Spring Session - Custom Cookie

Rob Winch, Eleftheria Stein-Kousathana

Version 2.3.0.RELEASE

Table of Contents

This guide describes how to configure Spring Session to use custom cookies with Java Configuration. The guide assumes you have already set up Spring Session in your project using your chosen data store. For example, HttpSession with Redis.

NOTE You can find the completed guide in the Custom Cookie sample application.

Index

Spring Java Configuration

Once you have set up Spring Session, you can customize how the session cookie is written by exposing a `CookieSerializer` as a Spring bean. Spring Session comes with `DefaultCookieSerializer`. Exposing the `DefaultCookieSerializer` as a Spring bean augments the existing configuration when you use configurations like `@EnableRedisHttpSession`. The following example shows how to customize Spring Session's cookie:

```
@Bean
public CookieSerializer cookieSerializer() {
    DefaultCookieSerializer serializer = new DefaultCookieSerializer();
    serializer.setCookieName("JSESSIONID"); ①
    serializer.setCookiePath("/"); ②
    serializer.setDomainNamePattern("^.+?\\.(\\w+\\.[a-z]+)$"); ③
    return serializer;
}
```

① We customize the name of the cookie to be `JSESSIONID`.

② We customize the path of the cookie to be `/` (rather than the default of the context root).

③ We customize the domain name pattern (a regular expression) to be `^.+?\\.(\\w+\\.[a-z]+)$`. This allows sharing a session across domains and applications. If the regular expression does not match, no domain is set and the existing domain is used. If the regular expression matches, the first grouping is used as the domain. This means that a request to https://child.example.com sets the domain to `example.com`. However, a request to http://localhost:8080/ or https://192.168.1.100:8080/ leaves the cookie unset and, thus, still works in development without any changes being necessary for production.

WARNING You should only match on valid domain characters, since the domain name is reflected in the response. Doing so prevents a malicious user from performing such attacks as HTTP Response Splitting.

Configuration Options

The following configuration options are available:

- `cookieName`: The name of the cookie to use. Default: `SESSION`.
- `useSecureCookie`: Specifies whether a secure cookie should be used. Default: Use the value of `HttpServletRequest.isSecure()` at the time of creation.
- `cookiePath`: The path of the cookie. Default: The context root.
- `cookieMaxAge`: Specifies the max age of the cookie to be set at the time the session is created. Default: `-1`, which indicates the cookie should be removed when the browser is closed.
- `jvmRoute`: Specifies a suffix to be appended to the session ID and included in the cookie. Used to identify which JVM to route to for session affinity. With some implementations (that is, Redis) this option provides no performance benefit. However, it can help with tracing logs of a particular user.
- `domainName`: Allows specifying a specific domain name to be used for the cookie. This option is simple to understand but often requires a different configuration between development and production environments. See `domainNamePattern` as an alternative.
- `domainNamePattern`: A case-insensitive pattern used to extract the domain name from the `HttpServletRequest#getServerName()`. The pattern should provide a single grouping that is used to extract the value of the cookie domain. If the regular expression does not match, no domain is set and the existing domain is used. If the regular expression matches, the first grouping is used as the domain.
- `sameSite`: The value for the `SameSite` cookie directive. To disable the serialization of the `SameSite` cookie directive, you may set this value to `null`. Default: `Lax`

WARNING You should only match on valid domain characters, since the domain name is reflected in the response. Doing so prevents a malicious user from performing such attacks as HTTP Response Splitting.

custom-cookie Sample Application

This section describes how to work with the `custom-cookie` sample application.

Running the custom-cookie Sample Application

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

```
$ ./gradlew :spring-session-sample-javaconfig-custom-cookie: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 custom-cookie Sample Application

Now you can use 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.

If you look at the cookies for the application, you can see the cookie is saved to the custom name of `JSESSIONID`.