

02 | 低代码到底是银弹，还是行业毒瘤？

2022-03-16 陈旭

《说透低代码》

课程介绍 >



讲述：陈旭

时长 19:34 大小 17.93M



你好，我是陈旭。

说到低代码，有人说它是毒瘤，也有人说它是银弹。那到底应该怎么看呢？这就是我们今天要解决的核心问题。

先上结论：**存在即合理**。

这里的“存在”包括两个角度：一是银弹论，二是毒瘤论，无论从哪个角度看，既然存在这样的论调，就有它们的合理性。

我们暂时不介入这两个言论的细节，而是先把关注点移到低代码本身，先回答这个问题：低代码到底是要革程序员的命？还是成为程序员工具箱里的另一个工具？

如果你觉得低代码的目的是为了革程序员的命，是要把程序员的手脚给捆住，是要束缚程序员的创造性，是要把复杂的现实世界强制机械化、特例化、流程化，那么大概率你会接受毒瘤论。你甚至会毫无保留地否定低代码，认为 **Low Code** 除了 **Low** 以外，不会有 **Code**。

毕竟，变革是有成本的，而且往往成本巨大，无论是一个人还是一个团队，保持惯性，留恋舒适区才是正常的。我们几十年来用的软件从来都是双手敲代码创造出来，抛弃这个已被无数次证实可行的方法，拥抱一个“饱受争议”的新方法，这本身就需要有莫大的勇气，也需要承受风险。

虽然会有很多勇于探索、期望尝试新方法的人和团队，也有很多受困于 **Pro Code**（手敲代码的方式）的各种痛点被迫自我变革的人和团队，但是更多的人和团队是倾向于保持现状的，即使嘴上不说，身体却很诚实。每次回顾会都能复盘出一堆的代码问题，然后一而再、再而三地立 **flag** 说要改进，但多数会给出各种借口说明问题无解，然后继续这样的循环。有惯性就有排斥，有排斥就有各种负面论调，毒瘤论是其中典型的代表。

Pro Code 创造的软件更“香”吗？

如果能换个角度，把低代码当做程序员工具箱里的另一个工具，我们再次审视银弹论和毒瘤论的冲突时，会发现一些不同点。工具不会革谁的命，只会让大家的日子更好过。工具有好有坏，不好用时就丢弃，好用时就多用用。

条条大路通罗马，我们前往罗马除了传统的步行、马车外，还能开汽车、坐飞机，甚至也可以在路况好的时候开汽车，风景好的时候下车漫步抑或纵马驰骋，只要能按时到达罗马，我相信没人会太关注过程，所以在去罗马的旅途只要你开心就好。

开发软件写 **bug** 也是类似的，**Pro Code** 方式创造的软件不会比其他方式生产出来的软件更香、不会卖更多的米，因为软件的用户完全不关注软件是怎么被创造出来的。

当然，软件研发里的“罗马”不仅仅是一个目的地，所以这里带上双引号了。在软件业里，“罗马”除了指代交付的业务功能外，它的内涵还有：交付物可维护性、可扩展性、可测试性、交付过程的成本，甚至是团队的新陈代谢等等许多因素。同样地，我们审视到底是“骑马”好，还是“开车”好的时候，需要关注的不仅仅是心情，而需要关注到所采用的方式是否能在各个方面都有比较好的表现。这是一个极复杂的评判过程，且各个团队有各自的侧重，没有标准答案。

即使评判过程如此复杂，但是有一点是明确的（甚至早已是业界共识）：**Pro Code 不是软件开发的银弹，要不然就不会有低代码等多种新方式被提出来甚至展开实践了。**

Pro Code 第一个问题是门槛高。虽然我们自贬为“码农”，但是根据 **GitHub** 的统计数据，去年国内只有大约 **755** 万多开发人员，一个人可能只会写 **hello world** 就被算进来，摊到各个细分研发领域后，人数就少得可怜了。我去年面试了几十人，但是最终入职的寥寥无几，招聘难成了我的痛点，我甚至将其写入年终报告。

Pro Code 第二个问题是跨界难。虽然都是写代码的，但是 **Java** 程序员可能很难玩得转 **C/C++**，前端程序员很难玩得转 **Java/Scala** 等后端技术，反之亦然。

一个典型例子是：全栈这个词是在在 **Node.js** 火热起来之后才被发明出来的，在这之前，前后端通吃的只能是极少数顶尖骨干的专属。但是，即使现在有了 **node.js** 实现前后端跨界，我们跨越到其他领域依然困难。总之，即使在具体业务场景下，要端到端交付一个完整业务，对一个人，甚至一个团队来说，都不是一件简单的事情。

