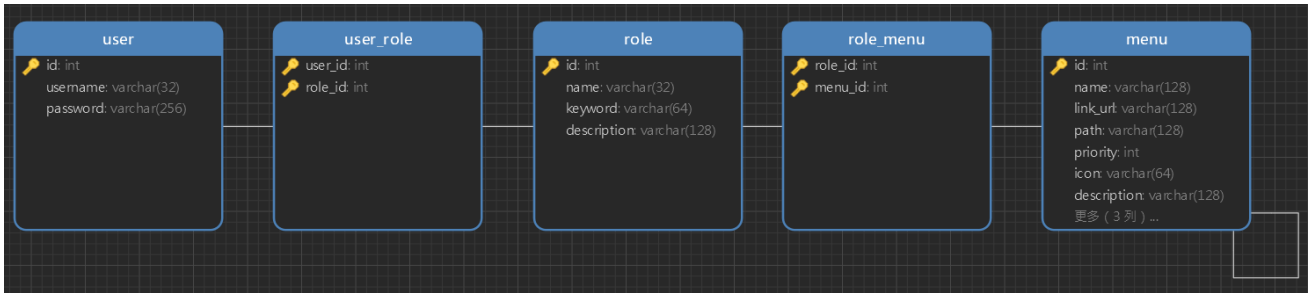
0 RBAC模型核心数据表



1 C++使用权限验证

C++服务器只需要判断是否登录即可，然后可以通过负载信息获取到它用户ID和权限列表。

1.1 开启凭证检查

修改`SystemInterceptor.cpp`

```
#ifndef CHECK_TOKEN

//开启凭证检查，解开下一行注释即可
//#define CHECK_TOKEN

#endif
```

1.2 生成凭证

可以通过单元测试模式中`test/jwt/test-jwt.cpp`生成测试凭证，要修改用户名或id可以通过下面的示例代码：

```
#define TEST_CREATE_PAYLOAD() \
PayloadDTO p; \
p.getAuthorities().push_back("SUPER_ADMIN"); \
p.setUsername(u8"roumiou"); \
p.setId("1"); \
p.setExp(3600 * 30)

TEST(JwtTest, TestHmac) {
    TEST_CREATE_PAYLOAD();
    p.setUsername("juliet");
    std::string token = JWTUtil::generateTokenByHmac(p, TEST_HAMC_SECRET);
    ASSERT_NE(token, "");
    std::cout << "HMAC-TOKEN:\n\n" << token << std::endl;
    auto pv = JWTUtil::verifyTokenByHmac(token, TEST_HAMC_SECRET);
    ASSERT_EQ(pv.getCode(), PayloadCode::SUCCESS);
    std::cout << "\nusername:" << pv.getUsername() << std::endl;
}
```

1.3 Swagger传递凭证

注意：使用凭证一定要开启开凭证验证（参考1.1开启凭证检查）

2 Spring Cloud权限框架应用

关于OAuth2：

OAuth 2.0（Open Authorization 2.0）是一种用于授权的开放标准协议，用于授权第三方应用程序访问用户在其他应用程序上托管的受保护资源。它允许用户提供给第三方应用程序有限的访问权限，而无需将用户名和密码直接提供给第三方应用程序。

OAuth 2.0 的核心思想是通过代理服务器（Authorization Server）来颁发访问令牌（Access Token），第三方应用程序使用该访问令牌来获取用户授权范围内的资源。它的设计目标是为了解决应用程序之间的身份验证和授权问题，并提供更安全、更可靠的用户授权机制。

OAuth 2.0 协议中的角色包括：

- 资源服务器(Resource server)：该服务器是用户拥有资源的服务器，它是受OAuth保护的服务器。一般我们叫做API提供商，因为它拥有受保护的数据，比如图片、视频、日历或者合同等。
- 资源拥有者(Resource owner)：通常是一个应用的用户，也是资源拥有者，它拥有授予访问在资源服务器上的资源。
- 客户端/客户(Client)：一个应用程序发出API请求对受保护资源执行一些行为，而这些行为是经过资源拥有授权的。
- 授权服务器(Authorization server)：当前授权服务器从资源拥有者达成共识时，它会发放访问令牌给客户以允许访问在资源服务器上的受保护的资源 。比较小的API提供商可能会使用相同的应用和URL空间作为授权服务器和资源服务器。

OAuth 2.0 的授权流程包括以下步骤：

1. 客户端向授权服务器请求授权，并提供身份验证凭据。
2. 授权服务器验证客户端身份，并要求用户授权。
3. 用户同意授权，授权服务器颁发访问令牌给客户端。
4. 客户端使用访问令牌向资源服务器请求受保护资源。
5. 资源服务器验证访问令牌，并提供受保护资源给客户端。

OAuth 2.0 的优点包括简化了用户授权流程，提高了安全性，允许用户选择性地授权资源访问，并支持多种授权方式（如授权码模式、密码模式、客户端模式等）。它广泛应用于各种互联网应用程序中，例如第三方登录、API 访问授权等场景。

1 搭建认证服务

创建 oauth2 模块，并按照下列流程完善模块。

1.1 添加依赖

```
<!— web —>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<!— alibaba nacos discovery —>
```

```
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
</dependency>
<!-- alibaba nacos config -->
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
</dependency>
<!-- oauth2 -->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-oauth2</artifactId>
</dependency>
<!-- common -->
<dependency>
    <groupId>com.zeroone.star</groupId>
    <artifactId>project-common</artifactId>
</dependency>
<!-- apis -->
<dependency>
    <groupId>com.zeroone.star</groupId>
    <artifactId>project-apis</artifactId>
</dependency>
<!-- mybatis plus -->
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
</dependency>
<!-- druid starter -->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid-spring-boot-starter</artifactId>
</dependency>
<!-- mysql driver -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
<!-- redis starter -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

1.2 修改相关配置文件

本模块application配置

```
server:
  port: ${sp.auth}
spring:
  application:
    name: ${sn.auth}
```

1.3 使用keytool生成RSA证书

详细步骤键项目README.md

1.4 数据层接口与实现

使用代码工具生成user、role、menu表映射代码。

RoleMapper添加对应接口和实现

