

Spring Authorization Server入门 (六) 自定义JWT中包含的内容与资源服务jwt解析器

叹雪飞花 2023-06-07 👁 2,558 ⌚ 阅读9分钟

关注

前言

在之前的文章中有提到过资源服务器解析access token时会将用户通过客户端请求的scope当做权限放入authorities属性中，当使用注解@PreAuthorize的hasAuthority校验用户权限时，实际上校验的是access token中拥有的权限；框架也提供了对应的定制内容，可以使开发者自定义jwt(access token)中的claims，同时对应的resource server也提供了对应的自定义解析配置。

OAuth2TokenCustomizer

[文档地址](#) 文档中对于OAuth2TokenCustomizer有这样一段描述:

An `OAuth2TokenCustomizer<JwtEncodingContext>` declared with a generic type of `JwtEncodingContext` (implements `OAuth2TokenContext`) provides the ability to customize the headers and claims of a `Jwt` . `JwtEncodingContext.getHeaders()` provides access to the `JwtHeader.Builder` , allowing the ability to add, replace, and remove headers. `JwtEncodingContext.getClaims()` provides access to the `JwtClaimsSet.Builder` , allowing the ability to add, replace, and remove claims.

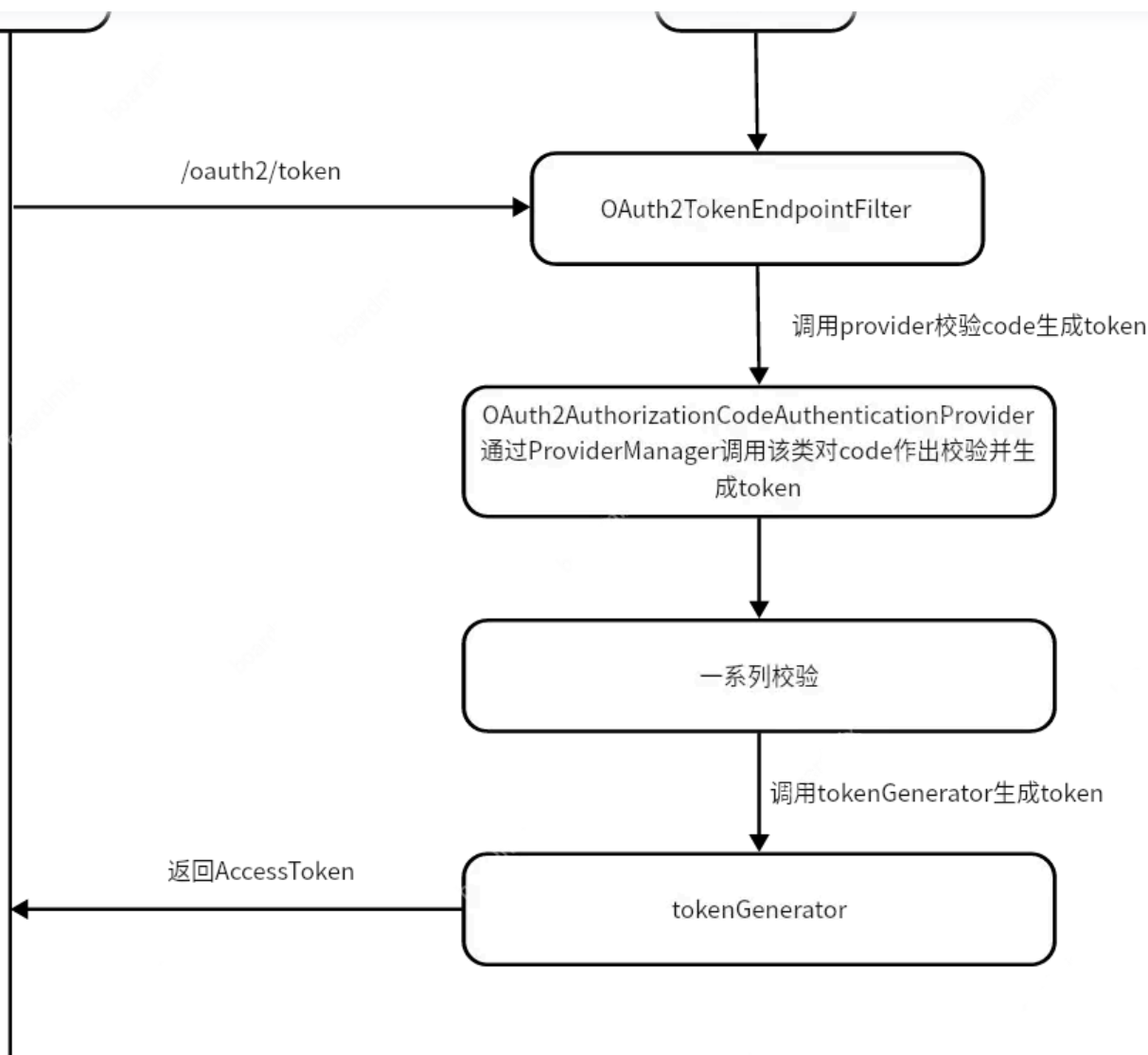
大概意思就是可以通过OAuth2TokenContext的实现类对jwt的header和claims部分进行修改。所以在认证服务器中实现 `OAuth2TokenCustomizer` 并将用户的权限信息放入jwt的claims中，并将实例注入IOC中。代码如下

