SERVLET

The terms "Internet application" or "web application" were coined when dynamic content was introduced. Loosely interpreted, a web application is a web site whose contents are generated dynamically before being sent to the browser.

When you surf the Internet, you basically request for a certain file located in a particular computer in the location you specify in the Uniform Resource Locator (URL). The computer where the file is stored is called the web server. This computer's main function is to serve anybody on the Internet who requests files it hosts. Because you never know when a user will visit and use your web application, your web server must be up and running all the time.

When you click or type in a URL in the Location or Address box of your browser, the following things happen:

• The client browser establishes a TCP/IP connection with the server.
• The browser sends a request to the server.
• The server sends a response to the client.
• The server closes the connection (After sending the requested page to the browser, the server always closes the connection, whether or not the user requests other pages from the server).

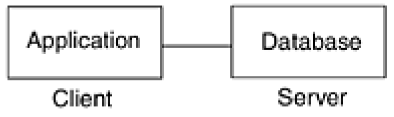
The Hypertext Transfer Protocol (HTTP) : HTTP is the protocol that allows web servers and browsers to exchange data over the web. It is a request and response protocol. HTTP uses reliable TCP connections—by default on TCP port 80. The client requests a file and the server responds to the request. It's always the client who initiates a transaction by establishing a connection and sending an HTTP request. The server is in no position to contact a client or make a callback connection tothe client. Either the client or the server can prematurely terminate a connection.

System Architecture : A well-designed software application is partitioned into separate logical parts called layers. Each layer has a different responsibility in the overall architecture. These layers are purely abstractions, and do not correspond to physical distribution.

Typical layers in a software system are as follows:

• **Presentation layer.** In this layer are parts that handle the user interface and userinteraction.
• **Business logic layer.** This layer contains components that handle the programming logicof the application.
• **Data layer.** This layer is used by the business logic layer to persist state permanently.This layer normally consists of one or more databases where data is stored. However,other types of datastore could also be used. For example, it is now very common to use XML documents as storage to keep data.

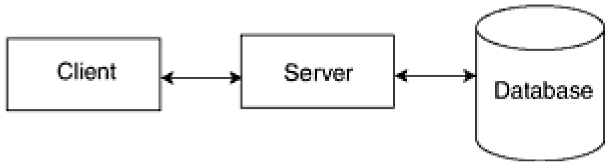
The Two-Tier Architecture : A two-tiered application is a simple client-server application in which the processing workload falls onto the client computer's shoulders and the server simply acts as a traffic controller between the client and the data. The term "fat client" for this type of architecture is due to the bulk of processing requirements at the client side. In this architecture, the presentation layer and the business logic layer are hosted in one tier and the data layer is on another tier.



The drawback of this type of architecture is that it starts to pose problems as the number of clients increases. The first problem is due to the fact that all processing happens at the client side. There is increased network traffic because each client has to make multiple requests for data from the server—even before presenting anything to the user. Another problem is cost because each client needs a machine with sufficient processing power. As the number of clients increase, the cost for providing client machines alone could be astronomical.

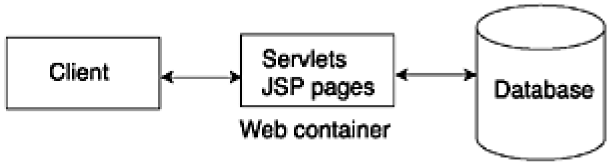
However, the most severe problem that this type of architecture can cause is probably a maintenance problem. Even a tiny change to the processing logic might require a complete rollout to the entire organization. Even though the process can be automated, there are always problems with larger organizations because some users may not be ready for an upgrade, whereas others insist it be performed immediately.

The Three-Tier Architecture : To overcome the problems in many client two-tiered applications, an application is broken up into three separate tiers, instead of two. The first tier contains the presentation layer, the second tier, or the middle tier, consists of the business logic layer, and the third tier contains the data layer.

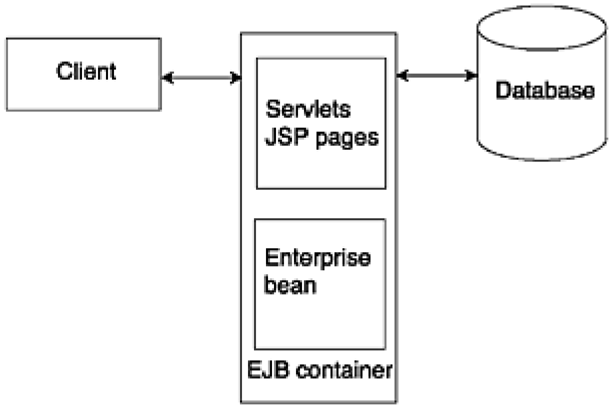


The n-Tier Architecture : To achieve more flexibility, the three tiers in the three-tiered application can be segregated even further. An application with this type of architecture is called an *n*-tiered application. In this architecture, the business logic layer is divided by function instead of being physically divided. It breaks down like the following:
• **A user interface.** This handles the user interaction with the application. In an Internet application, this is usually a web browser used to render HTML tags.
• **Presentation logic.** This defines what the user interface displays and how user requests are handled.
• **Business logic.** This models the application business logic.
• **Infrastructure services.** These provide additional functionality required by the application components.
• **The data layer.** Hosts the application data.

Java 2, Enterprise Edition (J2EE) : JEE is not a product. Rather, it is a specification that defines the contract between applications and the container. The container here refers to a standardized runtime environment, which provides specific services for components deployed in it.
Developing Web Applications in Java : We Can normally adopt two main architectures when developing web applications in Java.
The first architecture utilizes servlets and JSP in the middle tier to serve clients and process the business logic. Small to medium-size applications use this design model.

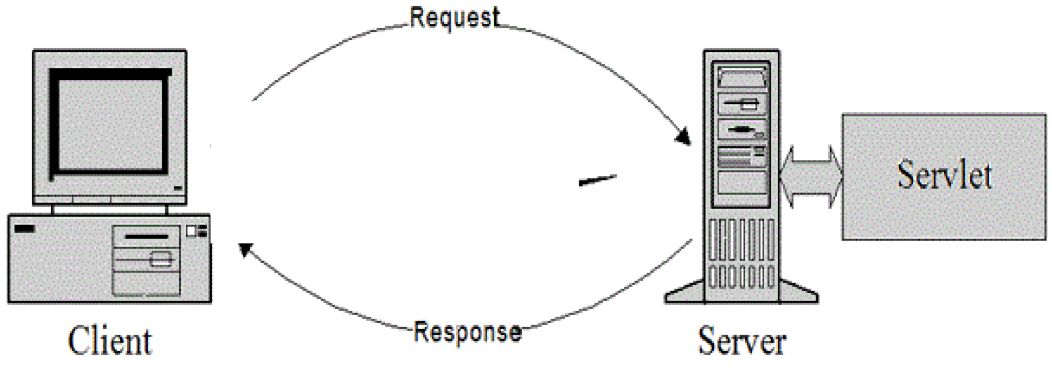


The second architecture includes the use of J2EE server and Enterprise JavaBeans (EJB) and this is especially useful for large enterprise applications that need scalability.



Introduction to Servlet
In the early days, web servers deliver static contents that are indifferent to users' requests. Java Servlet is server-side programs (running inside a web server) that handle clients' requests and return a customized or dynamic response for each request. The dynamic response could be based on user's input (e.g., search, online

shopping, online transaction) with data retrieved from databases or other applications, or time-sensitive data (such as news and stock prices).
Java Servlet typically runs on the HTTP protocol. HTTP is an asymmetrical request-response protocol. The client sends a request message to the server, and the server returns a response message.



What is Servlet?

- ✓ Servlet is a single instance multi thread java based server side web technology used to develop dynamic web resource program(page) having the capability to accept the request of the user and to provide respose(generate dynamic web pages).
- ✓ Servlet is a " single instance multiple threaded component" that accepts request from the user and provides response back to the user. It means when multiple request are given to Servlet program, the Servlet container creates only one object for that Servlet program class but multiple thread will be started on that object representing multiple request.
- ✓ It is a java program which executes in a web server for accepting request of the user and to provide appropriate response.
- ✓ This mechanism allows us to create dynamic web pages.
- ✓ A servlet is a Java class that can be loaded dynamically into and run by a special web server. This servlet-aware web server is called a servlet container, which also was called a servlet engine in the early days of the servlet technology.
- ✓ Servlets interact with clients via a request-response model based on HTTP. Because servlet technology works on top of HTTP, a servlet container must support HTTP as the protocol for client requests and server responses. However, a servlet container also can support similar protocols, such as HTTPS (HTTP over SSL) for secure transactions.
- ✓ It is one of the most important Java technologies, and it is the underlying technology for another popular Java technology for web application development called JavaServerPages (JSP).
- ✓ In a JSP application, the servlet container is replaced by a JSP container. Both the servlet container and the JSP container often are referred to as the web container or servlet/JSP container, especially if a web application consists of both servlets and JSP pages.

Static Web Page	**Dynamic web Pages**
Such type of page provides same context to all users.	Such type of page may provide different context for different users.
Response of this page is predefined before accepting user's request.	Response of this page gets predefined after accepting user's request.
This page does not need compilation or interpretation in sever.	This Page may need compilation or interpretation within the server.

Features of servlet : Since Servlet is object oriented hence it has all the features of java.

1. Object Oriented
2. Portability
3. Powerful
4. Efficiency due to Multi Threading
5. Safety.
6. Integration
7. Extensibility
8. Inexpensive

Object Oriented:- A Servlet is a java class. Hence it provides the benefits of code reuse by inheritance. A servlet is extendable.

Portability : As we know that the servlets are written in java and follow well known standardized APIs so they are highly portable across operating systems and server implementations.

We can develop a servlet on Windows machine running the tomcat server or any other server and later we can deploy that servlet effortlessly on any other operating system like Unix server running on the iPlanet/Netscape Application server. So servlets are writeonce, run anywhere (WORA) program.

Powerful : We can do several things with the servlets which were difficult or even impossible to do with CGI, for example the servlets can talk directly to the web server while the CGI programs can't do.

Servlets can share data among each other, they even make the database connection pools easy to implement. They can maintain the session by using the session tracking mechanism which helps them to maintain information from request to request. It can do many other things which are difficult to implement in the CGI programs.

Efficiency due to Multi Threading : As compared to CGI the servlets invocation is highly efficient. When the servlet get loaded in the server, it remains in the server's memory as a single object instance.

However with servlets there are N threads but only a single copy of the servlet class. Multiple concurrent requests are handled by separate threads so we can say that the servlets are highly scalable.(i.e. If multiple users will access a servlet, then the web server does not create multiple instances of the servlet. The web server creates separate thread for each user to access service methods of servlet. This process saves money of the server by not creating multiple instances of the servlets).

