Adding additional details to principal object stored in spring security context

Asked 11 years ago Modified 1 year, 11 months ago Viewed 110k times

58

I am using Spring 3.0 and Spring Security 3. I am able to authenticate a user against a database using Spring Security. Using:

```
SecurityContextHolder.getContext().getAuthentication().getPrincipal()
```

I am able to retrieve username of the current logged in user. I wish to add additional details like user id and the module accesses to the principal object stored in Spring Security context so that I can retrieve it later. How can I add additional details to the principal object and then how can I retrieve it later on a jsp or java class. Please provide an appropriate code snippet if possible.

Edit: I am using JDBC to access my database.

java spring spring-security

Share Edit Follow

edited Dec 24, 2022 at 20:19

starball
47.8k ● 28 ● 179 ● 837

asked Dec 3, 2013 at 11:13

ManeetK
685 ● 1 ● 5 ● 11

2 Creating your own `UserDetails` implementation and your own `UserDetailsService` implementation. With that you can do whatever you want. Add the properties you want etc. – M. Deinum Dec 3, 2013 at 11:32

Thanks. I have been trying to do the same but I think I am doing something wrong. @M.Deinum Could you please provide me a code snippet where this has been successfully implemented – ManeetK Dec 3, 2013 at 12:01

4 Answers

Sorted by: Highest score (default)

37

Here is what you need:

1. Extend spring `User` (`org.springframework.security.core.userdetails.User`) class and what ever properties you need.
2. Extend spring `UserDetailsService` (`org.springframework.security.core.userdetails.UserDetailsService`) and fill the above object. Override `loadUserByUsername` and return your extended user class

3. Set your custom `UserDetailsService` in `AuthenticationManagerBuilder`

For example

```
public class CurrentUser extends User{

   //This constructor is a must
    public CurrentUser(String username, String password, boolean enabled, boolean
accountNonExpired,
            boolean credentialsNonExpired, boolean accountNonLocked,
            Collection<? extends GrantedAuthority> authorities) {
        super(username, password, enabled, accountNonExpired,
credentialsNonExpired, accountNonLocked, authorities);
    }
    //Setter and getters are required
    private String firstName;
    private String lastName;

}
```

The Custom userdetails could be:

```
@Service("userDetailsService")
public class CustomUserDetailsService implements UserDetailsService {

@Override
public UserDetails loadUserByUsername(final String username) throws
UsernameNotFoundException {

    //Try to find user and its roles, for example here we try to get it from
database via a DAO object
   //Do not confuse this foo.bar.User with CurrentUser or spring User, this is a
temporary object which holds user info stored in database
    foo.bar.User user = userDao.findByUserName(username);

    //Build user Authority. some how a convert from your custom roles which are
in database to spring GrantedAuthority
    List<GrantedAuthority> authorities = buildUserAuthority(user.getUserRole());

    //The magic is happen in this private method !
    return buildUserForAuthentication(user, authorities);

}


//Fill your extended User object (CurrentUser) here and return it
private User buildUserForAuthentication(foo.bar.User user,
List<GrantedAuthority> authorities) {
    String username = user.getUsername();
    String password = user.getPassword();
    boolean enabled = true;
    boolean accountNonExpired = true;
    boolean credentialsNonExpired = true;
    boolean accountNonLocked = true;

    return new CurrentUser(username, password, enabled, accountNonExpired,
credentialsNonExpired,
            accountNonLocked, authorities);
```

```
    //If your database has more information of user for example firstname,... You
can fill it here
  //CurrentUser currentUser = new CurrentUser(....)
  //currentUser.setFirstName( user.getfirstName() );
  //.....
  //return currentUser ;
}

private List<GrantedAuthority> buildUserAuthority(Set<UserRole> userRoles) {

    Set<GrantedAuthority> setAuths = new HashSet<GrantedAuthority>();

    // Build user's authorities
    for (UserRole userRole : userRoles) {
        setAuths.add(new SimpleGrantedAuthority(userRole.getRole()));
    }

    return new ArrayList<GrantedAuthority>(setAuths);
}

}
```

Configure the spring security context

```
@Configuration
@EnableWebSecurity
@PropertySource("classpath://configs.properties")
public class SecurityContextConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    @Qualifier("userDetailsService")
    private UserDetailsService userDetailsService;

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws
Exception {
        auth.userDetailsService(userDetailsService);
    }
```

It's all done !

You can call `(CurrentUser)getAuthentication().getPrincipal()` to get you newly `CurrentUser` or set some properties.

Share Edit Follow

edited Feb 14, 2017 at 17:57

fujy

5,264 ● 5 ● 32 ● 51

answered Dec 26, 2016 at 6:21

Alireza Fattahi

45.2k ● 16 ● 131 ● 184

1 This works, but when the server restarts, we run into problems with existing sessions: java.lang.ClassCastException: be.storefront.imicloud.security.MyUserDetails cannot be cast to be.storefront.imicloud.security.MyUserDetails. How should this be handled? – Wouter Jan 10, 2017 at 10:00

@Wouter please ask a different question with the full stack trace! – Alireza Fattahi Jan 11, 2017 at 4:33

3 Solution is nice but make sure you don't cast `(CurrentUser)getAuthentication().getPrincipal()` into the controller/service methods which are accessible without authentication. Because for that `anonymous` would be returned by `getAuthentication().getPrincipal()` and hence `ClassCastException: java.lang.String cannot be cast to com.sample.CurrentUser` – The Coder Sep 26, 2019 at 19:00

