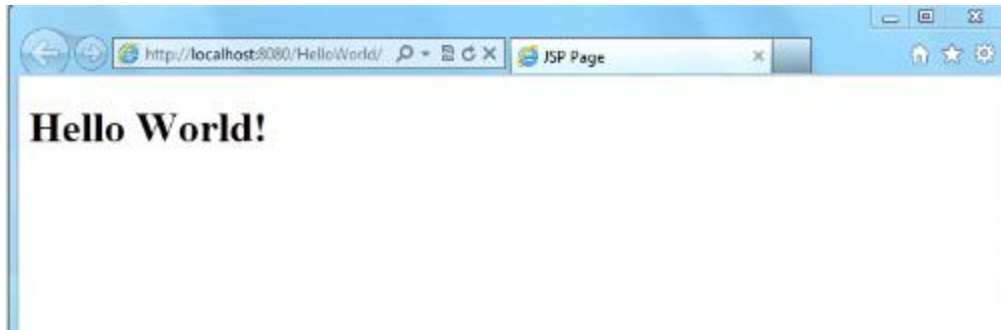
```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title> It's My First JSP Page</title>
    </head>
    <body>
        <h1><% out.println("Hello World!");%></h1>
    </body>
</html>
```

Step-8 Click the **run project** button to run **index.jsp** file

Output of the project is:

It displays the message "Hello World" as a standard heading H1 of the HTML page.

The *index.jsp* page is composed of HTML and Java code. The Java code is embedded inside the HTML document between notations <% and %>.

<% out.println("Hello World!");%>

In JSP, The code snippet above is known as **Scriptlet.** Inside the scriptlet block, we used the method println() of the out object to print the message "Hello World".

Servlet Technology

Servlets are the Java programs that runs on the Java-enabled web server or application server. They are used to handle the request obtained from the web server, process the request, produce the response, and then send response back to the web server.

<div align="center">OR</div>

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.

Also servlet can be defined as:

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

<div align="center">7</div>

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.

There are many interfaces and classes in the Servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse, etc.

Servlet Architecture
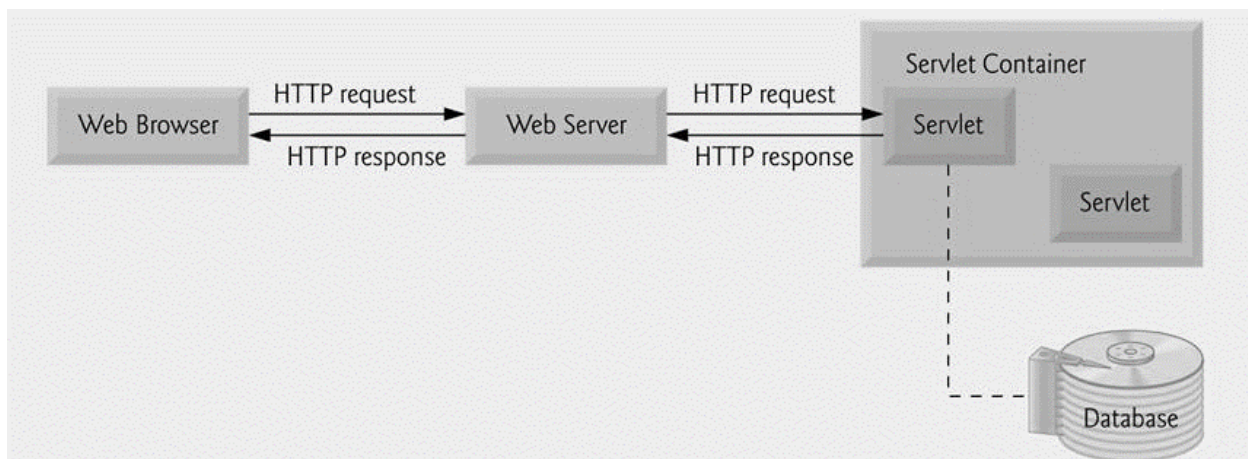
How Servlet Executes?

Execution of Servlets involves the six basic steps:

1. The clients send the request to the web server.
2. The web server receives the request.
3. The web server passes the request to the corresponding servlet.
4. The servlet processes the request and generate the response in the form of output.
5. The servlet send the response back to the web server.
6. The web server sends the response back to the client and the client browser displays it on the screen.

Servlet life Cycle

Servlet life cycle can be defined as the stages through which the servlet passes from its creation to its destruction.

The Servlet life cycle mainly goes through four stages,

❖ Loading a Servlet.
❖ Initializing the Servlet.
❖ Request handling.
❖ Destroying the Servlet.

