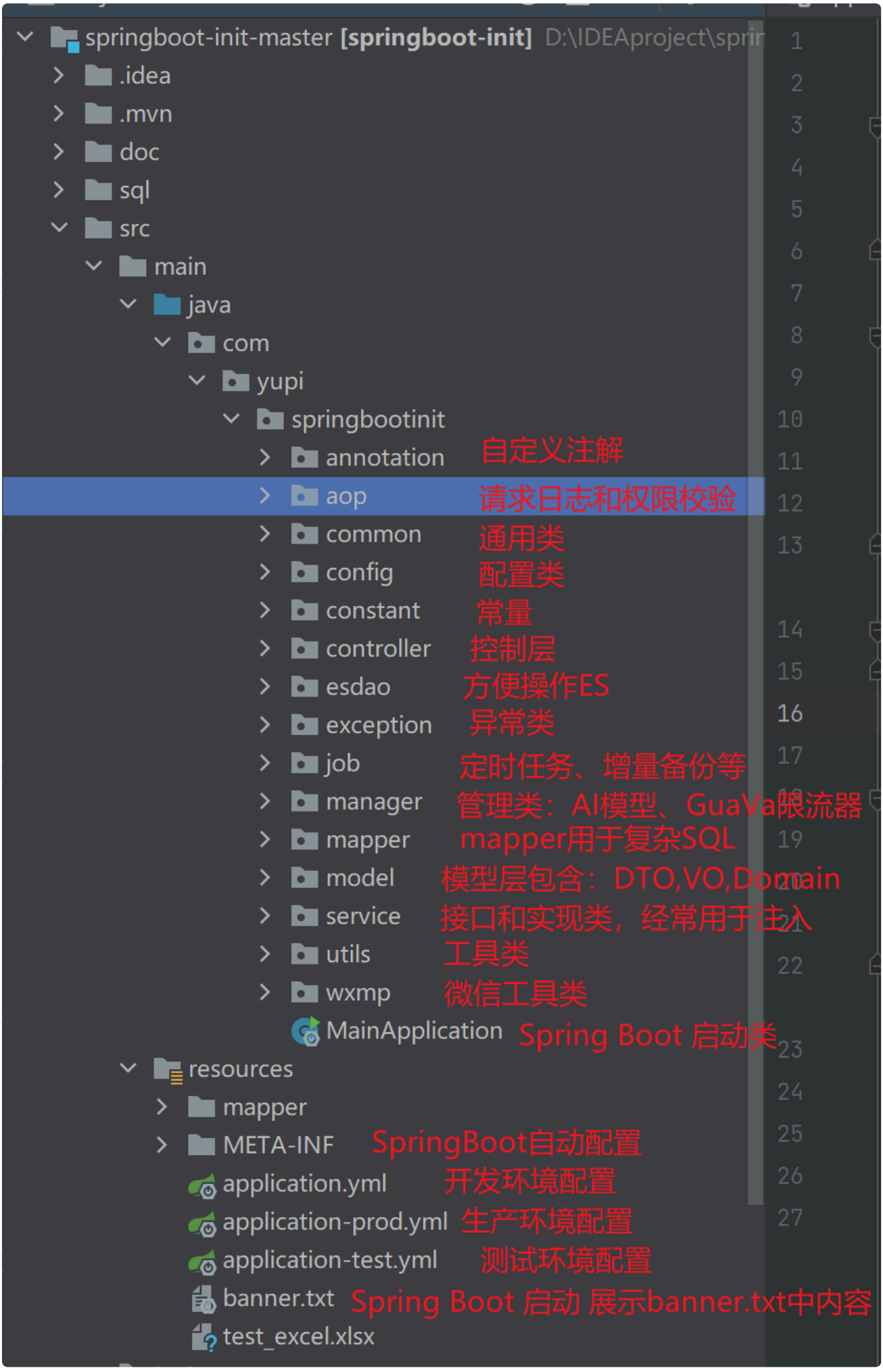
2-Java 后端万用项目模板使用教程

> 作者：小白条 <https://gitee.com/falle22222n-leaves> - 编程导航 <https://www.code-nav.cn/post/1816420035119853569> 编号 5563

今天来给大家介绍一下星球的后端 Spring Boot 万用模板的使用。

1、每个模块的简要概写


springboot-init-master [springboot-init] D:\IDEAproject\sprin
.idea
.mvn
doc
sql
src
main
java
com
yupi
springbootinit
annotation 自定义注解
aop 请求日志和权限校验
common 通用类
config 配置类
constant 常量
controller 控制层
esdao 方便操作ES
exception 异常类
job 定时任务、增量备份等
manager 管理类：AI模型、GuaVa限流器
mapper mapper用于复杂SQL
model 模型层包含：DTO,VO,Domain
service 接口和实现类，经常用于注入
utils 工具类
wxmp 微信工具类
MainApplication Spring Boot 启动类
resources
mapper
META-INF SpringBoot自动配置
application.yml 开发环境配置
application-prod.yml 生产环境配置
application-test.yml 测试环境配置
banner.txt Spring Boot 启动 展示banner.txt中内容
test_excel.xlsx

2023字

2、一些重点模块的讲解

2.1 全局项目配置

application.yml

```yaml
# 公共配置文件
# @author <a href="https://github.com/liyupi">程序员鱼皮</a>
# @from <a href="https://yupi.icu">编程导航知识星球</a>
spring:
  application:
    name: springboot-init
  # 默认 dev 环境
  profiles:
    active: dev
  # 支持 swagger3
  mvc:
    pathmatch:
      matching-strategy: ant_path_matcher
  # session 配置
  session:
    # todo 取消注释开启分布式 session（须先配置 Redis）
    # store-type: redis
    # 30 天过期
    timeout: 2592000
  # 数据库配置
  # todo 需替换配置
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/my_db
    username: root
    password: 123456
  # Redis 配置
  # todo 需替换配置，然后取消注释
  #  redis:
  #    database: 1
  #    host: localhost
  #    port: 6379
  #    timeout: 5000
  #    password: 123456
  # Elasticsearch 配置
  # todo 需替换配置，然后取消注释
```

数据库配置文件是每一个项目都要修改的，一般修改内容为：数据库库名，例如: my_db，用户名: xxxx 密码: xxxx。

Redis 在一伙伴匹配项目中用到，如有需要需要替换成自己的密码。并且在 Spring Boot 启动类中作出如下修改。

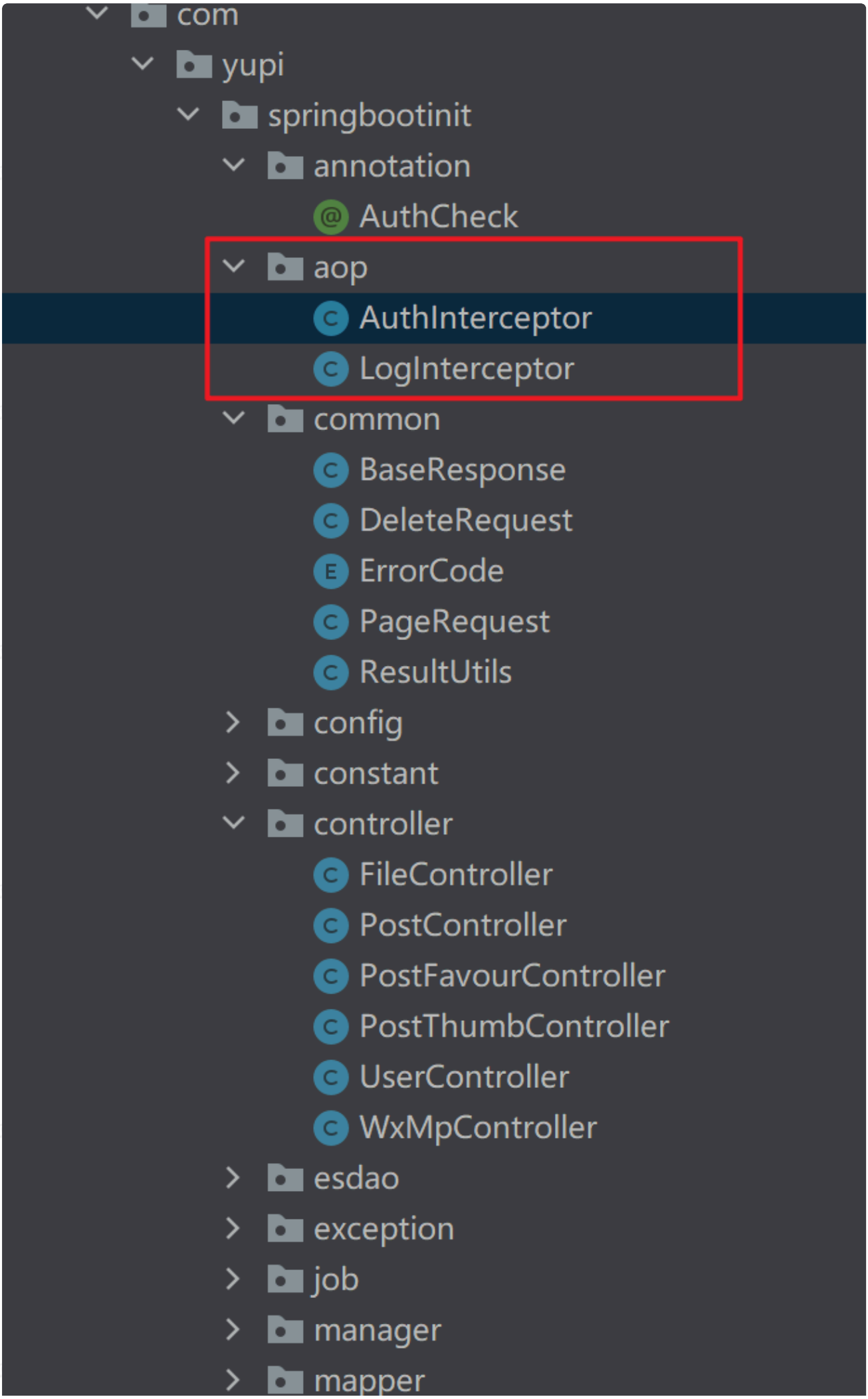
```java
*/
// todo 如需开启 Redis，须移除 exclude 中的内容
1 usage
@SpringBootApplication(exclude = {RedisAutoConfiguration.class})
@MapperScan("com.yupi.springbootinit.mapper")
@EnableScheduling
@EnableAspectJAutoProxy(proxyTargetClass = true, exposeProxy = true)
public class MainApplication {

    public static void main(String[] args) { SpringApplication.run(MainApplication.class, args); }

}
```