第三，代码编写只是第一步，之后还有许多问题需要解决。像代码所依赖的第三方库的开源合规治理、第三方和己方的代码安全漏洞检测和治理，还有代码性能、代码测试、运行时运维等，这些工作不是难度大，就是繁琐。最后，为了对抗代码库的熵增，避免代码仓库越来越混乱，越来越难维护，还必须引入代码走查机制，让经验丰富的程序员来把关。

Pro Code 存在的问题显然不止这些，但这些已经足够说明问题了。

既然 **Pro Code** 有如此多的问题，而且许多问题是由于代码自身导致的，那么引入一些工具来降低代码量，许多问题也就可以缓解甚至解决了。代码本身并无直接价值，业务才具有直接价值，从来没听说过哪家公司是凭着一个百万千万行代码 **repo** 作为资产上市的。公司之所以能上市肯定是由于资本市场认可它的业务价值，代码只不过是用于实现业务的一种场常见方式而已，但绝对不是唯一的。

总之，不管是白猫黑猫，能抓老鼠的就是好猫，吃喝拉撒越少的猫则是更好的猫。只要能够实现相似价值的业务，并且承载该业务的方式的副作用比代码要少，那么这就是一种更好的方式。

到这儿，我相信你已经有足够的理性来把低代码作为一种工具来看待，而不认为这是一种程序员自我革命的手段。

Low Code 银弹论合理吗？

既然 Pro Code 不是银弹，那 Low Code 是不是银弹呢？当然也不是银弹。

在理解这一点之前，我们先来搞清楚 Pro Code 和 Low Code 的区别。为了说清楚这两者的异同，我要引入“第三者”No Code，我们把这三个看起来很相似的概念放在一起比较。

Pro Code 和 No Code 实际上都很好理解，一个是纯代码，一个是无代码。假设 Pro Code 的代码量是 100，那 No Code 就是 0，所以 Pro Code 和 No Code 是截然不同的，甚至你可以认为这两者毫无关系。No Code 的最典型形态莫过于 SaaS 类的产品了。

那 Low Code 应该摆在哪个位置呢？似乎应该介于 0 到 100 之间？50？80？其实都不是的，关键点不在于多少，而在于有没有！

Low Code 当然是有代码的，所以它和 Pro Code 是一路货色，从创建业务价值的最根本上说，它们是一样的，都是通过代码来创建业务价值。而 No Code 则不是，它只是对已有业务的二次组合来创建业务价值，No Code 创建业务的全过程都没有源码的参与。

那 Pro Code 和 Low Code 的差异在哪呢？我认为本质差异在于源码在这两者创造业务价值的过程中所扮演的角色。Pro Code 是把代码当作关键输入来创建业务的，Low Code 则不是，它的输入是一些结构化的数据。

Low Code 工具有能力将结构化数据生成为源码，然后再采用与 Pro Code 相同的方式将源码转为业务能力。很显然的一点是，Low Code 把源码当做中间产物，而 Pro Code 则将源码做为关键输入。相信现在你应该可以分清楚 Pro Code、Low Code、No Code 之间的差异了。

那么 Low Code 为啥也不是银弹呢？关键就在于，Low Code 是采用无逻辑的结构化数据来描述业务的，对于相同业务，Low Code 的描述能力要弱于有逻辑的 Pro Code。所以，Pro Code 都做不到的事情，Low Code 当然也做不到。

根据前面与 Pro Code 的比对，我们可以看出，**Low Code 对业务的描述能力既弱于 Pro Code，也与 Pro Code 没有实质差异，看起来 Low Code 一无是处啊。**我认为这可能是低代码毒瘤论的理论基础。别急，继续往下看。

Pro Code 开发方式是指令式的，它的开发过程是告诉计算机如何一步步实现某个业务。**Low Code** 则不然，它的开发过程是不停地在描述和细化这个业务最终的样子。

可以说，**Low Code** 的开发人员的思维方式与 **Pro Code** 的开发人员完全不一样，他们始终在思考这个业务应该是啥样的，而 **Pro Code** 的开发人员则是在开发过程中不停地思考如何将业务翻译成一条条指令。其中关键的一点是，对于 **Pro Code** 开发人员来说，这个业务最终的样子不是天上掉下来的，抑或大风刮来的，而是要在他们着手翻译成指令之前先想清楚的。

所以结论就是，**Low Code** 开发人员要比 **Pro Code** 开发人员少做一个事情：无需将业务翻译成指令，从而他们得以用更高的效率实现相同的业务，在单位时间内创造更多业务价值。

