10讲炫技与克制：代码的两种味道与态度

虽然你代码可能已经写得不少了，但要真正提高代码水平，其实还需要多读代码。就像写作，写得再多，不多读书，思维和认知水平其实是很难提高的。

代码读得多了，慢慢就会感受到好代码中有一种味道和品质：克制。但也会发现另一种代码，它也会散发出一种味道：炫技。

炫技

什么是炫技的代码?

我先从一个读代码的故事说起。几年前我因为工作需要，去研究一个开源项目的源代码。这是一个国外知名互联网公司开源的工具项目，据说已在内部孵化了 6 年之久，这才开源出来。从其设计文档与代码结构来看，它高层设计的一致性还是比较好的，但到了源代码实现就显得凌乱了些，而且发现了一些炫技的痕迹。

代码中炫技的地方，具体来说就是关于状态机的使用。状态机程序本是不符合线性逻辑思维的，有点类似goto语句，程序执行会突然发生跳转，所以理解状态机程序的代码要比一般程序困难些。除此之外，它的状态机程序实现又是通过自定义的内存消息机制来驱动，这又额外添加了一层抽象复杂度。

而在我看来，状态机程序最适合的场景是一种真实领域状态变迁的映射。那什么叫真实领域状态呢？比如，红绿灯就表达了真实交通领域中的三种状态。而另一种场景，是网络编程领域，广泛应用在网络协议解析上，表达解析器当前的运行状态。

而但凡使用状态机来表达程序设计实现中引入的 “伪” 状态，往往都添加了不必要的复杂性，这就有点炫技的感觉了。但是我还是能常常在一些开源项目中看到一些过度设计和实现的复杂性，而这些项目往往还都是一些行业内头部大公司开源的。

在程序员的成长路径上，攀登公司的晋升阶梯时，通常会采用同行评审制度，而作为技术人就容易倾向性地关注项目或工程中的技术含量与难点。

这样的制度倾向性，有可能导致人为制造技术含量，也就是炫技了。就像体操运动中，你完成一个高难度动作，能加的分数有限，而一旦搞砸了，付出的代价则要惨重很多。所以，在比赛中高难度动作都是在关键的合适时刻才会选择。同样，项目中的

炫技，未必能加分，还有可能导致减分，比如其维护与理解成本变高了。

而**除了增加不必要的复杂性外，炫技的代码，也可能更容易出 Bug**。

刚工作的头一年，我在广东省中国银行写过一个小程序，就是给所有广东省中国银行的信用卡客户发邮件账单。由于当时广东中行信用卡刚起步，第一个月只有不到 10 万客户，所以算是小程序。

这个小程序就是个单机程序，为了方便业务人员操作，我写了个 GUI 界面。这是我第一次用 Java Swing 库来写 GUI，为了展示发送进度，后台线程每发送成功一封邮件，就通知页面线程更新进度条。

为什么这么设计呢？因为那时我正在学习 Java 线程编程，感觉这个技术很高端，而当时的 Java JDK 都还没标配线程 concurrent 包。所以，我选择线程间通信的方案来让后台发送线程和前端界面刷新线程通信，这就有了一股浓浓的炫技味道。

之后，就出现了界面动不动就卡住等一系列问题，因为各种线程提前通知、遗漏通知等情况没考虑到，代码也越改越难懂。其实后来想想，用个共享状态，定时轮询即可满足需要，而且代码实现会简单很多（前面《架构与实现》一文中，关于实现的核心我总结了一个字：简。这都是血泪教训啊），出 Bug 的概率也小了很多。

回头想想，成长的路上不免见猎心喜，手上拿个锤子看到哪里都是钉子。

炫技是因为你想表达得不一样，就像平常说话，你要故意说得引经据典去彰显自己有文化，但其实效果不一定佳，因为我们更需要的是平实、易懂的表达。

克制

在说克制之前，先说说什么叫不克制，写代码的不克制。

刚工作的第二年，我接手了一个比较大的项目中的一个主要子系统。在熟悉了整个系统后，我开始往里面增加功能时，有点受不了原本系统设计分层中的 DAO（Data Access Object， 数据访问对象）层，那是基于原生的 JDBC 封装的。每次新增一个 DAO 对象都需要复制粘贴一串看起来很类似的代码，难免生出厌烦的感觉。

