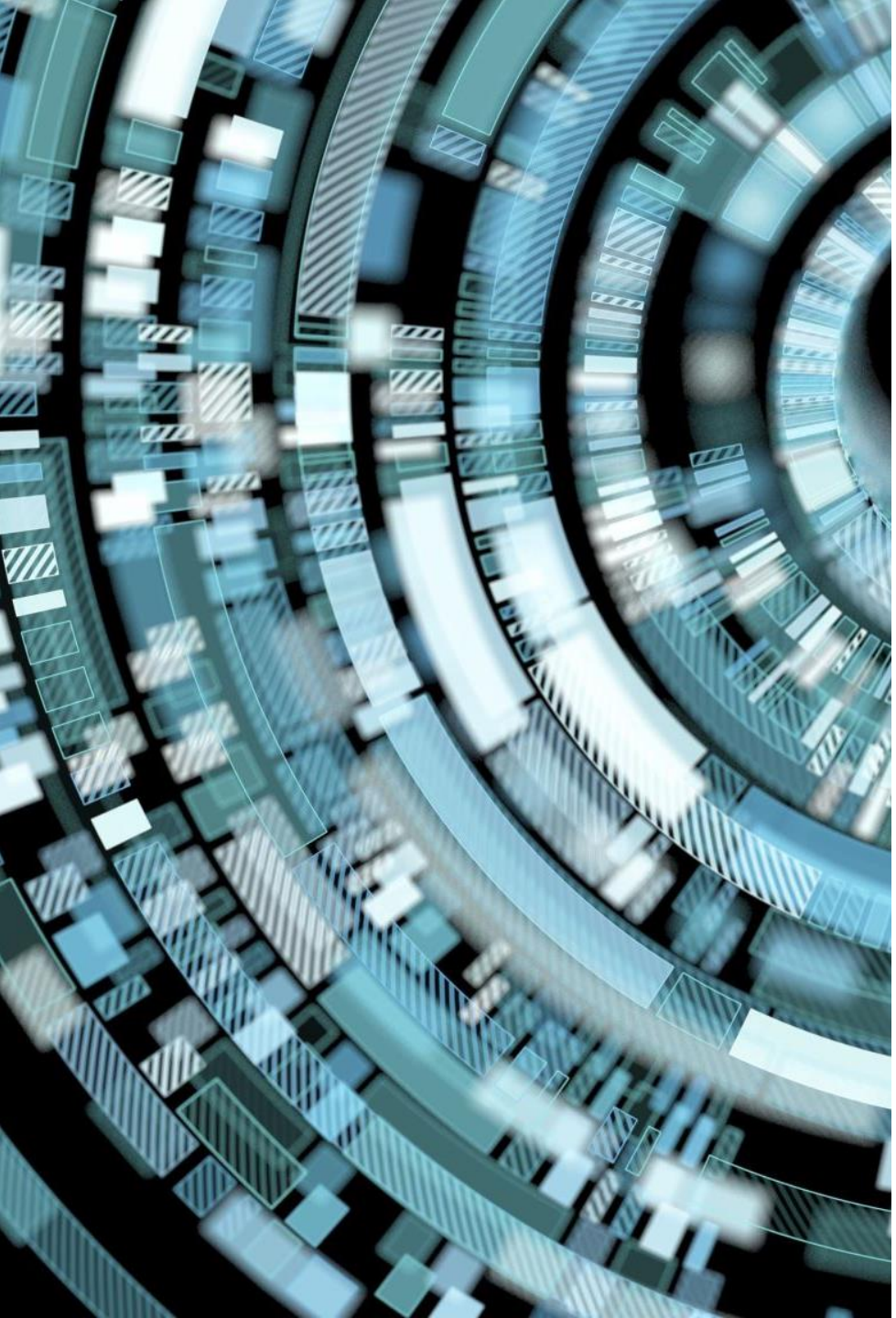
Computer System Design & Application

计算机系统设计与应用A

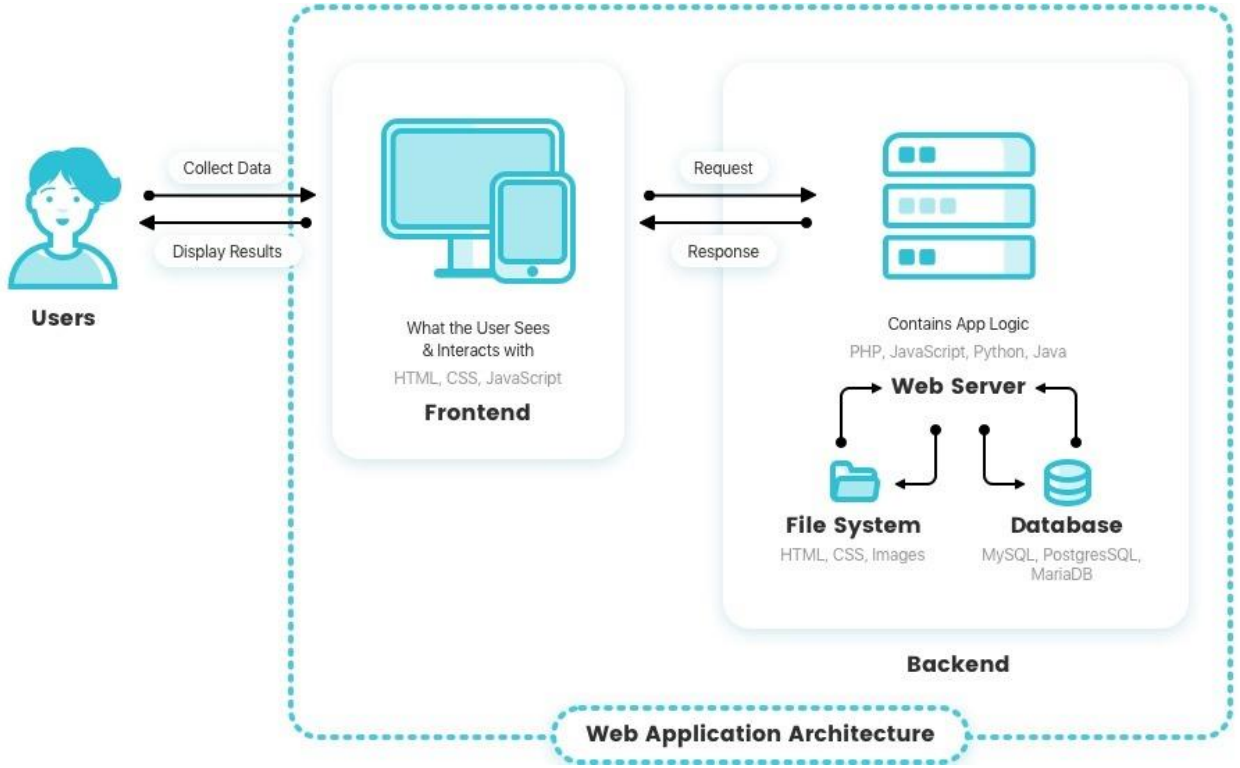
陶伊达 （TAO Yida）
taoyd@sustech.edu.cn



Lecture 12

- Web Development Overview
- Java EE
- Servlet & Containers
- JDBC & JPA

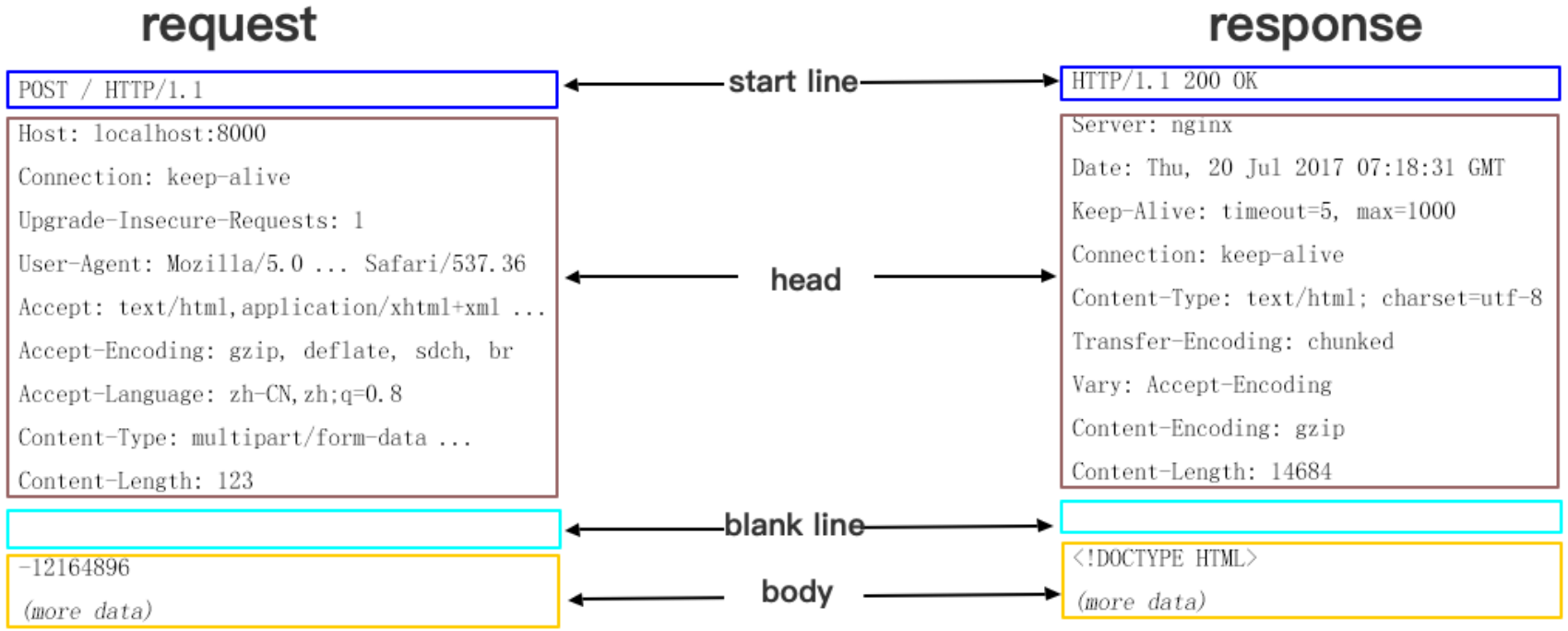
Web Application



- A web application (or web app) is application software that runs on a web server, unlike computer-based software programs that are run locally on the OS of the device.
- Web applications are accessed by the user through a web browser with an active network connection. These applications are programmed using a client-server modeled structure
- Example web applications: web-mail, online retail sales, online banking, etc.

Reference: Wikipedia
Image: https://reinvently.com/blog/fundamentals-web-application-architecture/

HTTP headers provide additional information about the data that will be sent

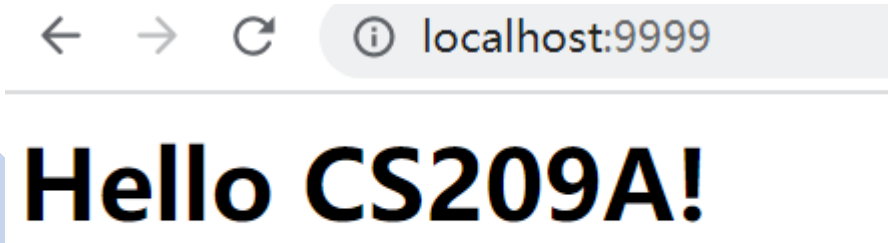


Data Exchange on the Web

- HTTP is the set of rules (protocol) for transferring files (e.g., text, images, sound, video) over the web
- Clients and servers exchange HTTP requests and responses, which follow specific syntax

Building a Web Server with ServerSocket

- Reply a fixed html whenever client is connected to the server
- Need proper HTTP header information for clients to parse
- Type localhost:9999 in browser (or localhost if port is 80)



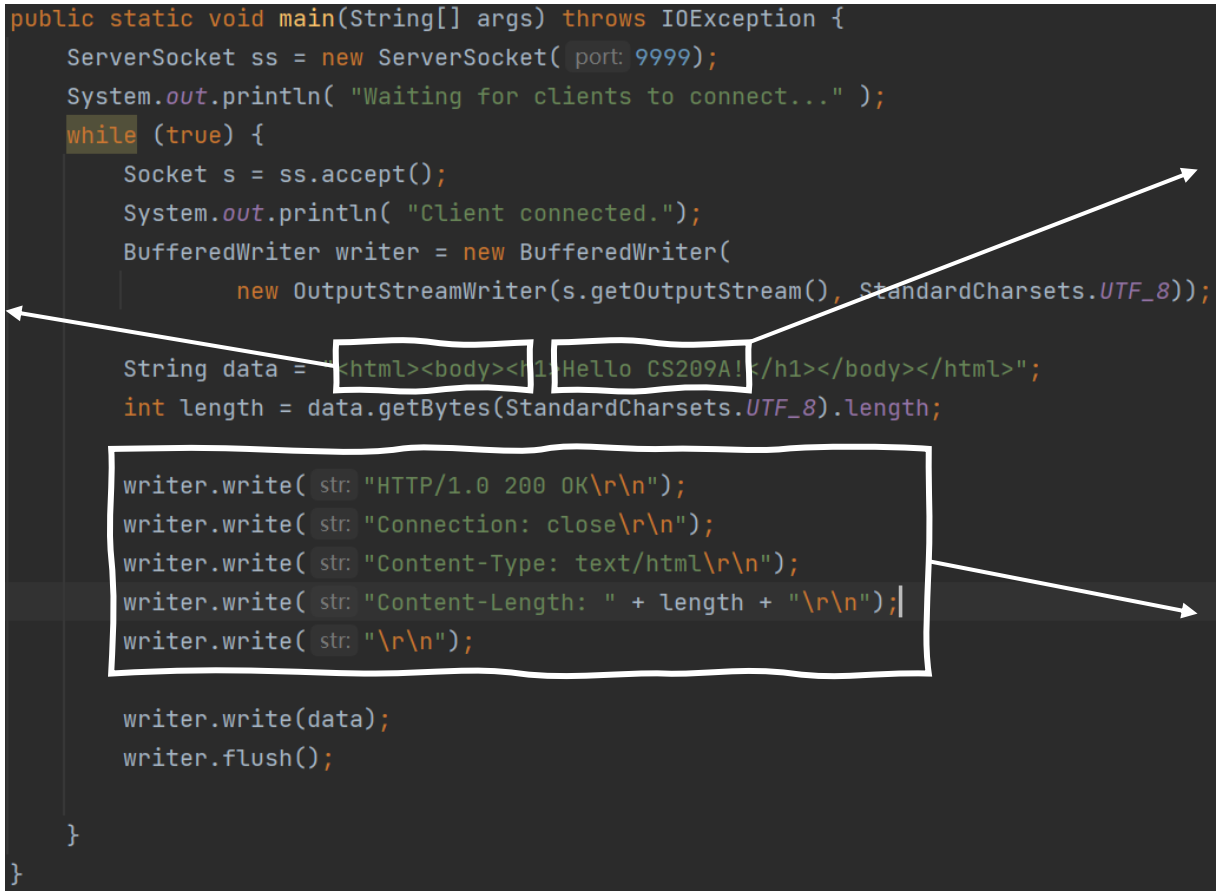

```java
public static void main(String[] args) throws IOException {
    ServerSocket ss = new ServerSocket( port: 9999);
    System.out.println( "Waiting for clients to connect..." );
    while (true) {
        Socket s = ss.accept();
        System.out.println( "Client connected.");
        BufferedWriter writer = new BufferedWriter(
                new OutputStreamWriter(s.getOutputStream(), StandardCharsets.UTF_8));

        String data = "<html><body><h1>Hello CS209A!</h1></body></html>";
        int length = data.getBytes(StandardCharsets.UTF_8).length;

        writer.write( str: "HTTP/1.0 200 OK\r\n");
        writer.write( str: "Connection: close\r\n");
        writer.write( str: "Content-Type: text/html\r\n");
        writer.write( str: "Content-Length: " + length + "\r\n");
        writer.write( str: "\r\n");

        writer.write(data);
        writer.flush();

    }
}
```

A web server is much more complex…

- We also have to:
  - Generate and parse correct HTTP headers/requests
  - Recognize and handle incorrect HTTP headers/requests
  - Handling concurrent requests
  - Handling network exceptions
  - Handling security issues
  - ….
- But we want to focus on application/business logic, instead of networking issues

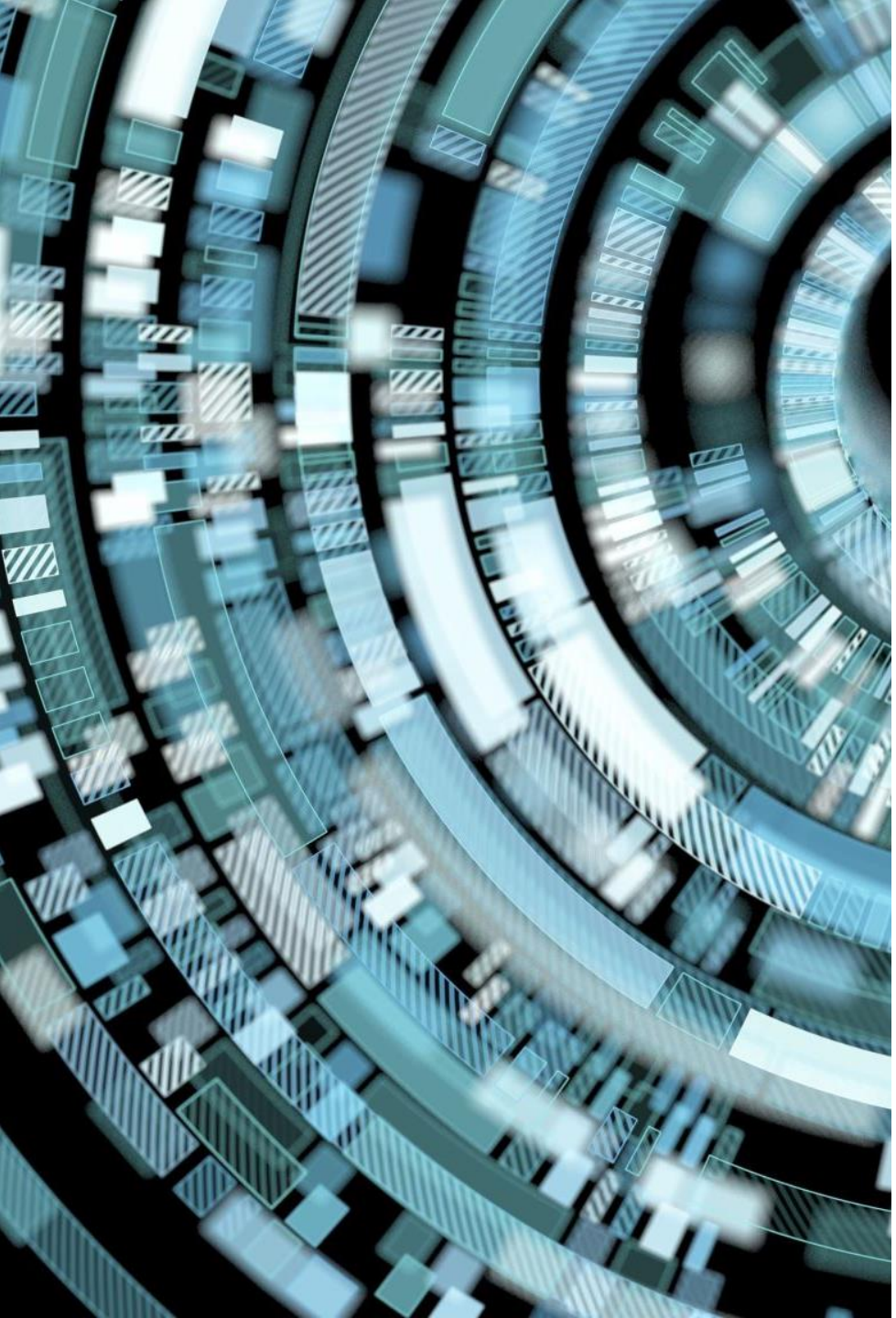
Division of Labor

User Interface: frontend developers/graphic designers



Application/Business Logic: developers

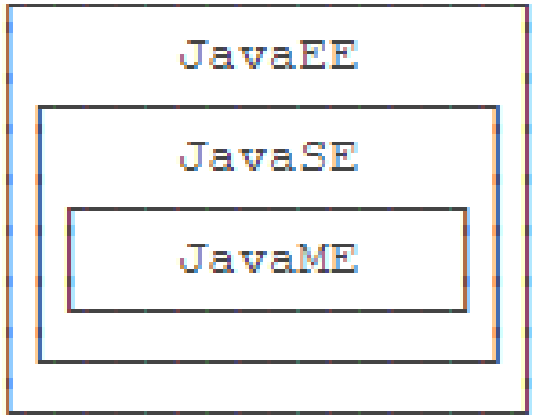
Reusable web technologies/framework



Lecture 12

- Web Development Overview
- Java EE
- Servlet & Containers
- JDBC & JPA

Java EE (Enterprise Edition)



