Client and Server Architecture:

The Client-server model is a distributed application structure that partitions task or workload between the providers of a resource or service, called servers, and service requesters called clients.

In the client-server architecture, when the client computer sends a request for data to the server through the internet, the server accepts the requested process and deliver the data packets requested back to the client.

Clients do not share any of their resources. Examples of Client-Server Model are Email, World Wide Web, etc.

Diagram

Description automatically generated

**How the Client-Server Model works ?**

* **Client:** When we talk the word **Client**, it mean to talk of a person or an organization using a particular service. Similarly in the digital world a **Client** is a computer (**Host**) i.e. capable of receiving information or using a particular service from the service providers (**Servers**).
* **Servers:** Similarly, when we talk the word **Servers**, It mean a person or medium that serves something. Similarly in this digital world a **Server** is a remote computer which provides information (data) or access to particular services.

So, its basically the **Client** requesting something and the **Server** serving it as long as its present in the database.

Diagram

Description automatically generated

### **Definition**

A [client server architecture](https://intellipaat.com/blog/what-is-client-server-architecture/) is a computing model wherein the server hosts, delivers, and manages most of the resources and services requested by the client.

It is also known as the networking computing model or client server network as all requests and services are delivered over a network. The client server architecture or model has another(other) system(s) connected over a network where resources are shared among the different computers.

Recognized as one of the most weirdly used networking models, the client server network model, generally, includes many clients or workstations interconnected in a network with a central server. While the server acts as a powerful system that stores all the data on it, the client functions as a machine that requests specific information from the server.

**How the browser interacts with the servers ?**  
There are few steps to follow to interacts with the servers a client.

* User enters the **URL**(Uniform Resource Locator) of the website or file. The Browser then requests the **DNS**(DOMAIN NAME SYSTEM) Server.
* **DNS Server** lookup for the address of the **WEB Server**.
* **DNS Server** responds with the **IP address** of the **WEB Server**.
* Browser sends over an **HTTP/HTTPS** request to **WEB Server’s IP** (provided by **DNS server**).
* Server sends over the necessary files of the website.
* Browser then renders the files and the website is displayed. This rendering is done with the help of **DOM** (Document Object Model) interpreter, **CSS** interpreter and **JS Engine** collectively known as the **JIT** or (Just in Time) Compilers.

Diagram

Description automatically generated

**Advantages of Client-Server model:**

* Centralized system with all data in a single place.
* Cost efficient requires less maintenance cost and Data recovery is possible.
* The capacity of the Client and Servers can be changed separately.

**Disadvantages of Client-Server model:**

* Clients are prone to viruses, Trojans and worms if present in the Server or uploaded into the Server.
* Server are prone to Denial of Service (DOS) attacks.
* Data packets may be spoofed or modified during transmission.
* Phishing or capturing login credentials or other useful information of the user are common and MITM(Man in the Middle) attacks are common.

**Client server architecture example**

Here are some of the client server model architecture examples from our daily life. Hope it helps you to understand the concept better.

* **Mail servers**: Email servers are used for sending and receiving emails. There are different software that allow email handling.
* **File servers**: File servers act as a centralized location for files. One of the daily life examples to understand this is the files that we store in Google Docs. The cloud services for Microsoft Office and Google Docs can be accessed from your devices; the files that you save from your computer, can be accessed from your phone. So, the centrally stored files can be accessed by multiple users.
* **Web servers**: [Web servers](https://intellipaat.com/blog/what-is-a-web-server/) are high-performance computers that host different websites. The server site data is requested by the client through high-speed internet.

**Components of client server architecture:**

Essentially, three components are required to make client server architecture work. The three components are workstations, servers, and networking devices. Let us, now, discuss them in detail:

* Workstations: Workstations are also called client computers. Workstations work as subordinates to servers and send them requests to access shared files and databases. A server requests information from the workstation and performs several functions as a central repository of files, programs, databases, and management policies. Workstations are governed by server-defined policies.
* Servers: Servers are defined as fast processing devices that act as centralized repositories of network files, programs, databases, and policies. Servers have huge storage space and robust memory to deal with multiple requests, approaching simultaneously from various workstations. Servers can perform many roles, such as mail server, database server, file server, and domain controller, in client server architecture at the same time.
* Networking devices: Now that we know about the roles that workstations and servers play, let us learn about what connects them, networking devices. Networking devices are a medium that connects workstations and servers in client server architecture. Many networking devices are used to perform various operations across the network. For example, a hub is used for connecting a server to various workstations . Repeaters are used to effectively transfer data between two devices. Bridges are used to isolate network segmentation.

