# SpringSecurity5.1.X

## Obtaining information about the current user

Inside the SecurityContextHolder we store details of the principal currently interacting with the application. Spring Security uses an Authentication object to represent this information. You won’t normally need to create an Authentication object yourself, but it is fairly common for users to query the Authenticationobject. You can use the following code block - from anywhere in your application - to obtain the name of the currently authenticated user, for example:

Object principal = SecurityContextHolder.getContext().getAuthentication().getPrincipal();

**if** (principal **instanceof** UserDetails) {

String username = ((UserDetails)principal).getUsername();

} **else** {

String username = principal.toString();

}

## What is authentication in Spring Security?

Let’s consider a standard authentication scenario that everyone is familiar with.

1. A user is prompted to log in with a username and password.
2. The system (successfully) verifies that the password is correct for the username.
3. The context information for that user is obtained (their list of roles and so on).
4. A security context is established for the user
5. The user proceeds, potentially to perform some operation which is potentially protected by an access control mechanism which checks the required permissions for the operation against the current security context information.

The first four items constitute the authentication process so we’ll take a look at how these take place within Spring Security.

1. The username and password are obtained and combined into an instance of UsernamePasswordAuthenticationToken (an instance of the Authenticationinterface, which we saw earlier).
2. The token is passed to an instance of AuthenticationManager for validation.
3. The AuthenticationManager returns a fully populated Authentication instance on successful authentication.
4. The security context is established by calling SecurityContextHolder.getContext().setAuthentication(…​), passing in the returned authentication object.

From that point on, the user is considered to be authenticated. Let’s look at some code as an example.

下面的案例为：用户名和密码一致则为通过

**import** org.springframework.security.authentication.\*;**import** org.springframework.security.core.\*;**import** org.springframework.security.core.authority.SimpleGrantedAuthority;**import** org.springframework.security.core.context.SecurityContextHolder;

**public** **class** AuthenticationExample {**private** **static** AuthenticationManager am = **new** SampleAuthenticationManager();

**public** **static** **void** main(String[] args) **throws** Exception {

BufferedReader in = **new** BufferedReader(**new** InputStreamReader(System.in));

**while**(true) {

System.out.println("Please enter your username:");

String name = in.readLine();

System.out.println("Please enter your password:");

String password = in.readLine();

**try** {

Authentication request = **new** UsernamePasswordAuthenticationToken(name, password);

Authentication result = am.authenticate(request);

SecurityContextHolder.getContext().setAuthentication(result);

**break**;

} **catch**(AuthenticationException e) {

System.out.println("Authentication failed: " + e.getMessage());

}

}

System.out.println("Successfully authenticated. Security context contains: " +

SecurityContextHolder.getContext().getAuthentication());

}

}

**class** SampleAuthenticationManager **implements** AuthenticationManager {**static** **final** List<GrantedAuthority> AUTHORITIES = **new** ArrayList<GrantedAuthority>();

**static** {

AUTHORITIES.add(**new** SimpleGrantedAuthority("ROLE\_USER"));

}

**public** Authentication authenticate(Authentication auth) **throws** AuthenticationException {

//用户名和密码相同

**if** (auth.getName().equals(auth.getCredentials())) {

**return** **new** UsernamePasswordAuthenticationToken(auth.getName(),

auth.getCredentials(), AUTHORITIES);

