SpringSecurity从入门到精通

简介

Spring Security 是 Spring 家族中的一个安全管理框架。相比与另外一个安全框架**Shiro**，它提供了更丰富的功能，社区资源也比Shiro丰富。

一般来说中大型的项目都是使用**SpringSecurity** 来做安全框架。小项目有Shiro的比较多，因为相比与SpringSecurity，Shiro的上手更加的简单。

一般Web应用的需要进行**认证**和**授权**。

认证：验证当前访问系统的是不是本系统的用户，并且要确认具体是哪个用户

授权：经过认证后判断当前用户是否有权限进行某个操作

而认证和授权也是SpringSecurity作为安全框架的核心功能。

1. 快速入门

1.1 准备工作

技术版本：

```
sprinboot3
mybatisplus3
redis6+
jwt
mysql8+
springsecurity 6.2+
```

我们先要搭建一个SpringBoot工程

① 创建工程 添加依赖

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.2.5</version>
    <relativePath/>
</parent>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
```

```
</dependencies>
```

② 创建启动类

```
package cn.hxzy;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SecurityApplication {
    public static void main(String[] args) {
        SpringApplication.run(SecurityApplication.class,args);
    }
}
```

③ 创建Controller

```
package cn.hxzy.controller;


import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloController {

    @RequestMapping("/hello")
    public String hello(){
        return "hello";
    }
}
```

1.2 引入SpringSecurity

在SpringBoot项目中使用SpringSecurity我们只需要引入依赖即可实现入门案例。

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

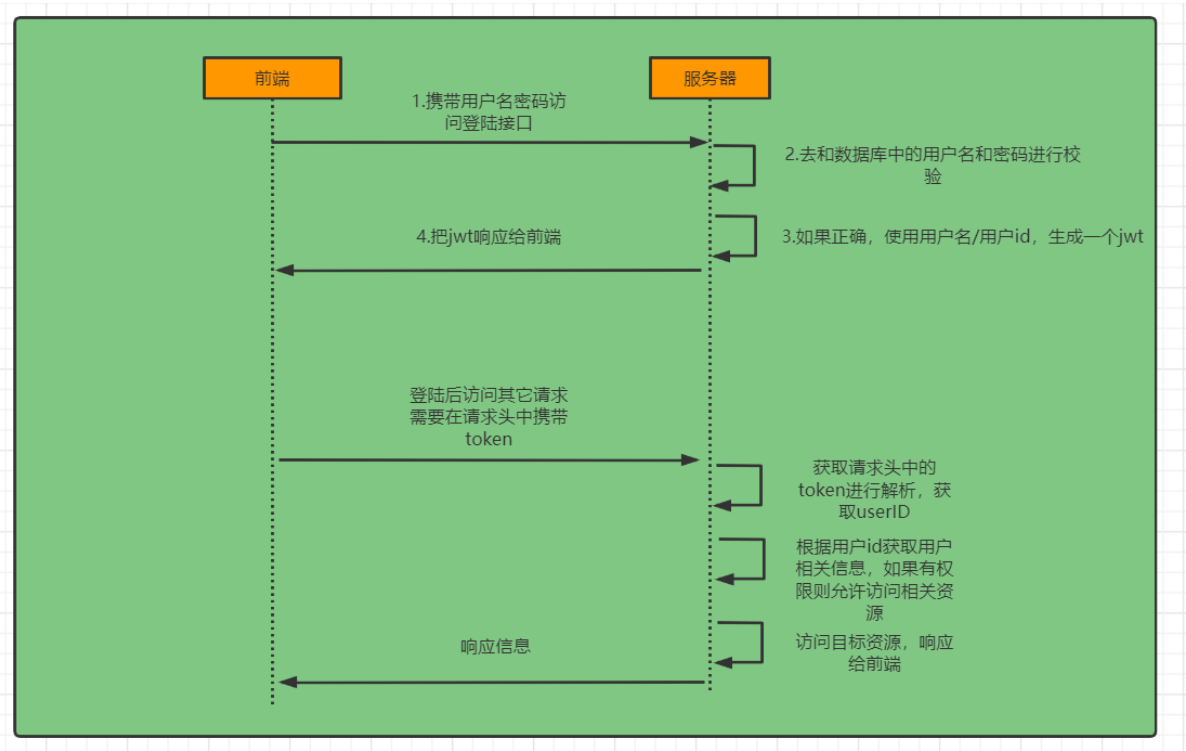
引入依赖后我们在尝试去访问之前的接口就会自动跳转到一个SpringSecurity的默认登陆页面，默认用户名是user,密码会输出在控制台。

必须登陆之后才能对接口进行访问。

退出：输入logout即可。

2. 认证

2.1 登录校验流程

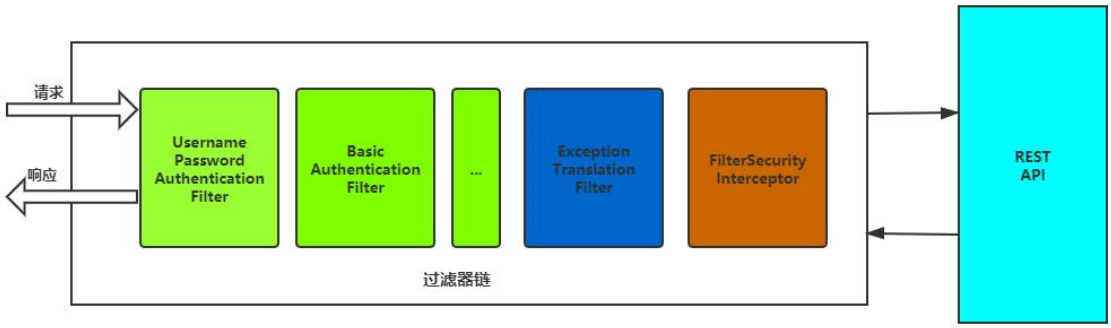


2.2 原理分析

想要知道如何实现自己的登陆流程就必须要先知道入门案例中SpringSecurity的流程。

2.2.1 SpringSecurity完整流程

