Lab: Servlets

Preamble

Throughout the lab, please check the documentation of the classes and the methods you use for the first time: https://javaee.github.io/javaee-spec/javadocs/

1 First Steps

Create a new Java EE project. Then create the package `controller`. This package will contain all the classes you will be developing in this lab.

1.1 Displaying the Parameters of an HTTP Request

Create a new servlet, named `Home`. The template of the servlet is provided on Campus. This template completes IntelliJ IDEA's by providing the URL patterns the servlet is associated to. These patterns are specified through the `urlPatterns` parameter of the `@WebServlet` annotation of the servlet.

Modify the servlet so that the HTML response it sends back contains the parameters of the incoming HTTP request. To do so, use the `getParameterNames` and `getParameter` methods of the `HttpServletRequest` class. You can use the following HTML construct to display a list of items:

```
<ul>
	<li>param1=value1</li>
	<li>param2=value2</li>
	<li>…</li>
</ul>
```

Check your code using various URLs, for example:

http://localhost:8080/<project name>/home
http://localhost:8080/<project name>/home?lastname=Dupont&firstname=Lamia
http://localhost:8080/<project name>/home?lastname=Dupont&firstname=Lamia&mark=18

Observe that parameters values are passed as `String` objects: you must convert them to the appropriate type if needed. For example, the mark parameter could be converted to a `float` number using the `Float.parseFloat` method.

1.2 Testing the Lifecycle of a Servlet

The lifecycle of a servlet is managed by the web container of the application server. Right after the web container instantiates a servlet, it calls the `init` method of the servlet. Likewise, the web container calls the `destroy` method of the servlet just before it deletes it.

Override these two methods in your servlet and check the typical behavior of a web container: (1) the container does not instantiate the servlet until it receives the first HTTP request for it, (2) the container re-use the same servlet instance for all subsequent HTTP requests, (3) the container deletes the servlet instance when the application server terminates or when it has to free memory resources (depending on the configuration of the server).

Next, check that (1) the container re-use the same servlet instance to process simultaneous HTTP requests, (2) each request is processed by a dedicated thread. To demonstrate this: (1) display the name of the current thread in the doGet method, (2) block the processing of the request for a while (e.g. 10 seconds) using the `Thread.sleep` method, (3) send two simultaneous HTTP requests using two distinct tabs of your browser.

1.3 Implementing Forms

Modify the Home servlet so that it returns the HTML form below when the method doGet is invoked. The form has two fields, *named* login and password, *labeled* `Login` and `Password`, and one button named Log in.

```
<form action='home' method='GET' >
   Login: <input type='text' name='login' />
   Password: <input type='password' name='password' />
   <input type='submit' value='Log in' />
</form>
```

When the user clicks on the Log in button (type='submit'), the browser sends a GET HTTP request (method='GET') to the relative URL named home (action='home'). Since the method is GET, the browser passes the values entered into the fields as parameters to the URL.

Check that the values entered are correctly received and returned by the servlet.

Since connecting to an application modifies its state, the connection request should be sent using the POST method rather than GET. Modify your code accordingly and test it all over again. Observe that the browser no longer sends the (field, value) pairs as parameters to the URL. Instead, it sends them in the header of the POST request.

2 Basic Application

We are now going to develop a simplified application, implementing a basic login – process – logout pattern.

2.1 Using Three Servlets

As a first step, develop the application using three servlets: `Login`, `Process` and `Logout`.

The `Login` servlet displays the login form. If the login and the password match, the application enters the process state, handled by the `Process` servlet. The `Process` servlet displays a page with the string "Logged in: <login>" and two buttons named Continue and Log out. If the user clicks on Continue, he/she stays on the same page. If he/she clicks on Log out, the `Logout` servlet displays a page with the string "Logged out".

