# 10 | 炫技与克制：代码的两种味道与态度

2018-08-24 胡峰


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。

极客时间

# 程序员进阶攻略

## 每个程序员都应该知道的成长法则

胡峰 京东成都研究院 技术专家