Servlets(API)

Servlets mostly used to build web applications to display raw web applications and how to access web applications from our browser.

A **Servlet** is a Java class used to **handle HTTP requests and responses** in a web application

It runs on a **Java-enabled server** (like Apache Tomcat).

Java Editions:

- SE (Standard Edition)
- EE (Enterprise Edition)
- ME (Mobile Edition)

Java SE Applications:

- **Standalone Apps**: Installed locally (e.g., AnyDesk, TeamViewer, Calculator)
- **Applets**: Browser-embedded programs (deprecated)
- **Network Socket Applications:** Client-server communication

  ## J2EE

  It is used to build a web-application

→ any application that deployed in the server and we access such application by using internet.

To build this type of web-application in our we are going to take the help of **Servlets**.

Servlet is a small piece of program that is going to run on server.

Without servlets we cannot able to build an application.

Servlets

- It is an **API** which is generally used to build web applications.

- All the classes and interfaces related to Servlets are present inside javax.servlet package.

- Servlet is also an interface present in the javax.servlet package.

- Servlet is a program which is going to run on the server

Servlet classes:

- We write servlet classes on the server
- When there is a request, these classes are triggered.

- These classes produce a **response** and send it back to the client.

To create a servlet class:

- You need to implement the Servlet interface
- Servlet has some methods which we must provide implementation for.

Servlet Methods

1. public void init(ServletConfig config
    - Called to initialize the class members
2. public void service(ServletRequest request, ServletResponse response)
    - Used to handle requests and give responses
3. public void destroy()
    - Called to unload the servlet class
4. public ServletConfig getServletConfig(
    - Returns servlet configuration object
5. public String getServletInfo()
    - Provides information about the servlet interface.

GenericServlet (C)

- It is an **abstract class**.
- Implements the Servlet interface.
- Provides implementation for all methods **except** service() method.
- Because service() remains abstract, the class is still abstract.
- To create your own servlet, extend GenericServlet and implement the service() method.

GenericServlet Methods

1. public void init()
2. public ServletContext getServletContext()
    - Returns ServletContext for the application.
3. public String getInitParameter(String name)

4. public String getServletName()

5. public Enumeration getInitParameterNames()

6. public void log(String msg)

    - Logs small pieces of info, written to log file.

Creating a Maven Project (Servlet Setup)

- To create a Maven project:

    - Search: maven-archetype-webapp

    - It is from: org.apache.maven.archetypes

    - Version: > 1.4
    - Add java servlet api 4.0.1 version

Servlet Mapping and Deployment

- **Servlet class name** should always be a fully qualified class name.

- After registering the servlet, you need to map it in web.xml:

```
<servlet-mapping>
    <servlet-name>First</servlet-name>
    <url-pattern>/message</url-pattern>
</servlet-mapping>
```

When the URL /message is hit, the servlet class First is triggered, and its method is invoked.

```
public class First extends GenericServlet {
    public void service(ServletRequest req, ServletResponse res)
        throws ServletException, IOException {

        PrintWriter pw = res.getWriter();
        pw.print("<html><body>");
```

```
        pw.print("<h1>Hi, I am called</h1>");
        pw.print("</body></html>");
    }
}
```

This will print data in the **browser**

Mapping is required for the servlet to be invoked

Based on URL pattern, the servlet will be triggered.

Read Form Data in Generic Servlet

1. Create HTML form in src/main/webapp/info.html
2. Build form with fields:

```
<form action="/submit">
    Name: <input type="text" name="user">
    Email: <input type="text" name="user-email">
```

```
        Age: <input type="number" name="user-age">

        <input type="submit" value="Submit">

    </form>
```

3. Create servlet to process form data:

```
public class UserInfo extends GenericServlet {
    public void service(ServletRequest req, ServletResponse res)
        throws ServletException, IOException {

        // Get form parameters
        String user = req.getParameter("user");
        String email = req.getParameter("user-email");
        String age = req.getParameter("user-age");

        // Send response
        PrintWriter out = res.getWriter();
        out.println("User name is: " + user);
        out.println("Email is: " + email);
        out.println("Age is: " + age);
    }
}
```

HttpServlet

HttpServlet is an abstract class in javax.servlet.http.
There are no abstract methods in HttpServlet, but it's made abstract to prevent direct instantiation.