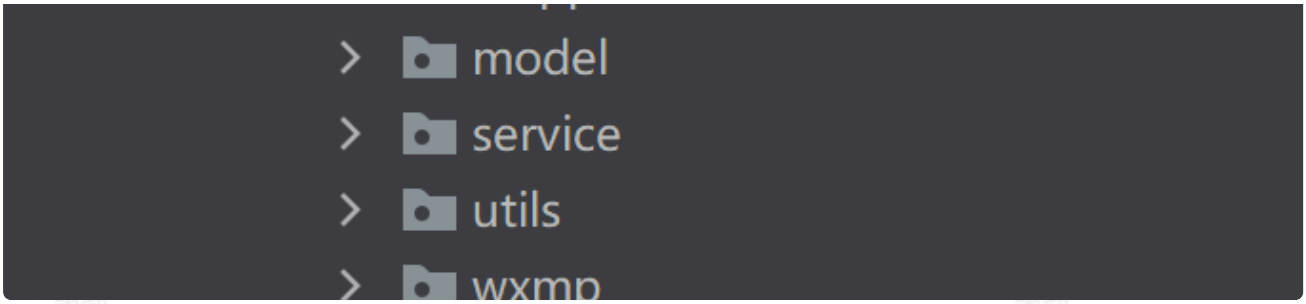
删除exclude的内容即可

2.2 全局请求、鉴权拦截器

2023字

AuthInterceptor 和 LogInterceptor


com
yupi
springbootinit
annotation
AuthCheck
aop
AuthInterceptor
LogInterceptor
common
BaseResponse
DeleteRequest
ErrorCode
PageRequest
ResultUtils
config
constant
controller
FileController
PostController
PostFavourController
PostThumbController
UserController
WxMpController
esdao
exception
job
manager
mapper



先介绍下AuthInterceptor,权限校验机制，判断用户的role(角色)是否为管理员、用户、ban(封号)三种情况。

```java
/**
 * 执行拦截
 *
 * @param joinPoint
 * @param authCheck
 * @return
 */
@Around("@annotation(authCheck)")
public Object doInterceptor(ProceedingJoinPoint joinPoint, AuthCheck authCheck) throws Throwable {
    String mustRole = authCheck.mustRole();
    RequestAttributes requestAttributes = RequestContextHolder.currentRequestAttributes();
    HttpServletRequest request = ((ServletRequestAttributes) requestAttributes).getRequest();
    // 当前登录用户
    User loginUser = userService.getLoginUser(request);
    // 必须有该权限才通过
    if (StringUtils.isNotBlank(mustRole)) {
        UserRoleEnum mustUserRoleEnum = UserRoleEnum.getEnumByValue(mustRole);
        if (mustUserRoleEnum == null) {
            throw new BusinessException(ErrorCode.NO_AUTH_ERROR);
        }
        String userRole = loginUser.getUserRole();
        // 如果被封号，直接拒绝
        if (UserRoleEnum.BAN.equals(mustUserRoleEnum)) {
            throw new BusinessException(ErrorCode.NO_AUTH_ERROR);
        }
        // 必须有管理员权限
        if (UserRoleEnum.ADMIN.equals(mustUserRoleEnum)) {
            if (!mustRole.equals(userRole)) {
                throw new BusinessException(ErrorCode.NO_AUTH_ERROR);
            }
        }
    }
    // 通过权限校验，放行
    return joinPoint.proceed();
}
}
```

登录的时候，已经在会话中放入了登录态，因此可以获取。

1.有权限，表示至少是三种中的一个

2.枚举值为null,说明没有权限访问。

3.ban封号直接拒绝

4.管理员权限才能访问，否则抛出异常，例如一些批量获取帖子的方法，敏感方法等。

```java
/**
 * 创建用户
 *
 * @param userAddRequest
 * @param request
 * @return
 */
@PostMapping("/add")
@AuthCheck(mustRole = UserConstant.ADMIN_ROLE)
public BaseResponse<Long> addUser(@RequestBody UserAddRequest userAddRequest, HttpServletRequest request) {
    if (userAddRequest == null) {
        throw new BusinessException(ErrorCode.PARAMS_ERROR);
    }
    User user = new User();
    BeanUtils.copyProperties(userAddRequest, user);
    boolean result = userService.save(user);
    ThrowUtils.throwIf(!result, ErrorCode.OPERATION_ERROR);
    return ResultUtils.success(user.getId());
}

/**
```

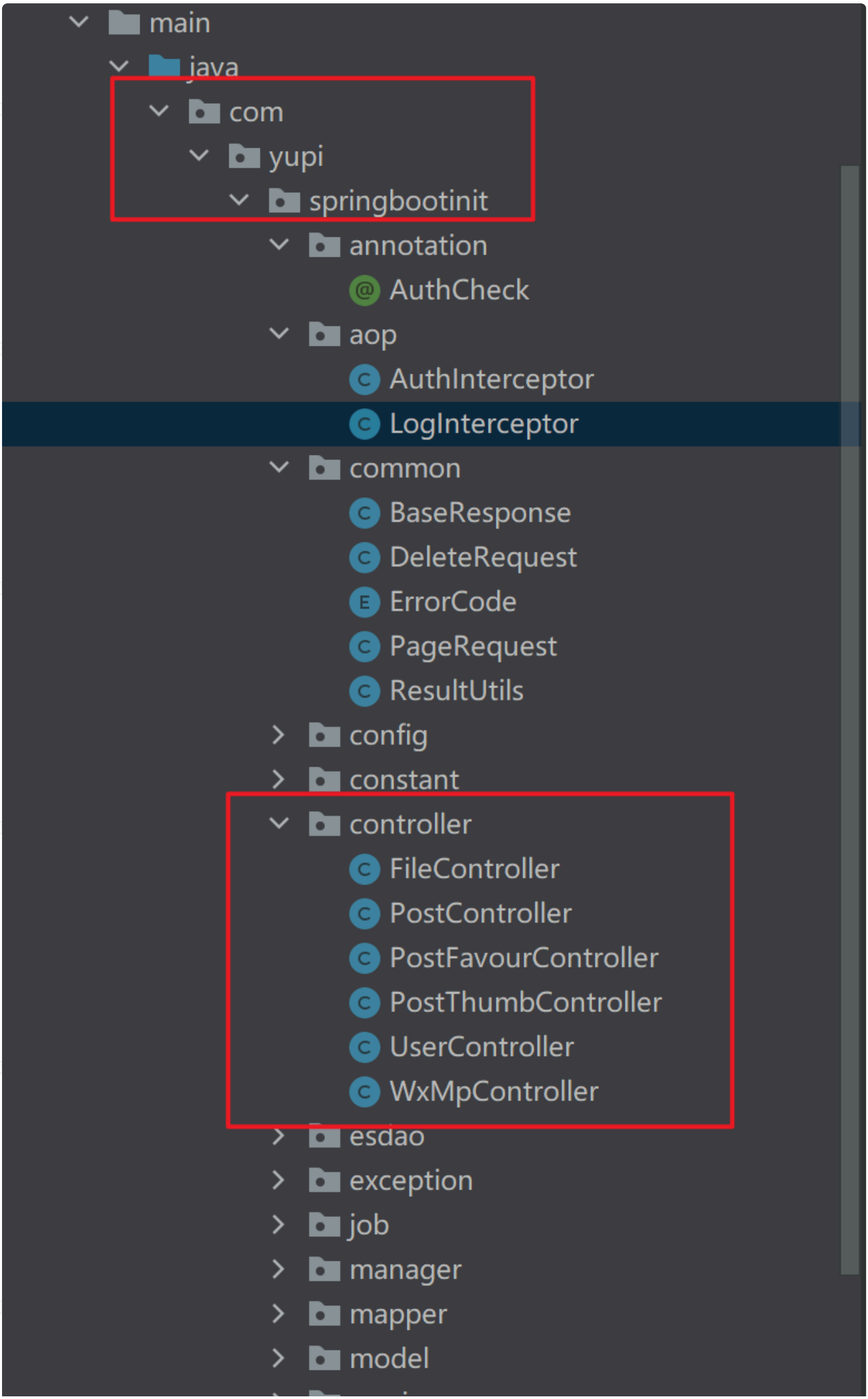
像创建用户这个方法就是只有管理员能够使用，用@AuthCheck自定义注解，然后写上使用该方法要有的权限即可。权限校验器因为是@Around环绕通知并且表明在有authCheck注解的方法周围执行如下逻辑。

