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

叹雪飞花 2023-07-05 1,108 阅读14分钟

前言

本篇文章会带大家实现自定义的UserDetailsService，从数据库获取用户及权限信息；也会带大家了解一下框架是怎么获取用户信息的。

实现步骤

1. 初始化数据库表结构
2. 编写相关表的实体、mapper接口和mapper文件
3. 实现 `UserDetailsService` 接口，实现 `loadUserByUsername` 抽象方法。

初始化数据库表结构

数据库表结构使用经典的RBAC模型，一共有五张表：用户、角色、权限、用户角色关联和角色权限关联表；关于三方登录账户信息表需要的可以加一下，该表主要存储三方登录获取到的用户信息。因为只是示例，所以表中字段都很简陋，大家替换成自己的用户表即可。

sql 复制代码

```sql
SET NAMES utf8mb4;
SET
FOREIGN_KEY_CHECKS = 0;

-- ----------------------------
-- Table structure for oauth2_basic_user
-- ----------------------------
```

```
    `id`          int(11) NOT NULL AUTO_INCREMENT COMMENT '自增id',
    `name`        varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NULL DEFAULT NULL COMME
    `account`     varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NULL DEFAULT NULL COMME
    `password`    varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NULL DEFAULT NULL COMME
    `mobile`      varchar(11) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NULL DEFAULT NULL COMMEN
    `email`       varchar(50) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NULL DEFAULT NULL COMMEN
    `avatar_url`  varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NULL DEFAULT NULL COMME
    `deleted`     tinyint(1) NULL DEFAULT NULL COMMENT '是否已删除',
    `source_from` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NULL DEFAULT NULL COMME
    `create_time` datetime(0) NULL DEFAULT NULL COMMENT '创建时间',
    `update_time` datetime(0) NULL DEFAULT NULL COMMENT '修改时间',
    PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 2 CHARACTER SET = utf8mb4 COLLATE = utf8mb4_bin COMMENT = '基

-- ----------------------------
-- Records of oauth2_basic_user
-- ----------------------------
BEGIN;
INSERT INTO `oauth2_basic_user`
VALUES (1, '云逸', 'admin', '$2a$10$K7nVcC.75YZSZU1Fq6G6buYujG.dolGYGPboh7eQbtkdFmB0EfN5K', '176
        '17683906991@163.com', NULL, 0, 'system', '2023-06-20 15:20:42', '2023-06-20 15:20:42');
COMMIT;

-- ----------------------------
-- Table structure for oauth2_third_Account
-- ----------------------------
DROP TABLE IF EXISTS `oauth2_third_Account`;
CREATE TABLE `oauth2_third_account`
(
    `id`          int(11) NOT NULL AUTO_INCREMENT COMMENT '自增id',
    `user_id`     int(11) NULL DEFAULT NULL COMMENT '用户表主键',
    `unique_id`   varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NULL DEFAULT NULL COMME
    `type`        varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NULL DEFAULT NULL COMME
    `blog`        varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NULL DEFAULT NULL COMME
    `location`    varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NULL DEFAULT NULL COMME
    `create_time` datetime(0) NULL DEFAULT NULL COMMENT '绑定时间',
    `update_time` datetime(0) NULL DEFAULT NULL COMMENT '修改时间',
    PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 1 CHARACTER SET = utf8mb4 COLLATE = utf8mb4_bin COMMENT = '三

-- ----------------------------
-- Table structure for sys_authority
-- ----------------------------
DROP TABLE IF EXISTS `sys_authority`;
CREATE TABLE `sys_authority`
(
```

```
    `url`            varchar(64) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NOT NULL COMMENT '跳转
    `authority`      varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NULL DEFAULT NULL CO
    `sort`           tinyint(4) NULL DEFAULT NULL COMMENT '排序',
    `type`           tinyint(4) NOT NULL COMMENT '0:菜单,1:接口',
    `deleted`        tinyint(1) NOT NULL COMMENT '0:启用,1:删除',
    `create_time`    datetime(0) NOT NULL COMMENT '创建时间',
    `create_user_id` int(11) NOT NULL COMMENT '创建人',
    PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 4 CHARACTER SET = utf8mb4 COLLATE = utf8mb4_bin COMMENT = '系

-- ----------------------------
-- Records of sys_authority
-- ----------------------------
BEGIN;
INSERT INTO `sys_authority`
VALUES (1, '系统管理', 0, '/system', 'system', 0, 0, 0, '2022-03-25 23:52:03', 1),
       (2, 'app', 0, '/**', 'app', 1, 1, 0, '2023-06-20 15:18:49', 1),
       (3, 'web', 0, '/**', 'web', 2, 1, 0, '2023-06-20 15:19:12', 1);
COMMIT;

-- ----------------------------
-- Table structure for sys_role
-- ----------------------------
DROP TABLE IF EXISTS `sys_role`;
CREATE TABLE `sys_role`
(
    `id`             int(11) NOT NULL AUTO_INCREMENT COMMENT '角色自增ID',
    `role_name`      varchar(16) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL COMMENT '角
    `deleted`        tinyint(1) NULL DEFAULT NULL COMMENT '0:启用,1:删除',
    `sort`           int(11) NULL DEFAULT NULL COMMENT '排序',
    `create_time`    datetime(0) NOT NULL COMMENT '创建时间',
    `create_user_id` int(11) NOT NULL COMMENT '创建人',
    PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 2 CHARACTER SET = utf8 COLLATE = utf8_general_ci COMMENT = '系

-- ----------------------------
-- Records of sys_role
-- ----------------------------
BEGIN;
INSERT INTO `sys_role`
VALUES (1, '管理员', 0, 0, '2022-03-25 23:51:04', 1);
COMMIT;

-- ----------------------------
-- Table structure for sys_role_authority
-- ----------------------------
```

```
    `id`           int(11) NOT NULL AUTO_INCREMENT COMMENT '角色菜单关联表自增ID',
    `role_id`      int(16) NOT NULL COMMENT '角色ID',
    `authority_id` int(11) NOT NULL COMMENT '权限菜单ID',
    PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 4 CHARACTER SET = utf8 COLLATE = utf8_general_ci COMMENT = '角

-- ----------------------------
-- Records of sys_role_authority
-- ----------------------------
BEGIN;
INSERT INTO `sys_role_authority`
VALUES (1, 1, 1),
       (2, 1, 2),
       (3, 1, 3);
COMMIT;

-- ----------------------------
-- Table structure for sys_user_role
-- ----------------------------
DROP TABLE IF EXISTS `sys_user_role`;
CREATE TABLE `sys_user_role`
(
    `id`      int(11) NOT NULL AUTO_INCREMENT,
    `role_id` int(16) NULL DEFAULT NULL COMMENT '角色ID',
    `user_id` int(18) NULL DEFAULT NULL COMMENT '用户ID',
    PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 2 CHARACTER SET = utf8 COLLATE = utf8_general_ci;

-- ----------------------------
-- Records of sys_user_role
-- ----------------------------
BEGIN;
INSERT INTO `sys_user_role`
VALUES (1, 1, 1);
COMMIT;

SET
FOREIGN_KEY_CHECKS = 1;
```

