Servlets

Basics of Web

Web applications are software programs that run on web servers and are accessed via web browsers over a network, typically the Internet. They provide functionality through a user-friendly interface and can range from simple websites to complex applications like online banking systems, e-commerce platforms, and social media sites.

Key Components:

1. Client: The user interface, typically a web browser, that interacts with the web application.
2. Server: The backend that processes requests from the client, performs business logic, and serves responses.
3. Database: Stores data required by the web application.

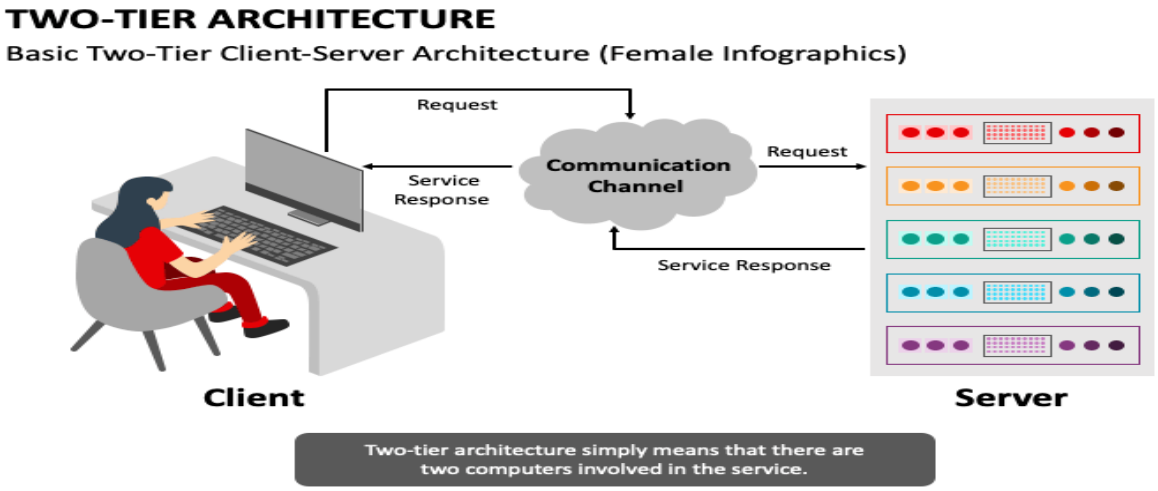
Client-Server Architecture

Client-server architecture is a computing model where clients (users) request services and resources from a centralized server.

Types of Client-Server Architecture:

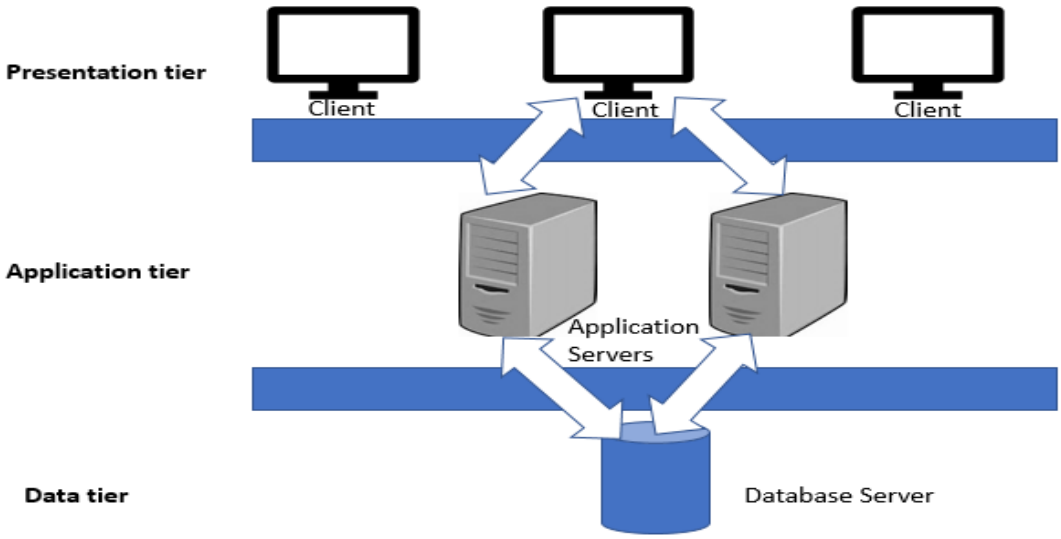
1. Two-Tier Architecture:

