护城河与塌方

导读

从大型机到 AI，软件公司为什么登顶、为什么崩塌、为什么有的能重生

1995 年，Netscape 上市那天，它是全世界最耀眼的软件公司。浏览器几乎人人都用，市值一飞冲天。四年后，它基本消失了。

差不多同一时期，微软错过了搜索，又错过了移动，两次都慢了整整一个身位。按理说，慢一步就该出局——Netscape 慢半步就出了局。可微软不但没死，二十年后还重新站上了全球市值第一。

同样是霸主，为什么一个几年就灰飞烟灭，另一个连错两次还能王者归来？如果你觉得答案是"微软钱多"或者"运气好"，那这本书想说服你：不是。

大白话

换句话说，公司的生死不太靠英雄或蠢货，更多靠它当时站在什么位置、被什么力量推着走。这本书想找的，就是那些力量。

这不是一本公司八卦编年史

从大型机、PC、桌面软件，到浏览器、搜索、手机、云，再到今天的 AI——每一波里都有登顶者和塌方者。把它们的故事按时间摆一排，很容易看成一部热闹的兴衰录：谁赢了，谁输了，谁东山再起。

但这本书的兴趣不在"谁"，在"为什么"。它假设：这些起伏背后有**可复用的因果机制**——一小把反复出现的规律，在不同年代、不同公司身上一次次上演。看懂一次，就能认出下一次。

所以书里每家公司，都被当成某条规律的一个实验样本：IBM 是"制度冲击如何凭空造出一个产业"的样本，微软是"控制标准就能收税"的样本，Netscape 是"当你的功能被上游平台吸收会怎样"的样本，柯达式的故事则是"赚钱的老业务如何亲手杀死公司"的样本。

一个词贯穿全书：护城河

全书真正的主角，是护城河——一家公司挡住对手、让自己躺着赚钱的那道屏障。它可能是网络效应、是标准的控制权、是分销渠道、是规模带来的成本优势。

有意思的地方在于：护城河既是登顶的原因，也常常是塌方的原因。同一道让你所向披靡的屏障，换个时代、换个范式，会变成困住你的墙。这本书想回答的，正是那个贯穿始终的问题：

> 护城河到底从哪来，又为什么会在一夜之间蒸发？

怎么读

两种读法都行：

- **顺着读**：十六章是一条线，时间往前走，机制层层递进，从软件为什么是门怪生意，一路推到今天的 AI 平台战争。
- **挑着读**：每一章自成一篇。对哪家公司、哪个机制好奇，直接翻进去就行，不必从头。

书里的专有名词，出现时都会用一句白话带过，不需要你先懂技术。

你能带走什么

读完，你手里会多一副思维工具：一套分析**任何**软件公司的框架。它能用来复盘过去，也能拿去看今天——比如那些正在疯狂融资、看起来护城河深不见底的 AI 公司，到底站在坚实的地基上，还是站在一块随时会移动的板块上。

至于每一章的结论，这里先不剧透。翻页就是。

第一章 · 软件为什么是门怪生意

假设你要开一家工厂造椅子。第一把椅子很贵——你得设计它、买机床、租厂房。但你心里有底：等生产线转起来，第二把、第三把、第一万把，每一把都要实打实地消耗一块木头、一份工时、一度电。造得越多，花的钱越多，只是每把便宜了一点。这是我们对"做生意"的全部直觉：要东西，就得付出成本；东西越多，成本越高。

软件把这条直觉整个掀翻了。你写一个 Excel，可能砸进去几百个工程师、好几年时间、几个亿的钱。但当第一份写完、能跑了，你复制第二份要多少钱？几乎是零。复制第一百万份呢？还是几乎零。你没多雇一个人,没多买一块木头。这就是软件这门生意"怪"的起点,也是这本书后面每一个登顶与塌方故事的第一块地基。

第一条怪律：写第一份贵得要死，复制第二份几乎免费

先把这个词说清楚。边际成本——多做一份东西，额外要掏的钱。造椅子的边际成本，就是那块木头加那份工时，是一个实实在在、不会消失的数。而软件的边际成本，趋近于零：把文件拷贝一遍、让人从网上下载一次，几乎不花你一分钱。

> 大白话
>
> 说人话：造椅子是"每多一把都要再花一把的钱"；软件是"第一份烧掉一座金山，之后随便印"。软件的成本几乎全压在动手之前，一旦造出来，卖一份和卖一亿份，成本差不多。

与之搭配的是另一个词。固定成本——不管你最后卖出去一份还是一亿份，都得先花掉的那笔钱，比如研发。软件是一门**固定成本极高、边际成本极低**的生意。这两句话拆开都平平无奇，合在一起就爆炸了：因为它决定了谁能赢、能赢多大。

第二条怪律：成本结构天然导向"赢家通吃"

我们来推一遍，看它怎么"天然"就走向一家独大。

假设市场上有两家做同类软件的公司，A 卖得多，B 卖得少。A 把几个亿的研发成本摊到一千万个用户头上，每个用户只分摊到几块钱；B 只有十万个用户，同样的成本摊下来，每个

用户身上压着几百块。**卖得多的那家，每一份的成本反而更低。**于是 A 可以卖得更便宜、或者拿更多利润去请更好的工程师、做更多功能——然后卖得更多，成本又更低。这是一个会自我加速的循环。

> 大白话
>
> 说人话：在造椅子的世界里，工厂大到一定程度就不划算了（木头要运更远、管理更乱），有个"最优规模"卡着你。软件几乎没有这个天花板——越大越省，越省越强，赢家会一路滚雪球滚到把对手甩没影。

这个词值得记住。赢家通吃——一个市场里，第一名不是比第二名好一点点，而是几乎拿走全部，第二名连汤都喝不上。为什么软件动不动就赢家通吃？因为它的成本结构就是往这个方向使劲。造椅子的世界能容下几十个椅子厂各活各的，软件的世界常常只给第一名留位置。**后面你会反复看到：某个领域一旦有人跑到第一，追赶者往往不是"差一点"，而是"根本上不来"。**记住这条，它是后面很多"一将功成万骨枯"故事的底层原因。

第三条怪律：用户被"粘住"了，换不动

光是赢家通吃还不够狠。真正让护城河变深的，是第二个机制。

锁定（lock-in）——用户就算想换别家，换的代价高到让他换不动，只好留下。注意，"代价"往往不是软件本身的价钱。假设一家公司全体员工都用某个办公软件，几万份文档是它的格式，所有人的操作习惯、所有的插件和流程都长在它上面。现在来了个功能更好、还更便宜的对手，你换不换？多半不换——因为要重新培训所有人、转换所有旧文档、重写所有流程，这笔账算下来贵得吓人。软件明明是虚的，却像水泥一样把用户浇筑在原地。

> 大白话
>
> 说人话：让你走不掉的，常常不是这个软件多好，而是"离开它太麻烦"。你的数据、你的习惯、你周围人的选择，全都成了拴住你的绳子。卖软件的最爽的地方就在这：客户一旦进来，就很难走。

锁定意味着，护城河不光靠"我比你好"，还靠"你离开我太亏"。这就是为什么软件公司常常在你没察觉时，已经悄悄把你圈住了。

第四条怪律：用的人越多，它就越值钱

最后一块地基，比前面几条更反直觉。

网络效应——一样东西，用的人越多，它对每个人就越有用；反过来用的人少，它就越没用。椅子没有网络效应：别人坐不坐同款椅子，跟你坐得舒不舒服毫无关系。但很多软件有。想想一种文件格式：用的人越多，你能把文件发给的人就越多、能顺利打开你文件的人越多，这个格式对你就越值钱。于是大家都去用最多人用的那个——它又变得更值钱。

> 大白话
>
> 说人话：一部电话是废铁，两部电话能互相打，一亿部电话才叫电话网。软件常常也这样——不是它本身多神，而是"别人都在用"这件事本身，成了它最硬的价值。

网络效应和前面三条会叠在一起：用的人多 → 更值钱 → 更多人用（网络效应），同时更多人用 → 成本更低 → 更能碾压对手（赢家通吃），而且用得越久 → 数据习惯越厚 → 越换不动（锁定）。三股劲拧成一根绳，护城河就是这么一层层堆深的。

> 边际成本近零 → 赢家通吃 → 锁定 → 网络效应。**记住这四条，本书后面所有的故事，都是它们的推论。**谁把这四条用到极致，谁就登顶；谁脚下的这四条被人抽走，谁就塌方。

那为什么一开始，软件是"白送"的？

怪的是，这门后来富得流油的生意，最初根本不被当成生意。

把时钟拨回 1950、60 年代的大型机时代——那是一种一屋子大、贵得只有大公司和政府买得起的巨型计算机。在那个年代，卖计算机的巨头 IBM，卖的是**机器**。软件是什么？是随机器附送的赠品，行话叫捆绑（bundled）——把软件塞进硬件里一起卖，软件不单独标价，等于白送。

为什么会这样？我们用"如果不是这样会怎样"来想一遍。当时一台机器没有配套软件就是一堆废铁——客户买了不会用、跑不起来，机器就卖不出去。所以对 IBM 来说，软件不是要拿去赚钱的商品，而是**为了把昂贵机器卖出去而必须附带的服务**，就像你买车，说明书和售后是白送的——没人会单独跟你收说明书的钱。软件当时就是那本"说明书"。

大白话

说人话：那会儿谁掏钱谁是大爷，机器才是大爷。软件只是让这台大爷机器能干活的附件。免费送软件，是为了多卖几台机器。

现在你回头看这四条怪律，就明白这个"免费赠品"时代有多反常了：一样边际成本近零、天生能赢家通吃、能把用户牢牢锁死的东西，居然被当成赠品白送。**软件这门生意最值钱的部分，在它诞生的头二十年里，压根没被当成生意。**这就像有人守着一座金矿，却只把它当成菜地边上的一块碍事的石头。

那块"碍事的石头"后来怎么被人认出是金矿、又是谁先动手把它从机器上撬下来单独卖——那是下一章的事。你只要先把这四条怪律刻在脑子里：**边际成本近零、赢家通吃、锁定、网络效应。**接下来的每一次登顶，都是有人把这几条用到了极致；接下来的每一次塌方，都是有人脚下的这几条，被别人换了一套玩法抽走。护城河从哪来，又为什么会塌——答案，全在这块地基里。

第二章 · 捆绑与拆分：反垄断如何催生一个产业

一门凭空冒出来的生意

上一章我们说到，1960 年代软件是硬件的**免费赠品**。你买一台 IBM 大型机，机器随附一堆程序，还有工程师帮你上门装好、教你用，全都不另收钱。软件不是商品，它是买机器时附送的糖果。

那时候如果有人跟你说「我想开一家专门写软件卖软件的公司」，你会觉得他疯了。不是因为他技术不行，而是因为——他卖的东西，市面上有一个一模一样、价格是零的替代品。你写个报表程序开价一万美元，客户扭头就说：「IBM 那套系统里自带一个差不多的，白送。」

> 大白话
>
> 说人话：不是没人会写软件，是没人能靠写软件活下去。你的对手不是别的软件公司，你的对手是「免费」。只要免费的东西还在，你这门生意就不成立。

可到了 1970 年代，专门卖软件的公司突然一家家冒了出来，而且活得下去。中间到底发生了什么？答案不是某个天才发明了什么，而是——**一条规则变了**。这一章讲的就是这件事：一门生意能不能存在，有时候不取决于技术，而取决于规则怎么定。

1969 年：IBM 把糖果单独标了价

1969 年 6 月，IBM 宣布了一个决定，行话叫 unbundling（拆分定价）——把原本捆在一起卖的东西拆开，各标各的价。

在这之前，硬件、软件、上门服务、技术培训，是打包成一个总价卖给你的，账单上只有一个数字。bundling（捆绑）就是这个意思：好几样东西塞进一个价格里，你看不出软件到底值多少钱——因为账面上它就是零。

拆分之后，IBM 的账单变了样：机器多少钱一项，某个软件多少钱一项，服务多少钱一项，分开列。这个动作看着像是财务上的小调整，实际上它**第一次给软件贴上了一个不是零的价格**。

> 大白话
>
> 打个比方：以前买车送导航，导航「不要钱」，卖导航的第三方根本没法活。现在车厂宣布导航单独收费五千块——就在这一刻，市面上所有做导航的公司，突然都有活路了。因为「免费」这个对手，消失了。

这就是关键机制。软件一旦有了独立的、非零的价格，客户心里的算盘就变了：既然 IBM 的软件也要单独花钱买，那我为什么不看看别家的？说不定别家的更好、更便宜。**比较，第一次成为可能。**有比较，才有市场。

IBM 为什么要动自己的奶酪

你可能会问：软件白送明明是 IBM 的杀手锏，能把小对手活活饿死，它干嘛主动放弃？

因为有人拿着更大的棒子站在它背后。1969 年 1 月，就在 unbundling 公告的几个月前，美国司法部对 IBM 提起了反垄断诉讼（antitrust suit）——这是政府用法律手段，指控一家公司靠不正当方式垄断市场、扼杀竞争。这个案子的正式名字叫 United States v. IBM（美国政府诉 IBM 案）。

政府的指控里就有一条：你把软件、服务白白塞进硬件价格里，表面是大方，实质是用「免费」这堵墙把独立软件公司挡在门外——它们没法跟「零」竞争。这种「用捆绑消灭竞争」的招数，正是反垄断法盯着的东西。

> 大白话
>
> 说人话：白送不总是好事。当一个巨头用「免费」来让所有小公司都活不下去，「免费」就从福利变成了武器。反垄断法管的，就是这种武器。

IBM 抢在诉讼全面展开前主动拆分，多少有「先自己松手、免得被法院强行掰开」的意思。有意思的是，这场官司一打就是十三年，缠讼到 1982 年，司法部才主动**撤诉**——最后谁也

没判谁赢。但结果已经不重要了：拆分这个动作，1969 年就做出来了，覆水难收。

大白话

记住这个反差：官司最后不了了之，可它在半路上逼出的那一个拆分动作，凭空造出了一整个产业。有时候一场诉讼真正的影响，不在判决书里，而在它逼对手做出的动作里。

市场的门，一开就有人涌进来

门一打开，早就在门外等着的人立刻涌了进来。独立软件商（ISV, independent software vendor）——不造机器、只卖软件的公司——这门生意，到这时才第一次真正立得住。

