spring-security-oauth2中添加自定义授权模式的思路

# 前言

在 oauth2 的授权模式中有4种：

* 授权码模式
* 隐式授权模式
* 密码模式
* 客户端模式

但如果我们想要增加一个自定义的授权模式，又该怎么做呢？

# 源码分析

## 判断是何种授权模式

1. **public** **class** CompositeTokenGranter **implements** TokenGranter {
3. **private** **final** List<TokenGranter> tokenGranters;
5. **public** CompositeTokenGranter(List<TokenGranter> tokenGranters) {
6. **this**.tokenGranters = **new** ArrayList<TokenGranter>(tokenGranters);
7. }
9. //四种授权模式+刷新令牌的模式根据grant\_type判断
10. **public** OAuth2AccessToken grant(String grantType, TokenRequest tokenRequest) {
11. **for** (TokenGranter granter : tokenGranters) {
12. OAuth2AccessToken grant = granter.grant(grantType, tokenRequest);
13. **if** (grant!=**null**) {
14. **return** grant;
15. }
16. }
17. **return** **null**;
18. }
20. **public** **void** addTokenGranter(TokenGranter tokenGranter) {
21. **if** (tokenGranter == **null**) {
22. **throw** **new** IllegalArgumentException("Token granter is null");
23. }
24. tokenGranters.add(tokenGranter);
25. }
27. }

## 授权模式的配置

1. **public** **final** **class** AuthorizationServerEndpointsConfigurer {
3. // 省略部分代码
5. **private** TokenGranter tokenGranter;
7. **public** AuthorizationServerEndpointsConfigurer tokenGranter(TokenGranter tokenGranter) {
8. **this**.tokenGranter = tokenGranter;
9. **return** **this**;
10. }
12. // 默认的四种授权模式+刷新令牌的模式的配置
13. **private** TokenGranter tokenGranter() {
14. **if** (tokenGranter == **null**) {
15. tokenGranter = **new** TokenGranter() {
16. **private** CompositeTokenGranter delegate;
18. @Override
19. **public** OAuth2AccessToken grant(String grantType, TokenRequest tokenRequest) {
20. **if** (delegate == **null**) {
21. delegate = **new** CompositeTokenGranter(getDefaultTokenGranters());
22. }
23. **return** delegate.grant(grantType, tokenRequest);
24. }
25. };
26. }
27. **return** tokenGranter;
28. }
30. // 默认的四种授权模式+刷新令牌的模式的配置
31. **private** List<TokenGranter> getDefaultTokenGranters() {
32. ClientDetailsService clientDetails = clientDetailsService();
33. AuthorizationServerTokenServices tokenServices = tokenServices();
34. AuthorizationCodeServices authorizationCodeServices = authorizationCodeServices();
35. OAuth2RequestFactory requestFactory = requestFactory();
37. List<TokenGranter> tokenGranters = **new** ArrayList<TokenGranter>();
38. // 添加授权码模式
39. tokenGranters.add(**new** AuthorizationCodeTokenGranter(tokenServices, authorizationCodeServices, clientDetails,
40. requestFactory));
41. // 添加刷新令牌的模式
42. tokenGranters.add(**new** RefreshTokenGranter(tokenServices, clientDetails, requestFactory));
43. // 添加隐式授权模式
44. ImplicitTokenGranter implicit = **new** ImplicitTokenGranter(tokenServices, clientDetails, requestFactory);
45. tokenGranters.add(implicit);
46. // 添加客户端模式
47. tokenGranters.add(**new** ClientCredentialsTokenGranter(tokenServices, clientDetails, requestFactory));
48. **if** (authenticationManager != **null**) {
49. // 添加密码模式
50. tokenGranters.add(**new** ResourceOwnerPasswordTokenGranter(authenticationManager, tokenServices,
51. clientDetails, requestFactory));
52. }
54. // 可以复制相关代码，然后这里添加自定义的授权模式
56. **return** tokenGranters;
57. }
58. }

看到这里就可以发现 spring 已经把默认的四种授权模式+刷新令牌的模式的配置在代码中写死了!