OR

❖ Servlet is borne
❖ Servlet is initialized
❖ Servlet is ready to service
❖ Servlet is servicing
❖ Servlet is not ready to service
❖ Servlet is destroyed

And Goes through THREE methods

There are three life cycle methods of a Servlet:
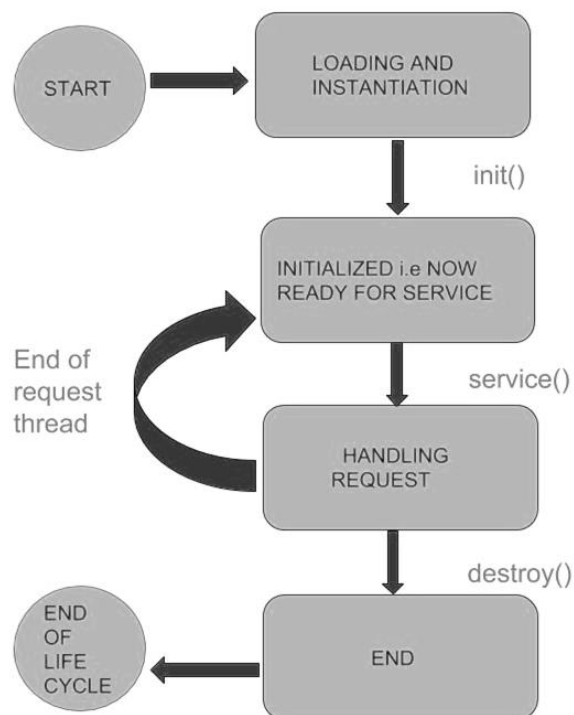
• init()
• service()
• destroy()

Figure: Servlet Life Cycle

Loading a Servlet:

- ❖ First stage of the Servlet lifecycle
- ❖ It involves loading and initializing the Servlet by the Servlet container.
- ❖ The Web container or Servlet Container can load the Servlet at either of the following two stages:
    1. Initializing the context, on configuring the Servlet with a zero or positive integer value.
    2. If the Servlet is not preceding stage, it may delay the loading process until the Web container determines that this Servlet is needed to service a request.
- ❖ The Servlet container performs two operations in this stage:
    - ✓ **Loading:** Loads the Servlet class.
    - ✓ **Instantiation:** Creates an instance of the Servlet. To create a new instance of the Servlet, the container uses the no-argument constructor.

Initializing a Servlet:

- ❖ After the Servlet is instantiated successfully, the Servlet container initializes the instantiated Servlet object.
- ❖ The container initializes the Servlet object by invoking the **Servlet.init(ServletConfig)** method which accepts ServletConfig object reference as parameter.
- ❖ The Servlet container invokes the **Servlet.init(ServletConfig)** method only once, immediately after the **Servlet.init(ServletConfig)** object is instantiated successfully. This method is used to initialize the resources, such as JDBC datasource.
- ❖ If the Servlet fails to initialize, then it informs the Servlet container by throwing the **ServletException** or **UnavailableException**.

Handling request:

- ❖ After initialization, the Servlet instance is ready to serve the client requests. The Servlet container performs the following operations when the Servlet instance is located to service a request:
    - ✓ Creates the **ServletRequest** and **ServletResponse** objects. In this case, if this is a HTTP request then the Web container creates **HttpServletRequest** and **HttpServletResponse** objects which are subtypes of the **ServletRequest** and **ServletResponse** objects respectively.

- ✓ After creating the request and response objects it invoke the Servlet.service (ServletRequest, ServletResponse) method by passing the request and response objects.
- ❖ The **service ()** method while processing the request may throw the **ServletException**or **UnavailableException** or **IOException**.

Destroying a Servlet:

- ❖ When a Servlet container decides to destroy the Servlet, it performs the following operations,
  - ✓ It allows all the threads currently running in the service method of the Servlet instance to complete their jobs and get released.
  - ✓ After currently running threads have completed their jobs, the Servlet container calls the **destroy()** method on the Servlet instance.
- ❖ After the **destroy()** method is executed, the Servlet container releases all the references of this Servlet instance so that it becomes eligible for garbage collection.

1. **init() method**: The **Servlet.init()** method is called by the Servlet container to indicate that this Servlet instance is instantiated successfully and is about to put into service.

//init() method

```
public class MyServlet implements Servlet{
  public void init(ServletConfig config) throws ServletException {
      //initialization code
  }
   //rest of code
}
```

2. **service() method**: The **service()** method of the Servlet is invoked to inform the Servlet about the client requests.
   - This method uses **ServletRequest** object to collect the data requested by the client.
   - This method uses **ServletResponse** object to generate the output content.

// service() method

public class MyServlet implements Servlet{

    public void service(ServletRequest res, ServletResponse res)

    throws ServletException, IOException {

        // request handling code

    }

    // rest of code

}

3. **destroy() method**: The **destroy()** method runs only once during the lifetime of a Servlet and signals the end of the Servlet instance.

//destroy() method

public void destroy()

As soon as the **destroy()** method is activated, the Servlet container releases the Servlet instance.

