Modern Servlets

Raúl Estrada

Octubre 2020

1. Overview

In this article, we will have a look at a core aspect of web development in Java – Servlets.

2. The Servlet and the Container

Simply put, a Servlet is a class that handles requests, processes them and reply back with a response.

For example, we can use a Servlet to collect input from a user through an HTML form, query records from a database, and create web pages dynamically.

Servlets are under the control of another Java application called a ***Servlet Container.*** When an application running in a web server receives a request, the Server hands the request to the Servlet Container – which in turn passes it to the target Servlet.

3. Maven Dependencies

To add Servlet support in our web app, the *javax.servlet-api* dependency is required:

```
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
</dependency>
```

The latest maven dependency can be found here.

Of course, we'll also have to configure a Servlet container to deploy our app to; this is a good place to start on how to deploy a WAR on Tomcat.

4. Servlet Lifecycle

Let's go through the set of methods which define the lifecycle of a Servlet.

4.1. *init()*

The *init* method is designed to be called only once. If an instance of the servlet does not exist, the web container:

1. Loads the servlet class
2. Creates an instance of the servlet class
3. Initializes it by calling the *init* method

The *init* method must complete successfully before the servlet can receive any requests. The servlet container cannot place the servlet into service if the *init* method either throws a *ServletException* or does not return within a time period defined by the Web server.

```
public void init() throws ServletException {
    // Initialization code like set up database etc....
}
```

4.2. *service()*

This method is only called after the servlet's *init()* method has completed successfully.

The Container calls the *service()* method to handle requests coming from the client, interprets the HTTP request type (*GET*, *POST*, *PUT*, *DELETE*, etc.) and calls *doGet*, *doPost*, *doPut*, *doDelete*, etc. methods as appropriate.

```java
public void service(ServletRequest request, ServletResponse response) 
  throws ServletException, IOException {
    // ...
}
```

4.3. *destroy()*

Called by the Servlet Container to take the Servlet out of service.

This method is only called once all threads within the servlet's *service* method have exited or after a timeout period has passed. After the container calls this method, it will not call the *service* method again on the Servlet.

```
public void destroy() {
    //
}
```

5. Example Servlet

Let's now setup a full example of handling information using a form.

To start, let's define a servlet with a mapping */calculateServlet* which will capture the information POSTed by the form and return the result using a RequestDispatcher:

```
@WebServlet(name = "FormServlet", urlPatterns = "/calculateServlet")
public class FormServlet extends HttpServlet {

    @Override
    protected void doPost(HttpServletRequest request,
      HttpServletResponse response)
      throws ServletException, IOException {

        String height = request.getParameter("height");
        String weight = request.getParameter("weight");
```

```java
        try {
            double bmi = calculateBMI(
              Double.parseDouble(weight),
              Double.parseDouble(height));

            request.setAttribute("bmi", bmi);
            response.setHeader("Test", "Success");
            response.setHeader("BMI", String.valueOf(bmi));

            RequestDispatcher dispatcher
              = request.getRequestDispatcher("index.jsp");
            dispatcher.forward(request, response);
        } catch (Exception e) {
            response.sendRedirect("index.jsp");
        }
    }

    private Double calculateBMI(Double weight, Double height) {
        return weight / (height * height);
    }
}
```

As shown above, classes annotated with *@WebServlet* must extend the *javax.servlet.http.HttpServlet* class. It is important to note that *@WebServlet* annotation is only available from Java EE 6 onward.

The *@WebServlet* annotation is processed by the container at deployment time, and the corresponding servlet made available at the specified URL patterns. It is worth noticing that by using the annotation to define URL patterns, we can avoid using XML deployment descriptor named *web.xml* for our Servlet mapping.

If we wish to map the Servlet without annotation, we can use the traditional *web.xml* instead:

```
<web-app ...>

    <servlet>
       <servlet-name>FormServlet</servlet-name>
       <servlet-class>com.root.FormServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>FormServlet</servlet-name>
        <url-pattern>/calculateServlet</url-pattern>
    </servlet-mapping>

</web-app>
```

Next, let's create a basic HTML *form*:

```html
<form name="bmiForm" action="calculateServlet" method="POST">
    <table>
        <tr>
            <td>Your Weight (kg) :</td>
            <td><input type="text" name="weight"/></td>
        </tr>
        <tr>
            <td>Your Height (m) :</td>
            <td><input type="text" name="height"/></td>
        </tr>
        <th><input type="submit" value="Submit" name="find"/></th>
        <th><input type="reset" value="Reset" name="reset" /></th>
    </table>
    <h2>${bmi}</h2>
</form>
```

Finally – to make sure everything's working as expected, let's also write a quick test:

```
public class FormServletLiveTest {

    @Test
    public void whenPostRequestUsingHttpClient_thenCorrect()
      throws Exception {

        HttpClient client = new DefaultHttpClient();
        HttpPost method = new HttpPost(
          "http://localhost:8080/calculateServlet");

        List<BasicNameValuePair> nvps = new ArrayList<>();
        nvps.add(new BasicNameValuePair("height", String.valueOf(2)));
        nvps.add(new BasicNameValuePair("weight", String.valueOf(80)));

        method.setEntity(new UrlEncodedFormEntity(nvps));
        HttpResponse httpResponse = client.execute(method);

        assertEquals("Success", httpResponse
          .getHeaders("Test")[0].getValue());
        assertEquals("20.0", httpResponse
          .getHeaders("BMI")[0].getValue());
    }
}
```

6. Servlet, HttpServlet and JSP

It's important to understand that **the Servlet technology is not limited to the HTTP protocol.**

In practice it almost always is, but *Servlet* is a generic interface and the *HttpServlet* is an extension of that interface – adding HTTP specific support – such as *doGet* and *doPost*, etc.

Finally, the Servlet technology is also the main driver a number of other web technologies such as JSP – JavaServer Pages, Spring MVC, etc.

7. Conclusion

In this quick article, we introduced the foundations of Servlets in a Java web application.