那又如何添加自定义的授权模式呢？

我的思路是这样的：

直接把这部分的代码复制，在其中添加自定义的授权模式。

我直接把密码模式复制，将其中的 GRANT\_TYPE 的值改为 sms\_code,然后使用 /oauth/token?grant\_type=sms\_code&scope=read&username=user&password=123456 来验证结果。

# 自定义授权模式

注意这个自定义授权模式代码只是复制密码模式的实现，只是改变了 GRANT\_TYPE 的值，来验证自定义授权模式的可行性。

1. /\*\*
2. \* @author fengxuechao
3. \* @version 0.1
4. \* @date 2019/5/17
5. \*/
6. **public** **class** SmsCodeTokenGranter **extends** AbstractTokenGranter {
8. // 仅仅复制了 ResourceOwnerPasswordTokenGranter，只是改变了 GRANT\_TYPE 的值，来验证自定义授权模式的可行性
9. **private** **static** **final** String GRANT\_TYPE = "sms\_code";
11. **private** **final** AuthenticationManager authenticationManager;
13. **public** SmsCodeTokenGranter(
14. AuthenticationManager authenticationManager,
15. AuthorizationServerTokenServices tokenServices,
16. ClientDetailsService clientDetailsService,
17. OAuth2RequestFactory requestFactory) {
18. **this**(authenticationManager, tokenServices, clientDetailsService, requestFactory, GRANT\_TYPE);
20. }
22. **protected** SmsCodeTokenGranter(
23. AuthenticationManager authenticationManager,
24. AuthorizationServerTokenServices tokenServices,
25. ClientDetailsService clientDetailsService,
26. OAuth2RequestFactory requestFactory,
27. String grantType) {
28. **super**(tokenServices, clientDetailsService, requestFactory, grantType);
29. **this**.authenticationManager = authenticationManager;
30. }
32. @Override
33. **protected** OAuth2Authentication getOAuth2Authentication(ClientDetails client, TokenRequest tokenRequest) {
34. Map<String, String> parameters = **new** LinkedHashMap<String, String>(tokenRequest.getRequestParameters());
35. String username = parameters.get("username");
36. String password = parameters.get("password");
37. // Protect from downstream leaks of password
38. parameters.remove("password");
40. Authentication userAuth = **new** UsernamePasswordAuthenticationToken(username, password);
41. ((AbstractAuthenticationToken) userAuth).setDetails(parameters);
42. **try** {
43. userAuth = authenticationManager.authenticate(userAuth);
44. } **catch** (AccountStatusException ase) {
45. //covers expired, locked, disabled cases (mentioned in section 5.2, draft 31)
46. **throw** **new** InvalidGrantException(ase.getMessage());
47. } **catch** (BadCredentialsException e) {
48. // If the username/password are wrong the spec says we should send 400/invalid grant
49. **throw** **new** InvalidGrantException(e.getMessage());
50. }
51. **if** (userAuth == **null** || !userAuth.isAuthenticated()) {
52. **throw** **new** InvalidGrantException("Could not authenticate user: " + username);
53. }
55. OAuth2Request storedOAuth2Request = getRequestFactory().createOAuth2Request(client, tokenRequest);
56. **return** **new** OAuth2Authentication(storedOAuth2Request, userAuth);
57. }
58. }