当时开源框架 Hibernate 刚兴起，我觉得它的设计理念优雅，代码写出来也简洁，所以就决定用 Hibernate 的方式来取代原本的实现。原来的旧系统里，说多不多，说少也不少，好几百个 DAO 类，而重新实现整个 DAO 层，让我连续加了一周的班。

这个替换过程，是个纯粹的搬砖体力活，弄完了还没松口气就又有了新问题：Hibernate 在某些场景下出现了性能问题。陆陆续续把这些新问题处理好，着实让我累了一阵子。后来反思这个决策感觉确实不太妥当，替换带来的好处仅仅是每次新增一个 DAO 类时少写几行代码，却带来很多当时未知的风险。

那时年轻，有激情啊，对新技术充满好奇与冲动。**其实对于新技术，即使从我知道、我了解到我熟悉、我深谙，这时也还需要克制，要等待合适的时机**。这让我想起了电影《勇敢的心》中的一个场景，是战场上华莱士看着对方冲过来，高喊：“Hold！Hold！”新技术的应用，也需要等待一个合适的出击时刻，也许是应用在新的服务上，也许是下一次架构升级。

不克制的一种形态是容易做出臆想的、通用化的假设，而且我们还会给这种假设安一个非常正当的理由：扩展性。不可否认，扩展性很重要，但扩展性也应当来自真实的需求，而非假设将来的某天可能需要扩展，因为扩展性的反面就是带来设计抽象的复杂性以及代码量的增加。

那么，如何才是克制的编程方式？我想可能有这样一些方面：

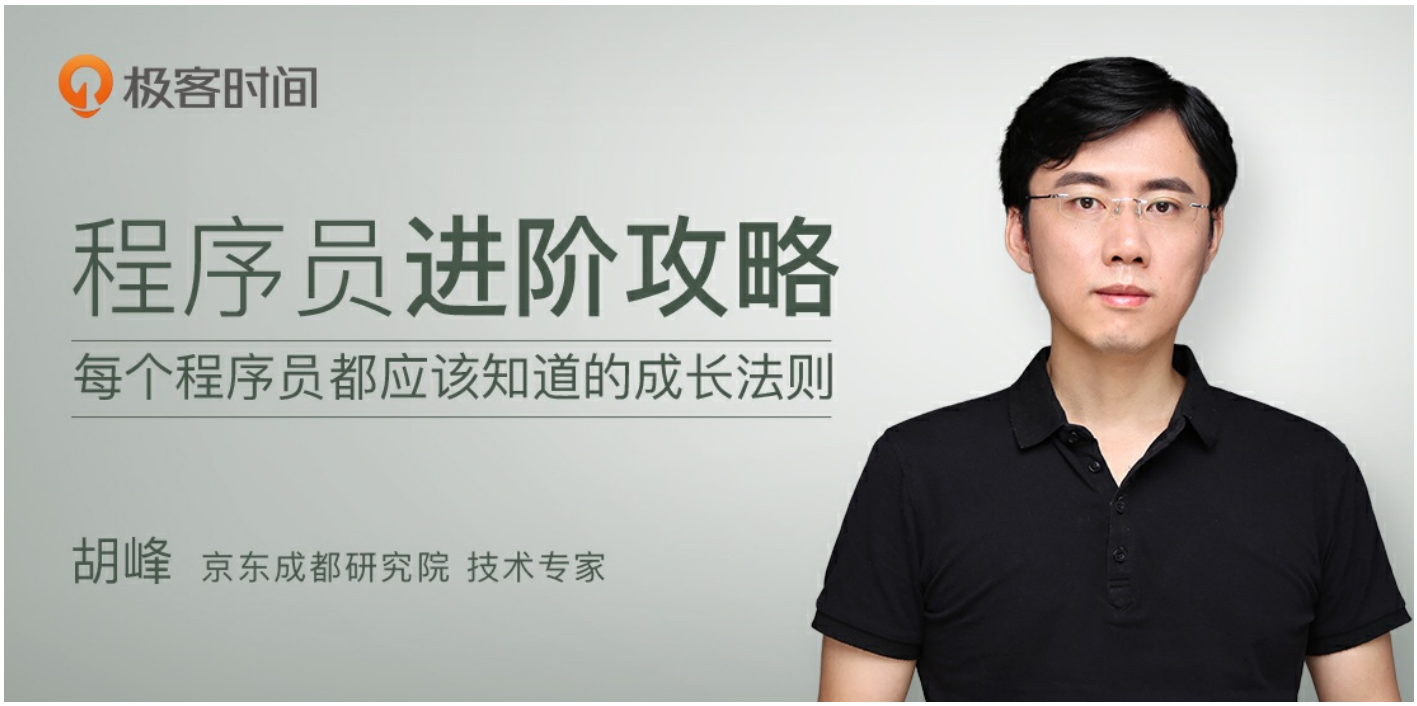
- 克制的编码，是每次写完代码，需要去反思和提炼它，代码应当是直观的，可读的，高效的。
- 克制的代码，是即使站在远远的地方去看屏幕上的代码，甚至看不清代码的具体内容时，也能感受到它的结构是干净整齐的，而非 “意大利面条” 似的混乱无序。
- 克制的重构，是每次看到 “坏” 代码不是立刻就动手去改，而是先标记圈定它，然后通读代码，掌握全局，重新设计，最后

