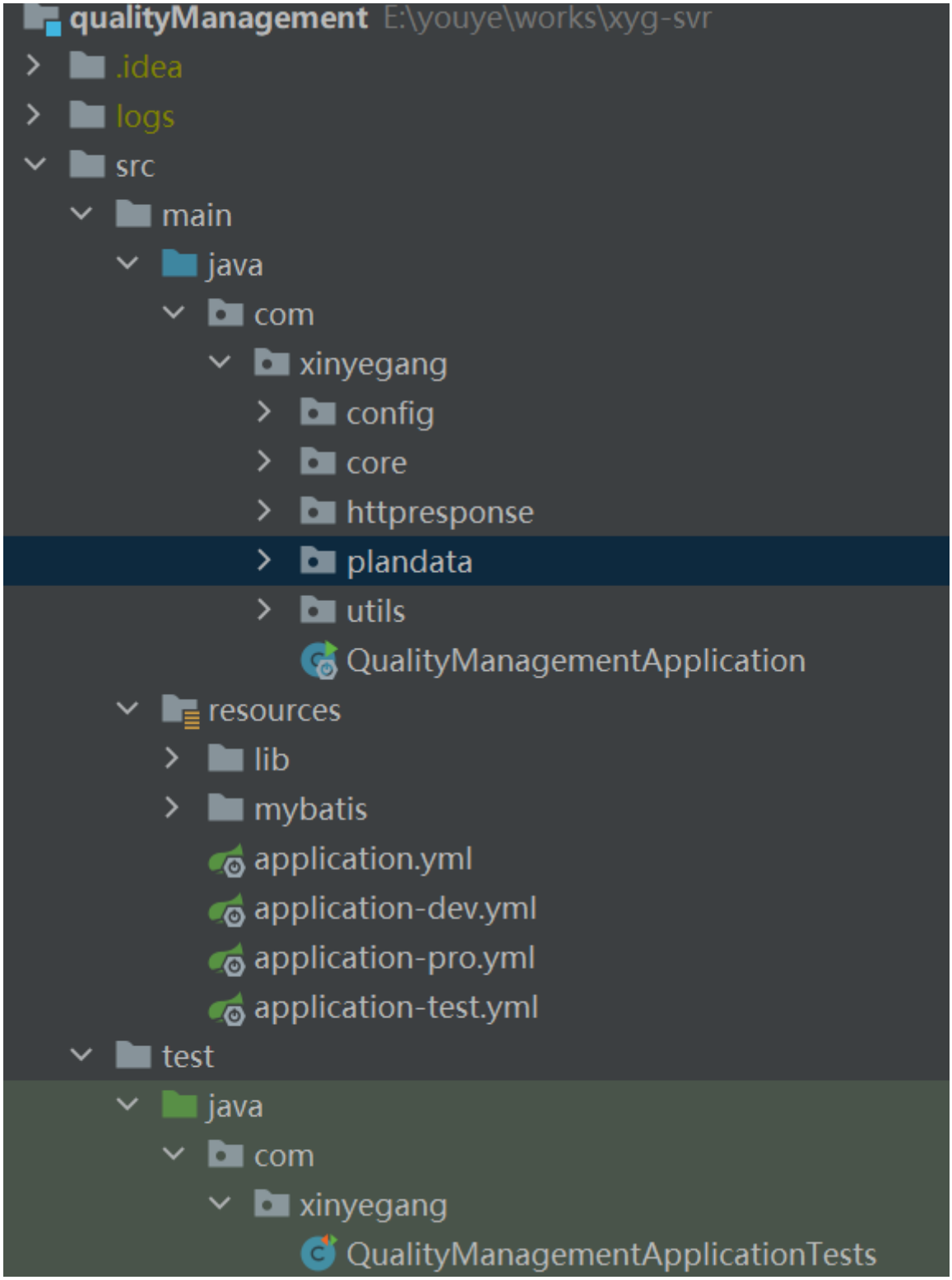
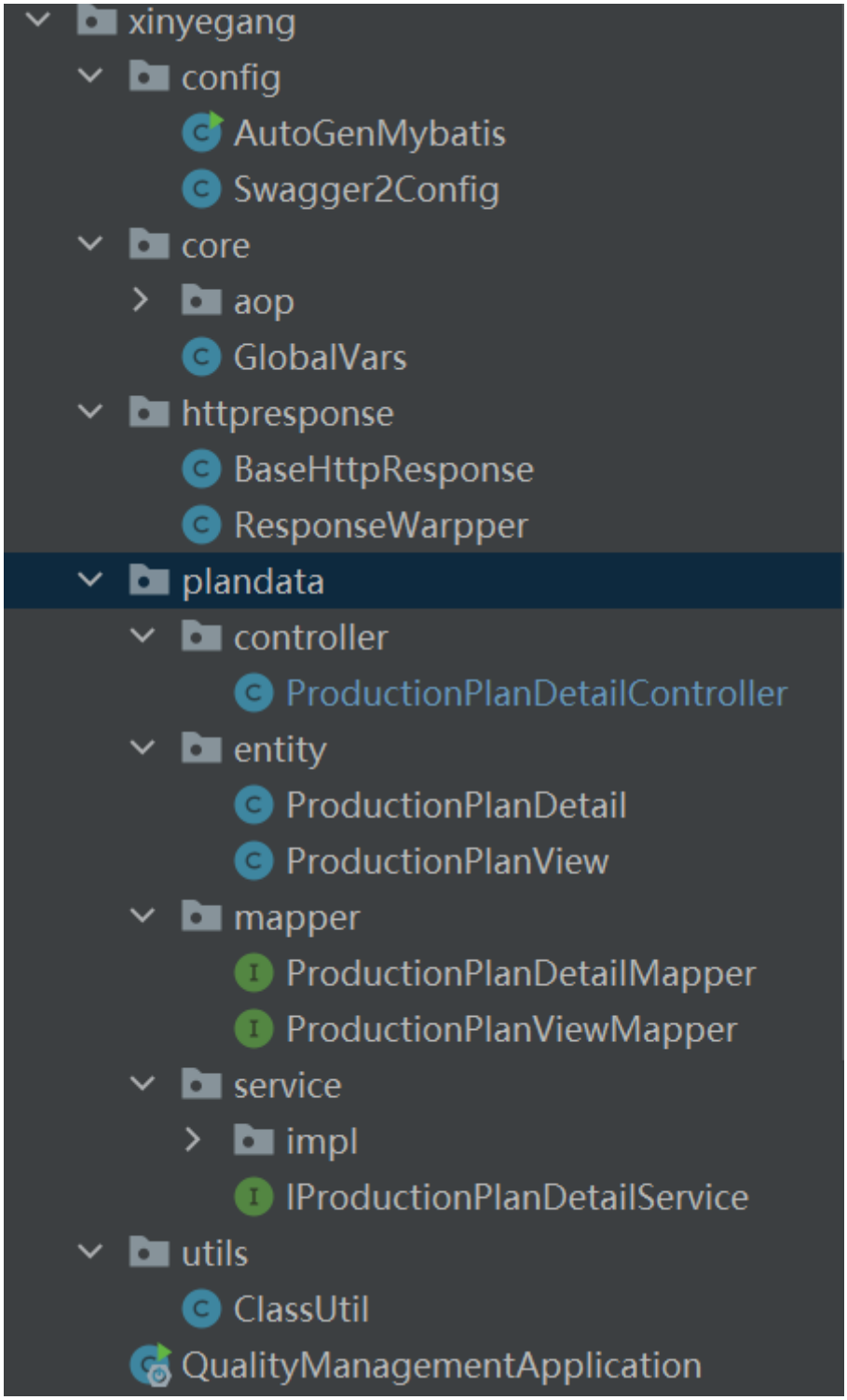
新冶钢后端实践之路

