spring-boot security + jwt + redis + swagger 详细配置

作者：jockming112
原文链接：https://ld246.com/article/1611372926794
来源网站：链滴

项目依赖

安全配置主要依赖：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
    <version>2.3.3.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-configuration-processor</artifactId>
    <version>2.3.3.RELEASE</version>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
    <version>2.3.3.RELEASE</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>
```

Security核心配置

```
import com.base.common.utils.Md5Utils;
import com.base.web.config.security.filter.JwtAuthenticationTokenFilter;
import com.base.web.config.security.handler.*;
import com.base.web.service.UserService;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builders.Authenticatio
ManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfig
rerAdapter;
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.authentication.UsernamePasswordAuthenticationFil
er;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;

import javax.annotation.Resource;
import java.time.Duration;

import static java.util.Collections.singletonList;

/**
 * Security config
 *
 * @author ming
 * @version 1.0.0
 * @date 2021/1/13 15:04
 **/
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Resource
    private UserService userService;

    @Resource
    private CustomAuthenticationFailureHandler authenticationFailureHandler;

    @Resource
    private CustomAuthenticationSuccessHandler authenticationSuccessHandler;

    @Resource
    private CustomAuthenticationEntryPointHandler authenticationEntryPointHandler;

    @Resource
    private CustomLogoutSuccessHandler logoutSuccessHandler;

    @Resource
    private CustomAccessDeniedHandler accessDeniedHandler;

    @Resource
    private JwtAuthenticationTokenFilter authenticationTokenFilter;

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

    /**
     * Implementation of user defined password encryption
     */
    @Bean
```

```
public PasswordEncoder customPasswordEncoder() {
    return new CustomPasswordEncoder();
}

/**
 * 跨域配置
 */
@Bean
public CorsConfigurationSource corsConfigurationSource() {
    UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
    CorsConfiguration configuration = new CorsConfiguration();
    configuration.setAllowCredentials(true);
    configuration.setAllowedOrigins(singletonList("*"));
    configuration.setAllowedMethods(singletonList("*"));
    configuration.setAllowedHeaders(singletonList("*"));
    configuration.setMaxAge(Duration.ofHours(1));
    source.registerCorsConfiguration("/**", configuration);
    return source;
}

/**
 * URL whitelist
 */
private static final String[] AUTH_WHITELIST = {
        // -- swagger ui
        "/v2/api-docs",
        "/swagger-resources/**",
        "/swagger-resources/configuration/ui",
        "/swagger-resources/configuration/security",
        "/swagger-ui.html/**",
        "/css/**",
        "/js/**",
        "/images/**",
        "/webjars/**",
        "**/favicon.ico",
        "/",
        "/csrf",
        // -- user & common
        "/user/registry",
        "/user/login",
        "/user/logout",
        "/common"
};

/**
 * 核心配置
 *
 * @param http HttpSecurity
 * @throws Exception e
 */
@Override
protected void configure(HttpSecurity http) throws Exception {

    http.formLogin()
```

```
            //登录处理接口
            //.loginPage("/login_page")
            .loginProcessingUrl("/user/login")
            //定义登录时，用户名的 key，默认为 username
            .usernameParameter("username")
            //定义登录时，用户密码的 key，默认为 password
            .passwordParameter("password")
            //登录成功处理
            .successHandler(authenticationSuccessHandler)
            //登录失败处理
            .failureHandler(authenticationFailureHandler)
            .permitAll()
            .and()
            //登出处理
            .logout()
            .logoutUrl("/user/logout")
            //清除身份信息
            .clearAuthentication(true)
            .logoutSuccessHandler(logoutSuccessHandler)
            .permitAll()
            .and()
            //开启跨域
            .cors()
            .configurationSource(corsConfigurationSource())
            .and()
            //取消跨站请求伪造防护
            .csrf().disable()
            //基于jwt，不需要session
            .sessionManagement()
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            //Url白名单
            .and()
            .authorizeRequests()
            .antMatchers(AUTH_WHITELIST)
            .permitAll()
            .anyRequest()
            .authenticated();

        // 禁用缓存
        http.headers().cacheControl();
        // 添加JWT filter
        http.addFilterBefore(authenticationTokenFilter, UsernamePasswordAuthenticationFilter.c
ass);

        //无token或者token无效的相关处理
        http.exceptionHandling()
            .authenticationEntryPoint(authenticationEntryPointHandler)
            .accessDeniedHandler(accessDeniedHandler);

    }

    /**
     * 配置用户签名服务 主要是user-details 机制
     * (可以采用内存、数据库、loap、自定义) 这里是自定义
```

```
     *
     * @param auth 签名管理器构造器，用于构建用户具体权限控制
     * @throws Exception e
     */
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userService).passwordEncoder(passwordEncoder());
    }

    /**
     * 自定义密码加密实现(md5)
     */
    private static class CustomPasswordEncoder implements PasswordEncoder {

        @Override
        public String encode(CharSequence plaintext) {
            return Md5Utils.md5(String.valueOf(plaintext));
        }

        @Override
        public boolean matches(CharSequence plaintext, String cipher) {
            return cipher.equals(Md5Utils.md5(String.valueOf(plaintext)));
        }
    }

}
```

2.处理器CustomAccessDeniedHandler.java

```
import com.base.common.ApiResponse;
import com.base.web.config.security.utils.WebMvcWriter;
import lombok.extern.slf4j.Slf4j;
import org.springframework.http.HttpStatus;
import org.springframework.security.access.AccessDeniedException;
import org.springframework.security.web.access.AccessDeniedHandler;
import org.springframework.stereotype.Component;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
 * 权限不足异常处理(用来解决认证过的用户访问无权限资源时的异常)
 *
 * @author ming
 * @version 1.0.0
 * @date 2021/1/15 14:35
 **/
@Component
@Slf4j
public class CustomAccessDeniedHandler implements AccessDeniedHandler {

    @Override
    public void handle(HttpServletRequest request, HttpServletResponse response, AccessDeni
```

