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

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

```java
package com.example.config;

import com.fasterxml.jackson.annotation.JsonAutoDetect;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import com.fasterxml.jackson.annotation.PropertyAccessor;
import com.fasterxml.jackson.databind.ObjectMapper;
import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
import org.springframework.data.redis.repository.configuration.EnableRedisRepositories;
import org.springframework.data.redis.serializer.Jackson2JsonRedisSerializer;
import org.springframework.data.redis.serializer.StringRedisSerializer;
import org.springframework.http.converter.json.Jackson2ObjectMapperBuilder;
import org.springframework.security.jackson2.CoreJackson2Module;

/**
 * Redis的key序列化配置类
 *
 * @author vains
 */
@Configuration
@RequiredArgsConstructor
@EnableRedisRepositories(enableKeyspaceEvents = RedisKeyValueAdapter.EnableKeyspaceEvents.ON_STARTUP
public class RedisConfig {

    private final Jackson2ObjectMapperBuilder builder;

    /**
     * 默认情况下使用
     *
     * @param connectionFactory redis链接工厂
     * @return RedisTemplate
     */
    @Bean
    public RedisTemplate<Object, Object> redisTemplate(RedisConnectionFactory connectionFactory)
        // 字符串序列化器
        StringRedisSerializer stringRedisSerializer = new StringRedisSerializer();

        // 创建ObjectMapper并添加默认配置
        ObjectMapper objectMapper = builder.createXmlMapper(false).build();

        // 序列化所有字段
        objectMapper.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);

        // 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：
        //     java.lang.ClassCastException: java.util.LinkedHashMap cannot be cast to XXX
        objectMapper.activateDefaultTyping(
                objectMapper.getPolymorphicTypeValidator(),
                ObjectMapper.DefaultTyping.NON_FINAL,
                JsonTypeInfo.As.PROPERTY);

        // 添加Security提供的Jackson Mixin
        objectMapper.registerModule(new CoreJackson2Module());

        // 存入redis时序列化值的序列化器
```

```
        RedisTemplate<Object, Object> redisTemplate = new RedisTemplate<>();

        // 设置值序列化
        redisTemplate.setValueSerializer(valueSerializer);
        // 设置hash格式数据值的序列化器
        redisTemplate.setHashValueSerializer(valueSerializer);
        // 默认的Key序列化器为：JdkSerializationRedisSerializer
        redisTemplate.setKeySerializer(stringRedisSerializer);
        // 设置字符串序列化器
        redisTemplate.setStringSerializer(stringRedisSerializer);
        // 设置hash结构的key的序列化器
        redisTemplate.setHashKeySerializer(stringRedisSerializer);

        // 设置连接工厂
        redisTemplate.setConnectionFactory(connectionFactory);

        return redisTemplate;
    }

}
```

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

```java
package com.example.entity.security;

import lombok.Data;
import org.springframework.data.annotation.Id;
import org.springframework.data.redis.core.RedisHash;
import org.springframework.data.redis.core.index.Indexed;

import java.io.Serializable;
import java.time.Instant;

/**
 * 基于redis存储的客户端实体
 *
 * @author vains
 */
@Data
@RedisHash(value = "client")
public class RedisRegisteredClient implements Serializable {

    /**
     * 主键
     */
    @Id
    private String id;

    /**
     * 客户端id
     */
    @Indexed
    private String clientId;

    /**
     * 客户端id签发时间
     */
    private Instant clientIdIssuedAt;

    /**
     * 客户端秘钥
     */
```

```
     * 客户端秘钥过期时间
     */
    private Instant clientSecretExpiresAt;

    /**
     * 客户端名称
     */
    private String clientName;

    /**
     * 客户端支持的认证方式
     */
    private String clientAuthenticationMethods;

    /**
     * 客户端支持的授权申请方式
     */
    private String authorizationGrantTypes;

    /**
     * 回调地址
     */
    private String redirectUris;

    /**
     * 登出回调地址
     */
    private String postLogoutRedirectUris;

    /**
     * 客户端拥有的scope
     */
    private String scopes;

    /**
     * 客户端配置
     */
    private String clientSettings;

    /**
     * 通过该客户端签发的access token设置
     */
    private String tokenSettings;

}
```

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

```java
package com.example.entity.security;

import lombok.Data;
import org.springframework.data.annotation.Id;
import org.springframework.data.redis.core.RedisHash;
import org.springframework.data.redis.core.TimeToLive;
import org.springframework.data.redis.core.index.Indexed;

import java.io.Serializable;
import java.time.Instant;
import java.util.concurrent.TimeUnit;

/**
 * 使用Repository将授权申请的认证信息缓存至redis的实体
 *
 * @author vains
 */
@Data
@RedisHash(value = "authorization")
public class RedisOAuth2Authorization implements Serializable {

    /**
     * 主键
     */
    @Id
    private String id;

    /**
     * 授权申请时使用的客户端id
     */
    private String registeredClientId;

    /**
     * 授权用户姓名
     */
    private String principalName;

    /**
     * 授权申请时使用 grant_type
     */
    private String authorizationGrantType;
```

```
     */
    private String authorizedScopes;

    /**
     * 授权的认证信息(当前用户)、请求信息(授权申请请求)
     */
    private String attributes;

    /**
     * 授权申请时的state
     */
    @Indexed
    private String state;

    /**
     * 授权码的值
     */
    @Indexed
    private String authorizationCodeValue;

    /**
     * 授权码签发时间
     */
    private Instant authorizationCodeIssuedAt;

    /**
     * 授权码过期时间
     */
    private Instant authorizationCodeExpiresAt;

    /**
     * 授权码元数据
     */
    private String authorizationCodeMetadata;

    /**
     * access token的值
     */
    @Indexed
    private String accessTokenValue;

    /**
     * access token签发时间
     */
    private Instant accessTokenIssuedAt;

