

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

叹雪飞花

2023-10-29

 1,250

 阅读15分钟

关注

## 怎么使用Spring Data Redis实现Spring Authorization Serv

本文对应的是文档中的 [How-to: Implement core services with JPA](#)，文档中使用Jpa实现了核心的三个服务类：授权信息、客户端信息和授权确认的服务；本文会使用Spring Data Redis参考文档来添加新的实现。在这里也放一下文档中的一句话：**「本指南的目的是为您自己实现这些服务提供一个起点，以便您可以根据自己的需要进行修改。」**

### 修改RedisConfig类 重要

**「添加注解 @EnableRedisRepositories(enableKeyspaceEvents = RedisKeyValueAdapter.EnableKeyspaceEvents.ON\_STARTUP)，在启动后添加 Redis数据失效时间，自动删除 @Indexed 注解生成的索引(Secondary Indexes)。」**

▼

java 复制代码

```
1 package com.example.config;
2
3 import com.fasterxml.jackson.annotation.JsonAutoDetect;
4 import com.fasterxml.jackson.annotation.JsonTypeInfo;
5 import com.fasterxml.jackson.annotation.PropertyAccessor;
6 import com.fasterxml.jackson.databind.ObjectMapper;
7 import lombok.RequiredArgsConstructor;
8 import org.springframework.context.annotation.Bean;
9 import org.springframework.context.annotation.Configuration;
```

```
13 import org.springframework.data.redis.repository.configuration.EnableRedisRepositories;
14 import org.springframework.data.redis.serializer.Jackson2JsonRedisSerializer;
15 import org.springframework.data.redis.serializer.StringRedisSerializer;
16 import org.springframework.http.converter.json.Jackson2ObjectMapperBuilder;
17 import org.springframework.security.jackson2.CoreJackson2Module;
18
19 /**
20  * Redis的key序列化配置类
21  *
22  * @author vains
23  */
24 @Configuration
25 @RequiredArgsConstructor
26 @EnableRedisRepositories(enableKeyspaceEvents = RedisKeyValueAdapter.EnableKeyspaceEvents.ON_STARTUP)
27 public class RedisConfig {
28
29     private final Jackson2ObjectMapperBuilder builder;
30
31     /**
32      * 默认情况下使用
33      *
34      * @param connectionFactory redis链接工厂
35      * @return RedisTemplate
36      */
37     @Bean
38     public RedisTemplate<Object, Object> redisTemplate(RedisConnectionFactory connectionFactory)
39         // 字符串序列化器
40         StringRedisSerializer stringRedisSerializer = new StringRedisSerializer();
41
42         // 创建ObjectMapper并添加默认配置
43         ObjectMapper objectMapper = builder.createXmlMapper(false).build();
44
45         // 序列化所有字段
46         objectMapper.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);
47
48         // 此项必须配置，否则如果序列化的对象里边还有对象，会报如下错误：
49         //      java.lang.ClassCastException: java.util.LinkedHashMap cannot be cast to XXX
50         objectMapper.activateDefaultTyping(
51             objectMapper.getPolymorphicTypeValidator(),
52             ObjectMapper.DefaultTyping.NON_FINAL,
53             JsonTypeInfo.As.PROPERTY);
54
55         // 添加Security提供的Jackson Mixin
56         objectMapper.registerModule(new CoreJackson2Module());
57
58         // 存入redis时序列化值的序列化器
```

```
62     RedisTemplate<Object, Object> redisTemplate = new RedisTemplate<>();
63
64     // 设置值序列化
65     redisTemplate.setValueSerializer(valueSerializer);
66     // 设置hash格式数据值的序列化器
67     redisTemplate.setHashValueSerializer(valueSerializer);
68     // 默认的Key序列化器为: JdkSerializationRedisSerializer
69     redisTemplate.setKeySerializer(stringRedisSerializer);
70     // 设置字符串序列化器
71     redisTemplate.setStringSerializer(stringRedisSerializer);
72     // 设置hash结构的key的序列化器
73     redisTemplate.setHashKeySerializer(stringRedisSerializer);
74
75     // 设置连接工厂
76     redisTemplate.setConnectionFactory(connectionFactory);
77
78     return redisTemplate;
79 }
80
81 }
```

## 实现步骤

因为本文使用的是Spring Data，所以需要先定义对应的实体，然后根据实体定义对应的Repository(Spring Data Repository)，最后实现核心的service，使用这些Repository操作Redis。

1. 定义实体
2. 定义 [Redis Repositories](#)
3. 实现核心服务类

## 具体实现

### 定义实体



java 复制代码

```
1 package com.example.entity.security;
2
3 import lombok.Data;
4 import org.springframework.data.annotation.Id;
5 import org.springframework.data.redis.core.RedisHash;
6 import org.springframework.data.redis.core.index.Indexed;
7
8 import java.io.Serializable;
9 import java.time.Instant;
10
11 /**
12  * 基于redis存储的客户端实体
13  *
14  * @author vains
15  */
16 @Data
17 @RedisHash(value = "client")
18 public class RedisRegisteredClient implements Serializable {
19
20     /**
21      * 主键
22      */
23     @Id
24     private String id;
25
26     /**
27      * 客户端id
28      */
29     @Indexed
30     private String clientId;
31
32     /**
33      * 客户端id签发时间
34      */
35     private Instant clientIdIssuedAt;
36
37     /**
38      * 客户端密钥
39      */
```



```
43     * 客户端秘钥过期时间
44     */
45     private Instant clientSecretExpiresAt;
46
47     /**
48     * 客户端名称
49     */
50     private String clientName;
51
52     /**
53     * 客户端支持的认证方式
54     */
55     private String clientAuthenticationMethods;
56
57     /**
58     * 客户端支持的授权申请方式
59     */
60     private String authorizationGrantTypes;
61
62     /**
63     * 回调地址
64     */
65     private String redirectUri;
66
67     /**
68     * 登出回调地址
69     */
70     private String postLogoutRedirectUri;
71
72     /**
73     * 客户端拥有的scope
74     */
75     private String scopes;
76
77     /**
78     * 客户端配置
79     */
80     private String clientSettings;
81
82     /**
83     * 通过该客户端签发的access token设置
84     */
85     private String tokenSettings;
86
87 }
```

