比较优雅地编码

公司文档 代码规范

笔者认为做到比较优雅地编码，需遵从如下约束，排名分先后：

- 良好的命名
- 清晰的结构
- 不十分差劲的算法

下面逐一说明：

良好的命名

> 名不正，则言不顺，言不顺，则事不成
> —— 孔子

孟子曰：“孔子说的对”。

命名很重要，随便一本逻辑学教材（如果读者有兴趣，此处推荐《逻辑学导论》）里都会有长篇大论来讨论命名的问题，我国古代在人才辈出的百家争鸣时期曾经出现过一个学派叫“名家”，专门讨论命名的问题，比如著名的“白马非马”、“离坚白”等，有空读读还是挺有趣的，比玩王者荣耀还有趣。笔者认为我国到目前为止出现了三个思想大解放时期：一，春秋战国时期的百家争鸣，可惜被秦皇的封建大一统所终结；二，明国时期，西方思想初涌，国人似惊雷轰顶，如梦方醒；三，当下，技术革命使得中央思想控制力度急剧下降，使得底层人民有了选择被谁控制思想的自由（请参考如今基于互联网的粉丝文化，圈子文化）。所以大家一定要好好写代码，才不负少年头呀。

回归主题，命名应该本着不拍长就怕不清楚的原则，尽量把一个类、方法、变量的含义交代清楚，实际上笔者比较喜欢OC的方法（消息）命名规则，比如：

```
(void) buy:(NSString*)something from:(NSString*)store at:(NSDate*)time;
```

长，但是很清楚。下面详述命名：

接口的命名

接口的命名一般都是前缀 **I** 加上**名词**或**形容词**，实际上.Net Framewrok类库就是这样做的，比如：

- System.Collections.Generic.ICollection（名词）
- System.Collections.Generic.IEnumerable（形容词）

接口往往用来分装一个或一组行为，实现接口的类意味着它具有了一个或一组行为的能力，所有常用形容词。当实在找不到一个恰当的形容词时，名词也可以。

抽象类

抽象类通常不需要加前缀或后缀，也有部分大牛喜欢给抽象类视情况加**Base**后缀。如下的命名笔者觉得就不错：

- Shape（抽象类）
- Ellipse（具体类）
- Rectangle（具体类）

具体类

尽量使用名词，不排除个别情况下把一个行为封装为类时使用动词。

方法

使用动词或动词性短语。

以上。笔者觉得其它的字段啊、属性啊、事件啊出现命名问题的比较少，不再讨论。比如有的人喜欢字段用下划线开头，有的人喜欢用**m**_开头，都无关紧要，求同存异，只要使用的词语

恰当，并不会带来太多麻烦。反倒是如果规定一个巨细无比的规范才有问题，一是耗费编撰者的时间，二是阅读者也记不住，三是阅读者记住了也可能因为和自己的习惯不同而心存抵触。

此外需强调一点就是少用拼音多用英文，少用缩写多用全称，尽量不用拼音缩写，甚至是方言拼音缩写，还有英语加拼音，甚至是英语加拼音缩写……。但也不能排除个别如“计划生育”这样的不好翻译的词语可以用拼音，需注释明确或维护词汇表。另外，缩写也不是不可以用，一些大家约定成俗的缩写比如“Id”，使用之并不会带来误解，还有利于代码的清晰度。

有时候由于英语水平的问题，毕竟我们的母语不是英语，难免会出现不知道用什么单词的情况。此时，可以试试codelf，感谢据说是网易工程师unbug开发的此工具。这个应该读作[koʊdɪf]吧，但是倒数第二个字母不是I,是L。

清晰的结构

基本要求

- if、while等关键字包裹的表达式，即使只有一句，也尽量使用大括号括起来
- 尽量避免使用直立人时代的goto语句
- 尽量避免过长的方法，把大方法拆分为数个小方法，方法的职能尽量单一
- 尽量避免重复代码，将其转移到公共工具中
- 尽量避免过大的类，拆分为数个类，各司其职
- 适当的注释，注释太多，说明代码本身的表达力可能还有待加强。有些实在难缠的逻辑，可以写详细注释，如果不写注释，离职时请注意不要泄露给擦屁股人自己的地址