```

```
    private Instant accessTokenExpiresAt;

    /**
     * access token元数据
     */
    private String accessTokenMetadata;

    /**
     * access token的类型
     */
    private String accessTokenType;

    /**
     * access token中包含的scope
     */
    private String accessTokenScopes;

    /**
     * refresh token的值
     */
    @Indexed
    private String refreshTokenValue;

    /**
     * refresh token签发使劲
     */
    private Instant refreshTokenIssuedAt;

    /**
     * refresh token过期时间
     */
    private Instant refreshTokenExpiresAt;

    /**
     * refresh token元数据
     */
    private String refreshTokenMetadata;

    /**
     * id token的值
     */
    @Indexed
    private String oidcIdTokenValue;

    /**
     * id token签发时间
```

```
    /**
     * id token过期时间
     */
    private Instant oidcIdTokenExpiresAt;

    /**
     * id token元数据
     */
    private String oidcIdTokenMetadata;

    /**
     * id token中包含的属性
     */
    private String oidcIdTokenClaims;

    /**
     * 用户码的值
     */
    @Indexed
    private String userCodeValue;

    /**
     * 用户码签发时间
     */
    private Instant userCodeIssuedAt;

    /**
     * 用户码过期时间
     */
    private Instant userCodeExpiresAt;

    /**
     * 用户码元数据
     */
    private String userCodeMetadata;

    /**
     * 设备码的值
     */
    @Indexed
    private String deviceCodeValue;

    /**
     * 设备码签发时间
     */
    private Instant deviceCodeIssuedAt;
```

```
     */
    private Instant deviceCodeExpiresAt;

    /**
     * 设备码元数据
     */
    private String deviceCodeMetadata;

    /**
     * 当前对象在Redis中的过期时间
     */
    @TimeToLive(unit = TimeUnit.MINUTES)
    private Long timeout;

}
```

java 复制代码

```java
package com.example.entity.security;

import lombok.Data;
import org.springframework.data.annotation.Id;
import org.springframework.data.redis.core.RedisHash;
import org.springframework.data.redis.core.index.Indexed;

import java.io.Serializable;

/**
 * 基于redis的授权确认存储实体
 *
 * @author vains
 */
@Data
@RedisHash(value = "authorizationConsent")
public class RedisAuthorizationConsent implements Serializable {

    /**
     * 额外提供的主键
     */
    @Id
    private String id;
```

```
     */
    @Indexed
    private String registeredClientId;

    /**
     * 当前授权确认用户的 username
     */
    @Indexed
    private String principalName;

    /**
     * 授权确认的scope
     */
    private String authorities;

}
```

「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。

```
package com.example.repository;

import com.example.entity.security.RedisRegisteredClient;
import org.springframework.data.repository.CrudRepository;

import java.util.Optional;

/**
 * 基于Spring Data Redis的客户端repository
 *
 * @author vains
 */
public interface RedisClientRepository extends CrudRepository<RedisRegisteredClient, String

    /**
     * 根据客户端Id查询客户端信息
     *
     * @param clientId 客户端id
     * @return 客户端信息
     */
    Optional<RedisRegisteredClient> findByClientId(String clientId);

}
```

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

```
package com.example.repository;

import com.example.entity.security.RedisOAuth2Authorization;
import org.springframework.data.repository.CrudRepository;

import java.util.Optional;

/**
 * oauth2授权管理
```

```
public interface RedisOAuth2AuthorizationRepository extends CrudRepository<RedisOAuth2Aut

    /**
     * 根据授权码获取认证信息
     *
     * @param token 授权码
     * @return 认证信息
     */
    Optional<RedisOAuth2Authorization> findByAuthorizationCodeValue(String token);

    /**
     * 根据access token获取认证信息
     *
     * @param token access token
     * @return 认证信息
     */
    Optional<RedisOAuth2Authorization> findByAccessTokenValue(String token);

    /**
     * 根据刷新token获取认证信息
     *
     * @param token 刷新token
     * @return 认证信息
     */
    Optional<RedisOAuth2Authorization> findByRefreshTokenValue(String token);

    /**
     * 根据id token获取认证信息
     *
     * @param token id token
     * @return 认证信息
     */
    Optional<RedisOAuth2Authorization> findByOidcIdTokenValue(String token);

    /**
     * 根据用户码获取认证信息
     *
     * @param token 用户码
     * @return 认证信息
     */
    Optional<RedisOAuth2Authorization> findByUserCodeValue(String token);

    /**
     * 根据设备码获取认证信息
     *
     * @param token 设备码
```

```

    /**
     * 根据state获取认证信息
     *
     * @param token 授权申请时的state
     * @return 认证信息
     */
    Optional<RedisOAuth2Authorization> findByState(String token);
}
```

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

```java
package com.example.repository;

import com.example.entity.security.RedisAuthorizationConsent;
import org.springframework.data.repository.CrudRepository;

import java.util.Optional;

/**
 * 基于redis的授权确认repository
 *
 * @author vains
 */
public interface RedisAuthorizationConsentRepository extends CrudRepository<RedisAuthori

    /**
     * 根据客户端id和授权确认用户的 username 查询授权确认信息
     *
     * @param registeredClientId 客户端id
     * @param principalName      授权确认用户的 username
     * @return 授权确认记录
     */
    Optional<RedisAuthorizationConsent> findByRegisteredClientIdAndPrincipalName(String regist