SpringSecurity的原理其实就是一个过滤器链，内部包含了提供各种功能的过滤器。这里我们可以看看入门案例中的过滤器。



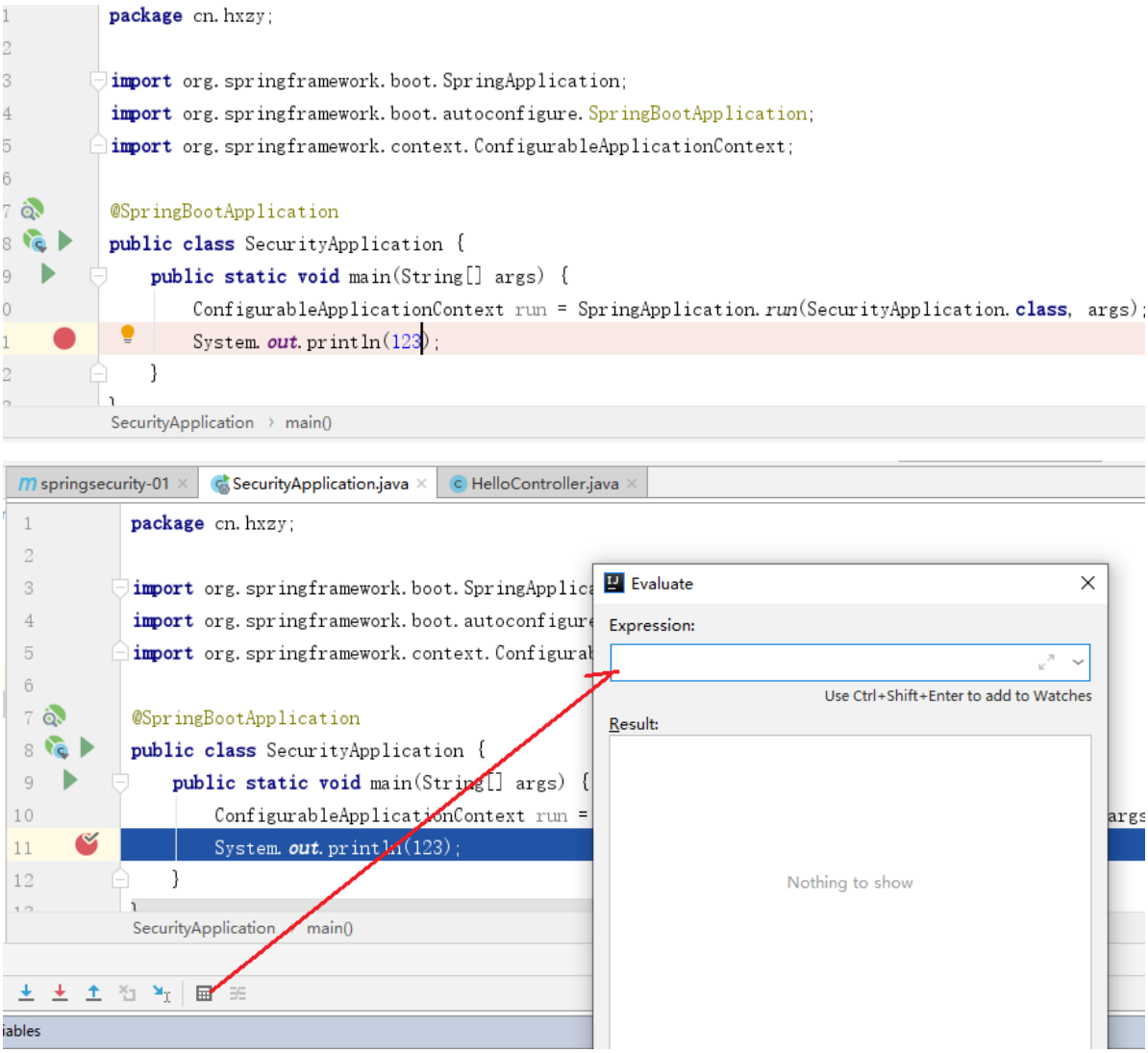
图中只展示了核心过滤器，其它的非核心过滤器并没有在图中展示。

UsernamePasswordAuthenticationFilter:负责处理我们在登陆页面填写了用户名密码后的登陆请求。入门案例的认证工作主要有它负责。

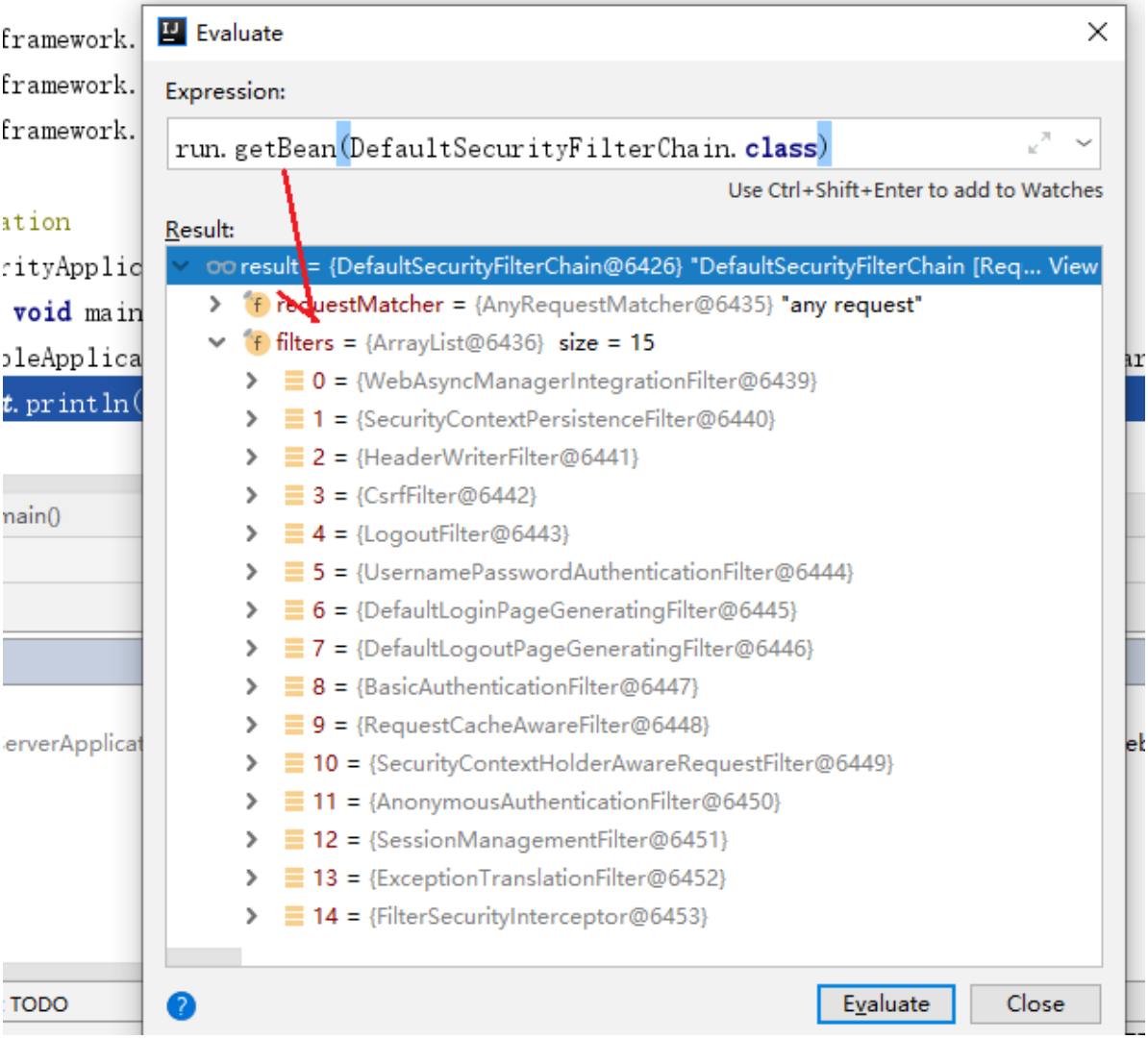
ExceptionTranslationFilter：处理过滤器链中抛出的任何AccessDeniedException和AuthenticationException 。

FilterSecurityInterceptor：负责权限校验的过滤器。通俗一点就是授权由它负责。（鉴权）

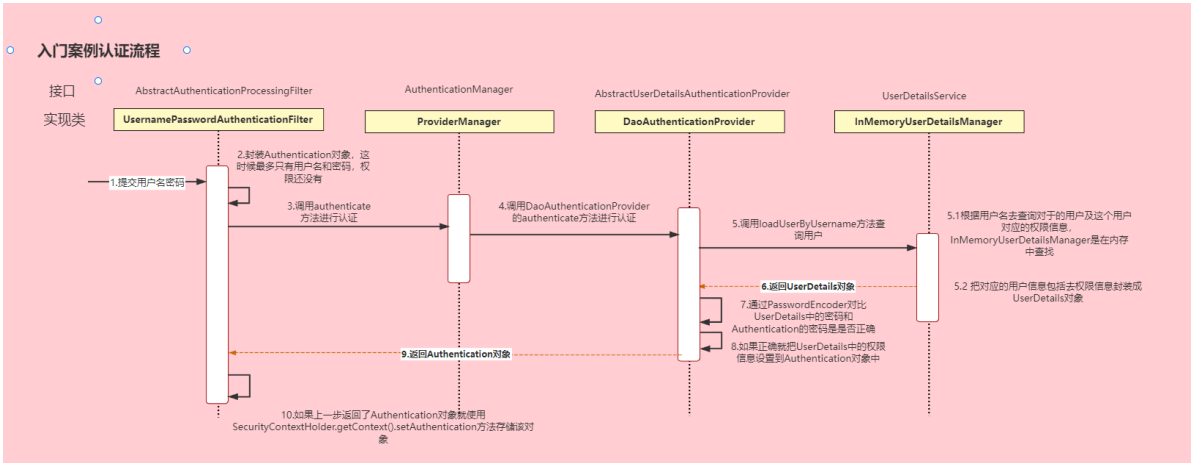
我们可以通过Debug查看当前系统中SpringSecurity过滤器链中有哪些过滤器及它们的顺序。



输入：run.getBean(DefaultSecurityFilterChain.class)，敲回车



2.2.2 认证流程详解



概念速查:

Authentication接口: 它的实现类，表示当前访问系统的用户，封装了用户相关信息。

AuthenticationManager接口：定义了认证Authentication的方法

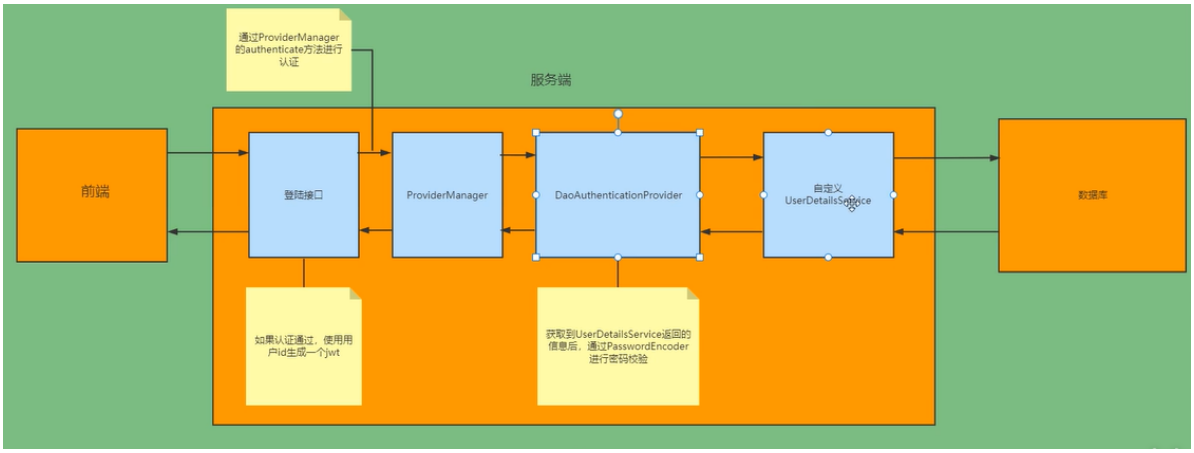
UserDetailsService接口：加载用户特定数据的核心接口。里面定义了一个根据用户名查询用户信息的方法。

UserDetails接口：提供核心用户信息。通过UserDetailsService根据用户名获取处理的用户信息要封装成UserDetails对象返回。然后将这些信息封装到Authentication对象中。

2.3 解决问题

2.3.1 思路分析

登录



①自定义登录接口

调用ProviderManager的方法进行认证 如果认证通过生成jwt

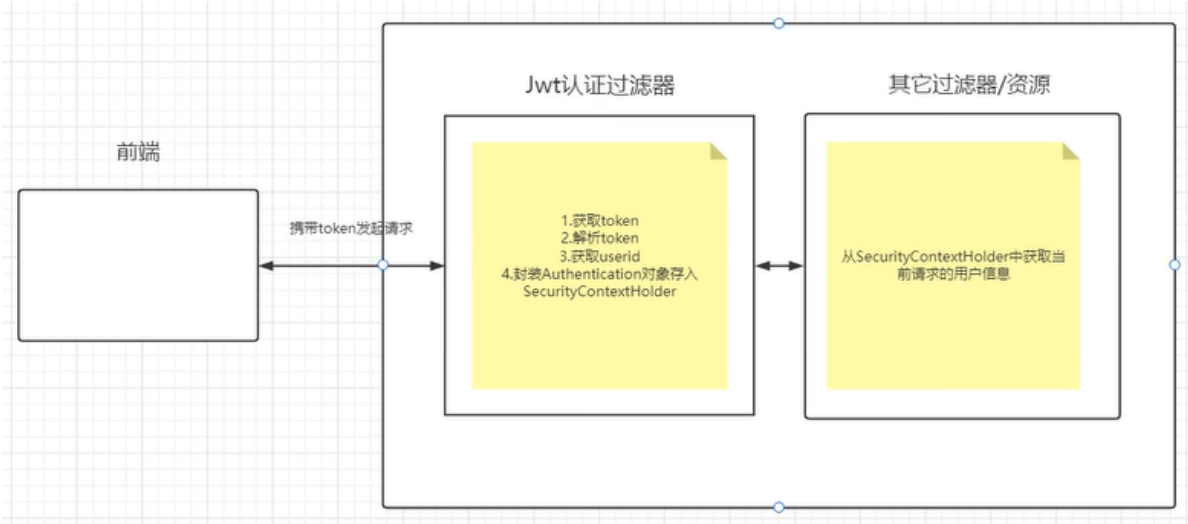
把用户信息存入redis中

②自定义UserDetailsService

在这个实现类中去查询数据库

校验：

思考：从JWT认证过滤器中获取到userid后怎么获取到完整的用户信息?



结论：如果认证通过，使用用户id生成一个jwt，然后用userid作为key，用户信息作为value存入redis，用户下一次访问的时候，到达JWT认证过滤器后，再去redis中就可以取到对应的用户信息（缓解一部分数据库的压力）

两种方案：

1.redis中存储jwt

2.不在redis中存储jwt，自解释

①定义Jwt认证过滤器

获取token

解析token获取其中的userid

从redis中获取用户信息（可选）

存入SecurityContextHolder

2.3.2 添加依赖及配置文件

```
<!--fastjson依赖-->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>1.2.33</version>
</dependency>

<!--jwt依赖-->
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.0</version>
</dependency>