- **Client: Manages the user interface and sends requests to the server.**
- **Server: Handles requests, processes data, and returns responses.**
- **Example: A web browser (client) requesting a web page from a web server.**



2. Three-Tier Architecture:

- **Presentation Tier (Client): The user interface layer (web browser).**
- **Logic Tier (Application Server): Processes business logic and handles communication between the client and the database.**
- **Data Tier (Database Server): Manages the storage and retrieval of data.**
- **Example: An e-commerce website where the client interacts with the application server, which in turn interacts with the database server.**



3. N-Tier Architecture:

- **Extends the three-tier architecture by adding more layers (e.g., additional application servers) to improve scalability and manageability.**
- **Example: Large-scale enterprise applications with multiple application servers handling different types of business logic.**

HTTP Protocol

Hypertext Transfer Protocol (HTTP) **is an application-layer protocol for transmitting hypermedia documents, such as HTML. It was designed for communication between web browsers and web servers, but it can also be used for other purposes. HTTP follows a classical client-server model, with a client opening a connection to make a request, then waiting until it receives a response. HTTP is a stateless protocol, meaning that the server does not keep any data (state) between two requests.** https://developer.mozilla.org/en-US/docs/Web/HTTP

HTTP request methods

HTTP defines a set of *request methods* to indicate the desired action to be performed for a given resource. Although they can also be nouns, these request methods are sometimes referred to as *HTTP verbs*. Each of them implements a different semantic, but some common features are shared by a group of them: e.g. a request method can be safe, idempotent, or cacheable.

https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

HTTP response status codes

HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes:

1. **Informational responses (100 – 199)**
2. **Successful responses (200 – 299)**
3. **Redirection messages (300 – 399)**
4. **Client error responses (400 – 499)**
5. **Server error responses (500 – 599)**

https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

Web Server

A web server is a software system that serves web pages to users in response to their requests. These requests are usually received via HTTP (HyperText Transfer Protocol), and the web server responds by delivering the requested web page or resource. Some common web servers include:

Apache HTTP Server: An open-source web server that is widely used and highly configurable.

Nginx: Known for its high performance, stability, and low resource usage, often used as a reverse proxy and load balancer.

Microsoft Internet Information Services (IIS): A web server for Windows servers, providing a robust and secure environment for web hosting.

Web Server vs. Gateway Interface

Web Server:

A web server is a piece of software that handles HTTP requests from clients (typically web browsers) and serves them web content (like HTML pages, images, and other resources).

Examples: Apache, Nginx, IIS.

Gateway Interface:

A gateway interface is a specification that defines how web servers can communicate with web applications. It acts as a bridge between the server and the application, allowing them to interact and exchange data.

Examples: CGI, WSGI, ASGI.

Common Gateway Interface (CGI)

CGI stands for Common Gateway Interface, which is a standard protocol for web servers to execute programs that generate web content dynamically. These programs, often written in languages like Perl, Python, or C, can generate HTML or other data formats and send them to the client's browser. CGI scripts enable the creation of dynamic and interactive web applications, but they can be resource-intensive and slower compared to modern alternatives.

Asynchronous Server Gateway Interface (ASGI)

ASGI stands for Asynchronous Server Gateway Interface, which is designed to handle asynchronous web applications. It is intended to provide a standard interface between asynchronous Python web servers, frameworks, and applications, making it possible to handle long-lived connections like WebSockets and background tasks in a more efficient manner. ASGI is an evolution of WSGI (Web Server Gateway Interface) to support the async features introduced in Python 3.5+.