}

**throw** **new** BadCredentialsException("Bad Credentials");

}

}

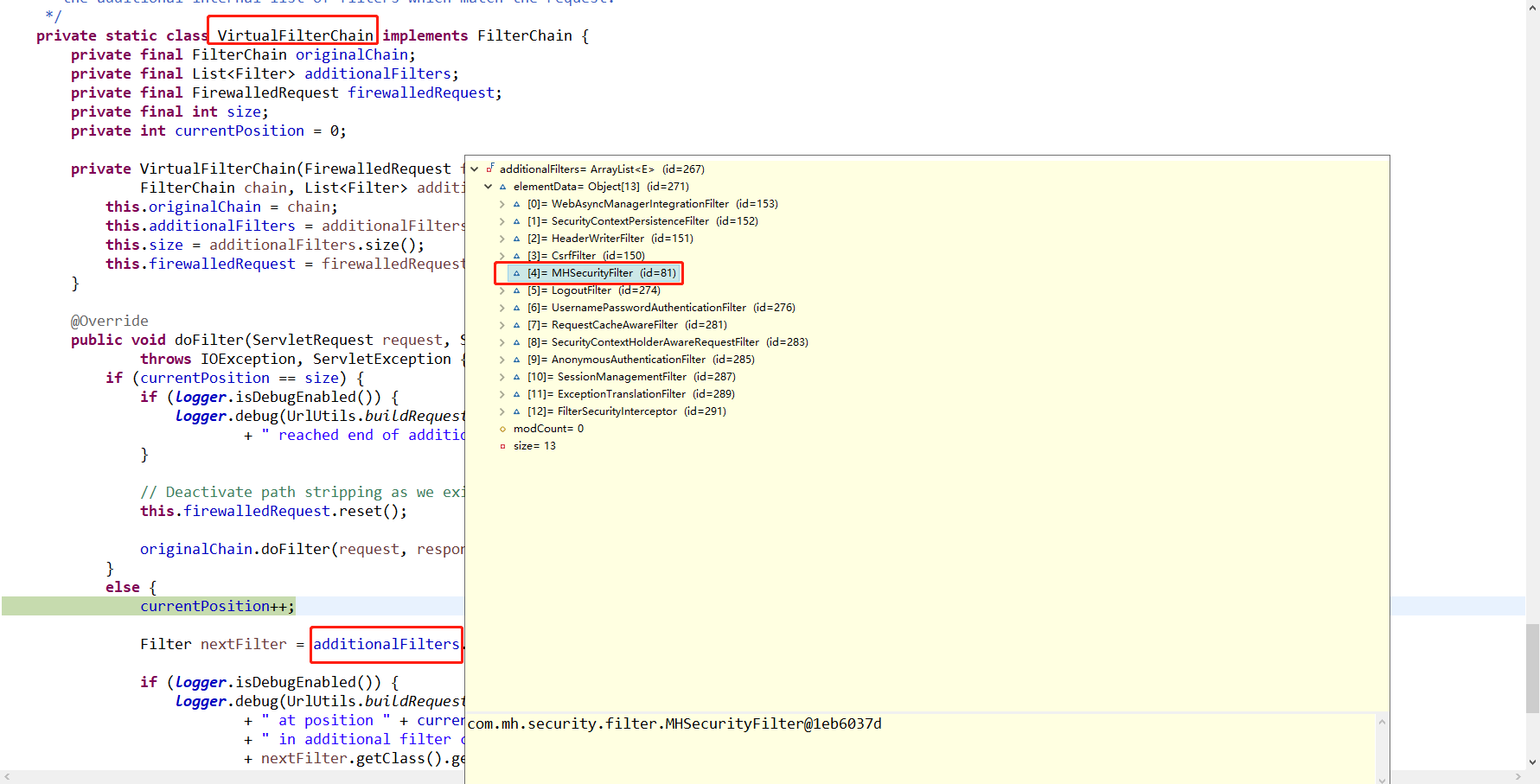
通过分析：

Authentication request = **new** UsernamePasswordAuthenticationToken(name, password);

Principal：当事人，理解为用户名

Credentials：证书，理解为密码

## 过滤器链：VirtualFilterChain



## 默认通过AnonymousAuthenticationFilter 类认证

核心代码解刨：在doFilter方法中主要创建认证对象，并设置给SecurityContext

SecurityContextHolder.*getContext*().setAuthentication(

createAuthentication((HttpServletRequest) req)); //设置认证对象

createAuthentication(req): 创建一个Authentication 的子类->AnonymousAuthenticationToken

代码实现：

**protected** Authentication createAuthentication(HttpServletRequest request) {

AnonymousAuthenticationToken auth = **new** AnonymousAuthenticationToken(key,

principal, authorities); //创建匿名认证对象

auth.setDetails(authenticationDetailsSource.buildDetails(request));

