Springboot 异常处理

1. 定义一个自己的异常类

```
public class MyException extends RuntimeException{
    private int code=200;
    private String msg;
    public MyException(String msg) {
        super(msg);
        this.msg = msg;
    }
}
```

2. 定义一个异常处理类

Springboot 处理异常主要依靠 @ExceptionHandler 注解

```
@RestControllerAdvice
public class ExceptionAdvice {
    @ResponseBody
    @ResponseStatus(HttpStatus.OK)
    @ExceptionHandler(Exception.class)
    public JsonResult exceptionHandler(Exception e){
        e.printStackTrace();
        return JsonResult.fail(e.getMessage());
    }
}
```

关于使用异常中断程序， 不同的公司有不同的看法。

按公司流程和团队习惯操作就好!

Springboot 权限控制

Spring Securty 整合项目， 一般都是通过 SecurityContextHolder 工具类实现的。

添加依赖

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

1. 我们创建自己的 过滤器

```
@Component
public class SpringSecurityFilter  extends OncePerRequestFilter {
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException {
        HttpSession session = request.getSession(true);
        Object user =  session.getAttribute("User");
        //如果能查到用户信息 则认为用户已登录 直接给 Security 赋值权限
        if (Objects.nonNull( user )) {
            //用户存在
            UsernamePasswordAuthenticationToken userToken =
                    new UsernamePasswordAuthenticationToken( "username", null, null );
            SecurityContextHolder.getContext().setAuthentication( userToken );
        } else {
            //如果 redis 查不到 用户信息 则认为 登录过期 清空Security上下文
            SecurityContextHolder.clearContext();
        }
        //过滤器放心
        filterChain.doFilter( request, response );
    }
}
```

2. 创建 Securty 配置文件

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {
    @Autowired
    SpringSecurityFilter springSecurityFilter;
    @Bean
    public SecurityFilterChain t(HttpSecurity http) throws Exception {
        HttpSecurity and = http
                //配置跨站攻击
                .csrf()
                //禁用跨站攻击
                .disable()
                //配置session
                .sessionManagement()
```

```
                //关闭session
                .sessionCreationPolicy(SessionCreationPolicy.NEVER)
                //结束session配置
                .and()
                //配置用户权限
                .authorizeRequests()
                //通过URL 匹配权限路径， 配置权限
                //URL 匹配了  /user/login 这个请求， 则放行
                .mvcMatchers("/user/login", "/user/logout").permitAll()
                //其他请求  全部必须登录才能访问
                .anyRequest().authenticated()
                //结束权限配置
                .and();
        //权限异常
        and.exceptionHandling().authenticationEntryPoint(new
RestAuthenticationEntryPoint())
        //把我们自定义的过滤器添加到  Security 过滤器链路的 前端， 保证我们的过滤器优先执行
        and.addFilterBefore( springSecurityFilter,
UsernamePasswordAuthenticationFilter.class);
        //返回鉴权对象
        return and.build();
    }
}
```

3. 异常处理

   a. AuthenticationEntryPoint

   b. AccessDeniedHandler

```
public class RestAuthenticationEntryPoint implements AuthenticationEntryPoint {
    @Override
    public void commence(HttpServletRequest request, HttpServletResponse response,
AuthenticationException authException) throws IOException, ServletException {
        response.setHeader("Access-Control-Allow-Origin", "*");
        response.setHeader("Cache-Control","no-cache");
        response.setCharacterEncoding("UTF-8");
        response.setContentType("application/json");

response.getWriter().println(JSONUtil.parse(CommonResult.unauthorized(authException.get
Message())));
        response.getWriter().flush();
    }
```

```
}
```

```
public class RestfulAccessDeniedHandler implements AccessDeniedHandler{
    @Override
    public void handle(HttpServletRequest request,
                       HttpServletResponse response,
                       AccessDeniedException e) throws IOException, ServletException {
        response.setHeader("Access-Control-Allow-Origin", "*");
        response.setHeader("Cache-Control","no-cache");
        response.setCharacterEncoding("UTF-8");
        response.setContentType("application/json");
response.getWriter().println(JSONUtil.parse(CommonResult.forbidden(e.getMessage())));
        response.getWriter().flush();
    }
}

```

权限验证注解

```
@PreAuthorize("hasAuthority('权限')")
```

启用控制器验证

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