活下来的机器

导读

从泡沫到 AI，硅谷上市软件公司的二十年因果链

2000 年 3 月，纳斯达克站上 5048 点，一屋子只有 PPT、没有收入的公司，市值超过老牌工业巨头。半年后，这个数字腰斩，再半年，再腰斩。烟尘散去，地上躺满尸体。可就在这片废墟里，有几家公司不但没死，还在往后的二十年里滚成了今天你每天都在用的庞然大物——它们卖的东西你摸不着，却让你离不开。

问题是：**凭什么是它们活下来?**

这本书想回答的，就是这一件事。2000 年之后这二十多年，一家美国上市软件公司——靠什么活过寒冬，靠什么滚成巨兽，又靠什么被后来者掀翻在地？我们不挨个讲它们的八卦，不比谁的创始人更传奇。市面上那种"公司百科"已经太多了。这本书要做的是另一件事：把二十年拉成**一条因果链**，让你看清同一出戏，是怎么在不同的年份、换着主角、反复上演的。

> 大白话
>
> 说白了：与其记住"谁在哪年上市了"，不如看懂"为什么每隔几年，牌桌上的赢家就要重新洗一次"。前者是知识点，后者是一副眼镜——戴上它，你能自己往下推。

一条藏在成本里的暗线

如果只让我给你一个抓手，那就是软件的交付成本——把一份软件送到用户手里、并让它跑起来，要花多少钱。

这条线，二十年里降了不止一个数量级：从往光盘上刻录、装箱、快递（`装光盘`），到按月收租的订阅，到租别人的机房按用量付费的云，到揣进每个人口袋的移动端，到把用户行为攒成金矿的数据，再到今天这波把"生成"本身变便宜的 AI。**每当这个成本掉一个数量级，收入怎么收、护城河挖在哪、谁能赢，就会被重新排一次座次。**

上一轮的王者，往往正是被自己那道又深又宽的护城河困死的——因为河还在，桥却已经改道了。这个规律具体怎么咬人、咬死过谁、又成全了谁，我留给你顺着读下去自己撞见。这里先埋个引子：**你会不止一次产生"这剧情我刚才见过"的错觉。**

读完你能带走什么

不是一堆年份和名字。是一种看兴衰的眼力：拿到一家软件公司，你能大致判断它站在哪一波成本浪潮上、护城河是真是假、下一个数量级的降价会不会正好冲垮它。

包括——尤其包括——看懂我们此刻正身处其中的这波 AI。它到底是又一次改朝换代，还是这次真的不一样？读完这本书，这个判断该由你自己来做，而不是听谁喊。

怎么读

时间线是这本书的骨架，每一章都接住上一章留下的疑问，所以**顺着读最省力**——你能亲眼看着一个悬念如何在下一个年代被解开、又如何埋下新的。但每一章也都能单独成篇：哪个年代、哪家公司先勾住了你，就从哪儿翻开。

> 软件的历史，不是一部技术进步史，而是一部"成本崩塌，牌桌重排"的循环史。看懂了这个循环，你看到的就不再是一堆公司，而是一台机器——以及，是什么让极少数玩家，一次次活了下来。

翻页吧。

第一章 · 泡沫破了之后，谁还站着

2000 年 3 月，纳斯达克（美国一个以科技股为主的股票市场，你可以把它想成"科技公司集中上市交易的大集市"）指数摸到了约 5000 点的高峰。谁都觉得它还会往上走。两年半后，它跌到 1200 点上下，蒸发了大约八成。中间没有战争，没有天灾，没有哪条海底光缆被咬断。什么都没坏，可几万亿美元的市值就这么没了。

这本书要讲的二十年，就从这堆废墟上开始。因为泡沫破掉的方式，恰好把一个问题逼到了所有人面前——**软件公司到底靠什么赚到真钱？**在此之前，大家忙着赚一种叫"故事"的钱，没人细想这个问题。等钱烧光了，才发现不想清楚就得死。

先说清楚：泡沫是什么，为什么会破

泡沫，说白了就是价格远远跑到了价值前面。一样东西真正能赚的钱是固定的，但如果所有人都相信"明天有人会用更高的价买走它"，价格就会脱离它本身，自己往上飘。这跟东西好不好、公司牛不牛没关系——关键是每个人都指望着下一个更急的接盘人。

> 大白话
>
> 你可以把它想成击鼓传花。花值不值钱不重要，重要的是你相信鼓还没停、后面还有人接。等某一刻大家同时怀疑"是不是没人接了"，鼓一停，花砸在谁手里谁认栽。泡沫破，就是所有人在同一秒钟不约而同地不信了。

1990 年代末，那朵"花"叫互联网。逻辑听上去无懈可击：全世界的人都要上网，上网就要用网站和软件，谁先占住地盘谁就赢者通吃。于是只要公司名字后面挂个 `.com`，投资人就往里塞钱。一家公司有没有利润没人问，大家问的是——**你有多少用户，涨得多快？**

"烧钱换眼球"：一个听起来很聪明的错误

当时最流行的打法，行话叫"烧钱换眼球"。翻成人话就是：先不管赚不赚钱，用巨额补贴、疯狂打广告，把用户抢到手，把规模做大。等你成了行业老大、对手都死光了，再回过头来慢慢收钱。

这里有几个词得当场讲明白，因为它们是理解整个故事的钥匙：

- 烧钱率（burn rate）：一家还不赚钱的公司，每个月净亏掉多少现金。比如账上有 6000 万，每月净烧 500 万，那它的"命"还剩 12 个月。烧钱率就是这家公司命的倒计时。
- IPO（首次公开募股）：公司第一次把自己的股票拿到股市上卖给公众，一次性拿回一大笔钱。对烧钱的公司来说，IPO 是给命续时间的关键补给。
- 市盈率：股价除以每股一年赚的钱，衡量"你要花多少年的利润才能买回这家公司"。可当年很多明星公司根本没有利润——分母是零甚至是负的，市盈率算都算不出来。于是投资人干脆换了把尺子：不看利润，看用户数、看点击量、看营收增速。**尺子一换，游戏就变味了。**

为什么这个打法在当时说得通？因为它偷偷藏了一个前提：**钱会一直有，命可以一直续。**只要下一轮融资、下一次 IPO 总能接上，你就能永远烧下去，直到熬死所有对手。整套逻辑像一座桥，桥墩就是"随时能拿到下一笔钱"。这个前提在 1999 年成立，在 2001 年塌了。

看两个死掉的样本

光讲道理太干，看两家真实烧死的公司，因果链会清楚得多。

Pets.com：一只木偶狗的葬礼

Pets.com 在网上卖宠物粮和用品。它有一只袜子木偶狗做吉祥物，广告做到了美国最贵的橄榄球超级碗决赛，还上了感恩节大游行。品牌知名度爆棚，用户也真在下单。

问题出在算术上。一袋狗粮又重又占地方，快递费经常比狗粮本身还贵。为了抢用户，它卖得比进货价还便宜——**每卖出一单，亏一单。**卖得越多，亏得越多。它指望用规模摊薄成本，可宠物粮这门生意压根没有"卖得越多、每单越便宜"的魔法：多卖一袋，还是得多付一份仓储和运费。

> 大白话
>
> 这里藏着全书最要紧的一个念头，先埋下：有的生意，做大了会自己变便宜；有的生意，做大了只是把亏损也做大了。分清这两者，几乎就分清了谁能活、谁会死。软件为什么后来那么让人着迷？因为它天生偏向前一种——这是后面十几章要反复回来的暗线。

Pets.com 从 IPO 到关门，只撑了大约九个月。它不是没人喜欢，是它每被人喜欢一次，就多亏一点钱。

Webvan：把菜送到家，太超前的一次尝试

Webvan 想让你在网上买菜，当天送到家门口——这事二十年后确实成了，但在 2000 年是灾难。它没有小步试错，而是一上来就砸下巨资，自建了好几座高度自动化的巨型仓库，铺开好几个城市。**固定开销像洪水一样先冲出来，收入却像滴水一样慢慢来。**

这就把烧钱率拉到了恐怖水平。仓库、冷链、车队、工资，每天一睁眼就是天文数字的支出，而愿意上网买菜的人当时还太少。等泡沫一破、资本市场关门，Webvan 借不到续命的钱了。它烧掉了大约十亿美元，然后消失。它错的不是方向，是**把"以后可能对"的事，用"现在没有的钱"提前干完了。**

桥塌的那一刻

2000 年春天开始，那座"随时能拿到下一笔钱"的桥垮了。触发点有几个：美联储（美国的中央银行，管着钱的松紧）为了压通胀连续加息，钱一下变贵、变紧；同时有些巨头公司的财报露了馅，让市场第一次认真怀疑"这些没利润的公司，真值这个价吗？"

怀疑一旦开始，就会自己滚雪球。投资人不敢再往 `.com` 里塞钱，IPO 的窗口砰地关上。对那些靠烧钱活着的公司，这等于氧气被抽走——它们的商业模式里从来没有"自己赚钱养活自己"这一项，全靠外部输血。血一停，烧钱率就成了死亡倒计时。一批公司在几个月里接连关门，员工连夜搬走电脑，纳斯达克指数一路跌回原点。

关键要看清因果的方向：**不是公司先垮了导致股市跌，是钱先停了导致公司垮。**模式没变，用户没跑，只是"永远有人接盘"这个假设被戳破了。一个把命押在别人钱包上的模式，在别人捂紧钱包的那天，必然出事。

废墟上剩下的人，长一个样

潮水退下去，光着屁股的都被冲走了，站着的是另一类公司。它们不一定最性感，但大都有一个共同点：**要么已经在自己赚钱，要么手里的现金能撑很久，不指望明天有人接盘。**

亚马逊也曾被骂成"最大的泡沫"，但它比 Pets.com 多想了一层：它把烧掉的钱变成了仓库、系统和越用越顺的规模，而不是一次性的广告；等资本市场关门，它咬牙走到了自己能造血的那天。差别不在谁更敢烧，而在**钱烧完之后，有没有留下一台能自己转、越转越省的机器。**

本书的视角，从这里定下

所以，泡沫留给我们的不是"互联网是骗局"这种蠢结论——互联网当然改变了世界。它留下的是一把更冷静的尺子。这本书接下来讲二十年的软件公司，会一直握着这把尺子，只问三个问题：

1. **钱从哪来？**是用户真金白银付的，还是投资人一轮轮输的血？
2. **成本长什么样？**做大之后，每多服务一个客户是更便宜了，还是又背上一份新开销？（这决定了一家公司是雪球还是漏斗。）
3. **护城河靠什么？**是烧钱换来的、停了就没的领先，还是别人抄不走、越久越深的东西？

> 大白话
>
> 一句话记住这一章：泡沫破的时候，被冲走的是那些"故事很好、算术很烂"的公司。软件这门生意接下来二十年的全部精彩，都是围绕一件事展开的——怎么让算术变好看。而算术好不好看，很大程度上取决于一件容易被忽略的事：把软件交到用户手里，到底有多贵。

而在 2001 年，这件事还很贵。那时候你想用一套软件，得买一张实体光盘，装进自己的机器，出了问题自己扛。软件公司卖的是一份份拷贝，卖一份收一次钱，卖完这单就得再找下一单。这种"卖光盘"的模式，天花板到底卡在哪儿、为什么注定要被更便宜的交付方式取代——就是下一章的事。全书的暗线，从那张光盘开始转动。

第二章 · 卖光盘的生意为什么是个天花板

卖光盘的生意为什么是个天花板

上一章我们看到泡沫散去之后，还站着的那批公司大多有一个共同点：他们卖的是软件本身，而且是用一种今天看起来很古怪的方式卖的——把程序刻进一张光盘，装进纸盒，摆上货架，一手交钱一手交货。这一章我们把这个模式拆开看，看清它的成本和收入长什么样。因为这条暗线的起点就在这里：**软件的交付方式一开始非常贵，而这种「贵」决定了谁能赢、能赢多久。**

大白话

一句话概括这一章：老式软件是「卖断」的生意，钱一次性收完，之后要再赚钱就得再卖一遍东西给你。这个模式能赚大钱，但天花板很硬，撑不到下一个时代。

先搞清「卖断」到底是怎么收钱的

传统软件的核心是永久授权——你付一笔钱，买到「永远可以用这个版本」的权利。注意，你买的不是那张光盘（塑料不值钱），是使用许可。这跟买一台冰箱很像：付一次钱，机器归你，用十年也不用再给厂商钱。

软件公司很清楚一次性收钱有个大问题：卖完这一单，客户就不再给你钱了。所以他们加了第二条腿，叫维护费——每年按授权价的百分之十几到二十再收一笔，换来「有 bug 帮你修、出了新小版本给你打补丁、打电话能找到人」。Oracle、SAP 这类企业软件公司，很大一块利润其实来自这个每年都收的维护费，它比新卖授权还稳。

大白话

为什么维护费这么香？因为一个大客户的核心系统一旦跑起来，是不敢轻易关掉的。停下来重换一套的代价太大，于是它只能每年乖乖续维护费。这笔钱几乎是纯利润——代码早写完了，多收一个客户的维护费，公司不用多花几个钱。

为什么这门生意能赚大钱：边际成本几乎是零

软件有个物理世界里没有的魔法：边际成本——每多卖一份要多花的钱——几乎是零。第一份 Windows 是一笔极其昂贵的一次性投入（几亿到十几亿美元量级的研发），但第二份、第一百万份，只是多刻一张光盘、多印一个盒子。造第一辆车和造第一百万辆车成本差不多，但复制一份软件几乎不要钱。

这就是为什么微软、Oracle 能有让制造业老板眼红的利润率。卖硬件的公司每多卖一台就得多买一堆零件，软件公司多卖一份几乎是白赚。**「一次开发、无限复制」是软件这门生意最底层的引力。**后面所有章节讲的商业模式变化，本质都是在争夺这份「复制几乎免费」带来的红利，只是分法一次次变了。

但天花板也从这里开始长出来

魔法有代价。复制虽然免费，可这个模式在四个地方结结实实地撞上了天花板。

天花板一：盗版几乎防不住

盗版就是不付钱就复制来用。既然复制一份软件几乎不要成本，那小偷复制一份同样不要成本——这个魔法对好人和坏人一视同仁。一张光盘可以拷贝无数次，一个序列号可以传遍全网。厂商想尽办法：序列号、加密狗（一个必须插上的硬件）、激活验证，但道高一尺魔高一丈，尤其在监管弱的市场，盗版率高得吓人。

> 大白话
>
> 这里藏着一个反直觉的点：正因为软件复制免费，它反而**特别难守住**。你卖冰箱，小偷得真把冰箱搬走；你卖光盘，小偷只要按一下复制。免费复制既是软件最大的优势，也是它最大的漏洞。

天花板二：升级要重新卖一遍

这是卖断模式最要命的结构问题。客户买了 Office 2003，就永久拥有 2003。你出了 Office 2007，想让他再掏钱，就得说服他「新版值得再买一次」。可对很多人来说，能打字、能算表格的旧版本已经够用了。

于是公司陷入一个尴尬循环：收入靠「卖新版本」，而卖新版本靠「让旧版本显得不够用」。每一代都得堆足够多的新功能逼客户升级，一旦某一代新功能不够诱人，收入立刻就断档。**你的收入不是稳定的水流，而是一波接一波的脉冲，每一波都要重新求客户一次。**这种「卖了还得再卖」的模式，让收入天然不连续、不可预测。

天花板三：大客户和销售团队互相绑架

企业级的企业软件（卖给公司、动辄几十上百万一单的那种，比如 Siebel 卖给大公司的客户管理系统）不像 Office 摆货架就能卖。它得靠一支昂贵的销售队伍，一单谈几个月，陪吃陪聊做关系，最后签一张大合同。

这就形成了一种互相绑架。公司为了签下大单，养着一支很贵的销售军队，销售的提成和公司的季度收入死死绑在一起；为了让季度数字好看，销售又会在季度末给大客户疯狂打折、塞进一堆客户其实用不上的模块。结果是：**收入被少数大客户和销售节奏牵着走，成本高、周期长、还不稳。**一个大单黄了，整个季度就难看。

> 大白话
>
> 对比一下就懂：卖断 + 大客户直销的模式，像是靠几场大生意撑一年，每一场都得重新谈、重新讨好。它赚的是「大鱼」的钱，可大鱼少、难钓、还容易跑。

