Context Initialization

Raúl Estrada

Octubre 2020

1. Overview

Servlets are plain Java classes that run in a servlet container.

HTTP servlets (a specific type of servlet) are first class citizens in Java web applications. The API of HTTP servlets is **aimed at handling HTTP requests through the typical request-processing-response cycle, implemented in client-server protocols**.

Furthermore, servlets can control the interaction between a client (typically a web browser) and the server using key-value pairs in the form of request/response parameters.

These parameters can be initialized and bound to an application-wide scope (context parameters) and a servlet-specific scope (servlet parameters).

In this tutorial, we'll learn **how to define and access context and servlet initialization parameters**.

2. Initializing Servlet Parameters

We can define and initialize servlet parameters using annotations and the standard deployment descriptor — the *"web.xml"* file. It's worth noting that these two options are not mutually exclusive.

Let's explore each of these options in depth.

2.1. Using Annotations

Initializing servlets parameters with annotations allows us to keep configuration and source code in the same place.

In this section, we'll demonstrate how to define and access initialization parameters that are bound to a specific servlet using annotations.

To do so, we'll implement a naive *UserServlet* class that collects user data through a plain HTML form.

First, let's look at the JSP file that renders our form:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Context and Initialization Servlet Parameters</title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    </head>
    <body>
        <h2>Please fill the form below:</h2>
        <form action="${pageContext.request.contextPath}/userServlet" method="post">
            <label for="name"><strong>Name:</strong></label>
            <input type="text" name="name" id="name">
            <label for="email"><strong>Email:</strong></label>
            <input type="text" name="email" id="email">
            <input type="submit" value="Send">
        </form>
    </body>
</html>
```

Note that we've coded the form's *action* attribute by using the EL (the Expression Language). This allows it to always point to the *"/userServlet"* path, regardless of the location of the application files in the server.

The *"${pageContext.request.contextPath}"* expression **sets a dynamic URL for the form, which is always relative to the application's context path**.

Here's our initial servlet implementation:

```
@WebServlet(name = "UserServlet", urlPatterns = {"/userServlet"}, initParams={
@WebInitParam(name="name", value="Not provided"),
@WebInitParam(name="email", value="Not provided")}))
public class UserServlet extends HttpServlet {
    // ...

    @Override
    protected void doPost(
      HttpServletRequest request,
      HttpServletResponse response)
      throws ServletException, IOException {
        processRequest(request, response);
        forwardRequest(request, response, "/WEB-INF/jsp/result.jsp");
    }
```

```
    protected void processRequest(
      HttpServletRequest request,
      HttpServletResponse response)
      throws ServletException, IOException {

        request.setAttribute("name", getRequestParameter(request, "name"));
        request.setAttribute("email", getRequestParameter(request, "email"));
    }

    protected void forwardRequest(
      HttpServletRequest request,
      HttpServletResponse response,
      String path)
      throws ServletException, IOException {
        request.getRequestDispatcher(path).forward(request, response);
    }

    protected String getRequestParameter(
      HttpServletRequest request,
      String name) {
        String param = request.getParameter(name);
        return !param.isEmpty() ? param : getInitParameter(name);
    }
}
```

In this case, we've defined two servlet initialization parameters, *name* and *email*, by **using *initParams* and the *@WebInitParam* annotations**.

Please note that we've used HttpServletRequest's *getParameter()* method to retrieve the data from the HTML form, and the *getInitParameter()* method to access the servlet initialization parameters.

The *getRequestParameter()* method checks if the *name* and *email* request parameters are empty strings.

If they are empty strings, then they get assigned the default values of the matching initialization parameters.

The *doPost()* method first retrieves the name and email that the user entered in the HTML form (if any). Then it processes the request parameters and forwards the request to a *"result.jsp"* file:

The *doPost()* method first retrieves the name and email that the user entered in the HTML form (if any). Then it processes the request parameters and forwards the request to a *"result.jsp"* file:

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>User Data</title>
    </head>
    <body>
        <h2>User Information</h2>
        <p><strong>Name:</strong> ${name}</p>
        <p><strong>Email:</strong> ${email}</p>
    </body>
</html>
```

If we deploy our sample web application to an application server, such as Apache Tomcat, Oracle GlassFish or JBoss WidlFly, and run it, it should first display the HTML form page.

Once the user has filled out the *name* and *email* fields and submitted the form, it will output the data:

```
User Information
Name: the user's name
Email: the user's email
```

If the form is just blank, it will display the servlet initialization parameters:

```
User Information
Name: Not provided
Email: Not provided
```

In this example, we've shown **how to define servlet initialization parameters by using annotations, and how to access them with the *getInitParameter()* method**.