Web Server Gateway Interface (WSGI)

WSGI stands for Web Server Gateway Interface, which is a specification for a simple and universal interface between web servers and web applications or frameworks for Python. It was developed to provide a standard way to serve Python web applications and improve the compatibility between various Python web frameworks and servers. WSGI applications are synchronous, meaning they handle one request at a time, which can be a limitation for highly concurrent or real-time applications.

Differences between CGI, ASGI, and WSGI

CGI (Common Gateway Interface)

- **Description: CGI is a standard protocol for web servers to execute external programs, often scripts, that generate web content dynamically.**
- **Working: Each HTTP request spawns a new process to handle the execution of the CGI script.**
- **Performance: Typically slower and more resource-intensive due to the overhead of creating a new process for each request.**
- **Concurrency: Limited due to process creation overhead.**
- **Use Case: Simple web applications and scripts where performance and scalability are not critical.**

WSGI (Web Server Gateway Interface)

- Description: WSGI is a specification for a simple and universal interface between web servers and Python web applications or frameworks.
- Working: Designed for synchronous applications. The web server handles incoming requests and passes them to the WSGI application, which processes the request and returns a response.
- Performance: More efficient than CGI as it avoids the overhead of process creation.
- Concurrency: Handles one request per thread or process, suitable for many typical web applications.
- Use Case: Most Python web applications and frameworks (e.g., Django, Flask).

ASGI (Asynchronous Server Gateway Interface)

- Description: ASGI is a specification designed to handle asynchronous web applications, supporting both synchronous and asynchronous applications.
- Working: Designed to handle long-lived connections, such as WebSockets, and background tasks more efficiently.
- Performance: Better performance and scalability for asynchronous tasks and real-time applications.
- Concurrency: Supports high concurrency and can handle multiple connections concurrently.
- Use Case: Applications requiring high concurrency, real-time updates, WebSockets (e.g., FastAPI, Django Channels).

Types of Web Servers:

1. Apache HTTP Server

- Supporting Programming Languages:

- PHP
- Perl
- Python (via mod_wsgi)
- Ruby
- Java (via Tomcat)
- C/C++ (via CGI and FastCGI)

2. Nginx

- Supporting Programming Languages:

- PHP (via PHP-FPM)
- Python (via uWSGI, Gunicorn)

- **Ruby (via Passenger, Puma)**
- **Node.js**
- **Java (via Tomcat or Jetty)**

3. Microsoft Internet Information Services (IIS)

- Supporting Programming Languages:

- **ASP.NET (C#, VB.NET)**
- **PHP**
- **Python (via IronPython or CGI)**
- **Node.js (via iisnode)**

4. Lighttpd

- Supporting Programming Languages:

- **PHP (via FastCGI)**
- **Python (via FastCGI)**
- **Perl (via FastCGI)**
- **Ruby (via FastCGI)**

5. LiteSpeed

- Supporting Programming Languages:

- **PHP**
- **Python (via LSAPI or FastCGI)**
- **Ruby (via LSAPI or FastCGI)**
- **Perl (via LSAPI or FastCGI)**
- **Java (via Servlet Engine)**

6. Caddy

- Supporting Programming Languages:

- **Go (native support)**
- **PHP (via FastCGI)**
- **Python (via FastCGI)**
- **Ruby (via FastCGI)**
- **Node.js (via reverse proxy)**

7. Tomcat (Apache Tomcat)

- Supporting Programming Languages:

- **Java**
- **JSP (JavaServer Pages)**

- Servlets

8. Jetty

- Supporting Programming Languages:

- Java
- JSP (JavaServer Pages)
- Servlets
- Kotlin (via JVM)

9. Node.js

- Supporting Programming Languages:

- JavaScript
- TypeScript (with transpilers)

10. Unicorn (uWSGI for Python)

- Supporting Programming Languages:

- Python
- Ruby (Gunicorn)
- Perl (uWSGI)
- PHP (uWSGI)

Each web server has its own strengths and is optimized for different types of workloads and environments. The choice of web server and supporting programming languages often depends on the specific requirements of the project, such as performance, scalability, and the development team's familiarity with the technologies.

https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/What_is_a_web_server

Web Services

Web services are software systems designed to allow for interoperable communication and interaction over a network, typically the internet. They facilitate the exchange of data and functionalities between different applications or systems, making it possible for them to work together. Web services follow specific communication protocols and use standard formats like XML or JSON for data exchange.

