# 边际成本不再是零

# 导读

大模型如何拆掉 SaaS 赖以成立的那条地基，以及新规则会长成什么样

## 导读：只拧一颗螺丝

关于"AI 会颠覆软件"的话已经太多了。热闹归热闹，很少有人能指着一处说清楚：到底颠覆的是哪根螺丝？

这本书不凑那个热闹。它只拧一颗螺丝，然后顺着这颗螺丝松动之后的裂缝，一路看下去——你会发现，动的从来不止一颗。

### 那颗螺丝是什么

过去二十年，SaaS（把软件做成在线订阅服务卖的生意）能繁荣、能高毛利、能撑起离谱的估值，靠的是一条几乎没人明说的地基：

> **软件写一次，多卖一份几乎不花钱。**

这就是边际成本近零——每多服务一个用户的额外成本，趋近于 `0`。多卖一份是纯利润，加上订阅按月印钞、用户懒得搬家，于是印钞机成立了。

大模型第一次把这条地基敲出了裂缝：每回答一个用户，都要真金白银地烧算力（术语叫 推理成本，按 token 计费）。**"多服务一个用户"重新开始花钱了。**成本从零，变成了正数。

这就是书名的意思——边际成本不再是零。听起来只是账本上一个数字从 0 变成了大于 0，可 SaaS 那套商业规则，正是架在"这个数字是 0"上面的。地基一动，上面的东西都得重新找平。

### 这本书不是什么

- 不是融资故事，也不吹哪家公司估值又翻了几倍；
- 不是"AI 会取代一切"的爽文，恐吓和亢奋都留给别人；
- 不是面面俱到的行业百科，不追求什么都讲一点。

它是**一条冷静的因果推理线**：一个变量变了，别的东西被迫怎么排。仅此而已。

## 贯穿全书的那个问题

> 当"多服务一个用户"重新要花钱，SaaS 二十年建立的商业规则——订阅、护城河、定价、组织——会怎样连锁重排？

顺着这个问题往下推，你会看到能力被拉平、界面开始贬值、护城河从"技术"迁到"位置、关系、责任"、定价从卖席位改成卖结果、毛利变薄、组织变小、谁死谁活……不同的章节，其实是同一颗螺丝松动后掉下来的不同零件。

## 读完你能带走什么

不是一堆要背的结论，而是**几把能自己称重的尺子**：

- **边际成本尺**——多服务一个用户，到底还花不花钱？
- **能力来源尺**——这个本事是产品自己的，还是从底层模型白捡的？
- **护城河迁移尺**——护城河还在"技术"上，还是已经挪到位置、关系、责任上了？

有了这几把尺，下一个热词扑面而来时，你能自己上手掂量，而不是被叙事牵着鼻子走。

## 谁适合读，怎么读

如果你对"它到底怎么运作"这件事天生好奇——尤其你是软件从业者、创业者，或投资人——这本书是写给你的。

大白话 全书 16 章，每章都能单独抽出来读。但它们是按因果顺下来的一条线：从头读，你拿到的是完整的推理链；跳读，你拿到的是能用的单件工具。两种读法都行，看你此刻想要哪一种。

下面，我们从那颗螺丝开始。

# 第一章 · SaaS 真正卖的是什么

你每天用的软件，几乎都长一个样：你交一笔月费，它给你一个网页或一个 App，你在里面打字、点按钮、拖来拖去。有人告诉你这叫「软件即服务」，英文缩写 SaaS。听起来像是一句解释，其实是一句咒语——它让你以为你懂了，其实什么也没说。

这一章我想干一件事：把这层漂亮的外壳拆开，让你看见底下真正撑着它的那根梁。我先把结论放在这儿，后面慢慢证给你看——

> SaaS 之所以是一门好到离谱的生意，跟它「跑在浏览器里」几乎没关系。真正的地基只有一句话：代码写一次，多卖一份几乎不花钱。

## 先说清楚「即服务」到底是什么意思

SaaS，全称 Software as a Service，字面是「软件当成一种服务来卖」。大白话讲：过去买软件像买一张光盘，一次性付钱，装在自己电脑上，坏了自己修；现在软件住在别人的服务器上，你按月或按年付费用它，厂商帮你维护、随时更新。你不拥有它，你订阅它。

大白话

光盘时代：你把软件搬回家养。SaaS 时代：软件养在厂商家里，你交房租进去用。

很多人以为，SaaS 的魔力就在「搬到云上」这件事上——不用装、随时能用、多方便啊。方便是真方便，但方便不赚钱。真正让投资人两眼放光、让一家几十人的公司能值上百亿的，是藏在方便背后的一条经济学规律。要看懂它，我们得先认识一个词，它会像一根线，串起这本书从头到尾的每一个故事。

## 本章的主角：边际成本

边际成本：你已经在做一件事了，再多做一份、多卖一个，额外要多花的那点钱。注意，是「额外多花的」，不是「总共花了多少」。

这个「多一份要多花多少」的问题，才是区分生意好坏的分水岭。我们拿两样东西对比一下，你立刻能摸到差别。

## 造一辆车 vs 多卖一份表格软件

一家车厂已经建好了工厂、雇好了工人。现在客户要多买一辆车，车厂得多花多少钱？钢板、玻璃、轮胎、电池、拧螺丝的工时、运到店里的物流——实实在在,一分都少不了。你多卖一辆,就得多买一辆的料。这些跟着产量一起涨的钱,就是这辆车的边际成本,而且它相当高。

再看一家做在线表格软件的公司。产品已经写好了、跑在服务器上。现在多来一个用户注册,公司要多花多少钱?几乎为零。这个新用户用的是同一份早就写好的代码,不需要为他重新写一遍;他占用的那点服务器和带宽,摊到头上是几分钱、几毛钱的量级。

> 大白话
>
> 多卖一辆车,你得再买一堆钢铁橡胶。多卖一份软件,你只是让同一份代码再被读一遍——代码不会因为多一个人用就磨损、就变少。

这就是软件最反常识、也最强大的地方：**它的边际成本接近于零。**制造业、餐饮业、几乎所有实体生意,都逃不掉「多做一份就得多花一份料」;而软件,把这条几千年的铁律直接踩碎了。

## 开餐馆 vs 加一个 Notion 用户

再换个能闻到味道的例子。一家餐馆今晚多来一桌客人,老板得多买菜、多用煤气、多请人洗碗。每多接一桌,成本就往上抬一截,抬到某个点,厨房塞不下、灶台不够用,他就接不动了。餐馆的产能有天花板,而且每一份产出都真金白银地烧料。

一款笔记软件多一个用户,情况完全反过来:不用多备一份纸,不用多雇一个人,那个人和第一个用户共用同一份代码、同一套服务器架子。从第一百个用户到第一百万个用户,产品这边几乎不用多做什么。软件没有「厨房塞满了」这回事——至少远比餐馆晚得多、缓得多。

## 数字有多夸张:毛利率

光说「差别很大」还不够,给你一个能量化的尺子:毛利率。它衡量你卖出去一块钱,刨掉「直接为这一份多花的成本」之后,兜里还能剩几毛。剩得越多,毛利率越高。

> 大白话
>
> 你卖一杯 10 块的奶茶,原料杯子花了 4 块,剩 6 块,毛利率就是 60%。它算的是「每一份自己养不养得活自己」,还没算房租、工资这些跟卖不卖都得付的大头。

差距大到什么程度?通常来说:

- **成熟的软件公司,毛利率常年在 75%–85% 这个区间**——卖一块钱软件,直接成本可能就一两毛。
- **而不少制造业,毛利率长期只有个位数到 20% 上下**——因为每一份产出都被原材料和人工实打实地吃掉一大块。

请注意,这里给的是行业里常见的量级,不是哪家公司某一年的具体财报——你不用记数字,你要记的是那个夸张的落差:同样收一块钱,一个能剩八毛,一个只能剩一两毛。剩下来的这部分,才是能拿去砸研发、砸市场、砸出下一轮增长的弹药。软件公司弹药多,不是因为它们更聪明,是因为它们的边际成本天生就低。

## 那份「几乎为零」不是天上掉的

说到这儿,得赶紧堵一个漏洞,免得你觉得软件是白捡的钱。第二份、第一百万份几乎免费,可**第一份贵得吓人。**

一款正经软件,常常是几十个工程师干上好几年的产物——工资、试错、推倒重来、反复打磨。这是一笔巨大的、沉甸甸的固定成本:不管你最后卖出去一份还是一亿份,这笔钱都得先花掉,而且花掉了就收不回来。

> 大白话
>
> 固定成本 = 开门之前就得先烧掉的钱,跟你后来卖多少无关。边际成本 = 开门之后,每多卖一份要额外添的钱。软件的怪就怪在:前者极高,后者极低。

所以软件生意的真实形状是这样的:一道极高的前期投入门槛,翻过去之后是一片几乎平坦的旷野——多卖十万份和多卖一份,成本差不了多少。这个形状,决定了这门生意后面所有的玩法。

## 为什么说这是「一切奇迹的源头」

现在把这条规律接到后面的章节上,你会发现它像地基一样,一层层往上顶。

1. **它让「订阅」变成印钞机。**既然多服务一个客户几乎不花钱,那把客户留住、让他年复一年地交月费,收上来的几乎全是净赚。这是第 2 章的事。
2. **它让规模变成压倒性的武器。**前期那道高门槛别人翻不过来,你翻过来了,之后每多一个客户你几乎零成本、对手却要从头烧一遍。护城河由此而来,那是第 3 章。
3. **它让「一份代码卖给全世界」成为可能。**复制不要钱,分发就没有物理上限,一个小团队的产品能同时服务几百万人。

你注意到没有:这三件事,没有一件是靠「技术多先进」。软件早就存在了几十年。真正在过去二十年里被彻底改写的,不是技术,而是**分发的经济学**——写一次、复制无数次、每一次复制几乎不要钱。这条「复制不要钱」的规则,就是 SaaS 一切商业奇迹的源头。

## 把这颗钉子钉进去

请你把这一章浓缩成一句话带走:

> SaaS 好赚,不是因为它跑在浏览器里,而是因为它的边际成本几乎为零——多卖一份,几乎不花钱。

这句话现在听起来像一条稳稳的地基,踩上去结结实实。但请你记住「边际成本」这四个字,因为整本书真正的戏,是从它松动的那一刻开始的。

前面我们反复说「多卖一份几乎不花钱」——那是因为代码被复制的时候,没有人在背后为你算一笔账。可如果有一天,你的软件每回答用户一句话、每处理一次请求,都有一台机器在别处实实在在地烧着电、跑着一次昂贵的计算,那个「几乎为零」还成立吗?

当大模型走进来,边际成本这根一直躺在零线上的曲线,第一次抬起了头。它从零,变成了正。这条地基怎么开始塌、塌了之后 SaaS 那套引以为傲的玩法哪些还灵、哪些成了包袱——是我们从第 4 章起,要一根螺丝一根螺丝拆给你看的事。

# 第二章 · 订阅制为什么是印钞机

## 第 2 章 订阅制为什么是印钞机

上一章我们拆到了地基：一份软件写好之后，再多卖一份，几乎不多花钱。这句话本身平平无奇，但它的下游后果非常凶猛。这一章我们就顺着这根链条往下走，看看「多服务一个人几乎免费」这件事，是怎么一步步长成一台印钞机的。

先给你一个能贯穿全章的画面：**把软件公司想成一个包租公**。它盖了一栋楼（写好了产品），接下来要做的不是把楼一次性卖掉，而是把房间一间间租出去，每月收租。楼已经盖好了，多住一个租客，水电损耗几乎为零。这就是接下来一切逻辑的起点。

### 为什么不卖断，而要按月收租

老式软件是买断的：你付一次钱，拿到一张光盘，装上就是你的了。这门生意有个天生的问题——每个客户一辈子只付你一次钱。你想再赚他，得逼他升级到下一个大版本，而他完全可以赖着不升级。

订阅制（subscription）就是把「一次买断」改成「按月或按年租用」：你不再拥有这份软件，你租用它，租金按周期持续付，停付就停用。

> 大白话
>
> 说人话：以前是把房子卖给你，一锤子买卖，钱货两清；现在是把房子租给你，你每个月给我打钱，哪个月不打就搬出去。

为什么软件公司敢这么干、也乐意这么干？正是因为上一章那个地基。既然多养一个租客几乎不花钱，那我最该在意的就不是「这一单赚多少」，而是「这个租客能住多久」。住得越久，我这栋盖好的楼就摊得越薄，赚得越多。买断制赚的是一次性的差价，订阅制赚的是**时间**。而时间，是可以复利的。

## 把租金加起来，就有了 ARR

当收入变成「每月都会到账的一笔租金」，投资人看这家公司的方式也就变了。他们不再盯着某个季度卖了多少套，而是问一个更要命的问题：**你现在手上这些租客，一年能稳定给你打多少钱？**

这个数就是 ARR（Annual Recurring Revenue，年度经常性收入）：把所有还在付费的客户的年租金加起来，得到的那个总额。它衡量的不是「过去赚了多少」，而是「只要大家不退租，未来一年能收上来多少」。

> 大白话
>
> 说人话：ARR 就是包租公算的「我手上这些房间，如果没人搬走，一年能收多少房租」。它是对未来的一个稳稳的预期，不是对过去的一次性统计。

关键词是**经常性**（recurring）——会重复发生的。一次性收入是「这个月卖了台冰箱」，明年这个月未必还有；经常性收入是「这个月收了房租」，只要租客还在，下个月照收。前者像打猎，运气好一顿饱；后者像收租，细水长流、可预测。可预测这三个字，后面会反复出现，它是整台印钞机的核心。

## 漏水的桶：churn

租金要能「持续」收，前提是租客别跑。跑掉的那部分，就是这门生意最致命的漏洞。

churn（流失率）指的是一段时间里，跑掉了多少客户（或多少收入）。年 churn 5%，就是一年里每 100 个客户走掉 5 个。

> 大白话
>
> 说人话：churn 就是水桶底下的漏洞。你从上面拼命往桶里灌水（拉新客户），桶底一直在漏水（老客户流失）。桶里的水位涨不涨，取决于灌得快还是漏得快。

好的 SaaS 公司，年 churn 常常压在个位数，甚至更低。别小看这个数字，它对长期结果的影响大得离谱。你直觉上会觉得，churn 从 10% 降到 5%，无非是「漏得慢了一半」。但这里藏着复利，长期差距是天差地别的——我们下一节就把这层反直觉掰开。

## 反直觉的核心：为什么留存决定一切

先立一个概念。把「一批同时进来的客户，随着时间推移还剩多少」画成一条曲线，就是 留存曲线：横轴是时间，纵轴是这批人里还有多少还在付费。churn 高，曲线掉得快；churn 低，曲线掉得慢、拖得长。

现在把上一章的地基接回来，见证一下复利。假设一个客户每年给你 100 块租金，而多服务他一年你几乎不花钱——也就是说，这 100 块**几乎全是纯利，而且是叠在往年之上的纯利**。

- 如果年 churn 是 **10%**，一个客户平均能陪你大约 10 年。
- 如果年 churn 降到 **5%**，平均陪你的年头大约翻倍到 20 年。

churn 只是从 10% 挪到 5%，看起来只差 5 个百分点，一个客户带来的总租金却差不多翻了一倍。原因就在于：因为多留一年几乎不多花成本，所以每多留住一年，收上来的租金几乎原封不动落进利润里，一年年往上叠。留存曲线的尾巴拖得越长，尾巴下面那块面积——也就是一个客户一辈子给你的钱——就大得越夸张。

**churn 低一点点，长期价值天差地别。**这不是修辞，是复利算出来的结果。这也是为什么 SaaS 公司把「降低 churn」「提高留存」挂在嘴边到近乎偏执——省下的每一个客户，赚的都是往后很多年的复利。

## LTV 和 CAC：赚回来要几年

顺着上面那条曲线，自然就有两个配套概念。

LTV（Lifetime Value，客户终身价值）：一个客户从进门到离开，一辈子总共给你贡献的钱。它就是留存曲线尾巴下面那块面积——租金越高、留得越久，LTV 越大。

CAC（Customer Acquisition Cost，获客成本）：为了拉进来这一个客户，你花了多少钱——广告费、销售的工资、请客吃饭，全算进去。

> 大白话
>
> 说人话：LTV 是这个租客一辈子给你交的总房租；CAC 是你为了把他招进来花掉的中介费和吆喝钱。生意划不划算，就看这一辈子的房租，够不够填掉当初招他的成本，还能剩多少。

业内常用 **LTV/CAC** 这个比值来体检：赚回来的是花出去的几倍。**一般认为大于 3 算健康**——花 1 块钱获客，这个客户一辈子给你赚回 3 块以上。低于这个数，说明你为了拉客花得太狠、或者客户留不住，桶漏得太快。注意，这个比值好不好，几乎完全被 churn 攥在手里：churn 一低，客户留得久，LTV 立刻变大，比值跟着好看——churn 这一个变量，牵着后面所有的账。

### 所以投资人到底在给什么估值

现在可以回答本章最开头那个问题了：为什么资本市场看 SaaS，盯的是 ARR 和 churn，而不是某个季度赚了多少？

因为对一家订阅制公司来说，某个季度的利润几乎没有信息量。真正决定它值多少钱的，是两件事：**手上有多少经常性收入（ARR），以及这些收入漏得快不快（churn）。**这两个数一确定，未来很多年的租金就大致可以推算出来——而且因为多服务一个客户几乎不花钱，这些未来租金里绝大部分是纯利。

换句话说，投资人买的不是这家公司这一季的利润，而是**它未来很多年那条可预测、还在复利增长的租金流**。可预测（churn 低、收入经常性）+ 可复利（多服务不花钱）——这两条一叠加，估值就被推得很高。这就是为什么一家当季还在亏钱的 SaaS 公司，市值能高得吓人：市场给的价，是对它未来十几年租金的贴现，不是对它这一季账本的奖励。