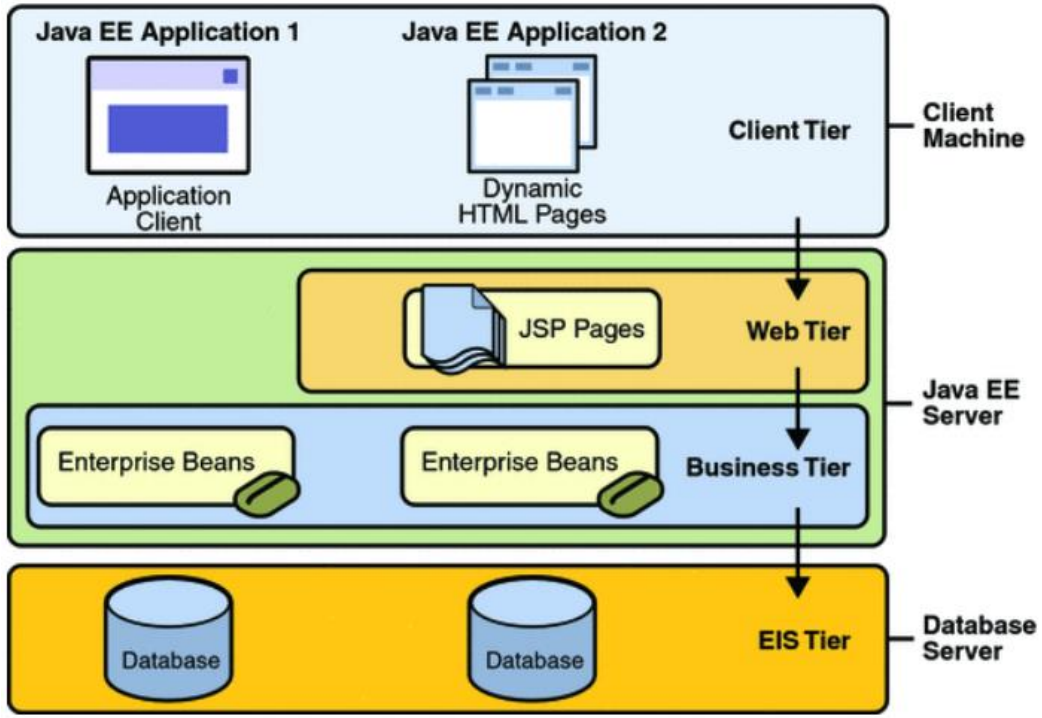
* Formerly known as J2EE and now known as Jakarta EE

- The Java technologies you'll use to create web applications are a part of Java EE platform
- Java EE is built on top of Java SE (Standard Edition), which contains core APIs that we use daily (java.lang, java.io, etc.), and adds libraries for database access (JDBC, JPA), servlets, remote method invocation (RMI), messaging (JMS), web services, XML processing, Enterprise Beans, etc.
- Java EE provides APIs and runtime environment to help developers create large-scale, multi-tiered, scalable, reliable, and secure web/business applications

Multitiered Applications

https://docs.oracle.com/javaee/7/firstcup/java-ee001.htm
https://docs.oracle.com/cd/E19575-01/819-3669/gfirp/index.html

- Java EE reduces the complexity of enterprise application by using a multitiered application model
- In a multi-tiered application, the functionality of the application is separated into isolated functional areas, called tiers; Typically, multi-tiered applications have a client tier, a middle tier, and a data tier



The client tier consists of a client program that makes requests to the middle tier

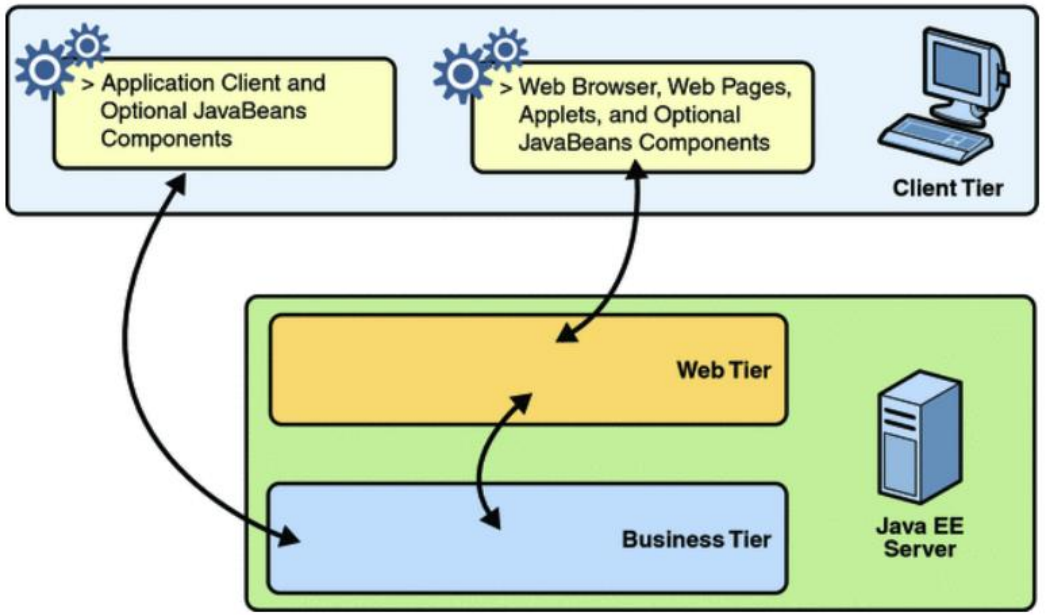
The middle tier is divided into a web tier and a business tier, which handle client requests and process application data, storing it in a permanent datastore in the data tier (often called the enterprise information systems tier).

Client Tier

- A Java EE client can be a web client or an application client.

Application client

- runs on a client machine and typically has a GUI (e.g., created from Swing or AWT)
- Can directly access enterprise beans running in the business tier or communicate with a servlet running in the web tier
- Can be written in other languages and interact with Java EE servers



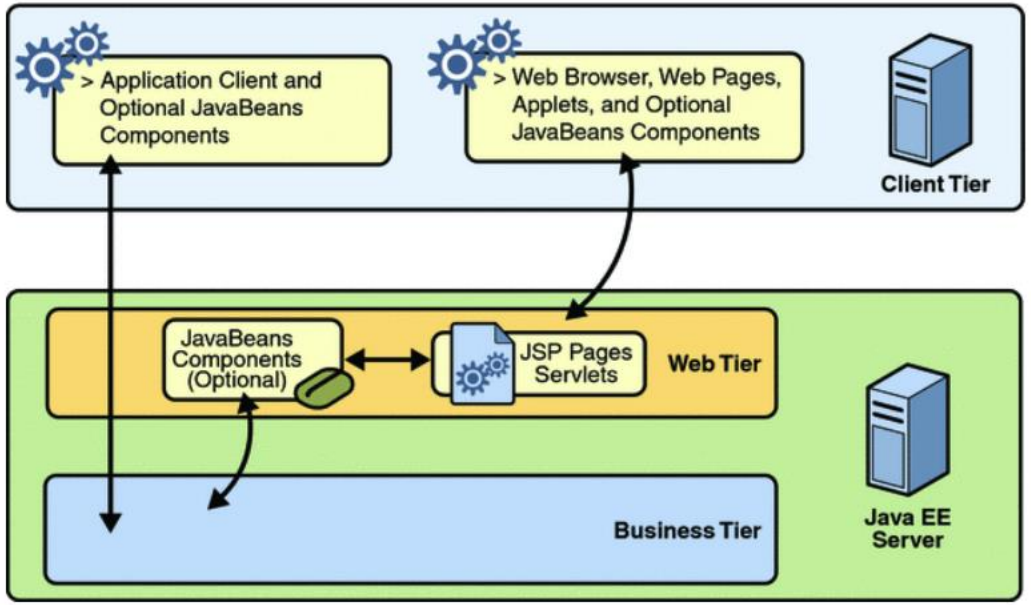
Web client

- consists of web pages and a web browser
- usually do not query databases, execute complex business rules
- A web page received from the web tier can include an embedded applet, a small client application written in Java that executes in JVM installed in the web browser

https://docs.oracle.com/cd/E19575-01/819-3669/gfirp/index.html

Web Tier

- The web tier consists of components that handle the interaction between clients and the business tier.



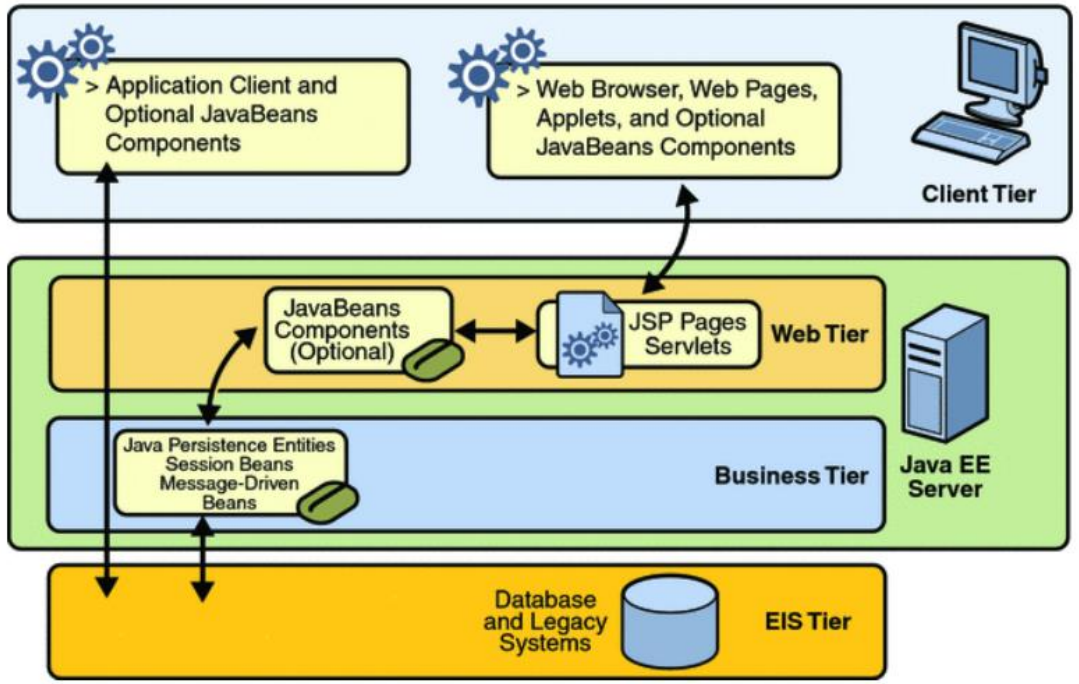
Java EE web-tier technologies

- **Servlet**: Java classes (APIs) that dynamically process requests and construct responses
- **JSP** (JavaServer Pages): extends/executes Servlet and intends to fullfill UI by generating web pages with HTML, XML, etc.
- **JSF** (JavaServer Faces): builds on servlets and JSP technology and provides a user interface component framework for web applications

https://docs.oracle.com/cd/E19575-01/819-3669/gfirp/index.html

Business Tier

- Business code that solves or meets the needs of a particular business domain (e.g., banking, retail, or finance), is handled by enterprise beans running in the business tier
- In a properly designed enterprise application, the core functionality exists in the business tier components



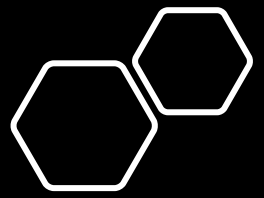
https://docs.oracle.com/cd/E19575-01/819-3669/gfirp/index.html

Java EE Business-tier technologies

- **EJB**
- **JAX-RS RESTful web services**
- **JAX-WS web service end points**

Enterprise JavaBeans (EJB)

- receives data from client programs, processes it (if necessary), and sends it to the enterprise information system tier for storage
- retrieves data from storage, processes it (if necessary), and sends it back to the client program

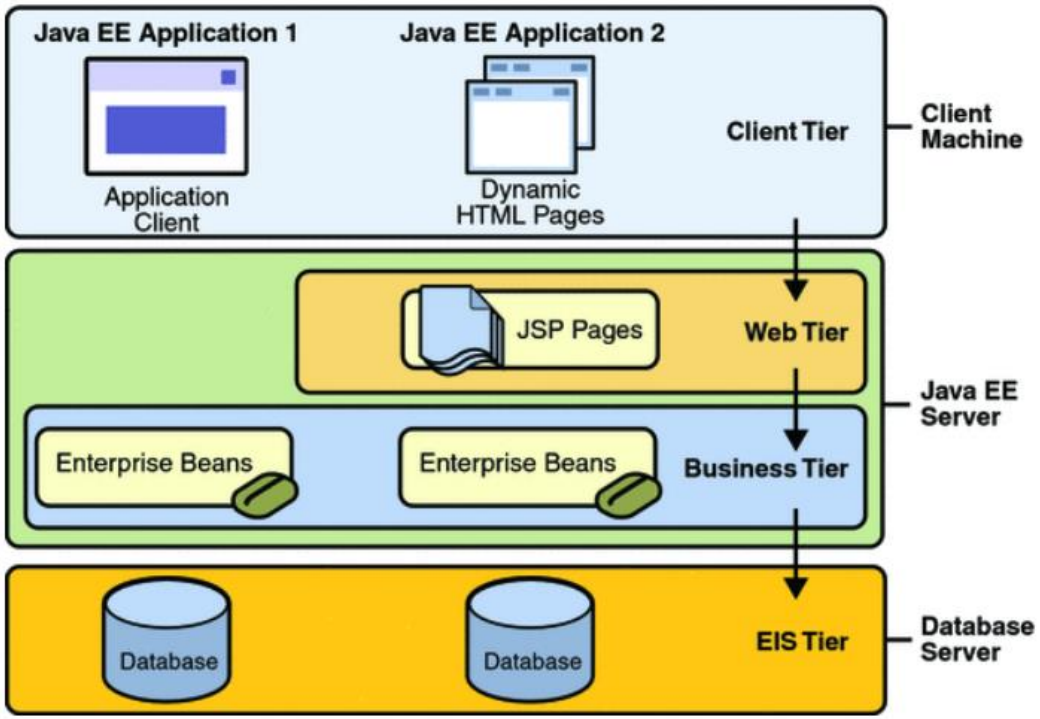


Terminology

- **POJO** (Plain Old Java Object): regular Java objects
- **JavaBean**: a POJO that conforms to certain conventions
  - All properties are private (use getters/setters)
  - A public no-argument constructor
  - Implements Serializable.
- **Enterprise JavaBean (EJB):** a server-side software component that encapsulates the business logic (e.g., sessions, security, etc.) of an enterprise application

Data Tier

- Also called the enterprise information systems (EIS) tier
- EIS consists of database servers, enterprise resource planning systems, and other legacy data sources, which typically locate on a separate machine from the Java EE server, and are accessed by the business tier



Java EE data-tier technologies

- The Java Database Connectivity API (JDBC)
- The Java Persistence API (JPA)
- The Java EE Connector Architecture
- The Java Transaction API (JTA)

https://docs.oracle.com/javaee/7/firstcup/java-ee001.htm
https://docs.oracle.com/cd/E19575-01/819-3669/gfirp/index.html

Java EE Servers, Components, Containers

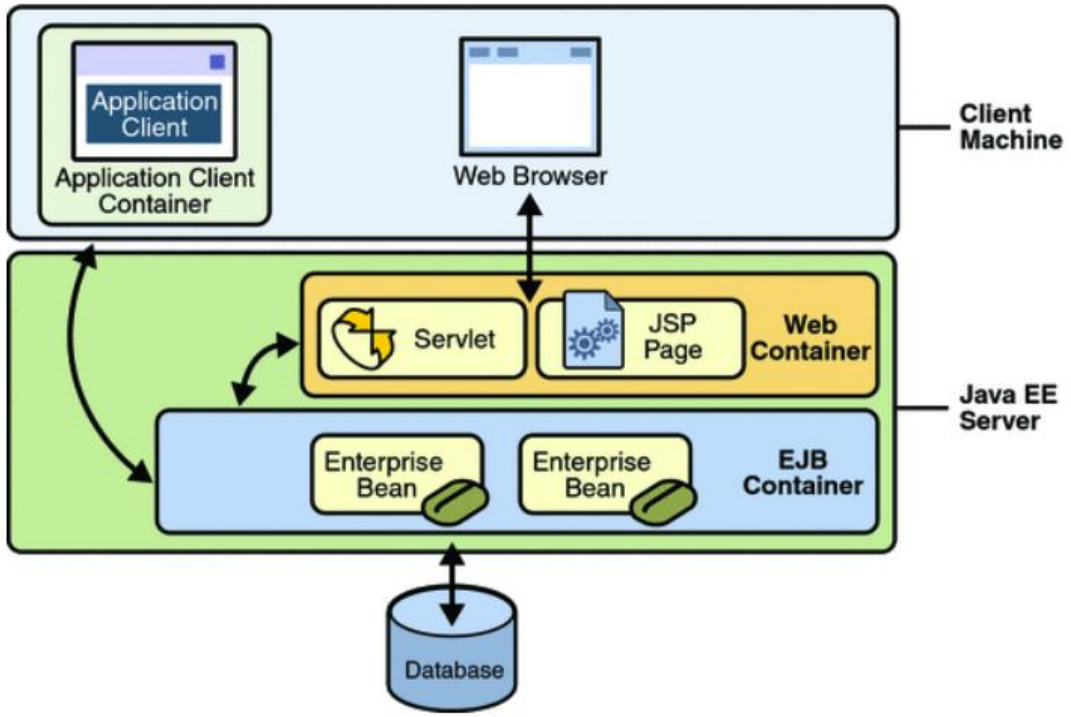
- Java EE applications are made up of components
  - Application clients and applets are components that run on the client.
  - Java Servlet, JSF, and JSP technology components are web components that run on the server.
  - Enterprise JavaBeans (EJB) components are business components that run on the server.
- Java EE components are written and compiled in Java
- Difference between Java EE components and standard Java classes
  - Java EE components are assembled into a Java EE application
  - Java EE components are verified to be well formed and in compliance with the Java EE specification
  - Java EE components are deployed to production, where they are run and managed by the Java EE server.