I found this dangerous when you directly change the other parameters on DBs like the parameter above "firstname", the persistence framework still shows the previous value.... – Andrew Kang Sep 10, 2020 at 14:59

@Wouter For the example above, The casting exception occurs when the properties on "//Setter and getters are required" and the ones on " //If your database has more information of user for example firstname,... You can fill it here " are not in accordance. – Andrew Kang Sep 11, 2020 at 3:14

32

In order to add more details to the authenticated user. You need to first create your own implementation of the User object which should extend the spring security User object. After that you can add the properties you want to add to the authenticated user. Once this is done you need to return your implementation of the user object in `UserDetailService` (If you are not using LDAP for authentication). This link provides the details for adding more details to the authenticated user--

http://javahotpot.blogspot.com/2013/12/spring-security-adding-more-information.html

Share Edit Follow

edited Jul 4, 2022 at 6:34

Apostolos
10.5k ● 5 ● 30 ● 44

answered Dec 8, 2013 at 4:34

Yogen
344 ● 2 ● 3

Could you also post the implementation of loadUserDetails(username) in LoginService of the example link. I want to know how the failed authentication requests will be handled. Thanks in advance – ManeetK Dec 11, 2013 at 8:21

Answered at the mentioned link only..javahotpot.blogspot.in/2013/12/... – Yogen Dec 13, 2013 at 14:12

5 Your link to your blog doesn't fully answer the question, as was asked for the implantation of LoginService. Typically you should answer the question on this site, not reference a post on your blog. – Shaggy Sep 1, 2016 at 20:12

Why you said **If you are not using LDAP for authentication** ? Will this solution not work with LDAP? – The Coder May 22, 2019 at 20:01

1 because with this service you need to return the password to spring, and you cannot read password from LDAP.. instead, to use LDAP you can implement a AuthenticationProvider – Thiago Suchorski May 23, 2019 at 17:47

15

(I will assume you have a basic Spring Security configuration working and know how the basic components work together)

The most "correct" way would be providing your own implementation of `AuthenticationProvider`, that return a custom `Authentication` implementation. Then you can fill in this `Authentication` instance with everything you need. For example:

```
public class MyAuthentication extends UsernamePasswordAuthenticationToken
implements Authentication {

    public MyAuthentication(Object principal, Object credentials, int moduleCode)
{
        super(principal, credentials);
        this.moduleCode = moduleCode;
    }

    public MyAuthentication(Object principal, Object credentials,  Collection<?
extends GrantedAuthority> authorities,int moduleCode) {
        super(principal, credentials, authorities);
        this.moduleCode = moduleCode;
    }

    private int moduleCode;

    public getModuleCode() {
        return moduleCode;
    }
}


public class MyAuthenticationProvider extends DaoAuthenticationProvider {

    private Collection<GrantedAuthority> obtainAuthorities(UserDetails user) {
        // return granted authorities for user, according to your requirements
    }

    private int obtainModuleCode(UserDetails user) {
        // return moduleCode for user, according to your requirements
    }

    @Override
    public Authentication createSuccessAuthentication(Object principal,
Authentication authentication, UserDetails user) {
        // Suppose this user implementation has a moduleCode property
        MyAuthentication result = new
MyAuthentication(authentication.getPrincipal(),

authentication.getCredentials(),
                                                    obtainAuthorities(user),
                                                    obtainModuleCode(user));
        result.setDetails(authentication.getDetails());
        return result;
    }
}
```

And then, in `applicationContext.xml`:

```
<authentication-manager>
    <authentication-provider ref="myAuthenticationProvider">
</authentication-manager>
```

```
<bean id="myAuthenticationProvider" class="MyAuthenticationProvider"
scope="singleton">
    ...
</bean>
```

I guess you could get it working by providing custom implementations of `AuthenticationDetails` and `AuthenticationDetailsSource`, but I think that would be a less clean approach.

Share Edit Follow

edited Sep 1, 2016 at 17:09

Shaggy
1,464 ● 1 ● 23 ● 34

answered Dec 3, 2013 at 11:58

gpeche
22.5k ● 5 ● 39 ● 52

5

The "only" things you need to do is create your own UserDetailsService implementation which returns your own implementation of a UserDetails object.

See here for a tutorial which implements a JPA based `UserDetailsService`.

Share Edit Follow

answered Dec 3, 2013 at 11:59

M. Deinum
124k ● 22 ● 230 ● 245

I am using JDBC to connect to the database. Can you specify the changes I would have to make in the tutorial since it is JPA based. – ManeetK Dec 5, 2013 at 5:42

I am not sure on how to handle failed authentication requests with JDBC when I override the loadUserByUsername(String username) of UserDetailsService. Could you help me with that. – ManeetK Dec 11, 2013 at 8:34

Why would you need to handle failed authentication requests? Spring does that for you and that doesn't change when you implement your own `UserDetailsService`. – M. Deinum Dec 11, 2013 at 9:31

Take for example the scenario when username and password do not match, how should I handle that when I override the method. Should I just return a null object in that case? – ManeetK Dec 11, 2013 at 9:36

You don't handle that, spring security handles that for you. The `UserDetailsService` is **ONLY** for looking up users NOT for checking the password. As you noticed there is only a method `loadUserByUsername` the name of the method says it all, the same for the attributes of the method, there is no password so how would you validate the password?! – M. Deinum Dec 11, 2013 at 9:44