代码架构





定义实体类

plandata - entity目录下，每个class定义一张数据表。

单个class内定义数据表中的列名。

```
@Data
@EqualsAndHashCode(callSuper = false)
@ApiModel(value="ProductionPlanDetail对象", description="")
public class ProductionPlanDetail implements Serializable {
```

@Data

要使用 @Data 注解要先引入 `lombok`，它是一个工具类库，可以用简单的注解形式来简化代码，提高开发效率。详细参考：https://www.jianshu.com/p/c1ee7e4247bf

@Data注在类上，提供类的get、set、equals、hashCode、canEqual、toString方法，而不用手动一个个声明，大大提高了开发效率。

@EqualsAndHashCode

自动的给类实现equals方法和hashcode方法，callSuper = false表示不继承父类。

@ApiModel

标记类是swagger的解析类。

```
@ApiModelProperty(value = "计划号")
@TableField("PLAN_NO")
public String planNo;

@ApiModelProperty(value = "计划执行顺序号")
@TableField("PLAN_SEQ")
public Integer planSeq;
```

@ApiModelProperty

使用在被 @ApiModel 注解的模型类的属性上，表示对model属性的说明或者数据操作更改 。参考链接：http://www.mamicode.com/info-detail-3010881.html

- value：属性简要说明
- name：运行覆盖属性的名称，重写属性名称

  ...