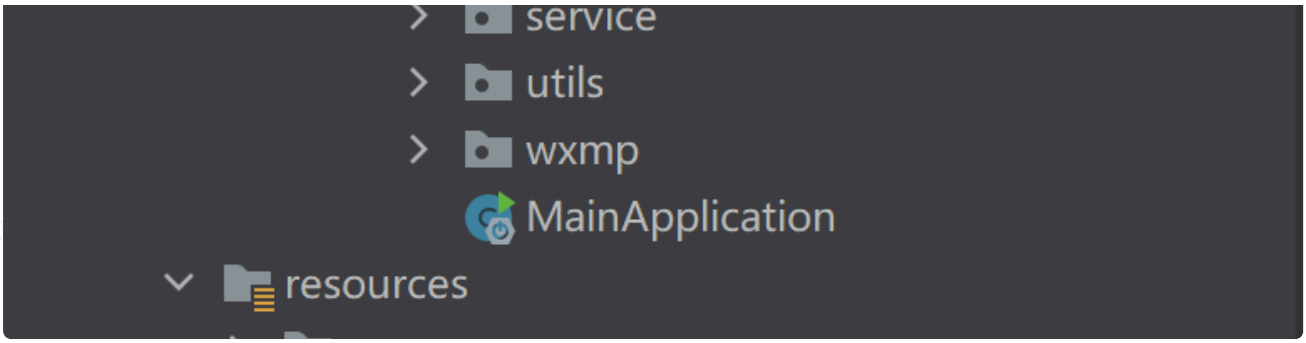
@Around环绕通知在也就是在方法执行前后额外添加的逻辑。AOP功能的诠释，如果有不懂这段逻辑的可以去回顾下Spring的AOP功能， **面试** 中经常会提及，能够回答出AOP的实际项目应用也是很不错的一个点。

接下来介绍一下 LogInterceptor

```java
 **/
@Aspect
@Component
@Slf4j
public class LogInterceptor {

    /**
     * 执行拦截
     */
    @Around("execution(* com.yupi.springbootinit.controller.*.*(..))")
    public Object doInterceptor(ProceedingJoinPoint point) throws Throwable {
        // 计时
        StopWatch stopWatch = new StopWatch();
        stopWatch.start();
        // 获取请求路径
        RequestAttributes requestAttributes = RequestContextHolder.currentRequestAttributes();
        HttpServletRequest httpServletRequest = ((ServletRequestAttributes) requestAttributes).getRequest();
        // 生成请求唯一 id
        String requestId = UUID.randomUUID().toString();
        String url = httpServletRequest.getRequestURI();
        // 获取请求参数
        Object[] args = point.getArgs();
        String reqParam = "[" + StringUtils.join(args, delimiter: ", ") + "]";
        // 输出请求日志
        log.info("request start, id: {}, path: {}, ip: {}, params: {}", requestId, url,
                httpServletRequest.getRemoteHost(), reqParam);
        // 执行原方法
        Object result = point.proceed();
        // 输出响应日志
        stopWatch.stop();
        long totalTimeMillis = stopWatch.getTotalTimeMillis();
        log.info("request end, id: {}, cost: {}ms", requestId, totalTimeMillis);
        return result;
    }
```

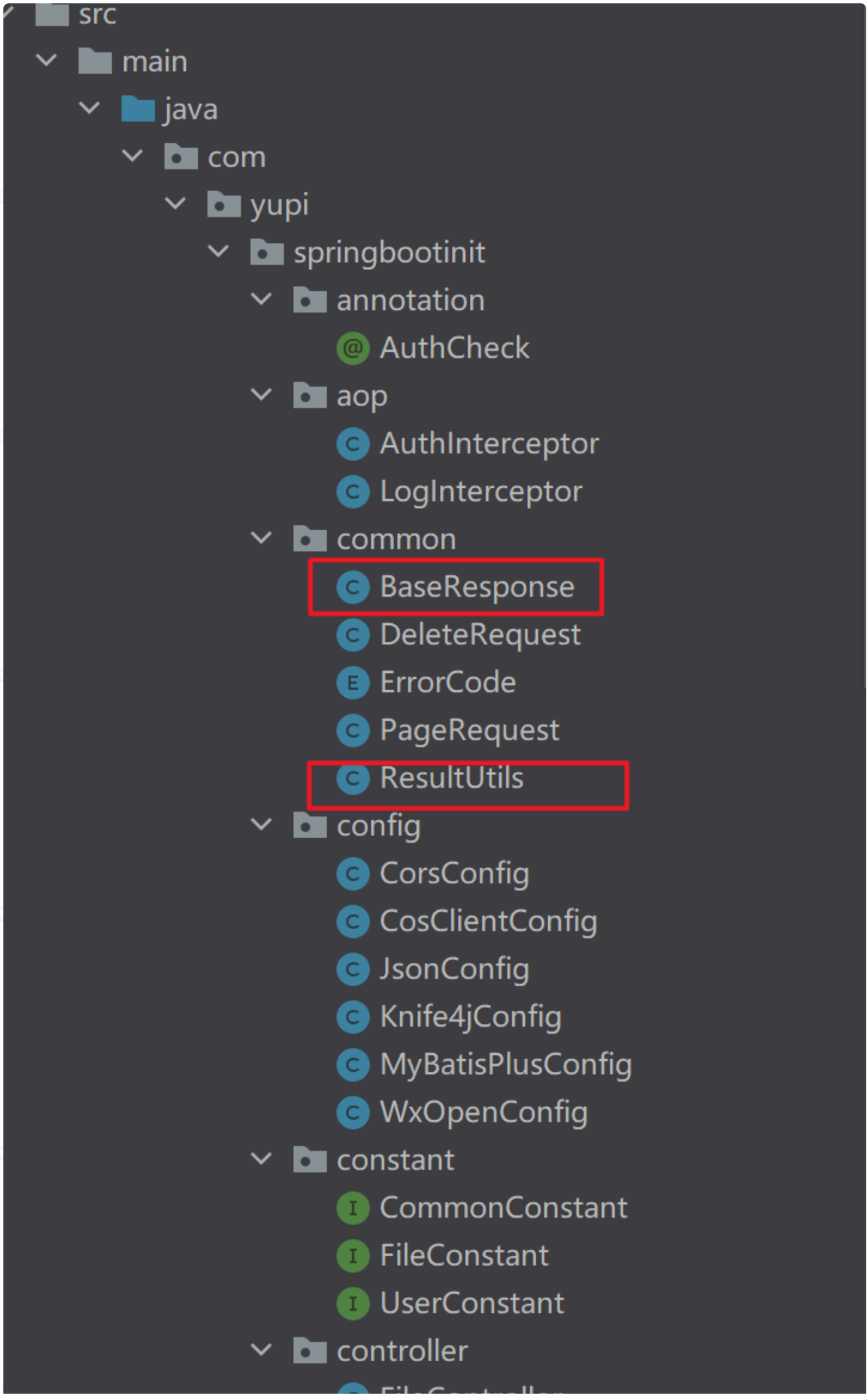
简要说明：请求日志拦截器，用于输出请求日志，@Around是环绕通知，然后用了切入点表达式，要拦截哪个包或者哪些包下面的哪个方法或者是全部方法。这段切入点表达式的功能就是对com.yupi.springbootinit.controller中所有方法进行拦截。也就是说控制层执行方法就会打印日志进行输出。有利用异常信息的捕获和后端调试debug。


main
java
com
yupi
springbootinit
annotation
AuthCheck
aop
AuthInterceptor
LogInterceptor
common
BaseResponse
DeleteRequest
ErrorCode
PageRequest
ResultUtils
config
constant
controller
FileController
PostController
PostFavourController
PostThumbController
UserController
WxMpController
esdao
exception
job
manager
mapper
model



2.3 通用响应类

BaseResponse、ResultUtils和 ErrorCode


src
main
java
com
yupi
springbootinit
annotation
AuthCheck
aop
AuthInterceptor
LogInterceptor
common
BaseResponse
DeleteRequest
ErrorCode
PageRequest
ResultUtils
config
CorsConfig
CosClientConfig
JsonConfig
Knife4jConfig
MyBatisPlusConfig
WxOpenConfig
constant
CommonConstant
FileConstant
UserConstant
controller

下面是 BaseResponse 的介绍：

通用返回类，code 表示响应状态码，data 存放返回的数据，message :成功或者失败的额外信息。

```java
package com.yupi.springbootinit.common;

import java.io.Serializable;
import lombok.Data;

/**
 * 通用返回类
 *
 * @param <T>
 * @author <a href="https://github.com/liyupi">程序员鱼皮</a>
 * @from <a href="https://yupi.icu">编程导航知识星球</a>
 */
43 usages
@Data
public class BaseResponse<T> implements Serializable {

    1 usage
    private int code;

    1 usage
    private T data;

    1 usage
    private String message;

    5 usages
    public BaseResponse(int code, T data, String message) {
        this.code = code;
        this.data = data;
        this.message = message;
    }

    public BaseResponse(int code, T data) { this(code, data, message: ""); }

    1 usage
    public BaseResponse(ErrorCode errorCode) { this(errorCode.getCode(), data: null, errorCode.getMessage()); }
}
```

下面是 ResultUtils 的介绍：

主要用于简化 BaseResponse 的操作，将成功，失败的一些通用情况进行的静态方法的封装，然后可以很方便的进行调用，比如调用 success 方法，响应状态码就是 0，然后会将data封装到 BaseResponse 的data属性，message为 "ok"。如果你的前端想要响应状态吗为 200，那么将这里的 0 改成 200就可以了,这边的message为固定消息 "ok"。可以再设置一个静态方法，进行方法重构,success形参为(T data,String message)，然后可以动态定义成功的时候的返回消息。

```java
package com.yupi.springbootinit.common;

/**
 * 返回工具类
 *
 * @author <a href="https://github.com/liyupi">程序员鱼皮</a>
 * @from <a href="https://yupi.icu">编程导航知识星球</a>
 */
public class ResultUtils {

    /**
     * 成功
     *
     * @param data
     * @param <T>
     * @return
     */
    26 usages
    public static <T> BaseResponse<T> success(T data) { return new BaseResponse<>( code: 0, data,  message: "ok"); }

    /**
     * 失败
     *
     * @param errorCode
     * @return
     */
    public static BaseResponse error(ErrorCode errorCode) { return new BaseResponse<>(errorCode); }

    /**
     * 失败
     *
     * @param code
     * @param message
     * @return
     */
    public static BaseResponse error(int code, String message) { return new BaseResponse(code,  data: null, message); }
```