其实有几家先行者赌得更早。**Applied Data Research（ADR）**做过一个叫 Autoflow 的程序，能自动帮程序员画流程图。他们发现最大的麻烦是：IBM 有个功能相近的东西白送，客户凭什么掏钱？ADR 甚至在 1969 年之前就自己告过 IBM 一状，理由正是「你用免费捆绑逼死我」。还有 **Informatics**，做了一个叫 Mark IV 的数据管理软件，成了历史上第一批卖出百万美元级别的软件产品之一。

这些公司在拆分之前就存在，但活得很挣扎，像在缺氧的房间里喘气。1969 年的拆分，等于突然把窗户打开了：软件有了价格，客户愿意为「更好的软件」单独付钱，独立软件商这门生意才从「勉强喘气」变成「能长大」。

为什么这件事值得单独讲一章

因为它藏着一条贯穿全书的规律：**市场不是天上掉下来的，它常常是被规则划出来的。**同样一堆会写软件的人、同样一批需要软件的客户，规则是「捆绑白送」时，这门生意不存在；规则一改成「拆分定价」，同一批人同一批客户，生意就存在了。变的不是技术，是规则划定的边界。

大白话

一句话记住这章：软件业不是被发明出来的，是被「拆」出来的。真正诞生的那一刻，不在实验室，在一纸定价公告和一场反垄断诉讼里。

这个「制度决定市场边界」的视角，后面还会一次次冒出来，而且往往长着相似的脸：

- 第三章，微软靠握住操作系统这个标准收「过路费」——为什么控制规则比控制产品更值钱；
- 第六章，微软反过来把浏览器**免费捆绑**进 Windows，用当年 IBM 那招绞杀 Netscape——「免费」这把武器又出现了，而这一次微软自己也吃上了反垄断官司。

你会发现历史像在照镜子：1969 年政府用反垄断逼 IBM 松手，造出了软件业；三十年后，同样的反垄断逻辑又追上了那个从软件业里长出来的巨头微软。**捆绑与拆分、免费与定价、垄断与反垄断——这几对力量，会在这本书里反复上演。**

下一章，我们就去看那个把「控制规则」这件事玩到极致的公司：它不造最好的机器，也不写最好的软件，却让整个行业都得向它交税。

第三章 · 谁掌握标准，谁收税

一场谁都没料到结局的胜利

1981 年 8 月，IBM 推出了它的第一台个人电脑，叫 IBM PC。当时的 IBM 是计算机世界的绝对霸主，造大型机赚得盆满钵满。它挟着全世界最响的招牌走进 PC 市场，几乎所有人都以为：这个新赛道，理所当然又是 IBM 的天下。

十几年后回头看，结果反直觉到令人发笑：**造机器的 IBM 输了，而给这台机器提供操作系统的微软、提供芯片的 Intel，成了整个 PC 时代最大的赢家。**

IBM 出钱、出人、出品牌、担风险，把一台划时代的机器造了出来，最后却在自己开创的市场里被挤到角落。两个当年的小配角，反倒躺着数了三十年钱。这一章要讲清楚：这不是运气，而是一条会在全书反复出现的铁律——**谁掌握了别人绕不开的那个接口标准，谁就等于在整条产业链上装了个收费站。**

> 大白话
>
> 一句话：IBM 造了车，微软和 Intel 造了车必须经过的收费站。车卖得再多再便宜，过路费一分不少地落进收费站老板的口袋。

IBM 的两个致命外包

IBM 平时做事极慢：什么都自己造，从螺丝到软件，层层审批。但 PC 这个项目很特殊——公司高层给了一个小团队一年期限，要求快速做出一台能上市的机器。为了赶时间，这个团队做了一个在当时看再自然不过、事后看却改写了行业命运的决定：**关键部件不自己造，直接对外采购。**

其中两个外包，是命门所在：

- **CPU（中央处理器，电脑的运算核心）外包给了 Intel**，用的是 Intel 的 8088 芯片。这颗芯片的指令集（芯片能听懂的一套命令语言，软件必须按这套语言写才能在它上面跑）从此成了 PC 世界的通用语。

- **操作系统外包给了微软**。微软提供了 MS-DOS（磁盘操作系统，负责管理硬件、加载和运行程序的那层底层软件——你按下开机键之后，真正指挥硬件干活的就是它）。

有意思的是，微软当时手里根本没有现成的操作系统。它花了很少的钱，从一家小公司买来一个叫 QDOS 的系统，改一改就交给了 IBM。真正让微软封神的，不是这套软件本身有多厉害，而是它和 IBM 签的那份合同里，藏着一个后来价值千亿美元的条款。

一份合同里的一句话，改写了行业

IBM 谈这份合同时，心思全在硬件上——在它的世界观里，软件不过是让机器能用起来的附属品，硬件才是利润和护城河所在。所以在操作系统的授权条款上，IBM 大方地让了一步：**微软可以把 MS-DOS 授权给别的厂商，而不只是卖给 IBM 一家。**

这句话，当时看毫不起眼。因为 IBM PC 是 IBM 的机器，别家又不能造，微软就算有权卖给别人，又能卖给谁呢？

问题就出在这里——**别家很快就能造了。**

> 大白话
>
> IBM 以为自己守住了硬件这座城，随手把软件授权当添头送了出去。它没想到，城墙很快会被人绕过去，而那张"添头"授权，恰好是绕过之后唯一还收得到钱的东西。

兼容机：城墙是怎么被绕过去的

IBM PC 里绝大多数零件都是市面上买得到的通用件，别的厂商照着买、照着拼，就能攒出一台几乎一样的机器。唯一的拦路虎，是一小段叫 BIOS 的固件（开机时最先运行的一段程序，负责把硬件和操作系统对接起来——相当于机器的"点火装置"）。这段代码是 IBM 自己写的、受版权保护，直接抄要吃官司。

1982 年，Compaq（康柏）想出了一个干净的办法：洁净室逆向工程（一组工程师只研究 BIOS 对外表现出什么行为、把规格写成说明书；另一组从没看过 IBM 代码的工程师，只照着说明书重新写一份功能一样的 BIOS）。因为写代码的人从没接触过原版，产出的东西合法，功能却完全兼容。

这一下，最后一道墙塌了。任何人都能造出"IBM 兼容机"——外面叫 PC compatible，意思是"和 IBM PC 兼容，能跑一样的软件"。市场瞬间涌进大批厂商：Compaq、Dell、后来的联想、以及无数叫不出名字的小工厂。

结果就是硬件行业的血海：

- 大家造的机器几乎一模一样，只能靠**降价**抢客户；
- 利润被一轮轮压到极薄，硬件被打成了白菜价；
- IBM 自己也被卷进这场价格战，最终在 2005 年把 PC 业务卖给了联想，彻底退场。

收费站：为什么造机器的反而输了

现在到了全章最关键的地方。请注意这个红海里一个谁也逃不掉的事实：

> 不管这台兼容机是谁造的、卖得多便宜、厂商赚不赚钱——它**都得装微软的 DOS/Windows，都得用 Intel 的芯片**。因为"兼容"的定义，就是能跑那套软件、认那套指令集。

这就是标准的威力。硬件厂商们拼死拼活、互相压价、九死一生，每卖出一台机器，都要给微软交一份操作系统的钱，给 Intel 交一份芯片的钱。它们出的力最多、担的风险最大、赚的利润最薄；而握着接口标准的两家，对每一台机器——无论谁造的——收一份稳稳的"过路费"。

大白话

造机器的像是在收费站前排队的司机：车多、竞争惨烈、赚辛苦钱。微软和 Intel 是收费站老板：不用造车、不用抢客，只要车流经过就收钱，而且车越多、越便宜、卖得越猛，它们收得越多。

为什么司机绕不开这个收费站？因为软件和指令集有**锁定效应**：用户买 PC 是为了跑上面的软件（表格、文字处理、游戏），这些软件都是照着 DOS/Windows 和 Intel 指令集写的。谁要造一台不兼容的机器，等于让用户放弃手上所有软件——没人会买。于是每个硬件厂商为了活下去，只能乖乖接入这套标准，也就等于亲手把过路费交上去。

这套格局有个流传至今的名字：Wintel（Windows 加 Intel 的合称）。两家公司，一家管操作系统这个软件接口，一家管 CPU 这个硬件接口，联手卡住了整个 PC 产业链最咽喉的两个位置。硬件厂商换了一批又一批，Wintel 的收费站三十年岿然不动。

护城河的第一种形态：标准即收税权

回到 IBM 那两个外包决定。它的错，错在一个根本的世界观：**它以为价值在"东西"里（机器造得好、品牌硬），没看出价值其实在"接口"里（谁定义了别人必须遵守的标准）。**

造机器这件事，出力最多、投入最大、风险最高，看起来像产业链的主角。但只要这件事能被很多人复制，它就会陷入价格战，利润被磨到接近于零。真正稀缺的、别人绕不开又复制不了的，是那个**标准接口**——操作系统的 API、CPU 的指令集。谁攥住它，谁就不必参与惨烈的竞争，只需在竞争者头顶架一个收费站。

把这条规律提炼出来，就是全书要反复用到的第一条深护城河：

> **控制了别人绕不开的标准接口，就等于在整个价值链上装了收费站。别人拼命造东西、互相压价，你对每一件经过的东西收一份过路费。**

这条规律不只属于 1980 年代的 PC。往后你会一次次看到它换装出现：谁定义了浏览器和网页的标准，谁定义了手机应用商店的规则，谁定义了云计算的接口，谁定义了 AI 模型调用的方式——每一次，胜负手都不是谁出力最多、谁的产品最精致，而是**谁握住了那个别人必须穿过、又无法绕开的接口。**

所以，当你下次看到一个行业里"最辛苦的那家反而没赚到钱"，别急着替它惋惜。先问一句：这条产业链的收费站在哪儿，站在收费站里的，又是谁。

第四章 · 开发者才是真正的用户

先做个思想实验。假设你手上有一台崭新的电脑，屏幕漂亮、鼠标灵巧、说明书印得像艺术品——但它上面一个软件都没有。你能用它干什么？答对了，什么都干不了。一台不能装软件的电脑，是一块昂贵的镇纸。

这句废话里藏着本章要讲的整个道理：**决定一个平台成败的，从来不是这台机器本身，而是有多少人愿意为它写软件。**而愿意为它写软件的那群人，就是开发者（写应用程序的程序员）。听上去反直觉——你以为操作系统是卖给用户的，其实操作系统真正要讨好的，是开发者。

操作系统真正的顾客，是写软件的人

我们平时说微软的用户是几亿装了 Windows 的普通人。这话没错，但没说到根子上。对一个平台（别人可以在上面搭软件的地基，比如操作系统）来说，普通用户是"看客"，开发者才是"供货商"。

> 大白话
>
> 打个比方：开一个商场（平台），进来逛街的人（用户）当然重要，但你首先得把店铺（应用）招满。店铺越多越热闹，逛街的人才越多；逛街的人越多，越多店家想进来开店。商场老板真正天天要哄的，是店家，不是顾客。

为什么？因为普通用户买电脑，从来不是为了"操作系统"这四个字。他们是为了用某个具体软件——报税、排版、算账、打游戏。操作系统只是那扇门，门后面有没有东西，才是他们掏钱的理由。

杀手级应用：为了一个软件，去买一整台机器

1983 年，一家叫莲花（Lotus）的公司推出了一款电子表格软件 Lotus 1-2-3（一个能自动算账、拉一下就重算整张表的软件）。它有多重要？重要到很多公司买 IBM PC，唯一的理由就是"我要跑 1-2-3"。机器是顺带买的，软件才是目的。

这就是杀手级应用（killer app，好到让人为了它专门去买某台机器或某个系统的软件）。一个平台一旦拥有一款杀手级应用，就等于有了一块巨大的磁铁：它不再需要费力解释"我这机器多好"，只要说"1-2-3 只在我这儿跑"，客户就来了。

> 大白话
>
> 你可以理解成：不是 PC 带火了 1-2-3，而是 1-2-3 带火了 PC。当年很多财务部门的采购单上写的是"买台能跑 1-2-3 的机器"——机器叫什么牌子，反倒无所谓。

杀手级应用之所以关键，是因为它启动了一台机器，我们叫它**飞轮**。

飞轮：越多应用越好用，越好用越多应用

把两句话连起来看：

- 一个平台上的应用越多，对用户就越有吸引力（想干的事都有软件）；
- 用户越多，就越多开发者愿意为它写应用（客户在哪，我就把软件卖到哪）。

注意这两句话互为因果——应用多带来用户，用户多又带来更多应用。它们首尾相接，转成了一个圈。这个圈一旦转起来，就会自己给自己加速，越转越快，别人越难追上。这就是正反馈飞轮（结果反过来加强原因，于是自我循环、越滚越大的机制）。

> 大白话
>
> 飞轮很重，一开始得使劲推（这就是杀手级应用的作用）；可一旦转起来，它自己的惯性就替你推了。平台竞争的残酷就在这里：领先者不是领先一点，而是领先一个"越跑越快"的加速度。

这个机制有个学名叫双边网络效应（同一个平台上，用户和开发者两群人互相吸引、互相壮大）。名字唬人，其实就是上面那个圈。记住那个圈就够了。

那台更先进的机器，为什么输了

现在到了本章最反直觉的地方。

1984 年，苹果发布了 Macintosh（麦金塔，第一台面向大众的图形界面电脑）。它有窗口、有图标、有鼠标——你今天用电脑的方式，基本是它定的。同一时期的 PC 呢？还停留在黑底白字、敲命令的 DOS 界面。单论技术和好用，Mac 领先了整整一个时代。

可十年之后，赢家是 Windows，尤其是 1990 年代的 Windows 3.x 和 1995 年那记引爆一切的 Windows 95（微软把图形界面做进大众市场、彻底铺开的版本）。到 90 年代末，个人电脑里九成以上装的是 Windows，Mac 缩在一个小角落里。

