


SpringBoot中使用Shiro和JWT做认证和鉴权

空挡 [关注](#) 11 2018.08.28 23:16:29 字数 3,172 阅读 90,753

最近新做的项目中使用了shiro和jwt来做简单的权限验证，在和springboot集成的过程中碰到了不少坑。做完之后对shiro的体系架构了解的也差不多了，现在把中间需要注意的点放出来，给大家做个参考。

相对于spring security来说，shiro出来较早，框架也相对简单。后面会另起一篇文章对这两个框架做一个简单的对比。

Shiro的关注点

首先看一下shiro中需要关注的几个概念。

- SecurityManager，可以理解成控制中心，所有请求最终基本上都通过它来代理转发，一般我们程序中不需要直接跟他打交道。
- Subject，请求主体。比如登录用户，比如一个被授权的app。在程序中任何地方都可以通过 `SecurityUtils.getSubject()` 获取到当前的subject。subject中可以获取到Principal，这个是subject的标识，比如登陆用户的用户名或者id等，shiro不对值做限制。但是在登录和授权过程中，程序需要通过principal来识别唯一的用户。
- Realm，这个实在不知道怎么翻译合适。通俗一点理解就是realm可以访问安全相关数据，提供统一的数据封装来给上层做数据校验。shiro的建议是每种数据源定义一个realm，比如用户数据存在数据库可以使用JdbcRealm；存在属性配置文件可以使用PropertiesRealm。一般我们使用shiro都使用自定义的realm。

当有多个realm存在的时候，shiro在做用户校验的时候会按照定义的策略来决定认证是否通过，shiro提供的可选策略有一个成功或者所有都成功等。

一个realm对应了一个CredentialsMatcher，用来做用户提交认证信息和realm获取得用户信息做对比，shiro已经提供了常用的比如用户密码和存储的Hash后的密码的对比。

JWT的应用场景

关于JWT是什么，请参考[JWT官网](#)。这里就不多解释了，可理解成使用带签名的token来做用户和权限验证，现在流行的公共开放接口用的OAuth 2.0协议基本也是类似的套路。这里只是说下选择使用jwt不用session的原因。

首先，是要支持多端，一个api要支持H5, PC和APP三个前端，如果使用session的话对app不是很友好，而且session有跨域攻击的问题。

其次，后端的服务器是无状态的，所以要支持分布式的权限校验。当然这个不是主要原因了，因为session持久化在spring里面也就是加一行注解就解决的问题。不过，spring通过代理httpsession来做，总归觉得有点复杂。

项目搭建

需求

需求相对简单，1) 支持用户首次通过用户名和密码登录；2) 登录后通过http header返回token；3) 每次请求，客户端需通过header将token带回，用于权限校验；4) 服务端负责token的定期刷新，刷新后新的token仍然放到header中返给客户端



空挡

总资产 24

[关注](#)