Life cycle methods

Life cycle methods are those methods which are used to control the life cycle of the servlet. These methods are called in specific order during the servlets's entire life cycle.
The class **Servlet** provides the methods to control and supervise the life cycle of servlet. There are three life cycle methods in the Servlet interface. There are as follows:
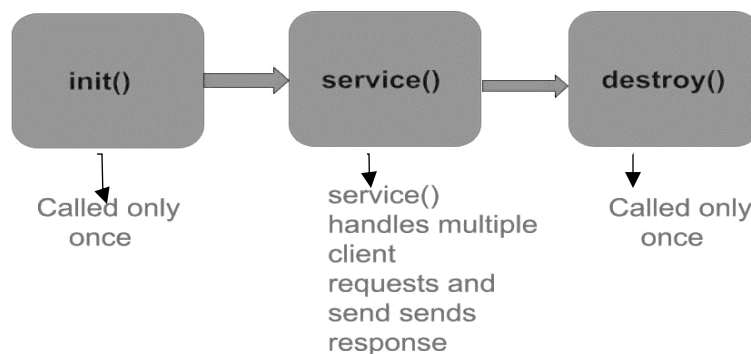

Figure: Life Cycle Method of Servlet

- **init() method :**
  - ❖ A servlet's life begins here.
  - ❖ This method is called only once to load the servlet.
  - ❖ This method receives only one parameter i.e **ServletConfig** object.
  - ❖ This method has the possibility to throw the ServletException.
  - ❖ Once the servlet is initialized, it is ready to handle the client request.
  - ❖ The prototype for the init() method:

  public void init(ServletConfig con)throws ServletException{ }

  where **con** is ServletConfig object

- **service() method :**
  - ❖ The service() method is the most important method to perform that provides the connection between client and server.
  - ❖ The web server calls the service() method to handle requests coming from the client(web browsers) and to send response back to the client.
  - ❖ This method determines the type of Http request (GET, POST, PUT, DELETE, etc.) .
  - ❖ This method also calls various other methods such as doGet(), doPost(), doPut(), doDelete(), etc. as required.
  - ❖ This method accepts two parameter.
  - ❖ The prototype for this method:

  public void service(ServletRequest req, ServletResponse resp)

  throws ServletException, IOException { }

  Where,

  - **req** is the ServletRequest object which encapualates the connection from client to server
  - **resp** is the ServletResponse object which encapualates the connection from server back to the client

- **destroy() method :**

  - ❖ The destroy() method is called only once .
  - ❖ It is called at the end of the life cycle of the servlet.

❖ This method performs various tasks such as closing connection with the database, releasing memory allocated to the servlet, releasing resources that are allocated to the servlet and other cleanup activities.

❖ When this method is called, the garbage collector comes into action

❖ The prototype for this method is:

Public void destroy() { // Finalization code...}

The Servlet Container

It is a system that manages Java Servlet components on top of the Web server to handle the Web client requests.

Services provided by the Servlet container:

- **Network Services:** Loads a Servlet class. The loading may be from a local file system, a remote file system or other network services. The Servlet container provides the network services over which the request and response are sent.
- **Decode and Encode MIME based messages:** Provides the service of decoding and encoding MIME-based messages.
- **Manage Servlet container:** Manages the lifecycle of a Servlet.
- **Resource management:** Manages the static and dynamic resource, such as HTML files, Servlets and JSP pages.
- **Security Service:** Handles authorization and authentication of resource access.
- **Session Management:** Maintains a session by appending a **session ID** to the URL path.

CGI (Common Gateway Interface)

CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.

CGI is actually an external application which is written by using any of the programming languages like **C** or **C++** and this is responsible for processing client requests and generating dynamic content.

In CGI application, when a client makes a request to access dynamic Web pages, the Web server performs the following operations:

- ❖ It first locates the requested web page *i.e.* the required CGI application using URL.
- ❖ It then creates a new process to service the client's request.
- ❖ Invokes the CGI application within the process and passes the request information to the server.
- ❖ Collects the response from CGI application.
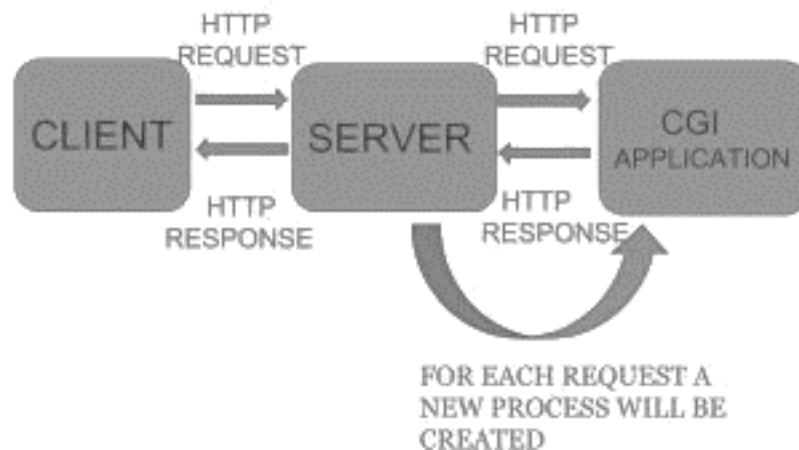- ❖ Destroys the process, prepares the HTTP response and sends it to the client.

Figure: Client request processing by CGI

So, in **CGI** server has to create and destroy the process for every request. It's easy to understand that this approach is applicable for handling few clients but as the number of client increases the workload on the server increases and so the time taken to process requests increases.