该类中包括了授权码、access\_token、refresh\_token、设备码和id\_token等数据。

java 复制代码

```
1 package com.example.entity.security;
2
3 import lombok.Data;
4 import org.springframework.data.annotation.Id;
5 import org.springframework.data.redis.core.RedisHash;
6 import org.springframework.data.redis.core.TimeToLive;
7 import org.springframework.data.redis.core.index.Indexed;
8
9 import java.io.Serializable;
10 import java.time.Instant;
11 import java.util.concurrent.TimeUnit;
12
13 /**
14  * 使用Repository将授权申请的认证信息缓存至redis的实体
15  *
16  * @author vains
17  */
18 @Data
19 @RedisHash(value = "authorization")
20 public class RedisOAuth2Authorization implements Serializable {
21
22     /**
23      * 主键
24      */
25     @Id
26     private String id;
27
28     /**
29      * 授权申请时使用的客户端id
30      */
31     private String registeredClientId;
32
33     /**
34      * 授权用户姓名
35      */
36     private String principalName;
37
38     /**
39      * 授权申请时使用 grant_type
40      */
41     private String authorizationGrantType;
```

```
45     */
46     private String authorizedScopes;
47
48     /**
49      * 授权的认证信息(当前用户)、请求信息(授权申请请求)
50      */
51     private String attributes;
52
53     /**
54      * 授权申请时的state
55      */
56     @Indexed
57     private String state;
58
59     /**
60      * 授权码的值
61      */
62     @Indexed
63     private String authorizationCodeValue;
64
65     /**
66      * 授权码签发时间
67      */
68     private Instant authorizationCodeIssuedAt;
69
70     /**
71      * 授权码过期时间
72      */
73     private Instant authorizationCodeExpiresAt;
74
75     /**
76      * 授权码元数据
77      */
78     private String authorizationCodeMetadata;
79
80     /**
81      * access token的值
82      */
83     @Indexed
84     private String accessTokenValue;
85
86     /**
87      * access token签发时间
88      */
89     private Instant accessTokenIssuedAt;
90
```



```
94     private Instant accessTokenExpiresAt;
95
96     /**
97      * access token元数据
98      */
99     private String accessTokenMetadata;
100
101     /**
102      * access token的类型
103      */
104     private String accessTokenType;
105
106     /**
107      * access token中包含的scope
108      */
109     private String accessTokenScopes;
110
111     /**
112      * refresh token的值
113      */
114     @Indexed
115     private String refreshTokenValue;
116
117     /**
118      * refresh token签发使劲
119      */
120     private Instant refreshTokenIssuedAt;
121
122     /**
123      * refresh token过期时间
124      */
125     private Instant refreshTokenExpiresAt;
126
127     /**
128      * refresh token元数据
129      */
130     private String refreshTokenMetadata;
131
132     /**
133      * id token的值
134      */
135     @Indexed
136     private String oidcIdTokenValue;
137
138     /**
139      * id token签发时间
```



```
143     /**
144      * id token过期时间
145      */
146     private Instant oidcIdTokenExpiresAt;
147
148     /**
149      * id token元数据
150      */
151     private String oidcIdTokenMetadata;
152
153     /**
154      * id token中包含的属性
155      */
156     private String oidcIdTokenClaims;
157
158     /**
159      * 用户码的值
160      */
161     @Indexed
162     private String userCodeValue;
163
164     /**
165      * 用户码签发时间
166      */
167     private Instant userCodeIssuedAt;
168
169     /**
170      * 用户码过期时间
171      */
172     private Instant userCodeExpiresAt;
173
174     /**
175      * 用户码元数据
176      */
177     private String userCodeMetadata;
178
179     /**
180      * 设备码的值
181      */
182     @Indexed
183     private String deviceCodeValue;
184
185     /**
186      * 设备码签发时间
187      */
188     private Instant deviceCodeIssuedAt;
```



```
192     */
193     private Instant deviceCodeExpiresAt;
194
195     /**
196     * 设备码元数据
197     */
198     private String deviceCodeMetadata;
199
200     /**
201     * 当前对象在Redis中的过期时间
202     */
203     @TimeToLive(unit = TimeUnit.MINUTES)
204     private Long timeout;
205
206 }
```



java 复制代码

```
1 package com.example.entity.security;
2
3 import lombok.Data;
4 import org.springframework.data.annotation.Id;
5 import org.springframework.data.redis.core.RedisHash;
6 import org.springframework.data.redis.core.index.Indexed;
7
8 import java.io.Serializable;
9
10 /**
11 * 基于redis的授权确认存储实体
12 *
13 * @author vains
14 */
15 @Data
16 @RedisHash(value = "authorizationConsent")
17 public class RedisAuthorizationConsent implements Serializable {
18
19     /**
20     * 额外提供的主键
21     */
22     @Id
23     private String id;
```

```
27     */
28     @Indexed
29     private String registeredClientId;
30
31     /**
32     * 当前授权确认用户的 username
33     */
34     @Indexed
35     private String principalName;
36
37     /**
38     * 授权确认的scope
39     */
40     private String authorities;
41
42 }
```

### 「注解解释」

1. `@RedisHash` 标注这是一个Spring Data Redis的实体类，同时也指定了该类保存在Redis时的key前缀。
2. `@Id` 指定id属性，id属性也会被当做key的一部分，本注解和上个这两个注解项负责创建用于持久化哈希的实际键。
3. `@Indexed` 注解标注的字段会创建一个基于该字段的索引，让Repository支持 `findBy`被注解标注的字段名 等方法。
4. `@TimeToLive` 注解标注的字段会被用来当做该对象在Redis的过期时间，虽然RedisHash也支持设置过期时间，但是不够灵活，所以额外添加一个字段针对某条数据设置过期时间。

## 定义Spring Data Repositories(Redis Repositories)

Spring Data Repository是Spring Data抽象出来的一个增删改查的interface接口，适用于Spring Data的不同实现，框架提供了支持增删改查的公共Repository：

`CrudRepository<实体类，主键类型>`。



像下边的接口中有一个 `findByClientId` 方法，但是 `ClientId` 属性并不是主键，如果不加 `@Indexed` 注解，则该方法就不会生效。

```
1 package com.example.repository;
2
3 import com.example.entity.security.RedisRegisteredClient;
4 import org.springframework.data.repository.CrudRepository;
5
6 import java.util.Optional;
7
8 /**
9  * 基于Spring Data Redis的客户端repository
10  *
11  * @author vains
12  */
13 public interface RedisClientRepository extends CrudRepository<RedisRegisteredClient, String> {
14
15     /**
16      * 根据clientId查询客户端信息
17      *
18      * @param clientId 客户端id
19      * @return 客户端信息
20      */
21     Optional<RedisRegisteredClient> findByClientId(String clientId);
22
23 }
```



提供根据

`state` , `authorizationCodeValue` , `accessTokenValue` , `refreshTokenValue` , `userCodeValue` 和 `deviceCodeValue` 属性查询的方法, 在`service`中组合使用。



java 复制代码

```
1 package com.example.repository;
2
3 import com.example.entity.security.RedisOAuth2Authorization;
4 import org.springframework.data.repository.CrudRepository;
5
6 import java.util.Optional;
7
8 /**
9  * oauth2授权管理
```



```
13 public interface RedisOAuth2AuthorizationRepository extends CrudRepository<RedisOAuth2Auth
14
15 /**
16  * 根据授权码获取认证信息
17  *
18  * @param token 授权码
19  * @return 认证信息
20  */
21 Optional<RedisOAuth2Authorization> findByAuthorizationCodeValue (String token);
22
23 /**
24  * 根据access token获取认证信息
25  *
26  * @param token access token
27  * @return 认证信息
28  */
29 Optional<RedisOAuth2Authorization> findByAccessTokenValue (String token);
30
31 /**
32  * 根据刷新token获取认证信息
33  *
34  * @param token 刷新token
35  * @return 认证信息
36  */
37 Optional<RedisOAuth2Authorization> findByRefreshTokenValue (String token);
38
39 /**
40  * 根据id token获取认证信息
41  *
42  * @param token id token
43  * @return 认证信息
44  */
45 Optional<RedisOAuth2Authorization> findByOidcIdTokenValue (String token);
46
47 /**
48  * 根据用户码获取认证信息
49  *
50  * @param token 用户码
51  * @return 认证信息
52  */
53 Optional<RedisOAuth2Authorization> findByUserCodeValue (String token);
54
55 /**
56  * 根据设备码获取认证信息
57  *
58  * @param token 设备码
```

```
62
63     /**
64      * 根据state获取认证信息
65      *
66      * @param token 授权申请时的state
67      * @return 认证信息
68      */
69     Optional<RedisOAuth2Authorization> findByState(String token);
70 }
```



