Servlet Tutorials

Web Technology refers to **the various tools and techniques that are utilized in the process of communication between different types of devices over the internet**.

A web browser is used to access web pages. Web browsers can be defined as programs that display text, data, pictures, animation, and video on the Internet.

**HTTP** stands for **H**yper **T**ext **T**ransfer **P**rotocol.

**WWW** is about communication between web **clients** and **servers**

Communication between client computers and web servers is done by sending **HTTP Requests** and receiving **HTTP Responses**

The World Wide Web is a system of interlinked hypertext documents and programs that can be accessed via the Internet primarily by using HTTP.

Web – Page A document which can be displayed in a web browser such as Firefox, Google Chrome, Opera, Microsoft Internet Explorer or Edge, or Apple's Safari. These are also often called just "pages. ν WebSite A collection of web pages which are grouped together and usually connected together in various ways. Often called a "web site" or simply a "site.“ ν Web Server A special high end computer that hosts a website on the Internet. Today we have Cloud services that act as web servers.

HTTP Request / Response

Communication between clients and servers is done by **requests** and **responses**:

1. A client (a browser) sends an **HTTP request** to the web
2. A web server receives the request
3. The server runs an application to process the request
4. The server returns an **HTTP response** (output) to the browser
5. The client (the browser) receives the response

Computers connected to the Internet can communicate with one another with a number of protocols such as HTTP, SMTP (Simple Mail Transfer Protocol), FTP (File Transfer Protocol), IRC (Internet relay chat), IM (instant messaging), Telnet, and P2P (peer-to-peer).

What is a Servlet?

A servlet is a Java™ technology-based Web component, managed by a container, that generates dynamic content. Like other Java technology-based components, servlets are platform-independent Java classes that are compiled to platform-neutral byte code that can be loaded dynamically into and run by a Java technology-enabled Web server. Containers, sometimes called servlet engines, are Web server extensions that provide servlet functionality. Servlets interact with Web clients via a request/response paradigm implemented by the servlet container.

A servlet is simply a class which responds to a particular type of network request - most commonly an HTTP request. Basically servlets are usually used to implement web applications - but there are also various frameworks which operate on top of servlets (e.g. Struts) to give a higher-level abstraction than the "here's an HTTP request, write to this HTTP response" level which servlets provide.

Servlets run in a servlet container which handles the networking side (e.g. parsing an HTTP request, connection handling etc). **One of the best-known open source servlet containers is**[**Tomcat**](http://tomcat.apache.org/)**.**

**What is a Servlet?**

* A servlet is simply a class which responds to a particular type of network request - most commonly an HTTP request.
* Basically servlets are usually used to implement web applications - but there are also various frameworks which operate on top of servlets (e.g. Struts) to give a higher-level abstraction than the "here's an HTTP request, write to this HTTP response" level which servlets provide.
* Servlets run in a servlet container which handles the networking side (e.g. parsing an HTTP request, connection handling etc). One of the best-known open source servlet containers is Tomcat.
* In a request/response paradigm, a web server can serve only static pages to the client
* To serve dynamic pages, a we require Servlets.
* Servlet is nothing but a Java program
* This Java program doesn’t have a main method. It only has some callback methods.
* How does the web server communicate to the servlet? Via container or Servlet engine.
* Servlet lives and dies within a web container.
* Web container is responsible for invoking methods in a servlets. It knows what callback methods the Servlet has.

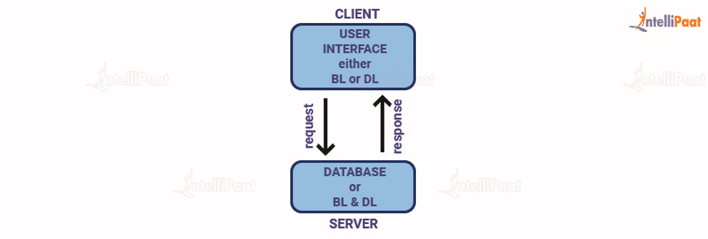
**Flow of Request**

* Client sends HTTP request to Web server
* Web server forwards that HTTP request to web container.
* Since Servlet can not understand HTTP, its a Java program, it only understands objects, so web container converts that request into valid request object
* Web container spins a thread for each request
* All the business logic goes inside doGet() or doPost() callback methods inside the servlets
* Servlet builds a Java response object and sends it to the container. It converts that to HTTP response again to send it to the client

What is a Servlet Container? The servlet container is a part of a Web server or application server that provides the network services over which requests and responses are sent, decodes MIME-based requests, and formats MIME-based responses. A servlet container also contains and manages servlets through their lifecycle. A servlet container can be built into a host Web server, or installed as an add-on component to a Web Server via that server’s native extension API. Servlet containers can also be built into or possibly installed into Web-enabled application servers. All servlet containers must support HTTP as a protocol for requests and responses, but additional request/response-based protocols such as HTTPS (HTTP over SSL) may be supported.

