Week 3: Introduction to Servlets

Introduction to Servlets

A **Servlet** is a special type of Java program that runs on a **web server**. Its main job is to handle requests from users (like when someone visits a webpage) and then send back a response (like showing a webpage or giving data).

Think of a servlet as a **waiter** in a restaurant:

- The **client** (like a person using a web browser) asks for something, like a webpage or information (just like a customer ordering food).
- The **servlet** (the waiter) takes the request, processes it, and then returns something useful to the client, like a webpage or the result of a form they filled out (just like the waiter bringing food).

Why Are Servlets Important?

- **Dynamic Web Pages** - Servlets allow web pages to change based on user input. For example, when you log in to a website, the page changes to show your name and personalized content. Servlets handle this behind the scenes.
- **Processing Forms** - When you submit a form online (like a login form or a registration form), a servlet processes the information, checks it, and gives you feedback (like "Login successful" or "Incorrect password").
- **Talking to Databases** - If a website needs to store or get information from a database (like storing your profile details or checking for available products), the servlet handles this task.

Where Do Servlets Run?

Servlets run on a **web server** (a powerful computer that hosts websites). They are part of something called the **Java Enterprise Edition (Java EE)** framework. This framework includes lots of tools and technologies that help create web applications.

Servlets specifically respond to **HTTP requests**, which are the requests your web browser makes when you visit a website or click on a link. The servlet receives these requests and decides what to do, such as loading a page, processing data, or sending information back to your browser.

In short, servlets are essential for creating interactive and dynamic web applications, allowing websites to respond to user actions and requests efficiently.

Example in Real Life:

When you visit an online store and search for a product, the following happens:

1. Your browser sends a **request** to the server, asking for product information.
2. A **Servlet** on the server receives that request, looks up the product details in the database, and sends back a webpage displaying the information you asked for.

Without servlets, the web wouldn't be nearly as dynamic or interactive!

Key Concepts of Servlets

1. What is a Servlet?

- A **Servlet** is a Java class that extends the capabilities of a server by generating dynamic content in response to a client's request. It is specifically used for building web applications.
- Servlets handle requests, process business logic, and respond to the client, often returning HTML pages or JSON/XML data.

2. Servlet Lifecycle

The lifecycle of a servlet is controlled by the servlet container (such as **Apache Tomcat**). The container is responsible for loading, initializing, executing, and destroying servlets.

The **Servlet lifecycle** includes the following phases:

- **Loading and Instantiation** - The servlet class is loaded and instantiated by the server.
- **Initialization (init())** - The init() method is called once when the servlet is first loaded. It's used to initialize resources like database connections.
- **Service (service())** - The service() method is called each time the servlet responds to a client request (HTTP request). It determines the type of request (GET, POST) and forwards it to the appropriate method (doGet(), doPost()).
- **Destruction (destroy())** - The destroy() method is called before the servlet is removed from memory. It's used for cleanup tasks, like closing database connections.

Servlet Lifecycle Phases:

1. **Initialization** - init() method
2. **Request Handling** - service() method (or doGet(), doPost())
3. **Destruction** - destroy() method

Advantages of Servlets

1. **Platform Independence -** Servlets are written in Java, making them platform-independent. They can run on any server that supports the Java Servlet API.

2. **Efficiency** - Servlets are loaded once, and multiple requests can be handled by the same instance, improving performance.
3. **Security** - Java provides a secure environment for servlets, including features like secure communication, authentication, and authorization.
4. **Integration** - Servlets can easily interact with databases, APIs, and other services.
5. **Extensibility** - Servlets can be used alongside JSP (JavaServer Pages) to separate business logic from presentation.

Setting Up the Development Environment: NetBeans and Apache Tomcat

To develop and run servlets, you need:

- **NetBeans IDE**: A popular IDE that supports Java web development.
- **Apache Tomcat**: A web server that supports servlets and JSP.