提供一个根据客户端Id和授权确认用户的username查询的方法。



java 复制代码

```
1 package com.example.repository;
2
3 import com.example.entity.security.RedisAuthorizationConsent;
4 import org.springframework.data.repository.CrudRepository;
5
6 import java.util.Optional;
7
8 /**
9  * 基于redis的授权确认repository
10  *
11  * @author vains
12  */
13 public interface RedisAuthorizationConsentRepository extends CrudRepository<RedisAuthorizationConsent, String> {
14
15     /**
16      * 根据客户端id和授权确认用户的 username 查询授权确认信息
17      *
18      * @param registeredClientId 客户端id
19      * @param principalName 授权确认用户的 username
20      * @return 授权确认记录
21      */
22     Optional<RedisAuthorizationConsent> findByRegisteredClientIdAndPrincipalName(String registeredClientId, String principalName);
23 }
```



以下内容摘抄自 [文档](#) 内容

「查询方法允许从方法名自动派生简单的查找器查询，请确保在查找器方法中使用的属性已设置为索引。」

下表提供了Redis支持的关键字概述，以及包含该关键字的方法本质上是什么：

Keyword	Sample	Redis snippet
And	<code>findByLastnameAndFirstname</code>	<code>SINTER ...:firstname:rand ...:lastname:al' thor</code>
Or	<code>findByLastnameOrFirstname</code>	<code>SUNION ...:firstname:rand ...:lastname:al' thor</code>
Is, Equals	<code>findByFirstname</code> , <code>findByFirstnameIs</code> , <code>findByFirstnameEquals</code>	<code>SINTER ...:firstname:rand</code>
IsTrue	<code>FindByAliveIsTrue</code>	<code>SINTER ...:alive:1</code>
IsFalse	<code>findByAliveIsFalse</code>	<code>SINTER ...:alive:0</code>
Top, First	<code>findFirst10ByFirstname</code> , <code>findTop5ByFirstname</code>	

## 实现核心service



小tip：我也不知道为什么这个这么特殊是Repository..