> 买断制卖的是一件商品，估值看的是利润；订阅制卖的是一段关系，估值看的是这段关系能拖多久、漏多慢。前者是打猎，后者是收租。

### 别忘了那块地基

把这一章的链条收一下：多服务一个客户几乎不花钱（第 1 章的地基）→ 所以不如把软件租出去、按月收租（订阅制）→ 于是收入变成可预测的经常性收入（ARR）→ 于是最要命的变量成了客户漏不漏（churn）、留多久（留存曲线）→ 于是一个客户一辈子的价值（LTV）远大于当初的获客成本（CAC）→ 于是资本市场愿意为这条复利的租金流，付出极高的估值。

每一环都扣得很紧。但请你盯住这条链子最上面那一环：**整台印钞机能转，前提是「多一个客户几乎不多花钱」。**纯利之所以能一年年叠、留存曲线的尾巴之所以那么值钱、churn 降

一点点之所以能撬动那么大的长期价值——全都建立在这个假设上。

那么问题来了：如果有一天，多服务一个客户开始实实在在地花钱了呢？如果每留住他一年，你都要为他持续掏出一笔成本呢？这条复利的链子，会从哪一环开始断？

这正是大模型动的那根螺丝。我们会在**第 4 章**回到这里，看看当边际成本从零变正，这台印钞机的地基会发生什么。先记住这台机器现在有多好用——因为接下来要拆的，就是它。

# 第三章 · 护城河的老配方

## 护城河的老配方

前两章我们算了一笔账：SaaS 真正的秘密是「代码写一次、多卖一份几乎不花钱」，于是订阅制变成印钞机。但这里有个问题——如果多卖一份不花钱，那对手多卖一份也不花钱。零边际成本是把双刃剑：它让你躺着收租，也让别人可以低价把你抄死。

那么在过去二十年里，为什么大多数成熟 SaaS 没被抄死？为什么 Salesforce 卖着一堆功能你都能列出来的表单和字段，却能收你一年几百万，而且你还真就不换？

答案是护城河——就是城堡外面那圈水沟的本义：让攻城的人过不来。搬到软件生意上，它指的是一种**结构性优势**，不是「我做得比你好一点」，而是「就算你做得比我好，你也追不上、抄不动、挖不走」。

> 大白话
>
> 护城河不是「我跑得快」，是「你追我这条路上有坑」。产品好不好是速度问题，护城河是地形问题。地形对手改不了。

老配方里有三味主药：数据网络效应、切换成本、集成锁定。这一章我们把这三样一个个拆开，看清楚它们当年为什么牢不可破。看清楚了，你才明白第四章开始大模型到底在拆哪根梁。

## 数据网络效应：越用越准，后来者追不上

先说最玄乎、也最被滥用的一个词。

数据网络效应：用户越多 → 你收集到的数据越多 → 产品越好用 → 又吸引来更多用户，转成一个自我加强的圈。注意它和普通「网络效应」不一样。普通网络效应是「人多所以有人跟你聊」（微信、电话），价值直接来自别的用户。数据网络效应绕了一道弯：别人用你，不是为了和这些人打交道，而是因为这些人的**行为数据**把产品喂得更聪明了。

举个工程师熟的例子。你做一个信用卡风控（判断一笔交易是不是盗刷、要不要拦下来）系统。你的模型好不好，取决于它见过多少真实的盗刷样本。你服务的商户越多，跑过你系统的交易越多，被标记为「这是盗刷」的确认样本就越多，你的模型判得越准。判得准，商户损失少，口碑好，更多商户来接你。

现在来了个后来者，代码写得比你还漂亮，架构比你新。他能追上你吗？追不上。因为他手里没有那几千万条带标注的历史交易。他冷启动的第一天，模型是「瞎」的——一个没见过盗刷的风控系统等于没有。而这个差距不会随时间缩小，反而会拉大：你每天还在吃新数据，他起步就落后，你跑得只会比他快。

> 大白话
>
> 这就是先发的狠处：不是我先到，是我先到之后每天都在囤别人囤不到的弹药。你想抄我的代码可以，代码不值钱；你抄不走我攒了五年的那堆数据。

推荐系统、搜索排序、反欺诈、地图路况——凡是「越多人用越准」的地方，都有这道河。它的地基是一句话：**关键数据攒在我这，而且只攒在我这。**记住这句，第五章我们会回来看它。

## 切换成本：明明不满意，也懒得换

第二味药，可能是三味里最被低估、却最实在的一味。

切换成本（switching cost）：你要把一个软件从生产系统里拆下来、换成另一个，得付出的全部代价。注意这里说的不是「新软件的订阅费」，而是**换掉旧的这个动作本身有多痛**。

为什么这道河深？因为一个用了三年的 SaaS，早就不只是「一个网页」了，它至少缠着你三样东西：

- **数据被它攒住了。**你三年的客户记录、工单、报表全在它库里，格式是它的私有格式。要搬走，得导出、清洗、映射字段、对账——工程师都知道数据迁移是个吞人的黑洞，导一半发现历史脏数据能让你怀疑人生。
- **人被它训练过了。**公司里两百号销售，每人都记得它那套操作路径，肌肉记忆都长好了。换一套，等于两百人重新培训、重新踩坑、生产力先掉一个季度。

- **流程被它固定住了。**你的审批、你的报销、你的季度结算，都是照着它的字段和按钮设计出来的。换掉它，很多流程要重画。

把这三样加起来，你就得到一个反直觉但每天都在发生的现象：**客户对产品不满意，但就是不换。**

> 大白话
>
> 不是它好到让人离不开，是拆它太疼。就像租的房子你嫌小，但一想到打包、搬家、重新办网、重新记路——算了，忍着。切换成本卖的不是「留下来的爽」，是「离开的痛」。

这里有个关键因果，别搞反：切换成本高，意味着厂商可以在你不满意的情况下继续涨价，只要涨价的痛还没超过搬家的痛，你就走不了。这就是为什么很多老牌企业软件用户体验烂得出名，还能年年提价——它赌的不是你爱它，是你懒得走。

大 CRM（管客户关系的系统，比如销售跟单、客户资料）和 ERP（管企业内部资源的系统，进销存、财务、供应链拧在一起那种）是切换成本的极致。一个大公司换 ERP，往往是几年、上千万、外加一个专门的项目团队的工程。很多公司宁可忍着二十年前的老系统，也不敢碰这个雷。这不是产品有多好，这是河有多深。

## 集成锁定：拔一个，牵一串

第三味药，是切换成本的放大器。

集成锁定：一个软件通过 API（系统之间对话的接口，A 调 B 的数据、B 调 A 的功能）深深嵌进你的工作流和别的系统里，嵌到你想拔它的时候会发现它牵着一大串别的东西一起动。

单个软件的切换成本已经不小了。但真实企业里，软件从来不是一座孤岛。你的 CRM 通过 API 把成交数据喂给财务系统，财务系统把发票状态推回 CRM，CRM 又触发营销工具发邮件，营销工具的打开数据再回流做客户画像……几十上百个系统用 API 互相缠在一起，织成一张网。

现在你想换掉中间那个 CRM。麻烦来了：所有连着它的那几十根 API 管子，得一根根拆下来、重新对到新系统上。新系统的接口长得不一样、字段对不齐、认证方式不同——每一根

都是一个小项目。你换的不是一个软件，你是在给一张活着的网做心脏移植，而且不能停机。

> 大白话
>
> 一个软件用久了，会像藤蔓一样把根须扎进你所有别的系统里。你拔它，扯的是一整片。工程师最怕的那句「这块动不得，动了不知道哪儿会炸」，说的就是集成锁定。

集成越深，河越宽。所以聪明的 SaaS 拼命鼓励你接它的 API、装它的插件、用它当各种流程的中枢——每接一根管子，都是往护城河里再灌一桶水。等你醒过来，它已经是你系统里那个「动了会炸」的核心节点了。

## 三味药是一副方子

回头看，这三样其实是彼此加固的：

1. **数据网络效应**让你先攒住数据、后来者追不上——这是*进攻方追不上*。
2. **切换成本**让已经上船的客户不满意也懒得下船——这是*守方客户走不掉*。
3. **集成锁定**把切换成本从「换一个软件」放大成「换一整片系统」——这是*把走不掉再乘上十倍*。

一个成熟 SaaS 三样都占齐了，它就基本上是攻不动的城堡：新对手代码再好也没数据、老客户再不爽也不敢拆、想拆还发现拔一个牵一串。二十年来，这就是 SaaS 生意「护城河牢不可破」的全部秘密。

## 但这副方子有个隐藏前提

现在，请你盯着上面这三味药，问一个问题：它们各自是**建立在什么假设上**的？

- 数据网络效应能成立，前提是——**核心能力来自你自己攒的数据**，别人拿不到就做不出好产品。
- 切换成本能成立，前提是——**界面和操作路径是你自己的**，用户重新学、数据重新导，才叫痛。
- 集成锁定能成立，前提是——**软件是你自己写死的一套接口**，别人对不齐、接不上。

换句话说，整副老配方都站在同一块地基上：**软件是你自己写的、界面是你自己的、数据攒在你这里。**这三件事在过去二十年是理所当然的常识，没人会去质疑。

而大模型来了之后，动的恰恰就是这三块常识。当核心能力可以从一个通用大模型里现取、不必自己攒数据；当界面从「你设计的按钮」变成「用户直接说人话」；当接一套系统不再需要死磕私有接口——这三味药会一味一味地开始漏水。

> 老护城河不是被填平的，是脚下的地基被换了。城堡还在，只是护城河里的水，正在从别处偷偷流走。

具体是哪根螺丝先松、每道河怎么漏、漏了之后新的沟该往哪挖——从下一章「边际成本从零变正」开始，我们一根一根拧给你看。

# 第四章 · 大模型动了哪根螺丝

前三章我们拆开了 SaaS 这台印钞机，看清了它靠什么运转：一份软件写好了，第二个用户、第一千个用户、第一百万个用户，服务他们几乎不额外花钱。这就是零边际成本（多服务一个用户，你要多掏的钱几乎是零）。订阅制、超高毛利、离谱的估值——全都是长在这块地基上的房子。

这一章，我们要看的是这块地基上第一次出现的一道裂缝。它不是又一次「技术升级」，不是数据库换了引擎、前端换了框架那种。它撬动的是「多服务一个用户几乎不花钱」这个前提本身。

先把话说死：**大模型让「多服务一次」重新变得要花钱了。**剩下的整章，都是在解释这句话为什么成立，以及它为什么这么要命。

## 查字典 vs 请专家现场思考

想理解裂缝，得先分清两种「计算机在干活」。

传统软件处理一个请求，本质是**查表和搬运**。你在淘宝点开一个商品，服务器做的事大致是：按 ID 去数据库里把这条记录捞出来，套进一个页面模板，发回给你。这中间没有「思考」，只有「查找」和「拼装」。这类操作快得惊人，一台普通服务器一秒钟能处理成千上万次。

> 大白话
>
> 说人话：传统软件像查字典。字典早就印好了，你要查「边际」这个词，翻到那一页念出来就行。翻一次和翻一万次，边际的力气几乎一样，纸也不会因为你多查一次就磨掉。

大模型干的事，是另一回事。

大模型（LLM，读了海量文本、学会预测下一个词，从而能读会写会推理的 AI）在回答你的时候，不是去某个表里把答案捞出来——答案根本不在任何表里。它是**现场算出来**的。你问它一句话，它要把你这句话拆成一串数字，然后让这串数字穿过一个有几百亿甚至上千亿个

参数的网络，一层一层做矩阵乘法，最后才挤出下一个词。然后把这个词接回去，再算一遍，挤出下下个词。如此反复，直到答案吐完。

> 大白话
>
> 说人话：大模型不像查字典，像请一个专家坐你面前现场思考。你问一句，他真的要动脑子——而且他每吐一个字，脑子就得从头到尾再转一圈。你付的不是「查一次」的钱，是「他思考了多久」的工时。

## 为什么现场思考这么贵

这里要引入一个词。模型每跑一次、真正吐出答案的这个过程，叫推理（inference，模型运行一次、实际生成答案的过程；注意它和「训练」不是一回事——训练是一次性地把模型教出来，推理是它之后每次干活）。

很多人以为大模型的钱都花在训练上：花几千万训一个模型，训完不就一劳永逸了吗？训练确实贵，但那是一次性的。真正会持续、无限次、每个用户每句话都要重来一遍的成本，是**推理**。

推理贵在哪？三个实打实的东西：

- **算力**：那几百上千亿个参数的矩阵乘法，跑在一种叫 GPU（专门做大规模并行数学运算的芯片）的昂贵硬件上。一张这样的卡本身就是天价，而且一次对话往往要占用不止一张。
- **电**：矩阵乘法是实打实地烧电、发热。数据中心为此要供电、要散热，这些都是每小时都在跳的电表。
- **时间占用**：专家在给你思考的时候，就没法同时给别人思考。这张昂贵的卡被你这次请求占住了，占的每一毫秒都是钱。

所以，模型每回答你一次，背后是一批昂贵的芯片实打实地转了一阵、烧了一阵电。**这笔钱，不会因为你是第一百万个用户就消失。**它对每一次请求都重新收一遍。

## 计价的那把尺子：token

这笔成本怎么量？行业用的单位叫 token（模型处理文本的最小计价单位，大致是一个词或半个词；你发进去的和它吐出来的，都按 token 数算）。

为什么用它计价，而不是「按次」？因为模型的活儿，前面说了，是一个字一个字挤出来的——你给的上下文越长，它要读的 token 越多；它答得越长，要生成的 token 越多；而每一个 token 都对应着「网络从头到尾再跑一圈」的算力。所以 token 数几乎直接正比于它烧掉的算力。**token 就是那把量「这次思考花了多少工时」的尺子。**

大白话

说人话：token 就是给专家的思考按字数计费。你问得啰嗦、要求他读一份长文档、要他写一篇长回答，字数一多，工时费就往上走。查字典没有这回事——翻到哪个词都一样便宜。

关键的一步来了。把它和第一章对照着看：

> 传统 SaaS：多服务一个请求 ≈ 多查一次字典 ≈ 几乎免费。
> 大模型：多服务一个请求 = 多请专家思考一次 = 每次都要付一笔 token 的工时费。

这正好是零边际成本的反面。SaaS 的地基是「多卖一份不花钱」，而大模型的机制是「多服务一次，就要真金白银付一笔」。

## 量级差多少：给点直觉

不报任何公司的具体单价——那些数字每季度都在变，报了反而误导人。但量级的直觉值得建立，因为差距大到会改变生意的形状。

粗略地说：

- 一次普通的 SaaS 请求（查条记录、渲染个页面），摊到的服务器成本可能是**几分之一分钱，甚至更低**。小到你根本不会去想它。
- 一次像样的大模型交互（一段对话、读一份长文档、生成一篇东西），成本可能是**几分、几毛，甚至更多**。

这中间的量级差，往往是**成百上千倍**。更要命的是，它不是固定的——用得越重，越贵：

- **上下文越长越贵**：让模型读整份合同、整个代码库，输入 token 就翻着番涨。
- **多轮对话越贵**：每问一轮，前面的对话往往要重新喂一遍当上下文，越聊越长，越长越贵。

- **agent 最贵**：让模型自己规划、自己反复调用工具、自己检查再重试（这类会自主多步行动的用法叫 agent），一个任务背后可能是它自己跟自己来回跑了几十上百次推理。用户按一个按钮，账单在后台翻了几十倍。

换句话说，成本不再是一个可以忽略的常数，而是一个**随使用强度往上爬的变量**。你越是把产品做得好用、做得「聪明」，每个用户越可能替你多烧钱。这在 SaaS 的世界里是不可想象的——那边是用户用得越多、你越赚，因为边际成本是零。

## 为什么这一下砸中的是地基

现在把镜头拉远，看清这道裂缝到底裂在哪。

SaaS 二十年的整套奇迹——订阅能印钞、毛利能到八九成、估值能给到营收的几十倍——追到根上，全都站在同一个前提上：**服务下一个用户的成本约等于零**。正因为约等于零：

- 收上来的订阅费几乎全是毛利，所以印钞；
- 用户用得再狠，成本也不涨，所以敢包月不限量；
- 规模越大越便宜，所以先烧钱换增长、后收割是理性的；
- 护城河可以建在网络效应、切换成本上，而不必操心「每一单还赚不赚钱」。

这四条，每一条都默默假设着「边际成本 = 0」。大模型做的事，就是把这个假设从 0 挪到了一个**正数**——而且是一个会随使用往上爬的正数。

> 大白话
>
> 说人话：前三章那套印钞逻辑不是错的，但它有个没写出来的前提——「多服务一个人不要钱」。大模型第一次让这句前提不成立了。地基上那条最粗的承重线，被动了。

这就是为什么它不是「又一个技术升级」。升级换的是房子里的家具、水管、电路；而这一次，动的是房子底下那块混凝土。地基一旦从「零成本」变成「每服务一次都要花钱」，上面盖的所有东西——订阅怎么收、毛利算不算得过来、护城河建在哪、价格该怎么定——都得重新受力，重新算一遍。

## 裂缝已经点破，接下来

这一章只干一件事：把裂缝指给你看。核心就一句——**推理是要花钱的，token 有成本，边际成本第一次从零变成了正数。**SaaS 赖以成立的那条地基，就在这里被撬开了一道缝。

缝已经在了。顺着它往下，一连串问题会自己冒出来：包月不限量还扛得住吗？八九成的毛利还保得住吗？如果能力不再来自你自己写的代码，护城河该往哪儿搬？卖结果的人，谁来替这笔越滚越大的成本买单？

这些，是后面几章要一格一格顺着裂缝推下去的事。这一章，我们先站在裂缝边上，把它看清楚——地基裂了，而且回不去了。

# 第五章 · 能力不再来自你的代码

上一章我们拧松了第一根螺丝——**成本**。多服务一次，从「几乎不花钱」变成了「每次都要付一笔」。那是被动的、藏在后台的一根螺丝，用户看不见，但账房先生一算就疼。

这一章拧第二根，同样被动，同样藏在你看不见的地方，但它动摇的东西更根本——它动的是你这款软件的**能力从哪来**。

先把这一章要证的那句话钉在墙上：**过去，一款软件有多强，约等于你自己写了多少代码；现在，让软件显得「聪明」的那部分能力，是你没写、也写不出的。**剩下整章，都是在解释这句话为什么成立，以及它成立之后，自研代码的价值到底还剩多少。