通过代码生成器生成entity、mapper、mapper文件

项目中引入生成器依赖

xml 复制代码

```xml
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-generator</artifactId>
    <version>3.5.3.1</version>
</dependency>
<dependency>
    <groupId>org.apache.velocity</groupId>
    <artifactId>velocity-engine-core</artifactId>
    <version>2.3</version>
</dependency>
```

编写生成代码

java 复制代码

```java
package com.example;

import com.baomidou.mybatisplus.generator.FastAutoGenerator;
import com.baomidou.mybatisplus.generator.config.DataSourceConfig;

import java.util.Arrays;
import java.util.Collections;
import java.util.List;

/**
 * 代码生成测试
 *
 * @author vains
 */
public class CodeGeneratorTest {

    private static final DataSourceConfig.Builder DATA_SOURCE_CONFIG = new DataSourceConfig.Buil

    public static void main(String[] args) {
        FastAutoGenerator.create(DATA_SOURCE_CONFIG)
                // 全局配置
                .globalConfig((scanner, builder) -> builder.author(scanner.apply("请输入作者名称？
```

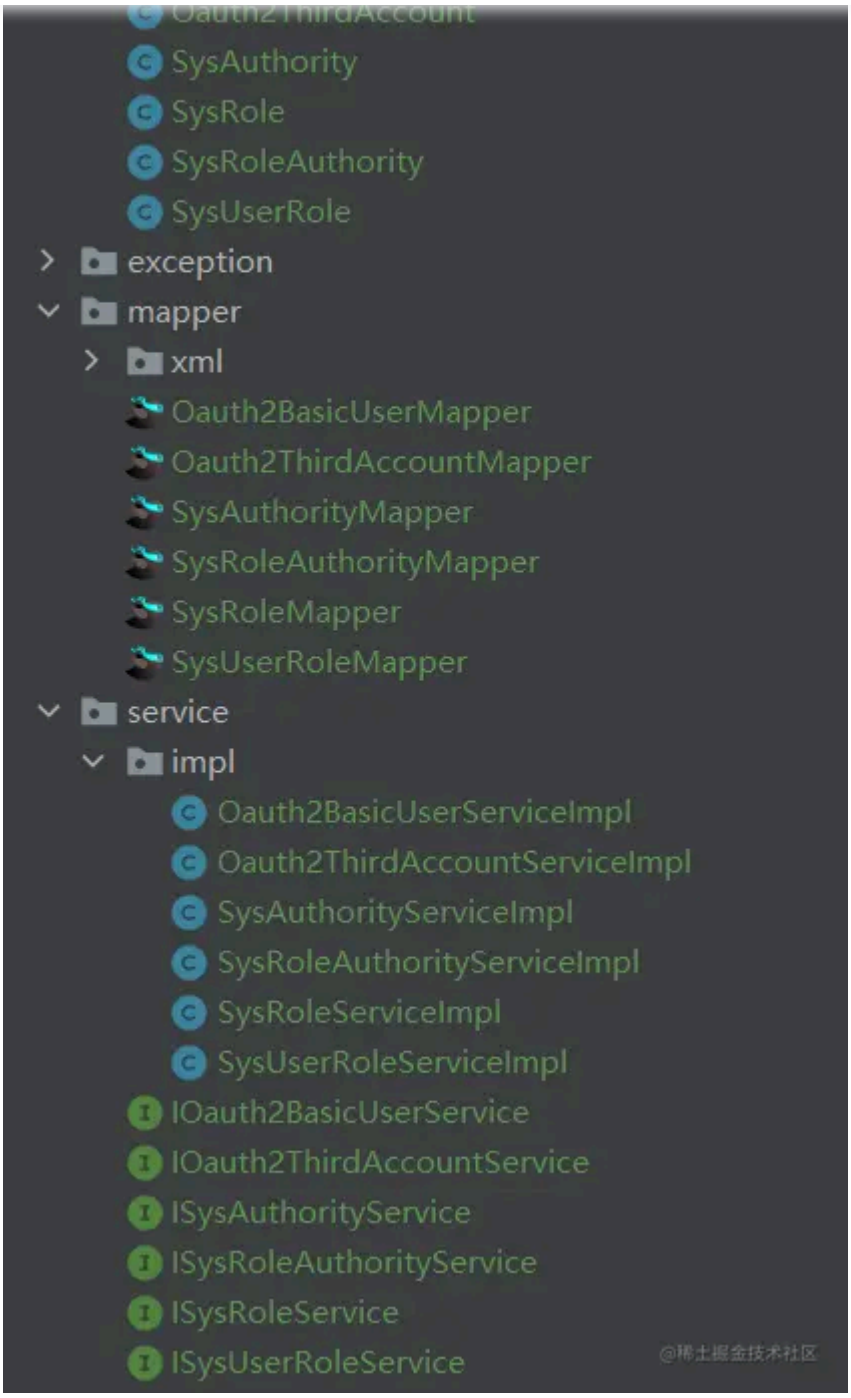
```
                .strategyConfig((scanner, builder) -> builder.addInclude(getTables(scanner.apply
                        .controllerBuilder().enableRestStyle().enableHyphenStyle()
                        .entityBuilder().enableLombok()
//                        .addTableFills(
//                                new Column("create_time", FieldFill.INSERT)
//                        )
                        .build())
                /*
                    模板引擎配置，默认 Velocity 可选模板引擎 Beetl 或 Freemarker
                   .templateEngine(new BeetlTemplateEngine())
                   .templateEngine(new FreemarkerTemplateEngine())
                 */
                .execute();
    }

    protected static List<String> getTables(String tables) {
        return "all".equals(tables) ? Collections.emptyList() : Arrays.asList(tables.split(","))
    }

}
```