Limitations / Disadvantages of CGI

There are many problems in CGI technology:

1. If the number of clients increases, it takes more time for sending the response.
2. For each request, it starts a process, and the web server is limited to start processes.
3. It uses platform dependent language e.g. C, C++, perl.

All these Limitations of CGI are overcome by Servlet.

Advantages of a Java Servlet over CGI

1. **Better performance:** because it creates a thread for each request, not process.
2. **Portability:** because it uses Java language.
3. **Robust:** JVM manages Servlets, so we don't need to worry about the memory leak, garbage collection, etc.
4. **Secure:** because it uses java language.

Difference between Servlet and CGI

SERVLET	CGI
Servlets are portable and efficient.	CGI is not portable
In Servlets, sharing of data is possible.	In CGI, sharing of data is not possible.
Servlets can directly communicate with the web server.	CGI cannot directly communicate with the web server.
Servlets are less expensive than CGI.	CGI are more expensive than Servlets.
Servlets can handle the cookies.	CGI cannot handle the cookies.

Difference between Servlet and JSP

SERVLET	JSP

Servlet is a java code.	JSP is an html based 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 MVC approach.	JSP is the view in MVC approach for showing output.
Servlet is faster than JSP.	JSP is slower than Servlet because the first step in JSP lifecycle is the translation of JSP to java code and then compile.
Servlet can accept all protocol requests.	JSP only accept 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 javaBeans.
Modification in Servlet is a time consuming task because it includes reloading, recompiling and restarting the server.	JSP modification is fast, just need to click the refresh button.

Creating and Deploying New Servlet

There are 6 steps to create a **servlet**. These steps are required for all the servers.

The servlet example can be created by three ways:

1. By implementing Servlet interface,
2. By inheriting GenericServlet class, (or)
3. By inheriting HttpServlet class

The mostly used approach is by extending HttpServlet because it provides http request specific method such as doGet(), doPost(), doHead() etc.

Let's use **apache tomcat server** and create one servlet example. The steps are as follows:

1. Create a directory structure
2. Create a Servlet
3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server and deploy the project
6. Access the servlet

Create a directory structure

The **directory structure** defines that where to put the different types of files so that web container may get the information and respond to the client.

The Sun Microsystem defines a unique standard to be followed by all the server vendors. Let's see the directory structure that must be followed to create the servlet.
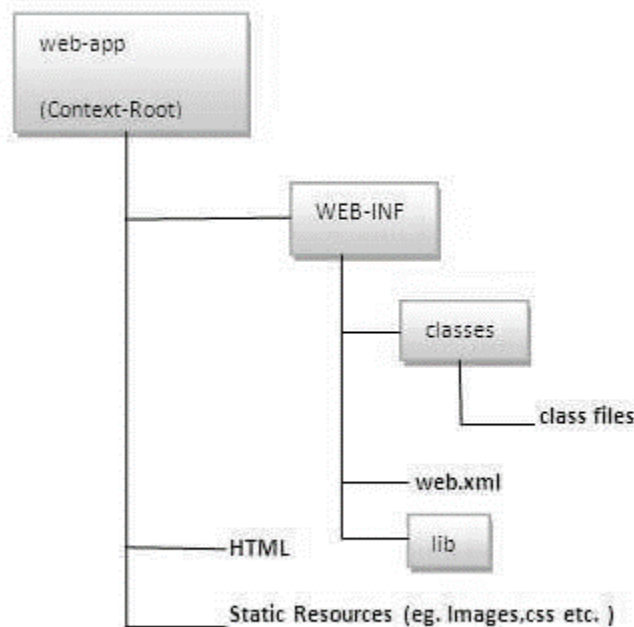
As you can see that the servlet class file must be in the classes folder. The web.xml file must be under the WEB-INF folder.

Create a servlet

- ❖ There are three different methods to create servlet but The HttpServlet class is widely used to create the servlet because it provides methods to handle http requests such as doGet(), doPost, doHead() etc.
- ❖ In the following example we are going to create a servlet that extends the HttpServlet class.
- ❖ we are inheriting the HttpServlet class and providing the implementation of the doGet() method. Notice that get request is the default request.

EecServlet.java

```java
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class EecServlet extends HttpServlet
{
public void doGet(HttpServletRequest req,HttpServletResponse res)
throws ServletException,IOException
{
res.setContentType("text/html");    //setting the content type
PrintWriter pw=res.getWriter(); //get the stream to write the data

//writing html in the stream
pw.println("<html><body>");
pw.println("Welcome to servlet from Everest Engineering College");
pw.println("</body></html>");

pw.close(); //closing the stream
}
}
```

Compile the Servlet

For compiling the Servlet, jar file is required to be loaded. Different Servers provide different jar files

Jar file	Server
1) servlet-api.jar	Apache Tomcat

2) weblogic.jar	Weblogic
3) javaee.jar	Glassfish
4) javaee.jar	JBoss

We are using Apache Tomcat so we need to load same server type jar file

There are two ways to load the jar file
1. set classpath
2. paste the jar file in JRE/lib/ext folder

Put the java file in any folder. After compiling the java file, paste the class file of servlet in **WEB-INF/classes** directory.