天花板四：装到客户机房里，麻烦全归自己

老式软件是本地部署——软件装在客户自己的机房、自己的电脑上运行，不在厂商这边。这意味着每个客户那里都是一套独立的、你看不见也控制不了的环境。

麻烦随之而来：得派工程师上门安装、调试；客户的机器五花八门，同一个软件在一千个客户那里可能出一千种毛病；出了问题你还没法远程看到现场，只能靠电话里客户描述。而且软件卖给谁之后往往还要经过渠道分销——就是不直接卖给用户，而是通过代理商、经销商层层转卖——厂商离真实用户更远了，用户到底怎么用、卡在哪，公司几乎两眼一抹黑。

交付方式的「贵」，贵就贵在这里：贵在防不住盗版，贵在每次升级都要重新卖，贵在养销售军队去签大单，贵在每个客户现场都要单独安装维护、还看不见用户。复制软件本身几乎免费，可把软件「送到客户手里、并持续让它跑好」这件事，成本高得惊人。

为什么它撑不起下一个时代

把上面四条连起来看，你会发现卖断模式的天花板是**结构性**的，不是努力一下就能突破的：

- 收入是脉冲式的——卖一波、等下一波，永远在求客户再买一次，没法预测下个季度稳稳有多少钱进账。
- 增长靠不断找新客户或逼老客户升级，可市场总会饱和，愿意升级的人总会变少。
- 成本压不下去——销售军队、上门部署、层层渠道，每一样都随规模一起变贵。
- 离用户太远——软件装在别人机房里，公司根本不知道用户怎么用、哪好用哪不好用，改进全靠猜。

所以这门生意能造出微软、Oracle 这样的巨头，却有一个共同的天然上限：它赚的是「一次性买卖」的钱，而一次性买卖的总量是有边界的。当增长撞上这堵墙，公司就只能靠越来越大的销售投入、越来越勉强的升级理由去硬撑——这不是能开启下一个时代的姿势。

> 大白话
>
> 这里要记住一件事，它会在下一章变成关键：上面这套「贵销售、大单子、装进客户机房」的打法，当年不是失败者的活法，恰恰是**赢家**的活法。以卖给大公司的客户管理软件为例，Siebel 就是靠这套模式做到了行业头名，一度拿下这块市场的大半。它没做错什么——它只是把一套注定要被换掉的模式，做到了极致。**被换掉的往往不是做得差的人，而是把旧规则玩得最好的人。**这正是本书那条暗线最扎心的地方。

结尾：如果不卖断，还能怎么收钱？

现在把问题倒过来问。既然卖断的所有毛病，几乎都源自「钱一次性收完、然后各奔东西」——那如果我们换个收法呢？

假设我不把软件卖给你，而是**租**给你：你不用买断，每个月付一笔小钱接着用；软件也不装进你的机房，而是跑在我这边，你用浏览器连过来。这样一来会发生什么？盗版没意义了（不给钱就断你连接）；升级不用再卖一遍了（我这边一改，所有人第二天就用上新版）；收入不再是脉冲，而是每个月准时到账的「房租」；我还能天天看着你怎么用，随时改。

大白话

把「卖一台冰箱」变成「每月收房租」——这一个念头，几乎把上一章列的每一堵墙都推倒了。听起来太美好，那它凭什么以前没人做、后来又是什么让它变得可行？这就是下一章的主角：订阅，以及那家把它做成一门大生意的公司。

交付方式第一次变便宜的故事，从这里开始。

第三章 · 订阅：把一锤子买卖变成每月房租

上一章我们把话停在一个尴尬的地方：卖光盘这门生意的天花板，是一次性授权模式自己焊死的。你把软件刻进光盘卖出去，收到一笔钱，然后呢？然后就没有然后了。想再收一笔，你得等到出下一个大版本，还得指望客户愿意再掏钱升级。收入像一次性的烟花，放完就黑。

于是问题变成：如果不卖断，还能怎么收钱？

1999 年，一个叫 Marc Benioff 的人给出了答案。他从 Oracle 出来——在那家卖了十几年数据库授权的公司干过——转身创办了 Salesforce，2004 年把它送上市。他卖的东西听起来平平无奇：一个帮销售团队记客户、记单子的软件。但他收钱的方式，把整个行业的地基换了一遍。

不卖光盘，改收房租

Benioff 的做法，用一句大白话就能讲清：**软件不再卖给你，而是租给你。**

这就是 SaaS（Software as a Service，软件即服务）：软件跑在厂商自己的服务器上，你不装、不买、不拥有，只是按月付钱使用它，就像用自来水、用电一样，打开水龙头才计费。

> 大白话
>
> 老模式像**买房**：一次性掏一大笔钱，房子归你，但装修、修水管、换锅炉全是你自己的事。
> SaaS 像**租房**：每月交房租，住着就行，屋顶漏了房东修。房东图什么？图你每个月都交钱，一交交好多年。

对客户来说，从"一次掏五十万买断"变成"每人每月付几百块"，门槛一下子低了，试错也不心疼。对厂商来说，变化更根本：**收入从一次性变成了每个月都来**。这就是订阅的全部魔力所在，值得我们把它拆开看。

为什么订阅更赚，也更稳

一、可预测的经常性收入

卖光盘的公司，每个季度开局都是零：上季度的钱早收完了，这季度能卖多少全靠重新去拼。做订阅的公司不一样——这个月的客户，只要不退，下个月还在交钱。于是有了一个软件工程师会喜欢的量词：ARR（Annual Recurring Revenue，年度经常性收入），把所有还在续费的客户一年会交的钱加起来，就是你还没开工就已经锁定的收入。

这东西的价值在于**确定性**。你可以像看仪表盘读数一样，年初就大致知道年末会有多少钱进账。资本市场偏爱这种确定性，愿意给它更高的估值——同样赚一块钱，"每年都会再来一块"比"这一块收完就走"值钱得多。

二、续费是复利，流失是漏水

既然客户是按月留下的，那核心就变成两件事：**怎么让他们别走，以及怎么让留下的人越用越多**。

走掉的客户占比，有个专门的词：流失率 churn，指一段时间里有多少客户退订走人。churn 之于订阅公司，就像水桶底下的漏洞之于蓄水——你在顶上拼命往里加水（拉新客户），底下一直漏，桶永远满不了。

> 大白话
>
> 假设你每月流失 5%，听着不多。但一年下来，就算一个新客户都不算，你会漏掉一大半的客户。反过来，如果 churn 压到每月 1%，同样一批客户能陪你走好几年。**churn 每降一点，客户平均能留的时间就成倍变长。**订阅生意的胜负，很大程度上是在这个小数点后面打的。

更妙的是，一个满意的老客户往往会加座位、买更贵的套餐，越用越多——这叫"负流失"，即老客户扩张带来的增收，超过了退订带来的损失。桶不但不漏，还自己往上涨。这是卖光盘的世界里根本不存在的东西：光盘卖出去那一刻，这段关系就基本结束了。

三、先亏后赚的获客经济学

订阅有个反直觉的地方：**刚签下一个客户时，你通常是亏的。**

为什么？因为拉来一个客户要花钱——打广告、养销售、请人吃饭签合同，这些加起来是CAC（Customer Acquisition Cost，获客成本），也就是搞定一个新客户平均花掉多少。而客户第一个月只交一点点房租，远远盖不住这笔前期开销。

能不能赚，取决于他之后交了多久的房租。一个客户从进门到离开，一共会付给你的钱（扣掉服务他的成本），叫 LTV（Customer Lifetime Value，客户终身价值）。

于是整门生意可以压缩成一个比值：

- **LTV 要远大于 CAC**——赚回来的得远多于当初花出去的，生意才成立；
- churn 越低，客户留得越久，**LTV 越大**；
- 获客越高效，**CAC 越小**。

这三个词其实是同一件事的三个面：**你花多少钱请客户进门（CAC），他能留多久别走（churn），一辈子给你交多少（LTV）。**把光盘时代"卖一次赚一次"的粗账，换成了一套可以精算、可以调参的仪表盘。工程师听到这里应该会心一笑——这不就是把生意变成了一个有明确目标函数的优化问题吗？

"No Software"：一句口号背后的算盘

Salesforce 早年最出名的营销，是一个红圈里划掉"SOFTWARE"字样的标志——`No Software`。销售软件的公司却喊"没有软件"，看着像行为艺术，其实精准得很。

它要划掉的不是软件本身，而是软件旧的**交付方式**：那张要你亲手安装、亲自维护、买断了却还得自己伺候的光盘。Benioff 的潜台词是——别再装了，别再养机房了，打开浏览器就能用，坏了我来修。

这句口号真正的对手，是当时企业软件的老大 Siebel。Siebel 也做客户管理软件，做得很好，但走的是老路子：把庞大的系统卖进企业，客户要买服务器、要请顾问装大半年、要养一支 IT 队伍伺候它。Salesforce 的整个叙事，就是站在 Siebel 的对立面：**你那套又贵又重又慢，我这套按月付、开箱即用。**

这里藏着本书那条暗线的又一次拨动：交付方式一变便宜，护城河就换地方了。光盘时代，护城河是"我的功能比你多、我的版本比你新"；订阅时代，护城河变成了**"客户已经把数据、流程、日常习惯都搬进了我的系统，搬走太麻烦"**。前者要靠不停出新版本来守，后者只要

客户不退订，护城河就自己一天天变深。这也解释了为什么订阅公司愿意先亏钱获客——它们赌的不是这一单，是这个客户往后好多年的房租。

可是，房子到底盖在哪？

讲到这儿，订阅模式的账我们算明白了：经常性收入、低流失、LTV 打败 CAC，护城河从"功能"挪到了"绑定"。软件从一锤子买卖，变成了每月房租。

但你有没有注意到一个被我们一直跳过的前提——

租房这个比喻里，房东能收租，是因为他手上真有一栋楼。Salesforce 让你"打开浏览器就能用"，让你不用买服务器、不用养机房，这话说得轻巧。可软件总得**在某台真实的机器上跑**。你甩掉的那些服务器、那些机房、那些 24 小时的运维，并没有凭空消失——它们只是从你的地下室，搬到了别人家里。

那么问题来了：**租来的软件，究竟跑在哪台机器上？那些机器谁来买、谁来管、谁来付电费？**当成千上万家公司都不想自己养机房，都想"打开就能用"的时候，总得有人把服务器盖成一片望不到边的楼群，再一间一间租出去。

这栋楼，就是下一章的主角。

第四章 · 云：把机房变成插座

上一章结尾我们把一个问题悬在了半空：Salesforce 让你"打开浏览器就能用"，可软件总得在某台真实的机器上跑。你甩掉的那些服务器、机房、运维，没有凭空消失，只是搬进了别人家里。那么——是谁家？谁来买这些机器、谁付电费、谁半夜起来换坏掉的硬盘？

这一章讲的就是这栋楼，以及盖楼的人为什么最后决定把它一间间租出去。楼的名字，你大概已经猜到了：云。

先说清楚：云不是天上的东西

云计算说白了，就是**把算力像水电一样，通过网络按用量卖给你**。你要跑一个程序、存一批数据，不必自己去买服务器，而是向一家有大机房的公司租——租多少台、租多久、要多大内存，网页上点几下就有，用完关掉就不再计费。

"云"这个字容易让人误会，好像数据飘在空中。其实一点也不飘。它就是散布在世界各地的一栋栋数据中心（塞满了服务器、空调和电缆的大仓库）里，实实在在的机器。之所以叫云，是因为对你来说，你看不见也不用管那些机器长在哪、怎么维护——就像你用电时不会去想电厂在哪座山后头。

> 大白话
>
> 老办法像**自己家里装一台发电机**：要用电，先花一大笔钱买机器，还要囤柴油、防它坏、留个人看着。云像**把插头接进市电**：电厂在几十公里外，你根本不关心，插上就有电，月底按用了多少度付钱。这一章从头到尾，你都可以拿"自购发电机 vs 接市电"这幅画来对照。

这一档，卖的是"基础设施"这一层

第三章的 SaaS，卖的是**做好的软件成品**——你直接用 Salesforce 管客户，底下怎么跑你不管。云这一档往下沉了一层，卖的不是成品软件，而是**盖软件用的地基和砖头**。

这一层有个正式名字：IaaS（Infrastructure as a Service，基础设施即服务）。意思是把服务器、存储、网络这些最底层的"硬"资源，也拆成按用量出租的服务。你租一台虚拟服务

器、租一块硬盘空间、租一段带宽，像点菜一样，要几份点几份。

2006 年，亚马逊旗下的 AWS（Amazon Web Services，亚马逊的云业务）陆续推出了两样东西，把这层地基第一次摆上货架卖：三月的 S3（租存储：给你一块可以无限往里塞文件的硬盘），八月的 EC2（租算力：给你一台随开随关的服务器）。AWS 由 Andy Jassy 一手带起来，亚马逊的 CTO 是 Werner Vogels——这两个名字后面还会出现，因为关于 AWS 的由来，正是他们出来辟过谣。

一台机器怎么切成很多台

你可能会问：租一台服务器给我，难道亚马逊就得实打实空出一台物理机？那也太浪费了。这里有个关键技术，让整件事划得来——虚拟化：用软件把一台物理服务器切成好多台互不打扰的"虚拟机"，每台看起来都像一台独立的电脑，装自己的系统、跑自己的程序，彼此看不见对方。

> 大白话
>
> 好比一栋大楼隔成很多间公寓：楼是同一栋、水电总管是同一套，但每家关起门来自成一体，A 家做饭不会呛到 B 家。虚拟化就是给服务器"隔公寓"，一台物理机因此能同时招待几十个互不相识的租户，每人只为自己那一间付钱。

正是虚拟化，让"按用量卖算力"从口号变成了能赚钱的生意：机器的利用率被榨高了，一台的成本摊给很多租户，每个租户的单价才能压到低得诱人。

它到底改了什么：把大笔预付款，变成用多少付多少

现在来说这一章最要紧的一件事——云真正翻转的，是**成本的形状**。这件事讲透了，后面很多结果都顺理成章。

在没有云的年代，你想上线一个软件，得先干这些事：估算最多会有多少用户，照着峰值去买一批服务器，租一个机房放它们，雇人 7×24 小时看着。注意，这些钱是**你还没有一个用户之前，就得先掏出去的**。这类"不管你用不用、先花下去的一大笔"叫 固定成本。对应地，在会计上先掏一大笔买下能用很多年的家当，叫 CapEx（Capital Expenditure，资本支出）——买服务器、盖机房就是典型的 CapEx。

云把这套彻底掉了个个儿。你不再预付，而是**用一台服务器算一台的钱，用一小时算一小时**，不用就关掉、一分不收。这种"随业务量涨落、用多少花多少"的钱叫 可变成本；在会计上就是按月付掉的日常开销，叫 OpEx（Operating Expenditure，运营支出）。**云做的事，一句话：把开一家软件公司的 CapEx，换成了 OpEx。**

别小看这一换，它直接决定了谁敢下场。为什么创业门槛会暴跌，就藏在这里：

- **起步不用一大笔本钱。**过去光是买服务器、租机房就得先烧掉几十上百万，很多好点子死在还没上线。现在几百块租几台虚拟机就能把产品跑起来，赔了也不心疼。
- **不用赌未来的用量。**自购服务器你必须押注一个数字：买少了扛不住流量高峰，买多了平时全在空转吃灰。云把这个赌局取消了——用量涨就多开，用量落就关掉。
- **失败变便宜了。**试错的代价从"沉没一屋子买断的机器"，降到"这个月的账单"。想关就关，明天再试别的。

这就是"随用随关"的威力，它有个名字：弹性伸缩，指资源能跟着实际用量自动地涨上去、缩回来。大促来了流量翻十倍，临时多开一批机器顶住；半夜没人用，缩回几台省钱。放在自购发电机的年代，这等于要求你按全年最热那一天的用电量去买发电机，然后一年里 360 天让它半空转——荒唐但你别无选择。接上市电，这问题根本不存在：夏天开空调多用点、冬天少用点，电网那头自己会调。

那亚马逊图什么：一个必须澄清的误会

讲到这，有个流传极广的说法你八成听过：亚马逊做电商，平时有一大堆闲置的服务器，为了不浪费就拿去对外出租，顺手做成了 AWS。这个故事很顺口，很符合直觉——**但它是错的，早被 AWS 的当事人公开辟过谣**。Werner Vogels、以及早期设计者 Benjamin Black 都说过：AWS 不是拿电商吃剩的余量二次利用，它从第一天起就是一门**专门为对外服务而单独采购、单独建设的独立生意**。

