Java Servlet

Introduction to Website

Website is a collection of related web pages that may contain text, images, audio and video. The first page of a website is called home page. Each website has specific internet address (URL) that you need to enter in your browser to access a website.

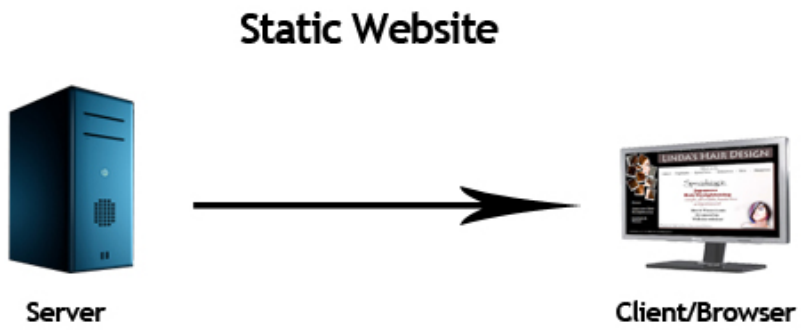
Website is hosted on one or more servers and can be accessed by visiting its homepage using a computer network. A website is managed by its owner that can be an individual, company or an organization.

Type of Website

a) Static website

Static website is the basic type of website that is easy to create. You don't need web programming and database design to create a static website. Its web pages are coded in HTML.

The codes are fixed for each page so the information contained in the page does not change and it looks like a printed page.



b) Dynamic website

Dynamic website is a collection of dynamic web pages whose content changes dynamically. It accesses content from a database or Content Management System (CMS). Therefore, when you alter or update the content of the database, the content of the website is also altered or updated.

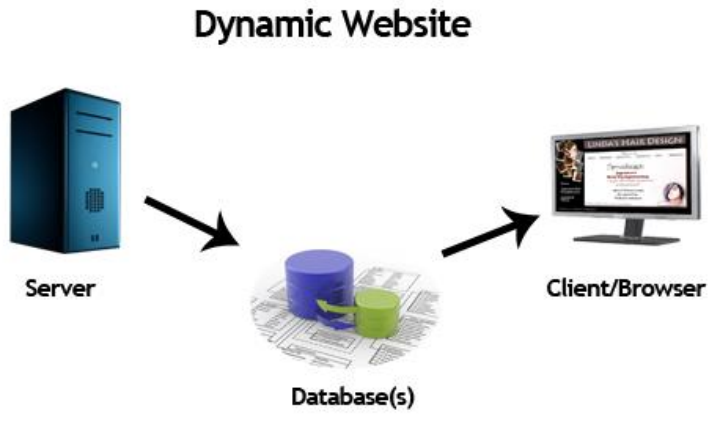
Dynamic website uses client-side scripting or server-side scripting, or both to generate dynamic content.

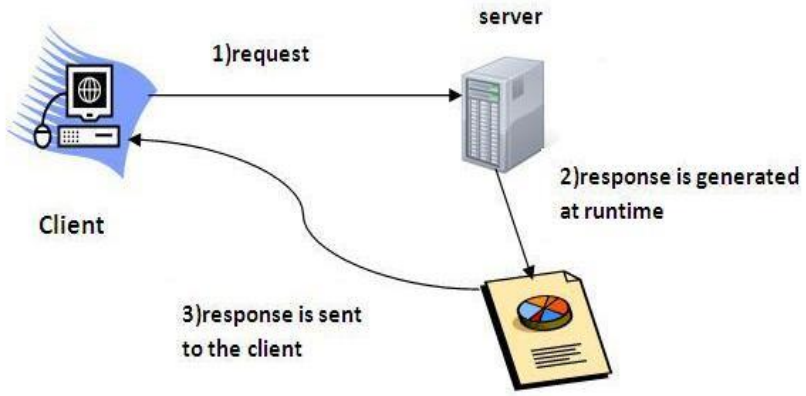
Client side scripting generates content at the client computer on the basis of user input. The web browser downloads the web page from the server and processes the code within the page to render information to the user.

In server side scripting, the software runs on the server and processing is completed in the server then plain pages are sent to the user.