再等待一个合适的时机，来一气呵成地完成重构。

总之，克制是不要留下多余的想象，是不炫技、不追新，且恰到好处地满足需要，是一种平实、清晰、易懂的表达。

克制与炫技，匹配与适度，代码的技术深度未必体现在技巧上。有句话是这么说的：“看山是山，看水是水；看山不是山，看水不是水；看山还是山，看水还是水。”转了一圈回来，机锋尽敛，大巧若拙，深在深处，浅在浅处。

最后，亲爱的读者朋友，在你的编码成长过程中，有过想要炫技而不克制的时候吗？欢迎你留言。



精选留言

third

心得如下：可能理解有误，望老师指点。

1.炫技俗称装B，有句老话说得好，装B被雷劈

2.炫技是指，为了展示自己的水平，人为的推高自己代码的复杂度和理解难度。（损人不利己）

3.克制俗称低调有内涵。

4.克制是指，为了程序的稳定，以及可读性，人为的降低代码的复杂度和理解难度（损己利人）

5.符合三个重要特征：
1，直观的，可读的，高效的
2，结构干净整齐
3，具有全局观

6.B还是要装的，只不过得趁早，因为成本比较低。

7.最后，每次我想要装B的时候，玩一下《绝地求生》这个游戏，感受一下

杀人最多的人不一定能取胜，幸存才是最大的胜利，闷声大发财。

2018-08-24 15:50

作者回复

哈哈，感谢third同学的总结心得，理解到位！喜欢第3条的阐释

2018-08-25 00:43

sprinty

只有炫技不克制过，才能克制不炫技。

2018-08-24 13:08

作者回复

嗯嗯，都是泪

2018-08-25 00:43

齐帜

不经历这些失败的尝试，可能也掌握不了这些技术

2018-08-24 11:15

作者回复

恩，每次炫技后都是给自己挖了坑

2018-08-25 00:53

Dream.

一句话道出真谛“手里拿个锤子，看见什么都是钉子”，真的是学了点新花样，就想立马运用在项目中，幻想着自己多么的牛逼。随意乱锤的结果就是换来各种坑以及加不完的班。真如楼上所说，只有炫技过后才能懂得克制。看见这篇文章，还不算太晚。

2018-08-24 19:19

作者回复

，都是加班换来的经验教训

2018-08-25 00:35

小名叫大明

我倒觉得程序员初期可以考虑炫技（装B）代码。

个人理解如下:

1. 程序员历史上确实存在某些祖传代码不易动，不能动，只能某些人维护的情况，某些人确实得到了想要的效果，获得了利益。（这不是我的追求，哈哈哈）

2. 炫，代表着复杂度，代表着更高程度的抽象，这对自己写代码的水平是一种提高的途径，将读到的东西，学以致用，才能是自己的，即使是个坑，也会有所收获。

从中学会了，能与不能的界限，拥有了以后能够一眼甄别那些小年轻的想法的"超能力"，哈哈

3.像您说的，升迁，生存之技。

2018-08-27 00:28

作者回复

就怕炫完了，坑挖完了，人跑了

2018-08-27 18:49

godtrue

实现一个功能越简单明了越好，否则真会害人不浅的，现在我们项目组维护的代码最早都有10年的，早一点没什么，关键是代码中的一个功能，代码是一坨一坨的，一个方法接一个方法看都看不完，又是核心代码一直计划重构，但是首先要理清代码业务逻辑，真是件令人头疼的事。