```

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	findByLastnameAndFirstname	SINTER …:firstname:rand …:lastname:al’ thor
Or	findByLastnameOrFirstname	SUNION …:firstname:rand …:lastname:al’ thor
Is, Equals	findByFirstname , findByFirstnameIs , findByFirstnameEquals	SINTER …:firstname:rand
IsTrue	FindByAliveIsTrue	SINTER …:alive:1
IsFalse	findByAliveIsFalse	SINTER …:alive:0
Top,First	findFirst10ByFirstname , findTop5ByFirstname	

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

```java
package com.example.repository;

import com.example.entity.security.RedisRegisteredClient;
import com.example.service.impl.RedisOAuth2AuthorizationService;
```

```
import jakarta.annotation.PostConstruct;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.jackson2.SecurityJackson2Modules;
import org.springframework.security.oauth2.core.AuthorizationGrantType;
import org.springframework.security.oauth2.core.ClientAuthenticationMethod;
import org.springframework.security.oauth2.core.oidc.OidcScopes;
import org.springframework.security.oauth2.jose.jws.SignatureAlgorithm;
import org.springframework.security.oauth2.server.authorization.client.RegisteredClient;
import org.springframework.security.oauth2.server.authorization.client.RegisteredClientRepository;
import org.springframework.security.oauth2.server.authorization.jackson2.OAuth2AuthorizationServerJ
import org.springframework.security.oauth2.server.authorization.settings.ClientSettings;
import org.springframework.security.oauth2.server.authorization.settings.OAuth2TokenFormat;
import org.springframework.security.oauth2.server.authorization.settings.TokenSettings;
import org.springframework.stereotype.Repository;
import org.springframework.util.Assert;
import org.springframework.util.StringUtils;

import java.time.Duration;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.UUID;

/**
 * 基于redis的客户端repository实现
 *
 * @author vains
 */
@Slf4j
@Repository
@RequiredArgsConstructor
public class RedisRegisteredClientRepository implements RegisteredClientRepository {

    /**
     * 提供给客户端初始化使用(不需要可删除)
     */
    private final PasswordEncoder passwordEncoder;

    private final RedisClientRepository repository;

    private final static ObjectMapper MAPPER = new ObjectMapper();

    static {
```

```
        List<Module> modules = SecurityJackson2Modules.getModules(classLoader);
        MAPPER.registerModules(modules);
        // 加载Authorization Server提供的Module
        MAPPER.registerModule(new OAuth2AuthorizationServerJackson2Module());
    }

    @Override
    public void save(RegisteredClient registeredClient) {
        Assert.notNull(registeredClient, "registeredClient cannot be null");
        this.repository.findByClientId(registeredClient.getClientId())
                .ifPresent(existingRegisteredClient -> this.repository.deleteById(existingRegistered
        this.repository.save(toEntity(registeredClient));
    }

    @Override
    public RegisteredClient findById(String id) {
        Assert.hasText(id, "id cannot be empty");
        return this.repository.findById(id)
                .map(this::toObject).orElse(null);
    }

    @Override
    public RegisteredClient findByClientId(String clientId) {
        Assert.hasText(clientId, "clientId cannot be empty");
        return this.repository.findByClientId(clientId)
                .map(this::toObject).orElse(null);
    }

    private RegisteredClient toObject(RedisRegisteredClient client) {
        Set<String> clientAuthenticationMethods = StringUtils.commaDelimitedListToSet(
                client.getClientAuthenticationMethods());
        Set<String> authorizationGrantTypes = StringUtils.commaDelimitedListToSet(
                client.getAuthorizationGrantTypes());
        Set<String> redirectUris = StringUtils.commaDelimitedListToSet(
                client.getRedirectUris());
        Set<String> postLogoutRedirectUris = StringUtils.commaDelimitedListToSet(
                client.getPostLogoutRedirectUris());
        Set<String> clientScopes = StringUtils.commaDelimitedListToSet(
                client.getScopes());

        RegisteredClient.Builder builder = RegisteredClient.withId(client.getId())
                .clientId(client.getClientId())
                .clientIdIssuedAt(client.getClientIdIssuedAt())
                .clientSecret(client.getClientSecret())
                .clientSecretExpiresAt(client.getClientSecretExpiresAt())
                .clientName(client.getClientName())
```

```
            .authorizationGrantTypes((grantTypes) ->
                    authorizationGrantTypes.forEach(grantType ->
                            grantTypes.add(resolveAuthorizationGrantType(grantType))))
            .redirectUris((uris) -> uris.addAll(redirectUris))
            .postLogoutRedirectUris((uris) -> uris.addAll(postLogoutRedirectUris))
            .scopes((scopes) -> scopes.addAll(clientScopes));

        Map<String, Object> clientSettingsMap = parseMap(client.getClientSettings());
        builder.clientSettings(ClientSettings.withSettings(clientSettingsMap).build());

        Map<String, Object> tokenSettingsMap = parseMap(client.getTokenSettings());
        builder.tokenSettings(TokenSettings.withSettings(tokenSettingsMap).build());

        return builder.build();
    }

    private RedisRegisteredClient toEntity(RegisteredClient registeredClient) {
        List<String> clientAuthenticationMethods = new ArrayList<>(registeredClient.getClientAuthe
        registeredClient.getClientAuthenticationMethods().forEach(clientAuthenticationMethod ->
                clientAuthenticationMethods.add(clientAuthenticationMethod.getValue()));

        List<String> authorizationGrantTypes = new ArrayList<>(registeredClient.getAuthorizationGr
        registeredClient.getAuthorizationGrantTypes().forEach(authorizationGrantType ->
                authorizationGrantTypes.add(authorizationGrantType.getValue()));

        RedisRegisteredClient entity = new RedisRegisteredClient();
        entity.setId(registeredClient.getId());
        entity.setClientId(registeredClient.getClientId());
        entity.setClientIdIssuedAt(registeredClient.getClientIdIssuedAt());
        entity.setClientSecret(registeredClient.getClientSecret());
        entity.setClientSecretExpiresAt(registeredClient.getClientSecretExpiresAt());
        entity.setClientName(registeredClient.getClientName());
        entity.setClientAuthenticationMethods(StringUtils.collectionToCommaDelimitedString(clientAut
        entity.setAuthorizationGrantTypes(StringUtils.collectionToCommaDelimitedString(authorization
        entity.setRedirectUris(StringUtils.collectionToCommaDelimitedString(registeredClient.getRedi
        entity.setPostLogoutRedirectUris(StringUtils.collectionToCommaDelimitedString(registeredClie
        entity.setScopes(StringUtils.collectionToCommaDelimitedString(registeredClient.getScopes()))
        entity.setClientSettings(writeMap(registeredClient.getClientSettings().getSettings()));
        entity.setTokenSettings(writeMap(registeredClient.getTokenSettings().getSettings()));

        return entity;
    }

    private Map<String, Object> parseMap(String data) {
        try {
            return MAPPER.readValue(data, new TypeReference<>() {
```

```
        }
    }

    private String writeMap(Map<String, Object> data) {
        try {
            return MAPPER.writeValueAsString(data);
        } catch (Exception ex) {
            throw new IllegalArgumentException(ex.getMessage(), ex);
        }
    }

    private static AuthorizationGrantType resolveAuthorizationGrantType(String authorizationG
        if (AuthorizationGrantType.AUTHORIZATION_CODE.getValue().equals(authorizationGrantType)) {
            return AuthorizationGrantType.AUTHORIZATION_CODE;
        } else if (AuthorizationGrantType.CLIENT_CREDENTIALS.getValue().equals(authorizationGrantTy
            return AuthorizationGrantType.CLIENT_CREDENTIALS;
        } else if (AuthorizationGrantType.REFRESH_TOKEN.getValue().equals(authorizationGrantType))
            return AuthorizationGrantType.REFRESH_TOKEN;
        }
        // Custom authorization grant type
        return new AuthorizationGrantType(authorizationGrantType);
    }

    private static ClientAuthenticationMethod resolveClientAuthenticationMethod(String client
        if (ClientAuthenticationMethod.CLIENT_SECRET_BASIC.getValue().equals(clientAuthenticationMet
            return ClientAuthenticationMethod.CLIENT_SECRET_BASIC;
        } else if (ClientAuthenticationMethod.CLIENT_SECRET_POST.getValue().equals(clientAuthentica
            return ClientAuthenticationMethod.CLIENT_SECRET_POST;
        } else if (ClientAuthenticationMethod.NONE.getValue().equals(clientAuthenticationMethod)) {
            return ClientAuthenticationMethod.NONE;
        }
        // Custom client authentication method
        return new ClientAuthenticationMethod(clientAuthenticationMethod);
    }

    /**
     * 容器启动后初始化客户端
     * (不需要可删除)
     */
    @PostConstruct
    public void initClients() {
        log.info("Initialize client information to Redis.");
        // 默认需要授权确认
        ClientSettings.Builder builder = ClientSettings.builder()
                .requireAuthorizationConsent(Boolean.TRUE);