Safety : As servlets are written in java, servlets inherit the strong type safety of java language. Java's automatic garbage collection and a lack of pointers mean that servlets are generally safe from memory management problems.

In servlets we can easily handle the errors due to Java's exception handling mechanism. If any exception occurs then it will throw an exception.

Integration

Servlets are tightly integrated with the server. Servlet can use the server to translate the file paths, perform logging, check authorization, and MIME type mapping etc.

Extensibility : The servlet API is designed in such a way that it can be easily extensible. As it stands today, the servlet API support Http Servlets, but in later date it can be extended for another type of servlets.

Inexpensive : There are number of free web servers available for personal use or for commercial purpose. Web servers are relatively expensive. So by using the free available web servers you can add servlet supports to it.

Server independent:- Servlet will execute in all type of web server with the support of a servlet container. The servlet container provides run time environment for the servlets.

Servlet (and JSP) offers the following benefits that are not necessarily available in other technologies:
Performance : The performance of servlets is superior to CGI because there is no process creation for each client request. Instead, each request is handled by the servlet container process. After a servlet is finished processing a request, it stays resident in memory, waiting for another request.

Portability : Similar to other Java technologies, servlet applications are portable. You can move them to other operating systems without serious hassles.

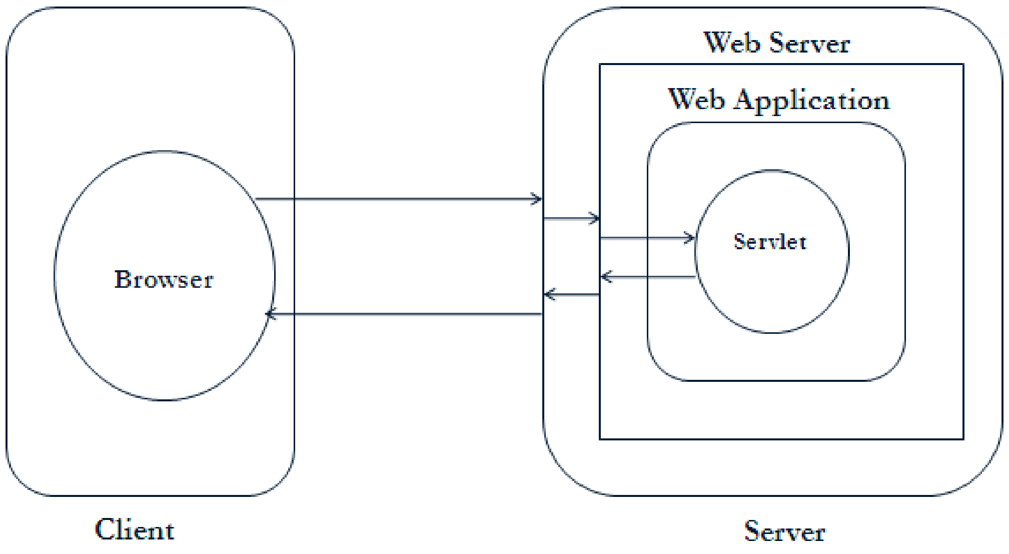
Rapid development cycle : As a Java technology, servlets have access to the rich Java library, which helps speed up the development process.

Robustness : Servlets are managed by the Java Virtual Machine. As such, you don't need to worry about memory leak or garbage collection, which helps you write robust applications.

Widespread acceptance : Java is a widely accepted technology. This means that numerous vendors work on Java-based technologies. One of the advantages of this widespread acceptance is that you can easily find and purchase components that suit your needs, which saves precious development time.

Software Requirement:-
* JDK:-This provides core packages and a compiler to the servlet.
*Servlet Container:-this Provide run time environment for the servlets. This can be Apache’s Tomcat, BEA’s Weblogic server, IBM’s Websphere, etc.

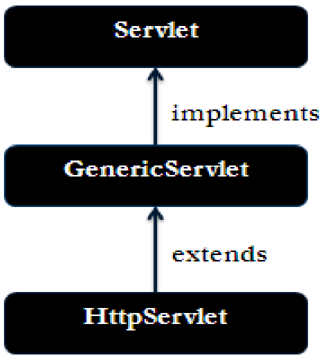
Servlet Architecture



Explanation:
- ✓ Browser window generates request to web application.
- ✓ Web server takes the request and passes to the appropriate web recourse program(servlet.jsp) of the web application.
- ✓ This web recourse program will process the request and return the result.
- ✓ Web resource program send the request to web server.
- ✓ Web server send the response to browser window as Http Response in form of web pages.

<u>Creation process of servlet :-</u> A servlet can be created by inheriting from Servlet interface or from the GenericServlet or HttpServlet class in a user defined class(The GenericServlet class is available in javax.servlet package and the HttpServlet class is defined in javax.servlet.http package).

Servlet Class Hierarchy



Servlet (Interface)

- ✓ javax.servlet.Servlet is one of pre-defined top most interface, which is containing life cycle methods init(), service() and destroy().
- ✓ Servlet interface contains five abstract methods, apart from above 2 method it has getServletConfig() & getServletInfo()

GenericServlet (Abstract Class)

- ✓ javax.servlet.GenericServlet is one of the predefined abstract class which is implementing javax.servlet.Servlet interface.
- ✓ This GenericServlet class is used for developing protocol independent applications
- ✓ GenericServlet contains one abstract method called sevice().

HttpServlet(Abstract Class)

- ✓ javax.servlet.http.HttpServlet is sub class of GenericServlet used for developing protocol dependent application.
- ✓ This abstract class contains no abstract method.

Developing Servlet Program

There are 3 approaches to develop the Servlet program:

Approach1 :

Take a java class and implements from javax.servlet.Servlet interface and provide implementation to all the five methods of that interface. Servlet interface contains following abstract method:

public void init(ServletConfig config)

- Called by the Servlet container to indicate to a Servlet that the Servlet is being placed into service.

public void service(ServletRequest req, ServletResponse res)throws ServletException,IOException

- Called by the Servlet container to allow the Servlet to respond to a request.

destroy()

- Called by the servlet container to indicate to a servlet that the servlet is being taken out of service.

public ServletConfig getServletConfig()

- Returns a ServletConfig object, which contains initialization and startup parameters for this servlet.

public String getServletInfo()

- Returns information about the servlet, such as author, version, and copyright.

Approach2:

Take a java class and extends from javax.servlet.GenericServlet class and provide implementation to only one abstract method of that class as given below :

public void service(ServletRequest req, ServletResponse res)throws ServletException,IOException

- Called by the Servlet container to allow the Servlet to respond to a request.

Approach3:

Take a java class and inherit from javax.servlet.http.HttpServlet class and provide logic for the servlet within doGet()/doPost()/doGet(), etc.

Note

Every Servlet program must implement Servlet interface directly or indirectly.

A Servlet's Life Cycle : A servlet life cycle can be defined as the entire process from its creation till the destruction. The Servlet interface in the javax.servlet package is the source of all activities in servlet programming. Servlet is the central abstraction of the Java servlet technology. Every servlet you write must implement this javax.servlet.Servlet interface, either directly or indirectly. The life cycle of a servlet is determined by three of its methods: init, service, and destroy. The following are the paths followed by a servlet:

- The servlet is initialized by calling the init () method.
- The servlet calls service() method to process a client's request.
- The servlet is terminated by calling the destroy() method. Finally, servlet is garbage collected by the garbage collector of the JVM.

The init() Method : public void init(ServletConfig config) throws ServletException

The init method is designed to be called only once. The init() method is called by the servlet container after it gets the first request from a user, then the web server instantiate the servlet to call the init(). The servlet container calls this method exactly once to indicate to the servlet that the servlet is being placed into service. The init method must complete successfully before the servlet can receive any further requests. You can override this method to write initialization code that needs to run only once, such as loading a database driver, initializing values, and so on. In other cases, you normally leave this method blank.

The init method is important also because the servlet container passes a ServletConfig object, which contains the configuration values stated in the web.xml file for this application. This method also can throw a ServletException. The servlet container cannot place the servlet into service if the init method throws a ServletException or the method does not return within a time period defined by the web server. (ServletException is the most important exception in servlet programming. In fact, a lot of methods in the javax.servlet and javax.servlet.http packages can throw this exception when a problem exists in a servlet).

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. Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT,DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Servlets typically run inside multithreaded servlet containers that can handle multiple requests concurrently. Therefore, you must be aware to synchronize access to any shared resources, such as files, network connections, and the servlet's class and instance variables. For example, if you open a file and write to that file from a servlet, you need to remember that a different thread of the same servlet also can open the same file. In that case we have to create Thread-Safe Servlets

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

The destroy() Method : public void destroy()

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, write cookie lists or hit counts to disk, and perform other such cleanup activities. After the destroy() method is called, the servlet object is marked for garbage collection.

The servlet container calls the destroy method before removing a servlet instance from service. This normally happens when the servlet container is shut down or the servlet container needs some free memory. This method is called only after all threads within the servlet's service method have exited or after a timeout period has passed. After the servlet container calls this method, it will not call the service method again on this servlet.

The destroy method gives the servlet an opportunity to clean up any resources that are being held (for example, memory, file handles, and threads) and make sure that any persistent state is synchronized with the servlet's current state in memory.

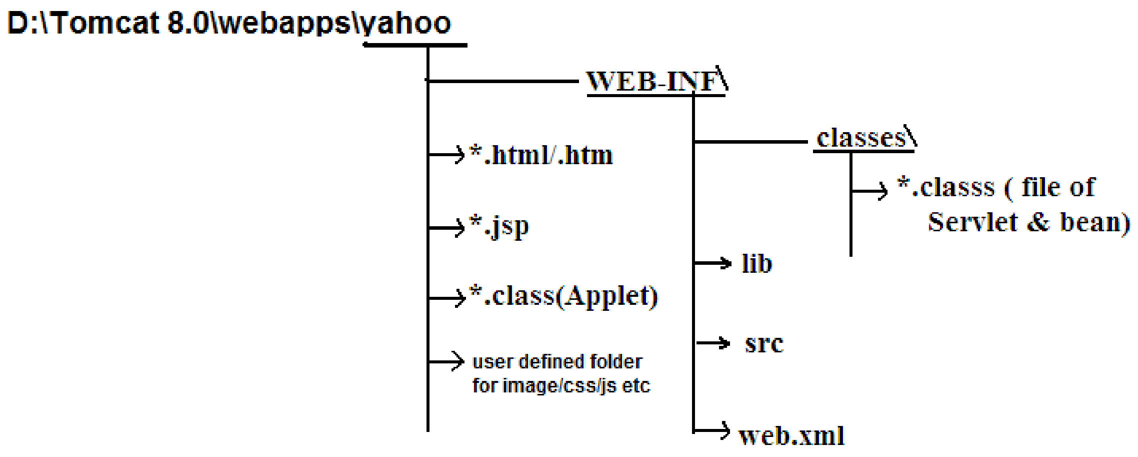
```
public void destroy() {
        // Finalization code...
 }
```

How Servlet Runs : A servlet is loaded by the servlet container the first time the servlet is requested. The servlet then is forwarded the user request, processes it, and returns the response to the servlet container, which in turn sends the response back to the user. After that, the servlet stays in memory waiting for other requests—it will not be unloaded from the memory unless the servlet container sees a shortage of memory. Each time the servlet is requested, however, the servlet container compares the timestamp of the loaded servlet with the servlet class file. If the class file timestamp is more recent, the servlet is reloaded into memory. This way, you don't need to restart the servlet container every time you update your servlet.

