Spring Session - find by username

Rob Winch

Version 2.3.0.RELEASE

Table of Contents

This guide describes how to use Spring Session to find sessions by username.

NOTE	You can find the completed guide in the findbyusername application.

Index

Assumptions

The guide assumes you have already added Spring Session to your application by using the built-in Redis configuration support. The guide also assumes you have already applied Spring Security to your application. However, we the guide is somewhat general purpose and can be applied to any technology with minimal changes, which we discuss later in the guide.

NOTE	If you need to learn how to add Spring Session to your project, see the listing of samples and guides

About the Sample

Our sample uses this feature to invalidate the users session that might have been compromised. Consider the following scenario:

- User goes to library and authenticates to the application.
- User goes home and realizes they forgot to log out.
- User can log in and terminate the session from the library using clues like the location, created time, last accessed time, and so on.

Would it not be nice if we could let the user invalidate the session at the library from any device with which they authenticate? This sample demonstrates how this is possible.

Using `FindByIndexNameSessionRepository`

To look up a user by their username, you must first choose a `SessionRepository` that implements `FindByIndexNameSessionRepository`. Our sample application assumes that the Redis support is already set up, so we are ready to go.

Mapping the User Name

`FindByIndexNameSessionRepository` can find a session only by the user name if the developer instructs Spring Session what user is associated with the `Session`. You can do so by ensuring that the session attribute with the name `FindByUsernameSessionRepository.PRINCIPAL_NAME_INDEX_NAME` is populated with the username.

Generally speaking, you can do so with the following code immediately after the user authenticates:

```
String username = "username";
this.session.setAttribute(FindByIndexNameSessionRepository.PRINCIPAL_NAME_INDEX_NA
ME, username);
```

Mapping the User Name with Spring Security

Since we use Spring Security, the user name is automatically indexed for us. This means we need not perform any steps to ensure the user name is indexed.

Adding Additional Data to the Session

It may be nice to associate additional information (such as the IP Address, the browser, location, and other details) to the session. Doing so makes it easier for the user to know which session they are looking at.

To do so, determine which session attribute you want to use and what information you wish to provide. Then create a Java bean that is added as a session attribute. For example, our sample application includes the location and access type of the session, as the following listing shows:

```
public class SessionDetails implements Serializable {

    private String location;

    private String accessType;

    public String getLocation() {
        return this.location;
    }

    public void setLocation(String location) {
        this.location = location;
    }

    public String getAccessType() {
        return this.accessType;
    }

    public void setAccessType(String accessType) {
        this.accessType = accessType;
    }

    private static final long serialVersionUID = 8850489178248613501L;

}
```

We then inject that information into the session on each HTTP request using a `SessionDetailsFilter`, as the following example shows:

```
@Override
public void doFilterInternal(HttpServletRequest request, HttpServletResponse
response, FilterChain chain)
        throws IOException, ServletException {
    chain.doFilter(request, response);

    HttpSession session = request.getSession(false);
    if (session != null) {
        String remoteAddr = getRemoteAddress(request);
        String geoLocation = getGeoLocation(remoteAddr);

        SessionDetails details = new SessionDetails();
        details.setAccessType(request.getHeader("User-Agent"));
        details.setLocation(remoteAddr + " " + geoLocation);

        session.setAttribute("SESSION_DETAILS", details);
    }
}
```

We obtain the information we want and then set the `SessionDetails` as an attribute in the `Session`. When we retrieve the `Session` by user name, we can then use the session to access our `SessionDetails` as we would any other session attribute.

NOTE You might wonder why Spring Session does not provide `SessionDetails` functionality out of the box. We have two reasons. The first reason is that it is very trivial for applications to implement this themselves. The second reason is that the information that is populated in the session (and how frequently that information is updated) is highly application-dependent.

Finding sessions for a specific user

We can now find all the sessions for a specific user. The following example shows how to do so:

```
@Autowired
FindByIndexNameSessionRepository<? extends Session> sessions;

@RequestMapping("/")
public String index(Principal principal, Model model) {
    Collection<? extends Session> usersSessions =
this.sessions.findByPrincipalName(principal.getName()).values();
    model.addAttribute("sessions", usersSessions);
    return "index";
}
```

In our instance, we find all sessions for the currently logged in user. However, you can modify this for an administrator to use a form to specify which user to look up.

`findbyusername` Sample Application

This section describes how to use the `findbyusername` sample application.

Running the `findbyusername` Sample Application

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

```
$ ./gradlew :spring-session-sample-boot-findbyusername:bootRun
```

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 security Sample Application

You can now try using the application. Enter the following to log in:

- **Username** *user*
- **Password** *password*

Now click the **Login** button. You should now see a message indicating your are logged in with the user entered previously. You should also see a listing of active sessions for the currently logged in user.

You can emulate the flow we discussed in the About the Sample section by doing the following:

- Open a new incognito window and navigate to http://localhost:8080/
- Enter the following to log in:
  - **Username** *user*
  - **Password** *password*
- Terminate your original session.
- Refresh the original window and see that you are logged out.