重构都停留在细枝末节上

产品名称	重构都停留在细枝末节上
公司名称	东莞市微三云大数据科技有限公司
价格	.00/个
规格参数	
公司地址	东莞市
联系电话	18665158422 18665158422

产品详情

经典著作《重构》这本书中有这么一段话：

一开始，我所做的重构都停留在细枝末节上。随着代码趋向简洁，我发现自己可以看到一些设计层面的东西了，这些是我以前理解不到的，如果没有重构，我达不到这种高度。

重构，着实是一件让程序员兴奋的事情。

今年年初，我们团队完成了一个复杂项目的重构工作，它属于广告系统核心的引擎部分，大概有 300 多个文件，3 万多行代码。

从技术方案设计到终全量上线仅仅花了 1 个月左右的时间，而且没有产生事故。

这应该是我 8
年程序生涯中，经历过的大型的同时成功的一次重构项目：速度足够快、计划比较周全、质量过关。

01 先聊聊这个系统的历史包袱

我们的广告引擎在这次重构前大概经历了1年半时间的迭代，初期针对的是搜索场景，业务单一，流程清晰。

2019年开始，公司的广告业务开始快速扩张，收入几乎是指数级的增长。在这个过程中，我们的广告引擎面临了两个挑战：

1、业务场景开始变得复杂，除了搜索广告，还需要支持信息流推荐以及相似推荐场景。

2、广告流量开始快速增加，除了满足功能性需求，还需要兼顾好性能。

经过梳理，整个引擎有大部分逻辑是可以公用的，因此我们定义了一个主体框架，同时将可扩展部分进行了抽象。这样，各个场景能够根据自身业务的特殊性实现某些公共接口即可。另外，从性能角度考虑，我们牺牲了一些代码可读性，把某些逻辑并行化了。

随着业务的发展，搜索场景开始进入快速迭代期，新增策略越来越多，我们的主体框架也是在这个时候逐渐变得不灵活。

如果动主体框架，搜索以外的场景都需要跟着重构。
在业务的快速发展期，工期根本不允许，因此我们只能在现有框架上进行补丁式的开发。
这样，带来了两个很明显的问题：

1、为了兼容搜索的特殊逻辑，我们需要在其他场景中增加各种 if 判断来绕过这些逻辑。

2、广告策略越来越多，累计了几十个，当框架失去清晰的结构后，有些策略的实现开始变得定制化，缺少层次化的划分和可插拔式的抽象设计。

在这样的背景下，随着改动的积累，代码开始偏离了设计的初衷，技术债务越来越重。但是，我们又始终找不到合适的时机进行重构。

转机出现在 2019
年年底，由于广告业务的特殊性，流量开始自然走低，另外产品运营团队将重心放在了第 2
年的工作规划上，因此给了我们非常好的窗口期开始此次重构。

我们将工期定成了 1 个月，终仅比预期晚上线了，虽然出现了两个线上问题，但是在灰度期都及时发现和修复了，并没有造成线上事故。

总体来说，这是一次难度颇大并且比较成功的重构项目，下面详细说一下我从这个项目中吸取到的宝贵经验。

02 重构前，我们做了哪些准备工作？

这次重构的代码量很大，3
万多行，而且是广告系统核心的引擎部分。启动前，我们能预期到下面这些困难：

1、业务侧的阻力：广告是极其以业务为导向的，本次重构虽然能带来长期研发效率的提升，但是没法直接提升业务收益，而且开发周期不会太短，如何才能得到业务同学的支持？

2、技术侧的顾虑 ：重构一旦引起线上事故，公司是有处罚制度的，如何让大家轻装上阵？同时，重构过程中如果还有非常重的业务迭代穿插，交付时间没人敢保证，质量也很难得到控制。