2.2. Using the Standard Deployment Descriptor

This approach differs from the one that uses annotations, as it allows us to keep configuration and source code isolated from each other.

To showcase how to define initialization servlet parameters with the *"web.xml"* file, let's first remove the *initParam* and *@WebInitParam* annotations from the *UserServlet* class:

```
@WebServlet(name = "UserServlet", urlPatterns = {"/userServlet"})
public class UserServlet extends HttpServlet { ... }
```

Next, let's define the servlet initialization parameters in the *"web.xml"* file:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
  xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
version="3.1">
    <servlet>
        <display-name>UserServlet</display-name>
        <servlet-name>UserServlet</servlet-name>
        <init-param>
            <param-name>name</param-name>
            <param-value>Not provided</param-value>
        </init-param>
        <init-param>
            <param-name>email</param-name>
            <param-value>Not provided</param-value>
        </init-param>
    </servlet>
</web-app>
```

As shown above, defining servlet initialization parameters using the *"web.xml"* file just boils down to using the *<init-param>*, *<param-name>* and *<param-value>* tags.

Furthermore, it's possible to define as many servlet parameters as needed, as long as we stick to the above standard structure.

When we redeploy the application to the server and rerun it, it should behave the same as the version that uses annotations.

3. Initializing Context Parameters

Sometimes we need to define some immutable data that must be globally shared and accessed across a web application.

Due to the data's global nature, we should **use application-wide context initialization parameters for storing the data, rather than resorting to the servlet counterparts**.

Even though it's not possible to define context initialization parameters using annotations, we can do this in the *"web.xml"* file.

Let's suppose that we want to provide some default global values for the country and province where our application is running.

We can accomplish this with a couple of context parameters.

Let's refactor the *"web.xml"* file accordingly:

```
<web-app
  xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
  http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" version="3.1">
    <context-param>
        <param-name>province</param-name>
        <param-value>Mendoza</param-value>
    </context-param>
    <context-param>
        <param-name>country</param-name>
        <param-value>Argentina</param-value>
    </context-param>
    <!-- Servlet initialization parameters -->
</web-app>
```

This time, we've used the *<context-param>*, *<param-name>*, and *<param-value>* tags to define the *province* and *country* context parameters.

Of course, we need to refactor the *UserServlet* class so that it can fetch these parameters and pass them on to the result page.

Here are the servlet's relevant sections:

```
@WebServlet(name = "UserServlet", urlPatterns = {"/userServlet"})
public class UserServlet extends HttpServlet {
    // ...

    protected void processRequest(
      HttpServletRequest request,
      HttpServletResponse response)
      throws ServletException, IOException {

        request.setAttribute("name", getRequestParameter(request, "name"));
        request.setAttribute("email", getRequestParameter(request, "email"));
        request.setAttribute("province", getContextParameter("province"));
        request.setAttribute("country", getContextParameter("country"));
    }

    protected String getContextParameter(String name) {-
        return getServletContext().getInitParameter(name);
    }
}
```

Please notice the *getContextParameter()* method implementation, which **first gets the servlet context through *getServletContext()*, and then fetches the context parameter with the *getInitParameter()* method**.

Next, we need to refactor the *"result.jsp"* file so that it can display the context parameters along with the servlet-specific parameters:

```
<p><strong>Name:</strong> ${name}</p>
<p><strong>Email:</strong> ${email}</p>
<p><strong>Province:</strong> ${province}</p>
<p><strong>Country:</strong> ${country}</p>
```

Lastly, we can redeploy the application and execute it once again.

If the user fills the HTML form with a name and an email, then it will display this data along with the context parameters:

```
User Information
Name: the user's name
Email: the user's email
Province: Mendoza
Country: Argentina
```

Otherwise, it would output the servlet and context initialization parameters:

```
User Information
Name: Not provided
Email: Not provided
Province: Mendoza
Country: Argentina
```

While the example is contrived, it shows **how to use context initialization parameters to store immutable global data**.

As the data is bound to the application context, rather than to a particular servlet, we can access them from one or multiple servlets, using the *getServletContext()* and *getInitParameter()* methods.

4. Conclusion

In this article, **we learned the key concepts of context and servlet initialization parameters and how to define them and access them using annotations and the *"web.xml"* file**.

For quite some time, there has been a strong tendency in Java to get rid of XML configuration files and migrate to annotations whenever possible.

CDI, Spring, Hibernate, to name a few, are glaring examples of this.

Nevertheless, there's nothing inherently wrong with using the *"web.xml"* file for defining context and servlet initialization parameters.

Even though the Servlet API has evolved at pretty fast pace toward this tendency, **we still need to use the deployment descriptor for defining context initialization parameters**.