# 授权模式配置类

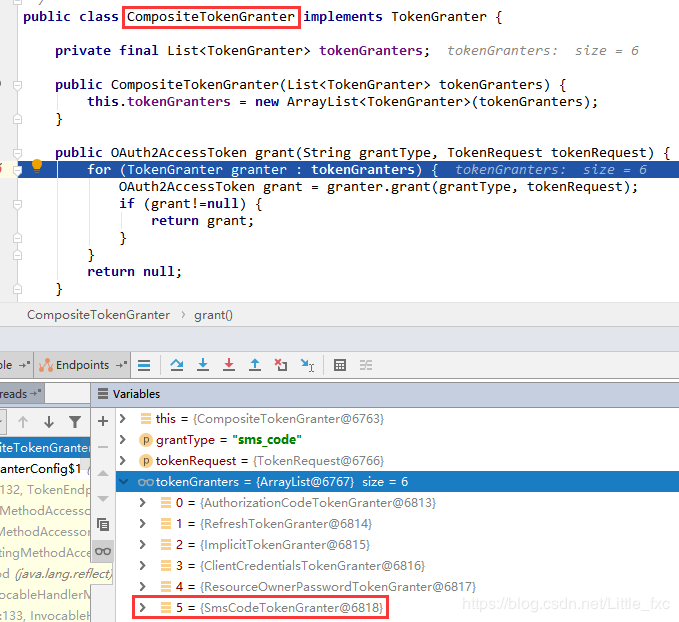
1. /\*\*
2. \* @author fengxuechao
3. \* @version 0.1
4. \* @date 2019/5/17
5. \*/
6. @Configuration
7. @Profile("inMemory")
8. **public** **class** TokenGranterConfig {
10. @Autowired
11. **private** ClientDetailsService clientDetailsService;
13. @Autowired
14. **private** UserDetailsService userDetailsService;
16. @Autowired
17. **private** AuthenticationManager authenticationManager;
19. @Autowired
20. **private** TokenStore tokenStore;
22. @Autowired
23. TokenEnhancer tokenEnhancer;
25. **private** AuthorizationCodeServices authorizationCodeServices;
27. **private** **boolean** reuseRefreshToken = **true**;
29. **private** AuthorizationServerTokenServices tokenServices;
31. **private** TokenGranter tokenGranter;
33. /\*\*
34. \* 授权模式
35. \*
36. \* @return
37. \*/
38. @Bean
39. **public** TokenGranter tokenGranter() {
40. **if** (tokenGranter == **null**) {
41. tokenGranter = **new** TokenGranter() {
42. **private** CompositeTokenGranter delegate;
44. @Override
45. **public** OAuth2AccessToken grant(String grantType, TokenRequest tokenRequest) {
46. **if** (delegate == **null**) {
47. delegate = **new** CompositeTokenGranter(getDefaultTokenGranters());
48. }
49. **return** delegate.grant(grantType, tokenRequest);
50. }
51. };
52. }
53. **return** tokenGranter;
54. }
56. /\*\*
57. \* 程序支持的授权类型
58. \*
59. \* @return
60. \*/
61. **private** List<TokenGranter> getDefaultTokenGranters() {
62. AuthorizationServerTokenServices tokenServices = tokenServices();
63. AuthorizationCodeServices authorizationCodeServices = authorizationCodeServices();
64. OAuth2RequestFactory requestFactory = requestFactory();
66. List<TokenGranter> tokenGranters = **new** ArrayList<TokenGranter>();
67. // 添加授权码模式
68. tokenGranters.add(**new** AuthorizationCodeTokenGranter(tokenServices, authorizationCodeServices, clientDetailsService, requestFactory));
69. // 添加刷新令牌的模式
70. tokenGranters.add(**new** RefreshTokenGranter(tokenServices, clientDetailsService, requestFactory));
71. // 添加隐士授权模式
72. tokenGranters.add(**new** ImplicitTokenGranter(tokenServices, clientDetailsService, requestFactory));
73. // 添加客户端模式
74. tokenGranters.add(**new** ClientCredentialsTokenGranter(tokenServices, clientDetailsService, requestFactory));
75. **if** (authenticationManager != **null**) {
76. // 添加密码模式
77. tokenGranters.add(**new** ResourceOwnerPasswordTokenGranter(authenticationManager, tokenServices, clientDetailsService, requestFactory));
78. // 添加自定义授权模式（实际是密码模式的复制）
79. tokenGranters.add(**new** SmsCodeTokenGranter(authenticationManager, tokenServices, clientDetailsService, requestFactory));
80. }
81. **return** tokenGranters;
82. }
84. /\*\*
85. \* TokenServices
86. \*
87. \* @return
88. \*/
89. **private** AuthorizationServerTokenServices tokenServices() {
90. **if** (tokenServices != **null**) {
91. **return** tokenServices;
92. }
93. **this**.tokenServices = createDefaultTokenServices();
94. **return** tokenServices;
95. }
97. /\*\*
98. \* 授权码API
99. \*
100. \* @return
101. \*/
102. **private** AuthorizationCodeServices authorizationCodeServices() {
103. **if** (authorizationCodeServices == **null**) {
104. authorizationCodeServices = **new** InMemoryAuthorizationCodeServices();
105. }
106. **return** authorizationCodeServices;
107. }
109. /\*\*
110. \* OAuth2RequestFactory的默认实现，它初始化参数映射中的字段，
111. \* 验证授权类型(grant\_type)和范围(scope)，并使用客户端的默认值填充范围(scope)（如果缺少这些值）。
112. \*
113. \* @return
114. \*/
115. **private** OAuth2RequestFactory requestFactory() {
116. **return** **new** DefaultOAuth2RequestFactory(clientDetailsService);
117. }
119. /\*\*
120. \* 默认 TokenService
121. \*
122. \* @return
123. \*/
124. **private** DefaultTokenServices createDefaultTokenServices() {
125. DefaultTokenServices tokenServices = **new** DefaultTokenServices();
126. tokenServices.setTokenStore(tokenStore);
127. tokenServices.setSupportRefreshToken(**true**);
128. tokenServices.setReuseRefreshToken(reuseRefreshToken);
129. tokenServices.setClientDetailsService(clientDetailsService);
130. tokenServices.setTokenEnhancer(tokenEnhancer);
131. addUserDetailsService(tokenServices, **this**.userDetailsService);
132. **return** tokenServices;
133. }
135. /\*\*
136. \* 添加预身份验证
137. \*
138. \* @param tokenServices
139. \* @param userDetailsService
140. \*/
141. **private** **void** addUserDetailsService(DefaultTokenServices tokenServices, UserDetailsService userDetailsService) {
142. **if** (userDetailsService != **null**) {
143. PreAuthenticatedAuthenticationProvider provider = **new** PreAuthenticatedAuthenticationProvider();
144. provider.setPreAuthenticatedUserDetailsService(**new** UserDetailsByNameServiceWrapper<PreAuthenticatedAuthenticationToken>(userDetailsService));
145. tokenServices.setAuthenticationManager(**new** ProviderManager(Arrays.<AuthenticationProvider>asList(provider)));
146. }
147. }
148. }