下面是 ErrorCode 的介绍：

ErrorCode配合上面的ResultUtils使用,可以定义枚举类将常规的响应状态码和响应信息进行封装。比如: 无权限访问(40300),服务器内部异常(50000),你可以添加自定义的一些专属你自己项目的响应状态码，例如: API 项目 接口调用失败,可以是 INTERFACE_ERROR(50003,"接口调用失败")。

```java
package com.yupi.springbootinit.common;

/**
 * 自定义错误码
 *
 * @author <a href="https://github.com/liyupi">程序员鱼皮</a>
 * @from <a href="https://yupi.icu">编程导航知识星球</a>
 */
public enum ErrorCode {

    SUCCESS( code: 0,  message: "ok"),
    43 usages
    PARAMS_ERROR( code: 40000,  message: "请求参数错误"),
    2 usages
    NOT_LOGIN_ERROR( code: 40100,  message: "未登录"),
    5 usages
    NO_AUTH_ERROR( code: 40101,  message: "无权限"),
    7 usages
    NOT_FOUND_ERROR( code: 40400,  message: "请求数据不存在"),
    1 usage
    FORBIDDEN_ERROR( code: 40300,  message: "禁止访问"),
    10 usages
    SYSTEM_ERROR( code: 50000,  message: "系统内部异常"),
    5 usages
    OPERATION_ERROR( code: 50001,  message: "操作失败");

    /**
     * 状态码
     */
    2 usages
    private final int code;

    /**
     * 信息
     */
    2 usages
    private final String message;
```

2.4 配置类

JsonConfig、Knife4jConfig、MyBatisPlusConfig、CorsConfig、CosClientConfig、WxOpenConfig

下面是 JsonConfig 的介绍：

```java
package com.yupi.springbootinit.config;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.module.SimpleModule;
import com.fasterxml.jackson.databind.ser.std.ToStringSerializer;
import org.springframework.boot.jackson.JsonComponent;
import org.springframework.context.annotation.Bean;
import org.springframework.http.converter.json.Jackson2ObjectMapperBuilder;

/**
 * Spring MVC Json 配置
 *
 * @author <a href="https://github.com/liyupi">程序员鱼皮</a>
 * @from <a href="https://yupi.icu">编程导航知识星球</a>
 */
@JsonComponent
public class JsonConfig {

    /**
     * 添加 Long 转 json 精度丢失的配置
     */
    @Bean
    public ObjectMapper jacksonObjectMapper(Jackson2ObjectMapperBuilder builder) {
        ObjectMapper objectMapper = builder.createXmlMapper(false).build();
        SimpleModule module = new SimpleModule();
        module.addSerializer(Long.class, ToStringSerializer.instance);
        module.addSerializer(Long.TYPE, ToStringSerializer.instance);
        objectMapper.registerModule(module);
        return objectMapper;
    }
}
```

@JsonComponent作用： 自定义序列化和反序列JSON数据，Spring Boot 默认使用 JackSon 进行序列化和反序列化。

怎么防止丢失？ 用@Bean覆盖组件后，重写逻辑代码，将包装类 Long 和基础数据类型 long 转化成字符串序列化成字符串防止在序列化的时候丢失精度。

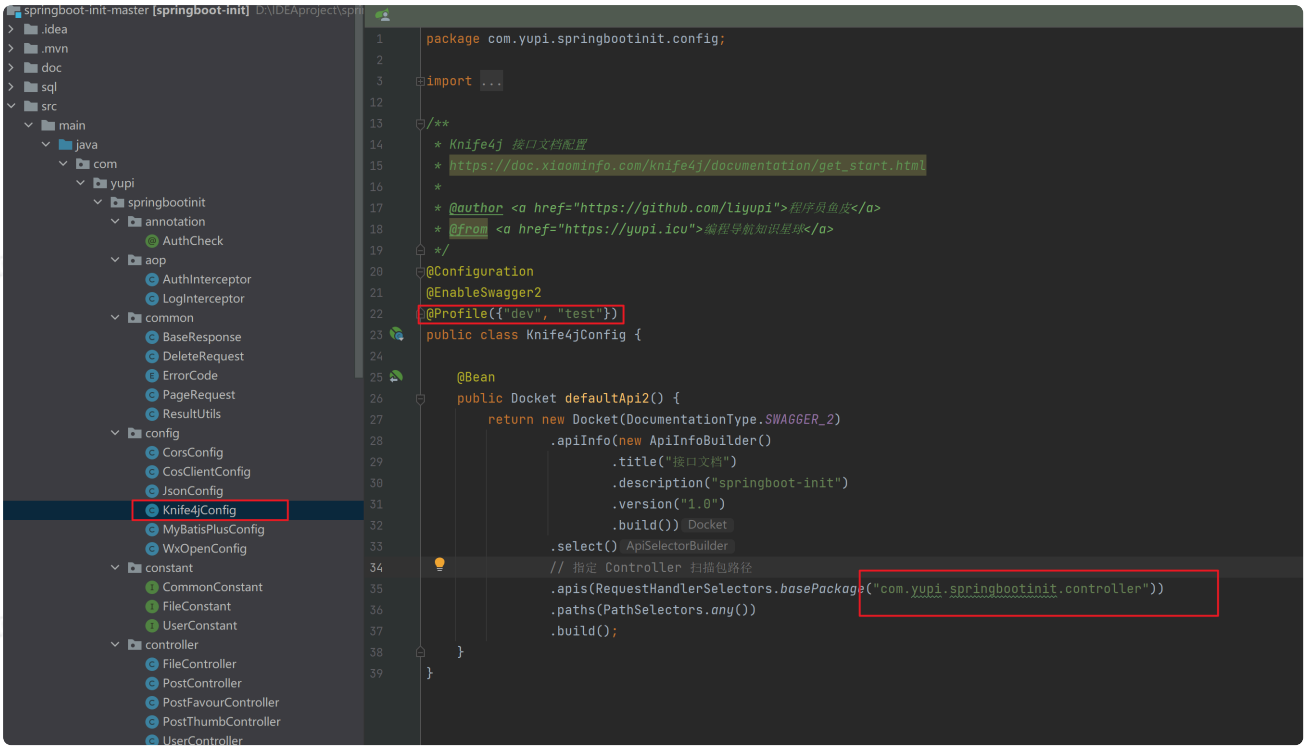
精度丢失场景： id在数据库是 BigInteger 类型，雪花算法生成id大于17位，因此在序列化的时候会产生精度丢失。

下面是 Knife4jConfig 的介绍:

Knife4jConfig 用于后端接口在线测试的配置类。

图中@Profile用于指定在什么环境配置，和配置文件的 spring.profiles.active 是相互关联的。因此只在开发环境和测试环境中生效。

basePackage:用于指定要扫描 Controller 层的包， **如果你的控制层的包名有所不同，需要进行修改，Knife4j才能生效。**

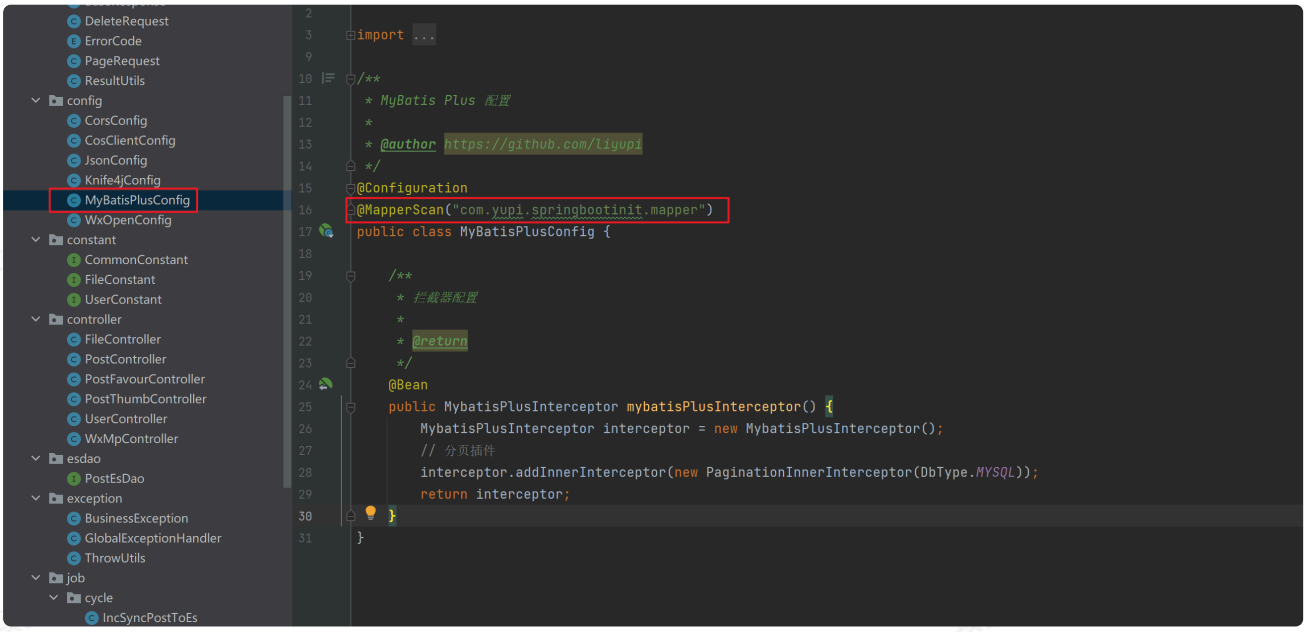


下面是 MyBatisPlusConfig 的介绍：

@MapperScan用于指定扫描的路径，如果你的项目的基础包名路径不同，需要修改为你自己的。

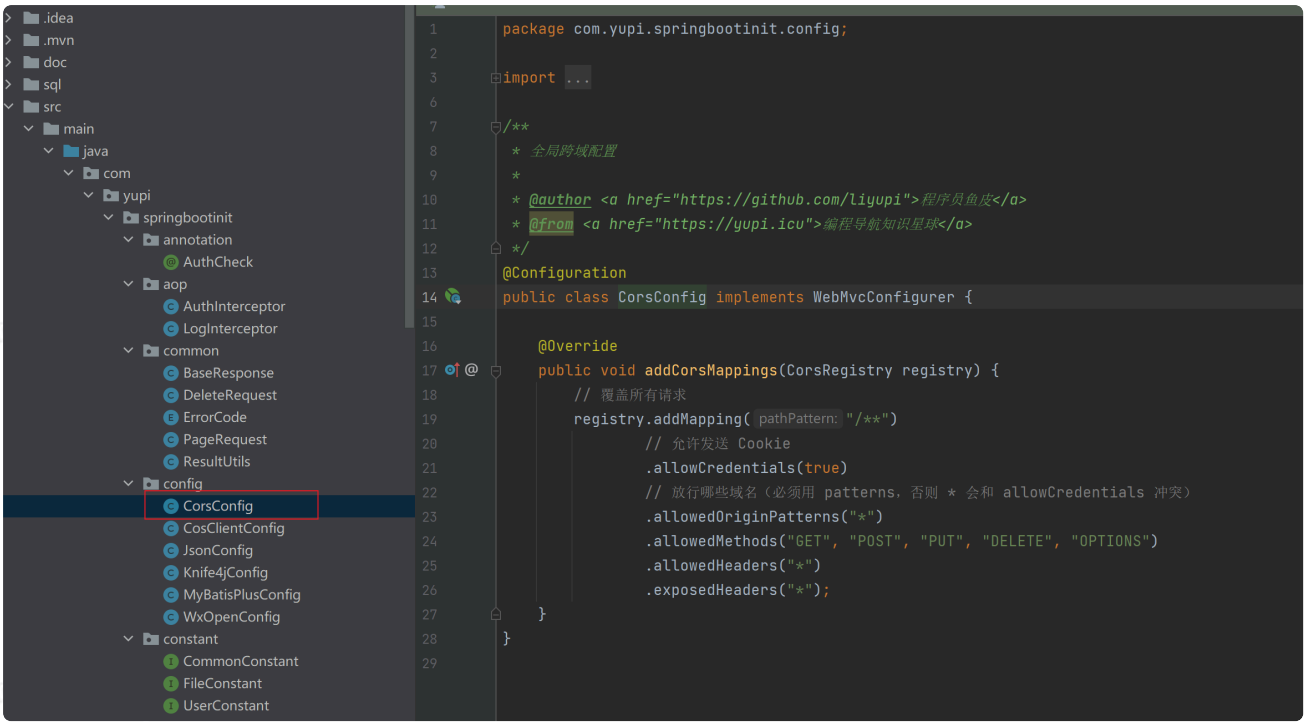
这边是用@Bean进行组件的注入，然后添加了分页插件。

MyBatisPlus还有很多插件，例如：乐观锁插件、多租户插件、动态表明插件、数据权限插件等等。可以去官网 <https://www.baomidou.com/pages/24112f/> 查看。



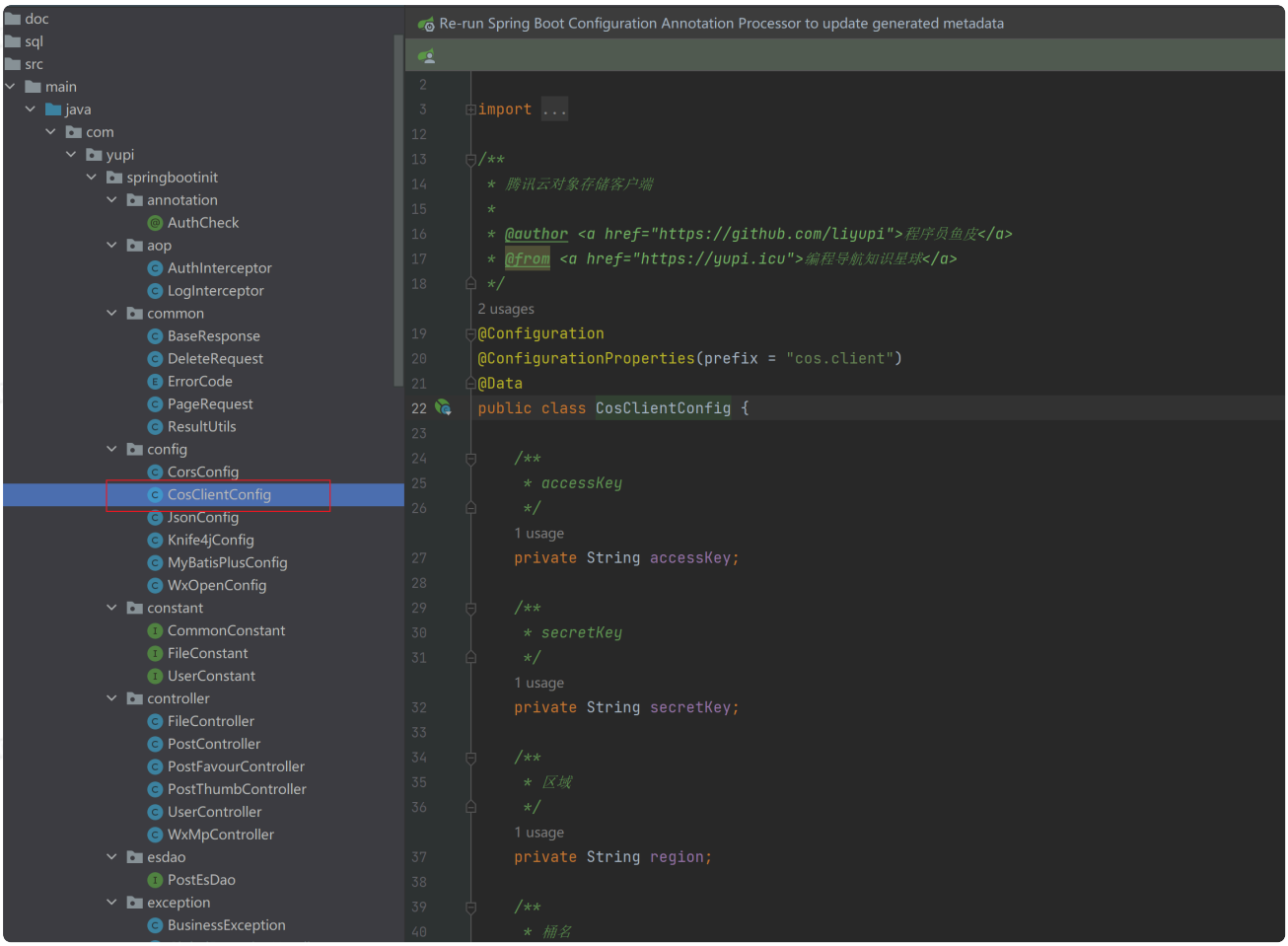
下面是 CorsConfig 的介绍：

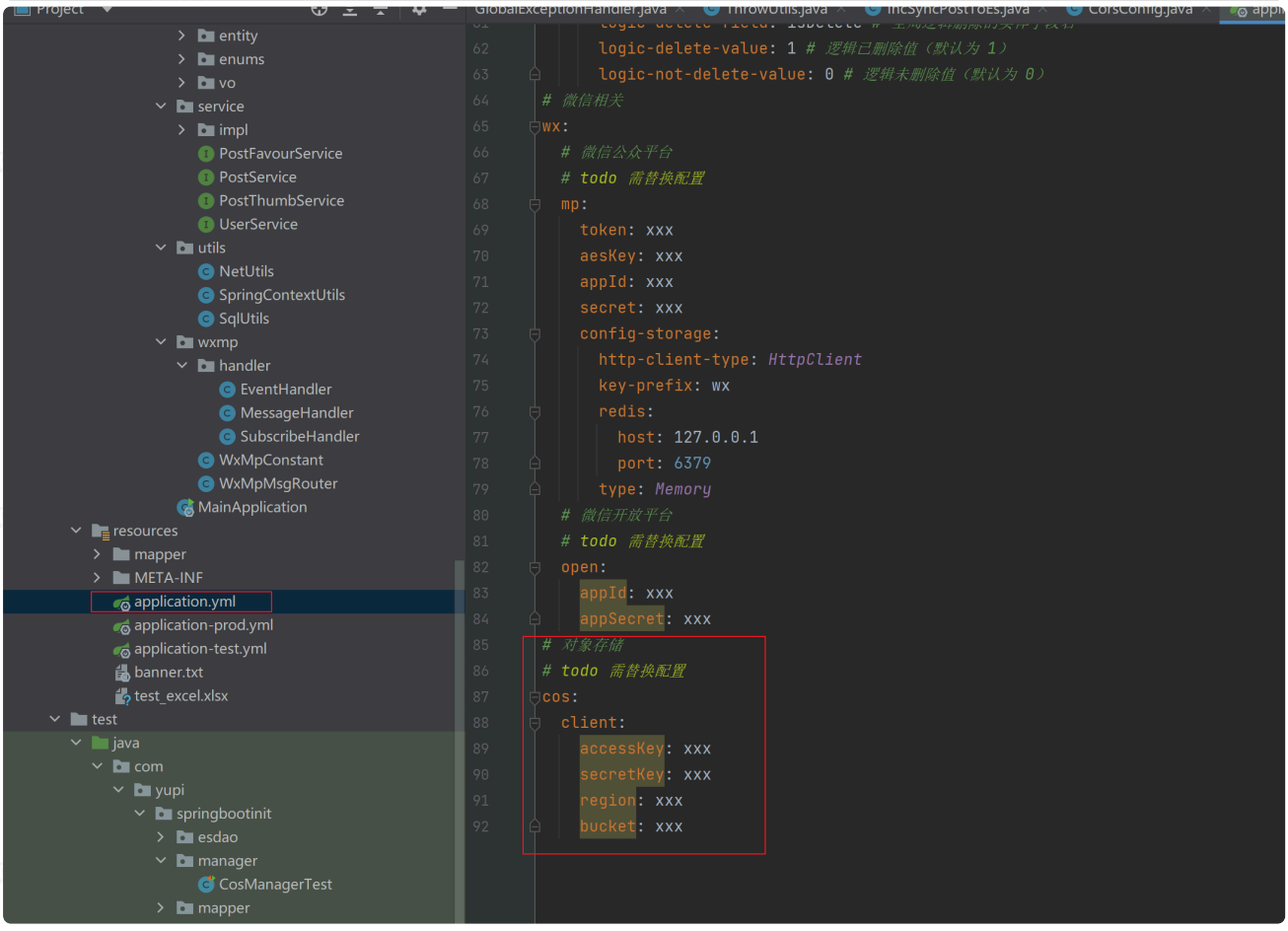
CorsConfig 用来解决全局跨域配置问题，可以指定请求方法、是否允许发送 Cookie、放行哪些特定域名或 ip、允许哪些请求头。