Installing and Configuring Apache Tomcat in NetBeans

1. **Download and Install Apache Tomcat**:
    - Download Apache Tomcat from the official website: Apache Tomcat.
    - Follow the installation instructions provided on the website.
2. **Configure Tomcat in NetBeans**:
    - Open **NetBeans**.
    - Go to **Tools > Servers**.
    - Click **Add Server**, select **Apache Tomcat**, and click **Next**.
    - Browse to your Tomcat installation folder, and then configure Tomcat with a username and password (required for management).
    - Click **Finish**.

Developing a Simple HelloWorld Servlet in NetBeans

Now that you have your environment set up, let's create a simple servlet to handle HTTP requests and respond with a basic HTML page.

Step-by-Step: Create a HelloWorld Servlet

1. **Create a New Java Web Project**:
    - Open **NetBeans**.
    - Go to File > New Project.

- In the **Categories** section, choose Java Web and select Web Application.
- Click Next, name the project HelloServletExample, and click Next.
- Choose **Apache Tomcat** as your server and click Finish.

2. **Create a Servlet Class**:
   - Right-click on the Source Packages folder, go to New > Servlet.
   - Name the servlet HelloWorldServlet, and click Finish.
3. **Modify the Servlet Code**:
   - Once your servlet is created, NetBeans will generate some basic code. Modify it to display "Hello, World!" when the user accesses the servlet.

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/hello")
public class HelloWorldServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // Set the content type to text/html
        response.setContentType("text/html");

        // Get the writer object to send response
        PrintWriter out = response.getWriter();

        // Send HTML response
        out.println("<html>");
        out.println("<head><title>Hello World Servlet</title></head>");
        out.println("<body>");
        out.println("<h1>Hello, World!</h1>");
        out.println("</body>");
        out.println("</html>");
    }

    @Override
```

```
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // Handle POST requests here
    }
}
```

```java
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/hello")
public class HelloWorldServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // Set the content type to text/html
        response.setContentType("text/html");

        // Get the writer object to send response
        PrintWriter out = response.getWriter();

        // Send HTML response
        out.println("<html>");
        out.println("<head><title>Hello World Servlet</title></head>");
        out.println("<body>");
        out.println("<h1>Hello, World!</h1>");
        out.println("</body>");
        out.println("</html>");
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // Handle POST requests here
    }
}
```

4. **Understanding the Code**:
    - The @WebServlet("/hello") annotation maps the servlet to the /hello URL path.

- The doGet() method handles HTTP GET requests. It generates an HTML response containing "Hello, World!".
- The PrintWriter is used to send text (in this case, HTML) to the client (web browser).

Deploying and Running the Servlet

1. **Deploy the Servlet**:
   - Right-click on the project in the **Projects** window, and select **Run**.
   - NetBeans will start Apache Tomcat, deploy your project, and open a browser window.
2. **Access the Servlet**:
   - In your web browser, navigate to http://localhost:8080/HelloServletExample/hello.
   - You should see the output: **"Hello, World!"** displayed on the web page.

Exercises

Exercise 1: Modify the HelloWorld Servlet

- **Objective**: Modify the HelloWorldServlet to display the current date and time along with the greeting.
- **Steps**:
  1. Import the java.util.Date class.
  2. In the doGet() method, create a Date object.
  3. Modify the response to include the current date and time below the "Hello, World!" message.
- **Expected Output**: The webpage should display "Hello, World!" followed by the current date and time.

Exercise 2: Create a Personalized Greeting Servlet

- **Objective**: Create a new servlet that accepts a user's name as a query parameter in the URL and displays a personalized greeting.
- **Steps**:
  1. Create a new servlet called GreetingServlet.
  2. Use request.getParameter() to retrieve the user's name from the URL query string (e.g., ?name=John).

3. Respond with "Hello, [name]!" where [name] is the name entered by the user.
4. Test by navigating to a URL like http://localhost:8080/HelloServletExample/greeting?name=John.

- **Expected Output**: If the user's name is "John," the output should display "Hello, John!".

Additional Resources

- **Java EE Servlet Documentation**: Java Servlet API Documentation
- **Apache Tomcat Official Documentation**: Tomcat Documentation

Recommended Books

- Marty Hall and Larry Brown, *Core Servlets and JavaServer Pages (JSP)*, Sun Microsystems Press.
- Jason Hunter, *Java Servlet Programming*, O'Reilly Media.