java 复制代码

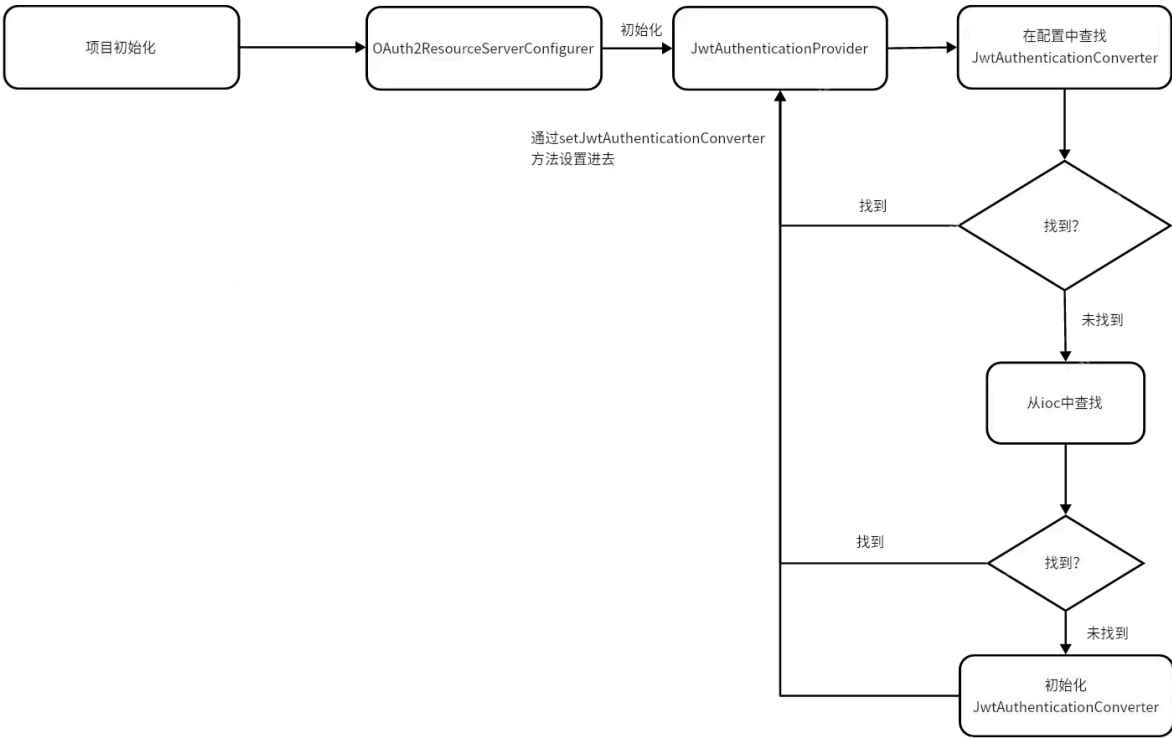
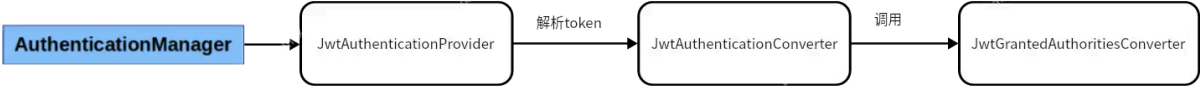
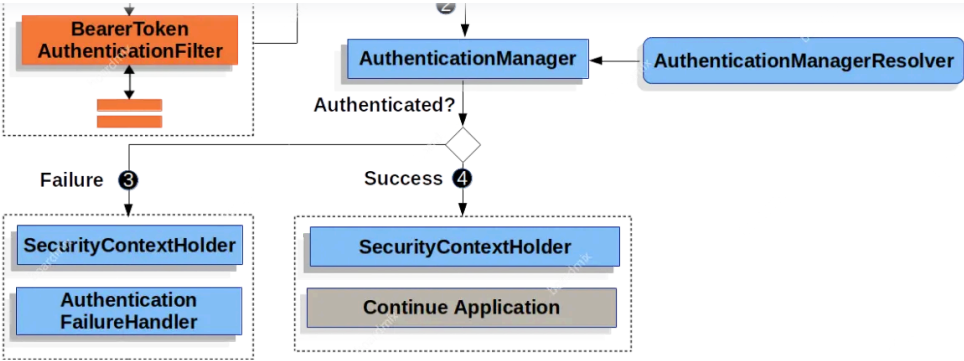
```
1 /**
2  * 自定义jwt，将权限信息放至jwt中
3  *
4  * @return OAuth2TokenCustomizer的实例
5  */
```

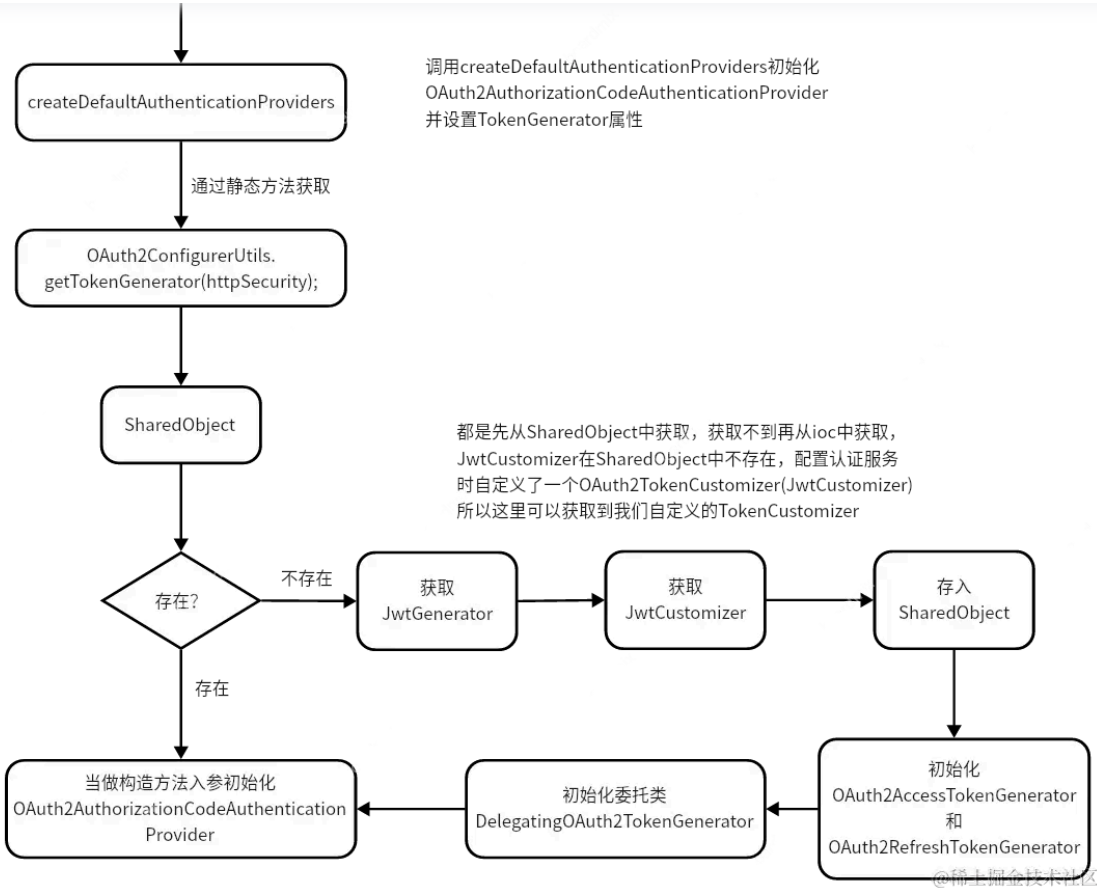
```
9      // 检查登录用户信息是不是UserDetails，排除掉没有用户参与的流程
10     if (context.getPrincipal().getPrincipal() instanceof UserDetails user) {
11         // 获取申请的scopes
12         Set<String> scopes = context.getAuthorizedScopes();
13         // 获取用户的权限
14         Collection<? extends GrantedAuthority> authorities = user.getAuthorities();
15         // 提取权限并转为字符串
16         Set<String> authoritySet = Optional.ofNullable(authorities).orElse(Collections.empty
17             // 获取权限字符串
18             .map(GrantedAuthority::getAuthority)
19             // 去重
20             .collect(Collectors.toSet()));
21
22         // 合并scope与用户信息
23         authoritySet.addAll(scopes);
24
25         JwtClaimsSet.Builder claims = context.getClaims();
26         // 将权限信息放入jwt的claims中（也可以生成一个以指定字符分割的字符串放入）
27         claims.claim("authorities", authoritySet);
28         // 放入其它自定内容
29         // 角色、头像...
30     }
31 };
32 }
```