针对这两方的顾虑，我认为下面这几项工作起到了很关键的作用。

1、让所有人看到痛点

前面提到：随着业务迭代，我们广告引擎的主体框架已经变得模糊不清，另外几十个广告策略散落在不同的业务场景中，配置凌乱。

针对这两个痛点，我们提前1个月启动了现有业务的梳理，走读旧代码、同时翻阅以前的需求文档，终我们将不同场景的核心流程以及广告策略归类成了一张清晰的表格。

正是这一张表格，让技术和产品次很清晰地看到了我们引擎部分的全貌，体会到了业务的复杂度以及当前技术上的瓶颈。

2、明确重构的目标和价值

让所有人感受到痛点后，我们规划了本次重构的两个核心目标：

1、主体框架的重构：将主流程模块化，重新定义上下层协议，确保接口清晰；各层级内部也需要做好抽象，具备良好的扩展性。

2、策略灵活可配置：将广告策略按照业务意图进行归类抽象，策略的执行条件动态可配置，同时策略可任意插拔。

此外，我们将这两个核心目标完成后可带来的预期收益进行了细化：

1、技术收益：代码结构更清晰，更容易理解和维护；可扩展性增强，引擎的开发效率将进一步提升。

2、业务收益：策略能做到更细粒度的配置和扩展，对业务支持更友好；研发提效后能进一步加快业务的迭代速度。

将重构的价值同步给大家后，进一步提升了所有人的兴奋度，让大家有了更强的动力参与进来。

3、整体节奏的把控

整体节奏的把控也是非常重要的一环，能让所有人对这件事情有一个时间上的预期。

首先，我们将工期定成了 1 个月，一方面考虑了业务侧可以接受的大周期，技术上也希望速战速决；另一方面，春节即将来临，我们必须赶在公司封网前上线，同时预留出1-2周的 buffer 以防意外情况发生。

此外，我们和业务侧达成了一致：重构期间，引擎部分的非紧急需求一律不接，这样可大限度地减少并行开发和代码冲突，让团队精力更集中。

03 执行过程中有哪些可分享的经验？

这次重构能够实施得如此顺利，有 4 点我认为很有价值的经验跟大家分享下。

1、高质量的技术设计方案

这一点得益于日常的要求，针对开发周期超过3天的项目我们都会进行技术方案设计，本次重构当然也不例外。

框架部分的整体架构、模块之间的协议设计、以及策略的可扩展性设计是本次技术方案的重点，团队前后讨论了不下3次。

在大方案定稿后，团队进一步对数据库、接口字段、缓存结构、日志埋点等公共部分进行了细化，因为涉及到多人协作开发，团队约定以文档作为沟通界面，文档始终保持和代码同步。

在这样的高要求下，团队产出了 5000 多字的技术方案文档，合计 36
页，这些为整体质量的保障打下了很好的基础。

2、 预重构出框架性代码

这一个 PR 非常关键，是我们从技术方案落地到代码重要的一步。我们把重构后的包结构、模块划分、各层之间的API定义、不同广告策略的抽象进行了梳理，先忽略实现的细节。

这样主体代码基本成型，能很清楚地描绘出我们理想中的框架。然后，我们组织了多次集中代码审查，终形成了统一意见。

这一步能很好地避免过早陷入实现细节，导致主体框架关注不够、代码不稳固，后期再返工反而会拖累效率。

3、 频繁沟通和成对代码 Review 机制

进入到细节实现阶段后，很重要的一点是：对现有逻辑的理解。引擎代码经过一年半的迭代，历史上被很多人开发过，但是本次只有 3 个同学参与重构。

整个过程中，我们遇到任何代码逻辑不明确的地方，都是反复沟通和求证，不主观猜想，这一份谨慎其实很关键。

另外在代码审查上，我们按模块分配了对这块业务比较熟悉的同学来负责，成对搭配，机制灵活。

4、 有效的测试方案

重构未动，测试先行。这个原则是《重构》一书中重点强调的，也是我们本次技术方案讨论的重点，我这里单独拎出来详细展开下。

首先，我们前期便约定好：不动任何老代码，完全建新的 package
进行重构。这样方便比对重构前后的结果，同时进行线上灰度实验。

测试方案上，以下 4 点值得借鉴：

1、端到端测试：本次重构不涉及功能性的调整，因此外层API的行为是不会有任何变化的，这样端到端的测试方法为有效，这个是研发和QA测试主要的手段。

2、冒烟测试：QA同学提供冒烟 Case，由研发同学进行冒烟，研发提测前必须保证所有冒烟 Case 执行通过。这一点在大部分互联网公司都不常见，但是对于大型项目有效。

3、沙箱环境双流程验证：前面提到我们重构前后的代码都保留了，因此可以通过脚本抓取线上环境的入参作为case，然后用自动化的方式对 API 的返回字段进行逐一比对。

4、线上环境灰度实验：灰度对于重构非常重要，我们利用已有的ABTest平台，逐步放开灰度流量，从5%、到10%、到30%、后到，制定了很谨慎的放量节奏，然后通过日志以及业务指标监控进行验证。