UNIT - 5

Introduction

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

→ Servlet technology is used to create a web application (reside at server side and generates a dynamic web Page.)

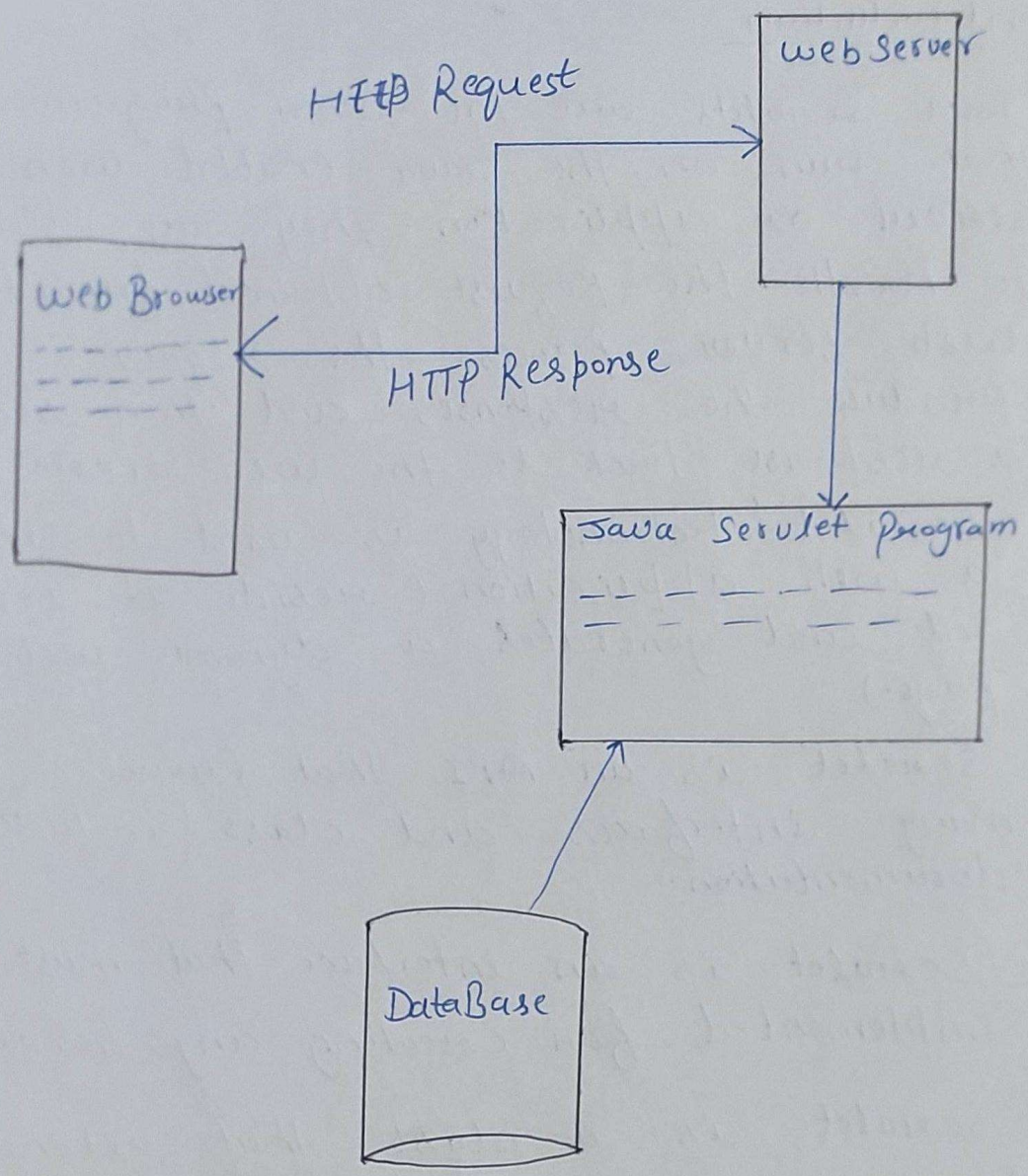
→ Servlet is an API that provides many interfaces and class including documentation.

→ Servlet is an interface that must be implemented for creating any servlet.

→ Servlet is a class that extends the capabilities of the server and responds to the incoming requests.

→ Servlet is a web component that is deployed on the server to create a dynamic web page.

Java Servlet Architecture



Execution of Servlet

It involves six steps

1) The client / web browser send the request to the web Server.

2) The web Server receives the Requests.
3) The web Server passes the request to the corresponding servlet.
4) The servlet processes the request and generates the response in the form of output
5) The Servlet Sends the response back to the webServer.
6) The WebServer sendo the back to the Client and the client browser displays it on the screen.

<u>HTTP Request</u> :- A HTTP request is a message sent by a client a (web browser or other application) to server to request data or action. HTTP requests are the Primary mean by which web Clients communicate with web server, allowing to retrieve data, submit data.

<u>HTTP Response</u> :- An HTTP response is the message sent by a server back to a Client in an ~~HTTP~~ response to an HTTP request. This response provides the client with the requested information or the result of the action performed by the Server.

CGI (Common Gateway Interface)

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

There are five steps to access dynamic web page, the web server performs the following steps:

1. It first locates the requested webpage
2. It then creates a new process to service the client request.
3. Invoke the CGI application within the process and passes the request information to the application.
4. Collects the response from CGI Application.
5. Destroy the process, prepare HTTP response, and send it to the client.

