****

**编 码 规 范**

题 目： “无微不至”的借阅伴侣

学 院： 计算机科学与工程学院

专 业： 计算机科学与技术

班 级： 2018级卓越工程师班

团队名称： 奋斗的小菜鸟

团队成员： 魏娜娜 马艳 牛建疆 王会娟

目录

[第一章 编码规范 2](#_Toc22991)

[1.1 好代码的原则 2](#_Toc7019)

[1.2 项目命名规范 2](#_Toc13412)

[1.3 方法参数规范 2](#_Toc12738)

[1.4 注释规范 2](#_Toc31673)

[第二章 项目规范 4](#_Toc18126)

[2.1 代码目录结构 4](#_Toc23825)

[2.2 common目录规范 4](#_Toc7282)

[2.3 config 目录规范 5](#_Toc24005)

[2.4 module 目录规范 5](#_Toc18842)

[2.5 domain 包中的 javabean 命名规范 6](#_Toc11294)

[第三章 Java编程规范 7](#_Toc12854)

[3.1 介绍 7](#_Toc30707)

[3.2 代码格式 9](#_Toc15324)

[3.3 Java注释 9](#_Toc6450)

# 第一章 编码规范

## 1.1好代码的原则

（1）为通过所有测试(Passes its tests)：强调的是外部需求，这是代码实现最重要的

（2）尽可能消除重复 (Minimizes duplication)：代码的模块架构设计，保证代码的正交性，保证代码更容易修改

（3）尽可能清晰表达 (Maximizes clarity)：代码的可阅读性，保证代码是容易阅读的

（4）更少代码元素 (Has fewer elements)：保证代码是简洁的，在简洁和表达力之间，我们更看重表达力

## 1.2项目命名规范

全部采用小写方式， 以中划线分隔。

正例：mall-management-system / order-service-client / user-api

反例：mall\_management-system / mallManagementSystem /

## 1.3方法参数规范

无论是controller，service，manager，亦或是其他的代码，每个方法最多3个参数，如果超出3个参数的话，要封装成javabean对象。

方便他人调用，降低出错几率。尤其是当参数是同一种类型，仅仅依靠顺序区分，稍有不慎便是灾难性后果，而且排查起来也极其恶心。

保持代码整洁、清晰度。当一个个方法里充斥着一堆堆参数的时候，再坚强的人，也会身心疲惫。

## 1.4注释规范

注释并不是越多越好，当注释过多，维护代码的同时，还需要维护注释，不仅变成了一种负担，也与我们当初添加注释的初衷背道而驰。

首先：应通过清晰的逻辑架构，好的变量命名来提高代码可读性；需要的时候，才辅以注释说明。注释是为了帮助阅读者快速读懂代码，所以要从读者的角度出发，按需注释。注释内容要简洁、明了、无二义性，信息全面且不冗余。

其次：无论是修改、复制代码时，都要仔细核对注释内容是否正确。只改代码，不改注释是一种不文明行为，破坏了代码与注释的一致性，会让阅读者迷惑、费解，甚至误解。

方法要尽量通过方法名自解释，不要写无用、信息冗余的方法头，不要写空有格式的方法头注释。

方法头注释内容可选，但不限于：功能说明、返回值，用法、算法实现等等。尤其是对外的方法接口声明，其注释，应当将重要、有用的信息表达清楚。

# 第二章 项目规范

## 2.1代码目录结构

统一的目录结构是所有项目的基础：

src 源码目录

|-- common 各个项目的通用类库

|-- config 项目的配置信息

|-- constant 全局公共常量

|-- handler 全局处理器

|-- interceptor 全局连接器

|-- listener 全局监听器

|-- module 各个业务

|-- |--- employee 员工模块

|-- |--- role 角色模块

|-- |--- login 登录模块

|-- third 三方服务，比如redis, oss，微信sdk等等

|-- util 全局工具类

|-- Application.java 启动类

## 2.2 common目录规范

common 目录用于存放各个项目通用的项目，但是又可以依照项目进行特定的修改。

src 源码目录

|-- common 各个项目的通用类库

|-- |--- anno 通用注解，比如权限，登录等等

|-- |--- constant 通用常量，比如 ResponseCodeConst

|-- |--- domain 全局的 javabean，比如 BaseEntity,PageParamDTO 等

|-- |--- exception 全局异常，如 BusinessException

|-- |--- json json 类库，如 LongJsonDeserializer，LongJsonSerializer

|-- |--- swagger swagger 文档

## 2.3 config 目录规范

config 目录用于存放各个项目通用的项目，但是又可以依照项目进行特定的修改。

src 源码目录

|-- config 项目的所有配置信息

|-- |--- MvcConfig mvc的相关配置，如interceptor,filter等

|-- |--- DataSourceConfig 数据库连接池的配置

|-- |--- MybatisConfig mybatis的配置

|-- |--- .... 其他

## 2.4 module 目录规范

module 目录里写项目的各个业务，每个业务一个独立的顶级文件夹，在文件里进行 mvc 的相关划分。 其中，domain 包里存放 entity, dto, vo，bo 等 javabean 对象

