Spring Session - HttpSession (Quick Start)

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` with Java Configuration.

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

Index

Updating Dependencies

Before you use Spring Session, you must update your dependencies. If you use Maven, you must add the following dependencies:

pom.xml

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

    <dependency>
        <groupId>org.springframework.session</groupId>
        <artifactId>spring-session-jdbc</artifactId>
        <version>2.3.0.RELEASE</version>
        <type>pom</type>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
        <version>5.2.6.RELEASE</version>
    </dependency>
</dependencies>
```

Spring Java Configuration

After adding the required dependencies, we can create our Spring configuration. The Spring configuration is responsible for creating a Servlet Filter that replaces the `HttpSession` implementation with an implementation backed by Spring Session. To do so, add the following Spring Configuration:

```
@EnableJdbcHttpSession ①
public class Config {

    @Bean
    public EmbeddedDatabase dataSource() {
        return new EmbeddedDatabaseBuilder() ②

.setType(EmbeddedDatabaseType.H2).addScript("org/springframework/session/jdbc/sche
ma-h2.sql").build();
    }

    @Bean
    public PlatformTransactionManager transactionManager(DataSource dataSource) {
        return new DataSourceTransactionManager(dataSource); ③
    }

}
```

① The `@EnableJdbcHttpSession` annotation 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. In this instance, Spring Session is backed by a relational database.

② We create a `dataSource` that connects Spring Session to an embedded instance of an H2 database. We configure the H2 database to create database tables by using the SQL script that is included in Spring Session.

③ We create a `transactionManager` that manages transactions for previously configured `dataSource`.

For additional information on how to configure data access related concerns, see the Spring Framework Reference Documentation.

Java Servlet Container Initialization

Our Spring 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 Session provides a utility class named `AbstractHttpSessionApplicationInitializer` to make both of these steps easy. The following example shows how to do so:

src/main/java/sample/Initializer.java

```
public class Initializer extends AbstractHttpSessionApplicationInitializer { ①

    public Initializer() {
        super(Config.class); ②
    }

}
```

NOTE The name of our class (Initializer) does not matter. What is important is that we extend `AbstractHttpSessionApplicationInitializer`.

① The first step is to extend `AbstractHttpSessionApplicationInitializer`. Doing so ensures that the Spring bean named `springSessionRepositoryFilter` is registered with our Servlet Container for every request.

② `AbstractHttpSessionApplicationInitializer` also provides a mechanism to ensure Spring loads our `Config`.

`httpsession-jdbc` Sample Application

This section describes how to work with the `httpsession-jdbc` Sample Application.

Running the `httpsession-jdbc` Sample Application

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

```
$ ./gradlew :spring-session-sample-javaconfig-jdbc:tomcatRun
```

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

Exploring the `httpsession-jdbc` Sample Application

Now you can try using the application. To do so, fill out the form with the following information:

- **Attribute Name:** *username*
- **Attribute Value:** *rob*

Now click the **Set Attribute** button. You should now see the values displayed in the table.

How Does It Work?

We interact with the standard `HttpSession` in the `SessionServlet` shown in the following listing:

src/main/java/sample/SessionServlet.java

```
@WebServlet("/session")
public class SessionServlet extends HttpServlet {

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
IOException {
        String attributeName = req.getParameter("attributeName");
        String attributeValue = req.getParameter("attributeValue");
        req.getSession().setAttribute(attributeName, attributeValue);
        resp.sendRedirect(req.getContextPath() + "/");
    }

    private static final long serialVersionUID = 2878267318695777395L;

}
```

Instead of using Tomcat's `HttpSession`, we persist the values in H2 database. 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).

If you like, 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 the attribute we added is no longer displayed.