<u>Note:</u> CGI server has to create and destroy the process for every Request.

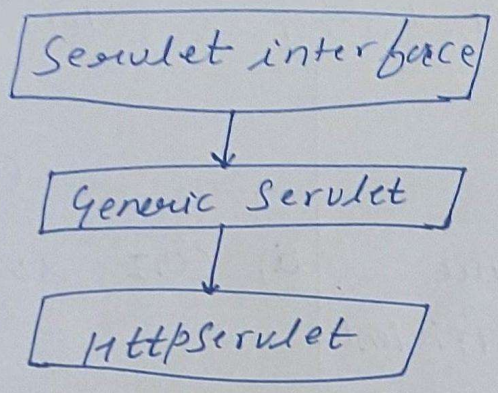
Difference between servlet and CGI

Servlet	CGI
a) Servlets are portable and efficient	a) CGI is not portable.
b) In Servlet, sharing data is possible	b) In CGI, sharing data is not possible.
c) Servlets can directly communicate with the webserver	c) CGI cannot directly communicate with the webser.
d) Servlet are less expensive than CGI	d) CGI is more expensive than Servlet.
e) Servlets can handle the cookies.	e) CGI cannot handle the cookies.

HTTP Servlet Basics

HttpServlet is a abstract class, it comes under package 'javax.servlet.http.HttpServlet'. To create a servlet the class must extends 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 Class

1. doGet() Method:-
   - This method is used to handle to GET request on the server-side.
   - This method also automatically supports HTTP HEAD.
   - The GET type request is usually used to preprocess a request

Modifier and Type Protected void

Syntax:

```
protected void doGet(HttpServletRequest req,
  HttpServletResponse resp) throws IOException,
                                   ServletException
```

2. doPost() Method

- This methods is used to handle the POST request on server side.
- This method allows the client to send data of unlimited length to the webserver at a time.
- The POST type request is usually used to post-process a request

Modifier and Type: protected void.

Syntax

```
protected void doPost(HttpServletRequest req,
HttpServletResponse resp) throws IOException,
                                 ServletException
```

3. doPut() Method.

- This method is overridden to handle the PUT request.
- This method allows the client to store the information on the server
- This method is called by the server to handle a PUT request.

Modifier and Type: protected void.

Syntax:

protected void doPut(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException.

5. doDelete() Method

- This method is overridden to handle the DELETE request.
- This method allows a client to remove a document or webpage from the server.

Modifier and Type: protected void.

protected void doDelete(HttpServletRequest req, HttpServletResponse resp) throw IOExce-ption, ServletException.

Servlet Mapping

Servlet Mapping is the process of defining a URL Pattern that is associated with a particular servlet. It tells the servlet Container which servlet to invoke when a client makes a request to specific URL.

Type of Servlet Mapping

1) Annotation- Based Mapping → @webservlet.
2) XML- Based Mapping

Servlet Container

A servlet Container, also Know as web Container, is a part of a web server or application server that provide the runtime environment for executing Java Servlet, JSP and other web component. It manages the lifecycle, configuration and execution of servlet and ensure they comply with the Java Servlet API

Annotation: Annotation represents the metadata. If you use annotation, deployment descriptor (web.xml) is not required.
eg → @webservlet("/ ")

web.xml or XML- Based The web.xml file is a deployment descriptor file that provide configuration the deployment information for the web component of a web application

```
<web-app - ---- >
< servlet>
<servlet_Name> abc </servlet_Name>
< servlet_class> package. classname </servlet_class>
</servlet>
<servlet-mapping>
   <servlet_name> abc </servlet_name>
   < Url_pattern> * </Url_pattern>
   </servlet_mapping>
                    </web-app>
```

Difference GET and 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 can not be bookmarked
4) GET request is idempotent	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

Servlet Life Cycle

The entire life cycle of a servlet is managed by the servlet container which uses the javax.servlet.Servlet interface to understand the servlet object and manage it. So, before creating a

Servlet object.

Stages of servlet life cycle

The servlet life cycle mainly goes through four stages.