Create the deployment descriptor (web.xml file)

- ❖ The deployment descriptor is an xml file, from which Web Container gets the information about the servlet to be invoked.
- ❖ The web container uses the Parser to get the information from the web.xml file. There are many xml parsers such as SAX, DOM and Pull.
- ❖ There are many elements in the web.xml file. Here is given some necessary elements to run the simple servlet program.

```xml
<web-app>

<servlet>
<servlet-name>EverestEngineeringCollege</servlet-name>
<servlet-class>EecServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name> EverestEngineeringCollege </servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

</web-app>
```

❖ There are too many elements in the web.xml file. Here is the illustration of some elements that is used in the above web.xml file. The elements are as follows:

<web-app> represents the whole application.

<servlet> is sub element of <web-app> and represents the servlet.

<servlet-name> is sub element of <servlet> represents the name of the servlet.

<servlet-class> is sub element of <servlet> represents the class of the servlet.

<servlet-mapping> is sub element of <web-app>. It is used to map the servlet.

<url-pattern> is sub element of <servlet-mapping>. This pattern is used at client side to invoke the servlet.

Start the Server and Deploy the Project

I assume that you have made the java runtime Environment and Apache Tomcat server configuration as per need of JSP/ Servlet.

I) to start Apache Tomcat server, double click on the startup.bat file under apache-tomcat/bin directory.

II) To deploy the project Copy the context (project) and paste it in the webapps folder under apache tomcat.

III) To do so, you need to use jar tool to create the war file.

IV) Go inside the project directory (before the WEB-INF), then write:

projectfolder> jar cvf myproject.war *

Creating war file has an advantage that moving the project from one location to another takes less time.

Access the Servlet

Open Browser and write
http://hostname:portNumber/contextroot/urlpatternofservlet.

http://localhost:8086 /Eec /welcome

 Following Output is displayed on browser:

Welcome to servlet from Everest Engineering College

Introduction to JSP Tag Library

- ❖ JSTL stands for Java server pages standard tag library, and it is a collection of custom JSP tag libraries that provide common web development functionality.
- ❖ The JSP Standard Tag Library (JSTL) represents a set of tags to simplify the JSP development.
- ❖ For the creation of JSTL application, you need to create the jstl.jar file.
- ❖ There JSTL mainly provides five types of tags:
    1. **Core Tags**: The JSTL core tag provide support for, variables, iteration, conditional logic, catch exception, URL management, flow control, etc. prefix for core tag is **c**

        Syntax:

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

    2. **Functions Tags:** The function tags provides support for string manipulation and string length. Prefix of function tag is **fn**
    3. **Formatting tags:** The Formatting tags provide support for message formatting, number and date formatting, etc. prefix is **fmt**.
    4. **XML Tags:** The XML tags provide flow control, transformation, etc. Prefix for XML tags is x.
    5. **SQL Tags**: The JSTL SQL tags provide SQL support. And prefix is **sql**.

HTTP Request Handling, Session Management

HTTP

- ❖ Hyper Text Transfer Protocol
- ❖ It is the data communication protocol used to establish communication between client and server.

❖ It is the protocol that allows web servers and browsers to exchange data over the web.
❖ It is a request response protocol.
❖ It is stateless means each request is considered as the new request. In other words, server doesn't recognize the user by default.

HTTP is TCP/IP based communication protocol, which is used to deliver the data like image files, query results, HTML files etc. on the World Wide Web (WWW) with the default port is TCP 80. It provides the standardized way for computers to communicate with each other.

HTTP Request

The request sent by the computer to a web server, contains all sorts of potentially interesting information; it is known as HTTP requests.

The HTTP client sends the request to the server in the form of request message which includes following information:

❖ The Request-line
❖ The analysis of source IP address, proxy and port
❖ The analysis of destination IP address, protocol, port and host
❖ The Requested URI (Uniform Resource Identifier)
❖ The Request method and Content
❖ The User-Agent header
❖ The Connection control header
❖ The Cache control header

The HTTP request method indicates the method to be performed on the resource identified by the **Requested URI (Uniform Resource Identifier)**. This method is case-sensitive and should be used in uppercase.

❖ The HTTP request methods are:

Method	Description
GET	Asks to get the resource at the requested URL.
POST	Asks the server to accept the body info attached. It is like GET request with extra info sent with the request.