一个技术明显更差的系统，赢了一个技术明显更好的系统。为什么？

不是因为 Windows 更好用——它当时确实更难用。是因为 Windows 继承了一整个已经转起来的飞轮：

- **它站在 DOS 的软件库上。**Windows 能兼容此前 DOS 时代积累的海量软件（1-2-3、各种行业程序都在这条线上）。开发者不用从零开始，用户升级也不用扔掉老软件。飞轮不是新推的，是接着转的。
- **它的装机量碾压 Mac。**因为第三章讲过的原因——DOS/Windows 授权给一大堆厂商造兼容机，机器便宜、到处都是。开发者写软件图什么？图卖得出去。客户都在 Windows 这边，我当然优先写 Windows 版。
- **苹果偏偏反着来。**软硬件封闭、只有自己一家造、机器又贵，装机量自然小。对开发者来说，为一个用户少的平台写软件，投入产出比太差，优先级只能往后排。

> 大白话
>
> 用商场的比方收个尾：Mac 是一座装修精美但刚开业、店铺稀稀拉拉的新商场；Windows 是一个又土又挤、但摊位早已排满、人山人海的老集市。你是店家（开发者），会去哪儿开店？答案不言而喻。技术再好，也顶不住"人气已经在别处了"。

这里要说句公道话：本章讲的是 80、90 年代那场桌面之争里苹果的失利，不是说苹果"一直输"。它后来靠另一套逻辑漂亮地翻身——那是后面章节的故事。此刻你只需记住这一战的教训。

把这条规律记进脑子

平台竞争里，人们本能地问："哪个技术更优？"但一次次的历史给出的答案是：这常常不是决定因素。真正决定胜负的是另一个问题——

> 哪个飞轮先转起来、转得更快？

谁先攒够应用和开发者，谁的圈就先转、就越转越快，后来者哪怕拿着更好的机器，也只能眼睁睁看着差距被"加速度"越拉越大。所以聪明的平台公司，第一件事永远是**讨好开发者**：把开发工具做顺手、把接口开放、把用户规模亮出来，让开发者相信"来我这儿写，卖得掉"。

"开发者才是真正的用户"——这句话不只属于 Windows 和 Mac。往后你会看到它一次次重演：浏览器要争前端开发者，手机要争 App 开发者，到了 AI 时代，模型要争的还是在它上面搭应用的开发者。载体年年在换，那个越转越快的圈，一直没变。

第五章 · 桌面软件大战：护城河也会一夜蒸发

把时钟拨回 1990 年。你走进任何一间美国办公室，屏幕上跑的桌面软件，几乎没有一个是微软的。要算账，用 Lotus 1-2-3（当时最强的电子表格软件，拉一下就重算整张表）；要写文件，用 WordPerfect（律师事务所人手一份的字处理软件，处理长文档、脚注、格式最靠谱）；要管数据、编程序、性价比至上的，找 Borland（一家以便宜好用著称的软件公司，旗下有数据库 dBASE、表格 Quattro Pro、编程工具 Turbo Pascal）。

这三家，每一家都是自己那个品类里公认的第一名。产品做到了同类最好，用户忠诚，市场份额领先。按常理，它们该稳坐十年、二十年。

可到 1995 年前后，它们几乎同时崩塌。Lotus 被 IBM 收购，1-2-3 慢慢消失；WordPerfect 被 Novell 买下、又贱价卖给 Corel；Borland 一路衰落，最后转型做别的去了。取代它们的，是同一家公司的同一套东西——微软 Office（把 Word、Excel、PowerPoint 打包一起卖的办公套件）。

本章要回答一个让人难受的问题：**为什么各个品类里最好的产品，会在同一段时间里一起死？** 不是它们各自犯了各自的错，而是它们撞上了同一套机器。这套机器有三个齿轮：平台更替、捆绑、分销。

第一个齿轮：换平台等于重新发牌

1980 年代的桌面软件，跑在 DOS（黑底白字、靠敲命令操作的老系统）上。到了 90 年代初，微软把图形界面的 Windows（有窗口、图标、鼠标的系统）铺开，一切要搬到新地基上。

关键在于：从 DOS 搬到 Windows，不是升级，是**重写**。DOS 程序和 Windows 程序，画面怎么画、鼠标怎么响应、菜单怎么弹，底层逻辑完全两码事。等于让一家做惯了收音机的工厂，一夜之间改产电视——不是调调产线，是整条重来。

大白话

这一步之所以致命，是因为它把所有人拉回同一条起跑线。你在 DOS 时代攒下的领先、口碑、代码，到了 Windows 上大半清零。市场被"重新发牌"，手里那副老好牌突然不算数了。

而在位者天生转身慢，原因很实在：老产品还在大把赚钱，用户还在用，你舍不得把精兵强将全抽去做一个"还没人用的新版本"。于是决策一犹豫，就误了船期。

WordPerfect 是最惨痛的例子。它的 DOS 版是标配，公司却在两件事上都押错了宝：先赌 IBM 的另一套系统会赢（结果没赢），又低估了 Windows；等回过神来做 Windows 版，产品迟迟难产、上市又晚又不好用。而这段时间里，微软的 Word 早就为 Windows 优化好了，抢先占住了新平台。等 WordPerfect 的 Windows 版终于像样，用户已经走光了。

Lotus 1-2-3 也一样。它在 DOS 上是霸主，但 Windows 版慢了一拍，而微软的 Excel 从一开始就是照着图形界面设计的——鼠标、下拉菜单、所见即所得，用起来就是顺。换平台这一步，冠军和挑战者被拉平，谁先站稳新地基，谁就赢。

第二个齿轮：单打冠军打不过一整套

就算你在新平台上也做出了好产品，还有第二关。

这三家有个共同点：都是**单点冠军**。Lotus 只强在表格，WordPerfect 只强在文字，各管一摊，各自为战。而微软的打法是把 Word、Excel、PowerPoint 三个装进一个盒子，起名 Office，捆绑（几个产品打包一起卖，不拆开）着卖。

捆绑的厉害之处有三层：

- **打包价更划算。**分开买三个单品冠军，比买一整套 Office 还贵。企业采购一算账，当然买套装——哪怕其中某一个不是全场最好，"够用又便宜一大截"就赢了。
- **套件内部能互相打通。**Excel 的表格一拖就进 Word，Word 的图一贴就进 PowerPoint，因为它们出自一家、生来就为配合。而你把 1-2-3 的表格塞进 WordPerfect，是两家不同公司的产品在硬凑，处处别扭。
- **操作是统一的。**Word 里学会的快捷键、菜单摆法，在 Excel 里照样管用。学一个等于学会一套。而混用不同厂家的软件，等于每换一个就得重新适应一套脾气。

大白话

换个比方：你去餐厅，一家端上来的是「前菜+主菜+甜点」搭配好、口味连贯、还打了套餐价的套餐；另一桌是三家不同馆子各送一道招牌菜，单看每道也许更精，但拼在一起既贵又不搭。多数人会点套餐。企业采购几万个座位，更是如此。

所以「我这一样做到全世界最好」这句话，在套件面前失去了分量。用户买的不是单项冠军，是「一整套能顺畅协作的活儿」。单点最优，挡不住整合。

第三个齿轮：又当裁判又当运动员

第三个齿轮最狠，也最有争议：微软既卖操作系统，又卖跑在这个系统上的应用。这叫掌握分销（把产品送到用户手里的渠道和路径）。

它至少带来两样别人没有的优势：

- **预装渠道。**微软控制着 Windows 授权给各家电脑厂商（OEM，贴牌造机器的厂商）的条件。新电脑出厂时装什么、搭不搭 Office，微软在谈判桌上说话最有分量。软件还没开卖，就已经躺在千百万台新机器里了。对手要一份份去卖，微软是随机器白送到你面前。
- **更懂自家的系统。**Windows 的各种接口（应用程序调用系统功能的入口和规则，英文叫 API）是微软自己定的。做应用的团队和做系统的团队是一家人，理论上能更早知道接口怎么变、更懂怎么榨干性能。当年就有很大争议：微软的应用是不是靠"提前知道内幕"占了便宜？无论内幕程度到底多深，光是"应用团队离系统团队最近"这一条，就已经是别家给不了的先手。

大白话

这就像一场足球赛，对手不但是场上最有钱的球队，还兼任了裁判和场地方。他不一定要明着吹黑哨，光是「主场、定规则、提前知道天气」，你就已经很难赢了。

三个齿轮咬在一起

现在把三个齿轮合起来看，就明白为什么各品类冠军会一起死了——它们不是各自倒下的，是被同一台机器一次碾过。

> 一次平台更替把大家拉回起跑线，一次捆绑让单品冠军的优势失效，一手分销让对手在渠道上根本挤不进来。

任意一个齿轮单独出现，冠军也许还能扛。DOS 到 Windows，如果只是换平台，反应快的公司能追上；如果只是被捆绑，产品够强或许还能守住高端;如果只是渠道吃亏，靠口碑也能卖。可这三件事叠在一起，同时发生，而且都由**同一个对手**发动——它控着平台，所以掌握换代节奏；它能打包，所以让你的单项优势贬值；它管着分销，所以你连货都铺不出去。三重挤压之下，最好的产品也没有活路。

这就是本章要你记住的那句反直觉的话：

> 在单一功能上做到最优，挡不住一个既控平台、又能打包、又掌分销的对手。

护城河看着很深——Lotus 有市占，WordPerfect 有律所的死忠，Borland 有性价比口碑。可这些护城河都只挖在"产品好不好用"这一层。而对手不在这一层跟你打，它在下面动地基（换平台）、在旁边改玩法（捆绑）、在门口截渠道（分销）。你挖得再深的河，一次平台更替加一次捆绑，就蒸发了。

所以「最好的产品」和「活下来的公司」，从来不是一回事。桌面软件这一仗把这个道理写得血淋淋。而这套逻辑没有随着 90 年代结束——往后你会看到它换上新载体一次次重演：浏览器、手机、AI。控平台、能打包、掌分销，这三件事一旦落到同一个对手手里，单点冠军就该睡不着觉了。

第六章 · 浏览器战争与「拥抱、扩展、消灭」

1995 年 8 月 9 日，一家成立还不到 16 个月、几乎没赚过什么钱的小公司上市了。它叫网景。开盘价定在 28 美元，第一天冲到 75 美元。华尔街第一次意识到：互联网可以是一门生意。那一天，很多人把它当成互联网时代的起跑枪。

可短短四年后，这家点燃了整场泡沫的公司，市占率崩塌，团队解散，被人以「白菜价」买走。杀死它的不是一个更好的竞争产品，而是一个更简单的动作——上游把它的产品，变成了操作系统里免费附送的一个零件。

这一章讲的就是这场「第一次浏览器战争」（1995–1999）。它是本书里最干净的一个案例，说明一条冷酷的规律：**当你的整个生意，是上游平台可以顺手免费送出的一个功能时，你几乎必死。**

浏览器：让普通人第一次点开网页

Netscape Navigator（网景导航者，1994 年底发布的图形网页浏览器）解决了一件今天看来理所当然、当年却很魔法的事：让不懂技术的人，也能用鼠标点开一个网页。

在它之前，互联网早就存在，但那是工程师的领地——你得敲命令、记地址、懂协议。Navigator 把这一切藏了起来：一个窗口，一个地址栏，蓝色带下划线的字点一下就跳转。它还有个当年很关键的贴心设计：网速慢的时候，文字先出来、图片慢慢加载，你不用干等。对第一次上网的人来说，这就是「互联网」本身的样子。

结果几乎是瞬间的统治。Navigator 免费给个人用户下载试用，靠向企业和开发者卖授权、卖服务器软件赚钱。到 1995 年，它占了浏览器市场的绝大多数份额——有的统计说八九成。一个刚出生的产品，就成了一整个新世界的入口。

大白话

说人话：Navigator 干的事，相当于给一片刚通电、却只有电工敢碰的城市，装上了家家户户都会用的电灯开关。开关一普及，用电的人就爆炸式增长——而网景，就是那个开关。

为什么微软会害怕一个浏览器?

如果网景只是「一个好用的上网软件」，微软大可以不理它。真正让微软坐立不安的，是网景创始人们放出的一句话——大意是：将来浏览器会让底层的 Windows 沦为「一堆不太重要的驱动程序」。

这句话背后是一个可能改天换地的设想：**浏览器可能变成新的平台。**你回想第三、第四章的逻辑——微软最深的护城河，是全世界的软件都为 Windows 而写，所以大家离不开 Windows。可如果软件不再写给 Windows，而是写成「网页」，在浏览器里跑呢?