Raft协议实现之etcd(三)：日志同步
阅读 602

Raft协议实现之etcd(二)：心跳及选举
阅读 1,001

推荐阅读

基于token身份认证的完整实例
阅读 24,589

SpringSecurity认证原理
阅读 162

Spring Cloud Gateway + Jwt +
Oauth2 实现网关的鉴权操作
阅读 3,152

基于Spring Boot的API测试
阅读 575

网关Spring Cloud Gateway
阅读 726

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org
5     <modelVersion>4.0.0</modelVersion>
6
7     <groupId>com.github.springboot</groupId>
8     <artifactId>shiro-jwt-demo</artifactId>
9     <version>1.0-SNAPSHOT</version>
10    <packaging>jar</packaging>
11
12    <name>Spring Boot with Shiro and JWT Demo</name>
13    <description>Demo project for Spring Boot with Shiro and JWT</description>
14    <parent>
15        <groupId>org.springframework.boot</groupId>
16        <artifactId>spring-boot-starter-parent</artifactId>
17        <version>2.0.4.RELEASE</version>
18    </parent>
19    <properties>
20        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
21        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
22        <java.version>1.8</java.version>
23        <shiro.spring.version>1.4.0</shiro.spring.version>
24        <jwt.auth0.version>3.2.0</jwt.auth0.version>
25    </properties>
26
27    <dependencies>
28        <dependency>
29            <groupId>org.springframework.boot</groupId>
30            <artifactId>spring-boot-starter-web</artifactId>
31        </dependency>
32        <dependency>
33            <groupId>org.springframework.boot</groupId>
34            <artifactId>spring-boot-starter-test</artifactId>
35        </dependency>
36        <!-- 使用redis做数据缓存, 如果不需要可不依赖 -->
37        <dependency>
38            <groupId>org.springframework.boot</groupId>
39            <artifactId>spring-boot-starter-data-redis</artifactId>
40        </dependency>
41        <dependency>
42            <groupId>org.apache.shiro</groupId>
43            <artifactId>shiro-spring-boot-web-starter</artifactId>
44            <version>${shiro.spring.version}</version>
45        </dependency>
46        <dependency>
47            <groupId>com.auth0</groupId>
48            <artifactId>java-jwt</artifactId>
49            <version>${jwt.auth0.version}</version>
50        </dependency>
51        <dependency>
52            <groupId>org.apache.httpcomponents</groupId>
53            <artifactId>httpclient</artifactId>
54            <version>4.5.5</version>
55        </dependency>
56        <dependency>
57            <groupId>org.apache.commons</groupId>
58            <artifactId>commons-lang3</artifactId>
59            <version>3.7</version>
60        </dependency>
61    </dependencies>
62    <build>
63        <plugins>
64            <plugin>
65                <groupId>org.springframework.boot</groupId>
66                <artifactId>spring-boot-maven-plugin</artifactId>
67            </plugin>
68            <plugin>
69                <groupId>org.apache.maven.plugins</groupId>
70                <artifactId>maven-compiler-plugin</artifactId>
71                <configuration>
72                    <source>${java.version}</source>
73                    <target>${java.version}</target>
74                </configuration>
75            </plugin>
76        </plugins>
```

```
82         </plugin>
83     </plugins>
84 </build>
85 </project>
```

shiro 配置

ShiroConfiguration

首先是初始化shiro的bean，主要是初始化Realm，注册Filter，定义filterChain。这些配置的用处后面会逐渐讲到。

```
1  @Configuration
2  public class ShiroConfig {
3      /**
4       * 注册shiro的Filter，拦截请求
5       */
6      @Bean
7      public FilterRegistrationBean<Filter> filterRegistrationBean(SecurityManager secur
8          FilterRegistrationBean<Filter> filterRegistration = new FilterRegistrationBean
9          filterRegistration.setFilter((Filter)shiroFilter(securityManager, userService)
10         filterRegistration.addInitParameter("targetFilterLifecycle", "true");
11         filterRegistration.setAsyncSupported(true);
12         filterRegistration.setEnabled(true);
13         filterRegistration.setDispatcherTypes(DispatcherType.REQUEST);
14
15         return filterRegistration;
16     }
17
18     /**
19     * 初始化Authenticator
20     */
21     @Bean
22     public Authenticator authenticator(UserService userService) {
23         ModularRealmAuthenticator authenticator = new ModularRealmAuthenticator();
24         //设置两个Realm，一个用于用户登录验证和访问权限获取；一个用于jwt token的认证
25         authenticator.setRealms(Arrays.asList(jwtShiroRealm(userService), dbShiroRealm
26         //设置多个realm认证策略，一个成功即跳过其它的
27         authenticator.setAuthenticationStrategy(new FirstSuccessfulStrategy());
28         return authenticator;
29     }
30
31     /**
32     * 禁用session，不保存用户登录状态。保证每次请求都重新认证。
33     * 需要注意的是，如果用户代码里调用Subject.getSession()还是可以用session，如果要完全禁用，要配
34     */
35     @Bean
36     protected SessionStorageEvaluator sessionStorageEvaluator(){
37         DefaultWebSessionStorageEvaluator sessionStorageEvaluator = new DefaultWebSess
38         sessionStorageEvaluator.setSessionStorageEnabled(false);
39         return sessionStorageEvaluator;
40     }
41     /**
42     * 用于用户名密码登录时认证的realm
43     */
44     @Bean("dbRealm")
45     public Realm dbShiroRealm(UserService userService) {
46         DbShiroRealm myShiroRealm = new DbShiroRealm(userService);
47         return myShiroRealm;
48     }
49     /**
50     * 用于JWT token认证的realm
51     */
52     @Bean("jwtRealm")
53     public Realm jwtShiroRealm(UserService userService) {
54         JWTShiroRealm myShiroRealm = new JWTShiroRealm(userService);
55         return myShiroRealm;
56     }
57
58     /**
59     * 设置过滤器，将自定义的Filter加入
60     */
61     @Bean("shiroFilter")
62     public ShiroFilterFactoryBean shiroFilter(SecurityManager securityManager, UserSer
```