Java EE Servers, Components, Containers

- A Java EE server is a server application that implements the Java EE platform APIs and provides standard Java EE services.
- Java EE servers host several application component types (e.g., servlet, EJB) that correspond to the tiers in a multi-tiered application. The Java EE server provides services to these components in the form of a container.
- Java EE containers are the interface between a component and the low-level platform-specific functionality (e.g., transaction and state management, multithreading, resource pooling) that supports the component

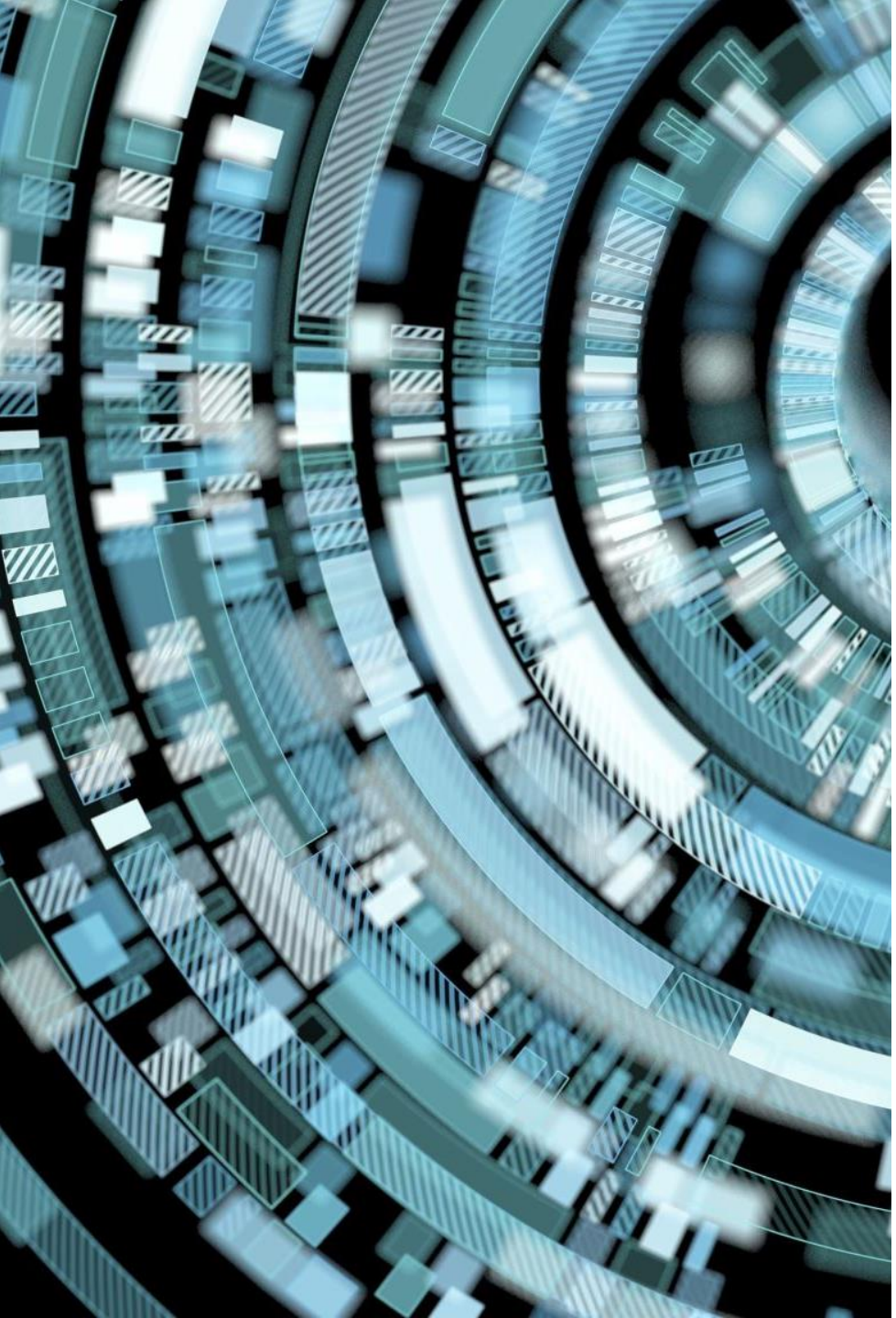
Container Types

- Before a web component, enterprise bean, or application client component can be executed, it must be assembled into a Java EE module and deployed into its container



- **Enterprise JavaBeans (EJB) container**: Manages the execution of enterprise beans for Java EE applications. Enterprise beans and their container run on the Java EE server.
- **Web container**: Manages the execution of JSP page and servlet components for Java EE applications. Web components and their container run on the Java EE server.
- **Application client container**: Manages the execution of application client components. Application clients and their container run on the client.
- **Applet container**: Manages the execution of applets. Consists of a web browser and Java Plug-in running on the client together.

https://docs.oracle.com/cd/E19575-01/819-3669/gfirp/index.html

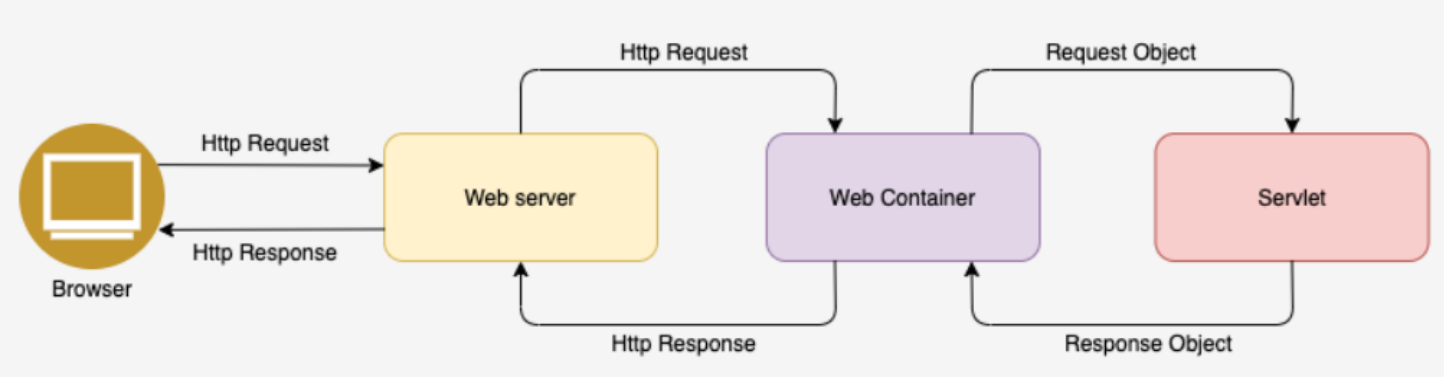


Lecture 12

- Web Development Overview
- Java EE
- Servlet & Containers
- JDBC & JPA

Workflow

- The client sends an HTTP Request to the web server
- Web server forwards requests to Web Container
- Web Container parse the HTTP request to objects and forward the request objects to the Servlet
- Servlet implement the application logic, builds the response object and sends it back to the Web Container
- Web container transforms the response object to equivalent HTTP response and sends it to the web server
- The web server sends the response via HTTP response back to the client.



https://codeburst.io/understanding-java-servlet-architecture-b74f5ea64bf4

Web Server vs. Web Container

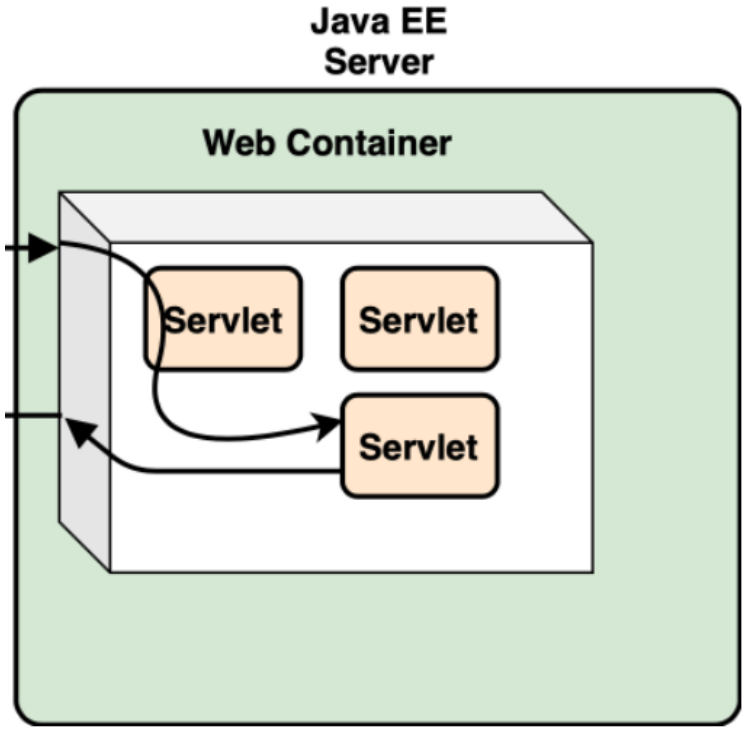
- Web Server
  - Serves static files (e.g., HTML, images, text) via the HTTP protocol
  - You can write a very simple one in Java in a few lines of code; or using an open source one (e.g., Apache HTTPD)
- Web Container
  - Serves dynamic content by executing the server-side web component (e.g., servlet)
  - Convert HTTP requests to request objects and convert response objects to HTTP response

What is Servlet?

- Servlet is nothing but a Java program/class
- Servlets respond to incoming requests by implementing application or business logics
- Servlet can not understand raw requests; its a Java program, which only understands objects
- Servlets run in a servlet container, which take care of the internal low-level details of parsing incoming requests and convert them to valid request objects

How containers & servlets work?

Mapping URL paths to corresponding servlets (typically by web.xml or annotations)



https://sergiomartinrubio.com/articles/get-started-with-java-servlets/

The `Servlet` Interface

Defines methods that all servlets must implement

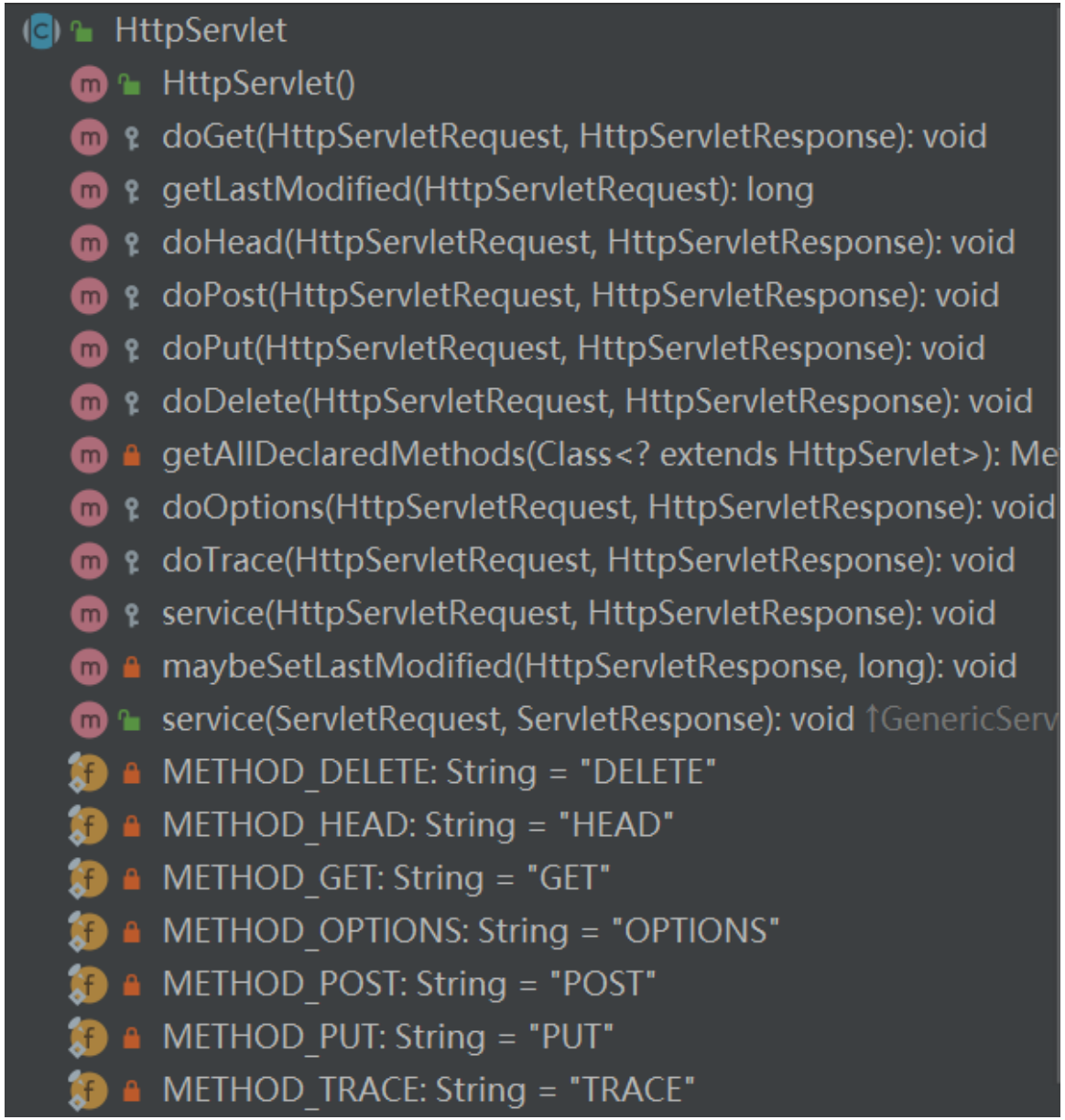
Method and Description
`destroy()` Called by the servlet container to indicate to a servlet that the servlet is being taken out of service.
`getServletConfig()` Returns a `ServletConfig` object, which contains initialization and startup parameters for this servlet.
`getServletInfo()` Returns information about the servlet, such as author, version, and copyright.
`init(ServletConfig config)` Called by the servlet container to indicate to a servlet that the servlet is being placed into service.
`service(ServletRequest req, ServletResponse res)` Called by the servlet container to allow the servlet to respond to a request.

- Methods to initialize a servlet, to service requests, and to remove a servlet from the server
- Methods to get basic information and startup configuration

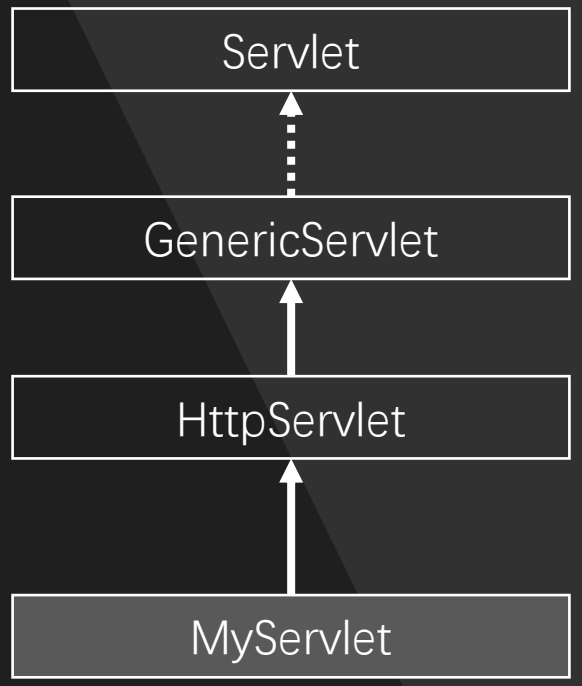
The `HttpServlet` Class

- `GenericServlet` implements `Servlet`
  - An abstract class
  - A generic, protocol-independent servlet.

- `HttpServlet` extends `GenericServlet`
  - An abstract class
  - Defines a HTTP protocol specific servlet.
  - Adds fields and methods that are specific to HTTP protocol



The `HttpServlet` Class



Typically, we would directly extend HttpServlet to create our own HTTP servlets

HttpServlet
- HttpServlet()
- doGet(HttpServletRequest, HttpServletResponse): void
- getLastModified(HttpServletRequest): long
- doHead(HttpServletRequest, HttpServletResponse): void
- doPost(HttpServletRequest, HttpServletResponse): void
- doPut(HttpServletRequest, HttpServletResponse): void
- doDelete(HttpServletRequest, HttpServletResponse): void
- getAllDeclaredMethods(Class<? extends HttpServlet>): Me
- doOptions(HttpServletRequest, HttpServletResponse): void
- doTrace(HttpServletRequest, HttpServletResponse): void
- service(HttpServletRequest, HttpServletResponse): void
- maybeSetLastModified(HttpServletResponse, long): void
- service(ServletRequest, ServletResponse): void ↑GenericServ
- METHOD_DELETE: String = "DELETE"
- METHOD_HEAD: String = "HEAD"
- METHOD_GET: String = "GET"
- METHOD_OPTIONS: String = "OPTIONS"
- METHOD_POST: String = "POST"
- METHOD_PUT: String = "PUT"
- METHOD_TRACE: String = "TRACE"

The `HttpServlet` Class

- Provides an abstract class to be subclassed
- HttpServlet overrides service(), which dispatches the HTTP requests to corresponding methods (e.g., GET -> doGet())
- A subclass of HttpServlet must override at least one method, usually one of these:
  - `doGet`, if the servlet supports HTTP GET requests
  - `doPost`, for HTTP POST requests
  - `doPut`, for HTTP PUT requests
  - `doDelete`, for HTTP DELETE requests
  - `init` and `destroy`, to manage resources that are held for the life of the servlet

```
protected void service(HttpServletRequest req, HttpServletResponse resp)
    String method = req.getMethod();
    long lastModified;
    if (method.equals("GET")) {...} else if (method.equals("HEAD")) {
        lastModified = this.getLastModified(req);
        this.maybeSetLastModified(resp, lastModified);
        this.doHead(req, resp);
    } else if (method.equals("POST")) {
        this.doPost(req, resp);
    } else if (method.equals("PUT")) {
        this.doPut(req, resp);
    } else if (method.equals("DELETE")) {
        this.doDelete(req, resp);
    } else if (method.equals("OPTIONS")) {
        this.doOptions(req, resp);
    } else if (method.equals("TRACE")) {
        this.doTrace(req, resp);
    } else {
        String errMsg = lStrings.getString( key: "http.method_not_implemer
        Object[] errArgs = new Object[]{method};
```

Example

doGet

- Called by the server (via the service method) to allow a servlet to handle a GET request.
- When overriding this method, read the request data, write the response headers, get the response's writer or output stream object, and finally, write the response data.
- It's best to include content type and encoding.

