加餐十 | 如何接手一坨烂业务代码？如何在烂业务代码中成长？

王争 · 设计模式之美

在我们的职业生涯中，很少有机会可以从零开发一个项目，大部分都是接手别人的代码继续开发，或者做些维护性开发。而且，对于大部分业务系统来说，因为业务导向，需求倒逼，开发工期紧，团队往往都不是很重视代码质量，快速上线是第一要务。所以，很多团队的代码质量一般都不怎么高。埋坑无数、没有文档、也没有注释，代码读不懂、也不敢改，这对于新人来说，会非常苦恼。今天，我们就聊一聊，如何接手一坨烂业务代码，以及如在烂业务代码中的成长？

话不多说，让我们正式开始今天的内容吧！

如何接手一坨烂业务代码？

在我过去 10 年的工作经历中，我接手过很多个代码质量比较烂的项目。这些项目都有很多共性的特点，大部分都已经维护了两三年，甚至五六年之久，代码量很大，有十几万行以上，并且大部分代码都没有任何注释，业务功能非常庞杂，也没有对应的业务文档。

除此之外，代码中还充斥着各种临时解决方案（Workaround）、硬编码（Hard Code）、遗留代码（Legacy Code），还有很多匪夷所思的设计。对于有些设计来说，我们称之为“反人类”设计或者“故意挖坑”，一点都不为过。如果没有老员工给你解释上下文，你万万都想不到它为什么这么设计和实现。

实际上，**要想接手一个业务系统，前提是要读懂代码，而读懂代码的关键，是要熟悉业务。**只要业务搞清楚了，代码只不过是对业务的翻译，对照着业务看代码实现，看懂并不是件难事。不过，我所接手的这几个项目，基本上都是零文档，所有的业务知识都是靠口口相传。所以，搞清楚业务，就成了接手项目最难的事情了。

面对如此庞大的项目代码，没有文档，几乎就是两眼一抹黑。原来参与这个项目开发的老同事，有的离职，有的去做其他新项目，一直问他们也不好意思，所以，大部分情况下，我都只能硬着头皮，通过阅读代码反推业务功能。

如果代码质量比较高，模块划分清晰，命名规范，那通过读代码反推业务，也并非不可能的事情。但真实的情况往往事与愿违，就像我们前面提到的，代码中充斥着临时解决方案、硬编码、遗留代码等各种坑，这就使反推业务变得非常困难。对于代码中的这些坑，尽管我不想一直麻烦同事，但也只有多问才能最快速地解决。

在读代码的过程中，我非常重视知识的文档化，我会把读懂的每个业务都写到文档中。当然，这其中也包括前面提到的各种坑。对于复杂的业务流程，我还会画一些流程图。读代码的过程非常痛苦，花了好几个月，我才有信心说，自己几乎把所有代码都搞清楚了。同时，我也做了一件过去几年都没有人做的事情，那就是补充完整了技术文档和业务文档，之后再有新同事加入，看了我的文档，就可以很快了解代码、了解业务，很快就能上手开发代码。

总结一下，即便代码再烂，只要有完善的业务文档，先理解业务，再去看代码，几乎就没啥难度了。对于零文档的项目，大部分情况下，我们只能通过代码来反推业务。当然，对于有些坑来说，必要的情况下，我们也要询问前辈来搞定。在读代码的过程中，我们要将得到的知识文档化，这也是对公司和团队来说最有价值的部分。

如何在烂业务代码中成长?