```
dException e) throws IOException {
        log.error("接口-> {} <-访问权限不足，请求失败: {}", request.getRequestURI(), e);
        response.setContentType("application/json;charset=UTF-8");
        response.setStatus(HttpStatus.FORBIDDEN.value());
        ApiResponse resp = ApiResponse.failed(HttpStatus.FORBIDDEN.value(), "权限不足!!");
        new WebMvcWriter().out(response, resp);
    }
}
```

3.CustomAuthenticationEntryPointHandler.java

```
import com.base.common.ApiResponse;
import com.base.web.config.security.utils.JwtTokenUtils;
import com.base.web.config.security.utils.WebMvcWriter;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.lang3.StringUtils;
import org.springframework.http.HttpStatus;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.AuthenticationEntryPoint;
import org.springframework.stereotype.Component;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
 * 令牌异常处理
 * (用来解决匿名用户访问无权限资源时的异常)
 *
 * @author ming
 * @version 1.0.0
 * @date 2021/1/15 14:36
 **/
@Component
@Slf4j
public class CustomAuthenticationEntryPointHandler implements AuthenticationEntryPoint {

    @Override
    public void commence(HttpServletRequest request, HttpServletResponse response, Authen
icationException e) throws IOException {
        String token = request.getHeader(JwtTokenUtils.TOKEN_HEADER);
        response.setContentType("application/json;charset=utf-8");
        int status = HttpStatus.UNAUTHORIZED.value();
        response.setStatus(status);
        if (StringUtils.isEmpty(token)) {
            String message = "未设置Token!!";
            log.error("接口访问-> {} <-请求失败: {} , {}", request.getRequestURI(), e, message);
            new WebMvcWriter().out(response, ApiResponse.failed(10000 + status, message));
            return;
        }
        if (StringUtils.startsWith(token, JwtTokenUtils.TOKEN_PREFIX)) {
            token = StringUtils.substring(token, JwtTokenUtils.TOKEN_PREFIX.length());
        }
        String username = JwtTokenUtils.getUsername(token);
```

```
        if (StringUtils.isEmpty(username)) {
            String message = "访问令牌不合法";
            log.error("接口访问-> {} <-请求失败: {} , {}", request.getRequestURI(), e, message);
            new WebMvcWriter().out(response, ApiResponse.failed(10000 + status, message));
        }
    }
}
```

4.CustomAuthenticationFailureHandler.java

```
import com.base.common.ApiResponse;
import com.base.web.config.security.utils.WebMvcWriter;
import lombok.extern.slf4j.Slf4j;
import org.springframework.http.HttpStatus;
import org.springframework.security.authentication.*;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.web.authentication.AuthenticationFailureHandler;
import org.springframework.stereotype.Component;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
 * 登录失败处理
 *
 * @author ming
 * @version 1.0.0
 * @date 2021/1/21 14:37
 **/
@Slf4j
@Component
public class CustomAuthenticationFailureHandler implements AuthenticationFailureHandler {

    @Override
    public void onAuthenticationFailure(HttpServletRequest request, HttpServletResponse resp
nse, AuthenticationException e) throws IOException {
        log.error("登录认证失败: {}", e);
        response.setContentType("application/json;charset=utf-8");
        response.setStatus(HttpStatus.UNAUTHORIZED.value());
        ApiResponse resp;
        if (e instanceof BadCredentialsException || e instanceof UsernameNotFoundException) {
            resp = ApiResponse.failed(1111, "账户名或者密码输入错误!");
        } else if (e instanceof LockedException) {
            resp = ApiResponse.failed(1112, "账户被锁定，请联系管理员!");
        } else if (e instanceof CredentialsExpiredException) {
            resp = ApiResponse.failed(1113, "密码过期，请联系管理员!");
        } else if (e instanceof AccountExpiredException) {
            resp = ApiResponse.failed(1114, "账户过期，请联系管理员!");
        } else if (e instanceof DisabledException) {
            resp = ApiResponse.failed(1115, "账户被禁用，请联系管理员!");
        } else {
            resp = ApiResponse.failed(1116, "登录失败!");
```

```
        }
        new WebMvcWriter().out(response, resp);
    }
}
```

5.CustomAuthenticationSuccessHandler.java

```
import cn.hutool.core.date.DatePattern;
import com.baomidou.mybatisplus.core.conditions.update.UpdateWrapper;
import com.base.common.ApiResponse;
import com.base.common.entity.sys.SysUser;
import com.base.common.handle.RequestHolder;
import com.base.common.utils.IpUtil;
import com.base.common.utils.JsonUtils;
import com.base.common.utils.RedisCacheUtil;
import com.base.web.config.security.utils.JwtTokenUtils;
import com.base.web.config.security.utils.WebMvcWriter;
import com.base.web.entity.vo.UserInfoVO;
import com.base.web.service.SysMenuService;
import com.base.web.service.UserService;
import com.github.hiwepy.ip2region.spring.boot.IP2regionTemplate;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.lang3.ObjectUtils;
import org.springframework.http.HttpStatus;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.web.authentication.AuthenticationSuccessHandler;
import org.springframework.stereotype.Component;

import javax.annotation.Resource;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.HashMap;
import java.util.Map;

/**
 * 登录成功处理
 *
 * @author ming
 * @version 1.0.0
 * @date 2021/1/21 14:49
 **/
@Slf4j
@Component
public class CustomAuthenticationSuccessHandler implements AuthenticationSuccessHandler
{

    @Resource
    private UserService userService;

    @Resource
```

```
    private RedisCacheUtil redisCacheUtil;

    @Resource
    private IP2regionTemplate ip2region;

    @Resource
    private SysMenuService menuService;

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse res
onse, Authentication auth) throws IOException {
        String username = auth.getName();
        log.info("用户-> {} <-登录成功", username);
        response.setContentType("application/json;charset=utf-8");
        response.setStatus(HttpStatus.OK.value());