那么用户底下装的是 Windows、Mac 还是别的什么，就不重要了——反正应用都在浏览器里。浏览器成了所有软件真正落脚的地方，而 Windows 退化成一个「负责开机、把浏览器启动起来」的底座。控制权，就从操作系统转移到了浏览器。

> 大白话
>
> 打个比方：Windows 是一栋盖满了商铺的大楼，商铺（软件）只能开在这栋楼里，所以人人都得进这栋楼。网景说：以后商铺都搬到「网页」这条街上，谁家的楼都能通向这条街——那大家还非得进你这栋楼干嘛？楼还在，但楼不再是必经之路了。对靠「收过路费」为生的微软，这是要命的。

所以这场仗从一开始就不只是「两个浏览器谁好用」。它是一场关于「未来的软件到底跑在谁的地盘上」的争夺。微软赌上的是它最核心的资产。

第一招：把浏览器变成免费的、操作系统自带的零件

微软的反击，第一步简单得近乎粗暴：做一个自己的浏览器 Internet Explorer（IE，互联网探索者），然后**免费、白送，并且直接捆进 Windows 里**——新买的电脑一开机，IE 就已经在那儿了。

这一招为什么致命，值得慢慢拆。网景是靠浏览器相关的生意赚钱的。而微软的做法，等于把网景赖以为生的那个东西，变成了 Windows 的一个「顺手附送的功能」。这里的杀伤力来自三层叠加：

- **价格被打到零。**你卖一样东西，对手把同类东西白送，你就没法定价了。更狠的是，微软不靠 IE 赚钱——它靠 Windows 赚钱，IE 亏多少都无所谓。这不是公平竞争，是一个不需要这块利润的巨人，来抢一个只有这块利润的小公司的饭碗。
- **分发被彻底垄断。**网景要用户「知道它、去下载它、装上它」；IE 则是电脑到手就在。对绝大多数普通人来说，「默认已经有的」永远赢过「需要自己去找的」。上游平台自带的分发渠道，是外人花多少钱都买不到的。
- **还被绑上了操作系统。**微软后来干脆宣称 IE 是 Windows「不可分割的一部分」——你没法只卸载浏览器而留下系统。这样一来，「换掉 IE」在技术上、心理上都变成了一件麻烦事。

大白话

说人话：想象你开了家卖矿泉水的公司，活得还不错。突然自来水公司宣布——从今往后，每家水龙头拧开就直接流出免费的瓶装水，而且水龙头拔不下来。你的水可能更好喝，但你还怎么卖？你不是被打败了，是被「白送」这件事本身抽干了呼吸。

第二招：拥抱、扩展、消灭

光免费还不够，因为已经在用 Navigator 的人未必愿意换。微软的第二招更精巧，后来被总结成一个专有名词——

Embrace, Extend, Extinguish（拥抱、扩展、消灭）：一套让对手「自己走进死角」的三步棋。它对付的不是网景这家公司，而是网景赖以生存的开放标准。

1. **拥抱：**先老老实实支持大家公认的网页标准，让 IE 显得「和别人一样兼容、一样规矩」。这一步是为了取得信任、拿到入场券——你不能一上来就搞特殊，那没人理你。
2. **扩展：**等 IE 装机量上来了，微软开始往里加只有 IE 才认、别的浏览器不认的私有功能和写法。对做网站的人来说，用这些功能能做出更炫、更方便的效果——诱惑很实在。
3. **消灭：**越来越多网站用了这些「只有 IE 认」的写法。于是用别的浏览器打开，页面就乱、就缺功能。网站被迫在角落贴上「本站最佳浏览器为 IE」。到这一步，标准已经悄悄从「大家共有的」变成「微软的」——不用 IE 的人被一点点挤出了这个网。

大白话

打个比方：一群人本来说同一种通用语，谁都能交流。微软先学会这门通用语（拥抱），再往里掺进一堆只有自己人懂的黑话，还让这些黑话变得又时髦又好用（扩展），最后大家发现不学黑话就听不懂重要的话——于是通用语名存实亡，语言的主导权落到了微软手里（消灭）。表面上它一直「支持标准」，实际上它偷走了标准。

这一招的阴险之处在于：每一步单看都不违法、甚至像是在「让产品更强」。但三步连起来，效果是把一条本该属于所有人的公共道路，慢慢改造成了一条只有自家车能走顺的私人车道。

塌方：从八九成到几乎归零

两招合力之下，网景的崩塌很快。免费捆绑抽干了它的收入，私有扩展侵蚀了它的生存土壤。IE 的市占率一路爬升，Navigator 一路下滑。到 1998 年，撑不下去的网景被 AOL（美国在线，当年最大的上网服务商）收购。第一次浏览器战争，实际上结束了——微软赢了，而且赢得彻底，IE 后来一度占据九成以上市场。

但网景在临终前做了一件影响深远的事：它把浏览器的源代码**开源**了（把软件的「配方」公开，任何人都能拿去看、去改、去接着做）。这段代码后来长成了 Mozilla，再后来结出了 Firefox。这条线我们按下不表——它是好几年后另一场故事的种子，这里只留一个伏笔：**被杀死的东西，有时会以开源的形式活过来。**

制度出手：United States v. Microsoft

如果故事到这里就结束，那结论会非常灰暗：上游平台想灭掉谁，顺手一个「免费捆绑」就够了，独立公司毫无还手之力。但这一次，还有第二只手介入了——监管。

United States v. Microsoft（美国政府诉微软案）是一场里程碑式的反垄断诉讼。政府 1998 年正式起诉，核心指控正是：微软把浏览器和 Windows 捆绑，是在利用操作系统的垄断地位，去压死浏览器市场的竞争。

请回到第二章那条规律——**制度决定市场的边界**。市场不是一片纯粹的丛林，游戏规则是人定的。反垄断法在这里问的问题是：一个已经在某个领域近乎垄断的公司，能不能用它的垄

断地位去碾平隔壁另一个市场的竞争？

官司的走向几经反复：

- **2000 年一审判决**认定微软违反了反垄断法，一度要求把公司拆分成两块——一块管操作系统，一块管其他软件，让操作系统这门生意不能再被拿来当武器。
- **2001 年**，经过上诉，最终以**和解**收场——微软没有被拆分，但接受了一系列行为约束，比如不得再用某些手段惩罚安装竞品的电脑厂商。

对网景本身，这来得太晚，救不回来了。但它改变了往后的空气。它给业界立了一条隐形的红线：平台巨头可以竞争，但用「捆绑 + 独家绑定」去系统性绞杀隔壁的独立公司，是要付出代价的。这条红线，为后来的一批公司（包括那些跳到 Web 上做生意的人）留下了一线可以呼吸的缝隙。

大白话

说人话：市场这场比赛，裁判是真的存在的。裁判管不了你比对手强，但能管你「趁自己是主场庄家，顺手把客队的水源掐了」。网景没等到裁判吹哨，但哨声本身，让下一批球员敢上场了。

这一章的机制落点

把这场战争抽干到只剩骨架，它讲的是一件事：**你的位置，比你的产品好不好更决定生死。**

网景的产品是划时代的，团队是顶尖的，眼光甚至看对了未来——软件真的越来越多地跑进了浏览器和网页里。可它站错了位置：它把整个身家，压在了「一个上游平台可以免费附送的功能」上。只要微软愿意，随时能把这个功能变成 Windows 自带的零件。这不是执行力问题，是结构问题。

于是这一章的规律可以写成一句话：

> 当你的整个生意，恰好是上游平台可以顺手免费送出的一个功能时，你几乎必死——除非监管出手拦住那只手，或者你能跳到平台够不着的那一层去。

「监管出手」，我们刚讲过；「跳到平台够不着的层」，是另一条活路——别只做一个「功能」，去做平台本身够不到的东西：一张更大的网络、一套别人离不开的协议、一个上游没法轻易复制的位置。下一章的 Sun 与 Java，恰恰想赌的就是这条路：不做浏览器这种会被吸收掉的功能，而去争夺更底层的「协议之王」。他们赌对了趋势——却没赌对怎么把趋势变成钱。那是另一种塌方。

第七章 · 网络效应与协议之王

看得见的界面，看不见的管道

前面几章的战场都在屏幕上：谁的操作系统更好用，谁的浏览器跑得更快，谁的窗口和菜单更顺手。用户能看见、能点、能骂。可当互联网真正铺开，故事换了地方——真正值钱的东西，普通人一辈子都没听说过它的名字。

你每天上网，数据在世界各地穿来穿去。它先被打包成一个个小包裹，经过一台台设备转发，最后拼回你手机上的一张图、一段视频。这一路上，有一家公司的设备几乎是必经之路；你打开的很多网站，背后跑的程序用的是另一家公司的语言、存的数据锁在第三家公司的仓库里。你看不见它们，可它们对经过的每一份"流量"、每一行"业务"，都在悄悄收钱。

> 大白话
>
> 一句话：桌面时代的护城河修在屏幕上（界面好不好用），互联网时代的护城河沉到了地底下（谁是大家默认都要用的管道、语言、仓库）。用户看不见的地方，才是真正装收费站的地方。

这一章讲一个反直觉的落点：**互联网时代的赢家，不靠让用户"喜欢"，而靠让所有人"绕不开"。**谁成了大家默认都用的那套协议、运行时、接口，谁就坐上了收费站——哪怕用户根本不知道它存在。而本章的悲剧主角 Sun，恰恰赌对了这个大方向，却站错了收费站的位置。

Sun：那句对了三十年的口号

Sun Microsystems（一家做工作站和服务器的公司，那种给企业和实验室用的高性能电脑）有一句响彻整个 90 年代的口号：**"The Network Is The Computer / 网络就是计算机"**。

这句话今天听起来像废话——我们的照片、聊天、文档不都在"云"里吗？可 Sun 在 80 年代末就把它喊了出来。那时候大多数人对电脑的理解还是"桌子上那个孤零零的箱子"，Sun 却

说：真正的计算不发生在某一台机器里，而发生在机器与机器连成的网络上。**这个方向，它比几乎所有人都看得早、看得准。**

看对了方向，Sun 也确实吃到了第一波红利。互联网泡沫最疯的那几年，全世界的网站、电商、门户都要买服务器来撑，而 Sun 的服务器又快又稳，成了搭建网站的"标配"。有句话在当时流传：Sun 是"点 com 背后的点 com"——别人做网站，Sun 卖铲子。钱像潮水一样涌进来。

Java：一门造福了世界、却救不了自己的语言

Sun 还干了一件真正改变世界的事：1995 年，它推出了 Java（一种编程语言，同时是一套"运行时"——即一层垫在程序和操作系统之间的软件）。

Java 的卖点是一句极有煽动力的话：**"一次编写，到处运行"**。意思是，同一个程序，写一遍，就能在 Windows、Mac、各种服务器上都跑起来，不用为每种系统改一遍。这在当时是天大的解放。

> 大白话
>
> 打个比方：以前你写的程序像一道只能在某一口特定灶台上做的菜，换个灶台就得重做。Java 相当于自带了一口标准锅——你把这口锅（运行时）搬到哪个灶台上，同一道菜都能照做。程序不再挑操作系统了。

这一招正好戳中互联网时代最痛的地方：网络上什么机器都有，你没法假设对方用的是哪种系统。于是 Java 迅速成了企业后端（银行、电商、政府系统那些看不见的服务器程序）的地基，后来又成了 Android 手机应用的主要语言。用它的人越多，教程、工具、招得到的程序员就越多；程序员越多，新项目就越倾向于继续用它——这就是**网络效应**滚起来了：一样东西用的人越多，它对每个人就越有用，于是更多人用。Java 的网络效应大到什么程度？直到今天，全世界最流行的编程语言里都有它一席之地。

问题来了，也是这一章最扎心的地方：**Sun 造出了一门统治世界的语言，公司却因此倒了。为什么？**

赌对了趋势，却赌错了怎么收钱

答案藏在一个致命的错位里：**Java 基本是免费的。**Sun 把它几乎白送给全世界，想的是——语言免费，用的人多了，大家自然会来买我又贵又好的 Sun 服务器来跑这些 Java 程序。语言是鱼饵，硬件才是鱼钩。

这个逻辑，在互联网蒸蒸日上、Sun 硬件卖到脱销的年头，看起来天衣无缝。可它把公司的命，全押在了"贵硬件卖得动"这一条腿上。

然后，这条腿被三件事同时打断了：

- **互联网泡沫破裂（2000—2001）**：无数烧钱的网站公司死掉，它们买服务器的订单凭空蒸发。Sun 最赚钱的客户群，一夜之间没了半壁。
- **便宜的 x86 服务器崛起**：过去买服务器认准 Sun 这种"高级货"，现在用一堆普通的、和 PC 同一套芯片（x86）的廉价机器攒起来，也能扛住网站的活儿，价格却是零头。
- **Linux 崛起**：一套免费的操作系统（下一章的主角）让这些廉价机器如虎添翼，进一步抹平了 Sun 贵硬件"跑得更好"的优势。

> 大白话
>
> Sun 的商业模式像一家餐厅：菜谱（Java）免费公开，靠卖自家昂贵的厨具（服务器）赚钱。可当外面出现了便宜好用的通用厨具，大家拿着你免费送的菜谱，跑到别处去做菜了。你越成功地推广菜谱，越是在给对手送客人。

这就是全章的核心反直觉：**拥有一门最流行的编程语言，救不了一家公司——因为流行的是语言，收钱的却是硬件，而这两者之间，没有一道别人绕不开的收费站。**任何人都能白拿 Java，然后买别家的便宜机器去跑。Sun 修的护城河，修在了收不到钱的位置上。2009 年，Sun 宣布被数据库巨头 Oracle 收购，2010 年完成。一家赌对了整个互联网方向、造出了影响世界的技术的公司，就这样谢幕了。

对照组一：Cisco，站在管道接头上

同一个互联网时代，有人把收费站修对了地方。

Cisco（思科，做路由器和交换机的公司）卖的东西，正是让数据在网络里穿行的那些设备——它们是互联网的"管道接头"。你发的每一个数据包，几乎都要经过这类设备一次次转发，才能到达目的地。

Sun 卖的是"网站放在哪台机器上"，这件事有大把替代品；Cisco 占的是"数据从哪儿流过去"，而当年几乎所有网络设备都得和它的设备对接、遵循它主导的那套接法。**它占住的是基础设施里那个别人绕不开的位置。**互联网每扩张一寸，就要多铺一段管道、多接一批接头，Cisco 的生意就随之涨一分。泡沫最高峰时，Cisco 一度成为全世界市值最高的公司之一——一家你几乎从没直接买过它产品的公司。

对照组二：Oracle，数据搬家难于登天

另一种收费站，靠的不是流量，而是**让你走不了**。

Oracle（甲骨文，企业数据库巨头。数据库就是企业存放核心数据的那个"仓库"——客户、订单、账目全在里面）几十年稳坐王座，靠的是一个朴素得可怕的机制：切换成本（想换掉它要付出的代价）。

一家银行、一家航空公司，把全部核心业务数据装进 Oracle 数据库之后，几十套系统、无数程序都是围着它写的。这时候想换成别家的数据库，等于给一栋住满人、通着水电、还在营业的大楼搬地基——迁移过程稍有闪失，账目错乱、业务停摆，谁担得起？**于是绝大多数企业宁可年年交着昂贵的授权费，也不敢动它。**

> 大白话
>
> Oracle 的护城河不是"我的仓库最好用"，而是"你的东西一旦搬进来，就搬不走了"。数据搬家难，比产品好用更能锁住客户。

Cisco 靠"绕不开的位置"，Oracle 靠"离不开的锁定"，它们都把护城河修在了能持续收到钱的地方。而 Sun 手握最先进的方向、最流行的语言，偏偏两样都没占住——它证明了技术上的胜利，可以和商业上的失败同时发生。