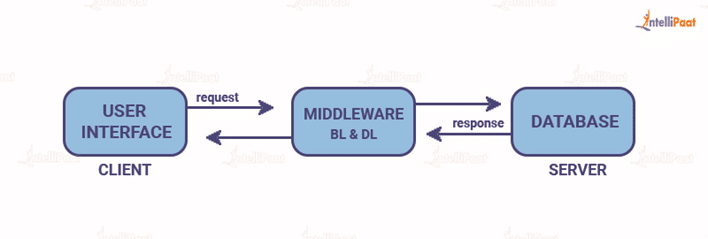
Client server architecture is **a computing model in which the server hosts, delivers, and manages most of the resources and services requested by the client**.

### **2-tier architecture**



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**



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.

Types of servers

The **proxy server** is a computer on the internet that accepts the incoming requests from the client and forwards those requests to the destination server. It works as a gateway between the end-user and the internet. It has its own IP address. It separates the client system and web server from the global network.

**Internet Relay Chat (IRC)** is an Internet application. IRC follows client-server model. It means that both client and server software are required in order to use it.

IRC client connects/communicates with IRC server on Internet. First, you have to log on to the server using a client and then pick the channel on which you want to chat. They are sent to your server when you type words on your keyboard. Now your server is part of global IRC server network. Your server sends your messages to other servers, which in turn, sends your messages to people who are part of your channel.

A [**groupware**](https://www.easytechjunkie.com/what-is-groupware.htm) server is a computer server utilized as a connection for various clients who use it to host and share files as part of a collaborative work environment. The number of clients connected to this server typically depends on the scope and nature of the project. Since this is part of a groupware project, there is typically software installed onto the various client computers to allow for better communication between the clients and access to the server.

**Creating servlet maven project**

<https://www.codejava.net/coding/java-servlet-and-jsp-hello-world-tutorial-with-eclipse-maven-and-apache-tomcat>

Tomcat is a [web server](http://en.wikipedia.org/wiki/Web_server) (can handle HTTP requests/responses) and [web container](http://en.wikipedia.org/wiki/Web_container) (implements [Java Servlet API](https://jcp.org/en/jsr/detail?id=340), also called servletcontainer) in one. Some may call it an [application server](http://en.wikipedia.org/wiki/Application_server), but it is definitely not an fullfledged Java EE application server (it does not implement [the whole Java EE API](http://docs.oracle.com/javaee/7/api/overview-summary.html)).

**Configuring Tomcat with Eclipse**

Download the latest version of eclipse IDE using[**this link**](https://www.eclipse.org/downloads/packages/release/2020-03/r/eclipse-ide-enterprise-java-developers-includes-incubating-components) and also configure the java environment. By default when Eclipse IDE is downloaded, it doesn’t come with Tomcat installed with it. Let us go over all detailed steps to configure Apache Tomcat in an Eclipse environment. **Step 1:** Download the latest version ( 9.0.34 ) of apache tomcat server from[this link](https://tomcat.apache.org/download-90.cgi) according to your platform.

<https://www.baeldung.com/java-web-app-without-web-xml>

The javax.\* package has been renamed to jakarta.\* package since Servlet API version 5.0 which is part of Jakarta EE 9 (Tomcat 10, TomEE 9, WildFly 22 Preview, GlassFish 6, Payara 6, Liberty 22, etc). So if you're targeting these server versions or newer, then you need to replace

import javax.servlet.\*;

import javax.servlet.http.\*;

by

import jakarta.servlet.\*;

import jakarta.servlet.http.\*;

JEE 6 shipped with Servlet 3.0 which enables us to use annotations for servlet definitions, minimizing the use of a web.xml file for a web application.

**For example, we can define a servlet and expose it with the @WebServlet annotation.**

@WebServlet(urlPatterns = "/uppercase", name = "uppercaseServlet") **public** **class** **UppercaseServlet** **extends** **HttpServlet** { **public** **void** **doGet**(HttpServletRequest request, HttpServletResponse response) **throws** IOException { **String** inputString = request.getParameter("input").toUpperCase(); **PrintWriter** out = response.getWriter(); out.println(inputString); } }

With the @WebServlet annotation, we're replacing the servlet and servlet-mapping sections from the web.xml file.

**Servlet Session Tracking**

**Session** simply means a particular interval of time.

**Session Tracking** is a way to maintain state (data) of an user. It is also known as **session management** in servlet.

We all know that **HTTP** is a stateless protocol. All requests and responses are independent. But sometimes you need to keep track of client's activity across multiple requests. For eg. When a User logs into your website, not matter on which web page he visits after logging in, his credentials will be with the server, until he logs out. So this is managed by creating a session.

The basic concept behind session is, whenever a user starts using our application, we can save a unique identification information about him, in an object which is available throughout the application, until its destroyed. So wherever the user goes, we will always have his information and we can always manage which user is doing what. Whenever a user wants to exit from your application, destroy the object with his information.