较高要求

熟悉以下常用面向对象的设计原则：

- 开放-封闭
- 单一职责
- 里氏替换
- 依赖倒置
- 接口隔离

再高要求

熟悉常用设计模式:

- 工厂方法
- 抽象工厂
- 建造者模式
- 单例模式
- 设配器模式
- 外观模式
- 观察者模式
- 命令模式
- 代理模式
- 策略模式

面向对象语言的三大特征人人都知道，但是真的用好却是不容易，三大特征仅仅是面向对象设计的基础，在此基础上建立起了面向对象的高楼大厦。笔者工作多年，常感叹虚度年华，至今尚未精通面向对象设计之十之五六。笔者想极力推荐一本书：《Head First 设计模式》，此书乃Head First系列成名之作。此外，这个网站不错，言简意赅，案例经典：www.oodesign.com。此外，《Java编程思想》开头部分关于面向对象设计的讲解也很精彩。

使用成熟的设计模式有两个好处：

- 这些解决方案已经经受过了考验，用了即使无功，但也不会犯错
- 设计模式的名称本来就一种在程序员间流传的概念，基于共同的对概念的认识能够降低沟通成本

当然也有坏处，代码的扩展性虽然是好了，但是代码的复杂性也高了，这就要求我们最好熟悉常用的设计模式，这样就得到了它的好处，规避了它的坏处。

必须要注意的是：不要乱用设计模式，只有需求产生了变化，或预见了需求将要发生变化才使用这些模式来封装发生的或可能发生的变化。有时候再重构代码时也会用到，比如外观模式和设配器模式等。

更高要求

> 能力越大，责任越大
> ——蜘蛛侠

大项目肯定不是一个人能完成的，人多了容易发生混乱，此时需要团队的领袖勇敢地承担起义不容辞的责任，包括但不限于：

- 定期维护代码框架、分层结构。写的时间长了难免乱，领袖人物要定期排除，及时消除臭味
- 引导小弟们做代码审查。审查的意义不在于监督，而在于学习和维持良好的态度。学习指通过代码审查学习别人的长处，同时帮助别人进步。保持良好的态度是指如果预知了有人会看自己的代码，那么就会自觉地尽量把代码写工整，即使审查代码的人偷懒没有真真看过，正所谓，如果人人都相信三尺之上有神灵，那么也就没人做坏事了，宗教的积极意义就在于此，扯远了。
- 过目数据库设计
- 及时更新或委派组员更新文档

关于文档，还需多说一句：只有再非常有必要写文档的情况下才写文档。如果写了一大堆可有可无的文档，代码更新后文档也要及时更新，很难做到的。一个新手面临着一堆和代码脱节的文档只会起误导作用，以代码本身的表达力和口头交流为主，以文档交流为辅，两手都要抓，一只手硬，一只手软。

不十分差劲的算法

相信大部人做的项目都是性能不敏感的，只要稍稍注意就能做好。比如该使用分页的地方就使用分页，该根据条件查询数据库的就别一次都查出来，该优化数据库就优化数据库。

此外，压力测试也很有必要，有的性能问题只有在数据量大了的情况下才出现。很多性能问题都是到了生产环境中才暴漏出来的。

代码之外

和团队建设比起来，以上只能算是雕虫小技，只起到了一个锦上添花的作用。如何使团队保持活力，保持积极性才是最重要的，通常也是最容易被领导忽视的，最难做好的，这些已超出笔者能力范围和本文的论述重点，有机会再写。祝大家工作愉快！

作文以记之

丁酉年夏，鹏镇守武汉。数月，政通人和，百废俱兴，乃重修代码规范，属予作文以记之。予观博客园胜状，驼峰命名，强制注释，迫之芸芸码士，俱迂腐巨细之论，此前人之述备矣。然，求同存异，团队建设，收买人心，论之甚少，此大道也，得之可平天下。