这一章的规律：押对方向，不等于押对商业模式

把这三家放在一起，一条清晰的规律浮出来。互联网时代，护城河从"桌面上的界面"沉到了"看不见的基础设施与协议"。谁成了大家默认都要用的那套东西——数据必经的管道、程序依赖的运行时、业务锁死的仓库——谁就坐在了收费站上。

但坐在收费站上，还得看你收的是"过路费"还是"入场券"，也就是**那个绕不开的位置，你能不能收到钱**。Java 是全世界都绕不开的运行时，可它免费，Sun 收不到过路费；真正要收钱的硬件，又偏偏是能被人绕开的。位置对了，收费口却错了。

> **押对技术方向 ≠ 押对商业模式。护城河必须同时满足两个条件：别人绕不开，而且你能收到钱。少了后一半，你可能造出改变世界的技术，却依然救不了自己的公司。**

Sun 的墓志铭，或许可以这样写：它看见了未来，为未来造好了地基，然后眼睁睁看着别人在这块地基上盖楼、收租。下一章，我们会看到那套"免费还能协作改进"的开源力量——正是它，把 Sun 这样卖贵硬件、卖授权的老巨头，一个接一个地掀翻在地。

第八章 · 开源如何吃掉商业软件

第一章埋下过一句话：软件的边际成本近乎为零——写完第一份要几百万，复制第二份几乎不要钱。前面几章我们看它怎么造出护城河：谁掌握标准谁收税，谁的用户多谁锁定人。这一章，我们把同一句话推到最狠的地方，问一个近乎耍赖的问题：

> 如果复制一份软件几乎不要钱，那么让一群人免费一起写、写完白送出去，会发生什么？

答案是一门叫开源软件 open source的东西——源代码公开，任何人都能免费拿去用、随便改、再送给别人。它没有正面攻破任何一家巨头的护城河。它做的事更狠：把护城河底下的水，悄悄抽干了。

先算一笔巨头没法算的账

七十到九十年代，服务器上跑的是Unix（一整套付费的企业级操作系统，Sun 的 Solaris、IBM 的 AIX、惠普的 HP-UX 各占一块）。它们贵，而且互相不兼容——你买了 Sun 的机器和系统，软件、人、流程都长在上面，想换厂商，等于重装整个公司。贵加锁定，这是标准的好生意。

1991 年，芬兰大学生 Linus Torvalds 在网上贴了个业余项目：他自己写的一个类 Unix 的操作系统内核，取名Linux（一个免费、开源、能当服务器操作系统用的类 Unix 系统）。配上早已存在的一整套免费编程工具（GNU，一个"重写一套自由版 Unix 工具"的运动，编译器、命令行这些都齐了），一个不要钱、还能自己改的完整系统就凑齐了。

一开始它很粗糙。但它有一个巨头永远给不出的属性：**免费，且开放。** 全世界的工程师顺手把自己修的 bug、加的功能提交回来，它一年一个样地变好。于是巨头的销售面前出现了一道无解的算术题。

大白话

你卖 Solaris，一套几万美元。对面摆着一个"够用、免费、还能自己改"的 Linux。你想打价格战？你能降到一万、五千、一千。但你降不到零——你养着一支要发工资的工程师队伍，零元卖你就死了。而对手的价格，从第一天起就是零，而且它不靠卖它活着。**你没法对 0 元报价。**

这就是全章的机制核心，值得记死：**当你的护城河是"这份代码只有我有、你得付钱买授权"，一旦有人把等价的东西免费做出来、还开放让大家继续改进，你的护城河不是被对手攻破的，是被抽干的。** 攻破要靠更强的产品，抽干只需要一个"够好且免费"的替代品长期存在。

Apache：半个互联网悄悄换了地基

同样的事，在互联网的另一层同时发生。网站要跑起来，得有一个"网页服务器"软件——用户在浏览器敲网址，它负责把网页递过去。九十年代中，这本来也可能长成一门卖授权的生意。

1995 年，一群人把一个开源网页服务器攒了出来，叫Apache（免费开源的网页服务器软件，长期占据互联网上大半网站）。它不要钱，谁都能装、能改、能塞进自己的产品里。结果，互联网基础设施这一层，从一开始就没给"卖网页服务器授权"留出生长空间——大半的网站默默跑在一个没人从中收授权费的软件上。

Apache 的意义在于示范了一件事：**开源不只在操作系统这种老地盘上蚕食巨头，它还会抢先占住新出现的地基**，让某些生意根本来不及诞生。等你想在这块地上盖收费站，地上已经站满了不收费的东西。

那开源公司靠什么活？Red Hat 的答案

到这里你可能会想：白送不就等于没生意吗？——这是最常见的误解，得把它讲透。开源抽掉的是"卖比特（卖代码副本）"这门生意，不是抽掉所有生意。真正活下来的公司，只是把收费的位置挪了个地方。

Red Hat（一家靠"把 Linux 做成企业敢用的发行版、再卖服务"起家的公司）把这条路走通了。它的逻辑是这样的：

- 软件本身照样免费、开源，谁想白拿都行；
- 但企业不敢拿一堆散装开源代码去跑银行的核心系统——他们要的是**一个测试过、能长期升级、出了事有人接电话负责的稳定版本**；
- Red Hat 卖的就是这份"省心和兜底"：技术支持、安全更新、责任担保。软件是零元的，信任是收费的。

> 大白话
>
> 打个比方：水是免费的，但装瓶、检测、送到你家、坏了负责，这些你愿意付钱。Red Hat 卖的不是水，是"你敢喝这口水"这件事。

这门生意成不成？2019 年，IBM 用约 **340 亿美元**把 Red Hat 买了下来。一家"把软件白送"的公司，值一家老牌巨头砸下几百亿。这就是本章要你记住的下半句：**护城河被抽干之后，活下来的人把钱从"卖比特"挪到了"卖服务与信任"。** 卖授权的生意死了，卖确定性的生意活了。

守不住的样本：Novell 为什么整个消失了

反面的机制，Novell 讲得最干净。

Novell（曾靠一款叫 NetWare 的网络操作系统称霸企业局域网的公司）在八九十年代是绝对的王者。那时候公司里几台电脑要连成一张网、共享打印机和文件，几乎离不开它的 NetWare——而这门生意的核心，正是一条一条卖授权。

然后地基被抽走了，从两头同时抽。一头，微软把网络能力直接做进了 Windows，你买系统就送；另一头，免费的 Linux 也能干同样的活，还能自己改。Novell 卖的那份"联网能力"，突然到处都是、还不要钱。

它的护城河不是被某个更强的对手一刀砍断的。是它赖以收费的那件事——"想联网你得买我的授权"——变成了免费的常识。**当你收费的理由被免费送掉，你的产品可以还很好，但客户已经找不到付钱的理由了。**

Novell 也挣扎过，而且方向看着挺聪明：它去买了 SUSE（一个 Linux 发行版），想跟 Red Hat 一样卖服务；它更早还买过王牌文字处理软件 WordPerfect，想在桌面上抢地盘。但这

些都没救回它——收购换不回一条已经被抽干的护城河。2011 年，Novell 被 Attachmate 收购，一个称霸过整个企业局域网的名字，就此散场。

对比看得很清楚：Red Hat 和 Novell 面对的是同一股力量，一个顺着它把生意重建在服务上，一个想守住旧的卖授权模式，结局天差地别。

Sun 的 Solaris：同一条河，又干了一次

第七章讲过 Sun 赌对了"网络就是计算机"的趋势，却没赌对商业模式。这里补上另一半：Sun 赖以卖钱的 Solaris，正是被免费 Linux 一口一口蚕食掉的那类付费 Unix。

互联网公司要成千上万台服务器，它们最不愿意做的事，就是为每一台机器付一份操作系统授权费。Linux 免费、能自己改、还能随规模无限复制，简直是为这种"用量爆炸"的场景量身长出来的。于是新增的服务器几乎全倒向 Linux，Solaris 守着存量慢慢缩小。Sun 甚至后来把 Solaris 也开源，想追上这股潮——但那更像是承认：**在这条河上，你要么把船改成靠服务划桨，要么随水位一起退场。**

把这章收进一句话

第一章说软件复制免费，是描述一个成本事实。这一章是把这个事实推到极致后看到的后果：

> 只要一样东西能被免费、开放地复制和改进，任何"靠独占这份代码来收授权费"的护城河，长期都守不住——不是被打穿，是被抽干。你没法对 0 元报价。

但别把它读成"开源 = 免费 = 没钱赚"。真正的规律是**商业模式被迫迁移**：钱从"卖代码副本"挪到"卖发行版、卖支持、卖稳定、卖出事有人负责"。Linux 和 Apache 抽干了卖授权的地基，Red Hat 在新地基上盖起了几百亿的房子，而 Novell 和 Solaris 守着旧地基，看着水一寸寸退去。

护城河能被抽干这件事，还会在后面反复出现——只不过抽水的手会换：下一次是把软件变成按月收租（第十二章），再下一次，是把整座机房变成别人云上的一行账单（第十三章）。

第九章 · 创新者的窘境：赚钱的业务会杀死你

前面八章，我们看的大多是"公司做错了什么"：赌错了商业模式、守不住标准、被上游平台吸掉了空气。这一章要换一个更让人后背发凉的问题：

> 如果一家公司足够聪明、足够赚钱、还特别会听客户的话——它会不会正因为这些优点而死？

答案是会。而且不是偶尔倒霉，是有规律地会。这条规律有个名字，叫创新者的窘境 (The Innovator's Dilemma)——哈佛商学院教授 Clayton Christensen 1997 年出版的一本书里提出的：让在位巨头掉进坑里的，往往不是它的愚蠢，恰恰是它的精明。

先破一个最流行的误解

大公司被小公司掀翻，大众版本的解释永远是"它变大了就变蠢了、变懒了、看不见新东西"。听起来解气，但基本是错的。柯达看见了数码相机——它自己发明的。微软看见了互联网——比尔·盖茨 1995 年就写过著名的"互联网浪潮"备忘录。诺基亚的工程师早就摸过触屏。它们不是没看见。

大白话

真正吓人的地方在这儿：它们看见了，认真算了账，然后**正确地**决定不做。用它们当时手里的那把尺子去量，"不做"是最理性、最负责任、股东最该鼓掌的决定。等这个决定被证明是错的，公司已经掉不了头了。杀死你的不是盲区，是一次算得清清楚楚的、正确的计算。

Christensen 把这套机制拆成了三步。我们一步步走。

第一步：那个又糙又便宜的小东西

故事总是从一个不起眼的新技术开始。Christensen 给它起名叫颠覆式创新 disruptive innovation——注意，"颠覆"不是指它一上来就更强大。恰恰相反，它**一开始又糙又便宜，性能还比不上主流产品，只够服务那些被主流看不上的边缘客户或低端市场。**

早期的数码相机像素低得可笑，照片糊、贵、还得配电脑；胶卷相机随手一拍就是清晰照片。早期的搜索引擎就是个丑陋的输入框，赚的钱少得可怜；那时候真正赚钱的是华丽的门户网站和卖软件授权。早期的 Linux 服务器（上一章讲过）粗糙、没人负责。

关键在于：**这些小东西第一天服务的，从来不是巨头最看重的那批高端客户。** 它先在一个巨头懒得要、利润又薄的角落里活下来——业余爱好者、穷学生、图便宜的小公司。对巨头来说，那块市场既小又不赚钱，让给你，不心疼。

第二步：巨头用错了尺子（而且它不知道那把尺错了）

现在到了整章最核心的一步。巨头当然会评估这个新东西——它有战略部门、有分析师、有MBA。它把新技术拿过来，用自己现有业务的标尺一量，得出三条结论：

- **利润率太低。** 我卖胶卷的毛利高得吓人，数码这门生意的利润率连零头都不到，做它等于自己拉低公司整体毛利。
- **市场太小。** 我现在是几百亿的盘子，那个新市场今年才几百万，就算全吃下来也不够我财报上四舍五入。
- **客户没在要。** 我最大、最赚钱的那些客户，没一个吵着要这个糙东西——听客户的话，是我一直以来做对的事啊。

请注意，**这三条每一条都是对的。** 它不是数据算错了，是数据算得太对了。一家管理良好的公司，本来就该把钱和最好的人投到利润最高、市场最大、客户最想要的地方去——这正是它当年打赢的方式。于是它"理性地"决定：这个小玩意儿不值得投，我们专注守好现金牛。

> 大白话
>
> 这就是"窘境"两个字的分量：不是"该做的没做"，而是**用来衡量该不该做的那把尺子本身，是为守护旧生意校准的**。护城河越深、现金牛越肥，这把尺子就越会告诉你"别碰那个小东西"。越成功的公司，尺子越硬，越量不出未来。

第三步：等你看清，已经掉不了头

然后时间开始工作。那个糙东西沿着它自己的曲线，一年一个样地变好——数码相机像素追上来了、又便宜又能立刻看；搜索框后面长出了广告拍卖，成了印钞机；Linux 变得企业敢

用了。等它终于好到够着主流市场时，巨头猛一抬头，发现对手已经站到自己家门口了。

这时候巨头想转身，却发现转不动了。不是不想，是它的整个身体都长在旧生意上：

- **利润结构掉不了头。** 公司习惯了 70% 的毛利，一个新业务只有 20% 毛利，投它就是"主动让财报变难看"，没有哪个高管扛得起这个短期代价。
- **组织和渠道掉不了头。** 最好的工程师、销售、预算，全绑在保护旧业务的考核指标上。任何威胁旧业务的新方向，会被组织当成异物本能地排斥掉——不是有人下令封杀，是无数个中层各自"理性"地把资源挪回了那个更赚钱的地方。
- **客户关系掉不了头。** 它引以为傲的"紧跟大客户"，此刻变成了枷锁——大客户要的还是旧东西，跟着他们走，正好走进死胡同。