@TableField

MybatisPlus中的注解。参考链接：https://blog.csdn.net/qq_40241957/article/details/101772536

- value：字段值，标定数据库中的列名，可与被其注释的变量名形成映射，我们在代码里只需要操作映射后的变量名。
- update：预处理set字段自定义注入

  `@TableField(.. , update="%s+1")` 使用mybatisplus自带的insert()方法向数据库插入数据时，假设我们给age字段赋值为1，但是我们在age字段上的@TableField注解里面加了update="%s+1"，那么插入到数据库的值就是age=2,而不是age=1了。
- exist：是否为数据库表字段，默认true存在，false不存在
- fill 字段填充标记

  ...

定义Service和其实现类

IProductionPlanDetailService.java

`public interface selfService extend IService<实体类>`

```
public interface IProductionPlanDetailService extends IService<ProductionPlanDetail> {

    /**
     * 获得待核对计划列表
     */
    public List<ProductionPlanView> GetList(String heatNo, String planNo, String groupId, String slabNo);

    /**
     * 初始化下拉框条件
     */
    public HashMap<String, List<Object>> GetSearchConditions();

}
```

ProductionPlanDetailServiceImpl.java

```
@Service
@Slf4j
public class ProductionPlanDetailServiceImpl extends
ServiceImpl<ProductionPlanDetailMapper, ProductionPlanDetail> implements
IProductionPlanDetailService {
    @Autowired
    ProductionPlanViewMapper productionPlanViewMapper;

    @Autowired
    ProductionPlanDetailMapper productionPlanDetailMapper;

    ...
```

Service接口继承于MyBatis-Plus封装的IService接口。

它采用 get 查询单行， remove 删除，list 查询集合， page 分页，前缀命名方式区分 Mapper 层接口避免混淆。

通用Service的使用场景是，如果方法很简单，例如，就是一个插入，或者根据条件更新，就不用在再Service写方法了，使用通用Service提供的方法即可。

如果有较多的业务逻辑，一般在继承后的Service中自定义方法，并在其实现类中实现即可。

注意这种自定义Service与其实体类和实现类的绑定方法：

`public class selfServiceImpl extend ServiceImpl<实体类的Mapper, 实体类> implements selfService`

- selfServiceImpl ： 自定义的实现类
- ServiceImpl：MyBatis-Plus提供的接口
- selfService：自定义的接口

定义Mapper

每个数据表中的实体类，在mapper目录下对应一个Mappper的接口。

```
public interface ProductionPlanDetailMapper extends BaseMapper<ProductionPlanDetail> {

    /*
    采用注解@Select方式查询数据库
     */
    @Select("select * from production_plan_detail")
    List<Map<String,Object>> getPlanZ();

    @Select("update loading_record set SLAB_NO = ${list}")
    List<Map<String,Object>> updateLoadTable(@Param("list")String list);

    @Select("set foreign_key_checks = 0")
    List<Map<String,Object>> deleteForeigrnKey();
    @Select("set foreign_key_checks = 1")
    List<Map<String,Object>> updateForeigrnKey();


}
```

接口继承于mybatis-plus的BaseMapper基类，并通过 `BaseMapper<ProductionPlanDetail>` 的方式与实体类进行映射。

@Select

使用注解的Mybatis与SpringBoot结合的方式。

不建议这么做，建议使用Mybatis-plus封装的BaseMapper中的方法。https://www.notion.so/stawary/BaseMapper-a38a468066654edea648277b04c1984d

条件构造器

BaseMapper中封装的方法有的可以直接传入主键进行操作，如 `deleteById` 、 `updateById` ，有些需要根据map进行操作，如 `selectByMap` ，有些需要写sql语句进行操作，sql的操作被封装到了条件构造器中，如 `selectList` 。

```
/**
 * 根据 entity 条件，查询全部记录
 *
 * @param queryWrapper 实体对象封装操作类（可以为 null）
 */
List<T> selectList(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
```