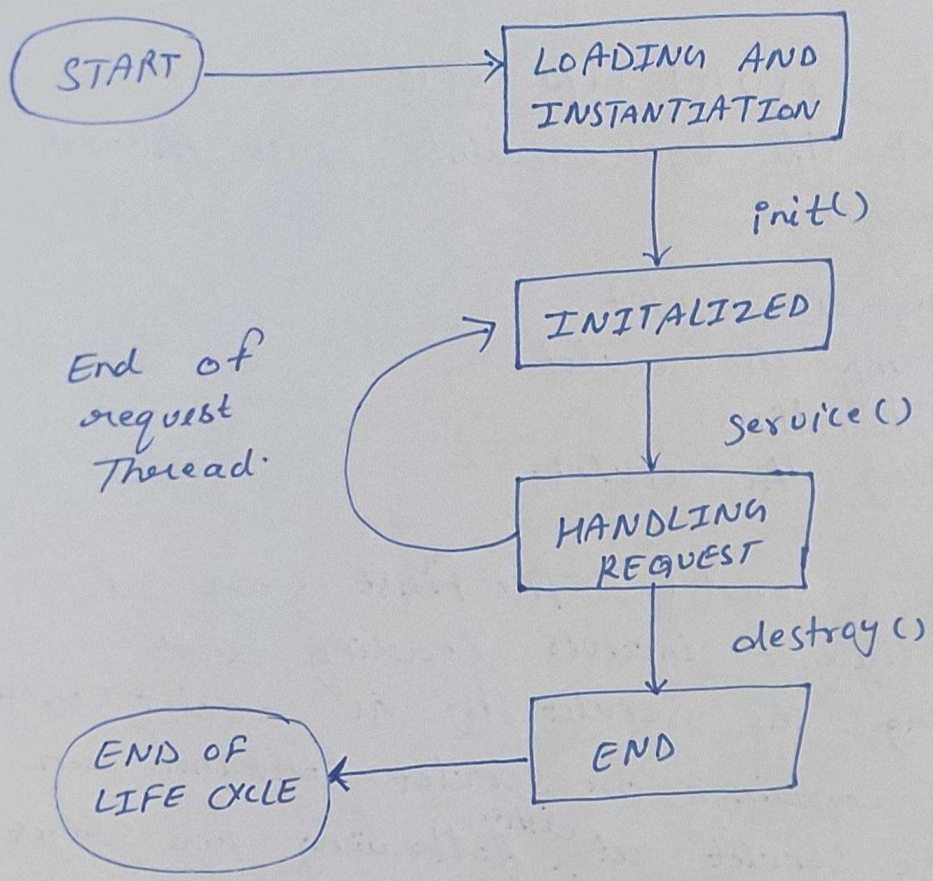
1. Loading a servlet.
2. Initializing the servlet.
3. Request handling.
4. Destroying the servlet.

1. Loading a servlet The first stage of servlet lifecycle involves loading and initializing the servlet by the servlet container. The web container or servlet container can load the servlet at either following two stages

- Initializing the context, on configuring the servlet with a zero or positive integer value.
- If the servlet is not preceding stage, it may delay the loading until the web container determine that this servlet is needed to service a request.

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.



2. <u>Initializing a Servlet</u>

After the servlet is instantiated sucessfully the servlet container initializes the instantiated servlet object. The containeer intialize the 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 This method is used to initialize the

resources, such as JDBC dataSource

3. Handling Request:

After initialization, the servlet instance is ready to serve the client requests. The servlet container perform the following operation when the servlet instance is located to a service a request.

- It create the ServletRequest and servletResponse objects.
- After creating the request and response objects it invokes the servlet.service (ServletRequest, ServletResponse) method by passing the request and response object.

Note: The service() method while processing the request may throw.

4. Destroying a Servlet

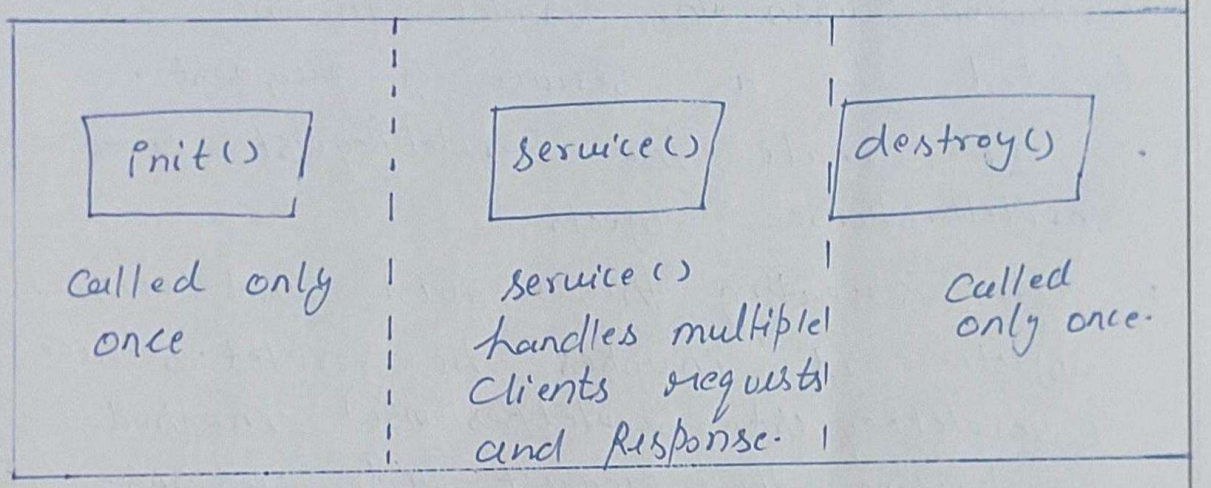
When a servlet container decides to destroy the servlet, it perform the following operations

- It allows all the threads currently running in the service method of the servlet instance to complete the task.
- After current running threads have complete their jobs, the servlet container calls the destroy() method on servlet instance.

Methods of Servlet Life cycle

There are three methods.

1. init()
2. Service()
3. destroy().



RETERIVING INFORMATION

When developing web application using the Servlet API, retrieving information typically refers to accessing request data sent by the client (eg, browser) to the server. This data can come in many form, such as form submission (via GET and POST methods), query Parameter, cookies, header and session-data.

1. Retrieving Parameter from the Http Request

When a user submits a form or when query parameter are passed in the URL, the server can retrieve those parameter using the HttpServletRequest object.