src

|-- module 所有业务模块

|-- |-- role 角色模块

|-- |-- |--RoleController.java controller

|-- |-- |--RoleConst.java role相关的常量

|-- |-- |--RoleService.java service

|-- |-- |--RoleDao.java dao

|-- |-- |--domain domain

|-- |-- |-- |-- RoleEntity.java 表对应实体

|-- |-- |-- |-- RoleDTO.java dto对象

|-- |-- |-- |-- RoleVO.java 返回对象

|-- |-- employee 员工模块

|-- |-- login 登录模块

|-- |-- email 邮件模块

|-- |-- .... 其他

## 2.5 domain 包中的 javabean 命名规范

1）javabean的整体要求：

①不得有任何的业务逻辑或者计算

②基本数据类型必须使用包装类型（Integer, Double、Boolean 等）

③不允许有任何的默认值

④每个属性必须添加注释，并且必须使用多行注释。

⑤必须使用lombok简化getter/setter方法

⑥建议对象使用lombok的@Builder ，@NoArgsConstructor，同时使用这两个注解，简化对象构造方法以及set方法。

2）数据对象；XxxxEntity，要求：

①以Entity为结尾

②Xxxx 与数据库表名保持一致

③类中字段要与数据库字段保持一致，不能缺失或者多余

④类中的每个字段添加注释，并与数据库注释保持一致

⑤不允许有组合

⑥项目内的日期类型必须统一，建议使用java.util.Date，java.sql.Timestamp，java.time.LocalDateTime其中之一。

3）传输对象；XxxxDTO，要求：

①不可以继承自Entity

②DTO可以继承、组合其他DTO，VO，BO等对象

③DTO只能用于前端、RPC 的请求参数

4）视图对象；XxxxVO，要求：

①不可继承自Entity

②VO可以继承、组合其他DTO，VO，BO等对象

③VO只能用于返回前端、rpc 的业务数据封装对象

5）业务对象BO，要求：

①不可以继承自Entity

②BO对象只能用于service，manager，dao层，不得用于controller层

# 第三章 Java编程规范

## 3.1 介绍

（1）标识符命名规范——标识符的命名力求做到统一、达意和简洁。

统一：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。统一是作为重要的，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。

达意：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。

简洁：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。

1. 骆驼法则：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。

英文VS拼音：尽量使用通俗易懂的英文单词，如果不会可以向队友求助，实在不行则使用汉语拼音，避免拼音与英文混用。比如表示归档，用archive比较好，用pigeonhole则不好，用guiDang尚可接受。

（3）包名：使用小写字母如com.xxx.settlment，不要com.xxx.Settlement；

单词间不要用字符隔开，比如com.xxx.settlment.jsfutil，而不要com.xxx.settlement.jsf\_util。

（4）类名：首字母大写，比如SupplierService，PaymentOrderAction；不要supplierService，paymentOrderAction。往往用不同的后缀表达额外的意思。