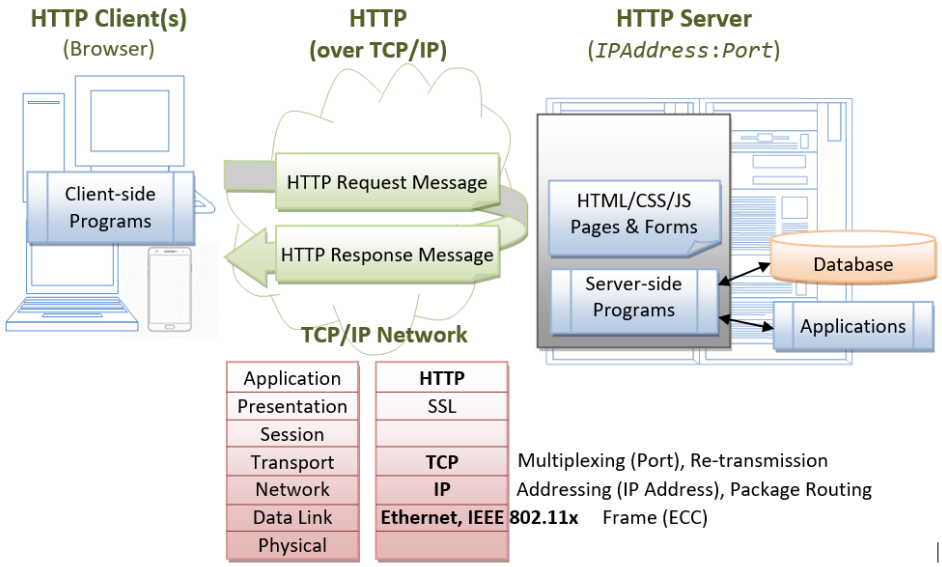
Static vs Dynamic website

Static Website	*Dynamic Website*
Prebuilt content is same every time the page is loaded.	Content is generated quickly and changes regularly.
It uses the HTML code for developing a website.	It uses the server side languages such as PHP, SERVLET, JSP, and ASP.NET etc. for developing a website.
It sends exactly the same response for every request.	It may generate different HTML for each of the request.
The content is only changes when someone publishes and updates the file (sends it to the web server).	The page contains "server-side" code it allows the server to generate the unique content when the page is loaded.
Flexibility is the main advantage of static website.	Content Management System (CMS) is the main advantage of dynamic website.





How Dynamic Web Application work:



Server side development in Java

Servlet technology is used to create web application in Java (resides at server side and generates dynamic web page).

Java Servlet

A **Servlet** is a Java class that responds to requests from web clients, often using HTTP. The most common type of servlet is an HTTP servlet, which handles HTTP requests and generates HTTP responses. Servlets run inside a servlet container (also known as a web container), which is part of a web server (e.g., Apache Tomcat, Jetty, or JBoss).

Servlets are a core part of the Java EE (Enterprise Edition) platform but can also be used in standalone applications or smaller web applications.

Servlet technology is robust and scalable because of java language.

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

Package for the Servlet interfaces and classes are :

javax.servlet.*;
javax.servlet.http.*;

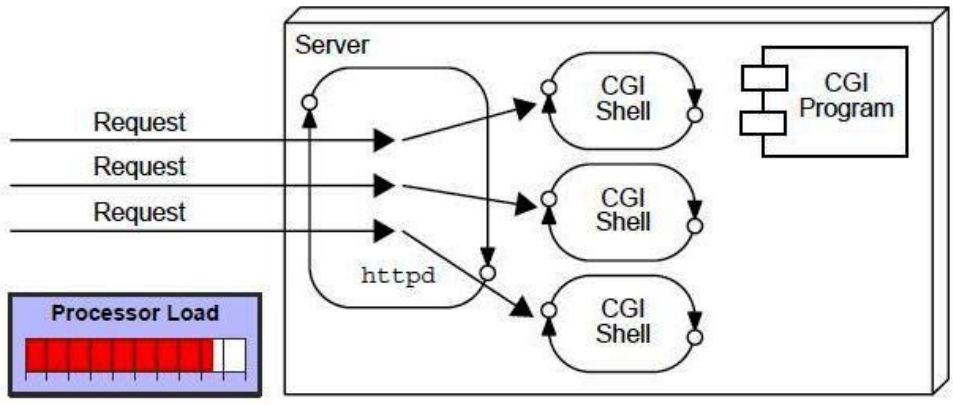
Servlet can be described in many ways, depending on the context.