<!--web依赖-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

```xml
<!--单元测试的坐标-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
</dependency>

<!--mybatisplus依赖-->
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-spring-boot3-starter</artifactId>
    <version>3.5.6</version>
</dependency>


<!--mysql驱动依赖-->
<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
</dependency>


<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid-spring-boot-starter</artifactId>
    <version>1.2.8</version>
</dependency>

<!--lombok依赖-->
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
</dependency>

<!--validation依赖-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>

<!--redis坐标-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>

<!--springdoc-openapi-->
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
    <version>2.1.0</version>
</dependency>

<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-api</artifactId>
    <version>2.1.0</version>
</dependency>
```

yml配置

```yaml
server:
  port: 8001
  #address: 127.0.0.1
#spring数据源配置
spring:
  application:
    name: token #项目名
  # 数据源
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/security_manager?
serverTimezone=GMT%2B8&useUnicode=true&useSSL=false&characterEncoding=utf8
    username: root
    password: root
    max-active: 20
    max-wait: 5000
    initial-size: 1
    filters: stat,log4j,wall
    validationQuery: SELECT 'x'   #验证连接 SQL心跳包
    enable: true
  # redis 配置
  redis:
    database: 0
    host: 127.0.0.1
    port: 6379
    #password: 123456
    lettuce:
      pool:
        #最大连接数
        max-active: 8
        #最大阻塞等待时间(负数表示没限制)
        max-wait: -1
        #最大空闲
        max-idle: 8
        #最小空闲
        min-idle: 0
      #连接超时时间
      timeout: 10000
  # jackson 配置
  jackson:
    date-format: yyyy-MM-dd HH:mm:ss
    time-zone: GMT+8
# mybatis-plus配置
mybatis-plus:
  global-config:
    db-config:
      logic-delete-field: deleted # 全局逻辑删除的实体字段名(since 3.3.0,配置后可以忽略
不配置步骤2)
      logic-delete-value: 1 # 逻辑已删除值(默认为 1)
      logic-not-delete-value: 0 # 逻辑未删除值(默认为 0)
  configuration:
    map-underscore-to-camel-case: true # 数据库下划线自动转驼峰标示关闭
    log-impl: org.apache.ibatis.logging.stdout.StdOutImpl #日志配置
  mapper-locations: classpath*:/mapper/**/*.xml
```

2.3.3 添加响应类

```
package cn.hxzy.common;

import lombok.Data;

import java.util.HashMap;
import java.util.Map;

@Data
public class R {

    private Boolean success;  //返回的成功或者失败的标识符
    private Integer code;  //返回的状态码
    private String message; //提示信息
    private Map<String, Object> data = new HashMap<String, Object>();  //数据


    //把构造方法私有
    private R() {}

    //成功的静态方法
    public static R ok(){
        R r=new R();
        r.setSuccess(true);
        r.setCode(ResultCode.SUCCESS);
        r.setMessage("成功");
        return r;
    }

    //失败的静态方法
    public static R error(){
        R r=new R();
        r.setSuccess(false);
        r.setCode(ResultCode.ERROR);
        r.setMessage("失败");
        return r;
    }

    //使用下面四个方法，方面以后使用链式编程
    // R.ok().success(true)
    //  r.message("ok).data("item",list)

    public R success(Boolean success){
        this.setSuccess(success);
        return this; //当前对象  R.success(true).message("操作成功").code().data()
    }

    public R message(String message){
        this.setMessage(message);
        return this;
    }

    public R code(Integer code){
        this.setCode(code);
```

```
        return this;
    }

    public R data(String key, Object value){
        this.data.put(key, value);
        return this;
    }

    public R data(Map<String, Object> map){
        this.setData(map);
        return this;
    }
}
```

接口

```
package cn.hxzy.common;

public interface ResultCode {
    Integer SUCCESS=20000;
    Integer ERROR=20001;
}
```

2.3.4 基于token的鉴权机制

1 什么是JWT?

官网：https://jwt.io/introduction

JSON Web Token令牌（JWT）是一种开放标准（RFC 7519)它定义了一种紧凑且自包含的方式，用于在各方之间作为JSON对象安全地传输信息。此信息可以验证和信任，因为它是经过数字签名的。JWT可以使用密钥（使用**HMAC**算法）或使用**RSA**或**ECDSA**的公钥/私钥对进行签名。

通俗的说：

JSON Web Token简称：JWT，也就是通过JSON形式作为Web应用的令牌，用于在各方之间安全地将信息作为JSON对象传输。在数据传输过程中还可以完成数据加密，签名等相关处理。

2 JWT的作用

JSON Web Token (JWT)是为了在网络应用环境间传递声明而执行的一种基于JSON的开放标准，它定义了一种紧凑的、自包含的方式，用于作为JSON对象在各方之间安全地传输信息。该信息可以被验证和信任，因为它是数字签名的。

3 JWT工作流程

Authorization (授权) : 这是使用JWT的最常见场景。一旦用户登录，后续每个请求都将包含JWT，允许用户访问该令牌允许的路由、服务和资源。

流程上是这样的：

- 用户使用用户名密码来请求服务器
- 服务器进行验证用户的信息
- 服务器通过验证发送给用户一个token
- 客户端存储token，并在每次请求时携带上这个token值
- 服务端验证token值，并返回数据

4 JWT长什么样?

JWT是由三段信息构成的，将这三段信息文本用`.`链接一起就构成了Jwt字符串。就像这样:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
```

5 JWT的构成

第一部分我们称它为头部（header)

第二部分我们称其为载荷（payload, 类似于飞机上承载的物品)

第三部分是签证（signature).

header

jwt的头部承载两部分信息：

- 声明类型，这里是jwt
- 声明加密的算法 通常直接使用 HMAC SHA256

完整的头部就像下面这样的JSON：

```
{
  'typ': 'JWT',
  'alg': 'HS256'
}
```

然后将头部进行base64加密，构成了第一部分

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
```

playload

载荷就是存放有效信息的地方。这个名字像是特指飞机上承载的货品，这些有效信息包含三个部分

- 标准中注册的声明
- 公共的声明
- 私有的声明

标准中注册的声明 (建议但不强制使用)：

- **iss**: jwt签发者
- **sub**: jwt所面向的用户
- **aud**: 接收jwt的一方
- **exp**: jwt的过期时间，这个过期时间必须要大于签发时间
- **nbf**: 定义在什么时间之前，该jwt都是不可用的.
- **iat**: jwt的签发时间
- **jti**: jwt的唯一身份标识，主要用来作为一次性token,从而回避重放攻击。

公共的声明：
公共的声明可以添加任何的信息，一般添加用户的相关信息或其他业务需要的必要信息.但不建议添加敏感信息，因为该部分在客户端可解密.

私有的声明：
私有声明是提供者和消费者所共同定义的声明，一般不建议存放敏感信息，因为base64是对称解密的，意味着该部分信息可以归类为明文信息。

定义一个payload: 用户信息

```
{
  "sub": "1234567890",
  "name": "mengshujun",
  "admin": true
}
```

然后将其进行base64加密，得到Jwt的第二部分。

```
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9
```

signature

jwt的第三部分是一个签证信息，这个签证信息由三部分组成：

- header (base64后的)
- payload (base64后的)
- secret (盐-密钥)

这个部分需要base64加密后的header和base64加密后的payload使用`.`连接组成的字符串，然后通过header中声明的加密方式进行加盐`secret`组合加密，然后就构成了jwt的第三部分。

```
// javascript
var encodedString = base64UrlEncode(header) + '.' + base64UrlEncode(payload);
var signature = HMACSHA256(encodedString, 'secret'); //
TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
```

将这三部分用`.`连接成一个完整的字符串,构成了最终的jwt:

注意：secret是保存在服务器端的，jwt的签发生成也是在服务器端的，secret就是用来进行jwt的签发和jwt的验证，所以，它就是你服务端的私钥，在任何场景都不应该流露出去。一旦客户端得知这个secret,那就意味着客户端是可以自我签发jwt了。

JwtUtils工具类

```
package cn.hxzy.utils;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.JwtBuilder;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;

import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
import java.util.Date;
import java.util.UUID;

/**
 * JWT工具类
 */
public class JwtUtils {

    //有效期为
    public static final Long JWT_TTL = 60 * 60 *1000L;// 60 * 60 *1000  一个小时
```

```
    //设置秘钥明文（盐）
    public static final String JWT_KEY = "qW21YIU&^%$";

    //生成令牌
    public static String getUUID(){
        String token = UUID.randomUUID().toString().replaceAll("-", "");
        return token;
    }


    /**
     * 生成jtw
     * @param subject token中要存放的数据（json格式） 用户数据
     * @param ttlMillis token超时时间
     * @return
     */

    public static String createJWT(String subject, Long ttlMillis) {
        JwtBuilder builder = getJwtBuilder(subject, ttlMillis, getUUID());// 设置
过期时间
        return builder.compact();
    }

    //生成jwt的业务逻辑代码
    private static JwtBuilder getJwtBuilder(String subject, Long ttlMillis,
String uuid) {

        SignatureAlgorithm signatureAlgorithm = SignatureAlgorithm.HS256;

        SecretKey secretKey = generalKey();

        long nowMillis = System.currentTimeMillis();//获取到系统当前的时间戳

        Date now = new Date(nowMillis);
        if(ttlMillis==null){
            ttlMillis=JwtUtil.JWT_TTL;
        }

        long expMillis = nowMillis + ttlMillis;

        Date expDate = new Date(expMillis);

        return Jwts.builder()
                .setId(uuid)          //唯一的ID
                .setSubject(subject) // 主题  可以是JSON数据
                .setIssuer("xx")     // 签发者
                .setIssuedAt(now)    // 签发时间
                .signWith(signatureAlgorithm, secretKey) //使用HS256对称加密算法签
名，第二个参数为秘钥
                .setExpiration(expDate);
    }

    /**
     * 创建token
     * @param id
     * @param subject
     * @param ttlMillis
```

```
     * @return
     */
    public static String createJWT(String id, String subject, Long ttlMillis) {
        JwtBuilder builder = getJwtBuilder(subject, ttlMillis, id);// 设置过期时间
        return builder.compact();
    }


    /**
     * 生成加密后的秘钥 secretKey
     * @return
     */
    public static SecretKey generalKey() {
        byte[] encodedKey = Base64.getDecoder().decode(JwtUtil.JWT_KEY);
        SecretKey key = new SecretKeySpec(encodedKey, 0, encodedKey.length,
"AES");
        return key;
    }

    /**
     * 解析jwt
     *
     * @param jwt
     * @return
     * @throws Exception
     */
    public static Claims parseJWT(String jwt) throws Exception {
        SecretKey secretKey = generalKey();
        return Jwts.parser()
                .setSigningKey(secretKey)
                .parseClaimsJws(jwt)
                .getBody();
    }


}
```

添加依赖

```
<dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.3.1</version>
</dependency>
```

2.3.5 认证的实现

1 配置数据库校验登录用户

从之前的分析我们可以知道，我们可以自定义一个UserDetailsService,让SpringSecurity使用我们的UserDetailsService。我们自己的UserDetailsService可以从数据库中查询用户名和密码。