HEAD	Asks for only the header part of whatever a GET would return. Just like GET but with no body.
TRACE	Asks for the loopback of the request message, for testing or troubleshooting.
PUT	Says to put the enclosed info (the body) at the requested URL.
DELETE	Says to delete the resource at the requested URL.
OPTIONS	Asks for a list of the HTTP methods to which the thing at the request URL can respond

Interfaces of HTTP Request &Response Interfaces

HttpServletRequest interface

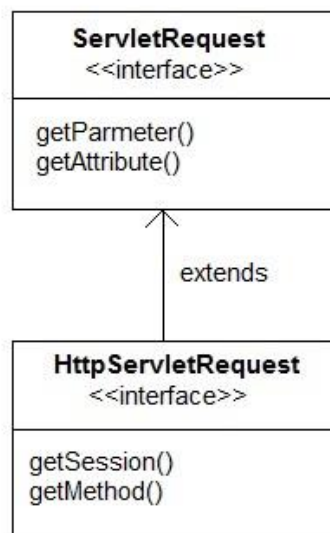**HttpServletRequest** interface adds the methods that relates to the **HTTP** protocol.

HttpServlet gets request from a client (browser) on the HttpServletRequest interface and can return response to the client through the HttpServletResponse interface in all the doXXXX methods.

Some of most used HttpServletRequest Methods:

Method	Description
public Enumeration getParameterNames()	Names of the parameters contained in this request.
public String[] getParameterValues(String name)	Used when the named parameter may have multiple values.
public Map getParameterMap()	Returns all the parameters stored in a Map object.
public BufferedReader getReader()	This will retrieve the body of a request as characters into a BufferedReader.

public ServletInputStream getInputStream()	This will retrieve the body of a request as binary data into a ServletInputStream.
public Enumeration getHeaderNames()	Returns an enumeration of all the header names this request contains.
public String getQueryString()	This will return the query string that is contained in the request URL after the path.
public RequestDispatcher getRequestDispatcher(String path)	A RequestDispatcher object can be used to forward a request to the resource or to include the resource in a response.

- You have to use either the getReader() method or the getInputStream(). You cannot combine to use these for the same request.

HttpServletResponse interface

- **HttpServletResponse** interface adds the methods that relates to the **HTTP** response.

MOST USED HTTPSERVLETRESPONSE METHODS:

Method	Description
public PrintWriter getWriter()	This will return a PrintWriter object that can send character text to the client.
public ServletOutputStream getOutputStream()	This will return a ServletOutputStream suitable for writing binary data in the response.
public addHeader(String name, String value)	This will add a property, which is a String name with a String value to the response header.
public addDateHeader(String name, long date)	This will add a property, which is a String name with a long date value to the response header.
public void sendRedirect(String location)	This will send a temporary redirect response to the client using the specified redirect location URL.
public Enumeration getHeaderNames()	Returns an enumeration of all the header names this request contains.

- You have to use either the getWriter() method or the getOutputStream().
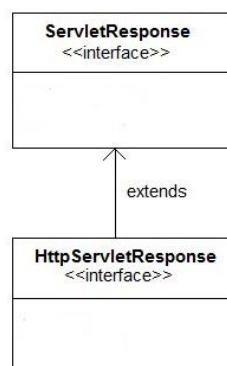 You cannot combine to use these for the same response

Figure: Servlet Response

Get vs. Post

GET	POST
1) In case of Get request, only **limited amount of data** can be sent because data is sent in header.	1) In case of post request, **large amount of data** can be sent because data is sent in body.
2) Get request is **not secured** because data is exposed in URL bar.	2) Post request is **secured** because data is not exposed in URL bar.
3) Get request **can be bookmarked.**	3) Post request **cannot be bookmarked.**
4) Get request is **idempotent.** It means second request will be ignored until response of first request is delivered	4) Post request is **non-idempotent.**
5) Get request is **more efficient** and used more than Post.	5) Post request is **less efficient** and used less than get.

Example to demonstrate Servlet Request

In this example a parameter is passed to a Servlet in a request object from HTML page.

index.html

```html
<form method="post" action="check">
    Name <input type="text" name="user" >
    <input type="submit" value="submit">
</form>
```

web.xml

```xml
<servlet>
    <servlet-name>check</servlet-name>
    <servlet-class>MyServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>check</servlet-name>
    <url-pattern>/check</url-pattern>
</servlet-mapping>
```

MyServlet.java

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MyServlet extends HttpServlet {