- Servlet is a technology i.e. used to create web application.
- Servlet is an API that provides many interfaces and classes including documentations.
- Servlet is an interface that must be implemented for creating any servlet.
- Servlet is a class that extend the capabilities of the servers and respond to the incoming request. It can respond to any type of requests.
- Servlet is a web component that is deployed on the server to create dynamic web page.

Before Java –Old Concept:

CGI(Common Gateway Interface)

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



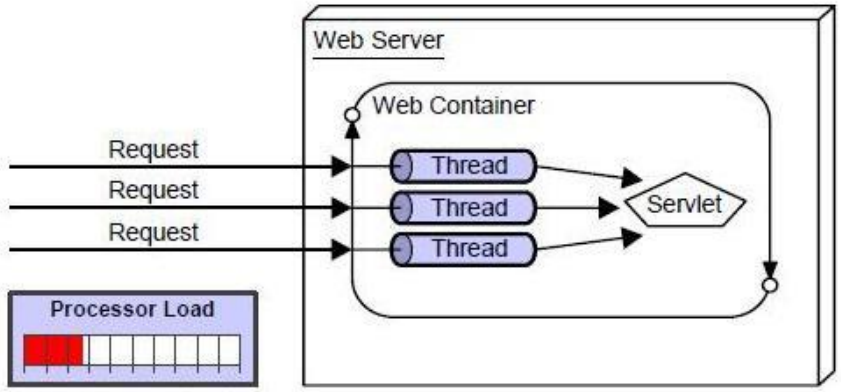
Disadvantages of CGI

There are many problems in CGI technology:

1. If number of client increases, it takes more time for sending response.

2. For each request, it starts a process and Web server is limited to start processes.
3. It uses platform dependent language e.g. C, C++, perl.

Advantage of Servlet



There are many advantages of servlet over old concept. The web container creates threads for handling the multiple requests to the servlet. Threads have a lot of benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The basic benefits of servlet are as follows:

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

Life Cycle of Servlet

Three methods are central to the life cycle of a servlet. These are init(), service() and destroy(). They are implemented by every servlet and invoked at specific time by server. Following is the life cycle of servlet:

i) User enters the URL to a web browser. The browser generate HTTP request for this URL. This request is then sent to the appropriate server.

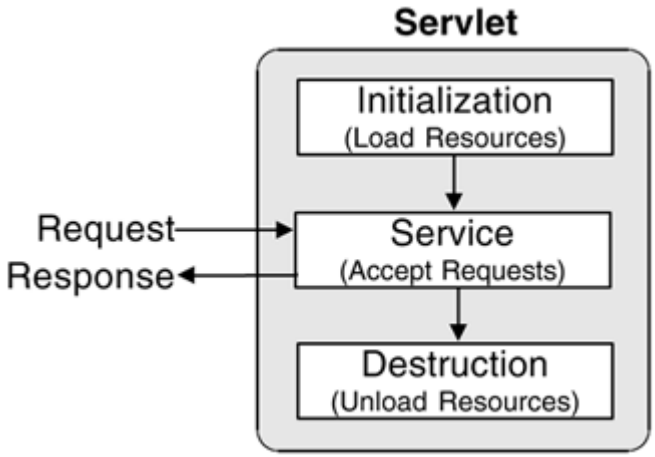
ii) HTTP request is received by the Web Server. The Server maps this request to a particular servlet. The servlet dynamically loaded into the address space of server.