```
@Mapper
public interface RoleMapper extends BaseMapper<Role> {
    /**
     * 通过用户编号查询角色
     * @param userId 用户编号
     * @return 角色列表
     */
    List<Role> selectByUserId(int userId);
    /**
     * 通过菜单路径获取对应的角色
     * @param path 菜单路径
     * @return 角色列表
     */
    List<Role> selectByMenuPath(String path);
}
```

实现代码

```
<mapper namespace="com.zeroone.star.oauth2.mapper.RoleMapper">
    <resultMap id="BaseResultMap" type="com.zeroone.star.oauth2.entity.Role">
        <id column="id" jdbcType="INTEGER" property="id" />
        <result column="name" jdbcType="VARCHAR" property="name" />
        <result column="keyword" jdbcType="VARCHAR" property="keyword" />
        <result column="description" jdbcType="VARCHAR" property="description" />
    </resultMap>
    <select id="selectByUserId" resultMap="BaseResultMap">
      select id,`name`,keyword,description from role where id in (select role_id
from user_role where user_id=#{userId})
    </select>
    <select id="selectByMenuPath" resultMap="BaseResultMap">
      select id,keyword from role where id in (select role_id from role_menu
where menu_id=(select id from menu where link_url=#{path}))
    </select>
</mapper>
```

1.5 服务层接口与实现

IRoleService 接口定义与实现

```
public interface IRoleService extends IService<Role> {
    /**
     * 通过用户编号获取角色列表
     * @param userId 用户编号
```

```
     * @return 角色列表
     */
    List<Role> listRoleByUserId(int userId);
    /**
     * 获取指定菜单路径有访问权限的角色
     * @param path 指定菜单路径
     * @return 角色列表
     */
    List<Role> listRoleByMenuPath(String path);
}
```

```
@Service
public class RoleServiceImpl extends ServiceImpl<RoleMapper, Role> implements
IRoleService {
    @Override
    public List<Role> listRoleByUserId(int userId) {
        return baseMapper.selectByUserId(userId);
    }
    @Override
    public List<Role> listRoleByMenuPath(String path) {
        return baseMapper.selectByMenuPath(path);
    }
}
```

`IMenuService` 接口与实现

```
public interface IMenuService extends IService<Menu> {
    /**
     * 获取菜单中的链接地址
     * @return 查询结果
     */
    List<Menu> listAllLinkUrl();
}
```

```
@Service
public class MenuServiceImpl extends ServiceImpl<MenuMapper, Menu> implements
IMenuService {
    @Override
    public List<Menu> listAllLinkUrl() {
        QueryWrapper<Menu> wrapper = new QueryWrapper<>();
        wrapper.select("link_url");
        wrapper.isNotNull("link_url");
        return baseMapper.selectList(wrapper);
    }
}
```

1.6 初始化数据库配置

```
@Configuration
@ComponentScan("com.zeroone.star.project.config.mybatis")
public class MyBatisInit {}
```

```java
@Configuration
@ComponentScan("com.zeroone.star.project.config.redis")
public class RedisInit {}
```

1.7 构建SecurityUser实体

```java
/**
 * <p>
 * 描述：权限认证用户实体
 * </p>
 * <p>版权：&copy;01星球</p>
 * <p>地址：01星球总部</p>
 * @author 阿伟学长
 * @version 1.0.0
 */
@Getter
@Setter
@ToString
public class SecurityUser extends
org.springframework.security.core.userdetails.User {
    /**
     * 关联一个用户对象
     */
    private User user;
    /**
     * 构造初始化
     *
     * @param user        数据库的User对象
     * @param authorities 权限列表
     */
    public SecurityUser(User user, Collection<? extends GrantedAuthority>
authorities) {
        super(user.getUsername(), user.getPassword(), authorities);
        this.user = user;
    }
}
```

1.8 实现UserDetailsService接口

创建 `UserDetailsServiceImpl` 类实现Spring Security的 `UserDetailsService` 接口，用于加载用户信息

```java
/**
 * <p>
 * 描述：用户权限权限服务实现
 * </p>
 * <p>版权：&copy;01星球</p>
 * <p>地址：01星球总部</p>
 * @author 阿伟学长
 * @version 1.0.0
 */
@Service
public class UserDetailsServiceImpl implements UserDetailsService {
```

```
    @Resource
    IUserService userService;
    @Resource
    IRoleService roleService;
    @Resource
    HttpServletRequest request;
    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        String clientId = request.getParameter("client_id");
        if (AuthConstant.CLIENT_MANAGER.equals(clientId)) {
            //1 通过用户名查找用户对象
            User user = new User();
            user.setUsername(username);
            user = userService.getOne(new QueryWrapper<>(user));
            if (user == null) {
                throw new UsernameNotFoundException("用户名或密码错误");
            }
            //2 通过用户ID获取角色列表
            List<Role> roles = roleService.listRoleByUserId(user.getId());
            //3 将数据库角色转换成Security权限对象
            List<GrantedAuthority> authorities = new ArrayList<>();
            roles.forEach(role -> authorities.add(new
SimpleGrantedAuthority(role.getKeyword())));
            //4 构建权限角色对象
            return new SecurityUser(user, authorities);
        } else if (AuthConstant.CLIENT_APP.equals(clientId)) {
            throw new UsernameNotFoundException("用户端查找用户尚未实现");
        }
        throw new UsernameNotFoundException("登录客户端ID错误");
    }
}
```

1.9 添加授权服务相关配置

需要配置加载用户信息的服务`UserDetailsServiceImpl`以及`RSA`的钥匙对`KeyPair`

```
/**
 * <p>
 * 描述：JWT Token增强实现，将用户ID记录到Token里面
 * </p>
 * <p>版权：&copy;01星球</p>
 * <p>地址：01星球总部</p>
 * @author 阿伟学长
 * @version 1.0.0
 */
@Component
class JwtTokenEnhancer implements TokenEnhancer {
    @Override
    public OAuth2AccessToken enhance(OAuth2AccessToken oAuth2AccessToken,
OAuth2Authentication oAuth2Authentication) {
        SecurityUser securityUser = (SecurityUser)
oAuth2Authentication.getPrincipal();
        Map<String, Object> info = new HashMap<>(1);
        //把用户ID设置到JWT中,如果要扩展负载信息可以在这里添加逻辑代码
        info.put("id", securityUser.getUser().getId());
```