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

@WebServlet(name = "helloServlet", value = "/hello-servlet")
public class HelloServlet extends HttpServlet {

    private String message;

    public void init() {
        message = "Hello CS209A!";
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
        response.setContentType("text/html");

        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("<h1>" + message + "</h1>");
        out.println("</body></html>");
    }

    public void destroy() {
    }
}
```

The @WebServlet annotation is used to declare a servlet. The annotated class must extend the HttpServlet class.

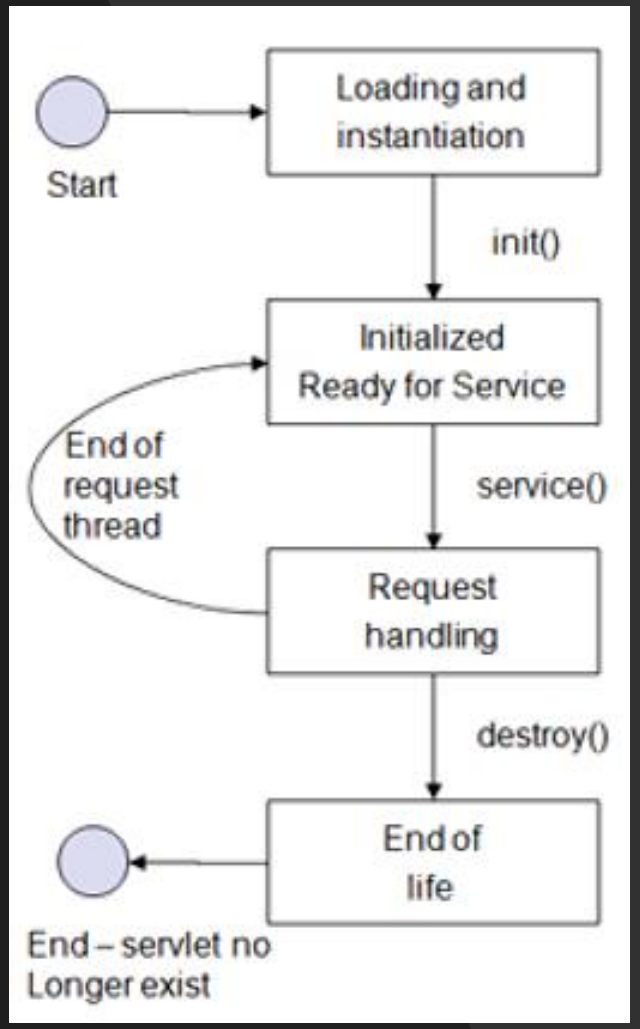
- value: required, specify the url of this servlet
- Name: optional, specify the name of this servlet

init() and destroy() manage resources that are held for the life of the servlet

Servlet Containers

- A servlet container is nothing but a compiled, executable program that runs on top of JVM
- The main function of the servlet container is to load, initialize, and execute servlets
- Servlet container manages the entire lifecycle of servlets

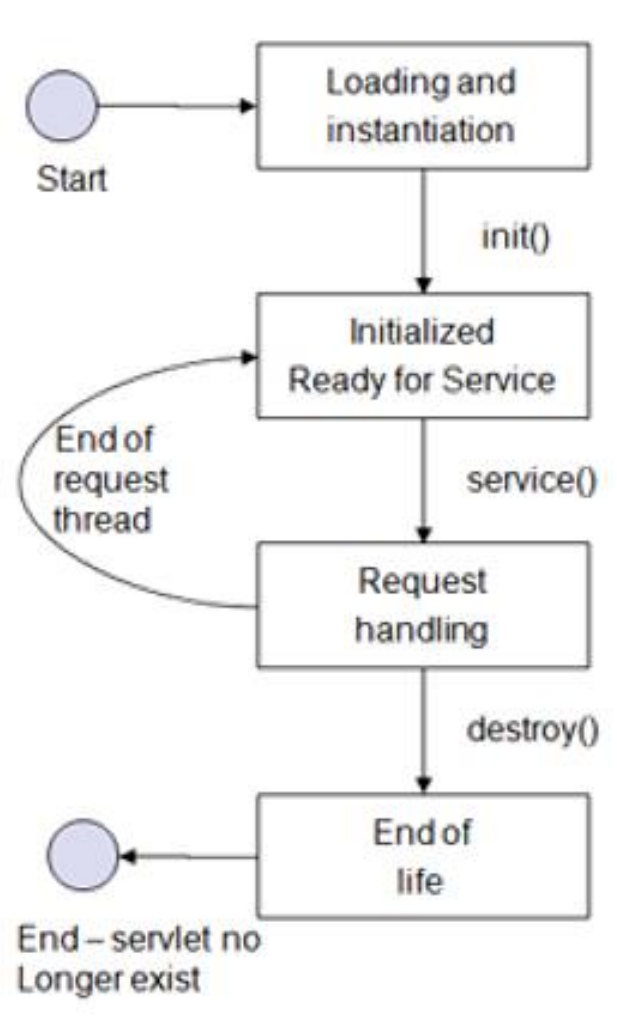
Servlet Lifecycle



- Managed by containers (all of these methods are invoked by containers, not by programmers)
- **init()** is called only once. It is invoked only when the servlet is created. So, it is used for one-time initializations.
- **service()** is the main method that performs the actual task. The servlet container calls the service() method to handle requests coming from the client.
  - Each time the server receives a request for a servlet, the web container spawns a new thread and calls service().
  - service() checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

https://codeburst.io/understanding-java-servlet-architecture-b74f5ea64bf4

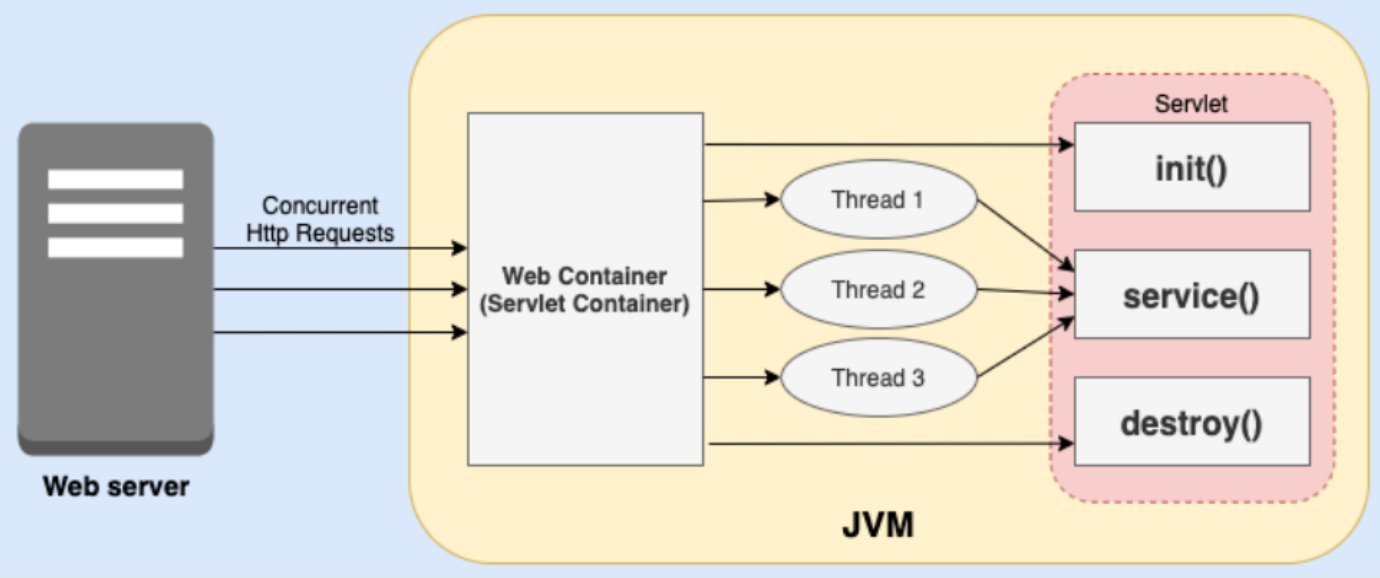
Servlet Lifecycle



- **destroy()** is invoked only once at the end of the life cycle of a servlet. This method gives the servlet a chance to close database connections, halt background threads, and perform other cleanup tasks.
  - This method is only called once all threads within the servlet's service method have exited or after a timeout period has passed
  - After destroy() is executed, the servlet object is marked available for garbage collection.
  - Once the destroy method is called on a servlet instance, the container may not route other requests to that instance of the servlet. If the container needs to enable the servlet again, it must do so with a new instance of the servlet’ s class.

https://codeburst.io/understanding-java-servlet-architecture-b74f5ea64bf4

Servlet Lifecycle