        SysUser user = userService.getSysUser(username);

        try {
            String jwtToken = JwtTokenUtils.createToken(userService.loadUserByUsername(usern
me), false);
            // 生成令牌并设置到响应头中
            // response.setHeader(JwtTokenUtils.TOKEN_HEADER, token);

            // refresh user info
            user.setLoginIp(IpUtil.getIp(new RequestHolder().getRequest()));
            user.setLoginAddress(ip2region.getRegion(user.getLoginIp()));
            user.setModifyTime(LocalDateTime.now().format(DateTimeFormatter.ofPattern(DatePa
tern.NORM_DATETIME_PATTERN)));

            if (redisCacheUtil.hasKey(jwtToken)) {
                //获得redis中用户的token刷新时效
                String userJson = redisCacheUtil.getString(jwtToken);
                UserInfoVO userInfoVO = JsonUtils.parseJsonToObj(userJson, UserInfoVO.class);
                UserDetails userDetails = userService.loadUserByUsername(username);
                if (ObjectUtils.isNotEmpty(userInfoVO)) {
                    // 说明用户存在且未过期
                    jwtToken = JwtTokenUtils.createToken(userDetails, false);
                    // TODO: 2021/1/22 这里可能还需要更新登录地点和登录IP
                    userInfoVO.setJwtToken(jwtToken);
                    // 刷新缓存
                    redisCacheUtil.delete(jwtToken);
                    redisCacheUtil.setString(jwtToken, JsonUtils.parseObjToJson(userInfoVO), JwtTok
nUtils.EXPIRATION);
                    Map<String, String> map = new HashMap<>(1);
                    map.put(JwtTokenUtils.TOKEN_HEADER, JwtTokenUtils.TOKEN_PREFIX + jwtToke
);
                    new WebMvcWriter().out(response, ApiResponse.successful(map));
                    return;
                }

            }

            // 修改登录IP和登录地址
```

```
            user.setLoginIp(user.getLoginIp());
            user.setLoginAddress(user.getLoginAddress());

            if (!userService.update(user, new UpdateWrapper<>(user))) {
                log.error("用户信息更新失败!!");
                ApiResponse.failed(2, "用户信息更新失败!!");
            }

            // 验证通过返回用户部分信息 以及 拥有的角色和菜单
            UserInfoVO userVO = userService.userInfo(user.getUsername());
            userVO.setMyMenus(userVO.getRoles());
            userVO.setUserRoutes(menuService.getMenuTree(userVO.getMenus()));
            userVO.setModifyTime(user.getModifyTime());
            userVO.setJwtToken(jwtToken);
            redisCacheUtil.setString(jwtToken, JsonUtils.parseObjToJson(userVO), JwtTokenUtils.E
PIRATION);

            Map<String, String> map = new HashMap<>(1);
            map.put(JwtTokenUtils.TOKEN_HEADER, JwtTokenUtils.TOKEN_PREFIX + jwtToken);
            new WebMvcWriter().out(response, ApiResponse.successful(map));
        } catch (Exception e) {
            log.error("登录认证失败: {}", e);
            new WebMvcWriter().out(response, ApiResponse.failed(HttpStatus.UNAUTHORIZED.va
ue(), "创建token失败，请与管理员联系"));
        }
    }
}
```

6.CustomLogoutSuccessHandler.java

```
import com.base.common.ApiResponse;
import com.base.common.entity.sys.SysUser;
import com.base.common.enumeration.ResponseMessageEnum;
import com.base.common.exception.BusinessException;
import com.base.common.utils.RedisCacheUtil;
import com.base.web.config.security.utils.JwtTokenUtils;
import com.base.web.config.security.utils.WebMvcWriter;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.lang3.StringUtils;
import org.springframework.http.HttpStatus;
import org.springframework.security.core.Authentication;
import org.springframework.security.web.authentication.logout.LogoutSuccessHandler;
import org.springframework.stereotype.Component;

import javax.annotation.Resource;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
 * 注销登录处理
 *
 * @author ming
 * @version 1.0.0
```

```
 * @date 2021/1/21 14:53
 **/
@Slf4j
@Component
public class CustomLogoutSuccessHandler implements LogoutSuccessHandler {

    @Resource
    private RedisCacheUtil redisCacheUtil;

    @Override
    public void onLogoutSuccess(HttpServletRequest request, HttpServletResponse response,
uthentication auth) throws IOException {
        String token = request.getHeader(JwtTokenUtils.TOKEN_HEADER);
        if (token != null && StringUtils.startsWith(token, JwtTokenUtils.TOKEN_PREFIX)) {
            token = StringUtils.substring(token, JwtTokenUtils.TOKEN_PREFIX.length());
        } else {
            log.error("登出失败，未设置Token");
            throw new BusinessException("登出失败");
        }
        String username = JwtTokenUtils.getUsername(token);
        log.debug(String.format("%s 用户正在登出!!", username));
        if (redisCacheUtil.hasKey(token)) {
            redisCacheUtil.delete(token);
        }
        log.info("用户-> {} <-登出成功", ((SysUser) auth).getUsername());
        response.setContentType("application/json;charset=utf-8");
        response.setStatus(HttpStatus.OK.value());
        //response.sendRedirect("/login_page");
        new WebMvcWriter().out(response, ApiResponse.successful(ResponseMessageEnum.US
R_LOGOUT_SUCCESS.getCode(), ResponseMessageEnum.USER_LOGOUT_SUCCESS.getMsg()));
    }
}
```

过滤器

JwtAuthenticationTokenFilter.java

```
import com.base.common.ApiResponse;
import com.base.common.enumeration.ResponseMessageEnum;
import com.base.common.utils.JsonUtils;
import com.base.common.utils.RedisCacheUtil;
import com.base.web.config.security.utils.JwtTokenUtils;
import com.base.web.config.security.utils.WebMvcWriter;
import com.base.web.entity.vo.UserInfoVO;
import com.base.web.service.UserService;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.lang3.ObjectUtils;
import org.apache.commons.lang3.StringUtils;
import org.springframework.http.HttpStatus;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
```

