

Spring Session - HttpSession (Quick Start)

Rob Winch

Version 2.3.0.RELEASE

Table of Contents

Updating Dependencies	2
Spring XML Configuration	3
XML Servlet Container Initialization	4
httpsession-xml Sample Application	6
Running the httpsession-xml Sample Application	6
Exploring the httpsession-xml Sample Application	6
How Does It Work?	6

This guide describes how to use Spring Session to transparently leverage Redis to back a web application's `HttpSession` with XML-based configuration.

NOTE | You can find the completed guide in the [httpsession-xml 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-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>
```

Spring XML 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:

src/main/webapp/WEB-INF/spring/session.xml

```
①
<context:annotation-config/>
<bean
class="org.springframework.session.data.redis.config.annotation.web.http.RedisHttp
SessionConfiguration"/>

②
<bean
class="org.springframework.data.redis.connection.lettuce.LettuceConnectionFactory"
/>
```

- ① We use the combination of `<context:annotation-config/>` and `RedisHttpSessionConfiguration` because Spring Session does not yet provide XML Namespace support (see [gh-104](#)). This 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](#).

XML 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, we need to instruct Spring to load our `session.xml` configuration. We can do so with the following configuration:

src/main/webapp/WEB-INF/web.xml

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>
    /WEB-INF/spring/session.xml
  </param-value>
</context-param>
<listener>
  <listener-class>
    org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>
```

The `ContextLoaderListener` reads the `contextConfigLocation` and picks up our `session.xml` configuration.

Last, we need to ensure that our Servlet Container (that is, Tomcat) uses our `springSessionRepositoryFilter` for every request. The following snippet performs this last step for us:

src/main/webapp/WEB-INF/web.xml

```
<filter>
  <filter-name>springSessionRepositoryFilter</filter-name>
  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-
class>
</filter>
<filter-mapping>
  <filter-name>springSessionRepositoryFilter</filter-name>
  <url-pattern>/*</url-pattern>
  <dispatcher>REQUEST</dispatcher>
  <dispatcher>ERROR</dispatcher>
</filter-mapping>
```

The `DelegatingFilterProxy` looks up a Bean by the name of `springSessionRepositoryFilter` and cast it to a `Filter`. For every request that `DelegatingFilterProxy` is invoked, the

springSessionRepositoryFilter is invoked.

httpsession-xml Sample Application

This section describes how to work with the `httpsession-xml` sample application.

Running the `httpsession-xml` Sample Application

You can run the sample by obtaining the [source code](#) and invoking the following command:

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.

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

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

Exploring the `httpsession-xml` Sample Application

Now you can try using the application. 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

```
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 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. To do so, 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.