真实的因果链更有意思，也更值得工程师琢磨。亚马逊做电商，尤其扛黑五这种大促峰值，被现实逼着修炼出了一身硬功夫：怎么**大规模、标准化、自动化地供应和运维成千上万台服务器**——怎么几分钟开出一批机器、怎么让运维不靠人手一台台伺候、怎么把这套东西做得又稳又便宜。这身内功是被自家业务的压力练出来的。

然后关键的一步来了：亚马逊意识到，**这身运维内功本身，就是一个能卖的产品**。别的公司也天天被同样的苦活折磨，凭什么每家都自己练一遍？于是亚马逊把这套能力打包、标准化，做成 AWS 对外卖。

> 大白话
>
> 所以别记成"有一堆闲机器拿去出租"。要记成：**峰值把亚马逊逼成了运维高手，它再把"当运维高手"这件事做成产品卖给所有人。**前者是废物利用，格局很小；后者是把一种能力产品化，才撑得起后来那么大的生意。

为什么后来者难追：越大越便宜

把内功产品化只是开头。AWS 冲在最前面，还吃到了一个自我强化的红利，工程师对它不会陌生——规模经济：买得越多、摊得越薄，单位成本越低。一次采购几十万台服务器，从芯片到电费的议价能力，是任何小公司比不了的；机房越多、租户越多，同一套运维系统和研发投入摊到每台机器上的成本就越低。

这就形成一个正循环：客户越多 → 规模越大 → 单位成本越低 → 越能降价 → 又吸引更多客户。后来者想追，不光要补上多年的技术积累，还得先跨过这道被规模拉开的成本鸿沟。这也是为什么云这门生意，最后集中在了少数几个玩家手里——它天生就奖励最早、最大的那个。

把机房变成了插座

回到这一章开头那个悬着的问题：租来的软件跑在哪台机器上、谁买谁管谁付电费？答案是——都搬进了云厂商的数据中心，由它们统一买、统一管、统一付电费，再按用量一份份卖回给你。

而它带来的最深远的一下，是把成本的形状从固定改成了可变，从 CapEx 改成了 OpEx。**过去要盖一间自己的机房才能开张，现在墙上有个插座，插上就能用电。**软件的"交付方式"又便宜了一档——这正是贯穿全书那条暗线的第三次拨动。

但请别急着为便宜叫好。当买服务器、盖机房这道最贵的门槛，被降到"注册一个账号、绑张信用卡"，一个连锁反应就要开始了：**门槛塌了，会发生什么？**如果任何一个揣着点子的人，

几百块钱、一个下午就能把产品跑上线，那么会有多少人涌进来？会冒出多少家公司？活下来的又凭什么活下来？

插座已经装好，电随时能取。下一章，我们去看闸门被拉开之后，涌进来的那群人。

第五章 · 创业变便宜之后：一场寒武纪大爆发

第五章 · 创业变便宜之后：一场寒武纪大爆发

上一章结尾，我们把一个问题留在了这里：当云把「先买一屋子服务器」的门槛拆掉之后，会发生什么？答案是——门槛不是被降低了，是被**拆到了地板**。而当做一件事的成本塌到接近零，数量就会井喷。生物界有过一次一模一样的剧情，叫寒武纪大爆发。

先讲那次生物界的爆发

五亿多年前，地球生命憋了很久，物种寥寥。然后在相对很短的一段时间里，海里突然冒出了各式各样的动物门类，眼睛、外壳、附肢、脊索，几乎今天所有动物的「基本身体方案」都在那时候一次性出现。古生物学家管这叫寒武纪大爆发（生命的多样性在很短时间内突然井喷）。

为什么突然爆发？一个被广泛接受的解释是：**身体的积木变便宜了。**当合成钙质外壳、长出感光细胞这些「零件」不再是难于登天的事，进化就可以疯狂地排列组合它们。积木一旦廉价且通用，物种数量就会爆炸。

软件在 2005 年前后，遇到了自己的寒武纪。

两块廉价积木：云 + 开源

第四章讲了第一块积木——云。AWS 把服务器从「买」变成「租」，从固定成本变成可变成本。你不再需要为一个还没人用的想法，先掏几十万买硬件。

但光有云还不够。就算服务器能按小时租，上面跑的操作系统、数据库、Web 服务器这些「软件基础设施」，过去也是要花大钱买许可证的——一套商业数据库授权，动辄五位数美元。这时候第二块积木补上了：开源（源代码公开、任何人可以免费使用和修改的软件，比如 Linux 操作系统、MySQL 数据库、后来层出不穷的各种开发框架）。

把这两块拼起来，就是那几年创业者人人挂在嘴边的 LAMP 栈（Linux 操作系统 + Apache 网页服务器 + MySQL 数据库 + PHP 编程语言，四样全开源全免费）。过去要花

大价钱一件件买齐的东西，现在**全部零成本起步**。再往后，Heroku（一种「你只管写代码、部署上线全帮你搞定」的平台）之类的服务，连「怎么把代码放到云上跑」这最后一点手工活都替你包了。

> 大白话
>
> 翻译成大白话：以前想开一家软件公司，好比想开饭馆得先花大钱买地皮、盖厨房、买全套厨具。现在云是「厨房按小时租」，开源是「锅碗瓢盆全部免费白送」。你剩下要做的，只是想清楚做什么菜、然后动手炒。

因果链：固定成本没了，试错就变得极便宜

顺着往下推，逻辑非常干净：

- **固定成本没了。**起步不用买服务器，不用买软件许可证。
- **于是试错变廉价。**做一个能上线的产品原型，从「一支团队几个月、烧掉几百万」，降到「一个人一个周末，加上几十美元云费用」。
- **于是失败的代价塌了。**过去一次失败可能赔掉全部身家；现在失败就是浪费一个周末和一顿饭钱。

这里最关键的不是「便宜」这个词本身，而是它改变了人对失败的态度。当失败只值一顿饭钱，你就敢多试。你会同时开几个坑，让其中不靠谱的自己死掉。**整个行业从「谋定而后动」变成了「先做出来再说」。**能亲手拼积木的人，从「有资本的少数」变成了「有想法的任何人」。

风投逻辑跟着重排：从重注几家到广撒网

钱的玩法也得跟着变。这一步的因果同样直白。

过去起步贵，风投（拿别人的钱去投资创业公司的机构）只能玩「重注」：仔细挑几家看起来最靠谱的，一次性砸进去一大笔，帮它买下那一屋子服务器。挑错了，代价惨重，所以要慎之又慎。

现在起步便宜了，逻辑翻转成广撒网（投很多很多家，每家先只给一小笔钱，让市场替你筛出谁能活）。既然一家公司起步只需要很少的钱就能跑出个样子，那与其赌上全部身家押三

五家，不如把同样的钱拆成几十份撒出去，让真实用户来告诉你哪个想法立得住。**筛选的活，从投资人的会议室，外包给了市场。**

这个转变催生了一种新物种：批量孵化器（一次招进一大批早期创业团队，每家给一小笔钱、加上三个月的密集辅导，像一届一届学生那样成批地送它们上路）。最有名的是 Y Combinator（简称 YC，2005 年成立于美国，把创业公司成批孵化的代表机构）。它一开始只是个不起眼的暑期小实验——招一小批人、每家给几千美元、换一点点股份，没人料到它会长成后来的样子（真正的「种子投资井喷」也是随后几年才成气候的，YC 和 AWS 更像点火的火星，不是同一秒的爆炸）。它的模式本身就是「创业变便宜」的直接产物：正因为每家启动成本极低，一次押注几十家、让市场淘汰大多数、只要跑出少数几个大赢家就能回本，这笔账才算得过来。

大白话

一句话：起步贵的时候，投资像「押宝」——挑少数几匹马下重注；起步便宜之后，投资像「播种」——撒一大把种子，浇点水，等着看哪几颗自己长成大树。

但廉价有个副作用：稀缺跑到了别处

到这儿，故事听起来全是好消息。但每一次「变便宜」都有同一个副作用，这一章尤其明显：**当人人都能做出来，「能不能做出来」这件事本身就不再值钱了。**

回到寒武纪。积木便宜了，物种是爆炸了没错——可食物、阳光、地盘并没有跟着变多。于是真正的竞争，从「能不能长出一副身体」，变成了「长出来之后，抢不抢得到吃的、活不活得下去」。

软件世界一模一样。供给爆炸，但一天还是只有 24 小时，用户的注意力就那么多，愿意掏钱的意愿更是有限。当成千上万个产品同时冒出来，能不能写出一个能用的 App 不再是门槛——**新的稀缺，变成了「有没有人知道你、有没有人用你、有没有人愿意为你付钱」。**

用一句话把这一档变便宜带来的重排收束一下：

> 做出来，从「最难的事」变成了「最容易的事」；于是最难的事，换成了「被人用起来」。

这就是寒武纪爆发之后每个物种立刻要面对的问题：世界突然挤满了同类，你怎么让别人注意到你、选择你、留下来。这个新稀缺，正是接下来两章的主角——第六章讲新的入口和网络效应怎样重画战场，第七章讲当获客本身成了生死线，人们发明了哪些具体打法去争夺那点有限的注意力。

第六章 · iPhone 与社交：软件长出了新入口

上一章我们停在一个别扭的地方：创业变便宜了，做软件的人多得像下饺子，可东西做出来不再稀缺——稀缺的是「人从哪来」。这一章先回答这个问题的前一半：用户走的是哪条路来到你面前，以及为什么有些产品一旦攒够了人，就再也搬不动了。

要讲清楚这个，得先认一个道理：软件再好，用户够不着，就等于零。

2007：软件长出了口袋大小的新入口

2007 年 1 月，iPhone 发布。2008 年，App Store 上线。对普通人来说，这是「手机能上网了、能装小程序了」；对软件行业来说，这是货架被重新发明了一次。

先把「货架」这个词说透。我们叫它入口（也叫分发渠道）：用户从哪里第一次遇到你的软件、又从哪里点开它。光盘时代的入口是百思买货架和邮购目录，你的软件想被人买到，得先被沃尔玛的采购看上；Web 时代的入口是浏览器地址栏和 Google 搜索框，用户敲个网址或搜一下就到你家门口。

> 大白话
>
> 入口值钱，是因为它卡在「用户」和「你的产品」中间。你写的代码再牛，用户得先能走到你面前，才谈得上用不用、付不付钱。控制入口的人，等于控制了所有软件的必经之路——这跟商场老板收租、跟古代关口收过路费，是同一件事。

iPhone 加 App Store，一下子造出了一个前所未有的入口，它有三个狠地方：

- **它在每个人口袋里。**PC 得坐到桌前才开机，手机一天摸几百次。软件第一次做到「随时随地」够得着人。
- **它直达十亿人，而且没有采购员。**过去上货架要求爷爷告奶奶，App Store 上你交个开发者费、审核过了就上架，理论上全球每一部 iPhone 都能搜到你。分发的门槛塌了。
- **它自带钱包。**手机里绑着信用卡，点一下就付款。收钱这一步——过去最难的一步——被平台顺手办了。

结果是一批公司凭「先占住手机这个新入口」起来了。用车、订餐、聊天、拍照——这些事本来就存在，只是过去没有一条又快又直的路把服务塞进你手心。移动这个新货架，让「亿级用户、随时随地」从愿望变成默认值。

但入口只解决了「怎么把用户送到你面前」。它没解决另一半：怎么让用户来了就不走、别人抄了也抢不走。这一半，2012 年上市的一家公司演示得最干净。

2012：Facebook 与最纯的网络效应

2012 年，Facebook 上市。它当然也是一个新入口——几亿人每天泡在信息流里，你想让人看见你，去那儿投广告最快。但 Facebook 真正示范的，是另一样东西，一种比「切换成本」更硬的护城河。

这东西叫网络效应。

> 大白话
>
> 网络效应，一句话：每多一个人加入，这个东西对已经在里面的所有人就更有用。
>
> 最老实的例子是电话。世界上只有一部电话，它一文不值，你打给谁？有两部，能通一次话。等全城都装了电话，你那部电话的价值大到离不开——可你自己那部电话从头到尾没变，变的是「别人也有」。传真机、微信，同理：价值不在机器里，在网里的人头上。

Facebook 就是这么长起来的。你注册它，不是因为它界面漂亮、代码优雅，而是因为**你想找的人都在上面**。你同学在、家人在、老同事在。你一个人在 Facebook 上什么也干不了，正是所有其他人在那儿，才让它对你有用。

现在把这个道理往前推一步，就能看懂那句反常识的话：**有网络效应的产品，越大越难被超越。**

为什么？假设你是个天才工程师，做了一个各方面都比 Facebook 强的社交网站——更快、更清爽、更尊重隐私。你能赢吗？大概率不能。因为用户不是来用「功能」的，是来找「人」的。而人全在老地方。你得说服的不是一个用户，是他要联系的一整圈人同时搬家。

这里藏着一个专门的坑，值得起个名：先有鸡还是先有蛋——新社交产品刚上线时空空荡荡，第一批用户进来发现没人，扭头就走；因为他们走了，第二批进来还是没人。老产品早

就跨过了这道坎，人吸引人，滚成了雪球；新产品连雪球的第一把雪都攒不起来。

> 大白话
>
> 所以网络效应护城河的厉害之处，不在于让对手做不出更好的产品——对手完全做得出。它厉害在：就算对手的产品更好，也搬不动大家。锁住用户的不是你的代码，是用户彼此。

但别把「难」听成「不可能」

话说到这儿要赶紧刹一脚，不然就把网络效应吹成了永动机。它是**更难被超越，不是不可能被超越**。最有说服力的反例，恰恰就是 Facebook 自己：它 2004 年起家的时候，社交网络的王座上坐着的是 Friendster 和 MySpace，那两家一样有几千万用户、一样有网络效应。Facebook 照样把它们掀翻了。所以「全村一起搬家」虽然难，但真会发生。

什么时候会发生？大致三种情况：**一是老产品自己变烂**——广告塞太多、体验垮掉，用户忍到某个点，一群人相约出走，雪球开始反向滚；**二是换了代人**，年轻人不想跟爸妈待在同一个网里，干脆去开辟新地盘（后来 Instagram、Snapchat、TikTok 就是这么从 Facebook 身上咬下肉来的）；**三是换了入口**——就像这一章讲的，手机这个新货架一出现，「在电脑上织好的网」并不自动等于「在手机上也赢」，规则一变，牌重发。

还得再分一层：网络效应有**全局**和**局部**之别。Facebook 是全局的——理论上全世界的人都该在同一个网里，所以赢家通吃、极难撼动。但打车、外卖、同城租房这类是**局部**网络：你在上海用得爽，靠的是上海的司机和乘客多，跟成都有多少人毫无关系。局部网络只需要在一座城市里从头攒起一张小网，就能撬开一个缺口——所以这类生意往往一城一城地打，很难一家独霸。**先发优势有多铁，得看这张网是全局的还是局部的。**

这一层不是给网络效应泼冷水，反而正是本书暗线的注脚：再硬的护城河也不是永久产权。每一次底层成本或入口发生变化，攒了很久的网都可能被重新洗牌——赢家会换，只是换起来比别的护城河更费劲。

这跟本书第 3 章讲的护城河不是一回事，得分清楚。第 3 章的护城河是**切换成本**：你数据在里面、流程绑死了、换一个太麻烦，所以你懒得走——那是「我自己走不动」。网络效应是**「我一个人走了也没用，得全村一起走」**。前者困住的是你，后者困住的是整张关系网。后者显然更硬。

（顺便划清另一条界：网络效应不等于「数据越多产品越好」。那种叫数据护城河，是本书第 9 章的活儿。这里的因果是「用户互相吸引」，不是「数据把产品磨得更准」，别混。）

把因果串起来：谁送货、谁锁人，被重排了

现在退一步看这一章到底发生了什么。移动和社交常被说成「两个新的产品品类」——多了 App，多了社交网站。但那是表象。真正被改写的，是两个更底层的问题：

1. **谁能把产品送到用户面前？** 答案从「浏览器和搜索框」变成了「手机主屏和 App Store」。掌握新入口的平台，成了所有软件的房东。
2. **谁能把用户锁住？** 答案从「让你换起来嫌麻烦」升级成了「让你身边所有人都在这儿，你根本换不动」。护城河从切换成本，进化到了网络效应。

这正好接上本书那条暗线：软件的交付方式每变便宜一档，赢家就重排一次。这一档变便宜的是**分发**——App Store 让「上货架、被十亿人搜到、顺手收钱」几乎免费。可越是分发免费、人人都能上架，「被看见」和「留得住」就越稀缺、越值钱。于是竞争的重心，从「能不能做出来」（第 5 章已经不稀缺了），彻底移到了「能不能占住入口、织成网络」。

