**团队项目编码规范**

**目录**

**[1. 目的 2](#_Toc20503)**

**[2. 概述 3](#_Toc22653)**

**[3. 代码规范 4](#_Toc21570)**

**[3.1 有关命名 4](#_Toc6786)**

**[3.1.1 包命名 4](#_Toc32648)**

**[3.1.2 类命名 5](#_Toc15318)**

**[3.1.3 方法命名 6](#_Toc19641)**

**[3.1.4 变量命名 6](#_Toc28890)**

**[3.1.5 常量命名 7](#_Toc18312)**

**[3.1.6 源文件命名 7](#_Toc25602)**

**[3.1.7 其他命名 7](#_Toc12280)**

**[3.2 有关注释 8](#_Toc7962)**

**[3.2.1 程序文件头注释 8](#_Toc29463)**

**[3.2.2 类注释 9](#_Toc930)**

**[3.2.3属性注释 9](#_Toc6762)**

**[3.2.4方法注释 9](#_Toc29450)**

**[3.2.5 构造方法注释 10](#_Toc28593)**

**[3.2.6 方法内部注释 10](#_Toc3545)**

**[4. 格式规范 10](#_Toc16376)**

**[4.1 缩进 10](#_Toc26885)**

**[4.2 换行 11](#_Toc16401)**

**[4.3 对齐 11](#_Toc21249)**

**[5. 成员行为规范 11](#_Toc26085)**

**[6. 补充 11](#_Toc20048)**

**1. 目的**

为了使软件开发过程顺畅，保证软件质量，于是有了这份开发规范文档

为什么要制定编码规范：

（1）能代码更加易于维护，程序并不是一次性的产品，它需要扩展和修改还有维护的。但是进行这次操作的人并一定就是自己，所以其他后续的成员也要将规范编码进行到底。

（2）可以提高代码质量，每个人编程都不是一次性完成的，是需要不断的测试与调试，以各种状态来修改自己的代码，而将代码规范化，就能对程序本身有更加清晰的结构思路，从而减少调试而成为快捷高效的代码。.

（3）是将每个人本身个性化溶于团队化的过程，当自己熟练运用编码规范了，就等于在以后的职场的道路上更加宽广。编码规范是一种习惯，-开始习惯不养好，永远写不出工程型代码。

**2. 概述**

（1）项目以功能模块来划分分工，模块功能要单一，新增功能时，尽量减少对已有模块的改动，在新增模块中实现功能，功能调整或修改bug时，影响的模块数量要尽可能少；功能调整或修改bug时，受影响的模块要尽可能的小（指客户端应用程序开发），模块间要松耦合，模块接口尽可能少、并且简单。

（2）数据库不能随意修改，本地数据库须统一；不管是数据库，还是代码变量，尽量使用简短，常见，准确，专业的单词，方便他人理解。数据库字段采用下划线命名法，单词之间必须用下划线隔开，简短单词也要遵循此原则，比如user\_id不能写成userid。代码中的变量名，应当和数据库中的字段名完全一致，由下划线转驼峰命名，保持客户端-数据库-服务端单词一致，方便代码自动生成和解析，前后端工程师在协作过程中，也不用在字段名这种小事情上浪费时间

（3）有标准的Java代码风格，有良好习惯；英文不好的，在命名时应当多花时间百度，尤其不能自己制造不存在的单词，单词禁止缩写，除非是单词特别长且有公认缩写形式的（比如connection->conn，department->dept），自己起的缩写，很可能让别人无法理解

（4）时刻考虑代码的可复用性，代码的可读性方面可以在规范注解等方面养成良好习惯，并且养成精益求精的重构好习惯，让自己的代码尽量简化，效率最大化。

（5）在现代化的开发当中，实体类是由数据库框架或插件一键生成的，数据库查询代码也是框架一键生成或封装好的，前端向后端发出的请求参数可以由框架一键转换为实体类，后端返回给前端的数据是由框架一键转换为ResponseBody，前端接收数据也是由框架一键将字符串转为实体类，APP将后台数据存储到本地数据库也是由框架一键完成，而这些全自动化的前提就是命名一致。如果命名不一致，那么每一个步骤，都需要手动去设置，前后端在协作过程中就有许多地方需要相互确认。可见，数据库和代码命名规范，不仅是为了看得舒服，已经直接影响到代码复杂度，前后端协作的复杂度，代码转手的复杂度。