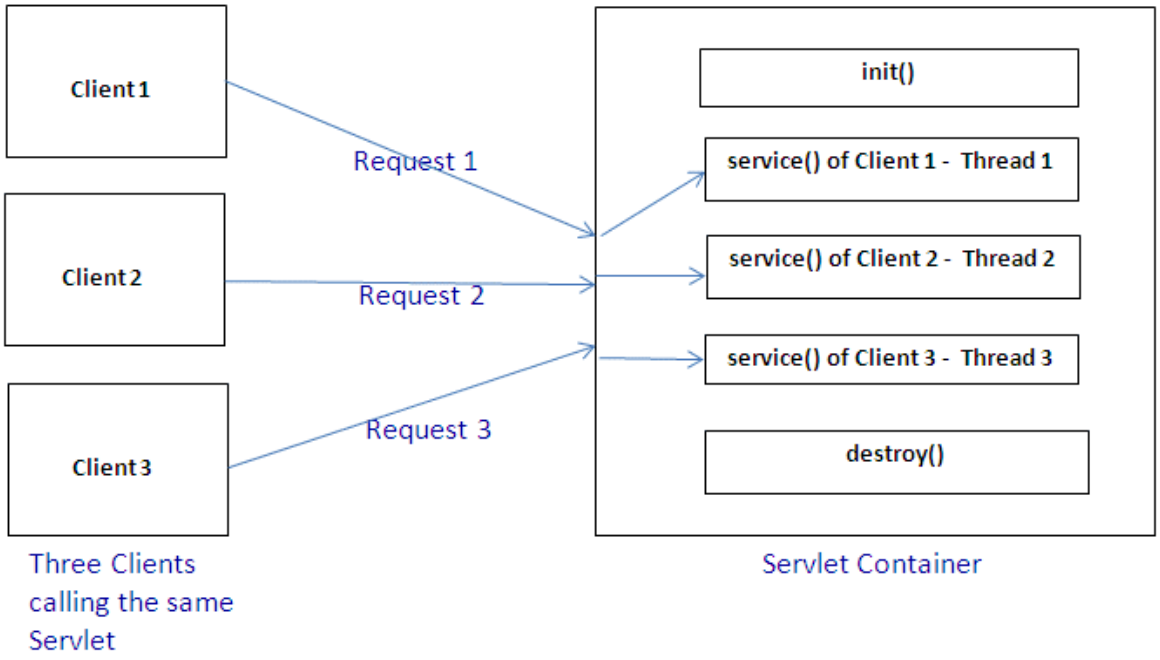
iii) The server invokes the **init()** method of the servlet. This method is invoked only when the servlet is first loaded into memory. It is possible to pass initialization parameters to the servlet so it may configure itself.

iv) The server invokes the **service()** method of the servlet. This method is called to process the HTTP request. It read data that has been provided in the HTTP request. It HTTP response for the client.

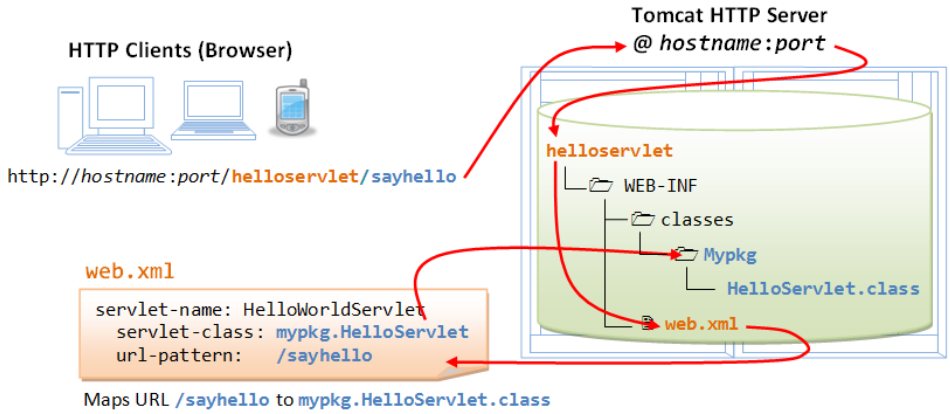
v) Finally, the server may decide to unload the servlet from its memory. The server calls the **destroy()** method to relinquish any resources such as file handles that are allocated for the servlet.