(a) Query Parameter (for GET request)

→ Query parameter are typically passed in the URL

→ Use getParameter() to retrieve a parameter.

```
String s1 = req.getParameter("name");
```

(b) Form Data (for POST Request)

→ Form data is sent in the body of the request, often using POST. You can retrieve form data in the same way as query parameters.

```
String s1 = req.getParameter("Username");
```

c) Retrieving Multiple Parameter (for Multiple values)

→ If the Parameter appear multiple times in the request (checkbox, multi-select lists) you can retrieve all values.

```
String[] s1 = req.getParameter("item");
```

2. Retrieving Header

→ HTTP headers contains additional information about the request, such as the User agent, content type or authentication details.

→ Using getHeader() method.

String s1 = req.getHeader("User");

3. Retrieving Cookies

→ Cookies are small pieces of data sent by the client and store on the client machine.

→ Use getCookies() to get an array of cookies objects.

Cookies[] cookies = req.getCookies();

4. Retrieving Attributes

Attributes are used to pass data between servlets, or between a servlet and a JSP. They are often set by using setAttribute() can be retrieved using getAttribute().

Object myAttribute = req.getAttribute("Attribute");

Example

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

public class Demo extends HttpServlet
  {
  @override        // annotation
protected void doGet(HttpServlet_Request req,
HttpServletResponse resp) throws ServletException,
                                 IOException.
          {
      resp.setContentType("text/html");
      //Get the PrintWriter to send output
      PrintWriter out= resp.getWriter();
     //Retrieve a query Parameter
      String name= req.getParameter("name");
     //Retrieve a header
      String UserAgent = req.getHeader("User-Agent");
    //Retrieve a cookies
      Cookie[] cookies = req.getCookies();
      for(int i=0; i<cookies.length; i++)
        {
          out.print("<br>"+ cookies[i].getName()
                    + " "+ cookies[i].getValue());
        }
//Get session data.
  HttpSession session = req.getSession();
String sessiondata= (String) session.getAttribute
                                ("sessionKey");
```

```
// output the retrieved data.

out.println("<html><body>");
out.println("<h1> Retrieved </h1>");
out.println("<p> Name: " + name + "</p>");
out.println("<p> User-Agent: " + UserAgent + "</p>");
out.println("<p> Cookie Username: " + cookievalue
                                    + "</p>");
out.println("<p> Session Data: " + sessionData
                                    + "</p>");
out.println("</body></html>");
}}
```

Sending HTML Information

Sending HTML information refers to the process of sending data in HTML format from a web server to client. HTML is the standard language used for creating web pages.

There are following steps involved in Sending HTML information.

1. **Generating HTML content:**

The Html content is generated by the server-side code. This can be done using a variety by methods, including dynamic HTML generation, template engines and Content Managment.

2. **Setting the response content type:**

The content type of the response must be set to "text/html" to indicate that the response is in HTML format.

3. **Writing HTML content to the response object:** The HTML content is written to the response object using the response objects "write" method.

The "write" method takes the HTML content as a parameter and writes it to the response.

4. Sending the response to the Client

The response is sent to the client by the web server. The client recieves the HTML content and display it in the browser.

Example

```
@Override
protected void doPost(HttpServletRequest req,
    HttpServletResponse resp) throws ServletException, IOException.
{
        String username = req.getParameter("User");

    resp.setContentType("text/html");
    PrintWriter out = resp.getWriter();
    out.println("<html><body>");
    out.println("<h1> welcome, "+user+"!</h1>");
    out.println("</body>");
    out.println("</html>");
        out.close();
}
```

RequestDispatcher in Servlet

The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. This interface can also be used to include the content of another resource also. It is one of the way of servlet collaboration.

There are two methods defined in the RequestDispatcher interface.

1.public void forward(ServletRequestrequest,ServletResponse response)throws ServletException,java.io.IOException: Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.

2.public void include(ServletRequestrequest,ServletResponse response)throws ServletException,java.io.IOException: Includes the content of a resource (servlet, JSP page, or HTML file) in the response.

Example:

index.html

```
<form action="servlet1" method="post">
Name:<input type="text" name="userName"/><br/>
Password:<input type="password" name="userPass"/><br/>
<input type="submit" value="login"/>
</form>
```

Login.java

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

@WebServlet("/servlet1")
public class Login extends HttpServlet {
@override
public void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

   response.setContentType("text/html");
   PrintWriter out = response.getWriter();

   String n=request.getParameter("userName");
   String p=request.getParameter("userPass");

   if(p.equals("servlet"){
      RequestDispatcher rd=request.getRequestDispatcher("servlet2");
      rd.forward(request, response);
```

```
    }
    else{
        out.print("Sorry UserName or Password Error!");
        RequestDispatcher rd=request.getRequestDispatcher("/index.html");
        rd.include(request, response);

        }
    }

}
```

WelcomeServlet.java

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

@WebServlet("/servlet2")
public class WelcomeServlet extends HttpServlet {

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

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    String n=request.getParameter("userName");
    out.print("Welcome "+n);
    }

}
```

SendRedirect in servlet

The **sendRedirect()** method of **HttpServletResponse** interface can be used to redirect response to another resource, it may be servlet, jsp or html file.

It accepts relative as well as absolute URL.

It works at client side because it uses the url bar of the browser to make another request. So, it can work inside and outside the server.

Difference between forward() and sendRedirect() method

forward() method	sendRedirect() method
The forward() method works at server side.	The sendRedirect() method works at client side.
It sends the same request and response objects to another servlet.	It always sends a new request.
It can work within the server only.	It can be used within and outside the server.
Example: request.getRequestDispacher("servlet2").forward(request,response);	Example: response.sendRedirect("servlet2");

Example:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>sendRedirect example</title>
</head>
<body>


<form action="MySearcher">
<input type="text" name="name">
<input type="submit" value="Google Search">
</form>

</body>
</html>
```

