Spring Session - Spring Boot

Rob Winch, Vedran Pavić

Version 2.3.0.RELEASE

Table of Contents

This guide describes how to use Spring Session to transparently leverage a relational database to back a web application's `HttpSession` when you use Spring Boot.

NOTE You can find the completed guide in the httpsession-jdbc-boot sample application.

Index

Updating Dependencies

Before you use Spring Session, you must update your dependencies. We assume you are working with a working Spring Boot web application. If you use Maven, you must add the following dependencies:

pom.xml

```xml
<dependencies>
    <!-- ... -->

    <dependency>
        <groupId>org.springframework.session</groupId>
        <artifactId>spring-session-jdbc</artifactId>
    </dependency>
</dependencies>
```

Spring Boot provides dependency management for Spring Session modules, so you need not explicitly declare the dependency version.

Spring Boot Configuration

After adding the required dependencies, we can create our Spring Boot configuration. Thanks to first-class auto configuration support, setting up Spring Session backed by a relational database is as simple as adding a single configuration property to your `application.properties`. The following listing shows how to do so:

src/main/resources/application.properties

```
spring.session.store-type=jdbc # Session store type.
```

If a single Spring Session module is present on the classpath, Spring Boot uses that store implementation automatically. If you have more than one implementation, you must choose the StoreType that you wish to use to store the sessions, as shows above.

Under the hood, Spring Boot applies configuration that is equivalent to manually adding the `@EnableJdbcHttpSession` annotation. This creates a Spring bean with the name of `springSessionRepositoryFilter`. That bean implements `Filter`. The filter is in charge of replacing the `HttpSession` implementation to be backed by Spring Session.

You can further customize by using `application.properties`. The following listing shows how to do so:

src/main/resources/application.properties

```
server.servlet.session.timeout= # Session timeout. If a duration suffix is not
specified, seconds are used.
spring.session.jdbc.initialize-schema=embedded # Database schema initialization
mode.
spring.session.jdbc.schema=classpath:org/springframework/session/jdbc/schema-
@@platform@@.sql # Path to the SQL file to use to initialize the database schema.
spring.session.jdbc.table-name=SPRING_SESSION # Name of the database table used to
store sessions.
```

For more information, see the Spring Session portion of the Spring Boot documentation.

Configuring the `DataSource`

Spring Boot automatically creates a `DataSource` that connects Spring Session to an embedded instance of an H2 database. In a production environment, you need to update your configuration to point to your relational database. For example, you can include the following in your application.properties:

src/main/resources/application.properties

```
spring.datasource.url= # JDBC URL of the database.
spring.datasource.username= # Login username of the database.
spring.datasource.password= # Login password of the database.
```

For more information, see the Configure a DataSource portion of the Spring Boot documentation.

Servlet Container Initialization

Our Spring Boot Configuration created a Spring bean named `springSessionRepositoryFilter` that implements `Filter`. The `springSessionRepositoryFilter` bean is responsible for replacing the `HttpSession` with a custom implementation that is backed by Spring Session.

In order for our `Filter` to do its magic, Spring needs to load our `Config` class. Last, we need to ensure that our Servlet Container (that is, Tomcat) uses our `springSessionRepositoryFilter` for every request. Fortunately, Spring Boot takes care of both of these steps for us.

`httpsession-jdbc-boot` Sample Application

The httpsession-jdbc-boot Sample Application demonstrates how to use Spring Session to transparently leverage an H2 database to back a web application's `HttpSession` when you use Spring Boot.

Running the `httpsession-jdbc-boot` Sample Application

You can run the sample by obtaining the source code and invoking the following command:

```
$ ./gradlew :spring-session-sample-boot-jdbc:bootRun
```

You should now be able to access the application at http://localhost:8080/

Exploring the Security Sample Application

You can now try using the application. To do so, enter the following to log in:

- **Username** *user*
- **Password** *password*

Now click the **Login** button. You should now see a message indicating that your are logged in with the user entered previously. The user's information is stored in the H2 database rather than Tomcat's `HttpSession` implementation.

How Does It Work?

Instead of using Tomcat's `HttpSession`, we persist the values in the H2 database. Spring Session replaces the `HttpSession` with an implementation that is backed by a relational database. When Spring Security's `SecurityContextPersistenceFilter` saves the `SecurityContext` to the `HttpSession`, it is then persisted into the H2 database.

When a new `HttpSession` is created, Spring Session creates a cookie named `SESSION` in your browser. That cookie contains the ID of your session. You can view the cookies (with Chrome or Firefox).

You can remove the session by using the H2 web console available at: http://localhost:8080/h2-console/ (use `jdbc:h2:mem:testdb` for JDBC URL).

Now you can visit the application at http://localhost:8080/ and see that we are no longer authenticated.