```
        DefaultOAuth2AccessToken result = (DefaultOAuth2AccessToken) oAuth2AccessToken;
        result.setAdditionalInformation(info);
        return result;
    }
}

/**
 * <p>
 * 描述：权限授权服务配置
 * </p>
 * <p>版权：&copy;01星球</p>
 * <p>地址：01星球总部</p>
 * @author 阿伟学长
 * @version 1.0.0
 */
@Configuration
@EnableAuthorizationServer
@AllArgsConstructor
public class Oauth2ServerConfig extends AuthorizationServerConfigurerAdapter {
    private final PasswordEncoder passwordEncoder;
    private final UserDetailsServiceImpl userDetailsService;
    private final AuthenticationManager authenticationManager;
    private final JwtTokenEnhancer jwtTokenEnhancer;
    @Override
    public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
        // 针对于第三方客户端配置
        // 指定哪些应用可以访问授权服务并颁发令牌
        // 访问模式是什么
        clients.inMemory()
                //配置客户端ID（管理端）
                .withClient(AuthConstant.CLIENT_MANAGER)
                //配置访问密码
                .secret(passwordEncoder.encode(AuthConstant.CLIENT_PASSWORD))
                //配置授权范围，客户端传递scope必须为下面的值或者不携带scope
                .scopes("all")
                //配置支持的授权模式，支持用户名密码认证和凭证刷新
                .authorizedGrantTypes("password", "refresh_token")
                //配置颁发令牌的有效时间，这里修改成了24小时
                .accessTokenValiditySeconds(3600 * 24)
                //配置颁发刷新令牌的有效时间，这里修改成了1周
                .refreshTokenValiditySeconds(3600 * 24 * 7)
                .and()
                //配置客户端ID（用户端，如app）
                .withClient(AuthConstant.CLIENT_APP)
                .secret(passwordEncoder.encode(AuthConstant.CLIENT_PASSWORD))
                .scopes("all")
                .authorizedGrantTypes("password", "refresh_token")
                .accessTokenValiditySeconds(3600 * 24)
                .refreshTokenValiditySeconds(3600 * 24 * 7);
    }
    @Override
    public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
        //定义JWT内置增强内容
        List<TokenEnhancer> delegates = new ArrayList<>();
        delegates.add(jwtTokenEnhancer);
```

```
        delegates.add(accessTokenConverter());
        //配置JWT的内容增强器
        TokenEnhancerChain enhancerChain = new TokenEnhancerChain();
        enhancerChain.setTokenEnhancers(delegates);
        //授权服务端点访问配置
        endpoints
                //配置授权管理器
                .authenticationManager(authenticationManager)
                //配置加载用户信息的服务
                .userDetailsService(userDetailsService)
                //设置凭证签名转换器
                .accessTokenConverter(accessTokenConverter())
                //设置凭证增强附加额外信息到凭证中
                .tokenEnhancer(enhancerChain);
    }
    @Override
    public void configure(AuthorizationServerSecurityConfigurer security) {
        // 访问安全性配置
        // 在BasicAuthenticationFilter之前添加
ClientCredentialsTokenEndpointFilter,
        // 使用ClientDetailsUserDetailsService来进行登陆认证
        security.allowFormAuthenticationForClients();
    }
    @Bean
    public JwtAccessTokenConverter accessTokenConverter() {
        //使用非对称加密算法来对Token进行签名
        JwtAccessTokenConverter jwtAccessTokenConverter = new
JwtAccessTokenConverter();
        jwtAccessTokenConverter.setKeyPair(keyPair());
        return jwtAccessTokenConverter;
    }
    @Bean
    public KeyPair keyPair() {
        //从classpath下的证书中获取秘钥对
        KeyStoreKeyFactory keyStoreKeyFactory = new KeyStoreKeyFactory(new
ClassPathResource("jwt.jks"), "123456".toCharArray());
        return keyStoreKeyFactory.getKeyPair("01star", "123456".toCharArray());
    }
}
```

1.10 公钥获取接口（选用）

目前系统采用读取密钥文件的方式来获取。

由于我们的网关服务需要RSA的公钥来验证签名是否合法，所以认证服务需要有个接口把公钥暴露出来。

```
@RestController
@RequestMapping("rsa")
public class KeyPairController implements KeyPairApis {
    private final KeyPair keyPair;
    @Autowired
    public KeyPairController(KeyPair keyPair) {
        this.keyPair = keyPair;
    }
    @Override
    @GetMapping("public-key")
```

```
    public Map<String, Object> getPublicKey() {
        RSAPublicKey publicKey = (RSAPublicKey) keyPair.getPublic();
        RSAKey key = new RSAKey.Builder(publicKey).build();
        return new JWKSet(key).toJSONObject();
    }
    @Override
    @GetMapping("private-key")
    public Map<String, Object> getPrivateKey() {
        return null;
    }
}
```

1.11 构建SpringSecurity配置

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
                // 内置端点请求放行
                .requestMatchers(EndpointRequest.toAnyEndpoint()).permitAll()
                // 公钥获取放行
                .antMatchers("/rsa/public-key").permitAll()
                // 其他请求需要已认证才能访问
                .anyRequest().authenticated();
    }
    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

1.12 构建认证接口

```
@RestController
@RequestMapping("oauth")
public class AuthController implements AuthApis {
    private final TokenEndpoint tokenEndpoint;
    @Autowired
    public AuthController(TokenEndpoint tokenEndpoint) {
        this.tokenEndpoint = tokenEndpoint;
    }
    @Override
    @PostMapping("token")
    public JsonVO<Oauth2TokenDTO> postAccessToken(Principal principal,
                                                  @RequestParam Map<String,
String> parameters) {
        //调用oAuth2AccessToken的postAccessToken接口来刷新或颁发token
```