```
69     factoryBean.setFilterChainDefinitionMap(shiroFilterChainDefinition().getFilterChainDefinitionMap());
70
71     return factoryBean;
72 }
73
74 @Bean
75 protected ShiroFilterChainDefinition shiroFilterChainDefinition() {
76     DefaultShiroFilterChainDefinition chainDefinition = new DefaultShiroFilterChainDefinition();
77     chainDefinition.addPathDefinition("/login", "noSessionCreation,anon"); //login
78     chainDefinition.addPathDefinition("/logout", "noSessionCreation,authcToken[permissive]"); //logout
79     chainDefinition.addPathDefinition("/image/**", "anon");
80     chainDefinition.addPathDefinition("/admin/**", "noSessionCreation,authcToken,authz");
81     chainDefinition.addPathDefinition("/article/list", "noSessionCreation,authcToken,authz");
82     chainDefinition.addPathDefinition("/article/**", "noSessionCreation,authcToken[permissive]");
83     chainDefinition.addPathDefinition("/**", "noSessionCreation,authcToken"); // 默认
84     return chainDefinition;
85 }
86 //注意不要加@Bean注解，不然spring会自动注册成filter
87 protected JwtAuthFilter createAuthFilter(UserService userService){
88     return new JwtAuthFilter(userService);
89 }
90 //注意不要加@Bean注解，不然spring会自动注册成filter
91 protected AnyRolesAuthorizationFilter createRolesFilter(){
92     return new AnyRolesAuthorizationFilter();
93 }
94
95 }
```

校验流程

我们使用Shiro主要做3件事情，1) 用户登录时做用户名密码校验；2) 用户登录后收到请求时做JWT Token的校验；3) 用户权限的校验

登录认证流程

登录controller

从前面的 `ShiroFilterChainDefinition` 配置可以看出，对于登录请求，Filter直接放过，进到controller里面。Controller会调用shiro做用户名和密码的校验，成功后返回token。

```
1  @PostMapping(value = "/login")
2      public ResponseEntity<Void> login(@RequestBody UserDto loginInfo, HttpServletRequest request) {
3      Subject subject = SecurityUtils.getSubject();
4      try {
5          //将用户请求参数封装后，直接提交给Shiro处理
6          UsernamePasswordToken token = new UsernamePasswordToken(loginInfo.getUsername(), loginInfo.getPassword());
7          subject.login(token);
8          //Shiro认证通过后会user信息放到subject内，生成token并返回
9          UserDto user = (UserDto) subject.getPrincipal();
10         String newToken = userService.generateJwtToken(user.getUsername());
11         response.setHeader("x-auth-token", newToken);
12
13         return ResponseEntity.ok().build();
14     } catch (AuthenticationException e) {
15         // 如果校验失败，shiro会抛出异常，返回客户端失败
16         logger.error("User {} login fail, Reason:{}", loginInfo.getUsername(), e.getMessage());
17         return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
18     } catch (Exception e) {
19         return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
20     }
21 }
```

登录的Realm

从上面的controller实现我们看到，controller只负责封装下参数，然后扔给Shiro了，这时候Shiro收到后，会到所有的realm中找能处理 `UsernamePasswordToken` 的Realm（我们这里是 `DbShiroRealm`），然后交给Realm处理。Realm的实现一般直接继承 `AuthorizingRealm` 即可，只需要实现两个方法，`doGetAuthenticationInfo()` 会在用户验证时被调用，我们看下实现。