柯达：把自己发明的未来锁进了抽屉

为了让你相信这不是软件业独有的病，先看隔壁行业一个教科书级的例子——柯达 Kodak（曾经统治全球胶卷市场的美国影像巨头）。

1975 年，柯达自己的工程师造出了世界上第一台数码相机原型。公司高层看懂了它，也看懂了它的威胁——正因为看懂了，才把它雪藏。逻辑冷酷又清晰：柯达真正赚钱的不是卖相机，是卖胶卷、卖冲印，那是一条每按一次快门就要复购的高毛利耗材生意。而数码相机不用胶卷。**推数码，等于亲手拆掉自己最肥的那条现金牛。**

于是柯达做了当时"最负责任"的决定：小心翼翼地保护胶卷利润，让数码慢慢来。结果数码浪潮没有慢慢来，它按自己的曲线一路变好、变便宜，最终把整个胶卷市场淹了。2012 年，柯达申请破产。它不是没看见数码，它是发明了数码、算清了账、然后正确地选择了保护自己——正是这个正确，杀死了它。

微软：现金牛太香，就让搜索长成了帝国

同样的病，在软件业发作过很多次。最典型的一次，是微软对互联网和搜索的态度。

整个九十年代到两千年代，微软最肥的两头现金牛是 Windows 和 Office——高毛利、近乎垄断、每年稳稳印钱。用这把尺子去量早期的互联网搜索，结论一目了然：那就是个不赚钱的边缘玩意儿，一个丑输入框，利润薄、还看不清怎么变现。微软的资源、考核、组织重心，全都绑在"保护 Windows"这根轴上——任何一个新方向，只要威胁到 Windows 或者

不能反哺 Windows，就天然被组织排斥。（第六章的浏览器战争，其实就是这个病的一次早期发作：微软的第一反应不是"互联网是不是新平台"，而是"它会不会绕过 Windows"。）

于是微软眼睁睁看着 Google 从那个"不起眼的搜索框"，一步步长成了广告帝国。等微软反应过来搜索是门天大的生意、砸重金做 Bing 去追时，Google 早已在渠道、数据和规模上筑起了新护城河——追是能追，但再也回不到起跑线了。（搜索到底怎么变成印钞机的，留给下一章讲；这一章只回答一个问题：微软那么聪明，**为什么会允许它长起来**。答案就是那把尺子——早期搜索用 Windows 的标准量，怎么量都"不值得做"。）

落点：护城河越深，转身越难

把这一章收束成一句话，正好接得住全书的贯穿问题：

> 护城河越深、现金牛越肥，组织就越有理由拒绝那个将来会取代它的小东西——护城河本身，成了转身的枷锁。

这是前面几章的一个反转。之前我们羡慕护城河：标准、锁定、网络效应，都是让公司稳坐钓鱼台的好东西。这一章告诉你，同一条护城河，在换代的关口会反过来把公司焊死在原地——它让"守住旧生意"永远显得比"押注新生意"更理性，直到那个新生意变成了别人的帝国。

大白话

要说清楚一件事：这不等于"大公司必死"。有的公司真的完成了那次几乎不可能的转身——IBM 从卖硬件转向卖服务，苹果和微软后来也各自跳过了这道坎。它们是怎么做到的、和柯达差在哪，是第十四章的事。这里只需要记住那个最反直觉的诊断：**让你掉进坑里的，常常不是你没看见新东西，而是你看见了、算过账、然后"正确地"决定不做。**

第十章 · 搜索与「注意力」的商业模式革命

一个不卖东西的公司，怎么成了最赚钱的公司

前面九章的主角，做生意的路子其实都一样：写好软件，卖给你。卖操作系统的授权、卖办公套件的光盘、卖捆着硬件的机器——花样再多，本质都是"这是我的商品，你付钱把它买走"。用户掏钱，公司收钱，天经地义。

然后来了一家公司，把这套规矩整个掀翻了：它最核心的产品，对用户**一分钱都不收**；而它，却长成了地球上最赚钱的公司之一。

这家公司叫 Google（谷歌，1998 年成立，做搜索起家）。它没在"我的软件比你的好，所以卖得更贵"这条老路上和谁较劲。它换了一个问题：如果产品干脆不要钱，那钱从哪儿来？

大白话

一句话：前面的公司都在琢磨"怎么把软件卖出更高的价"；Google 琢磨的是"怎么让软件免费，然后从别的地方把钱赚回来"。这不是价格战，是换了一门生意。

免费的东西，为什么反而更赚钱

先想通第一件反直觉的事：**产品免费，为什么反而能赚更多的钱？**

关键在于，Google 想清楚了一件事——搜索这件事，用户根本不是它的**客户**，而是它的**货**。

这话听着刺耳，但机制就是这么运转的。你在搜索框里打字，说明你此刻正想找点什么——想买跑鞋、想订机票、想修水管。你的这份"我正想要某样东西的注意力"，对某些人来说值真金白银：卖跑鞋的、卖机票的、修水管的，做梦都想在你正想要的那一刻，把广告递到你眼前。

于是 Google 做的事，是把你的注意力打包，卖给这些广告主。**用户不掏钱，因为用户不是买东西的人——用户是被卖的那批"库存"。**真正掏钱的，是想够到你注意力的广告主。

大白话

就像免费的电视台：你看节目不花钱，因为电视台真正在卖的，是"正在看节目的这群观众"——卖给插播广告的厂商。你以为你是观众，其实你是商品。Google 把这套搬到了互联网上，而且做得精细得多。

想通这层，"免费"的威力就出来了。收费会劝退人，免费则让分发毫无摩擦——不用付费、不用安装、打开网页就能用。于是用它的人像决堤一样涨。而用的人越多，Google 手里能卖的"注意力库存"就越多，广告位就越值钱。**免费不是慈善，是把用户规模做到最大的最快办法，而规模本身就是要卖的货。**

真正精巧的地方：把广告位拿去拍卖

光是"免费换规模、卖广告"还不算革命——报纸电视早就这么干了。Google 真正精巧、也真正改写行业的，是它**怎么定广告的价**。

老办法是这样的：广告主找上门，销售报个价，谈个人情，一个位置卖多少钱基本靠拍脑袋和讨价还价。这套慢、贵、还容易掺水。

Google 在 2000 年 10 月首次推出 AdWords（关键词广告系统）时，其实还很传统：广告按**千次展示计费**（CPM，广告每被看到一千次收一笔钱），价也是谈出来的。真正的转折发生在 **2002 年 2 月**——Google 推出 AdWords Select，把玩法彻底改成了另一种：**关键词竞价 + 按点击付费。**广告位不再由销售报价，而是让所有想买的广告主，对着一个个**关键词**公开竞价——谁愿意为"跑鞋"这个词出的钱高，谁的广告就更可能被排在前面；而且只有真被点了才收钱。

这就是 关键词拍卖：把每一个搜索词，都变成一个随时在开的小拍卖会。有人搜"跑鞋"，后台一瞬间，一群卖鞋的广告主就为这次展示的机会竞了一次价，价高者上。

大白话

把它想成一个永不落幕的菜市场：每个词是一个摊位，谁出价高谁站摊位前排。只不过这个市场里没有吆喝的商贩，全靠机器在毫秒之间自动撮合——你搜索的那一刹那，一场拍卖已经拍完了。

为什么让"最相关"的广告赢，对所有人都好

但只按出价高低排，会出大问题：一个财大气粗但和你八竿子打不着的广告，会把真正对你有用的广告挤下去。用户被烦到，广告也没人点，最后谁都不赚。

Google 加了第二个变量：点击率（CTR，一条广告被人点开的概率）。排序不只看出价，还看这条广告有多可能被点。真正的排位，大致等于**出价 × 点击率**——出价再高，如果没人愿意点，也排不上去；出价一般，但特别对路、大家都爱点，反而能靠前。

这一步藏着全章最漂亮的一个道理：**为什么让"最相关"的广告赢，对所有人都好？**

- **对用户**：胜出的广告是大家最可能点的，也就是最对路、最有用的——广告体验反而更好。
- **对广告主**：你不必靠砸钱压过所有人，把广告做得更对路（更多人点）也能赢，出价压力小了。
- **对 Google**：广告是按点击付费的（没人点就不收钱），越多人点，Google 越赚。所以让"最可能被点"的广告胜出，正好就是让 Google 最赚钱。

这是精妙之处：**用户体验和平台收入，头一次朝着同一个方向使劲，而不是互相打架。**让用户满意的广告，恰好也是最赚钱的广告。

付"下一名的价"：一个借来的聪明主意，和它没说出口的小秘密

还有一层设计，值得单独说，因为它藏着一个特别值得玩味的故事。这套系统用的不是"你出多少就付多少"的普通拍卖，而是一种叫 广义次价拍卖（Generalized Second-Price，简称 GSP）的机制。名字唬人，但通行的说法很朴素——

赢家不用付自己出的价，而是付排在他下一名那个人的出价。你出价最高、排第一，但你实际付的，是排第二那位出的钱（再加一点点）；排第二的付第三名的价，以此类推。

为什么要这么绕？因为这么一来，广告主直觉上就不必把价"咬到最满"——反正你付的是下一名的价，不是自己那个数。系统于是每次大致能收到"下一名愿意出的那个价"，这通常很接近这次点击的真实身价。**用户看到最相关的广告、广告主不必你死我活地互相加价、平台又收得饱满——三方同向的那股劲，在这里又加了一道。**

到这儿都很美好。但这里有个常被讲错、也确实值得较真的地方——

大白话

你多半听过这样一个漂亮的说法："付第二高价，所以大家干脆说真话、报真实价位就是最优。"这个结论是**真的**，但它只在一种极简单的情形下成立：**只拍一件东西**。就一幅画，出价最高的人赢、只付第二高的价——这时你确实该直接报真实估值，因为你付多少由别人决定、跟你报多少无关，算计没好处。这叫 维克里拍卖（单物品的次价拍卖），"说真话最优"是它一条能被严格证明的数学性质。

问题在于，搜索页上不是一件东西，而是**好几个广告位**：第一位比第二位显眼得多，值的钱也差一截。把"付下一名的价"从"一件"搬到"好几件"上，它就**不再**保证"说真话最优"了——此时诚实报价既不是稳赢的占优策略，也不总是均衡；精明的广告主反而会**策略性地少报一点**，用更低的成本换到第二、第三位那些同样不错的位置。GSP 借来了次价拍卖"让人不必咬满价"的直觉，却没有继承它"说真话最优"的严格保证。

真要在多个位置上也严格保证"说真话最优"，经济学里是有答案的，叫 VCG 机制。它数学上更漂亮、更"正确"。可 Google 用的偏偏**不是** VCG，而是更土、更好懂的 GSP。

这才是全章我最想留给你的洞见：**"理论上最优"和"工程上够用又好解释"，常常不是同一件事——真正能印钞的系统，往往选了后者。**VCG 更完美，但算起来绕、跟广告主讲不清；GSP 不完美，却一句话就说得明白（"你付下一名的价"），好实现、好解释、跑起来又足够好。Google 要的不是一篇能拿奖的论文，是一台今天就能开动、还让人听得懂的印钞机——于是它选了那个"不够完美但恰到好处"的答案。

数据飞轮：越搜越准，越准越搜

免费换来了规模，拍卖把规模变成了钱。但 Google 护城河最深的一层，是规模会**自我加固**。

看这个循环：搜的人越多 → 用户点了什么、跳过了什么、改了什么词，这些数据就越多 → 搜索结果和广告排序就调得越准 → 结果越准，越多人来搜 → 广告越值钱 → 越有钱投入服务器、人才、把系统做得更好 → 又吸引更多人来搜……

大白话

这是一个越滚越大的雪球。第四章讲过生态飞轮：应用越多越好用、越好用越多人写应用。这里是同一台机器，只不过轮子是建在**数据**上的——搜的人喂给它数据，数据让它更聪明，更聪明又招来更多人。后来者就算抄走全部代码，也抄不走这些年积累的海量真实点击数据。

于是护城河变了性质。以前是"我有你没有的代码、你没有的授权"；现在是**"我有你追不上的规模和数据"**。代码可以重写，甚至可以开源白送，但十几亿人用出来的数据，你没法凭空造出来。这是一种连价格战都省了的护城河——大家都免费，你根本没有价格可打，只能眼看对手的雪球越滚越大。

反面教材：Yahoo，想什么都自己做的门户

把 Google 衬托得最清楚的，是它的老前辈 Yahoo（雅虎）。

Yahoo 早年是互联网的头号入口，起家靠的是**人工编辑的网站目录**——一群编辑像图书管理员一样，把网站按类别一条条手工整理进目录，你顺着分类往下点，就能找到想去的地方。这在网站还不算多的年代很好用。它后来长成了一个"门户"：新闻、邮箱、财经、体育、聊天……什么都往里塞，想做一个用户上网的大杂烩总入口，什么都自己做。

这正是旧范式的样子：靠人力堆内容，靠"大而全"圈住用户，广告位很大程度上还是靠销售一单一单去谈。它的两条腿，都跑不过 Google 的机器：

- **专注 vs 大杂烩**：Google 把"搜索"这一件事做到极致，用算法自动整理全网，规模一上来，人工编辑的目录根本追不上网页增长的速度。什么都做的 Yahoo，反而每一样都不够深。
- **拍卖 vs 人情**：Google 的广告是机器自动竞价、按点击结算、还自动挑最相关的；Yahoo 的广告体系长期更依赖人去谈、去定价，慢、贵、榨不干每一次展示的价值。

当分发免费、边际成本几乎为零、规模还自我强化，Yahoo 那台靠人力驱动的引擎就慢慢熄了火。它不是某一天被打败的，是被规模效应一年一年碾过去的。2017 年，Yahoo 把自己的核心业务以约 48 亿美元卖给了电信公司 Verizon——对一家曾定义了整个互联网入口的公司，这是个近乎贱卖的价格。

这一章的规律：护城河从"代码"变成了"规模和数据"

把 Google 和 Yahoo 放一起，一条新规律浮出来。

前面几章，护城河的材料是**代码和标准**：我掌握了你没有的操作系统、你依赖的运行时、你锁死的仓库。到了 Google 这里，材料换了——产品免费送出去，收入从广告拍卖里来，而真正让别人追不上的，是**规模和规模喂出来的数据**。

> **当分发免费、边际成本为零、且规模能自我强化时，护城河就从"我有你没有的代码"变成了"我有你追不上的规模和数据"。这是一种连价格战都免了的护城河——因为大家都免费，你连打的价都没有。**