下面是 CosClientConfig 的介绍：

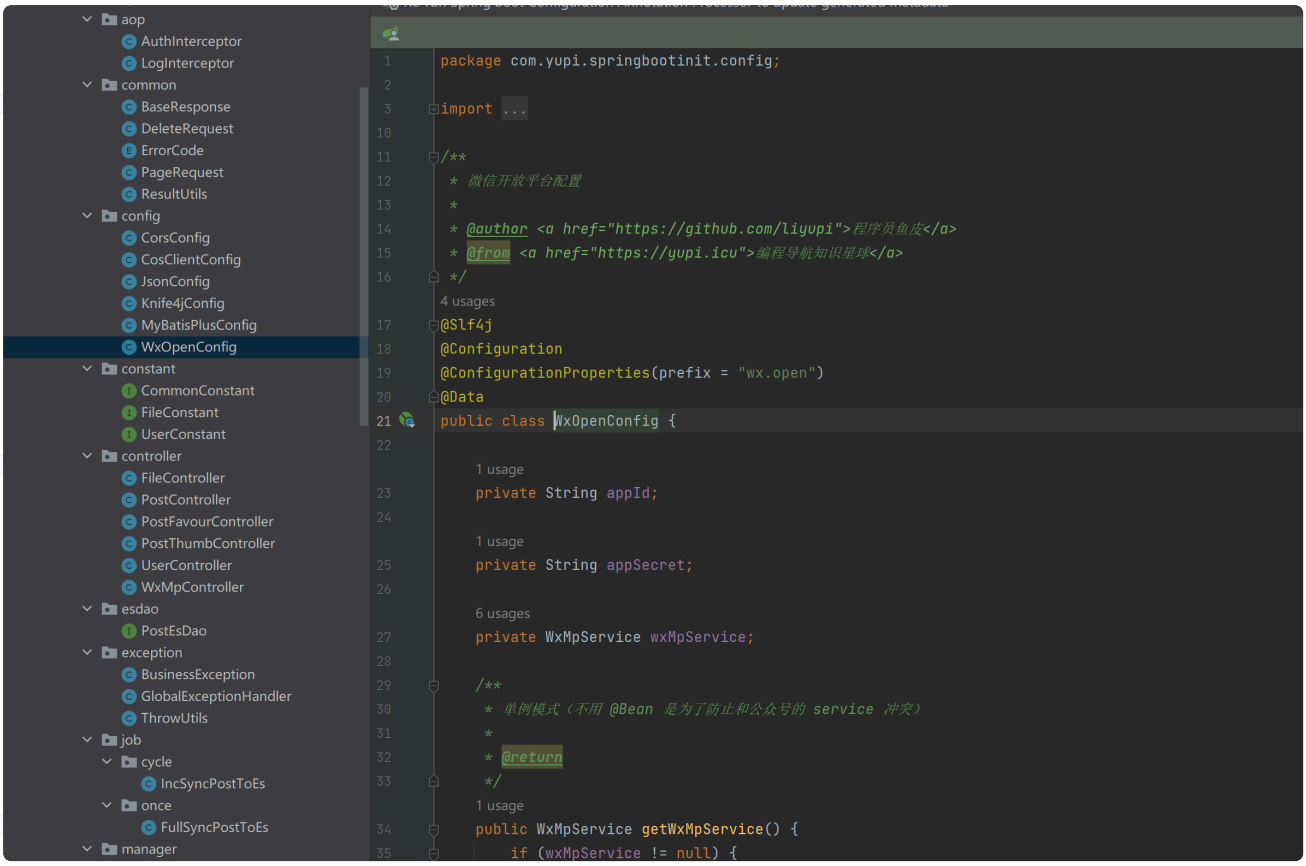
直接将你的accessKey、secretKey、region、bucket进行替换即可，在application.yml中做替换，然后配合工具类即可使用对象云存储的功能。