## 过去：产品能力 ≈ 你写的代码量

回想大模型之前，一款软件的能力是怎么来的。答案朴素得近乎无聊：**你的团队实现了多少功能，产品就有多强。**

要做一个能读懂用户评论、自动判断好评差评的功能，你得雇一队人，标数据、调模型、写一大坨代码，磨上一两年。要做一个能把长文档总结成三行的功能，同理，又是一支队伍、又是若干人年。人年（一个人干一年的工作量，工程里用来量一件事有多大）是这套世界观里的硬通货：能力是拿人年一铲一铲堆出来的。

这里藏着一个当年天经地义、现在要被推翻的等式：

> 功能越难 = 要堆的人年越多 = 对手要追的时间越长 = 你的护城河越深。

这就是第三章那副老配方的底层逻辑。你花三年写出一个别人写不出的复杂引擎，这三年本身就是墙。对手就算今天开始抄，也得从头再走三年——而这三年里你没闲着，还在往前跑。**难，是护城河的原料。**谁能把难的东西做出来，谁就把对手挡在了三年之外。

大白话

说人话：过去，最聪明的功能都是最贵、最难写的功能。正因为贵、因为难，对手才追不上你——难度本身就是那圈水沟。你的代码写得越硬核，墙就越高。

## 现在：最「聪明」的那部分，是租来的

大模型把这个等式从中间劈开了。

那些过去最难写、最能当墙用的能力——理解一句话到底什么意思、把一长篇总结成几句、写一段通顺的文案、顺着线索推理、甚至写代码——今天不用你写了。你去接一个 通用大模型（general-purpose model，一个什么领域都能干的模型；它不是为你的场景专门训练的，但你的场景要的那点「聪明」，它顺手就有）的 API，一句调用，这些能力当场就到你手上。

关键在「通用」这两个字。它不是给你定制的，正因为不是给你定制的，**它也同样给你的对手。**你和对手接的是同一个模型、同一个 API、同一套能力。你们俩产品里那颗最聪明的脑子，是同一颗，从同一个地方租的。

请把这一下和上一段的老等式对照着看，看清楚断在了哪：

- 过去：难的功能 → 你花三年写 → 对手要追三年 → 这三年是你的墙。
- 现在：难的功能 → 你接一行 API → 对手也接一行 API → 谁都没墙，起点一样平。

三年的差距，被一行 API 抹平了。不是你变强了，是**那道原本靠「难」堆起来的墙，塌了**——对所有人一起塌。你能现取的聪明，对手也能现取。难度不再稀缺，也就不再是护城河的原料。

大白话

说人话：过去比谁写得出那个难功能，比的是内功，练三年才有。现在这门内功挂在墙上明码标价，谁都能当场买一份一模一样的。你练的三年，白练了——不是没用，是不再独一份了。

## 能力平权：门槛塌了

把这件事推到极致，就是一个值得单独起名的现象。

能力平权：过去要一整支专家队伍、砸下大把人年才做得出的能力，现在一个小团队接上大模型，当场就有了——能力的门槛，从「少数人翻得过」塌成了「几乎人人都能迈过去」。

具体到语言这件事上尤其刺眼。「让机器读懂人话」这个方向，过去有个专门的学科叫 NLP（自然语言处理，专门研究怎么让计算机理解和生成人类语言）。要做出像样的语言能力，你得招一屋子这方向的博士，磨好几年。这道门槛，二十年来把绝大多数团队挡在了门外。

今天，一个三人小队，接上通用大模型，写几行调用代码，一个下午就拥有了过去要几十个 NLP 博士才做得出的语言能力——理解、总结、改写、问答，全都开箱即用。那道挡了二十年的门槛，不是被谁翻过去了，是**地基被抽掉、整道门塌平了**。三个人和三百个人，在「语言能力」这条起跑线上，第一次站到了一起。

> 大白话
>
> 说人话：过去「机器懂人话」是名校博士才碰得动的绝活，普通团队想都别想。现在这门绝活变成了水电煤——拧开就有，谁家都通着同一根管子。绝活一旦变成水电煤，它就不再是谁的本事了。

## 但别急着下结论：租来的是「聪明」，不是「你的业务」

说到这，一个工程师该警觉了：照这么说，自研代码岂不是一文不值，大家都成套壳的了？

没那么快。这里必须辩证，否则就滑进了另一个坑。你从 API 租到的，是**通用的聪明**——它会读会写会推理，但它对**你的业务**一无所知。它不知道你那行的规矩，不认识你的客户，没见过你后台那堆数据，也不担保它吐出来的东西一定对、一定能用。租来的是一颗聪明的脑子，可这颗脑子是空的，还时不时说胡话。

于是有价值的自研，就从「实现聪明」挪到了「模型给不了的那些地方」。至少这么几块，模型租不来，还得你自己攒、自己写：

- **你的专有数据。**通用模型没见过你后台那些独家数据。这正是第三章说的那道数据的河——它没被填平，反而更值钱了。

- **你对具体工作流的理解。**模型懂「语言」，但不懂你这行到底怎么干活、每一步卡在哪、什么叫做对了。这份理解得你自己钻进业务里磨。
- **把不可靠变可靠的工程。**模型会说胡话、会算错、会跑偏。围着它建一圈评测（拿题去考模型、量它到底对不对）、纠错、约束的工程，让一颗时灵时不灵的脑子交付出稳定可用的结果——这活儿模型自己干不了，全靠你写。
- **分发和信任。**用户凭什么找到你、凭什么把要紧的事交给你。这跟模型多聪明没关系。

这几块具体怎么变成新的护城河、套壳到底有没有价值——是第 7 章和第 9 章的正题，这里只点到，不展开。你先记住这一章的分寸就够了：**塌掉的是「靠难度堆起来的墙」，没塌的是「模型给不了的那些东西」。**

## 换个厨房想：大厨不再是壁垒

这一整章的逻辑，用一个餐馆的比方能一口咽下去。

过去开餐馆，老板最好自己是个好厨师，或者能养住一个别人挖不走的大厨——**厨艺就是护城河**。你店里的招牌菜别家做不出，客人就只能来你这。这门手艺练十年，对手追十年，跟「花三年写个复杂引擎」是一回事。

现在变天了：最好的那位「大厨」（大模型）站在街口，明码标价，谁都能雇，而且雇到的是同一个。你雇他，对面那家也雇他，做出来的菜一样香。**厨艺，一夜之间不再是壁垒了。**

那餐馆之间还比什么？比选址（分发）、比菜单和口味的定位（你对业务的理解）、比供应链（你的专有数据）、比上菜稳不稳定、服务好不好（把不可靠变可靠的工程）。**厨艺不再是墙，别的东西才是。**能雇到同一个大厨，恰恰逼着大家去比拼那些大厨给不了的东西。

> 大白话
>
> 说人话：当全世界最好的厨子谁都能雇，「会不会做菜」就不再是本事了。真正拉开差距的，反倒是那些跟厨艺无关、你得自己一点点攒的东西。

## 这一章拧到哪了

收个尾。第二根螺丝，拧完了。

核心就一句：**软件的核心「聪明」，第一次不再来自你自己写的代码，而是从一个人人都能接的通用大模型里租来的。**难度被抹平，门槛塌掉，你和对手在「聪明」这条起跑线上被拉回了同一个起点。三年的墙，一行 API 就翻过去了。

但也别读偏了：租来的只是通用的聪明，不是你的业务。真正还值钱的自研，从「实现聪明」搬到了模型给不了的地方——数据、工作流、可靠性工程、分发。

第 4 章那根「成本」螺丝、和这一章「能力来源」螺丝，是两根一起松的承重螺丝。它们松开之后，一个更直观、用户当场就能看见的变化，正憋着要冒出来——当聪明是租来的、当机器真的听得懂人话，你辛辛苦苦设计的那一屏按钮、菜单、表单，还有存在的必要吗？下一章，我们从「点按钮」讲到「说人话」。

# 第六章 · 从点按钮到说人话

## 从点按钮到说人话

前面两章拆了两根螺丝：软件的能力不再只来自你写的代码（第 5 章），边际成本从零变正（第 4 章）。这一章拆第三根，而且这根最贴近用户的手指——**交互方式**。过去几十年，我们跟软件打交道的方式是点按钮、拉菜单、填表单；现在越来越多是直接说人话。别小看这个变化，它动的不是外观，是护城河。

先说清楚这一章要争的那个点：界面（也就是 UI，你在屏幕上看到并操作的那一堆菜单、按钮、面板）曾经是软件最值钱的资产之一。而现在，它正在贬值。更要命的是，它贬值的方式，恰好把第 3 章讲的那条老护城河——切换成本（换一个软件要付出的重新学习、迁移、适应的代价）——一起漏掉了水。

## 界面曾经是壁垒，不是装饰

工程师容易把 UI 当成"套在功能外面的皮"。这是低估了它。在 GUI 时代，界面本身就是产品价值的一大块，也是壁垒的一大块。

想想你最熟的那个专业软件——Excel、Photoshop、你司那套内部 ERP。它的价值有多少藏在"功能"里，有多少藏在"你练了三个月才顺手的操作方式"里？菜单藏在哪一层、哪个快捷键对应哪个动作、什么工作流该先点哪儿后点哪儿——这些你早就形成了**肌肉记忆**。

> 大白话
>
> 肌肉记忆就是：你不用想，手指自己知道该去哪儿。就像老司机换挡不用看挡位。这种熟练是真金白银练出来的，一旦练成，你就懒得从头再来。

这份熟练，从用户角度是"顺手"，从厂商角度就是切换成本。你会 Excel 的三百个快捷键，让你换一个功能一模一样但布局全变的表格软件，你会本能地抗拒——不是因为新软件差，是因为你不想再当一次新手。UI 越复杂、越"专业"，这层锁就越死。这不是 bug，这是过

去二十年很多软件公司心照不宣的护城河设计：**让用户在你的界面里投入得越多，他就越走不掉。**

换句话说，复杂的界面是一座迷宫。你花力气把迷宫走熟了，这份"走熟"本身就把你留在了原地。厂商乐得如此。

## 自然语言把迷宫削平了

现在来了自然语言交互——你不再点菜单，而是直接用日常说话的方式告诉软件你要干嘛。

> 大白话
>
> 过去："这三张表要合并……菜单在哪来着？数据 → 合并计算 → 选区域 → 按月分组……异常值怎么标红来着？条件格式 → 新建规则……"。现在：直接打一句"把这三个表按月份合并，标红异常值"。

看出关键了吗？迷宫没了。你不再需要知道"合并"这个功能藏在第几层菜单，也不需要记住标红的操作路径。界面从一个**你要征服的迷宫**，变成了一个**你一句话下达的意图**。

因果链条很直：

- 你不用学菜单结构 → **学习成本**大幅下降；
- 学习成本下降 → 你在这个软件里"练出来"的那份熟练不值钱了；
- 那份熟练本来就是切换成本的主要来源 → **切换成本**跟着下降；
- 切换成本是老护城河 → 护城河漏水。

过去锁住你的是"换软件要重学"。可如果新旧软件都听人话，那"重学"这一步几乎消失了——反正你对两边都只需要说人话。你练熟一个界面的那几个月，从"沉没成本变护城河"，变成了"白练"。厂商精心设计的迷宫，被一句自然语言绕了过去。

## 别急，不是所有界面都得死

这里要踩一脚刹车，否则就滑进"AI 要取代所有 UI"的空泛口号了。那是错的。

自然语言擅长的是**说清意图**，不擅长**高带宽的精确操控**。有一类操作，GUI 到今天依然吊打说话：

- **高频、需要即时反馈的操控**：专业绘图里拖动一个锚点、微调贝塞尔曲线的弧度。你没法用嘴说"把这个点往右挪 3 个像素、再往上一点点、不对回来半点"——太慢，太糊。手直接拖，一秒钟的事。
- **需要一眼看全局的场景**：交易终端上十几个实时跳动的数字、一整块盘口，眼睛扫一遍就有判断。让 AI 念给你听？信息带宽差了几个数量级。
- **精密编辑**：写代码时的光标、多选、跳转、重构快捷键，是极高带宽的精确输入。"说人话"在这儿是降速。

> 大白话
>
> 说人话的带宽低：一句话能传的信息有限，且模糊。眼睛看、手直接操作的带宽高：一屏信息一眼吸收，一个拖拽精确到像素。带宽差异决定了哪种交互更合适，跟"新旧"无关。

## 真正的变化：意图和执行分层了

所以别把这一章读成"界面消失"。真正发生的事，是一次**分层**——把过去揉在一起的两件事，掰成了两层：

- 意图：你到底想要什么结果。"把这三个表合并、标红异常"。
- 执行：为了达到这个结果，具体要点哪些按钮、走哪些步骤。

在 GUI 时代，这两层是压在一起的：你想要一个结果，就必须亲手把执行的每一步都点出来。你不得不学会执行，才能表达意图。自然语言把它们撬开了：**人负责说意图，模型负责翻译成一堆执行操作。**

这个分层，才是界面贬值的真正机制。不是界面不存在了，而是**用户不再需要直接面对它**。很多软件会退到对话后面，变成一台"藏在对话背后的引擎"——它照样在点那些按钮、跑那些工作流，只是点按钮的人从你换成了模型。引擎还在干活，但用户看不见它长什么样了。

一旦用户看不见你的界面，你界面上那些精心设计的品牌感、那套"用起来就知道是我家产品"的辨识度，价值就掉下来了。用户记住的是那个帮他转达需求的助手，不是你。

## 一个埋在这里的推论

顺着分层往下想一步，会撞见一个反直觉的结论。

当用户不再直接戳你的界面，而是对着一个 AI 助手说"帮我把这几个表合并了"，那么**这个助手，就卡在了你和用户中间**。用户的意图先到助手那里，助手再决定调哪个引擎去执行。谁拥有那个"助手入口"，谁就站在了你和用户之间的关口上。

你和用户之间，第一次隔了一个不属于你的中间人。这件事会把"入口该归谁、分发该归谁"变成一场硬仗——那是第 8 章和第 9 章要打的仗。这里只点到，先记住：**界面贬值，顺手就把"谁离用户最近"这个问题重新翻了个底朝天。**

## 用一个类比收尾

过去用软件，像开手动挡的车。你得学离合、记挡位、练到脚和手配合默契，才能把车开好。这套技能是门槛，也是你和"不会开的人"之间的护城河——顺带也把你锁在你熟的那辆车上。

自然语言交互，像打车。你钻进后座，对司机说一句"去机场"，就完事了。你不再需要懂车、懂路、懂怎么操作——你只要会说目的地。

> 大白话
>
> 但注意：车还在，发动机还在干活，只是你不用碰它了。会开手动挡在赛道上、在越野时依然有价值（对应那些高带宽精密操控的场景）。变的不是"车没用了"，是"大多数人不再需要亲自开"。

软件的界面就是那套"怎么开车"的操作。当你只需要说目的地，那些年你练出来的驾驶技术——那份构成切换成本的肌肉记忆——就不再把你拴在原地了。这是第三根松掉的螺丝。下一章，我们去掀开那个越来越常见的东西：如果引擎藏在对话后面，那前面那层"套壳"，到底还有没有价值。

# 第七章 · 套壳有没有价值

前两章我们把两把刀递给了「套壳没价值」这个论点：能力是租来的（第五章，核心智能来自你没写的通用模型），界面在贬值（第六章，说人话就能用，图形界面这个老资产不再稀缺）。顺着这两刀往下，一个从业者最爱吵、也最扎心的问题就摆上了桌面。

先把词说清。套壳（wrapper，在别人的大模型外面包一层自己的产品，核心智能是租来的，你自己主要是调 API、做界面、接流程）——这类应用到底有没有门槛、有没有价值？这一章只干一件事：把这个问题拆到底。读完你要能回答两件事——壳的价值到底在哪，以及什么样的壳，三个月就被模型自己吃掉。

## 先把「没价值」这一派讲到最狠

要诚实，就得先把对手的论点擦亮，讲到它最强的版本，而不是找个稻草人来打。「套壳没价值」这一派，手里有四张实牌：

- **核心能力是租的。**你产品里最聪明的那部分——读、写、推理——不是你写的，是模型厂商的。你退租了，产品就变哑巴。你没有护住那个「聪明」。
- **界面在贬值。**过去你精心设计的菜单、表单、快捷键是资产；现在一个对话框谁都能拼，这层壁垒薄了。
- **一个周末就能拼个 demo。**调通一个 API、套个聊天框、写几句提示词，一个工程师一个周末就能做出一个「看起来能用」的东西。门槛低到令人不安。
- **模型厂商随手就把你覆盖了。**你辛辛苦苦做的「AI 帮你总结长文」，模型厂商下个版本原生支持了，你整个产品一夜之间成了多余的中间商。

这四条不是危言耸听，全是真的。任何想给套壳辩护的人，只要绕过这四条，就是在写软文。所以我们不绕。**薄壳确实没价值，而且确实会被吃掉。**问题不在这句话对不对，而在它漏掉了一个关键区分——它把所有壳当成了同一种壳。

## 壳 ≠ 没价值，价值在「壳里装了什么模型给不了的东西」

「壳」是个形状，不是个判决。同样一个塑料壳，里面可以是空气，也可以塞满模型永远拿不到、也懒得做的东西。价值不在壳这层皮上，在皮底下你装了什么。有门槛的壳，靠的是下面这四样，逐条讲清。

### 一、工作流嵌入：把 AI 缝进用户每天真实的流程里

工作流嵌入（把 AI 的能力，接进用户每天真实干活的那条流程里——数据从哪个系统来、结果发到哪个系统去、中间要谁审批、和谁协作）。这活儿又脏又具体：一家医院的病历怎么流转、一家律所的合同要过哪三道签字、一个电商客服的退款要卡在哪一步——每个行业、每家公司都不一样。

> 大白话
>
> 说人话：模型是个绝顶聪明但对你公司一无所知的空降兵。他会思考，但不知道你们的活是怎么一步步流下来的、谁管盖章、错了找谁。把他塞进你们真实流水线里、让他不添乱地干活——这套脏活模型厂商不会为每个行业挨个做，因为对他们不划算。

### 二、专有数据与上下文：你有的，模型没有

模型读遍了公开的互联网，但它没读过、也读不到你客户的私有数据：这家公司三年的工单、这个用户的历史订单、谁有权限看哪份文件。这些数据是你在业务里一天天沉淀下来的，模型再聪明，手里也是空的。