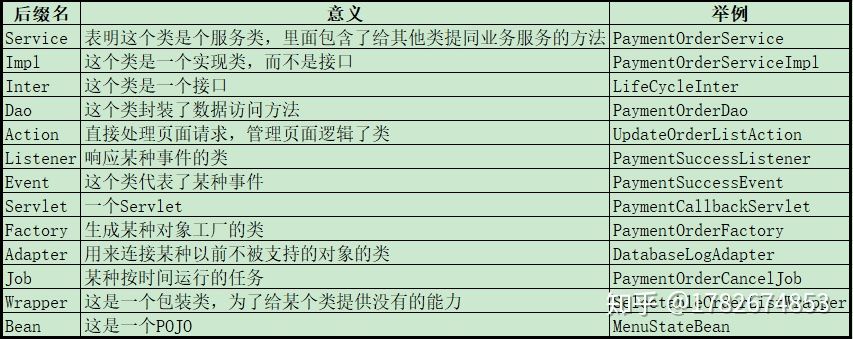


图3-1 后缀

（5）方法名：首字母小写，比如addOrder不要AddOrder；动词在前，比如addOrder，不要orderAdd；动词前缀往往表达特定的含义。

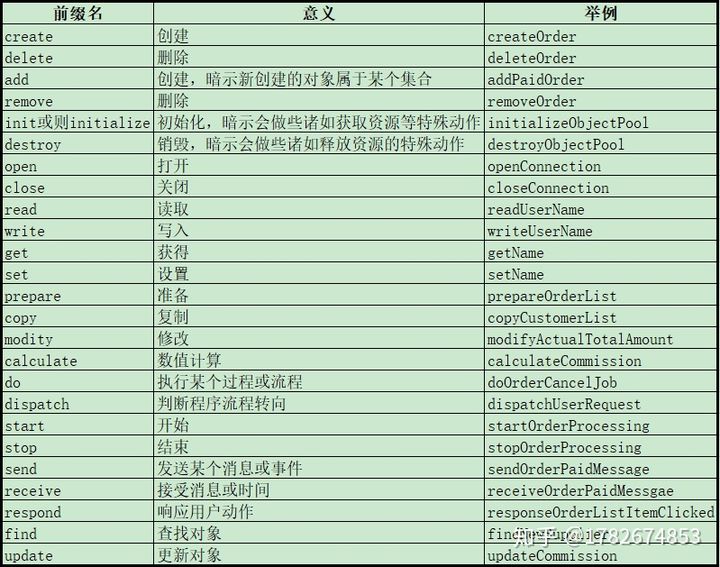


图3-2 前缀

find方法在业务层尽量表达业务含义，比如 findUnsettledOrders，查询未结算订单，而不要findOrdersByStatus。数据访问层，find，update等方法可以表达要执行的sql，比如：findByStatusAndSupplierIdOrderByName。

静态常量：全大写用下划线分割，比如：

public static final String ORDER\_PAID\_EVENT = "ORDER\_PAID\_EVENT";

枚举：全大写，用下划线分割，比如：

public enum Events{

ORDER\_PAID,

ORDER\_CREATED}

局部变量名：参数和局部变量名首字母小写，骆驼法则。尽量不要和域冲突，尽量表达这个变量在方法中的意义。

## 3.2 代码格式

**代码格式**——用空格字符缩进源代码，不要用Tab，每个缩进4个空格。

**源文件编码**：源文件使用utf-8编码，结尾用unix n分格。

**行宽**：不要超过120。

**包的导入**：删除不用的导入，尽量不要使用整个包的导入。

**域格式**：每行只能声明一个域，域的声明用空行隔开。

**代码块格式：**

缩进风格：大括号的开始在代码块开始的行尾，闭合在和代码块同一缩进的行首。

空格的使用：表示分割时用一个空格，不能这样，比如：

if ( a > b ) {//do something here};

二元、三元运算符两边用一个空格隔开，比如：

a + b = c;b - d = e;return a == b ? 1 : 0;

逗号语句后如不换行，紧跟一个空格，比如：

call(a, b, c);

空行的使用：空行可以表达代码在语义上的分割，注释的作用范围，等等。将类似操作，或一组操作放在一起不用空行隔开，而用空行隔开不同组的代码。

空行使注释的作用域很明显，连续两行的空行代表更大的语义分割；方法之间用空行

分割；域之间用空行分割；超过十行的代码如果还不用空行分割，就会增加阅读困难

## 3.3 Java注释

注释宜少而精，不宜多而滥，更不能误导；命名达意，结构清晰，类和方法等责任明确，往往不需要，或者只需要很少注释，就可以让人读懂，相反，代码混乱，再多的注释都不能弥补，所以，应当先在代码本身下功夫；不能正确表达代码意义的注释，只会损害代码的可读性；过于详细的注释，对显而易见的代码添加的注释，啰嗦的注释，还不如不写，注释要和代码同步，过多的注释会成为开发的负担；注释不是用来管理代码版本的，如果有代码不要了，直接删除，svn会有记录的，不要注释掉，否则以后没人知道那段注释掉的代码该不该删除。

**Java Doc**：表明类、域和方法等的意义和用法等的注释，要以javadoc的方式来写。Java Doc是个类的使用者来看的，主要介绍是什么，怎么用等信息。凡是类的使用者需要知道，都要用Java Doc 来写。非Java Doc的注释，往往是个代码的维护者看的，着重告述读者为什么这样写，如何修改，注意什么问题等。

**块级别注释**：单行时用//，多行时用/\* .. \*/；较短的代码块用空行表示注释作用域；较长的代码块要用/\*------ start: ------\*/和/\*-------- end: -------\*/包围，可以考虑使用大括号来表示注释范围。

o