There are several types of web services, with two fundamental categories being:

1. SOAP (Simple Object Access Protocol) Web Services:

- SOAP is a protocol for exchanging structured information in web services.

- **It uses XML for message formatting and relies on other protocols like HTTP and SMTP for message negotiation and transmission.**
- **It defines a set of rules for structuring messages, including headers and bodies.**

2. RESTful Web Services (Representational State Transfer):

- **REST, or Representational State Transfer,**
- **is an architectural style for designing networked applications.**
- **A RESTful API (Application Programming Interface) is a set of rules and conventions for building and interacting with web services.**
- **It typically uses standard HTTP methods like GET, POST, PUT, DELETE to perform operations on resources, and it relies on stateless communication, meaning each request from a client contains all the information needed to understand and fulfill that request.**
- **JSON is commonly used for data interchange in REST APIs.**
- **It often utilizes JSON or XML for data representation.**

Eclipse Jakarta Project

The Eclipse Jakarta Project, also known as Jakarta EE, is an open-source project that is managed by the Eclipse Foundation. It is the successor to Java EE (Java Platform, Enterprise Edition) and provides a set of specifications for developing enterprise-level applications in Java.

1. History and Evolution:

- **Originally developed by Sun Microsystems as Java EE.**
- **Acquired by Oracle, which later donated it to the Eclipse Foundation.**
- **Renamed from Java EE to Jakarta EE to reflect its new governance under the Eclipse Foundation.**

2. Goals:

- **To provide a set of stable, scalable, and enterprise-level specifications for Java.**
- **To promote the adoption of cloud-native and microservices architectures.**
- **To ensure compatibility and interoperability between implementations.**

3. Specifications:

Jakarta EE includes a wide range of specifications, such as:

- **Jakarta Servlet: For creating web applications.**
- **Jakarta Persistence (JPA): For managing relational data.**
- **Jakarta Contexts and Dependency Injection (CDI): For managing dependencies.**
- **Jakarta RESTful Web Services (JAX-RS): For building RESTful web services.**
- **Jakarta Faces (JSF): For building component-based user interfaces.**
- **Jakarta Transactions (JTA): For managing transactions.**

- **Jakarta Messaging (JMS): For sending messages between applications.**

4. Governance and Community:

- **Managed by the Eclipse Foundation, which ensures open governance, vendor neutrality, and community-driven development.**
- **Supported by a large community of developers, organizations, and vendors.**

5. Implementations:

- **Several Jakarta EE-compatible application servers and frameworks are available, such as Eclipse GlassFish, Red Hat's WildFly, IBM's Open Liberty, and Payara Server.**

Benefits of Using Jakarta EE:

- **Standardization: Provides a consistent set of APIs and standards for building enterprise applications.**
- **Portability: Applications built using Jakarta EE can be easily moved between different application servers.**
- **Scalability: Designed to handle large-scale, distributed, and transactional applications.**
- **Productivity: Rich set of tools and libraries that help developers build robust applications quickly.**

Explore Documentation and Tutorials:

- **Official Jakarta EE documentation: Jakarta EE Specifications | The Eclipse Foundation**
- **Tutorials and examples:**
    1) **Overview :: Jakarta EE Documentation**
    2) **The Jakarta® EE Tutorial (eclipse-ee4j.github.io)**

By understanding and leveraging the Eclipse Jakarta Project, you can build powerful, scalable, and maintainable enterprise applications in Java.