**return** auth;

}

## 开始做认证：FilterSecurityInterceptor 实现了filter接口

### 代码解刨：在doFilter -->调用invoke(fi)-->调用beforeInvocation

在beforeInvocation 核心代码：

//从SecurityContext获取认证对象，即：

//在AnonymousAuthenticationFilter 设置的认证对象

Authentication authenticated = authenticateIfRequired();

// Attempt authorization

**try** {

**this**.accessDecisionManager.decide(authenticated, object, attributes);

}

attributes: 为url 允许权限 例如：permitAll

### 解析authenticateIfRequired()方法

/\*\*

\* Checks the current authentication token and passes it to the AuthenticationManager

\* if {@link org.springframework.security.core.Authentication#isAuthenticated()}

\* returns false or the property <tt>alwaysReauthenticate</tt> has been set to true.

\*

\* **@return** an authenticated <tt>Authentication</tt> object.

\*/

**private** Authentication authenticateIfRequired() {

Authentication authentication = SecurityContextHolder.*getContext*()

.getAuthentication();

//如果authentication 已经被认证过了，则直接返回authentication

**if** (authentication.isAuthenticated() && !alwaysReauthenticate) {

**if** (logger.isDebugEnabled()) {

logger.debug("Previously Authenticated: " + authentication);

}

**return** authentication;

}

//重新鉴别认证

authentication = authenticationManager.authenticate(authentication);

// We don't authenticated.setAuthentication(true), because each provider should do

// that

**if** (logger.isDebugEnabled()) {

logger.debug("Successfully Authenticated: " + authentication);

}

SecurityContextHolder.*getContext*().setAuthentication(authentication);

**return** authentication;

}

## Authorize Requests 认证请求

Our examples have only required users to be authenticated and have done so for every URL in our application. We can specify custom requirements for our URLs by adding multiple children to our http.authorizeRequests() method. For example:

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http

.authorizeRequests() IMG_256

.antMatchers("/resources/\*\*", "/signup", "/about").permitAll() IMG_257

.antMatchers("/admin/\*\*").hasRole("ADMIN") IMG_258

.antMatchers("/db/\*\*").access("hasRole('ADMIN') and hasRole('DBA')") IMG_259

.anyRequest().authenticated() IMG_260

.and()

*// ...*

.formLogin();

}

|  |  |
| --- | --- |
| [IMG_261](https://docs.spring.io/spring-security/site/docs/5.1.6.RELEASE/reference/htmlsingle/#CO3-1) | There are multiple children to the http.authorizeRequests() method each matcher is considered in the order they were declared. |
| [IMG_262](https://docs.spring.io/spring-security/site/docs/5.1.6.RELEASE/reference/htmlsingle/#CO3-2) | We specified multiple URL patterns that any user can access. Specifically, any user can access a request if the URL starts with "/resources/", equals "/signup", or equals "/about". |
| [IMG_263](https://docs.spring.io/spring-security/site/docs/5.1.6.RELEASE/reference/htmlsingle/#CO3-3) | Any URL that starts with "/admin/" will be restricted to users who have the role "ROLE\_ADMIN". You will notice that since we are invoking the hasRole method we do not need to specify the "ROLE\_" prefix. |
| [IMG_264](https://docs.spring.io/spring-security/site/docs/5.1.6.RELEASE/reference/htmlsingle/#CO3-4) | Any URL that starts with "/db/" requires the user to have both "ROLE\_ADMIN" and "ROLE\_DBA". You will notice that since we are using the hasRole expression we do not need to specify the "ROLE\_" prefix. |
| [IMG_265](https://docs.spring.io/spring-security/site/docs/5.1.6.RELEASE/reference/htmlsingle/#CO3-5) | Any URL that has not already been matched on only requires that the user be authenticated |

## 设计方案

### 自定义Fileter（起名为MHSecurityFilter） 类，并放在 ExceptionTranslationFilter执行链 后面。

在MHSecurityFilter 中有两大属性：

自定义实现 AuthenticationManager 的子类 ：UserAuthenticationManager

自定义实现Authentication 的子类：