```
6 public DbShiroRealm(UserService userService) {
7     this.userService = userService;
8     //因为数据库中的密码做了散列，所以使用shiro的散列Matcher
9     this.setCredentialsMatcher(new HashedCredentialsMatcher(Sha256Hash.ALGORITHM_N
10 }
11 /**
12  * 找它的原因是这个方法返回true
13  */
14 @Override
15 public boolean supports(AuthenticationToken token) {
16     return token instanceof UsernamePasswordToken;
17 }
18 /**
19  * 这一步我们根据token给的用户名，去数据库查出加密过用户密码，然后把加密后的密码和盐值一起发给shi
20  */
21 @Override
22 protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) th
23     UsernamePasswordToken userpasswordToken = (UsernamePasswordToken)token;
24     String username = userpasswordToken.getUsername();
25     UserDto user = userService.getUserInfo(username);
26     if(user == null)
27         throw new AuthenticationException("用户名或者密码错误");
28
29     return new SimpleAuthenticationInfo(user, user.getEncryptPwd(), ByteSource.Uti
30 }
31 }
32 }
```

我们可以看到 `doGetAuthenticationInfo` 里面只判断了用户存不存在，其实也没做密码比对，只是把数据库的数据封装一下就返回了。真正的比对逻辑在Matcher里实现的，这个shiro已经替我们实现了。如果matcher返回false，shiro会抛出异常，这样controller那边就会知道验证失败了。

登出

登出操作就比较简单了，我们只需要把用户登录后保存的salt值清除，然后调用shiro的logout就可以了，shiro会将剩下的事情做完。

```
1 @GetMapping(value = "/logout")
2 public ResponseEntity<Void> logout() {
3     Subject subject = SecurityUtils.getSubject();
4     if(subject.getPrincipals() != null) {
5         UserDto user = (UserDto)subject.getPrincipals().getPrimaryPrincipal();
6         userService.deleteLoginInfo(user.getUsername());
7     }
8     SecurityUtils.getSubject().logout();
9     return ResponseEntity.ok().build();
10 }
```

这样整个登录/登出就结束了，我们可以看到shiro对整个逻辑的拆解还是比较清楚的，各个模块各司其职。

请求认证流程

请求认证的流程其实和登录认证流程是比较相似的，因为我们的服务是无状态的，所以每次请求带来token，我们就是做了一次登录操作。

JwtAuthFilter

首先我们先从入口的Filter开始。从 `AuthenticatingFilter` 继承，重写isAccessAllow方法，方法中调用父类executeLogin()。父类的这个方法首先会createToken()，然后调用shiro的 `Subject.login()` 方法。是不是跟 `LoginController` 中的逻辑很像。

```
1 public class JwtAuthFilter extends AuthenticatingFilter {
2     /**
3      * 父类会在请求进入拦截器后调用该方法，返回true则继续，返回false则会调用onAccessDenied()。这里
4      */
5     @Override
6     protected boolean isAccessAllowed(ServletRequest request, ServletResponse response
```

```
12         } catch(IllegalStateException e){ //not found any token
13             log.error("Not found any token");
14         }catch (Exception e) {
15             log.error("Error occurs when login", e);
16         }
17         return allowed || super.isPermissive(mappedValue);
18     }
19     /**
20     * 这里重写了父类的方法，使用我们自己定义的Token类，提交给shiro。这个方法返回null的话会直接抛出
21     */
22     @Override
23     protected AuthenticationToken createToken(ServletRequest servletRequest, ServletRe
24         String jwtToken = getAuthzHeader(servletRequest);
25         if(StringUtils.isNotBlank(jwtToken)&&!JwtUtils.isTokenExpired(jwtToken))
26             return new JWTToken(jwtToken);
27
28         return null;
29     }
30     /**
31     * 如果这个Filter在之前isAccessAllowed () 方法中返回false,则会进入这个方法。我们这里直接返回
32     */
33     @Override
34     protected boolean onAccessDenied(ServletRequest servletRequest, ServletResponse se
35         HttpServletResponse httpResponse = WebUtils.toHttp(servletResponse);
36         httpResponse.setCharacterEncoding("UTF-8");
37         httpResponse.setContentType("application/json;charset=UTF-8");
38         httpResponse.setStatus(HttpStatus.SC_NON_AUTHORITATIVE_INFORMATION);
39         fillCorsHeader(WebUtils.toHttp(servletRequest), httpResponse);
40         return false;
41     }
42     /**
43     * 如果Shiro Login认证成功，会进入该方法，等同于用户名密码登录成功，我们这里还判断了是否要刷新
44     */
45     @Override
46     protected boolean onLoginSuccess(AuthenticationToken token, Subject subject, Servi
47         HttpServletResponse httpResponse = WebUtils.toHttp(response);
48         String newToken = null;
49         if(token instanceof JWTToken){
50             JWTToken jwtToken = (JWTToken)token;
51             UserDto user = (UserDto) subject.getPrincipal();
52             boolean shouldRefresh = shouldTokenRefresh(JwtUtils.getIssuedAt(jwtToken.g
53                 if(shouldRefresh) {
54                     newToken = userService.generateJwtToken(user.getUsername());
55                 }
56             }
57             if(StringUtils.isNotBlank(newToken))
58                 httpResponse.setHeader("x-auth-token", newToken);
59
60             return true;
61         }
62     /**
63     * 如果调用shiro的login认证失败，会回调这个方法，这里我们什么都不做，因为逻辑放到了onAccessDe
64     */
65     @Override
66     protected boolean onLoginFailure(AuthenticationToken token, AuthenticationExceptio
67         log.error("Validate token fail, token:{}, error:{}", token.toString(), e.getMe
68         return false;
69     }
70 }
```

JWT token封装

在上面的Filter中我们创建了一个Token提交给了shiro，我们看下这个Token，其实很简单，就是把jwt的token放在里面。