MySearcher.java

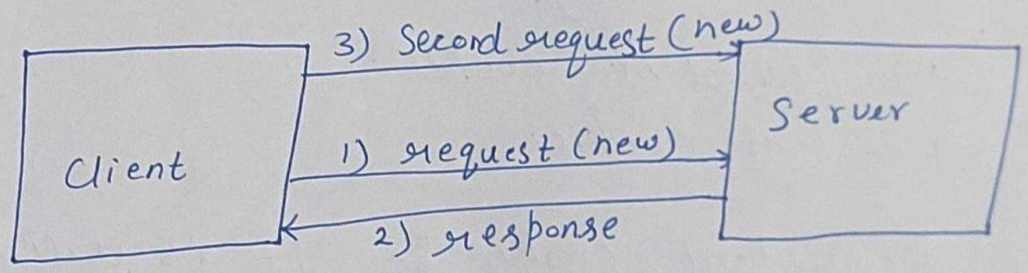
```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
@Webservlet("/MySearcher")
public class MySearcher extends HttpServlet {

@override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {

      String name=request.getParameter("name");
      response.sendRedirect("https://www.google.co.in/#q="+name);
   }
}
```

Session Tracking

Session simply means a particular interval of time.

Session Tracking is a way to maintain state (data) of user. It is also known as session management in servlet. Http Protocol is a stateless so we need to maintain state using session tracking technique. Each time user request 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.



Note: we use session tracking to recognize the user.

Session Tracking Technique

1) Cookies
2) Hidden Form Field
3) URL Rewritting
4) HttpSession

Cookies

A cookies is a small piece of information that is persisted between client requests.
A cookies has a name, a single value and optional attribute such as a comment, path and domain qualifier.

Note :- By default each request is considered as a new Request And Cookie is stored in the cache of the browser.

Types of cookies

There are two type of cookies

1) Non Persistent cookie.
2) Persistent cookie.

Non Persistent cookie :- It is valid for single session only. It is removed each time when user close the browser.

Persistent cookie :- It is valid for multiple sessions. It is not removed each time when use closes the browser. It is removed only if user logout or singout.

How to create a cookie

```
Cookie ck = new Cookie("user", "Ajay");
```

or

```
Cookie ck = new Cookie("user", name);
```

How to setting up lifespan for a cookie

```
ck.setMaxAge(30*60);
```

How to Sending the cookie to client

```
response.addCookie(ck);
```

Getting cookie from client request

```
Cookie[] cks = request.getCookies();
```

Delete the cookies

```
Cookie ck = new Cookie("user" );
ck.setMaxAge(0);
response.addCookie(ck);
```

Example of cookie

Step-I :- Create a Html file in web content

```
// cookie.html file name.

<form method = "post" action = "ActionCookie">
<label for = "first"> UserName </label>
<input type = "text" name = "user">
```

```
< label for = "password" > Password < / Label>
< input type = "password" name= "pass" >
< Button type = "submit" > Submit < / Button>
  < / form >
```

Step-2 Create a servlet

```
@WebServlet ("/ActionCookie")
public class ActionCookie extends HttpServlet
{

  @Override
protected void doPost (HttpServletRequest req,
          HttpServletResponse resp) throws
          ServletException, IOException

    resp. setContentType ("text / html");

    String name = req. getParameter ("user");
    String pass= req. getParameter ("pass");
    PrintWriter out = resp. getWriter();

  if ( pass. equals ("123") )
  {
    Cookie ck= new Cookie ("user", name);
     resp. addCookie (ck);
     resp. sendRedirect ("welcomeCookie");
  }
else
  {  out. print ("<h1> Error </h1>");
  RequestDispatcher rd = resp. getRequestDispatch
                -er ("cookie. html");
```

```
rd.include(req, resp);
  }
}
```

<u>Step-3</u> create a servlet to welcomeCookie

```
public class WelcomeCookie extends HttpServlet
{

protected void doGet(HttpServletRequest req,
        HttpServletResponse resp) throws
    IOException, ServletException

  resp.setContentType("text/html");
   Cookie[] ck = req.getCookies();
  PrintWriter out = resp.getWriter();
 out.println("Welcome"+ ck[0].getValue());

}
```