Google 真正的发明，与其说是搜索引擎，不如说是一门生意：把免费产品变成一座自动运转的广告拍卖场，用一条数学规则让用户、广告主、平台三方同向，再让数据飞轮把领先越滚越大。它证明了软件公司起伏的又一条底层规律——**商业模式本身可以是护城河**，而且往往比任何一行代码都更深、更难被抄。

下一章我们会看到，再深的护城河也有天敌：当脚下的地基整个移动——从 PC 挪到手机——上一代所有的规模和优势，会在一夜之间被清零。

第十一章 · 平台转移：脚下的地基会移动

2007 年 1 月，乔布斯在台上掏出第一代 iPhone 的时候，旧世界的王者们其实没太当回事。当时黑莓的联席 CEO 看完发布会，评价大意是：一块全是玻璃、没有键盘的手机，商务人士打字打不快，续航还差，掀不起什么风浪。三年后，说这话的公司开始塌方。

这一章讲的，是软件史上最残酷、也最容易被误解的一种死法。前面几章的失败者，多少是"打输了同一场仗"——同一个赛道上，被更强的对手碾过去。这一章不一样：**输家没有打输，他们脚下的地面直接塌了。**规则换了，比赛项目换了，他们还站在原来的跑道上，握着上一届冠军的奖杯。

先认识旧世界的三个王

要理解这场塌方有多反直觉，得先看清楚：在 2007 年之前，这三家不是弱者，他们是各自领域做到极致的强者。

BlackBerry（黑莓）是加拿大公司 RIM 的产品，它几乎发明了"随身收邮件"这件事。一排小小的实体全键盘，配上一套省流量的加密邮件推送系统——你的邮件到了服务器，它主动"推"到你手机上，不费流量、不用刷新，还加密。这在流量按字节收费、网络又慢又贵的年代，是杀手级的本事。华尔街的交易员、公司的高管、政府官员，人手一台。更狠的是，黑莓卖的不只是手机，还有一整套让企业 IT 部门放心的后台系统。企业一旦采购部署，就被牢牢锁住。护城河看上去深不见底：键盘手感、超长续航、企业安全、采购锁定，样样都是硬功夫。

Palm更早，它是PDA（掌上电脑）的开创者——一块能用触控笔戳着记日程、存通讯录的小板子。在还没有智能手机这个词的年代，Palm 教会了整整一代人"把电脑装进口袋"。触控操作的很多早期探索，都是 Palm 做的。

诺基亚就更不用说了。功能机时代的全球霸主，一年卖出去的手机数以亿计，供应链、渠道、品牌，全球第一，没有之一。

大白话

记住这个画面：2007 年初，如果你要评"最好的手机公司"，答案是这三家里挑。他们不是没落贵族，他们是当红霸主。这一点很重要——正因为他们太成功，才死得那么快。

iPhone 一开始，很多方面确实更差

这里是全章最关键的一个事实，请慢慢读：iPhone（2007 年苹果发布的全触屏手机）刚出来的时候，用旧世界的标准去衡量，它样样不如黑莓。

- 第一代不支持 3G（当时更快的移动网络），上网慢；
- 没有实体键盘，在玻璃上打字又慢又容易错，商务人士嫌弃；
- 续航差，黑莓能撑好几天，它一天一充；
- 没有黑莓那套企业邮件和 IT 后台，公司采购根本看不上。

如果你是黑莓高管，拿着一张评分表逐项打勾，你会得出一个完全"理性"的结论：这玩意儿是个玩具，我们赢定了。他们没看错任何一项——键盘、续航、邮件、安全，iPhone 确实都更差。

他们错的不是评分，是**那张评分表本身已经作废了。**

地基移动了：竞争的定义被换掉

iPhone 真正做的事，不是造了一台更好的手机，而是**重新定义了"手机"这个词**。在它之前，手机是"一个能打电话、顺便能收邮件的通讯设备"；在它之后，手机是"一块能装任何软件的全触屏电脑，顺便能打电话"。

大白话

打个比方。旧世界大家都在比谁的马车跑得快——马选得好、车轴造得精、车夫经验足。黑莓是跑得最快的马车。然后有人开来一辆汽车。第一辆汽车又慢又爱抛锚，比好马车差远了。可问题是：一旦大家认定"以后要开汽车"，你那身养马、造车轴、驯车夫的绝活，一夜之间不值钱了。你不是输给了更快的马，你是输给了"不再需要马"。

当"手机"变成"口袋里的电脑"，衡量它好坏的标准就整个换了：从"打字快不快、续航久不久、邮件稳不稳"，变成了"能装多少软件、浏览器好不好用、体验顺不顺手"。黑莓在旧标准上的所有优势，在新标准里几乎不计分。这就是平台转移（范式更替）——脚下的地基整块移动，评判规则被替换。

飞轮又来了：这次搬到了手机上

如果只是标准变了，黑莓还有机会跟。真正锁死结局的，是第四章那个"越转越快的圈"被搬到了手机上。

2008 年，苹果推出 App Store（苹果的手机应用商店）：任何开发者都能把软件放上去卖，用户点一下就装。同一年，Google 的 Android（一套开源、免费给各家手机厂用的手机操作系统）商用。iOS 和 Android，两大生态就此成型。

还记得那个飞轮吗？应用越多，用户越多；用户越多，越多开发者来写应用。这个圈一旦在 iPhone 和 Android 上转起来，全世界第三方开发者的重力就全倒向了这两边——因为用户在那里，钱在那里。开发者不会为一个用户在流失、生态在萎缩的平台写软件。

> 手机的胜负，最终不是手机厂之间打出来的，是开发者用脚投票投出来的。

黑莓和诺基亚眼睁睁看着这个飞轮在别人那边越转越满，自己这边越来越冷清。等他们想推自己的应用商店，货架上空空荡荡，没人来开店。

为什么"太成功"反而让他们转不动身

现在到了最扎心的问题：他们不是傻子，趋势后来看得明明白白，为什么不掉头？

答案是：**正因为他们在旧范式里太成功，才转身极慢。**成功不是护身符，成功本身变成了绑住手脚的绳子。

- **企业客户锁定，成了包袱。**黑莓最赚钱的是企业采购和那套邮件后台。要拥抱新范式，就得亲手弱化键盘、弱化邮件那套卖点——等于砍掉自己的利润来源。哪个高管敢干？
- **自有操作系统，成了包袱。**黑莓、诺基亚都有自己的老系统。转向触屏 App 生态，意味着推倒重来。黑莓的 BB10（黑莓 2013 年才推出的新系统）、诺基亚死守的 Symbian（诺基

亚的老手机系统），都来得太晚，等它们上市时，飞轮早在对面转满了。

- **键盘信仰，成了包袱。**整个公司从上到下都相信"商务人士离不开实体键盘"。这个信仰曾经是对的，也曾经是护城河。可当用户开始接受玻璃屏，这份信仰就变成了看不见新大陆的眼罩。

> 大白话
>
> 这正是第九章"创新者的窘境"在手机上的重演：看清趋势的人未必救得了自己，因为整个组织的重力——利润结构、客户承诺、员工技能、老板的骄傲——全都焊死在旧范式上。掉头，等于对着自己最赚钱的业务开刀。绝大多数公司做不到，不是因为蠢，是因为"理性"。

结局：几年之内，深沟变成壕堑

剩下的就是塌方的速度，快得吓人。

- 黑莓在 2009 年前后还是北美智能机的领先者，几年内市占崩到近乎归零，2016 年宣布不再自研手机；
- 诺基亚的手机业务在 2013–14 年卖给微软，此后基本消亡；
- Palm 这个开创者，2010 年被惠普（HP）收购，随后关停。

三个曾经的王，握着旧世界最好的产品，输给了重新定义品类的人。

把这条规律记进脑子

本章要你带走的，是一句会一次次应验的话：

> 护城河，是相对于"当前地形"挖的。地形一变，你挖得最深的那条沟，可能恰好变成让你转身最慢的那道壕堑。

黑莓的键盘、企业锁定、邮件后台，在 PC-手机的旧地形里是又深又宽的护城河；范式一换到"口袋电脑 + App 生态"，同一条沟就把它自己困在了里面——想过河，先得填掉自己的沟，而填沟就是放血。

所以判断一家公司危不危险，光看它护城河多深没用，得多问一句：**它的城，是不是修在一块正在移动的地基上？**看趋势不难，难的是当你全部的利润、组织、骄傲都长在旧地基上时，还能不能狠下心搬家。这个问题，我们留到第十四章——那些真的搬成功了的公司——再来回答。

第十二章 · 从卖授权到收租：SaaS 与订阅革命

前面几章的护城河，都建在"代码"上：谁的软件更强、谁掌握标准、谁的用户更难搬走。这一章要讲一件看起来无聊透顶的事——公司怎么收钱。把"一次性买断"改成"按月续费"，听上去只是财务部门换个账本。但你会看到，这一改，把整间公司的激励、留存和身价全部重写了一遍。

> 为什么同样是一年赚一亿，"每月自动续费的一亿"能让公司值好几倍于"一锤子买卖的一亿"？为什么改成收租之后，公司反而被逼着变好？

旧模式：卖一次，然后求你再买一次

先看老玩法。永久授权 perpetual license——你一次性买断某个版本的软件，付一大笔钱（比如一套 Photoshop 几百上千美元），然后它永远归你用。

大白话

这像买 DVD：一次付清，碟片是你的，想看几遍看几遍，厂商从此和你没关系。要想再从你身上赚钱，它得出一部新电影，再说服你去买。

软件厂商靠什么持续赚钱？靠每隔一两年出个新版本，堆一堆新功能，然后想办法逼老用户掏钱升级。这门生意有三个天生的毛病：

- **收入像脉冲。** 新版本发布那年营收冲高，之后逐季回落，直到下一个版本再冲一次。财报忽高忽低，像心电图。
- **增长要靠"再说服一次"。** 每一代都得重新证明"这次的新功能值得你再花一次钱"。用户觉得手上这版够用，就一分钱不给了。
- **厂商和用户利益是拧着的。** 卖完这单，厂商巴不得你觉得旧版本过时、快去买新版；至于你买回去用得爽不爽，它其实没那么在乎——反正钱已经到手了。

还有个隐痛：**老版本钉子户**。总有一大批人守着三年前的版本死活不升级，照样用得好好的。对厂商来说，这些人是"已经不再付钱、却还在占用支持资源"的存量，怎么撬都撬不

动。再加上盗版——买断软件装在用户自己机器上，破解一份就能复制无数份。

新模式：软件不卖了，改成"接进来用"

另一条路，是SaaS（Software as a Service，软件即服务）——软件不再装进你的电脑，而是跑在厂商的服务器上，你通过浏览器或网络连过去用，按月或按年订阅 subscription付费。不续费，就断供。

> 大白话
>
> 如果买断像买 DVD，订阅就像 Netflix 会员：你不再拥有任何一张碟，你买的是"这个月能一直看"的资格。停止付费，画面立刻黑掉。片子始终在 Netflix 那边，你只是接进去。

很多人第一反应：这不就是"分期付款"吗？把一次几百块拆成每月几十块，本质一样。**这是本章最大的误解，必须掰开。** 订阅制改的根本不是付款节奏，而是三样更深的东西：收入的形状、公司的激励、和护城河的位置。一个一个说。

改变一：收入从脉冲变成细水长流——而市场为"稳定"多付好几倍

订阅制下，收入变成一条平稳流入的溪水：这个月订的人，大概率下个月还在订，钱月月自动进账。这种"能持续重复、可预测"的收入，行业里叫ARR（Annual Recurring Revenue，年度经常性收入）——每年稳定重复收上来的那部分钱。

关键在于：**资本市场给这两种收入的估值倍数，天差地别。** 同样是一年一亿利润——

- 买断制的一亿：明年得重新卖、重新说服，谁也不敢保证还有。市场看它像一份不确定的合同，只肯按较低倍数出价。
- 订阅制的一亿：明年这批用户大部分还在，钱几乎自动重来一遍，还能在此基础上再叠加新用户。市场看它像一份能复利滚动的年金，愿意按高得多的倍数出价。

大白话

换个说法：你更愿意买一棵"今年结一次果、明年结不结不知道"的树，还是一棵"每年都稳定结果、还越长越大"的树？同样一筐果子，后者这棵树本身贵得多。订阅制卖的不是果子，是那棵会一直结果的树。

这就是订阅制的第一层魔力：**不改一分钱利润，只把收入从"一次性"变成"稳定重复"，公司的身价就能翻好几倍。** 这也是为什么两千年后，一批又一批软件公司哪怕短期难看，也要拼命往订阅制转——它们不只是在换收钱方式，是在换一个更高的估值锚。

改变二：激励反转——厂商被逼着让你"每个月都还想用"

这是订阅制最反直觉、也最重要的一层。回想买断制：厂商赚完这单，注意力立刻转向下一个版本、下一个新客户，你用得爽不爽它管不着。

订阅制把这套激励整个掉了个头。你的钱不是一次交清，而是**每个月都要重新决定要不要继续给**。只要哪个月你觉得不值、觉得烦、觉得有更好的，随手就能退订。于是厂商最怕的一个数字诞生了：流失率 churn——每个月退订用户占总用户的比例。

大白话

买断制盯的是"这个月签了多少新客户"，像追新猎物；订阅制盯的是"这个月跑了多少老客户"，像守一个漏水的桶。桶再大，底下漏得快，也存不住水。

这个转变的后果极其深远：**为了让你每个月都不想退，厂商被迫持续地把产品改好。** 不能再攒两年放个大版本了事，而要月月修 bug、月月加实用的小功能、月月盯着你用得顺不顺。用户的满意，从"卖出去之前要哄"的一次性任务，变成了"卖出去之后要天天维护"的长期义务。