```
1 public class JWTToken implements HostAuthenticationToken {
2     private String token;
3     private String host;
4     public JWTToken(String token) {
5         this(token, null);
6     }
7     public JWTToken(String token, String host) {
8         this.token = token;
9         this.host = host;
10    }
```

```
17 @Override
18 public Object getPrincipal() {
19     return token;
20 }
21 @Override
22 public Object getCredentials() {
23     return token;
24 }
25 @Override
26 public String toString(){
27     return token + ':' + host;
28 }
29 }
```

JWT Realm

Token有了，filter中也调用了shiro的login()方法了，下一步自然是Shiro把token提交到Realm中，获取存储的认证信息来做比对。

```
1 public class JWTShiroRealm extends AuthorizingRealm {
2     protected UserService userService;
3
4     public JWTShiroRealm(UserService userService){
5         this.userService = userService;
6         //这里使用我们自定义的Matcher
7         this.setCredentialsMatcher(new JWTCredentialsMatcher());
8     }
9     /**
10      * 限定这个Realm只支持我们自定义的JWT Token
11      */
12     @Override
13     public boolean supports(AuthenticationToken token) {
14         return token instanceof JWTToken;
15     }
16
17     /**
18      * 更controller登录一样，也是获取用户的salt值，给到shiro，由shiro来调用matcher来做认证
19      */
20     @Override
21     protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authcToken) {
22         JWTToken jwtToken = (JWTToken) authcToken;
23         String token = jwtToken.getToken();
24
25         UserDto user = userService.getJwtTokenInfo(JwtUtils.getUsername(token));
26         if(user == null)
27             throw new AuthenticationException("token过期，请重新登录");
28
29         SimpleAuthenticationInfo authenticationInfo = new SimpleAuthenticationInfo(user, token, "realm");
30
31         return authenticationInfo;
32     }
33 }
```

JWT Matcher

跟controller登录不一样，shiro并没有实现JWT的Matcher，需要我们自己来实现。代码如下：

```
1 public class JWTCredentialsMatcher implements CredentialsMatcher {
2     /**
3      * Matcher中直接调用工具包中的verify方法即可
4      */
5     @Override
6     public boolean doCredentialsMatch(AuthenticationToken authenticationToken, AuthenticationInfo authenticationInfo) {
7         String token = (String) authenticationToken.getCredentials();
8         Object stored = authenticationInfo.getCredentials();
9         String salt = stored.toString();
10
11         UserDto user = (UserDto) authenticationInfo.getPrincipals().getPrimaryPrincipal();
12         try {
13             Algorithm algorithm = Algorithm.HMAC256(salt);
14             JWTVerifier verifier = JWT.require(algorithm)
15                 .withClaim("username", user.getUsername())
16                 .build();
17             verifier.verify(token);
18         } catch (Exception e) {
19             return false;
20         }
21         return true;
22     }
23 }
```