可是，普通软件公司怎么办？

到这儿有个尴尬的事实得摊开讲。入口和网络效应是好东西，但它们不是人人有份。

Facebook 那种网络效应，是社交产品的天赋——用户之间天然要互相连接。可绝大多数软件根本没有这种天赋：一个报销工具、一个建站服务、一个数据库，你多一个客户，并不会让我这个客户用得更爽，我俩八竿子打不着。这类普通公司站在同一个拥挤的新货架上，没有天生的雪球，也没有把用户焊在一起的关系网。

那它们靠什么在十亿人的货架上被发现？靠什么让来的人留下来、掏钱、别走？

答案是：既然天上不掉网络效应，那就把「获客」这件事，从碰运气变成一门可以设计、可以拆解、可以拿数据一遍遍调的**工程**——让软件自己带来下一个用户，让免费的人心甘情愿变成付费的人。这门工程后来有了名字：freemium、增长黑客、PLG。

那，就是第 7 章的事了。

第七章 · 增长黑客与免费增值：把获客做成工程

第七章 · 增长黑客与免费增值：把获客做成工程

上一章我们看到，iPhone 和 App Store 给了软件公司新入口，社交网络给了新势能。但入口是所有人的入口，货架是所有人的货架。当供给爆炸、货架拥挤（第五章的伏笔），一家没背景没流量的普通软件公司，怎么让人知道你、用上你、还愿意付钱？

过去的答案是：花钱。买广告，雇销售，投电视。谁钱多谁赢。这一章讲的是一群工程师给出的另一个答案——**获客不该是花钱的事，应该是改产品的事**。

市场部拍脑袋，工程师看漏斗

先说一个词。增长黑客（growth hacking），说白了就是**用工程和数据的方法做增长**，而不是靠市场部拍脑袋买广告。

它跟传统市场营销的分水岭，在于对「用户从哪来」这件事的态度。传统市场部把它当成一门艺术：想个好口号，投个大广告，然后看销量涨没涨——涨了也说不清是哪句话起了作用。增长黑客把它当成一套转化漏斗：用户从看见你，到点进来，到注册，到用起来，到付钱，每一步都会漏掉一批人，像水流过一层层漏斗，越往下越少。

> 大白话
>
> 漏斗的意思是：100 个人看到你的落地页，30 个点进来，10 个注册，3 个天天用，1 个付钱。每一层的比例都是一个可以量出来的数字。你不再问「用户喜不喜欢我」这种没法回答的问题，而是问「注册页为什么漏掉了 70% 的人」这种能动手的问题。

一旦每一步都是数字，增长就变成了工程师熟悉的活儿：**找到漏得最狠的那一层，改产品，A/B 对比，看数字有没有变好，留下好的那个版本。**这就是「黑客」二字的由来——不是攻击谁，而是像调代码一样，一个变量一个变量地拧，把漏斗的每个环节调到最优。注册流程

少填一个字段、按钮从蓝色改成绿色、把「免费试用」四个字挪到首屏，都可能让转化率动几个百分点。市场部靠灵感，增长黑客靠迭代。

让用户替你拉用户：K 因子

漏斗只是把「买来的流量」用得更省。真正的魔法，是让产品自己生产流量。

最早也最经典的一击来自 Hotmail。1996 年，这个免费邮箱在每一封用户发出的邮件末尾，自动加上一行小字：「`PS: I love you. Get your free email at Hotmail`」。每个用户写信，都在替 Hotmail 打广告；收信的人里总有几个点进去也注册了，然后他们发的信又带上了这行字。几乎没花钱，用户数在一年多里冲到千万级。这不是买来的增长，是**用户自己长出来的增长**。

工程师看到这里，本能地想量化它。于是有了病毒系数 K 因子：**一个用户平均能带来几个新用户**。K 大于 1，意味着 1 个带来 1 个多，新用户又去带更多，像链式反应一样自我繁殖，不花一分广告费也能滚雪球；K 小于 1，火种再旺也会慢慢熄灭，你得不停往里砸钱续命。

> 大白话
>
> 把 K 因子想成传染病的传播数。K=2，一个传两个，指数爆炸。K=0.5，一个传半个，几轮就传没了。增长黑客毕生在干的一件事，就是想尽办法把这个数字从 0.9 拱到 1.1——因为跨过 1 这条线，增长的性质就从「烧钱」变成了「白嫖」。

Dropbox 把这件事做成了产品里的一个按钮。它不投广告，而是设计了推荐机制：**你邀请一个朋友注册，你和朋友都白得几百 MB 免费空间。**对一家云存储公司来说，送出去的空间几乎不花钱（后面会讲为什么），但换来的是一个自带动机去拉人的老用户，和一个已经上钩的新用户。奖励直接焊在产品里，不经过市场部的手。据称这套机制让 Dropbox 的注册量在一段时间里翻了好几倍。

还有一类更安静的病毒式设计：LinkedIn 和 Facebook 的「你可能认识的人」。它读取你的通讯录，猜出你还没连上的熟人，推到你面前一个「加为好友」的按钮。你每加一个人，就把那个人也更深地黏进了这张网——网里的人越多，每个人离开的成本越高。这就是第六章说的网络效应，被增长团队用工程手段一点点喂大。

先白送，再从重度用户身上赚：免费增值

漏斗要转化的人，得先进到漏斗里。怎么让海量陌生人零门槛地用起来？答案是免费增值（freemium）：**基础功能永久免费，让所有人先用起来；只把一小撮用得最重、最离不开的用户，转成付费。**free（免费）+ premium（增值）拼在一起，就是这个词。

Dropbox 给你 2GB 白用，不够了才掏钱升级；Evernote、Spotify、后来无数 SaaS 都是这个套路。免费的那群人不是慈善对象，他们是**漏斗最顶端最宽的那一层**——是活广告、是 K 因子的燃料、是未来付费用户的蓄水池。

但这里有个关键问题：**凭什么养得起一大群不给钱的人？**换到光盘时代，这套打法根本不成立——每多一个用户，你得多压一张盘、多印一本手册、多寄一趟快递，白送就是纯亏。

成立的前提，是这本书的暗线又变便宜了一档：**云让边际成本趋近于零**。当软件跑在云端、按需扩容，多服务一个免费用户，无非是多占一点几乎免费的存储和计算。多一个白嫖用户的成本，从「一张光盘」降到了「几分钱电费」。**正是因为多养一个免费用户几乎不花钱，你才养得起一大群不付钱的人，把他们当漏斗顶端。**freemium 不是谁发明的营销妙招，是成本结构变了之后自然长出来的商业模式。

产品即销售：PLG

把漏斗、病毒系数、免费增值拧成一股绳，得到的就是这十几年硅谷最主流的打法：PLG，产品驱动增长（Product-Led Growth）。一句话，**让产品自己带来增长**——用户用着用着，自然就邀请了同事，自然就撞到了付费墙想升级，全程不需要销售在旁边盯着。

对比一下第二章那支昂贵的销售军队。老派企业软件的卖法是：销售约到客户的 CIO，飞过去讲三个月 PPT，签一份百万美元的年度合同，装一整套系统。获客成本高得吓人，所以只能卖给付得起的大客户。

> 大白话
>
> PLG 反过来：产品先免费给一线员工用，一个程序员觉得好用，拉进来三个同事，一个部门用顺了手，某天撞上「团队版才能共享」的墙，就自己刷卡升级了。销售（如果还需要的话）是最后来收单的，不是最前面来敲门的。用第二章的话说——**过去是销售军队卖产品，现在是产品自己当销售。**

为什么 PLG 能取代销售军队？因为前面几档「变便宜」叠在了一起：云让白送不花钱，社交和邀请机制让传播不花钱，数据和漏斗让优化有据可依。当获客的边际成本被压到接近零，「先让所有人免费用，再从中捞付费的」在数学上就跑得通了。软件不再需要一支军队去推，它自己会走。

留存：漏斗底下有没有堵住

还差最后一个词。拉来再多人，留不住也是白搭，所以增长团队盯的第二个核心指标是留存率：**一批新用户过了一周、一个月、三个月，还有多少人回来用。**

留存是所有指标里最诚实的一个。K 因子高但留存低，等于用漏水的桶接水——顶上疯狂拉人，底下的人不停流失，桶永远装不满，还得一直烧钱补水。留存高，才说明产品真解决了问题，滚雪球才滚得起来。所以成熟的增长黑客有句共识：**先把留存做扎实，再谈拉新**；否则你只是在给一个漏桶加大水龙头。

看起来完美，但账本上有个洞

把这一章串起来：获客从「打广告砸钱」变成了「改产品」，从市场部的艺术变成了工程师的漏斗。免费增值当燃料，病毒系数当引擎，PLG 当整套发动机，留存当油箱。一台漂亮的增长机器。

但请盯住这台机器赖以运转的两个前提，因为它们后面会出事。

第一个洞是结构性的：**免费的人太多，付费的人太少。**freemium 的转化率天生很低——绝大多数白嫖用户永远不会付钱。养他们的成本虽然低，但**低不等于零**：几百万免费用户的存储、带宽、客服，加起来是一笔真金白银的持续支出。只要付费的那一小撮撑不起免费的一大群，账本上就有个洞。

第二个洞更致命：**整套打法极度依赖「获客几乎免费、钱很便宜」的环境。**K 因子、免费增值、PLG，全建立在一个假设上——现在把规模做大不要紧，钱多的是，用户以后总能想办法变现。在低利率、热钱涌动的年代，这个假设成立，投资人乐意为「增长」买单，哪怕公司年年亏损。

> 「先把规模做大，以后再想怎么赚钱」——这句话是这个时代所有增长故事的潜台词。它成立的唯一条件，是钱一直很便宜。

可风向是会变的。当利率抬头、热钱退潮（第十章、第十二章的故事），投资人不再为纯增长鼓掌，转而追问一个朴素的问题：这些免费用户，到底什么时候、能不能变成钱？那一刻，养着百万白嫖用户、转化率却上不去的公司，会被一笔一笔清算。这台在便宜资本里运转如飞的增长机器，会第一次感受到重力。

下一章我们先不谈清算，而是看增长黑客的一个更聪明的变种：当你的用户本身就是开发者，产品该怎么设计才能让他们自下而上地把你带进公司。

第八章 · 开发者成了新客户：卖铲子的人

第八章 · 开发者成了新客户：卖铲子的人

上一章讲了怎么把东西卖给普通人——免费注册、用顺手了再掏钱，一整套面向大众的获客工程。这一章要换一群客户，一群很特殊、以前从没人正经把他们当客户的人：**工程师自己**。你在读这本书，多半就是他们中的一个。所以这一章讲的，其实是你是怎么被卖东西的，以及为什么这种卖法，比过去那套厉害得多。

先回忆一下过去软件是卖给谁的

第二章讲过老规矩：一套企业软件，卖给「老板和采购部门」。销售穿西装、飞过去、请吃饭、做几十页 PPT，跟客户公司的高层谈半年，签一份几十万美元的合同。真正每天要用这软件的工程师，全程没有发言权——软件是别人替他挑好、买好、装好，然后塞给他用的。好不好用不重要，采购签字的那位觉得靠谱就行。

这套打法叫 自上而下（top-down，从公司高层往下推：先说服有预算、能签字的老板，再一层层压到基层员工手里）。它贵、慢，而且经常买回来一堆没人爱用的东西。

这一波，一批公司把顺序彻底反了过来。

自下而上：绕开老板，直接俘获工程师

新打法叫 自下而上（bottom-up，不去敲老板的门，而是让最底层的工程师先自己用起来，用上瘾了，再由他们反过来推动公司付钱）。它凭什么打得过西装革履的传统销售？就凭一条：**信任的来源不一样了。**

老板的信任，来自 PPT、来自商务饭局、来自「同行都在用」这种话术——都是二手的。工程师的信任，来自**亲手用过**。他今天下午免费注册一个账号，十分钟就在自己的真实项目里跑通了，明天就知道这东西到底行不行。这种信任骗不了也吹不了，因为代码不会给面子：能跑就是能跑，不能跑吹上天也没用。

大白话

老办法像相亲：媒人（销售）把对象夸得天花乱坠，你信不信全看媒人嘴皮子。新办法像同居一周再谈：你先住进去，锅碗瓢盆全试过一遍，合不合适自己心里有数——这时候谁的话都不好使，只有体验作数。

顺着这条线，因果就很清楚了。工程师能自己免费注册、当天用起来、在真实项目里验证，于是**决策链被压到极短**：不用开会、不用打报告、不用等预算审批，一个人一个下午就能决定「这玩意儿好用，以后就用它」。等到项目做大、团队都在用、免费额度不够了，付费这一步几乎是水到渠成——不是老板被说服了要买，而是**东西早已经长进了公司的日常，拔不掉了，只好补一张合同把它转正。**

销售的动作，从「跪着求大客户签字」，变成了「让工程师先上瘾，再让上瘾的工程师去替你说服他老板」。替你卖货的人，从西装销售，换成了你的用户本人。

那么，把源代码白送出去，靠什么赚钱?

自下而上要成立，前提是让工程师零成本、零门槛地先用上。最极致的做法，就是把软件的源代码整个公开、免费给全世界用——这就是开源（源代码公开、任何人都能免费拿去用和改的软件）。可问题立刻来了：东西都免费送人了，公司喝西北风吗?

这里有个反直觉但很关键的账：**先让全世界免费用上、攒起海量采用，再从其中「用得最重、最离不开」的那批人身上收费。**免费的是「用」，收费的是「用得省心、用得放心、用得合规」。具体有这么几种赚钱的机制：

- **托管 / 云服务。**源代码免费，你当然可以自己下载、自己搭、自己运维——但你多半不想。自己搭一套数据库还得盯着它别宕机、别丢数据、半夜爬起来扩容，太累。于是公司说：这些脏活累活我帮你干，你按月付钱，打开就能用。MongoDB（一款开源数据库）就是这么干的：数据库本身开源免费，但它的托管服务 MongoDB Atlas 帮你把运维全包了，这才是收入大头。
- **开放核心。**叫 open-core（核心功能开源免费，企业才需要的高级功能另外收费）。个人和小团队用免费版足够；但大公司需要的那些东西——精细的权限管理、审计日志、单点登录、企业级支持——放进付费版。谁最需要这些？恰恰是最有钱付的大客户。

- **支持与合规。**免费自己用没问题，可一家银行、一家医院要把开源软件用在核心系统上，它需要有人签合同背书、出问题有人半夜接电话、能证明这套东西合规。这份「有人兜底」的安心，本身就值钱。

这条路成立的底层逻辑是：交付变便宜（云 + 开源）之后，「让一个人开始用你」的成本几乎是零，于是你可以先不计代价地铺开、把采用量做到千万级，再从这千万人里那一小撮「规模化使用、离不开你」的重度用户身上，把钱赚回来。Elastic（做搜索引擎的开源公司）、HashiCorp（做云基础设施工具的开源公司）走的都是这条路。

大白话

这里有个后来才爆出的张力值得先埋个扣子：正因为「靠托管赚钱」，这些公司最怕的不是小偷，而是**财大气粗的云厂商**——亚马逊这类平台把它们的开源版本直接拿去做成自己的托管服务，白赚了本该属于它们的托管钱。于是 MongoDB（2018）、Elastic（2021）、HashiCorp（2023）后来都改了授权协议，从严格意义的「开源」收紧成「源码可见 source-available」（代码你还能看、能自己用，但不许拿去开一个跟我打对台的托管服务）。这恰恰印证了本章那句话：一旦收入靠托管，「谁能托管」就成了必须焊死的护城河。

卖铲子：不赌哪个 App 会火

现在讲这一章标题里的那个词。淘金热里，真正稳赚不赔的，往往不是挖金子的人——挖金子看运气，十个里九个空手而归。稳赚的是**在矿场边上卖铲子、卖牛仔裤、卖水的人**：不管哪个矿工挖到金子、哪个颗粒无收，他们都得来买铲子。这就是卖铲子的智慧——不下场赌哪个人能中彩票，而是给所有下场赌的人供货。

软件世界里，「铲子」就是 基础设施 / 开发者工具（数据库、通信、支付、部署、监控这些每个应用都要用的底层零件）。做这层的公司精明就精明在：它**不赌哪个 App 会火**。上面跑的应用死掉一大半没关系，因为不管哪个活下来、哪个火了，都得踩着它这层底座。它卖的是所有淘金者共用的那把铲子。