  protected void doPost(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
       PrintWriter out = response.getWriter();
       try {
           String user = request.getParameter("user");
           out.println("<h2> Welcome to "+user+"</h2>");
       } finally {
           out.close();
       }
    }
}
```

Output

If you run this code then you will get a text box with label name and submit button

If you write Everest Engineering College inside that text box then you will get
Welcome to Everest Engineering College as an output.

Session

❖ **Session** simply means a particular interval of time.
❖ Tracking any kind of user activity is known as Session tracking,

 Session comes in two favor

> 1.   Statefull session
>
> 2.   Stateless Session

Statefull session:

❖ It creates a channel between client and server, session will be closed if any
   one of the end get disturbed. And they need to re-authenticate again to
   continue their process.

❖ Example: SSH, FTP, Telnet etc

Stateless session:

❖ Once the user is authenticated it maintains a token to do the further process, so there is no need to maintain any channel between client and server, the client needs to re-authenticate if the token is lost or expired.
❖ Example: http protocol, all web based application.
❖ The token can be maintained on either client or server side

Consider a web based application, token can be maintained on client using 3 ways

1. Cookie   2. URL-rewriting   3. HiddenForm

Server side Servlets programming.

1. HttpSession can be used to maintain token on server side.

Use of the Session:

❖ Session is used to store everything that we can get from the client from all the requests the client makes.
❖ Any kind of object can be stored into a session, be it a text, database, dataset etc.
❖ Usage of sessions is not dependent on the client's browser.
❖ Sessions are secure and transparent

How Session Works?
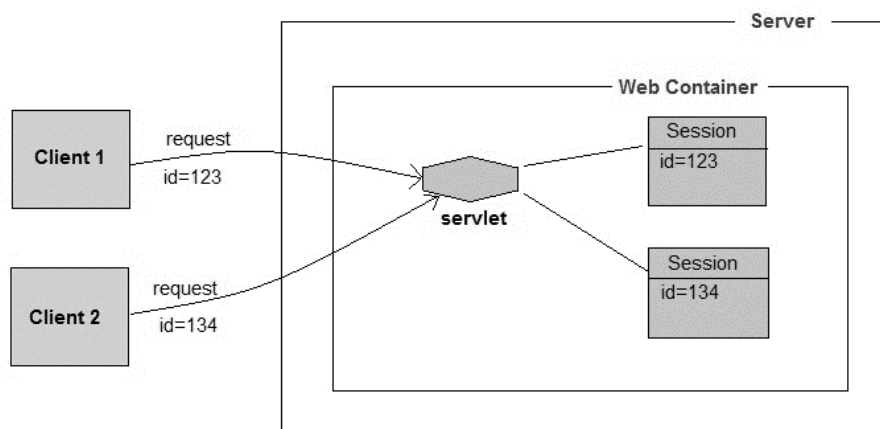
❖ The basic concept behind session is, whenever a user starts using our application, we can save a unique identification information about him, in an object which is available throughout the application, until it's destroyed.

❖ So wherever the user goes, we will always have his information and we can always manage which user is doing what. Whenever a user wants to exit from your application, destroy the object with his information.

Session Management:

❖ **Session Tracking** is a way to maintain state (data) of a user. It is also known as **session management** in servlet.
❖ **HTTP** is a stateless protocol. All requests and responses are independent. But sometimes you need to keep track of client's activity across multiple requests.
❖ For e.g. When a User logs into your website, not matter on which web page he visits after logging in, his credentials will be with the server, until he logs out. So this is managed by creating a session.

Session Management is a mechanism used by the Web container to store session information for a particular user.

❖ **There are following four different techniques used by Servlet application for session management**.

1. Cookies

2. Hidden form field

3. URL Rewriting

4. HttpSession

Cookies

❖ A **cookie** is a small piece of information that is persisted between the multiple client requests.
❖ A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

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.

Hidden Form Field

- ❖ **A hidden (invisible) textfield** is used in case of Hidden from Field for maintaining the state of a user.
- ❖ In such case, we store the information in the hidden field and get it from another servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.

URL Rewriting

- ❖ In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource. We can send parameter name/value pairs using the following format:
  url?name1=value1&name2=value2&??
- ❖ A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&). When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a Servlet, we can use getParameter() method to obtain a parameter value.

HTTP Session Interfaces

In this case, container creates a session id for each user.The container uses this id to identify the particular user.An object of HttpSession can be used to perform two tasks:

1. bind objects
2. View and manipulate information about a session, such as the session identifier, creation time, and last accessed time.
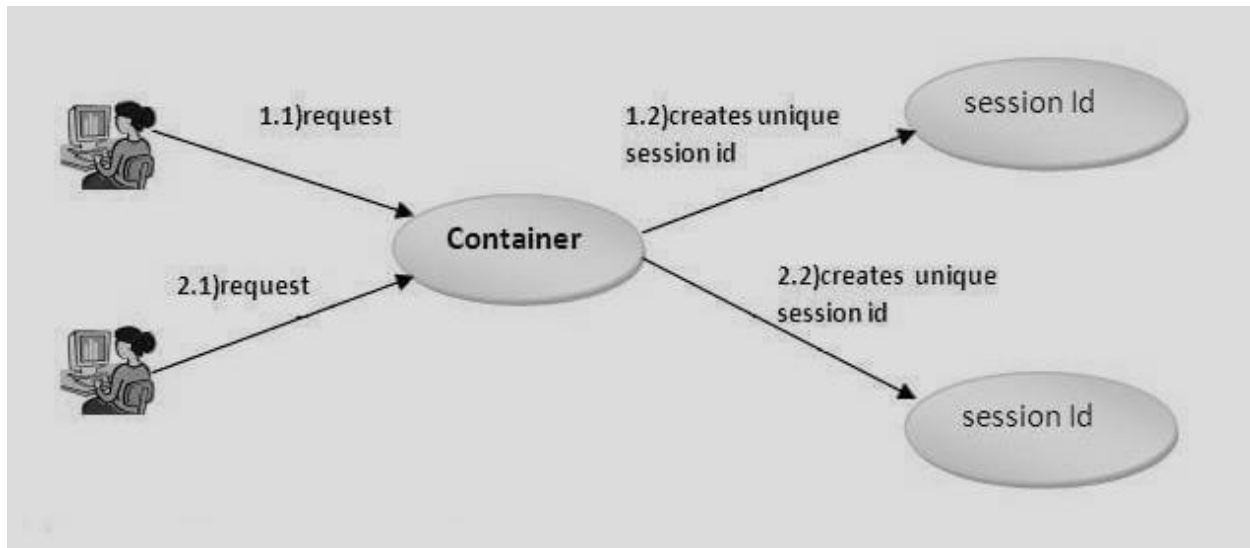
How to get the HttpSession object?

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.

Commonly used methods of HttpSession interface

1. **public String getId():**Returns a string containing the unique identifier value. OR, returns the unique session id.
2. **public long getCreationTime():**Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
3. **public long getLastAccessedTime():**Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
4. **public void invalidate():**Invalidates this session then unbinds any objects bound to it.

Example of Session tracking using HttpServlet Interface:

In the below example the setAttribute() and getAttribute() methods of the HttpSession interface is used to create an attribute in the session scope of one servlet and fetch that attribute from the session scope of another servlet.

Index.html