https://codeburst.io/understanding-java-servlet-architecture-b74f5ea64bf4

1. Concurrent HTTP requests coming to the server are forwarded to the web container.
2. The web container creates an instance of the servlet and executes init()
3. The container handles multiple requests to the same instance by spawning multiple threads, each thread executing the service() method of a same instance of the servlet.
4. The container calls destroy() once all threads for a servlet exited; the servlet instance is removed from the container

Multithreading

- A Java servlet container is typically multithreaded: multiple requests to the same servlet may be executed at the same time.
  - The container takes care of multithreading
- By default, a container may have only one instance per servlet declaration
  - The container handles concurrent requests to the same servlet by concurrent execution of the service method on different threads.
  - We make sure that the servlet code is implemented to be thread-safe (i.e., accessing shared resource like instance/class variables)

Where is `javax.servlet`?

We mostly use Java SE JDK

javax.servlet is part of Java EE; we should install Java EE SDK

Alternatively, simple servlet containers (e.g., Tomcat) also come with this API (servlet-api.jar).

https://stackoverflow.com/questions/860022/wheres-javax-servlet

What exactly is Java EE?

What exactly is Java EE?

- Java EE is indeed an abstract specification, which describes the standards, expected behaviors, and interactions between APIs (what we have learned so far)
- Anybody (companies, providers, developers) is open to develop and provide a working, concrete implementation of (part of) the specification.
- An application is “Java EE compliant” if it meets the requirements of Java EE specification

Reference Implementations

- In Java specifications' cases, you usually have a **reference implementation (RI)** created while drafting the specification
- Then other providers who may create their own implementation of the specification (often claiming it's "better" in some way).
- Java EE developers should write code following the specification (`import javax...`), then the code would run correctly on any concrete implementation.

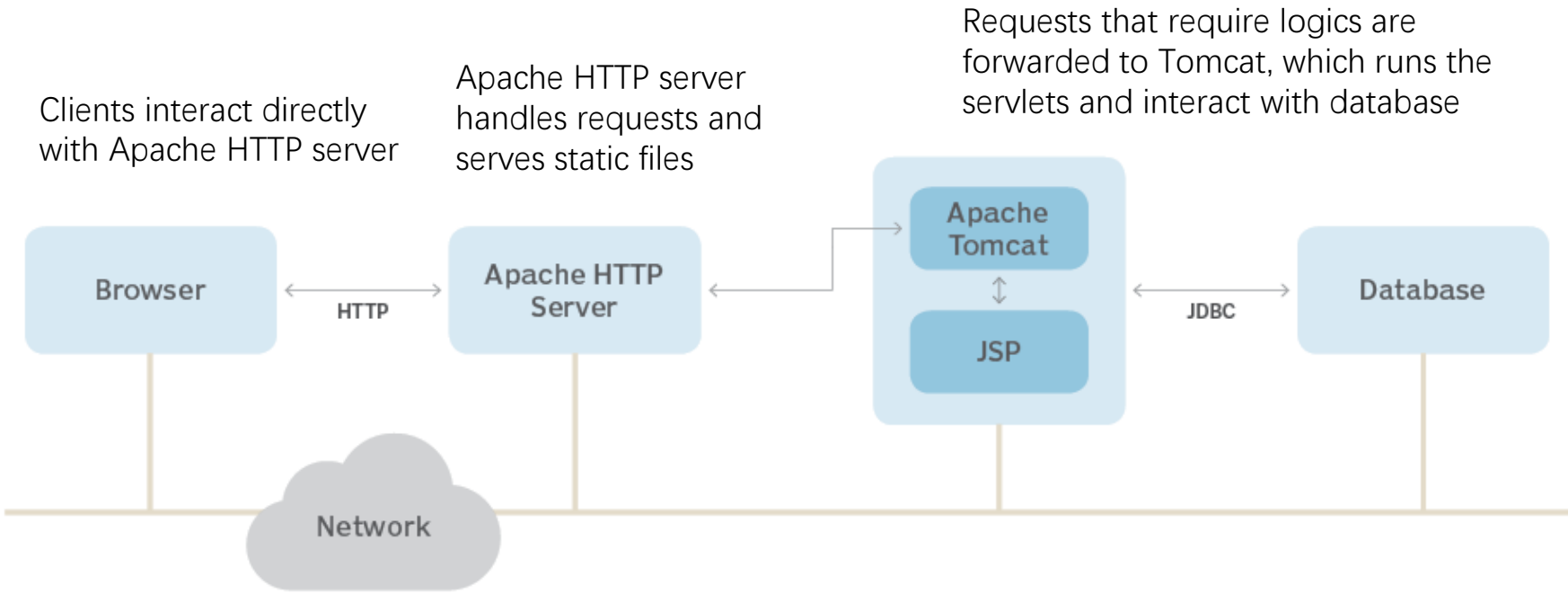
Java EE 6 RIs and Alternatives

https://stackoverflow.com/questions/2296678/where-can-i-find-a-list-of-all-the-reference-implementations-for-java-ee-6

- **Java EE full-fledged**
  - ***Oracle Glassfish (RI)***
  - **JBoss AS**
  - **IBM WebSphere**
- **Servlet & JSP**
  - *Oracle Glassfish (RI)*
  - Apache Tomcat
  - Eclipse Jetty
  - Resin
- **Enterprise JavaBeans (EJBs)**
  - *Oracle Glassfish (RI)*
  - Apache TomEE and OpenEJB
  - BuzyBeans
