SpringBoot+Vue博客系统

3.1 登录功能实现

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

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

一般Web应用的需要进行**认证**和**授权**。

认证：验证当前访问系统的是不是本系统的用户，并且要确认具体是哪个用户

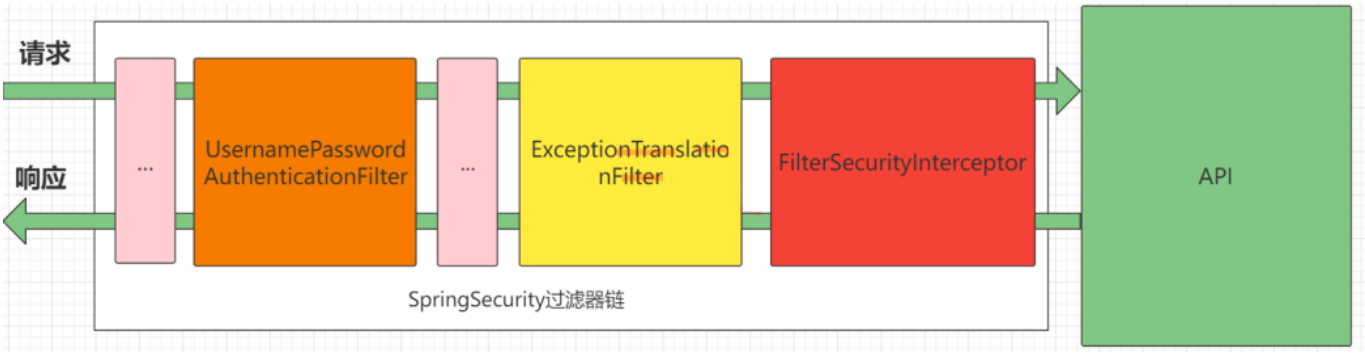
授权：经过认证后判断当前用户是否有权限进行某个操作

而认证和授权也是SpringSecurity作为安全框架的核心功能。

我们前台和后台的认证授权统一都使用SpringSecurity安全框架来实现。