Six Steps to Running Your First Servlet : After you have installed and configured Tomcat, you can put it into service. Basically, you need to follow six steps to go from writing your servlet to running it. These steps are as follows:

1. Create a **directory structure** under Tomcat for your application(context folder/root directory/home directory)
2. Write the **servlet source code**. You need to import the javax.servlet package and the javax.servlet.http package in your source file & store the file under classes folder
3. **Compile** your source code, by using temporary/permanent class.
4. Create a **deployment descriptor** in web.xml file.
5. **Run Tomcat web server**.
6. **Call your servlet** from a web browser

<u>STEP 1 :- Context folder:</u> This is a name which refers to a directory structure of a file system. This directory contains the web pages and the files required for the web site. It is the **home/root** directory of a web site, sometimes the name of this folder remains equal to the web site name. The context directory follows a pre defined rule for having the name of its sub directories. IT is also called home directory of the web site(say yahoo will be the context folder name).


D:\Tomcat 8.0\webapps\yahoo
WEB-INF\
*.html/.htm
*.jsp
*.class(Applet)
user defined folder for image/css/js etc
classes\
*.classs (file of Servlet & bean)
lib
src
web.xml

- classes contains all .class file of servlet, bean, etc, it also contains all user defined packages.
- lib contains all **.jar** files, **.rar** files, etc.

<u>How to create the context folder :</u>

1. Create a sub directory(i.e. folder) under Tomcat 8.0\webapps folder using name as yahoo and this directory will be called as context directory
2. Create a folder within the context directory by using the name "WEB-INF" in upper case Eg: - D:\Tomcat 8.0\webapps\yahoo\WEB-INF
3. Create a sub directory with in the WEB-INF directory by using the class name "classes". Eg: -
4. D:\Tomcat 8.0\webapps\yahoo\ WEB-INF\ classes
5. Search for web.xml within Tomcat's installation directory. Copy any one of the web.xml file and save it inside WEB-INF directory of this context folder.
6. Open this file in "notepad" and remove all the contents present within <web-app> & </web-app> tags.

<u>STEP- 2: Writing Servlet class by inheriting From GenericServlet class</u> :-
Store the file as GenServer.java under classes folder

```
import  java.io.*;
import  javax.servlet.*;
public class GenServer extends GenericServlet {
  public void service(ServletRequest req, ServletResponse res) throws IOException, ServletException {
      PrintWriter  out = res.getWriter();
      out.println("<html><body bgcolor=blue>");
      out.println("<h1>First servlet</h1>");
      out.println("</body></html>");
}}
```

✓ The class of our Servlet program must be public class in order to make it visibile to Servlet container.
✓ Servlet container creates object of Servlet class and calls service() method on that object for multiple times, for multiple requests on one per request basis.
✓ For every request is given to Servlet program the Servlet container creates one set of (request, response) objects and calls service() method having request, response objects as arguments.

<u>STEP -3 : Storing & Compiling the servlet: -</u>

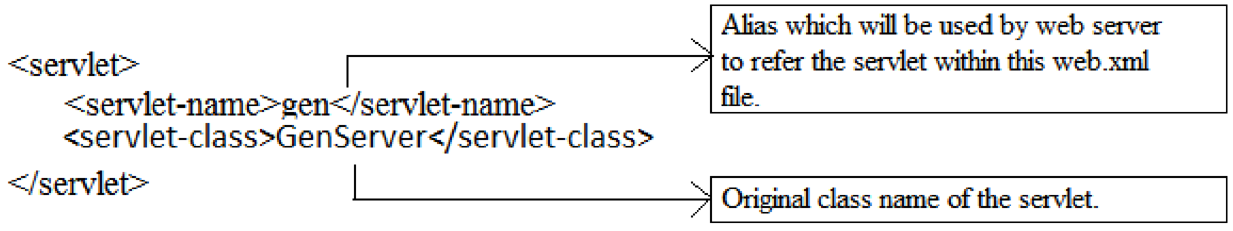
- Let us **store** the GenServer.java under the "classes" directory of the context folder.

- To compile the GenServer.java we have to copy **servlet-api.jar** file from the Tomcat installations directory (….\Tomcat x.x\lib) into the "classes " folder of the context folder.
- The following command must be used to **compile** the GenServer.java :
  -------\WEB-INF\classes> javac –cp servlet-api.jar;. GenServer.java

STEP-4 : Creating Deployment Descripter File(web.xml) :-

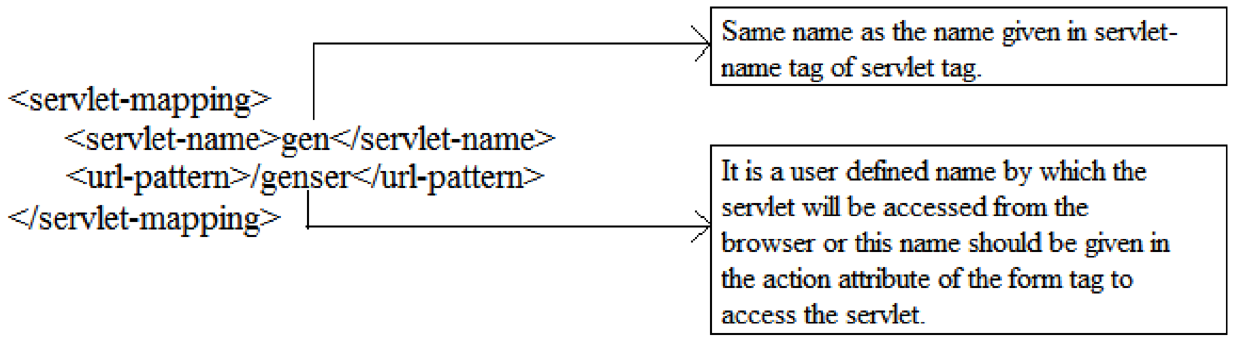
This is a process to make available the servlet class to the web site, so that the servlet can be accessible throw an URL given in the address bar. This information can be given in a deployment descriptor file (web.xml). The web.xml file specifies the details about certain resource and making underlying container or server recognizing that resource, these called as resource configuration. Every Servlet program must be configured in web.xml specifying logical name, class name, and identity name (URL name) to make the Servlet container recognizing the Servlet program.

This can be done by providing information about the servlet in the predefined tag within web.xml file. Open web.xml in notepad and providing the following tag within<web-app> & </web-app>



```
<servlet>
     <servlet-name>gen</servlet-name>
     <servlet-class>GenServer</servlet-class>
</servlet>
```

Alias which will be used by web server to refer the servlet within this web.xml file.
Original class name of the servlet.



```
<servlet-mapping>
     <servlet-name>gen</servlet-name>
     <url-pattern>/genser</url-pattern>
</servlet-mapping>
```

Same name as the name given in servlet-name tag of servlet tag.
It is a user defined name by which the servlet will be accessed from the browser or this name should be given in the action attribute of the form tag to access the servlet.

- ✓ The above given logical name becomes object name for our Servlet class, when Servlet container creates object for it.
- ✓ Every Servlet program is identified with its url-pattern in Servlet based executing environment.
- ✓ Servlet container reads web.xml file the moment web application deployed, so it is called DD file (Deployment Descriptor).
- ✓ Servlet container, web server, clients other web resource programs can't identified Servlet program through its logical name or class name. but they identified Servlet program through its URL pattern.

STEP-5 : Run tomcat

STEP-6 : Calling Servlet from browser:

Open the browser and type http://localhost:8085/yahoo to access the web site.
Open the browser and type http://localhost:8085/yahoo/genser to access the servlet.

The javax.servlet Package contains seven interfaces, three classes, and two exceptions.

Creating servlet by using HttpServlet :-

1. Createie a class by inheriting javax.servlet.http.HttpServlet.

2. Override method of "HttpServlet" class to implement logic of the servlet . those method are as follows
i. <u>public void init()</u> <u>:</u>- Executes during initialization of servlet class.
ii. public viod doGet/doPost(HttpServletRequest req, HttpServletResponse res);
This method execute on acceptance of request from the user. If the request comes in 'get' method then doGet()to be overridden if the request comes in 'post' method then doPost()to be overridden.
iii. <u>public void destroy()</u> :- this executes on removal of servlet's instance.
3. Create an object of "PrintWriter" byusing"getWriter()" of HttpServerletRespons use "println()" of "PrintWriter" to print the data on the browser.

<u>USING HttpServlet</u> :-

```
import   javax.servlet.*.;
import   javax.servlet.http.*;
import   java.io.*;
public class ServDemo extends HttpServlet {
public void doGet(HttpServletRequest req, HttpServletResponse res)throws IOException, ServletException {
     PrintWriter out = res.getWriter();
     out.println("<html><body>");
     out.println("<h1>The Http type servlet</h1>");
     out.println("</body></html>");
}}
```

Q-1: Create a servlet which will create a HTML table having 30 rows and 20column Each call of the table will contain your name

Q-2: Create a servlet to display current data and time.