把私有数据、历史、权限喂给模型当上下文，让它的回答贴着「这一家客户的真实情况」——这件事只有握着数据的人能做。数据不是壳，是壳里最硬的那块芯。

### 三、可靠性工程：把「经常对」变成「可交付地对」

这条最容易被低估，也最是护城河。模型有个天生的毛病，叫幻觉（hallucination，模型会一本正经地编出听起来很对、其实是错的答案，而且它自己往往不知道自己错了）。它天生不稳定——同一个问题，今天答对，明天可能编瞎话。

可靠性工程（一整套把「经常对」变成「可交付地对」的工程手段）就是专门对付这个的。它包括：

- **评测**：建一套题库，每次模型或提示词一改，先跑一遍，量出它到底对多少，而不是凭感觉说「好像变好了」。
- **约束输出**：逼模型只能从合法选项里选、只能按固定格式吐，堵住它自由发挥编瞎话的路。
- **纠错与兜底**：让另一道程序或另一个模型来复查，查出可疑的就打回、就换保守答案，绝不硬着头皮往下走。
- **人审**：关键环节留一个人把关，模型给草稿，人拍板。

> 大白话
>
> 说人话：一个周末能拼出的 demo，是「十次里对八次」——演示够炸，上线要命。真实生意要的是「十次里对九十九次半，剩下那半次也不会闯大祸」（这两个数字只是打比方的示意，不是什么实测阈值，别当硬指标）。从 demo 到产品之间这道鸿沟，全靠这套又慢又不性感的工程去填。这道鸿沟，就是壳最深的那条护城河。

### 四、分发与信任：你已经在客户面前

你已经站在客户面前了——有品牌、有销售关系、有多年攒下的信任，还有一堆合规资质（数据安全认证、行业许可）。这些是新玩家进不来的门。一个技术再强的周末 demo，也没法一夜之间让一家银行的采购部信它、让监管点头放它进生产系统。分发和信任本身就是壁垒，而且是模型再强也帮你搭不起来的那种。

## 一把判断尺子：看它离模型的「通用能力」有多近

四条讲完，把它们收成一句能随身带走的话。判断一个壳会不会被吃掉，用这一把尺子——

> 看这个壳，离模型的「通用能力」有多近。越近，越危险；越远，越安全。

所谓「离通用能力近」，就是你干的事，恰好是模型天生就会、或者下个版本随手就能会的。「帮你把提示词写得漂亮点」「给聊天框换个皮肤」「AI 帮你总结/AI 帮你写作」——这些

价值全建在模型的通用能力正上方，模型升一级，你就蒸发一层。反过来，你把价值越是往**专有数据、真实工作流、可靠性、分发信任**这四个方向挪，就离模型的通用能力越远，模型再升级，也踩不到你头上——因为它升级的是「更聪明」，而你护住的根本不是「聪明」。

> 大白话
>
> 但这把尺子只给「默认判决」，不给「死刑判决」，别拿它一刀切。它量的是第一、二、三条那种能力上的门槛；可上面第四条「分发与信任」自己就是个例外——它是一种能对冲距离的护城河。所以会出现这种拧巴事：一个能力很薄、离通用能力近得要命的壳，照样能靠分发活得挺滋润。银行、医院里那个平平无奇的「AI 帮你总结」，功能上模型随手就能覆盖，可它是走完了数据安全认证、卡进了既有采购关系、机构信得过它才进得了生产系统的——换个技术再强的新玩家来，一时半会儿也顶不掉它。记住这句：**「离通用能力的距离」给的是默认判决，分发、合规、信任这类护城河能把这个判决往回拉。**离得近不等于必死，只是默认危险、得靠别的东西续命。

拿两个例子对着看：

- **反例（薄壳，会被吃）：**一个「AI 写作助手」，本质是把用户输入转一手喂给模型、再把模型的输出美化一下返回。它没有专有数据、没嵌进谁的流程、没做可靠性、也没有分发优势。它站的位置，正是模型通用能力的正上方——模型自带写作，它就没有存在的理由了。
- **正例（厚壳，吃不动）：**一个深嵌某行业合规流程的助手——它接着这家公司的私有台账，懂这个行业每一步该过哪道审批，对每条输出都做了评测和人审兜底，还带着监管认可的资质摆在客户面前。模型再强，也不会平白拥有这家公司的数据、这个行业的流程、这套已经跑通的信任。它护住的，全是模型给不了的东西。

## 类比：你卖的不是电，是「能用上电」

把这一章收进一个画面。你自己不发电，你转卖国家电网的电。只是倒卖裸电——谁都能从电网买，你在中间加个价，这没有门槛，明天任何人都能干，电网自己也随时能把你甩掉。这就是薄壳。

但换个活法：你在一个偏远山村，把电线一根根架进每户人家，装好插座，出了故障半夜也去修。你卖的其实不是电，是「这个村能用上电」——是最后一公里的线、插座、和随叫随到的维修。电还是电网的电，但没有你，这村子点不亮。

大白话

说人话：倒卖裸电（模型的通用能力）谁都能干，没门槛；铺好最后一公里（工作流、数据、可靠性、信任）才有门槛。壳的价值，从来不在「转卖电」，在「让电真能用起来」的那一整套脏活上。

所以「套壳有没有价值」这个问法，本身就问错了。壳只是形状，价值取决于壳里装了什么、离通用能力有多远。这一章给了你判断单个壳生死的那把尺子——但它没回答一个更大的问题：把整条价值链摊开，模型层和应用层，最后到底谁吃到肉、谁替谁扛成本？那是下一章的事。至于这些「模型给不了的东西」为什么恰恰构成了新时代的护城河、该系统性地往哪儿挖沟，我们留到第九章一次收束。

# 第八章 · 应用层和模型层谁吃肉

## 应用层和模型层谁吃肉

前面几章我们拆完了旧地基：边际成本从零变正（第4章），能力是从通用模型租来的（第5章），交互变成了自然语言（第6章），套壳有没有价值取决于壳自己攒了什么（第7章）。现在把镜头拉远，看整条价值链——钱到底沿着这条链流到哪一格去了。

读完这一章，你应该能回答两个问题：为什么"卖铲子的"不一定最赚？以及，这条链上最危险的位置在哪里？

### 先把三层拆开，都讲人话

一个大模型产品从技术底座到最终用户，中间站着三层玩家。

模型层，英文叫 foundation model，就是从头训练那个底层大模型的厂商——做 GPT 的、做 Claude 的那类公司。它们干的是最重、技术最深的活：搜集海量数据、租下成千上万块显卡、跑几个月的训练，产出一个能力通用的大脑。

应用层，就是贴着具体客户、具体场景做产品的公司，也就是第7章讲的"壳"。它们不自己训模型，而是调模型层的接口，包一层界面、流程、行业知识，卖给律师、卖给客服团队、卖给写代码的人。它们离真实需求和真实的钱最近。

夹在两者之间的，是中间件（也叫基础设施层）：帮别人把模型接起来、用起来的工具。比如向量库（把文本变成一串数字存起来，方便按语义检索的数据库）、编排框架（把"调模型、查资料、调工具"这些步骤串成流水线的代码库）、以及监控、日志、评测这些配套。它们不直接面对终端用户，卖的是"让接模型更省事"。

大白话

模型层造发动机，应用层造整车卖给司机，中间件卖火花塞、卖变速箱这类零件。三层各挣各的钱——问题是，这三块的钱厚薄差得很远，而且和你的直觉相反。

### "卖铲子的最赚"，这个类比在这里会骗你

科技圈最爱讲一个淘金热的类比：淘金的人十个九个亏，真正稳赚的是路边卖铲子、卖牛仔裤的。于是很多人顺手推论：模型层是卖铲子的，稳赚；应用层是下场淘金的，风险大。

这个推论听起来很顺，但它悄悄借用了一个1849年成立、今天却不成立的前提。当年卖铲子为什么稳？因为——

- **铲子不用天天重造。**一把铲子的设计几十年不变，造好了库存卖就行，没有持续的天价研发。
- **卖铲的不跟买铲的抢金子。**五金店老板不会晚上偷偷跑去淘金，跟自己的客户竞争。
- **铲子之间不打价格战打到骨折。**你家铲子和我家铲子差不多，但没人天天砸钱只为让铲子便宜一美分。

现在把这三条对着模型层看，会发现每一条都反过来了。

第一，模型层的"铲子"每个季度都要重造，而且一次比一次贵。这就是军备竞赛——不停砸几十亿美元买算力、抢顶尖研究员、训更大的模型，谁停下来谁的模型就落后、就被客户抛弃。铲子不是造好一次卖十年，而是造好三个月就得推倒重来，成本还翻着番往上涨。

第二，卖铲子的自己跳下去挖金了。模型厂商不满足于只卖 API，它们直接下场做应用——做聊天助手、做写代码工具、做企业套件，跟自己 API 的客户正面竞争。它既卖铲子，又用铲子挖矿，还常常把挖矿工具免费送，逼得单纯买铲子的应用层很难受。

第三，几家头部模型能力越来越像，客户切换只是改一行配置，于是价格战开打；再加上开源模型（把训练好的模型权重公开、谁都能免费下载自己跑的模型）在后面追赶，等于有人在旁边免费发铲子。三面夹击，卖铲子的利润被军备竞赛和价格战一起吃掉。

> 大白话
>
> 结论不是"模型层不赚钱"，而是：模型层烧钱换来的是规模和领先，不等于换来利润。花一百亿美元训出全球第二好的模型，客户可能一分钱不给你，直接用便宜10%的第一名或者免费的开源货。技术最深的那一格，恰恰是资本消耗最凶、护城河最容易被填平的那一格。所以"卖铲子"在这儿不是一个安全位置，而是一场必须持续下注、下错就出局的豪赌。

## 最危险的位置：中间那层薄夹层

如果说模型层是"贵但不一定赚"，那真正九死一生的，是夹在中间、既不够深也不够近的**薄夹层**。

什么叫薄夹层？就是那种只做"通用一层"的中间件：只做一个通用的编排框架、只做一套通用的 prompt 优化、只做一个谁都能替代的转接头。它的处境是两头受挤：

- **上边被模型厂商一个新功能抹掉。**这正是第7章说的"离通用能力太近"——你辛苦做的通用编排、通用记忆管理、通用工具调用，模型厂商觉得该内置的时候，出个新版本 API 就自带了，你这一层瞬间没了存在理由。
- **下边被应用层直接绕开。**应用层发现你这层没提供什么独有价值，干脆自己直接调模型、少付一道过路费。你的过路费桥，随时被人从旁边趟过去。

回到淘金类比：中间件本来像"修路搭桥收过路费"的。但这条链上，桥的两头——模型层往下延、应用层往上够——随时能自己修一条便道绕开你。你既没有模型层的技术纵深和资本壁垒（挖井太深别人跟不进），又没有应用层贴着客户攒下的分发渠道和专有数据（客户和数据都在别人手里）。两样护城河都不占，就只剩下"暂时比较方便"这一条，而方便是最容易被复制的东西。

> 大白话
>
> 薄夹层的死法很典型：它做的事有价值，但价值不归它。它替整条链解决了一个真问题，可这个问题一旦被上游内置、被下游自建，它创造的价值就被两头分走，自己什么都没剩下。不可替代性不够，就会被碾平。

## 那到底谁能吃到肉

把结构性因果理一遍，能吃到肉的其实是两头的少数赢家，条件都很硬：

- **真正贴客户的应用层。**注意"真正"两个字——不是随便包个壳，而是握着别人拿不到的专有数据（客户在你这儿长期积累、竞争对手复制不了的数据）和分发渠道，能把上涨的 token 成本转嫁出去（这条留到第11章细讲），还能把可靠性、准确性这些"脏活"做扎实。它离钱最近，且这份"近"是别人短期抢不走的。

- **少数赢家通吃的头部模型层。**军备竞赛是残酷的淘汰赛，但活到最后、能把巨大算力成本摊到海量调用上的那一两家，规模效应会反过来变成壁垒。吃肉的不是"模型层"这个身份，而是"跑赢军备竞赛"这个结果。
- **成了事实标准的中间件。**薄夹层会死，但如果一个中间件深到成了行业默认选择、大家的系统都围着它建、换掉的代价高到没人愿意换——它就从"可绕开的过路费"变成了"绕不开的地基"。要么不可替代到这个程度，要么被两头挤没，中间没有安全区。

你会注意到，这三种赢家的共同点，都不是"站在哪一层"，而是"在那一层里筑起了别人填不平的护城河"。护城河这东西没有消失，它是从旧地方搬走了——搬到哪去了，是第9章要正面重建的话题，这里先不展开。

### 本章落点：你在价值链的哪一格

与其记住"应用层必胜"或"模型层必死"这种站队口号——两句都是错的——不如拿下面这组问题，给你手上的产品做个自检：

1. 如果模型层下个版本把你这层能力内置了，你还剩什么？（剩得越少，你越像薄夹层）
2. 如果你的客户明天绕过你、直接调模型，他会损失什么你独有的东西？（答不上来，说明你没攒下专有资产）
3. 你手里的数据、分发、客户关系，是租来的还是自己的？（租来的能力，别人也能租；第5章）
4. token 成本涨一倍，你能转嫁出去吗，还是只能自己扛？（转不动的，往下看第11章）

把这四个问题诚实答一遍，你大概就知道自己站在哪一格、离两头的挤压有多近。价值链上没有天生安全的层，只有筑没筑起护城河的位置。下一章，我们就去看那条护城河，到底被搬到了哪里。

# 第九章 · 护城河搬到了哪里

## 护城河搬到了哪里

前面六章像拆房子：边际成本从零变正（第4章）、能力是从通用模型租来的（第5章）、界面变成了说人话（第6章）、套壳的价值只在壳自己攒了什么（第7章）、钱沿着价值链流到最贴客户的那头（第8章）。拆得很爽，但读者迟早会问一句正经的：那今天还有护城河吗？如果有，该去哪儿挖？

这一章负责正面回答。前面几章零散点到的线索，这里收束成一个体系。

### 先把护城河这个词按住一秒

第3章讲过，护城河不是「我做得比你好」，而是一种**结构性优势**——就算你做得比我好，你也追不上、抄不动、挖不走。它是地形，不是速度。这一点没变，一个字都没变。变的只是：地形从一个地方，挪到了另一个地方。

旧的那几条为什么漏水，一句话带过就够，第5、6章已经拧过螺丝：**能力是租的，界面在贬值**。既然对手能租到和你一样的模型，用户又能对着谁的产品都说人话，那「我功能多、我界面熟」就守不住了——这两样恰恰是租得到、学得会的东西。护城河不会因为旧配方失效就消失，它只是从「你拥有的技术」，搬到了别处。

### 换个类比：金矿边上，人人买得到同一台挖矿机

想象一片金矿，边上有家店，把全世界最好的那台挖矿机（大模型）敞开卖，谁掏钱谁都能搬一台回去。这时候「我有挖矿机」不再是本事——因为你有，我也有，隔壁老王也有。

> 大白话
>
> 挖矿机平权了，比的就不再是机器，而是四件机器给不了你的东西：谁的矿占得好，谁修好了通往矿的路和仓库，谁手里攥着别人没有的藏宝图，谁能保证挖出来的是真金不是黄铜。

这四件事，正好对应新护城河的四条。下面一条条讲：机制是什么，以及为什么模型再强也抹不平它。

## 一、分发：谁占住了通往矿的位置

分发（distribution）：你已经站在用户面前了——你有入口、有触达渠道、有装机量，用户打开某个东西时第一眼看到的是你。

这是最朴素、也最容易被工程师看轻的一条。工程师爱谈技术强不强，但商业上有个冷冰冰的事实：**模型再强，也得有人真的用你的产品，它才产生价值。**而「让人用上」这件事本身极贵——获客、建立信任、挤进用户每天会打开的那几个东西，这条路又长又烧钱，且和模型能力毫无关系。

所以已经卡住入口、已经攒下信任的公司，等于赢在起跑线。这正呼应第6章那句：**谁拥有助手入口，谁就卡在用户和模型中间**。你哪怕训出全世界第二好的模型，用户接触不到，就等于没有。而占着入口的人，可以随时把最好的模型接到自己后面——用户根本不在乎背后是谁的模型，只在乎那个他每天都在用的入口还在。

> 大白话
>
> 模型是发动机，分发是那条已经通到用户家门口的路。发动机谁都能买，路你得自己修很多年。占住路口的人，换个发动机就行，从不担心发动机。

## 二、工作流嵌入：修好了通往矿的路和仓库

工作流嵌入：把 AI 缝进用户每天真实跑的流程里——它的数据从哪来、要过谁的审批、和哪些人协作、和哪些别的系统对接。这些活又脏又具体，缝得越深，用户越拔不动。

你会发现这其实是第3章「切换成本」的新形态。旧切换成本靠「界面熟、肌肉记忆」——第6章说了，界面被说人话削掉了，这条在贬值。但切换成本没死，它换了个地方扎根：从**界面熟**，迁到了**流程绑**。

举个工程师熟的场景。一个 AI 工具如果只是「你贴文本，它给你回答」，那它是浮在流程外面的，换掉毫无痛感。但如果它接进了公司的工单系统，自动从数据库拉客户历史，生成的结果要走三级审批链，审批完自动回写进 ERP，出了问题还能顺着日志追责——这时候它不

是一个工具，它是流程的一段血管。你想换掉它，扯的是一整片，跟第3章那个「拔一个牵一串」的集成锁定一模一样，只是这次缠住的是 AI。

模型抹不平这条，是因为通用模型只懂通用知识，它不懂你公司这条审批链怎么走、这个字段对哪个字段、这个数据要不要脱敏。这些脏活是一家一家手工缝进去的，没有一个通用模型能替你缝。缝得越深越具体，越是别人（包括模型厂商）短期内够不着的地形。

## 三、私有数据：手里那张别人没有的藏宝图

私有数据（proprietary data）：通用模型几乎什么公开的东西都读过，但它没读过、也读不到你客户的私有数据、你系统里的实时数据、你行业里的专有知识。你能喂给模型的这份独家上下文，才是别人租同一个模型也复制不出来的东西。