▼ java 复制代码

```
1 package com.example.repository;
2
3 import com.example.entity.security.RedisRegisteredClient;
4 import com.example.service.impl.RedisOAuth2AuthorizationService;
```

```
8 import jakarta.annotation.PostConstruct;
9 import lombok.RequiredArgsConstructor;
10 import lombok.extern.slf4j.Slf4j;
11 import org.springframework.security.crypto.password.PasswordEncoder;
12 import org.springframework.security.jackson2.SecurityJackson2Modules;
13 import org.springframework.security.oauth2.core.AuthorizationGrantType;
14 import org.springframework.security.oauth2.core.ClientAuthenticationMethod;
15 import org.springframework.security.oauth2.core.oidc.OidcScopes;
16 import org.springframework.security.oauth2.jose.jws.SignatureAlgorithm;
17 import org.springframework.security.oauth2.server.authorization.client.RegisteredClient;
18 import org.springframework.security.oauth2.server.authorization.client.RegisteredClientRepository;
19 import org.springframework.security.oauth2.server.authorization.jackson2.OAuth2AuthorizationServerJ
20 import org.springframework.security.oauth2.server.authorization.settings.ClientSettings;
21 import org.springframework.security.oauth2.server.authorization.settings.OAuth2TokenFormat;
22 import org.springframework.security.oauth2.server.authorization.settings.TokenSettings;
23 import org.springframework.stereotype.Repository;
24 import org.springframework.util.Assert;
25 import org.springframework.util.StringUtils;
26
27 import java.time.Duration;
28 import java.util.ArrayList;
29 import java.util.List;
30 import java.util.Map;
31 import java.util.Set;
32 import java.util.UUID;
33
34 /**
35  * 基于redis的客户端repository实现
36  *
37  * @author vains
38  */
39 @Slf4j
40 @Repository
41 @RequiredArgsConstructor
42 public class RedisRegisteredClientRepository implements RegisteredClientRepository {
43
44     /**
45      * 提供给客户端初始化使用(不需要可删除)
46      */
47     private final PasswordEncoder passwordEncoder;
48
49     private final RedisClientRepository repository;
50
51     private final static ObjectMapper MAPPER = new ObjectMapper();
52
53     static {
```



```
57     List<Module> modules = SecurityJackson2Modules.getModules(classLoader);
58     MAPPER.registerModules(modules);
59     // 加载Authorization Server提供的Module
60     MAPPER.registerModule(new OAuth2AuthorizationServerJackson2Module());
61 }
62
63 @Override
64 public void save(RegisteredClient registeredClient) {
65     Assert.notNull(registeredClient, "registeredClient cannot be null");
66     this.repository.findById(registeredClient.getClientId())
67         .ifPresent(existingRegisteredClient -> this.repository.deleteById(existingRegisteredClient.getId()));
68     this.repository.save(toEntity(registeredClient));
69 }
70
71 @Override
72 public RegisteredClient findById(String id) {
73     Assert.hasText(id, "id cannot be empty");
74     return this.repository.findById(id)
75         .map(this::toObject).orElse(null);
76 }
77
78 @Override
79 public RegisteredClient findById(String clientId) {
80     Assert.hasText(clientId, "clientId cannot be empty");
81     return this.repository.findById(clientId)
82         .map(this::toObject).orElse(null);
83 }
84
85 private RegisteredClient toObject(RedisRegisteredClient client) {
86     Set<String> clientAuthenticationMethods = StringUtils.commaDelimitedListToSet(
87         client.getClientAuthenticationMethods());
88     Set<String> authorizationGrantTypes = StringUtils.commaDelimitedListToSet(
89         client.getAuthorizationGrantTypes());
90     Set<String> redirectUris = StringUtils.commaDelimitedListToSet(
91         client.getRedirectUris());
92     Set<String> postLogoutRedirectUris = StringUtils.commaDelimitedListToSet(
93         client.getPostLogoutRedirectUris());
94     Set<String> clientScopes = StringUtils.commaDelimitedListToSet(
95         client.getScopes());
96
97     RegisteredClient.Builder builder = RegisteredClient.withId(client.getId())
98         .clientId(client.getClientId())
99         .clientIdIssuedAt(client.getClientIdIssuedAt())
100        .clientSecret(client.getClientSecret())
101        .clientSecretExpiresAt(client.getClientSecretExpiresAt())
102        .clientName(client.getClientName())
```

```
106         .authorizationGrantTypes((grantTypes) ->
107             authorizationGrantTypes.forEach(grantType ->
108                 grantTypes.add(resolveAuthorizationGrantType(grantType))))
109         .redirectUris((uris) -> uris.addAll(redirectUris))
110         .postLogoutRedirectUris((uris) -> uris.addAll(postLogoutRedirectUris))
111         .scopes((scopes) -> scopes.addAll(clientScopes));
112
113     Map<String, Object> clientSettingsMap = parseMap(client.getClientSettings());
114     builder.clientSettings(ClientSettings.withSettings(clientSettingsMap).build());
115
116     Map<String, Object> tokenSettingsMap = parseMap(client.getTokenSettings());
117     builder.tokenSettings(TokenSettings.withSettings(tokenSettingsMap).build());
118
119     return builder.build();
120 }
121
122 private RedisRegisteredClient toEntity(RegisteredClient registeredClient) {
123     List<String> clientAuthenticationMethods = new ArrayList<>(registeredClient.getClientAuthen
124     registeredClient.getClientAuthenticationMethods().forEach(clientAuthenticationMethod ->
125         clientAuthenticationMethods.add(clientAuthenticationMethod.getValue()));
126
127     List<String> authorizationGrantTypes = new ArrayList<>(registeredClient.getAuthorizationGr
128     registeredClient.getAuthorizationGrantTypes().forEach(authorizationGrantType ->
129         authorizationGrantTypes.add(authorizationGrantType.getValue()));
130
131     RedisRegisteredClient entity = new RedisRegisteredClient();
132     entity.setId(registeredClient.getId());
133     entity.setClientId(registeredClient.getClientId());
134     entity.setClientIdIssuedAt(registeredClient.getClientIdIssuedAt());
135     entity.setClientSecret(registeredClient.getClientSecret());
136     entity.setClientSecretExpiresAt(registeredClient.getClientSecretExpiresAt());
137     entity.setClientName(registeredClient.getClientName());
138     entity.setClientAuthenticationMethods(StringUtils.collectionToCommaDelimitedString(clientAut
139     entity.setAuthorizationGrantTypes(StringUtils.collectionToCommaDelimitedString(authorization
140     entity.setRedirectUris(StringUtils.collectionToCommaDelimitedString(registeredClient.getRedi
141     entity.setPostLogoutRedirectUris(StringUtils.collectionToCommaDelimitedString(registeredClie
142     entity.setScopes(StringUtils.collectionToCommaDelimitedString(registeredClient.getScopes()))
143     entity.setClientSettings(writeMap(registeredClient.getClientSettings().getSettings()));
144     entity.setTokenSettings(writeMap(registeredClient.getTokenSettings().getSettings()));
145
146     return entity;
147 }
148
149 private Map<String, Object> parseMap(String data) {
150     try {
151         return MAPPER.readValue(data, new TypeReference<>() {
```

```
155     }
156 }
157
158 private String writeMap(Map<String, Object> data) {
159     try {
160         return MAPPER.writeValueAsString(data);
161     } catch (Exception ex) {
162         throw new IllegalArgumentException(ex.getMessage(), ex);
163     }
164 }
165
166 private static AuthorizationGrantType resolveAuthorizationGrantType(String authorizationG
167     if (AuthorizationGrantType.AUTHORIZATION_CODE.getValue().equals(authorizationGrantType)) {
168         return AuthorizationGrantType.AUTHORIZATION_CODE;
169     } else if (AuthorizationGrantType.CLIENT_CREDENTIALS.getValue().equals(authorizationGrantTy
170         return AuthorizationGrantType.CLIENT_CREDENTIALS;
171     } else if (AuthorizationGrantType.REFRESH_TOKEN.getValue().equals(authorizationGrantType))
172         return AuthorizationGrantType.REFRESH_TOKEN;
173     }
174     // Custom authorization grant type
175     return new AuthorizationGrantType(authorizationGrantType);
176 }
177
178 private static ClientAuthenticationMethod resolveClientAuthenticationMethod(String client
179     if (ClientAuthenticationMethod.CLIENT_SECRET_BASIC.getValue().equals(clientAuthenticationMet
180         return ClientAuthenticationMethod.CLIENT_SECRET_BASIC;
181     } else if (ClientAuthenticationMethod.CLIENT_SECRET_POST.getValue().equals(clientAuthentica
182         return ClientAuthenticationMethod.CLIENT_SECRET_POST;
183     } else if (ClientAuthenticationMethod.NONE.getValue().equals(clientAuthenticationMethod)) {
184         return ClientAuthenticationMethod.NONE;
185     }
186     // Custom client authentication method
187     return new ClientAuthenticationMethod(clientAuthenticationMethod);
188 }
189
190 /**
191  * 容器启动后初始化客户端
192  * (不需要可删除)
193  */
194 @PostConstruct
195 public void initClients() {
196     log.info("Initialize client information to Redis.");
197     // 默认需要授权确认
198     ClientSettings.Builder builder = ClientSettings.builder()
199         .requireAuthorizationConsent(Boolean.TRUE);
200 }
```

```
204 // Access Token 存活时间: 2小时
205 .accessTokenTimeToLive(Duration.ofHours(2L))
206 // 授权码存活时间: 5分钟
207 .authorizationCodeTimeToLive(Duration.ofMinutes(5L))
208 // 设备码存活时间: 5分钟
209 .deviceCodeTimeToLive(Duration.ofMinutes(5L))
210 // Refresh Token 存活时间: 7天
211 .refreshTokenTimeToLive(Duration.ofDays(7L))
212 // 刷新 Access Token 后是否重用 Refresh Token
213 .reuseRefreshTokens(Boolean.TRUE)
214 // 设置 Id Token 加密方式
215 .idTokenSignatureAlgorithm(SignatureAlgorithm.RS256)
216 .build();
217
218 // 正常授权码客户端
219 RegisteredClient registeredClient = RegisteredClient.withId(UUID.randomUUID().toString())
220     .clientId("messaging-client")
221     .clientName("授权码")
222     .clientSecret(passwordEncoder.encode("123456"))
223     // 客户端认证方式, 基于请求头的认证
224     .clientAuthenticationMethod(ClientAuthenticationMethod.CLIENT_SECRET_BASIC)
225     // 配置资源服务器使用该客户端获取授权时支持的方式
226     .authorizationGrantType(AuthorizationGrantType.AUTHORIZATION_CODE)
227     .authorizationGrantType(AuthorizationGrantType.REFRESH_TOKEN)
228     .authorizationGrantType(AuthorizationGrantType.CLIENT_CREDENTIALS)
229     // 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,
230     .redirectUri("http://127.0.0.1:8000/login/oauth2/code/messaging-client-oidc")
231     .redirectUri("https://www.baidu.com")
232     // 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
233     .scope(OidcScopes.OPENID)
234     .scope(OidcScopes.PROFILE)
235     // 指定scope
236     .scope("message.read")
237     .scope("message.write")
238     // 客户端设置, 设置用户需要确认授权
239     .clientSettings(builder.build())
240     // token相关配置
241     .tokenSettings(tokenSettings)
242     .build();
243
244 // 设备码授权客户端
245 RegisteredClient deviceClient = RegisteredClient.withId(UUID.randomUUID().toString())
246     .clientId("device-message-client")
```

```
253         // 设备码授权
254         .authorizationGrantType(AuthorizationGrantType.DEVICE_CODE)
255         .authorizationGrantType(AuthorizationGrantType.REFRESH_TOKEN)
256         // 指定scope
257         .scope("message.read")
258         .scope("message.write")
259         // token相关配置
260         .tokenSettings(tokenSettings)
261         .build();
262
263     // PKCE客户端
264     RegisteredClient pkceClient = RegisteredClient.withId(UUID.randomUUID().toString())
265         .clientId("pkce-message-client")
266         .clientName("PKCE流程")
267         // 公共客户端
268         .clientAuthenticationMethod(ClientAuthenticationMethod.NONE)
269         // 设备码授权
270         .authorizationGrantType(AuthorizationGrantType.AUTHORIZATION_CODE)
271         .authorizationGrantType(AuthorizationGrantType.REFRESH_TOKEN)
272         // 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,
273         .redirectUri("http://127.0.0.1:8000/login/oauth2/code/messaging-client-oidc")
274         // 开启 PKCE 流程
275         .clientSettings(builder.requireProofKey(Boolean.TRUE).build())
276         // 指定scope
277         .scope("message.read")
278         .scope("message.write")
279         // token相关配置
280         .tokenSettings(tokenSettings)
281         .build();
282
283     // 初始化客户端
284     this.save(registeredClient);
285     this.save(deviceClient);
286     this.save(pkceClient);
287 }
288
289 }
```