```

```
            // Access Token 存活时间：2小时
            .accessTokenTimeToLive(Duration.ofHours(2L))
            // 授权码存活时间：5分钟
            .authorizationCodeTimeToLive(Duration.ofMinutes(5L))
            // 设备码存活时间：5分钟
            .deviceCodeTimeToLive(Duration.ofMinutes(5L))
            // Refresh Token 存活时间：7天
            .refreshTokenTimeToLive(Duration.ofDays(7L))
            // 刷新 Access Token 后是否重用 Refresh Token
            .reuseRefreshTokens(Boolean.TRUE)
            // 设置 Id Token 加密方式
            .idTokenSignatureAlgorithm(SignatureAlgorithm.RS256)
            .build();

        // 正常授权码客户端
        RegisteredClient registeredClient = RegisteredClient.withId(UUID.randomUUID().toString())
            // 客户端id
            .clientId("messaging-client")
            // 客户端名称
            .clientName("授权码")
            // 客户端秘钥，使用密码解析器加密
            .clientSecret(passwordEncoder.encode("123456"))
            // 客户端认证方式，基于请求头的认证
            .clientAuthenticationMethod(ClientAuthenticationMethod.CLIENT_SECRET_BASIC)
            // 配置资源服务器使用该客户端获取授权时支持的方式
            .authorizationGrantType(AuthorizationGrantType.AUTHORIZATION_CODE)
            .authorizationGrantType(AuthorizationGrantType.REFRESH_TOKEN)
            .authorizationGrantType(AuthorizationGrantType.CLIENT_CREDENTIALS)
            // 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，
            .redirectUri("http://127.0.0.1:8000/login/oauth2/code/messaging-client-oidc")
            .redirectUri("https://www.baidu.com")
            // 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时
            .scope(OidcScopes.OPENID)
            .scope(OidcScopes.PROFILE)
            // 指定scope
            .scope("message.read")
            .scope("message.write")
            // 客户端设置，设置用户需要确认授权
            .clientSettings(builder.build())
            // token相关配置
            .tokenSettings(tokenSettings)
            .build();

        // 设备码授权客户端
        RegisteredClient deviceClient = RegisteredClient.withId(UUID.randomUUID().toString())
            .clientId("device-message-client")
```

```
                // 设备码授权
                .authorizationGrantType(AuthorizationGrantType.DEVICE_CODE)
                .authorizationGrantType(AuthorizationGrantType.REFRESH_TOKEN)
                // 指定scope
                .scope("message.read")
                .scope("message.write")
                // token相关配置
                .tokenSettings(tokenSettings)
                .build();

        // PKCE客户端
        RegisteredClient pkceClient = RegisteredClient.withId(UUID.randomUUID().toString())
                .clientId("pkce-message-client")
                .clientName("PKCE流程")
                // 公共客户端
                .clientAuthenticationMethod(ClientAuthenticationMethod.NONE)
                // 设备码授权
                .authorizationGrantType(AuthorizationGrantType.AUTHORIZATION_CODE)
                .authorizationGrantType(AuthorizationGrantType.REFRESH_TOKEN)
                // 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，
                .redirectUri("http://127.0.0.1:8000/login/oauth2/code/messaging-client-oidc")
                // 开启 PKCE 流程
                .clientSettings(builder.requireProofKey(Boolean.TRUE).build())
                // 指定scope
                .scope("message.read")
                .scope("message.write")
                // token相关配置
                .tokenSettings(tokenSettings)
                .build();

        // 初始化客户端
        this.save(registeredClient);
        this.save(deviceClient);
        this.save(pkceClient);
    }

}
```