我们先创建一个用户表， 建表语句如下：

```sql
CREATE TABLE `sys_user` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT COMMENT '主键',
  `user_name` VARCHAR(64) NOT NULL DEFAULT 'NULL' COMMENT '用户名',
  `nick_name` VARCHAR(64) NOT NULL DEFAULT 'NULL' COMMENT '昵称',
  `password` VARCHAR(64) NOT NULL DEFAULT 'NULL' COMMENT '密码',
  `status` CHAR(1) DEFAULT '0' COMMENT '账号状态（0正常 1停用）',
  `email` VARCHAR(64) DEFAULT NULL COMMENT '邮箱',
  `phonenumber` VARCHAR(32) DEFAULT NULL COMMENT '手机号',
  `sex` CHAR(1) DEFAULT NULL COMMENT '用户性别（0男，1女，2未知）',
  `avatar` VARCHAR(128) DEFAULT NULL COMMENT '头像',
  `user_type` CHAR(1) NOT NULL DEFAULT '1' COMMENT '用户类型（0管理员，1普通用户）',
  `create_by` BIGINT(20) DEFAULT NULL COMMENT '创建人的用户id',
  `create_time` DATETIME DEFAULT NULL COMMENT '创建时间',
  `update_by` BIGINT(20) DEFAULT NULL COMMENT '更新人',
  `update_time` DATETIME DEFAULT NULL COMMENT '更新时间',
  `deleted` INT(11) DEFAULT '0' COMMENT '删除标志（0代表未删除，1代表已删除）',
  PRIMARY KEY (`id`)
) ENGINE=INNODB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8mb4 COMMENT='用户表'
```

实体类

```java
package cn.hxzy.entity;

import com.baomidou.mybatisplus.annotation.TableId;
import com.baomidou.mybatisplus.annotation.TableName;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.io.Serializable;
import java.util.Date;


/**
 * 用户表(User)实体类
 *
 * @author mengshujun
 */
@Data
@AllArgsConstructor
@NoArgsConstructor
@TableName("sys_user")
public class User implements Serializable {
    private static final long serialVersionUID = -40356785423868312L;

    /**
    * 主键
    */
    @TableId
    private Long id;
    /**
    * 用户名
    */
    private String userName;
    /**
    * 昵称
    */
```

```
    private String nickName;
    /**
    * 密码
    */
    private String password;
    /**
    * 账号状态（0正常 1停用）
    */
    private String status;
    /**
    * 邮箱
    */
    private String email;
    /**
    * 手机号
    */
    private String phonenumber;
    /**
    * 用户性别（0男，1女，2未知）
    */
    private String sex;
    /**
    * 头像
    */
    private String avatar;
    /**
    * 用户类型（0管理员，1普通用户）
    */
    private String userType;
    /**
    * 创建人的用户id
    */
    private Long createBy;
    /**
    * 创建时间
    */
    private Date createTime;
    /**
    * 更新人
    */
    private Long updateBy;
    /**
    * 更新时间
    */
    private Date updateTime;

     /**
     * 删除标志（0代表未删除，1代表已删除）
     */
    private Integer deleted;
}
```

定义Mapper接口

```
public interface UserMapper extends BaseMapper<User> {
}
```

配置Mapper扫描

```
@MapperScan("cn.hxzy.mapper")
```

测试MP是否能正常使用

```
@SpringBootTest
public class MapperTest {

    @Autowired
    private UserMapper userMapper;

    @Test
    public void testUserMapper(){
        List<User> users = userMapper.selectList(null);
        System.out.println(users);
    }
}
```

核心代码实现

因为UserDetailsService方法的返回值是UserDetails类型，所以需要定义一个类，实现该接口，把用户信息封装在其中。

```
package cn.hxzy.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import java.util.Collection;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class LoginUser implements UserDetails {

    private User user;

    //用来返回权限信息
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return null;
    }

    //获取密码
    @Override
    public String getPassword() {
        return user.getPassword();
    }
```

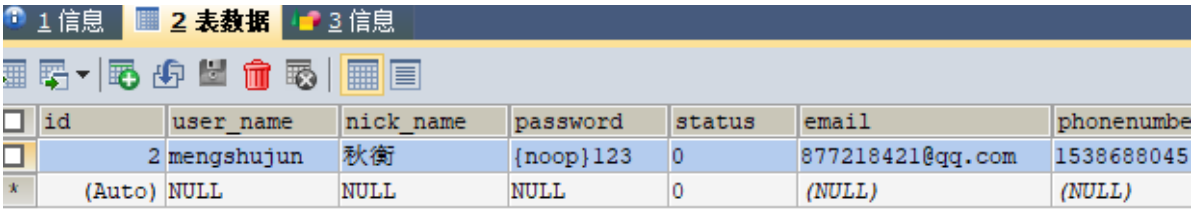
```
    //获取用户名
    @Override
    public String getUsername() {
        return user.getUserName();
    }

    //判断账号是否未过期
    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    //判断账号是否没有锁定
    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    //判断账号是否没有超时
    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    //判断账号是否可用
    @Override
    public boolean isEnabled() {
        return true;
    }
}
```

创建一个类实现UserDetailsService接口，重写其中的方法。使用用户名从数据库中查询用户信息

```
package cn.hxzy.service.impl;

import cn.hxzy.entity.LoginUser;
import cn.hxzy.entity.User;
import cn.hxzy.mapper.UserMapper;
import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
import org.springframework.beans.factory.annotation.Autowired;
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.Objects;

@Service
public class UserDetailsServiceImpl implements UserDetailsService {
    @Autowired
    private UserMapper userMapper;

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
```

```
        //查询用户信息
        QueryWrapper<User> queryWrapper=new QueryWrapper<>();
        queryWrapper.eq("user_name",username);
        User user = userMapper.selectOne(queryWrapper);
        //如果没有查询到用户，就抛出异常
        if(Objects.isNull(user)){
            throw  new RuntimeException("用户名或密码错误");
        }
        //TODO 查询用户对应的权限信息

        //如果有，把数据封装成UserDetails对象返回
        return new LoginUser(user);
    }
}
```

注意：如果要测试，需要往用户表中写入用户数据，并且如果你想让用户的密码是明文存储，需要在密码前加{noop}。例如



id	user_name	nick_name	password	status	email	phonenumbe
2	mengshujun	秋衡	{noop}123	0	877218421@qq.com	1538688045
(Auto)	NULL	NULL	NULL	0	(NULL)	(NULL)

这样登陆的时候就可以用mengshujun作为用户名，123作为密码来登陆了。

2 密码加密存储

实际项目中我们不会把密码明文存储在数据库中。

默认使用的PasswordEncoder要求数据库中的密码格式为：{id}password 。它会根据id去判断密码的加密方式。但是我们一般不会采用这种方式。所以就需要替换PasswordEncoder。

我们一般使用SpringSecurity为我们提供的BCryptPasswordEncoder。

我们只需要使用把BCryptPasswordEncoder对象注入Spring容器中，SpringSecurity就会使用该PasswordEncoder来进行密码校验。

我们可以定义一个SpringSecurity的配置类，SpringSecurity要求这个配置类要继承WebSecurityConfigurerAdapter。

```
package cn.hxzy.config;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConf
igurerAdapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    //创建BCryptPasswordEncoder注入容器
```

```
    @Bean
    public PasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder();
    }

}
```

测试：

```
package cn.hxzy;

import cn.hxzy.entity.User;
import cn.hxzy.mapper.UserMapper;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.security.crypto.password.PasswordEncoder;

import java.util.List;

@SpringBootTest
public class SpringsecurityO1ApplicationTests {


    @Autowired
    private PasswordEncoder passwordEncoder;


    @Test
    public void testPassWord(){
        //随机的盐
        String encode1 = passwordEncoder.encode("123");
        String encode2 = passwordEncoder.encode("123");
        System.out.println(encode1);
        System.out.println(encode2);

    }

}
```

相同的明文，它内部使用了随机的盐，导致产生不同的密文。为了安全性

比较明文与加密后的密文

```
@Test
public void test1(){
    boolean bool = passwordEncoder.matches("123",
"$2a$10$OPRsPDxlBCOp6bQd3oyzq.WFA4BayXfdVzhrlaRRIaSR.DbAYvcli");
    System.out.println(bool);
}
```

3 自定义登陆接口

接下我们需要自定义登陆接口，然后让SpringSecurity对这个接口放行,让用户访问这个接口的时候不用登录也能访问。

在接口中我们通过AuthenticationManager的authenticate方法来进行用户认证,所以需要在SecurityConfig中配置把AuthenticationManager注入容器。

认证成功的话要生成一个jwt，放入响应中返回。并且为了让用户下回请求时能通过jwt识别出具体的是哪个用户，我们需要把用户信息存入redis，可以把用户id作为key。

```
package cn.hxzy.controller;

import cn.hxzy.common.R;
import cn.hxzy.domain.User;
import cn.hxzy.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.util.StringUtils;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/user")
public class UserController {

    @Autowired
    private UserService userService;

    @PostMapping("/login")
    public R login(@RequestBody User user) {
       String jwt=  userService.login(user);
       if(StringUtils.hasLength(jwt)){
           return R.ok().message("登陆成功").data("token",jwt);
       }
       return R.error().message("登陆失败");
    }
}
```

业务逻辑层:

```
package cn.hxzy.service;

import cn.hxzy.domain.User;
import com.baomidou.mybatisplus.extension.service.IService;

public interface UserService extends IService<User> {
    //登录
    String login(User user);
}
```

使用AuthenticationManager对象

```java
package cn.hxzy.config;


import cn.hxzy.filter.JwtAuthenticationTokenFilter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.authentication.ProviderManager;
import
org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import
org.springframework.security.config.annotation.authentication.builders.Authentic
ationManagerBuilder;
import
org.springframework.security.config.annotation.authentication.configuration.Auth
enticationConfiguration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecuri
ty;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFi
lter;


@EnableWebSecurity
@Configuration
public class SecurityConfig {

    //创建BCryptPasswordEncoder注入容器
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }


    /**
     * 登录时需要调用AuthenticationManager.authenticate执行一次校验
     *
     */
    @Bean
    public AuthenticationManager
authenticationManager(AuthenticationConfiguration config) throws Exception {
        return config.getAuthenticationManager();
    }
}
```

在AuthenticationManager中找到authenticate方法，查看参数authentication，在参数的类型上按住Ctrl+Alt点击左键，可以查看到这个接口的全部实现类列表，这里使用UsernamePasswordAuthenticationToken实现类

4 SecurityConfig配置文件

4.1 HttpSecurity参数说明

`SecurityFilterChain`：一个表示安全过滤器链的对象

- `http.antMatchers(...).permitAll()` 通过 `antMatchers` 方法，你可以指定哪些请求路径不需要进行身份验证。
- `http.authorizeRequests()` 可以配置请求的授权规则。 例如，`.anyRequest().authenticated()` 表示任何请求都需要经过身份验证。
- `http.requestMatchers` 表示某个请求不需要进行身份校验，permitAll 随意访问。
- `http.httpBasic()` 配置基本的 HTTP 身份验证。
- `http.csrf()` 通过 `csrf` 方法配置 CSRF 保护。
- `http.sessionManagement()` 不会创建会话。这意味着每个请求都是独立的，不依赖于之前的请求。适用于 RESTful 风格的应用。