这段代码将申请的scope与用户本身自带的权限合并后放入jwt中。



@稀土掘金技术社区





JwtAuthenticationConverter

自定义token部分就完成了，那么接下来就到resource server部分，早在最开始就添加了resource server的配置，将认证服务器也当做一个资源服务器，所以接下就在资源服务器文档中找到关于 `JwtAuthenticationConverter` 的[说明文档](#)。文档中有如下一段说明： *However, there are a number of circumstances where this default is insufficient. For example, some authorization servers don't use the scope attribute, but instead have their own custom attribute. Or, at other times, the resource server may need to adapt the attribute or a composition of attributes into internalized authorities.* 正好对应了上文说的自定义token，所以按照示例添加自己的jwt解析器

▼

java 复制代码

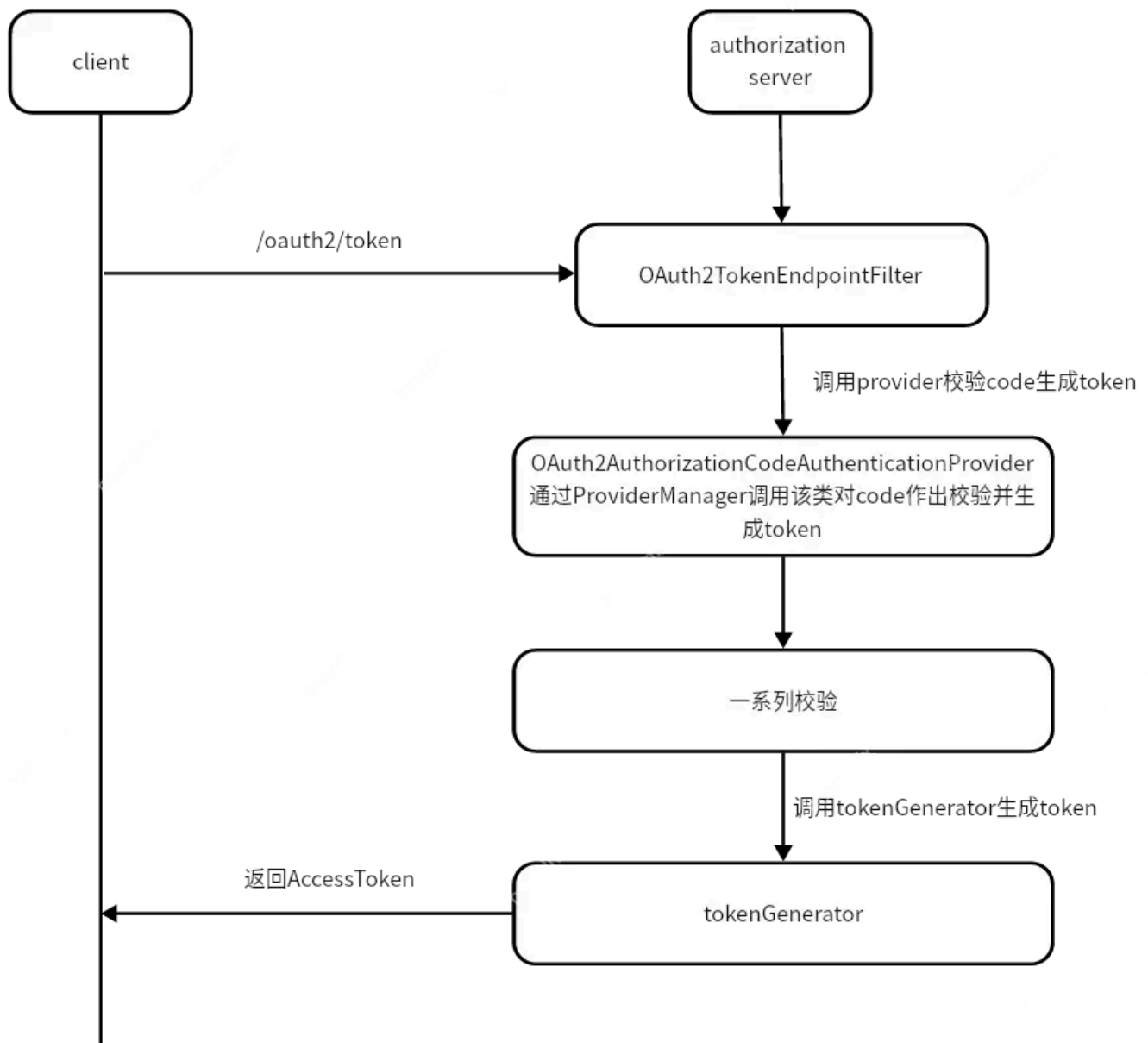
```
1 /**
2  * 自定义jwt解析器，设置解析出来的权限信息的前缀与在jwt中的key
3  *
4  * @return jwt解析器 JwtAuthenticationConverter
5  */
6 @Bean
```



```
10    grantedAuthoritiesConverter.setAuthorityPrefix("");
11    // 设置权限信息在jwt claims中的key
12    grantedAuthoritiesConverter.setAuthoritiesClaimName("authorities");
13
14    JwtAuthenticationConverter jwtAuthenticationConverter = new JwtAuthenticationConverter();
15    jwtAuthenticationConverter.setJwtGrantedAuthoritiesConverter(grantedAuthoritiesConverter);
16    return jwtAuthenticationConverter;
17 }
```

这里设置解析jwt时将权限key设置为上文中存入时的key，去除**SCOPE_**前缀。

流程图



@稀土掘金技术社区



查看token中的信息

生成一个access token, 在[JSON Web Tokens \(JWT\) 在线解密](#)中解析token, 查看token内容

JSON Web Tokens (JWT) 在线解密

提示: JWT是目前最流行的跨域认证解决方案,是一个开放式标准(RFC 7519),用于在各方之间以JSON对象安全传输信息。我们不记录令牌,所有验证和调试都在客户端进行。