INTERFACES	CLASSES	EXCEPTION CLASSES
• RequestDispatcher • Servlet • ServletConfig • ServletContext • ServletRequest • ServletResponse • SingleThreadModel	• GenericServlet • ServletInputStream • ServletOutputStream	• ServletException • UnavailableException

Passing values from html to Servlet

Html file	**Servlet file (url-pattern is 'addser')**

<html><body bgcolor="pink">
<h1>Addition of Numbers
 	
<form action="addser" method="get">
First Number
<input type="text" name="t1">
 	
Second Number
<input type="text" name="t2">
 	
<input type="submit" value="Submit">
<input type="reset" value="Reset">
</form></h1>
</body></html>
	import java.io.*;
	import javax.servlet.*;
	public class AddServlet extends GenericServlet {
	public void service(ServletRequest req, ServletResponse res)
	throws IOException, ServletException {
	PrintWriter out = res.getWriter();
	out.println("<html><body bgcolor=cyan>");
	int x= Integer.parseInt(req.getParameter("t1"));
	int y= Integer.parseInt(req.getParameter("t2"));
	out.println("<h1> Addtion of "+x+" & "+y +" = " + (x+y));
	out.print("</body></html>");
	}};

<u>Submitting the data to itself :</u> When a servlet generates a response in the form of html file consisting of a form tag, then the data given by the user in the html file will be submitted to the webpage(servlet/jsp) whose name/URL pattern is given in action attribute of form tag. If we do not provide the action attribute or we provide null string("")as the value of action attribute, then the data will be submitted to the same webpage.

```
import   java.io.*;
import   java.sql.*;
import   javax.servlet.*;
import   javax.servlet.http.*;
public  class  SameSer extends HttpServlet  {
public  void  doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException  {
try {
PrintWriter out= res.getWriter();
out.println("<html><body bgcolor=cyan><h1>");
out.println("Sending  data from SERVLET to itself<br>");
out.println("Even Odd test<br>");
out.println("<form>Enter a No. <input  type=text name=t1> <input   type=submit value=Go></form>");
try{
  int x= Integer.parseInt(req.getParameter("t1"));
  if(  x%2==0)
    out.println(x+" is Even No.");
  else
    out.println(x+" is Odd No.");
}catch(Exception e){    }
out.println("</h1></body></html>");
}};
```

Creating a servlet by inheriting from Servlet interface: The PrimServlet class implements Servlet interface of javax.servlet and provides implementations for all the five methods of servlet.

```
import javax.servlet.*;
import java.io.*;
public class PrimServlet implements Servlet {
public void init(ServletConfig config) throws ServletException {
        System.out.println("init");
}
public void service(ServletRequest request, ServletResponse  response) throws ServletException, IOException {
        System.out.println("service");
}
public void destroy() {
        System.out.println("destroy");
}
public String getServletInfo() {
        return null;
}
public ServletConfig getServletConfig() {
        return null;
}}
```

Configure this class with url-pattern as 'primitive' in web-xml, call this servlet from your browser by typing the following URL:

```
http://localhost:8085/yahoo/primitive
```

The first time the servlet is called, the console displays these two lines:

```
init
service
```

This tells you that the init method is called, followed by the service method. However, on subsequent requests, only the service method is called. The servlet adds the following line to the console:

```
service
```

You can return any string in the getServletInfo method, such as your company name or the author name or other information whatever is necessary, The getServletConfig is more important.

Example :

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

public class First implements Servlet{
        ServletConfig config=null;
        public void init(ServletConfig config){
                this.config=config;
                System.out.println("servlet is initialized");
        }
        public void service(ServletRequest req,ServletResponse res)  throws IOException,ServletException{
                res.setContentType("text/html");
                PrintWriter out=res.getWriter();
                out.print("<html><body>");
                out.print("<b>Lakshya</b>");
                out.print("</body></html>");
        }
        public void destroy(){
                System.out.println("servlet is destroyed");
        }
        public ServletConfig getServletConfig(){
                return config;
        }
        public String getServletInfo(){
                return "copyright 2014";
        }
 }
```

Database Access from Servlet :

Html file

```
<html><body bgcolor="cyan">
<h1>Enter A record of Student Table<br>      <form action="instud" method="get">
Name <input name="t1" type="text" /><br>   Roll <input name="t2" type="text" /><br>
Sem <input name="t3" type="text" /><br>    Mark <input name="t4" type="text" /><br>
<input type="submit" value="Submit"  />      <input type="reset" value="Reset" />
</form></h1></body></html>
```

Servlet file (url-pattern is `instud')

```
import  java.io.*;
import  java.sql.*;
import  javax.servlet.*;
public  class InsertStud  extends  GenericServlet {
public void service(ServletRequest  req, ServletResponse  res) throws IOException, ServletException {
PrintWriter out = res.getWriter();
out.print("<html><body bgcolor=orange>");
String  s1 = req.getParameter("t1");    String  s2 = req.getParameter("t2");
String  s3 = req.getParameter("t3");    String  s4 = req.getParameter("t4");
try{
        Class.forName("com.mysql.jdbc.Driver");
        Connection con = DriverManager.getConnection("jdbc:mysql://localhost/college", "root", "");
        Statement  st=con.createStatement();
        String  qry="insert into stud values('"+s1+"', "+s2+", '"+s3+"', "+s4+")";
        st.executeUpdate(qry);
        out.print("Record inserted");
        con.close();
}catch(Exception e) {
  out.print(e);
}
out.print("</body></html>");
}};
```

ServletConfig   interface : The SerlvetConfig is used to receive initial parameter to a servlet. Why do you want to use an initial parameter ? For practicality, hard coding information in the servlet code means that you have to recompile the servlet if the information changes. A web.xml file is plain text. You can edit its content easily using a text editor.

An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file. If the configuration information is modified from the web.xml file, then no need to change the servlet. So it is easier to manage the web application if any specific content is modified from time to time.

The getServletConfig() method of Servlet interface returns the object of ServletConfig interface.

1. public String **getInitParameter**(String name):Returns the parameter value for the specified parameter name.
2. public Enumeration **getInitParameterNames**():Returns an enumeration of all the initialization parameter names.
3. public String **getServletName**():Returns the name of the servlet.
4. public ServletContext **getServletContext**():Returns an object of ServletContext

Example :

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class DemoServlet extends HttpServlet  {
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException  {
                     response.setContentType("text/html");
                    PrintWriter out = response.getWriter();
                    ServletConfig config=getServletConfig();
                    String driver=config.getInitParameter("driver");
                    String url=config.getInitParameter("jdbcurl");
                    String user=config.getInitParameter("user");
                    String pwd=config.getInitParameter("pwd");
                    out.print("Driver is: "+driver);
                    out.print("JDBC URL is: "+url);
                    out.print("User is: "+user);
```

```
                out.print("Password is: "+ pwd);
} }
```

web.xml :

```
<web-app>
<servlet>
        <servlet-name>DemoServlet</servlet-name>
        <servlet-class>DemoServlet</servlet-class>
        <init-param>
                <param-name>driver</param-name>
                <param-value>oracle.jdbc.driver.OracleDriver</param-value>
        </init-param>
        <init-param>
                <param-name>jdbcurl</param-name>
                <param-value>jdbc:oracle:thin:@localhost:1521:XE</param-value>
        </init-param>
        <init-param>
                <param-name>user</param-name>
                <param-value>system</param-value>
        </init-param>
        <init-param>
                <param-name>pwd</param-name>
                <param-value>manager</param-value>
        </init-param>
        </servlet>
<servlet-mapping>
        <servlet-name>DemoServlet</servlet-name>
        <url-pattern>/servletconfig</url-pattern>
</servlet-mapping>
</web-app>
```

The servlet container passes a ServletConfig object to the init method, it is easiest to write the code in the init method. The program loops through the Enumeration object called parameters that is returned from the getInitParameterNames method. For each parameter, it outputs the parameter name and value. The parameter value is retrieved using the getInitParameter method.

Example :

```
import javax.servlet.*;
import java.util.Enumeration;
import java.io.IOException;
public class ConfigDemoServlet implements Servlet {
public void init(ServletConfig  cfg) throws ServletException {
        Enumeration p = cfg.getInitParameterNames();
        while (p.hasMoreElements()) {
                String para = (String) p.nextElement();
                System.out.println("Parameter name : " + para);
                System.out.println("Parameter value : " +cfg.getInitParameter(para));
        }
}
public void destroy() {         }
public void service(ServletRequest req, ServletResponse  res) throws ServletException, IOException {}
public String getServletInfo() {            return null;       }
public ServletConfig getServletConfig() {          return null;       }
}
```

Write the following code in **web.xml** file :

```
<servlet>
<servlet-name>ConfigDemo</servlet-name>
<servlet-class>ConfigDemoServlet</servlet-class>
<init-param>
<param-name>adminName</param-name>
<param-value>Gitesh </param-value>
</init-param>
<init-param>
<param-name>adminEmail</param-name>
<param-value>giteshmallick@gmail.com</param-value>
</init-param>
<init-param>
<param-name>ContactNumber</param-name>
<param-value>09437100566</param-value>
</init-param>
</servlet>
```

User Tracking

Session/User Tracking is a way to maintain state (data) of an user. It is also known as session management in servlet. This mechanism allows the website to propagate the user's information to all pages(i.e. servlets or JSP's) of the web site or this mechanism allows the web server to track/remember the user's activity in all pages of the web site.

Whenever a form is being submitted to a servlet(A) then the value of the form fields can be available to that servlet(A) only. Whenever user sends another request from current servlet(A) to the server then the value of those form fields are being lost in the next page(the next page may be another servelet(B), or jsp). Hence consecutive request cannot have any relation with each other . This nature is known as state-less nature of the protocol. 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.

User-tracking is a mechanism to avoid state-less nature within the web application. This makes available information about the user to more than one servlet or request. This can be done by using

(a) URL rewriting
(b) Hidden field
(c) Cookie
(d) Session

URL Rewriting : This is the mechanism to create an explicit query string by using hyperlink . The 'href' attribute of <A> contains the query string with the destination's URL . Whenever the user clicks on the hyper text to visit the destination then the query string also travels to the destination. If the destination is the servlet then it can find the value of strings present in query string through getParameter() method of request.

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&??

Store it as URLTest.html :

```
<html><body bgcolor="yellow">
<h1>Url Demo<br>
<form action="ur1" method="get">
User Name <input name="t1" type="text" /><br>
Password <input name="t2" type="password" /><br>
<input type="submit" value="Submit"  /> <input type="reset" value="Reset" /> </form>
</h1></body></html>
```

Note: When we click the submit button in above html file by providing 'ram' in username & 'r' in Password field then the address bar will hold the URL as :
http://localhost:8085/yahoo/url1?t1=ram&t2=r
The "?t1=rama&t2=r" is called query string.

Store it as URServ1.java : (url-pattern is url1)

```
import   java.io.*;
import   javax.servlet.*;
import   javax.servlet.http.*;
public  class URServ1 extends HttpServlet {
 public void  doGet(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException {
 PrintWriter out=res.getWriter();
 out.print("<html><body bgcolor=cyan>");
 String  s1= req.getParameter("t1");
 String  s2= req.getParameter("t2");
 out.println("<h1>Welcome to Url rewriting</h1>");
 out.println("Hello "+s1);
  out.println("<a href=ur2?a1="+s1+"&a2="+s2+">Click Me</a>");
 out.print("</body></html>");
}};
```

Here "a1="+s1+"&a2="+s2+" is called user defined query string, where a1 & a2 are user defined variable name which will be used by getParameter() in the next servlet.