这里必须点破它和第3章「数据网络效应」的区别，否则很容易当成一回事。

- **老的是「量的飞轮」。**用户越多 → 数据越多 → 产品越准 → 又吸引更多用户，滚雪球。重点在*越滚越大*。
- **新的是「独家的墙」。**重点不在多，在**模型进不去**。你独家握着一批数据——客户内部的、实时的、合同保密的——通用模型再全能，也从没见过它，也拿不到它。

> 大白话
>
> 老护城河比谁的雪球滚得大，新护城河比谁的墙关得严。模型什么都读过，唯独没读过你墙里的东西——那堵墙就是你的沟。

为什么模型抹不平？因为模型的能力来自公开语料，它的强恰恰建立在「大家都能读到的东西」上。你墙里的私有数据，正好落在它训练照不到的盲区。你接同一个模型、我接同一个模型，但我往它嘴里喂的是我独家的上下文，你喂不进去——同一台挖矿机，你没有那张图，就是不知道往哪儿挖。

## 四、结果可靠性：保证挖出来的是真金不是黄铜

结果可靠性：模型的输出天生不稳定，同样的问题问两次可能给两个答案，还会一本正经地胡说。谁能把这份不稳定，做成「可交付地对」——上评测、上纠错、加约束、设兜底、并且愿意为结果负责——谁就筑起了一道信任壁垒。

这条在高风险场景里几乎是唯一算数的护城河。医疗、金融、法律——这些地方，一个「大概对」的答案等于没有答案，甚至是负资产。客户买的从来不是「模型能生成」，而是**「出错了有人兜、出事了有人担」**。裸模型给不了这个承诺，它只会生成，不会负责。

模型抹不平这条，是因为可靠性不是模型能力的一部分，而是模型能力*之外*那一整套工程和责任：你得建评测集去量它到底多准、写规则去卡住它的胡说、设计人工兜底去接住它出错的那些次、最后还得有一个法律和商业上的主体，敢在合同里签字说「这个结果我负责」。模型厂商在 API 条款里写的是「不保证输出正确」——那份没人签的责任，恰恰是应用层能捡起来的、最硬的一道墙。

## 一句统领：护城河从「技术」搬到了「位置和关系」

把这四条摞起来看，会浮出这一章真正的骨架：

> 护城河从「你拥有的技术」，迁移到了「你拥有的关系与位置」。

过去问一家公司的护城河，问的是**「你写了什么难的代码、你界面做得多好」**。这些今天都在平权——代码能力从模型里租，界面被说人话削平。而新的四条，没有一条是在问技术：分发问的是**你离用户多近**，工作流嵌入问的是**你缝进流程多深**，私有数据问的是**你独家握着什么模型进不去的东西**，结果可靠性问的是**你敢对结果负多大责任**。

大白话

一句话：技术在平权，位置在增值。你写的代码越来越不值钱，你卡的位置、你攒的关系、你担的责任越来越值钱。护城河没干涸，它只是从你机房里，搬到了你和用户、你和数据之间那段地形上。

## 本章落点：给你一把尺子

拿到一家公司，别急着夸它技术多牛，先把它的护城河分个类，问一句：

1. 它的护城河是**能力型**的吗？——靠「我模型调得好、我功能多、我界面顺」。如果是，警惕：这类东西正在被平权抹平，对手租同一个模型就能追上来。

2. 还是**位置/关系/责任型**的？——靠「我卡在用户入口、我缝进了流程、我握着独家数据、我为结果负责」。如果是，这几样模型抹不平，沟是真的。

能力型的护城河，是在流沙上挖沟，水一冲就平；位置、关系、责任型的护城河，才是挖在别人搬不走的地形上。今天该去哪儿挖沟，答案就在这把尺子里。

当然，光有护城河还不够——挖出了真金，怎么定价、怎么把它变成能收上来的钱，是另一回事（第10章）。而这把「能力型 vs 位置型」的尺子，到第15章判断「谁死谁强」时，还会再拿出来用一次。这里先埋下。

# 第十章 · 不再按人头收费

## 不再按人头收费

前面几章把旧地基一块块撬开了：边际成本从零变正（第4章），能力是租来的（第5章），护城河搬了家（第9章）。这一章我们把这些变化收拢到一件最实在的事上——**钱怎么收**。因为地基一动，账单的算法就得跟着重写。你会看到，SaaS 时代最经典的那种收费方式，正在从"印钞机"变成"漏水的桶"。

读完这一章，你应该能回答两个问题：为什么"按人头卖席位"这套收法会漏？以及，"卖完成的活"是怎么把收入重新搭起来的？

### 先看旧收法：按席位订阅

按席位订阅（per-seat，按用户人头每月收一笔固定费，比如"每人每月30美元，你们公司100人就是3000"）是 SaaS 时代的默认收法，也是第2章那台印钞机的核心零件。它之所以好用，是因为它踩在一个前提上：多一个用户，你几乎不多花钱（零边际成本），所以每多卖一个席位，几乎全是纯利。席位数一涨，利润就涨，账还特别好算——数人头就行。

> 大白话
>
> 过去卖软件像卖健身房年卡：办多少张卡是收入，而多一个会员来不来、来了练多久，成本几乎不变。所以你只管数卖了多少张卡。

问题是，大模型把这个前提抽走了。而且是从两头一起抽——成本端和价值端。

### 成本端：收入按人头走，成本按用量走，两头脱钩了

健身房那套之所以成立，是因为会员来不来你成本不变。但大模型产品不是这样：每次调用都要烧 token，用户每多干一件活，你就多付一笔推理费。于是账本上出现了一道裂缝——

- **收入这条线，随"有多少人"走。**你按席位收费，100个席位就是固定的100份月费，跟他们用多用少无关。

- **成本这条线，随"用了多少"走。**token 账单只认调用量，跟你卖了几个席位无关。

两条线本来在零边际成本时代是平行的（成本那条几乎贴着地面），现在成本这条翘起来了，还跟收入不同步。结果就很难看：

- **重度用户把你毛利吃穿。**公司里那几个天天让 AI 跑长文档、批量处理的人，一个人烧的 token 可能顶几十个席位的月费。你按人头只收了他一份钱，他却单独把你这单的利润啃光，甚至让你倒贴。
- **轻度用户觉得不值，转身就走。**另一头那些一个月点开两次的人，白白付着全额月费，心里盘算"这钱不值"，续费时就砍席位。你想靠他们补贴重度用户，他们先跑了。

> 大白话
>
> 一句话：席位数和你的真实成本，不再是一回事。你收的是"人头钱"，付的是"用量钱"，这两个数各走各的，中间的缝就是你毛利漏掉的地方。

### 价值端：AI 越能干，越砸了按人头收费的逻辑

成本端的漏还只是难受，价值端这一刀才是真正致命的，而且反直觉。

按席位收费其实偷偷绑定了一个假设：**席位数 ≈ 干活的人数 ≈ 你创造的价值**。一个客服团队有50人，你卖50个席位，因为这软件在帮50个人干活，收50份钱天经地义。

现在 AI 把这个假设打穿了。当 AI 让一个人干出过去十个人的活，客户会怎么想？他不会想"我该多付钱"，他会想"那我能不能裁掉九个席位"。

这就是反直觉的地方：**AI 让每个席位更能干，恰恰砸了"按席位数收费"的逻辑。**你越是帮客户把活干得又快又好、把人省下来，客户需要的席位就越少，你按人头能收的钱就越少。你把自己做得越强，你的账单反而越薄。

> 大白话
>
> 你努力帮客户省人，客户就顺手帮你省钱——省的是你的钱。席位不再代表价值，它现在代表"客户还没来得及裁掉的人"。把你的收费锚在一个客户正想方设法减少的数字上，这本身就是个漏。

所以按席位这条路，成本端在漏、价值端在塌。人们开始换收法。下面三种，是沿着"把账单重新贴回真实价值"这条线，一步步往前挪的。

### 第一步挪回来：按用量计费

按用量计费（usage-based，按调用次数、token 数、处理的条目量来收，像水表电表，用多少收多少）是最直接的修补。它把收入那条线，重新焊回到成本那条线上——你多烧 token，客户就多付钱，毛利不再被重度用户吃穿，也不用逼轻度用户为没用到的东西买单。成本和收入重新同步了。

但它引入一个新麻烦，而且是人性层面的：**客户讨厌不可预测的账单。**企业做预算要的是一个能写进表格的确定数字，而按用量意味着月初不知道月底要付多少，用猛了月底账单吓一跳。云计算这些年被人吐槽的"账单焦虑"就是这毛病。你把不确定性从自己身上，甩给了客户，客户不一定接。

> 大白话
>
> 按用量把成本的账修对了，但把不可预测这件事推给了客户。它解决了你的毛利问题，没解决客户的安全感问题。所以它常常只是过渡形态，不是终点。

### 第二步，也是想象力最大的一步：按结果计费

按结果计费（outcome-based，不按你用了多少，也不按你有几个人，而是按"完成了多少件活"收钱——解决一个客服工单收一笔、跑通一次财务对账收一笔、成交一个销售线索收一笔）是这一章真正的转折点。因为它把你卖的东西，从"工具的使用权"换成了"事情被做完"。

这里值得停下来讲个类比，因为它一下能说清楚这一步跨了多大。

> 过去买软件，像买一把电钻：一次性买断也好、订阅也好，你付的是"这把钻归你用"，至于你拿它在墙上钻几个孔、钻得好不好，是你自己的事。现在有一种卖法变了——我不卖你电钻，我按"帮你在墙上打好的孔"收钱。

关键在于：**客户要的从来不是电钻，是墙上的孔。**电钻只是拿到孔的手段。过去因为"打孔"这件事没法标准化、没法自动完成，卖工具的只能把工具交给你，让你自己去打。现在 AI 能把"打孔"这件事端到端做完，于是卖家第一次可以跳过工具，直接卖那个孔。

谁能直接卖"孔"，谁就跳过了"卖工具"的天花板。因为客户心里给"一把电钻"的预算，和给"墙上打好的孔"的预算，根本不是一个量级——他愿意为结果付的钱，远高于愿意为工具付的钱。

### 把这一步推到底：按 agent 工时、按完成任务

顺着"卖结果"再往前一步，就到了最激进的收法。这里得先讲清一个词。

agent（智能体，指能自己把一个大任务拆成多步、然后一步步自动执行的 AI——不是你问一句它答一句，而是你交代一件事，它自己查资料、调工具、干完再回来交活）出现之后，AI 不再只是"帮人干活的工具"，它开始像"一个会干活的人"。

于是有了按 agent 工时/按完成任务计费：把一个 AI agent 当成一名"数字员工"，按它干了多少活、占了多少工时来收钱。它对标的价格锚点变了——不再对标一份软件订阅（一个月几十上百块），而是对标**一个人的人力成本**（一个岗位一个月几千上万块，外包价更是明码标价）。

想象空间就在这个锚点的切换里。软件行业一直只能在"软件预算"这块蛋糕里分食，而"软件预算"在一家公司的总开支里，往往是很小的一块。真正大的那块，是**人力预算**——工资、外包、人力外包合同。过去软件够不到这块钱，因为软件是工具，工具不领工资。现在 agent 能把活整件干完，软件第一次可以名正言顺地去咬"人力预算"这块大得多的蛋糕。

> 大白话
>
> 同一件事，你叫它"一个软件功能"，客户按软件的心理价给钱；你叫它"一个数字员工替你完成了这个岗位的活"，客户就会拿它跟"雇一个人要多少钱"去比。名字没变，参照物变了，能收的钱就是两个世界。

### 把这条线拎直：定价的锚，整个搬家了

回头看这四种收法，你会发现它们不是零散的花样，而是同一个锚点在一路搬家：

1. **按席位**——锚在"你有几个用户"。本质是拿用户数当价值的替身，AI 时代这个替身失效了。
2. **按用量**——锚在"你烧了多少成本"。修好了成本，把不确定性推给了客户。
3. **按结果**——锚在"你替客户做完了几件事"。从卖工具变成卖成果。
4. **按 agent/任务**——锚在"你替客户省下了多少人力"。去对标人力市场，天花板一下抬起来了。

一句话概括这场搬家：**定价的锚，从"你的成本、你的席位"，移到了"你替客户创造的价值、省下的人力"。**从"卖许可证"到"卖成果"，你能收的钱的天花板，从"软件预算"抬到了"人力预算"，这是一次量级的跃升。

但这里必须点破一件事，不然这一章会给你一个太美的错觉。这场跃升不是白拿的。当你从"卖工具"改成"卖结果"，有一样东西也悄悄换了手——**风险**。卖席位的时候，客户用得好不好是客户的事，钱照收；一旦你改成"事情做完才收钱"，那"要是没做成呢"这个问题，就从客户那边，转回到你自己头上了。天花板抬高的同时，你把不确定性从客户身上接了过来。

> 大白话
>
> 卖电钻的旱涝保收，钻不出孔是你手艺问题；卖孔的，孔没打好就一分钱收不到——想象空间和风险，是同一枚硬币的两面。

接下来会发生什么，这一章只点到为止：一旦你开始卖"完成的活"，你就得替客户扛下"完成"背后的全部成本，而这些成本正是正边际成本（第4章）。它会怎么啃你的毛利、把财务模型重新逼成什么样——那是下一章的正题。

# 第十一章 · 卖结果的人要扛成本

## 第 11 章 卖结果的人要扛成本

上一章我们看着定价从「按人头」松动开，长出了三种新收法：按用量收、按结果收、按 agent 工时收。那一章讲的是钱怎么进来。这一章讲的是它的背面——**钱怎么出去**。因为你一旦不再按人头收，而是按「干了多少活」收，那么第 4 章讲的那笔 token 成本，就不再是账本角落里可以忽略的小数点，它变成了你每做一单都要真金白银掏出去的料钱。

换个画面。过去开软件公司，像开一家**「复印店，但纸和墨水都不要钱」**。你印一份和印一万份，成本几乎一样，客户给的钱几乎原封不动落进兜里。现在不一样了：每印一份，纸和墨水都要花钱——这个纸墨，就是 token。你重新变回了一家要算料的普通店。这一章，就是教你像个开店的人那样，把料钱算清楚。

### 先把「毛利」这把尺子立起来

要看清成本怎么啃利润，得先有一把尺。这把尺叫毛利率。

毛利率（gross margin）：你收进来一块钱，先扣掉「为了多服务这一单而直接多花的钱」，剩下的占多大比例。注意，扣的只是「多服务一单的直接成本」，不含你办公室房租、研发工资这些不管卖几单都要花的固定开销。

> 大白话
>
> 说人话：一杯奶茶卖 20 块，这一杯的茶叶、牛奶、杯子成本 6 块，那这杯的毛利就是 14 块，毛利率 70%。毛利率量的是「每多卖一单，能净落下多少」，不管你的门店租金和店长工资。

现在把第 1、2 章接回来。传统 SaaS 为什么是印钞机？因为它「多服务一单的直接成本≈0」——多一个用户，多花的服务器钱小到可以忽略。于是收进来的那块钱几乎全是毛利。这就是为什么**软件公司的毛利常年在 75%–85%**，高得不像实体生意。这不是它们特别会做生意，是它们的料本来就不要钱。

### token 成本，是直接从毛利里扣的

关键的一步来了。token 成本是什么性质的成本？它是**每多服务一单就要多花一次的成本**——用户多问一句，你就多付一次；这单任务更重，你就付更多。按定义，它正好落在「多服务一单的直接成本」这一格里。也就是说，它不是从固定开销里出，而是**直接从毛利里扣**。

于是同样收进来一块钱：

- 传统 SaaS：料钱≈0，一块钱里八九毛是毛利。
- 大模型应用：料钱是实打实的 token，一块钱里可能只剩五毛、四毛，甚至更薄——取决于这单用得多重。

用得越重，扣得越狠。一个轻活儿（分个类、改句话），token 花不了几个钱，毛利还能保住七八成；一个重活儿（读整个代码库、让 agent 自己反复调工具跑几十轮，见第 4 章），token 能把毛利啃到**四五成甚至更低**。软件公司第一次，毛利要看「这一单具体干了什么」才知道。

> 大白话
>
> 说人话：SaaS 的一块钱收入，含金量八九成；大模型应用的一块钱收入，含金量可能只剩四五成。同样是一块钱，一个几乎是纯金，一个掺了不少料钱进去。

### 这会反噬估值——收入的形状变了

现在把第 2 章那条估值逻辑拽回来。资本为什么肯给 SaaS 那么高的估值（营收的几十倍）？我们当时拆过，就三样叠在一起：**经常性收入**（可预测）+ **极高毛利**（一块钱几乎全是利润）+ **可复利**（多服务不花钱，客户留得越久越赚）。

大模型应用动的，正是中间那样——毛利。当一块钱收入的含金量从八九成掉到四五成，这块钱在资本眼里就「变便宜」了。因为估值本质上是给「未来能落下多少利润」定价，而不是给「未来能收上来多少收入」定价。毛利薄一半，同样一块钱收入背后能落下的利润就薄一半，市场愿意为它付的倍数自然往下走。

往哪走？往**服务业、外包业**那边走。一家人力外包公司、一家咨询公司，为什么市销率（市值是营收的几倍）远低于软件公司？因为它们每收一块钱，背后都压着实打实的人力成本，毛利本来就薄。大模型应用按结果收费、每单压着 token 成本，收入的形状越来越像它们，而不像纯软件。

> 收入的形状变了，市场给收入的价格也会变。同样一块钱营收，长在零边际成本上，值几十倍；压着料钱，可能只值几倍。

### 成本转嫁：这笔料钱，到底谁扛

薄毛利不是命中注定的，它取决于一个选择：成本转嫁——token 这笔料钱，是你自己吞，还是转手让客户付。上一章那三种收法，其实就是在回答这个问题。

- **按用量收 = 转嫁给客户**。客户用多少 token、你就收多少（加个加价）。模型涨价、用户滥用，账单最终落在客户头上。你的毛利稳如老狗——因为你根本没扛成本。代价是：客户拿到一张**不可控的账单**，月底才知道花了多少，用起来提心吊胆，砍需求、装限额，甚至干脆不敢放开用。
- **按结果 / 固定价收 = 你自己扛**。「修好一个 bug 收 50 块」「一篇稿 10 块」，客户省心，账单可预测。但料钱这头是敞开的：模型一涨价、某个用户的任务比预期难十倍、失败重试了五轮、还得拉个人来兜底——这些成本全砸在你身上。极端情况下，你会**亏钱做单**：收了 50 块，这单烧了 60 块的 token。