- **Java Persistence API (JPA)**
  - *EclipseLink (RI, used in Glassfish)*
  - OpenJPA
  - Hibernate

Apache Tomcat

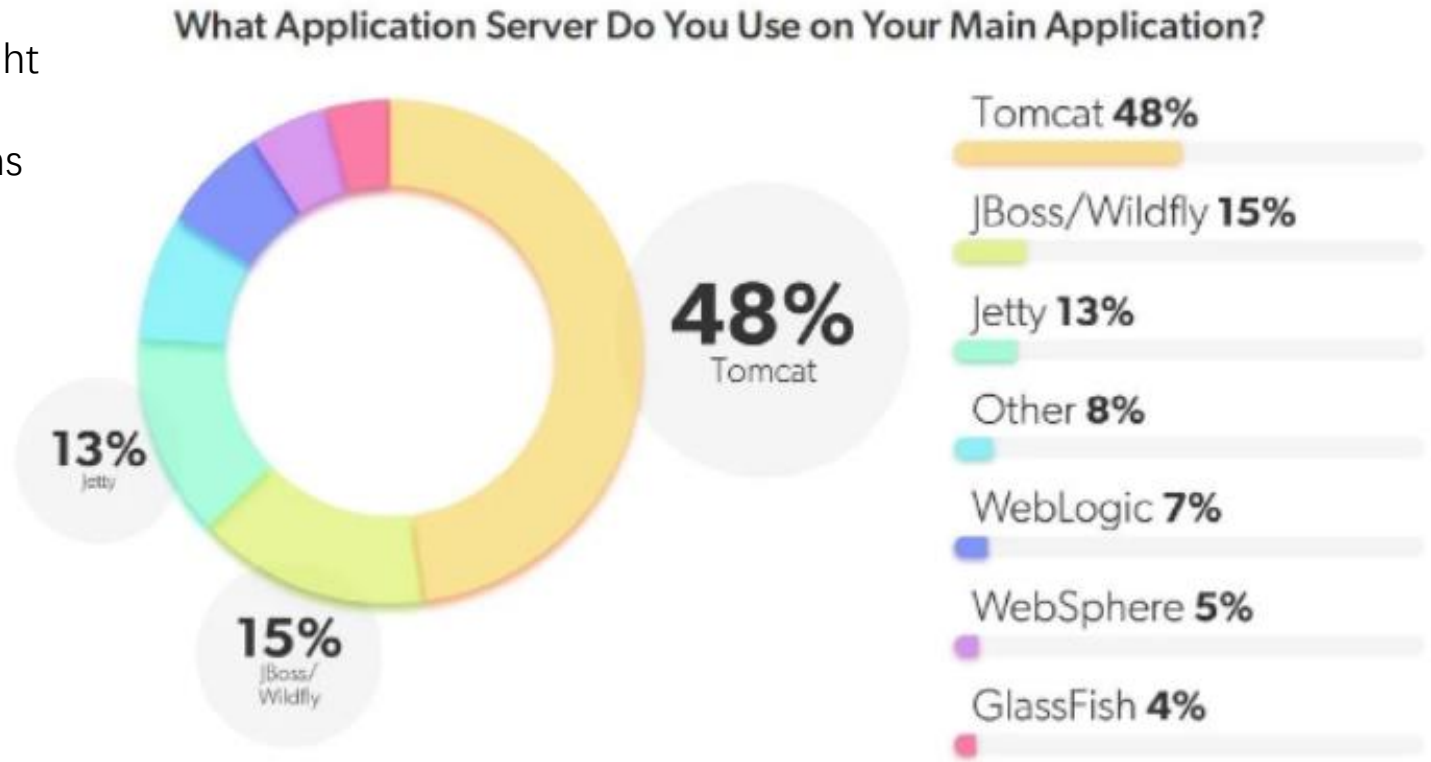
- Tomcat is a webcontainer which allows running servlet and JSP based web applications
- Tomcat is written in Java and requires JDK to run



- Tomcat also has it's own HTTP server built into it, and is fully functional at serving static content too.
- But the performance of Tomcat as HTTP server is not as good as the performance of a designated web server, e.g., Apache HTTP server.
- For simple (production) applications, Tomcat alone is sufficient and good enough

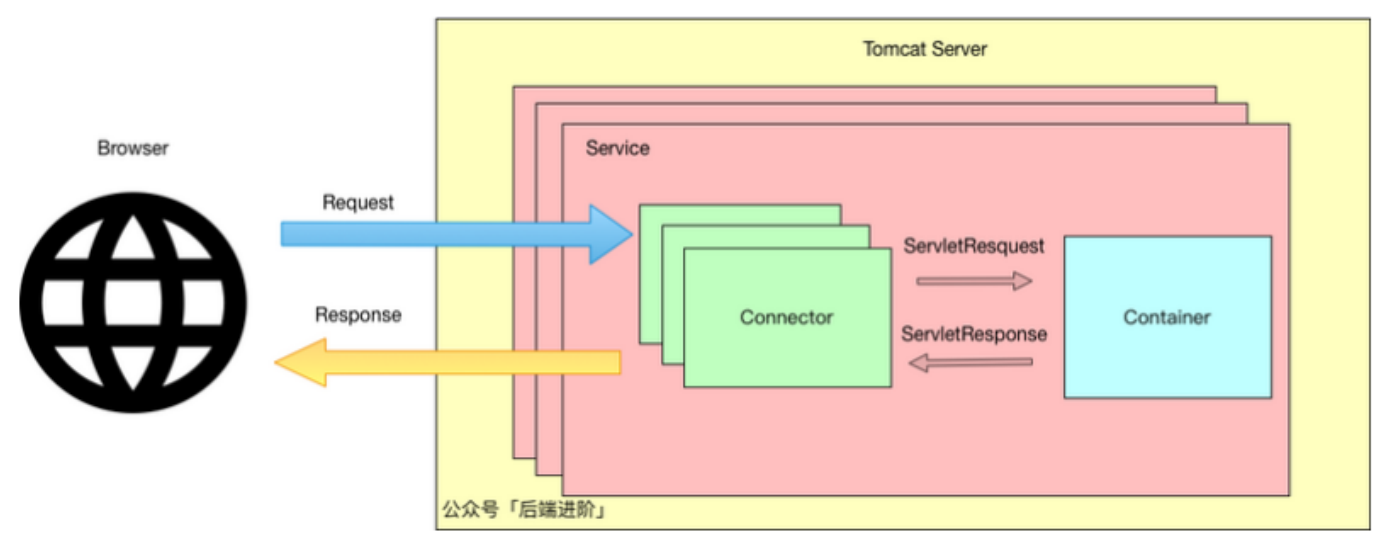
Is Tomcat Still Popular?

- Tomcat is free and lightweight
- It offers basic functionalities needed by many applications



Source: https://www.jrebel.com/resources/java-developer-productivity-report-2022

Architecture of Tomcat



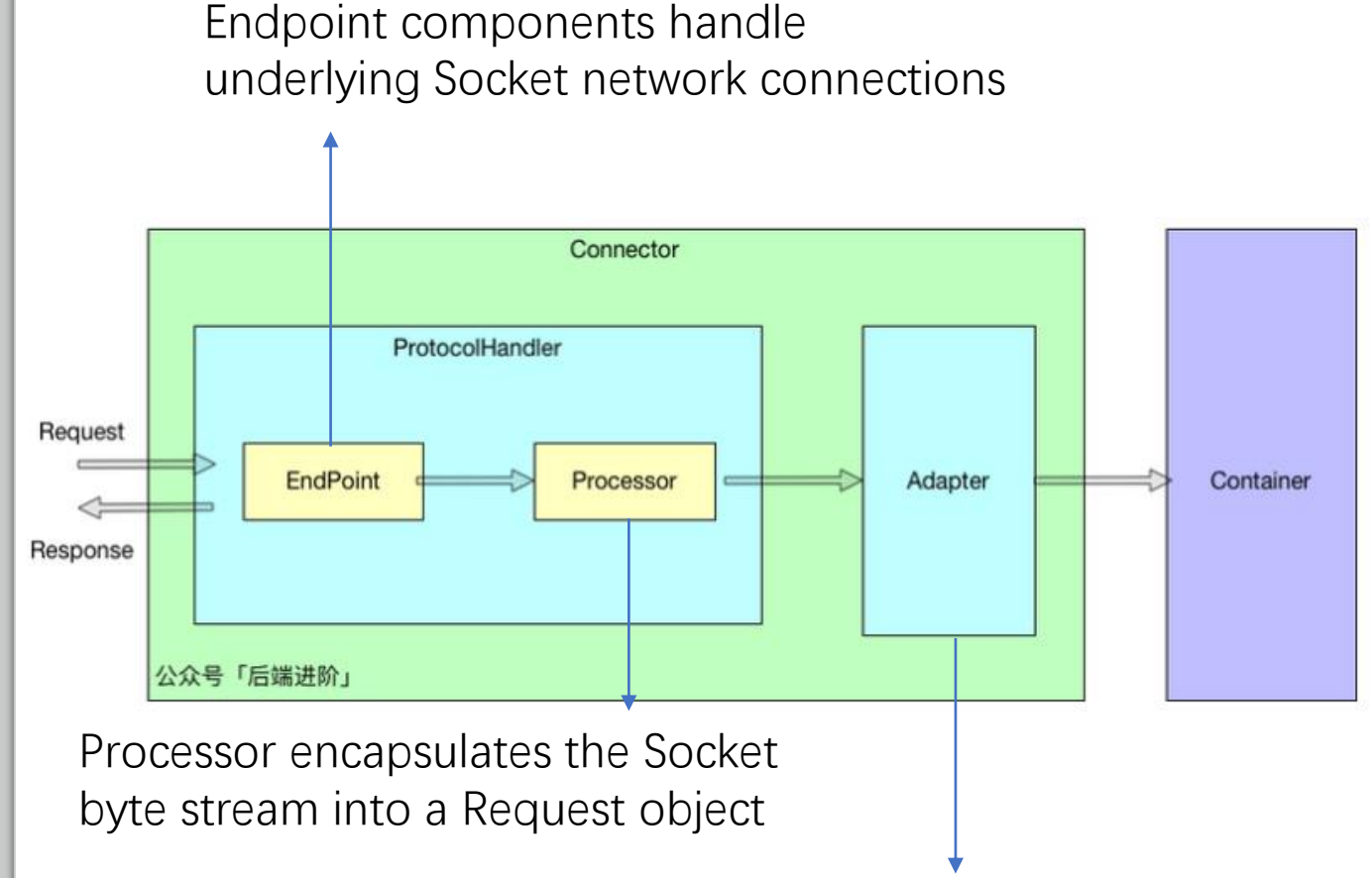
https://developpaper.com/talking-about-tomcats-architecture-design/

- Tomcat represents a server
- A server can provide multiple services (multiple ports)
- A service can contain multiple connectors
  - Connector handles network connection
  - Multiple connectors support different network protocols
- A service contains a container, which handles internal Servlets
- Connectors communicate with containers through ServletRequest and ServletResponse objects.

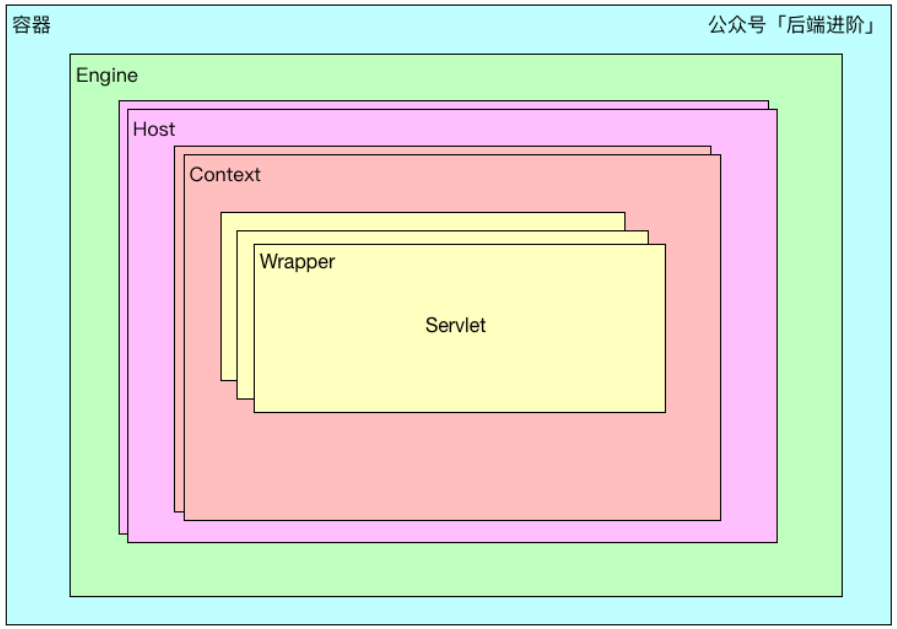
Connector

- Connector is responsible for encapsulating all kinds of network protocols, shielding the details of network connection and IO processing, and passing the processed Request object to container
- Tomcat encapsulates the details of processing requests to the ProtocolHandler interface

https://developpaper.com/talking-about-tomcats-architecture-design/

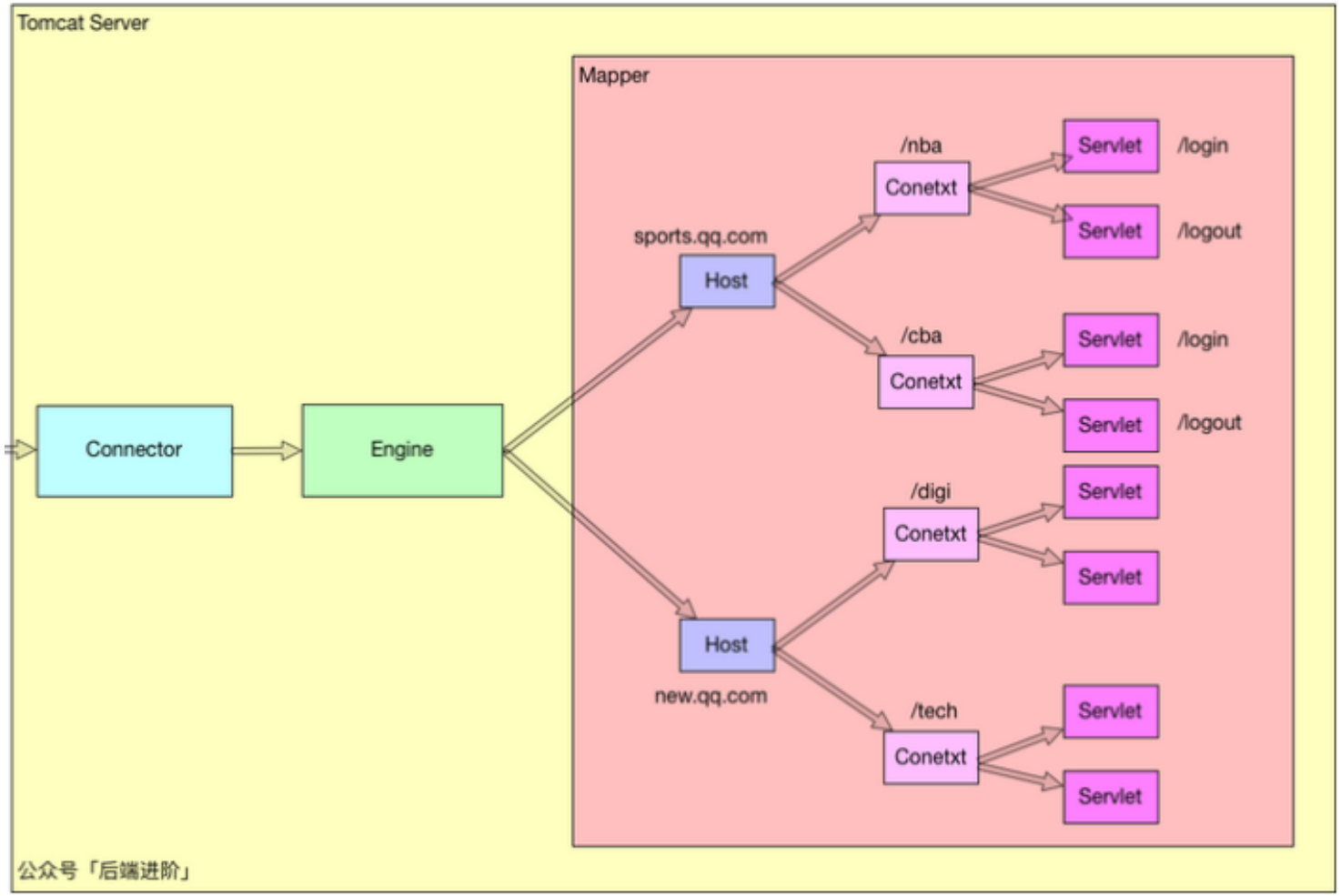


Container

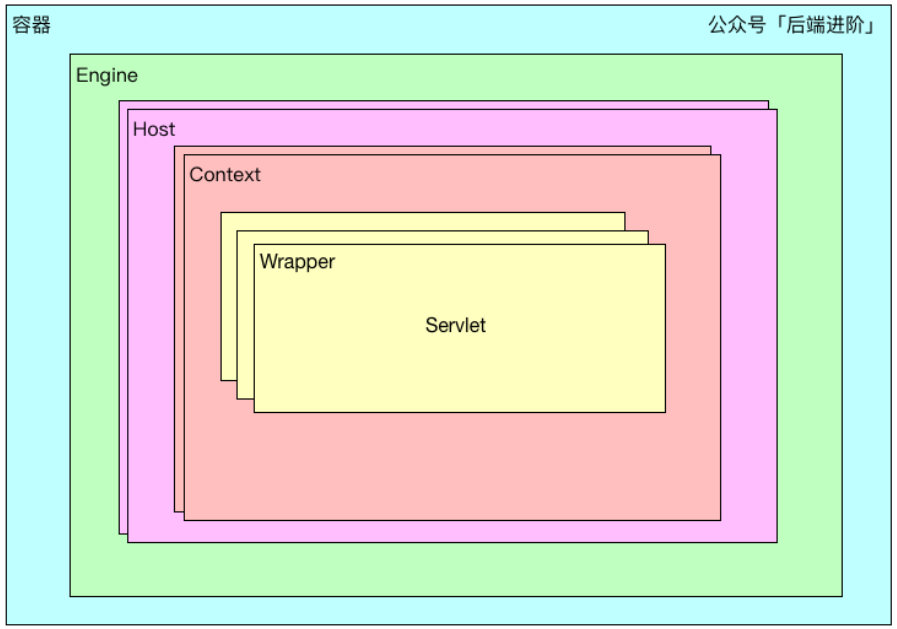


Engine

- The engine of a virtual host.
- A Tomcat Server has only one engine.
- All requests of the connector are handed over to the engine, then to the corresponding virtual host

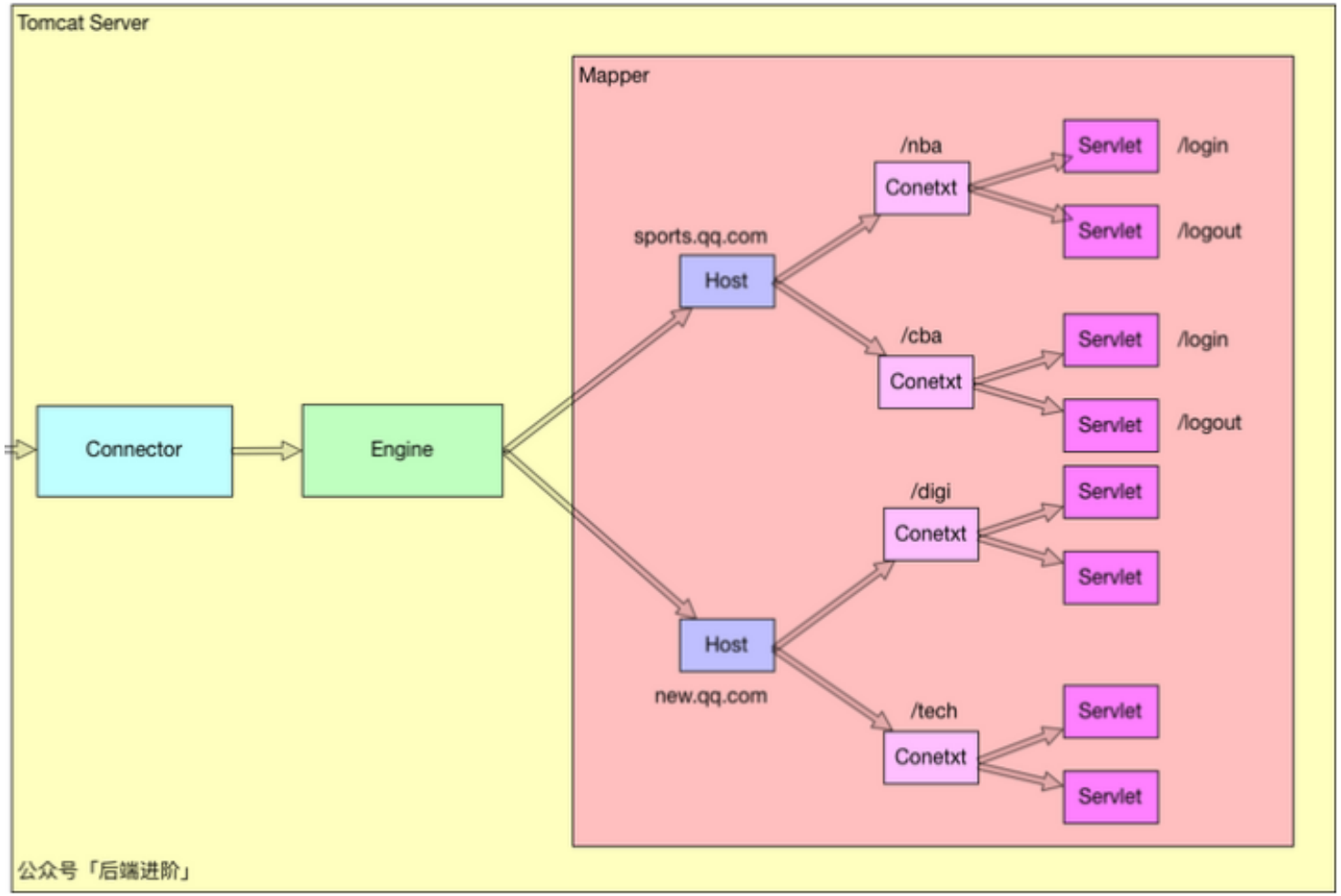


Container

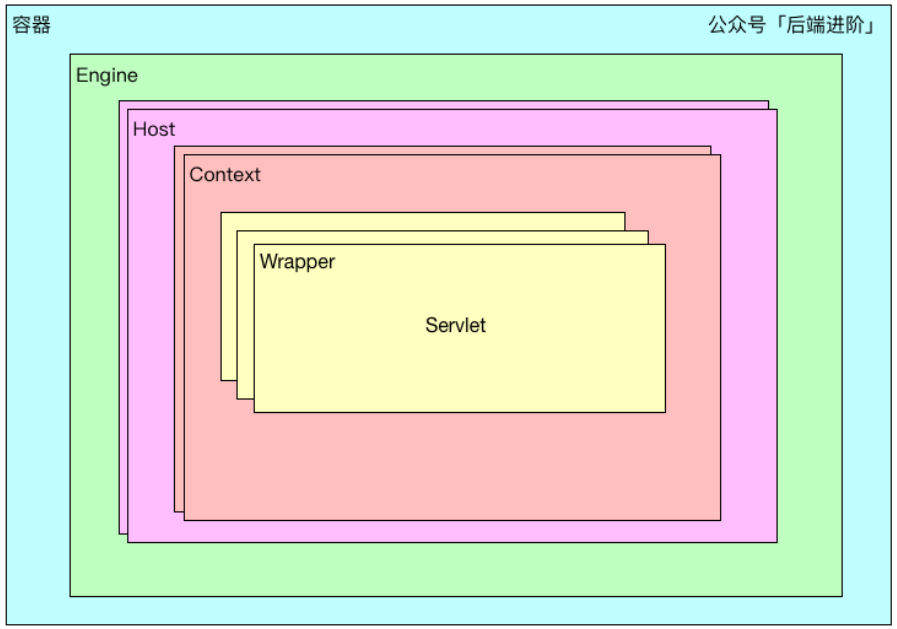


Host

- A virtual host.
- A container can have more than one virtual host.
- Each host has its own domain name.

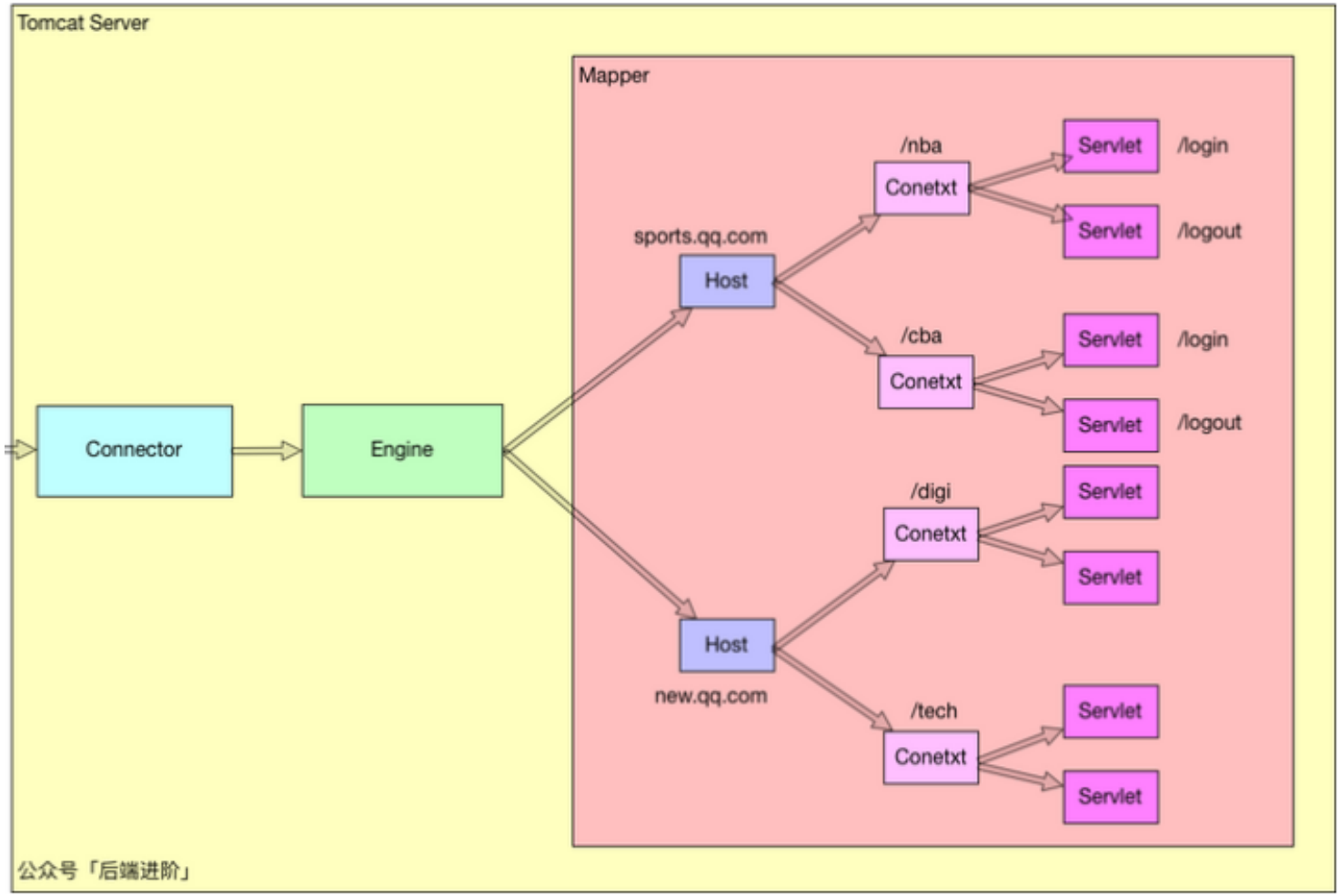


Container

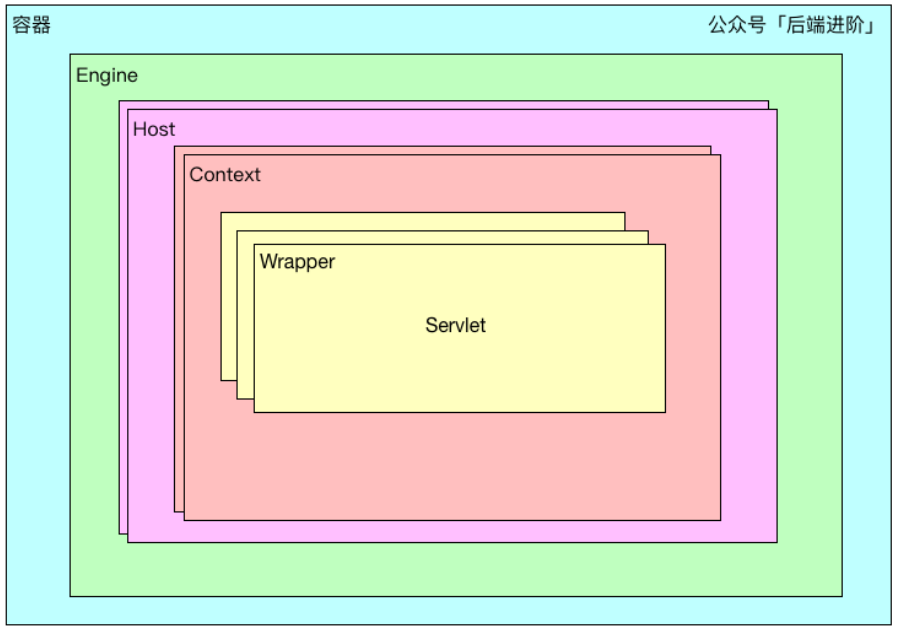


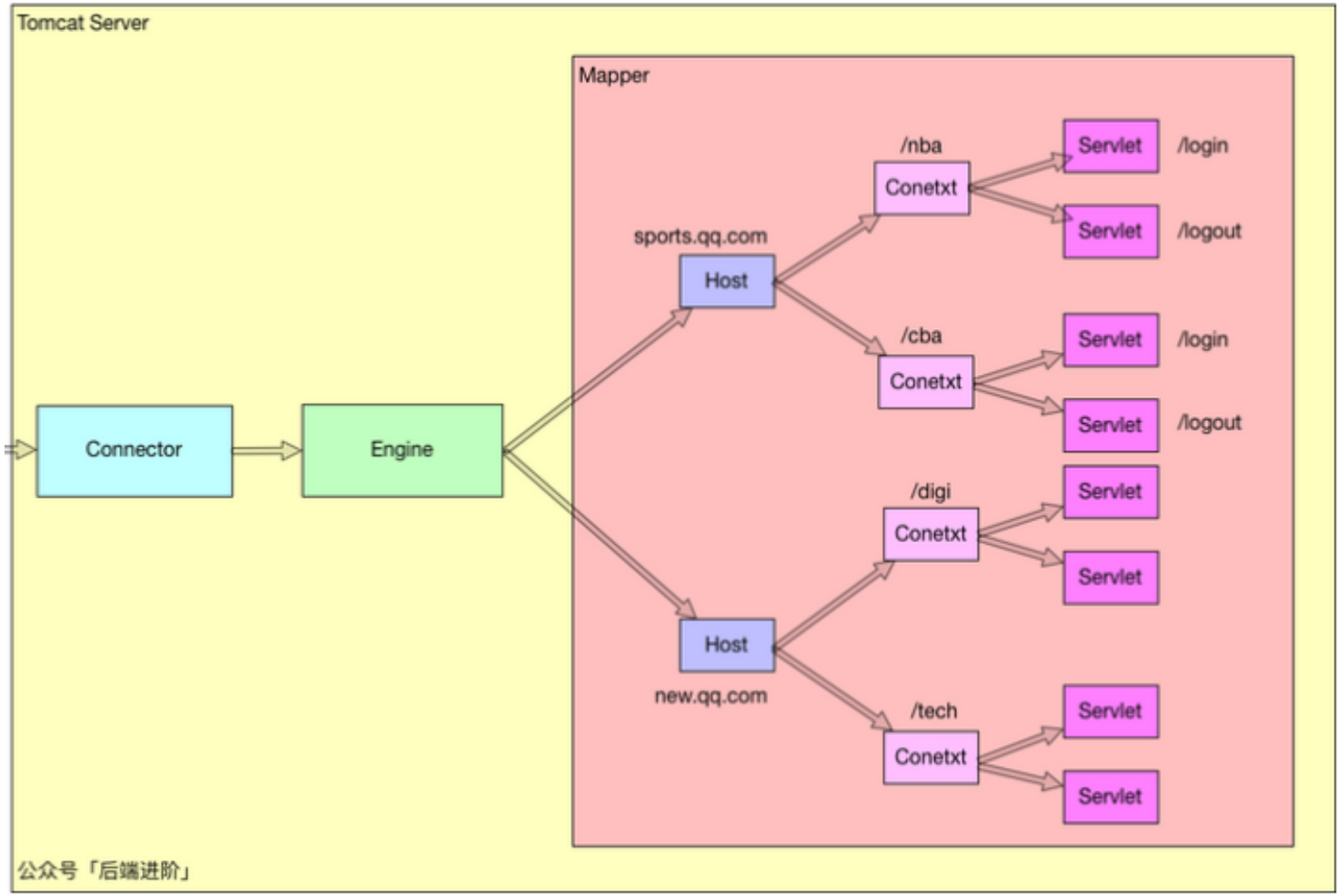
Context

- Represents an application
- A virtual host can have multiple applications
- Each application can configure multiple servlets..



Container

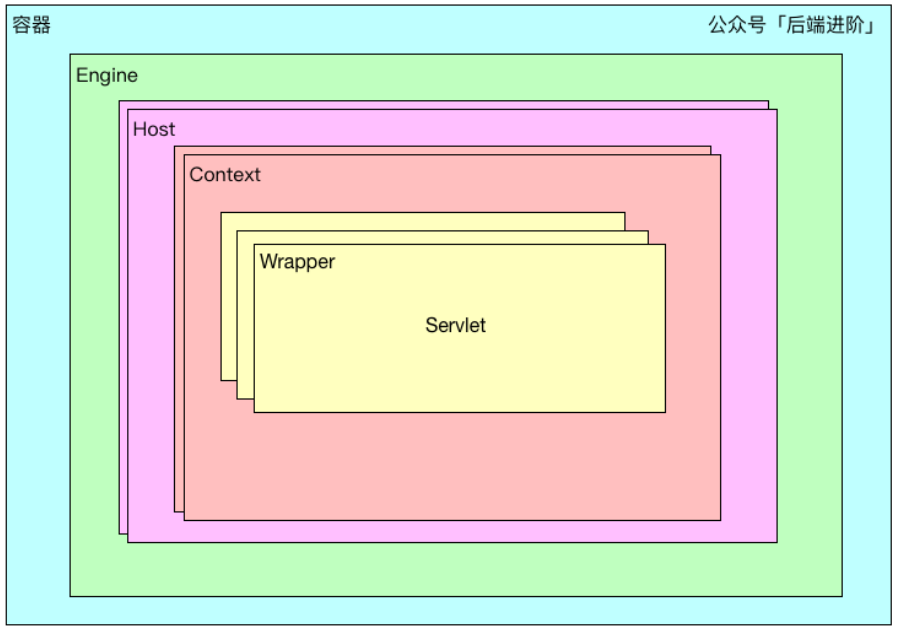


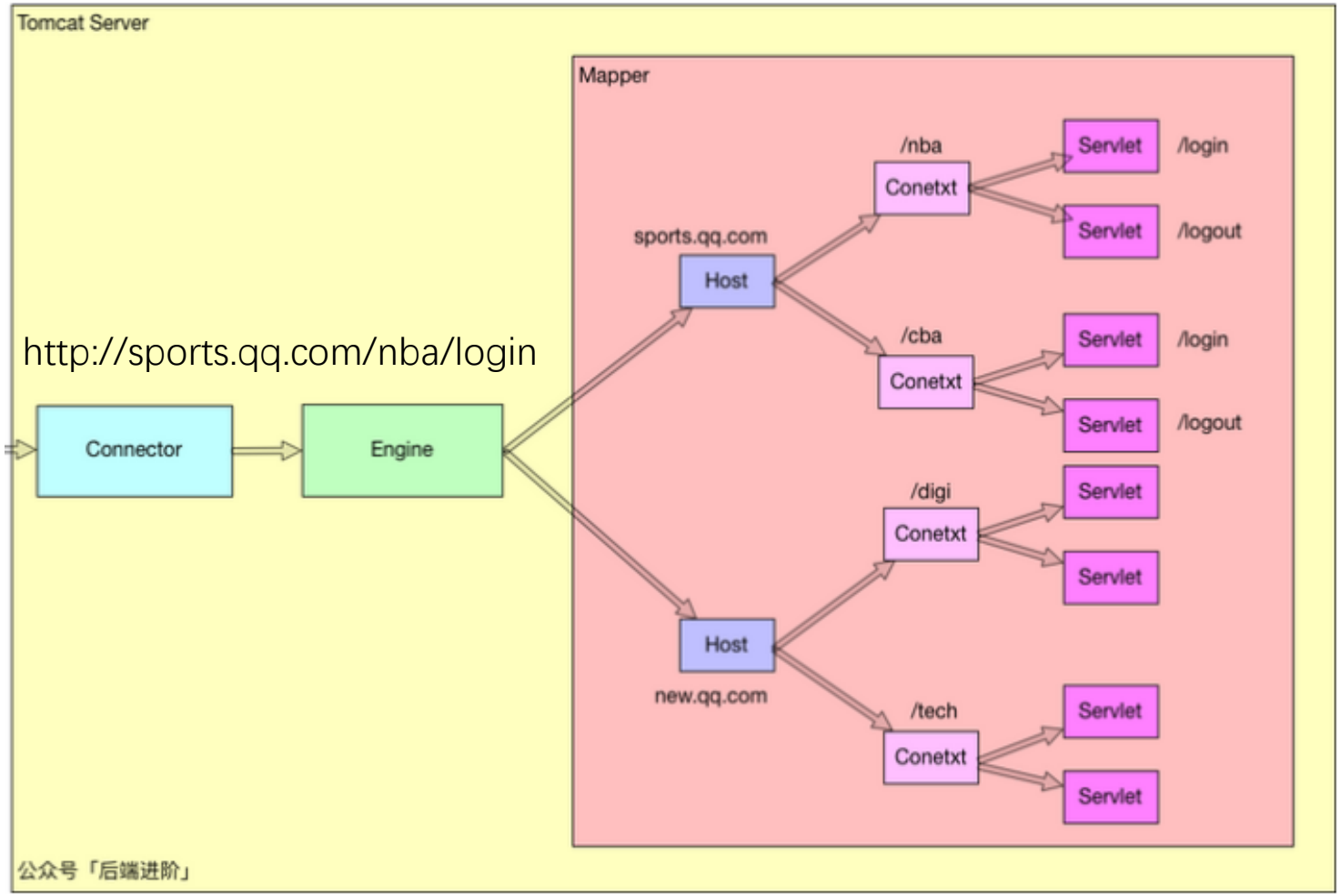


Wrapper

- Represents an individual servlets