4.2 具体配置

```
package cn.hxzy.config;


import cn.hxzy.filter.JwtAuthenticationTokenFilter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.authentication.ProviderManager;
import
org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import
org.springframework.security.config.annotation.authentication.builders.Authentic
ationManagerBuilder;
import
org.springframework.security.config.annotation.authentication.configuration.Auth
enticationConfiguration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecuri
ty;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
```

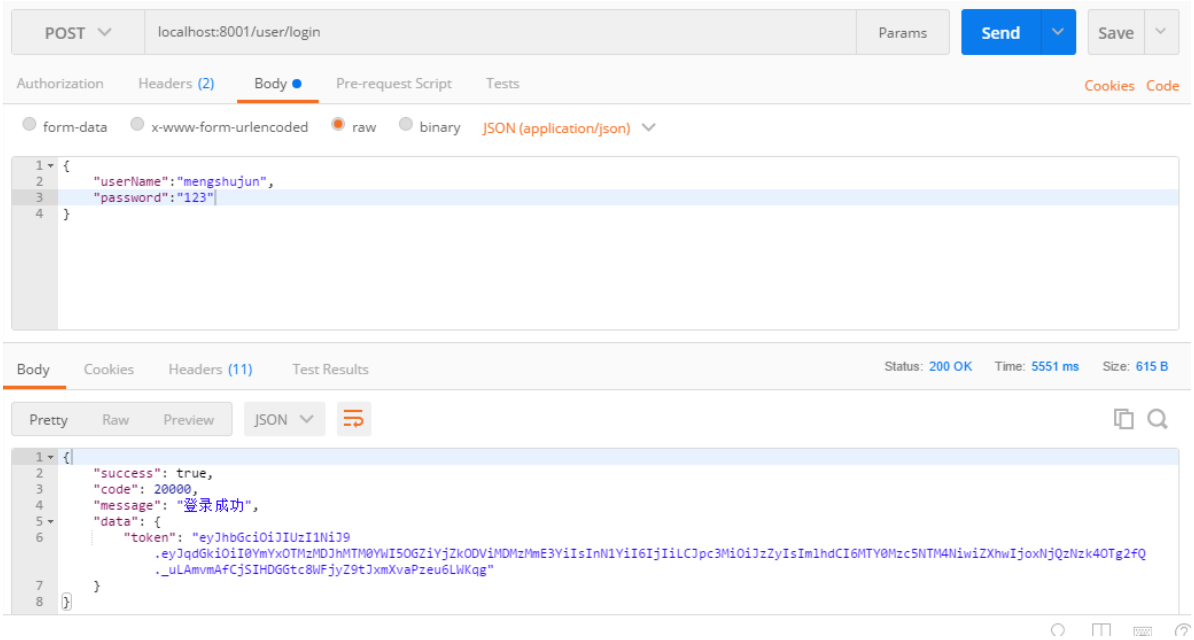
```
import org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFi
lter;


@EnableWebSecurity
@Configuration
public class SecurityConfig {


    @Autowired
    private JwtAuthenticationTokenFilter jwtAuthenticationTokenFilter;

    //创建BCryptPasswordEncoder注入容器
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        //配置关闭csrf机制
        http.csrf(csrf -> csrf.disable());
        //配置请求拦截方式
        //permitAll:随意访问
        http.authorizeHttpRequests(auth -> auth.requestMatchers("/user/login")
                .permitAll().
                 anyRequest()
                .authenticated());

        //把token校验过滤器添加到过滤器链中
        http.addFilterBefore(jwtAuthenticationTokenFilter,
UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }

    /**
     * 登录时需要调用AuthenticationManager.authenticate执行一次校验
     *
     * @param config
     * @return
     * @throws Exception
     */
    @Bean
    public AuthenticationManager
authenticationManager(AuthenticationConfiguration config) throws Exception {
        return config.getAuthenticationManager();
    }

}
```

登录的业务逻辑层实现类

```
package cn.hxzy.service.impl;

import cn.hxzy.domain.User;
```

```
import cn.hxzy.domain.vo.LoginUser;
import cn.hxzy.mapper.UserMapper;
import cn.hxzy.service.UserService;
import cn.hxzy.utils.JwtUtils;
import com.alibaba.fastjson.JSON;
import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Service;

import java.util.Objects;

@Service
public class UserServiceImpl extends ServiceImpl<UserMapper, User> implements
UserService {

    @Autowired
    private AuthenticationManager authenticationManager;

    @Override
    public String login(User user) {

        //1、封装Authentication对象 ,密码校验，自动完成
        UsernamePasswordAuthenticationToken authentication=new
UsernamePasswordAuthenticationToken(user.getUserName(),user.getPassword());
        //2、进行校验
        Authentication authenticate =
authenticationManager.authenticate(authentication);
        //3、如果authenticate为空
        if(Objects.isNull(authenticate)){
            throw new RuntimeException("登录失败");
        }
        //放入的用户信息
        LoginUser loginUser=(LoginUser) authenticate.getPrincipal();
        //生成jwt，使用fastjson的方法，把对象转成字符串
        String loginUserString = JSON.toJSONString(loginUser);
        //调用JWT工具类，生成jwt令牌
        String jwt = JwtUtils.createJWT(loginUserString, null);
        return jwt;
    }
}
```

打开postman进行测试



5 认证校验过滤器

我们需要自定义一个过滤器，这个过滤器会去获取请求头中的token，对token进行解析取出其中的userid。

使用userid去redis中获取对应的LoginUser对象。

然后封装Authentication对象存入SecurityContextHolder

```
package cn.hxzy.filter;

import cn.hxzy.domain.vo.LoginUser;
import cn.hxzy.utils.JwtUtils;
import com.alibaba.fastjson.JSON;
import io.jsonwebtoken.Claims;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;
import org.springframework.util.StringUtils;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;

/**
 * token验证过滤器
 */

@Component
public class JwtAuthenticationTokenFilter extends OncePerRequestFilter {

    @Override
```

```
    protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain filterChain) throws ServletException,
IOException {

        //获取当前请求的url地址
        String url = request.getRequestURI();
        System.out.println(url);
        //如果当前请求不是登录请求，则需要进行token验证
        if (url.equals("/user/login")) {
            filterChain.doFilter(request, response);
            return;
        }

        //获取token
        String token = request.getHeader("Authorization");
        if (!StringUtils.hasText(token)) {
            throw new RuntimeException("token不存在");
        }
        //解析token
        String loginUserStr=null;
        LoginUser loginUser=null;
        try {
            Claims claims = JwtUtils.parseJWT(token);
            loginUserStr = claims.getSubject();
            loginUser = JSON.parseObject(loginUserStr,LoginUser.class);

        } catch (Exception e) {
            e.printStackTrace();
            throw new RuntimeException("token非法");
        }

        //存入SecurityContextHolder
        //TODO 获取权限信息封装到Authentication中
        UsernamePasswordAuthenticationToken authenticationToken =
                new UsernamePasswordAuthenticationToken(loginUser,null,null);
        //设置到Spring Security上下文

 SecurityContextHolder.getContext().setAuthentication(authenticationToken);
        //放行
        filterChain.doFilter(request, response);
    }
}
```

6 注册认证过滤器

```
package cn.hxzy.config;


import cn.hxzy.filter.JwtAuthenticationTokenFilter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;
```

```
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.authentication.ProviderManager;
import
org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import
org.springframework.security.config.annotation.authentication.builders.Authentic
ationManagerBuilder;
import
org.springframework.security.config.annotation.authentication.configuration.Auth
enticationConfiguration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecuri
ty;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFi
lter;


@EnableWebSecurity
@Configuration
public class SecurityConfig {


    @Autowired
    private JwtAuthenticationTokenFilter jwtAuthenticationTokenFilter;

    //创建BCryptPasswordEncoder注入容器
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        //配置关闭csrf机制
        http.csrf(csrf -> csrf.disable());
        //配置请求拦截方式
        //permitAll:随意访问
        http.authorizeHttpRequests(auth -> auth.requestMatchers("/user/login")
                .permitAll().
                 anyRequest()
                .authenticated());

        //把token校验过滤器添加到过滤器链中
        http.addFilterBefore(jwtAuthenticationTokenFilter,
UsernamePasswordAuthenticationFilter.class);
        //允许跨域
        return http.build();
    }

    /**
     * 登录时需要调用AuthenticationManager.authenticate执行一次校验
     *
```

```
     * @param config
     * @return
     * @throws Exception
     */
    @Bean
    public AuthenticationManager
authenticationManager(AuthenticationConfiguration config) throws Exception {
        return config.getAuthenticationManager();
    }

}
```

3. 授权

3.0 权限系统的作用

例如一个学校图书馆的管理系统，如果是普通学生登录就能看到借书还书相关的功能，不可能让他看到并且去使用添加书籍信息，删除书籍信息等功能。但是如果是一个图书馆管理员的账号登录了，应该就能看到并使用添加书籍信息，删除书籍信息等功能。

总结起来就是**不同的用户可以使用不同的功能。**这就是权限系统要去实现的效果。

我们不能只依赖前端去判断用户的权限来选择显示哪些菜单哪些按钮。因为如果只是这样，如果有人知道了对应功能的接口地址就可以不通过前端，直接去发送请求来实现相关功能操作。

所以我们还需要在后台进行用户权限的判断，判断当前用户是否有相应的权限，必须具有所需权限才能进行相应的操作。

3.1 授权基本流程

在SpringSecurity中，会使用默认的FilterSecurityInterceptor来进行权限校验。在FilterSecurityInterceptor中会从SecurityContextHolder获取其中的Authentication，然后获取其中的权限信息。当前用户是否拥有访问当前资源所需的权限。

所以我们在项目中只需要把当前登录用户的权限信息也存入Authentication。然后设置我们的资源所需要的权限即可。

3.2 授权实现

3.2.1 限制访问资源所需权限

SpringSecurity为我们提供了基于注解的权限控制方案，这也是我们项目中主要采用的方式。我们可以使用注解去指定访问对应的资源所需的权限。

但是要使用它我们需要先开启相关配置。

```
@EnableGlobalMethodSecurity(prePostEnabled = true)
```

然后就可以使用对应的注解。@PreAuthorize

```
@RestController
public class HelloController {

    @RequestMapping("/hello")
    @PreAuthorize("hasAuthority('test')")
    public String hello(){
        return "hello";
    }
}
```