「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。」

```
package com.example.service.impl;

import com.example.entity.security.RedisOAuth2Authorization;
import com.example.repository.RedisOAuth2AuthorizationRepository;
import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.Module;
import com.fasterxml.jackson.databind.ObjectMapper;
import lombok.RequiredArgsConstructor;
import org.springframework.dao.DataRetrievalFailureException;
import org.springframework.security.jackson2.SecurityJackson2Modules;
import org.springframework.security.oauth2.core.*;
import org.springframework.security.oauth2.core.endpoint.OAuth2ParameterNames;
import org.springframework.security.oauth2.core.oidc.OidcIdToken;
import org.springframework.security.oauth2.core.oidc.endpoint.OidcParameterNames;
import org.springframework.security.oauth2.server.authorization.OAuth2Authorization;
import org.springframework.security.oauth2.server.authorization.OAuth2AuthorizationCode;
import org.springframework.security.oauth2.server.authorization.OAuth2AuthorizationService;
import org.springframework.security.oauth2.server.authorization.OAuth2TokenType;
import org.springframework.security.oauth2.server.authorization.client.RegisteredClient;
import org.springframework.security.oauth2.server.authorization.client.RegisteredClientRepository;
import org.springframework.security.oauth2.server.authorization.jackson2.OAuth2AuthorizationServerJ
import org.springframework.stereotype.Service;
import org.springframework.util.Assert;
import org.springframework.util.StringUtils;

import java.time.Duration;
import java.time.Instant;
import java.util.*;
import java.util.function.Consumer;

/**
 * 基于redis的授权管理服务
 *
 * @author vains
 */
@Service
@RequiredArgsConstructor
public class RedisOAuth2AuthorizationService implements OAuth2AuthorizationService {

    private final RegisteredClientRepository registeredClientRepository;

    private final RedisOAuth2AuthorizationRepository oAuth2AuthorizationRepository;

    private final static ObjectMapper MAPPER = new ObjectMapper();

    static {
```

```
        List<Module> modules = SecurityJackson2Modules.getModules(classLoader);
        MAPPER.registerModules(modules);
        // 加载Authorization Server提供的Module
        MAPPER.registerModule(new OAuth2AuthorizationServerJackson2Module());
    }

    @Override
    public void save(OAuth2Authorization authorization) {
        Optional<RedisOAuth2Authorization> existingAuthorization = oAuth2AuthorizationRepository.fin

        // 如果已存在则删除后再保存
        existingAuthorization.map(RedisOAuth2Authorization::getId)
                .ifPresent(oAuth2AuthorizationRepository::deleteById);

        // 过期时间，默认永不过期
        long maxTimeout = -1L;
        // 所有code的过期时间，方便计算最大值
        List<Instant> expiresAtList = new ArrayList<>();

        RedisOAuth2Authorization entity = toEntity(authorization);

        // 如果有过期时间就存入
        Optional.ofNullable(entity.getAuthorizationCodeExpiresAt())
                .ifPresent(expiresAtList::add);

        // 如果有过期时间就存入
        Optional.ofNullable(entity.getAccessTokenExpiresAt())
                .ifPresent(expiresAtList::add);

        // 如果有过期时间就存入
        Optional.ofNullable(entity.getRefreshTokenExpiresAt())
                .ifPresent(expiresAtList::add);

        // 如果有过期时间就存入
        Optional.ofNullable(entity.getOidcIdTokenExpiresAt())
                .ifPresent(expiresAtList::add);

        // 如果有过期时间就存入
        Optional.ofNullable(entity.getUserCodeExpiresAt())
                .ifPresent(expiresAtList::add);

        // 如果有过期时间就存入
        Optional.ofNullable(entity.getDeviceCodeExpiresAt())
                .ifPresent(expiresAtList::add);

        // 获取最大的日期
```

```
            Duration between = Duration.between(Instant.now(), maxInstant.get());
            // 转为分钟
            maxTimeout = between.toMinutes();
        }

        // 设置过期时间
        entity.setTimeout(maxTimeout);

        // 保存至redis
        oAuth2AuthorizationRepository.save(entity);
    }

    @Override
    public void remove(OAuth2Authorization authorization) {
        Assert.notNull(authorization, "authorization cannot be null");
        oAuth2AuthorizationRepository.deleteById(authorization.getId());
    }

    @Override
    public OAuth2Authorization findById(String id) {
        Assert.hasText(id, "id cannot be empty");
        return oAuth2AuthorizationRepository.findById(id)
                .map(this::toObject).orElse(null);
    }

    @Override
    public OAuth2Authorization findByToken(String token, OAuth2TokenType tokenType) {
        Assert.hasText(token, "token cannot be empty");

        Optional<RedisOAuth2Authorization> result;

        if (tokenType == null) {
            result = oAuth2AuthorizationRepository.findByState(token)
                    .or(() -> oAuth2AuthorizationRepository.findByAuthorizationCodeValue(token))
                    .or(() -> oAuth2AuthorizationRepository.findByAccessTokenValue(token))
                    .or(() -> oAuth2AuthorizationRepository.findByOidcIdTokenValue(token))
                    .or(() -> oAuth2AuthorizationRepository.findByRefreshTokenValue(token))