```
        OAuth2AccessToken oAuth2AccessToken;
        try {
            oAuth2AccessToken = tokenEndpoint.postAccessToken(principal,
parameters).getBody();
        } catch (Exception e) {
            return JsonVO.create(null, ResultStatus.FAIL.getCode(),
                                 "postAccessToken:" +
e.getClass().getSimpleName() + ":" + e.getMessage());
        }
        if (oAuth2AccessToken != null) {
            Oauth2TokenDTO oauth2TokenDto = new Oauth2TokenDTO(
                    oAuth2AccessToken.getValue(),
                    oAuth2AccessToken.getRefreshToken().getValue(),
                    "Bearer ",
                    oAuth2AccessToken.getExpiresIn(),
                    parameters.get("client_id"));
            return JsonVO.create(oauth2TokenDto, ResultStatus.SUCCESS);
        }
        return JsonVO.create(null, ResultStatus.FAIL.getCode(),
"oAuth2AccessToken为null");
    }
}
```

1.13 构成初始化角色与资源匹配服务

```
/**
 * <p>
 * 描述：路径与角色资源服务器初始化服务
 * </p>
 * <p>版权：&copy;01星球</p>
 * <p>地址：01星球总部</p>
 * @author 阿伟学长
 * @version 1.0.0
 */
@Service
public class ResourceServiceImpl {
    @Resource
    private RedisTemplate<String, Object> redisTemplate;
    @Resource
    private IMenuService menuService;
    @Resource
    private IRoleService roleService;
    @PostConstruct
    public void init() {
        // 定义缓存map
        Map<String, List<String>> resourceRolesMap = new TreeMap<>();
        // 1 获取所有菜单
        List<Menu> tMenus = menuService.listAllLinkUrl();
        tMenus.forEach(menu -> {
            // 2 获取菜单对应的角色
            List<Role> rolesMenu =
roleService.listRoleByMenuPath(menu.getLinkUrl());
            List<String> roles = new ArrayList<>();
            rolesMenu.forEach(role -> roles.add(role.getKeyword()));
            resourceRolesMap.put(menu.getLinkUrl(), roles);
        });
```

```
        //将资源缓存到redis
        redisTemplate.opsForHash().putAll(RedisConstant.RESOURCE_ROLES_MAP,
resourceRolesMap);
    }
}
```

2 搭建网关服务器

创建`gateway`模块，并按照后续流程配置完善模块。

它将作为`Oauth2`的资源服务、客户端服务使用，对访问微服务的请求进行统一的校验认证和鉴权操作。

2.1 添加依赖

```
<!-- alibaba nacos discovery -->
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
</dependency>
<!-- alibaba nacos config -->
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
</dependency>
<!-- gateway -->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-gateway</artifactId>
</dependency>
<!-- redis starter -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
<!-- common -->
<dependency>
    <groupId>com.zeroone.star</groupId>
    <artifactId>project-common</artifactId>
</dependency>
<!-- domain -->
<dependency>
    <groupId>com.zeroone.star</groupId>
    <artifactId>project-domain</artifactId>
</dependency>
<!-- spring security need-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
```

```
    <artifactId>spring-security-oauth2-resource-server</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-oauth2-client</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-oauth2-jose</artifactId>
</dependency>
```

2.2 修改application配置

主要是路由规则的配置、`Oauth2`中`RSA`公钥的配置及路由白名单的配置；

```yaml
server:
  port: ${sp.gateway}
spring:
  application:
    name: ${sn.gateway}
  security:
    oauth2:
      resourceserver:
        jwt:
          # 公钥文件配置
          public-key-location: classpath:public.pem
  cloud:
    gateway:
      discovery:
        locator:
          # 开启从注册中心动态创建路由的功能
          enabled: true
      # 注意：这里的路径配置需要移植到nacos配置中心system.yaml中
      # 提示：predicates 中如果要匹配多个地址前缀，每个前缀用,分割，如：-Path=/auth/**,/oauth/**
      routes:
        - id: oauth2-auth-route
          uri: lb://${sn.auth}
          predicates:
            - Path=/auth/**
          filters:
            - StripPrefix=1
        - id: login-route
          uri: lb://${sn.login}
          predicates:
            - Path=/login/**
        - id: sample-route
          uri: lb://${sn.sample}
          predicates:
            - Path=/sample/**
        - id: cpp-route
          uri: lb://${sn.cpp}
          predicates:
            - Path=/cpp/**
secure:
  # 配置是否开启鉴权
```

```yaml
  openauthorization: true
  # 配置白名单路径
  white:
    urls:
      - "/actuator/**"
      - "/auth/oauth/token"
      - "/login/auth-login"
      - "/login/refresh-token"
```

初始化 `Redis` 配置

```
@Configuration
@ComponentScan("com.zeroone.star.project.config.redis")
public class RedisInit {}
```

2.3 实现鉴权管理器

在 `WebFluxSecurity`中自定义鉴权操作需要实现 `ReactiveAuthorizationManager` 接口

```
/**
 * <p>
 * 描述：鉴权管理器，用于判断是否有资源的访问权限
 * </p>
 * <p>版权：&copy;01星球</p>
 * <p>地址：01星球总部</p>
 * @author 阿伟学长
 * @version 1.0.0
 */
@RefreshScope
@Component
public class AuthorizationManager implements
ReactiveAuthorizationManager<AuthorizationContext> {
    @Resource
    private RedisTemplate<String, Object> redisTemplate;

    @Value("${secure.openauthorization}")
    private boolean isOpenAuthorization;

    /**
     * 通过路径查询角色列表
     * @param path 路径名
     * @return 没有查询到返回空列表
     */
    private List<String> queryRoleListByPath(String path) {
        Object obj =
redisTemplate.opsForHash().get(RedisConstant.RESOURCE_ROLES_MAP, path);
        if (obj == null) {
            return new ArrayList<>();
        }
        List<String> authorities = Convert.toList(String.class, obj);
        authorities = authorities.stream().map(roleName -> roleName =
AuthConstant.AUTHORITY_PREFIX + roleName).collect(Collectors.toList());
        return authorities;
    }

```

```
    @Override
    public Mono<AuthorizationDecision> check(Mono<Authentication> mono, AuthorizationContext authorizationContext) {
        //如果关闭鉴权功能，直接放行
        if (!isOpenAuthorization) {
            return Mono.just(new AuthorizationDecision(true));
        }