「类中初始化客户端信息的操作针对第一次使用启动的项目, 同时每次启动也是更新客户端的操作, 如果不需要读者可自行去除。」



```
1 package com.example.service.impl;
2
3 import com.example.entity.security.RedisOAuth2Authorization;
4 import com.example.repository.RedisOAuth2AuthorizationRepository;
5 import com.fasterxml.jackson.core.type.TypeReference;
6 import com.fasterxml.jackson.databind.Module;
7 import com.fasterxml.jackson.databind.ObjectMapper;
8 import lombok.RequiredArgsConstructor;
9 import org.springframework.dao.DataRetrievalFailureException;
10 import org.springframework.security.jackson2.SecurityJackson2Modules;
11 import org.springframework.security.oauth2.core.*;
12 import org.springframework.security.oauth2.core.endpoint.OAuth2ParameterNames;
13 import org.springframework.security.oauth2.core.oidc.OidcIdToken;
14 import org.springframework.security.oauth2.core.oidc.endpoint.OidcParameterNames;
15 import org.springframework.security.oauth2.server.authorization.OAuth2Authorization;
16 import org.springframework.security.oauth2.server.authorization.OAuth2AuthorizationCode;
17 import org.springframework.security.oauth2.server.authorization.OAuth2AuthorizationService;
18 import org.springframework.security.oauth2.server.authorization.OAuth2TokenType;
19 import org.springframework.security.oauth2.server.authorization.client.RegisteredClient;
20 import org.springframework.security.oauth2.server.authorization.client.RegisteredClientRepository;
21 import org.springframework.security.oauth2.server.authorization.jackson2.OAuth2AuthorizationServerJ
22 import org.springframework.stereotype.Service;
23 import org.springframework.util.Assert;
24 import org.springframework.util.StringUtils;
25
26 import java.time.Duration;
27 import java.time.Instant;
28 import java.util.*;
29 import java.util.function.Consumer;
30
31 /**
32  * 基于redis的授权管理服务
33  *
34  * @author vains
35  */
36 @Service
37 @RequiredArgsConstructor
38 public class RedisOAuth2AuthorizationService implements OAuth2AuthorizationService {
39
40     private final RegisteredClientRepository registeredClientRepository;
41
42     private final RedisOAuth2AuthorizationRepository oAuth2AuthorizationRepository;
43
44     private final static ObjectMapper MAPPER = new ObjectMapper();
45
46     static {
```

```
50     List<Module> modules = SecurityJackson2Modules.getModules(classLoader);
51     MAPPER.registerModules(modules);
52     // 加载Authorization Server提供的Module
53     MAPPER.registerModule(new OAuth2AuthorizationServerJackson2Module());
54 }
55
56 @Override
57 public void save(OAuth2Authorization authorization) {
58     Optional<RedisOAuth2Authorization> existingAuthorization = oAuth2AuthorizationRepository.find
59
60     // 如果已存在则删除后再保存
61     existingAuthorization.map(RedisOAuth2Authorization::getId)
62         .ifPresent(oAuth2AuthorizationRepository::deleteById);
63
64     // 过期时间，默认永不过期
65     long maxTimeout = -1L;
66     // 所有code的过期时间，方便计算最大值
67     List<Instant> expiresAtList = new ArrayList<>();
68
69     RedisOAuth2Authorization entity = toEntity(authorization);
70
71     // 如果有过期时间就存入
72     Optional.ofNullable(entity.getAuthorizationCodeExpiresAt())
73         .ifPresent(expiresAtList::add);
74
75     // 如果有过期时间就存入
76     Optional.ofNullable(entity.getAccessTokenExpiresAt())
77         .ifPresent(expiresAtList::add);
78
79     // 如果有过期时间就存入
80     Optional.ofNullable(entity.getRefreshTokenExpiresAt())
81         .ifPresent(expiresAtList::add);
82
83     // 如果有过期时间就存入
84     Optional.ofNullable(entity.getOidcIdTokenExpiresAt())
85         .ifPresent(expiresAtList::add);
86
87     // 如果有过期时间就存入
88     Optional.ofNullable(entity.getUserCodeExpiresAt())
89         .ifPresent(expiresAtList::add);
90
91     // 如果有过期时间就存入
92     Optional.ofNullable(entity.getDeviceCodeExpiresAt())
93         .ifPresent(expiresAtList::add);
94
95     // 获取最大的日期
```

```
99         Duration between = Duration.between(Instant.now(), maxInstant.get());
100         // 转为分钟
101         maxTimeout = between.toMinutes();
102     }
103
104     // 设置过期时间
105     entity.setTimeout(maxTimeout);
106
107     // 保存至redis
108     oAuth2AuthorizationRepository.save(entity);
109 }
110
111 @Override
112 public void remove(OAuth2Authorization authorization) {
113     Assert.notNull(authorization, "authorization cannot be null");
114     oAuth2AuthorizationRepository.deleteById(authorization.getId());
115 }
116
117 @Override
118 public OAuth2Authorization findById(String id) {
119     Assert.hasText(id, "id cannot be empty");
120     return oAuth2AuthorizationRepository.findById(id)
121         .map(this::toObject).orElse(null);
122 }
123
124 @Override
125 public OAuth2Authorization findByToken(String token, OAuth2TokenType tokenType) {
126     Assert.hasText(token, "token cannot be empty");
127
128     Optional<RedisOAuth2Authorization> result;
129
130     if (tokenType == null) {
131         result = oAuth2AuthorizationRepository.findByState(token)
132             .or(() -> oAuth2AuthorizationRepository.findByAuthorizationCodeValue(token))
133             .or(() -> oAuth2AuthorizationRepository.findByAccessTokenValue(token))
134             .or(() -> oAuth2AuthorizationRepository.findByOidcIdTokenValue(token))
135             .or(() -> oAuth2AuthorizationRepository.findByRefreshTokenValue(token))
136             .or(() -> oAuth2AuthorizationRepository.findByUserCodeValue(token))
137             .or(() -> oAuth2AuthorizationRepository.findByDeviceCodeValue(token));
138     } else if (OAuth2ParameterNames.STATE.equals(tokenType.getValue())) {
139         result = oAuth2AuthorizationRepository.findByState(token);
140     } else if (OAuth2ParameterNames.CODE.equals(tokenType.getValue())) {
141         result = oAuth2AuthorizationRepository.findByAuthorizationCodeValue(token);
142     } else if (OAuth2TokenType.ACCESS_TOKEN.equals(tokenType)) {
143         result = oAuth2AuthorizationRepository.findByAccessTokenValue(token);
144     } else if (OidcParameterNames.ID_TOKEN.equals(tokenType.getValue())) {
```



```
148         } else if (OAuth2ParameterNames.USER_CODE.equals(tokenType.getValue())) {
149             result = oAuth2AuthorizationRepository.findByUserCodeValue(token);
150         } else if (OAuth2ParameterNames.DEVICE_CODE.equals(tokenType.getValue())) {
151             result = oAuth2AuthorizationRepository.findByDeviceCodeValue(token);
152         } else {
153             result = Optional.empty();
154         }
155
156         return result.map(this::toObject).orElse(null);
157     }
158
159     /**
160      * 将redis中存储的类型转为框架所需的类型
161      *
162      * @param entity redis中存储的类型
163      * @return 框架所需的类型
164      */
165     private OAuth2Authorization toObject(RedisOAuth2Authorization entity) {
166         RegisteredClient registeredClient = this.registeredClientRepository.findById(entity.getRegisteredClientId());
167         if (registeredClient == null) {
168             throw new DataRetrievalFailureException(
169                 "The RegisteredClient with id '" + entity.getRegisteredClientId() + "' was not found"
170             );
171         }
172
173         OAuth2Authorization.Builder builder = OAuth2Authorization.withRegisteredClient(registeredClient)
174             .id(entity.getId())
175             .principalName(entity.getPrincipalName())
176             .authorizationGrantType(resolveAuthorizationGrantType(entity.getAuthorizationGrantType()))
177             .authorizedScopes(StringUtils.commaDelimitedListToSet(entity.getAuthorizedScopes()))
178             .attributes(attributes -> attributes.putAll(parseMap(entity.getAttributes())));
179
180         if (entity.getState() != null) {
181             builder.attribute(OAuth2ParameterNames.STATE, entity.getState());
182         }
183
184         if (entity.getAuthorizationCodeValue() != null) {
185             OAuth2AuthorizationCode authorizationCode = new OAuth2AuthorizationCode(