Servlet may remain in server's address space and is available to process any other HTTP requests received from client. The service() method is called for each HTTP request.





How Servlet Works



Server Development Environment

To create servlets, a servlet development environment is needed.

A number of Web Servers that support servlets are available in the market. Some web servers are freely downloadable for example: Apache Tomcat, Glassfish, JBoss, WildFly, XAMPP

Apache Tomcat

Apache Tomcat is an open source software implementation of the Java Servlet and Java Server Pages technologies and can act as a standalone server for testing servlets and can be integrated with the Apache Web Server.

Here are the steps to setup Tomcat on your machine –

Configuring Tomcat on Eclipse

i) Download from the Apache website.

ii) Unzip it and create a folder in the computer

iii) To configure in Eclipse Go to:

Windows-Preferences-Server-Runtime Environment-Add Server - Select the version you have downloaded – Select the folder where you unzip it – Apply

Servlet interfaces and classes

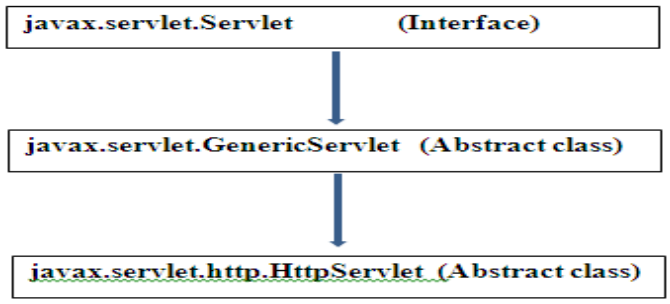
Name	Type	Pakage
Servlet	Interface	javax.servlet.*
GenericServlet	Abstract Class	javax.servlet.*
HttpServlet	Abstract Class	javax.servlet.http.*

Protocol Independent Servlet: If you are creating protocol independent Servlet you extend GenericServlet class.

Protocol Dependent Servlet: If you are creating protocol dependent servlet such as http servlet then you should extend HttpServlet class.

Servlet vs GenericServlet vs HttpServlet

Following figure shows the hierarchy of Servlet vs GenericServlet vs HttpServlet and to know from where HttpServlet comes.



Servlet is the super interface. Servlet interface contains 5 abstract methods. These abstract methods are :

i) init(), ii) service(), iii) getServiceConfig(), iv) getServletInfo(), v) destroy().

These 5 abstract methods are inherited by GenericServlet and HttpServlet.

Observe the hierarchy and understand the relationship between the three (involved in multilevel inheritance). With the observation, a conclusion can be arrived, to write a Servlet three ways exist.

a) by implementing Servlet (it is interface)
b) by extending GenericServlet (it is abstract class)
c) by extending HttpServlet (it is abstract class)

- **Servlet Interface** : Disadvantage of the Servlet Interface is, all the 5 abstract methods of the interface Servlet should be overridden even though programmer is not interested in all.
  These 5 methods are : i) init(), ii) service(), iii) getServiceConfig(), iv) getServletInfo(), v) destroy().

- **GenericServlet Class** : By Inheriting GenericServlet there is need to verriding its only one abstract method **service()**. It is enough to the programmer to override only this method. It is a callback method (called implicitly).

- HttpServlet Class : By extending HttpServlet, need not to override any methods as HttpServlet contains no abstract methods. Even though the HttpServlet does not contain any abstract methods, it is declared as abstract class by the designers to not to allow the programmer to create an object directly because a Servlet object is created by the system (here system is Servlet Container).

Servlet Interface implementation

Methods of Servlets

A Generic servlet contains the following five methods:

init()

public void init(ServletConfig config) throws ServletException

The init() method is called only once by the servlet container throughout the life of a servlet. By this init() method the servlet get to know that it has been placed into service.

The servlet cannot be put into the service if the init() method does not return within a fix time set by the web server. It throws a ServletException.

Parameters - The init() method takes a ServletConfig object that contains the initialization parameters and servlet's configuration.

service()

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

Once the servlet starts getting the requests, the service() method is called by the servlet

container to respond. The servlet services the client's request with the help of two objects. These two objects **javax.servlet.ServletRequest** and **javax.servlet.ServletResponse** are passed by the servlet container.