### **Why use Session Tracking?**

**To recognize the user** It is used to recognize the particular user.

### **Session Tracking Techniques**

There are four techniques used in Session tracking:

1. **Cookies**
2. **Hidden Form Field**
3. **URL Rewriting**
4. **HttpSession**

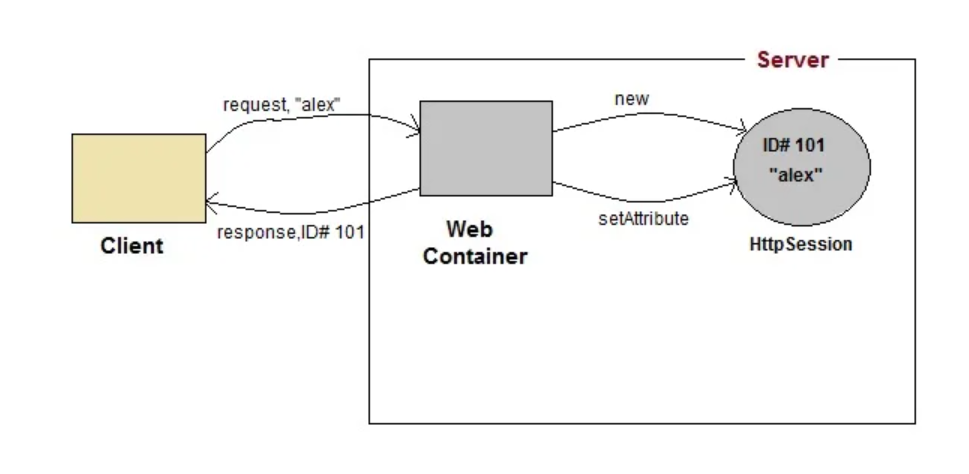
# **What is HttpSession?**

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.

## Servlet: How HttpSession works

1. On client's first request, the **Web Container** generates a unique session ID and gives it back to the client with response. This is a temporary session created by web container.
2. The client sends back the session ID with each request. Making it easier for the web container to identify where the request is coming from.
3. The **Web Container** uses this ID, finds the matching session with the ID and associates the session with the request.



### **How to get the HttpSession object ?**

The HttpServletRequest interface provides two methods to get the object of HttpSession:

1. **public HttpSession getSession():**Returns the current session associated with this request, or if the request does not have a session, creates one.
2. **public HttpSession getSession(boolean create):**Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

### **Commonly used methods of HttpSession interface**

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



# **Using Cookies for Session Management in Servlet**

https://www.javatpoint.com/cookies-in-servlet

**Cookies** are small pieces of information that are sent in response from the web server to the client. **Cookies** are the simplest technique used for storing client state.

**Cookies** are stored on client's computer. They have a lifespan and are destroyed by the client browser at the end of that lifespan.

A **cookie** is a small piece of information that is persisted between the multiple client requests.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

Using Cookies for storing client state has one shortcoming though, if the client has turned of COokie saving settings in his browser then, client state can never be saved because the browser will not allow the application to store cookies.

How cookies work?

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.



For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are:

1. **public void addCookie(Cookie ck):**method of HttpServletResponse interface is used to add cookie in response object.
2. **public Cookie[] getCookies():**method of HttpServletRequest interface is used to return all the cookies from the browser.

### **How to create Cookie?**

Let's see the simple code to create cookie.

1. Cookie ck=**new** Cookie("user","sonoo jaiswal");//creating cookie object
2. response.addCookie(ck);//adding cookie in the response

### **How to delete Cookie?**

Let's see the simple code to delete cookie. It is mainly used to logout or signout the user.

1. Cookie ck=**new** Cookie("user","");//deleting value of cookie
2. ck.setMaxAge(0);//changing the maximum age to 0 seconds
3. response.addCookie(ck);//adding cookie in the response

### **How to get Cookies?**

Let's see the simple code to get all the cookies.

1. Cookie ck[]=request.getCookies();
2. **for**(**int** i=0;i<ck.length;i++){
3. out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing name and value of cookie
4. }

**URL Rewriting**

**Listeners**

Events are basically occurrence of something. Changing the state of an object is known as an event.

We can perform some important tasks at the occurrence of these exceptions, such as counting total and current logged-in users, creating tables of the database at time of deploying the project, creating database connection object etc.

There are many Event classes and Listener interfaces in the javax.servlet and javax.servlet.http packages.

# **ServletContextEvent and ServletContextListener**

The ServletContextEvent is notified when web application is deployed on the server.

If you want to perform some action at the time of deploying the web application such as creating database connection, creating all the tables of the project etc, you need to implement ServletContextListener interface and provide the implementation of its methods.

### **Constructor of ServletContextEvent class**

There is only one constructor defined in the ServletContextEvent class. The web container creates the instance of ServletContextEvent after the ServletContext instance.

1. ServletContextEvent(ServletContext e)

### **Method of ServletContextEvent class**

1. **public ServletContext getServletContext()**: returns the instance of ServletContext.

### **Methods of ServletContextListener interface**

There are two methods declared in the ServletContextListener interface which must be implemented by the servlet programmer to perform some action such as creating database connection etc.

1. **public void contextInitialized(ServletContextEvent e)**: is invoked when application is deployed on the server.
2. **public void contextDestroyed(ServletContextEvent e)**: is invoked when application is undeployed from the server.

**Like previously, we can define a listener with the @WebListener annotation. – important**