使用方法：

```
QueryWrapper<T> wrapper = new QueryWrapper<>();
wrapper.eq("standard_craft_card_id", id);
List<T> t = Mapper.selectList(wrapper);
```

详细内容查看：https://baomidou.com/guide/wrapper.html#alleq

定义Controller

controller文件目录下是用spring mvc框架编写的restful API 服务，定义了页面的路由和对应路由下的返回结果。

在 Spring Boot 中使用到 @Controller 及相关的注解如下，主要分为三个层面进行，请求前，处理中，返回。

应用场景	注解	注解说明
处理请求	@Controller	处理 Http 请求
处理请求	@RestController	@Controller 的衍生注解
路由请求	@RequestMapping	路由请求 可以设置各种操作方法
路由请求	@GetMapping	GET 方法的路由
路由请求	@PostMapping	POST 方法的路由
路由请求	@PutMapping	PUT 方法的路由
路由请求	@DeleteMapping	DELETE 方法的路由
请求参数	@PathVariable	处理请求 url 路径中的参数 /user/{id}
请求参数	@RequestParam	处理问号后面的参数
请求参数	@RequestBody	请求参数以json格式提交
返回参数	@ResponseBody	返回 json 格式

- @RestController 是 @Controller 的子集。
    - @Controller 一般应用在有返回界面的应用场景下.例如，管理后台使用了 thymeleaf 作为模板开发，需要从后台直接返回 Model 对象到前台，那么这时候就需要使用 @Controller 来注解。
    - @RestController 如果只是接口，那么就用 RestController 来注解。例如前端页面全部使用了 Html、Jquery来开发，通过 Ajax 请求服务端接口，那么接口就使用 @RestController 统一注解。
- @GetMapping、@PostMapping、@PutMapping、@DeleteMapping 是 @RequestMapping 的子集。

@RequestMapping

示例	说明
@RequestMapping("/index")	默认为 GET 方法的路由 /index
@RequestMapping(value="/index",method = RequestMethod.GET)	同上面一条
@RequestMapping(value="/add",method = RequestMethod.POST)	路由为 /add 的 POST 请求
@RequestMapping(value="/add",method = RequestMethod.POST),consumes="application/json"	路由为 /add 的 POST 请求，但仅仅处理 application/json 的请求
@RequestMapping(value="/add",method = RequestMethod.POST),produces="application/json"	路由为 /add 的 POST 请求，强调返回为 JSON 格式
@RequestMapping(value="/add",method = RequestMethod.POST),params="myParam=xyz"	路由为 /add 的 POST 请求，但仅仅处理头部包括 myParam=xyz 的请求
@RequestMapping(value="/add",method = RequestMethod.POST),headers="Referer=http://www.xyz.com/"	路由为 /add 的 POST 请求，但仅仅处理 来源为 www.xyz.com 的请求

- @GetMapping是一个组合注解，是@RequestMapping(method = RequestMethod.GET)的缩写。
- @PostMapping是一个组合注解，是@RequestMapping(method = RequestMethod.POST)的缩写。

@RequestParam

将请求参数绑定到controller的方法参数上。

语法：`@RequestParam(value=”参数名”,required=”true/false”,defaultValue=””)`

- value：参数名
- required：是否包含该参数，默认为true，表示该请求路径中必须包含该参数，如果不包含就报错。
- defaultValue：默认参数值，如果设置了该值，required=true将失效，自动为false,如果没有传该参数，就使用默认值

```
@RequestMapping("/test")
    public Method(@RequestParam(value="name",required=true,defaultValue="hello")
String name){
        // 可以直接操作变量name继续编写代码
    }
```

@PathVariable

通过 @PathVariable 可以将URL中占位符参数 `{xxx}` 绑定到处理器类的方法形参中 @PathVariable(“xxx“)

```
@RequestMapping("test/{id}/{name}")
public Method(@PathVariable("id") Long ids ,@PathVariable("name") String names){
        // 可以直接操作变量ids和names继续编写代码,传参的时候类型要一致
    }
```

@RequestBody

- 'content-type' : application/json

```
@PostMapping(path = "/demo")
public void demo1(@RequestBody Person person) {
    System.out.println(person.toString());
}
```

问题梳理

1. Controller、Mapper、实体类、Service类和实现类间的映射关系
2. 关于spring boot应用自动注入出现Consider defining a bean of type 'xxx' in your configuration问题解决方案。

   在xxxService上加上@Service注解，让spring能够扫到。