**HTTP Methods in Servlets**

HTTP methods define the type of request a client can make to a server. Servlets handle these methods through HttpServlet's methods like doGet(), doPost(), etc.

**Common HTTP Methods**

**1. GET**

* **Description**: Retrieves data from the server.
* **Characteristics**:
  + Data sent as query parameters in the URL.
  + Limited data size due to URL length restrictions.
  + No sensitive information (parameters are visible in the URL).
  + Can be bookmarked or cached.
* **Example**:
* <a href="/search?query=java">Search for Java</a>
* @Override
* protected void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
* String query = request.getParameter("query");
* response.getWriter().println("Search query: " + query);
* }
* **Use Case**: Searching, retrieving public data.

**2. POST**

* **Description**: Submits data to the server for processing.
* **Characteristics**:
  + Data sent in the request body (not visible in the URL).
  + Can handle large amounts of data.
  + Often used for secure data (e.g., passwords).
* **Example**:
* <form action="/login" method="POST">
* <input type="text" name="username">
* <input type="password" name="password">
* <button type="submit">Login</button>
* </form>
* @Override
* protected void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException {
* String username = request.getParameter("username");
* response.getWriter().println("Welcome, " + username);
* }
* **Use Case**: Submitting forms, uploading files.

**3. PUT**

* **Description**: Updates or creates a resource on the server.
* **Characteristics**:
  + Often idempotent (multiple requests result in the same state).
  + Replaces the existing resource or creates a new one if it doesn't exist.
* **Example**:
* @Override
* protected void doPut(HttpServletRequest request, HttpServletResponse response) throws IOException {
* String resourceId = request.getParameter("id");
* // Update or create logic here
* response.getWriter().println("Resource updated: " + resourceId);
* }
* **Use Case**: Updating records, creating resources.

**4. DELETE**

* **Description**: Deletes a resource on the server.
* **Characteristics**:
  + Usually idempotent.
  + Removes the specified resource.
* **Example**:
* @Override
* protected void doDelete(HttpServletRequest request, HttpServletResponse response) throws IOException {
* String resourceId = request.getParameter("id");
* // Delete logic here
* response.getWriter().println("Resource deleted: " + resourceId);
* }
* **Use Case**: Deleting user accounts, removing files.

**5. HEAD**

* **Description**: Similar to GET, but retrieves only headers (no body).
* **Use Case**: Checking if a resource exists, verifying metadata like content type and size.

**6. OPTIONS**

* **Description**: Describes the available HTTP methods for a resource.
* **Use Case**: Preflight requests in CORS.

**7. PATCH**

* **Description**: Partially updates a resource.
* **Use Case**: Updating specific fields of a user profile (e.g., just the email).

**Summary Table**

| **Method** | **Purpose** | **Data Location** | **Idempotent?** | **Use Case** |
| --- | --- | --- | --- | --- |
| GET | Retrieve data | Query params | Yes | Search queries, public data |
| POST | Submit data | Request body | No | Forms, file uploads |
| PUT | Create/replace resource | Request body | Yes | Updating user info |
| DELETE | Remove resource | Query params | Yes | Deleting files or accounts |
| HEAD | Retrieve headers only | - | Yes | Resource existence check |
| OPTIONS | List allowed HTTP methods | - | Yes | CORS, method introspection |
| PATCH | Partially update resource | Request body | No | Updating specific fields |

The **Servlet Hierarchy** in Java explains the structure of classes and interfaces in the Servlet API. Below is a breakdown of the hierarchy:

**Servlet Interface (Root of the Hierarchy)**

* It defines the basic methods that all servlets must implement.
* Core methods:
  1. init(ServletConfig config)
  2. service(ServletRequest req, ServletResponse res)
  3. destroy()

**GenericServlet (Abstract Class)**

* A base class that implements the Servlet and ServletConfig interfaces.
* Provides simple implementations of many methods, leaving only the service() method to be implemented by subclasses.
* Not specific to HTTP.

public abstract class GenericServlet implements Servlet, ServletConfig {

// Methods like init(), getServletConfig(), getServletInfo() are implemented here

}

**HttpServlet (Abstract Class)**

* Extends GenericServlet to provide HTTP-specific functionality.
* Contains methods to handle HTTP requests:
  1. doGet(HttpServletRequest req, HttpServletResponse res)
  2. doPost(HttpServletRequest req, HttpServletResponse res)
  3. doPut(), doDelete(), doHead(), etc.

**Hierarchy Visualization**

java.lang.Object

↳ javax.servlet.Servlet (Interface)

↳ javax.servlet.GenericServlet (Abstract Class)

↳ javax.servlet.http.HttpServlet (Abstract Class)

**Usage in Development**

1. **Implementing Servlet Interface**: Rare, as you have to implement all methods yourself.
2. **Extending GenericServlet**: Used for non-HTTP protocols.
3. **Extending HttpServlet**: Common for most web applications as it provides HTTP-specific methods.

To run a servlet program, you need to configure it in the **web.xml** file (also called the deployment descriptor). This file tells the server about your servlet and how it should handle requests. Here's a step-by-step explanation:

**Steps to Add a Servlet in web.xml**

**1. Basic Structure of web.xml**

The web.xml file resides in the WEB-INF directory of your project. It must follow the XML structure and syntax.

xml

Copy code

<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="3.0">

<!-- Servlet and Mapping entries go here -->

</web-app>

**2. Define the Servlet**

You need to specify the servlet class in the web.xml file.

xml

Copy code

<servlet>

<servlet-name>MyServlet</servlet-name>

<servlet-class>in.sp.backend.MyServlet</servlet-class>

</servlet>

* **<servlet-name>**: A unique name for the servlet (used in mappings).
* **<servlet-class>**: Fully qualified class name of the servlet (package + class name).

**3. Map the Servlet to a URL**

Associate the servlet with a specific URL pattern so the server knows when to call it.

xml

Copy code

<servlet-mapping>

<servlet-name>MyServlet</servlet-name>

<url-pattern>/myservlet</url-pattern>

</servlet-mapping>

* **<servlet-name>**: Matches the name defined earlier in the <servlet> tag.
* **<url-pattern>**: Defines the URL path to access the servlet (e.g., http://localhost:8080/myapp/myservlet).

**4. Full Example of web.xml**

Here’s the complete file with everything put together:

xml

Copy code

<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="3.0">

<!-- Servlet Definition -->

<servlet>

<servlet-name>MyServlet</servlet-name>

<servlet-class>in.sp.backend.MyServlet</servlet-class>

</servlet>

<!-- Servlet Mapping -->

<servlet-mapping>

<servlet-name>MyServlet</servlet-name>

<url-pattern>/myservlet</url-pattern>

</servlet-mapping>

</web-app>