研发中心代码管理规范
（试用）

广东国地规划科技股份有限公司

2021 年 8 月 17 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	郝明才	添加环境、技术框架、目录及版本管理规范	2021/8/17
1.6	郝明才	修改笔误	2021/9/24

1 管理规范

1.1 前端开发环境

IDE 工具：Visual Studio Code 1.59.0 以上；

编译打包环境：Nodejs v12.18.3、webpack 4.44.1

中间件：Vue 2.6.10、ElementUI2.15.1、国地 UI（扩展 ElementUI）

运行环境：IE11.0 及以上、firefox、谷歌浏览器

推荐应用服务器 Tomcat10 或 Nginx

1.2 后端开发环境

IDE 工具：Idea 2019.2 x64（推荐） 以上；

编译打包环境：Maven3.6.1（推荐）以上

底层依赖：gd-sys-common4.0.0.RELEASE 以上

中间件：Oracle12c 以上、PG 等

1.3 代码管理

新入职员工由管理员（彭辉、郝明才、吴灿）统一创建账号。

由开发人员向部门负责人申请，在 GIT 服务器创建项目。创建项目必须在所属部门文件夹下。

GIT 地址：http://10.0.1.250:9090/

完全的版本号定义，分三项：<主版本号>.<次版本号>.<修订版本号>，如 1.0.0。另，对于开发过程中的版本（某版本没有最终发布，

还处于调试阶段的不稳定版本，即快照版本），需要添加-beta，如1.0.0-beta。

版本号升级原则：

- ✧ 主版本号：功能模块有大的变动，比如增加了多个模块或者整体架构发生变化。
- ✧ 次版本号：和主版本相对而言，次版本号的升级对应的只是局部的变动。但该局部的变动造成了程序和以前版本不能兼容，或者对该程序以前的协作关系产生了破坏，或者是功能上有大的改进或增强。
- ✧ 修订版本号：局部的变动，主要是局部函数的功能改进，或者 bug 的修正，或者功能的扩充

每个工程都要有代码版本管理，使用 GIT 的版本管理模式，分开发库、发布库（标签库）：

Master：开发主干，所有开发工作都基于这个主干提交代码。

Branches：为了不影响主干的工作，对一些重大 bug 修改，拉出分支开发；当开发工作完成后，必须合并到主干上。

Tags：标签库的作用为项目的某个里程碑版本做备份用；每发布一个生产包，必须打一个标签版本；或者某个重要的 beta 版本也可以拷贝一个到标签分支；标签分支最主要的作用是保留每个发布版本的源码。

1.4 技术引进规范

为了保证各部门技术框架一致性、可维护性、兼容性、开放性，原则上前后端新项目必须基于当前公司底层开发框架进行搭建。当个

别特殊项目开发需要引进新的框架时，需要充分考虑引进新框架的各项风险，并以书面形式提交申请（列明原因和存在分险），经过研发中心领导审批方可进行。

同时，原则上基础运维等底层框架源码统一由基础研发部进行维护，各部门不得私自下载修改源码。特殊情况，以书面形式提交申请（列明原因和存在风险），经过研发中心领导审批方可进行。修改源码后引起的各种问题由申请源码的部门自行解决，基础研发部不再提供后续运维服务。

2 通用规范

2.1 规范意义

2.1.1 提高可读性

编码规范，帮助我们写出人类容易理解的代码，它为我们提供了最基本的模板，良好的编码风格，使代码具有一定的描述性，可以通过名字来获取一些需要 IDE 才能得到的提示，如可访问性、继承基类等。

2.1.2 统一全局，促进团队协作

开发软件是一个团队活动，而不是个人的英雄主义。编码规范，要求团队成员遵守这一统一的全局决策，这样成员之间可以轻松地阅读对方的代码，所有成员一种清晰而一致的风格进行编码。而且，开发人员也可以集中精力关注他们真正应该关注的问题——自身代码的业务逻辑，与需求的契合度等局部问题。

2.1.3 有助于知识传递，加快工作交接

风格的相似性，能让开发人员更迅速，更容易理解一些陌生的代码，更快速地理解别人的代码。因为，他和你的代码风格是一样的，你没有必要对他的一些个性化风格进行揣测。这样的好处是开发人员可以很快的接手项目组其他成员的工作，快速完成工作交接。