Encoded 请在以下文本框粘贴令牌

[illegible]

Decode 以下是解密的内容

HEADER

```
{
  "kid": "1739fea4-052b-4f51-85e4-d828e58a5c55",
  "alg": "RS256"
```

PAYLOAD

```
{
  "sub": "admin",
  "aud": "device-message-client",
  "nbf": 1686106301,
  "scope": ["message.read"],
  "iss": "http://127.0.0.1:8080",
  "exp": 1686106601,
  "iat": 1686106301,
  "authorities": ["app", "ROLE_normal", "/test3", "web", "/test2",
    "ROLE_admin", "message.read", "ROLE_unAuthentication"]
}
```

STATUS

Decode Success

②③④世居柔佛-朥越社区

现在token中已经有权限信息了，接下来就编写一个测试接口并设置需要app的权限，来测试下解析token时是否会按照配置通过**authorities**来提取权限信息。

测试接口

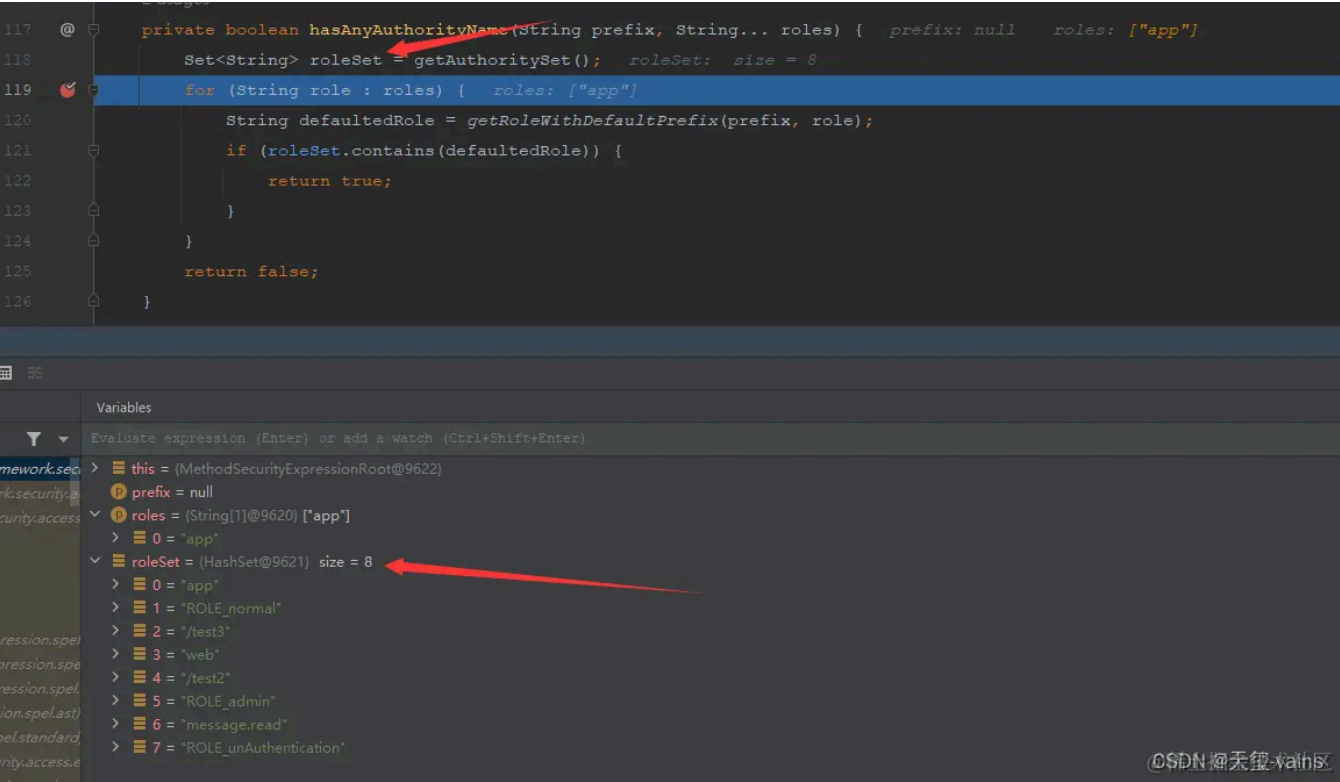


java 复制代码

```
1 @GetMapping("/app")
2 @PreAuthorize("hasAuthority('app')")
3 public String app() {
4     return "app";
5 }
```



断点查看拥有的权限



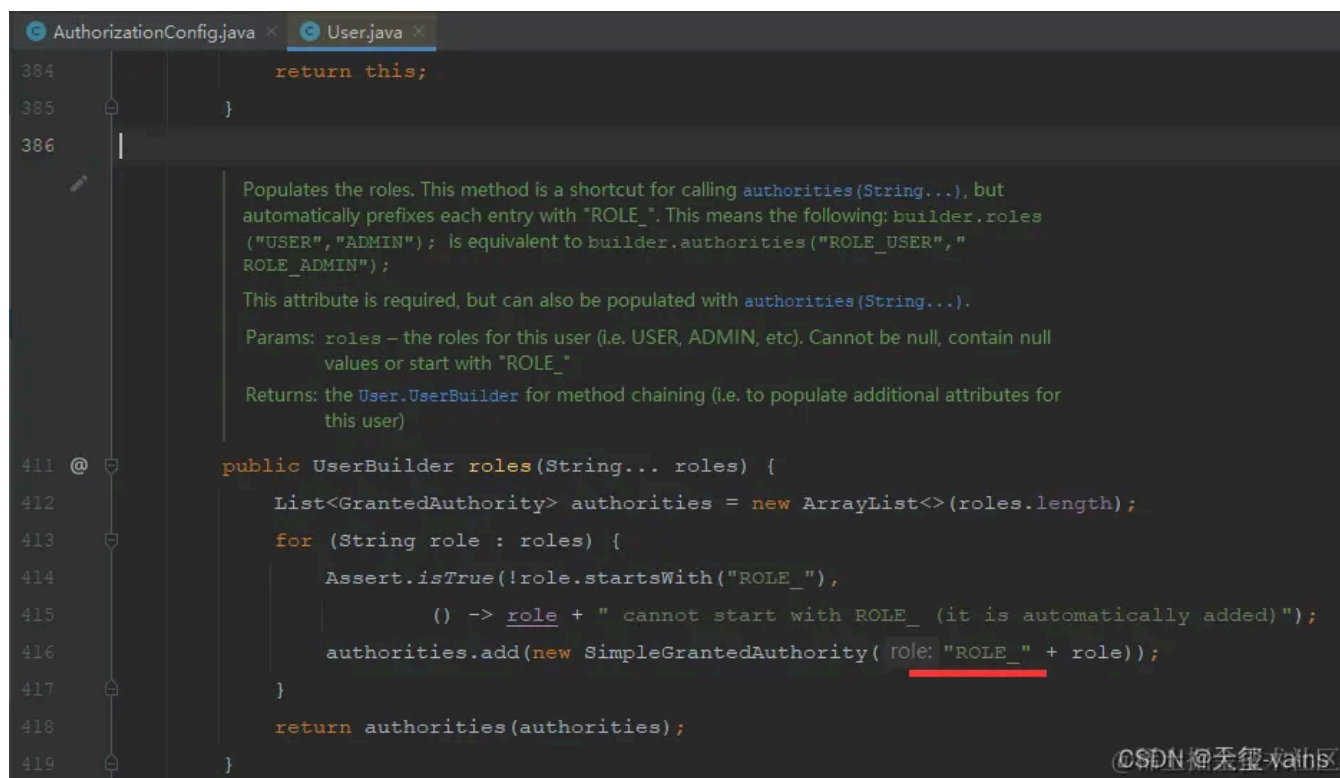
通过断点可以看到之前的自定token内容与自定jwt解析都已经生效了，已经将权限信息解析到当前用户信息的authorities中了。补一张用户信息的图片


```
* {@link UserDetailsService#loadUserByUsername(String)} 方法根据
* 账号查询用户信息，一般是重写该方法实现自己的逻辑
*
* @param passwordEncoder 密码解析器
* @return UserDetailsService
*/
@Bean
public UserDetailsService users(PasswordEncoder passwordEncoder) {
    UserDetails user = User.withUsername("admin")
        .password(passwordEncoder.encode( rawPassword: "123456"))
        .roles("admin", "normal", "unAuthentication")
        .authorities("app", "web", "/test2", "/test3")
        .build();
    return new InMemoryUserDetailsManager(user);
}
```

