01_Introduction

June 18, 2024

1 What is Servelets?

Servlets are Java classes which service HTTP requests and implement the `javax.servlet.Servlet` interface. Web application developers typically write servlets that extend `javax.servlet.http.HttpServlet`, an abstract class that implements the `Servlet` interface and is specially designed to handle HTTP requests.

A servlet is basically a small Java program that runs within a web server. The servlet takes an HTTP request from a browser, generates dynamic content like querying a database, and provides an HTTP response back to the browser.

Servlets are a key component of Java-based web applications and are a server-side technology. They provide a powerful, efficient, portable, and secure alternative to CGI for generating dynamic web content.

2 Why to Learn Servlet?

Learning Servlets can be beneficial for several reasons:

1. **Web Development:** Servlets are a fundamental technology for developing Java-based web applications. They provide server-side processing, creating dynamic web content.
2. **Performance:** Servlets offer better performance than traditional CGI scripts. They use a thread-based model rather than a process-based model, making them more efficient for handling multiple simultaneous requests.
3. **Integration:** Servlets can be integrated with other Java technologies like JSP, JDBC, and EJB, making them part of a powerful suite of tools for building complex enterprise applications.
4. **Portability:** Servlets are platform-independent. They can run on any servlet container that complies with the Java Servlet API, making your web applications portable across a wide range of server environments.
5. **Career Opportunities:** Knowledge of Servlets and JSP is often a requirement for Java developer positions, especially those involving web application development.
6. **Foundation for Other Technologies:** Learning Servlets provides a foundation for learning more advanced Java web technologies and frameworks like Spring and JSF.
7. **Community and Support:** Servlets are part of the Java ecosystem, which has a large, active community and a wealth of resources for learning and problem-solving.

3 Applications of Servlet?

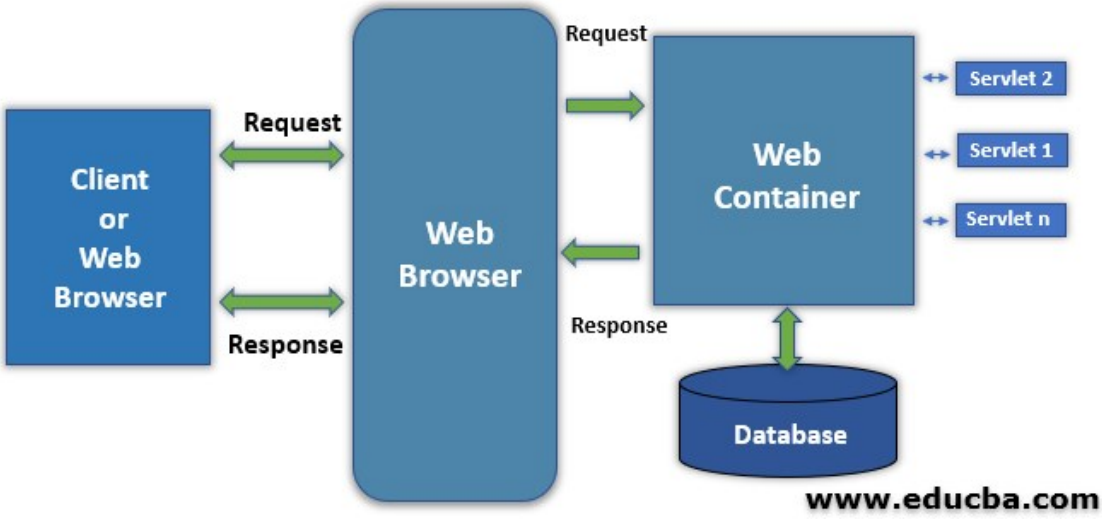
Servlets are used in a variety of ways to create dynamic web content. Here are some common applications of Servlets:

1. **Web Applications:** Servlets are commonly used to create web applications that interact with users through browsers. They can generate HTML, XML, JSON, or any other type of content that a client can handle.
2. **Form Processing:** Servlets can handle form data submitted by users, performing tasks such as validating user input, storing data in a database, and generating a response.
3. **Database Interaction:** Servlets can interact with databases using JDBC or other data access technologies. They can perform CRUD (Create, Read, Update, Delete) operations and implement complex business logic.
4. **Session Tracking:** Servlets can keep track of user sessions using several methods, including cookies, URL rewriting, and the HttpSession object.
5. **Collaboration Systems:** Servlets can be used to build web-based collaboration systems, such as online chat, forums, and social networking sites.
6. **Enterprise Applications:** In combination with other Java technologies like JSP, EJB, and JPA, Servlets can be used to build complex enterprise applications.
7. **Web Services:** Servlets can be used to implement SOAP or RESTful web services, allowing different applications to communicate over the network.
8. **Content Management Systems:** Servlets can be used to build content management systems, where users can create, edit, and publish web content.
9. **E-commerce Applications:** Servlets are often used in the development of e-commerce applications, where they can handle tasks like product catalog management, shopping cart management, and order processing.

4 Servlets Architecture?

[ ]:
```
from IPython.display import Image
Image(filename='Images/Servlet-Architecture-2.jpg')
```

[ ]:



The architecture of Servlets involves interaction between a web client (usually a web browser) and the web server. Here's a step-by-step explanation of how it works:

1. **Client Request:** The client sends an HTTP request to the web server.
2. **HTTP Request to Servlet Container:** The web server receives the HTTP request and forwards it to the Servlet container. The Servlet container could be a part of the web server or it could be a separate software.
3. **Load and Instantiate Servlet:** If the Servlet is not loaded, the Servlet container loads the Servlet class and creates an instance of it. If the Servlet is already loaded, it skips to the next step.
4. **Initialize Servlet:** If the Servlet is newly loaded, the Servlet container calls the `init()` method to initialize the Servlet.
5. **Service Request:** The Servlet container creates `HttpServletRequest` and `HttpServletResponse` objects, and then calls the `service()` method of the Servlet, passing these objects as arguments. The `service()` method then calls the appropriate method (`doGet()`, `doPost()`, etc.) based on the HTTP method of the request.
6. **Generate Response:** The Servlet generates the response, which is captured in the `HttpServletResponse` object.
7. **Send Response:** The Servlet container sends the response back to the client via the web server.
8. **Remove Servlet:** If the Servlet is no longer needed, the Servlet container calls the `destroy()` method to take the Servlet out of service.

This architecture allows Servlets to handle multiple requests concurrently, with each request being processed in a separate thread.

5 Servlets Packages?

Servlets in Java use a number of packages for their operation. The most commonly used are:

1. **javax.servlet:** This package contains a number of classes and interfaces that describe and define the contracts between a servlet class and the runtime environment provided for an instance of such a class by a conforming servlet container.
2. **javax.servlet.http:** This package provides interfaces and classes for writing servlets that use the HTTP protocol. The classes in this package are responsible for handling HTTP-specific services such as requests, responses, cookies, and sessions.
3. **javax.servlet.annotation:** This package provides annotation types that can be used to annotate a servlet class to specify metadata about the servlet. For example, the `@WebServlet` annotation is used to define a servlet component in a web application.
4. **javax.servlet.descriptor:** This package provides types that correspond to elements in the `web.xml` deployment descriptor.

Here's an example of how these packages might be used in a servlet:

[ ]:
```java
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;

@WebServlet("/example")
public class ExampleServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<h1>Hello, World!</h1>");
    }
}
```

In this example, the `javax.servlet.http.HttpServlet` class is extended to create an HTTP servlet, the `javax.servlet.annotation.WebServlet` annotation is used to specify the URL pattern for the servlet, and the `javax.servlet.http.HttpServletRequest` and `javax.servlet.http.HttpServletResponse` classes are used to handle the HTTP request and response.

6 Setting up Web Server — Tomcat?

Here are the steps to install Apache Tomcat on Ubuntu:

1. **Update your system:** Always start by updating your system to the latest packages.

[ ]:
```
sudo apt update
sudo apt upgrade
```

2. **Install Java Development Kit (JDK):** Tomcat requires Java to run. You can install OpenJDK with the following command:

```
sudo apt install default-jdk
```

3. **Create a Tomcat User:** For security purposes, Tomcat should be run as an unprivileged user (i.e., not root). You can create a new user and group that will run the Tomcat service with the following command:

```
sudo groupadd tomcat
sudo useradd -s /bin/false -g tomcat -d /opt/tomcat tomcat
```