2.1.4 减少命名增生，降低维护成本

在没有规范的情况下，很容易为同一类型的实例起不同的名字。对于以后维护这些代码程序员来说会产生疑惑。

2.1.5 强调变量之间的关系，降低缺陷出现的机会

命名可以表示一定的逻辑关系，是开发人员在使用时保持警惕，从而一定程度上减少缺陷出现的机会。

2.1.6 提高程序员的个人能力

不可否认，每个程序员都应该养成良好的编码习惯，而编码规范无疑是教材之一。从一个程序员的代码本身能看出很多东西。所以，即便是为了自身发展，作为程序员也没有理由抵制这种规则的存在。你可能没有认识到，我们正默默地得益于编码规范。

2.2 统一

统一是指对于同一个概念，在程序中用同一种表示方法，比如对于供应商，既可以用 supplier，也可以用 provider，但是我们只能选定一个使用，至少在一个 Java 项目中保持统一。统一的作用是非常重要的，如果对同一概念有不同的表示方法，会使代码混乱难以理解。即使不能取得好的名称，但是只要统一，阅读起来也不会太困难，

因为阅读者只要理解一次。

2.3 达意

达意是指，标识符能准确的表达出它所代表的意义，比如：newSupplier, OrderPaymentGatewayService 等；而 supplier1, service2，idtts 等则不是好的命名方式。准确有两成含义，一是正确，二是丰富。如果给一个代表供应商的变量起名是 order，显然没有正确表达。同样的，supplier1, 远没有 targetSupplier 意义丰富。

2.4 简洁

简洁是指，在统一和达意的前提下，用尽量少的标识符。如果不能达意，宁愿不要简洁。比如：theOrderNameOfTheTargetSupplierWhichIsTransfered 太长，transferedTargetSupplierOrderName 则较好，但是 transTgtSplOrdNm 就不好了。省略元音的缩写方式不要使用，我们的英语往往还没有好到看得懂奇怪的缩写。

3 后端编程规范

3.1 java 编码规范

3.1.1 目录规范

（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）。

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

3.1.2 注释规范

注释是软件可读性的具体体现。程序注释量一般占程序编码量的20%，软件工程要求不少于20%。

统一格式的注释（导入现成的模板），（IDEA使用说明1.4.docx有）。

IDEA使用说明1.4.docx

3.1.3 请求路径规范

控制器Controller类规范，请求路径为类名去掉Controller，这样客户端调用时可以很清晰的知道URL那个部分是控制层的类，如：

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

3.1.4 接口返回值

接口返回统一用类com.guodi.core.tool.api.R进行封装。

3.1.5 分页实现规范

分页方法：PageHelper的startPage()或mybatis plus默认分页方式;推荐自己扩展的接口分页基于PageHelper实现，因为这样的接口具备兼容性，可以用来作为分页接口也可以查询返回所有数据的接口。

3.1.6 通用接口规范

控制层 Controller 命名为"表名 Controller"；

业务逻辑层："表名 Service"

实体："表名"，其中表名用英文，使用骆驼法则进行命名。

新增方法名：save；

修改方法名：update；

删除方法名：delete;

列表查询：list。

*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

3.1.7 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】Java 中，除了包名，静态常量等特殊情况，大部分情况下标识符使用骆驼法则，即单词之间不使用特殊符号分割，而是通过

首字母大写来分割。比如: supplierName, addNewContract，而不是 supplier_name, add_new_contract。

【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】Java 源文件还遵循以下规则：开头注释、包和引入语句、类和接口声明。

【推荐】单个方法的总行数不超过 200 行。

4 前端编程规范

4.1 前端文件目录

标准的前端目录框架：

```
├── build                          构建脚本目录
│   ├── build.js                   生产环境构建脚本
│   ├── check-versions.js          版本
│   ├── utils.js                   构建相关工具方法
│   ├── vue-loader.conf.js         vue 的相关 loader
│   ├── webpack.base.conf.js       wabpack 基础配置
│   ├── webpack.dev.conf.js        wabpack 开发环境配置
│   └── webpack.prod.conf.js       wabpack 生产环境配置
├── coder                          构建相关命令目录
│   ├── commonJs                   构建通用 js 目录
│   ├── templates                  构建模板，包括纯净版子系统等
│   ├── addSystem.js               新增纯净版子系统脚本
│   ├── removeSystem.js            删除子系统脚本
│   ├── setSystemConfig.js         设置子系统相关配置脚本
│   ├── indexNew.js                生成文件目录树脚本
│   └── utils.js                   相关工具方法
│   └── system.json                系统名称集合
├── config                         项目配置
│   ├── index.js                   项目配置文件
│   ├── webpack.dll.conf.js        生成 dll 文件脚本
├── src                            项目源码目录
    ├── projects                       子系统目录
        ├── index                      子系统，多个系统并行
```