Tomcat Web Server

https://tomcat.apache.org/

https://tomcat.apache.org/tomcat-10.1-doc/index.html

https://dlcdn.apache.org/tomcat/tomcat-10/v10.1.25/bin/apache-tomcat-10.1.25-windows-x64.zip

Apache Tomcat is an open-source implementation of the Java Servlet, JavaServer Pages (JSP), and WebSocket technologies. It is developed and maintained by the Apache Software Foundation. Tomcat provides a "pure Java" HTTP web server environment for Java code to run in.

Apache Tomcat version 10.1 implements the Servlet 6.0 and Pages 3.1 specifications from Jakarta EE, and includes many additional features that make it a useful platform for developing and deploying web applications and web services.

Key Features of Apache Tomcat

1. Java Servlet Support: Tomcat supports Java Servlets, which are Java programs that run on the server and handle client requests and responses.

2. JavaServer Pages (JSP) Support: JSP allows developers to create dynamically generated web pages based on HTML, XML, or other document types.

3. WebSocket Support: Tomcat supports WebSocket, enabling two-way communication between the client and server for real-time applications.

4. Java Expression Language (EL) Support: EL allows the easy integration of Java code in JSP pages.

5. Extensibility and Integration: Tomcat can be extended and integrated with other technologies and frameworks, such as Spring, Hibernate, and more.

Why Use Apache Tomcat?

1. Open Source and Free: Tomcat is open-source software, which means it is free to use, modify, and distribute. This makes it an attractive option for developers and organizations looking for a cost-effective solution.

2. Java EE Compatibility: Tomcat supports key specifications of the Java Enterprise Edition (Java EE) platform, including Servlets and JSP. This makes it a popular choice for Java-based web applications.

3. Lightweight and Efficient: Tomcat is relatively lightweight compared to full-fledged Java EE application servers like JBoss or WebSphere. This makes it suitable for applications that do not require the full Java EE stack.

4. Scalability: Tomcat can handle a large number of concurrent requests and can be scaled horizontally by deploying multiple instances behind a load balancer.

5. Wide Adoption and Community Support: Tomcat has a large user base and a robust community. This means plenty of resources, tutorials, and community support are available.

6. Integration with Development Tools: Tomcat integrates well with popular development tools and environments such as Eclipse, IntelliJ IDEA, and NetBeans, making the development process smoother.

Common Use Cases for Apache Tomcat

1. Web Application Hosting: Tomcat is commonly used to host Java-based web applications, ranging from small websites to large-scale enterprise applications.

2. Development and Testing: Developers use Tomcat for developing and testing Java Servlets and JSPs due to its ease of use and configuration.

3. Microservices: Tomcat's lightweight nature makes it suitable for deploying microservices, especially when combined with frameworks like Spring Boot.

4. RESTful Services: Tomcat can be used to deploy RESTful web services, which are commonly used in modern web and mobile applications.

Apache Tomcat 10.1 is an open-source implementation of the Jakarta Servlet, Jakarta Server Pages (JSP), and WebSocket technologies, developed and maintained by the Apache Software Foundation. Tomcat 10.1 provides a "pure Java" HTTP web server environment for Java code to run in, adhering to the latest Jakarta EE standards.

Key Features of Tomcat 10.1

1. Jakarta Servlet 6.0: Supports the latest features and updates in the Jakarta Servlet API, which is used to create dynamic web applications.

2. Jakarta Server Pages (JSP) 3.1: Supports the latest JSP technology, allowing the creation of dynamically generated web pages based on HTML, XML, or other document types.

3. Jakarta Expression Language (EL) 5.0: Provides a way to easily access application data stored in JavaBeans components.

4. WebSocket 2.1: Supports WebSocket technology for creating bi-directional communication channels over a single, long-lived connection, useful for real-time applications.

5. JSP Tag Library API 3.1: Supports the use of custom tags in JSP pages, which can encapsulate complex server-side logic into simple, reusable tags.