4. **Download and Install Tomcat:** Now, download the latest version of Tomcat from the Tomcat Downloads page. As of writing, the latest version is Tomcat 9.0. You can download it to the `/tmp` directory with the following command:

```
cd /tmp
wget https://downloads.apache.org/tomcat/tomcat-9/v9.0.53/bin/apache-tomcat-9.0.
↪53.tar.gz
```

Then, create a directory for Tomcat and extract the archive to it:

```
sudo mkdir /opt/tomcat
sudo tar xzf apache-tomcat-9*.tar.gz -C /opt/tomcat --strip-components=1
```

5. **Update Permissions:** Make the Tomcat user the owner of the Tomcat directory and update the permissions:

```
sudo chown -R tomcat: /opt/tomcat
sudo chmod +x /opt/tomcat/bin/*.sh
```

6. **Create a systemd Service File:** To run Tomcat as a service, you need to create a systemd service file. Open a new service file in your text editor with root privileges:

```
sudo nano /etc/systemd/system/tomcat.service
```

Then, add the following content to the file:

```
[Unit]
Description=Apache Tomcat Web Application Container
After=network.target

[Service]
Type=forking

User=tomcat
Group=tomcat

ExecStart=/opt/tomcat/bin/startup.sh
```

```
ExecStop=/opt/tomcat/bin/shutdown.sh

[Install]
WantedBy=multi-user.target
```

Save and close the file.

7. **Start Tomcat:** Now, you can start the Tomcat service with the following command:

[ ]:
```
sudo systemctl start tomcat
```

To enable the Tomcat service to start on boot, use the following command:

[ ]:
```
sudo systemctl enable tomcat
```

8. **Access Tomcat:** You can now access the Tomcat server by opening your browser and visiting `http://localhost:8080`.

Please note that these instructions are for a basic Tomcat installation which is suitable for development purposes. For a production environment, additional configuration would be required to secure the server.

7 Servlets - Life Cycle?

The lifecycle of a servlet is managed by the servlet container (like Apache Tomcat). The lifecycle consists of the following phases:

1. **Loading and Instantiation:** The servlet container loads the servlet during startup or when the first request is made. The loading of the servlet depends on the configuration in the `web.xml` file. The servlet container creates an instance of the servlet after loading it.
2. **Initialization:** After creating the servlet instance, the servlet container calls the `init()` method and passes an object implementing the `javax.servlet.ServletConfig` interface. This `ServletConfig` object allows the servlet to access name-value initialization parameters from the web.xml file. The `init()` method is called only once throughout the lifecycle of the servlet.
3. **Request Handling:** After initialization, the servlet instance can service client requests. Each client request results in a new thread. The servlet container calls the `service()` method for each request, passing request (`HttpServletRequest`) and response (`HttpServletResponse`) objects. The `service()` method then calls the `doGet()`, `doPost()`, `doPut()`, `doDelete()`, etc. methods as per the client request. The `service()` method is called for each client request, so it's called multiple times.
4. **Removal from Service (Destruction):** If the servlet is no longer needed for servicing client requests, it is removed from service. The servlet container calls the `destroy()` method to relinquish any resources such as file handles, database connections etc. The `destroy()` method, like `init()`, is also called only once throughout the lifecycle of the servlet, i.e., at the end.

Here's a simple representation of the lifecycle:

[ ]:
```
Load Servlet --> Instantiate Servlet --> Initialize Servlet (call init method)
        ^                                       |
        |                                       |
        | <----- Process Request (call service method) <-----
        |
        |
Destroy Servlet (call destroy method) --> Garbage Collection
```

Remember, the servlet container manages the lifecycle of the servlet, creating and initializing the servlet as needed and destroying it when it's no longer needed.

[ ]:
```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloWorld extends HttpServlet {

   private String message;

   public void init() throws ServletException {
      // Do required initialization
      message = "Hello World";
   }

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

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

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

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

8 Compiling a Servlet?

To compile a Servlet, you need to have the Java Development Kit (JDK) and Servlet API jar file in your classpath. The Servlet API jar file is typically provided by your Servlet container (like Apache Tomcat) and can usually be found in the `lib` directory of the container's installation directory.

Here are the steps to compile a Servlet:

1. **Set the classpath:** You need to include the path to the Servlet API jar file in your classpath. If you're using Tomcat, the Servlet API jar file is typically located in the `lib` directory of your Tomcat installation. You can set the classpath in your shell with a command like this:

```
[ ]: export CLASSPATH=/path/to/servlet-api.jar:$CLASSPATH
```

Replace `/path/to/servlet-api.jar` with the actual path to the Servlet API jar file on your system.

2. **Compile the Servlet:** Use the `javac` command to compile your Servlet. If your Servlet source code is in a file named `HelloWorld.java`, you would compile it like this:

```
[ ]: javac HelloWorld.java
```

This will create a `HelloWorld.class` file in the current directory. This is the compiled Servlet that can be loaded by your Servlet container.

Remember, the Servlet class file should be placed in the correct location in the web application directory structure (i.e., under `WEB-INF/classes`), and the web application needs to be properly configured to use the Servlet (usually in the `web.xml` file).

9 Servlet Deployment?

Deploying a servlet involves packaging the servlet class files and any associated resources into a web application archive (WAR) file, and then deploying this WAR file to a servlet container like Apache Tomcat.

Here are the steps to deploy a servlet:

1. **Organize your files:** Servlets should be placed in the `WEB-INF/classes` directory of your web application. If your servlet is in a package, the package structure should be reflected in the directory structure under `WEB-INF/classes`. For example, if your servlet class is `com.example.MyServlet`, the class file should be placed in `WEB-INF/classes/com/example/MyServlet.class`.

2. **Create a web.xml file:** The `web.xml` file is the deployment descriptor for your web application. It should be placed in the `WEB-INF` directory. This file is used to configure your servlets and other components of your web application. A simple `web.xml` for a servlet might look like this:

```
[ ]: <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                           http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
       version="3.1">
       <servlet>
         <servlet-name>MyServlet</servlet-name>
         <servlet-class>com.example.MyServlet</servlet-class>
       </servlet>
```

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

3. **Package your web application:** Your web application should be packaged into a WAR file for deployment. The WAR file is simply a ZIP file with a `.war` extension that contains your web application directory structure and files. You can create a WAR file using the `jar` command like this:

[ ]:
```
jar cvf myapp.war *
```

This command should be run from the root directory of your web application, and will create a `myapp.war` file in the current directory.

4. **Deploy your web application:** The process for deploying the WAR file depends on your servlet container. For Apache Tomcat, you can simply copy the WAR file to the `webapps` directory of your Tomcat installation, and Tomcat will automatically deploy it. Alternatively, you can use the Tomcat Manager web application to deploy your WAR file.

After deployment, your servlet should be accessible at the URL mapped to it in the `web.xml` file. For the example `web.xml` above, if your Tomcat server is running on `localhost` port `8080`, the servlet would be accessible at `http://localhost:8080/myapp/myServlet`.

10 Write note on Tomcat server?

Apache Tomcat, often referred to as Tomcat Server, is an open-source Java Servlet Container developed by the Apache Software Foundation (ASF). It implements several Java EE specifications including Java Servlet, JavaServer Pages (JSP), Java EL, and WebSocket, and provides a "pure Java" HTTP web server environment in which Java code can run.

Key Features of Tomcat Server:

1. **Lightweight and Flexible:** Tomcat is lighter and requires less resources compared to other application servers. It's often used for simple web applications or microservices.
2. **Java Servlet Container:** Tomcat implements the Java Servlet and the JavaServer Pages (JSP) specifications from Oracle, providing an environment for Java code to run in.
3. **Webserver:** Tomcat includes its own HTTP server. It is often used as a standalone server for applications that only require servlet/JSP processing.
4. **Integration:** Tomcat can also be used in conjunction with other web servers like Apache HTTP Server for serving static content, while dynamically generated content is handled by Tomcat.
5. **Tooling:** It provides tools for configuration and management, which can be graphical or command line based. It can also be configured by editing XML configuration files.
6. **Security:** Tomcat includes a realm mechanism that can be configured to support a variety of authentication methods, including SSL and integration with enterprise security systems.

7. **Community Support:** Being open-source, it has a large community of developers who contribute to its development and provide support.
8. **Portability:** Applications developed with Tomcat and the Servlet API are portable to other Servlet/JSP containers.

Tomcat is not a full Java EE server. It doesn't support the full range of Java EE technologies such as Java Message Service (JMS), Enterprise JavaBeans (EJBs), or Java Transaction API (JTA). However, it's often used in development environments and for applications that don't require the full Java EE platform.

11 Thank You!