研发部代码开发规范

广东国地规划科技股份有限公司

2019 年 7 月 8 日

前言

本手册的旨在码出高效，码出质量。现代软件架构的复杂性需要协同开发完成，如何高效地协同呢？无规矩不成方圆，无规范难以协同。对软件来说，适当的规范和标准绝不是消灭代码内容的创造性、优雅性，而是限制过度个性化，以一种普遍认可的统一方式一起做事，提升协作效率，降低沟通成本。代码的字里行间流淌的是软件系统的血液，质量的提升是尽可能少踩坑，杜绝踩重复的坑，切实提升系统稳定性，码出质量。

目　录

版本	提出人	修改内容	日期
0.1	郝明才		2018/10/19
0.2	彭辉	jsp，css,js 文件命名改为驼峰	2018/10/19
0.3	彭辉	加入 B.7	2018/10/24
0.4	郝明才	更新细节	2019/1/7
1.0	郝明才	格式调整，文件夹规范添加	2019/7/8
1.1	肖俊杰	包目录规范说明、统一 API 响应	2022/8/25

1 后端编程规约

1.1 java 文件夹命名

（一） 全小写英文（尽量用一个英文单词，多个单词时全部小写拼接起来即可）。

（二）目录结构

com.guodi.子系统名.[controller | service | persistence].模块名。

可以参考 bds。

(3) 【参考】包目录规范说明

```
|- com
  |- guodi
    |- platpro                项目包名
      |- global               全局配置
        |- config             全局配置类
        |- constant           全局常量类
        |- enums              全局枚举类
        |- util               全局工具类
      |- module               模块
        |- platpro            模块一名称
          |- constant         常量类
          |- controller       控制层
          |- domain
```

```
            |- bo          业务对象
            |- dto         数据传输对象
            |- entity      实体类，一个 Entity 对应一张表
            |- vo          页面对应的对象，控制层与视图层传输交换
         |- enums          枚举类
         |- mapper         数据库层
         |- service        业务层
         |- util           工具类
      |- platpro2          模块二
         |- constant       常量类
         |- controller     控制层
         |- domain
            |- bo          业务对象
            |- dto         数据传输对象
            |- entity      实体类，一个 Entity 对应一张表
            |- vo          页面对应的对象，控制层与视图层传输交换
         |- enums          枚举类
         |- mapper         数据库层
         |- service        业务层
         |- util           工具类
```

1.2 java 编码规范

（一）控制器 Controller 类规范

A：类名遵守驼峰原则，请求路径为类名去掉 Controller；

如：

控制器的注解@Controller value 值统一为类名去掉 Controller，首字母大写。

如下：

```
@Controller（value="Login"）
public class LoginController{
 }
```

B：方法命名遵守驼峰原则，首字母小写；

C：常用方法约定：

分页方法：startPage()或 mybatis plus 默认分页方式；

打开列表页面

```
@RequestMapping("/toList")
public ModelAndView toList()
```

打开新增页面

```
@RequestMapping("/toAdd")
public ModelAndView toAdd
```

保存实体

```
@RequestMapping("/save")
@ResponseBody
public AjaxResult save
```

打开编辑页面

```
@RequestMapping("/toEdit")
public ModelAndView toEdit
```

保存更新

```
@RequestMapping("/update")
@ResponseBody
public AjaxResult update
```

获取列表数据

```
@RequestMapping("/listByEntity")
@ResponseBody
public AjaxResult listByEntity
```

D:统一 API 响应

```
public class R<T> implements Serializable {

    private static final long serialVersionUID = 1L;

    @ApiModelProperty(value = "状态码", required = true)
    private int code;
```

```
        @ApiModelProperty(value = "是否成功", required = true)
        private boolean success;
        @ApiModelProperty(value = "承载数据")
        private T data;
        @ApiModelProperty(value = "返回消息", required = true)
        private String msg;
    }
```

```
@PostMapping("/listByIds")
@ApiImplicitParams({
        @ApiImplicitParam(name = "ids", value = "主键,多个逗号分隔",required = true,
paramType = "query", dataType = "java.lang.String"),
})
@ApiOperationSupport(order = 2)
@ApiOperation(value = "批量根据 ID 查询对象", notes = "传入 ids")
public R<List<PlatPro>> listByIds(@ApiParam(value = "ids", required = true) @RequestParam
String ids) {
    if(StringUtil.isBlank(ids)){
        return R.fail(ResultCode.PARAM_MISS);
    }
    List<PlatPro> list = this.platProService.listByIds(Func.toLongList(ids));
    return R.data(list);
}
```

（二）Mapper 规范

*Mapper.xml 文件中表名、字段全部小写，命名规则如下：

namespace="表名(去掉前缀)Mapper"

例如：namespace="UserMapper"

增删改统一命名为：save、delete、update，涉及到具体业务操作方法也已该 save、delete、update 作为方法前缀，例如：deleteByOrgId、updateStatusById