New Features and Improvements in Tomcat 10.1

1. Jakarta EE 9+ Compliance: Tomcat 10.1 is fully compliant with Jakarta EE 9 and beyond, which involves a package namespace change from `javax` to `jakarta`.

2. Enhanced Security: Improved security features to protect web applications against various types of attacks and vulnerabilities.

3. Performance Improvements: Optimizations to enhance the performance and efficiency of the server, particularly under high load conditions.

4. Modernized Codebase: Updates and modernizations to the codebase to ensure compatibility with the latest Java versions and standards.

5. Support for New HTTP/2 Features: Enhanced support for HTTP/2, which includes multiplexing of streams, header compression, and server push capabilities.

Deployment and Use Cases

1. Web Application Hosting: Tomcat 10.1 is widely used to host web applications written in Java. It is particularly popular for enterprise-level applications.

2. Development and Testing: Developers use Tomcat to test Java Servlets and JSPs during the development process due to its simplicity and ease of configuration.

3. Microservices and Cloud Deployments: Tomcat is lightweight and can be easily deployed in cloud environments and microservices architectures.

Configuration and Administration

- Server Configuration: Configured through XML configuration files such as `server.xml` and `web.xml`.

- Deployment: Web applications are typically packaged as WAR (Web Application Archive) files and deployed to the `webapps` directory.

- Management: Comes with a web-based administration interface for managing deployed applications, configuring server settings, and monitoring server status.

Servlet Basics

Jakarta Servlet :: Jakarta EE Documentation

https://jakarta.ee/specifications/servlet/5.0/jakarta-servlet-spec-5.0#what-is-a-servlet

17 Java Servlet Technology (Release 7) (oracle.com)

A servlet is a Jakarta technology-based web component, managed by a container, that generates dynamic content. Like other Jakarta 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 Jakarta 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.