把一件复杂的事，变成一个 API 调用

这些铲子好卖、粘性还极强，核心秘密在一个词：API（一段让程序之间互相调用的接口——你不用自己造轮子，调一下别人现成的服务就行）。最漂亮的一类公司，做的事可以一

句话概括：**把一件原本极其复杂的事，收成一个简单的 API 调用。**

举两个工程师都熟的例子：

- Twilio（做通信 API 的公司）。自己实现「给用户发条短信」或「打个电话」有多恶心，做过的人都知道：要跟一堆运营商谈接入、处理各国的号码和法规、维护一套通信底层。Twilio 把这一整坨复杂，收成了**几行代码**：你调一下它的 API，短信就发出去了、电话就打通了。你根本不用懂运营商是怎么回事。
- Stripe（做支付 API 的公司）。自己实现「在网站上收一笔钱」同样是个噩梦：对接银行、过风控、处理各种卡、扛合规。Stripe 也把它收成了几行代码——贴上去，你的网站就能收钱了。

请注意，Twilio 是通信、Stripe 是支付、MongoDB 是数据库、GitHub 是代码托管协作（把代码放上去、多人一起改、记录每一次改动的平台），各管各的一摊，别记混了。它们的共同点只有一个：**把一件你不想自己碰的复杂事，变成了一个调用。**

粘性从哪来：写进代码里就拔不掉了

为什么这类公司护城河这么深？因为一旦你把它的 API 嵌进了自己的代码，**换它的成本高得吓人**。你的支付逻辑里到处是 Stripe 的调用，你的短信、通知全走 Twilio，想换成别家，等于把自己代码里成百上千处地方一个个抠出来重写、重测、重新上线——没人愿意干这种既有风险又不产生新价值的活。

> 大白话
>
> 这跟第七章那种「用户嫌麻烦懒得换」的粘性不是一回事，它硬得多。前者是习惯，后者是**物理焊死**：它的接口已经缝进了你几万行代码的血肉里，拔出来会流血。所以护城河从「客户关系」，变成了「已经写进千万人的代码里，拔不掉」。

把这一档变便宜，收束一下

顺着全书那条暗线看：交付变便宜（云），带来的不只是「东西便宜了」，连**「怎么卖」本身都被重排了**。既然让一个工程师开始用你几乎不要钱，那最优策略就不再是雇一队西装去跪求大客户，而是**让开发者自己先用上瘾**，再让上瘾的他们把你带进公司。

> 过去：说服一个签字的人，卖一份合同。现在：让一百万个工程师亲手用上，再从其中最离不开你的那批人身上收钱。

赢家也跟着换了一批人：不是谁的销售最能喝，而是谁的东西**让工程师第一次用就想留下、留下了就拔不掉**。GitHub、MongoDB、Twilio、Stripe、Elastic、HashiCorp——这一批，都是靠着把客户从「老板」换成「开发者」跑出来的。

结尾：这些铲子每天都在挖出同一样东西

但故事还没完。你有没有注意到一件事：这些跑在云上的海量应用、这些被调来调去的 API、这些被无数人使用的开发者工具，它们运转的每一秒钟，都在源源不断地产出**同一样东西——数据**。

谁在这条路上通过谁、谁的短信发给了谁、谁的代码改了哪一行、谁买了什么——这些数据起先只是运营的副产品，没人太当回事。可是当它攒到足够大的量，它会变成一种全新的资产，甚至一道比「写进代码里」更深的护城河。铲子卖着卖着，卖铲子的人低头发现：手里攒下的这堆数据，可能比铲子本身还值钱。

数据凭什么成为下一道护城河，它又会怎样再一次重排赢家？这是第九章的事。

第九章 · 数据变成金矿，也变成护城河

上一章结尾，卖铲子的人低头发现：铲子挖了半天，脚底下堆起的这堆数据，可能比铲子还值钱。这一章我们就蹲下来，把这堆东西看清楚——它凭什么值钱，凭什么能变成一道护城河，以及它没被吹的那面：它也会失效。

先说一句让全章成立的话：**云和移动，把「用户的一举一动」全搬到了线上。**过去你在实体店买东西，店员记不住你摸过哪件没买；现在你在 App 里划一下、停三秒、加进购物车又删掉——每一个动作都是一条记录，自动落进服务器。数据不再靠人工填表攒出来，它是软件运转时顺手漏下来的。

数据本来是「废气」

早年间，做业务的人是这么看数据的：它是业务的副产品——你为了完成一笔交易，顺带产生的一堆记录（谁在几点买了什么、点了哪个按钮）。就像工厂开机器，主产品是零件，副产品是废气，没人冲着废气去开厂。这些记录躺在日志里，占着硬盘，定期还得删掉腾地方。

> 大白话
>
> 「数据是废气」不是个比喻上的贬低，是当时的经济账：存它要花钱，分析它要花钱，而它能换回的钱看不见。在存储和算力都贵的年代，把废气收集起来提纯，不划算。所以大家真的就把它放掉了。

让废气变成资产的，是第四章那件事：**存储和算力，从「一次性买断的大件」变成了「按用量买的水电」。**攒一年的点击记录，过去意味着先砸钱买一屋子服务器；上云之后，变成每月账单上多出的一小笔，用多少付多少。于是「先把数据都留着，回头再想怎么用」这个念头，第一次在经济上说得通了。地基一便宜，上面立刻长出新东西。

数据怎么就变成了钱

数据变钱，靠的不是「卖数据」——真正的高手很少直接卖。它变钱的方式是**让产品变得更准**，而更准的产品能赚更多钱、留住更多人。这里有个自我强化的循环，是这一章的核心，叫数据飞轮（也叫数据护城河）：

1. 用的人越多 → 你积累的数据越多；
2. 数据越多 → 你的产品/推荐/判断越准；
3. 越准 → 越多人愿意来用；
4. 回到第一步，数据又更多。

轮子一旦转起来，它会自己越转越快，而且后来者很难追——因为对手缺的不是代码，是**你已经攒下、他还没有的那堆样本**。举三个转得最漂亮的例子：

- **搜索。**越多人在搜索框里搜，越多人点了第几条、又退回来重搜，引擎就越知道「对这个词，什么才是好结果」。你的每一次点击都在替它打分。新来的搜索引擎代码可以照抄，但它没有「几十亿次真人点击」这本账，所以结果就是没那么准。
- **推荐。**越多人看了什么、跳过了什么，系统越知道「看了 A 的人多半也想看 B」。你在电商、短视频、音乐里感到「它怎么这么懂我」，懂你的不是某段聪明算法，是它见过**几亿人**的观看轨迹。
- **反欺诈。**见过的欺诈样本越多，越会认下一次欺诈长什么样。一家处理支付的公司，看过千万笔真实盗刷，它的风控就是比只看过几百笔的对手灵。这里数据的价值特别直白：多抓一笔骗子，就是省下一笔真钱。

它和另外两道护城河，别混

这本书到现在，讲过三种「别人抢不走你」的机制。它们能同时出现在一个产品上，但发动机完全不同，混着说就全糊了：

- **网络效应（第六章）：价值在人头。**用户之间互相吸引——你朋友都在这个社交网络上，所以你也来。去掉别的用户，产品对你就没意义了。
- **切换成本（第三章）：价值在「搬家太麻烦」。**不是产品多好，是换掉它要重新导数据、重新培训、重新接口，你嫌烦就不换了。

- **数据护城河（本章）：价值在积累的样本。**数据把产品磨得更准，而对手缺的是那堆样本。哪怕全世界只有你一个用户，只要你喂了它足够多数据，它也会更准——这一点，恰恰是它和网络效应最干净的分界：网络效应要的是「别人也在」，数据护城河只要「样本够多」。

> 大白话
>
> 一句话记法：网络效应是「人拉人」，切换成本是「拔出来会疼」，数据护城河是「练得比你久」。一个产品可以三样都占——比如一个社交 App 既有人拉人，又攒了你多年数据难搬走，又靠数据把推荐磨准。但你要拆一家公司凭什么活着，就得看清它到底靠哪台发动机。

别把它吹成万能

数据护城河听着无敌，其实有两个软肋，这也是费曼式的诚实：

一，边际收益会递减。数据不是越多越好，多到一定程度，再翻十倍也帮不上什么。教一个模型认猫，头一万张照片让它突飞猛进；到第一千万张，再加一百万张几乎看不出区别——好处早被前面吃光了。所以「数据多」本身不等于护城河深，得看多出来的数据还能不能换来看得见的更准。

二，数据会过时。世界一变，旧数据就成了误导。疫情来了，之前所有的出行、消费轨迹一夜之间失真；一个热点过去，围着它攒的偏好数据就是包袱。数据护城河不是护城河一旦挖好就永远在那，它更像一个**需要新鲜水源不断补给的池塘**，断了活水就会发臭。

撑起这一切的三样基础设施

数据大到一定量，老办法就装不下、算不动了，于是催生了一套新工具。三个名词，一次说白：

大数据：数据量大到一台机器根本装不下、算不完，只能**叫来一大群普通机器分工干**——把一份巨大的活切成小块，一人算一块，最后汇总。关键不在某台机器多强，而在「让几百台便宜机器像一台超级机器那样协作」。

数据仓库：把散落在各处的业务数据（订单系统一份、App 日志一份、客服记录一份）**集中搬进一个大仓库，专门用来做分析**。它和你天天用的数据库不一样——数据库是「账房」，负责快准狠地记一笔笔当下的账（这单成交了、这个用户改了密码）；数据仓库是「档案馆

+参谋部」，负责回答「为什么上个月华东掉单」「照这势头下季度会怎样」。一个管当下记账，一个管回头看和往前看。

数据湖：比仓库更随意的一种存法——先不管数据整不整齐，原样倒进一个大池子存着，等要用了再收拾。仓库要求你进门前把货码好，湖是先堆着、用时再挑。

> 大白话
>
> 为什么这些偏偏在云出现之后才真正跑得起来？因为它们全都吃「按需的海量存储 + 临时叫一大群机器」。搁在自建机房年代，你得为「万一要分析」提前买齐一屋子服务器，平时闲着也烧钱，多数公司算不过账。上云之后（呼应第四章）：存海量数据按用量付，要分析时临时租五百台、算完就还，账一下子就平了。云不是让分析变强了，是让「攒海量数据、按需分析」这件事第一次变便宜、变得普通公司也玩得起。

把这一档收束一下

顺着全书那条暗线看：这一次变便宜的，是**「存储和分析海量数据」的成本**。成本一塌，废气变资产，资产又能磨出更准的产品，护城河于是在「人际网络」旁边，又长出一条新的——「数据积累」。谁先把数据攒起来、并且真把它转成了产品优势，谁就吃到这一波；只会囤数据不会用的，攒的还是废气。

为什么这为后面 Snowflake 那批公司铺了路，这里先埋一句：既然每家公司都突然明白「数据是金矿」，那「帮别人把金矿存好、挖好」本身，就成了一门能上市的大生意。这是后话。

结尾：地基下面还垫着一样东西

讲到这儿，软件公司看着简直无懈可击：它有订阅（第七章那种细水长流的现金流），有云（弹性又便宜的交付），现在又叠上了数据护城河。三样叠在一起，估值一飞冲天，投资人愿意为「未来还很远的收入」预支很高的价钱。

但你有没有想过：投资人凭什么愿意为「未来的收入」现在就掏这么多钱？这份繁荣的地基，其实还悄悄垫着一样和软件本身毫无关系的东西——**钱的价格，也就是利率。**钱便宜的

时候，「以后才赚到的钱」在今天就显得很值；钱一贵，同样一份未来收入，今天就不值那么多了。

这只看不见的手，一直在幕后决定着所有软件公司值多少钱。它松一松，泡沫就吹起来；它一收紧，塌方就开始。它，是第十章的主角。

第十章 · 零利率：一场用便宜的钱吹起来的盛宴

第十章 · 零利率：一场用便宜的钱吹起来的盛宴

前面九章，我们一直盯着一件事：软件的**交付方式**怎么一次次变便宜——从压光盘到装机，从装机到订阅，从订阅到云，从云到「多养一个免费用户几乎不花钱」。每便宜一档，赢家就重排一次。这是这本书的主暗线，我们叫它「成本的塌方」。

但决定软件世界冷暖的，从来不止这一只手。这一章，我们把另一只手正式请上台。它不改交付方式，不写一行代码，却决定了整个游戏在什么水温里进行。它的名字，你在新闻里天天听见，却很可能从没真正搞懂——**利率**。

利率是什么：钱也要付租金

先把这个词讲成人话。利率就是**钱的租金**——你借别人的钱用，得为「用这笔钱」付一个价钱，这个价钱占本金的比例，就是利率。

> 大白话
>
> 你租房子，占用房东的房子，要付房租。你借钱，占用别人的钱，也要付「钱租」。利率 5%，就是借 100 块，一年得还 105 块，那 5 块就是钱的租金。利率高，钱贵，借钱肉疼；利率低，钱便宜，借钱像白捡。

那么什么是零利率 / 低利率时代？大致从 2008 年金融危机之后，一直到 2021 年，美国的中央银行（美联储）为了把濒死的经济抢救回来，做了两件事：**把基准利率一路压到接近零**——钱的租金几乎为零；同时**大量印钱放水**，让市场上到处是找不到出路的闲钱。2020 年疫情来了，为了托住经济，这套「便宜钱」的水龙头又开得更大。

结果就是：有十几年时间，**钱几乎不要钱**。这是一个非常不正常的状态，但它持续得够久，久到一整代人以为这就是世界的默认设置。软件行业最猛的一段繁荣，恰好整个泡在这缸温水里。

关键一步：为什么「明年的一块钱」不值一块钱

现在讲全章最要命、也最反直觉的一个金融常识。请慢一点看，因为整条因果链都从这里长出来。

问你一个问题：**今天给你一块钱，和明年这时候给你一块钱，一样值钱吗？**

不一样。今天这块钱更值钱。因为如果我今天就拿到手，我可以立刻拿去钱生钱——存起来吃利息，或者投出去。假设利率是 5%，我今天的 1 块，明年就变成 1.05 块。反过来说，**明年的 1 块钱，换算到今天，只值大约 0.95 块。**「未来的钱」要打个折，才能跟「今天的钱」放在一起比较。这个折，就叫贴现（贴现 / 钱的时间价值）：**把未来的钱打个折，换算成今天的价值。**

> 大白话
>
> 把它想成快递运费。「未来的钱」要运到「今天」才能用，路上得扣一笔运费，扣完剩下的才是它今天真正值多少。运得越远（越晚才能拿到），扣得越多——十年后的一块钱，比明年的一块钱，折得狠得多。

而**这笔「运费」的高低，正是由利率决定的**。这是本章的枢纽，务必吃透：

- 利率**高**，运费贵——未来的钱折得狠，遥远的未来几乎不值钱。十年后的一百万，今天可能只当几十万看。
- 利率**低**，运费便宜——未来的钱几乎不打折，「以后才能赚到的钱」在今天就几乎能按原价算。
- 利率**趋近于零**，运费几乎为零——**未来的钱和今天的钱，差不多一样值钱了。**十年后的一百万，今天也当一百万看。

记住这最后一句。它听起来只是个算术，但接下来你会看到，它能凭空吹起一整个时代的估值。

软件公司：一台把利润押在遥远未来的机器

现在把贴现这把尺子，量到一家 SaaS 公司身上。

回想前面几章那台增长机器：一家典型的成长期软件公司，**今天是亏钱的**。它把每一块收入、甚至借来的每一块钱，都砸进获客、砸进烧钱换规模。它的价值不在此刻的账本上——此刻账本是红的——而在**很多年以后**：等它做到足够大、订阅用户足够多、护城河足够深，那时才会开始吐出巨额利润。

换句话说，**软件公司的价值，绝大部分堆在遥远的未来**。你买它的股票，买的不是它今天赚了多少，而是你相信它十年后能赚多少。它是一台把利润押在未来的机器。

那么，一台价值全在未来的机器，值多少钱？这不再是拍脑袋——它就等于**把那些遥远未来的利润，用贴现这把尺子，一年一年折回到今天，再加起来**。于是刚才那个「运费」的高低，就成了生死攸关的事：

