Java Authentication Guide with Apache Shiro

Table of Contents

Authentication is the process of identity verification– you are trying to prove a user is who they say they are. To do so, a user needs to provide some sort of proof of identity that your system understands and trust.

The goal of this guide is to walk you through how Authentication in Java is performed in Shiro. If you haven't already please take moment and go through Shiro's 10-Minute Tutorial (/10-minute-tutorial.html) so that you get a basic understanding of how to work with Shiro.

Terminology you'll need

Subject

Security specific user 'view' of an application user. It can be a human being, a third-party process, a server connecting to your application, or even a cron job. Basically, it is anything or anyone communicating with your application.

Principals

A subjects identifying attributes. First name, last name, social security number, username

Credentials

secret data that are used to verify identities. Passwords, Biometric data, x509 certificates,

Realms

Security specific DAO, data access object, software component that talks to a backend data source. If you have usernames and password in LDAP, then you would have an LDAP Realm that would communicate with LDAP. The idea is that you would use a realm per back-end data source and Shiro would know how to coordinate with these realms together to do what you have to do.

How to Authenticate in Java with Shiro

In Shiro's framework, and almost every other framework for that matter, the Java authentication process can be broken up into three distinct steps.

1. Collect the subject's principals and credentials
2. Submit the principals and credentials to an authentication system.
3. Allow access, retry authentication, or block access

Here is some code on how you do this in Shiro Specifically.

Step 1 - Collect the subject's principals and credentials

```
//Example using most common scenario:
//String username and password.  Acquire in
//system-specific manner (HTTP request, GUI, etc)
UsernamePasswordToken token = new UsernamePasswordToken( username, password );

//"Remember Me" built-in, just do this:
token.setRememberMe(true);
```

In this particular case, we're using a class called UsernamePasswordToken (/static/current/apidocs/org/apache/shiro/authc/UsernamePasswordToken.html). It is the most common authentication token used in the framework.

We use this token to bundle the username and password we acquired in someway in our Java application. Maybe they were submitted via a user web form, an HTTP header, or a command line. In Shiro, it does not matter how you acquire them – it is protocol agnostic.

In this example, we have decided that we want the application to remember users when they return. So once the token is created, we use Shiro's built-in "Remember-me" feature by setting it to true on the token. This is done using the token's `setRememberMe()` (/static/current/apidocs/org/apache/shiro/authc/UsernamePasswordToken.html#setRememberMe(boolean)) method

Step 2 - Submit the principals and credentials to an authentication system.

So we've collected the information in a token and set it to remember returning users. The next step is in the Authentication process is to submit the token to an authentication system. Your authentication system is represented in Shiro by security-specific DAOs, that are referred to as Realms (/static/current/apidocs/). For more information on realms please check out the Shiro Realm Guide (/realm.html).

In Shiro, we try to make this part as quick and easy as humanly possible. We have it down to one line of Java code!

```
//With most of Shiro, you'll always want to make sure you're working with the currently
//executing user, referred to as the subject
Subject currentUser = SecurityUtils.getSubject();

//Authenticate the subject by passing
//the user name and password token
//into the login method
currentUser.login(token);
```

First, we need to acquire the currently executing user, referred to as the subject. A subject is just a security specific view of the user—-it can be a human, a process, cron job, doesn't matter. In Shiro, there is always a subject instance available to the currently executing thread. The concept of a subject is core to Shiro and most of the framework is centered around working with subjects. In this example, we will name this instance of subject currentUser.

To acquire the subject, we use the SecurityUtils (/static/current/apidocs/org/apache/shiro/SecurityUtils.html) class which is also a core part of Shiro's API. It will acquire the currently executing user via the `getsubject()` (/static/current/apidocs/org/apache/shiro/SecurityUtils.html#getSubject()) method call. And we get back a subject instance that is representing who the current user is who is interacting with the system. At this point in the example, the subject currentUser is anonymous. There is no identity associated with them.

Now with the user representation in hand, we authenticate them by just calling the `login()` (/static/current/apidocs/org/apache/shiro/subject/Subject.html#login(org.apache.shiro.authc.AuthenticationToken))) method and submit the token we just constructed a second ago.

