09 | 粗放与精益：编程的两种思路与方式

2018-08-22 胡峰

几年前，我给团队负责的整个系统写过一些公共库，有一次同事发现这个库里存在一个Bug，并告诉了我出错的现象。然后我便去修复这个Bug，最终只修改了一行代码，但发现一上午就这么过去了。

一上午只修复了一个Bug，而且只改了一行代码，到底发生了什么？时间都去哪里了？以前觉得自己写代码很快，怎么后来越来越慢了？我认真地思考了这个问题，开始认识到我的编程方式和习惯在那几年已经慢慢发生了变化，形成了明显的两个阶段的转变。这两个阶段是：

- 写得粗放，写得多
- 写得精益，写得好

多与粗放

粗放，在软件开发这个年轻的行业里其实没有确切的定义，但在传统行业中确实存在相近的关于“粗放经营”的概念可类比。引用其百科词条定义如下：

> 粗放经营（Extensive Management），泛指技术和管理水平不高，生产要素利用效率低，产品粗制滥造，物质和劳动消耗高的生产经营方式。

若把上面这段话里面的“经营”二字改成“编程”，就很明确地道出了我想表达的粗放式编程的含义。

一个典型的粗放式编程场景大概是这样的：需求到开发手上后，开始编码，编码完成，人肉测试，没问题后快速发布到线上，然后进入下一个迭代。

我早期参与的大量项目过程都与此类似，不停地重复接需求，快速开发，发布上线。在这个过程中，我只是在不停地堆砌功能代码，每天产出的代码量不算少，但感觉都很类似，也很粗糙。这样的过程持续了挺长一个阶段，一度让我怀疑：这样大量而粗放地写代码到底有什么作用和意义？

后来读到一个故事，我逐渐明白这个阶段是必要的，它因人、因环境而异，或长或短。而那个给我启发的故事，是这样的。

有一个陶艺老师在第一堂课上说，他会把班上学生分成两组，一组的成绩将会以最终完成的陶器作品数量来评定；而另一组，则会以最终完成的陶器品质来评定。

在交作业的时候，一个很有趣的现象出现了：“数量”组如预期一般拿出了很多作品，但出乎意料的是质量最好的作品也全部是由“数量”组制作出来的。

按“数量”组的评定标准，他们似乎应该忙于粗制滥造大量的陶器呀。但实际情况是他们每做出一个垃圾作品，都会吸取上一次制作的错误教训，然后在做下一个作品时得到改进。

而“品质”组一开始就追求完美的作品，他们花费了大量的时间从理论上不断论证如何才能做出一个完美的作品，而到了最后拿出来的东西，似乎只是一堆建立在宏大理论上的陶土。

读完这个故事，我陷入了沉思，感觉故事里的制作陶器和编程提升之路是如此类似。很显然，“品质”组的同学一开始就在追求理想上的“好与精益”，而“数量”组同学的完成方式则似我早期堆砌代码时的“多与粗放”，但他们正是通过做得多，不断尝试，快速迭代，最后取得到了更好的结果。

庆幸的是，我在初学编程时，就是在不断通过编程训练来解答一个又一个书本上得来的困惑；后来工作时，则是在不断写程序来解决一个又一个工作中遇到的问题。看到书上探讨各种优雅的代码之道、编程的艺术哲学，那时的我也完全不知道该如何通往这座编程的“圣杯”，只能看着自己写出的蹩脚代码，然后继续不断重复去制作下一个丑陋的“陶器”，不断尝试，不断精进和进阶。

《黑客与画家》书里说：“编程和画画近乎异曲同工。”所以，你看那些成名画家的作品，如果按时间顺序来排列展示，你会发现每幅画所用的技巧，都是建立在上一幅作品学到的东西之上；如果某幅作品特别出众，你往往也能在更早期的作品中找到类似的版本。而编程的精进过程也是类似的。

总之，这些故事和经历都印证了一个道理：**在通往“更好”的路上，总会经过“更多”这条路。**

好与精益

精益，也是借鉴自传统行业里的一个类比：精益生产。

> 精益生产（Lean Production），简言之，就是一种以满足用户需求为目标、力求降低成本、提高产品的质量、不断创新的资源节约型生产方式。

若将定义中的“生产”二字换成“编程”，也就道出了精益编程的内涵。它有几个关键点：质量、成本与效率。但要注意：在编程路上，如果一开始就像“品质”组同学那样去追求完美，也许你就会被定义“完美”的品质所绊住，而忽视了制作的成本与效率。