## ****Types of client server architecture****

The functionality of client server architecture is in various tiers.

Diagram

Description automatically generated

In this category of client server architecture, the architecture contains all kinds of settings, such as configuration setting and marketing logic, on a single device.  While the diversity of services offered by 1-tier architecture makes it one of the reliable sources, handling such an architecture is difficult. This is primarily due to the data variance.  It often results in replication of work. 1-tier architecture consists of several layers, such as presentation layer, business layer, and data layer, that are combined with the help of a unique software package. The data present in this layer is usually stored in local systems or on a shared drive.

### **2-tier architecture**

Diagram

Description automatically generated

This architecture has the best environment. In this architecture, the user interface is stored on the client’s side and the database is stored on the server, while database logic and business logic is maintained either on the client’s side or on the server’s side.  
  
The 2-tier architecture is faster in comparison to the 1-tier architecture; this is because the 2-tier architecture does not have any intermediary between the client and the server. It is often utilized to avoid confusion between clients. One of the popular examples of 2-tier architecture is the online ticket reservation system.

### **3-tier architecture**

Diagram

Description automatically generated

Unlike 2-tier architecture that has no intermediary, in 3-tier client server architecture, a middleware lies between the client and the server. If the client places a request to fetch specific information from the server, the request will first be received by the middleware. It will then be dispatched to the server for further actions. The same pattern will be followed when the server sends a response to the client. The framework of 3-tier architecture is categorized into three main layers, presentation layer, application layer, and database tier.

All three layers are controlled at different ends. While the presentation layer is controlled at the client’s device, the middleware and the server handle the application layer and the database tier respectively. Due to the presence of a third layer that provides data control, 3-tier architecture is more secure, has invisible database structure, and provides data integrity.

Diagram

Description automatically generated

N-tier architecture is also called multi-tier architecture. It is the scaled form of the other three types of architecture. This architecture has a provision for locating each function as an isolated layer that includes presentation, application processing, and management of data functionalities.

## ****Advantages and disadvantages of client-server architecture****

The advantages and disadvantages of client-server architecture are mentioned below:

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| The centralized network has complete leverage to control the processes and activities. | If the primary server goes down, the entire architecture is disrupted. |
| All devices in the network can be controlled centrally. | It is expensive to operate because of the cost of heavy hardware and software tools. |
| Users have the authority to access any file, residing in the central storage, at any time. | This architecture requires particular OSs related to networking. |
| It provides a good user interface, easy file finding procedure, and management system for organizing files. | Too many users at once can cause the problem of traffic congestion. |
| Easy sharing of resources across various platforms is possible. | It requires highly technical stuff, such as server machines, for maintenance of the network. |

# Server side and Client side Programming

**Server-side Programming :**

It is the program that runs on server dealing with the generation of content of web page.  
1) Querying the database  
2) Operations over databases  
3) Access/Write a file on server.  
4) Interact with other servers.  
5) Structure web applications.  
6) Process user input. For example if user input is a text in search box, run a search algorithm on data stored on server and send the results.

**Examples :**  
The Programming languages for server-side programming are :  
1) PHP  
2) C++  
3) Java and JSP  
4) Python  
5) Ruby on Rails

**Client-side Programming :**

It is the program that runs on the client machine (browser) and deals with the user interface/display and any other processing that can happen on client machine like reading/writing cookies.

1) Interact with temporary storage  
2) Make interactive web pages  
3) Interact with local storage  
4) Sending request for data to server  
5) Send request to server  
6) work as an interface between server and user

The Programming languages for client-side programming are :  
1) Javascript  
2) VBScript  
3) HTML  
4) CSS  
5) AJAX

Servlets

## What are Servlets?

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

Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI.