3.2.2 封装权限信息

我们前面在写UserDetailsServiceImpl的时候说过，在查询出用户后还要获取对应的权限信息，封装到UserDetails中返回。

我们先直接把权限信息写死封装到UserDetails中进行测试。

LoginUser修改完后我们就可以在UserDetailsServiceImpl中去把权限信息封装到LoginUser中了。我们写死权限进行测试，后面我们再从数据库中查询权限信息。

```
package cn.hxzy.service.impl;

import cn.hxzy.domain.User;
import cn.hxzy.domain.vo.LoginUser;
import cn.hxzy.mapper.UserMapper;
import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
import org.assertj.core.util.Arrays;
import org.springframework.beans.factory.annotation.Autowired;
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.ArrayList;
import java.util.List;
import java.util.Objects;

@Service
public class UserDetailsServiceImpl implements UserDetailsService {

    @Autowired
    private UserMapper userMapper;
    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        //1、连接数据库，根据用户名查询账号信息
        QueryWrapper<User> queryWrapper=new QueryWrapper<>();
        queryWrapper.eq("user_name",username);
        User user = userMapper.selectOne(queryWrapper);
        if(Objects.isNull(user)){
            throw  new RuntimeException("不存在此用户");
        }
        //TODO
        //2、赋权操作  死的数据
        List<String> list=new ArrayList<>();
        list.add("select");
        list.add("delete");
```

```
        // 3、返回UserDetails对象
        return new LoginUser(user,list);
    }
}
```

我们之前定义了UserDetails的实现类LoginUser，想要让其能封装权限信息就要对其进行修改。

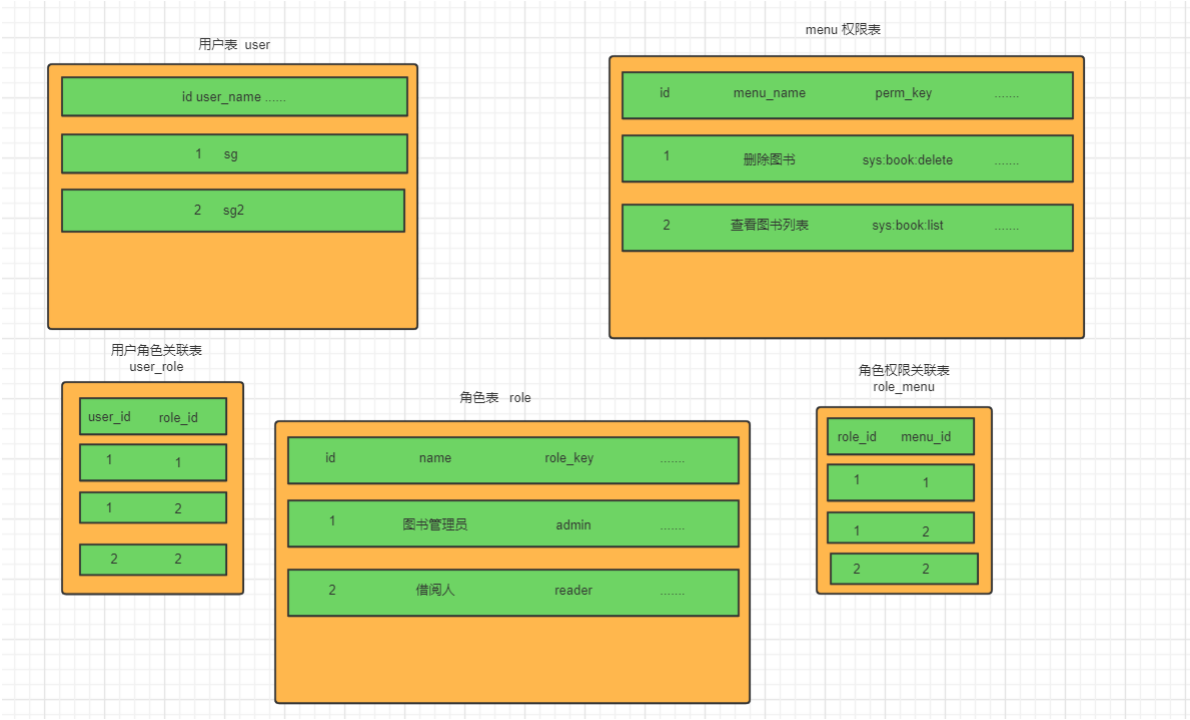
```
package cn.hxzy.domain.vo;

import cn.hxzy.domain.User;
import com.alibaba.fastjson.annotation.JSONField;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.function.Consumer;


@Data
@NoArgsConstructor
public class LoginUser implements UserDetails {
    //传入的参数权限列表
    private List<String> list; //select delete

    private User user;

    //带参数的构造
    public LoginUser(User user,List<String> list){
        this.list=list;
        this.user=user;
    }

    //自定义一个权限列表的集合  中转操作
    @JSONField(serialize = false)
    List<SimpleGrantedAuthority> authorities; //子类
    //权限列表
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        //父类
        if(authorities!=null){
            return authorities;
        }
        authorities=new ArrayList<>(); //对象
        for (String item : list) {
            SimpleGrantedAuthority authority=new SimpleGrantedAuthority(item);
            authorities.add(authority);
        }

        return authorities;
```

```
    }

    //返回密码
    @Override
    public String getPassword() {

        return user.getPassword();
    }

    //用户名
    @Override
    public String getUsername() {
        return user.getUserName();
    }

    //账号是否未过期
    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    //判断账号是否没有锁定
    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    //判断账号是否没有超时
    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }


    //判断账号是否可用
    @Override
    public boolean isEnabled() {
        return true;
    }
}
```

3.3 从数据库查询权限信息

3.3.1 RBAC权限模型

RBAC权限模型（Role-Based Access Control）即：基于角色的权限控制。这是目前最常被开发者使用也是相对易用、通用权限模型。



RBAC权限表设计

1、用户表

用户ID	用户账号	用户密码
1000	mengshujun	
1001	zhangsan	

2、角色表

角色ID	角色名称	角色说明
100	super_admin	超级管理员
101	admin_add	添加管理员

3、用户与角色的关联关系表

ID	角色ID	用户ID
1	100	1000
2	101	1001

4、权限表 (菜单表)：整个系统所有的功能api

权限ID	请求URL	权限名	权限标识符
1	/addUser	新增用户	addUser
2	/updateUser	修改用户	updateUser
3	/delUser	删除用户	delUser
4	/showUser	查询用户	showUser

5、角色与权限的关联关系表

ID	角色ID	权限ID
1	100	1
2	100	2
3	100	3
4	100	4
5	101	1

3.3.2 准备工作

MySQL在5.5.3之后增加了这个utf8mb4的编码，mb4就是most bytes 4的意思，专门用来兼容四字节的unicode。好在utf8mb4是utf8的超集，除了将编码改为utf8mb4外不需要做其他转换。当然，为了节省空间，一般情况下使用utf8也就够了。

```
CREATE DATABASE /*!32312 IF NOT EXISTS*/`security_manager` /*!40100 DEFAULT
CHARACTER SET utf8mb4 */;

USE `security_manager`;

/*Table structure for table `sys_menu` */

DROP TABLE IF EXISTS `sys_menu`;

CREATE TABLE `sys_menu` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `menu_name` varchar(64) NOT NULL DEFAULT 'NULL' COMMENT '菜单名',
  `path` varchar(200) DEFAULT NULL COMMENT '路由地址',
  `component` varchar(255) DEFAULT NULL COMMENT '组件路径',
  `visible` char(1) DEFAULT '0' COMMENT '菜单状态（0显示 1隐藏）',
  `status` char(1) DEFAULT '0' COMMENT '菜单状态（0正常 1停用）',
  `perms` varchar(100) DEFAULT NULL COMMENT '权限标识',
  `icon` varchar(100) DEFAULT '#' COMMENT '菜单图标',
  `create_by` bigint(20) DEFAULT NULL,
  `create_time` datetime DEFAULT NULL,
  `update_by` bigint(20) DEFAULT NULL,
  `update_time` datetime DEFAULT NULL,
  `del_flag` int(11) DEFAULT '0' COMMENT '是否删除（0未删除 1已删除）',
  `remark` varchar(500) DEFAULT NULL COMMENT '备注',
```

```
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8mb4 COMMENT='菜单表';

/*Table structure for table `sys_role` */

DROP TABLE IF EXISTS `sys_role`;

CREATE TABLE `sys_role` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `name` varchar(128) DEFAULT NULL,
  `role_key` varchar(100) DEFAULT NULL COMMENT '角色权限字符串',
  `status` char(1) DEFAULT '0' COMMENT '角色状态（0正常 1停用）',
  `del_flag` int(1) DEFAULT '0' COMMENT 'del_flag',
  `create_by` bigint(200) DEFAULT NULL,
  `create_time` datetime DEFAULT NULL,
  `update_by` bigint(200) DEFAULT NULL,
  `update_time` datetime DEFAULT NULL,
  `remark` varchar(500) DEFAULT NULL COMMENT '备注',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8mb4 COMMENT='角色表';

/*Table structure for table `sys_role_menu` */

DROP TABLE IF EXISTS `sys_role_menu`;

CREATE TABLE `sys_role_menu` (
  `role_id` bigint(200) NOT NULL AUTO_INCREMENT COMMENT '角色ID',
  `menu_id` bigint(200) NOT NULL DEFAULT '0' COMMENT '菜单id',
  PRIMARY KEY (`role_id`,`menu_id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8mb4;

/*Table structure for table `sys_user` */

DROP TABLE IF EXISTS `sys_user`;

CREATE TABLE `sys_user` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT '主键',
  `user_name` varchar(64) NOT NULL DEFAULT 'NULL' COMMENT '用户名',
  `nick_name` varchar(64) NOT NULL DEFAULT 'NULL' COMMENT '昵称',
  `password` varchar(64) NOT NULL DEFAULT 'NULL' COMMENT '密码',
  `status` char(1) DEFAULT '0' COMMENT '账号状态（0正常 1停用）',
  `email` varchar(64) DEFAULT NULL COMMENT '邮箱',
  `phonenumber` varchar(32) DEFAULT NULL COMMENT '手机号',
  `sex` char(1) DEFAULT NULL COMMENT '用户性别（0男，1女，2未知）',
  `avatar` varchar(128) DEFAULT NULL COMMENT '头像',
  `user_type` char(1) NOT NULL DEFAULT '1' COMMENT '用户类型（0管理员，1普通用户）',
  `create_by` bigint(20) DEFAULT NULL COMMENT '创建人的用户id',
  `create_time` datetime DEFAULT NULL COMMENT '创建时间',
  `update_by` bigint(20) DEFAULT NULL COMMENT '更新人',
  `update_time` datetime DEFAULT NULL COMMENT '更新时间',
  `deleted` int(11) DEFAULT '0' COMMENT '删除标志（0代表未删除，1代表已删除）',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8mb4 COMMENT='用户表';

/*Table structure for table `sys_user_role` */

DROP TABLE IF EXISTS `sys_user_role`;
```

```
CREATE TABLE `sys_user_role` (
  `user_id` bigint(200) NOT NULL AUTO_INCREMENT COMMENT '用户id',
  `role_id` bigint(200) NOT NULL DEFAULT '0' COMMENT '角色id',
  PRIMARY KEY (`user_id`,`role_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

mysql查询权限SQL语句

```
SELECT
    DISTINCT t4.`perms`
FROM
    sys_user_role t1
    LEFT JOIN `sys_role` t2 ON t1.`role_id` = t2.`id`
    LEFT JOIN `sys_role_menu` t3 ON t1.`role_id` = t3.`role_id`
    LEFT JOIN `sys_menu` t4 ON t4.`id` = t3.`menu_id`
WHERE
    user_id = 2
    AND t2.`status` = 0
    AND t4.`status` = 0
```

创建Menu实体类

@JsonInclude用法

JsonJsonInclude.Include.ALWAYS 这个是默认策略，任何情况下都序列化该字段，和不写这个注解是一样的效

果。
JsonJsonInclude.Include.NON_NULL这个最常用，即如果加该注解的字段为null,那么就不序列化这个字段了。

作用在类上表明该类为NULL的字段不参加序列化!