说明

生成后代码会在D盘根目录下，按照控制台提示输入包名、作者和表名就会在d盘根目录生成对应的持久层代码，生成的代码就不贴出来了，太长了，后边会贴出核心代码；结构如下



实现自定义的UserDetailsService

`Oauth2BasicUser` 实体类实现 `UserDetails` 接口

实现该接口的原因有两个，一是可以用自己的用户属性替换框架默认的用户属性，二是因为 `UserDetailsService` 的 `loadUserByUsername` 方法返回的类型只能是 `UserDetails` 及其子类。

java 复制代码

```java
package com.example.entity;

import com.baomidou.mybatisplus.annotation.*;

import java.io.Serial;
import java.io.Serializable;
import java.time.LocalDateTime;
import java.util.Collection;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import com.fasterxml.jackson.databind.annotation.JsonSerialize;
import lombok.Getter;
import lombok.Setter;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

/**
 * <p>
 * 基础用户信息表
 * </p>
 *
 * @author vains
 */
@Getter
@Setter
@JsonSerialize
@TableName("oauth2_basic_user")
@JsonIgnoreProperties(ignoreUnknown = true)
public class Oauth2BasicUser implements UserDetails, Serializable {

    @Serial
    private static final long serialVersionUID = 1L;

    /**
     * 自增id
     */
    @TableId(value = "id", type = IdType.AUTO)
    private Integer id;

    /**
     * 用户名、昵称
     */
    private String name;

```

```
    private String account;

    /**
     * 密码
     */
    private String password;

    /**
     * 手机号
     */
    private String mobile;

    /**
     * 邮箱
     */
    private String email;

    /**
     * 头像地址
     */
    private String avatarUrl;

    /**
     * 是否已删除
     */
    private Boolean deleted;

    /**
     * 用户来源
     */
    private String sourceFrom;

    /**
     * 创建时间
     */
    @TableField(fill = FieldFill.INSERT)
    private LocalDateTime createTime;

    /**
     * 修改时间
     */
    @TableField(fill = FieldFill.INSERT_UPDATE)
    private LocalDateTime updateTime;

    /**
     * 权限信息
```

```java
    private Collection<? extends GrantedAuthority> authorities;

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return this.authorities;
    }

    @Override
    public String getUsername() {
        return this.account;
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return !this.deleted;
    }
}
```

Oauth2BasicUserServiceImpl 实现 UserDetailsService 接口

java 复制代码

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

import com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;
import com.baomidou.mybatisplus.core.toolkit.ObjectUtils;
import com.baomidou.mybatisplus.core.toolkit.Wrappers;
import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import com.example.entity.Oauth2BasicUser;
```

```
import com.example.mapper.Oauth2BasicUserMapper;
import com.example.mapper.SysAuthorityMapper;
import com.example.mapper.SysRoleAuthorityMapper;
import com.example.mapper.SysUserRoleMapper;
import com.example.service.IOauth2BasicUserService;
import lombok.RequiredArgsConstructor;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import java.util.Collections;
import java.util.List;
import java.util.Optional;
import java.util.Set;
import java.util.stream.Collectors;

/**
 * <p>
 * 基础用户信息表 服务实现类
 * </p>
 *
 * @author vains
 */
@Service
@RequiredArgsConstructor
public class Oauth2BasicUserServiceImpl extends ServiceImpl<Oauth2BasicUserMapper, Oauth2BasicUs

    private final SysUserRoleMapper sysUserRoleMapper;

    private final SysAuthorityMapper sysAuthorityMapper;

    private final SysRoleAuthorityMapper sysRoleAuthorityMapper;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        // 在Security中“username”就代表了用户登录时输入的账号，在重写该方法时它可以代表以下内容：账号、
        // “username”在数据库中不一定非要是一样的列，它可以是手机号、邮箱，也可以都是，最主要的目的就是
        // 通过传入的账号信息查询对应的用户信息
        LambdaQueryWrapper<Oauth2BasicUser> wrapper = Wrappers.lambdaQuery(Oauth2BasicUser.class
                .or(o -> o.eq(Oauth2BasicUser::getEmail, username))
                .or(o -> o.eq(Oauth2BasicUser::getMobile, username))
                .or(o -> o.eq(Oauth2BasicUser::getAccount, username));
        Oauth2BasicUser basicUser = baseMapper.selectOne(wrapper);
        if (basicUser == null) {
```