While we could extend GenericServlet, it's recommended to extend HttpServlet and override its methods.

Hierarchy:

[Servlet] (interface)
↑
[GenericServlet] (class)
↑
[HttpServlet] (class)
↑
[CustomServlet]

HttpServlet Methods:
public void service(ServletRequest req, ServletResponse res)
public void service(HttpServletRequest req, HttpServletResponse res)
protected void doGet(HttpServletRequest req, HttpServletResponse res)
protected void doPost(HttpServletRequest req, HttpServletResponse res)
protected void doHead(HttpServletRequest req, HttpServletResponse res)
protected void doOptions(HttpServletRequest req, HttpServletResponse res)
protected void doPut(HttpServletRequest req, HttpServletResponse res)
protected void doTrace(HttpServletRequest req, HttpServletResponse res)
protected void doDelete(HttpServletRequest req, HttpServletResponse res)
public long getLastModified(HttpServletRequest req)

Reading form data via GET (default method)

```
public class Register extends HttpServlet {
   protected void doGet(HttpServletRequest req, HttpServletResponse res)
      throws ServletException, IOException {

      String user = req.getParameter("user-name");
      String email = req.getParameter("user-email");
      String age = req.getParameter("user-age");
   }
}
```

HTML Form & GET Handler

```
<form action="/register" method="get">
   Name: <input type="text" name="user-name">
   Email: <input type="text" name="user-email">
   Age: <input type="text" name="user-age">
   <input type="submit" value="Register User">
</form>

PrintWriter out = res.getWriter();
out.print("<html><body>");
out.print("<h1>Name: " + un + "</h1>");
out.print("<h1>Email: " + e + "</h1>");
out.print("<h1>Age: " + a + "</h1>");
out.print("</body></html>");
```

POST Method Handler

```
public class UserRegister extends HttpServlet {
   protected void doPost(HttpServletRequest req, HttpServletResponse res)
      throws ServletException, IOException {

      String username = req.getParameter("user-name");
```

```
        String userEmail = req.getParameter("user-email");
        String userAge = req.getParameter("user-age");

        PrintWriter out = res.getWriter();
        out.print("<html><body>");
        out.print("<h1>Name: " + username + "</h1>");
        out.print("<h1>Email: " + userEmail + "</h1>");
        out.print("<h1>Age: " + userAge + "</h1>");
        out.print("</body></html>");
    }
}

// Important: HTML must specify method="post"
// Otherwise it defaults to GET and causes issues
```

Query Parameters Example

Create two servlets to demonstrate query parameter handling:

```
@WebServlet(value = "/page-b")
public class PageB extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {

        PrintWriter out = resp.getWriter();
        out.print("<h1>Hi, I'm in page B</h1>");
    }
}

@WebServlet(value = "/page-a")
public class PageA extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
```

```
        String myId = req.getParameter("id");
        PrintWriter out = resp.getWriter();
        out.print("<h1>Hi, I'm in page A</h1>");
        out.println("ID: " + myId);
    }
}
```

HTML Menu (menu.html):

```
<a href="page-a">Click to go to Page-A</a>
<a href="page-b">Click to go to Page-B</a>
```

To pass parameters in URL:

```
<a href="page-a?id=123">Go to Page A (ID=123)</a>
```

How to pass the data in the url

```
<a href="page-a?id=10">click to go to Page-A</a>
<a href="page-b?name='Samu'& id=10 k Email='Samu@123'"> </a>
```

How does Servlet works

Servlet is a small piece of program that run on the Server.

Now with any device we will send the request to Servlet when our device will interact with Servlet class and Servlet class sends back the request response to client.

Ex: Assume facebook, whatsapp, insta all belong to one Server in our Servlet all the application will be running. Now when client sends the request to open insta, it will open the insta itself.

The Server Controls which application & which Servlet class has to be triggered.

From the client (browser), you will send a request to get specific information (e.g., animal data).

The request must be in proper HTTP format and follow standard protocols.