```
23 |     }  
24 | }
```

这样非登录请求的认证处理逻辑也结束了，看起来是不是跟登录逻辑差不多。其实对于无状态服务来说，每次请求都相当于做了一次登录操作，我们用session的时候之所以不需要做，是因为容器代替我们把这件事干掉了。

关于permissive

前面Filter里面的isAccessAllow方法，除了使用jwt token做了shiro的登录认证之外，如果返回false还会额外调用isPermissive()方法。这里面干了什么呢？我们看下父类的方法：

```
1 | /**  
2 |  * Returns <code>true</code> if the mappedValue contains the {@link #PERMISSIVE} q  
3 |  *  
4 |  * @return <code>true</code> if this filter should be permissive  
5 |  */  
6 | protected boolean isPermissive(Object mappedValue) {  
7 |     if(mappedValue != null) {  
8 |         String[] values = (String[]) mappedValue;  
9 |         return Arrays.binarySearch(values, PERMISSIVE) >= 0;  
10 |     }  
11 |     return false;  
12 | }
```

逻辑很简单，如果filter的拦截配置那里配置了permissive参数，即使登录认证没通过，因为isPermissive返回true，还是会让请求继续下去的。细心的同学或许已经发现我们之前shiroConfig里面的配置了，截取过来看一下：

```
1 | chainDefinition.addPathDefinition("/logout", "noSessionCreation,authToken[permissive]
```

就是这么简单直接，字符串匹配。当然这里也可以重写这个方法插入更复杂的逻辑。

这么做的目的是什么呢？因为有时候我们对待请求，并不都是非黑即白，比如登出操作，如果用户带的token是正确的，我们会将保存的用户信息清除；如果带的token是错的，也没关系，大不了不干啥，没必要返回错误给用户。还有一个典型的案例，比如我们阅读博客，匿名用户也是可以看的。只是如果是登录用户，我们会显示额外的东西，比如是不是点过赞等。所以认证这里的逻辑就是token是对的，我会把人认出来；是错的，我也直接放过，留给controller来决定怎么区别对待。

JWT Token刷新

前面的Filter里面还有一个逻辑（是不是太多了😂），就是如果用户这次的token校验通过后，我们还会顺便看看token要不要刷新，如果需要刷新则将新的token放到header里面。这样做的目的是防止token丢了之后，别人可以拿着一直用。我们这里是固定时间刷新。安全性要求更高的系统可能每次请求都要求刷新，或者是每次POST，PUT等修改数据的请求后必须刷新。判断逻辑如下：

```
1 | protected boolean shouldTokenRefresh(Date issueAt){  
2 |     LocalDateTime issueTime = LocalDateTime.ofInstant(issueAt.toInstant(), ZoneId.  
3 |     return LocalDateTime.now().minusSeconds(tokenRefreshInterval).isAfter(issueTim  
4 | }
```

以上就是jwt token校验的所有逻辑了，是不是有点绕，画一个流程图出来，对比着看应该更清楚一点。

jwt filter逻辑

角色配置

认证讲完了，下面看下访问控制。对于角色检查的拦截，是通过继承一个 `AuthorizationFilter` 的Filter来实现的。Shiro提供了一个默认的实现 `RolesAuthorizationFilter`，比如可以这么配置：

```
1 | chainDefinition.addPathDefinition("/article/edit", "authc,role[admin]");
```

表示要做文章的edit操作，需要满足两个条件，首先authc表示要通过用户认证，这个我们上面已经讲过了；其次要具备admin的角色。shiro是怎么做的呢？就是在请求进入这个filter后，shiro会调用所有配置的Realm获取用户的角色信息，然后和Filter中配置的角色做对比，对上了就可以通过了。

所以我们所有的Realm还要另外一个方法 `doGetAuthorizationInfo`，不得不吐槽一下，realm里面要实现的这两个方法的名字实在太像了。

在JWT Realm里面，因为没有存储角色信息，所以直接返回空就可以了：

```
1 | @Override
2 |     protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals)
3 |     {
4 |         return new SimpleAuthorizationInfo();
5 |     }
```

在DbRealm里面，实现如下：