                    .or(() -> oAuth2AuthorizationRepository.findByUserCodeValue(token))
                    .or(() -> oAuth2AuthorizationRepository.findByDeviceCodeValue(token));
        } else if (OAuth2ParameterNames.STATE.equals(tokenType.getValue())) {
            result = oAuth2AuthorizationRepository.findByState(token);
        } else if (OAuth2ParameterNames.CODE.equals(tokenType.getValue())) {
            result = oAuth2AuthorizationRepository.findByAuthorizationCodeValue(token);
        } else if (OAuth2TokenType.ACCESS_TOKEN.equals(tokenType)) {
            result = oAuth2AuthorizationRepository.findByAccessTokenValue(token);
        } else if (OidcParameterNames.ID_TOKEN.equals(tokenType.getValue())) {
```

```
        } else if (OAuth2ParameterNames.USER_CODE.equals(tokenType.getValue())) {
            result = oAuth2AuthorizationRepository.findByUserCodeValue(token);
        } else if (OAuth2ParameterNames.DEVICE_CODE.equals(tokenType.getValue())) {
            result = oAuth2AuthorizationRepository.findByDeviceCodeValue(token);
        } else {
            result = Optional.empty();
        }

        return result.map(this::toObject).orElse(null);
    }

    /**
     * 将redis中存储的类型转为框架所需的类型
     *
     * @param entity redis中存储的类型
     * @return 框架所需的类型
     */
    private OAuth2Authorization toObject(RedisOAuth2Authorization entity) {
        RegisteredClient registeredClient = this.registeredClientRepository.findById(entity.getRegis
        if (registeredClient == null) {
            throw new DataRetrievalFailureException(
                    "The RegisteredClient with id '" + entity.getRegisteredClientId() + "' was not f
        }

        OAuth2Authorization.Builder builder = OAuth2Authorization.withRegisteredClient(registeredCli
                .id(entity.getId())
                .principalName(entity.getPrincipalName())
                .authorizationGrantType(resolveAuthorizationGrantType(entity.getAuthorizationGrantTy
                .authorizedScopes(StringUtils.commaDelimitedListToSet(entity.getAuthorizedScopes()))
                .attributes(attributes -> attributes.putAll(parseMap(entity.getAttributes())));
        if (entity.getState() != null) {
            builder.attribute(OAuth2ParameterNames.STATE, entity.getState());
        }

        if (entity.getAuthorizationCodeValue() != null) {
            OAuth2AuthorizationCode authorizationCode = new OAuth2AuthorizationCode(
                    entity.getAuthorizationCodeValue(),
                    entity.getAuthorizationCodeIssuedAt(),
                    entity.getAuthorizationCodeExpiresAt());
            builder.token(authorizationCode, metadata -> metadata.putAll(parseMap(entity.getAuthoriz
        }

        if (entity.getAccessTokenValue() != null) {
            OAuth2AccessToken accessToken = new OAuth2AccessToken(
                    OAuth2AccessToken.TokenType.BEARER,
                    entity.getAccessTokenValue(),
```

```
            builder.token(accessToken, metadata -> metadata.putAll(parseMap(entity.getAccessTokenMet
        }

        if (entity.getRefreshTokenValue() != null) {
            OAuth2RefreshToken refreshToken = new OAuth2RefreshToken(
                    entity.getRefreshTokenValue(),
                    entity.getRefreshTokenIssuedAt(),
                    entity.getRefreshTokenExpiresAt());
            builder.token(refreshToken, metadata -> metadata.putAll(parseMap(entity.getRefreshTokenM
        }

        if (entity.getOidcIdTokenValue() != null) {
            OidcIdToken idToken = new OidcIdToken(
                    entity.getOidcIdTokenValue(),
                    entity.getOidcIdTokenIssuedAt(),
                    entity.getOidcIdTokenExpiresAt(),
                    parseMap(entity.getOidcIdTokenClaims()));
            builder.token(idToken, metadata -> metadata.putAll(parseMap(entity.getOidcIdTokenMetadat
        }

        if (entity.getUserCodeValue() != null) {
            OAuth2UserCode userCode = new OAuth2UserCode(
                    entity.getUserCodeValue(),
                    entity.getUserCodeIssuedAt(),
                    entity.getUserCodeExpiresAt());
            builder.token(userCode, metadata -> metadata.putAll(parseMap(entity.getUserCodeMetadata(
        }

        if (entity.getDeviceCodeValue() != null) {
            OAuth2DeviceCode deviceCode = new OAuth2DeviceCode(
                    entity.getDeviceCodeValue(),
                    entity.getDeviceCodeIssuedAt(),
                    entity.getDeviceCodeExpiresAt());
            builder.token(deviceCode, metadata -> metadata.putAll(parseMap(entity.getDeviceCodeMetad
        }

        return builder.build();
    }

    /**
     * 将框架所需的类型转为redis中存储的类型
     *
     * @param authorization 框架所需的类型
     * @return redis中存储的类型
     */
    private RedisOAuth2Authorization toEntity(OAuth2Authorization authorization) {
```

```
        entity.setPrincipalName(authorization.getPrincipalName());
        entity.setAuthorizationGrantType(authorization.getAuthorizationGrantType().getValue());
        entity.setAuthorizedScopes(StringUtils.collectionToDelimitedString(authorization.getAuthoriz