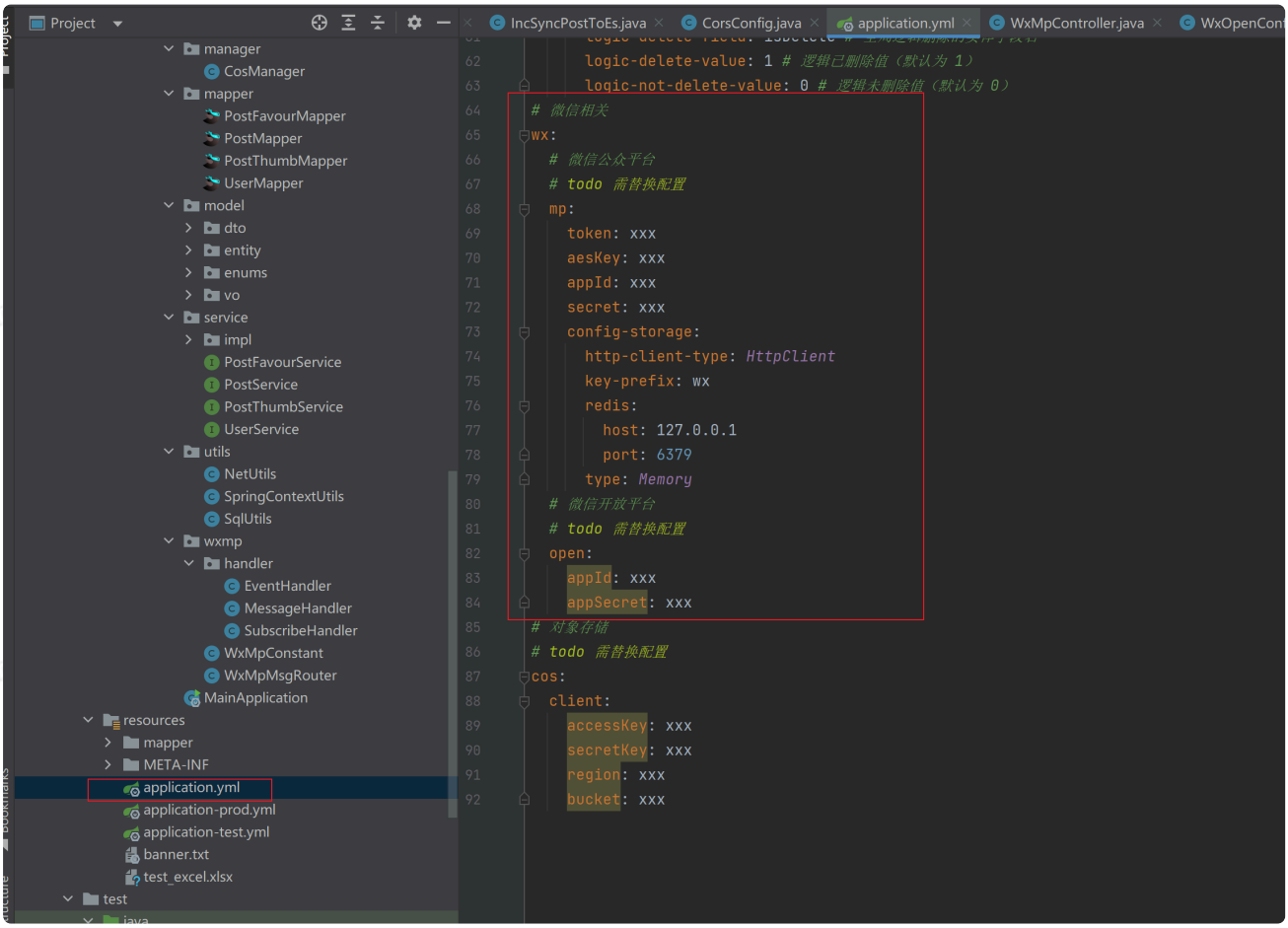




下面是WxOpenConfig 的介绍：

在微信开放平台获取 appId、appSecret、等配置后，在 applicaiton.yml 中替换即可。



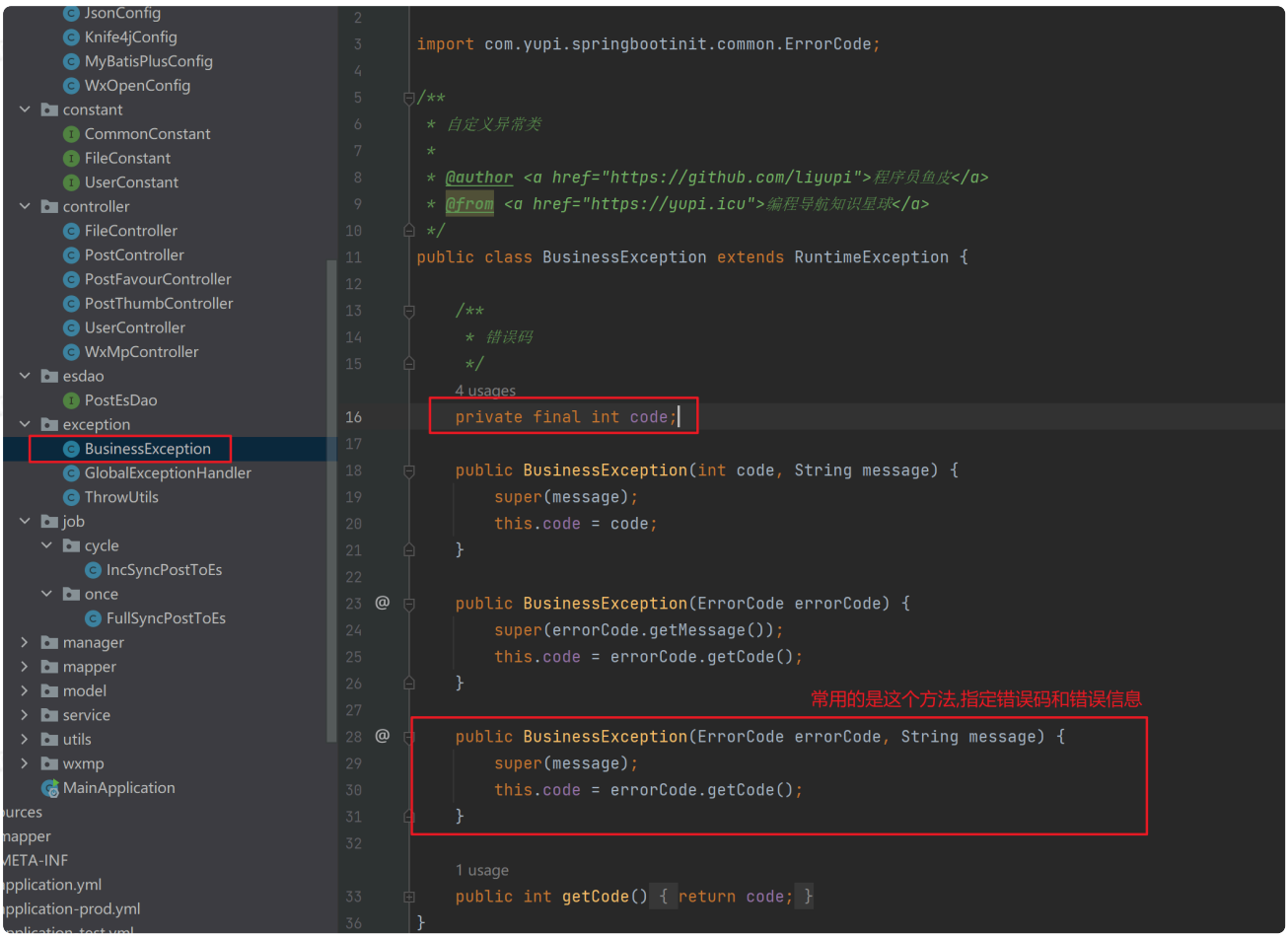


2.5 全局异常处理

BusinessException、GlobalExceptionHandler和ThrowUtils

下面是 BusinessException 的介绍：

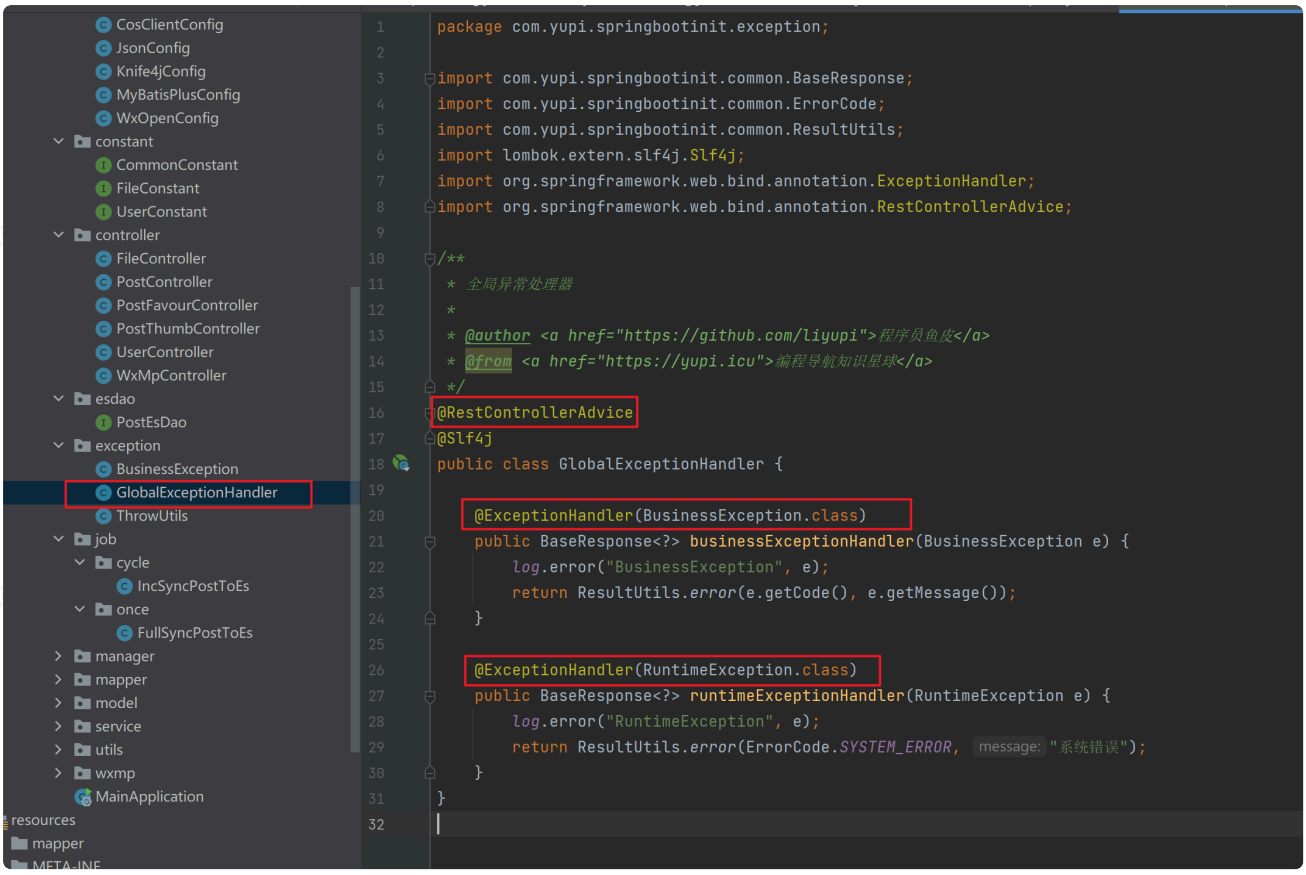
code：错误码，message因为继承了父类RunTimeException，因此就有属性message。 **结合ErrorCode使用** 。



下面是GlobalExceptionHandler的介绍:

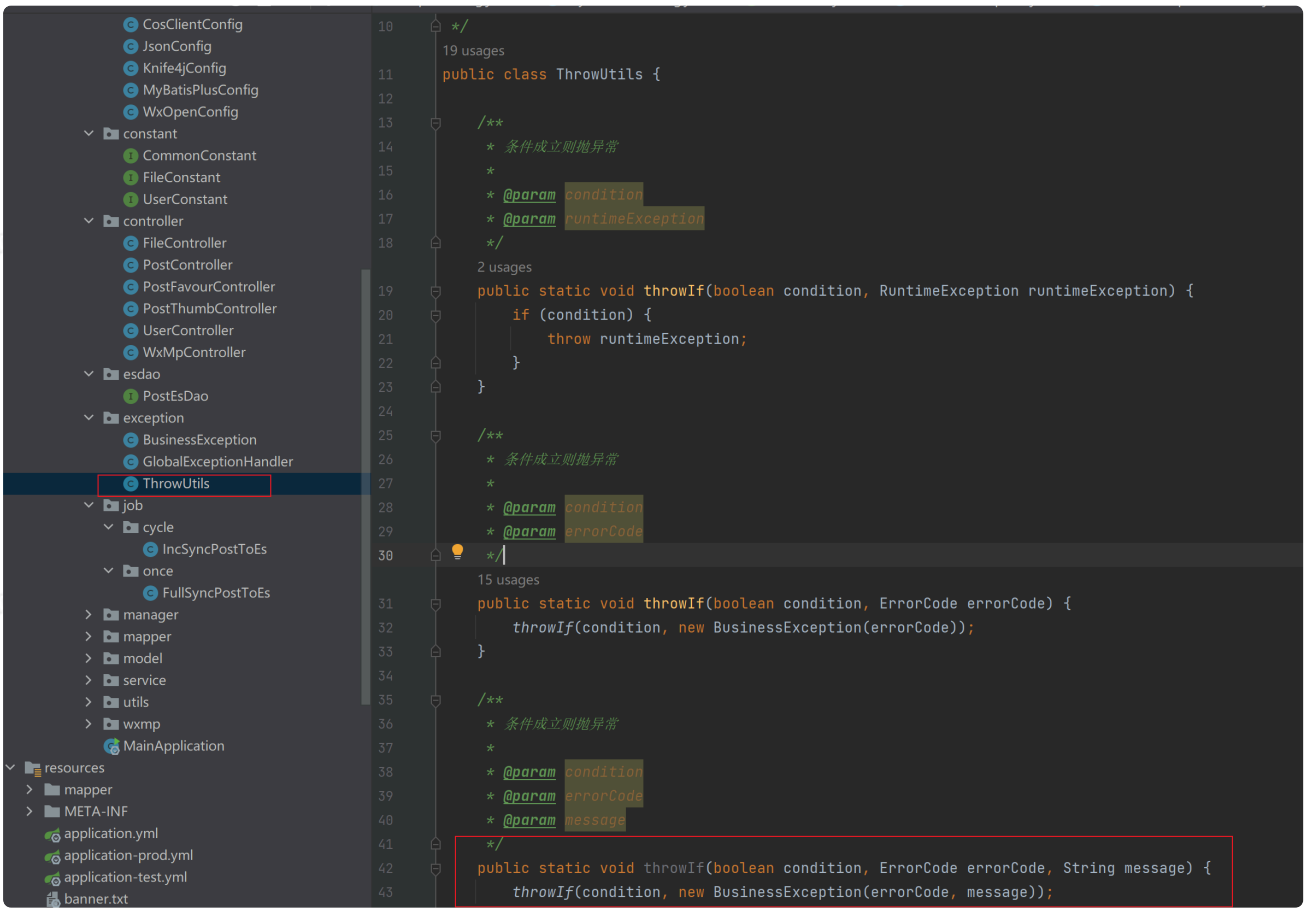
@RestControllerAdvice注解是@ControllerAdvie和@ResponseBody注解的组合，先捕获整个应用程序中抛出的异常，然后将异常处理方法的返回值将自动转换为HTTP响应的主体。