3.1.0 前置知识-Spring Security

SpringSecurity的原理其实就是一个**过滤器链**，内部包含了提供各种功能的过滤器。



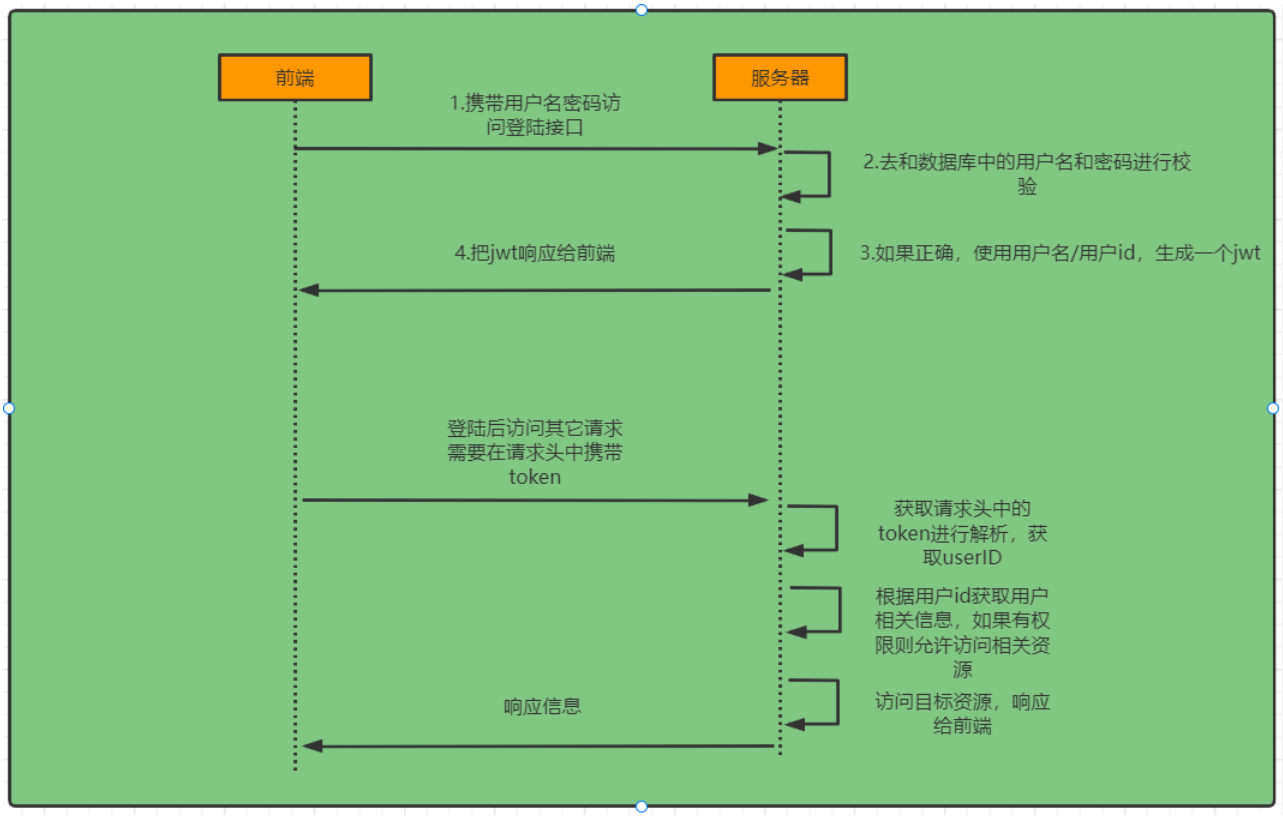
图中只展示了核心过滤器，其它的非核心过滤器并没有在图中展示。

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

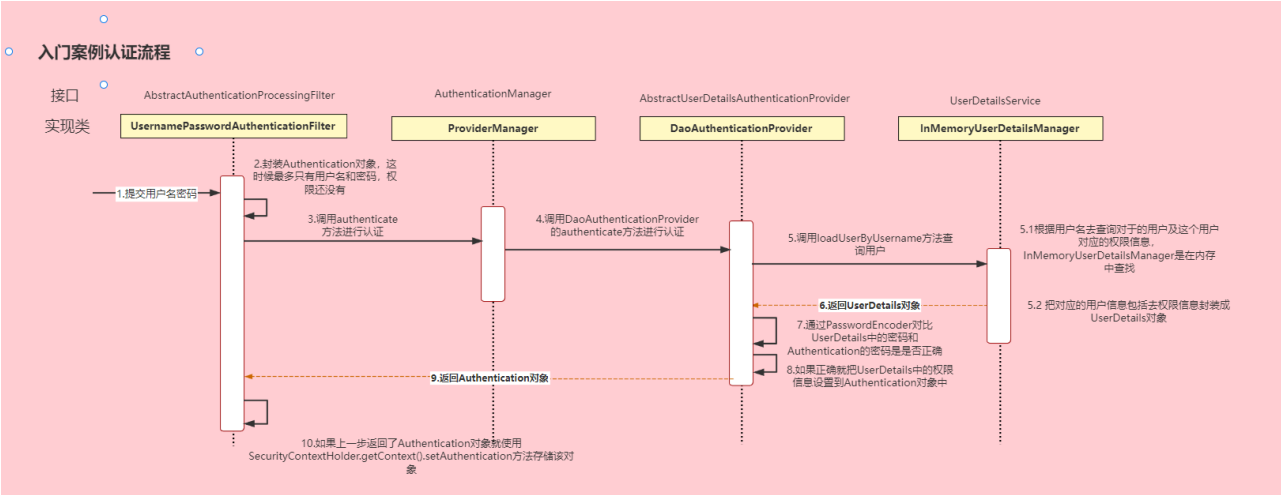
ExceptionTranslationFilter:处理过滤器链中抛出的任何AccessDeniedException和AuthenticationException 。