The status code of the response always should be set for a servlet that throws or sends an error.

Parameters - The service() method takes the **ServletRequest** object that contains the client's request and the object of **ServletResponse** contains the servlet's response. The service() method throws ServletException and IOExceptions exception.

getServletConfig()

public ServletConfig getServletConfig()

This method contains parameters for initialization and startup of the servlet and returns a ServletConfig object. This object is then passed to the init method. When this interface is implemented then it stores the ServletConfig object in order to return it. It is done by the generic class which implements this inetrface.

Returns - the ServletConfig object

getServletInfo()

public String getServletInfo()

The information about the servlet is returned by this method like version, author etc. This method returns a string which should be in the form of plain text and not any kind of markup.

Returns - a string that contains the information about the servlet

destroy()

public void destroy()

This method is called when we need to close the servlet. That is before removing a servlet instance from service, the servlet container calls the destroy() method. Once the servlet container calls the destroy() method, no service methods will be then called . That is after the exit of all the threads running in the servlet, the destroy() method is called. Hence, the servlet gets a chance to clean up all the resources like memory, threads etc which are being held.

Servlet Interface Example:

DemoServlet.java

```
import java.io.*;
import javax.servlet.*;
public class DemoServlet implements Servlet{
  ServletConfig config=null;

  public void init(ServletConfig config){
    this.config=config;
    System.out.println("Initialization complete");
  }

//ServletRequest : To accept the request
//ServletResponse : To send the response

  public void service(ServletRequest req,ServletResponse res)
  throws IOException,ServletException{
     res.setContentType("text/html");
     PrintWriter pwriter=res.getWriter();
     pwriter.print("<html>");
     pwriter.print("<body>");
     pwriter.print("<h1>Servlet Example Program</h1>");
     pwriter.print("</body>");
     pwriter.print("</html>");
  }
  public void destroy(){
     System.out.println("servlet life cycle finished");
  }
  public ServletConfig getServletConfig(){
     return config;
  }
  public String getServletInfo(){
     return "Servlet Demo";
  }
}
```

GenericServlet class

It is the immediate subclass of Servlet interface. In this class, only one abstract method service() exist. Other 4 abstract methods of Servlet interface are given implementation (given body). Anyone who extends this GenericServlet should override service() method. It was used by the Programmers when the Web was not standardized to HTTP protocol. It is protocol independent; it can be used with any protocol, say, SMTP, FTP, CGI including HTTP etc.

Signature:

public abstract class GenericServlet extends java.lang.Object implements Servlet, ServletConfig, java.io.Serializable

GenericServlet class Example:

DemoServlet.java

```
import java.io.*;
import javax.servlet.*;
public class DemoServlet extends GenericServlet{


//ServletRequest : To accept the request
//ServletResponse : To send the response

  public void service(ServletRequest req,ServletResponse res)
  throws IOException,ServletException{
     res.setContentType("text/html");
     PrintWriter pwriter=res.getWriter();
     pwriter.print("<html>");
     pwriter.print("<body>");
     pwriter.print("<h1>Servlet Example Program</h1>");
     pwriter.print("</body>");
     pwriter.print("</html>");
  }
}
```

HttpServlet (Creating Web Application in Java)

When HTTP protocol was developed by W3C people to suit more Web requirements, the Servlet designers introduced HttpServlet to suit more for HTTP protocol. HttpServlet is protocol dependent and used specific to HTTP protocol only.

The immediate super class of HttpServlet is GenericServlet. HttpServlet overrides the service() method of GenericServlet.

HttpServlet is abstract class but without any abstract methods.

With HttpServlet extension, service() method can be replaced by doGet() or doPost() with the same parameters of service() method.

Signature:

public abstract class HttpServlet extends GenericServlet implements java.io.Serializable

Being subclass of GenericServlet, the HttpServlet inherits all the properties (methods) of GenericServlet. So, if you extend HttpServlet, you can get the functionality of both.