When the request arrives, the Servlet Container (which manages secured servlet classes storing
various information) must determine which specific servlet class to trigger.

The Servlet Container:
1. Receives client requests
2. Converts HTTP requests into Java objects (HttpServletRequest)
3. Routes the request object to the appropriate servlet class
4. The servlet class processes the request and prepares a response

The servlet sends the response as an object (HttpServletResponse) back to the Servlet Container,
which then sends the final response to the client.

Note:
- When the HttpServletRequest object is created, the container checks if the servlet instance exists
- If not, it loads the servlet class, creates an instance in memory, and initializes it
- The servlet instance processes the request and returns an HttpServletResponse object
- The container converts this response object into an HTTP response and sends it to the client

Servlet Life Cycle

The Servlet Life Cycle is controlled and managed by the Web Container.

1. Loading Servlet Class
→ Class Loader is responsible for loading the Servlet class into memory.
  - Loads on first request
  - Not reloaded for subsequent requests (single class load)
After class loading:

2. Instantiation of Servlet Class
→ Web Container creates an instance of the Servlet class.
The Web Container:

- Instantiates the servlet class
- Provides access to non-static members
Only one servlet instance is created during the entire lifecycle.

3. Initialization: init()
→ Web Container initializes the servlet by calling init(ServletConfig)
  - Configures servlet parameters
  - Performed only once per servlet instance

4. Request Handling: service()
→ Invoked for every client request
→ Routes requests to appropriate methods (doGet(), doPost(), etc.)

5. Destruction: destroy()
→ Called before unloading the servlet

Before unloading the servlet, destroy() is invoked:
→ Web Container removes servlet instance when unloading
→ Only occurs once in the servlet lifecycle

Lifecycle Summary:
1. Class Loading
2. Instantiation
3. Initialization (init())
4. Request Handling (service()/doXxx())
5. Destruction (destroy())

Welcome File List

Purpose:
- Specify which page should open automatically when launching the web application
- Configured in web.xml

Example:

Set Home.html as the welcome page

```
<!-- web.xml Configuration -->
<web-app>
    <welcome-file-list>
        <welcome-file>pageA.html</welcome-file>
        <welcome-file>pageB.html</welcome-file>
        <welcome-file>pageC.html</welcome-file>
    </welcome-file-list>
</web-app>
```

How Multiple Welcome Files Work:
1. Container checks files in specified order:
   - First looks for pageA.html → serves if exists
   - If missing → checks pageB.html
   - If missing → checks pageC.html
2. If none found → throws 404 "Page not found" error

Default Behavior (if no configuration):
1. Container searches for:
   - index.html
   - index.htm
   - index.jsp
2. If none exist → 404 error

Purpose:
- Determines initial page when accessing application root
- Applies to first visit (e.g., http://yourapp.com/)

Implementation Example:

1. Create HTML files:
   - welcome.html: <h1>Hello</h1>
   - pageA.html: <h1>PageA</h1>
   - pageB.html: <h1>PageB</h1>

2. Configure web.xml:
xml
<welcome-file-list>
  <welcome-file>welcome.html</welcome-file>
  <welcome-file>pageA.html</welcome-file>
  <welcome-file>pageB.html</welcome-file>
</welcome-file-list>

Behavior:

- Serves welcome.html if exists
- Else pageA.html → Else pageB.html → Else 404

<welcome-file-list>
  <welcome-file>index.html</welcome-file>
  <welcome-file>index.htm</welcome-file>
  <welcome-file>index.jsp</welcome-file>
</welcome-file-list>

RequestDispatcher

It is an interface in javax.servlet package.