186                 entity.getAuthorizationCodeValue(),
187                 entity.getAuthorizationCodeIssuedAt(),
188                 entity.getAuthorizationCodeExpiresAt());
189             builder.token(authorizationCode, metadata -> metadata.putAll(parseMap(entity.getAuthorizationCodeMetadata())));
190         }
191
192         if (entity.getAccessTokenValue() != null) {
193             OAuth2AccessToken accessToken = new OAuth2AccessToken(
194                 OAuth2AccessToken.TokenType.BEARER,
195                 entity.getAccessTokenValue(),
```

```
197         builder.token(accessToken, metadata -> metadata.putAll(parseMap(entity.getAccessTokenMetad
198     })
199
200     if (entity.getRefreshTokenValue() != null) {
201         OAuth2RefreshToken refreshToken = new OAuth2RefreshToken(
202             entity.getRefreshTokenValue(),
203             entity.getRefreshTokenIssuedAt(),
204             entity.getRefreshTokenExpiresAt());
205         builder.token(refreshToken, metadata -> metadata.putAll(parseMap(entity.getRefreshTokenM
206     })
207
208     if (entity.getOidcIdTokenValue() != null) {
209         OidcIdToken idToken = new OidcIdToken(
210             entity.getOidcIdTokenValue(),
211             entity.getOidcIdTokenIssuedAt(),
212             entity.getOidcIdTokenExpiresAt(),
213             parseMap(entity.getOidcIdTokenClaims()));
214         builder.token(idToken, metadata -> metadata.putAll(parseMap(entity.getOidcIdTokenMetadat
215     })
216
217     if (entity.getUserCodeValue() != null) {
218         OAuth2UserCode userCode = new OAuth2UserCode(
219             entity.getUserCodeValue(),
220             entity.getUserCodeIssuedAt(),
221             entity.getUserCodeExpiresAt());
222         builder.token(userCode, metadata -> metadata.putAll(parseMap(entity.getUserCodeMetadatat
223     })
224
225     if (entity.getDeviceCodeValue() != null) {
226         OAuth2DeviceCode deviceCode = new OAuth2DeviceCode(
227             entity.getDeviceCodeValue(),
228             entity.getDeviceCodeIssuedAt(),
229             entity.getDeviceCodeExpiresAt());
230         builder.token(deviceCode, metadata -> metadata.putAll(parseMap(entity.getDeviceCodeMetad
231     })
232
233     return builder.build();
234 }
235
236 /**
237  * 将框架所需的类型转为redis中存储的类型
238  *
239  * @param authorization 框架所需的类型
240  * @return redis中存储的类型
241  */
242 private RedisOAuth2Authorization toEntity(OAuth2Authorization authorization) {
```

```
246     entity.setPrincipalName(authorization.getPrincipalName());
247     entity.setAuthorizationGrantType(authorization.getAuthorizationGrantType().getValue());
248     entity.setAuthorizedScopes(StringUtils.collectionToDelimitedString(authorization.getAuthoriz
249     entity.setAttributes(writeMap(authorization.getAttributes()));
250     entity.setState(authorization.getAttribute(OAuth2ParameterNames.STATE));
251
252     OAuth2Authorization.Token<OAuth2AuthorizationCode> authorizationCode =
253         authorization.getToken(OAuth2AuthorizationCode.class);
254     setTokenValues(
255         authorizationCode,
256         entity::setAuthorizationCodeValue,
257         entity::setAuthorizationCodeIssuedAt,
258         entity::setAuthorizationCodeExpiresAt,
259         entity::setAuthorizationCodeMetadata
260     );
261
262     OAuth2Authorization.Token<OAuth2AccessToken> accessToken =
263         authorization.getToken(OAuth2AccessToken.class);
264     setTokenValues(
265         accessToken,
266         entity::setAccessTokenValue,
267         entity::setAccessTokenIssuedAt,
268         entity::setAccessTokenExpiresAt,
269         entity::setAccessTokenMetadata
270     );
271     if (accessToken != null && accessToken.getToken().getScopes() != null) {
272         entity.setAccessTokenScopes(StringUtils.collectionToDelimitedString(accessToken.getToken
273     }
274
275     OAuth2Authorization.Token<OAuth2RefreshToken> refreshToken =
276         authorization.getToken(OAuth2RefreshToken.class);
277     setTokenValues(
278         refreshToken,
279         entity::setRefreshTokenValue,
280         entity::setRefreshTokenIssuedAt,
281         entity::setRefreshTokenExpiresAt,
282         entity::setRefreshTokenMetadata
283     );
284
285     OAuth2Authorization.Token<OidcIdToken> oidcIdToken =
286         authorization.getToken(OidcIdToken.class);
287     setTokenValues(
288         oidcIdToken,
289         entity::setOidcIdTokenValue,
290         entity::setOidcIdTokenIssuedAt,
291         entity::setOidcIdTokenExpiresAt,
```

```
295         entity.setOidcIdTokenClaims(writeMap(oidcIdToken.getClaims()));
296     }
297
298     OAuth2Authorization.Token<OAuth2UserCode> userCode =
299         authorization.getToken(OAuth2UserCode.class);
300     setTokenValues(
301         userCode,
302         entity::setUserCodeValue,
303         entity::setUserCodeIssuedAt,
304         entity::setUserCodeExpiresAt,
305         entity::setUserCodeMetadata
306     );
307
308     OAuth2Authorization.Token<OAuth2DeviceCode> deviceCode =
309         authorization.getToken(OAuth2DeviceCode.class);
310     setTokenValues(
311         deviceCode,
312         entity::setDeviceCodeValue,
313         entity::setDeviceCodeIssuedAt,
314         entity::setDeviceCodeExpiresAt,
315         entity::setDeviceCodeMetadata
316     );
317
318     return entity;
319 }
320
321 /**
322  * 设置token的值
323  *
324  * @param token          Token实例
325  * @param tokenValueConsumer set方法
326  * @param issuedAtConsumer  set方法
327  * @param expiresAtConsumer set方法
328  * @param metadataConsumer set方法
329  */
330 private void setTokenValues(
331     OAuth2Authorization.Token<?> token,
332     Consumer<String> tokenValueConsumer,
333     Consumer<Instant> issuedAtConsumer,
334     Consumer<Instant> expiresAtConsumer,
335     Consumer<String> metadataConsumer) {
336     if (token != null) {
337         OAuth2Token oAuth2Token = token.getToken();
338         tokenValueConsumer.accept(oAuth2Token.getTokenValue());
339         issuedAtConsumer.accept(oAuth2Token.getIssuedAt());
340         expiresAtConsumer.accept(oAuth2Token.getExpiresAt());
```

```
344
345  /**
346   * 处理授权申请时的 GrantType
347   *
348   * @param authorizationGrantType 授权申请时的 GrantType
349   * @return AuthorizationGrantType的实例
350   */
351  private static AuthorizationGrantType resolveAuthorizationGrantType(String authorizationG
352      if (AuthorizationGrantType.AUTHORIZATION_CODE.getValue().equals(authorizationGrantType)) {
353      return AuthorizationGrantType.AUTHORIZATION_CODE;
354      } else if (AuthorizationGrantType.CLIENT_CREDENTIALS.getValue().equals(authorizationGrantTy
355      return AuthorizationGrantType.CLIENT_CREDENTIALS;
356      } else if (AuthorizationGrantType.REFRESH_TOKEN.getValue().equals(authorizationGrantType))
357      return AuthorizationGrantType.REFRESH_TOKEN;
358      } else if (AuthorizationGrantType.DEVICE_CODE.getValue().equals(authorizationGrantType)) {
359      return AuthorizationGrantType.DEVICE_CODE;
360      }
361      // Custom authorization grant type
362      return new AuthorizationGrantType(authorizationGrantType);
363  }
364
365  /**
366   * 将json转为map
367   *
368   * @param data json
369   * @return map对象
370   */
371  private Map<String, Object> parseMap(String data) {
372      try {
373          return MAPPER.readValue(data, new TypeReference<>() {
374          });
375      } catch (Exception ex) {
376          throw new IllegalArgumentException(ex.getMessage(), ex);
377      }
378  }
379
380  /**
381   * 将map对象转为json字符串
382   *
383   * @param metadata map对象
384   * @return json字符串
385   */
386  private String writeMap(Map<String, Object> metadata) {
387      try {
388          return MAPPER.writeValueAsString(metadata);
389      } catch (Exception ex) {
```



```
393
394 }
395
```