HttpServlet Example :

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;     // Extend HttpServlet class
public class HelloWorld extends HttpServlet {

  private String message;


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

     // Set response content type
     response.setContentType("text/html");

     // Actual logic goes here.
     PrintWriter out = response.getWriter();
     out.println("<h1>" + message + "</h1>");
  }

  public void destroy() {
     // do nothing.
  }
}
```

doGet and doPost methods:

The doGet and doPost methods are called in response to an HTTP GET and an HTTP POST respectively which are submission methods used in an HTML FORM. On an HTTP GET the form data is part of the URL whereas on an HTTP POST the form data appears in the message body. You should always have one method call the other in your servlet as processing the form dat*a in a servlet is consistent regardless of the submission method.

Client/frontend :-

<form>

<input type = “text” name=”user” id=”us”/>

<input type = “text” name=”password” id=”pw”/>

</form>

For example:

Backend :-

```
protected void doPost(HttpServletRequest req, HttpServletResponse res) throws IOException
{
	String un = req.getParameter("user");
	String pw = req.getParameter("password");
	DataConnection dc = new DataConnection();
	dc.connect();


	try {
	ResultSet rs=dc.stmt.executeQuery("select * from login");
	PrintWriter writer = res.getWriter();

	while(rs.next())
	{
	if(un.compareTo(rs.getString(1))==0 && pw.compareTo(rs.getString(2))==0)
	{
		try {
			res.sendRedirect("Welcome.html");
			break;
		} catch (IOException e) {

		}
	}
	else
	{
		String htmlcode ="<html>";
		htmlcode+= "<body>";
		htmlcode+="<h1> You have entered wrong user name or password</h1>";
		htmlcode+="</body>";
		htmlcode+="</html>";
		writer.write(htmlcode);
		break;
	}
```

```
        }
        }catch(SQLException sq)
        {

        }
}
```

Difference between doGet and doPost methods:

DoGet
1)In doGet Method the parameters are appended to the URL and sent along with header information.

2)Maximum size of data that can be sent using doGet is 240 bytes.
3)Parameters are not encrypted.

4)DoGet is faster if we set the response content length since the same connection is used.

5) Increasing the performance

6)DoGet should be idempotent. i.e. doget should be able to be repeated safely many times
7)DoGet should be safe without any side effects for which user is held responsible

DoPost
1)In doPost, parameters are sent in separate line in the body.
2)There is no maximum size for data
3)Parameters are encrypted
4)Dopost is generally used to update or post some information to the server
5)DoPost is slower compared to doGet since doPost does not write the content length
6)This method does not need to be idempotent. Operations requested through POST can have side effects for which the user can be held accountable, for example, updating stored data or buying items online.

Request and Response in Servlet

HttpServletRequest interface :

- HttpServletRequest is an interface that provides methods to handle the HTTP request sent by the client (usually a browser) to the server.
- It allows access to request data like parameters, headers, cookies, and session information.

- It provides methods to read form data (e.g., getParameter()), access request headers (getHeader()), and interact with session attributes (getSession()).
- You can also use getMethod() to determine whether the request is a GET, POST, PUT, etc.
- It's commonly used in servlets to retrieve user input from the URL, query parameters, or form submissions.

HttpServletResponse interface:

- HttpServletResponse is an interface that provides methods to send the HTTP response from the server back to the client.
- It allows you to set the response's content type (e.g., HTML, JSON) with setContentType() and control HTTP status codes (e.g., setStatus()).
- You can use getWriter() to write response data, or sendRedirect() to redirect the client to a different URL.
- Methods like addHeader() let you add custom headers to the response, e.g., for setting cookies or caching directives.
- It's used in servlets to generate dynamic content and control how the response is sent back to the client.

  Syntax :

  protected void doPost(HttpServletRequest req, HttpServletResponse res) **throws** IOException

Deployment Descriptor

web.xml file deployment descriptor

A web application's deployment descriptor describes the classes, resources and configuration of the application and how the web server uses them to serve web requests. When the web server receives a request for the application, it uses the deployment descriptor to map the URL of the request to the code that ought to handle the request.