        //1 从Redis中获取当前路径可访问角色列表
        String path = authorizationContext.getExchange().getRequest().getURI().getPath();
        List<String> authorities = queryRoleListByPath(path);
        // 没有查询到结果，尝试查找通配符
        if (authorities.isEmpty()) {
            // 1.1 处理二级地址，比如：请求地址为 "/fc/get"，此时匹配 "/fc/**"
            String[] arr = path.split("/");
            String currPath = "/" + arr[1] + "/**";
            authorities = queryRoleListByPath(currPath);
            // 1.2 处理参数变量，比如请求地址为 "/fc/get/参数变量"，此时匹配 "/fc/get/**"
            if (authorities.isEmpty()) {
                currPath = path.substring(0, path.lastIndexOf("/") + 1) + "**";
                authorities = queryRoleListByPath(currPath);
            }
        }

        //2 认证通过且角色匹配的用户可访问当前路径
        return mono
                // 判断是否认证
                .filter(Authentication::isAuthenticated)
                // 获取权限对象列表(即：角色列表)
                .flatMapIterable(Authentication::getAuthorities)
                // 遍历获取权限对象(即：获取单个角色名)
                .map(GrantedAuthority::getAuthority)
                // 查看当前路径中是否包对应权限（即：是否包含角色名）
                .any(authorities::contains)
                // 根据判断结果构建一个鉴权对象
                .map(AuthorizationDecision::new)
                // 没有权限列表的情况，构建一个无权访问对象
                .defaultIfEmpty(new AuthorizationDecision(false));
    }
}
```

2.4 实现没有登录或token过期处理器

新增一个消息发送工具类

```
/**
 * <p>
 * 描述：通用消息发送工具类
 * </p>
 * <p>版权：&copy;01星球</p>
 * <p>地址：01星球总部</p>
 * @author 阿伟学长
 * @version 1.0.0
 */
```

```
class CommonSender {
    static Mono<Void> sender(ServerWebExchange exchange, ResultStatus resultStatus, String data){
        ServerHttpResponse response = exchange.getResponse();
        response.setStatusCode(HttpStatus.OK);
        response.getHeaders().add(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE);
        ObjectMapper mapper = new ObjectMapper();
        String body = "";
        try {
            body = mapper.writeValueAsString(JsonVO.create(data, resultStatus));
        } catch (JsonProcessingException e1) {
            e1.printStackTrace();
        }
        DataBuffer buffer = response.bufferFactory().wrap(body.getBytes(StandardCharsets.UTF_8));
        return response.writeWith(Mono.just(buffer));
    }
}
```

构建处理器

```
/**
 * <p>
 * 描述：没有登录或token过期时下发消息
 * </p>
 * <p>版权：&copy;01星球</p>
 * <p>地址：01星球总部</p>
 * @author 阿伟学长
 * @version 1.0.0
 */
@Component
public class RestfulAuthenticationEntryPoint implements ServerAuthenticationEntryPoint {
    @Override
    public Mono<Void> commence(ServerWebExchange exchange, AuthenticationException e) {
        return CommonSender.sender(exchange, ResultStatus.UNAUTHORIZED, e.getMessage());
    }
}
```

2.5 实现无权访问处理器

```
/**
 * <p>
 * 描述：没有权限访问时下发消息
 * </p>
 * <p>版权：&copy;01星球</p>
 * <p>地址：01星球总部</p>
 * @author 阿伟学长
 * @version 1.0.0
 */
@Component
public class RestfulAccessDeniedHandler implements ServerAccessDeniedHandler {
```

```
    @Override
    public Mono<Void> handle(ServerWebExchange exchange, AccessDeniedException denied) {
        return CommonSender.sender(exchange, ResultStatus.FORBIDDEN, denied.getMessage());
    }
}
```

2.6 实现白名单配置

```
/**
 * <p>
 * 描述：网关白名单配置
 * </p>
 * <p>版权：&copy;01星球</p>
 * <p>地址：01星球总部</p>
 * @author 阿伟学长
 * @version 1.0.0
 */
@Data
@Component
@ConfigurationProperties(prefix="secure.white")
public class WhitePathConfig {
    private List<String> urls;
}
```

2.7 白名单路径访问过滤器

```
/**
 * <p>
 * 描述：白名单路径处理
 * </p>
 * <p>版权：&copy;01星球</p>
 * <p>地址：01星球总部</p>
 * @author 阿伟学长
 * @version 1.0.0
 */
@Component
public class WhitePathFilter implements WebFilter {
    @Resource
    WhitePathConfig whitePathConfig;
    @SuppressWarnings("NullableProblems")
    @Override
    public Mono<Void> filter(ServerWebExchange exchange, WebFilterChain chain) {
        ServerHttpRequest request = exchange.getRequest();
        URI uri = request.getURI();
        PathMatcher pathMatcher = new AntPathMatcher();
        //白名单路径移除JWT请求头
        List<String> whiteUrls = whitePathConfig.getUrls();
        for (String whiteUrl : whiteUrls) {
            if (pathMatcher.match(whiteUrl, uri.getPath())) {
                request = exchange.getRequest().mutate().header("Authorization", "").build();
                exchange = exchange.mutate().request(request).build();
```

```
                return chain.filter(exchange);
            }
        }
        return chain.filter(exchange);
    }
}
```

2.8 实现跨域过滤器

```
/**
 * <p>
 * 描述：跨域过滤器
 * </p>
 * <p>版权：&copy;01星球</p>
 * <p>地址：01星球总部</p>
 * @author 阿伟学长
 * @version 1.0.0
 */
@Component
public class CorsFilter implements WebFilter {
    @SuppressWarnings("NullableProblems")
    @Override
    public Mono<Void> filter(ServerWebExchange exchange, WebFilterChain chain) {
        ServerHttpRequest request = exchange.getRequest();
        if (CorsUtils.isCorsRequest(request)) {
            ServerHttpResponse response = exchange.getResponse();
            HttpHeaders headers = response.getHeaders();
            headers.set(HttpHeaders.ACCESS_CONTROL_ALLOW_ORIGIN, "*");
            headers.add(HttpHeaders.ACCESS_CONTROL_ALLOW_HEADERS, "*");
            headers.add(HttpHeaders.ACCESS_CONTROL_ALLOW_METHODS, "*");
            headers.add(HttpHeaders.ACCESS_CONTROL_ALLOW_CREDENTIALS, "false");
            headers.add(HttpHeaders.ACCESS_CONTROL_EXPOSE_HEADERS, "*");
            headers.add(HttpHeaders.ACCESS_CONTROL_MAX_AGE, "3600");
            if (request.getMethod() == HttpMethod.OPTIONS) {
                response.setStatusCode(HttpStatus.OK);
                return Mono.empty();
            }
        }
        return chain.filter(exchange);
    }
}
```

2.9 实现全局过滤器