> 一家利润在远方的公司，它今天的估值对利率极其敏感。**利率一低，未来的钱几乎不打折，市场就敢为「还没到来的利润」付很高的价。**

大白话

同一家公司、同样的产品、同样十年后能赚一百亿的前景：利率高的时候，这一百亿折回今天可能只值二十亿；利率接近零的时候，它折回今天几乎还值一百亿。公司没变，产品没变，只是钱的租金变了，它的身价就能翻好几倍。这不是玄学，就是那把运费尺子在起作用。

（严格说一句，好让较真的读者安心：给一家公司未来利润打折用的那把尺子，不只是美联储那个基准利率，还得加上一块「你这生意有风险」的风险溢价——所以就算基准利率真到了零，未来的钱也不会「完全」不打折，只是折得很轻。但方向不变：基准利率越低，这把尺子整体就越松，远期利润在今天就越值钱。为讲清主线，下文说「不打折」都是这个「折得很轻」的简写。）

这就解释了一个让工程师困惑的现象：为什么那些年年亏损、看不到赚钱迹象的软件公司，估值却高得离谱？因为在零利率的世界里，**市场根本没在为「今天亏多少」定价，它在为「未来能赚多少」定价，而未来的钱此刻几乎不打折。**亏损被原谅了，因为亏损买的是未来，而未来很便宜。

于是「增长优先于利润」成了信条

一旦上面这套逻辑成立，一个在别的行业听起来疯狂的信条，就顺理成章地立住了：增长优先于利润——**先不赚钱，甚至拼命亏钱，只要换来增长就行。**

为什么理性人会这么干？因为在便宜钱的环境里，两个条件同时成立：其一，**未来的钱值钱**，所以把资源全押在「做大规模、以后再收割」这件事上，划算；其二，**融资又便宜**，钱多得是，投资人排队送钱，你根本不缺弹药去烧。既然如此，把每一块钱都砸进增长，就是最优解。亏损不但被容忍，**甚至被当成一枚勋章**——亏得越狠，越说明你在猛踩油门抢地盘。有一句话精准概括了那几年的风气：

> 「Growth at all costs」——不惜一切代价增长。利润是懦夫的追求，市场份额才是英雄的勋章。

这股风气最极致的例证，是那些年从天而降的巨额输血。软银的「愿景基金」拎着几百上千亿美元，在全球到处给创业公司注资，一出手就是十亿量级，逼着被投公司把规模冲到最大、把对手烧到出局。这里不展开这些公司的故事（它们不全是软件，也不是本章主角），我们只借它当一个氛围的注脚：**当钱便宜到这个地步，「快速做大」压倒一切，「什么时候赚钱」这种扫兴的问题，没人愿意问出口。**

一间开着暖气的温室

把整章收进一个类比，你就再也不会忘。

便宜的钱，就像温室里的暖气。它把水温调得又高又稳，于是各种平时活不下去的物种，都能在这里疯长。

大白话

那些「常年亏损、却高速烧钱」的公司，就是这类娇贵物种。在正常水温（正常利率）下，它们撑不了多久——投资人会追问利润，会掐掉输血。但在零利率这间开足暖气的温室里，未来的钱不打折、融资取之不尽，它们不但活了下来，还长得比谁都快、比谁都高，甚至让人误以为这就是它们本该有的样子。

整个 2010 年代到 2021 年的软件繁荣，很大一部分是这间温室的产物。满园的高速增长、天价估值、亏损却被追捧的明星公司，看上去枝繁叶茂、生机勃勃。**但请记住一件事：它们的繁茂，一半靠自己的产品，一半靠这间温室的暖气。**

埋在地基里的那个前提

现在，把这套繁荣的逻辑翻过来看它的背面，你会找到一个所有人都默认、却几乎没人说出口的**隐藏前提**：

> 「增长优先、亏损无所谓」这套信条能成立，唯一的地基是——**钱一直很便宜。**

这句话值得盯着多看几秒。整个盛宴的地基，**不在产品里，不在代码里，而在利率这只看不见的手上**。是接近零的利率，让未来的钱不打折；是不打折的未来，让亏损换增长变得理性；是便宜的融资，让烧钱可以一直烧下去。抽掉「钱一直便宜」这一块砖，上面所有的逻辑都会哗啦垮掉。

而暖气，不会永远开着。

利率是会掉头的。当有一天它从接近零开始往上抬（这就是第十二章的故事），运费尺子会瞬间变贵，未来的钱重新被狠狠打折。到那一刻，**同一家公司、同样的产品、同样的团队，估值逻辑会在几乎一夜之间反过来**：昨天还被追捧的「增长故事」，今天就被追问「你到底什么时候赚钱」。温室的暖气一关，那些靠暖气才长得起来的物种，会第一次感受到真实的寒冷。

两只手，一缸水温

最后，回到这本书的骨架，把这一章摆正它的位置。

这一章讲的，是一个**外部变量**——它和贯穿全书的那条「成本塌方」主暗线，是**并行的两只手**：

- 一只手是**技术**：交付成本一次次塌方，决定了谁能用更便宜的方式把软件送到用户手里，谁就重排为赢家。这是前九章的故事。

- 另一只手是**金融**：钱的价格（利率）高低，决定了整缸水的水温——市场愿意为「遥远的未来」付多高的价，容忍多大的亏损。

软件世界的冷暖，**一半由技术决定，一半由金融决定。**前九章我们只看见第一只手，容易误以为赢家全靠产品和成本结构取胜。这一章要你记住第二只手的存在：过去十几年那场盛宴的水温，是被便宜的钱人为烧高的。

下一章（第十一章），我们就走进这缸温水最烫的时候——2018 到 2021 年那波 SaaS 上市潮的顶点，看看具体是哪些公司、顶着怎样的财务长相，登上了浪尖。而它们脚下这块「钱一直便宜」的地基，能撑多久，我们留到第十二章去清算。

第十一章 · SaaS 军团上市潮：神话的顶点

第十一章 · SaaS 军团上市潮：神话的顶点

上一章讲了一只看不见的手：便宜的钱。利率低到贴地，市场愿意为「未来某天才会到来的利润」付今天的高价，「增长压倒利润」从一句口号变成了整个行业的信条。这一章讲这套逻辑被推到顶点的那几年——2018 到 2021，一大批做订阅软件的公司排着队冲上股市，市场用真金白银给它们盖了个章：这就是最优解。

先说清楚舞台。IPO（首次公开募股，一家公司第一次把自己的股票卖给公众、正式登上证券交易所）在这几年密得像下饺子。而且冲上来的不是杂牌军，是一支长相高度一致的队伍。

点名一支军团

随手就能列一长串：Zoom、Slack、Snowflake、Datadog、CrowdStrike、Zscaler，再往前一两年还有 Okta、Atlassian 打了头阵。它们做的事各不相同——开视频会、发消息、存数据、看监控、防黑客、管登录——但它们全是一个物种：SaaS（软件即服务，software as a service，软件不再是你买一张光盘装在自己电脑上，而是跑在人家云上、你按月按年订阅、打开浏览器就能用）。

我不打算一家家给你写公司百科，那没意思。重要的是它们凑在一起的那张**共同的脸**——四条财务特征，几乎每一家都长这样。看懂这张脸，你就看懂了整个时代的疯狂。

共同的财务长相：四条一起看

第一条：高增长

高增长指的是营收（一年卖了多少钱）同比涨得飞快。传统公司一年营收涨个百分之十几就算不错了，这批公司常年是几十个百分点，最猛的那阵子能翻倍——去年卖一个亿，今年卖两个亿。这条是所有故事的引擎：只要这个数字还在猛冲，市场就愿意原谅它别的一切毛病。

第二条：高毛利

毛利率是个必须讲成人话的词。它问的是：你卖出一块钱的产品，扣掉「为了交付这一块钱产品直接花掉的成本」之后，手里还剩几毛。剩得越多，毛利率越高。

> 大白话
>
> 开个饭馆，卖一碗一百块的面，光食材就得三四十块，毛利率也就六成出头，还得天天买菜。软件不一样：第一份产品造出来很贵，但复制第二份、第一万份、第一百万份，几乎不花钱——多一个客户，无非是服务器上多跑一个账号。这就是第二章讲过的**边际成本近乎为零**。所以这批公司的毛利率常年在七八成以上：卖一块钱，成本就一两毛，剩下七八毛是「毛利」。

这条极关键。它意味着这门生意**天生就该很赚钱**——只要它愿意。记住这个「只要它愿意」，一会儿要用。

第三条：常年不盈利

诡异的地方来了。这批毛利高得吓人的公司，净利润（把所有成本、开销全扣完，最后到底赚是赔）却常年是**负的**——一直在亏，年年亏。

钱去哪了？没进产品成本（那部分很省），全烧在了「抢地盘」上：疯狂招销售、疯狂打广告、疯狂做市场，为的是把上面第一条那个增长数字顶得越高越好。它们是主动选择亏损的——把本该落进口袋的利润，全都再投出去买增长。

放在过去，一家常年亏损的公司股价该被打到地板。可这几年，市场不但不嫌弃，反而**追捧**。亏得越凶、增长越猛的，估值往往越高。这不是市场瞎了，这是市场在按一套特定的逻辑下注——马上就讲。

（这里点一个异类：Zoom。它 2019 年上市时居然是**盈利**的，在这支「亏损军团」里是个罕见的另类。这说明高增长和盈利并非天生对立，只是绝大多数公司主动选了前者。）

第四条：经常性收入 + 高留存

最后一条，是这套模式的定海神针，直接呼应第三章。经常性收入指订阅制带来的、会自动重复的收入——客户不是买一次就走，而是每个月、每年自动续费。这让公司的收入变得极其**可预测**：年初就大致知道今年能收多少，因为大部分是去年老客户续过来的。

更妙的是一个明星指标：NRR（净收入留存率，net revenue retention）。它问的是：**光看去年那批老客户，不算任何新客户，他们今年给你的钱是多了还是少了？**算法是拿老客户群今年掏的钱除以去年掏的钱。

- 如果是 100%，说明老客户群不多不少，钱没变。
- 如果**超过 100%**——比如 120%——意味着一分钱新客户没拉，光靠老客户「越买越多」（多开账号、升级套餐、加购功能），收入就自己长了两成。

> 大白话
>
> 这相当于一个漏水的水桶，往里加水（拉新客户）的同时，桶底还在漏（老客户流失）。NRR 超过 100% 是什么概念？是这个桶不但不漏，桶底还在**往外冒水**——你哪怕停止加水，水位自己都在涨。华尔街看到这个数字会两眼放光，因为它意味着增长有一部分是「白捡」的、自我生长的。

那么，市场当时到底在为什么买单？

把四条拼起来，答案就浮出来了。市场买的**根本不是当下的利润**——当下没有利润，是负的。市场买的是这么一个推理链：

> 这家公司锁定了一大笔每年自动续费的经常性收入（第四条），这些收入的毛利高达七八成（第二条），而且它还在高速扩张、疯狂圈地（第一条）。今天它之所以亏，纯粹是因为它**主动**把利润全砸进了增长（第三条）。那么总有一天，等地盘圈得差不多了，它只要**把增长这个开关关小一点**，不再猛砸销售和广告，那七八成的高毛利就会哗哗变成利润——它就能开始印钱了。

这就是核心。市场付的是**「未来某天关掉增长开关就能印钱」的预期**。它情愿现在忍受亏损，是因为它相信亏损是一种战略选择，而不是无能——今天亏得越狠、地盘圈得越大，将来印钱的机器就越大。

没有利润，拿什么估值？市销率登场

这套买单逻辑，逼出了一个当年人人挂嘴边的估值方式。传统上给公司估值，常看市盈率（股价相对于每股利润的倍数，用市值除以利润）——可这批公司**根本没有利润可除**，分母是负的，这个尺子直接失灵。

于是大家换了把尺子：市销率（P/S，price-to-sales，用公司的总市值去除以它的年营收，而不是除以利润）。既然利润还没到，那就先按「卖了多少钱」来估。

问题是这把尺子被拉到了荒诞的刻度。正常公司市销率也就几倍，可当年一些当红 SaaS 被推到了**几十倍**——市场愿意为它每一块钱的营收，付几十块钱的市值。翻译成大白话：**市场按这家公司未来好多年后才可能有的样子，给今天定价。**

这份狂热有个标志性的顶点：Snowflake（做云上数据仓库的公司）2020 年 9 月上市，是软件史上最大的 IPO 之一，挂牌**首日股价就翻着番往上冲**。连一向躲着科技新股、几十年不碰 IPO 的巴菲特（他的伯克希尔公司），都罕见地参与了这次 IPO——保守派的教父都下场了，你可以想象当时空气里那股「不上车就错过时代」的味道有多冲。（顺带一提，2019 年 Slack 上市走的是不太一样的直接上市（direct listing，老股东的股票直接挂牌交易，不新发股票融资）——形式各异，但都被裹进了同一场狂欢。）

点破地基：这份自信踩在哪只手上

现在把这一整套「亏着也值钱」的自信翻过来看它的地基。你会发现它整个悬在**第十章那只手**上。

市场为什么敢为「未来某天才印得出的利润」付今天的高价？因为利率低。回想第十章的道理：利率越低，「未来的钱」在今天就越不打折，未来某天的一块钱利润，几乎能当今天的一块钱来估值。正是这只手，让「今天亏、将来印钱」这笔账算得过来——如果未来利润几乎不打折，那么一家「将来必印钱」的公司，今天当然配得上天价。

换句话说，那几十倍的市销率里，藏着一个**从没被写进招股书、却被默认为真的假设**：

> 利率会一直这么低下去。钱会永远这么便宜。

市场不是在为这些公司的产品定价，而是在为「低利率永不结束」这个隐藏前提定价。所有那些漂亮的 NRR、高毛利、高增长，都是真的；但把它们翻译成「几十倍市销率」的那道汇率，是那只看不见的手偷偷给的。手在，汇率就在；手要是撤了……先按下不表。

收束到暗线：订阅这一档，红利吃到顶了

顺着全书那条暗线看：软件的交付方式一次次变便宜，每变一档，赢家就重排一次。到这几年，「订阅 + 云 + 高毛利 + 高留存」这套 SaaS 模式，被市场**盖章认证为最优解**——不光是好生意，是好到能顶着连年亏损还被抢着买。这是「订阅这一档红利」被吃到顶、被推到极致的时刻。整个行业都相信：找到终极答案了，往后就照这个抄。

结尾：所有人都以为这是新常态

2021 年底，空气里全是笃定。扎堆的 IPO、翻番的首日、几十倍的市销率、连巴菲特都下了场——所有人都认定，这不是泡沫，这是**新常态**，是软件生意进化出的最终形态，往后的世界就长这样了。

可你已经看见那块藏起来的地基了。这一整套光鲜定价，押的其实是一个赌注：**钱，永远这么便宜。**只要那只看不见的手不动，这场盛宴就能一直办下去。

2022 年初，那只手，动了。

第十二章 · 利率一涨，潮水退了

第 10 章我们埋了一个问题：低利率像给整个软件行业开了暖气，把估值烘得暖洋洋——那要是有一天，暖气被人一把关掉呢？

2022 年，那只手动了。

钱的价格，突然涨了

先说一件在 2022 年之前十来年里几乎没发生过的事：借钱，变贵了。

那些年通货膨胀低、经济需要刺激，美联储把利率压得很低很低，钱几乎是白送的。可 2022 年物价开始猛涨，为了把通胀压下去，美联储掉头就是一连串又快又猛的加息——把持续了十来年的低利率环境，在一年之内几乎翻了个面。

对普通人来说，这意味着房贷车贷变贵。对一家软件公司来说，这件事的杀伤力，要拐一个弯才看得清。而这个弯，正是第 10 章那把尺子的反面。

> 大白话
>
> 第 10 章说过：一家公司值多少钱，本质是把它**未来能赚的所有钱**，按利率"打个折"折算到今天。利率就是那个折扣的力度。利率低，未来的钱几乎不打折，公司显得很值钱；利率高，未来的钱被狠狠打折，公司立刻就显得没那么值钱了。

为什么什么都没变，股价却能跌一半

这里有个让很多工程师第一次听会愣住的事实：2022 年那一波高估值软件股、成长股整体大幅回撤，很多公司股价**普遍腰斩、跌去大半**——可它们的产品没变，客户没跑，增长也还在。

怎么会这样？

关键在于：软件公司的价值，大头压在**很遥远的未来**。一家 SaaS 公司今天可能还在亏钱，市场愿意给它高估值，是因为大家相信它十年、二十年后会赚很多很多。也就是说，它的身