```
        // 通过用户角色关联表查询对应的角色
        List<SysUserRole> userRoles = sysUserRoleMapper.selectList(Wrappers.lambdaQuery(SysUserR
        List<Integer> rolesId = Optional.ofNullable(userRoles).orElse(Collections.emptyList()).s
        if (ObjectUtils.isEmpty(rolesId)) {
            return basicUser;
        }
        // 通过角色菜单关联表查出对应的菜单
        List<SysRoleAuthority> roleMenus = sysRoleAuthorityMapper.selectList(Wrappers.lambdaQuer
        List<Integer> menusId = Optional.ofNullable(roleMenus).orElse(Collections.emptyList()).s
        if (ObjectUtils.isEmpty(menusId)) {
            return basicUser;
        }

        // 根据菜单ID查出菜单
        List<SysAuthority> menus = sysAuthorityMapper.selectBatchIds(menusId);
        Set<SimpleGrantedAuthority> authorities = Optional.ofNullable(menus).orElse(Collections.
        basicUser.setAuthorities(authorities);
        return basicUser;
    }
}
```

编写MybatisPlus配置类

java 复制代码

```java
package com.example.config;

import com.baomidou.mybatisplus.annotation.DbType;
import com.baomidou.mybatisplus.core.handlers.MetaObjectHandler;
import com.baomidou.mybatisplus.extension.plugins.MybatisPlusInterceptor;
import com.baomidou.mybatisplus.extension.plugins.inner.PaginationInnerInterceptor;
import org.apache.ibatis.reflection.MetaObject;
import org.mybatis.spring.annotation.MapperScan;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.time.LocalDateTime;

/**
 * MybatisPlus分页插件
 *
 * @author vains
 */
```

```
public class MybatisPlusConfig {

    /**
     * 新的分页插件,一缓和二缓遵循mybatis的规则,
     * 需要设置 MybatisConfiguration#useDeprecatedExecutor = false
     * 避免缓存出现问题(该属性会在旧插件移除后一同移除)
     */
    @Bean
    public MybatisPlusInterceptor mybatisPlusInterceptor() {
        MybatisPlusInterceptor interceptor = new MybatisPlusInterceptor();
        interceptor.addInnerInterceptor(new PaginationInnerInterceptor(DbType.MYSQL));
        return interceptor;
    }

    /**
     * 这里对应的是实体类中的`@TableField(fill = FieldFill.INSERT_UPDATE)`注解
     * fill的值可以是INSERT、UPDATE和INSERT_UPDATE
     * INSERT：插入时填充字段
     * UPDATE：修改时填充字段
     * INSERT_UPDATE：插入与修改时都触发
     */
    @Bean
    public MetaObjectHandler metaObjectHandler() {
        return new MetaObjectHandler() {

            @Override
            public void insertFill(MetaObject metaObject) {
                // 添加自动填充逻辑
                this.strictInsertFill(metaObject, "createTime", LocalDateTime::now, LocalDateTim
                this.strictInsertFill(metaObject, "updateTime", LocalDateTime::now, LocalDateTim
            }

            @Override
            public void updateFill(MetaObject metaObject) {
                // 修改自动填充逻辑
                this.strictUpdateFill(metaObject, "updateTime", LocalDateTime::now, LocalDateTim
            }

        };
    }

}
```

application.yml中添加配置

```
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/authorization-example?serverTimezone=UTC&userUnicode=true&c
    username: root
    password: root

# Mybatis-Plus 配置
mybatis-plus:
  # 扫描mapper文件
  mapper-locations:
    - classpath:com/vains/mapper/xml/*Mapper.xml
```

移除 `AuthorizationConfig` 的用户配置

完整代码如下

java 复制代码

```java
package com.example.config;

import com.example.authorization.device.DeviceClientAuthenticationConverter;
import com.example.authorization.device.DeviceClientAuthenticationProvider;
import com.example.authorization.sms.SmsCaptchaGrantAuthenticationConverter;
import com.example.authorization.sms.SmsCaptchaGrantAuthenticationProvider;
import com.example.constant.SecurityConstants;
import com.example.util.SecurityUtils;
import com.nimbusds.jose.jwk.JWKSet;
import com.nimbusds.jose.jwk.RSAKey;
import com.nimbusds.jose.jwk.source.ImmutableJWKSet;
import com.nimbusds.jose.jwk.source.JWKSource;
import com.nimbusds.jose.proc.SecurityContext;
import lombok.SneakyThrows;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.MediaType;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.security.access.annotation.Secured;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.Customizer;
import org.springframework.security.config.annotation.authentication.configuration.Authenticatio
import org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
```

```
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
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.jwt.JwtClaimsSet;
import org.springframework.security.oauth2.jwt.JwtDecoder;
import org.springframework.security.oauth2.server.authorization.JdbcOAuth2AuthorizationConsentSe
import org.springframework.security.oauth2.server.authorization.JdbcOAuth2AuthorizationService;
import org.springframework.security.oauth2.server.authorization.OAuth2AuthorizationConsentServic
import org.springframework.security.oauth2.server.authorization.OAuth2AuthorizationService;
import org.springframework.security.oauth2.server.authorization.client.JdbcRegisteredClientRepos
import org.springframework.security.oauth2.server.authorization.client.RegisteredClient;
import org.springframework.security.oauth2.server.authorization.client.RegisteredClientRepositor
import org.springframework.security.oauth2.server.authorization.config.annotation.web.configurat
import org.springframework.security.oauth2.server.authorization.config.annotation.web.configurer
import org.springframework.security.oauth2.server.authorization.settings.AuthorizationServerSett
import org.springframework.security.oauth2.server.authorization.settings.ClientSettings;
import org.springframework.security.oauth2.server.authorization.token.JwtEncodingContext;
import org.springframework.security.oauth2.server.authorization.token.OAuth2TokenCustomizer;
import org.springframework.security.oauth2.server.authorization.token.OAuth2TokenGenerator;
import org.springframework.security.oauth2.server.resource.authentication.JwtAuthenticationConve
import org.springframework.security.oauth2.server.resource.authentication.JwtGrantedAuthoritiesC
import org.springframework.security.web.DefaultSecurityFilterChain;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.LoginUrlAuthenticationEntryPoint;
import org.springframework.security.web.util.matcher.MediaTypeRequestMatcher;

import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import java.util.*;
import java.util.stream.Collectors;