有人一遇到这种烂业务代码，就觉得很心烦，我反倒不一样。恰恰相反，相比接手好代码，我觉得接手烂代码，虽然过程更加痛苦，但同时也会给我更多施展才华的空间、锻炼技术的机会，我的成长也会更多。

除此之外，很多人觉得做偏底层的开发（基础架构、框架、中间件等开发）才锻炼技术，做业务系统没有挑战，技术上没有成长，对此非常苦恼。实际上，我觉得这种看法是比较片面的。做业务开发的难度不亚于底层开发，做好也不是件容易的事情，同样可以积累技术、锻炼能力。

偏底层的开发更加考验程序员在某一细分领域的技术深度，偏业务的开发更加考验程序员的能力，比如沟通能力、分析问题解决问题能力、权衡取舍能力、架构能力等，毕竟业务多种多样，问题千奇百怪，单一细分领域的经验很难应对所有问题。

实际上，业务系统的开发难度一般来自两个方面：高性能要求和业务复杂。

解决性能问题，你需要具备一定的架构能力，有一定的技术广度，需要对各种基础架构、框架、中间件都有所了解。光了解还不够，还要有一定的技术深度，最好能对原理甚至是源码有所研究。除此之外，还要有一定的使用经验。广度、深度、经验三者配合，这样才能做到恰到好处组合这些技术搭建架构，解决性能问题，并且在出现问题之后才能快速地解决。

应对大型项目的业务复杂性，要想让项目代码一直在你的掌控范围内，你需要有很强的业务建模能力、复杂逻辑的抽象能力、代码翻译能力等。对于一个人的基本素质、基础能力的要求要更高。实际上，对于复杂业务系统来说，对业务的熟悉也能成为你的竞争壁垒，成为升职加薪的砝码。我前面也讲到，低级别的晋升靠技术，比如升阿里的 P7，高级别的晋升靠业务，比如升阿里的 P8、P9，或者换个说法，高级别的晋升，靠业务比单纯靠技术，更容易一些。

如果你参与的项目，性能要求高、业务也复杂，那恭喜你，好好干就成了。如果你参与的项目，在性能和复杂度上，只兼具其中一点，那也不错，值得一做。如果你参与的项目，既没有性能压力、业务也不复杂，那也别太着急，走着瞧，实在不行再跳槽。

课堂讨论

在过往的项目经历中，你有没有像我一样，接手过代码质量比较差的代码？你又是如何顺利接手的呢？

欢迎留言和我分享你的想法。如果有收获，也欢迎你把这篇文章分享给你的朋友。

AI智能总结

接手烂业务代码并不容易，但通过熟悉业务、阅读代码、文档化知识等方法，可以成功应对挑战。在接手烂代码的过程中，不仅可以锻炼技术能力，还能通过解决性能问题和应对业务复杂性来提升自己。业务系统开发的难度在于高性能要求和业务复杂性，因此需要具备架构能力、技术广度和深度，以及对业务的熟悉。通过接手烂业务代码，可以获得更多施展才华的空间，锻炼技术的机会，以及提升自己的成长。

全部留言 (37)

最新　精选

沧浪之水
2020-08-26

不知道作者有没有在小公司工作的经历，我感觉所有的问题，归根结底都是人的问题。人的素质高了，再难也能一步步解决，因为大家都是奔着解决问题去的。 人的素质不行，再简单的问题，也能搞成无解，中间各种推诿，甩锅，真的是不厌其烦