* Performance is significantly better.
* Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
* Servlets are platform-independent because they are written in Java.
* Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.
* The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

## Servlets Architecture

Diagram

Description automatically generated

## Servlets Tasks

Servlets perform the following major tasks −

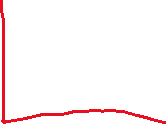
* Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
* Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
* Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
* Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
* Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

# Servlet API

There are two packages that you must remember while using API, the javax.servlet package that contains the classes to support generic servlet (protocol-independent servlet) and the javax.servlet.http package that contains classes to support http servlet.

Java.lang.Obiects

Javax.servlet.GenericServlet



Javax.servlet.http.httpServlet

## Generic Servlet

**service() method:**

**public abstract void service(ServletRequest req,ServletResponse Res) throws ServletException**

**{**

**}**

The request object tells the servlet about the request made by client while the response object is used to return a response back to the client.

Http Servlet

The request object tells the servlet about the request made by client while the response object is used to return a response back to the client.

* **doGet()** – This method is called by servlet service method to handle the HTTP GET request from client. The Get method is used for getting information from the server
* **doPost()** – Used for posting information to the Server
* **doPut()** – This method is similar to doPost method but unlike doPost method where we send information to the server, this method sends file to the server, this is similar to the FTP operation from client to server
* **doDelete()** – allows a client to delete a document, webpage or information from the server
* **init() and destroy()** – Used for managing resources that are held for the life of the servlet
* **getServletInfo()** – Returns information about the servlet, such as author, version, and copyright.

#### Interfaces in javax.servlet package

* Servlet
* ServletRequest
* ServletResponse
* ServletConfig
* ServletContext
* SingleThreadModel
* RequestDispatcher
* ServletRequestListener
* ServletRequestAttributeListener
* ServletContextListener
* ServletContextAttributeListener
* Filter
* FilterConfig
* FilterChain

#### Classes in javax.servlet package

* GenericServlet
* ServletInputStream
* ServletOutputStream
* ServletException
* ServletRequestWrapper
* ServletRequestEvent
* ServletResponseWrapper
* ServletContextEvent
* ServletRequestAttributeEvent
* ServletContextAttributeEvent
* UnavailableException

#### Interfaces in javax.servlet.http package

* HttpSession
* HttpServletRequest
* HttpServletResponse
* HttpSessionAttributeListener
* HttpSessionListener
* HttpSessionBindingListener
* HttpSessionActivationListener
* HttpSessionContext

#### Classes in javax.servlet.http package

* HttpServlet
* Cookie
* HttpSessionEvent
* HttpSessionBindingEvent
* HttpServletRequestWrapper
* HttpServletResponseWrapper
* HttpUtils

# Servlet Interface

1. By extending HttpServlet class  
   2) By extending GenericServlet class  
   3) By implementing Servlet interface\

### Servlet Interface methods

1. **void destroy()**:

**void destroy()**: This method is called by Servlet container at the end of [**servlet life cycle**](https://beginnersbook.com/2013/05/servlet-life-cycle/).

1. **void init(ServletConfig config)**: After this init() method gets called for each instantiated servlet, this method initializes the servlet.
2. **void service(ServletRequest req, ServletResponse res)**: This is the only method that is called multiple times during servlet life cycle. This methods serves the client request, it is called every time the server receives a request.
3. **ServletConfig getServletConfig()**: Returns a ServletConfig object, which contains initialization and startup parameters for this servlet.
4. **java.lang.String getServletInfo()**: Returns information about the servlet, such as author, version, and copyright.

## Life Cycle of Servlet

Servlet life cycle contains five steps:

1) Loading of Servlet

2) Creating instance of Servlet

3) Invoke init() once

4) Invoke service() repeatedly for each client request

5) Invoke destroy()

**Step 1: Loading of Servlet**  
When the web server (e.g. Apache Tomcat) starts up, the servlet container deploy and loads all the servlets.

**Step 2: Creating instance of Servlet**  
Once all the Servlet classes loaded, the servlet container creates instances of each servlet class. Servlet container creates only once instance per servlet class and all the requests to the servlet are executed on the same servlet instance.

**Step 3: Invoke init() method**  
Once all the servlet classes are instantiated, the init() method is invoked for each instantiated servlet. This method initializes the servlet. There are certain init parameters that you can specify in the deployment descriptor (web.xml) file. For example, if a servlet has value >=0 then its init() method is immediately invoked during web container startup.