/**
 * 认证配置
 * {@link EnableMethodSecurity} 开启全局方法认证，启用JSR250注解支持，启用注解 {@link Secured} 支持，
 * 在Spring Security 6.0版本中将@Configuration注解从@EnableWebSecurity，@EnableMethodSecurity，@E
 * 和 @EnableGlobalAuthentication 中移除，使用这些注解需手动添加 @Configuration 注解
 * {@link EnableWebSecurity} 注解有两个作用:
 * 1. 加载了WebSecurityConfiguration配置类，配置安全认证策略。
 * 2. 加载了AuthenticationConfiguration，配置了认证信息。
 *
 * @author vains
 */
```

```
77  public class AuthorizationConfig {
78
79      private static final String CUSTOM_CONSENT_PAGE_URI = "/oauth2/consent";
80
81      /**
82       * 配置端点的过滤器链
83       *
84       * @param http spring security核心配置类
85       * @return 过滤器链
86       * @throws Exception 抛出
87       */
88      @Bean
89      public SecurityFilterChain authorizationServerSecurityFilterChain(HttpSecurity http,
90                                                                        RegisteredClientRepository
91                                                                        AuthorizationServerSetting
92          // 配置默认的设置，忽略认证端点的csrf校验
93          OAuth2AuthorizationServerConfiguration.applyDefaultSecurity(http);
94
95          // 新建设备码converter和provider
96          DeviceClientAuthenticationConverter deviceClientAuthenticationConverter =
97                  new DeviceClientAuthenticationConverter(
98                          authorizationServerSettings.getDeviceAuthorizationEndpoint());
99          DeviceClientAuthenticationProvider deviceClientAuthenticationProvider =
100                 new DeviceClientAuthenticationProvider(registeredClientRepository);
101
102
103         http.getConfigurer(OAuth2AuthorizationServerConfigurer.class)
104                 // 开启OpenID Connect 1.0协议相关端点
105                 .oidc(Customizer.withDefaults())
106                 // 设置自定义用户确认授权页
107                 .authorizationEndpoint(authorizationEndpoint -> authorizationEndpoint.consentPag
108                 // 设置设备码用户验证url(自定义用户验证页)
109                 .deviceAuthorizationEndpoint(deviceAuthorizationEndpoint ->
110                         deviceAuthorizationEndpoint.verificationUri("/activate")
111                 )
112                 // 设置验证设备码用户确认页面
113                 .deviceVerificationEndpoint(deviceVerificationEndpoint ->
114                         deviceVerificationEndpoint.consentPage(CUSTOM_CONSENT_PAGE_URI)
115                 )
116                 .clientAuthentication(clientAuthentication ->
117                         // 客户端认证添加设备码的converter和provider
118                         clientAuthentication
119                                 .authenticationConverter(deviceClientAuthenticationConverter)
120                                 .authenticationProvider(deviceClientAuthenticationProvider)
121                 );
122         http
```

```
                                new LoginUrlAuthenticationEntryPoint("/login"),
                                new MediaTypeRequestMatcher(MediaType.TEXT_HTML)
                        )
                )
                // 处理使用access token访问用户信息端点和客户端注册端点
                .oauth2ResourceServer((resourceServer) -> resourceServer
                        .jwt(Customizer.withDefaults()));

        // 自定义短信认证登录转换器
        SmsCaptchaGrantAuthenticationConverter converter = new SmsCaptchaGrantAuthenticationConv
        // 自定义短信认证登录认证提供
        SmsCaptchaGrantAuthenticationProvider provider = new SmsCaptchaGrantAuthenticationProvid
        http.getConfigurer(OAuth2AuthorizationServerConfigurer.class)
                // 让认证服务器元数据中有自定义的认证方式
                .authorizationServerMetadataEndpoint(metadata -> metadata.authorizationServerMet
                // 添加自定义grant_type——短信认证登录
                .tokenEndpoint(tokenEndpoint -> tokenEndpoint
                        .accessTokenRequestConverter(converter)
                        .authenticationProvider(provider));

        DefaultSecurityFilterChain build = http.build();

        // 从框架中获取provider中所需的bean
        OAuth2TokenGenerator<?> tokenGenerator = http.getSharedObject(OAuth2TokenGenerator.class
        AuthenticationManager authenticationManager = http.getSharedObject(AuthenticationManager
        OAuth2AuthorizationService authorizationService = http.getSharedObject(OAuth2Authorizati
        // 以上三个bean在build()方法之后调用是因为调用build方法时框架会尝试获取这些类，
        // 如果获取不到则初始化一个实例放入SharedObject中，所以要在build方法调用之后获取
        // 在通过set方法设置进provider中，但是如果在build方法之后调用authenticationProvider(provider
        // 框架会提示unsupported_grant_type，因为已经初始化完了，在添加就不会生效了
        provider.setTokenGenerator(tokenGenerator);
        provider.setAuthorizationService(authorizationService);
        provider.setAuthenticationManager(authenticationManager);

        return build;
    }

    /**
     * 配置认证相关的过滤器链
     *
     * @param http spring security核心配置类
     * @return 过滤器链
     * @throws Exception 抛出
     */
    @Bean
    public SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http) throws Exception {
```