```html
<html>
<head>
<body>
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="submit"/>
</form>
</body>
</html>
```

First.java

```java
// The first servlet
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

< div class= "noIdeBtnDiv" > public class First extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)
    {
    try {
 /*Declaration of the get method*/
response.setContentType("text/html"); // Setting the content type to text
PrintWriter out = response.getWriter();

String n = request.getParameter("userName"); /*Fetching the contents of the
 userName field from the form*/
out.print("Welcome to " + n); // Printing the username
HttpSession session = request.getSession(); /* Creating a new session*/
session.setAttribute("uname", n); /*Setting a variable uname containing the value
 as the fetched username as an attribute of the session which will be shared amon
g different servlets of the application*/
out.print("<a href='servlet2'>GoToNextServlet</a>"); // Link to the second servle
t
out.close();
    }
catch (Exception e) {
System.out.println(e);
    }
}
```

```
}
```

Second.java

```java
// The second servlet
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SecondServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response) try {

        /*Declaration of the get method*/
response.setContentType("text/html");

PrintWriter out = response.getWriter();

HttpSession session = request.getSession(false); /*Resuming the session created i
n the previous servlet using the same method that was used to create the session.
The boolean parameter 'false' has been passed so that a new session is not create
d since the session already exists*/

String n = (String)session.getAttribute("uname");

out.print("Hello " + n);

out.close();
    }
catch (Exception e) {

System.out.println(e);
    }
  }
}
```

```xml
Web.xml
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>First</servlet-class>
</servlet>
```

```
<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>Second</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>

</web-app>
```
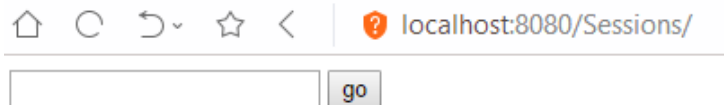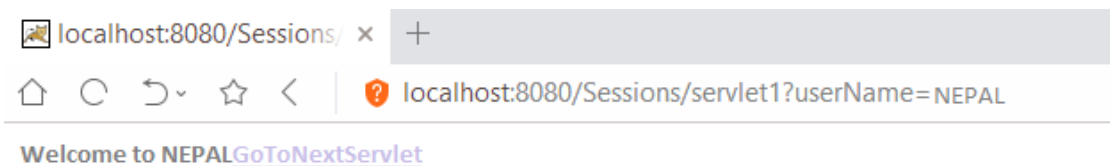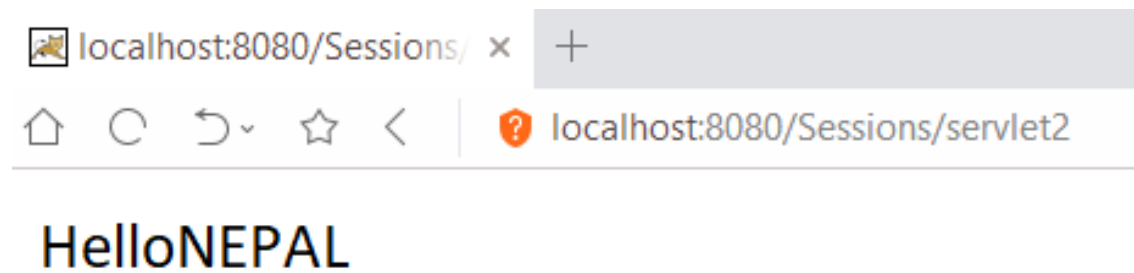
Output:

Index.html:

localhost:8080/Sessions/

[   ] go

You will get this output because of the interface using above.

If you write NEPAL inside that text box then,

Servlet1:

localhost:8080/Sessions   ×   +

localhost:8080/Sessions/servlet1?userName=NEPAL

Welcome to NEPALGoToNextServlet

Servlet2:

localhost:8080/Sessions/ × +

localhost:8080/Sessions/servlet2

HelloNEPAL

*** The End***