```
/**
 * <p>
 * 描述：将登录用户的JWT转化成用户信息的全局过滤器
 * </p>
 * <p>版权：&copy;01星球</p>
 * <p>地址：01星球总部</p>
 * @author 阿伟学长
 * @version 1.0.0
 */
@Component
```

```java
@Slf4j
public class AuthGlobalFilter implements GlobalFilter, Ordered {
    @Override
    public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {
        String token = exchange.getRequest().getHeaders().getFirst("Authorization");
        if (StrUtil.isEmpty(token)) {
            return chain.filter(exchange);
        }
        //TODO: 判断凭证是否注销需要在此补充逻辑
        try {
            //从token中解析用户信息并设置到Header中去
            String realToken = token.replace("Bearer ", "");
            JWSObject jwsObject = JWSObject.parse(realToken);
            String userStr = jwsObject.getPayload().toString();
            log.info("AuthGlobalFilter.filter() user:{}", userStr);
            ServerHttpRequest request = exchange.getRequest().mutate().header("user", userStr).build();
            exchange = exchange.mutate().request(request).build();
        } catch (ParseException e) {
            e.printStackTrace();
        }
        return chain.filter(exchange);
    }
    @Override
    public int getOrder() {
        return 0;
    }
}
```

2.10 实现资源服务器配置

```java
/**
 * <p>
 * 描述：资源服务器配置
 * </p>
 * <p>版权：&copy;01星球</p>
 * <p>地址：01星球总部</p>
 * @author 阿伟学长
 * @version 1.0.0
 */
@AllArgsConstructor
@Configuration
@EnableWebFluxSecurity
public class ResourceServerConfig {
    private final AuthorizationManager authorizationManager;
    private final WhitePathConfig whitePathConfig;
    private final RestfulAccessDeniedHandler restfulAccessDeniedHandler;
    private final RestfulAuthenticationEntryPoint restfulAuthenticationEntryPoint;
    private final WhitePathFilter whitePathFilter;
    private final CorsFilter corsFilter;

    @Bean
```

```
22      public SecurityWebFilterChain springSecurityFilterChain(ServerHttpSecurity
    http) {
23          //跨域支持
24          http.cors().and().csrf().disable();
25          http.addFilterAt(corsFilter,
    SecurityWebFiltersOrder.SECURITY_CONTEXT_SERVER_WEB_EXCHANGE);
26          //jwt凭证转换器
27          http.oauth2ResourceServer().jwt()
28                  .jwtAuthenticationConverter(jwtAuthenticationConverter());
29          //自定义处理JWT请求头过期或签名错误的结果
30
     http.oauth2ResourceServer().authenticationEntryPoint(restfulAuthenticationEntryP
    oint);
31          //对白名单路径，直接移除JWT请求头
32          http.addFilterBefore(whitePathFilter,
    SecurityWebFiltersOrder.AUTHENTICATION);
33          http.authorizeExchange()
34                  //白名单配置
35                  .pathMatchers(ArrayUtil.toArray(whitePathConfig.getUrls(),
    String.class)).permitAll()
36                  //OPTIONS预检请求直接放行
37                  .pathMatchers(HttpMethod.OPTIONS).permitAll()
38                  //鉴权管理器配置
39                  .anyExchange().access(authorizationManager)
40                  .and().exceptionHandling()
41                  //处理未授权
42                  .accessDeniedHandler(restfulAccessDeniedHandler)
43                  //处理未认证
44                  .authenticationEntryPoint(restfulAuthenticationEntryPoint);
45          return http.build();
46      }
47
48      @Bean
49      public Converter<Jwt, ? extends Mono<? extends AbstractAuthenticationToken>>
50          jwtAuthenticationConverter() {
51          JwtGrantedAuthoritiesConverter jwtGrantedAuthoritiesConverter = new
    JwtGrantedAuthoritiesConverter();
52
     jwtGrantedAuthoritiesConverter.setAuthorityPrefix(AuthConstant.AUTHORITY_PREFIX)
    ;
53
     jwtGrantedAuthoritiesConverter.setAuthoritiesClaimName(AuthConstant.AUTHORITY_CL
    AIM_NAME);
54          JwtAuthenticationConverter jwtAuthenticationConverter = new
    JwtAuthenticationConverter();
55
     jwtAuthenticationConverter.setJwtGrantedAuthoritiesConverter(jwtGrantedAuthoriti
    esConverter);
56          return new
    ReactiveJwtAuthenticationConverterAdapter(jwtAuthenticationConverter);
57      }
58  }
```

3 实现登录认证功能

创建`login`模块，并按照后续流程配置完善模块。

3.1 配置依赖

```xml
<!-- spring mvc -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<!-- apis -->
<dependency>
    <groupId>com.zeroone.star</groupId>
    <artifactId>project-apis</artifactId>
</dependency>
<!-- common -->
<dependency>
    <groupId>com.zeroone.star</groupId>
    <artifactId>project-common</artifactId>
</dependency>
<!-- alibaba nacos discovery -->
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
</dependency>
<!-- alibaba nacos config -->
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
</dependency>
<!-- alibaba sentinel -->
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-sentinel</artifactId>
</dependency>
<!-- openfeign -->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

3.2 修改相关配置

`application.yaml`

```yaml
server:
  port: ${sp.login}
spring:
  application:
    name: ${sn.login}
  cloud:
    sentinel:
      transport:
        # 控制台地址
        dashboard: ${sentinel.dashboard}
        port: 8720
      eager: true