**3. 代码规范**

**3.1 有关命名**

总体原则上，变量名与方法命名时应该遵循良好的命名习惯，尽量使得命名能够体现变量或者方法本身的含义。

**3.1.1 包命名**

* 包命名采用全小写命名；
* 通过功能来命名；
* 类命名采用Pascal命名法；
* 大写字母开头，各个单词首字母大写；
* Activity、Fragment、Service等命名必须以Activity、Fragment、Service相应后缀结束；
* Model命名变量采用public修饰；
* 方法命名采用Camel命名法，小写字母开头，各个单词首字母大写；
* 属性的getter和setter方法尽量使用自动生成，并放在程序的后面；
* Boolean类型的属性的get方法应形如isProperty()；
* 使用getXXX()形式和setXXX()形式进行获取数据和设置数据的基本方法命名；
* 采用Camel命名法，小写字母开头，各个单词首字母大写；
* 特有大写缩写词汇保持大写如：SQL
* 尽量采用全命名方式，名字确实过长时，可适当采用缩减英文元音字母来缩短长度；
* 假如缩短后名字重复，可以保留其中一个的部分元音字母；
* 类内部变量命名时，需要带上m开头，表示此为类内部变量成员；
* 控件在命名时保持与layout中id一致命名原则；
* 采用全大写命名法，有意义的单词之间使用"\_"进行分割；
* 采用全小写命名法；
* 所有的字母均小写，单词之间以下划线'\_'分隔；
* 按照分类+功能+所属页面等命名，如：layout\_main.xml，item\_bluetooth\_device.xml,view\_switch\_button.xml,dialog\_add.xml等；
* 图标文件默认使用ic\_开头，例如ic\_logo；
* 控件样式文件等以控件缩写开头，描述控件目标，例如btn\_ background \_light.9.png；
* 原则上采用全小写命名法，并采用下划线分割法；
* 控件id在冲突的情况下需要加入页面限定，例如：tv\_username\_activity\_main等；

**3.1.2 类命名**

类名使用大驼峰命名形式，类命通常时名词或名词短语，接口名除了用名词和名词短语以外，还可以使用形容词或形容词短语，如Cloneable，Callable等，表示实现该接口的类有某种功能或能力。对于测试类则以它要测试的类开头，以Test结尾，如HashMapTest。

对于一些特殊特有名词缩写也可以使用全大写命名，比如XMLHttpRequest，不过笔者认为缩写三个字母以内都大写，超过三个字母则按照要给单词算。这个没有标准，如阿里巴巴中fastjson用JSONObject作为类命，而google则使用JsonObjectRequest命名，对于这种特殊的缩写，原则是统一就好。

| **属性** | **约束** | **例** |
| --- | --- | --- |
| 抽象类 | Abstract 或者 Base 开头 | BaseUserService |
| 枚举类 | Enum 作为后缀 | GenderEnum |
| 工具类 | Utils作为后缀 | StringUtils |
| 异常类 | Exception结尾 | RuntimeException |
| 接口实现类 | 接口名+ Impl | UserServiceImpl |
| 领域模型相关 | /DO/DTO/VO/DAO | 正例：UserDAO 反例： UserDo， UserDao |
| 设计模式相关类 | Builder，Factory等 | 当使用到设计模式时，需要使用对应的设计模式作为后缀，如ThreadFactory |
| 处理特定功能的 | Handler，Predicate, Validator | 表示处理器，校验器，断言，这些类工厂还有配套的方法名如handle，predicate，validate |
| 测试类 | Test结尾 | UserServiceTest， 表示用来测试UserService类的 |
| MVC分层 | Controller，Service，ServiceImpl，DAO后缀 | UserManageController，UserManageDAO |

**3.1.3 方法命名**

**方法命名采用小驼峰的形式**，首字小写，往后的每个单词首字母都要大写。 和类名不同的是，方法命名一般为**动词或动词短语**，与参数或参数名共同组成动宾短语，即动词 + 名词。一个好的函数名一般能通过名字直接获知该函数实现什么样的功能。

**3.1.4 变量命名**

**3.1.4.1 一般性原则**

**3.1.4.2 控制命名原则**

变量是指在程序运行中可以改变其值的量，包括成员变量和局部变量。变量名由多单词组成时，第一个单词的首字母小写，其后单词的首字母大写，俗称骆驼式命名法（也称驼峰命名法），如 computedValues，index、变量命名时，尽量简短且能清楚的表达变量的作用，命名体现具体的业务含义即可。

变量名不应以下划线或美元符号开头，尽管这在语法上是允许的。变量名应简短且富于描述。变量名的选用应该易于记忆，即，能够指出其用途。尽量避免单个字符的变量名，除非是一次性的临时变量。pojo中的布尔变量，都不要加is(数据库中的布尔字段全都要加 is\_ 前缀)。