2018-08-26 10:12

作者回复

恩，理清再动手，此处坑多

2018-08-27 18:52

@XP

之前一家小公司里做安卓独立开发，看到网上兴起kotlin写安卓，我就整合到项目中，一边学习一边开发，之后大量重写代码，然后就开始出现很多兼容问题，想反悔都没办法，硬着头皮修bug

2018-08-24 07:40

作者回复

冲动了吧……

2018-08-25 00:57

湮汐

胡老师，在这里想请教你一个问题。本篇中提到阅读代码，我们很多程序员也提议大家阅读有名的框架源码。
可是，自己在阅读源码的过程中，有时候有点无从下手，一个框架的源码量太大了，不知道从哪读起。再就是因为自己基础的原因，很多源码读不懂，看了半天不知道那一块到底是做啥的。
这里有没有读源码的方法呢?

2018-09-26 12:28

作者回复

我读源码的方式是，发现这个框架实现了某个能力，我自己想不出怎么做，就会去看看别人怎么实现的。带着问题去读，可能有针对性一些

2018-09-26 21:45

登高

毕业时写了一个android app，那时对高层设计很感兴趣。在网上发现了MVP模式和clean架构，于是将它用起来。但是没想到这给开发增加了太多工作量，需要理解掌握的知识超出了预想，结果拖累了app一定的完成度。现在看当时有点冲动了
面对新技术，基本态度希望是积极拥抱，但在实际工作中要有克制。

2018-08-27 08:57

作者回复

恩，心态要开放，行动要克制

2018-08-27 18:48

艾尔欧唯伊

一针见血。。虽然干了四年多但是感觉有效代码量真的不多，哎。

2018-08-25 01:35

作者回复

如果你能清晰分辨出有效代码，并重构迭代出来，就又前进成长了一大步了

2018-08-25 11:41

edcSam

处处戳中我痛点，收获颇多，以后定当谨记克制与时机。

2018-08-24 08:41

作者回复

赞

2018-08-25 00:46

艾尔欧唯伊

这一篇很有道理，但是读起来有点抽象。。不知道实践中怎么落实

2018-08-24 08:40

作者回复

估计是代码写得还不够多，量到了就能感觉得到

2018-08-25 00:46

Hurt

什么是状态机程序啊。 什么又是goto 语句呢

2018-08-24 08:34

作者回复

哦……

2018-08-25 00:45

无聊夫斯基

当时做用户模版，需求还是比较简单的，本可以自定义快速实现，但为了炫技直接引入没用过的shiro，大部分功能没用上不说，还增加了其他功能的复杂度，坑嘛是肯定很多的，爬了好一阵子

2018-08-24 08:20

作者回复

哈哈

2018-08-25 00:54

维维

过度设计会增加代码的复杂度，增加维护成本。有时候一开始想的很好，让代码有很好的可扩展性，但过了很久，这份代码几乎不会怎么变。

我们项目中也有些代码，一个方法两三百行，复杂度很高，谁也不敢动。

2018-08-24 07:48

作者回复

几百行，还好，啃得动

2018-08-25 00:56

忆灬星辰

炫技和追新往往在自己的项目里，平时看看一些新书或是代码整洁方面的，全当练手。公司里比较老顽固。有的用的还是很早的技术。不过能够优化好代码就好，新技术往往要经过很长时间才会放进公司项目

2018-08-24 00:36

作者回复

业务要稳定，技术可以渐变，不要突变

2018-08-25 00:59

苦行僧

大部分情况是技没炫成反漏了拙：）

2018-12-17 13:05

作者回复

总有这样尴尬的时刻

2018-12-17 15:02

绿鲤鱼与驴。

使用合适的架构，合适的需要去实现适合的需求，不炫技，不装逼，活的久

2018-11-15 17:41

作者回复

务实

2018-11-18 15:03

米斯特粥

真正的牛人应该能直接闻出代码的坏味道，有时候代码注释越多可能代码越臭...

2018-10-05 13:48

作者回复

有这种情况存在

2018-10-07 20:26

王超

有时候自己压根没觉得那是炫技，是大义凛然，最后发现自己挖了一堆坑，看了老师的文章，才知道啥是炫技，啥是克制。。这就是年轻么。。

2018-09-25 07:32

作者回复

年经时都挺大义凛然的，填坑到天明常有的事^_^

2018-09-25 23:53