**Step 4: Invoke service() method**  
Each time the web server receives a request for servlet, it spawns a new thread that calls service() method. If the servlet is GenericServlet then the request is served by the service() method itself, if the servlet is HttpServlet then service() method receives the request and dispatches it to the correct handler method based on the type of request.



**Step 5: Invoke destroy() method**  
When servlet container shuts down(this usually happens when we stop the web server), it unloads all the servlets and calls destroy() method for each initialized servlets.

Example:

GENERIC :

**package** com.genric;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** jakarta.servlet.GenericServlet;

**import** jakarta.servlet.ServletException;

**import** jakarta.servlet.ServletRequest;

**import** jakarta.servlet.ServletResponse;

**public** **class** ExampleGeneric **extends** GenericServlet{

@Override

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

// **TODO** Auto-generated method stub

res.setContentType("text/html");

PrintWriter pwriter=res.getWriter();

pwriter.print("<html>");

pwriter.print("<body>");

pwriter.print("<h1> welcome to GENERIC SERVLET</h1>");

pwriter.print("</body>"); pwriter.print("</html>");

}

}

WEB.xml

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd" id="WebApp\_ID" version="2.5">

<display-name>GenericServlet</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

<welcome-file>index.htm</welcome-file>

<welcome-file>index.jsp</welcome-file>

<welcome-file>default.html</welcome-file>

<welcome-file>default.htm</welcome-file>

<welcome-file>default.jsp</welcome-file>

</welcome-file-list>

<servlet>

<servlet-name>MyGenericServlet</servlet-name>

<servlet-class>com.genric.ExampleGeneric</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>MyGenericServlet</servlet-name>

<url-pattern>/welcome</url-pattern>

</servlet-mapping>

</web-app>

HTTPSERVLET

**package** com.http;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** jakarta.servlet.ServletException;

**import** jakarta.servlet.http.HttpServlet;

**import** jakarta.servlet.http.HttpServletRequest;

**import** jakarta.servlet.http.HttpServletResponse;

**public** **class** Exampleservlet **extends** HttpServlet {

/\*\*

\*

\*/

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** String mymsg;

**public** **void** init() **throws** ServletException

{

mymsg = "Http Servlet Demo";

}

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

{

res.setContentType("text/html");

PrintWriter out = res.getWriter();

out.println("<h1>" + mymsg + "</h1>");

out.println("<p>" + "Hello Friends!" + "</p>");

}

}

Web.XML

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd" id="WebApp\_ID" version="2.5">

<display-name>Httpserv</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

<welcome-file>index.htm</welcome-file>

<welcome-file>index.jsp</welcome-file>

<welcome-file>default.html</welcome-file>

<welcome-file>default.htm</welcome-file>

<welcome-file>default.jsp</welcome-file>

</welcome-file-list>

<servlet>

<servlet-name>MyHttpServlet</servlet-name>

<servlet-class>com.http.Exampleservlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>MyHttpServlet</servlet-name>

<url-pattern>/welcome</url-pattern>

</servlet-mapping>

</web-app>

package com.application;

import java.io.\*;

import jakarta.servlet.Servlet;

import jakarta.servlet.ServletConfig;

import jakarta.servlet.ServletException;

import jakarta.servlet.ServletRequest;

import jakarta.servlet.ServletResponse;

public class App implements Servlet {

ServletConfig config=null;

@Override

public void destroy() {

System.out.println("servlet life cycle finished");

}

@Override

public ServletConfig getServletConfig() {

// TODO Auto-generated method stub

return config;

}

@Override

public String getServletInfo() {

return "A Demo program written by karthika";

}

@Override

public void init(ServletConfig arg0) throws ServletException {

this.config=config;

System.out.println("Initialization complete");

}

@Override

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

res.setContentType("text/html");

PrintWriter pw=res.getWriter();

pw.print("<html>");

pw.print("<body>");

pw.print("<h1>welcome to servlet</h1>");

pw.print("</body>");

pw.print("</html>");

}

}

Web.XML

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd" id="WebApp\_ID" version="2.5">

<display-name>Myserv1</display-name>

<servlet>

<description></description>

<display-name>Ser1</display-name>

<servlet-name>Ser1</servlet-name>

<servlet-class>com.application.App</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>Ser1</servlet-name>

<url-pattern>/welcome</url-pattern>

</servlet-mapping>

</web-app>