```
package cn.hxzy.entity;

import com.baomidou.mybatisplus.annotation.TableId;
import com.baomidou.mybatisplus.annotation.TableName;
import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.io.Serializable;
import java.util.Date;

/**
 * 菜单表(Menu)实体类
 */
@TableName(value="sys_menu")
@Data
@AllArgsConstructor
@NoArgsConstructor
@JsonInclude(JsonInclude.Include.NON_NULL)
public class Menu implements Serializable {
    private static final long serialVersionUID = -54979041104113736L;
```

```
    @TableId
    private Long id;
    /**
    * 菜单名
    */
    private String menuName;
    /**
    * 路由地址
    */
    private String path;
    /**
    * 组件路径
    */
    private String component;
    /**
    * 菜单状态（0显示 1隐藏）
    */
    private String visible;
    /**
    * 菜单状态（0正常 1停用）
    */
    private String status;
    /**
    * 权限标识
    */
    private String perms;
    /**
    * 菜单图标
    */
    private String icon;

    private Long createBy;

    private Date createTime;

    private Long updateBy;

    private Date updateTime;
    /**
    * 是否删除（0未删除 1已删除）
    */
    private Integer delFlag;
    /**
    * 备注
    */
    private String remark;
}
```

3.3.3 代码实现

我们只需要根据用户id去查询到其所对应的权限信息即可。

所以我们可以先定义个mapper，其中提供一个方法可以根据userid查询权限信息。

```
package cn.hxzy.mapper;
```

```
import cn.hxzy.entity.Menu;
import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import org.apache.ibatis.annotations.Param;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface MenuMapper extends BaseMapper<Menu> {

    List<String> selectPermsByUserId(@Param("userId") Long userId);
}
```

尤其是自定义方法，所以需要创建对应的mapper文件，定义对应的sql语句

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="cn.hxzy.mapper.MenuMapper">
    <select id="selectPermsByUserId" resultType="java.lang.String">
       SELECT
        DISTINCT t4.`perms`
    FROM
        sys_user_role t1
        LEFT JOIN `sys_role` t2 ON t1.`role_id` = t2.`id`
        LEFT JOIN `sys_role_menu` t3 ON t1.`role_id` = t3.`role_id`
        LEFT JOIN `sys_menu` t4 ON t4.`id` = t3.`menu_id`
    WHERE
        user_id = #{userId}
        AND t2.`status` = 0
        AND t4.`status` = 0
    </select>
</mapper>
```

在application.yml中配置mapperXML文件的位置

```
 mapper-locations: classpath*:/mapper/**/*.xml
```

测试MenuMapper

```
@Test
public void testMenu(){
    List<String> list = menuMapper.selectPermsByUserId(2L);
    for (String item:list) {
        System.out.println("item:"+item);
    }
}
```

然后我们可以在UserDetailsServiceImpl中去调用该mapper的方法查询权限信息封装到LoginUser对象中即可。

```
package cn.hxzy.service.impl;
```

```
import cn.hxzy.entity.LoginUser;
import cn.hxzy.entity.User;
import cn.hxzy.mapper.MenuMapper;
import cn.hxzy.mapper.UserMapper;
import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
import org.springframework.beans.factory.annotation.Autowired;
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.List;
import java.util.Objects;

@Service
public class UserDetailsServiceImpl implements UserDetailsService {
    @Autowired
    private UserMapper userMapper;

    @Autowired
    private MenuMapper menuMapper;

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        //查询用户信息
        QueryWrapper<User> queryWrapper=new QueryWrapper<>();
        queryWrapper.eq("user_name",username);
        User user = userMapper.selectOne(queryWrapper);
        //如果没有查询到用户，就抛出异常
        if(Objects.isNull(user)){
            throw  new RuntimeException("用户名或密码错误");
        }
        //TODO 查询用户对应的权限信息
        //测试写法
        //List<String> list = new ArrayList<>(Arrays.asList("select","delete"));
        List<String> list =  menuMapper.selectPermsByUserId(user.getId());
        //如果有，把数据封装成UserDetails对象返回
        return new LoginUser(user,list);
    }
}
```

3.3.4 测试接口权限

```
@RestController
public class NewsController {

    @GetMapping("/getNews")
    @PreAuthorize("hasAnyAuthority('sys:news:select')")
    public String getNews(){
        return "news列表";
    }

    @PostMapping("/insertNews")
    @PreAuthorize("hasAnyAuthority('sys:news:add')")
```

```
    public String insertNews(){
        return "保存成功";
    }

    @PutMapping("/updateNews")
    @PreAuthorize("hasAnyAuthority('sys:news:update')")
    public String updateNews(){
        return "修改成功";
    }

    @DeleteMapping("/deleteNews")
    @PreAuthorize("hasAnyAuthority('sys:news:delete')")
    public String deleteNews(){
        return "删除成功";
    }

}
```

4. 项目优化-自定义处理器

我们还希望在认证失败或者是授权失败的情况下也能和我们的接口一样返回相同结构的json，这样可以让前端能对响应进行统一的处理。要实现这个功能我们需要知道SpringSecurity的异常处理机制。

在SpringSecurity中，如果我们在认证或者授权的过程中出现了异常会被ExceptionTranslationFilter捕获到。在ExceptionTranslationFilter中会去判断是认证失败还是授权失败出现的异常。

4.1 自定义验证异常类

创建exception包，在exception包下创建自定义CustomerAuthenticationException类，继承AuthenticationException类

```
package cn.hxzy.exception;


import org.springframework.security.core.AuthenticationException;

/**
 * 自定义验证异常类
 */
public class CustomerAuthenticationException extends AuthenticationException {
    public CustomerAuthenticationException(String message){
        super(message);
    }
}
```

4.2 编写认证用户无权限访问处理器

在cn.hxzy.handler包下创建CustomerAccessDeniedHandler认证用户访问无权限资源时处理器类。

```
package cn.hxzy.handler;


import cn.hxzy.common.R;
import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.serializer.SerializerFeature;
import jakarta.servlet.ServletException;
import jakarta.servlet.ServletOutputStream;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.security.access.AccessDeniedException;
import org.springframework.security.web.access.AccessDeniedHandler;
import org.springframework.stereotype.Component;
import java.io.IOException;
import java.nio.charset.StandardCharsets;

/**
 * 认证用户访问无权限资源时处理器
 */
@Component
public class CustomerAccessDeniedHandler implements AccessDeniedHandler {
    @Override
    public void handle(HttpServletRequest request, HttpServletResponse response,
AccessDeniedException accessDeniedException) throws IOException,
ServletException {
        //设置客户端的响应的内容类型
        response.setContentType("application/json;charset=UTF-8");
        //获取输出流
        ServletOutputStream outputStream = response.getOutputStream();
        //消除循环引用
        String result = JSON.toJSONString(R.error().code(700).message("无权限访问,
请联系管理员！"), SerializerFeature.DisableCircularReferenceDetect);
        outputStream.write(result.getBytes(StandardCharsets.UTF_8));
        outputStream.flush();
        outputStream.close();
    }
}
```

4.3 编写匿名用户访问资源处理器

在cn.hxzy.handler包下创建 AnonymousAuthenticationHandler匿名用户访问资源处理器类。

```
package cn.hxzy.handler;

import cn.hxzy.common.R;
import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.serializer.SerializerFeature;
import jakarta.servlet.ServletException;
import jakarta.servlet.ServletOutputStream;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.security.authentication.BadCredentialsException;
```

```
import
org.springframework.security.authentication.InternalAuthenticationServiceExcepti
on;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.AuthenticationEntryPoint;
import org.springframework.stereotype.Component;

import java.io.IOException;
import java.nio.charset.StandardCharsets;

/**
 * 匿名用户访问无权限资源的处理类
 */
@Component
public class AnonymousAuthenticationHandler implements AuthenticationEntryPoint
{
    @Override
    public void commence(HttpServletRequest request, HttpServletResponse
response, AuthenticationException authException) throws IOException,
ServletException {

        //设置客户端的响应的内容类型
        response.setContentType("application/json;charset=UTF-8");
        String result = null;
        //获取输出流
        ServletOutputStream outputStream = response.getOutputStream();
        // System.out.println("异常消息:"+authException.getMessage()+",对
象:"+authException);
        if (authException instanceof BadCredentialsException) {
            // 用户名未找到，可以在这里添加自定义处理逻辑
            result = JSON.toJSONString(R.error()
                            .code(HttpServletResponse.SC_UNAUTHORIZED)
                            .message(authException.getMessage()),
                    SerializerFeature.DisableCircularReferenceDetect);
        } else if (authException instanceof
InternalAuthenticationServiceException) {
            result = JSON.toJSONString(R.error()
                            .code(HttpServletResponse.SC_UNAUTHORIZED)
                            .message("用户名为空!"),
                    SerializerFeature.DisableCircularReferenceDetect);
        } else {
            // 其他身份验证异常处理
            result = JSON.toJSONString(R.error()
                            .code(600).message("匿名用户无权限访问！"),
                    SerializerFeature.DisableCircularReferenceDetect);  //消除循环
引用
        }
        outputStream.write(result.getBytes(StandardCharsets.UTF_8));
        outputStream.flush();
        outputStream.close();
    }
}
```