CSDN @天玺-vains

通

过对比可以更清楚的观察到前文作出的自定义配置已经生效。



CSDN @天玺-vains

至于设置的角色在权限中有一个 **ROLE_** 的前缀是因为roles的方法实际上是调用authorities方法，并且自动加上该前缀，如上图所示，这种情况在自己实现loadUserByUsername时避免这种情况。

最后经过两个简单的配置实现了自定义的token内容与解析器，这也正是框架灵活与支持高度自定义的体现，实际上这些操作的代码部分都比较少，多的是理论部分，结合文档与源码能够更好地理解框架，文档中对于某些类的应用一般会有详细的说明与示例存在。

如果有问题请在评论区指出，谢谢

AuthorizationConfig.java

最后在附一下AuthorizationConfig.java的代码

java 复制代码

```
1 package com.example.config;
2
3 import com.example.authorization.DeviceClientAuthenticationConverter;
4 import com.example.authorization.DeviceClientAuthenticationProvider;
5 import com.example.util.SecurityUtils;
6 import com.nimbusds.jose.jwk.JWKSet;
7 import com.nimbusds.jose.jwk.RSAKey;
8 import com.nimbusds.jose.jwk.source.ImmutableJWKSet;
9 import com.nimbusds.jose.jwk.source.JWKSource;
10 import com.nimbusds.jose.proc.SecurityContext;
11 import org.springframework.context.annotation.Bean;
12 import org.springframework.context.annotation.Configuration;
13 import org.springframework.http.MediaType;
14 import org.springframework.jdbc.core.JdbcTemplate;
15 import org.springframework.security.access.annotation.Secured;
16 import org.springframework.security.config.Customizer;
17 import org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
18 import org.springframework.security.config.annotation.web.builders.HttpSecurity;
19 import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
20 import org.springframework.security.core.GrantedAuthority;
21 import org.springframework.security.core.userdetails.User;
22 import org.springframework.security.core.userdetails.UserDetails;
23 import org.springframework.security.core.userdetails.UserDetailsService;
24 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
25 import org.springframework.security.crypto.password.PasswordEncoder;
26 import org.springframework.security.oauth2.core.AuthorizationGrantType;
27 import org.springframework.security.oauth2.core.ClientAuthenticationMethod;
28 import org.springframework.security.oauth2.core.oidc.OidcScopes;
29 import org.springframework.security.oauth2.jwt.JwtClaimsSet;
30 import org.springframework.security.oauth2.jwt.JwtDecoder;
31 import org.springframework.security.oauth2.server.authorization.JdbcOAuth2AuthorizationConsentSe
32 import org.springframework.security.oauth2.server.authorization.JdbcOAuth2AuthorizationService;
```



```
36 import org.springframework.security.oauth2.server.authorization.client.RegisteredClient;
37 import org.springframework.security.oauth2.server.authorization.client.RegisteredClientRepositor
38 import org.springframework.security.oauth2.server.authorization.config.annotation.web.configurat
39 import org.springframework.security.oauth2.server.authorization.config.annotation.web.configurer
40 import org.springframework.security.oauth2.server.authorization.settings.AuthorizationServerSett
41 import org.springframework.security.oauth2.server.authorization.settings.ClientSettings;
42 import org.springframework.security.oauth2.server.authorization.token.JwtEncodingContext;
43 import org.springframework.security.oauth2.server.authorization.token.OAuth2TokenCustomizer;
44 import org.springframework.security.oauth2.server.resource.authentication.JwtAuthenticationConve
45 import org.springframework.security.oauth2.server.resource.authentication.JwtGrantedAuthoritiesC
46 import org.springframework.security.provisioning.InMemoryUserDetailsManager;
47 import org.springframework.security.web.SecurityFilterChain;
48 import org.springframework.security.web.authentication.LoginUrlAuthenticationEntryPoint;
49 import org.springframework.security.web.util.matcher.MediaTypeRequestMatcher;
50
51 import java.security.KeyPair;
52 import java.security.KeyPairGenerator;
53 import java.security.interfaces.RSAPrivateKey;
54 import java.security.interfaces.RSAPublicKey;
55 import java.util.Collection;
56 import java.util.Collections;
57 import java.util.Optional;
58 import java.util.Set;
59 import java.util.UUID;
60 import java.util.stream.Collectors;
61
62 /**
63  * 认证配置
64  * {@link EnableMethodSecurity} 开启全局方法认证, 启用JSR250注解支持, 启用注解 {@link Secured} 支持,
65  * 在Spring Security 6.0版本中将@Configuration注解从@EnableWebSecurity, @EnableMethodSecurity, @En
66  * 和 @EnableGlobalAuthentication 中移除, 使用这些注解需手动添加 @Configuration 注解
67  * {@link EnableWebSecurity} 注解有两个作用:
68  * 1. 加载了WebSecurityConfiguration配置类, 配置安全认证策略。
69  * 2. 加载了AuthenticationConfiguration, 配置了认证信息。
70  *
71  * @author vains
72  */
73 @Configuration
74 @EnableWebSecurity
75 @EnableMethodSecurity(jsr250Enabled = true, securedEnabled = true)
76 public class AuthorizationConfig {
77
78     private static final String CUSTOM_CONSENT_PAGE_URI = "/oauth2/consent";
79
80     /**
81      * 配置端点的过滤器链
```

```
85     * @throws Exception 抛出
86     */
87     @Bean
88     public SecurityFilterChain authorizationServerSecurityFilterChain(HttpSecurity http,
89                                                                     RegisteredClientRepository
90                                                                     authorizationServerSetting
91     // 配置默认的设置, 忽略认证端点的csrf校验
92     OAuth2AuthorizationServerConfiguration.applyDefaultSecurity(http);
93
94     // 新建设备码converter和provider
95     DeviceClientAuthenticationConverter deviceClientAuthenticationConverter =
96         new DeviceClientAuthenticationConverter(
97             authorizationServerSettings.getDeviceAuthorizationEndpoint());
98     DeviceClientAuthenticationProvider deviceClientAuthenticationProvider =
99         new DeviceClientAuthenticationProvider(registeredClientRepository);
100
101
102     http.getConfigurer(OAuth2AuthorizationServerConfigurer.class)
103         // 开启OpenID Connect 1.0协议相关端点
104         .oidc(Customizer.withDefaults())
105         // 设置自定义用户确认授权页
106         .authorizationEndpoint(authorizationEndpoint -> authorizationEndpoint.consentPag
107         // 设置设备码用户验证url(自定义用户验证页)
108         .deviceAuthorizationEndpoint(deviceAuthorizationEndpoint ->
109             deviceAuthorizationEndpoint.verificationUri("/activate")
110         )
111         // 设置验证设备码用户确认页面
112         .deviceVerificationEndpoint(deviceVerificationEndpoint ->
113             deviceVerificationEndpoint.consentPage(CUSTOM_CONSENT_PAGE_URI)
114         )
115         .clientAuthentication(clientAuthentication ->
116             // 客户端认证添加设备码的converter和provider
117             clientAuthentication
118                 .authenticationConverter(deviceClientAuthenticationConverter)
119                 .authenticationProvider(deviceClientAuthenticationProvider)
120         );
121     http
122         // 当未登录时访问认证端点时重定向至login页面
123         .exceptionHandling((exceptions) -> exceptions
124             .defaultAuthenticationEntryPointFor(
125                 new LoginUrlAuthenticationEntryPoint("/login"),
126                 new MediaTypeRequestMatcher(MediaType.TEXT_HTML)
127             )
128         )
129         // 处理使用access token访问用户信息端点和客户端注册端点
130         .oauth2ResourceServer((resourceServer) -> resourceServer
```



```
134     }
135
136     /**
137      * 配置认证相关的过滤器链
138      *
139      * @param http spring security核心配置类
140      * @return 过滤器链
141      * @throws Exception 抛出