Store it as URServ2.java : (url-pattern is url2)

```
import   java.io.*;
import   javax.servlet.*;
import   javax.servlet.http.*;
public  class URServ2 extends HttpServlet {
 public void  doGet(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException {
 String  s1= req.getParameter("a1");
 String  s2= req.getParameter("a2");
 PrintWriter out=res.getWriter();
 out.print("<html><body bgcolor=orange>");
 out.print("<h1>Welcome to Url rewriting</h1>");
 out.print("Hello "+s1);
 out.print("<br>ur password "+s2);
 out.print("</body></html>");
}};
```

Advantage of URL Rewriting

It will always work whether cookie is disabled or not (browser independent).
Extra form submission is not required on each pages.

Disadvantage of URL Rewriting

It will work only with links.

It can send only textual information.

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

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

This is a mechanism to create a hidden field within a form and the hidden field contains value of a variable present in current request. Whenever the form containing the hidden field is being submitted then the destination servlet can use the value available in hidden field through getParameter() method.

Store it as Hiddentest.html :

```
<html><body bgcolor=olive><h1>using hidden field<br>
<form action="hid1"> Enter Book Name: &nbsp;&nbsp;&nbsp; <input type="text" name="t1"><br>
Auther: &nbsp;&nbsp;&nbsp;&nbsp; <input type="text" name="t2"><br>
<input type=submit value="submit" name="b1"></h1></body></html>
```

Store it as Hidden1.java : (url-pattern is hid1)

```
import    java.io.*;
import    javax.servlet.*;
import    javax.servlet.http.*;
public  class Hidden1 extends HttpServlet{
  public void doGet(HttpServletRequest  req,  HttpServletResponse  res) throws IOException, ServletException {
String  s1 = req.getParameter("t1");
String  s2 =req.getParameter("t2");
PrintWriter pw= res.getWriter();
pw.println("<html><body bgcolor=yellow><h1>");
pw.println("First Servlet <form action=hid2>");
pw. println("UR looking for "+s1 +" book writen by "+s2 +"<br>Enter UR Name");
pw. println("<input type=text name=tf><br>");
pw. println("<input type=submit value=submit> <br>");
pw. println("<input type=hidden name=t2 value="+s1+"><br>");
pw. println("<input type=hidden name=t3 value="+s2+"><br>");
pw.println("</h1></form></body></html>");
}};
```

Store it as Hidden2.java : (url-pattern is hid2)

```
import    java.io.*;
import    javax.servlet.*;
import    javax.servlet.http.*;
public  class Hidden2 extends HttpServlet{
  public void doGet(HttpServletRequest  req,  HttpServletResponse  res) throws IOException, ServletException {
String  s1 = req.getParameter("tf");
String  s2 = req.getParameter("t2");
String  s3 = req.getParameter("t3");
PrintWriter pw= res.getWriter();
pw.println("<html><body bgcolor=yellow><h1>");
pw. println("Hello Mr :"+s1 +"<br> The "+s2+ " book writen by " + s3 +"<br> is available</h1>");
pw.println("</body></html>");
}};
```

Advantage of Hidden Form Field : It will always work whether cookie is disabled or not.

Disadvantage of Hidden Form Field:

It is maintained at server side.
Extra form submission is required on each pages.
Only textual information can be used.

We can implement url rewriting when we visit from one place to another by clicking on a hyperlink, similarly we can implement hidden field when we visit from on place to another by clicking on the submit button.

We can use url rewriting and hidden field to track the user or to keep the user's activity, but as the number of web pages(servlet,jsp)increases then it is an overhead
to the programmer to implement above two mechanism i.e the programmer has to write the code for url-rewriting and the hidden field in all web pages.

To reduce the burden of the programmer Sun has given the concept of cookie and session.These two mechanism behaves like a global variable which
can be accessible by all the webpagesof the website.

COOKIE : Cookie is a memory location available in user’s machine to store information about the user. The web server can create cookie object and sends it to the client through the response. Whenever user sends any request then all cookie variables created by the server goes back to the server as part of the request automatically. This variables can be available in user’s memory for a period of 24 hours by default. The default life span of cookie can be changed by the server. Each browser has the option to allow or restrict the cookie.

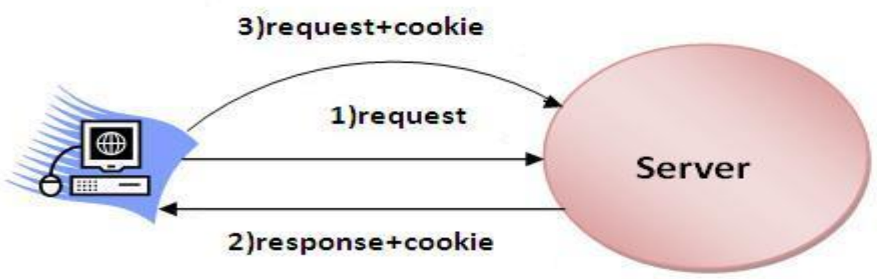
Cookies are small bits of textual information that a Web server sends to a browser and that the browser returns unchanged when visiting the same Web site or domain later. By having the server read information it sent the client previously, the site can provide visitors with a number of conveniences

Identifying a user during an e-commerce session. Many on-line stores use a ”shopping cart” metaphor in which the user selects an item, adds it to his shopping cart, then continues shopping. Since the HTTP connection is closed after each page is sent, when the user selects a new item for his cart, how does the store know that he is the same user that put the previous item in his cart? Cookies are a good way of accomplishing this. In fact, this is so useful that servlets have an API specifically for this, and servlet authors don’t need to manipulate cookies directly to make use of it.

Avoiding username and password. Many large sites require you to register in order to use their services, but it is inconvenient to remember the username and password. Cookies are a good alternative for low-security sites. When a user registers, a cookie is sent with a unique user ID. When the client reconnects at a later date, the user ID is returned, the server looks it up, determines it belongs to a registered user, and doesn’t require an explicit username and password.

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.

- Non-persistent cookie
- Persistent cookie

In-memory / Non-persistent cookie
It is valid for single session only. It is removed each time when user closes the browser.
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 sign out.

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.

Methods of Cookie class

Method	Description
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie. The name cannot be changed after creation.
public String getValue()	Returns the value of the cookie.
public void setValue(String value)	changes the value of the cookie.
public void setName(String nm)	To change name of a cookie.

<u>Creation of Cookie in Servlet</u> :-

1. Create a servlet by inheriting javax.servlet.http.HttpServlet .
2. Create an object of javax.servlet.http.Cookie class by using variable name and value to its constructor as

   String str = req.getParameter("t1"); //say str holds the user name
   Cookie c = new Cookie("nm", str); //nm is the user defined Cookie variable name, and str is the value to the Cookie variable.
3. Add the cookie into the response by using addCookie() method of HttpServletResponse as

   res.addCookie(c);

<u>Reading Value Of Cookie Variables</u> :-

1. Creating a servlet by inheriting javax.servlet.http.HttpServlet .
2. Read all cookies present in the request by using getCookies() method of HttpServletRequest. This method returns an array of cookies. As

   Cookie c[] = req.getCookies();
3. Read the variable name by using getName() method and read variable's value by using getValue() method. As

```
                String s=null;
                for(int i = 0;  i<c.length; i++){
                        if (c[i].getName().equals("nm"))
                                s = c[i].getValue();
                }
```

<u>Store it as Cookietest.html :</u>

```
<html><body bgcolor="orange"><h1><form action="cook1">
Enter  Your  Name<input type=text  name=t1><br>
Enter  Password<input type=text  name=t2><br>
<input type="submit"></form></h1></body></html>
```

<u>Store it as   Cookie1.java :</u> (url-pattern is cook1)

```
import   java.io.*;
import   javax.servlet.*;
import   javax.servlet.http.*;
public class  Cookie1  extends HttpServlet {
public  void  doGet( HttpServletRequest   req,
HttpServletResponse    res) throws  ServletException, IOException {
String s1=req.getParameter("t1");
String s2=req.getParameter("t2");
Cookie c1=new  Cookie("name", s1);
Cookie c2=new  Cookie("pass", s2);
res.addCookie(c1);
res.addCookie(c2);
PrintWriter pw = res.getWriter();
pw.println("<html><body bgcolor=red>" );
pw.println("<h1>Welcome to COOKIE");
pw.println("<form action=cook2><h1>Testing ur Cookie<br><br><input type=submit
value=Submit></form>");
pw.println("</h1></body></html>");
}};
```

<u>Store it as   Cookie2.java :</u> (url-pattern is cook2)

```
import   java.io.*;
import   javax.servlet.*;
import   javax.servlet.http.*;
public class  Cookie2  extends HttpServlet {
public  void  doGet( HttpServletRequest   req, HttpServletResponse   res) throws  ServletException, IOException
{
Cookie ck[]= req.getCookies();
PrintWriter pw = res.getWriter();
pw.println("<html><body bgcolor=red>" );
pw.println("<h1>cookie name & values<br>");
for(int i=0; i< ck.length;i++)
        pw.println("Cookie name : "+ ck[i].getName() + "  value is : " + ck[i].getValue()+"<br>");
pw.println("</h1></body></html>");
}};
```

<u>Creating the session:</u> The amount of time period for which a user is stay connected to the server is called session.

In case of cookie the server creates the cookie object and sends it to the client machine. The client resends the cookie back to the server in next/further request ,but the session is created by server and stored in server only in a session id(an integer value), which is given to the client. To keep multiple information about the user we have to create multiple cookie object,but in case of session the multiple information of the user can be kept within one session. The session behaves like an index table or hash table to store user's information. In case of cookie we have to create many object of Cookie class to store user's information, but in one reference of Session we can provide many variable-value

In such 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:

public HttpSession getSession(): Returns the current session associated with this request, or if the request does not have a session, creates one.

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

public String getId(): Returns a string containing the unique identifier value.

public long getCreationTime(): Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.

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.

public void invalidate(): Invalidates this session then unbinds any objects bound to it.

Creating and storing value in session :

The HttpServletRequest interface has a method called getSession() by which we will get a reference of HttpSession interface, and to store different information the setAttribute() of the HttpSession interface must be used as given below:

setAttribute(String variablename, Object values);

ex: HttpSession ss = req.getSession();
setAttribute("name", "raja"); // 'name' is the user defined Session variable name, and 'raja' is the value to the Session variable.
setAttribute("roll", 101);

Accessing the value of the session : A webpage in which we like to fetch the values of the session must use getAttribute() of the HttpSession interface by providing the variable name in the session, but first of all the webpage must get the refernce of session from the HttpServletRequest interface as given above, then we can use getAtribute() method of HttpSession interface as given below:

public Object getAttribute(String variablename)

ex: HttpSession ss = req.getSession();
String na = (String) ss.getAttribute("name");
int ro = (Integer) ss.getAttribute("roll");

Methods of HttpSession :-

- public void setAttribute(String var-name, Object value)
- public Object getAttribute(String var-name)
- public String getId() : It returns Session-Id
- public void setMaxInactiveInterval(long seconds) : To change default inactive interval.
- public void invalidate() : To destroy the session .
- public void removeAttribute(String var-name) : To remove a variable from session .