FilterSecurityInterceptor:负责权限校验的过滤器。

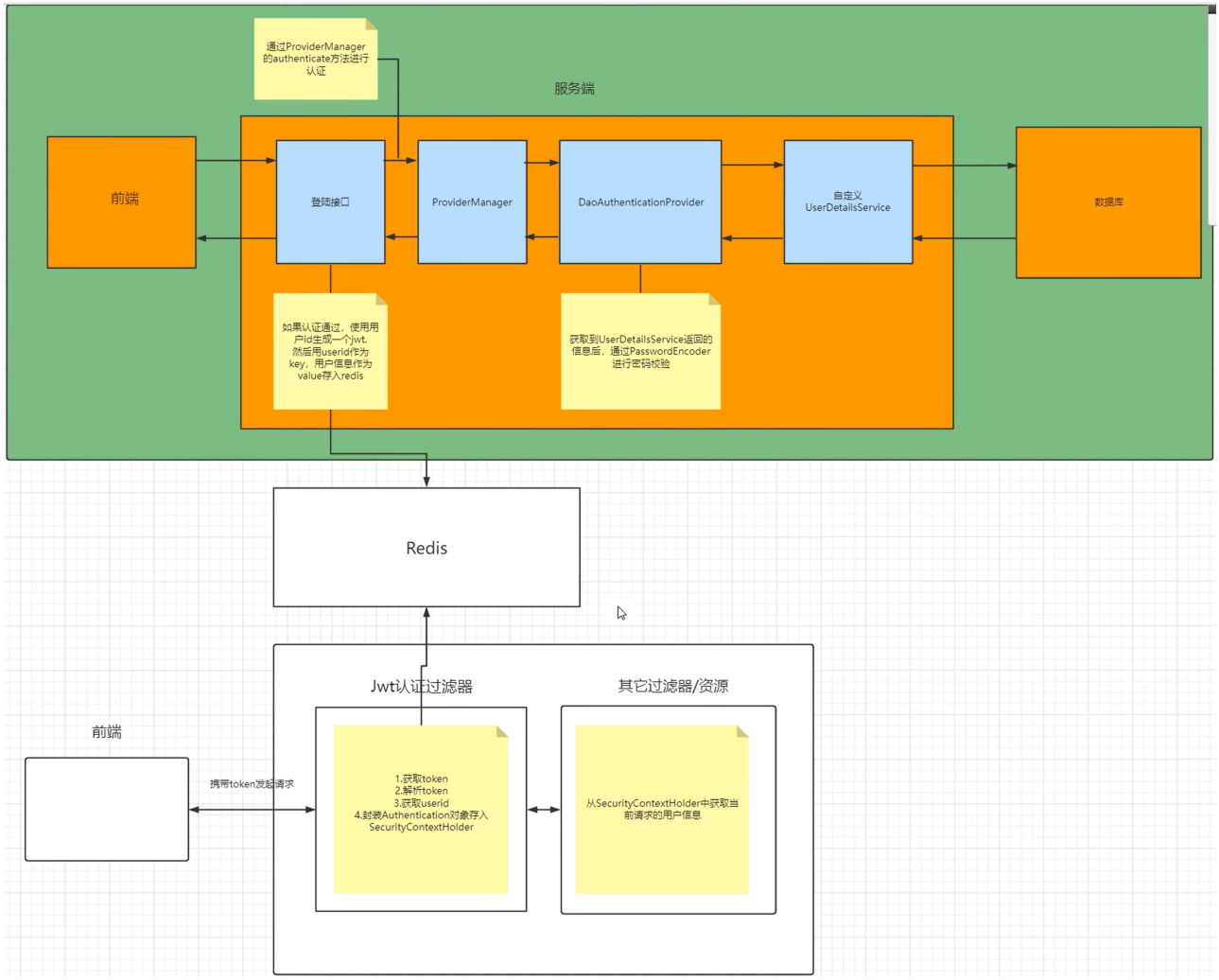
- 登陆校验流程



- 认证流程详解



- 自定义登录认证流程



概念速查:

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

AuthenticationManager接口：定义了认证Authentication的方法

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

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

3.1.1 需求

需要实现登录功能

有些功能必须登录后才能使用，未登录状态是不能使用的。

3.1.2 接口设计

请求方式	请求路径
POST	/login

请求体:

```
{
    "userName":"test",
    "password":"1234"
}
```

响应格式：

```
{
    "code": 200,
    "data": {
        "token":
"eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI0ODBmOThmYmJkNmI0NjM0OWUyZjY2NTM0NGNjZWY2NSIsInN1
YiI6IjEiLCJpc3MiOiJzZyIsImlhdCI6MTY0Mzg3NDMxNiwiZXhwIjoxNjQzOTYwNzE2fQ.ldLBUvNIxQC
GemkCoMgT_0YsjsWndTg5tqfJb77pabk",
        "userInfo": {
            "avatar":
"http://i0.hdslb.com/bfs/article/3bf9c263bc0f2ac5c3a7feb9e218d07475573ec8.gif",
            "email": "test@ptu.edu.cn",
            "id": 1,
            "nickName": "test",
            "sex": "1"
        }
    },
    "msg": "操作成功"
}
```

3.1.3 思路分析

登录

①自定义登录接口

调用ProviderManager的方法进行认证 如果认证通过生成jwt

把用户信息存入redis中

②自定义UserDetailsService

在这个实现类中去查询数据库

注意配置passwordEncoder为BCryptPasswordEncoder

校验：

①定义Jwt认证过滤器

获取token

解析token获取其中的userid

从redis中获取用户信息

存入SecurityContextHolder

3.1.4 准备工作

①添加依赖(framework)

注意取消Security依赖的注释

```
        <!--redis依赖-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
        </dependency>
        <!--fastjson依赖-->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.33</version>
        </dependency>
        <!--jwt依赖-->
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt</artifactId>
            <version>0.9.0</version>
        </dependency>
```

②工具类和相关配置类

见： 登录功能所需资源

maven菜单：lifecycle-install

3.1.5 登录接口代码实现

BlogLoginController

blog模块下

```
@RestController
public class BlogLoginController {
    @Autowired
    private IBlogLoginService blogLoginService;

    @PostMapping("/login")
    public ResponseResult login(@RequestBody User user){
        return blogLoginService.login(user);
    }
```

```
}
```

IBlogLoginService

framework模块下

```
public interface IBlogLoginService {
    ResponseResult login(User user);
}
```

SecurityConfig

blog模块下

- 旧版配置

```
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Bean
    public PasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder();
    }
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                //关闭csrf
                .csrf().disable()
                //不通过Session获取SecurityContext

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                .and()
                .authorizeRequests()
                // 对于登录接口 允许匿名访问
                .antMatchers("/login").anonymous()
                // 除上面外的所有请求全部不需要认证即可访问
                .anyRequest().permitAll();


        http.logout().disable();
        //允许跨域
        http.cors();
    }
    @Override
    @Bean
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
```

```
    }
}
```

- 新版配置: Spring Security without the WebSecurityConfigurerAdapter, stackoverflow: Global AuthenticationManager without the WebSecurityConfigurerAdapter

```
@Configuration
public class SecurityConfig {

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

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
                //关闭csrf
                .csrf().disable()
                //不通过Session获取SecurityContext

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                .and()
                .authorizeRequests()
                // 对于登录接口 允许匿名访问
                .antMatchers("/login").anonymous()
                // 除上面外的所有请求全部不需要认证即可访问
                .anyRequest().permitAll();


        http.logout().disable();
        //允许跨域
        http.cors();

        return http.build();
    }

    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration
authenticationConfiguration) throws Exception {
        return authenticationConfiguration.getAuthenticationManager();
    }
}
```