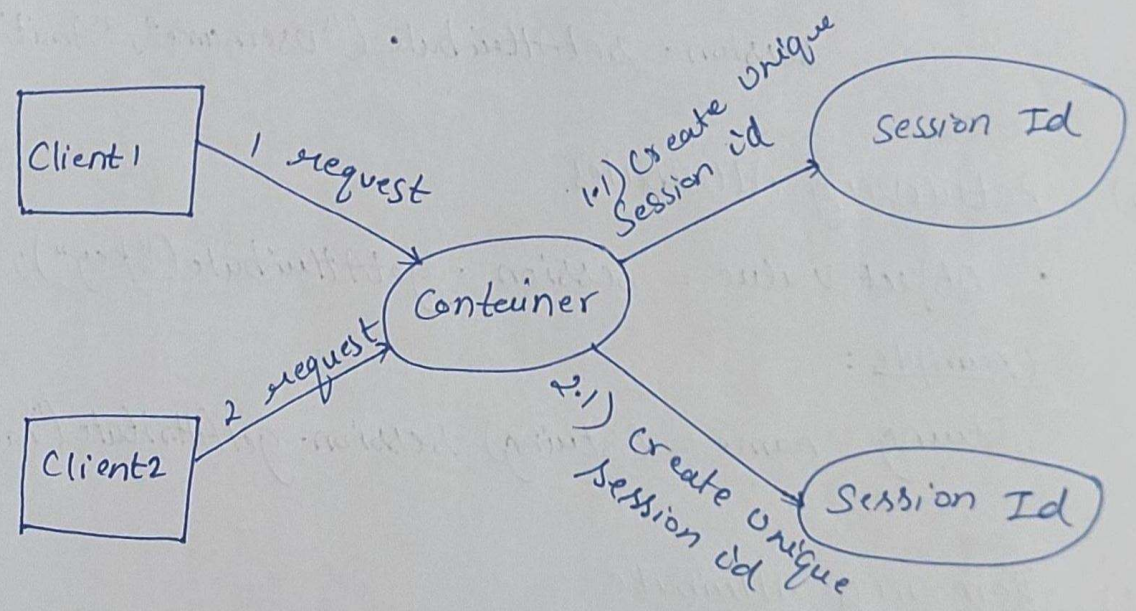
Http Session

Http Session is used to maintain state and data across multiple requests made by the same user during their interaction. with a web application

Http Session can be used to perform two tasks:

(i) bind object

(ii) View and manipulate information about a session, such as session identifier, creation time



Note:

HTTP Provide two method to get Object of HTTPSession

(a) public HttpSession getSession();

(b) public HttpSession getSession(boolean create);

Creating and Managing an HTTP Session

1) Retrieve or Create a Session

- HttpSession session = req.getSession();
- This retrieve the current session if its exists, or creates a new session if it dosnot.

2) Storing Attribute in Session

- Session.setAttribute("key", value);

Example:

session.setAttribute("username", "Amit");

3) Retrieving Attributes

- Object value = session.getAttribute("key");

Example:

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

4) Removing Attribute

- session.removeAttribute("key");

5) Invalidating a Session

- session.invalidate();

This destroy the session and remove

all associated data.

6) Getting Session Information

- session.getId(): Return unique session ID.
- session.getCreationTime(): Return the time of session was created.
- session.isNew(): Checks if the session is new.

Example

Step-I To create a login page and it is same a cookies step I

Step-II Create a servlet

```
public class Login extends HttpServlet
  {
  @override
  protected void doPost (HttpServletRequest req,
HttpServletResponse resp) throws IoException,
      ServletException
      {

  String username = req.getParameter("user");
    HttpSession session = req.getSession();
    session.setAttribute ("Username", username);
```

```
response.sendRedirect("welcome");
```

Step-3 create a servlet

```
public class welcome extends HttpServlet
{
@override
protected void doGet(HttpServletRequest req,
HttpServletResponse resp) throws IOException,
                                 ServletException

{
    resp.setContentType("text/html");

  HttpSession session = req.getSession(false);

  PrintWriter out = resp.getWriter();

if(session != null && session.getAttribute("username")
                                    != null)

    {

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

      out.println("<h1> Username </h1>");
      out.print("<a href='logout'> logout
                                    </a>");
  }

  else
  {

    ~~out~~
```

~~out.print~~

```
out.println("<h1> No session found </h1>");
out.println("<a href='login.html'> Login </a>");
}
}
}
```

Step-4 Create Logout servlet

```
public class Logout extends HttpServlet
{
@override
protected void doGet(HttpServletRequest req,
HttpServletResponse resp) throws ServletException,
        IoException
{
HttpSession session = req.getSession(false);
if(session != null)
{
    session.invalidate();
}
response.sendRedirect("login.html");
}
}
```

Hidden Form Field

In case of Hidden Form Field a hidden (invisible) textfield is used for maint--aining the state of an user.

We can 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 donot want to depend on browser.

```
<input type="hidden" name="uname"
    value="vimal">
```