作者回复: 是的，核心是人的问题。

共 4 条评论>　29

Obed
2020-07-11

加餐的文章还是挺有收货的,都是争哥工作这么多年的感触

作者回复: 有的也比较水，多包涵哈，整体有收获、大部分有收获就值！

12

大头
2020-07-10

业务建模，该从哪些方面去考虑考虑？争哥有没有推荐资料？

作者回复: 现在比较火的DDD可以参考下，不过，技巧层面的东西并不多，业务建模更重要的还是看你对业务的理解程度。

共 2 条评论>

6

咕咕噜噜

2020-08-30

争哥，能说说梳理业务的一些技巧吗，包括产出一些技术文档

作者回复: 感觉也没啥技巧，先把核心业务理顺，边整理边输出，把不清楚的也记下来，慢慢记下来的所有问题都搞清楚，基本上就搞清楚的差不多了。

5

Jxin

2020-07-10

1.只要你编码意识在提高，接手烂代码应该是常态。
2.这次有幸和业务架构师合作搞中台。（这算不上我理解的中台，只是这么叫而已）

3.首先说一个问题，业务架构师在领域的理解和建模上都很不错，但是在实际开发落地却显得有点刚。我们知道，老单体拆解往往是从圈表开始，这就先动了数据结构。然而既然是烂代码，它的隔离性必然不好(业务层直接调dao层，dao层做数据分发等待)。那么改了数据结构，连带的一切都受牵连，置底向上的重构代码，往往很难发挥新的数据模型的价值，而且这个工作量还很大，把数据迁移算在内，可能2-3个月就这么一层都不见得能重构成功。而对外这一层重构的价值是很不明显的。干2个月，没有看得到的成效，可能就要被砍掉。

4.也感谢团队和架构师的认可，整个中台改造的规划以我的方案为基础。我们做中台是为了增效，主要两个方面。首先，已有业务功能要可以复用（业务流程的可控制，功能的可编排）；其次，新功能的加入要尽量少受原有代码的影响（提供扩展性，隔离复杂性或者说分离关注点）。所以我也就这两点自上而下做重构。首先，搭建最外层编排功能的调度层；接着，对已有功能和策略做标准化的适配（适配器:统一多个类的接口设计）,这里既有适配和参数转换的工作，也有部分"大方法"拆解重构成小的标准方法的工作。那么一期的重构就在上层编排层的实现和已有功能的适配，加上部分"大方法"的拆解。整个工作差不多在2-3礼拜就能完成。后续的工作就是对每个适配的标准化接口做实现上的重构。（这里数据库的表结构不做变化，但新的功能会基于抽象的领域模型来实现<领域模型到do会兼容老数据结构的转换>。当所有功能基于领域模型实现完后，再对领域模型到do这一层做重构，圈表拆表合并表。）

5.以上，2-3周我们的重构就可以跟着版本迭代发出去。上游编排支持新流程的诉求也能满足。后续工作就落在每个标准功能的重构上，关注点比较收敛。（同一时间，重构过程里也把技术相关的代码抽离业务代码，重定义本项目的dto<老项目没有dto的概率，对外的api包啥都有，有do也有其他组api包的do，非常恶心>）

6.实时上，我认为做重构重来都不只是开发的事。像栏主这样捏着鼻子啃烂代码梳理业务逻辑，我也干过。不同的是，我一直都拉着产品对业务逻辑。重构过程会换业务逻辑实现，甚至会删除整块已经没什么价值的功能。一个项目烂，开发是主要原因，产品也难逃其责。

程序员小跃

2020-10-22

回忆目前为止我主要经历了三个项目，仔细归类的话，这三个竟然还都有所不同。

第一个项目，是我刚毕业时候进到项目组的，是一个Android App项目。之前App项目的主要业务逻辑是通过JavaScript编写的，Android端调用webview展现即可，所以很多流程都是依赖于其他小组。

刚好领导们觉得有必要全部原生化，就让我赶上了重构的时代，说是重构，其实就是对之前在JavaScript 上的业务迁移到Android原生上。

也很庆幸那个项目一开始做的就比较好，文档之类的相对来说也齐全，做主流业务的同事一直在项目组里，哪怕文档里没有的业务，自己总结起来抽时间麻烦他，也能得到自己想要的答案。

也许是注定需要给我一次锻炼的机会，在重构初始，我的师傅当时的Android端负责人因为身体原因休息了一个多月，就让我这个徒弟去接手了当时复杂的，核心的业务，得到了一次快速成长的机会。

就这样坑次坑次完成了我人生中第一次商业化项目的重构，因为项目很庞大，经历了几个月的加班加点，上架的时候狂松了一口气。几个月的努力终于看到了回报，因为前期准备的材料都很充分，这次重构对初入职场的我是很大的能力提升。

第二个项目，是我去新公司之后的项目，做一个即时通信的项目。

我来公司之前，有一个即时通信的在用，是基于Flash编写的，从响应速度和稳定性来说都没有让领导和用户满意，当时项目组里的后端都是PHP，领导想内部发掘Java的员工来，所

