Spring Session and Spring Security with Hazelcast

Tommy Ludwig, Rob Winch

Version 2.3.0.RELEASE

Table of Contents

This guide describes how to use Spring Session along with Spring Security when you use Hazelcast as your data store. It assumes that you have already applied Spring Security to your application.

NOTE	You cand find the completed guide in the Hazelcast Spring Security 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>com.hazelcast</groupId>
        <artifactId>hazelcast</artifactId>
        <version>3.12.7</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
        <version>5.2.6.RELEASE</version>
    </dependency>
</dependencies>
```

Spring 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:

```
@EnableHazelcastHttpSession ①
@Configuration
public class HazelcastHttpSessionConfig {

    @Bean
    public HazelcastInstance hazelcastInstance() {
        Config config = new Config();
        MapAttributeConfig attributeConfig = new MapAttributeConfig()

.setName(HazelcastIndexedSessionRepository.PRINCIPAL_NAME_ATTRIBUTE)
                .setExtractor(PrincipalNameExtractor.class.getName());

config.getMapConfig(HazelcastIndexedSessionRepository.DEFAULT_SESSION_MAP_NAME) ②
                .addMapAttributeConfig(attributeConfig).addMapIndexConfig(
                        new
MapIndexConfig(HazelcastIndexedSessionRepository.PRINCIPAL_NAME_ATTRIBUTE,
false));
        return Hazelcast.newHazelcastInstance(config); ③
    }

}
```

① The `@EnableHazelcastHttpSession` annotation creates a Spring bean named `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 Hazelcast.

② In order to support retrieval of sessions by principal name index, an appropriate `ValueExtractor` needs to be registered. Spring Session provides `PrincipalNameExtractor` for this purpose.

③ We create a `HazelcastInstance` that connects Spring Session to Hazelcast. By default, the application starts and connects to an embedded instance of Hazelcast. For more information on configuring Hazelcast, see the reference documentation.

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 `SessionConfig` class. Since our application is already loading Spring configuration by using our `SecurityInitializer` class, we can add our `SessionConfig` class to it. The following listing shows how to do so:

src/main/java/sample/SecurityInitializer.java

```
public class SecurityInitializer extends AbstractSecurityWebApplicationInitializer
{

    public SecurityInitializer() {
        super(SecurityConfig.class, SessionConfig.class);
    }

}
```

Last, we need to ensure that our Servlet Container (that is, Tomcat) uses our `springSessionRepositoryFilter` for every request. It is extremely important that Spring Session's `springSessionRepositoryFilter` is invoked before Spring Security's `springSecurityFilterChain`. Doing so ensures that the `HttpSession` that Spring Security uses is backed by Spring Session. Fortunately, Spring Session provides a utility class named `AbstractHttpSessionApplicationInitializer` that makes this doing so easy. The following example shows how to do so:

src/main/java/sample/Initializer.java

```
public class Initializer extends AbstractHttpSessionApplicationInitializer {

}
```

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

By extending `AbstractHttpSessionApplicationInitializer`, we ensure that the Spring Bean named `springSessionRepositoryFilter` is registered with our servlet container for every request before Spring Security's `springSecurityFilterChain`.

Hazelcast Spring Security Sample Application

This section describes how to work with the Hazelcast Spring Security sample application.

Running the Sample Application

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

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

NOTE	By default, Hazelcast runs in embedded mode with your application. However, if you want to connect to a standalone instance instead, you can configure it by following the instructions in the reference documentation.

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 Hazelcast rather than Tomcat's `HttpSession` implementation.

How Does It Work?

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

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).

Interacting with the Data Store

You can remove the session by using a Java client, one of the other clients, or the management center.

Using the Console

For example, to remove the session by using the management center console after connecting to your Hazelcast node, run the following commands:

```
default> ns spring:session:sessions
spring:session:sessions> m.clear
```

TIP The Hazelcast documentation has instructions for the console.

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

```
spring:session:sessions> m.remove 7e8383a4-082c-4ffe-a4bc-c40fd3363c5e
```

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

Using the REST API

As described in the section of the documentation that cover other clients, there is a REST API provided by the Hazelcast node(s).

For example, you could delete an individual key as follows (being sure to replace `7e8383a4-082c-4ffe-a4bc-c40fd3363c5e` with the value of your SESSION cookie):

```
$ curl -v -X DELETE
http://localhost:xxxxx/hazelcast/rest/maps/spring:session:sessions/7e8383a4-082c-
4ffe-a4bc-c40fd3363c5e
```

TIP The port number of the Hazelcast node is printed to the console on startup. Replace `xxxxx` with the port number.

Now you can see that you are no longer authenticated with this session.