```

```yaml
# 开启sentinel声明式服务熔断
feign:
  sentinel:
    enabled: true

# 负载均衡配置
ribbon:
  MaxAutoRetries: 0 #（默认1次 不包括第一次）最大重试次数，当注册中心中可以找到服务，但是服务连不上时将会重试，如果注册中心中找不到服务则直接走断路器
  MaxAutoRetriesNextServer: 1 #（默认0次 不包括第一次）切换实例的重试次数
  OkToRetryOnAllOperations: false  #对所有操作请求都进行重试，如果是get则可以，如果是post，put等操作没有实现幂等的情况下是很危险的,所以设置为false
  ConnectTimeout: 5000  #（默认1s，即1000）请求连接的超时时间
  ReadTimeout: 5000 #（默认1s，即1000）请求处理的超时时间
```

3.3 初始化配置

```java
/**
 * <p>
 * 描述：Swagger配置
 * </p>
 * <p>版权：&copy;01星球</p>
 * <p>地址：01星球总部</p>
 * @author 阿伟学长
 * @version 1.0.0
 */
@Configuration
@EnableSwagger2WebMvc
public class SwaggerConfig {
    @Bean
    Docket loginApi() {
        return SwaggerCore.defaultDocketBuilder("登录模块", "com.zeroone.star.login.controller", "login");
    }
}
```

```java
/**
 * <p>
 * 描述：初始化自定义组件
 * </p>
 * <p>版权：&copy;01星球</p>
 * <p>地址：01星球总部</p>
 * @author 阿伟学长
 * @version 1.0.0
 */
@Configuration
@ComponentScan({
        "com.zeroone.star.project.components.jwt",
        "com.zeroone.star.project.components.user"
})
public class ComponentInit {
}
```

3.4 登录认证声明式服务

定义接口

```
/**
 * <p>
 * 描述：授权声明式服务接口
 * </p>
 * <p>版权：&copy;01星球</p>
 * <p>地址：01星球总部</p>
 * @author 阿伟学长
 * @version 1.0.0
 */
@FeignClient(value = "${sn.auth}", fallbackFactory =
OauthServiceFallbackFactory.class)
public interface OauthService {
    /**
     * 声明式调用授权服务
     * @param parameters 参数列表
     * @return 授权数据
     */
    @PostMapping("/oauth/token")
    JsonVO<Oauth2TokenDTO> postAccessToken(@RequestParam Map<String, String>
parameters);
}
```

服务降级处理

```
/**
 * <p>
 * 描述：授权服务降级实现
 * </p>
 * <p>版权：&copy;01星球</p>
 * <p>地址：01星球总部</p>
 * @author 阿伟学长
 * @version 1.0.0
 */
@AllArgsConstructor
public class OauthServiceImpl implements OauthService {
    private Throwable throwable;
    private void setMessage(JsonVO<?> vo) {
        if (throwable.getMessage() != null) {
            vo.setMessage(throwable.getMessage());
        } else {
            vo.setMessage(throwable.getClass().toGenericString());
        }
    }
    @Override
    public JsonVO<Oauth2TokenDTO> postAccessToken(Map<String, String> parameters)
{
        JsonVO<Oauth2TokenDTO> vo = JsonVO.fail(null);
        setMessage(vo);
        return vo;
    }
}
```

```
27
28  /**
29   * <p>
30   * 描述：授权服务异常回调工厂
31   * </p>
32   * <p>版权：&copy;01星球</p>
33   * <p>地址：01星球总部</p>
34   * @author 阿伟学长
35   * @version 1.0.0
36   */
37  @Component
38  public class OauthServiceFallbackFactory implements
    FallbackFactory<OauthServiceImpl> {
39      @Override
40      public OauthServiceImpl create(Throwable throwable) {
41          return new OauthServiceImpl(throwable);
42      }
43  }
```

3.5 实现登录认证接口

```
1   /**
2    * <p>
3    * 描述：登录接口
4    * </p>
5    * <p>版权：&copy;01星球</p>
6    * <p>地址：01星球总部</p>
7    * @author 阿伟学长
8    * @version 1.0.0
9    */
10  @RestController
11  @RequestMapping("login")
12  @Api(tags = "login")
13  public class LoginController implements LoginApis {
14      @Resource
15      OauthService oAuthService;
16      @Resource
17      UserHolder userHolder;
18      @ApiOperation(value = "授权登录")
19      @PostMapping("auth-login")
20      @Override
21      public JsonVO<Oauth2TokenDTO> authLogin(LoginDTO loginDTO) {
22          //TODO:未实现验证码验证
23          //账号密码认证
24          Map<String, String> params = new HashMap<>(5);
25          params.put("grant_type", "password");
26          params.put("client_id", loginDTO.getClientId());
27          params.put("client_secret", AuthConstant.CLIENT_PASSWORD);
28          params.put("username", loginDTO.getUsername());
29          params.put("password", loginDTO.getPassword());
30          return oAuthService.postAccessToken(params);
31          //TODO:未实现认证成功后如何实现注销凭证（如记录凭证到内存数据库）
32      }
33      @ApiOperation(value = "刷新登录")
34      @PostMapping("refresh-token")
35      @Override
```

```
    public JsonVO<Oauth2TokenDTO> refreshToken(Oauth2TokenDTO oauth2TokenDTO) {
        //TODO:未实现注销凭证验证
        Map<String, String> params = new HashMap<>(4);
        params.put("grant_type", "refresh_token");
        params.put("client_id", oauth2TokenDTO.getClientId());
        params.put("client_secret", AuthConstant.CLIENT_PASSWORD);
        params.put("refresh_token", oauth2TokenDTO.getRefreshToken());
        return oAuthService.postAccessToken(params);
    }
    @ApiOperation(value = "获取当前用户")
    @GetMapping("current-user")
    @Override
    public JsonVO<LoginVO> getCurrUser() {
        UserDTO currentUser;
        try {
            currentUser = userHolder.getCurrentUser();
        } catch (Exception e) {
            return JsonVO.create(null, ResultStatus.FAIL.getCode(),
e.getMessage());
        }
        if (currentUser == null) {
            return JsonVO.fail(null);
        } else {
            //TODO:这里需要根据业务逻辑接口，重新实现
            LoginVO vo = new LoginVO();
            BeanUtil.copyProperties(currentUser, vo);
            return JsonVO.success(vo);
        }
    }
    @ApiOperation(value = "退出登录")
    @GetMapping("logout")
    @Override
    public JsonVO<String> logout() {
        //TODO:登出逻辑，需要配合登录逻辑实现
        return null;
    }
    @ApiOperation(value = "获取菜单")
    @GetMapping("get-menus")
    @Override
    public JsonVO<List<MenuTreeVO>> getMenus() throws Exception {
        return null;
    }
}
```