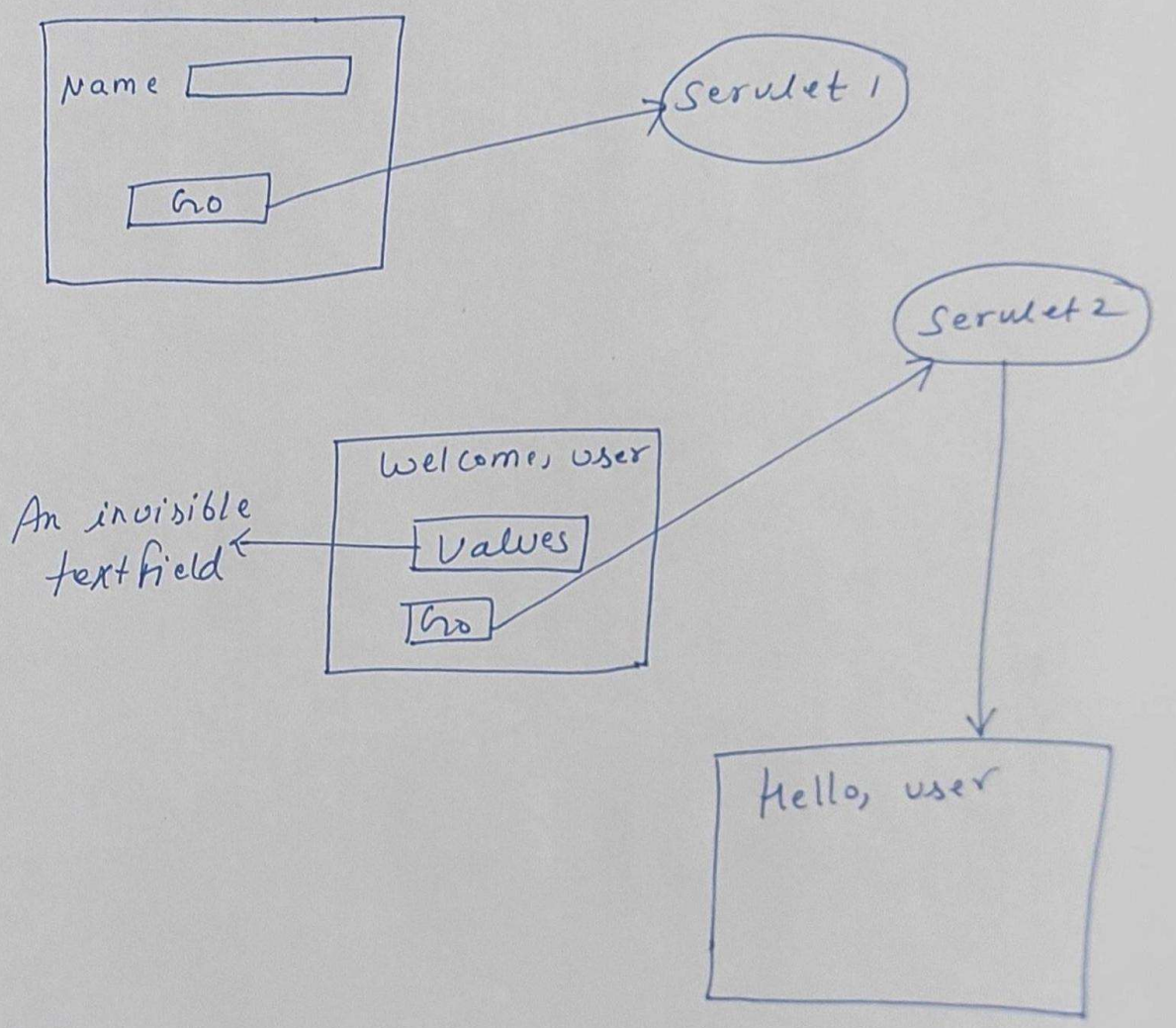
Here uname is the hidden field name and vimal is the hidden field value.

Advantage

- Easy to setup and use.
- No need for cookies or URL Rewriting.
- works without relying on server-side storage.

Disadvantage

- It is maintained at server side
- Extra form submission is required on each page.
- only textual information can be used.

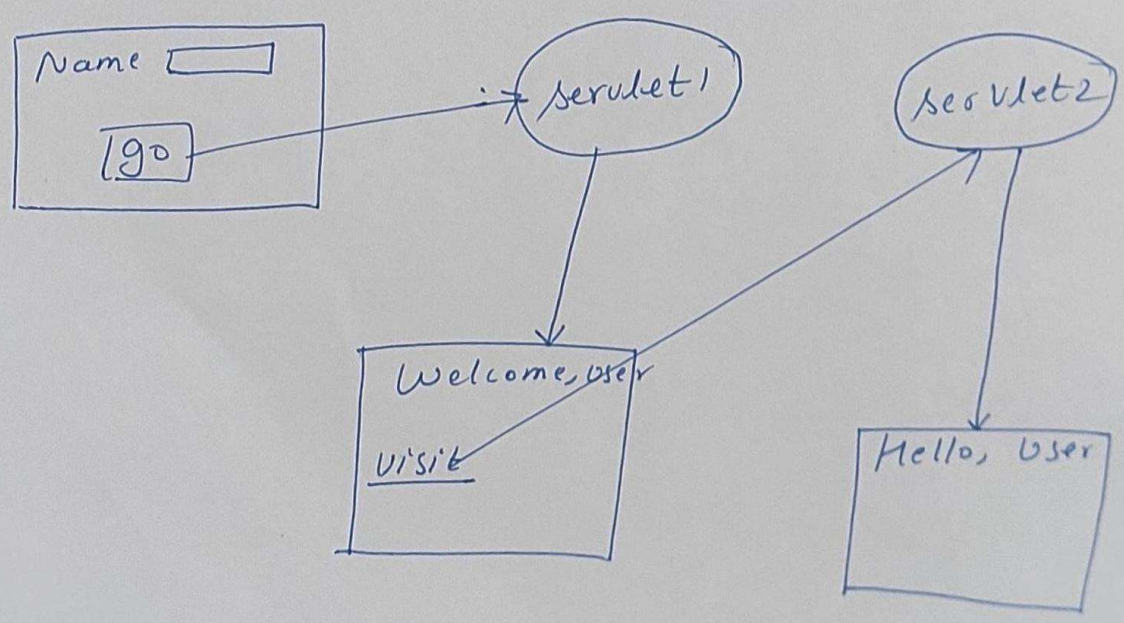


URL Rewriting

It is a technique used in java servlets to maintain state across multiple request by appending session information or other parameter to the URL. This method is useful when cookies are disabled or not supported by the client browser.