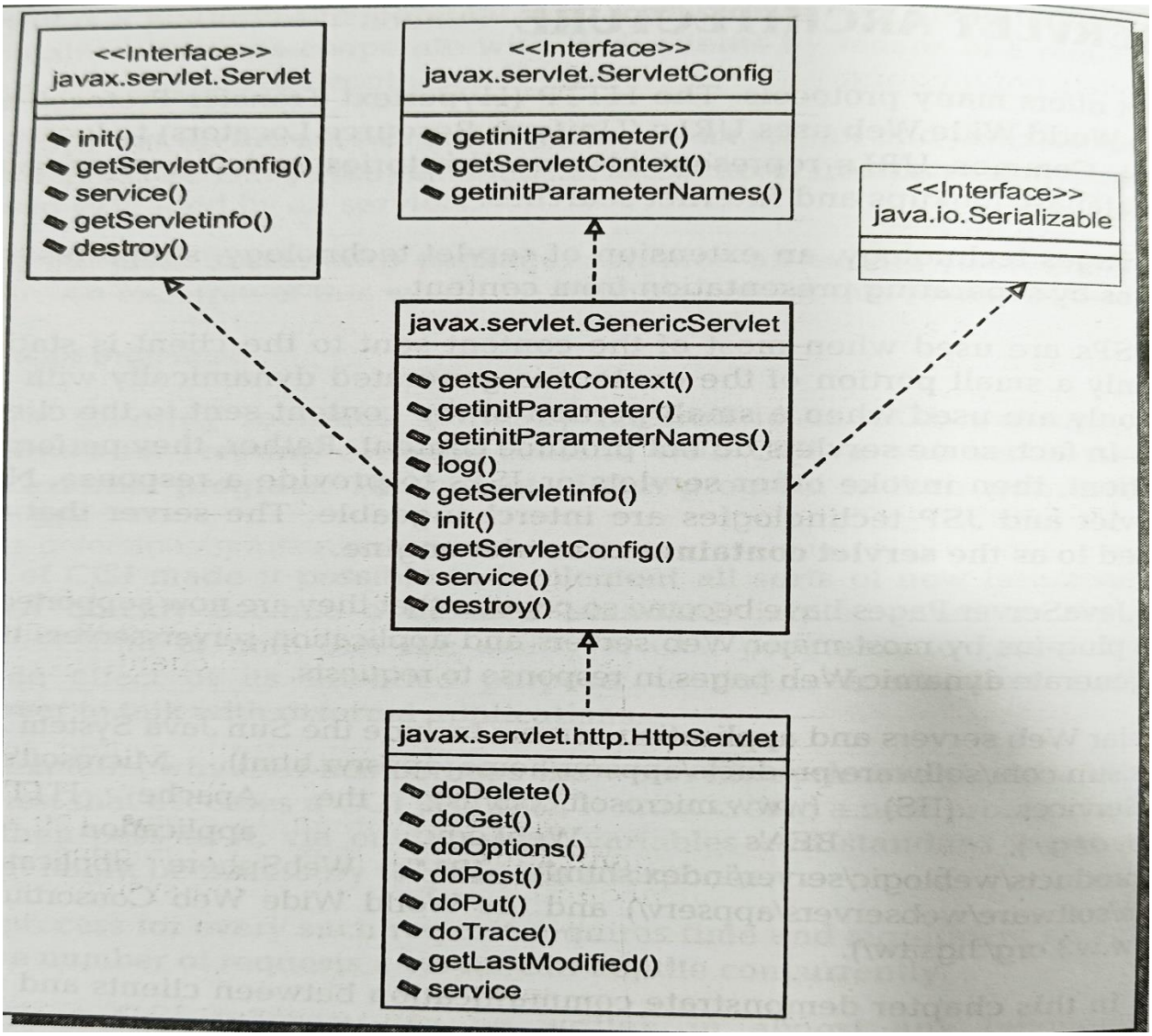
A servlet is a Java programming language class that directly or indirectly implements the jakarta.servlet.Servlet interface. The jakarta.servlet and jakarta.servlet.http packages provide interfaces and classes for writing servlets. All servlets must implement the jakarta.servlet.Servlet interface, which defines lifecycle methods such as init, service, and destroy. When implementing a generic service, you can extend the jakarta.servlet.GenericServlet class which already implements the Servlet interface. When implementing an HTTP service, you can extend the jakarta.servlet.http.HttpServlet class which already extends the GenericServlet class.

In a typical Jakarta Servlet based web application, the class must extend jakarta.servlet.http.HttpServlet and override one of the doXxx methods where Xxx represents the HTTP method of interest.

Servlet API

https://jakarta.ee/specifications/servlet/5.0/apidocs/jakarta/servlet/package-summary

The Servlet API is a set of Java interfaces and classes that provide a standardized way to create and manage web components on a server, such as servlets and filters. These components can handle requests and generate responses, typically in the context of web applications. The Servlet API is part of the Jakarta EE platform (previously Java EE).



Key Components of the Servlet API

1. Servlet Interface (jakarta.servlet.Servlet):

- **The central interface in the Servlet API, which must be implemented by any servlet.**
- **Key methods include init(), service(), and destroy().**

2. GenericServlet Class (jakarta.servlet.GenericServlet):

- **An abstract class that implements the Servlet interface and is designed to make it easier to write servlets.**
- **Provides simple versions of the lifecycle methods.**

3. HttpServlet Class (jakarta.servlet.http.HttpServlet):

- **Extends GenericServlet and provides methods specific to handling HTTP requests.**
- **Commonly used methods include doGet(), doPost(), doPut(), doDelete(), etc.**

4. ServletConfig Interface (jakarta.servlet.ServletConfig):

- **Provides configuration information to a servlet.**
- **Methods include getServletName(), getServletContext(), and getInitParameter().**

5. ServletContext Interface (jakarta.servlet.ServletContext):

- **Provides a servlet with information about its environment and allows interaction with the server (e.g., logging).**
- **Methods include getInitParameter(), getAttribute(), setAttribute(), and getRequestDispatcher().**