家里，"眼前这点"占比很小，"远方那一大坨"占比很大。

而折现有个特点：**越远的钱，对利率越敏感**。利率涨一点，明年的钱缩水一丁点，但二十年后的钱会缩水一大截。所以对一家"价值主要在远方"的公司，折扣力度稍微一变，今天该值多少就天翻地覆。

> 大白话
>
> 换个说法。同样一栋房子，你用不同的尺子去量，读数就不一样。2022 年发生的，不是房子塌了，是**量房子的那把尺子换了刻度**。公司还是那家公司，产品还是那个产品，变的是"钱的价格"——而软件公司偏偏是那种"离尺子最远的一头"，尺子一动,它晃得最狠。

这就是第 10 章伏笔的答案：暖气关掉，最先冷的、冷得最厉害的，就是那些"全靠未来故事撑估值"的软件公司。它们涨的时候涨得最凶，是同一个道理反着走了一遍。

连锁反应：便宜的钱没了以后

估值腰斩只是第一张倒下的多米诺骨牌。钱一贵，一连串事情跟着来。

一、融资变贵变难，续命的输液管被拔了

融资变贵变难，说的是：过去很多烧钱的公司，账上的钱不够花没关系，随时能再融一轮便宜钱续上。投资人排着队给钱，因为钱本来就不值钱，投出去总比躺着强。

2022 年这条路断了。钱变贵，投资人捂紧钱包，估值又在跌——这时候融资，等于用更低的身价卖更多股份，谁都不乐意。于是那些一直靠"外部输血"活着的公司，输液管被拔了，只能靠自己造血活下去。可很多公司从成立起就没学过怎么造血。

二、裁员潮，"养一大堆人赌未来"的账算不过来了

紧接着是 2022 到 2023 年的科技行业大规模裁员潮——从大厂到创业公司，累计裁掉的人以**数十万**计。很多读这本书的工程师，就是那两年亲历者。

为什么突然就裁？因为账变了。便宜钱时代，市场只奖励增长，招人越多、摊子铺得越大，故事讲得越性感，估值越高——所以"养着一大堆人去赌一个遥远的未来"是划算的。钱一

贵，市场不再无条件奖励增长，这笔账立刻算不过来：养这么多人，烧这么多钱，换来的增长却撑不起这个估值了。

三、增长失速，疫情拉高的需求回落了

更难堪的是增长失速。2020、2021 年疫情期间，全世界被迫远程办公、上网买东西，云和 SaaS 的需求被硬生生拉高。很多公司把这段特殊时期的高增长，当成了会一直持续的常态，照着这个速度招人、定目标、给自己估值。

等生活恢复正常，那部分被"提前透支"的需求回落，增长曲线一下就平了。高增长神话破掉——偏偏是在市场最不能容忍"只有故事没有业绩"的时候破的。

四、评判标准，翻了个面

把上面三件事合起来，你会看到市场的评判尺子发生了一次根本翻转：

- 便宜钱时代：**只看增长，不管亏损**。你增长快，亏多少都行，反正未来会赚回来。
- 2022 年之后：**既要增长，也要能自己造血**。光会烧钱换增长，不行了。

就是在这个翻转的当口，一个朴素得不像"生死线"的指标，被所有人翻出来天天念叨。

Rule of 40：一把突然管用的照妖镜

40 法则 Rule of 40，是一条简单到近乎粗暴的加法：把一家软件公司的**营收增长率**，加上它的**利润率**，两个数加起来，最好达到或超过 **40%**。

大白话

用人话说：增长和赚钱，你至少得占一样，加起来够 40 分。

- 想高速增长、暂时亏钱？**可以**——比如增长 60%、利润率 -20%，加起来 40，及格。
- 想增长慢一点、但踏实赚钱？**也可以**——比如增长 10%、利润率 30%，加起来 40，及格。
- 又不怎么增长、又一直亏钱？**不行**——比如增长 10%、利润率 -20%，加起来 -10，这就是在裸泳。

它为什么突然成了生死线？因为它恰好衡量一件在便宜钱时代没人 care、钱一贵却要命的事：**这家公司到底是在"健康地"增长，还是纯靠烧钱买增长？**

增长本身不值钱——花两块钱买一块钱的增长，谁都会。值钱的是"不用一直往里贴钱也能长"。Rule of 40 把"增长"和"造血"两件事强行捆在一根秤上称：你可以偏科，但不能两门都不及格。在钱不要钱的年代，没人拿这把尺子量谁；钱一贵，它就成了照妖镜——一照，谁只有故事没有业绩，现出原形。

潮水退了，才知道谁在裸泳

这一章其实在说一件贯穿全书、但和"交付成本主线"平行的事：**决定一套商业模式能不能成，是技术；决定它在市场眼里值多少钱，是金融环境。**

技术让这些公司造出了漂亮的订阅模式、造出了"亏着也值钱"的成长故事。但这套故事能成立，一直悄悄押着一个前提——钱永远便宜。2022 年这个前提被抽掉，同一批公司、同样的产品、同样的团队，只因为"钱的价格"变了，就集体从云端摔了下来。

潮水退去，谁在裸泳一目了然：那些真有造血能力的公司，被打折但站得住；那些只有一个增长故事、从来没证明过自己能赚钱的公司，露了底。

效率，从没人在乎变成生死线

于是，活下来的公司被逼着回答一个突然变得要命的问题——

> 怎么用更少的钱、更少的人，办成同样多、甚至更多的事？

在便宜钱时代，"效率"是个没人在乎的词。钱多、人好招、增长被无限奖励，谁会去抠一块钱换回多少产出？可暖气一关，效率一夜之间从一个不起眼的词，变成了区分生死的那条线。

那么，那些活下来的公司，具体是**怎么**做到用更少的钱和人、办成同样多的事的？他们怎么把烧钱换来的增长，变成能自己造血的增长？这就是下一章的事。

第十三章 · 效率时代：少花钱，多办事怎么做到

第十三章 · 效率时代：少花钱，多办事怎么做到

第 12 章的最后，潮水退了。加息、估值腰斩、裁员、Rule of 40 上台——这些是"退潮时发生了什么"。这一章讲的是退潮之后：那些没被冲走、还站在沙滩上的公司，接下来到底在干什么。

答案听起来平平无奇，却是整个十年繁荣的账单：**它们开始学着用更少的人、更少的钱，换回同样多、甚至更多的收入。**行话叫"高效增长"，人话叫——终于开始好好过日子了。

> 大白话
>
> 便宜钱的时候，公司比的是谁跑得快，没人问"这一步划不划算"。钱一贵，问题变成了"每花一块钱，能换回多少"。同样一件事，思路完全反过来。

先想明白一件事：为什么以前不算账

要理解效率时代，得先理解它在还什么债。

过去十年融资太容易了。增长快，就有人给钱；给了钱，就能继续砸增长。在这个循环里，"浪费"几乎没有代价——多招几个人、多花点获客费、留着一堆用不上的云服务器，都无所谓，反正下一轮融资能补上。**增长像涨潮，海面一升，水下的礁石全被盖住了：低效、亏损、烧钱的坏习惯，一样都看不见。**

钱一贵，潮水退下去，礁石全露出来了。公司这才发现，自己身上到处是过去十年"先做大、再说"欠下的账。所谓效率时代，本质就是把这些账一笔笔还清。

所以要先破一个误会：**效率时代不是什么新发明，它是回归常识——一门生意，终究得自己能赚钱。**只不过便宜钱让大家忘了这条常识十年而已。

把生意拆到最小单位：单位经济学

还账的第一步，是把糊涂账拆开算清楚。这就是单位经济学 unit economics——把整个生意拆到"服务一个客户"这个最小单位上，看这一个客户到底赚不赚钱。

算这个客户，主要看三个数（前两个第 3 章已经出现过）：

- **获客花多少（CAC）**：把一个新客户拉进门，销售、市场、广告加起来花了多少钱。
- **这个客户一辈子赚回多少（LTV）**：他从签约到最后离开，一共给你交了多少钱、其中多少是真利润。
- **多久回本（回收期 payback period）**：你为拉他花的那笔获客钱，要多少个月才从他交的费里赚回来。

> 大白话
>
> 就像开一家店：拉一个客人进门花了 12 块，这客人平均每月给你带来 1 块利润，那要 12 个月才回本，之后才是净赚。回收期太长，说明你的客人还没帮你把成本挣回来就走了——生意再大，也是漏的。

便宜钱时代没人细算这个，因为不用算：回本慢没关系，融资能补。钱一贵，每个客户赚不赚钱、多久回本，全都得摊开在桌上看。回收期从"几年"逼到"一年出头"，成了硬指标。

现金流：比营收诚实的那个数

拆完单个客户，再看整盘生意。这里发生了效率时代最关键的一次视角切换：**从盯着"营收增长"，转向盯着"真金白银的现金流"。**

为什么？因为营收会骗人。你只要肯烧钱，营收几乎想堆多高堆多高——疯狂打折、给销售发天价提成、亏本卖，账面数字唰唰往上涨。营收好看，不代表公司真赚到了钱。

要看真的，得看自由现金流 FCF——公司经营一圈、该收的收了、该花的都花了（包括买设备、发工资）之后，真正能自由支配、揣进兜里的那笔现金。

大白话

营收像化了妆的脸，打光一好谁都精神；自由现金流像体检报告上的血压血糖，化妆盖不住。一个公司可能营收涨得很漂亮，体检单却一片红——钱全烧出去了。效率时代，投资人开始只信体检单。

顺带一个搭档概念：毛利率——每卖出一块钱的软件，扣掉直接成本（主要是服务器、带宽这些运行成本）之后，还剩几毛。软件生意本该毛利很高，一份代码卖一万个客户，多一个客户几乎不多花钱。毛利率低，往往说明成本结构出了问题。效率时代，这一毛一毛都被抠出来看。

人效：一个人到底能扛多少活

裁员是第 12 章的关键词，但效率时代的裁员不只是"省工资"。它背后是一个更冷静的问题：人效——平均一个人，能创造多少收入、多少利润。

便宜钱时代招人像囤货，"先招进来占着，业务总会用上"。结果很多公司是**虚胖**：人头翻了三倍，产出没翻三倍。裁员，某种程度上是在重新校准"一个人到底该扛多少活"这把尺子。

于是出现了一个耐人寻味的现象：不少公司裁掉两三成人之后，收入非但没塌，反而稳住、甚至继续涨。这不是奇迹，恰恰反过来证明——之前那些人里，有相当一部分本就是虚胖堆出来的，砍掉不疼。

销售效率：每花一块销售费，带回多少经常性收入

最后一块命根子是销售效率。旧打法是"猛砸销售换增长"——销售团队使劲扩，只要能签单，成本先不管。新打法是算一笔账：**每多花一块钱销售费用，能带回多少能年年续费的收入。**

业内有个粗略指标叫 magic number，你不用记公式，记它想问的事就行：一块钱销售投入，换回来的那份"每年都能再收一次"的收入，是多还是少。多，说明这台销售机器高效，值得继续加油；少，说明在往漏桶里灌水，得先把桶补上。

但别搞错：效率不是一味紧缩

讲到这里容易走偏，以为效率时代就是全面勒紧、什么都砍。恰恰相反。

真正的高手是"该省的狠省、该投的照投"——把钱从低效的地方，挪到高回报的地方。砍掉回不了本的获客渠道、砍掉虚胖的人头、砍掉没人用的项目；但对那些真能带来高效增长的产品线、真能提人效的工具，反而加码投。

> 大白话
>
> 紧缩和高效是两回事。一刀切全砍，是把肌肉和脂肪一起削掉——短期数字好看，却把未来的增长也杀死了。过度紧缩的公司，往往下一年就没故事可讲了。

这一章在暗线上的位置

回到本书那条暗线：软件的交付方式一次次变便宜，每变一档，赢家重排。

但这一章有点特别——**它不是"交付成本又降了一档"，而是上一轮红利吃完、泡沫出清、大家一起回到地面的整固期。**没有新技术让软件更便宜，只是把过去十年欠的账还清、把身上的虚肉削掉。

还账很痛，但它把这批公司锻炼得精瘦、务实：懂得算每一个客户的账，懂得盯现金流而不是盯营收，懂得把钱花在刀刃上。这套"精打细算"的肌肉，本身没那么性感，却是接下来所有故事的地基。

结尾：肌肉刚练好，地就开始晃

就在这批公司刚学会精打细算、把每一块钱都花在刀刃上的时候，一个新变量出现在地平线上。

它有两副面孔。一副正合效率时代的胃口——它能让"办同样一件事"的成本再降一大截，写代码、做客服、跑营销，样样都能更省。刚练出降本增效肌肉的公司，看到它简直如获至宝。

但它的另一副面孔更吓人：它可能把前十二章辛苦建立的整套规则——怎么定价、怎么获客、护城河靠什么、一个人能扛多少活——连根拔起。

它就是 AI。下一章，我们看看这个既能顺着效率时代往下走、又可能把整张牌桌掀翻的东西，到底会带来什么。

第十四章 · AI：当写软件本身变便宜

第十四章 AI：当写软件本身变便宜

上一章我们停在了一个岔路口。效率时代教会了每家公司算清单位经济学、把现金流看得比故事重，AI 站在门口，一手拿着"再帮你砍一刀成本"的刀，另一手却按着整套规则的开关。这一章，我们就走进这扇门，看清它到底动了什么。

先说结论式的那句话，因为它足够重要，值得放在最前面：

前十三章里，每一次"变便宜"的都是把软件送到你手里的方式——光盘、订阅、云、移动、数据。这一次，变便宜的是"造出软件"这件事本身。

> 大白话
>
> 大白话：以前是运货的路越修越便宜，货还得一件件造出来。这次是造货的车间突然便宜了。位置往上游挪了一大截，所以震动也更大。

到底什么变便宜了

2022 年底，ChatGPT 出现，普通人第一次能用大白话跟一个程序对话，让它写文章、写代码、答问题。它背后是一类叫 大语言模型 LLM（用海量文本训练出来、能理解和生成语言与代码的 AI 模型，GPT、Claude 都是这一类）的东西。

对我们这本书的主角——上市软件公司——来说，重点不是"AI 好神奇"，而是一件很具体的事：**过去要养一队人才能干的活，现在很多能让模型干。**写代码、写文案、做客服、做数据分析，这些活的边际成本正在往下掉。一个工程师配上模型，能顶过去几个人；一段以前要排期两周的功能，可能一个下午就出了雏形。

把这件事翻译成前面章节的语言：过去十三章里，软件公司卖的是"帮你把事做了"——帮你管账、管客户、管代码。而现在，**"做这些事本身"的边际成本正在塌方**。造软件的车间便宜了，那么，靠"造软件很贵、所以做出来就值钱"建立起来的每一条规则，都得重新称一称还剩几斤几两。

下面逐条来看。请注意基调：这些是**正在发生、结局未定**的事，不是已经盖棺的历史。我会尽量把两边的逻辑都摆清楚，而不是替你选边。

规则一：按用量付费回潮，动摇纯订阅

回忆第 3、11 章的地基：软件复制一份几乎不花钱，边际成本近乎为零。正因如此，"固定月费、随便用"的纯订阅模式才成立——你用一次还是一万次，我的成本差不多，那我按人头收月费最省事、最好算、最容易滚成高毛利。

AI 把这块地基敲了一道裂缝。**LLM 每接一次请求，都真实地烧一次算力。**模型跑在昂贵的 GPU（一种原本用来渲染游戏画面、后来发现极擅长做 AI 计算的芯片）上，每一次调用都有实打实的电费和硬件折旧。这不再是"复制一份几乎免费"，而是"每答一句话都在花钱"。

于是一个老朋友回来了：按用量计费 usage-based（用多少付多少，正是第 4 章云计算那套逻辑）。在 AI 产品里，你会越来越常看到"按调用次数""按处理的字数（token）"计费，而不是一口价随便用。因为厂商扛不住——你用得越猛，它亏得越多。

> 大白话
>
> 大白话：纯订阅像自助餐，一口价随便吃，因为每盘菜的成本几乎为零。AI 这道菜，每上一盘后厨都真在花钱，自助餐就不好开了，得改成按盘算。

这件事往下推，动的是软件公司最看重的那个数字：**毛利**。SaaS 之所以被市场追捧，很大原因是七八成的高毛利。现在给产品塞进 AI 功能，等于在成本里塞进一块会随用量长大的算力账单。有人正在赌算力会持续变便宜、这块成本终将被摊薄回去；也有人担心，AI 功能会把 SaaS 那套漂亮的高毛利，实实在在往下拉一截。哪一边对，此刻还没有答案。