142      */
143     @Bean
144     public SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http) throws Exception {
145         http.authorizeHttpRequests((authorize) -> authorize
146             // 放行静态资源
147             .requestMatchers("/assets/**", "/webjars/**", "/login").permitAll()
148             .anyRequest().authenticated()
149         )
150         // 指定登录页面
151         .formLogin(formLogin ->
152             formLogin.loginPage("/login")
153         );
154         // 添加BearerTokenAuthenticationFilter, 将认证服务当做一个资源服务, 解析请求头中的token
155         http.oauth2ResourceServer((resourceServer) -> resourceServer
156             .jwt(Customizer.withDefaults())
157             .accessDeniedHandler(SecurityUtils::exceptionHandler)
158             .authenticationEntryPoint(SecurityUtils::exceptionHandler)
159         );
160
161         return http.build();
162     }
163
164     /**
165      * 自定义jwt, 将权限信息放至jwt中
166      *
167      * @return OAuth2TokenCustomizer的实例
168      */
169     @Bean
170     public OAuth2TokenCustomizer<JwtEncodingContext> oAuth2TokenCustomizer() {
171         return context -> {
172             // 检查登录用户信息是不是UserDetails, 排除掉没有用户参与的流程
173             if (context.getPrincipal().getPrincipal() instanceof UserDetails user) {
174                 // 获取申请的scopes
175                 Set<String> scopes = context.getAuthorizedScopes();
176                 // 获取用户的权限
177                 Collection<? extends GrantedAuthority> authorities = user.getAuthorities();
178                 // 提取权限并转为字符串
179                 Set<String> authoritySet = Optional.ofNullable(authorities).orElse(Collections.e
```

```
183         .collect(Collectors.toSet());
184
185         // 合并scope与用户信息
186         authoritySet.addAll(scopes);
187
188         JwtClaimsSet.Builder claims = context.getClaims();
189         // 将权限信息放入jwt的claims中（也可以生成一个以指定字符分割的字符串放入）
190         claims.claim("authorities", authoritySet);
191         // 放入其它自定内容
192         // 角色、头像...
193     }
194 };
195 }
196
197 /**
198  * 自定义jwt解析器，设置解析出来的权限信息的前缀与在jwt中的key
199  *
200  * @return jwt解析器 JwtAuthenticationConverter
201  */
202 @Bean
203 public JwtAuthenticationConverter jwtAuthenticationConverter() {
204     JwtGrantedAuthoritiesConverter grantedAuthoritiesConverter = new JwtGrantedAuthoritiesCo
205     // 设置解析权限信息的前缀，设置为空是去掉前缀
206     grantedAuthoritiesConverter.setAuthorityPrefix("");
207     // 设置权限信息在jwt claims中的key
208     grantedAuthoritiesConverter.setAuthoritiesClaimName("authorities");
209
210     JwtAuthenticationConverter jwtAuthenticationConverter = new JwtAuthenticationConverter()
211     jwtAuthenticationConverter.setJwtGrantedAuthoritiesConverter(grantedAuthoritiesConverter
212     return jwtAuthenticationConverter;
213 }
214
215 /**
216  * 配置密码解析器，使用BCrypt的方式对密码进行加密和验证
217  *
218  * @return BCryptPasswordEncoder
219  */
220 @Bean
221 public PasswordEncoder passwordEncoder() {
222     return new BCryptPasswordEncoder();
223 }
224
225 /**
226  * 配置客户端Repository
227  *
228  * @param jdbcTemplate db 数据源信息
```




```
232 @Bean
233 public RegisteredClientRepository registeredClientRepository(JdbcTemplate jdbcTemplate, Pass
234     RegisteredClient registeredClient = RegisteredClient.withId(UUID.randomUUID().toString())
235         // 客户端id
236         .clientId("messaging-client")
237         // 客户端密钥, 使用密码解析器加密
238         .clientSecret(passwordEncoder.encode("123456"))
239         // 客户端认证方式, 基于请求头的认证
240         .clientAuthenticationMethod(ClientAuthenticationMethod.CLIENT_SECRET_BASIC)
241         // 配置资源服务器使用该客户端获取授权时支持的方式
242         .authorizationGrantType(AuthorizationGrantType.AUTHORIZATION_CODE)
243         .authorizationGrantType(AuthorizationGrantType.REFRESH_TOKEN)
244         .authorizationGrantType(AuthorizationGrantType.CLIENT_CREDENTIALS)
245         // 授权码模式回调地址, oauth2.1已改为精准匹配, 不能只设置域名, 并且屏蔽了localhost, 本
246         .redirectUri("http://127.0.0.1:8080/login/oauth2/code/messaging-client-oidc")
247         .redirectUri("https://www.baidu.com")
248         // 该客户端的授权范围, OPENID与PROFILE是IdToken的scope, 获取授权时请求OPENID的scope#
249         .scope(OidcScopes.OPENID)
250         .scope(OidcScopes.PROFILE)
251         // 自定scope
252         .scope("message.read")
253         .scope("message.write")
254         // 客户端设置, 设置用户需要确认授权
255         .clientSettings(ClientSettings.builder().requireAuthorizationConsent(true).build
256         .build());
257
258 // 基于db存储客户端, 还有一个基于内存的实现 InMemoryRegisteredClientRepository
259 JdbcRegisteredClientRepository registeredClientRepository = new JdbcRegisteredClientRepo
260
261 // 初始化客户端
262 RegisteredClient repositoryByClientId = registeredClientRepository.findByClientId(regist
263 if (repositoryByClientId == null) {
264     registeredClientRepository.save(registeredClient);
265 }
266 // 设备码授权客户端
267 RegisteredClient deviceClient = RegisteredClient.withId(UUID.randomUUID().toString())
268     .clientId("device-message-client")
269     // 公共客户端
270     .clientAuthenticationMethod(ClientAuthenticationMethod.NONE)
271     // 设备码授权
272     .authorizationGrantType(AuthorizationGrantType.DEVICE_CODE)
273     .authorizationGrantType(AuthorizationGrantType.REFRESH_TOKEN)
274     // 自定scope
275     .scope("message.read")
276     .scope("message.write")
277     .build();
```

```
281     }
282
283     // PKCE客户端
284     RegisteredClient pkceClient = RegisteredClient.withId(UUID.randomUUID().toString())
285         .clientId("pkce-message-client")
286         // 公共客户端
287         .clientAuthenticationMethod(ClientAuthenticationMethod.NONE)
288         // 授权码模式, 因为是扩展授权码流程, 所以流程还是授权码的流程, 改变的只是参数
289         .authorizationGrantType(AuthorizationGrantType.AUTHORIZATION_CODE)
290         .authorizationGrantType(AuthorizationGrantType.REFRESH_TOKEN)
291         // 授权码模式回调地址, oauth2.1已改为精准匹配, 不能只设置域名, 并且屏蔽了localhost, 本
292         .redirectUri("http://127.0.0.1:8080/login/oauth2/code/messaging-client-oidc")
293         .clientSettings(ClientSettings.builder().requireProofKey(Boolean.TRUE).build())
294         // 自定scope
295         .scope("message.read")
296         .scope("message.write")
297         .build();
298     RegisteredClient findPkceClient = registeredClientRepository.findByClientId(pkceClient.g
299     if (findPkceClient == null) {
300         registeredClientRepository.save(pkceClient);
301     }
302     return registeredClientRepository;
303 }
304
305 /**
306  * 配置基于db的oauth2的授权管理服务
307  *
308  * @param jdbcTemplate          db数据源信息
309  * @param registeredClientRepository 上边注入的客户端repository
310  * @return JdbcOAuth2AuthorizationService
311  */
312 @Bean
313 public OAuth2AuthorizationService authorizationService(JdbcTemplate jdbcTemplate, Registered
314     // 基于db的oauth2认证服务, 还有一个基于内存的服务实现InMemoryOAuth2AuthorizationService
315     return new JdbcOAuth2AuthorizationService(jdbcTemplate, registeredClientRepository);
316 }
317
318 /**
319  * 配置基于db的授权确认管理服务
320  *
321  * @param jdbcTemplate          db数据源信息
322  * @param registeredClientRepository 客户端repository
323  * @return JdbcOAuth2AuthorizationConsentService
324  */
325 @Bean
326 public OAuth2AuthorizationConsentService authorizationConsentService(JdbcTemplate jdbcTempla
```