查询命名：

返回单个记录：find*，例如：findById

返回多条记录：list*，例如：listByOrgId

（三）Java 通用规范

【强制】代码中的命名严禁使用拼音与英文混合的方式，更不允许直接使用中文的方式。

【强制】类名、方法名、参数名、成员变量、局部变量都统

一使用 CamelCase 风格，必须遵从驼峰形式。

【强制】常量命名全部大写，单词间用下划线隔开。

【强制】杜绝完全不规范的缩写，避免望文不知义。

【推荐】单个方法的总行数不超过 200 行。

2 前端编程规约

2.1 前端文件文件夹命名

全小写英文（尽量用一个英文单词,多个单词时全部小写,然后用中划线拼接起来）；

2.2 前端文件目录

WEB-INFO/jsp/子系统名/模块名/**.jsp;
static /子系统名/[css|js|images]/模块名/**.[css|js|png];

2.3 前端文件命名

jsp/js/css 文件命名为（首字母小写的数据库表实体名+List/Edit）。

3 注释规约

统一格式的注释（导入现成的模板），（IDEA 使用说明 1.4.docx 有）。

要求：

A:所有 java 类 public 方法必须加注释；

B:方法体内特殊的业务逻辑加注释。

C:所有业务文件逻辑处理，或特殊部分加注释。

4 单元测试

4.1 【强制】单元测试必须遵守 AIR 原则

说明：单元测试在线上运行时，感觉像空气（AIR）一样并不存在，但在测试质量

的保障上，却是非常关键的。好的单元测试宏观上来说，具有自动化、独立性、可重复执行的特点。

A：Automatic（自动化）

I：Independent（独立性）

R：Repeatable（可重复）

4.2 【强制】单元测试应该是全自动执行的，并且非交互式的

测试用例通常是被定期执行的，执行过程必须完全自动化才有意义。输出结果需要人工检查的测试不是一个好的单元测试。单元测试中不准使用 System.out 来进行人肉验证，必须使用 assert 来验证。

4.3 【强制】保持单元测试的独立性

为了保证单元测试稳定可靠且便于维护，单元测试用例之间决不能互相调用，也不能依赖执行的先后次序。

反例：method2 需要依赖 method1 的执行，将执行结果作为 method2 的输入。

4.4 【强制】单元测试是可以重复执行的，不能受到外界环境的影响

说明：单元测试通常会被放到持续集成中，每次有代码 check in 时单元测试都会被执行。如果单测对外部环境（网络、服务、中间件等）有依赖，容易导致持续集成机制的不可用。

正例： 为了不受外界环境影响，要求设计代码时就把 SUT 的依赖改成注入，在测试时用 spring 这样的 DI 框架注入一个本地（内存）实现或者 Mock 实现。

4.5 【强制】对于单元测试，要保证测试粒度足够小，有助于精确定位问题

单测粒度至多是类级别，一般是方法级别。

说明：只有测试粒度小才能在出错时尽快定位到出错位置。单测不负责检查跨类或者跨系统的交互逻辑，那是集成测试的领域。

4.6 【强制】核心业务、核心应用、核心模块的增量代码确保单元测试通过

说明：新增代码及时补充单元测试，如果新增代码影响了原有单元测试，请及时修正。

4.7 【强制】单元测试代码必须写在 test 目录

如下工程目录： src/test/java，不允许写在业务代码目录下。

说明：源码构建时会跳过此目录，而单元测试框架默认是扫描此目录。

4.8 【推荐】单元测试的基本目标

语句覆盖率达到 70%；核心模块的语句覆盖率和分支覆盖率都要达到 100%

说明：在工程规约的应用分层中提到的接口层，实现层，可重用度高的 Service，都应该进行单元测试。

5 安全规约

5.1 【强制】隶属于用户个人的页面或者功能必须进行权限控制校验

说明：防止没有做水平权限校验就可随意访问、修改、删除别人的数据，比如查看他人的私信内容、修改他人的信息。

5.2 【强制】参数绑定

用户输入的 SQL 参数严格使用参数绑定或者 METADATA 字段值限定，防止 SQL 注入

5.3 【强制】用户请求传入的任何参数必须做有效性验证

说明：忽略参数校验可能导致：

page size 过大导致内存溢出

恶意 order by 导致数据库慢查询

任意重定向

SQL 注入

反序列化注入

正则输入源串拒绝服务 ReDoS

说明：Java 代码用正则来验证客户端的输入，有些正则写法验证普通用户输入没有问题，

但是如果攻击人员使用的是特殊构造的字符串来验证，有可能导致死循环的结果。

6 数据库规约

1：数据库表命名

尽量英文，不使用复数名词；

格式如下：

子系统缩写+”_”+表名，如：

SYS_USER;WF_USER;

2：表字段

尽量英文，中间”_”分割；

如：

USER_NAME;USER_CODE;

表达是与否概念的字段，必须使用 is_xxx 的方式命名；

禁用保留字；

主键索引名为 pk_字段名；