Spring Session - HttpSession (Quick Start)

Rob Winch, Vedran Pavić

Version 2.1.5.RELEASE

Table of Contents

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

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

1. 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

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

    <dependency>
        <groupId>org.springframework.session</groupId>
        <artifactId>spring-session-jdbc</artifactId>
        <version>2.1.5.RELEASE</version>
        <type>pom</type>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
        <version>5.1.6.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. The following listing shows how to add the following Spring Configuration:

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

XML

```xml
1
<context:annotation-config/>
<bean
class="org.springframework.session.jdbc.config.annotation.web.http.JdbcHttpSessionConfigura
tion"/>

2
<jdbc:embedded-database id="dataSource" database-name="testdb" type="H2">
    <jdbc:script location="classpath:org/springframework/session/jdbc/schema-h2.sql"/>
</jdbc:embedded-database>

3
<bean class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <constructor-arg ref="dataSource"/>
</bean>
```

We use the combination of `<context:annotation-config/>` and `JdbcHttpSessionConfiguration` because Spring Session does not yet provide XML Namespace support (see gh-104 (https://github.com/spring-projects/spring-session/issues/104)).

1 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. 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 (https://docs.spring.io/spring/docs/5.1.6.RELEASE/spring-framework-reference/data-access.html).

== 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 do so with the following configuration:

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

XML

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

The `ContextLoaderListener` (https://docs.spring.io/spring/docs/5.1.6.RELEASE/spring-framework-reference/core.html#context-create) 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

XML

```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` (https://docs.spring.io/spring-framework/docs/5.1.6.RELEASE/javadoc-api/org/springframework/web/filter/DelegatingFilterProxy.html) looks up a bean named `springSessionRepositoryFilter` and casts it to a `Filter`. For

every request on which `DelegatingFilterProxy` is invoked, the `springSessionRepositoryFilter` is invoked.
== `httpsession-jdbc-xml` Sample Application

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

=== Running the `httpsession-jdbc-xml` Sample Application

You can run the sample by obtaining the source code (https://github.com/spring-projects/spring-session/archive/2.1.5.RELEASE.zip) and invoking the following command:

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

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

=== Exploring the `httpsession-jdbc-xml` 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 following `SessionServlet` :

src/main/java/sample/SessionServlet.java

JAVA

```java
public class SessionServlet extends HttpServlet {

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, 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 the 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 (https://developers.google.com/web/tools/chrome-devtools/manage-data/cookies) or Firefox (https://developer.mozilla.org/en-US/docs/Tools/Storage_Inspector)).

You can remove the session by using 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 observe that the attribute we added is no longer displayed.

Version 2.1.5.RELEASE
Last updated 2019-03-29 12:38:10 +00:00