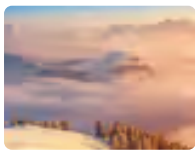
```
330
331 /**
332  * 配置jwk源，使用非对称加密，公开用于检索匹配指定选择器的JWK的方法
333  *
334  * @return JWKSource
335  */
336 @Bean
337 public JWKSource<SecurityContext> jwkSource() {
338     KeyPair keyPair = generateRsaKey();
339     RSAPublicKey publicKey = (RSAPublicKey) keyPair.getPublic();
340     RSAPrivateKey privateKey = (RSAPrivateKey) keyPair.getPrivate();
341     RSAKey rsaKey = new RSAKey.Builder(publicKey)
342         .privateKey(privateKey)
343         .keyID(UUID.randomUUID().toString())
344         .build();
345     JWKSet jwkSet = new JWKSet(rsaKey);
346     return new ImmutableJWKSet<>(jwkSet);
347 }
348
349 /**
350  * 生成rsa密钥对，提供给jwk
351  *
352  * @return 密钥对
353  */
354 private static KeyPair generateRsaKey() {
355     KeyPair keyPair;
356     try {
357         KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
358         keyPairGenerator.initialize(2048);
359         keyPair = keyPairGenerator.generateKeyPair();
360     } catch (Exception ex) {
361         throw new IllegalStateException(ex);
362     }
363     return keyPair;
364 }
365
366 /**
367  * 配置jwt解析器
368  *
369  * @param jwkSource jwk源
370  * @return JwtDecoder
371  */
372 @Bean
373 public JwtDecoder jwtDecoder(JWKSource<SecurityContext> jwkSource) {
374     return OAuth2AuthorizationServerConfiguration.jwtDecoder(jwkSource);
375 }
```

```
379      *
380      * @return AuthorizationServerSettings
381      */
382      @Bean
383      public AuthorizationServerSettings authorizationServerSettings() {
384          return AuthorizationServerSettings.builder().build();
385      }
386
387      /**
388       * 先暂时配置一个基于内存的用户，框架在用户认证时会默认调用
389       * {@link UserDetailsService#loadUserByUsername(String)} 方法根据
390       * 账号查询用户信息，一般是重写该方法实现自己的逻辑
391       *
392       * @param passwordEncoder 密码解析器
393       * @return UserDetailsService
394       */
395      @Bean
396      public UserDetailsService users(PasswordEncoder passwordEncoder) {
397          UserDetails user = User.withUsername("admin")
398              .password(passwordEncoder.encode("123456"))
399              .roles("admin", "normal", "unAuthentication")
400              .authorities("app", "web", "/test2", "/test3")
401              .build();
402          return new InMemoryUserDetailsManager(user);
403      }
404
405 }
```