```
1 | @Override
2 |     protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals)
3 |     {
4 |         SimpleAuthorizationInfo simpleAuthorizationInfo = new SimpleAuthorizationInfo();
5 |         UserDto user = (UserDto) principals.getPrimaryPrincipal();
6 |         List<String> roles = user.getRoles();
7 |         if(roles == null) {
8 |             roles = userService.getUserRoles(user.getId());
9 |             user.setRoles(roles);
10 |        }
11 |        if (roles != null)
12 |            simpleAuthorizationInfo.addRoles(roles);
13 |        return simpleAuthorizationInfo;
14 |    }
```

这里需要注意一下的就是Shiro默认不会缓存角色信息，所以这里调用service的方法获取角色强烈建议从缓存中获取。

自己实现RoleFilter

在实际的项目中，对同一个url多个角色都有访问权限很常见，shiro默认的RoleFilter没有提供支持，比如上面的配置，如果我们配置成下面这样，那用户必须同时具备admin和manager权限才能访问，显然这个是不合理的。

`AuthorizationFilter` 中两个方法就可以了：

```
1 public class AnyRolesAuthorizationFilter extends AuthorizationFilter {
2
3     @Override
4     protected boolean isAccessAllowed(ServletRequest servletRequest, ServletResponse s
5         Subject subject = getSubject(servletRequest, servletResponse);
6         String[] rolesArray = (String[]) mappedValue;
7         if (rolesArray == null || rolesArray.length == 0) { //没有角色限制，有权限访问
8             return true;
9         }
10        for (String role : rolesArray) {
11            if (subject.hasRole(role)) //若当前用户是rolesArray中的任何一个，则有权限访问
12                return true;
13        }
14        return false;
15    }
16    /**
17     * 权限校验失败，错误处理
18     */
19    @Override
20    protected boolean onAccessDenied(ServletRequest request, ServletResponse response)
21        HttpServletResponse httpResponse = WebUtils.toHttp(response);
22        httpResponse.setCharacterEncoding("UTF-8");
23        httpResponse.setContentType("application/json;charset=utf-8");
24        httpResponse.setStatus(HttpStatus.SC_UNAUTHORIZED);
25        return false;
26    }
27 }
28 }
```

禁用session

因为用了jwt的访问认证，所以要把默认session支持关掉。这里要做两件事情，一个是

`ShiroConfig` 里面的配置：

```
1 @Bean
2 protected SessionStorageEvaluator sessionStorageEvaluator(){
3     DefaultWebSessionStorageEvaluator sessionStorageEvaluator = new DefaultWebSess
4     sessionStorageEvaluator.setSessionStorageEnabled(false);
5     return sessionStorageEvaluator;
6 }
```

另外一个是在对请求加上 `noSessionCreationFilter` ,具体原因上面的代码中已经有解释，用法如下：

```
1 chainDefinition.addPathDefinition("/**", "noSessionCreation,authToken");
```

跨域支持

对于前后端分离的项目，一般都需要跨域访问，这里需要做两件事，一个是在JwtFilter的postHandle中在头上加上跨域支持的选项（理论上应该重新定义一个Filter的，图省事就让它多干点吧😂）。

```
1 @Override
2 protected void postHandle(ServletRequest request, ServletResponse response){
3     this.fillCorsHeader(WebUtils.toHttp(request), WebUtils.toHttp(response));
4 }
```

在实际使用中发现，对于controller返回@ResponseBody的请求，filter中添加的header信息会丢失。对于这个问题spring已经给出解释，并建议实现ResponseBodyAdvice类，并添加@ControllerAdvice。

Allows customizing the response after the execution of an `@ResponseBody` or a `ResponseBody` controller method but before the body is written with an `HttpMessageConverter`.

Implementations may be registered directly with `RequestMappingHandlerAdapter` and `ExceptionHandlerExceptionResolver` or more likely annotated with `@ControllerAdvice` in which case they will be auto-detected by both.

所以如果存在返回`@ResponseBody`的controller，需要添加一个 `ResponseBodyAdvice` 实现类