```
                        .anyRequest().authenticated()
                )
                // 指定登录页面
                .formLogin(formLogin ->
                        formLogin.loginPage("/login")
                );
        // 添加BearerTokenAuthenticationFilter，将认证服务当做一个资源服务，解析请求头中的token
        http.oauth2ResourceServer((resourceServer) -> resourceServer
                .jwt(Customizer.withDefaults())
                .accessDeniedHandler(SecurityUtils::exceptionHandler)
                .authenticationEntryPoint(SecurityUtils::exceptionHandler)
        );

        return http.build();
    }

    /**
     * 自定义jwt，将权限信息放至jwt中
     *
     * @return OAuth2TokenCustomizer的实例
     */
    @Bean
    public OAuth2TokenCustomizer<JwtEncodingContext> oAuth2TokenCustomizer() {
        return context -> {
            // 检查登录用户信息是不是UserDetails，排除掉没有用户参与的流程
            if (context.getPrincipal().getPrincipal() instanceof UserDetails user) {
                // 获取申请的scopes
                Set<String> scopes = context.getAuthorizedScopes();
                // 获取用户的权限
                Collection<? extends GrantedAuthority> authorities = user.getAuthorities();
                // 提取权限并转为字符串
                Set<String> authoritySet = Optional.ofNullable(authorities).orElse(Collections.e
                        // 获取权限字符串
                        .map(GrantedAuthority::getAuthority)
                        // 去重
                        .collect(Collectors.toSet());

                // 合并scope与用户信息
                authoritySet.addAll(scopes);

                JwtClaimsSet.Builder claims = context.getClaims();
                // 将权限信息放入jwt的claims中（也可以生成一个以指定字符分割的字符串放入）
                claims.claim(SecurityConstants.AUTHORITIES_KEY, authoritySet);
                // 放入其它自定内容
                // 角色、头像...
            }
```

```
    /**
     * 自定义jwt解析器，设置解析出来的权限信息的前缀与在jwt中的key
     *
     * @return jwt解析器 JwtAuthenticationConverter
     */
    @Bean
    public JwtAuthenticationConverter jwtAuthenticationConverter() {
        JwtGrantedAuthoritiesConverter grantedAuthoritiesConverter = new JwtGrantedAuthoritiesCo
        // 设置解析权限信息的前缀，设置为空是去掉前缀
        grantedAuthoritiesConverter.setAuthorityPrefix("");
        // 设置权限信息在jwt claims中的key
        grantedAuthoritiesConverter.setAuthoritiesClaimName(SecurityConstants.AUTHORITIES_KEY);

        JwtAuthenticationConverter jwtAuthenticationConverter = new JwtAuthenticationConverter()
        jwtAuthenticationConverter.setJwtGrantedAuthoritiesConverter(grantedAuthoritiesConverter
        return jwtAuthenticationConverter;
    }


    /**
     * 将AuthenticationManager注入ioc中，其它需要使用地方可以直接从ioc中获取
     * @param authenticationConfiguration 导出认证配置
     * @return AuthenticationManager 认证管理器
     */
    @Bean
    @SneakyThrows
    public AuthenticationManager authenticationManager(AuthenticationConfiguration authenticatio
        return authenticationConfiguration.getAuthenticationManager();
    }

    /**
     * 配置密码解析器，使用BCrypt的方式对密码进行加密和验证
     *
     * @return BCryptPasswordEncoder
     */
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    /**
     * 配置客户端Repository
     *
     * @param jdbcTemplate    db 数据源信息
     * @param passwordEncoder 密码解析器
     * @return 基于数据库的repository
```

```
        RegisteredClient registeredClient = RegisteredClient.withId(UUID.randomUUID().toString()
                // 客户端id
                .clientId("messaging-client")
                // 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)
                // 客户端添加自定义认证
                .authorizationGrantType(new AuthorizationGrantType(SecurityConstants.GRANT_TYPE_
                // 授权码模式回调地址，oauth2.1已改为精准匹配，不能只设置域名，并且屏蔽了localhost，本
                .redirectUri("http://127.0.0.1:8080/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)
                // 自定scope
                .scope("message.read")
                .scope("message.write")
                // 客户端设置，设置用户需要确认授权
                .clientSettings(ClientSettings.builder().requireAuthorizationConsent(true).build
                .build();

        // 基于db存储客户端，还有一个基于内存的实现 InMemoryRegisteredClientRepository
        JdbcRegisteredClientRepository registeredClientRepository = new JdbcRegisteredClientRepo

        // 初始化客户端
        RegisteredClient repositoryByClientId = registeredClientRepository.findByClientId(regist
        if (repositoryByClientId == null) {
            registeredClientRepository.save(registeredClient);
        }
        // 设备码授权客户端
        RegisteredClient deviceClient = RegisteredClient.withId(UUID.randomUUID().toString())
                .clientId("device-message-client")
                // 公共客户端
                .clientAuthenticationMethod(ClientAuthenticationMethod.NONE)
                // 设备码授权
                .authorizationGrantType(AuthorizationGrantType.DEVICE_CODE)
                .authorizationGrantType(AuthorizationGrantType.REFRESH_TOKEN)
                // 自定scope
                .scope("message.read")
                .scope("message.write")
                .build();
```