低代码模式除了效率外，还有另一个卖点，那就是低技术门槛。它可以赋能更多人，使其可以快速参与业务开发。没错，就是实现外卷！

根据我们前面说的，**Low Code** 依然有代码，但 70%~90% 甚至更多比例的代码是自动生成的。这里要特别注意的是，其中 100% 的框架性的代码都是自动生成的，这部分代码不但与业务无直接关系，即没有直接业务价值，还是最难开发的，一出问题都是大问题。而剩余的小部分需要开发者填写的代码，则基本上都是表达式，或者部分复杂业务逻辑。

表达式也好，强业务逻辑代码也好，都是在直接实现业务功能，所以即使在低代码平台上写代码，那也是在直接实现业务逻辑。那些只有业务能力但无开发能力的人、需要写 UI 的后端开发，或者是需要写业务的前端开发等人群，都可以在低代码平台上实现跨界开发，实现端到端的业务交付能力。赋予原本没有技术能力的人快速参与的能力，是低代码的另一个重要能力。

基于以上提效和赋能两点，我们有足够的理由来认为低代码银弹论是合理的，银弹论并非一群“不务正业”、“整天想着再造一个轮子”的人的自嗨。

Low Code 离银弹还有多远？

低代码毒瘤论者只看到表面，或者思考深度不足，没能从实质上看到 **Pro Code** 和 **Low Code** 的差异。即使如此，我也认为毒瘤论是有合理性的。毕竟很多实现 **Low Code** 的人自己也没想明白就匆匆动手，为了达到少代码（而非低代码）的目的而做出各种各样无理约束，你不能这个、不能那个，最终不但没能发挥出低代码的优势，反而降低了效率，被他人诟病。

即使抛开这类情形不说，Low Code 本身的成熟度也需要改进，**一个重要改进点是：如何有效解决强逻辑场合下的可视化配置方法**，这是 Low Code 采用无逻辑的结构化数据描述业务常见的重要短板。这个问题至今也在困扰着我，我认为我们目前给出的解决方案并非最优解，仍然需要孜孜不倦地探索出更合适的手段。

低代码要真正成为银弹，除了解决强逻辑的可视化配置外，要做的事情还有很多。有一部分我在前文已经有过描述了，这里再给你简单总结和拓展一下，一方面是为了用作毒瘤论非合理性的额外论据，另一方面也是为了给这个专栏后续的内容埋点伏笔。

首先，快速部署能力是要有的，最基本的需要一键导出所有运行时需要的文件，最好能做到一键部署和运行时，这样的能力将给业务的小伙伴提供巨大的便利，大幅减少他们的试错时间。

其次，也是最重要的，低代码平台还需要关注生成的 App 的可测试性。如果生成的 App 是一个黑盒，出错时无从下手，日志打印惜字如金，要是这时候平台再对业务团队的困境置之不理甚至推诿，那无论是谁都会厌恶这样的开发方式的。

可测试性是一个非常综合的能力，从最基本的部署运行开始，到丰富准确的日志，再到直接给予业务团队调试支持，甚至到直接提供自动化测试的能力，这里头涉及的不仅是技术，还有团队间如何协作的问题。

还有一个容易被无视的能力是，低代码平台需要支持多人协作开发，当业务越来越复杂，多人协作是刚需。这个问题往往在平台建设初期就容易被无视，在后期随着应用的复杂度上升后才被提出，这时候再去实现，成本会非常大。此时平台有可能会简单粗暴地采用互斥的方式来掩耳盗铃。

要知道，业务团队往往面临巨大的交付工期压力，如果小李无法在小王编辑时上手，一急之下他们可能将数据复制一份来同时编辑，结果发现无法手工合并结构化数据，这样的情形放谁身上都会抓狂。所以支持多人协作开发是低代码平台非常必要的一个能力，而这背后其实还隐藏着编辑历史管理、分支管理等潜在需求，这些都对业务团队多人协作效率有很大的影响。

再者，兜底能力也是很重要的。低代码平台不可能面面俱到，它总有能力边界，但这个能力边界不能束缚业务团队的探索。业务需要紧随市场甚至引领市场，而市场是千变万化的，任何公司都无法决定，所以要把“业务提出低代码平台能力之外的需求”当做一种常态。此时，低代码平台需要有一种策略帮助应用快速实现需求，哪怕直接上手编码乃至 Hack。这样的策略就是兜底策略。

此外还有一些增值功能，包括 UX 设计规范自动对齐、提供 UX 设计稿转代码（D2C）能力、App 的可维护性、App 的埋点 & 数据采集、App 的开源合规治理、App 的安全漏洞治理、App 的性能等等。简单来说，这些都是低代码平台的亮点能力，并且是拉开与 Pro Code 差距的重要能力。

讲到这里我们可以看到，低代码平台只关注开发能力是远远不够的。而毒瘤论的另一个重要基础就是来自于许多低代码平台只注重开发能力，而无视业务交付过程的其他能力，特别是对 App 可测试性的漠视。