BlogLoginServiceImpl

framework模块下

redis key为`bloglogin:id`

```
@Service
public class BlogLoginServiceImpl implements IBlogLoginService {

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private RedisCache redisCache;

    @Override
    public ResponseResult login(User user) {
        UsernamePasswordAuthenticationToken authenticationToken = new
UsernamePasswordAuthenticationToken(user.getUserName(),user.getPassword());
        Authentication authenticate =
authenticationManager.authenticate(authenticationToken);
        //判断是否认证通过
        if(Objects.isNull(authenticate)){
            throw new RuntimeException("用户名或密码错误");
        }
        //获取userid 生成token
        LoginUser loginUser = (LoginUser) authenticate.getPrincipal();
        String userId = loginUser.getUser().getId().toString();
        String jwt = JwtUtil.createJWT(userId);
        //把用户信息存入redis
        redisCache.setCacheObject("bloglogin:"+userId,loginUser);

        //把token和userinfo封装 返回
        //把User转换成UserInfoVo
        UserInfoVo userInfoVo = new UserInfoVo();
        BeanUtils.copyProperties(loginUser.getUser(), userInfoVo);
        BlogUserLoginVo vo = new BlogUserLoginVo(jwt,userInfoVo);
        return ResponseResult.okResult(vo);
    }
}
```

UserDetailsServiceImpl

framework模块下

```
@Service
public class UserDetailsServiceImpl implements UserDetailsService {

    @Autowired
    private UserMapper userMapper;

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        //根据用户名查询用户信息
```

```
        LambdaQueryWrapper<User> queryWrapper = new LambdaQueryWrapper<>();
        queryWrapper.eq(User::getUserName,username);
        User user = userMapper.selectOne(queryWrapper);
        //判断是否查到用户  如果没查到抛出异常
        if(Objects.isNull(user)){
            throw new RuntimeException("用户不存在");
        }
        //返回用户信息
        // TODO 查询权限信息封装
        return new LoginUser(user);
    }
}
```

LoginUser

framework模块下

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

    private User user;


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

    @Override
    public String getPassword() {
        return user.getPassword();
    }

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

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

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

    @Override
```

```
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }
}
```

BlogUserLoginVo

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class BlogUserLoginVo {

    private String token;
    private UserInfoVo userInfo;
}
```

UserInfoVo

```
@Data
@Accessors(chain = true)
public class UserInfoVo {
    /**
     * 主键
     */
    private Long id;

    /**
     * 昵称
     */
    private String nickName;

    /**
     * 头像
     */
    private String avatar;

    private String sex;

    private String email;
```

```
}
```

安装、启动redis，如果未在项目里配置redis主机端口，默认使用本机6379

使用Postman测试/login接口

3.1.6 登录校验过滤器代码实现

思路

①定义Jwt认证过滤器

获取token

解析token获取其中的userid

从redis中获取用户信息

存入SecurityContextHolder

JwtAuthenticationTokenFilter

```
@Component
public class JwtAuthenticationTokenFilter extends OncePerRequestFilter {

    @Autowired
    private RedisCache redisCache;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain filterChain) throws ServletException,
IOException {
        //获取请求头中的token
        String token = request.getHeader("token");
        if(!StringUtils.hasText(token)){
            //说明该接口不需要登录  直接放行
            filterChain.doFilter(request, response);
            return;
        }
        //解析获取userid
        Claims claims = null;
        try {
            claims = JwtUtil.parseJWT(token);
        } catch (Exception e) {
            e.printStackTrace();
            //token超时  token非法
            //响应告诉前端需要重新登录
            ResponseResult result =
ResponseResult.errorResult(AppHttpCodeEnum.NEED_LOGIN);
            WebUtils.renderString(response, JSON.toJSONString(result));
```

```
            return;
        }
        String userId = claims.getSubject();
        //从redis中获取用户信息
        LoginUser loginUser = redisCache.getCacheObject("bloglogin:" + userId);
        //如果获取不到
        if(Objects.isNull(loginUser)){
            //说明登录过期  提示重新登录
            ResponseResult result =
ResponseResult.errorResult(AppHttpCodeEnum.NEED_LOGIN);
            WebUtils.renderString(response, JSON.toJSONString(result));
            return;
        }
        //存入SecurityContextHolder
        UsernamePasswordAuthenticationToken authenticationToken = new
UsernamePasswordAuthenticationToken(loginUser,null,null);
        SecurityContextHolder.getContext().setAuthentication(authenticationToken);

        filterChain.doFilter(request, response);
    }


}
```