Store it as Sessiontest.html :

```
<html><body bgcolor="yellow" ><form   action="sess1">
<h1>Enter ur name  &nbsp;&nbsp; <input type="text" name="t1"> <br>
Enter password  &nbsp;&nbsp; <input type="text" name="t2">
<br><input type="submit" value="Submit"></h1></form></body></html>
```

Store it as Session1.java (url-pattern is sess1):

```
import   java.io.*;
import   javax.servlet.*;
import   javax.servlet.http.*;
public class  Session1  extends HttpServlet {
public  void  doGet( HttpServletRequest   req,
HttpServletResponse   res) throws  ServletException, IOException {
String s1=req.getParameter("t1");
String s2=req.getParameter("t2");
HttpSession ss =req.getSession();
ss.setAttribute("uname",s1);
ss.setAttribute("pwd", s2);
PrintWriter pw = res.getWriter();
pw.println("<html><body bgcolor=red>" );
pw.println("<h1>Welcome to INBOX : success ");
pw.println("<form action=sess2><h1>Tessting ur Session<br><br><input type=submit
value=Submit></form>");
pw.println("</h1></body></html>");
}}
```

Store it as Session2.java (url-pattern is sess2):

```
import   java.io.*;
import   javax.servlet.*;
import   javax.servlet.http.*;
public class  Session2  extends HttpServlet {
public  void  doGet( HttpServletRequest   req,
HttpServletResponse   res) throws  ServletException, IOException {
HttpSession ss =req.getSession();
String p1=(String) ss.getAttribute("uname");
String p2=(String) ss.getAttribute("pwd");
PrintWriter pw = res.getWriter();
pw.println("<html><body bgcolor=red>" );
pw.println("<h1>The value of ur session is :<br>");
pw.println("user name : " + p1+ "<br>");
pw.println("password  : " + p2+ "<br>");
pw.println("</h1></body></html>");
}};
```

Applet-Servlet InterCommunication :

Store it as AppServ.html :

```
<html><body bgcolor= "pink"><h1>Applet-Servlet intercommunication <br>
 </h1><applet code="AppServ.class" width="700" height="400"></h1></applet></body></html>
```

Store it as AppServ.java :

```
import  java.applet.*;
import  java.awt.*;
import  java.awt.event.*;
import  java.net.*;
import  java.io.*;
public class AppServ extends Applet  implements  ActionListener {
  Button  b1 ;
  TextField t1;
  TextArea ta;
public   void  init(){
   setFont(new  Font("ZSDf", Font.BOLD, 45));
   b1 = new Button("Ok");
   t1=new TextField(13);
   ta=new TextArea(3,12);
   b1.addActionListener(this);
   add(t1);
   add(b1);
   add(ta);
}
public  void actionPerformed(ActionEvent ar){
 try {
    String  a =  t1.getText();
    URL ur= new URL("http://localhost:8085/Yahoo/AppSe?t1="+a);
    URLConnection  uc= ur.openConnection();
    BufferedReader br =new BufferedReader(new InputStreamReader(uc.getInputStream()));
    String  z="";
   while( (z=br.readLine()) != null)
     ta.append(z+ "\n");
     br.close();
  }catch(Exception e) {  }
 }};
```

Store it as AppServlet.java :

```
import   java.io.*;
import   javax.servlet.*;
import   javax.servlet.http.*;
public class AppServlet  extends HttpServlet {
 public  void  doGet( HttpServletRequest req,
HttpServletResponse res) throws IOException, ServletException {
String s1 = req.getParameter("t1");
PrintWriter pw=res.getWriter();
pw.println("Welcome "+ s1);
/* the string given in println()  will be available to the applet , bcoz the string given in the println() behaves like
the response of the servlet */
}};
```

URL(Uniform Resource Locator)- This class of java.net package is used to hold the path of a resource on the internet/intranet. The resource may be a website or a file on the internet/intranet. Hence the constructor of this class will accept an URL (address given in browser). The openConnection() of URL class will establish a connection between the java program & the resource ,it will return an object of URLConnection class.

URLConnection- This class is used to perform read /write operation to the resource by its getInputStream()/getOutputStream()respectively in terms of byte.

Permanent servlet :A servlet which is started when the website is started as well as the servlet is destroyed when the website is stopped /undeployed is called a permanent servlet. Generally this kind of servlet is not accessed by the user, it does not provide any user interaction.This kind of servlet are used to keep information about the error or the exception that occurs during the execution of any we page available in the website,such as Log.txt file that keeps all valuable information of the website or information about the exception&error.

Store it as PermServ.java

```
import   java.io.*;
import   java.util.*;
import   javax.servlet.*;
import   javax.servlet.http.*;
public  class PermServ extends HttpServlet{
public  void  init() {
  try {
  FileWriter  fw= new FileWriter("abc.txt", true);
  PrintWriter pw = new PrintWriter(fw, true);
  pw.println("init  is invoked---");
  Date d= new Date();
  pw.println(d);
  pw.println();
  pw.close();
  fw.close();
 }catch(Exception e) {   }
}
public  void  destroy() {
  try {
  FileWriter  fw= new FileWriter("abc.txt", true);
  PrintWriter pw = new PrintWriter(fw, true);
  pw.println("destroy  is invoked---");
  Date d= new Date();
  pw.println(d);
  pw.println();
  pw.close();
  fw.close();
 }catch(Exception e) {   }
}};
```

The <servlet>of permanent servlet must contain another subtag called <Load-on-startup>.If we donot provide this tag then the servlet will not be started by the webserver during the deployment/hosting of the website. In our program we have given value as 1 within <load-on-startup>which indicates that it is the first permanent servlet which will execute during the hosting of the website. If we have another permanent servlet which we want to execute after "permserv" then the <load-on-startup>value must be 2.

If you pass the positive value in <load-on-startup>, the lower integer value servlet will be loaded before the higher integer value servlet. In other words, container loads the servlets in ascending integer value. The 0 value will be loaded first then 1, 2, 3 and so on. If you pass the negative value, servlet will be loaded at request time, at first request.

Servlet Context: This interface represents current runtime environment/web container/apache tomcat in a java program. The getServletContxt() of Servlet interface will return the runtime environment of the web server to the java program as a reference of ServletContext interface. The servlet container creates a ServletContext object that you can use to access information about the servlet's environment.

All servlets belong to one servlet context. The javax.servlet.ServletContext interface in the Java Servlet API is responsible for the state of its servlets and knows about resources and attributes available to the servlets in the context. Here we will only look at how ServletContext attributes can be used to share information among a group of servlets.

In addition the servlet engine may provide ways to configure a servlet context with initial attribute values. This serves as a welcome addition to the servlet to provide initialization arguments for configuration information used by a group of servlets, for instance a database identifier, a style sheet URL for an application, the name of a mail server, etc.

A servlet gets a reference to its ServletContext object through the getServletContext(). The HttpServlet class actually provides a convenience method (through its super class GenericServlet) named getServletContext to make it really easy:

There are three ServletContext methods dealing with context attributes:

- getAttribute(String appvar)
- setAttribute (String appvar, Object val)
- removeAttribute(String appvar)

A servlet also can bind an object attribute(2nd parameter) into the context by name(1st parameter) using setAttibute(). Any object bound into a context is available to any other servlet that is part of the same web application, to achieve this the ServletContxt Interface has given a method called setAttribute() that accepts two parameters, the first parameter must be a string which will be considered as an application scope variable name,the second parameter must be the value to the variable .Hence the setAttribute() is used to create application level variable .Similarly the getAttribute() will accept the application level variable name so that it will return the application level variable's value.

Example :

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class DemoServlet extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServletResponse res)throws ServletException,IOException {
        res.setContentType("text/html");
        PrintWriter pw=res.getWriter();
        ServletContext context=getServletContext();
        String driverName=context.getInitParameter("dname");
        pw.println("driver name is="+driverName);
        pw.close();
}}
```

Web.xml :

```
<web-app>
```

```
        <context-param>
                <param-name>dname</param-name>
                <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
        </context-param>

        <servlet>
                <servlet-name>test1</servlet-name>
                <servlet-class>ServletContextDemo</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>test1</servlet-name>
                <url-pattern>/context</url-pattern>
        </servlet-mapping>
</web-app>
```

The ServletContext object is created per application (i.e. only one) whereas ServletConfig object is created per servlet (i.e. if there are 4 servlets, there will be 4 ServletConfig objects). We may get the instance of ServletContext by calling the getServletContext() method of GenericServlet class where as getServletConfig() method of Servlet interface returns the instance of ServletConfig.

Type of variable : Depending upon the declaration of variable & their scope there are 4 types of variables in servlet/jsp as given below;

1- local scope
2- page scope
3- session scope
4- application scope

Local Scope: When a variable is declared within any function of a servlet, such as within service(), doGet(), doPost() etc then each servlet will get individual copy of these variable. Hence variables behave like data member for individual servlet.

Page Scope: When a variable is declared as a data member of a servlet class, then the scopeof the variable will remain to the entire servlet. All the servlet available in the web site will get same copy of the variable .Hence as a static variables of a class is shared by the object, simillarly all user of the web page will share this variable

Session Scope: When a variable is declared within a cookie/session then the scope of the variable will remain to the entire session(time spent by the user on a website). It is generally used to hold user's information .All the the servlet /jsp will get same copy of this variable per a user i.e this variable is available per user's basis.

Application Scope: The scope of this variable will remain throughout the website & it can be accessed by any no of web page by any no of user.Hence it behaves like a global variable for the website that will be accessed by any webpage by any user. Generally this variables are used to provide common values for all users.

Store it as AppVar1.java

```
import   java.io.*;
import   javax.servlet.*;
import   javax.servlet.http.*;
public  class AppVar1 extends HttpServlet{
  public void doGet(HttpServletRequest  req,  HttpServletResponse  res) throws IOException, ServletException {
ServletContext sc= getServletContext();
Integer x =(Integer) sc.getAttribute("hit");
if(x==null)
     sc.setAttribute("hit", 1);
else
   sc.setAttribute("hit", ++x);