A token (parameter) is added at the end of URL. The token consist of name/value pair seperated by an equals (=) sign.

Url? name1 = value1 & name2 = value2 & ??



Advantage

- Work even if cookies are disabled.
- Easy to implement & understand.
- Extra form submission is not required on each page.

Disadvantage

- It will works only with links.
- It can send only textual information
- URLs are subject to length limitations in browsers.

DataBase Connectivity

DataBase connectivity refers to the process of connecting to a database from web Application in order to retrieve and store data. A dataBase Connection is required in order to interact with a database

How to create JDBC connection

1) Import the packages
2) Load the drivers using the forName() method
3) Register the drivers using DriverManager
4) Establish a connection using the connection Class object
5) Create Statement
6) Execute the Query
7) Close the connection

Detail

→ Class.forName("com.mysql.jdbc.Driver");

→ DriverManager.getConnection(Url, user, password)

Url="jdbc:mysql://localhost:3306/DataBase";

→ Statement statement = connection.createStatement();

→ executeQuery() method

→ Connection.close();

CRUD Operation Perform

C → Create (POST)
R → Read (GET)
U → Update (PUT)
D → Delete (DELETE)

Example Insert a data.

1) Create a connection class

```
public class Conn
  {
  public static final String DRIVER = "com.mysql.jdbc.
                                        Driver";
  public static Connection con;
  public static final String URL = "jdbc:mysql://local
                                        host/";
  public static final String USERNAME = "root";
  public static final String PASSWORD = "root";
  public static final String DATABASE = "ABC";

      public static Connection getCon()

  {     try
        {
            Class.forName(DRIVER);
            con = DriverManager.getConnection(URL+
                    DATABASE, USERNAME, PASSWORD);
        System.out.println("Create connection");
        }
      Catch (Exception e)
```

```
{
    e.printStackTrace();
    return null;
}
return con;
}
}
```

2) <u>Create a Login form in HTML file.</u>

```
<form action = "abc" method = "post">
<label for="user"> UserName: </label>
<input type="text" name="user">
<label for = "password"> Password: </label>
<input type= "password" name = "pass">
<button type= "submit"> Click </button>
</form>
```

3) <u>Create a Servlet</u>

```
@WebServlet("/abc")
public class Demo
{
@Override
protected void doPost(HttpServletRequest req,
    HttpServletResponse resp) throws ServletException,
                                     IOException
    {  resp.setContentType("text/html");
       String s1 = req.getParameter("User");
       String s2 = req.getParameter("pass");
       int i = 0;
```

```
PrintWriter out = resp.getWriter();
try
{ Connection database = Conn.getCon();
  String sql = " insert into signup (name, email)
                 values(?,?)";
    PreparedStatement ps = database.prepareStatement
                                (sql);

  ps.setString(1, s1);
  ps.setString(2, s2);
   i = ps.executeUpdate();
}

  catch(Exception e)
    {
      e.printStackTrace();
    }
  if (i>0)
    {
      out.print("<h1> Data Inserted </h1>");
  RequestDispatcher rd = req.getRequestDispatcher
                                ("signup.html");

      rd.include(req, resp);

    }
  else
  {
   out.print(" Data not inserted");
RequestDispatcher rd= req.getRequestDispatcher
                                ("signup.html");
      rd.include(req, resp);
  }
}}
```

GET a data in database

Step I and Step II are same in insert data Example

Step-3

Create a servlet

```
@WebServlet("/abc")
public Class Demo
{
protected void doGet(HttpServletRequest req,
 HttpServletResponse resp) throws IoException,
                                 ServletException.

 {
    resp.setContentType("text/html");
   PrintWriter out = resp.getWriter();

  try
    {
     Connection database= Conn.getCon();
  String ds= "select * from signup";
PreparedStatement ps= database.prepareStatement(ds);
  ResultSet rs= ps.executeQuery();

  while (rs.next())
    {
      String name = rs.getString(1);
      String password = rs.getString(2);

    out.println(name);
    out.println(password);
  }
}
```

```
catch(Exception e)
{
    e.printStackTrace();
}
}}
```

Update a data in dataBase

Step I and II are same in insert a data to pic

Step III

```
@webServlet("/abc")
public class Demo
{
@ovrride
protected void doPost(HttpServletRequest req,
HttpServletResponse resp) throws ServletException,
                                IOException
{
    resp.setContentType("text/html");
    String s1 = req.getParameter("id");
      int a = Integer.ParseInt(s1);

    String s2 = req.getParameter("user");
    String s3 = req.getParameter("pass");

    PrintWriter out = resp.getWriter();

    try
      {
```

```
Connection database = Conn.getCon();
String sql ="update signup set name=?,
            password=? where id=?";
PreparedStatement ps= database.prepareStatement(sql);
  ps.setString(1, s2);
  ps.setString(2, s3);
  ps.set.      (3, a);
    i= ps.executeUpdate();

  }
catch (Exception e)
  {
    e.printStackTrace();

  }
  if (i>0)
  {   out.print(" Update successfully");
  RequestDispatcher rd = req.getRequestDispatcher
                          ("index.html");

          rd.include(req, resp);

  }
  else
  {   out.print(" Error occured");
RequestDispatcher rd = req.getRequestDispatcher
                          ("index.html");

        rd.include(req, resp);

  }
}}
```