```
        }

        // PKCE客户端
        RegisteredClient pkceClient = RegisteredClient.withId(UUID.randomUUID().toString())
                .clientId("pkce-message-client")
                // 公共客户端
                .clientAuthenticationMethod(ClientAuthenticationMethod.NONE)
                // 授权码模式，因为是扩展授权码流程，所以流程还是授权码的流程，改变的只是参数
                .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，本
                .redirectUri("http://127.0.0.1:8080/login/oauth2/code/messaging-client-oidc")
                .clientSettings(ClientSettings.builder().requireProofKey(Boolean.TRUE).build())
                // 自定scope
                .scope("message.read")
                .scope("message.write")
                .build();
        RegisteredClient findPkceClient = registeredClientRepository.findByClientId(pkceClient.g
        if (findPkceClient == null) {
            registeredClientRepository.save(pkceClient);
        }
        return registeredClientRepository;
    }

    /**
     * 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, Registered
        // 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 jdbcTempla
```

```

    /**
     * 配置jwk源，使用非对称加密，公开用于检索匹配指定选择器的JWK的方法
     *
     * @return JWKSource
     */
    @Bean
    public JWKSource<SecurityContext> jwkSource() {
        KeyPair keyPair = generateRsaKey();
        RSAPublicKey publicKey = (RSAPublicKey) keyPair.getPublic();
        RSAPrivateKey privateKey = (RSAPrivateKey) keyPair.getPrivate();
        RSAKey rsaKey = new RSAKey.Builder(publicKey)
                .privateKey(privateKey)
                .keyID(UUID.randomUUID().toString())
                .build();
        JWKSet jwkSet = new JWKSet(rsaKey);
        return new ImmutableJWKSet<>(jwkSet);
    }

    /**
     * 生成rsa密钥对，提供给jwk
     *
     * @return 密钥对
     */
    private static KeyPair generateRsaKey() {
        KeyPair keyPair;
        try {
            KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
            keyPairGenerator.initialize(2048);
            keyPair = keyPairGenerator.generateKeyPair();
        } catch (Exception ex) {
            throw new IllegalStateException(ex);
        }
        return keyPair;
    }

    /**
     * 配置jwt解析器
     *
     * @param jwkSource jwk源
     * @return JwtDecoder
     */
    @Bean
    public JwtDecoder jwtDecoder(JWKSource<SecurityContext> jwkSource) {
        return OAuth2AuthorizationServerConfiguration.jwtDecoder(jwkSource);
    }
```

```
     *
     * @return AuthorizationServerSettings
     */
    @Bean
    public AuthorizationServerSettings authorizationServerSettings() {
        return AuthorizationServerSettings.builder()
                /*
                    设置token签发地址(http(s)://{ip}:{port}/context-path, http(s)://domain.com/co
                    如果需要通过ip访问这里就是ip，如果是有域名映射就填域名，通过什么方式访问该服务这里就
                 */
                .issuer("http://192.168.120.33:8080")
                .build();
    }

}
```

到这里就完成了自定义的UserDetailsService

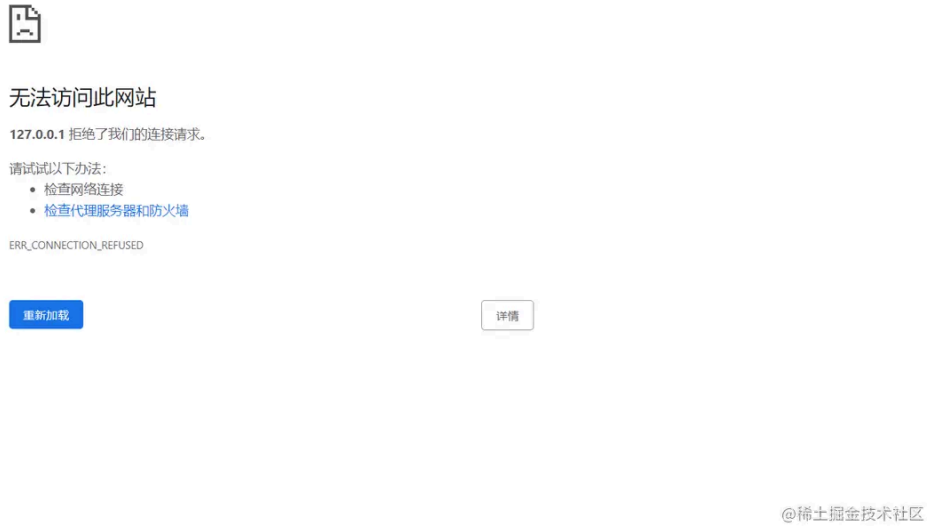
测试

访问/oauth2/authorize接口

http://127.0.0.1:8080/oauth2/authorize?response_type=code&client_id=pkce-message-client&redirect_uri=http%3A%2F%2F127.0.0.1%3A8000%2Flogin%2Foauth2%2Fcode%2Fmessaging-client-oidc&scope=message.read&code_challenge=kfis_wJYpmCAPO-Ap1Sc6GXyz_x2dhhMsm9FOA7eEWY&code_challenge_method=S256 重定向至登录页面

输入数据库中的手机号或者邮箱

登录成功后携带code重定向至回调地址



框架在什么时候调用自定义的UserDetailsService

在loadUserByUsername方法中打一个断点，查看请求执行经过的类，发现是在 `DaoAuthenticationProvider` 中调用的loadUserByUsername方法；所以在上边为了符合框架的规则去实现UserDetails与UserDetailsService接口，这样注入ioc中后不需要别的配置即可生效。如果看过我之前文章的读者可能就比较熟悉了，这一快儿的东西在之前的文章中也有提到

```
protected final UserDetails retrieveUser(String username, UsernamePasswordAuthenticationToken authenti
        throws AuthenticationException {
    prepareTimingAttackProtection();
    try {
        UserDetails loadedUser = this.getUserDetailsService().loadUserByUsername(username);
        if (loadedUser == null) {
            throw new InternalAuthenticationServiceException(
                    "UserDetailsService returned null, which is an interface contract violation");
        }
        return loadedUser;
    }
```