# 授权认证服务端点配置

1. **package** com.fengxuechao.examples.auth.config.inmemory;
2. /\*\*
3. \* @author fengxuechao
4. \* @version 0.1
5. \* @date 2019/5/8
6. \*/
7. @Slf4j
8. @EnableAuthorizationServer
9. @Configuration
10. @Profile("inMemory")
11. **public** **class** AuthorizationServerConfigInMemory **extends** AuthorizationServerConfigurerAdapter {
13. // 省略部分代码
15. @Autowired
16. **private** TokenGranter tokenGranter;
18. /\*\*
19. \* 认证服务器节点配置
20. \*
21. \* @param endpoints
22. \* @throws Exception
23. \*/
24. @Override
25. **public** **void** configure(AuthorizationServerEndpointsConfigurer endpoints) **throws** Exception {
26. endpoints
27. .tokenGranter(tokenGranter) // 四种授权模式+刷新令牌的模式+自定义授权模式
28. .tokenStore(tokenStore)
29. .approvalStore(approvalStore)
30. .userDetailsService(userDetailsService)
31. .authenticationManager(authenticationManager)
32. .setClientDetailsService(clientDetailsService);
33. }
34. }

# 演示效果

下面这张图显示了我们系统中拥有了自定义的授权模式



获取token请求演示：

POST http://localhost:8080/oauth/token?grant\_type=sms\_code&scope=read&username=user&password=123456

HTTP/1.1 200

X-Application-Context: application:inMemory

Cache-Control: no-store

Pragma: no-cache

X-Content-Type-Options: nosniff

X-XSS-Protection: 1; mode=block

X-Frame-Options: DENY

Content-Type: application/json;charset=UTF-8

Transfer-Encoding: chunked

Date: Tue, 18 Jun 2019 07:13:12 GMT

{

  "access\_token": "19f2e44a-6c9a-45c4-be7e-0aada6a0a9e6",

  "token\_type": "bearer",

  "refresh\_token": "f59336a8-03c4-4c85-bc31-16c6d80f1381",

  "expires\_in": 359,

  "scope": "read",

  "organization": "userWqTI"

}

Response code: 200; Time: 335ms; Content length: 190 bytes