因为编程的难点是，无论你在开始动手编程时看过多少有关编程理论、方法、哲学与艺术的书，一开始你还是无法领悟到什么是编程的正确方法，以及什么是“完美”的程序。毕竟纸上得来终觉浅，绝知此事要躬行。

曾经，还在学校学习编程时，有一次老师布置了一个期中课程设计，我很快完成了这个课程设计中的编程作业。而另一位同学，刚刚看完了那本经典的《设计模式》书。

他尝试用书里学到的新概念来设计这个编程作业，并且又用 UML 画了一大堆交互和类图，去推导设计的完美与优雅。然后兴致勃勃向我（因为我刚好坐在他旁边）讲解他的完美设计，我若有所悟，觉得里面确实有值得我借鉴的地方，就准备吸收一些我能听明白的东西，重构一遍已经写好的作业程序。

后来，这位同学在动手实现他的完美设计时，发现程序越写越复杂，交作业的时间已经不够了，只好借用我的不完美的第一版代码改改凑合交了。而我在这第一版代码基础上，又按领悟到的正确思路重构了一次、改进了一番后交了作业。

所以，别被所谓“完美“ 的程序所困扰，只管先去盯住你要用编程解决的问题，把问题解决，把任务完成。

编程，其实一开始哪有什么完美，只有不断变得更好。

工作后，我做了大量的项目，发现这些项目都有很多类似之处。每次，即使项目上线后，我也必然重构项目代码，提取其中可复用的代码，然后在下一个项目中使用。循环往复，一直干了七八年。每次提炼重构，都是一次从“更多”走向“更好”的过程。我想，很多程序员都有类似的经历吧？

回到开头修改Bug 的例子，我用半天的时间改一个Bug，感觉效率不算高，这符合精益编程的思路吗？先来回顾下这半天改这个Bug 的过程。

由于出问题的那个公共库是我接到Bug 时的半年前开发的，所以发现那个Bug 后，我花了一些时间来回忆整个公共库的代码结构设计。然后我研究了一下，发现其出现的场景比较罕见，要不不至于线上运行了很久也没人发现，属于重要但不紧急。

因此，我没有立刻着手去修改代码，而是先在公共库的单元测试集中新写了一组单元测试案例。单元测试构建了该Bug的重现场景，并顺利让单元测试运行失败了，之后我再开始去修改代码，并找到了出问题的那一行，修改后重新运行了单元测试集，并顺利看见了测试通过的绿色进度条。

而作为一个公共库，修改完成后我还要为本次修改更新发布版本，编写对应的文档，并上传到Maven 仓库中，才算完成。回想这一系列的步骤，我发现时间主要花在了构建重现Bug 的测试案例场景中，有时为了构建一个测试场景编写代码的难度可能比开发功能本身更困难。

为修改一个Bug 付出的额外单元测试时间成本，算一种浪费吗？虽说这确实提高了代码的修复成本，但也带来了程序质量的提升。按前面精益的定义，这似乎是矛盾的，但其实更是一种权衡与取舍。

就是在这样的过程与反复中，我渐渐形成了属于自己的编程价值观：世上没有完美的解决方案，任何方案总是有这样或那样一些因子可以优化。一些方案可能面临的权衡取舍会少些，而另一些方案则会更纠结一些，但最终都要做取舍。

以上，也说明了一个道理：**好不是完美，好是一个过程，一个不断精益化的过程。**

编程，当写得足够多了，也足够好了，你才可能自如地在“多”与“好”之间做出平衡。

编程的背后是交付程序系统，交付关心的是三点：功能多少，质量好坏，效率快慢。真实的编程环境下， 你需要在三者间取得平衡，哪些部分可能是多而粗放的交付，哪些部分是好而精益的完成，同时还要考虑效率快慢（时间）的需求。

编程路上，“粗放的多”是“精益的好和快”的前提，而好和快则是你的取舍：是追求好的极致，还是快的极致，或者二者的平衡？

在多而粗放和好而精益之间，现在你处在哪个阶段了？欢迎留言谈谈你的看法。

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