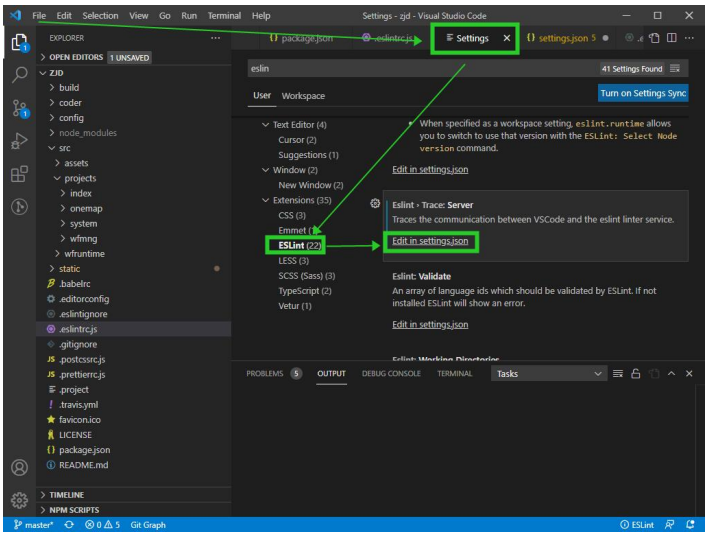
```
        ├── system                运维系统
    ├── modularize                多个子系统通用的模块
        ├── assets                相关静态文件目录
        ├── config                通用配置文件目录
        ├── directive             通用指令
        ├── helper                通用请求方式处理目录
        ├── iconfont-common       通用 iconfont 目录，具体规范请查看规范 iconfont 模块
        ├── icons                 通用 svg 目录
        ├── util                  通用工具函数
        ├── index.js              暴露通用功能
        ├── modularize.js         通用功能集合
├── static                        纯静态资源，不会被 wabpack 构建。
│   ├── common                    通用的静态文件
│   ├── index                     子系统的静态文件，文件夹名称和系统名称要一致，多个系统并行
├── favicon.ico                   页面图标
├── .babelrc                      babel 规则
├── .editorconfig                 编辑器配置
├── .eslintignore                 eslint 忽略规律
├── .eslintrc.js                  eslint 规则
├── .gitignore                    git 忽略规则
├── package.json                  npm 包配置文件，里面定义了项目的 npm 脚本，依赖包等信息
├── readmd.md                     项目描述文件
```

标准子应用目录

```
├── api                           接口目录
├── assets                        相关静态文件目录
├── components                    业务组件目录
├── config                        项目配置
├── mixin                         mixin 目录
├── router                        router 目录
├── store                         store 目录
├── styles                        styles 样式目录
├── utils                         子系统相关函数
├── views                         页面模块，这里注意，层级尽量不要超过三层
├── App.vue                       根组件
├── main.js                       入口 js 文件
├── index.html                    入口 html 页面
```