java 复制代码

```
1 package com.example.service.impl;
2
3 import com.example.entity.security.RedisAuthorizationConsent;
4 import com.example.repository.RedisAuthorizationConsentRepository;
5 import lombok.RequiredArgsConstructor;
6 import org.springframework.dao.DataRetrievalFailureException;
7 import org.springframework.security.core.GrantedAuthority;
8 import org.springframework.security.core.authority.SimpleGrantedAuthority;
9 import org.springframework.security.oauth2.server.authorization.OAuth2AuthorizationConsent;
10 import org.springframework.security.oauth2.server.authorization.OAuth2AuthorizationConsentService;
11 import org.springframework.security.oauth2.server.authorization.client.RegisteredClient;
12 import org.springframework.security.oauth2.server.authorization.client.RegisteredClientRepository;
13 import org.springframework.stereotype.Service;
14 import org.springframework.util.Assert;
15 import org.springframework.util.StringUtils;
16
17 import java.util.HashSet;
18 import java.util.Set;
19 import java.util.UUID;
20
21 /**
22  * 基于redis的授权确认服务实现
23  *
24  * @author vains
25  */
26 @Service
27 @RequiredArgsConstructor
28 public class RedisOAuth2AuthorizationConsentService implements OAuth2AuthorizationConsentService {
29
30     private final RegisteredClientRepository registeredClientRepository;
31
32     private final RedisAuthorizationConsentRepository authorizationConsentRepository;
33
34     @Override
```

```
38     // 如果存在就先删除
39     this.authorizationConsentRepository.findByRegisteredClientIdAndPrincipalName(
40         authorizationConsent.getRegisteredClientId(), authorizationConsent.getPrincipalName()
41     ).ifPresent(existingConsent -> this.authorizationConsentRepository.deleteById(existingConsent.getId()));
42
43     // 保存
44     RedisAuthorizationConsent entity = toEntity(authorizationConsent);
45     entity.setId(UUID.randomUUID().toString());
46     this.authorizationConsentRepository.save(entity);
47 }
48
49 @Override
50 public void remove(OAuth2AuthorizationConsent authorizationConsent) {
51     Assert.notNull(authorizationConsent, "authorizationConsent cannot be null");
52     // 如果存在就先删除
53     this.authorizationConsentRepository.findByRegisteredClientIdAndPrincipalName(
54         authorizationConsent.getRegisteredClientId(), authorizationConsent.getPrincipalName()
55     ).ifPresent(existingConsent -> this.authorizationConsentRepository.deleteById(existingConsent.getId()));
56 }
57
58 @Override
59 public OAuth2AuthorizationConsent findById(String registeredClientId, String principalName) {
60     Assert.hasText(registeredClientId, "registeredClientId cannot be empty");
61     Assert.hasText(principalName, "principalName cannot be empty");
62     return this.authorizationConsentRepository.findByRegisteredClientIdAndPrincipalName(
63         registeredClientId, principalName).map(this::toObject).orElse(null);
64 }
65
66 private OAuth2AuthorizationConsent toObject(RedisAuthorizationConsent authorizationConsent) {
67     String registeredClientId = authorizationConsent.getRegisteredClientId();
68     RegisteredClient registeredClient = this.registeredClientRepository.findById(registeredClientId)
69     if (registeredClient == null) {
70         throw new DataRetrievalFailureException(
71             "The RegisteredClient with id '" + registeredClientId + "' was not found in the database"
72         );
73     }
74     OAuth2AuthorizationConsent.Builder builder = OAuth2AuthorizationConsent.withId(
75         registeredClientId, authorizationConsent.getPrincipalName());
76     if (authorizationConsent.getAuthorities() != null) {
77         for (String authority : StringUtils.commaDelimitedListToSet(authorizationConsent.getAuthorities())) {
78             builder.authority(new SimpleGrantedAuthority(authority));
79         }
80     }
81
82     return builder.build();
83 }
```

```
87     entity.setRegisteredClientId(authorizationConsent.getRegisteredClientId());
88     entity.setPrincipalName(authorizationConsent.getPrincipalName());
89
90     Set<String> authorities = new HashSet<>();
91     for (GrantedAuthority authority : authorizationConsent.getAuthorities()) {
92         authorities.add(authority.getAuthority());
93     }
94     entity.setAuthorities(StringUtils.collectionToCommaDelimitedString(authorities));
95
96     return entity;
97 }
98
99 }
```