PrintWriter pw= res.getWriter();
pw.println("<html><body bgcolor=yellow><h1>");
pw.println("<a href=http://localhost:8085/Yahoo/AppLev2>Click Me</a>");
pw.println("</h1></body></html>");
}};
```

Store it as AppVar2.java

```
import    java.io.*;
import    javax.servlet.*;
import    javax.servlet.http.*;
public  class AppVar2 extends HttpServlet{
  public void doGet(HttpServletRequest  req,  HttpServletResponse  res) throws IOException, ServletException {

ServletContext sc= getServletContext();
int  x =(Integer) sc.getAttribute("hit");

PrintWriter pw= res.getWriter();
pw.println("<html><body bgcolor=yellow><h1>");
pw.println("Ur user number : "+ x);
pw.println("</h1></body></html>");
}};
```

We can use the ServletContext interface's various methods to get the information you need. These methods include the following:

- **getMajorVersion.** This method returns an integer representing the major version for the servlet API that the servlet container supports. If the servlet container supports the servlet API version 2.3, this method will return 2.
- **getMinorVersion.** This method returns an integer representing the minor version of the servlet API that the servlet container supports. For the servlet API version 2.3, this method will return 3.
- **getAttributeNames.** This method returns an enumeration of strings representing the names of the attributes currently stored in the ServletContext.
- **getAttribute.** This method accepts a String containing the attribute name and returns the object bound to that name.
- **setAttribute.** This method stores an object in the ServletContext and binds the object to the given name. If the name already exists in the ServletContext, the old bound object will be replaced by the object passed to this method.
- **removeAttribute.** This method removes from the ServletContext the object bound to a name. The removeAttribute method accepts one argument: the name of the attribute to be removed.

Retrieving Servlet Context Information

```
import javax.servlet.*;
import java.util.Enumeration;
import java.io.IOException;
public class ContextDemoServlet implements Servlet {
ServletConfig  servletConfig;
public void init(ServletConfig config) throws ServletException {
servletConfig = config;
}
public void destroy() {
}
public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException {
ServletContext sct = servletConfig.getServletContext();
Enumeration attrib = sct.getAttributeNames();
while (attrib.hasMoreElements()) {
String atb = (String) attrib.nextElement();
System.out.println("Attribute name : " +  atb);
System.out.println("Attribute value : " +  sct.getAttribute(attribute));
}
System.out.println("Major version : " +  sct.getMajorVersion());
System.out.println("Minor version : " +  sct.getMinorVersion());
System.out.println("Server info : " +  sct.getServerInfo());
}
public String getServletInfo() {
return null;
}
public ServletConfig getServletConfig() {
return null;
}}
```

Forwarding & including request(Servlet chaining or Servlet Collaboration):

When a webpage will receive a request from the user (i.e from html,servlet/jsp)then it will process the request & provides the response to the user, but the user request will not be propagated to the next webpage. In case of request forwarding mechanism when the first webpage receive the client's request .then it will process the request or it may forward the client's request to the second webpage but the first webpage will not provide response to the client .when the second webpage will receive the user's request from the first webpage then it may process the request or it may forward the request to the third webpage and so on.The last webpage involved in this mechanism will provide the response to the client ,other webpages remain hidden to the client.This mechanism is also called as servlet chaining. This mechanism allows us to propagate the same request of the user to multiple webpages otherwise the user's request would always be available only to the first web page.

To achieve the forwarding request mechanism we have to get the reference of RequestDispatcher interface by the getRequestDispatcher() of ServletContext interface as given below:

```
ServletContext  sc = getServletContext();
RequestDispatcher  rd = sc.getRequestDispatcher(String url –of-another-webpage);
```

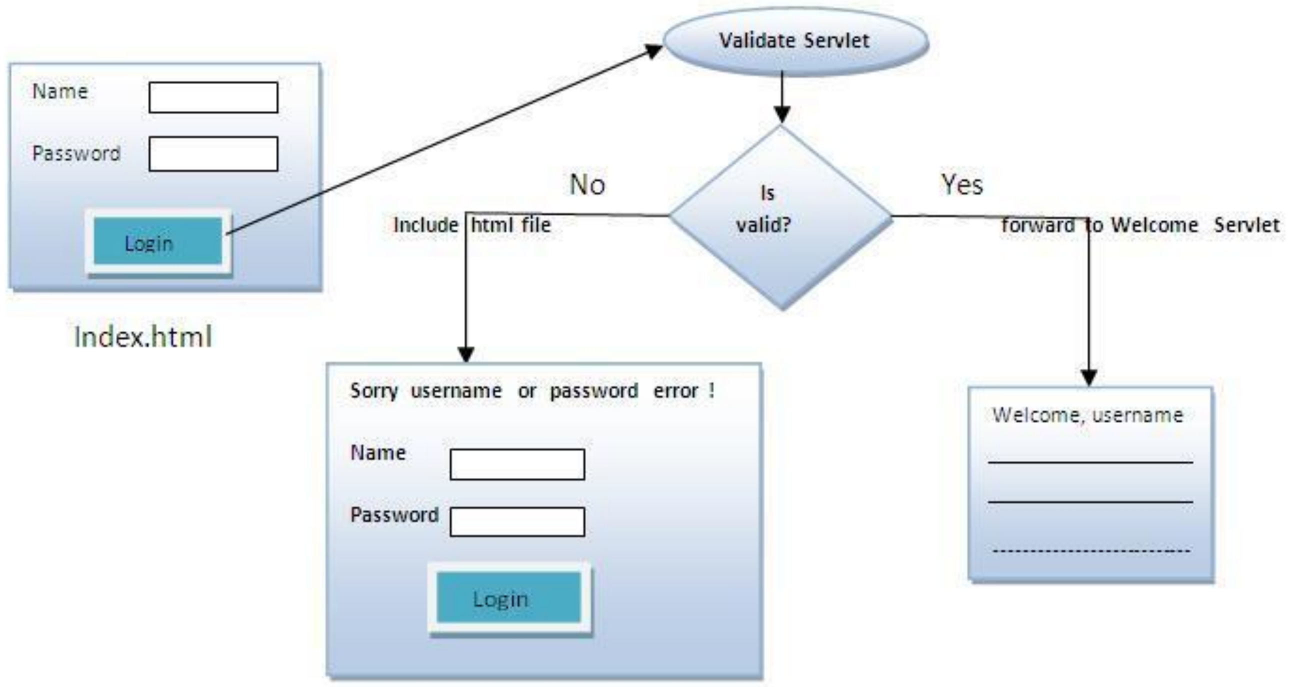
RequestDispatcher- The RequestDispacher interface provides the facility of dispatching/transferring the request from current web resource to another web resource it may be html, servlet or jsp. This interface can also be used to include the content of another resource also. It is one of the way of servlet collaboration.
There are two methods defined in the RequestDispatcher interface.

Methods

1. **public void forward(ServletRequest request,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(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:**Includes the content of a resource (servlet, JSP page, or HTML file) in the response.

The getRequestDispatcher() which will hold the url pattern of another servlet or the name of a jsp file to whom the request will be forwarded, and it returns a reference of RequestDispatcher interface.

Once we get refernce of RequestDispatcher interface then we can forward the user's request by using the forward method which will accept HttpServletRequest &HttpServletResponse interface as first & second parameter respectively. The RequestDispatcher interface has another method called include() that accepts HttpServletRequest & HttpServletResponse as the first and second parameter ,so that the content of another webpage can be included in the current webpage.



Stroe it as index.html :

```
<html><body bgcolor="orange"><h1>Sign In Here<form action="for1">
User  Name<input type=text  name=t1><br>
Enter  Password<input type=password  name=t2><br>
<input type="submit"></form></h1></body></html>
```

Store it as Forward1.java(url pattern is for1)

```
import    java.io.*;
import    java.sql.*;
import    javax.servlet.*;
import    javax.servlet.http.*;
public  class Forward1 extends HttpServlet{
  public void  doGet(HttpServletRequest  req,  HttpServletResponse  res) throws IOException, ServletException {
String s1= req.getParameter("t1");
String s2= req.getParameter("t2");
ServletContext  sc = getServletContext();
RequestDispatcher rd1= sc.getRequestDispatcher("/for2");
RequestDispatcher rd2= sc.getRequestDispatcher("/index.html");
PrintWriter pw= res.getWriter();
pw.println("<html><body bgcolor=yellow><h1>");
try{
        Class.forName("com.mysql.jdbc.Driver");
        Connection con = DriverManager.getConnection("jdbc:mysql://localhost/college", "root", "");
        Statement  st=con.createStatement();
        String  qry="select * from login where username='"+s1+"'  and pwd='"+s2+"'";
        ResultSet rs = st.executeQuery(qry);
        if(rs.next())
                rd1.forward(request, response);
        else{
                out.print("Sorry User Name or Password Error!");
                rd2.include(request, response);
        }
         con.close();
}catch(Exception e){  }
pw.println("</h1></body></html>");
}};
```

Store it as Forward2.java (url-pattern is for2)

```
import    java.io.*;
import    javax.servlet.*;
import    javax.servlet.http.*;
public  class  Forward2 extends HttpServlet{
  public void  doGet(HttpServletRequest  req,  HttpServletResponse  res) throws IOException, ServletException {
String s1= req.getParameter("t1");
String s2= req.getParameter("t2");
PrintWriter pw= res.getWriter();
pw.println("<html><body bgcolor=cyan><h1>");
pw.println("welcome "+s1 + " and it is home page");
pw.println("</h1></body></html>");
}};
```

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 :

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

public class DemoServlet extends HttpServlet{
        public void doGet(HttpServletRequest req,HttpServletResponse res)
throws ServletException,IOException  {
                res.setContentType("text/html");
                PrintWriter pw=res.getWriter();
                res.sendRedirect("http://www.google.com");

        }
}
```

Servlet Concurrency or Thread-Safe Servlets (Implementing SingleThreadModel interface) :

A servlet container allows multiple requests for the same servlet by creating a different thread to service each request. In many cases, each thread deals with its own ServletRequest and ServletResponse objects that are isolated from other threads. Problems start to arise, when two threads of the same servlet need to access an external resource, such as opening a file and writing to it.

```
import javax.servlet.*;
import java.io.*;
public class SingleThreadedServlet extends GenericServlet{
public void service(ServletRequest req, ServletResponse res)throws ServletException, IOException {
int counter = 0;
// get saved value
try {
BufferedReader reader = new BufferedReader(new FileReader("counter.txt"));
counter = Integer.parseInt( reader.readLine() );
reader.close();
}catch (Exception e) {   }
// increment counter
counter++;

// save new value
try {
BufferedWriter writer = new BufferedWriter(new FileWriter("counter.txt"));
writer.write(Integer.toString(counter));
writer.close();
}catch (Exception e) {   }

try {
PrintWriter out = response.getWriter();
out.println("You are visitor number " + counter);
}catch (Exception e) {  }
}}
```

This is exactly how you solve the problem in a servlet needing to service two users at the sametime: by making the second user wait until the first servlet finishes serving the first user. This solution makes the servlet single-threaded. This solution is very easy to do because of the marker SingleThreadedServlet interface. You don't need to change your code; you need only to implement the interface.

```
import javax.servlet.*;
import java.io.*;
public class SingleThreadedServlet extends GenericServlet implements SingleThreadModel {
public void service(ServletRequest req, ServletResponse res)throws ServletException, IOException {
  //same as above
}}
```
Now, if a user requests the service of the servlet while the servlet is servicing another user, the user who comes later will have to wait. We need to remember following points to make a thread safe servlet.