大白话

说人话：这就像装修。按实报实销（按用量），师傅不亏，但业主看着账单心惊肉跳；包工包料（按结果），业主省心，可万一料涨价、活儿比想的难，师傅自己吃这个亏，甚至倒贴。省心和稳赚，天平只能偏一头。

这就是「卖成果」的隐藏账单。你越是给客户「省心、可预测」的体验，就越是把成本的不确定性揽到自己身上。定价越性感，风险越靠你这边。

## 单位经济：软件公司第一次要像餐馆一样算账

把上面这些逼到最尖的一点，就是一个实体生意才熟悉、软件公司过去根本不用碰的概念。

单位经济（unit economics）：做成「一单」，到底是赚还是亏，赚多少、亏多少。餐馆天天算这个——这道菜卖 38，食材成本 12，人工水电摊 8，一盘净赚 18；哪道菜是赔本赚吆喝，老板心里门儿清。

SaaS 时代，软件公司几乎不用算这个，因为「一单」的直接成本约等于零，做成一单基本等于纯赚，没什么可算的。大模型应用第一次把这道题摆回桌上：**做成这一单，到底烧了多少 token？失败重试了几次？要不要人工兜底？**算完才知道这单是赚是赔。

> 大白话
>
> 说人话：软件公司第一次要像餐馆算「每道菜的食材成本」一样，去算「每单活儿的 token 成本」。做成一单不再天然赚钱，得一单一单精算，才知道自己是在赚钱还是在给模型厂商打工。

这不是坏消息本身，而是一种成熟——它逼你把过去糊在「反正边际成本是零」里的糊涂账，一笔一笔摊开来算清楚。

## 另一面：料价每年在跌

讲到这里像在唱衰，但先别急着下结论。上面这幅画少了很重要的一笔：**token 这个料，价格不是恒定的，它每年在大幅往下掉**。同等能力的模型，单位成本逐年以肉眼可见的幅度下降——今天贵得肉疼的一单，一两年后可能便宜到零头。这跟餐馆不一样：餐馆的牛肉不会年年跌价，而 token 会。

而且，薄下去的毛利，是可以靠工程一分一分抠回来的：

- **模型路由**：简单活儿不用请最贵的专家。分个类、判个是非，用又小又便宜的模型就够了，只有真正难的活儿才上顶级大模型。模型路由就是让系统按任务难度自动挑模型——把料钱按需分配，别拿五星大厨去切葱花。
- **缓存**：一样的问题、重复的上下文，答过一次就存下来，下次直接取，不必让模型从头再算一遍。省下的每一次重算，都是省下的料钱。

- **用传统代码兜底**：能用一行确定的代码算清楚的（比如加减法、查一条记录），就别丢给模型现场推理。把能确定的部分交还给零边际成本的老办法，只在真正需要「思考」的地方才烧 token。

这几招合起来，能把一单的 token 成本压下相当一截，毛利跟着回升。所以长期看，大模型应用的毛利未必会一直趴在四五成——它有机会靠模型降价和工程优化，慢慢往上爬。

## 白吃的午餐结束了，但生意没死

把这一章收一下。承接第 10 章的新定价，它的代价是：当你按结果、按用量收钱，token 成本就变成了**你的**成本，直接从毛利里扣。于是——

- 毛利从 SaaS 那种 75%–85%，可能被压到 50%、40%，看你用得多重；
- 毛利一薄，收入的含金量下降，估值倍数向服务业、外包业靠拢，而不是软件业（反噬第 2 章那套高估值逻辑）；
- 谁扛 token 成本成了关键选择：转嫁给客户则毛利稳但账单吓人，自己扛则客户省心但你可能亏本做单；
- 单位经济第一次要像餐馆一样一单一单精算。

但结论不是「大模型应用是门烂生意」。真正准确的说法是：**「零边际成本的白吃午餐」结束了，毛利不再是天上掉下来的，得靠工程一分一分抠出来。**料价每年在跌，路由、缓存、代码兜底都能回收毛利——这门生意能不能算得过账，从此是个真问题，而不是像 SaaS 那样默认成立。

而这件事——「要精算单位成本、毛利更薄、每单都得算得过账」——会顺着一直往下渗，渗进一个更现实的地方：**创业和融资的算术**。当你不能再靠「边际成本是零」这句话糊过投资人，当每一单都压着料钱，一家公司该怎么起步、怎么烧钱、怎么讲估值故事，全都要重算。这是第 13 章的事。这一章先记住一句：你开的不再是那家纸墨免费的复印店了，你重新变回了一家要算料的普通店——只不过，你的料价每年都在往下掉。

# 第十二章 · 一人公司与小团队

## 一人公司与小团队：为什么组织会变小

前面十一章都在拆机制：成本、能力、界面、护城河、定价。全在讲「产品」和「钱」。这一章把同一套机制推到最后一层，也是最贴身的一层——**组织**。同样一套逻辑，落到「一家公司该有多少人」这个问题上，会得出一个很多人不敢相信的结论：很多公司会变小，而且是小很多。

先把这一章要证的话钉在墙上：**过去公司要养一堆人，很大一部分是为了买「产能」；现在产能可以按需从模型租，于是养人的必要性塌了一大截。**但——这一章不写爽文，后半段会老老实实讲：哪些人怎么都缩不掉，为什么。

## 先看清：过去公司为什么非得那么多人

回到第5章那个等式：过去，产品有多强 ≈ 你写了多少代码、你雇了多少人。把这句话翻个面，就是这一章的起点——**过去，产出多少 ≈ 你养了多少人。**

一家公司里，相当一部分岗位的本质，是**把人当执行器**：写模板文案、初筛简历、一审代码、做基础客服、把 Excel 里的数据整理成报表。你想多产出，办法很朴素——多招人。招一个人，多一份产能。

> 大白话
>
> 说人话：过去的公司，很大程度是一台「囤人力」的机器。你不知道下个季度要处理多少客服工单、要看多少简历，所以你按峰值囤人——旺季够用，淡季养着。人头是固定成本，招进来就得发工资，不管这个月他忙不忙。

这台机器有个绕不开的毛病：产能是**固定**的、**按人头结算**的。它不像水电，用多少付多少；它像你买断了一整个乐团，不管今晚演不演、演几首，工资照发。

## 那些「执行器」岗位，正是模型最擅长的

现在把第5章的「能力平权」端过来，对着这些岗位照一遍。你会发现一件很不舒服的事：**上面列的那些活，几乎全是「读—理解—生成」。**

读—理解—生成：把一段输入读进来（一份简历、一条工单、一段代码、一堆原始数据），理解它想说什么，再生成一段输出（一句回复、一个初判、一段文案、一张报表）。这正好是大模型的主场——它天生就是干这个的。

- 写模板文案 = 读需求，生成文案。
- 初筛简历 = 读简历，理解岗位要求，生成一个「过/不过」的初判。
- 一审代码 = 读 diff，理解意图，生成一批评论。
- 基础客服 = 读用户问题，生成一句得体的回复。
- 整理数据 = 读原始表格，生成规整的结构。

这些岗位过去要人，不是因为它们多难，而是因为**除了人，没有别的东西能读得懂、写得出**。第5章讲过，这道门槛塌了。门槛一塌，「非人不可」就变成了「人可以，一堆 AI agent 也可以」。

于是一个人 + 一批 AI agent（能自己读任务、调工具、一步步把活干完的 AI 程序，你给目标它给结果）能干过去一整个小组的活。这就是 人效（人均产出，一个人能顶多少活）的跃升——不是这个人变强了，是他背后挂了一排随叫随到的执行器，他从「自己干」变成了「调度一群 AI 干」。

## 从「囤人力」到「调度人力 + 机器」

这里发生的，是组织形态的一次相变，值得单独说透。

过去养人，本质是把「产能」买成了**固定成本**：人头在那儿，工资就在那儿。现在，产能可以从模型按次租——多处理一份工单，多付一笔 API 钱；不处理，不付钱。**固定的人头开销，变成了可变的算力开销。**

大白话

说人话：这跟第4章那根「成本」螺丝、第11章那笔「单位经济」的账是同一根逻辑，只是这次算的是人。过去你为「以防万一要处理很多活」而囤的那些人，现在可以退掉，改成「真有活了再去租算力」。旱涝保收的工资单，变成了随业务量涨落的账单。

组织于是从「囤人力」变成了「调度人力 + 机器」。这才是 一人公司（solo company，一个人或极小团队，却能服务大量用户、做出过去要几十人才做得出的产品）第一次成为可能的真正原因——**不是工具更顺手了，是「产能」这件东西，从必须雇人，变成了可以按需租。**

一个人做不出过去几十人的产品，从来不是因为他不够聪明，而是因为一天只有 24 小时，产能是他一个人身体的上限。现在产能可以外挂、可以并行、可以按需扩，这个上限被拆掉了。这也顺手接住了第8章那句「夹层在变薄」——中间那层靠人力堆起来的产能，正在被抽走。

## 硬边界：哪些活，模型怎么都替不掉

到这儿必须踩刹车。如果只讲上面这些，这一章就成了「一人公司乌托邦」爽文。真实世界有几堵墙，模型撞不穿。把它们讲清楚，比讲那些缩小的部分更重要——**因为分界线就画在这儿。**

### 一、需要「担责任、建信任、跑线下关系」的活

企业级销售、合规审查、供应链谈判、线下履约——这些活的核心不是「读—理解—生成」，是**有一个人格化的主体，站出来负责、被信任、被追责**。一家大公司愿意签下千万级合同，买的一半是产品，一半是「出了事我知道找谁、他跑不掉」。AI 生成得出合同文本，但签不了字，也蹲不了号。这条呼应了第9章那道「结果可靠性」护城河——责任本身，模型给不了。

### 二、需要「真实世界原子操作」的活

制造、物流、医疗操作、维修——这些活要动的是**原子，不是比特**。模型再强，它动的永远是屏幕里的字节；把一个零件装上、把一箱货搬到车上、给病人做一台手术，这些需要物理

世界里真实的手和真实的身体。

大白话

说人话：模型是一个只会读写、不会动手的天才。凡是最后要落到「有人真的去做一件物理的事」的环节，那个人就省不掉。这是这一章最硬的一条边界——比特可以无限复制，原子不行。

不过这条得加个时间戳：**就现在而言**成立。机器人和具身智能正在把一部分原子操作也慢慢「比特化」——让机器的手去动真实的零件、货箱。当前这堵墙还立着，是因为具身智能的通用化还没到位；哪天它到位了，这条边界该往回收多少，就得重新画。别把「现在动不了原子」写成一条永远的定律。

### 三、需要「长期攒下的专有资产」的活

独家数据、行业资质、牌照、品牌信任——这些是拿时间熬出来的，不是拿算力换来的。你租得到模型的聪明，租不到一张十年才批下来的牌照，也租不到客户对一个老字号攒了二十年的信任。这正是第9章「私有数据」那道墙的延伸：模型进不去的东西，一人公司也变不出来。

### 四、协调本身的成本

这条最容易被忽略。人多了要管理，这是负担；但反过来，**有些大事，本质上就需要很多人当场协同**——造一架飞机、办一场大型演出的现场、跑一个覆盖全国的线下网络。AI 能帮每个人干得更快，但它减少不了「这件事需要 N 个人协调配合」这个根本需求。协调不是「读—理解—生成」，它是人和人之间的实时耦合，模型压不掉。

## 把分界线画出来

四条边界摞起来，其实是同一句话的四个侧面：

> 能被「读—理解—生成」覆盖的工作会被压缩；需要责任、原子操作、信任、协调的工作，仍然需要人。

所以「公司会变小」不是一句普适的预言，它有精确的适用范围：**一家公司里「读—理解—生成」占比越高，它就缩得越狠；越是压在责任、原子、信任、协调上，它就越缩不动。**

## 一个反直觉的推论：更多小公司，和更强巨头，会同时发生

顺着往下推，会撞出一个看似矛盾的结论。工程师爱较真，这里正好拿因果捋一遍。

一方面，**公司数量会变多**。创业门槛塌了——过去要凑齐一个小组、烧一大笔钱才能启动的事，现在一两个人挂上一排 agent 就能干起来。能开公司的人变多了，公司自然变多。

> 大白话
>
> 但别把「变多」听成「活得好」。门槛低了，进来的多，淘汰也快——同一扇门谁都进得来，你能一两个人干起来的事，别人也能，竞争只会更惨烈。所以变多的是**数量**，不是**存活**：一茬一茬冒出来，也一茬一茬倒下去，大多数小公司活不长。

另一方面，**头部会更集中**。为什么？因为赢家也在用同一套小团队杠杆。一个卡住了分发、握着独家数据、担得起责任的头部玩家（第9章那四条护城河），现在能用**比过去小得多的团队**，服务多得多的用户、吃下多得多的市场。规模不再需要靠堆人换，赢家于是赢得更省、也更狠。

> 大白话
>
> 但注意，别把巨头也想象成能「一人化」。它缩的，是上面说的那些「读—理解—生成」执行器岗位——文案、初筛、一审、基础客服这类。它缩不掉的，恰恰是本章后面列的那几堵硬边界：真正吃下大市场，往往还是得有相当规模的销售去谈、合规去审、线下去履约、以及一大群人当场协调。**巨头缩的是执行器，不是责任、协调、线下这些岗**——所以它变的是「同样的产出用更少的人」，而不是变成一个人的公司。

大白话

说人话：「更多小公司」和「更强巨头」不打架，它们是同一件事的两头。被这股力量挤扁的，是中间那一层——那些「靠雇很多人堆出规模、但没有一条独特护城河」的中型公司。它们的存在理由本来就是「人多、能干活」，可现在「能干活」变便宜了，它们上不去、下不来，最先塌。

这正呼应第8章那个「薄夹层」，也给第15章「谁死谁强」埋下伏笔：死的往往不是最小的，也不是最大的，而是那个靠人力撑规模、却没独特地形的中间层。

## 回到那个乐团

用一个比方收口。过去要办一场交响乐，你得养一整个乐团——每个乐手都是一笔固定成本，不管今晚演不演，工资照发。乐团的规模，就是你产能的上限。

现在，一个独奏者配上一套顶级的 AI 乐手，随叫随到、按曲目付费。他第一次能一个人撑起一台交响乐——过去要几十人的声部，现在从他背后的机器里现取。这就是一人公司。

但别忘了台上那些真正需要人的部分：现场的杂技、需要演员之间当场默契配合的段落、要有人站出来对整场演出负责的那个总监。这些，AI 乐手替不了。**能被谱子写下来、按下去就响的声部，可以外包给机器;需要现场的身体、现场的信任、现场的配合的部分，还得是人。**

## 给读者的一把尺子

这一章最后落一句能直接上手的话。下次你拿到一家公司，别急着看它人多人少，先看它的**人员构成**，问一句：

> 这些人干的活，有多少是「读—理解—生成」？

占比越高，这家公司越会被重构——它的人头正站在模型的正下方，随时可能被换成一行算力账单。占比越低、越压在责任、原子、信任、协调上，它就越稳。

组织变小这件事，讲到这儿逻辑就闭合了。但小公司也好、小团队也好，它要真跑起来，得先回答几个很俗但很致命的问题：启动它要多少钱？怎么烧？靠什么融资？——「人」的算

术算完了，下一章我们算「钱」的算术。

# 第十三章 · 创业和融资的新算术

## 第 13 章 · 创业和融资的新算术

上一章算完了「人」的算术：产能能按需从模型租，公司缩小了。这一章接着算「钱」的算术。同一股力量往融资这头一推，会撞倒一个用了二十年、几乎被当成常识的剧本——**「先融一大轮、招一堆人、慢慢建产品、用规模和先发建护城河」**。它不是被人质疑倒的，是被成本结构悄悄抽掉了地基。

先把话说死，免得读成标题党：**不是「VC 死了」，是「钱这件武器的用法变了」。**钱还有用，但它能买到和买不到的东西，跟五年前不是一张清单了。整章都在重画这张清单：老剧本为什么失灵，投资人现在该怕什么、该赌什么。

## 启动成本塌了：既是好消息，也是坏消息

先看最直接的一根螺丝。

启动成本：从零到做出一个「能用、能给人看的产品」，你得先烧掉多少钱、多少时间。过去做个软件产品，这个数字很吓人——要凑一支工程团队，几个月到几年，烧掉一大笔种子钱（公司最早期、还没啥收入时融的第一笔钱，专门用来把产品从想法做到能演示），才能有个像样的 demo。

现在，接着第 5、12 章的逻辑：核心能力从模型租，执行器岗位挂给 agent，一两个人几周就能做出一个能用的产品。启动成本从「一大笔种子钱」塌到「几乎不要钱」。

> 大白话
>
> 说人话：过去做产品像盖房子，得先备砖、请工人、打地基；现在像搭乐高，一个人几个晚上就拼出个能玩的东西。入场门票从几十万一张，变成几乎白送。

这里有个容易被讲成爽文、其实是双刃的关键点，得钉住：**门票便宜，是对所有人便宜。**你能几周启动，你的对手、你的模仿者、明天冒出来的陌生人，也都能几周启动。于是第一条反直觉结论就出来了：**启动变容易，不等于活下来变容易，反而更难。**这呼应第 12 章那句

——门槛低了，进来的多，淘汰也快。你省下的那笔启动钱，换来的不是安全，是一个「谁都能进、进来就贴身肉搏」的赛道。对投资人，这里还埋着本章后面最要命的一个陷阱，先记着。

## 老融资剧本为什么失灵

风险投资（VC，venture capital）：投资人拿一笔钱换公司一部分股份，赌它未来暴涨、卖股份时赚几十上百倍。因为大多数会失败，它靠「少数几家暴涨」覆盖「大多数归零」——所以必须找「能长到极大」的公司，并想尽办法帮它长大。由这个赌法，长出经典三段式：

1. **融一大轮**——先把弹药备足；
2. **招人建团队**——把钱变成产能，快速把产品和市场做出来；
3. **用规模和先发建护城河**——比对手更早更大，靠先发优势和团队规模把地盘焊死。