```
import org.springframework.web.filter.OncePerRequestFilter;

import javax.annotation.Resource;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
 * Jwt Token Effectiveness Filter
 *
 * @author ming
 * @version 1.0.0
 * @date 2021/1/15 14:35
 **/
@Slf4j
@Component
public class JwtAuthenticationTokenFilter extends OncePerRequestFilter {

    @Resource
    private UserService userService;

    @Resource
    private RedisCacheUtil redisCacheUtil;

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
FilterChain filterChain) throws ServletException, IOException {
        String token = request.getHeader(JwtTokenUtils.TOKEN_HEADER);
        if (token != null && StringUtils.startsWith(token, JwtTokenUtils.TOKEN_PREFIX)) {
            token = StringUtils.substring(token, JwtTokenUtils.TOKEN_PREFIX.length());
        } else {
            filterChain.doFilter(request, response);
            return;
        }
        String username = JwtTokenUtils.getUsername(token);

        if (username != null && redisCacheUtil.hasKey(token)) {
            //获得redis中用户的token刷新时效
            String userJson = redisCacheUtil.getString(token);
            UserInfoVO userInfoVO = JsonUtils.parseJsonToObj(userJson, UserInfoVO.class);
            UserDetails userDetails = userService.loadUserByUsername(username);
            if (ObjectUtils.isNotEmpty(userInfoVO)) {
                // 说明用户存在且未过期
                String jwtToken = JwtTokenUtils.createToken(userDetails, false);
                // TODO: 2021/1/22 这里可能还需要更新登录地点和登录IP
                userInfoVO.setJwtToken(jwtToken);
                // 刷新缓存
                redisCacheUtil.delete(token);
                redisCacheUtil.setString(jwtToken, JsonUtils.parseObjToJson(userInfoVO), JwtToken
tils.EXPIRATION);

                response.setHeader(JwtTokenUtils.TOKEN_HEADER, JwtTokenUtils.TOKEN_PREFIX +
```

```
wtToken);
            }

        }
        try {
            if (username != null && SecurityContextHolder.getContext().getAuthentication() == n
ll) {
                UserDetails userDetails = userService.loadUserByUsername(username);

                if (JwtTokenUtils.validateToken(token, userDetails)) {
                    UsernamePasswordAuthenticationToken authentication =
                            new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.g
tAuthorities());
                    authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(req
est));

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

            }
        } catch (Exception e) {
            log.error("接口访问-> {} <-请求失败,不合法的令牌!!", request.getRequestURI());
            response.setContentType("application/json;charset=UTF-8");
            response.setStatus(HttpStatus.UNAUTHORIZED.value());
            new WebMvcWriter().out(response, ApiResponse.failed(HttpStatus.UNAUTHORIZED.va
ue(), ResponseMessageEnum.USER_TOKEN_INVALID.getMsg()));
            return;
        }

        filterChain.doFilter(request, response);
    }

}
```

工具类

1.JwtTokenUtils.java

```
import com.base.common.exception.BusinessException;
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.AuthorityUtils;
import org.springframework.security.core.userdetails.UserDetails;

import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Set;

/**
 * JWT utils
 *
```

```
 * @author : ming
 * @version 1.0
 */
public class JwtTokenUtils {

    public static final String TOKEN_HEADER = "Authorization";

    public static final String TOKEN_PREFIX = "Bearer ";
    /**
     * 密钥key
     */
    private static final String SECRET = "security";

    /**
     * JWT的发行人
     */
    private static final String ISS = "ming";

    /**
     * 自定义用户信息
     */
    private static final String ROLE_CLAIMS = "rol";

    /**
     * 过期时间是3600秒，既是1个小时
     */
    public static final long EXPIRATION = 3600L * 1000;

    /**
     * 选择了记住我之后的过期时间为7天
     */
    public static final long EXPIRATION_REMEMBER = 604800L * 1000;

    /**
     * 创建token
     *
     * @param details      登录名
     * @param isRememberMe 是否记住我
     * @return String
     */
    public static String createToken(UserDetails details, boolean isRememberMe) throws Busin
ssException {
        // 如果选择记住我，则token的过期时间为
        long expiration = isRememberMe ? EXPIRATION_REMEMBER : EXPIRATION;

        HashMap<String, Object> map = new HashMap<>();
        // 角色名字,"ROLE_"开头的权限才是用户角色
        map.put(ROLE_CLAIMS, details.getAuthorities());
        return Jwts.builder()
                // 加密算法
                .signWith(SignatureAlgorithm.HS512, SECRET)
                // 自定义信息
                .setClaims(map)
                // jwt发行人
```

```
            .setIssuer(ISS)
            // jwt面向的用户
            .setSubject(details.getUsername())
            // jwt发行时间
            .setIssuedAt(new Date())
            // jwt过期时间
            .setExpiration(new Date(System.currentTimeMillis() + expiration))
            .compact();
    }

    /**
     * 从token获取用户信息
     *
     * @param token token
     * @return username
     */
    public static String getUsername(String token) throws BusinessException {
        return getTokenBody(token).getSubject();
    }

    /**
     * 从token中获取用户角色
     *
     * @param token token
     * @return GrantedAuthority
     */
    @SuppressWarnings("unchecked")
    public static Set<String> getUserRole(String token) throws BusinessException {
        List<GrantedAuthority> userAuthorities = (List<GrantedAuthority>) getTokenBody(toke
).get(ROLE_CLAIMS);
        return AuthorityUtils.authorityListToSet(userAuthorities);
    }

    /**
     * 是否已过期
     *
     * @param token token
     * @return boolean
     */
    private static boolean isExpiration(String token) throws BusinessException {
        return getTokenBody(token).getExpiration().before(new Date());
    }

    /**
     * 解析token
     *
     * @param token token
     * @return Claims
     * @throws BusinessException e
     */
    private static Claims getTokenBody(String token) throws BusinessException {

        return Jwts.parser().setSigningKey(SECRET).parseClaimsJws(token).getBody();
    }
```