SecurityConfig

```
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    @Bean
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }

    @Autowired
    private JwtAuthenticationTokenFilter jwtAuthenticationTokenFilter;

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                //关闭csrf
                .csrf().disable()
                //不通过Session获取SecurityContext

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                .and()
                .authorizeRequests()
                // 对于登录接口 允许匿名访问
                .antMatchers("/login").anonymous()
                //jwt过滤器测试用，如果测试没有问题吧这里删除了
```

```
                .antMatchers("/link/getAllLink").authenticated()
                // 除上面外的所有请求全部不需要认证即可访问
                .anyRequest().permitAll();


        http.logout().disable();
        //把jwtAuthenticationTokenFilter添加到SpringSecurity的过滤器链中
        http.addFilterBefore(jwtAuthenticationTokenFilter,
UsernamePasswordAuthenticationFilter.class);
        //允许跨域
        http.cors();
    }

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

3.2 认证授权失败处理

目前我们的项目在认证出错或者权限不足的时候响应回来的Json是Security的异常处理结果。但是这个响应的格式肯定是不符合我们项目的接口规范的。所以需要自定义异常处理。

AuthenticationEntryPoint 认证失败处理器

AccessDeniedHandler 授权失败处理器

```
@Component
public class AuthenticationEntryPointImpl implements AuthenticationEntryPoint {

    @Override
    public void commence(HttpServletRequest request, HttpServletResponse response,
AuthenticationException authException) throws IOException, ServletException {
        authException.printStackTrace();
        //InsufficientAuthenticationException
        //BadCredentialsException
        ResponseResult result = null;
        if(authException instanceof BadCredentialsException){
            result =
ResponseResult.errorResult(AppHttpCodeEnum.LOGIN_ERROR.getCode(),authException.get
Message());
        }else if(authException instanceof InsufficientAuthenticationException){
            result = ResponseResult.errorResult(AppHttpCodeEnum.NEED_LOGIN);
        }else{
            result =
ResponseResult.errorResult(AppHttpCodeEnum.SYSTEM_ERROR.getCode(),"认证或授权失
败");
        }
        //响应给前端
```

```
        WebUtils.renderString(response, JSON.toJSONString(result));
    }
}
```

```
@Component
public class AccessDeniedHandlerImpl implements AccessDeniedHandler {
    @Override
    public void handle(HttpServletRequest request, HttpServletResponse response,
AccessDeniedException accessDeniedException) throws IOException, ServletException
{
        accessDeniedException.printStackTrace();
        ResponseResult result =
ResponseResult.errorResult(AppHttpCodeEnum.NO_OPERATOR_AUTH);
        //响应给前端
        WebUtils.renderString(response, JSON.toJSONString(result));
    }
}
```

配置Security异常处理器

```
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    @Bean
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }

    @Autowired
    private JwtAuthenticationTokenFilter jwtAuthenticationTokenFilter;
    @Autowired
    AuthenticationEntryPoint authenticationEntryPoint;
    @Autowired
    AccessDeniedHandler accessDeniedHandler;


    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                //关闭csrf
                .csrf().disable()
                //不通过Session获取SecurityContext

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                .and()
                .authorizeRequests()
```

```
                // 对于登录接口 允许匿名访问
                .antMatchers("/login").anonymous()
                //jwt过滤器测试用，如果测试没有问题吧这里删除了
                .antMatchers("/link/getAllLink").authenticated()
                // 除上面外的所有请求全部不需要认证即可访问
                .anyRequest().permitAll();