6. ServletRequest Interface (jakarta.servlet.ServletRequest):

- **Encapsulates client request information.**
- **Methods include getParameter(), getAttribute(), getInputStream(), etc.**

7. HttpServletRequest Interface (jakarta.servlet.http.HttpServletRequest):

- **Extends ServletRequest to provide HTTP-specific request information.**
- **Methods include getHeader(), getCookies(), getSession(), etc.**

8. ServletResponse Interface (jakarta.servlet.ServletResponse):

- **Encapsulates the response sent to the client.**
- **Methods include getWriter(), getOutputStream(), setContentType(), etc.**

9. HttpServletResponse Interface (jakarta.servlet.http.HttpServletResponse):

- **Extends ServletResponse to provide HTTP-specific functionality.**
- **Methods include setStatus(), addCookie(), sendRedirect(), etc.**

10. ServletException Class (jakarta.servlet.ServletException):

- **Represents a general exception that a servlet can throw when it encounters difficulty.**

11. Filter Interface (jakarta.servlet.Filter):

- **Provides a mechanism for filtering requests and responses.**
- **Methods include init(), doFilter(), and destroy().**

Servlet Lifecycle

lifecycle methods are defined in the jakarta.servlet.GenericServlet class and implemented in the jakarta.servlet.http.HttpServlet class.

Below set of methods which define the lifecycle of a Servlet.

init()

- **The init method is designed to be called only once. If an instance of the servlet does not exist, the web container:**
- **Loads the servlet class**
- **Creates an instance of the servlet class**
- **Initializes it by calling the init method**
- **The init method must complete successfully before the servlet can receive any requests. The servlet container cannot place the servlet into service if the init method either throws a ServletException or does not return within a time period defined by the Web server.**

service()

- **This method is only called after the servlet's init() method has completed successfully.**
- **The Container calls the service() method to handle requests coming from the client, interprets the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.**

destroy()

- **Called by the Servlet Container to take the Servlet out of service.**
- **This method is only called once all threads within the servlet's service method have exited or after a timeout period has passed. After the container calls this method, it will not call the service method again on the Servlet.**

HTTP Servlets

The HttpServlet abstract subclass adds additional methods beyond the basic Servlet interface that are automatically called by the service method in the HttpServlet class to aid in processing HTTP-based requests. These methods are:

- **doGet for handling HTTP GET requests**
- **doPost for handling HTTP POST requests**
- **doPut for handling HTTP PUT requests**
- **doDelete for handling HTTP DELETE requests**
- **doHead for handling HTTP HEAD requests**
- **doOptions for handling HTTP OPTIONS requests**
- **doTrace for handling HTTP TRACE requests**

Servlets Configuration

Deployment Descriptor (web.xml): Understand how to configure servlets using the `web.xml` file.

Annotations: Learn how to use annotations for servlet configuration (`@WebServlet`, `@WebInitParam`).

Servlet Context

ServletContext Interface: Understand the `ServletContext` interface and its methods.

Context Parameters: Learn how to configure and access context parameters.

Servlets Collaboration

RequestDispatcher: Learn about the `RequestDispatcher` interface for forwarding and including requests.

Servlet Communication: Understand how servlets communicate with each other.

Session Tracking

Session Management: Explore different techniques for session management, including cookies, URL rewriting, and `HttpSession` API.

Session Attributes: Learn how to store and retrieve session attributes.

CRUD Operations

Database Connectivity: Understand how to connect to a database using JDBC.

CRUD Operations: Learn to implement Create, Read, Update, and Delete operations using servlets.

Books

- *Head First Servlets and JSP* by Bryan Basham, Kathy Sierra, and Bert Bates

- *Java Servlet & JSP Cookbook* by Bruce W. Perry

Online Tutorials

- [Oracle's Official Servlet Documentation](https://docs.oracle.com/javaee/7/tutorial/servlets.htm)