Introduction to Servlets

Web or application server acts as a middle layer between requests coming from Web browsers or other HTTP clients and databases or applications on the HTTP server. **Tasks of web applications:**

- **Read explicit data from clients** (for example, data from a HTML form on a Web page).
- **Read the implicit HTTP request data sent by a web browser** (behind-the-scenes HTTP information, like cookies, media types and compression schemes).
- **Generate and present data** within a document (HTML or JSP).
- **Send the implicit HTTP response data** (behind-the-scenes HTTP information. like setting cookies and caching parameters).

Basic Servlet structure

A basic servlet handles **GET** requests. **GET** requests are the usual type of browser HTTP requests for Web pages generated in the following cases:

- User enters a URL on the address line.
- User follows a link on a web page
- User submits a HTML form that either does not specify a method or specifies METHOD="GET".

Servlets can also easily handle POST requests, which are generated when a user submits an HTML form that specifies METHOD="POST".

Servlet basic template:

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

public class ServletTemplate extends HttpServlet {

      public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
                  throws ServletException, IOException {
            PrintWriter out = response.getWriter();
            pw.println("Hello World from doGet method in my first servlet");

      }

public void doPost(HttpServletRequest request,
                  HttpServletResponse response)
                  throws ServletException, IOException {
            PrintWriter out = response.getWriter();
            pw.println("Hello World from doPost method in my first servlet");
      }

}
```

Servlets extend class HttpServlet and override doGet or doPost, depending on whether the data is being sent by GET or by POST methods. If you want a servlet to take the same action for both GET and POST requests, simply have doGet call doPost, or vice versa.

Methods doGet and doPost

Both doGet and doPost take two **arguments**:

- **HttpServletRequest** - used for getting all incoming data:
  - Form data,
  - HTTP request headers,
  - The client's hostname.
- **HttpServletResponse** - used for specifying outgoing information to clients such as:
  - HTTP status codes (200, 404, etc.),
  - Response headers (Content-Type, Set-Cookie, etc.),
  - Document content by obtaining a **`PrintWriter`** for preparing and sending back data to the client.

За више детаља о класи HttpServlet треба проучити Servlet 4.0 API спецификацију на адреси

https://tomcat.apache.org/tomcat-9.0-doc/servletapi/overview-summary.html

са посебним осврtом на класу HttpServlet:

https://tomcat.apache.org/tomcat-9.0-doc/servletapi/index.html

Servlet life cycle

Only a single instance of each servlet is created, while each user request results in a new thread that is handed off to **`doGet`** or **`doPost`** as appropriate.

When the servlet is first created, its **`init`** method is invoked, which is used for one-time setup code. After this, each user request results in a thread that calls the **`service`** method of the previously created instance. Multiple concurrent requests normally result in multiple threads calling **`service`** simultaneously. In some cases a servlet can implement a special interface **`SingleThreadModel`** that stipulates that allows a single thread to be run at any time. The service method then calls **`doGet`**, **`doPost`**, or another **`doXxx`** (**`doPut`**, **`doDelete`**) method, depending on the type of HTTP request it received. When the server decides to unload a servlet, it calls the servlet's **`destroy`** method.

Method **`init`** is called when a servlet is loaded to a web server, and it is used for configuring the servlet for the next HTTP calls. For example, it can be used for setting cookies, setting up a connection to database etc. When there is no need to work with the servlet, **`destroy`** method is invoked, which enables saving files, closing connections to databases etc. However, it is better to do all these activities with specially created methods.

NOTE: Life cycle methods such as **`init`** method or **`destroy`** method should be called by the servlet container (e.g. Tomcat). The advice is not to call them within the code. They can contain code, but the next advice is to do all necessary initialization and clean up with specially created methods that are called from the servlet **`doGet`** or **`doPost`** methods.

Literature and Links

[1] **The Apache Tomcat**. http://tomcat.apache.org/

[2] Marty Hall and Larry Brown. *Core Servlets and JavaServer Pages, Free Online Version of Second Edition*. http://pdf.coreservlets.com/

[3] https://tomcat.apache.org/tomcat-9.0-doc/servletapi/overview-summary.html

[4] https://tomcat.apache.org/tomcat-9.0-doc/servletapi/index.html.

[5] Marty Hall and Larry Brown. *Core Servlets and JavaServer Pages, Free Online Version of Second Edition*. http://pdf.coreservlets.com/