```
    /**
     * 验证token
     *
     * @param token       token
     * @param userDetails user
     * @return boolean
     */
    public static boolean validateToken(String token, UserDetails userDetails) throws BusinessE
ception {
        final String username = getUsername(token);
        return (username.equals(userDetails.getUsername()) && !isExpiration(token));
    }

}
```

2.WebMvcWriter.java

```
import com.base.common.ApiResponse;
import com.fasterxml.jackson.databind.ObjectMapper;

import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

/**
 * Web mvc out
 *
 * @author ming
 * @version 1.0.0
 * @date 2021/1/21 14:42
 **/
public class WebMvcWriter {

    public void out(HttpServletResponse response, ApiResponse resp) throws IOException {
        ObjectMapper om = new ObjectMapper();
        PrintWriter out = response.getWriter();
        out.write(om.writeValueAsString(resp));
        out.flush();
        out.close();
    }
}
```

权限判断

CustomPermissionEvaluator.java

```
import lombok.extern.slf4j.Slf4j;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.access.PermissionEvaluator;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;

import java.io.Serializable;
```

```
/**
 * 自定义权限判断
 * 注解使用方式(@PreAuthorize("hasPermission('user','add')" ))
 *
 * @author ming
 * @version 1.0.0
 * @date 2021/1/13 18:52
 **/
@Configuration
@Slf4j
public class CustomPermissionEvaluator implements PermissionEvaluator {

    @Override
    public boolean hasPermission(Authentication auth, Object o, Object o1) {
        boolean access = false;
        log.info(auth.getPrincipal().toString());
        // 权限判断
        if (auth.getPrincipal().toString().compareToIgnoreCase("anonymousUser") != 0) {
            String privilege = o + ":" + o1;
            for (GrantedAuthority authority : auth.getAuthorities()) {
                if (privilege.equalsIgnoreCase(authority.getAuthority())) {
                    access = true;
                    break;
                }
            }
        }
        return access;
    }

    @Override
    public boolean hasPermission(Authentication authentication, Serializable serializable, String
s, Object o) {
        return false;
    }
}
```

MethodSecurityConfig.java

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.access.expression.method.DefaultMethodSecurityExpress
onHandler;
import org.springframework.security.access.expression.method.MethodSecurityExpressionHa
dler;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMe
thodSecurity;
import org.springframework.security.config.annotation.method.configuration.GlobalMethodS
curityConfiguration;

/**
 * 全局方法安全配置(权限评估)
 *
 * @author ming
 * @version 1.0.0
```

```
 * @date 2021/1/21 15:49
 **/
@EnableGlobalMethodSecurity(prePostEnabled = true, securedEnabled = true)
public class MethodSecurityConfig extends GlobalMethodSecurityConfiguration {
    private CustomPermissionEvaluator customPermissionEvaluator;

    @Autowired
    public void setCustomPermissionEvaluator(CustomPermissionEvaluator customPermissionE
aluator) {
        this.customPermissionEvaluator = customPermissionEvaluator;
    }

    @Override
    protected MethodSecurityExpressionHandler createExpressionHandler() {
        DefaultMethodSecurityExpressionHandler expressionHandler = new DefaultMethodSecur
tyExpressionHandler();
        expressionHandler.setPermissionEvaluator(customPermissionEvaluator);
        return expressionHandler;
    }
}
```

同意响应类

ApiResponse.java

```
import com.base.common.enumeration.ResponseMessageEnum;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.AllArgsConstructor;
import lombok.Data;

import java.io.Serializable;

/**
 * Global Return
 *
 * @author ming
 * @version 1.0.0
 * @since 2020/4/14 14:30
 **/
@Data
@ApiModel("统一响应类")
@AllArgsConstructor
public class ApiResponse<T> implements Serializable {

    private static final long serialVersionUID = 9024462031856060619L;
    @ApiModelProperty("状态码")
    private Integer code;
    @ApiModelProperty("信息")
    private String msg;
    @ApiModelProperty("数据")
    private T data;

    private ApiResponse() {
```

```
        super();
    }

    public static <T> ApiResponse<T> successful(Integer code, String msg) {
        return getApiResponseWithoutData(code, msg);
    }

    public static <T> ApiResponse<T> successful(T data) {
        ApiResponse<T> apiResponse = new ApiResponse<>();
        apiResponse.setCode(ResponseMessageEnum.OPERATION_SUCCESS.getCode());
        apiResponse.setMsg("");
        apiResponse.setData(data);
        return apiResponse;
    }

    public static <T> ApiResponse<T> successful(Integer code, String msg, T data) {
        return getApiResponseWithData(code, msg, data);
    }

    public static <T> ApiResponse<T> failed(Integer code, String msg) {
        return getApiResponseWithoutData(code, msg);
    }

    private static <T> ApiResponse<T> getApiResponseWithoutData(Integer code, String msg)
{
        ApiResponse<T> apiResponse = new ApiResponse<>();
        apiResponse.setCode(code);
        apiResponse.setMsg(msg);
        apiResponse.setData(null);
        return apiResponse;
    }

    private static <T> ApiResponse<T> getApiResponseWithData(Integer code, String msg, T
ata) {
        ApiResponse<T> apiResponse = new ApiResponse<>();
        apiResponse.setCode(code);
        apiResponse.setMsg(msg);
        apiResponse.setData(data);
        return apiResponse;
    }
}
```