        //配置异常处理器
        http.exceptionHandling()
                .authenticationEntryPoint(authenticationEntryPoint)
                .accessDeniedHandler(accessDeniedHandler);

        http.logout().disable();
        //把jwtAuthenticationTokenFilter添加到SpringSecurity的过滤器链中
        http.addFilterBefore(jwtAuthenticationTokenFilter,
UsernamePasswordAuthenticationFilter.class);
        //允许跨域
        http.cors();
    }

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

3.3 统一异常处理

实际我们在开发过程中可能需要做很多的判断校验，如果出现了非法情况我们是期望响应对应的提示的。但是如果我们每次都自己手动去处理就会非常麻烦。我们可以选择直接抛出异常的方式，然后对异常进行统一处理。把异常中的信息封装成ResponseResult响应给前端。

SystemException

```
public class SystemException extends RuntimeException{

    private int code;

    private String msg;

    public int getCode() {
        return code;
    }

    public String getMsg() {
        return msg;
    }
```

```
    public SystemException(AppHttpCodeEnum httpCodeEnum) {
        super(httpCodeEnum.getMsg());
        this.code = httpCodeEnum.getCode();
        this.msg = httpCodeEnum.getMsg();
    }

}
```

GlobalExceptionHandler

```
@RestControllerAdvice
@Slf4j
public class GlobalExceptionHandler {

    @ExceptionHandler(SystemException.class)
    public ResponseResult systemExceptionHandler(SystemException e){
        //打印异常信息
        log.error("出现了异常！ {}",e);
        //从异常对象中获取提示信息封装返回
        return ResponseResult.errorResult(e.getCode(),e.getMsg());
    }


    @ExceptionHandler(Exception.class)
    public ResponseResult exceptionHandler(Exception e){
        //打印异常信息
        log.error("出现了异常！ {}",e);
        //从异常对象中获取提示信息封装返回
        return
ResponseResult.errorResult(AppHttpCodeEnum.SYSTEM_ERROR.getCode(),e.getMessage());
    }
}
```

3.4 退出登录接口

3.4.1 接口设计

请求方式	请求地址	请求头
POST	/logout	需要token请求头

响应格式:

```
{
    "code": 200,
    "msg": "操作成功"
}
```

3.4.2 代码实现

要实现的操作：

删除redis中的用户信息

BlogLoginController

```
    @PostMapping("/logout")
    public ResponseResult logout(){
        return blogLoginService.logout();
    }
```

BlogLoginService

```
ResponseResult logout();
```

BlogLoginServiceImpl

```
    @Override
    public ResponseResult logout() {
        //获取token 解析获取userid
        Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
        LoginUser loginUser = (LoginUser) authentication.getPrincipal();
        //获取userid
        Long userId = loginUser.getUser().getId();
        //删除redis中的用户信息
        redisCache.deleteObject("bloglogin:"+userId);
        return ResponseResult.okResult();
    }
```

SecurityConfig

要关闭默认的退出登录功能。并且要配置我们的退出登录接口需要认证才能访问

```
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                //关闭csrf
                .csrf().disable()
                //不通过Session获取SecurityContext

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
```

```
                .and()
                .authorizeRequests()
                // 对于登录接口 允许匿名访问
                .antMatchers("/login").anonymous()
                //注销接口需要认证才能访问
                .antMatchers("/logout").authenticated()
                //jwt过滤器测试用，如果测试没有问题吧这里删除了
                .antMatchers("/link/getAllLink").authenticated()
                // 除上面外的所有请求全部不需要认证即可访问
                .anyRequest().permitAll();

        //配置异常处理器
        http.exceptionHandling()
                .authenticationEntryPoint(authenticationEntryPoint)
                .accessDeniedHandler(accessDeniedHandler);
        //关闭默认的注销功能
        http.logout().disable();
        //把jwtAuthenticationTokenFilter添加到SpringSecurity的过滤器链中
        http.addFilterBefore(jwtAuthenticationTokenFilter,
UsernamePasswordAuthenticationFilter.class);
        //允许跨域
        http.cors();
    }
```