规则二：护城河从"功能"转向"数据和分发"

这本书从头到尾在讲一件事：护城河一直在迁移。现在它可能又要挪一次窝。

逻辑很直接：**如果任何人都能用 AI 快速拼出一个功能相近的产品，那"功能多"就不再是护城河。**过去你家产品比对手多三十个功能、多两年积累，这是壁垒。可当做功能这件事本身变便宜，壁垒就漏水了——对手用模型追平你，可能只要几周。

那什么还守得住？答案要回到前面几章去找：

- **专有数据**（呼应第 9 章）：别人拿不到的数据。模型再强，也变不出你独家的客户交易记录、行业沉淀。数据喂给模型，才是别人抄不走的那部分。
- **分发渠道和用户关系**（呼应第 6 章）：你已经躺在几千万用户的手机里、已经嵌进企业每天必开的那个后台。做出功能是一回事，把功能送到用户面前、让他们习惯用你，是另一回事——而这一步，AI 并没有变便宜。
- **深度嵌入的工作流**：当一个软件缠进了报销流程、审批链、合规审计，换掉它的成本高得吓人。代码好写，但把这条流程重新接一遍、重新取得信任，不好办。

所以会出现一个有点反直觉的结果：**"AI 让做功能变便宜"，反而让"谁手里有独家数据、谁离用户近"更值钱了。**越是人人都能造功能，那些造不出来的东西——数据、渠道、信任——就越稀缺、越贵。护城河没有消失，它只是又往里退了一层。

规则三：SaaS 会不会被自己写的代码吃掉

这是眼下最激烈的一场争论，值得把两边的逻辑都摆开。

一种极端说法是"**SaaS 已死**"。它的逻辑是这样：SaaS 的本质，是有人提前把一个软件写好，你按月租来用。可如果 AI 能**按需、现场**生成软件——你要什么功能，当场生成一个给你——那还需要买一个个现成的、固定的 SaaS 吗？为什么要为别人预先写好的功能付月费，而不是让模型即时给我造一个正好合身的？照这个逻辑，一大批"就是一个数据库加几个表单"的中小 SaaS，最危险。

另一种声音要冷静得多，它提醒你：**软件的价值，从来就不只是"那段代码"。**

- 代码背后有**数据**——多年积累的、干净的、结构化的业务数据，不会因为代码好写就自动出现。
- 有**集成**——和上下游几十个系统对接好的管道，现场生成的代码接不上。
- 有**合规**——过了审计、符合监管、扛得住法务的那套东西，不是即兴生成能保证的。
- 有**可靠性和责任承担**——出了事有人负责、有 SLA、半夜宕机有人接电话。你敢把工资发放交给一段"刚刚现场生成、没人担保"的代码吗？

大白话

大白话：会盖房子的机器变便宜了，不等于你会住进一栋刚现浇、没验收、没人担保的楼。写代码只是盖毛坯，能住的房子还要水电、验收、物业和"出事找谁"。

我不替你选边。把两边的逻辑摆清楚，你自己判断：这更像"整个品类被淘汰"，还是"最薄的那一层被吃掉、有数据有信任的那层反而更稳"。**大概率不是非黑即白**——某些薄 SaaS 会被现场生成取代，某些重 SaaS 会因为 AI 而更深地扎进客户。真正的问题不是"SaaS 死没死"，而是"你这个 SaaS，除了那段代码，还剩下什么别人夺不走的"。

规则四：谁吃到这一波、谁被威胁

第 8 章讲过一条老规律：淘金热里最稳的生意是卖铲子。AI 时代的铲子是谁？

1. **算力/GPU**：所有 AI 都得在芯片上跑，不管上层谁输谁赢，算力都在收钱。这是最底、最硬的一层铲子。
2. **基础模型**：训出 GPT、Claude 这类底座模型的厂商。别人的 AI 功能，很多是调它们的模型接口——它们卖的是"智能"本身。
3. **AI 开发工具**：帮工程师用好 AI 的那层工具——代码助手、模型编排、评测监控。淘金的人多了，卖工具的跟着赚。

但铲子这个比喻有个陷阱，得点破：**并不是所有站在 AI 边上的公司都在卖铲子，很多只是在别人的铲子外面套了层壳。**

这就是眼下最大的隐忧，业内叫 套壳 wrapper（只是在别人的大模型外面包一层薄薄的界面，自己没有独家的东西）。你做了个漂亮的对话框，后面接的是别家的模型——问题来了：

- 模型厂商随时可能**自己往上做一层**，把你这层壳顺手覆盖掉；
- 下一个套壳者可能做得**更漂亮、更便宜**，而你没有任何拦得住他的东西；
- 你把"智能"这个核心外包给了别人，等于把命门交了出去，模型一涨价、一改规则，你就被动。

注意，这恰好又绕回了规则二。**套壳危险，不是因为"套壳"这个动作本身，而是因为它往往什么护城河都没有——没有独家数据、没有分发、没有嵌进工作流。**反过来，如果一

层"壳"背后压着别人拿不到的数据、已经触达的用户、缠进业务的工作流，那它就不只是壳，它是有护城河的产品，只是恰好用了 AI 而已。区别不在于用不用模型，而在于**除了模型，你还有没有别人夺不走的东西**。

为什么这一档冲击可能更大

把这四条规则合起来看，你会发现 AI 这一档和前面几次的区别。

光盘变订阅、订阅变云、云变移动、移动变数据——前面每一次，变便宜的都是"**把软件送到用户手里**"这件事，动的主要是分发和交付。而这一次，变便宜的是"**造出软件**"这件事，位置更靠上游。

正因为它踩在更上游，所以它一次踩中了三条线：**成本**（算力烧钱，规则一）、**护城河**（功能不再稀缺，规则二）、**收入模式**（纯订阅松动、SaaS 边界被质疑，规则一和三）。前面几次"变便宜"，大多先冲击其中一条，再慢慢传导到别处；AI 这一次，三条线几乎同时开始晃。这是它可能量级不同的原因。

> 大白话
>
> 大白话：以前是改了送货方式，慢慢逼着你改定价和打法。这次是改了造货方式，成本、壁垒、收钱方式一起开始动，所以让人心里更没底。

站在岔路口

请把这一章当成一张"进行中"的地图，而不是一份判决书。到 2026 年的此刻，没人真的知道：算力会不会一路便宜下去、把毛利救回来；纯订阅会退到什么程度、按用量计费会占多大盘子；哪些 SaaS 会被现场生成吃掉、哪些会因 AI 扎得更深；今天风光的套壳里，哪些会长出真护城河、哪些会在下一个季度消失。**有人赌算力会救毛利，有人赌毛利回不来了；有人赌 SaaS 已死，有人赌它换个活法继续长——这些赌注都还没开盘。**

但有一件事是清楚的：走到这里，我们已经数完了整整六次"变便宜"——光盘、订阅、云、移动、数据，再到这一次的 AI。它们摆在一起，隔着二十多年，样子却越来越像。

它们像不像同一件事，在反复上演？如果真是同一件事，那么能不能从这六次里，提炼出一条共同的规律——一副眼镜，让我们不只看懂已经发生的，还能自己往前推演下一次？

这，就是最后一章要做的事。

第十五章 · 一条暗线：交付成本每降一档，赢家就换一批

第十五章 · 一条暗线：交付成本每降一档，赢家就换一批

走到这里，你已经看完了二十年、十四段故事。光盘、订阅、云、手机、增长黑客、卖铲子、数据、零利率、加息、AI……它们表面上各说各的，像十四件互不相干的事。这一章我们做一件事：把它们排成一条线，让你看清——**这从来不是十四件事，而是同一个动作，反复上演了六七遍。**

大白话

这一章不讲新故事。它给你一副眼镜。戴上它，前面的每一章会突然对齐成一条直线；摘下它，你还能自己往前看——看 AI，看下一个还没出现的东西。这是全书真正想留给你的东西。

先把那条线钉在墙上

从第二章起，我们其实一直在追一个数字：**把软件送到一个新客户手里、并让它持续跑好，要多花多少钱。**不是研发的钱（那是一次性的），是"每多服务一个人"的钱。这个数字，决定了一门软件生意的天花板。

把二十年的账摊开，你会看到这个数字一档一档往下塌，每塌一档，同一套剧情就重演一次：

1. **光盘时代**：复制免费，可"送到手里、跑好"极贵——盗版、上门部署、销售军队、逐单谈判。天花板是硬的。
2. **订阅**：软件从卖断变成租，不装进你机房，跑在我这边。交付的边际成本塌了一档——升级一次全员生效，盗版失去意义。
3. **云**：连"我这边的服务器"都不用自己买了，固定成本变可变成本，创业的门槛跟着塌。
4. **手机 + 社交**：把软件送到人眼前的"分发"成本塌了——不再需要货架、代理商、地推。

5. **数据 / 卖铲子**：把软件卖给工程师、让它嵌进代码和数据里，获客与嵌入的成本又换了一种更低的算法。
6. **AI**：这一次塌的是最核心的一环——**生产软件本身**。写代码、做客服、跑营销的边际成本，正在往下掉一个数量级。

六次塌方，每一次都不只是"更便宜"这么简单。每一次，都会接连发生三件事，顺序几乎从不变：

- **旧护城河贬值**：上一档成本结构撑起来的那道墙，突然不值钱了。
- **新护城河浮现**：值钱的东西挪到了下一个还稀缺的环节。
- **赢家名单重排**：把上一套规则玩得最溜的人，恰恰最容易被换掉。

为什么"变便宜"总是先骗人，再筛人

这里有一个反复出现、值得单独钉住的规律。**每一次成本塌方，都会先制造一场"人人都能赢"的幻觉，紧接着，新的稀缺浮出水面，把赢家重新筛一遍。**

云和开源让创业变便宜时（第五章那场"寒武纪大爆发"），无数人以为门槛没了、大家都能做软件了。确实，"做出来"不再稀缺了——可正因为人人都做得出来，"有人用"立刻变成了新的稀缺。门槛没消失，它只是往后挪了一格。

> 大白话
>
> 这就像修了一条更宽的路。所有人都涌上去，以为从此畅通无阻，结果堵在了下一个路口。便宜从不消灭竞争，它只是把竞争推到下一个还没变便宜的环节。稀缺永远在，只是换了个地方站。

把这条规律接成一串，你会看到稀缺一路搬家：**能不能做出来 → 有没有人用 → 留不留得住 → 有没有独家数据 → 送不送得到人面前……**每一次塌方，都把队伍往后赶一格。谁只顾守着上一格，谁就被留在原地。

眼镜：五条能自己往前推的透镜

下面五条，是这本书真正想交到你手上的东西。它们不是结论，是一套可以对着任何一家软件公司、任何一次技术变化去问的问题。当检查清单用。

透镜一：先看成本结构，别先看故事

拿到一门软件生意，第一个问题永远是：**它每多服务一个客户，要多花多少钱？这个数字是往上走还是往下走？**

边际成本趋近于零，天花板就高（这是软件最底层的引力，第二章讲过）；边际成本压不下去——像光盘时代的上门部署、销售军队——天花板就是结构性的，再努力也顶不破。故事讲得再漂亮，先把这个数字问出来，很多幻觉当场破掉。

透镜二：护城河会迁移，且没有永久产权

数一数这本书里出现过的护城河：功能领先 → 切换成本（订阅把你绑住） → 网络效应（社交） → 数据飞轮 → 分发渠道。**它们不是并列的选项，是一条被成本塌方一次次往前推的迁徙路线。**

关键在于：**每一道护城河都是租来的，没有一张地契是永久的。**上一次塌方立起来的墙，下一次塌方就可能把它冲平。所以别爱上任何一条护城河——要问的不是"它现在有多深"，而是"下一次成本塌方，会不会让这条河一夜干涸"。

透镜三：便宜先造过剩，过剩再逼出新稀缺

这是"变便宜先骗人再筛人"那条规律的操作版。每当某环节变便宜，先预判：**供给会在哪里爆炸性过剩？过剩之后，真正值钱的东西会挪到哪个还稀缺的环节？**

做出来不稀缺了，就去看谁能让人用；用起来不稀缺了，就去看谁能留住人、谁攥着别人没有的数据。**钱永远流向队伍里还没排到的那一格。**盯住那一格，别在已经过剩的环节里卷。

透镜四：技术定"能造什么"，钱的价格定"此刻值多少"

这本书有两只手在同时拨动这批公司。一只是**技术**——它决定你能造出什么、成本能压到多低。另一只是**金融**，也就是"钱的价格"（利率）——它决定市场此刻愿意为同一样东西付多少。

零利率那几章最能说明问题：公司没变，产品没变，可"钱的价格"这把尺子一变，同一家公司的估值就腰斩（第十二章）。**技术决定长期能不能成，钱的价格决定短期市场愿不愿意为**

它买单。只看一只手，你会把"钱变贵了"错当成"公司变差了"，或者把"钱太便宜"催出来的泡沫错当成"技术真的赢了"。两只手要一起看。

透镜五：最危险的，是把上一套规则玩到极致的人

回到第二章那个最扎心的例子。Siebel 没做错任何事——它把"贵销售、大单子、装进客户机房"这套卖断打法做到了行业头名，拿下市场大半。可正因为它把上一套规则玩到了极致，当订阅塌方来临时，它船大难掉头，最危险的恰恰是它。

被淘汰的往往不是做得差的人，而是把上一套规则玩得最好的人。做得差的人本来就没什么可失去；玩到极致的人，整个身家、整套组织、全部肌肉记忆都长在旧规则上——换规则，就是要他把自己最擅长的东西亲手拆掉。所以看一家公司危不危险，别只看它现在多强，要看它的强，是不是全押在那道正在贬值的护城河上。

把眼镜架到 AI 这个岔路口上

现在，戴着这五条透镜，走到我们此刻正站着的岔路口——AI（第十四章）。我不替你下结论，也不预言谁会赢。我只把该问的问题，一个个摆到你面前，剩下的推演交给你。

这一次塌的是哪一环？前几次塌的是交付、分发、创业门槛；这一次，塌的是"生产软件"本身。用透镜一问：当造软件的边际成本掉一个数量级，什么会过剩？"能写出这个功能"还稀缺吗？

哪条旧护城河要贬值？用透镜二问：如果软件本身能被 AI 快速仿造，那"功能领先"这道最老的墙还剩多少？靠"做得比你好一点"守身家的公司，是不是站在最松的土上？订阅那套高毛利，会不会因为生产成本塌了而被重新定价？

新稀缺会浮现在哪？用透镜三问：做软件不稀缺了，值钱的东西往哪挪？算力？别人拿不到的独家数据？把东西送到人面前的分发？还是——当满世界都是 AI 生成的东西时，**"信任"**本身成了新的稀缺？

此刻的高估值，是技术真赢了，还是钱又变便宜了？用透镜四问：分清哪一部分是 AI 真的改了成本结构，哪一部分只是市场情绪又把尺子拉长了。

谁是"把上一套规则玩到极致、因而最危险"的那个？用透镜五问：今天谁最像当年的 Siebel——整套身家都押在"卖高毛利订阅软件"这套正在被 AI 动摇的规则上？

大白话

看，我一个答案都没给。但如果这五个问题你现在能自己问出来、自己往下推，那这本书的活就干完了。授人以鱼，不如给你这副能一直戴下去的眼镜。

结尾：活下来的，从来不是某一家公司

回到书名——《活下来的机器》。

走完这二十年你会发现一件事：**没有哪一家公司是永远的赢家。**每一次成本塌方，都会掀翻一批曾经不可一世的名字，捧起一批新的。Siebel 输给了订阅，某些订阅巨头正被 AI 逼问。名单一直在换。

那"活下来的机器"到底是谁？

它不是任何一家公司。它是那台更大的东西——**整个行业本身：一台每当成本塌方，就自动把护城河、收入模式、赢家名单全部重排一遍的机器。**它不心疼任何一个旧赢家，也不偏爱任何一个新贵。它只忠于一条铁律：成本降一档，一切重排一次。

所以真正"活下来"的，从来不是某家具体的公司，而是**一种能力**——不断随着成本结构，把自己拆掉、重造一遍的能力。谁攥着这种能力，谁就能一次次搭上下一班车；谁把身家押死在某一档规则上，哪怕曾经是头名，也终将被这台机器筛下去。

合上书之前，把这句话带走：

> 这个行业里，没有永远的赢家，只有反复自我重排的机器。看懂它怎么排，你就不必再猜谁会赢——因为你已经会自己往前推了。