```
1 @ControllerAdvice
2 public class ResponseHeaderAdvice implements ResponseBodyAdvice<Object> {
3     @Override
4     public boolean supports(MethodParameter methodParameter, Class<?> extends HttpMessageConverter.class) {
5         return true;
6     }
7
8     @Override
9     public Object beforeBodyWrite(Object o, MethodParameter methodParameter, MediaType mediaType,
10                                  ServletHttpRequest serverHttpRequest, ServletHttpResponse serverHttpResponse) {
11         ServletHttpRequest serverRequest = (ServletHttpRequest)serverHttpRequest;
12         ServletHttpResponse serverResponse = (ServletHttpResponse)serverHttpResponse;
13         if(serverRequest == null || serverResponse == null || serverRequest.getServletRequest() == null || serverResponse.getServletResponse() == null) {
14             return o;
15         }
16
17         // 对于未添加跨域消息头的响应进行处理
18         HttpServletRequest request = serverRequest.getServletRequest();
19         HttpServletResponse response = serverResponse.getServletResponse();
20         String originHeader = "Access-Control-Allow-Origin";
21         if(!response.containsHeader(originHeader)) {
22             String origin = request.getHeader("Origin");
23             if(origin == null) {
24                 String referer = request.getHeader("Referer");
25                 if(referer != null) {
26                     origin = referer.substring(0, referer.indexOf("/", 7));
27                 }
28             }
29             response.setHeader("Access-Control-Allow-Origin", origin);
30         }
31
32         String allowHeaders = "Access-Control-Allow-Headers";
33         if(!response.containsHeader(allowHeaders)) {
34             response.setHeader(allowHeaders, request.getHeader(allowHeaders));
35         }
36
37         String allowMethods = "Access-Control-Allow-Methods";
38         if(!response.containsHeader(allowMethods)) {
39             response.setHeader(allowMethods, "GET,POST,OPTIONS,HEAD");
40         }
41         //这个很关键，要不然ajax调用时浏览器默认不会把这个token的头属性返回给JS
42         String exposeHeaders = "access-control-expose-headers";
43         if(!response.containsHeader(exposeHeaders)) {
44             response.setHeader(exposeHeaders, "x-auth-token");
45         }
46
47         return o;
48     }
49 }
```

好了，到这里使用shiro和jwt做用户认证和鉴权的实现就结束了。详细代码地址：[shiro-jwt-demo](#)

下一篇：[Spring Security做JWT认证和授权](#)



229人点赞 >



spring



更多精彩内容，就在简书APP



"小礼物"  关注我"

赞赏支持

还没有人赞赏，支持一下



空挡 奋斗中的80后

总资产24 共写了9.8W字 获得618个赞 共525个粉丝

关注

被以下专题收入，发现更多相似内容



Java



security



spring-...



springb...



Spring 学习



java web



spring ...

展开更多

推荐阅读

更多精彩内容

Shiro整合Web项目及整合后的开发

title: Shiro整合Web项目及整合后的开发tags: shirocategories: shiro 将S...



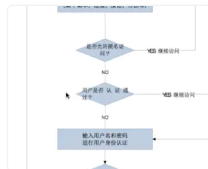
codingXiaxw 阅读 2,745 评论 3 赞 24

从权限控制到shiro框架的应用

说明：本文很多观点和内容来自互联网以及各种资料，如果侵犯了您的权益，请及时联系我，我会删除相关内容。权限管理 基...



寇寇寇先森 阅读 6,761 评论 8 赞 75

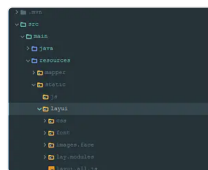


springboot1.5.9 + mybatis + layui + shiro后台权限管理系统

后台管理系统 业务场景 spring boot + mybatis后台管理系统框架； layUI前端界面； shi...



汉若已认证 阅读 8,958 评论 2 赞 68

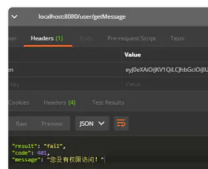


教你 Shiro + SpringBoot 整合 JWT

本篇文章将教大家在 shiro + springBoot 的基础上整合 JWT （JSON Web Token）如...



Howie_Y 阅读 77,734 评论 42 赞 118



2017.2.15

今天，一个朋友的同事生了重病去国外治疗了，感恩自己有一个健康的身体 跑步的路上，看到很多阿姨在跳广场舞，很幸福！感...



samsara伟萍 阅读 79 评论 0 赞 1