标签： [Spring](#) [Spring Boot](#) 话题： [我的技术写作成长之路](#)

本文收录于以下专栏

◀ 1 / 2 ▶



Spring Authorization Server
Spring Authorization Server系列文章
177 订阅 · 25 篇文章

专栏目录

订阅

上一篇 [Spring Authorization Server...](#) 下一篇 [Spring Authorization Server...](#)



平等表达，友善交流



0 / 1000  发送

最热 最新

咸蛋超超人 Java工程师

大佬 jwt 越来越长 返回的权限标识也越来越多咋办

1月前  点赞  1 ...

叹雪飞花 [作者](#)：用匿名token(opaque token)或者自定义一下token生成器用自定义加密算法包装一下，记得解析时要加对应的处理。

1月前  点赞  回复 ...

wfhusb

JwtDecoder 不是 自动注入到容器中的吗？ 不用 再@Bean吧？

3月前  点赞  2 ...

叹雪飞花 [作者](#)：认证服务需要注册的，我是这样理解的：自动注册说到底也是通过配置认证服务的issue-uri来进行注册的，是通过这个地址请求认证服务来获取相关信息的，但是认证服务不能自己配置自己

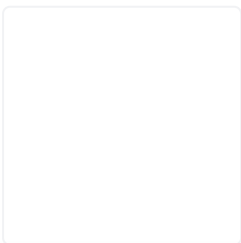
3月前  点赞  回复 ...

叹雪飞花 [作者](#)：还有就是认证服务需要生成一个jwtDecoder让其它资源服务获取

3月前  点赞  回复 ...

开发者充电站 @公众号：开发者充电站

不知道是不是版本不一致，我这看到 User源码里,每次调用 Builder 的authorities实际上都是 new 了一个 list 出来，也就是说用户的权限和角色只保留其一，你在调试的时候可以同时看到 roles 和authorities，估计是老版本没有重新 new list



userDetailsService的，这种生成固定用户的方式不一致也没事儿

6月前 点赞 回复

...

查看全部 14 条评论

目录 收起

前言

OAuth2TokenCustomizer

JwtAuthenticationConverter

流程图

测试

查看token中的信息

测试接口

断点查看拥有的权限

总结

AuthorizationConfig.java

相关推荐

Spring Authorization Server入门 (五) 自定义异常响应配置

1.8k阅读 · 4点赞

SpringSecurityOAuth已停更，来看一看进化版本Spring Authorization Server

372阅读 · 0点赞

Spring Authorization Server优化篇：自定义UserDetailsService实现从数据库获取用户信息

1.1k阅读 · 5点赞

Spring Data Redis工具类

207阅读 · 3点赞

Spring Authorization Server优化篇：持久化JWKSource，解决重启后无法解析AccessToken问题

精选内容

入职一家新公司如何配置Git及远程开发？

安妮的心动录 · 760阅读 · 4点赞

ThreadPoolExecutor在执行过程中出现异常

Lxlxxx · 694阅读 · 1点赞

SSE：ChatGPT的对话传输协议！

石头聊技术 · 862阅读 · 2点赞

Python数据可视化-销售数据可视化：Bokeh 条形图示例

CodeJourney · 122阅读 · 1点赞

没看过CountDownLatch源码，别说是Java高级开发

一灯架构 · 147阅读 · 0点赞

为你推荐

Spring Authorization Server入门 (二) Spring Boot整合Spring Authorization Server

叹雪飞花 9月前 👁 6.6k 👍 31 💬 76 Java

Spring Authorization Server入门 (十二) 实现授权码模式使用前后端分离的登录页面

叹雪飞花 8月前 👁 4.7k 👍 24 💬 65 后端 Spring ... Spring

Spring Authorization Server入门 (十) 添加短信验证码方式登录

叹雪飞花 9月前 👁 3.4k 👍 20 💬 9 Spring Spring ...

Spring Authorization Server入门 (八) Spring Boot引入Security OAuth2 Client对接认...

叹雪飞花 9月前 👁 2.7k 👍 13 💬 43 Spring ... Spring

Spring Authorization Server入门 (十六) Spring Cloud Gateway对接认证服务

叹雪飞花 6月前 👁 2.5k 👍 17 💬 44 Spring ... Spring ... 安全

Spring Authorization Server入门 (十三) 实现联合身份认证，集成Github与Gitee的OAu...

叹雪飞花 7月前 👁 2.3k 👍 13 💬 51 Spring Spring ... 安全

Spring Authorization Server入门 (十一) 自定义grant_type(短信认证登录)获取token

叹雪飞花 9月前 👁 2.4k 👍 15 💬 39 Spring Spring ... 安全

SpringBoot3.x最简集成SpringDoc-OpenApi

叹雪飞花

4月前

 2.3k

 16

 评论

后端

Spring ...

Java

Spring Authorization Server入门 (九) Spring Boot引入Resource Server对接认证服务

叹雪飞花

9月前

 1.8k

 13

 8

Spring

Spring ...

Spring Authorization Server优化篇：添加Redis缓存支持和统一响应类

叹雪飞花

8月前

 1.7k

 6

 12

Spring

Spring ...

安全

Spring Authorization Server入门 (十五) 分离授权确认与设备码校验页面

叹雪飞花

7月前

 1.6k

 14

 8

Spring ...

Spring

Vue.js


Spring Authorization Server入门 (十九) 基于Redis的Token、客户端信息和授权确认信...

叹雪飞花

4月前

 1.2k

 8

 19

Spring ...

后端

Redis

Spring Authorization Server入门 (二十) 实现二维码扫码登录

叹雪飞花

1月前

 890

 16

 6

Spring ...

Spring

Java

Spring Authorization Server入门 (十七) Vue项目使用授权码模式对接认证服务

叹雪飞花

6月前

 751

 9

 12

Vue.js

安全

Spring ...