## 去除认证服务配置文件中这三个核心service的注入

▼ java 复制代码

```
1  /**
2   * 配置客户端Repository
3   *
4   * @param jdbcTemplate db 数据源信息
5   * @return 基于数据库的repository
6   */
7  @Bean
8  public RegisteredClientRepository registeredClientRepository(JdbcTemplate jdbcTemplate) {
9      // 基于db存储客户端，还有一个基于内存的实现 InMemoryRegisteredClientRepository
10     return new JdbcRegisteredClientRepository(jdbcTemplate);
11 }
12
13 /**
14 * 配置基于db的oauth2的授权管理服务
15 *
16 * @param jdbcTemplate db数据源信息
17 * @param registeredClientRepository 上边注入的客户端repository
18 * @return JdbcOAuth2AuthorizationService
19 */
20 @Bean
21 public OAuth2AuthorizationService authorizationService(JdbcTemplate jdbcTemplate, RegisteredClientRepository registeredClientRepository) {
22     // 基于db的oauth2认证服务，还有一个基于内存的服务实现InMemoryOAuth2AuthorizationService
23     return new JdbcOAuth2AuthorizationService(jdbcTemplate, registeredClientRepository);
24 }
```



```
27 * 配置基于db的授权确认管理服务
28 *
29 * @param jdbcTemplate          db数据源信息
30 * @param registeredClientRepository 客户端repository
31 * @return JdbcOAuth2AuthorizationConsentService
32 */
33 @Bean
34 public OAuth2AuthorizationConsentService authorizationConsentService(JdbcTemplate jdbcTemplate,
35     // 基于db的授权确认管理服务，还有一个基于内存的服务实现InMemoryOAuth2AuthorizationConsentService
36     return new JdbcOAuth2AuthorizationConsentService(jdbcTemplate, registeredClientRepository
37 }
```

## 写在最后

到此为止基本就结束了，本文章和前边的所有系列文章没有必要的关联，如果是第一次看到文章的读者也是可以很顺畅的将文章中的内容引入项目，当然，因为引用了Spring Data Redis，所以项目必须要先有Redis支持。

文章看起来很长，但是实际上就是定义三个实体类，定义三个Repository，然后实现核心的service；逻辑并不复杂，操作Redis的内容因为使用了Spring Data Repositories，所以这两部分内容很少，内容多得地方就在每个service中实体与默认实体的转换中，一大堆的转换内容导致文章看起来内容很多，但是这些内容在文档中都已经实现，所以说这部分内容直接Copy就行，哈哈。

## 附录

1. [How-to: Implement core services with JPA](#)
2. [Spring Data Redis](#)
3. [Redis Repositories](#)
4. [@TimeToLive](#)
5. [@Indexed](#)
6. 代码仓库: [Gitee](#)、[Github](#)

本文收录于以下专栏

1 / 2



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

专栏目录

订阅

上一篇 Spring Authorization Server...

下一篇 Spring Authorization Server...

评论 19



平等表达，友善交流



0 / 1000 ? 发送

最热 最新



Unravel

您好，感谢分享，想问一下，token中心化存储后还使用jwt的意义在哪里，是否可以换成Opaque Token。

1月前 点赞 3



叹雪飞花 作者：可以换Opaque Token，现在使用jwt在解析令牌时无需从库中查询，但是使用Opaque Token就需要从库里查询了

1月前 点赞 回复



Unravel 回复 叹雪飞花 作者：嗯嗯，确实。看网上讲token的大多也都在说jwt，确实很方便，在面对大多数场景都够用了，但是在一些管理端感觉不适合。

1月前 点赞 回复

查看全部 3 条回复



用户9367754597345

3月前

 点赞

 5

...

 叹雪飞花

作者

:

注入ioc中就行了，原先的也是通过注入ioc的方式添加配置



3月前

 点赞

 回复

...

 用户9367754...

回复


叹雪飞花

作者

:

在哪里注入了，没找到😂


3月前

 点赞

 回复

...

查看全部 5 条回复 ▾



用户7728733569040

我想问一下：postLogoutRedirectUri这个属性要怎么用呢，或者是怎么配置登出，可以详细说一下嘛

4月前

 点赞

 5

...

 叹雪飞花

作者

:


这个应该是在调用OIDC登出端点(/connect/logout)后重定向的地址

4月前

 点赞

 回复

...

 用户7728733...

回复

叹雪飞花

作者

:

现在有一个问题，我在客户端A登出后，如果客户端A不重启，就不用登录，如果客户端A重启了，那就需要登录。这是为什么呢

4月前

 点赞

 回复

...

查看全部 5 条回复 ▾

查看全部 19 条评论 ▾


具体实现

- 定义实体
- 客户端实体(RegisteredClient)
- 授权信息实体(OAuth2Authorization)
- 授权确认信息实体(OAuth2AuthorizationConsent)
- 定义Spring Data Repositories(Redis Repositories)
- 客户端Repository
- 授权信息Repository
- 授权确认信息Repository
- 实现核心service
- 客户端Repository(RegisteredClientRepository)
- 授权信息的service
- 授权确认信息的service
- 去除认证服务配置文件中这三个核心service的注入

写在最后

附录

相关推荐

- 

Spring Authorization Server (1) 认证、授权、oauth2概念和流程初步介绍

474阅读 · 3点赞
- Spring Authorization Server入门 (八) Spring Boot引入Security OAuth2 Client对接认证服务

2.8k阅读 · 13点赞
- Spring Authorization Server入门 (十三) 实现联合身份认证，集成Github与Gitee的OAuth登录

2.3k阅读 · 13点赞
- 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

4.8k阅读 · 24点赞
- Spring Authorization Server优化篇：Redis值序列化器添加Jackson Mixin，解决Redis反序列化失败问题

547阅读 · 4点赞



避免 FastAPI 多进程环境下 ApScheduler 定时任务重复触发的方法

郝同学的测开笔记 · 608阅读 · 2点赞

修复等保漏洞：升级ssh到9.4版本

玛奇玛丶 · 267阅读 · 1点赞

一篇搞懂springsecurity项目实战(纯干货)

璐画 · 269阅读 · 1点赞

如何在Node.js中实现一个缓存系统？

Knockkk · 246阅读 · 2点赞

关于类和接口设计的11个好习惯

菜菜的后端私房菜 · 328阅读 · 5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 6.8k 31 86 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 4.8k 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 2.8k 13 43 Spring ... Spring

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

叹雪飞花 7月前 2.6k 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 8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 2.5k 15 39 Spring Spring ... 安全

Spring Authorization Server入门 (七) 登录添加图形验证码

Springboot3.x最简集成SpringDoc-OpenApi

叹雪飞花   4月前    2.3k    17    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    1.9k    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 (二十) 实现二维码扫码登录

叹雪飞花   2月前    936    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    762    9    12

Vue.js   安全   Spring ...

Spring Cloud Gateway集成SpringDoc，集中管理微服务API

叹雪飞花   3月前    767    7    评论

Spring ...   Spring ...   Java