这套打法在 SaaS 时代天衣无缝，因为每一节都踩在当时的成本结构上。现在，第一节和第三节同时断了。

**第一节断在「不那么烧钱了」。**启动成本一塌，很多公司根本不需要那么多钱就能起步——甚至能bootstrapping（不靠外部融资，用自己赚的收入一点点滚动着长大，字面是「拽着自己的鞋带把自己拎起来」）。一家公司能自己养活自己、自己滚动增长，「融一大轮」就从必需品变成了选项，甚至变成负担——拿了钱就得给出暴涨的回报，反而把一条本可以稳稳赚钱的小生意，架上了「不暴涨就算失败」的火。

**第三节断在「先发+堆人建的护城河，正是被抹平的那种」。**这是最狠的一刀。经典打法靠什么把规模变成护城河？靠功能比对手多、团队比对手大。可第 5、8 章已经证过：功能和能力如今是从通用模型租来的，谁都租得到；靠堆人堆出来的产能，第 12 章说了，正在被抽走。**钱砸出来的「功能多、团队大」，恰恰是平权最先抹平的东西。**

> 过去钱能买到护城河，是因为护城河由「功能数量」和「团队规模」构成——而这两样，正好是今天钱买了也守不住的。你花一个亿堆的功能，别人租个模型几周就追平了。

## 投资人该怕什么：三个新陷阱

把创业者视角切成投资人视角。同一套变化，落到「我这笔钱投得值不值」上，藏着三个过去不存在、现在能让一笔投资血本无归的陷阱。

### 陷阱一：护城河是「能力型」的，模型厂商一升级就没了

你投的应用，如果核心竞争力是「它比别人更会用模型、功能更强」，那第 7、8 章那把刀就悬在头顶：**这种「能力型护城河」，模型厂商升一级、或干脆自己下场，就蒸发了。**你投的是一家公司，赌的却是模型厂商未来几年「刚好不往这个方向走」——这不是护城河，是运气。

### 陷阱二：毛利比经典 SaaS 薄，却按老 SaaS 倍数估值

第 11 章算过：大模型应用每一单都压着 token 料钱，毛利从 SaaS 那种八九成被啃到四五成。估值倍数（用营收或利润的多少倍来给公司定价，比如「按营收的 20 倍估值」）本该跟着毛利走。可现实里，很多投资人看见「软件公司 + 快速增长」，条件反射就套上 SaaS 那套高倍数。**用高毛利生意的倍数去给薄毛利生意估值，必然高估。**同样一块钱营收，长在零边际成本上值几十倍，压着料钱可能只值几倍——认错了形状，价就付贵了。

### 陷阱三：「增长很快」可能只是「启动很容易」的假象

这条最隐蔽，也最要命，正好接上开头埋的陷阱。VC 一向把「增长快」当成好公司的铁证——涨得快，说明有需求、有本事。但在启动成本塌了的世界里，这个信号被污染了：**启动容易，曲线就陡；但正因为对所有人都容易，这条陡曲线随时会被下一个同样几周做出来的对手抹平。**

> 大白话
>
> 说人话：过去能快速起量是本事，因为起步很难，冲得出来的都不简单；现在起步谁都会，一个几周做出来的产品自然也能几周就涌进一批尝鲜用户。快速增长可能只是「太容易做、太容易试」的副作用，不是「能守住」的证据。

快速增长不再自动等于能守住。投资人得学会问一句新问题：这增长，是因为「别人做不出来」，还是因为「谁都做得出来、所以谁都来试一下」？

## 投资人该赌什么：赌位置和关系，别赌技术和人数

怕完了得说赌什么，否则又成唱衰。答案第 9 章已备好：**赌那些模型抹不平的护城河。**第 9 章列过四道墙——分发、独家数据、深工作流、结果可靠性；再加第 12 章那条：能对结果站出来负责、扛得住责任的团队。这几样有个共同点：**都不是「技术」和「人数」，而是「位置」和「关系」。**

- **分发**——你已经站在用户面前，别人挤不进来（位置）。
- **独家数据**——你手里有别人拿不到、模型也没见过的数据（资产）。
- **深工作流**——你嵌进客户离不开的流程里，拔出来伤筋动骨（关系）。
- **结果可靠性 + 扛责任的团队**——你敢对结果签字、出事扛得住（信任）。

> 老打法赌「谁技术强、谁人多」；新打法赌「谁站的位置好、谁有别人没有的关系和资产」。前者模型能抹平，后者模型碰不到。

这也顺手校正了看人的眼光。过去尽调爱看团队里有多少顶尖工程师、代码有多牛；现在这些是租得到的通用能力，含金量在掉。真正该问的是：**这支团队有没有别人没有的分发、数据、流程、信任？**——不再问「你们多能干」，而是问「你们占了什么别人占不了的地形」。

## 估值倍数怎么调

把上面三个陷阱翻成一条能上手的准则。别再无脑给软件的高倍数了——过去给软件估值几乎是「营收 × 一个高倍数」这么简单，因为毛利和护城河默认都好。现在这个默认没了，估值时得先过三道筛：

- **看毛利结构**：每一块钱营收含金量是八九成还是四五成？薄的那头，倍数得往服务业、外包业靠，而不是往软件业靠。
- **看护城河类型**：是「能力型」（模型一升级就没）还是「位置/关系型」（第 9 章那四道墙）？前者要打折，后者才配溢价。
- **看增长的「可守性」**：这条曲线是「别人做不出来」撑起来的，还是「谁都能做、都来试」堆出来的？守得住的增长值钱，守不住的是幻觉。

大白话

说人话：三样都过关，才配老 SaaS 那种高倍数；缺一样，价就得往下打。

## 反直觉推论：钱作为武器，威力下降了

把整章的力往一个点上收，会撞出一个让老 VC 不舒服的结论。过去有句几乎当真理的话：**「谁融得多谁赢。」**钱多，就能砸市场（补贴烧到对手扛不住）、砸人才（高薪挖光顶尖工程师）、砸速度（招一堆人比对手先做出来）。融资额本身，就是一种压倒性的武器。

可现在这三样砸法一个个失灵：启动不缺钱了，砸速度砸不出领先，因为对手也能几周起步；能力能租了，砸人才砸不出独占，因为顶尖能力挂在通用模型上人人可取；市场里全是几周就能冒出来的小团队，砸补贴砸来的用户，转头就被下一个免费的替代品带走。**钱能砸出来的优势，越来越容易被一个拿着模型的小团队抹平。**

但——这正是不能写成「钱没用了」的地方。钱依然是压倒性武器，**前提是它砸在「只有钱才买得到」的东西上**：独家数据（买断别人拿不到的数据源）、分发渠道（拿下别人挤不进的入口）、监管牌照（用钱、时间、合规熬出的资质）、线下履约网络（铺一张模型够不着、动真实原子的网，呼应第 12 章「原子不是比特」那条硬边界）。这些贵、慢、重——而「贵、慢、重」，在一个启动变得又快又轻的世界里，第一次成了护城河而不是包袱。

> 融资的意义，从「买产能」转向「买那些模型给不了、只有钱能撬动的稀缺资源」。过去融钱买「更快、更多」，现在这些能租了；钱该改去买「更独、更难」。

## 回到那家餐馆

用一个开店的比方收口，接上第 11 章那家「要算料的普通店」。过去开餐馆是一场重投入的赌博：先砸钱装修、雇厨师、备一屋子货，才能开张。**本钱厚本身就是最大的优势**——谁装修更气派、厨师更多、备货更足，谁就压得住对手。融资，就是这场赌博的入场券，票价越高越占便宜。

现在，共享厨房、外卖平台、预制菜一摆开，一个人花很小的成本就能开张试菜。「谁本钱厚」不再是决定性优势——因为开张谁都做得起。真正决定生死的换成了两样：**你选的赛道好不好，你有没有别人没有的东西**——一个抢手的地段（分发）、一条独家的供应链（独家数据）、一块砸不烂的老招牌（信任与牌照）。钱不该再砸在「把店开起来」上，那太便宜了；钱该砸在「拿下地段、锁死供应链、熬出招牌」上——那些，才是一个人一台模型怎么都变不出来、只有钱能撬动的东西。

创业和融资的新算术，到这儿就闭合了。但整章都站在创业者和投资人这一侧——都是新玩家怎么进场、怎么下注。牌桌另一头，还坐着一群手握重金、客户、渠道的**大企业**。同样这套变化摆到它们面前，反应却常常出人意料地慢，甚至干脆按兵不动。这是为什么？下一章，我们坐到巨头那一侧，算算它们的账。

# 第十四章 · 企业为什么按兵不动

## 企业为什么按兵不动

前面几章都站在卖方那头看：做软件的成本怎么变了（第4章）、护城河搬去了哪（第9章）、一个人怎么撑起一家公司（第12章）、拿这套东西怎么去融资（第13章）。这一章换个座位——从卖软件的，挪到**买软件的**。

因为有个巨大的反差，绕不过去：推特上天天在喊「AI 颠覆一切」，可你真去一家大银行、大制造厂、大保险公司里问一句「你们上了没」，得到的答案往往是「在评估」「在走流程」「明年吧」。**C 端天天革命，B 端三年不动。**这一章负责把这条裂缝讲透——不是靠「大企业笨」「官僚保守」这种偷懒话，而是逐条拆清楚：那个「慢」，到底是由什么零件组成的，以及一个反直觉的结论——这慢，恰恰是新护城河本身。

## 先分清：to C 和 to B 是两个世界

to C（to Consumer）＝把东西卖给个人。你想用某个 AI 工具，掏出手机，下载，输入卡号，五分钟后开始用——**决定用不用的，就你一个人**。

to B（to Business）＝把东西卖给企业组织。这里没有「一个人说了算」这回事。想让一家公司用上一个新工具，得走招标、评估、法务、安全、预算——**要一堆人依次点头，缺一个都推不动**。

> 大白话
>
> 说人话：to C 是一个人下决心，to B 是一群人凑同意。前者的阻力是「我愿不愿意」，后者的阻力是「这一长串关卡里，有没有哪一关卡住」。C 端叙事跑得飞快，是因为它只需要说服一个人；B 端叙事慢得像卡住，是因为它要穿过一层厚厚的组织。这一层厚东西，就是这一章要拆的主角。

## 把「慢」拆成五个零件

企业不动，不是一个原因，是五个机制叠在一起。工程师爱较真，我们一条条看它到底卡在哪。

### 一、采购流程：以季度为单位的关卡

采购流程：大企业买任何软件，都不是「点一下购买」，而是一条流水线——先招标（把需求发出去，让几家来投标）、再评估（技术选型、比价）、过法务（审合同条款、数据归属、违约责任）、过安全（做安全审查，看会不会引入风险）、最后走预算审批（这笔钱谁批、从哪个池子出）。

这条线天然以**季度、年**为单位跑。不是哪个环节故意拖，是每一环都要真人签字、真部门背书。你一个人下载 App 是「秒级」，企业采购是「季度级」——这中间差的不是决心，是**制度的采样频率**。C 端按天迭代的东西，撞进一个按季度采样的系统里，看起来自然像卡死了。

### 二、集成：买模型容易，缝进流程难

集成：新工具不是孤立运行的，它得接进企业已经在跑的那一整套东西——账号权限体系、数据管道、还有一堆遗留系统（legacy，十几年前搭的、又老又乱、文档早丢了，但核心业务还压在上面跑的旧软件，动一下都怕它塌）。

这是最被 C 端叙事低估的一条。买一个模型能力，是最容易的一步——难的是把它**缝**进去：数据从哪个库拉、拉之前要不要脱敏、结果回写到哪张表、要不要经过哪几级审批、和那个谁也不敢碰的遗留系统怎么对接。这活又脏又具体，和模型聪不聪明毫无关系。

> 大白话
>
> 说人话：这正是第9章那条「工作流嵌入」护城河的买方视角。卖方觉得难缝，买方也觉得难被缝进来——同一件脏活，两头都嫌重。模型能力可以一夜平权，但「缝进这家公司这套二十年老系统」这件事，一天都省不掉，得一个字段一个字段手工对。

## 三、数据合规：谁能保证数据不出门

企业手里最值钱、也最怕出事的，是数据——客户名单、交易记录、商业机密、受法律保护的个人信息。它们最大的恐惧是：**把这些喂给一个外部模型，会不会泄露，会不会被拿去训练，回头从别人的对话里吐出来。**

于是就有了 数据合规这道关：数据能不能出公司的门、出了要满足哪些法规、能不能被第三方看到。谁能给出「数据不出门」的答案，谁才进得了门。答案通常叫 私有化部署（把模型直接装进企业自己的机房或私有云里，数据在自家墙内闭环流转，一个比特都不外流）。

这条直接把很多「只能调云端 API」的玩家挡在门外——你模型再强，只要数据得离开企业的墙，一大批受监管的行业（金融、医疗、政府）连谈都不跟你谈。**会做私有化，本身就是一张入场券。**

## 四、可靠性与责任：出错要有边界、有人担

模型会胡说（幻觉，第7章讲过）。在 C 端，胡说一句顶多是个笑话，用户自己一眼看穿。在 B 端，一次胡说可能是一起合规事故、一笔真金白银的赔付、一场上了新闻的事故。

所以企业要的从来不是「平均很聪明」，而是 可靠性与责任：出错的概率有没有边界、错了能不能被兜住、出了事**有没有一个主体站出来负责**。这正接住第9章那条「结果可靠性」护城河——裸模型只会生成，不会负责；API 条款里白纸黑字写着「不保证输出正确」。而企业要签的合同里，必须有人敢在「我为这个结果负责」那一行签字。这份没人愿签、模型也签不了的责任，恰恰是应用层能捡起来的最硬的一道墙。

## 五、变革成本：组织越大越沉

就算前四关全过了，还有最后一堵墙，在企业内部：改流程本身是有成本的。要重新培训一批人、要动某些人手里的既得利益（一个流程被 AI 接管，意味着有人的活、有人的权没了）、要承担「换了新的万一更糟」的试错风险。

大白话

说人话：这跟第4章那根「成本螺丝」是同一种账，只是这次算的不是钱，是组织的惯性。一家公司越大，牵动的人越多、动的利益越多、试错一旦翻车赔得越狠——所以它天然会慢。这不是笨，是体量决定的物理性质：越重的东西，越难拐弯。

## 骨架：这些「慢」，恰恰在建新护城河

把五条摞起来，很容易读成「大企业真麻烦」。但这一章真正的骨，是把这句话翻个面：

> 正因为落地又慢、又脏、又难，谁能把这些脏活干成，谁就有了模型厂商和小团队都跨不过去的墙。

顺着想一遍就通了。假如落地是「秒级、干净、零门槛」的——就像 C 端下载个 App——那谁都能进，模型一平权，壁垒瞬间归零。可现实恰恰相反：落地要熬过采购的季度周期、缝进老系统的脏活、拿下合规和私有化的资质、还要有人敢担责。**每一道慢，都是一道别人也得同样慢慢熬的门槛。**

于是「慢」在这里有了双重身份：它一边拖慢了颠覆，一边保护了熬过这套流程的人。这几样东西——合规资质、私有化能力、和企业老系统缝合的经验、担责的主体——没有一样是模型能力能直接给的。它们对「模型能力平权」**天生免疫**，因为它们压根不是技术问题，是组织性的、非技术的摩擦。第9章说护城河从「技术」搬到了「位置和关系」，to B 这边看得最清楚：护城河很大一部分，就藏在这个「慢」里。

大白话

说人话：C 端比的是谁的模型更聪明，因为除此之外没什么门槛。B 端比的是谁扛得过这套脏活累活——合规、私有化、缝老系统、担责任。模型再强，抹不平这些，因为它们根本不在模型的赛道上。这也顺手接住第12章那句「仍需规模的活」：把这套脏活干成，往往还得靠一支有相当规模的队伍去谈、去审、去缝、去担。

## 一个类比：新药再神，也得过临床和医院

想象一款疗效惊人的新药。它再神，也不能第二天就用到病人身上——得先过临床试验（证明它安全有效）、过监管审批、进医院的采购目录、最后还得医生真的信它、敢开它。这一路好几年。

你可以骂这套流程慢。但它慢，很大一部分是**刻意的安全阀**——药直接进人体，错一次是人命，所以整个医疗系统宁可慢，也要把风险卡在门外。企业采购 AI 是一模一样的道理：AI 直接接管一家公司的核心流程，错一次可能是合规事故、是巨额赔付。那套慢吞吞的采购、评估、安全、合规，不全是官僚空转，有相当一部分是必要的安全阀。

而这里藏着最关键的一层：**会走完这套流程本身，就是门槛。**就像不是谁有个好分子就能把药铺到全国的病床上——你得能跑通临床、能过审批、能进采购、能建立起医生的信任，这套能力本身比分子更稀缺。to B 也一样：不是谁有个强模型就能进大企业，你得能熬过采购、能缝进老系统、能拿下合规、能担得起责。**能走这套流程，就是那道模型平权抹不平的沟。**

## 推论：雷声大雨点小，两头都对

现在回到开头那条裂缝，答案清楚了。

C 端看起来「AI 颠覆一切」，是真的——因为它只需说服一个人，摩擦极薄，价值一推就落地。B 端「雷声大雨点小」，也是真的——不是企业没看见价值，是**价值得先穿过采购、集成、合规、责任、变革这五层组织摩擦，才落得了地**。同一股技术浪潮，撞上薄摩擦就掀翻一切，撞上厚摩擦就被吸得没了声。

> 大白话
>
> 说人话：别再问「AI 到底颠不颠覆」——这问题问错了。该问的是「它撞上的那层摩擦有多厚」。C 端摩擦薄，所以快、所以乱、所以谁都能被替；B 端摩擦厚，所以慢、所以稳、所以熬过去的人反而被这层厚东西护住。慢和快不是矛盾，是同一股力量撞在两种厚度上的两种结果。

而这层厚摩擦，是个双刃的东西：它既拖住了挑战者，也保护了在位者。谁能把这套慢活熬成自己的资产，谁就在 B 端站稳；谁只有一个更聪明的模型、却过不了这五关，谁就永远卡