Your code needs to transfer execution control from one servlet to another, e.g. from the `Login` servlet to the `Process` servlet and then from the `Process` servlet to the `Logout` servlet. This can be implemented using a HTTP redirect, which involves a HTTP response and request between the server and the client. To avoid the incurred delay, control can transferred *internally*, using the web container dispatcher, as follows:

```
RequestDispatcher dispatcher = request.getRequestDispatcher(<next URL>);
dispatcher.forward(request, response);
```

Note that the calling servlet cannot complete the `response` after `forward` is called.

2.2 Using a Single Servlet

Implement again the application, this time using a single servlet named `Application`. The servlet acts as a state-machine: based on the current state of the application and the current user inputs in the HTTP request, the servlet determines the new state of the application and the HTTP response to send back. This forms the basis of the MVC Model 2 Architecture, a Java EE design pattern that we will study in the JavaServer Pages lesson.

Tip: To enhance readability, isolate the code that generates the various HTML responses into helper methods of the `Application` servlet.

3 Advanced Features

We are now going to add more features to our web application by using cookies and managing sessions. All the exercises below relate the `Application` servlet.

3.1 Cookies

Related Javadocs:
https://javaee.github.io/javaee-spec/javadocs/javax/servlet/http/Cookie.html
https://javaee.github.io/javaee-spec/javadocs/javax/servlet/http/HttpServletRequest.html#getCookies--
https://javaee.github.io/javaee-spec/javadocs/javax/servlet/http/HttpServletResponse.html#addCookie-javax.servlet.http.Cookie-

Modify you code so that the application remembers the login used on the last successful connection using permanent cookies. The application will display the login as the default value of the Login field, using the following construct:

```
<input type='text' name='<field name>' value='<default value>' />
```

By default, cookies are session cookies: they are deleted when the browser is closed. To make them permanent, you must call the method `setMaxAge` with a large value. Remember that cookies are included in the header of the HTTP response: the servlet must add cookies to the HTTP response before it writes anything to the output stream of the response, which represents its body.

Use your regular browser to test your code. Make sure you prevent your browser from storing the login and password you have just entered. If you don't, the browser will provide them on the next login and your test will not be significant.

Next, check the value and the properties of the cookie using the setting menu of your browser. Finally, check what happens if you delete the cookie.

3.2 Session Management

Related Javadocs:
https://javaee.github.io/javaee-spec/javadocs/index.html?javax/servlet/http/HttpSession.html
https://javaee.github.io/javaee-spec/javadocs/index.html?javax/servlet/http/HttpServletRequest.html

Cookies allow the application to store long-lived information on the client side. They are not suited for storing information about the current *session*, i.e. the time period between a user logs in and he/she logs out.

This information can be stored using an object of type `HttpSession`, in the form of (attribute name, attribute value) pairs. This information is stored and managed by the servlet container, on the server side.

The `HttpServletRequest.getSession` method returns the session object associated with the current session, or creates such an object if it does not exist yet. The session will last until the `invalidate` method is called or until the inactivity period expires. The inactivity period can be set through the `setMaxInactiveInterval` method. In the meantime, one can set, get and remove attributes from the session object using the `set/get/removeAttribute` methods, respectively.

Complete your code so that (1) a session is created between the time a user logs in and he/she logs out, with an inactivity interval of 30s, (2) the user can add and remove session attributes through a simple form with two fields named Attribute and Value and three buttons named Add, Remove and Log out.

By default, the web container implements sessions using cookies: the ID of the session is passed back and forth using a cookie named `JSESSIONID`. Check that your navigator actually stores that cookie; further check that the value of the cookie is the value returned by the `getID` method of the `HttpSession` class.

Next, configure your browser so that cookies are disabled or use an “incognito” or “private” window. Check that the session mechanism still works: the web container now uses a request parameter to pass the ID of the session from the client to the server and vice-versa.

Finally, emulate two simultaneous users and check that your code can handle multiple sessions at the same time. Note that if you use the same client machine for both connections, you must use two distinct browsers (e.g. Chrome and Firefox) for the test to work.