```
Debug: AuthorizationExampleApplication
Debugger  Console  Actuator
Frames
"http-nio-8080-exec-7"@8,768 in group "main": RUNNING
loadUserByUsername:61, Oauth2BasicUserServiceImpl (com.example.service.impl)
invoke0:-1, NativeMethodAccessorImpl (jdk.internal.reflect)
invoke:77, NativeMethodAccessorImpl (jdk.internal.reflect)
invoke:43, DelegatingMethodAccessorImpl (jdk.internal.reflect)
invoke:568, Method (java.lang.reflect)
invokeJoinpointUsingReflection:343, AopUtils (org.springframework.aop.support)
intercept:698, CglibAopProxy$DynamicAdvisedInterceptor (org.springframework.aop.framework)
loadUserByUsername:-1, Oauth2BasicUserServiceImpl$$SpringCGLIB$$0 (com.example.service.impl)
retrieveUser:103, DaoAuthenticationProvider (org.springframework.security.authentication.dao)
authenticate:133, AbstractUserDetailsAuthenticationProvider (org.springframework.security.authentication.dao)
authenticate:80, CaptchaAuthenticationProvider (com.example.authorization.captcha)
authenticate:182, ProviderManager (org.springframework.security.authentication)
Switch frames from anywhere in the IDE with Ctrl+Alt+向上箭头 and Ctrl+Alt+向下箭头
@稀土掘金技术社区
Git  Find  Debug  TODO  Problems  Profiler  Dependencies  Spring  Terminal  Build
```

代码已提交至码云authorization-example

如果有什么问题请在评论区指出，以防误人子弟

标签： Spring Boot Spring 话题： 我的技术写作成长之路

本文收录于以下专栏

评论 12

0 / 1000 发送

最热 最新

微醺冰淇凌 Java coder @家里蹲集团

大佬，RegisteredClient客户端、授权管理和授权确认服务能自己实现。现有提供的三个表还想加几个字段

5月前 点赞 4 ···

叹雪飞花 作者 ：都是可以自定义的，包括OAuth2AuthorizationService(授权管理服务)、RegisteredClientRepository(客户端管理)和OAuth2AuthorizationConsentService(授权确认服务)，你可以自己编写一个子类实现他们，然后定义自己的逻辑

5月前 点赞 回复 ···

微醺冰淇凌 回复 叹雪飞花 作者 ：这些接口类倒是可以实现。但是我发现方法参数（实体类）还是原来的。实体类我怎么替换呢

5月前 点赞 回复 ···

查看全部 4 条回复

职业挖坑 JAVA

手机号密码密码登录点击授权后，你那有这个问题嘛，我下载的是master 分支代码 java.lang.IllegalArgumentException: The class with com.example.entity.Oauth2BasicUser and name of com.example.entity.Oauth2BasicUser is not in the allowlist. If you believe this class is safe to deserialize, please provide an explicit mapping using Jackson annotations or by providing a Mixin. If the serialization is only done by a truste...

8月前 点赞 6 ···

叹雪飞花 作者 ：我重新拉取一下代码试试，就是通过访问/oauth2/authorize接口重定向至登录页，然后在登录页面登录吧，流程是这样吗？

8月前 点赞 回复 ···

问题

8月前 点赞 回复 ...

查看全部 6 条回复

目录 收起

相关推荐

Spring Authorization Server入门 (十四) 联合身份认证添加微信登录

1.2k阅读 · 5点赞

Spring Authorization Server优化篇：持久化JWKSource，解决重启后无法解析AccessToken问题
1.1k阅读 · 5点赞

Spring Authorization Server入门 (九) Spring Boot引入Resource Server对接认证服务
1.8k阅读 · 13点赞

Spring Authorization Server常见问题解答(FAQ)
927阅读 · 1点赞

精选内容

关于防重，我是这么设计的
山间小僧 · 592阅读 · 3点赞

让页面动起来！Axios数据请求与加载动画完美结合教程
尘缘_ · 310阅读 · 0点赞

Express中间件全攻略：成为Node.js路由处理大师的秘诀
snakeshe1010 · 299阅读 · 0点赞

听说map删除元素占用竟然不会减少?
低飞的蜜蜂 · 284阅读 · 2点赞

SpringBoot3集成PostgreSQL
知了一笑 · 262阅读 · 0点赞

为你推荐

Spring Authorization Server入门 (二) Spring Boot整合Spring Authorization Server
叹雪飞花 9月前 6.8k 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
叹雪飞花 9月前 2.9k 18 4 Spring ...

SpringBoot3.x最简集成SpringDoc-OpenApi
叹雪飞花 4月前 2.3k 16 评论 后端 Spring ... Java

Spring Authorization Server入门 (九) Spring Boot引入Resource Server对接认证服务
叹雪飞花 9月前 1.8k 13 8 Spring Spring ...

Spring Authorization Server优化篇：添加Redis缓存支持和统一响应类
叹雪飞花 8月前 1.7k 6 12 Spring Spring ... 安全

Spring Authorization Server入门 (十五) 分离授权确认与设备码校验页面
叹雪飞花 7月前 1.6k 14 8 Spring ... Spring Vue.js

Spring Authorization Server入门 (十九) 基于Redis的Token、客户端信息和授权确认信...
叹雪飞花 4月前 1.2k 8 19 Spring ... 后端 Redis

Spring Authorization Server入门 (二十) 实现二维码扫码登录
叹雪飞花 2月前 922 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 759 9 12 Vue.js 安全 Spring ...