Container



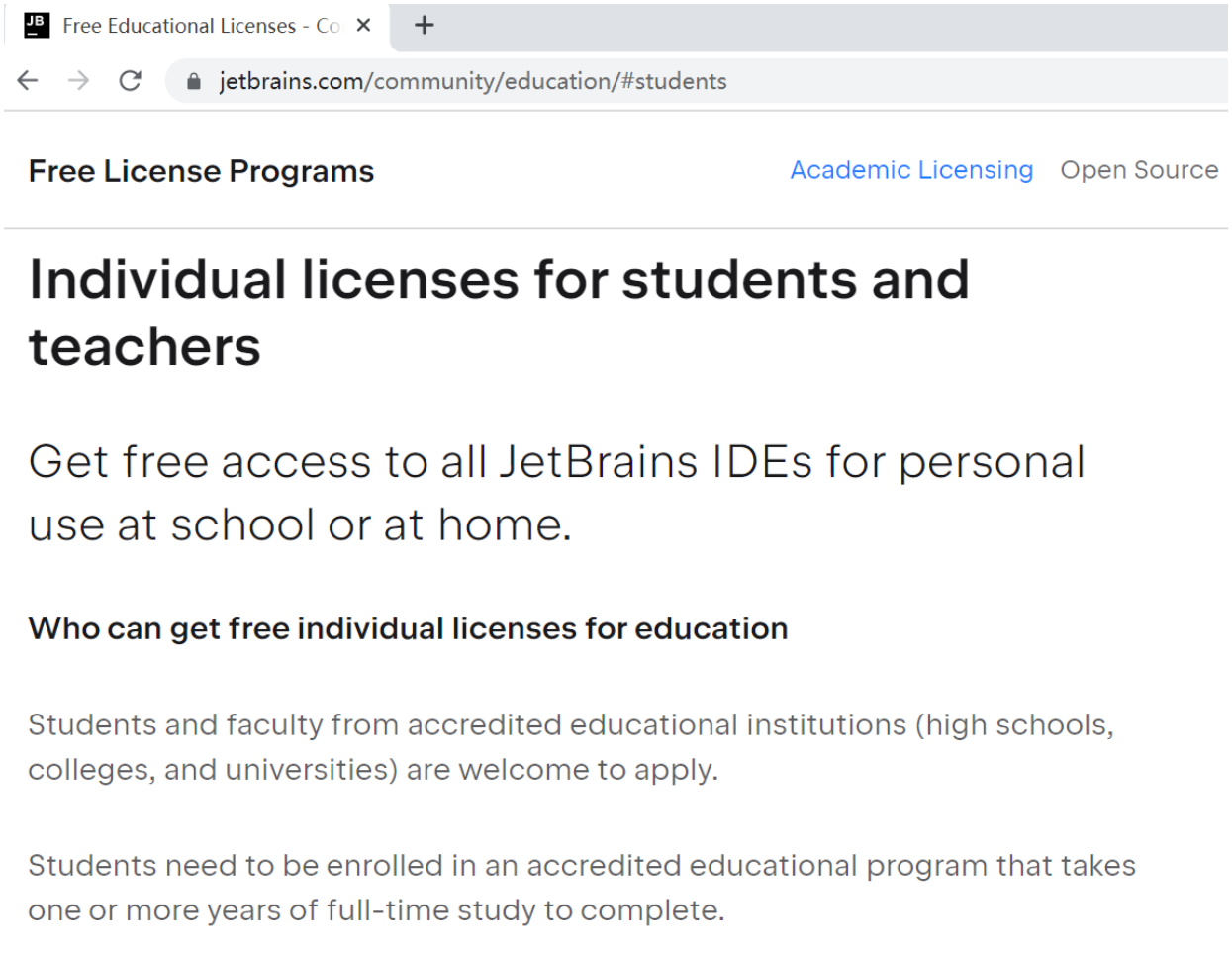


Mapper

- Locate the host and the context given the URL
- Locate the servlet using web.xml

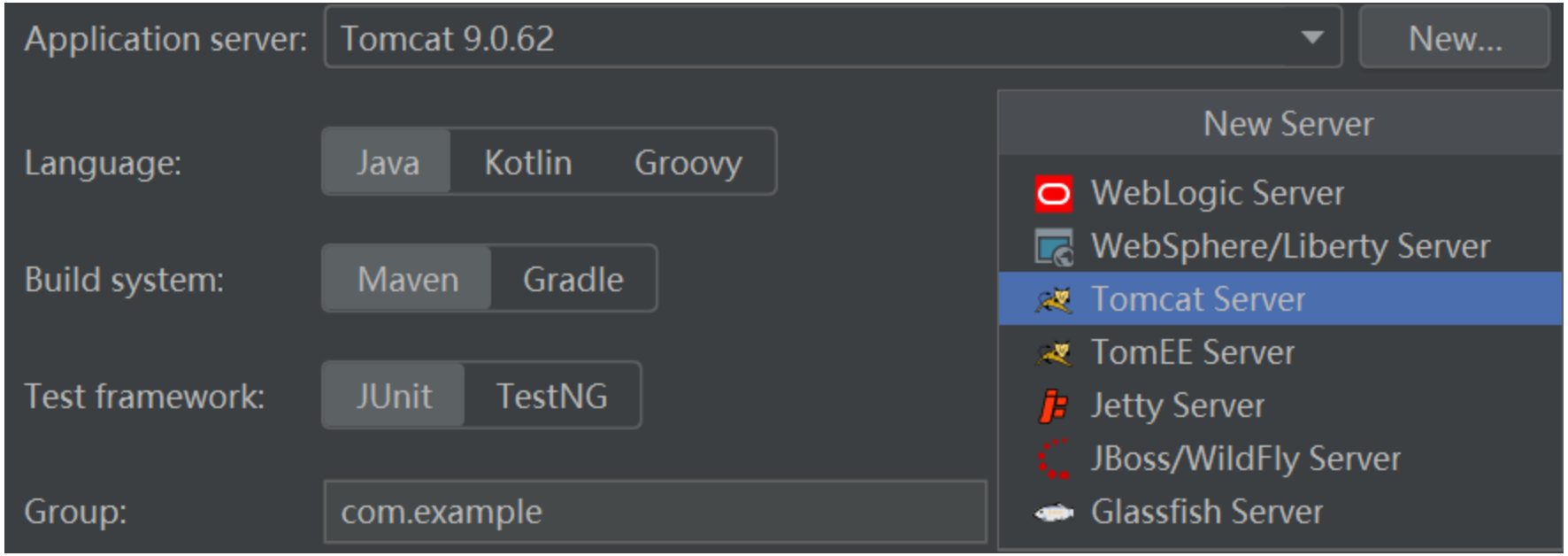
Working with Tomcat in IntelliJ IDEA

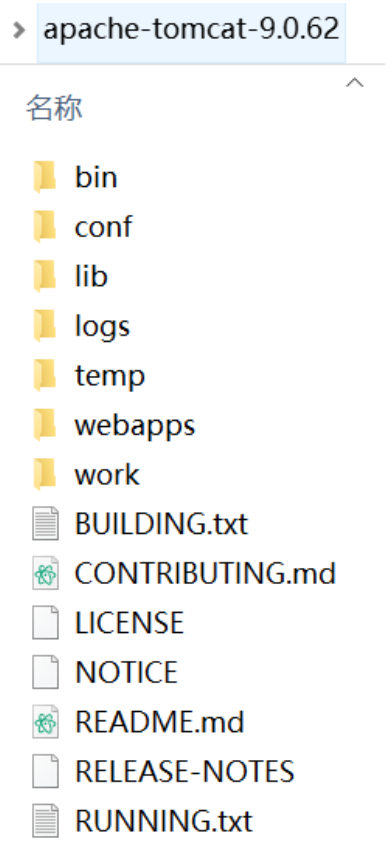
- Java EE is available in IntelliJ Ultimate (instead of IntelliJ Community)
- You may apply for a free individual license as students



Working with Tomcat in IntelliJ IDEA

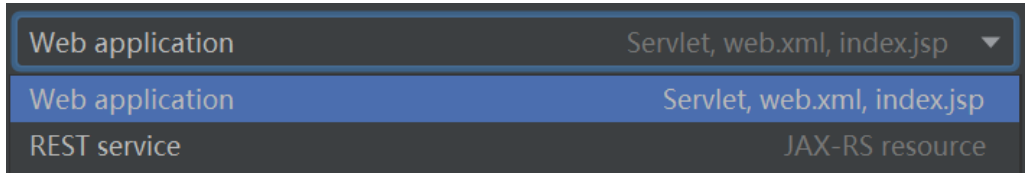
Download tomcat (.zip) and install (unzip) it



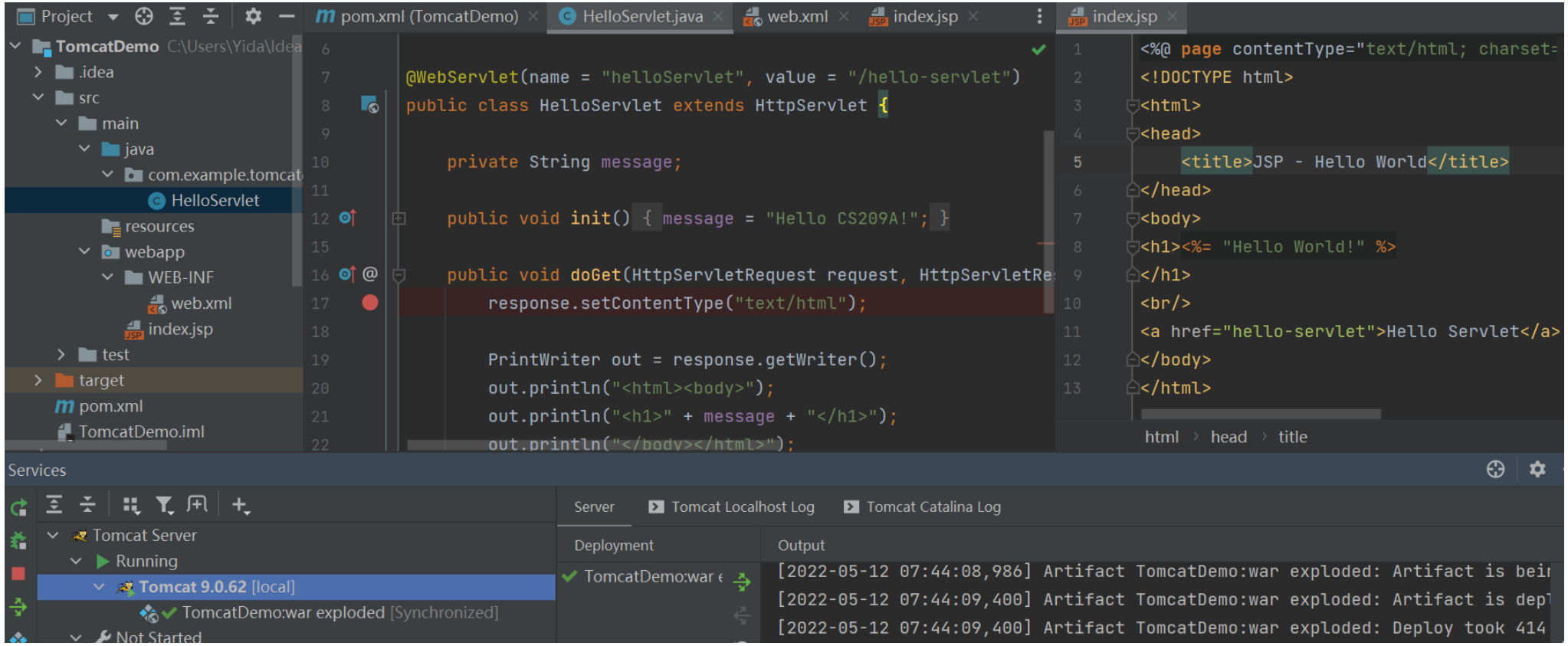


Then specify the installation path in IDEA. Done!

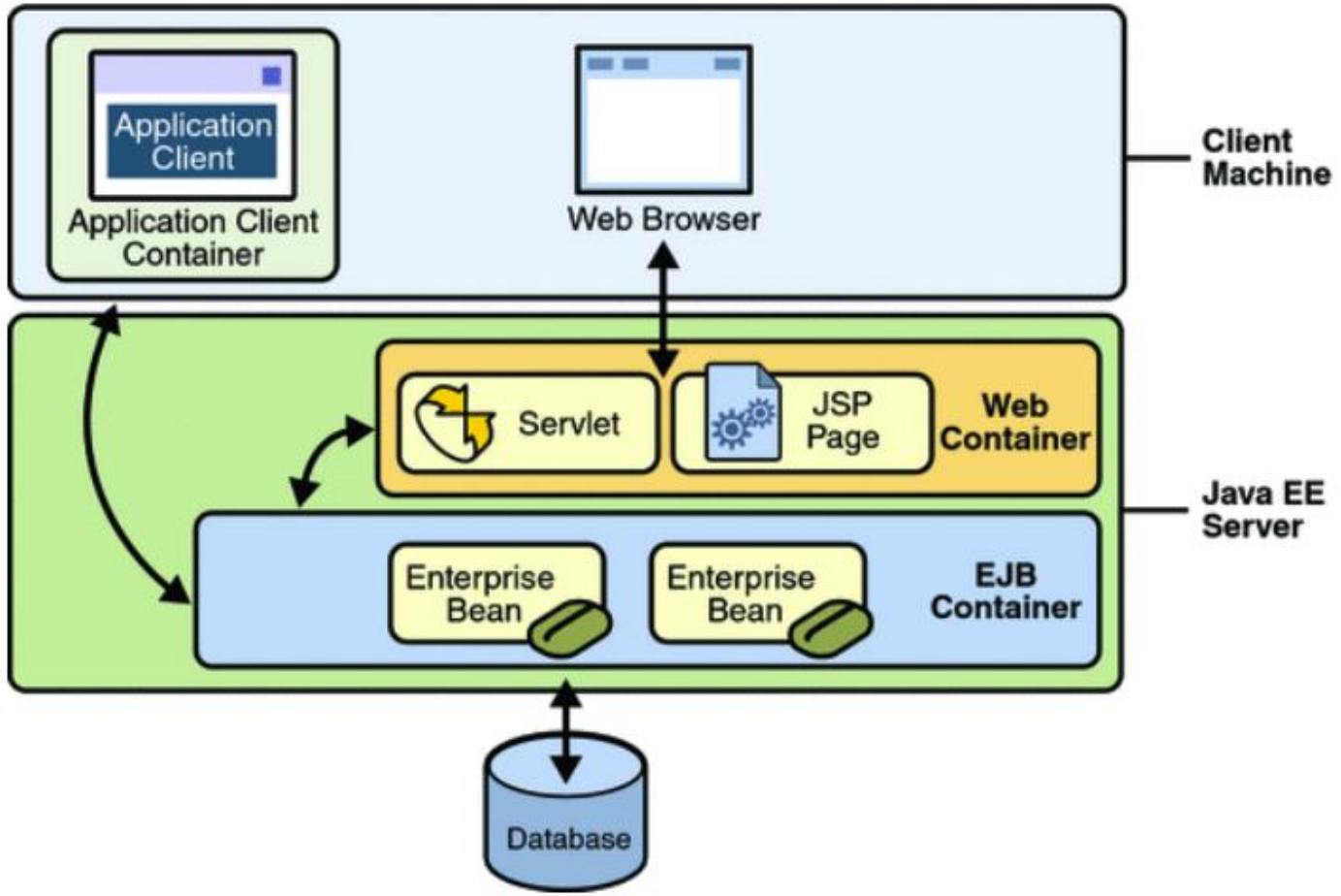
To put it altogether……

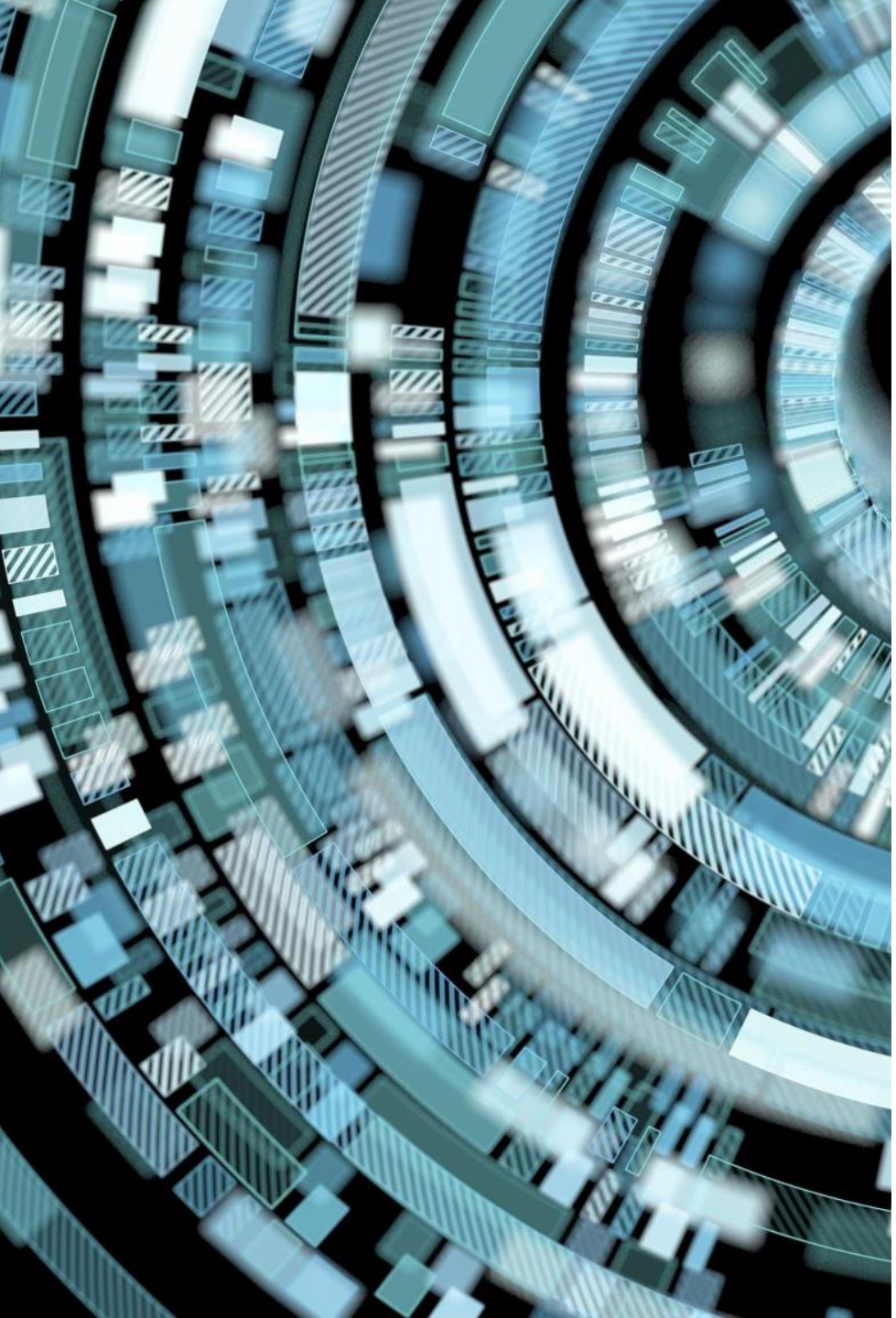


localhost:8080/TomcatDemo/hello-servlet



What we have learned so far ……





Lecture 12

- Web Development Overview
- Java EE
- Servlet & Containers
- JDBC & JPA

Recall Data Persistence (数据持久化)

- Objects created in Java programs live in memory; they are removed by the garbage collector once they are not used anymore
- What if we want to persist the objects?

VS

File Systems vs Database

File system stores unstructured, unrelated data. Better used when:

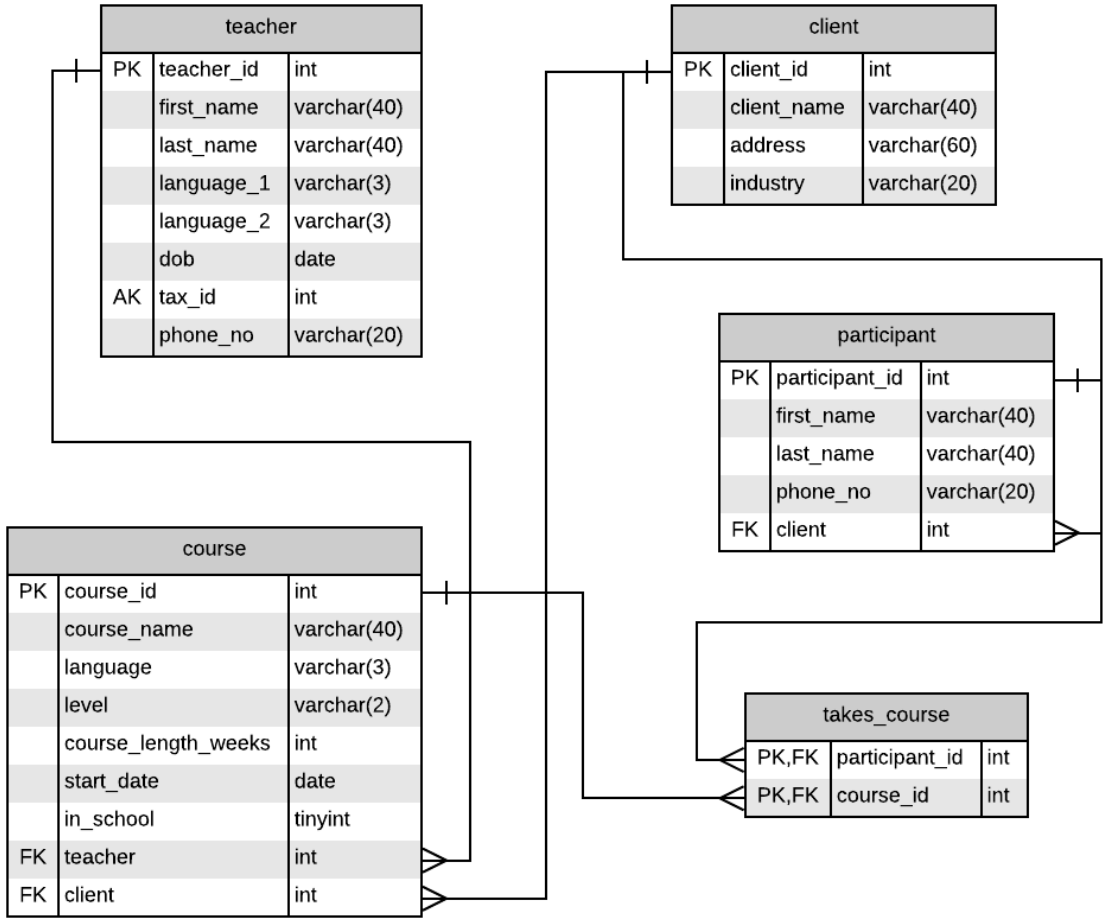
- You like to use version control on your data (a nightmare with dbs)
- You have big chunks of data that grow frequently (e.g., logfiles)
- You want other apps to access your data without API (e.g., text editors)
- You want to store lots of binary content (e.g., pictures, mp3s)

Database stores related, structured data in an efficient manner for insert, update and/or retrieval. Better used when:

- You want to store many rows with the exact same structure
- You need lightning-fast lookup and query processing
- You need to support multiple users andatomic transactions (data safety)

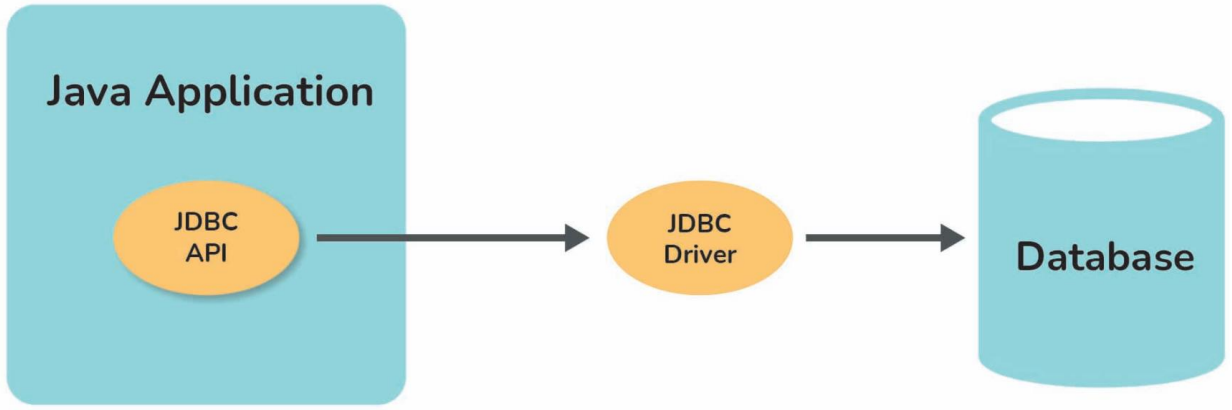
Relational Database

- A relational database organizes data into rows and columns, which collectively form a table.
- Data is typically structured across multiple tables, which can be joined together via a primary key or a foreign key.
- These unique identifiers demonstrate the different relationships which exist between tables



JDBC (Java Database Connectivity)

- To store, organize and retrieve data, most applications use relational databases.
- Java EE applications access relational databases through the JDBC API
- JDBC classes are contained in the `java.sql` and `javax.sql` packages



JDBC drivers

- Client-side adapters (installed on the client machine, not on the server)
- Convert requests from Java programs to a protocol that the DBMS can understand.

Example of Using JDBC https://docs.oracle.com/javase/tutorial/jdbc/TOC.html