RedisUtils

```
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.lang3.StringUtils;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.stereotype.Component;

import javax.annotation.Resource;
import java.util.*;
import java.util.concurrent.TimeUnit;

/**
```

```
 * redis 缓存工具类
 *
 * @author ming
 * @version 1.0.0
 * @since 2019/8/22 17:27
 **/
@Slf4j
@Component
public class RedisCacheUtil {

    // 注： 这里不能用Autowired按类型装配注入,必须用@Resource
    // StringRedisTemplate默认采用的是String的序列化策略,
    // RedisTemplate默认采用的是JDK的序列化策略，保存的key和value都是采用此策略序列化保存

    /**
     * 注入redis
     */
    @Resource
    private RedisTemplate<String, Object> redisTemplate;

    /**
     * 生成业务查询使用的缓存key , 带分页
     *
     * @param username 用户名
     * @param obj      业务相关命名
     * @return key
     */
    public String generateCachePageKey(String username, String obj, Integer pageNo, Integer
ageSize) {
        return generateCacheKey(username, obj) + "_page[" + pageNo + "," + pageSize + "]";
    }

    /**
     * 生成业务查询使用的缓存key
     *
     * @param username 用户名
     * @param obj      业务相关命名
     * @return key
     */
    public String generateCacheKey(String username, String obj) {
        if (StringUtils.isBlank(username) || StringUtils.isBlank(obj)) {
            return null;
        }
        return username + "_" + obj;
    }

    /**
     * 指定缓存失效时间
     *
     * @param key  键
     * @param time 时间(秒)
     */
    private void expire(String key, long time) {
        if (time > 0) {
```

```
            redisTemplate.expire(key, time, TimeUnit.SECONDS);
        }
    }

    /**
     * 根据key 获取过期时间
     *
     * @param key 键 不能为null
     * @return 时间(秒) 返回0代表为永久有效
     */
    public Long getExpire(String key) {
        return redisTemplate.getExpire(key, TimeUnit.SECONDS);
    }

    /**
     * 判断key是否存在
     *
     * @param key 键
     * @return true 存在 false不存在
     */
    public Boolean hasKey(String key) {
        try {
            return redisTemplate.hasKey(key);
        } catch (Throwable e) {
            return false;
        }
    }

    /**
     * 删除缓存
     *
     * @param key 可以传一个值 或多个
     */
    public Long delete(String... key) {
        if (key != null && key.length > 0) {
            Collection<String> collection = Arrays.asList(key);
            if (key.length == 1) {
                Boolean re = redisTemplate.delete(key[0]);
                if (re) {
                    return 1L;
                } else {
                    return 0L;
                }
            } else {
                return redisTemplate.delete(collection);
            }
        }
        return 0L;
    }

    // ============================= String =========================
====

    /**
```

```
     * 获取字符串
     *
     * @param key key
     * @return String value
     */
    public String getString(String key) {
        Object obj = redisTemplate.opsForValue().get(key);
        return obj.toString();
    }

    /**
     * 保存字符串
     *
     * @param key   键
     * @param value 值
     */
    public void setString(String key, Object value) {
        redisTemplate.opsForValue().set(key, value);
    }

    /**
     * 普通缓存放入并设置时间
     *
     * @param key        键
     * @param value      值
     * @param expireTime 时间(秒) time要大于0 如果time小于等于0 将设置无限期
     */
    public void setString(String key, Object value, long expireTime) {
        if (expireTime > 0) {
            redisTemplate.opsForValue().set(key, value, expireTime, TimeUnit.SECONDS);
        } else {
            setString(key, value);
        }
    }

    // ============================= Object =========================
====

    /**
     * 获取对象
     *
     * @param key key
     * @return obj
     */
    public Object getObject(String key) {
        return redisTemplate.opsForValue().get(key);
    }

    /**
     * 普通缓存放入并设置时间
     *
     * @param key   键
     * @param value 值
     * @param time  时间(秒) time要大于0 如果time小于等于0 将设置无限期
```

```
     */
    public void setObject(String key, Object value, long time) {
        //JsonUtils.toJson
        if (time > 0) {
            redisTemplate.opsForValue().set(key, value, time, TimeUnit.SECONDS);
        } else {
            setObject(key, value);
        }
    }

    /**
     * 普通缓存放入
     *
     * @param key   键
     * @param value 值
     */
    public void setObject(String key, Object value) {
        redisTemplate.opsForValue().set(key, value);
    }

    /**
     * 递增
     *
     * @param key   键
     * @param delta 要增加几(大于0)
     * @return long
     */
    public Long incr(String key, long delta) {
        if (delta < 0) {
            throw new RuntimeException("递增因子必须大于0");
        }
        return redisTemplate.opsForValue().increment(key, delta);
    }

    /**
     * 递减
     *
     * @param key   键
     * @param delta 要减少几(小于0)
     * @return long
     */
    public Long decr(String key, long delta) {
        if (delta < 0) {
            throw new RuntimeException("递减因子必须大于0");
        }
        return redisTemplate.opsForValue().increment(key, -delta);
    }

    // ================================= Map ======================
===========

    /**
     * HashGet
     *
```

```
 * @param key  键 不能为null
 * @param item 项 不能为null
 * @return 值
 */
public Object hget(String key, String item) {
    return redisTemplate.opsForHash().get(key, item);
}

/**
 * 获取hashKey对应的所有键值
 *
 * @param key 键
 * @return 对应的多个键值
 */
public Map<Object, Object> hmget(String key) {
    return redisTemplate.opsForHash().entries(key);
}

/**
 * HashSet
 *
 * @param key 键
 * @param map 对应多个键值
 */
public void hmset(String key, Map<String, Object> map) {