```
 <web-app>
<servlet>
<servlet-name>DemoServlet</servlet-name>
<servlet-class>DemoServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>DemoServlet</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>
</web-app>
```

Example 1 : Complete Login Page Example without Database

a) HTML Code for Login Page

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>

<form name="myForm" action="/TestServlet" method="get">
User Name: <input type="text" name=uname">
Password: <input type="text" name="password">
  <input type="submit" value="submit">
</form>
</body>
</html>
```

b) Servlet class to Process Login Page Request

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



// classes used in this code :
//1. HttpServlet, 2. HttpServletResponse, 3. HttpServletRequest

public class TestServlet extends HttpServlet {

        private static final long serialVersionUID = 1L;

        private int count;

        public void init()
        {
                count=0;

        }
        public void doGet(HttpServletRequest req, HttpServletResponse res)
                        throws ServletException, IOException
        {

                PrintWriter pw  =  res.getWriter();
                // Access the user name and password from the html file
                String name = req.getParameter("uname");
```

```
                String password = req.getParameter("password");

                if(name.compareTo("gvmitm")==0)
                        if(password.compareTo("abc")==0)
                        {
                pw.println("Welcome "+ name + "You are visitor no " + (++count));
                                res.sendRedirect("Welcome.html");

 //Welcome.html file will open if username and password is correct
                        }

                }
}
```

c) **Deployment descriptor (web.xml file)**

```
 <servlet>
 <servlet-name>TestServlet</servlet-name>
 <servlet-class>bcakend.TestServlet</servlet-class>
 </servlet>

 <servlet-mapping>
  <servlet-name>TestServlet</servlet-name>
  <url-pattern>/TestServlet</url-pattern>
 </servlet-mapping>
</web-app>
```

Example 2 : Complete Login Page Example using Database Connectivity

a) **HTML Code for Login Page**

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>

<form name="myForm" action="/TestServlet" method="get">
User Name: <input type="text" name=”nm">
Password: <input type="text" name="pw">
  <input type="submit" value="submit">
</form>
```

```
</body>
</html>
```

b) Servlet class to Process Login Page Request using database connectivity

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



// 1. HttpServlet, 2. HttpServletResponse, 3. HttpServletRequest

public class TestServlet extends HttpServlet {


        //private static final long serialVersionUID = 1L;
        Statement st;
        ResultSet rs;
        ConnectData cd;   //Data connectivity class

        public void doGet(HttpServletRequest req, HttpServletResponse res)
                        throws ServletException, IOException
        {

                cd = new ConnectData();
                cd.DataConnection();
                try {
                st  = cd.c.createStatement(); //statement object initialization
                }
                catch(SQLException sqe)
                {

                }

                //Handling of Request
                String name = req.getParameter("nm");
                String password = req.getParameter("pw");

                try
                 {
                rs = st.executeQuery("Select * From Login");

                while(rs.next())
                {
                if(name.compareTo(rs.getString(1))==0)
```

```
                    if(password.compareTo(rs.getString(2))==0)
                    {
                        res.sendRedirect("Welcome.html");
                        break;
                    }
                }
                }catch(SQLException sqe)
                {
                    System.out.println(sqe);
                }

                    }
}
```

c) Database Connectivity Class

```
import java.sql.*;

public class ConnectData {

    Connection c;
    void DataConnection()
    {
        //Driver Registration
        try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        }
        catch(ClassNotFoundException cnfe)
        {
            System.out.println(cnfe);
        }

        //Establishing the connection
        try
        {
c=DriverManager.getConnection("jdbc:mysql://localhost:3306/BCA2021","root","Abc@123
");

        }
        catch(SQLException se)
        {
            System.out.println(se);
        }

    }
```

```
}
```

d) Deployment Descriptor (web.xml file)

```
  <servlet>
  <servlet-name>TestServlet</servlet-name>
  <servlet-class>bcakend.TestServlet</servlet-class>
  </servlet>

  <servlet-mapping>
   <servlet-name>TestServlet</servlet-name>
   <url-pattern>/TestServlet</url-pattern>
  </servlet-mapping>
</web-app>
```