Spring Session - HttpSession (Quick Start)

Rob Winch

Version 2.3.0.RELEASE

Table of Contents

This guide describes how to use Spring Session to transparently leverage Redis to back a web application's `HttpSession` with Java Configuration.

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

Index

Updating Dependencies

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

pom.xml

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

    <dependency>
        <groupId>org.springframework.session</groupId>
        <artifactId>spring-session-data-redis</artifactId>
        <version>2.3.0.RELEASE</version>
        <type>pom</type>
    </dependency>
    <dependency>
        <groupId>io.lettuce</groupId>
        <artifactId>lettuce-core</artifactId>
        <version>5.2.2.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
        <version>5.2.6.RELEASE</version>
    </dependency>
</dependencies>
```

Since we are using a SNAPSHOT version, we need to ensure to add the Spring Snapshot Maven Repository. You must have the following in your pom.xml:

pom.xml

```
<repositories>

    <!-- ... -->

    <repository>
        <id>spring-snapshot</id>
        <url>https://repo.spring.io/libs-snapshot</url>
    </repository>
</repositories>
```

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:

```
@EnableRedisHttpSession ①
public class Config {

    @Bean
    public LettuceConnectionFactory connectionFactory() {
        return new LettuceConnectionFactory(); ②
    }

}
```

① The `@EnableRedisHttpSession` annotation creates a Spring Bean with the name of `springSessionRepositoryFilter` that 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 Redis.

② We create a `RedisConnectionFactory` that connects Spring Session to the Redis Server. We configure the connection to connect to localhost on the default port (6379). For more information on configuring Spring Data Redis, see the 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 shows an example:

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 by the name of `springSessionRepositoryFilter` is registered with our Servlet Container for every request.

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

httpsession Sample Application

Running the `httpsession` Sample Application

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

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

NOTE	For the sample to work, you must install Redis 2.8+ on localhost and run it with the default port (6379). Alternatively, you can update the `RedisConnectionFactory` to point to a Redis server. Another option is to use Docker to run Redis on localhost. See Docker Redis repository for detailed instructions.

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

Exploring the `httpsession` Sample Application

Now you can try to use 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 Redis. 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 redis-cli. For example, on a Linux based system you can type the following:

```
    $ redis-cli keys '*' | xargs redis-cli del
```

TIP The Redis documentation has instructions for installing redis-cli.

Alternatively, you can also delete the explicit key. Enter the following into your terminal, being sure to replace `7e8383a4-082c-4ffe-a4bc-c40fd3363c5e` with the value of your SESSION cookie:

```
    $ redis-cli del spring:session:sessions:7e8383a4-082c-4ffe-a4bc-c40fd3363c5e
```

Now you can visit the application at http://localhost:8080/ and see that the attribute we added is no longer displayed.