在「在评估」里。

到底是熬过去的人赢，还是被摩擦护住的老玩家赢，还是根本没人跨得过去——这就是「谁死谁强」的问题了，留到第15章正面算。这里先把这颗种子埋下：**在 B 端，决定生死的往往不是谁的模型更强，而是谁扛得住那个「慢」。**

# 第十五章 · 谁会死，谁会更强

## 审判：拿到一家公司，你站在浪的哪一侧

前面十四章，我们一件一件地把工具打磨出来：边际成本从零变正、能力平权、界面贬值、套壳尺子、薄夹层最危险、四条新护城河、席位收入的自噬、to B 的"慢"是护城河。现在把这些工具拧成一句话，然后拿它去审判。

审判不是骂人。审判是分类——给你一家公司，你能判断它的核心价值站在浪的哪一侧。

## 一句话判据

把所有尺子拧成一句话，就是这个问题：

> 这家公司的核心价值，是**能力型**，还是**位置·关系·责任型**？

能力型：它值钱是因为它"会做某件事"——读、写、总结、转换、分类、生成。这类能力是通用的，模型平权就能抹平。今天你比别人做得好，只是因为你早做了三年、界面顺手、模板攒得多；这些优势模型一来就归零。

位置·关系·责任型：它值钱不是因为它"会做"，而是因为它"在那儿"、"握着别人拿不到的东西"、"敢为结果签字"。分发入口、独家数据、深嵌工作流、对结果担责、组织性壁垒——这些模型抹不平，因为它们根本不是"能力"问题。

大白话

能力型的价值是"租来的"——你租了一段本事，模型现在把这段本事白送给所有人。位置·关系·责任型的价值是"长出来的"——它扎在真实世界的入口、数据、信任和责任里，模型再聪明也拔不动。

记住：一个能力越通用、越接近"任何人对模型说句话就能得到"，它就越危险；越是绑死在某个具体位置、某份独家数据、某条脏活流程上，它就越安全。下面用这把尺子逐类过一遍。

## 会被吃掉、承压的那一侧

### 一、薄工具型 SaaS

核心功能就是"用规则或模板，做一件通用的读写转换"，且背后没有独家数据、没有深工作流。通用文案生成、通用总结、简单表单、格式转换、通用润色——这些是模型的原生能力，你写的规则模板，模型内部早就有了更好的版本。

更致命的是第七、八章讲的：它离通用能力太近，模型厂商顺手就覆盖了。你不是被某个对手打败，你是被平台的默认功能顺路碾过。`value = 界面 + 提示词工程`，而这两样都在贬值。

### 二、靠"界面好用、功能齐全"活着的

底层能力可平权，切换成本只靠"用户用熟了这个界面"。第六章说过：当交互从"你学会点哪个按钮"变成"你说人话它就懂"，界面本身贬值。界面熟练度这种切换成本，是最先被对话式交互溶解的那种。

> 大白话
>
> "功能齐全"曾经是护城河——一个工具做一百件事，对手要追得做一百件。但当这一百件事的底层都是通用能力，齐全就不再稀缺，谁都能一夜之间齐全。

### 三、卖"席位数"、而 AI 让每席位更能干的

第十章的自噬逻辑：你按人头收费，你的产品让客户每个人干三个人的活，客户于是裁掉三分之二的人——你的收入基数被你自己的产品侵蚀。你越成功，你的计价单位越少。这不是被对手吃掉，是被自己的定价模型反噬。

### 四、夹在中间的中型公司

第八、十二章的薄夹层放大版：上不着天（没有平台级的分发和数据），下不着地（没有深嵌到某条脏活流程里），中间那点技术领先又被平权抹平。它既没有大厂的资本纵深去自建模型能力，又没有小而深的公司那种"就是这条流程离不开我"的黏性。夹层最危险，中型的夹层公司尤其危险。

## 会更强的那一侧

### 一、握着独家数据、且把数据喂给了模型的在位者

第九章的"独家的墙"。模型能力是公共的，但模型需要上下文才能干活，而最好的上下文是别人拿不到的私有数据。谁握着独家数据，谁就能让同一个公共模型，在自己的场景里跑出别人复制不了的结果。

注意后半句——**且把数据喂给了模型**。光坐在数据上不算数,见下面的辩证。

### 二、卡住分发入口、装机量、用户信任的

第九、十三章。已经在几亿人手机里、已经是某件事的默认入口、用户已经信任的——对这些公司，AI 是给现有产品加涡轮，不是掀桌子。用户不会为了一个更聪明的模型，去离开一个已经在用、已经信任、已经绑好数据的入口。分发是把公共能力变成私有优势的通道。

### 三、深嵌企业工作流、能干 to B 脏活的

第十四章："慢"是护城河。合规、私有化部署、和一堆老系统做集成、满足行业审计要求——这些活又慢又脏，恰恰因为慢和脏，模型平权抹不平它。模型让"写代码"变便宜了，但没让"通过某国金融监管的现场审计"变便宜。谁扛住了这些脏活，谁就站在了模型够不着的地方。

### 四、能对结果负责、在高风险场景扛住可靠性的

医疗、金融、法律。模型能生成一份诊断建议、一份合规意见、一份合同——但它不能**签字**，不能承担签错了的后果。在这些场景里，客户付费买的从来不是"生成"，是"有人为这个结果担责"。第十二章讲过：需要责任的活，模型抹不平，因为责任不是能力，是主体。

**五、需要真实世界原子操作、线下履约的**

第十二章：模型动比特，不动原子（就现阶段而言）。要有人上门、要把货真的搬过去、要在物理世界完成一次交割——模型再强，也得有人把这一步落地。握着线下履约网络的，模型是它的工具，不是它的对手。

## 关键辩证：别写成非黑即白

上面像是把公司分成两堆，但真实世界没这么干净。三条必须记住：

**第一，同一家公司可能一半被吃、一半更强。**它的薄工具业务在被平权碾过，同时它的独家数据业务在变强。审判的对象不是"公司"，是"这家公司的每一块核心价值"。

**第二，护城河是"潜在的"，不会自动兑现。**握着独家数据不等于赢——你得*主动把数据喂给模型*，把它变成别人复制不了的上下文。卡着分发入口不等于赢——你得*主动把 AI 装进你的产品*，给它加涡轮。在位优势是一块没上膛的护城河，你不激活它，反应更快的挑战者会绕过你、用 AI 把你在位的那点优势掀翻。历史上最危险的位置，就是"手里有牌但坐着不打"的在位者。

> 大白话
>
> 独家数据是弹药，分发是炮管，AI 是火药。三样都有还不开火，你只是个抱着军火库睡觉的人——迟早被更饿的人偷家。

**第三，给两个反直觉的例子。**

- **看起来稳、其实危险**：一家功能极其齐全、界面口碑极好、用户量很大的工具型公司。看着稳如泰山，但如果它的底层全是通用能力、切换成本只靠界面熟、又没握住独家数据和分发——它稳的全是正在贬值的东西。用户量大不等于护城河，用户为"工具"来的用户量，模型一句话就能召走。
- **看起来危险、其实稳**：一家又土又慢、界面像上个时代、增长平平的 to B 公司。看着要被时代淘汰，但如果它深嵌在客户的合规流程里、握着十年攒下的行业私有数据、还为结果签字担责——它土的地方恰恰是模型够不着的地方。它慢，但它站在浪打不到的高地上。

## 自检清单：拿到任何一家公司，套这五问

这是本章要交到你手上的东西。任何一家公司，问这五个问题：

1. **核心能力是租的还是独家的？**它会做的事，是任何人对模型说句话就能得到的通用能力（租的），还是绑死在独家数据、独家位置上的（独家的）？
2. **价值在界面，还是在数据/流程/责任？**如果把它漂亮的界面剥掉，只留下功能，它还剩什么？剩得越多越稳。
3. **客户为它付费，是为"工具"还是为"结果和信任"？**为工具付费的，会被更便宜的工具召走；为结果和信任付费的，不会。
4. **它握着分发入口吗？**它是用户完成某件事的默认入口，还是需要用户特意想起来才打开？
5. **它的护城河已激活，还是只是潜在？**它有没有主动把独家数据喂给模型、把 AI 装进产品、把在位优势变成实弹？坐着不动的护城河，等于没有。

五个问题答完，你不需要任何内部消息，就能判断一家公司站在浪的哪一侧。答案越偏向"独家、数据/流程/责任、结果和信任、握着入口、已激活"，它越站在安全的那侧；越偏向"租的、界面、工具、没入口、只是潜在"，它越站在被吃掉的那侧。

但审判只是第一步。判断一家公司站在哪侧，靠的是看清它的真实结构；而看清真实结构，最大的敌人是漫天飞的叙事和炒作——每家公司都会把自己说成握着独家数据、卡着分发入口的那一种。下一章，我们讲怎么把真变化从叙事幻觉里拆出来：当所有人都在喊"这次不一样"，你怎么分辨哪句是真的。

# 第十六章 · 真变化与叙事幻觉

写到这儿，这本书要证的那件事已经证完了：SaaS 二十年繁荣，地基是「边际成本近零 + 订阅锁定」；大模型第一次让「多服务一个用户」重新要花钱，这根钉子一松，上面那栋楼的每一层——能力、界面、护城河、定价、组织——都跟着晃。前面十五章，是我一根一根螺丝拆给你看的过程。

这最后一章，我不想再拆新东西了。我想把这一整套逻辑，拧成几把你能揣进兜里、随身带走的**尺子**。因为你合上这本书之后，真正的考验才开始——明天又会冒出一个新的 AI 热词，后天还有一个。你需要的不是记住我的结论，是能自己称重。

## 先说清楚你在跟什么东西打架：叙事

叙事（narrative）：一个讲得动听、能融到钱、能带货的故事——但它未必对应任何真实的机制变化。它可能对，也可能只是个好听的壳。

> 大白话
>
> 说人话：叙事就是「PPT 上那句让你热血上头的话」。它不一定是骗子，很多时候讲的人自己都信。问题在于——一个故事讲得好不好，和它背后有没有真发生一件事，是两码事。这两件事经常被搅在一起，你得学会把它们拆开看。

AI 这个领域，叙事密度高得吓人。热词一个接一个，每个都配一套「这次不一样」的说辞，多数三个月就凉。作为一个爱较真、烦空话的人，你早就受够了「颠覆」「革命」「范式转移」这类词——它们说了等于没说。

但注意，我不是要教你当个见啥都撇嘴的犬儒。犬儒和轻信一样蠢，只是蠢得更体面。你要的是一套**能称重的工具**：把叙事放上去，看它到底压下去几两真东西。下面三把尺子，是全书的浓缩。

## 第一把：边际成本尺

> 这件事，让「多服务一个用户」的成本变了吗？往上，还是往下？

这是全书的锚，也是最该先拿出来的一把。别看它讲什么功能、什么体验，直接问：多来一个用户、多处理一次请求，厂商的账单是变了、还是没动？

- **真变**：一个产品，过去多服务一个用户几乎零成本，现在每次回答都要烧一次推理、跑一次昂贵计算——它的毛利结构被实实在在地改写了（第4、11章）。这不是包装，是账本变了。
- **幻觉**：一个老功能，被贴上「AI 版」的标签重新发布，但底下的成本结构一分没动——它多服务一个用户，还是几乎不花钱。这种「AI」是营销词，不是机制变化。

大白话

怎么用：看它的钱怎么流。真的动了地基的东西，一定在成本这一栏留下痕迹——要么多花了推理钱，要么把某件过去要花钱的事变得不要钱了。成本栏纹丝不动，那多半只是换了张海报。

## 第二把：能力来源尺

> 这里的核心「聪明」，是它自己独有的，还是租来的、人人可得的？

这把尺子专治「独家 AI 能力」这类说法。第5章讲过能力平权——通用大模型把一大批过去要专门团队才做得出的能力，变成了一个 API 调用，谁都调得到。那么问题就来了：你吹的这个聪明，究竟是你独家熬出来的，还是你从公共货架上租来、再转手包装的？

- **真变**：能力平权真的抹平了某道旧壁垒——过去某件事非得一支专业团队几年功夫，现在一个通用模型直接干了。旧的护城河被填平，是真的填平了（第5章）。这时候，站在填平后一侧的玩家，优势是真的没了。

- **幻觉**：一家公司大讲自己的「专有 AI 能力」多神，可你稍微一想就发现，调个通用 API、拼几段提示词，一下午就能复现个八九不离十。那不是护城河，那是别人也能领的免费样品。

大白话

怎么用：问一句「我自己周末能不能糊一个差不多的？」。如果答案是「能」，那它的核心聪明就是租来的，不是它的。租来的东西，房东可以租给任何人——包括它的对手。

## 第三把：护城河迁移尺

> 它的优势，是「能力型」，还是「位置·关系·责任型」？

这是三把尺子里最要命的一把，因为它回答的是「守不守得住」。第9章讲过一次护城河的大搬家：技术能力这种护城河正在被平权抹平，真正守得住的，迁到了模型碰不到的地方——分发位置、独家数据、深工作流、担得起的责任。

- **能力型**护城河：「我们模型更强」「我们算法更好」。这类优势正站在平权的正下方，随时会被抹平。
- **位置·关系·责任型**护城河：卡在别人绕不过的分发口子上、握着别人拿不到的独家数据、嵌进客户改不动的深工作流、出了事有个主体站出来担责。这些，模型再强也抹不平——因为它们根本不是「聪明」的问题。
- **真变**：一家公司把护城河真的搬到了模型碰不到的地方——它值钱的不再是「AI 多强」，而是它卡的位置、攒的数据、担的责任。这样的迁移是真的稳固了。
- **幻觉**：护城河明明还建在会被平权的能力上，却被讲成「不可撼动」。听上去很硬，其实建在流沙上——平权的水一涨，就塌。

大白话

怎么用：把它的优势翻译成一句大白话，然后问「这句话，一个租得到同款模型的对手，抄不抄得走？」。抄得走的，是能力型，早晚被平权；抄不走的（分发、数据、工作流、责任），才是真墙。

## 几种典型的叙事幻觉：不是刻薄，是讲机制

三把尺子之外，有几个反复出现的坑，值得单拎出来。我不是要挖苦谁——很多讲这些故事的人是真心相信的。我只想把底下的机制讲清楚，让你一眼看穿。

### 把「demo 惊艳」当「产品可用」

一个 demo 让全场惊呼，不等于它能交付。第7章讲过，demo 到可交付之间，隔着一整条**可靠性工程**的鸿沟——demo 只要在挑好的例子上跑通一次就行，产品得在成千上万个刁钻的真实场景里，次次都不出错、出了错还得有人担着。惊艳是「偶尔对得漂亮」，可用是「稳定地、担得起责任地对」。这中间的距离，往往比从零到 demo 还长。

### 用「增长快」证明「护城河深」

「它涨得这么猛，护城河一定很深」——这是个偷换。第13章讲过，AI 时代**启动**变得极容易，所以「快」只说明门槛低、谁都进得来，它恰恰不能说明「守得住」。同一扇低门槛的门，你冲得进去，对手也冲得进去。快，很多时候是护城河浅的症状，不是深的证据。

### 把「接了个 API」包装成「AI 驱动的革命」

这个用第二把尺子一称就现形。看它离通用能力有多近（第7章）：如果它的「革命性 AI」本质就是调了通用大模型的 API，那它的能力是租来的，护城河的水位就是那个 API 的公开程度——你能接，人人能接。

### 混淆 to C 的热闹和 to B 的落地

叙事最喧闹的地方永远在 to C——推特上的刷屏、发布会的欢呼。但第14章讲过，真正的商业价值要穿过**组织摩擦**：采购流程、合规审查、和旧系统的对接、员工习惯的改变。to C

的热闹是一夜之间的，to B 的落地是以季度和年计的。看到一个热词在网上炸了，别急着判断它在企业里也成了——那是两个时间尺度完全不同的世界。

## 关键的一句：别把真变化也一起扔掉

讲到这儿，我必须踩一脚急刹车，否则你会误会我这本书的立场。

**怀疑叙事，绝不等于否认变化。**这套尺子是用来区分「真变化」和「蹭变化的故事」的，不是用来否定一切的。恰恰相反——这本书从头到尾在说的那件事，是真的：地基真的动了。边际成本从零变正，这不是又一次炒作，这是一次实打实的位移。

所以你面前有两种错误，它们一样致命：

- **把幻觉当真**：见一个热词追一个，被每一套动听的叙事牵着跑，三个月一轮，跑得精疲力尽，什么也没沉淀下来。
- **把真变当幻觉**：因为受够了炒作，从此见 AI 就撇嘴，一句「又在吹」把所有东西挡在门外——结果错过了一次真正的地基位移。这种错误更隐蔽，因为它披着「清醒」的外衣，其实只是另一种偷懒。

这套尺子的全部意义，就是让你**两头都不踩空**：既不被叙事骗，也不被自己的逆反情绪骗。

> 大白话
>
> 说人话：真正的变化，往往不喧哗。它不在发布会的形容词里，不在那些「颠覆」「革命」的大词里；它藏在成本结构、护城河位置、定价形态这些「不性感但要命」的地方。喊得最响的，通常是叙事；真变了的东西，反而安安静静地改着那本没人爱看的账。

## 回到那四个字

整本书拆了这么多层，如果只让你带走一样东西，就是书名那四个字——**边际成本不再是零**。

这是全书唯一的锚。所有的推演——能力平权、界面贬值、护城河搬家、定价从卖席位到卖结果、组织变小——全是从「多服务一个用户重新要花钱」这一个变量长出来的。你记住这一条，别的都可以忘。

因为下一个热词来的时候，我给你的三把尺子，其实都是这一条的分身：边际成本尺，直接问它；能力来源尺，问的是「这份聪明的成本落在谁头上」；护城河迁移尺，问的是「成本结构变了之后，还守得住吗」。三把尺子，一个锚。

我想给你的，从来不是一套「AI 会怎样」的结论——结论会过期，热词会换代。我想塞给你的是一杆**秤**：下次那个讲得天花乱坠的故事摆到你面前，你不必等谁来告诉你它真不真，你自己就能把它放上去，称一称到底压下去几两。

> 拿到结论的人，只能等下一个结论；拿到秤的人，谁来了都能自己称。

这本书讲完了。边际成本这根曲线，抬起头了。剩下的，交给你的秤。