        entity.setAttributes(writeMap(authorization.getAttributes()));
        entity.setState(authorization.getAttribute(OAuth2ParameterNames.STATE));

        OAuth2Authorization.Token<OAuth2AuthorizationCode> authorizationCode =
                authorization.getToken(OAuth2AuthorizationCode.class);
        setTokenValues(
                authorizationCode,
                entity::setAuthorizationCodeValue,
                entity::setAuthorizationCodeIssuedAt,
                entity::setAuthorizationCodeExpiresAt,
                entity::setAuthorizationCodeMetadata
        );

        OAuth2Authorization.Token<OAuth2AccessToken> accessToken =
                authorization.getToken(OAuth2AccessToken.class);
        setTokenValues(
                accessToken,
                entity::setAccessTokenValue,
                entity::setAccessTokenIssuedAt,
                entity::setAccessTokenExpiresAt,
                entity::setAccessTokenMetadata
        );
        if (accessToken != null && accessToken.getToken().getScopes() != null) {
            entity.setAccessTokenScopes(StringUtils.collectionToDelimitedString(accessToken.getToken
        }

        OAuth2Authorization.Token<OAuth2RefreshToken> refreshToken =
                authorization.getToken(OAuth2RefreshToken.class);
        setTokenValues(
                refreshToken,
                entity::setRefreshTokenValue,
                entity::setRefreshTokenIssuedAt,
                entity::setRefreshTokenExpiresAt,
                entity::setRefreshTokenMetadata
        );

        OAuth2Authorization.Token<OidcIdToken> oidcIdToken =
                authorization.getToken(OidcIdToken.class);
        setTokenValues(
                oidcIdToken,
                entity::setOidcIdTokenValue,
                entity::setOidcIdTokenIssuedAt,
                entity::setOidcIdTokenExpiresAt,
```

```
            entity.setOidcIdTokenClaims(writeMap(oidcIdToken.getClaims()));
        }

        OAuth2Authorization.Token<OAuth2UserCode> userCode =
                authorization.getToken(OAuth2UserCode.class);
        setTokenValues(
                userCode,
                entity::setUserCodeValue,
                entity::setUserCodeIssuedAt,
                entity::setUserCodeExpiresAt,
                entity::setUserCodeMetadata
        );

        OAuth2Authorization.Token<OAuth2DeviceCode> deviceCode =
                authorization.getToken(OAuth2DeviceCode.class);
        setTokenValues(
                deviceCode,
                entity::setDeviceCodeValue,
                entity::setDeviceCodeIssuedAt,
                entity::setDeviceCodeExpiresAt,
                entity::setDeviceCodeMetadata
        );

        return entity;
    }

    /**
     * 设置token的值
     *
     * @param token              Token实例
     * @param tokenValueConsumer set方法
     * @param issuedAtConsumer   set方法
     * @param expiresAtConsumer  set方法
     * @param metadataConsumer   set方法
     */
    private void setTokenValues(
            OAuth2Authorization.Token<?> token,
            Consumer<String> tokenValueConsumer,
            Consumer<Instant> issuedAtConsumer,
            Consumer<Instant> expiresAtConsumer,
            Consumer<String> metadataConsumer) {
        if (token != null) {
            OAuth2Token oAuth2Token = token.getToken();
            tokenValueConsumer.accept(oAuth2Token.getTokenValue());
            issuedAtConsumer.accept(oAuth2Token.getIssuedAt());
            expiresAtConsumer.accept(oAuth2Token.getExpiresAt());
```

```
    /**
     * 处理授权申请时的 GrantType
     *
     * @param authorizationGrantType 授权申请时的 GrantType
     * @return AuthorizationGrantType的实例
     */
    private static AuthorizationGrantType resolveAuthorizationGrantType(String authorizationG
        if (AuthorizationGrantType.AUTHORIZATION_CODE.getValue().equals(authorizationGrantType)) {
            return AuthorizationGrantType.AUTHORIZATION_CODE;
        } else if (AuthorizationGrantType.CLIENT_CREDENTIALS.getValue().equals(authorizationGrantTy
            return AuthorizationGrantType.CLIENT_CREDENTIALS;
        } else if (AuthorizationGrantType.REFRESH_TOKEN.getValue().equals(authorizationGrantType))
            return AuthorizationGrantType.REFRESH_TOKEN;
        } else if (AuthorizationGrantType.DEVICE_CODE.getValue().equals(authorizationGrantType)) {
            return AuthorizationGrantType.DEVICE_CODE;
        }
        // Custom authorization grant type
        return new AuthorizationGrantType(authorizationGrantType);
    }

    /**
     * 将json转为map
     *
     * @param data json
     * @return map对象
     */
    private Map<String, Object> parseMap(String data) {
        try {
            return MAPPER.readValue(data, new TypeReference<>() {
            });
        } catch (Exception ex) {
            throw new IllegalArgumentException(ex.getMessage(), ex);
        }
    }

    /**
     * 将map对象转为json字符串
     *
     * @param metadata map对象
     * @return json字符串
     */
    private String writeMap(Map<String, Object> metadata) {
        try {
            return MAPPER.writeValueAsString(metadata);
        } catch (Exception ex) {
```

```

}