4.2 代码格式化

编码完后可以使用检查工具检查代码：ESLint，Eslint 配置参考如下：



settings.json

4.3 VUE 组件和工具函数

一致性规范

尽量使用 ElementUi 和国地组件库提供的组件，不重复开发功能重复的组件。

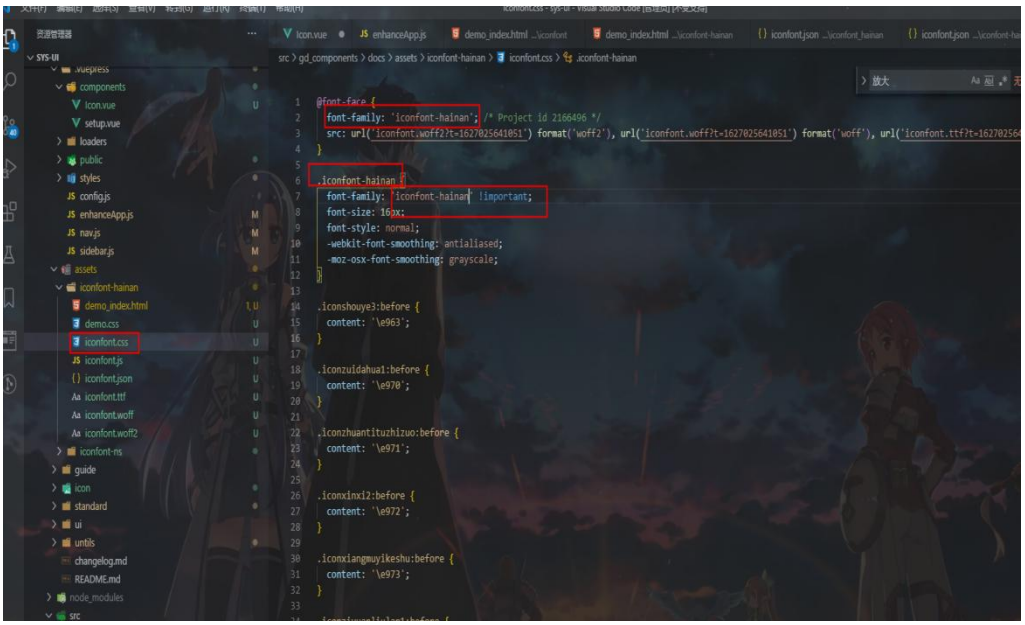
尽量使用内置的国地工具函数，不重复开发功能重复的函数。

命名规范

- 组件名应该始终是多个单词的，根组件 App 除外；
- 有意义的名词、简短、具有可读性；
- 命名遵循 PascalCase(单词首字母大写命名) 约定；
- 公用组件以 Abcd（公司名缩写简称）开头，如（AbcdDatePicker,AbcdTable）；
- 页面内部组件以组件模块名简写为开头，Item 为结尾，如（StaffBenchToChargeItem，StaffBenchAppNotArrItem）；
- 使用遵循 kebab-case(短横线分隔命名) 约定；
- 在页面中使用组件需要前后闭合，并以短线分隔；
- 导入及注册组件时，遵循 PascalCase(单词首字母大写命名) 约定）。

4.3.1 iconfont 使用规范

iconfont 的 Iconfont.css 文件添加项目编码标识。参阅国地前端开发组件库规范章节。如下图：



4.4 前端文件文件及文件夹命名

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）；

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

4.5 注释

代码注释在一个项目的后期维护中显的尤为重要，所以我们要为每一个被复用的组件编写组件使用说明，为组件中每一个方法编写方法说明。

以下情况，务必添加注释：

- 公共组件使用说明；
- 各组件中重要函数或者类说明；
- 复杂的业务逻辑处理说明；
- 特殊情况的代码处理说明,对于代码中特殊用途的变量、函数中使用的变量、使用了某种算法或思路等需要进行注释描述；
- 注释块必须以/（至少两个星号）开头/，添加作者和时间；
- 单行注释使用//；
- 多重 if 判断语句自定义类。

4.6 变量命名

变量名应为英文字母，数字，下划线“_”，美元符号“$”组成，变量名第一个字符不能是数字。

统一使用小写字母，多个单词时使用驼峰式命名，即后面的单词首字母大写

如：userName = “xxx”。

4.7 常量名

应全为大写，单词与单词之间使用下划线“_”连接，如：PAGE_SIZE=“20”。

4.8 对象属性命名

在声明 prop 的时候，其命名应该始终使用 camelCase， 而在模板中应该始终使用 kebab-case。

```
// 推荐
<script>
  props: {
    greetingText: String;
  }
</script>
<welcome-message greeting-text="hi"></welcome-message>
//不推荐
<script>
  props: {
    'greeting-text': String
  }
</script>
<welcome-message greetingText="hi"></welcome-message>
```

4.9 行数

一行语句的字符数不能太长，一般不超过 80 个字符，多出的应该换行显示。

5 单元测试

5.1 【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）

5.2 【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。

5.3 【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。

5.4 【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。

5.5 【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。

5.6 【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。

5.7 【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。

5.8 【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。

6 安全规范

6.1 【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。

6.2 【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

6.3 【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。

7 数据库规范

- 数据库设计

设计关系数据库时，遵从不同的规范要求，设计出合理的关系型数据库。越高的范式数据库冗余越小。在满足性能要求的情况下，减少数据冗余，节省存储空间，保持数据的一致性。

- 数据库表命名

尽量英文，不使用复数名词，一般表名长度不超过 30 个字符；

格式如下：

子系统缩写+”_”+表名，如：

SYS_USER;WF_USER;

- 表字段

尽量英文，中间”_”分割，长度不超过 30 个字符；；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；

字段长度够用即可，不是越大越好。

- 外键

存在关联性的字段尽量建立外键，保证数据的完整性，外键约束以 FK_开头。

- 视图命名

  视图命名统一以 VIEW_开头，提高表的辨识度。

- 索引

  数据量较大的表，尽量利用建立索引，利用 SQL 优化技术，提升访问性能。