以，我的第二次能力提升就在这个时候出来了。

我到公司的时候是Android开发，因为当时项目组有3个Android，没有Java，领导在征求大家的建议，问有没有想转的，我分析了自己的情况之后，主动要求转Java，和老大一起去做这个即时通信框架。

我们两个选择Netty框架来进行，又一次加班加点的拼命时刻到来，这次没有文档，只有代码，也还是有幸运的部分，之前框架的负责人还在岗。

难点就是，我需要看懂Flash的代码，然后一步步迁移过去。不过也有一点遗憾，整个框架是老大搭建好的，我的核心任务是在计划的时间之内，完全迁移即时通信的功能，然后把项目跑通，调试上线完成。

所以还是有那么点遗憾，搭建框架的时候我没参与，但不妨碍我对Netty的理解，为此我还在付费购买了Netty学习的一个专栏，加深我对Netty的理解。

从客户端转到后端，给我最大的感触就是我看项目看的范围更大了，之前客户端只是很片面的看到自己所负责的功能，后端能把整个项目都看透了，尤其是业务方面的知识点。

给我最大的教训就是：后端编码和客户端还是存在不同，因为我的不熟练，在项目上线的第一个晚上，因为扛不住峰值的压力，把网站给瘫痪了，业务宕机了一个小时，也是老大帮我解决的，给我当头一棒，原来代码不是这样写的。这也是让我坚定，在做业务的同时，需要持续的精进自己的技术。

第三个项目，是最近几个月在做的，帮公司的合作伙伴半路救急。

一开始是在他们外包的项目上进行接口的新增，但是外包的项目估计收钱给代码之后就不来管了，让我在新增的时候做的很痛苦。没有文档，没有业务说明，只有一个勉强能用的App，以及一份新需求的文档提供过来，全程懵逼，这就是项目？

没办法，硬着头皮做呗。加了两个多月的班，完成了近40个接口的编写和调试，过程很痛苦，遇到问题问对方的负责人，他也只是一知半解，所以只能自己通过数据库，通过简单的注释进行编写，苦啊。

完成第一版之后，老大和我们着手进行重构，现在正在进行中。根据App提供出来需要使用的接口，从接口中重构代码，完善文档，把代码分层，轻便化。

因为项目复杂，又拉动了一个比我工作经验更丰富的C++转岗到Java，减轻了压力，也感受

到转岗同事的超强适应力，给我的警示就是争哥在加餐里说的：**觉得自己做得很好，领导不识货，同事都没他强**。我有部分符合，所以以后还需要虚心写代码，提升技术，冲冲冲

共 1 条评论>

17

Jackey

2020-07-10

刚接手一个这样的项目，没办法，也是读代码，梳理代码逻辑，然后再拉上产品对比很久不更新的文档，最后总结一个新的文档，同时把代码中不合理的地方重构掉。希望我们的文档可以持续维护

13

悠游

2020-07-11

最近接手的一个项目，真的是一堆烂代码，业务员逻辑又复杂无比，只能是硬着头皮去啃了

共 1 条评论>

9

业余爱好者

2020-07-10

这个专栏不仅可以学习知识和技术，还有端正态度，纠正“价值观”的“正念”效果。

9

微末凡尘

2020-07-17

年初加入一家公司，项目差不多有5 6年了，几乎没有任何的文档，熟悉该项目的产品经理，程序员几乎都离职了，唯一熟悉的老大每天又很忙，不能经常问他，实在是太痛苦了，不敢动，不敢改，数据库字段也没有注释，代码写的很随意，果断跳槽了，我之前写代码也特别不注意规范和命名，自从来了新公司，虽然公司不大，后端开发差不多就三个人，但是代码结构看起来非常的舒服，命名规范，老大还是对我写的代码进行CR，老大是一个非常注重细节和规范的人，对自我的提升还是挺大的，有时候工期紧张，保证代码规范还是有一定的难度的，都是先上线再说~

7