4.4 改造认证校验过滤器

```
package cn.hxzy.filter;


import cn.hxzy.domain.vo.LoginUser;
import cn.hxzy.exception.CustomerAuthenticationException;
import cn.hxzy.handler.LoginFailureHandler;
import cn.hxzy.utils.JwtUtils;
import com.alibaba.fastjson.JSON;
import io.jsonwebtoken.Claims;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;
import org.springframework.util.ObjectUtils;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;


//每一个servlet请求，只会执行一次
@Component
public class JwtAuthenticationTokenFilter extends OncePerRequestFilter {

    @Autowired
    private LoginFailureHandler loginFailureHandler;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain filterChain) throws ServletException,
IOException {
        try {
            //获取当前请求的url地址
            String url = request.getRequestURI();
            //如果当前请求不是登录请求，则需要进行token验证
            if (!url.equals("/user/login")) {
                this.validateToken(request);
            }
        } catch (AuthenticationException e) {
            System.out.println(e);
            loginFailureHandler.onAuthenticationFailure(request, response, e);
        }
        //登录请求不需要验证token
        doFilter(request, response, filterChain);
    }


    /**
     * 验证token
     *
```

```
     * @param request
     */
    private void validateToken(HttpServletRequest request) throws
            AuthenticationException {
        //从头部获取token信息
        String token = request.getHeader("Authorization");
        //如果请求头部没有获取到token，则从请求的参数中进行获取
        if (ObjectUtils.isEmpty(token)) {
            token = request.getParameter("Authorization");
        }
        if (ObjectUtils.isEmpty(token)) {
            throw new CustomerAuthenticationException("token不存在");
        }
        //如果存在token，则从token中解析出用户名
        Claims claims = null;
        try {
            claims = JwtUtils.parseJWT(token);
        } catch (Exception e) {
            throw new CustomerAuthenticationException("token解析失败");
        }
        //获取到主题
        String loginUserString = claims.getSubject();
        //把字符串转成loginUser对象
        LoginUser loginUser = JSON.parseObject(loginUserString,
LoginUser.class);

        //创建身份验证对象
        UsernamePasswordAuthenticationToken authenticationToken = new
UsernamePasswordAuthenticationToken(loginUser, null,
loginUser.getAuthorities());
        //设置到Spring Security上下文

 SecurityContextHolder.getContext().setAuthentication(authenticationToken);
    }

}
```

4.5 自定义认证失败处理器

```
package cn.hxzy.handler;


import cn.hxzy.common.R;
import cn.hxzy.exception.CustomerAuthenticationException;
import com.alibaba.fastjson.JSON;
import jakarta.servlet.ServletException;
import jakarta.servlet.ServletOutputStream;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.security.authentication.*;
import org.springframework.security.core.AuthenticationException;
import
org.springframework.security.web.authentication.AuthenticationFailureHandler;
import org.springframework.stereotype.Component;
```

```
import java.io.IOException;
import java.nio.charset.StandardCharsets;

/**
 * 认证校验失败处理类
 */
@Component
public class LoginFailureHandler implements AuthenticationFailureHandler {
    @Override
    public void onAuthenticationFailure(HttpServletRequest request,
HttpServletResponse response, AuthenticationException exception) throws
IOException, ServletException {
        //设置客户端响应编码格式
        response.setContentType("application/json;charset=UTF-8");
        //获取输出流
        ServletOutputStream outputStream= response.getOutputStream();
        String message = null;//提示信息
        int code = 500;//错误编码
        //判断异常类型
        if(exception instanceof AccountExpiredException){
            message = "账户过期,登录失败！";
        }else if(exception instanceof BadCredentialsException){
            message = "用户名或密码错误,登录失败！";
        }else if(exception instanceof CredentialsExpiredException){
            message = "密码过期,登录失败！";
        }else if(exception instanceof DisabledException){
            message = "账户被禁用,登录失败！";
        }else if(exception instanceof LockedException){
            message = "账户被锁,登录失败！";
        }else if(exception instanceof InternalAuthenticationServiceException){
            message = "账户不存在,登录失败！";
        }else if(exception instanceof CustomerAuthenticationException){
            message = exception.getMessage();
            code = 600;
        }else{
            message = "登录失败！";
        }
        //将错误信息转换成JSON
        String result =JSON.toJSONString(R.error().code(code).message(message));
        outputStream.write(result.getBytes(StandardCharsets.UTF_8));
        outputStream.flush();
        outputStream.close();
    }
}
```

4.6 配置SecurityConfig

```
package cn.hxzy.config;

import cn.hxzy.filter.JwtAuthenticationTokenFilter;
import cn.hxzy.handler.AnonymousAuthenticationHandler;
import cn.hxzy.handler.CustomerAccessDeniedHandler;
import cn.hxzy.handler.LoginFailureHandler;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.authentication.configuration.Auth
enticationConfiguration;
import
org.springframework.security.config.annotation.method.configuration.EnableGlobal
MethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFi
lter;

@Configuration
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfig {

    @Autowired
    private CustomerAccessDeniedHandler customerAccessDeniedHandler;

    @Autowired
    private LoginFailureHandler loginFailureHandler;

    @Autowired
    private AnonymousAuthenticationHandler anonymousAuthenticationHandler;

    //自定义的用于认证的过滤器，进行jwt的校验操作
    @Autowired
    private JwtAuthenticationTokenFilter jwtAuthenticationTokenFilter;

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    /**
     * 登录时需要调用AuthenticationManager.authenticate执行一次校验
     */
    @Bean
    public AuthenticationManager
authenticationManager(AuthenticationConfiguration config) throws Exception {
        return config.getAuthenticationManager();
    }
```

```
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        //关闭csrf
        http.csrf(csrf -> csrf.disable());

        http.formLogin(configurer -> {
            configurer.failureHandler(loginFailureHandler);
        });

        // STATELESS（无状态）： 表示应用程序是无状态的，不会创建会话。
        http.sessionManagement(configurer ->
configurer.sessionCreationPolicy(SessionCreationPolicy.STATELESS));
        //配置请求的拦截方式
        http.authorizeHttpRequests(auth ->
                //登陆请求放行
                auth.requestMatchers("/user/login").permitAll()
                        .anyRequest().authenticated());

        //配置过滤器的执行顺序
        http.addFilterBefore(jwtAuthenticationTokenFilter,
UsernamePasswordAuthenticationFilter.class);

        //配置异常处理器
        http.exceptionHandling(configurer -> {
            configurer.accessDeniedHandler(customerAccessDeniedHandler);
            configurer.authenticationEntryPoint(anonymousAuthenticationHandler);
        });
        return http.build();
    }

}
```

4.7 用户退出系统

因为JWT是无状态的，去中心化的，在服务器端无法清除，服务器一旦进行颁发，就只能等待自动过期才会失效，所以需要redis配合才能完成登录状态的记录。

实现思路：

登录后在redis中添加一个白名单，把认证成功的用户的JWT添加到redis中。

在退出的时候，服务清空springsecurit保存认证通过的Authentication对象，其次在redis中进行删除

改造登录接口：

```
//把生成的token存到redis
String tokenKey = "token_"+token;
stringRedisTemplate.opsForValue().set(tokenKey,token,JwtUtils.JWT_TTL() / 1000);
```

退出后台代码实现：

```
/**
* 用户退出
```

```
* @param request
* @param response
* @return
*/
@PostMapping("/logOut")
public R logout(HttpServletRequest request, HttpServletResponse response) {
    //获取token
    String token = request.getParameter("token");
    //如果没有从头部获取token，那么从参数里面获取
    if (ObjectUtils.isEmpty(token)) {
       token = request.getHeader("token");
    }
    //获取用户相关信息
    Authentication authentication
=SecurityContextHolder.getContext().getAuthentication();
    if (authentication != null) {
        //清空用户信息
        new SecurityContextLogoutHandler().logout(request,
response,authentication);
        //清空redis里面的token
        String key = "token_" + token;
        stringRedisTemplate.delete(key);
    }
    return R.ok().message("用户退出成功");
}
```

认证过滤器再次添加校验的代码信息：

```
//判断redis中是否存在该token
String tokenKey = "token_" + token;
String redisToken = stringRedisTemplate.opsForValue().get(tokenKey);
//如果redis里面没有token,说明该token失效
if (ObjectUtils.isEmpty(redisToken)) {
    throw new CustomerAuthenticationException("token已过期");
}
```

配置完成后，测试系统。

5.扩展OAuth2.0

5.1 OAuth2.0介绍

什么是OAuth2.0？第三方登录授权，比如QQ,微信，微博，微信的授权

比如：

在此场景中，用户想要登录百度，但并没有百度的账号和密码，而是基于第三方平台微信上的账号密码来快速完成登录操作，这个过程百度需要获取用户在微信上的信息。那么问题来了，百度如何获取用户在微信上的信息呢?

OAuth2.0(Open Authorization)

- 一个关于授权的开放网络标准
- 允许用户授权第三方应用访问用户存储在其他服务提供者上的信息
- 不需要将用户名和密码提供给第三方应用

5.2 集成JustAuth

JustAuth登录：https://www.justauth.cn/guide/

JustAuth 集成了诸如：Github、Gitee、支付宝、新浪微博、微信、Google、Facebook、Twitter、StackOverflow等国内外数十家第三方平台。

5.2.1 引入依赖

```
<dependency>
    <groupId>me.zhyd.oauth</groupId>
    <artifactId>JustAuth</artifactId>
    <version>1.16.6</version>
</dependency>

<dependency>
    <groupId>cn.hutool</groupId>
    <artifactId>hutool-all</artifactId>
    <version>5.8.25</version>
</dependency>

<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
</dependency>
```

5.2.2 创建Request

```
AuthRequest authRequest = new AuthGiteeRequest(AuthConfig.builder()
                .clientId("Client ID")
                .clientSecret("Client Secret")
                .redirectUri("应用回调地址")
                .build());
```

5.2.3 代码示例

用户类

```
package cn.hxzy.domain;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.Getter;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class UserVo {

    private Long userId;
    private String userName;
    private String nickName;
    private String avatar;

}
```

授权控制器

```
package cn.hxzy.controller;

import cn.hutool.json.JSONObject;
import cn.hutool.json.JSONUtil;
import cn.hxzy.domain.UserVo;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import jakarta.servlet.http.HttpServletResponse;
import me.zhyd.oauth.config.AuthConfig;
import me.zhyd.oauth.request.AuthGiteeRequest;
import me.zhyd.oauth.model.AuthCallback;
import me.zhyd.oauth.request.AuthRequest;
import me.zhyd.oauth.utils.AuthStateUtils;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;


import java.io.IOException;
```

```
@RestController
@RequestMapping("/oauth")
public class RestAuthController {

    @RequestMapping("/render")
    public void renderAuth(HttpServletResponse response) throws IOException {
        AuthRequest authRequest = getAuthRequest();

 response.sendRedirect(authRequest.authorize(AuthStateUtils.createState()));
    }

    @RequestMapping("/callback")
    public Object login(AuthCallback callback){
        AuthRequest authRequest = getAuthRequest();
        JSONObject jsonObject=
JSONUtil.parseObj(authRequest.login(callback),false);
        JSONObject jsonStr=JSONUtil.parseObj(jsonObject.get("data"),false);
        UserVo userVo=new UserVo();
        userVo.setUserId(Long.parseLong(jsonStr.get("uuid").toString()));
        userVo.setUserName(jsonStr.get("username").toString());
        userVo.setNickName(jsonStr.get("nickname").toString());
        userVo.setAvatar(jsonStr.get("avatar").toString());
        return userVo;
    }

    private AuthRequest getAuthRequest() {
        return new AuthGiteeRequest(AuthConfig.builder()

.clientId("08eed381d2e8cad508de59d99e9d2111e542e646e9eed2bfbf194aa3f706750c")

.clientSecret("432e007c2f4f556cdac320577174821b0cffc457afeb4f14bb8c9b0ff157d5ff"
)
                .redirectUri("http://localhost:8080/oauth/callback")
                .build());
    }
}
```