作为低代码的坚定支持者和践行者，我当然不同意低代码是毒瘤这个论断，但是我不轻易去喷这些言论，而是努力去理解和挖掘毒瘤论的言论基础，从中分析出那些容易被我们无视或者轻视的需求。只要我们常常能细心、谦虚地观察业务开发过程中的方方面面，并及时发现业务团队的痛点和需求并及时解决，以一种服务者的态度来为业务提供服务，我相信，毒瘤论会渐渐消散的。

总结

今天这一讲到这里就结束了。其实很少有一种技术像低代码这样同时被两种极端言论评价，今天我们从一个理性的角度审视了这两种言论，并尝试去理解它们的思路和痛点。

我们从 Pro Code、Low Code、No Code 的对比中了解到 Pro Code 与 Low Code 的关键差异不在于代码量的多少，而在于：

- 代码在这两者创造业务价值的过程中所扮演的角色，对 Pro Code 来说，代码是关键输入，而对 Low Code 来说，代码仅仅是中间产物、是副产品；
- 使用 Low Code 开发虽然也需要少量编码，但是基本上都是在填写表达式，直接实现业务价值，与业务价值无关的代码则几乎全都被自动生成；
- 使用 Low Code 开发业务的人几乎时时刻刻都在描述和细化业务最终的样子，使用 Pro Code 的开发人员不仅需要思考业务最终的样子，还要将其翻译成一条条计算机指令。

这些差异正是 Low Code 的比较优势，这也是为啥 Low Code 可以帮助业务提效和赋能的原因：无需长篇累牍地编码可以大幅降低使用的门槛，实际上就是对无编码技能者进行赋能；几乎所有框架性、非功能代码全部自动生成，以及无需将业务翻译成一条条计算机指令，可以大幅压缩开发周期，实际上就是在提效。

但即使如此，当下的 **Low Code** 也不是银弹，因为我们在分析毒瘤论的过程中意识到 **Low Code** 除了注重开发能力之外，还应该具有诸多其他能力以覆盖业务研发全生命周期，包括一键导出和部署、较高的可测试性甚至自动化测试能力、多人协作、兜底能力等等，若低代码平台不具备这些能力，则很难发挥低代码技术的优势。

当然，我们也发现了 **Low Code** 带来了许多增值功能，包括自动对齐 **UX** 设计规范、**UX** 设计稿转代码（**D2C**）能力、**App** 的埋点 & 数据采集、开源合规治理、安全漏洞治理等等，这些功能是拉开 **Pro Code** 差距的重要抓手。


思考题


低代码平台是否只需把开发能力发挥到极致就可以了？除了开发能力之外，低代码平台还需要注重哪些能力的建设？你认为其中最重要的是哪些？欢迎在评论区留言。

我是陈旭，我们下节课见。

分享给需要的人，Ta 订阅超级会员，你最高得 50 元

Ta 单独购买本课程，你将得 20 元

 生成海报并分享

 赞 3  提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 01 | 低代码平台到底是什么样的？

下一篇 03 | 低代码的天花板：一个完备的低代码平台应该具备哪些条件？

说透低代码

拨开迷雾，解析低代码平台架构

陈旭

中兴通讯
软件研发资深专家



新版升级：点击「👤请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

精选留言 (5)

写留言



抱紧我的小鲤鱼

2022-03-18

在国内，架构是为业务服务的，没有业务的需求也就不需要架构的支撑，也谈不上到底是Pro code还是low code了

作者回复: 确实，你的留言让我想起多年前听到的一句话，脱离业务谈架构的都是耍流氓。同样的，脱离业务谈低代码的也是要流氓



1



coder

2022-03-23

low code要结合固定的业务场景，通用的话，灵活度不够。



卡特

2022-03-20

集成能力 报表设计 开放能力 以及扩散能力



Tree New Bee

2022-03-16

结合业务能力抽象业务组件，集成外部能力的封装成平台业务组件。比如集成ocr识别可以使用到费控报销场景 发票拍照识别

作者回复: 是的，低代码平台具有非常强的综合性属性，一定要做成开放式架构，这样才能接入更多业务定制功能，如你说的OCR扫描



Light 胖虎

2022-03-16

我认为还需要注重业务能力吧，在开发过程中会存在很多业务相关的重复代码，因此低代码也需要解决一些业务功能

作者回复: 的你说得没错，任何低代码解决方案最终的目标都是要解决业务问题，从来没有哪个低代码解决方案是只专注于解决技术问题的，技术是没有价值的业务才有价值。但是凡事都有一个发展的过程，一般都先解决技术问题，然后再聚焦到业务问题。