Step 3 - Allow access, retry authentication, or block access

Again really, really easy, single method call. If the `login()` method call is successful, then the user is logged in and associated with a user account or identity. From here, the user can go about using your application and retain their identity through their session or longer since we have set the "Remember Me" in our example.

But what happens if something fails in the authentication attempt? What if they give you the wrong password, or they accessed the system too many times, maybe their account is locked? In this case, Shiro will throw an exception. This is where Shiro's rich exception hierarchy comes into play.

```
try {
    currentUser.login(token);
} catch  ( UnknownAccountException uae ) { ...
} catch  ( IncorrectCredentialsException ice ) { ...
} catch  ( LockedAccountException lae ) { ...
} catch  ( ExcessiveAttemptsException eae ) { ...
} ...  your own ...
} catch ( AuthenticationException ae ) {
    //unexpected error?
}
//No problems, show authenticated view…
```

You can take that method call and wrap it in a try/catch block, and you can catch all sort of exceptions if you want to handle them and react accordingly. In addition to a rich set of exceptions that Shiro offers, you can create your own if you need custom functionality. For more information, follow this link documentation on AuthenticationException (/static/current/apidocs/org/apache/shiro/authc/AuthenticationException.html).

Security Tip

Security best practice is to give generic login failure messages to users because you do not want to aid an attacker trying to break into your system.

"Remember Me" Support

As shown in the example above, Shiro supports the notion of "remember me" in addition to the normal login process.

In Shiro, the Subject object supports two methods : `isRemembered()` (/static/current/apidocs/org/apache/shiro/subject/Subject.html#isRemembered()) and `isAuthenticated()` (/static/current/apidocs/org/apache/shiro/subject/Subject.html#isAuthenticated()).

A "remembered" subject has an identity (it is not anonymous) and their identifying attributes,referred to as principals, are remembered from a successful authentication during a previous session.

An authenticated subject has proved their identity *during their current session*.

> If a subject is remembered, it DOES NOT mean they are authenticated.

Remembered vs Authenticated

In shiro it is very important to note that a remembered subject is not an authenticated subject. A check against `isAuthenticated()` is a much more strict check because authentication is the process of proving you are who you say you are. When a user is only remembered, the remembered identity gives the system an idea who that user probably is, but in reality, has no way of absolutely guaranteeing if the remembered Subject represents the user currently using the application. Once the subject is authenticated, they are no longer considered only remembered because their identity would have been verified during the current session.

So although many parts of the application can still perform user-specific logic based on the remembered principals, such as customized views, it should never perform highly-sensitive operations until the user has legitimately verified their identity by executing a successful authentication attempt.

For example, a check to see if a subject can access financial information should almost always depend on `isAuthenticated()`, not `isRemembered()`, to guarantee a verified identity.

Here is a scenario to help illustrate why the distinction between isAuthenticated and isRemembered is important.

Let's say you're using Amazon.com. You log in, and you add some books to your shopping cart. A day goes by. Of course your user session has expired, and you've been logged out. But Amazon "remembers" you, greets you by name, and is still giving you personalized book recommendations. To Amazon, `isRemembered()` would return `TRUE`. What happens if you try to use one of the credit cards on file or change your account information? While Amazon "remembers" you, `isRemembered() = TRUE`, it is not certain that you are in fact you, `isAuthenticated()=FALSE`. So before you can perform a sensitive action Amazon needs to verify your identity by forcing an authentication process which it does through a login screen. After the login, your identity has been verified and `isAuthenticated()=TRUE`.

This scenario happens very often over the web so the functionality is built into Shiro helping you easily make the distinction yourself.

Logging Out

Finally, when the user is done using the application, they can log out. And in Shiro, we make logging out quick and easy with a single method call.

```
currentUser.logout(); //removes all identifying information and invalidates their session too.
```

When you log out in Shiro it will close out the user session and removes any associated identity from the subject instance. If you're using RememberMe in a web environment, then `.logout()` will, by default, also delete the RememberMe cookie from the browser.

Edit this page on GitHub (https://github.com/apache/shiro-site/edit/main/src/site/content/java-authentication-guide.adoc)