```

java 复制代码

```java
package com.example.service.impl;

import com.example.entity.security.RedisAuthorizationConsent;
import com.example.repository.RedisAuthorizationConsentRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.dao.DataRetrievalFailureException;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.oauth2.server.authorization.OAuth2AuthorizationConsent;
import org.springframework.security.oauth2.server.authorization.OAuth2AuthorizationConsentService;
import org.springframework.security.oauth2.server.authorization.client.RegisteredClient;
import org.springframework.security.oauth2.server.authorization.client.RegisteredClientRepository;
import org.springframework.stereotype.Service;
import org.springframework.util.Assert;
import org.springframework.util.StringUtils;

import java.util.HashSet;
import java.util.Set;
import java.util.UUID;

/**
 * 基于redis的授权确认服务实现
 *
 * @author vains
 */
@Service
@RequiredArgsConstructor
public class RedisOAuth2AuthorizationConsentService implements OAuth2AuthorizationConse

    private final RegisteredClientRepository registeredClientRepository;

    private final RedisAuthorizationConsentRepository authorizationConsentRepository;

    @Override
```

```
        // 如果存在就先删除
        this.authorizationConsentRepository.findByRegisteredClientIdAndPrincipalName(
                        authorizationConsent.getRegisteredClientId(), authorizationConsent.getPrinci
                .ifPresent(existingConsent -> this.authorizationConsentRepository.deleteById(existin

        // 保存
        RedisAuthorizationConsent entity = toEntity(authorizationConsent);
        entity.setId(UUID.randomUUID().toString());
        this.authorizationConsentRepository.save(entity);
    }

    @Override
    public void remove(OAuth2AuthorizationConsent authorizationConsent) {
        Assert.notNull(authorizationConsent, "authorizationConsent cannot be null");
        // 如果存在就删除
        this.authorizationConsentRepository.findByRegisteredClientIdAndPrincipalName(
                        authorizationConsent.getRegisteredClientId(), authorizationConsent.getPrinci
                .ifPresent(existingConsent -> this.authorizationConsentRepository.deleteById(existin
    }

    @Override
    public OAuth2AuthorizationConsent findById(String registeredClientId, String principalName) {
        Assert.hasText(registeredClientId, "registeredClientId cannot be empty");
        Assert.hasText(principalName, "principalName cannot be empty");
        return this.authorizationConsentRepository.findByRegisteredClientIdAndPrincipalName(
                registeredClientId, principalName).map(this::toObject).orElse(null);
    }

    private OAuth2AuthorizationConsent toObject(RedisAuthorizationConsent authorizationConsent) {
        String registeredClientId = authorizationConsent.getRegisteredClientId();
        RegisteredClient registeredClient = this.registeredClientRepository.findById(registeredClien
        if (registeredClient == null) {
            throw new DataRetrievalFailureException(
                    "The RegisteredClient with id '" + registeredClientId + "' was not found in the
        }

        OAuth2AuthorizationConsent.Builder builder = OAuth2AuthorizationConsent.withId(
                registeredClientId, authorizationConsent.getPrincipalName());
        if (authorizationConsent.getAuthorities() != null) {
            for (String authority : StringUtils.commaDelimitedListToSet(authorizationConsent.getAuth
                builder.authority(new SimpleGrantedAuthority(authority));
            }
        }

        return builder.build();
    }
```

```
        entity.setRegisteredClientId(authorizationConsent.getRegisteredClientId());
        entity.setPrincipalName(authorizationConsent.getPrincipalName());

        Set<String> authorities = new HashSet<>();
        for (GrantedAuthority authority : authorizationConsent.getAuthorities()) {
            authorities.add(authority.getAuthority());
        }
        entity.setAuthorities(StringUtils.collectionToCommaDelimitedString(authorities));

        return entity;
    }

}
```

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

```java
/**
 * 配置客户端Repository
 *
 * @param jdbcTemplate db 数据源信息
 * @return 基于数据库的repository
 */
@Bean
public RegisteredClientRepository registeredClientRepository(JdbcTemplate jdbcTemplate) {
    // 基于db存储客户端，还有一个基于内存的实现 InMemoryRegisteredClientRepository
    return new JdbcRegisteredClientRepository(jdbcTemplate);
}

/**
 * 配置基于db的oauth2的授权管理服务
 *
 * @param jdbcTemplate               db数据源信息
 * @param registeredClientRepository 上边注入的客户端repository
 * @return JdbcOAuth2AuthorizationService
 */
@Bean
public OAuth2AuthorizationService authorizationService(JdbcTemplate jdbcTemplate, RegisteredClie
    // 基于db的oauth2认证服务，还有一个基于内存的服务实现InMemoryOAuth2AuthorizationService
    return new JdbcOAuth2AuthorizationService(jdbcTemplate, registeredClientRepository);
```

```
 * 配置基于db的授权确认管理服务
 *
 * @param jdbcTemplate              db数据源信息
 * @param registeredClientRepository 客户端repository
 * @return JdbcOAuth2AuthorizationConsentService
 */
@Bean
public OAuth2AuthorizationConsentService authorizationConsentService(JdbcTemplate jdbcTemplate,
    // 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
    return new JdbcOAuth2AuthorizationConsentService(jdbcTemplate, registeredClientRepository
}
```

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

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

目录 收起

相关推荐

为你推荐

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