```
public Connection getConnection() throws SQLException {

    Connection conn = null;
    Properties connectionProps = new Properties();
    connectionProps.put("user", this.userName);
    connectionProps.put("password", this.password);

    if (this.dbms.equals("mysql")) {
        conn = DriverManager.getConnection(
                    "jdbc:" + this.dbms + "://" +
                    this.serverName +
                    ":" + this.portNumber + "/",
                    connectionProps);
    } else if (this.dbms.equals("derby")) {
        conn = DriverManager.getConnection(
                    "jdbc:" + this.dbms + ":" +
                    this.dbName +
                    ";create=true",
                    connectionProps);
    }
    System.out.println("Connected to database");
    return conn;
}
```

```
public static void viewTable(Connection con) throws SQLException {
  String query = "select COF_NAME, SUP_ID, PRICE, SALES, TOTAL from COFFEES";
  try (Statement stmt = con.createStatement()) {
    ResultSet rs = stmt.executeQuery(query);
    while (rs.next()) {
      String coffeeName = rs.getString("COF_NAME");
      int supplierID = rs.getInt("SUP_ID");
      float price = rs.getFloat("PRICE");
      int sales = rs.getInt("SALES");
      int total = rs.getInt("TOTAL");
      System.out.println(coffeeName + ", " + supplierID + ", " + price +
                         ", " + sales + ", " + total);
    }
  } catch (SQLException e) {
    JDBCTutorialUtilities.printSQLException(e);
  }
}
```

One needs to write SQL queries and manually map between Java object's data and relational DB, which can be complicated

Object-Relational Mapping (ORM)

Technical difficulties of matching the relational model (DB) and the object model (Java)

Granularity	The object model has various levels of granularity but a database table has only two, tables and columns, for example you could have two classes Person and Address but only one table that contains both this information.
Inheritance	objects have the ability to inherit but database tables do not.
Identity	Databases use a primary key to identify a row but Java uses both object identity (==) and equality (equals)
Associations	In java you use references to associate objects and they can be bi-directional but in databases we use a foreign key which are not directional.
Data Navigation	In Java you use the object graph to walk the associations, for example a Person object may contain references to an Address Object which in turn has references to a PostCode object, in order to get to the PostCode object you have walk both Person and Address objects. Databases use SQL joins, which joins tables together to retrieve data.

http://www.datadisk.co.uk/html_docs/java_persistence/persistence_1.html

Object-Relational Mapping (ORM)

- ORM techniques/libraries let us query and manipulate data from a database using an object-oriented paradigm
- We don't use SQL anymore; we interact directly with Java object

```
book_list = new List();
sql = "SELECT book FROM library WHERE author = 'Linus'";
data = query(sql); // I over simplify ...
while (row = data.next())
{
     book = new Book();
     book.setAuthor(row.get('author');
     book_list.add(book);
}
```

With JDBC

```
book_list = BookTable.query(author="Linus");
```

With ORM libraries

https://stackoverflow.com/questions/1279613/what-is-an-orm-how-does-it-work-and-how-should-i-use-one

Java Persistence API (JPA)

- JPA is the Java EE standard specification for ORM.
- Reference implementation
  - Eclipselink (used in GlassFish)
- Other implementations
  - Hibernate
  - Apache OpenJPA

Hibernate

- Hibernate is a framework which is used to develop persistence logic which is independent of Database software.
- In JDBC, to develop persistence logic we deal with primitive types.
- In Hibernate framework, we use Objects to develop persistence logic which are independent of database software

https://www.geeksforgeeks.org/hibernate-architecture/

Hibernate Architecture

- We write the persistence logic to perform some specific operations to the database with the help of Hibernate Configuration file and Hibernate mapping file.
- After that, we create an object of the particular class on which we wrote the persistence logic.
- Our classes, which contain the persistence logic, interact with the Hibernate framework
- Hibernate in turn interacts with JDBC, JNDI, JTA to connect to the database to perform that persistence logic, with the help of JDBC driver

https://www.geeksforgeeks.org/hibernate-architecture/

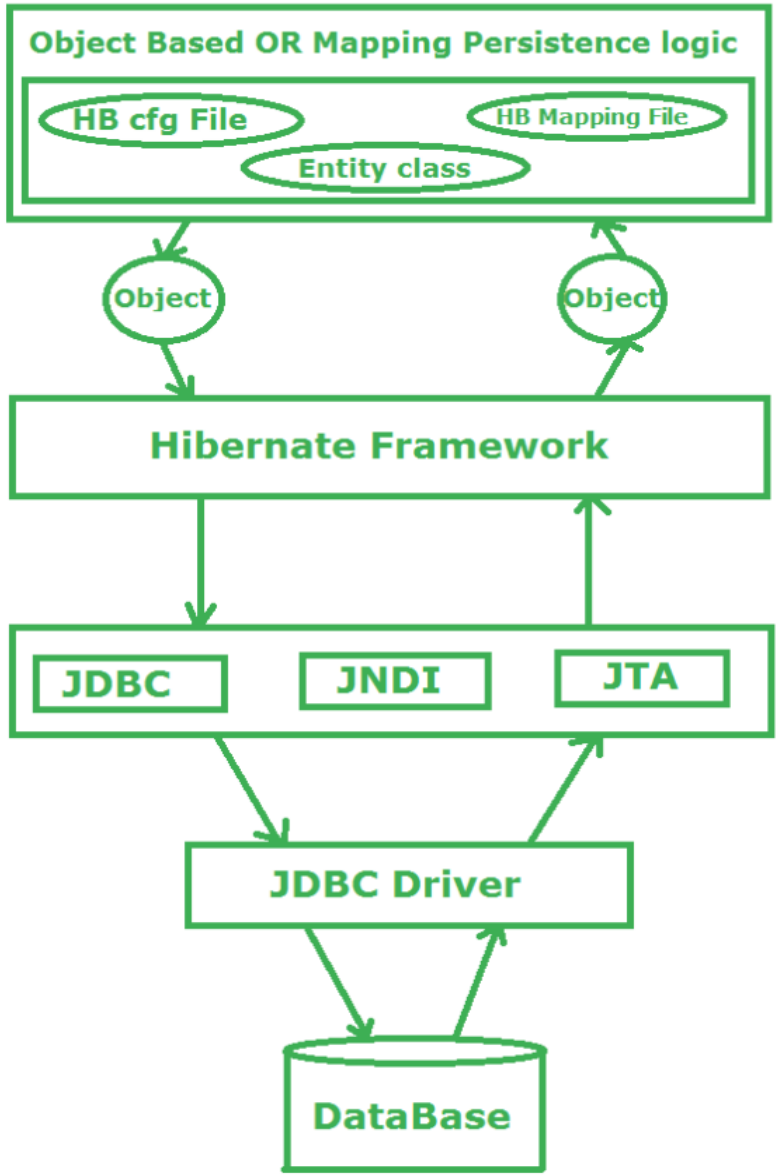


Fig: Working Flow of Hibernate framework to save/retrieve the data from the database in form of Object

An Analogy of JPA vs Hibernate

```
public interface JPA {

        public void insert(Object obj);

        public void update(Object obj);

        public void delete(Object obj);

        public Object select();

}
```

```
public class Hibernate implements JPA {

        public void insert(Object obj) {
       //Persistence code
        }

        public void update(Object obj) {
           //Persistence code

        }

        public void delete(Object obj) {
          //Persistence code

        }
```

http://tothought.com/post/2

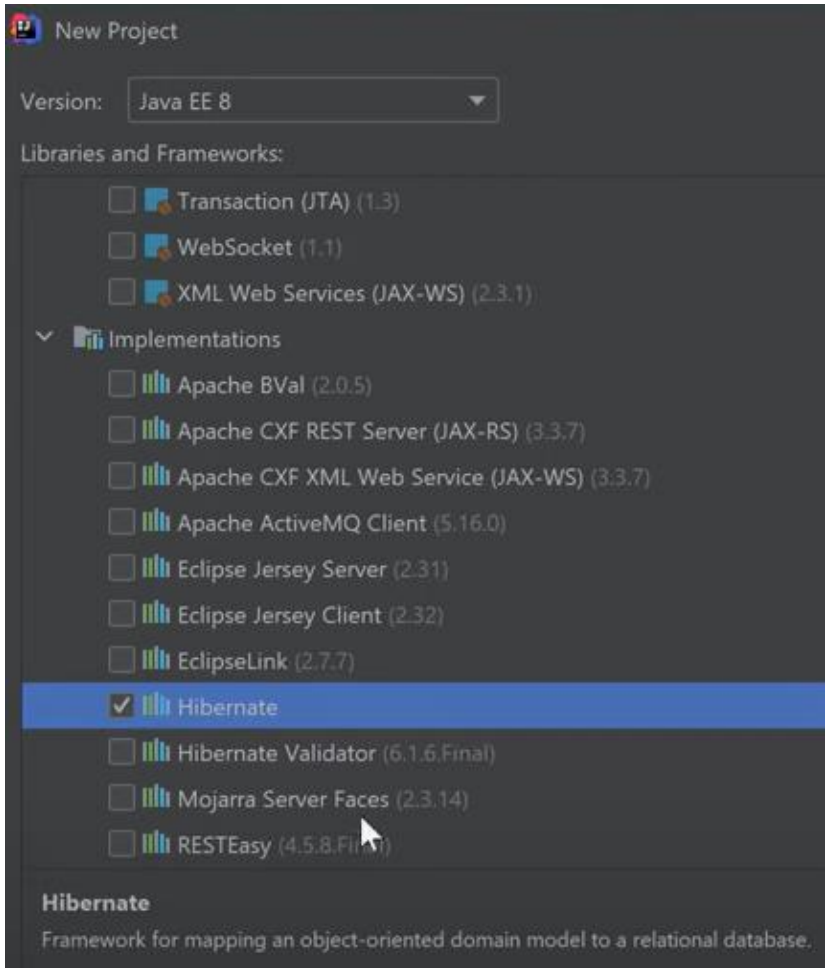
An Analogy of JPA vs Hibernate

```
public class MyApplication {

        public static JPA jpa = new Hibernate();

        public static void main(String[] args) {
                Object object = new Object();
                jpa.insert(object);  //writes to DB
        }
}
```

We could switch to other JPA implementations easily

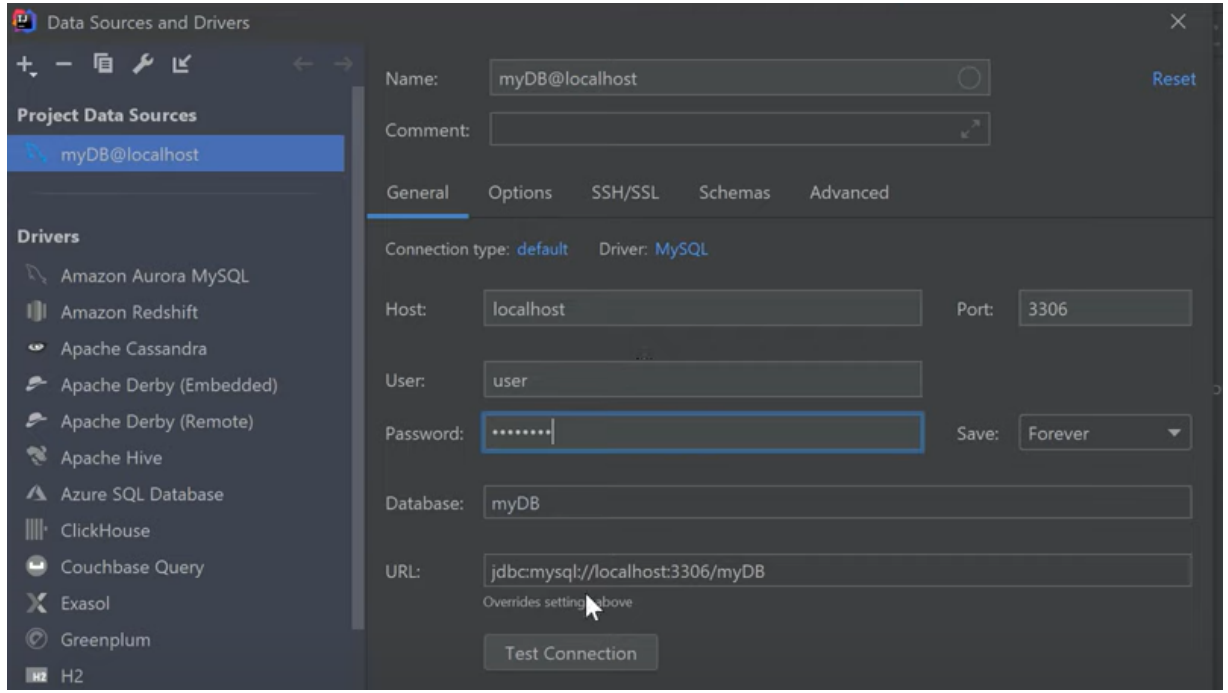
http://tothought.com/post/2

Working with Hibernate in IntelliJ IDEA

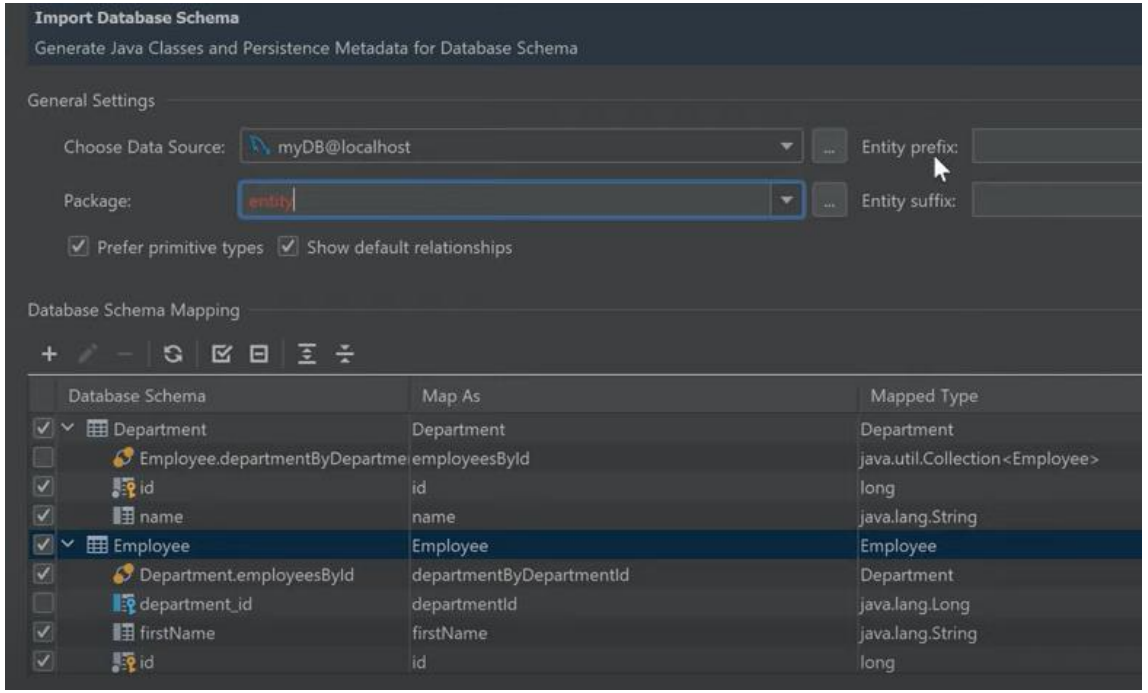



```
<dependencies>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.22</version>
    </dependency>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>5.4.25.Final</version>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
```

Working with Hibernate in IntelliJ IDEA



Connect to database



Configuring mappings between objects and db table & columns

Working with Hibernate in IntelliJ IDEA

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/
             version="2.2">
    <persistence-unit name="default">

        <class>entity.Department</class>
        <class>entity.Employee</class>
        <properties>
            <property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/myDB"/>
            <property name="hibernate.connection.driver_class" value="com.mysql.cj.jdbc.Driver"/>
            <property name="hibernate.connection.user" value="user"/>
            <property name="hibernate.connection.password" value="password"/>
            <property name="hibernate.show_sql" value="true"/>
        </properties>
    </persistence-unit>
</persistence>
```

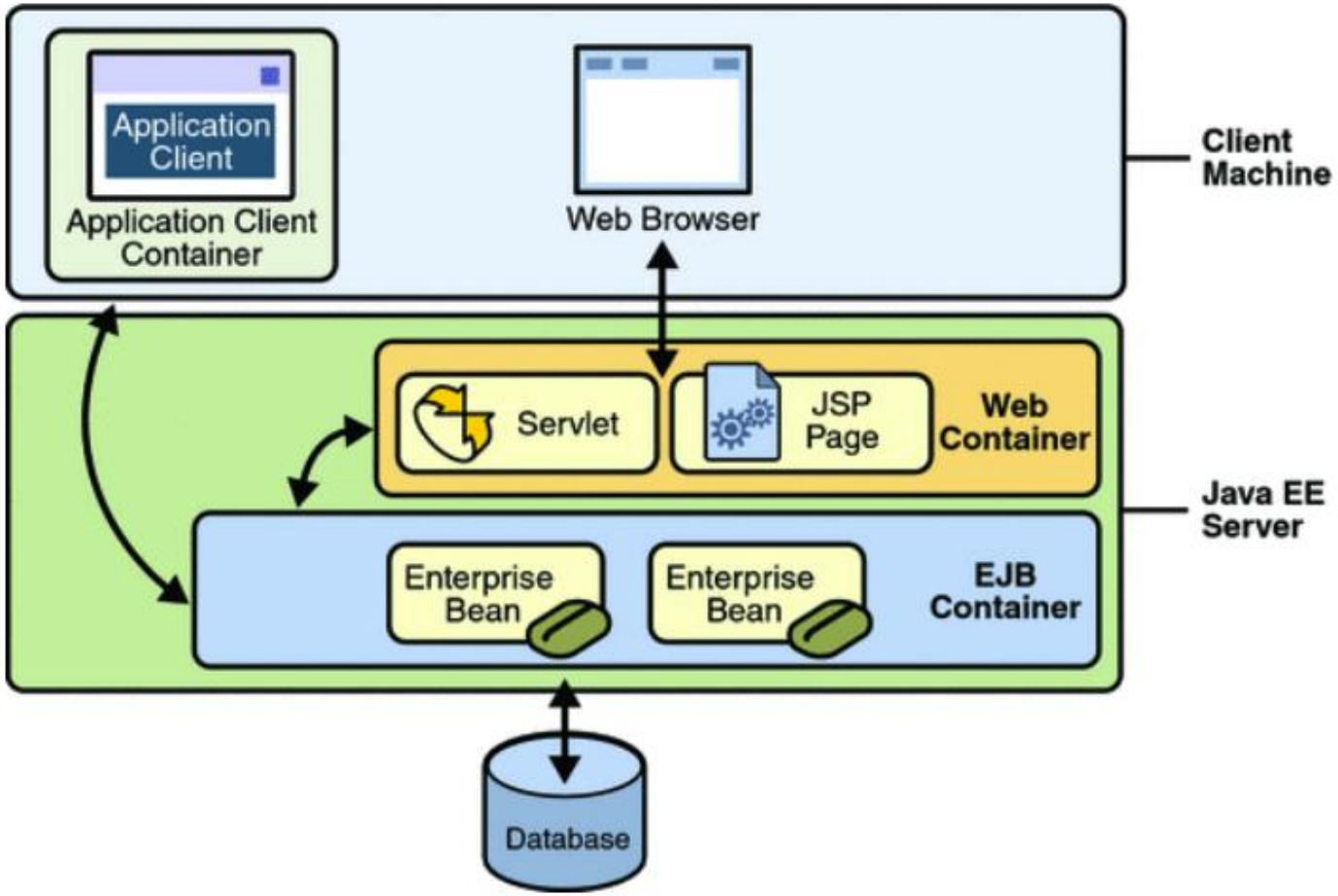
Hibernate configurations

Main code to use JPA APIs and
OO syntax to work with database

```java
public class Main {
    public static void main(String[] args) {
        EntityManagerFactory entityManagerFactory = Persistence.createEntityManagerFactory( persi
        EntityManager entityManager = entityManagerFactory.createEntityManager();
        EntityTransaction transaction = entityManager.getTransaction();

        try {
            transaction.begin();

            Employee dalia = new Employee();
            dalia.setId(6);
            dalia.setFirstName("Dalia");
            dalia.setLastName("Abo Sheasha");
            entityManager.persist(dalia);

            transaction.commit();
        } finally {
            if (transaction.isActive()) {
```

```
Main
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platfor
Hibernate: insert into Employee (department_id, firstName, lastName, id) values (?, ?, ?, ?)
Jan 25, 2021 5:33:48 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionPr
INFO: HHH10001008: Cleaning up connection pool [jdbc:mysql://localhost:3306/myDB]
```

Summary


Application Client
Application Client Container
Web Browser
Client Machine
Servlet
JSP Page
Web Container
Java EE Server
Enterprise Bean
Enterprise Bean
EJB Container
Database


Rod Johnson
Rod Johnson is an enterprise Java architect specializing in scalable web applications. He has worked with both Java and J2EE since their release, and he is a member of JSR 154 Expert Group defining the Servlet 2.4 specification.
expert one-on-one
J2EE™ Design and Development
wrox
Updates, source code, and Wrox technical support at www.wrox.com

Next Lecture

- The Spring Framework
- Spring Boot