    redisTemplate.opsForHash().putAll(key, map);
}

/**
 * HashSet 并设置时间
 *
 * @param key  键
 * @param map  对应多个键值
 * @param time 时间(秒)
 */
public void hmset(String key, Map<String, Object> map, long time) {
    redisTemplate.opsForHash().putAll(key, map);
    if (time > 0) {
        expire(key, time);
    }
}

/**
 * 向一张hash表中放入数据,如果不存在将创建
 *
 * @param key   键
 * @param item  项
 * @param value 值
 */
public void hset(String key, String item, Object value) {
    redisTemplate.opsForHash().put(key, item, value);
}

/**
```

```
 * 向一张hash表中放入数据,如果不存在将创建
 *
 * @param key   键
 * @param item  项
 * @param value 值
 * @param time  时间(秒) 注意:如果已存在的hash表有时间,这里将会替换原有的时间
 */
public void hset(String key, String item, Object value, long time) {
    redisTemplate.opsForHash().put(key, item, value);
    if (time > 0) {
        expire(key, time);
    }
}

/**
 * 删除hash表中的值
 *
 * @param key  键 不能为null
 * @param item 项 可以使多个 不能为null
 */
public void hdel(String key, Object... item) {
    redisTemplate.opsForHash().delete(key, item);
}

/**
 * 判断hash表中是否有该项的值
 *
 * @param key  键 不能为null
 * @param item 项 不能为null
 * @return true 存在 false不存在
 */
public boolean hHasKey(String key, String item) {
    return redisTemplate.opsForHash().hasKey(key, item);
}

/**
 * hash递增 如果不存在,就会创建一个 并把新增后的值返回
 *
 * @param key  键
 * @param item 项
 * @param by   要增加几(大于0)
 * @return double
 */
public double hincr(String key, String item, double by) {
    return redisTemplate.opsForHash().increment(key, item, by);
}

/**
 * hash递减
 *
 * @param key  键
 * @param item 项
 * @param by   要减少记(小于0)
 * @return double
```

```
     */
    public double hdecr(String key, String item, double by) {
        return redisTemplate.opsForHash().increment(key, item, -by);
    }

    // ============================= set ==========================
==

    /**
     * 根据key获取Set中的所有值
     *
     * @param key 键
     * @return Set
     */
    public Set<Object> sGet(String key) {
        return redisTemplate.opsForSet().members(key);
    }

    /**
     * 根据value从一个set中查询,是否存在
     *
     * @param key   键
     * @param value 值
     * @return Boolean
     */
    public Boolean sHasKey(String key, Object value) {
        return redisTemplate.opsForSet().isMember(key, value);
    }

    /**
     * 将数据放入set缓存
     *
     * @param key    键
     * @param values 值 可以是多个
     * @return Long 成功个数
     */
    public Long sSet(String key, Object... values) {
        return redisTemplate.opsForSet().add(key, values);
    }

    /**
     * 将set数据放入缓存
     *
     * @param key    键
     * @param time   时间(秒)
     * @param values 值可以是多个
     * @return 成功个数
     */
    public Long sSetAndTime(String key, long time, Object... values) {
        try {
            Long count = redisTemplate.opsForSet().add(key, values);
            if (time > 0) {
                expire(key, time);
            }
```

```
            return count;
        } catch (Throwable e) {
            return 0L;
        }
    }

    /**
     * 获取set缓存的长度
     *
     * @param key 键
     * @return Long
     */
    public Long sGetSetSize(String key) {
        try {
            return redisTemplate.opsForSet().size(key);
        } catch (Throwable e) {
            return 0L;
        }
    }

    /**
     * 移除值为value的
     *
     * @param key    键
     * @param values 值可以是多个
     * @return 移除的个数
     */
    public Long setRemove(String key, Object... values) {
        try {
            return redisTemplate.opsForSet().remove(key, values);
        } catch (Throwable e) {
            return 0L;
        }
    }

    // ================================list=========================
========

    /**
     * 通过索引 获取list中的值
     *
     * @param key   键
     * @param index 索引 index>=0时， 0 表头，1 第二个元素，依次类推；index<0时，-1，表
，-2倒数第二个元素，依次类推
     * @return Obj
     */
    public Object lGetIndex(String key, long index) {
        try {
            return redisTemplate.opsForList().index(key, index);
        } catch (Throwable e) {
            return null;
        }
    }
```

```
/**
 * 将list放入缓存
 *
 * @param key   键
 * @param value 值
 */
public void lSet(String key, Object value) {
    redisTemplate.opsForList().rightPush(key, value);
}

/**
 * 将list放入缓存
 *
 * @param key   键
 * @param value 值
 * @param time  时间(秒)
 */
public void lSet(String key, Object value, long time) {
    redisTemplate.opsForList().rightPush(key, value);
    if (time > 0) {
        expire(key, time);
    }
}

/**
 * 将list放入缓存
 *
 * @param key   键
 * @param value 值
 */
public void lSet(String key, List<Object> value) {
    redisTemplate.opsForList().rightPushAll(key, value);
}

/**
 * 将list放入缓存
 *
 * @param key   键
 * @param value 值
 * @param time  时间(秒)
 */
public void lSet(String key, List<Object> value, long time) {
    redisTemplate.opsForList().rightPushAll(key, value);
    if (time > 0) {
        expire(key, time);
    }
}

/**
 * 根据索引修改list中的某条数据
 *
 * @param key   键
 * @param index 索引
 * @param value 值
```

```
     * @return boolean
     */
    public boolean lUpdateIndex(String key, long index, Object value) {
        try {
            redisTemplate.opsForList().set(key, index, value);
            return true;
        } catch (Throwable e) {
            return false;
        }
    }

    /**
     * 移除N个值为value
     *
     * @param key   键
     * @param count 移除多少个
     * @param value 值
     * @return 移除的个数
     */
    public Long lRemove(String key, long count, Object value) {
        try {
            return redisTemplate.opsForList().remove(key, count, value);
        } catch (Throwable e) {
            return 0L;
        }
    }
}
```