**3.1.5 常量命名**

常量命名CONSTANT\_CASE，一般采用全部大写（作为方法参数时除外），单词间用下划线分割。

常量是在作用域内保持不变的值，一般使用final进行修饰。一般分为三种，全局常量（public static final修饰），类内常量（private static final 修饰）以及局部常量（方法内，或者参数中的常量），局部常量比较特殊，通常采用小驼峰命名即可。

常量一般都有自己的业务含义,不要害怕长度过长而进行省略或者缩写。如，用户消息缓存过期时间的表示，那种方式更佳清晰，交给你来评判。

**3.1.6 源文件命名**

（1）一个源文件只能有一个public class中，不能在已经已经有public class的源文件中再次定义public class。

（2）编译源文件，报错。

f:\JavaStudy>javac Welcome.java

Welcome.java:20: 类 Welcome2 是公共的，应在名为 Welcome2.java 的文件中声明public class Welcome2

（3）一个源文件可以定义多个类，但是，只能有一个是public class。

（4）源文件的名字必须和public class的名字一样，大写敏感。

**3.1.7 其他命名**

备注：各个控件以首字母缩写为基本原则，个别控件可根据实际情况使用全命名方式，但必须小写开头，命名前缀一览；

|  |  |  |
| --- | --- | --- |
| 控件名称 | 前缀 | 示例 |
| TextView | tv\_ | tv\_add |
| EditText | ev\_ | ev\_add |
| View | v\_ | v\_split |
| ProgressBar | pb\_ | pb\_loading |
| ... | ... | ... |

* 其它情况下需要提出问题进行商讨解决；

**3.2 有关注释**

团队成员都应该形成良好的写注释的习惯，方便以后阅读，以及为了后期生成可读性良好的Java Doc。

**3.2.1 程序文件头注释**

应该包含如下：   
\* 文件描述（Description）：描述此类的作用；  
\* 作者（Author）：创建者或者修改者名；  
\* 版本（Version）：创建或者修复时的编号，需要自行在bug管理系统中创建bug号，使用bug号进行命名（若无bug管理工具的临时办法：如无bug号，从1开始，修改时依次增加）  
\* 日期（Date）：创建或者修改时的日期，使用“-”进行年月日分割；

\* 记录（Record）：创建或者修改的工作内容描述；

提前设置好文件的模板Template， 模板以下：

/\*\*

 \* Description:

 \* Author Version Date Record

 \*KevinLee 1 2017-11-7 版本创建

\*/

**3.2.2** **类注释**

在每个类前面必须加上类注释，注释模板如下：

/\*\*

\* Copyright (C), 2006-2010, ChengDu Lovo info. Co., Ltd.

\* FileName: Test.java

\* 类的详细说明

\*

\* @author 类创建者姓名  
    \* @Date    创建日期

\* @version 1.00

\*/

**3.2.3属性注释**

在每个属性前面必须加上属性注释，注释模板如下：

/\*\* 提示信息 \*/

private String strMsg = null;

**3.2.4方法注释**

在每个方法前面必须加上方法注释，注释模板如下：

/\*\*

\* 类方法的详细使用说明

\*

\* @param 参数1 参数1的使用说明

\* @return 返回结果的说明

\* @throws 异常类型.错误代码 注明从此类方法中抛出异常的说明

\*/

**3.2.5 构造方法注释**

在每个构造方法前面必须加上注释，注释模板如下：

/\*\*

\* 构造方法的详细使用说明

\*

\* @param 参数1 参数1的使用说明

\* @throws 异常类型.错误代码 注明从此类方法中抛出异常的说明

\*/

**3.2.6 方法内部注释**

在方法内部使用单行或者多行注释，该注释根据实际情况添加。

如：//背景颜色

       Color bgColor = Color.RED

**4. 格式规范**

**4.1 缩进**

应注意使用format来格式化代码，使用Tab键来缩进代码，相当于4个空格。折行使用120字符宽度；

**4.2 换行**

* {}花括号应该另起一行，左花括号与方法名、类名在同一行。(除了数组初始化时的花括号)；
* if、while等语句，假如体内只有一句代码也不要省略{}，为了方便以后的增删；
* 字符串过长考虑拆分成多行；
* {}括号等应该对齐；
* 类和方法的块注释必须紧贴类和方法；
* 单独起行的//注释必须对齐被注释语句；

**4.3 对齐**

找到"Source",点击在弹出的下拉框内，找到"Format",然后点击然后对比一下，就可以看到代码自动对齐了；还有一种方法是直接使用快捷键“Ctrl+shift+f”，就ok了，返回代码区，然后对比一下，就可以看到代码自动对齐了

**5. 成员行为规范**

希望各位成员遵守这份开发规范文档，养成良好的开发习惯；

**6. 补充**

如果有问题，请及时反馈。