1. Your servlet service() method should not access any instance variables, unless these member variables are thread safe themselves.
2. Your servlet service() should not reassign instance variables, as this may affect other threads executing inside the service() method. If you really, really need to reassign a instance variable, make sure this is done inside a synchronized block.

Servlet Filter : A filter is an object that is invoked at the preprocessing and post processing of a request. It is mainly used to perform filtering tasks such as conversion, logging, compression, encryption and decryption, input validation etc.

The servlet filter is pluggable, i.e. its entry is defined in the web.xml file, if we remove the entry of filter from the web.xml file, filter will be removed automatically and we don't need to change the servlet.

Usage of Filter

- recording all incoming requests
- logs the IP addresses of the computers from which the requests originate
- conversion
- data compression
- encryption and decryption
- input validation etc.

Advantage of Filter

1. Filter is pluggable.
2. One filter don't have dependency onto another resource.
3. Less Maintenance

Filter API
Like servlet filter have its own API. The javax.servlet package contains the three interfaces of Filter API.
- Filter
- FilterChain
- FilterConfi**g**

Life cycle Methods of Filters :

Method	**Description**
public void init(FilterConfig config)	This method is invoked only once. It is used to initialize the filter.
public void doFilter(HttpServletRequest request,HttpServletResponse response, FilterChain chain)	This method is invoked every time when user request to any resource, to which the filter is mapped. It is used to perform filtering tasks. It is called each time a filter needs to perform any function. This method performs the actual work of a filter, either modifying the request or the response.
public void destroy()	This is invoked only once to perform any cleanup operation before the container removes a filter instance out of the service.

FilterChain interface

The object of FilterChain is responsible to invoke the next filter or resource in the chain.This object is passed in the doFilter method of Filter interface.The FilterChain interface contains only one method:

Filtertest.html:

```
<html><body bgcolor=asdf <form method="post" action="first">
   Name:<input type="text" name="t1" /><br/>
   Password:<input type="text" name="t2" /><br/>
   <input type="submit" value="submit" />
</form></body></html>
```

MyFilter.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class MyFilter implements Filter {
  public void init(FilterConfig fc) throws ServletException {}
  public void doFilter(ServletRequest request,
ServletResponse response,
         FilterChain chain) throws IOException,
ServletException {
      PrintWriter out = response.getWriter();
      response.setContentType("text/html");
      String usr = request.getParameter("t1");
      String pwd = request.getParameter("t2");
      if(usr.equals("gitesh") && pwd.equals("1234"))   {
         chain.doFilter(request, response);
      }else{
         out.println("You have enter a wrong password");
         RequestDispatcher rs =
request.getRequestDispatcher("filtertest.html");
         rs.include(request, response);
      }
  }
  public void destroy() { }
}
```

First.java

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

public class FirstServlet extends HttpServlet {
 protected void doPost(HttpServletRequest
request, HttpServletResponse response) throws
ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String user = request.getParameter("t1");
      out.println("Welcome "+user);
    }
}
```

web.xml

```
<web-app>
<servlet>
        <servlet-name>f1</servlet-name>
        <servlet-class>FirstServlet</servlet-class>
</servlet>
<servlet-mapping>
        <servlet-name>f1</servlet-name>
        <url-pattern>/first</url-pattern>
</servlet-mapping>

<filter>
        <filter-name>mf1</filter-name>
        <filter-class>MyFilter</filter-class>
</filter>
```

```
<filter-mapping>
        <filter-name>mf1</filter-name>
        <url-pattern>/first</url-pattern>
</filter-mapping>

</web-app>
```

How Filters Works?

1) When a request reaches the Web Container, it checks if any filter has URL patterns that matches the requested URL.
2) The Web Container locates the first filter with a matching URL pattern and filter's code is executed.
3) If another filter has a matching URL pattern, its code is then executed. This continues until there are no filters with matching URL patterns left.
4) If no error occurs, the request passes to the target servlet. Hence we know, that the request will be passed to the target servlet only when all the related Filters are successfully executed.
5) The servlet returns the response back to its caller. The last filter that was applied to the request is the first filter applied to the response.
6) At last the response will be passed to the Web Container which passes it to the client.

PrintWriter vs ServletOutputStream :

The response job is to send data to client. To achieve this the response uses IO Streams. There are two streams are involved for text data, the response uses PrintWriter and for binary data it uses ServletOutputStream. Two methods exist in ServletResponse interface (inherited by HttpServletResponse) has two methods :

PrintWriter **getWriter()** throws IOException: Returns a PrintWriter object that can send character text to the client. The PrintWriter uses the character encoding returned by getCharacterEncoding(). If the response's character encoding has not been specified as described in getCharacterEncoding (i.e., the method just returns the default value ISO-8859-1), getWriter updates it to ISO-8859-1.

```
PrintWriter out = response.getWriter();
out.println("VALID");                    // sending text data
out.println("<B>INVALID</B>");           // sending HTML data
```

ServletOutputStream **getOutputStream()** throws java.io.IOException: Returns a ServletOutputStream suitable for writing binary data in the response. The servlet container does not encode the binary data.

```
ServletOutputStream sos = resonse.getOutputStream();
sos.write(barray);
```

PROPERTY	PRINTWRITER	SERVLETOUTPUTSTREAM
CLASS NATURE	Concrete class	Abstract class
METHOD THAT RETURNS	getWriter()	getOutputStream()
NATURE OF DATA	To send character-based information like textual/HTML data	Mostly used to send binary data and less times to send primitive values and character-based information
CHARACTER ENCODING	Does character encoding for ASCII/Unicode characters	Does not perform any character conversion as it is meant for binary data

	(as per the getCharacterEncoding() method)	(container does not encode binary data).
PERFORMANCE	Very expensive for character conversion	Faster as no character conversion exists and particularly the entire data is sent as a byte array at a time
TYPE OF STREAM	Character stream	Byte stream
USAGE	Mostly used	Limited use.

Servlet Events & Listeners :

Why do we have Servlet Listener? : We know that using ServletContext, we can create an attribute with application scope that all other servlets can access but we can initialize ServletContext init parameters as String only in deployment descriptor (web.xml). What if our application is database oriented and we want to set an attribute in ServletContext for Database Connection. If you application has a single entry point (user login), then you can do it in the first servlet request but if we have multiple entry points then doing it everywhere will result in a lot of code redundancy. Also if database is down or not configured properly, we won't know until first client request comes to server. To handle these scenario, servlet API provides Listener interfaces that we can implement and configure to listen to an event and do certain operations.

Event is occurrence of something. Changing the state of an object is known as an event, in web application world an event can be initialization of application, destroying an application, request from client, creating/destroying a session, attribute modification in session etc.

Servlet API provides different types of Listener interfaces that we can implement and configure in web.xml to process something when a particular event occurs. For example, in above scenario we can create a Listener for the application startup event to read context init parameters and create a database connection and set it to context attribute for use by other resources, Counting total and current logged-in users, creating tables of the database at time of deploying the project, creating database connection object etc.

2. **Servlet Listener Interfaces and Event Objects**

Servlet API provides different kind of listeners for different types of Events. Listener interfaces declare methods to work with a group of similar events, for example we have ServletContext Listener to listen to startup and shutdown event of context. Every method in listener interface takes Event object as input. Event object works as a wrapper to provide specific object to the listeners.

Servlet API provides following event objects.

A. javax.servlet.AsyncEvent – Event that gets fired when the asynchronous operation initiated on a ServletRequest (via a call to ServletRequest#startAsync or ServletRequest#startAsync(ServletRequest, ServletResponse)) has completed, timed out, or produced an error.

B. javax.servlet.http.HttpSessionBindingEvent – Events of this type are either sent to an object that implements HttpSessionBindingListener when it is bound or unbound from a session, or to a HttpSessionAttributeListener that has been configured in the web.xml when any attribute is bound, unbound or replaced in a session.
The session binds the object by a call to HttpSession.setAttribute and unbinds the object by a call to HttpSession.removeAttribute.
We can use this event for cleanup activities when object is removed from session.

C. javax.servlet.http.HttpSessionEvent – This is the class representing event notifications for changes to sessions within a web application.
D. javax.servlet.ServletContextAttributeEvent – Event class for notifications about changes to the attributes of the ServletContext of a web application.
E. javax.servlet.ServletContextEvent – This is the event class for notifications about changes to the servlet context of a web application.
F. javax.servlet.ServletRequestEvent – Events of this kind indicate lifecycle events for a ServletRequest. The source of the event is the ServletContext of this web application.
G. javax.servlet.ServletRequestAttributeEvent – This is the event class for notifications of changes to the attributes of the servlet request in an application.

Servlet API provides following Listener interfaces.
H. javax.servlet.AsyncListener – Listener that will be notified in the event that an asynchronous operation initiated on a ServletRequest to which the listener had been added has completed, timed out, or resulted in an error.
I. javax.servlet.ServletContextListener – Interface for receiving notification events about ServletContext lifecycle changes.
J. javax.servlet.ServletContextAttributeListener – Interface for receiving notification events about ServletContext attribute changes.
K. javax.servlet.ServletRequestListener – Interface for receiving notification events about requests coming into and going out of scope of a web application.
L. javax.servlet.ServletRequestAttributeListener – Interface for receiving notification events about ServletRequest attribute changes.
M. javax.servlet.http.HttpSessionListener – Interface for receiving notification events about HttpSession lifecycle changes.
N. javax.servlet.http.HttpSessionBindingListener – Causes an object to be notified when it is bound to or unbound from a session.
O. javax.servlet.http.HttpSessionAttributeListener – Interface for receiving notification events about HttpSession attribute changes.
P. javax.servlet.http.HttpSessionActivationListener – Objects that are bound to a session may listen to container events notifying them that sessions will be passivated and that session will be activated. A container that migrates session between VMs or persists sessions is required to notify all attributes bound to sessions implementing HttpSessionActivationListener.