Swagger配置

1.SwaggerConfig.java

```
import com.base.web.config.security.utils.JwtTokenUtils;
import lombok.AllArgsConstructor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.env.Environment;
import org.springframework.core.env.Profiles;
import springfox.documentation.builders.ApiInfoBuilder;
import springfox.documentation.builders.ParameterBuilder;
import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.schema.ModelRef;
import springfox.documentation.service.ApiInfo;
import springfox.documentation.service.Contact;
import springfox.documentation.service.Parameter;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

import java.util.*;

/**
 * Swagger config
```

```
 *
 * @author ming
 * @version 1.0.0
 * @date 2020/04/15
 **/
@Configuration
@EnableSwagger2
@AllArgsConstructor
public class SwaggerConfig {

    private final SwaggerProperties properties;

    /**
     * frps-console (7000)
     * frp_0.34.3_linux_386 (ali Centos 7)
     * <p>
     * Client
     * cmd : frpc: frpc.exe -c frpc.ini
     * <p>
     * http://gua.esbug.com:7000/swagger-ui/index.html
     * swagger-ui V3:
     * http://localhost:8818/swagger-ui/index.html
     * swagger-ui V2:
     * http://localhost:8818/swagger-ui.html
     */

    @Bean
    public Docket createRestApi(Environment environment) {
        //设置要显示swagger的环境
        Profiles profiles = Profiles.of("uat", "dev");
        //判断不同环境中profiles的布尔值,并将enable传到enable(enable)方法中
        Boolean enable = environment.acceptsProfiles(profiles);

        ParameterBuilder parameterBuilder = new ParameterBuilder();
        List<Parameter> parameters = new ArrayList<>();
        parameterBuilder.name(JwtTokenUtils.TOKEN_HEADER).description("令牌").modelRef(ne
ModelRef("string")).parameterType("header").required(false).build();
        parameters.add(parameterBuilder.build());

        return new Docket(DocumentationType.SWAGGER_2)
                .pathMapping("/")
                .groupName("通用模块")
                .enable(enable)
                // 将api的元信息设置为包含在json ResourceListing响应中。
                .apiInfo(apiInfo())
                // 选择哪些接口作为swagger的doc发布
                .select()
                .apis(RequestHandlerSelectors.basePackage(properties.getBasePackage()))
                .paths(PathSelectors.any())
                .build()
                // 支持的通讯协议集合
                .protocols(newHashSet("https", "http"))
                .globalOperationParameters(parameters)
                //.globalRequestParameters()
```

```
            // 授权信息设置，必要的header token等认证信息
            /*.securitySchemes(Collections.singletonList(securitySchema()))
            // 授权信息全局应用
            .securityContexts(Collections.singletonList(securityContext()))*/;
    }

    /**
     * API 文档相关信息
     */
    private ApiInfo apiInfo() {
        return new ApiInfoBuilder()
                .title(properties.getTitle())
                .description(properties.getDescription())
                .termsOfServiceUrl(properties.getUrl())
                .version(properties.getVersion())
                .contact(new Contact(properties.getContact().getName(), properties.getContact().ge
Url(), properties.getContact().getEmail()))
                .build();
    }

    @SafeVarargs
    private final <T> Set<T> newHashSet(T... ts) {
        if (ts.length > 0) {
            return new LinkedHashSet<>(Arrays.asList(ts));
        }
        return null;
    }

}
```

2.SwaggerContact.java

```
import lombok.Data;

/**
 * @author ming
 * @version 1.0.0
 * @date 2021/1/5 10:44
 **/
@Data
class SwaggerContact {
    private String name;
    private String url;
    private String email;
}
```

3.SwaggerProperties.java

```
import lombok.Data;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

/**
 * @author ming
```

```
 * @version 1.0.0
 * @date 2021/1/5 10:14
 **/
@ConfigurationProperties(prefix = "swagger-doc")
@Component
@Data
public class SwaggerProperties {
    private String accessTokenKey;
    private String basePackage;
    private String title;
    private String description;
    private String url;
    private String version;
    private SwaggerContact contact;
}
```

4.自定义yml配置

```
###### swagger-ui ######
### doc
swagger-doc:
  access-token-key: ""
  base-package: "com.base.web.controller"
  title: "基础服务API"
  description: "基础服务API"
  url: "http://localhost:${server.port}/swagger-ui.html"
  version: "0.0.1"
  contact:
    name: "ming"
    url: "http://www.itwetouch.com/"
    email: "an23gn@163.com"
```

权限注解使用

不需要权限判断的开放的接口可以加入到核心配置的白名单中

```
@PreAuthorize("hasPermission('user','list')")
@GetMapping("list")
@ApiOperation(value = "用户列表", notes = "后台用户列表")
@ApiImplicitParams({
        @ApiImplicitParam(name = "current", value = "当前页码", dataType = "int", paramType = "query", defaultValue = "1"),
        @ApiImplicitParam(name = "size", value = "每页显示数", dataType = "int", paramType = query", defaultValue = "10", example = "10", allowableValues = "10,20,40,60"),
        @ApiImplicitParam(name = "userName", value = "用户名", dataType = "string", paramTy e = "query"),
        @ApiImplicitParam(name = "telephone", value = "手机号", dataType = "string", paramTy e = "query"),
        @ApiImplicitParam(name = "email", value = "邮箱", dataType = "string", paramType = " uery"),
})
public ApiResponse getUserList(@RequestParam(name = "current", required = false, defaultV lue = WebConstant.PAGE_NO) Integer current,
                               @RequestParam(name = "size", required = false, defaultValue = WebCons
```

```
ant.PAGE_SIZE) Integer size,
                        @RequestParam(name = "userName", required = false, defaultValue = "")
tring userName,
                        @RequestParam(name = "telephone", required = false, defaultValue = "")
tring telephone,
                        @RequestParam(name = "email", required = false, defaultValue = "") Strin
 email) {
    return userService.userList(current, size, userName, telephone, email);
}
```

登录接口

```
@PostMapping("login")
@ApiOperation(value = "登录", notes = "用户登录(支持用户名/手机号码/邮箱)")
@ApiImplicitParams({
        @ApiImplicitParam(name = "username", value = "用户名", dataType = "string", paramTy
e = "query", required = true, defaultValue = "admin"),
        @ApiImplicitParam(name = "password", value = "密码", dataType = "string", paramType
= "query", required = true, defaultValue = "123456")
})
public void login(@RequestParam(name = "username") String username,
                @RequestParam(name = "password") String password) throws Exception {
    // nothing to do
}
```

登出接口

```
@PostMapping("logout")
@ApiOperation(value = "登出", notes = "用户登出")
@ResponseStatus(HttpStatus.OK)
public void logout() throws Exception {
    // nothing to do
}
```