4 实现菜单获取

4.1 添加依赖

`login`模块添加依赖

```
<!— mp —>
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
</dependency>
```

```xml
<!-- druid -->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid-spring-boot-starter</artifactId>
</dependency>
<!-- mysql driver -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
```

4.2 初始化配置

`login`模块添加配置

```java
/**
 * <p>
 * 描述：初始化mybatis
 * </p>
 * <p>版权：&copy;01星球</p>
 * <p>地址：01星球总部</p>
 * @author 阿伟学长
 * @version 1.0.0
 */
@Configuration
@ComponentScan("com.zeroone.star.project.config.mybatis")
public class MyBaitsInit {}
```

4.4 菜单数据操作

使用代码生成工具生成menu表对于的模块代码，代码生成到`login`模块中。

生成完成删除掉`MenuController`

4.4.1 实现数据查询

```java
@Mapper
public interface MenuMapper extends BaseMapper<Menu> {
    /**
     * 通过角色名获取对应的菜单资源
     * @param roleName 角色名
     * @return 返回菜单列表
     */
    List<Menu> selectByRoleName(String roleName);
}
```

```xml
<mapper namespace="com.zeroone.star.login.mapper.MenuMapper">
    <resultMap id="BaseResultMap" type="com.zeroone.star.login.entity.Menu">
        <id column="id" jdbcType="INTEGER" property="id" />
        <result column="name" jdbcType="VARCHAR" property="name" />
        <result column="link_url" jdbcType="VARCHAR" property="linkUrl" />
        <result column="path" jdbcType="VARCHAR" property="path" />
        <result column="priority" jdbcType="INTEGER" property="priority" />
```

```
        <result column="icon" jdbcType="VARCHAR" property="icon" />
        <result column="description" jdbcType="VARCHAR" property="description" />
        <result column="parent_menu_id" jdbcType="INTEGER"
property="parentMenuId" />
        <result column="level" jdbcType="INTEGER" property="level" />
        <result column="is_enable" jdbcType="TINYINT" property="isEnable" />
    </resultMap>
    <select id="selectByRoleName" resultMap="BaseResultMap">
      SELECT id,`name`,path,icon,parent_menu_id FROM menu WHERE path IS NOT NULL
AND id IN (SELECT menu_id FROM role_menu WHERE role_id=(SELECT id FROM role WHERE
keyword=#{roleName})) ORDER BY priority
    </select>
</mapper>
```

4.4.2 实现服务接口

```
public interface IMenuService extends IService<Menu> {
    /**
     * 通过角色名称获取，菜单资源
     * @param roleNames 角色名称
     * @return 返回菜单列表
     */
    List<MenuTreeVO> listMenuByRoleName(List<String> roleNames);
}
```

```
/**
 * <p>
 * 描述：定义一个Menu数据到MenuTreeNode的属性映射配器
 * </p>
 * <p>版权：&copy;01星球</p>
 * <p>地址：01星球总部</p>
 * @author 阿伟学长
 * @version 1.0.0
 */
class MenuTreeNodMapper implements TreeNodeMapper<Menu> {
    @Override
    public TreeNode objectMapper(Menu menu) {
        MenuTreeVO treeNode = new MenuTreeVO();
        // 首先设置TreeNode计算层数使用属性
        treeNode.setTnId(menu.getId().toString());
        if (menu.getParentMenuId() == null) {
            treeNode.setTnPid(null);
        } else {
            treeNode.setTnPid(menu.getParentMenuId().toString());
        }
        // 设置扩展属性
        treeNode.setId(menu.getId());
        treeNode.setIcon(menu.getIcon());
        treeNode.setText(menu.getName());
        treeNode.setHref(menu.getPath());
        treeNode.setPid(menu.getParentMenuId());
        return treeNode;
    }
}

```

```java
/**
 * <p>
 * 描述：菜单服务实现类
 * </p>
 * <p>版权：&copy;01星球</p>
 * <p>地址：01星球总部</p>
 * @author 阿伟学长
 * @version 1.0.0
 */
@Service
public class MenuServiceImpl extends ServiceImpl<MenuMapper, Menu> implements
IMenuService {
    @Override
    public List<MenuTreeVO> listMenuByRoleName(List<String> roleNames) {
        //1 定义一个存储数据库查询菜单数据的容器
        List<Menu> menus = new ArrayList<>();
        //2 遍历获取角色获取所有的菜单列表
        roleNames.forEach(roleName -> {
            //通过角色名获取菜单列表
            List<Menu> tMenus = baseMapper.selectByRoleName(roleName);
            if (tMenus != null && !tMenus.isEmpty()) {
                menus.addAll(tMenus);
            }
        });
        //3 转换树形结构并返回
        return TreeUtils.listToTree(menus, new MenuTreeNodMapper());
    }
}
```

4.4.3 实现登录控制器

```java
@Resource
IMenuService menuService;

@ApiOperation(value = "获取菜单")
@GetMapping("get-menus")
@Override
public JsonVO<List<MenuTreeVO>> getMenus() throws Exception {
    //TODO:未实现根据实际数据库设计业务逻辑，下面逻辑属于示例逻辑
    //1 获取当前用户
    UserDTO currentUser = userHolder.getCurrentUser();
    //2 获取当前用户拥有的菜单
    List<MenuTreeVO> menus =
menuService.listMenuByRoleName(currentUser.getRoles());
    return JsonVO.success(menus);
}
```

3 与前端联调

- 使用apipost演示
- 演示knife4j填充凭证
- 启动前端联调
- 移植网关配置