@ExceptionHandler注解用于指定什么异常需要被捕获。



下面是ThrowUtils的介绍:

一般用于请求参数的校验，如果请求参数为空，直接抛出业务异常，然后指明错误码ErroCode和message错误信息。



2.6 数据库和 ES 同步

IncSyncPostToEs和FullSyncPostToEs

下面是IncSyncPostToEs的介绍：

@Component 注解： 如果取消注解，就将这个定时任务加入到Spring 容器中，Spring Boot 启动类启动后将会开启这个定时任务。

@Scheduled: Spring Boot 的定时任务控制的注解，这里用于每分钟执行同步帖子的逻辑。

应用场景：

1.想要统计Top10的接口调用次数(API接口),在数据库量大后，每个用户去发送请求获取Top10的接口调用次数会造成数据库请求压力过大，此时可以写个定时任务，可以定时24小时，每天将Top10的接口调用次数同步到Redis存储，以接口名称和键以接口调用次数为值保存即可。实时性要求不太强的功能，可以采用定时任务。

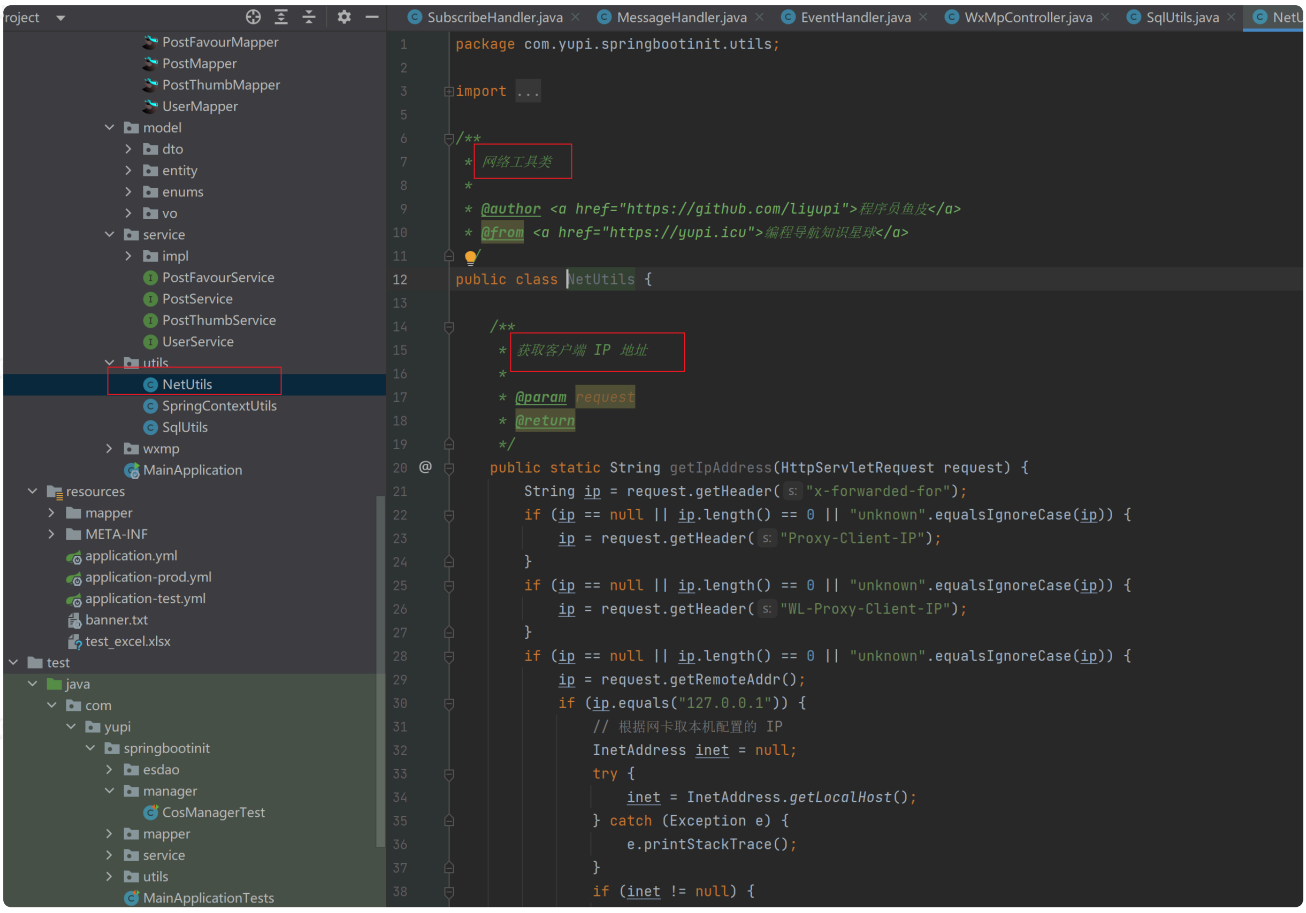
2.某个API接口不用用户传参，而且大多数时间回复的调用结果都是相通的，那么就可以采取定时任务，将这些不用用户传参，并且调用结果都是相同的接口定时同步到Redis存储，相对于数据库的IO读取，Redis存储会大大提升这些接口的QPS，可以自己在简历上进行量化处理。

2.7 工具类

NetUtils、SpringContextUtils、SqlUtils

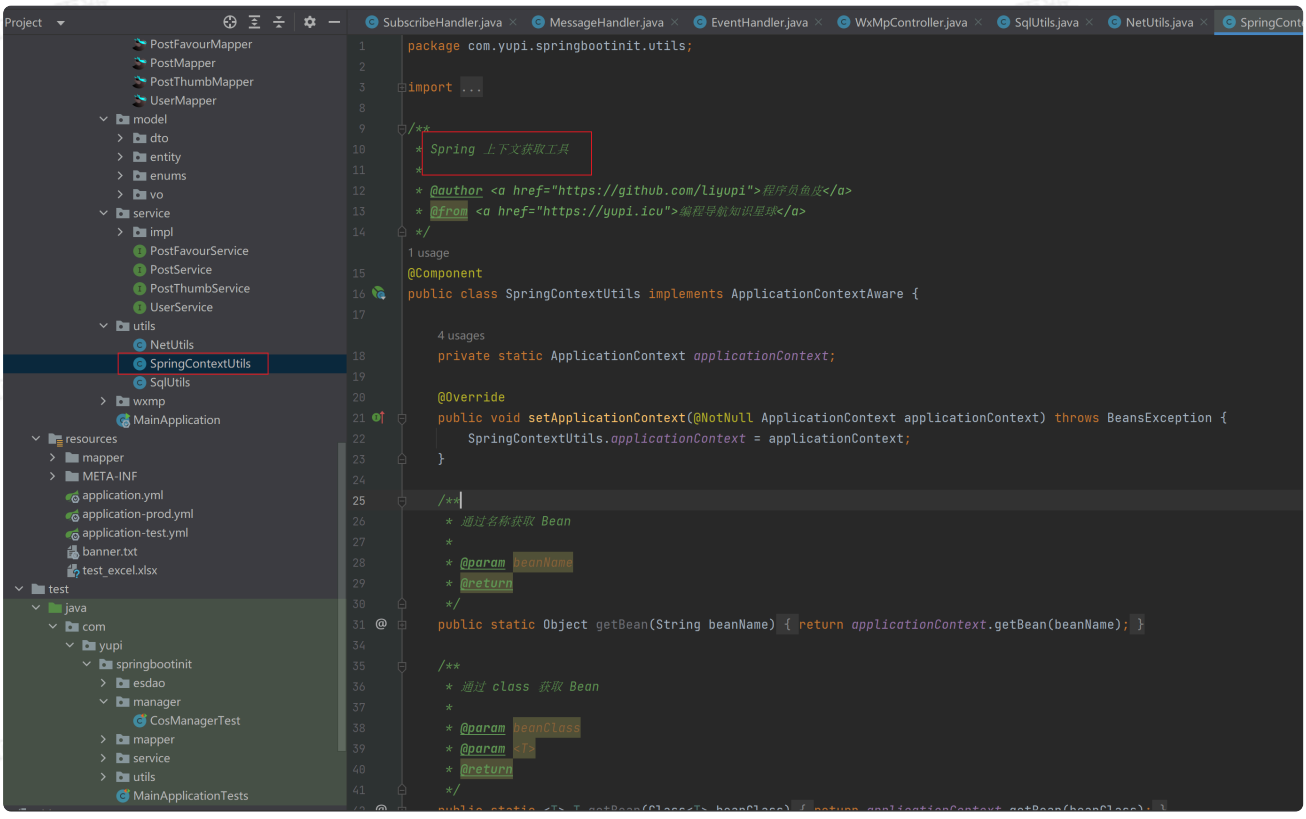
下面是 NetUtils 的介绍：

主要用于获取客户端的 IP 地址



下面是 SpringContextUtils 的介绍：

用于通过名称、类型、名称和类型获取 Spring 上下文的容器。



下面是 SqlUtils 的介绍：

主要用于检查 SQL 注入问题