As the name suggests, it helps dispatch requests:
- From one servlet to another servlet
- From servlet to HTML page
- From JSP to another JSP
- From JSP to servlet

RequestDispatcher provides two key methods:
1. forward(request, response)
2. include(request, response)

We obtain RequestDispatcher object by:

→ Calling request.getRequestDispatcher(path)
→ Using the request object in a servlet

forward() Method Example:

Scenario: BMI Calculator
- User submits data to first servlet (BMI Calculator)
- Servlet calculates BMI
- Forwards request to second servlet (Result Processor) for:
  - Age-based adjustments
  - Health risk assessment
  - Personalized recommendations

Flow:
[Client] → [BMIServlet] → (forward) → [ResultServlet] → [Response]

Key Points:
- Forwarding happens server-side (client unaware)
- Same request/response objects passed
- Used for modular processing

The include() method is used to:
- Include one servlet's output inside another servlet
- Embed a JSP page within another JSP page

Example: Login System
1. User submits credentials via Login Page (Page A)
2. Servlet validates email/password
3. If credentials are invalid:
   - Servlet includes Page A again
   - Adds error message: "Email or password is wrong"

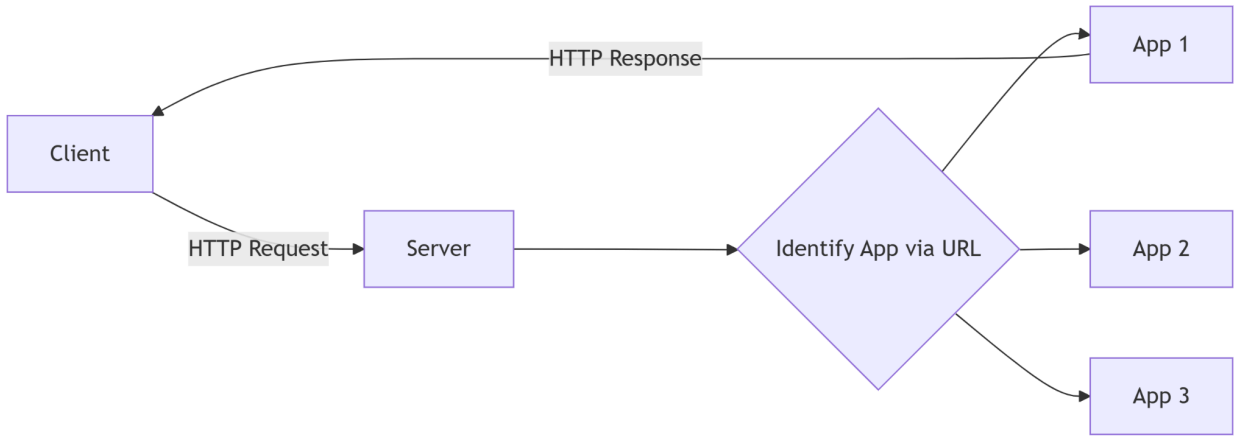
Flow:
[Login Page A] → [Validation Servlet] → (include) → [Login Page A with error]

Result:
- Original login page is displayed with embedded error message
- Appears as single page to user

Server:
- A server is a computer system with:
  - Operating system (OS)
  - Capability to host applications
- Web applications are installed on servers similar to local software installation
- Key principle:
  → Every web application deployed on a server has a unique identifier
  → This uniqueness is provided by a dedicated URL

Client-Server Communication:
- To connect from client to server:
  → Must use standard protocols (HTTP/HTTPS)
- Applications accessed via these protocols are web applications

Critical Requirement:
- All requests to the server MUST follow HTTP protocol
- Without HTTP, applications cannot be invoked because:
  - Servers host multiple applications
  - HTTP provides:
    * Standard communication format
    * URL-based application identification
    * Request routing mechanism



`HttpServletRequest` is an interface in Java Servlets that provides details about the **client's request** to the server. It helps in retrieving form data, headers, session details, and request information.

Common Methods:

- `getParameter(String name)`: Gets form data from the request.
- `getMethod()`: Returns the HTTP method (`GET`, `POST`, etc.).
- `getRequestURI()`: Retrieves the requested URL. `getHeader(String name)`: Fetches request headers. `getSession()`: Gets the session object for the client.

`HttpServletResponse` is an interface in Java Servlets that helps send a response from the server to the client. It allows setting **status codes, headers,and response content**.

Common Methods:

- `setContentType(String type)`: Sets the response content type (e.g., HTML, JSON).
- `getWriter()`: Returns a `PrintWriter` to send text response.
- `sendRedirect(String url)`: Redirects the client to another page.

- `setStatus(int statusCode)`: Sets the HTTP response status (e.g., `200 OK`, `404 Not Found`).