换句话说，订阅制用一条冷冰冰的财务机制，把"公司必须持续变好"这件事，从口号变成了生死线。**不是厂商变善良了，是收租模式让"变好"变成了唯一活路。**

改变三：交付搬到云端，顺手解决一堆老问题、长出新的锁定

因为软件改在厂商服务器上跑，好几件老大难顺带就解决了：

- **盗版基本消失。** 软件不在你机器上，没东西可破解——你只有一个"接不接得进去"的账号。
- **钉子户绝迹。** 所有人始终用的是服务器上那唯一一份最新版，厂商也不用再维护一堆历史版本。
- **数据沉淀在厂商那边。** 你的文件、项目、协作记录都存在云端。这一条最关键——它长出了一种全新的护城河。

想想：当你几年的工作文件、团队的协作历史、和别的工具打通的工作流，全都长在这个订阅服务里，退订的代价就不再是"换个软件"，而是"搬走整个家当、打断整套习惯"。**锁定你的不再只是代码，而是你自己沉淀在里面的东西。** 这句话，是本章落点的一半，先记住。

Adobe：一次惊险的纵身一跳

Adobe（Photoshop、Illustrator 等设计软件的厂商）是这场革命最经典的教科书，因为它是"半路上车"——它有几十年的买断制家业要拆掉重来。

2013 年，Adobe 做了个当时看着近乎疯狂的决定：把 Photoshop 这些王牌软件打包成 **Creative Cloud**，从此**只租不卖**，取消永久授权，全部改成按月订阅。

短期账面惨不忍睹，两头受压：

- **营收和口碑双双承压。** 习惯了"一次买断永久用"的老用户炸了锅，觉得这是变相逼他们永远交租，网上骂声一片。
- **报表更难看。** 这里有个反直觉的会计效应：过去卖一套买断软件，一大笔钱当场计入当期收入；改成订阅后，一年的订阅费要摊到十二个月里一点点确认（这叫递延收入 deferred revenue——钱收了，但要按月慢慢算成收入）。于是转型初期，同样的生意，当期报表上的收入反而缩水了。

这就是"惊险一跳"的惊险之处：**你要主动把眼前好看的数字砸烂，去换一个几年后才兑现的更好结构。** 中间那段账面难看、用户骂街的日子，需要极大的定力才扛得住。

但 Adobe 挺过了转换期。之后的故事你大概猜到了：收入更高、更稳、可预测，用户始终在最新版，盗版被摁住，数据沉在云端。它的股价此后走出一条长牛。Adobe 用自己证明

了：老牌买断公司能完成这一跳，代价是一段咬牙硬扛的至暗时刻。

Salesforce：一出生就长在新地基上

如果说 Adobe 是艰难转身，Salesforce（1999 年成立的企业软件公司，做客户管理系统起家）就是天生的原住民——它从第一天起就生在云上、纯订阅，根本没有旧包袱要拆。

它当年的口号极其挑衅："**No Software**"。字面是"不要软件"，真意不是不用软件，而是：**别再在你自己的机器上装、自己维护那一大坨软件了。** 传统企业软件（比如 Oracle、SAP 卖的那些）动辄要企业自己买服务器、装系统、雇一队人运维、每隔几年痛苦升级一次。Salesforce 说：这些统统交给我，你打开浏览器登进来用就行，按月付费。

它一上来就把订阅制三层好处全占齐了：细水长流的经常性收入、逼自己持续改进的激励、数据全沉在云端的锁定。等于它不是在旧规则里竞争，而是**直接改写了企业软件的玩法**。这一改写的压力，最终反噬到那些买断制巨头身上——连 Oracle、SAP 这样的老王，也不得不掉头往云和订阅转，去追一个当年被它们轻视的年轻人定的新规矩。

把这章收进一句话

回到开头那个"只是换收钱方式"的错觉。走完一圈你会看到，从买断到订阅，改的从来不只是账本：

> 把生意从"卖一次"改成"绑一辈子"，收入从脉冲变成可复利的经常性收入（市场因此多付好几倍身价）；激励从"卖完就走"反转成"每月都得让你想留"（公司因此被逼着持续变好）；护城河从"独占的代码"迁移到"你自己沉淀在服务里的数据、工作流和习惯"。

这正好呼应全书那条主线：**护城河不一定建在代码上，也可以建在"关系与持续价值"上。** 当用户的家当长在一个你每月都在改进的服务里，退订的痛，比换个软件大得多。谁先把生意从卖授权改成收租，谁就换到了一块更稳的地基。

而这块地基还能再往下挖一层——如果软件都跑在别人服务器上了，那机房本身、算力本身，是不是也能变成一门按月收租的生意？下一章，我们把镜头对准云（第十三章）。

第十三章 · 云：把资本开支变成别人的生意

2003 年前后，亚马逊的工程师们碰到一个反复出现的麻烦：每次要上线一个新功能，团队都得先花几个月准备服务器、数据库、存储、网络——这些底层的东西每个团队都在重复造。更要命的是，为了扛住「黑色星期五」这种大促峰值，亚马逊必须按最高峰的流量来买机器。可一年里真正的峰值就那么几天，剩下的三百多天，那一大片机房里的服务器大多在空转。

这是一笔巨大的、看起来只能认栽的浪费。谁能想到，正是这笔浪费，长成了今天全世界最赚钱的生意之一。

为什么闲置的成本，能变成最赚钱的生意

先讲清一个词。资本开支 CapEx (Capital Expenditure)——就是「先砸一大笔钱去买能用很多年的大件资产」，比如盖一座数据中心、买上万台服务器。它的特点是：钱要提前花，而且花得很重。对一家公司来说，CapEx 通常是成本中心——是往外流的钱，是财报上让人皱眉的数字。

亚马逊为撑自家电商建了一整套基础设施，也就背上了一大笔 CapEx。按常理，接下来该做的是「省」：怎么让机器少空转、怎么把这块成本压下去。

但 2006 年，亚马逊做了一件反直觉的事。它推出了 AWS (Amazon Web Services)，把自己内部用的那套基础设施，标准化成「按用量出租」的服务，卖给所有人。第一批产品是 S3（按量计费的存储，你存多少数据、付多少钱）和 EC2（按小时租的虚拟服务器，用几小时算几小时的钱）。

这就是 IaaS (Infrastructure as a Service，基础设施即服务)——把服务器、存储、网络，变成像水和电一样的东西：你不用自己挖井、自己发电，打开水龙头、插上插座，用多少交多少钱。

大白话

大白话：以前你想开一家网店，得先自己买服务器、租机房、雇人运维，几十万起步，还没开张就先出血。有了云，你注册个账号，按小时租几台虚拟机，几十块钱就能起步，半夜火了就多租几台，凌晨没人了就退掉——像交水电费一样。

这一翻转的精妙之处在于：亚马逊没有消灭那笔闲置成本，而是**把它卖了出去**。原本只是自家的成本中心，现在成了对外收租的利润引擎；原本三百天空转的服务器，现在租给了成千上万家创业公司。更妙的是，外部客户的租金，反过来把亚马逊自己的基础设施成本给摊薄了——它用别人的钱，养起了自己的机房。

> 亚马逊没有把闲置成本省掉，而是把它租出去。成本中心一旦学会收租，就变成了印钞机。

规模经济：一种新的「收税权」

第三章讲过，微软靠控制操作系统标准，向整个 PC 产业「收税」。云的收税权来自另一个地方——不是标准，而是**规模**。

数据中心这门生意有个硬道理：越大越便宜。你买 100 台服务器和买 100 万台，单价完全不同；你为一个小机房谈电价，和为占全国用电百分之几的巨型机房谈电价，也完全不同。运维、冷却、网络、安全，这些成本几乎都随规模摊薄。规模最大的玩家，单位算力的成本天然就比别人低。

这带来一个可怕的正反馈：

- 规模最大 → 单位成本最低 → 能报出别人「亏本才敢给」的价格；
- 价格最低 → 抢到更多客户 → 规模变得更大；
- 规模更大 → 成本更低 → 还能继续降价，把对手一点点挤出去。

这就是「越大越便宜、越便宜越大」的飞轮。AWS 上线十几年里降价上百次——不是发善心，而是每一次降价都在把成本更高的对手往悬崖边推。对小玩家来说，这几乎是无解的：你的成本地板，就是巨头的天花板。

大白话

为什么这算「收税」？因为它和第三章的微软是同一类东西——占住整个产业绕不开的那一层，然后从每一笔流过的生意里抽成。区别在于：微软的税建在「控制标准」上，云的税建在「资本和规模」上。前者是聪明，后者是砸钱砸出来的厚。

为什么云比操作系统更深

操作系统的护城河已经够深了，但云更深。深在哪？深在**搬家几乎搬不动**。

一家公司一旦把业务扎根在某朵云上，会发生三件事：

1. **数据的重量**：几十上百 TB 的数据存在这朵云里，搬走不只要付一大笔「搬运费」（云厂商往往对数据「搬出」收费），还要停机、要重新验证，风险极高。
2. **架构长在了云里**：你的系统不是简单跑在虚拟机上，而是深度用了这家云的几十种「托管服务」——它的数据库、它的消息队列、它的身份认证。这些服务各家的接口都不一样，换一家等于把代码大改一遍。
3. **能力也绑住了**：你的工程师熟的是这朵云的操作方式，运维、监控、自动化脚本全是围着它建的。换云，团队也得重新学。

> 把业务从一朵云搬到另一朵，像给一架正在飞行的飞机换引擎——不是不能换，是你不敢在半空中试。

而且云厂商不会只让你租算力。它顺着往上卖：托管数据库、大数据分析、机器学习和 AI 服务、各种现成的托管组件。你用得越多、绑得越深，每一层都在往锁上加一道扣。操作系统锁的是「你在哪个系统上装软件」，云锁的是「你整个公司的命根子跑在谁家机房里」。这就是为什么云的切换成本，比当年换操作系统高出一个量级。

三家的牌桌：门票是每年几百亿美元

今天的云是一场三巨头的游戏。

AWS 先发，2006 年起步，领先对手好几年，长期稳坐市占第一。先发优势在这门生意里格外值钱——它最早开始砸钱建规模，也就最早享受到规模带来的成本优势和飞轮。

微软 Azure（Azure，微软的云平台，2010 年商用）紧追在后。它没有和 AWS 硬拼算力单价，而是打自己最擅长的牌：企业关系。全世界的大公司早就在用 Windows、用 Office、用微软的各种企业软件，微软顺势对它们说——你们那些搬不上云的老系统，就留在自家机房，新业务放上 Azure，两头打通，这叫混合云 (Hybrid Cloud)（一部分在公有云、一部分留在自己机房，两者连成一体）。这一招精准地接住了「不敢一步搬空」的大企业，也正是第十四章要讲的微软重生故事的一部分。

Google Cloud 排第三，技术底子厚，尤其在数据和 AI 上有独到之处，但入场晚、企业关系不如微软深，追赶得更吃力。

值得停下来看一眼的，是这三家的进入壁垒。要坐上这张牌桌，你得每年拿出几百亿甚至上千亿美元去建数据中心、买芯片、拉电力——这已经不是「有个好点子」能进的门了。一家再聪明的创业公司，也无法凭一段绝妙的代码复制出一片覆盖全球的机房网络。资本本身，成了壁垒。

护城河从「技术秘密」变成了「资产负债表」

回到贯穿全书的问题：护城河到底从哪来？

前面十二章，我们看到的护城河大多建在「聪明」上——聪明的标准控制、聪明的网络效应、聪明的商业模式迁移。云给出了一个不一样的答案：**最深的护城河，往往不在「聪明的代码」，而在「别人复制不起的规模与资本」。**

当一门生意的门票，是每年几百亿美元的机房投入时，护城河就从「技术秘密」变成了「资产负债表」。你没法偷走它，因为它不是一段能被抄的代码；你也没法绕过它，因为绕过它意味着你也得先砸下同样的千亿资本。这是一种笨重的、却极其结实的深——它不怕被人想到，只怕对手也有钱。

大白话

一句话记住这一章：亚马逊把「自家闲置的成本」翻转成「对外收租的利润」，靠规模压出「越大越便宜、越便宜越大」的飞轮，再用极高的搬家成本把客户锁死。云之所以比操作系统更深，是因为它的护城河不是靠聪明，而是靠钱——多到别人复制不起的钱。

但请记住全书反复出现的那句话：再深的护城河，也怕脚下的地基移动。云是建在「算力像水电一样按用量卖」这个前提上的。当下一场范式转移——比如 AI 对算力的重新定义——到来时，这套资本壁垒会继续加固，还是会被重新洗牌？这正是第十五章要追问的。

第十四章 · 重生者的共同点：苹果、微软、IBM、Adobe 为什么能转身

前面十三章，倒下的公司排成了一长队：Netscape、Lotus、WordPerfect、Borland、Sun、Novell、Yahoo、黑莓、Palm……它们的死法各不相同，但你把它们并排一放，会闻到一股相同的味道——它们几乎都是在**拼命守住一条正在干涸的护城河**时被淹死的。守桌面份额、守协议、守门户、守键盘手机，守到最后一刻。

可同一片战场上，也有几家公司崩过、濒死过，却又活了过来，甚至比崩之前还高。这一章不讲它们的完整传记，只问一个问题：

> 为什么绝大多数巨头一崩就再也起不来，偏偏苹果、微软、IBM、Adobe 能转身？它们在最要命的那个路口，做对的是同一件什么事？

先把答案摆在桌上，后面再拆：**死掉的公司多半在保卫旧护城河；活过来的公司，主动放弃旧护城河，把自己整个搬到了新的价值层。** 我们看四个"转身的瞬间"，每个只讲那一个关键决定。

IBM：亲手革掉自己硬件的命

1993 年的 IBM（那个曾经定义了整个计算机行业的蓝色巨人）差点死了。它的看家生意是大型机 mainframe——一屋子那么大、卖给银行政府的超级贵重计算机。可 PC 和小型服务器越来越强，客户不再需要一屋子的巨兽，大型机需求萎缩，IBM 一年亏掉近八十亿美元，是当时美国公司史上最惨的亏损之一。华尔街的普遍建议是：把这头大象大卸八块分开卖。

新 CEO 郭士纳 Lou Gerstner（一个卖饼干和信用卡出身、完全不懂技术的外来者）上任后做了个反直觉的决定：不拆，但要换活法。他看清了一件事——客户真正头疼的从来不是"缺一台机器"，而是"这一堆乱七八糟的机器和软件怎么拼到一起、怎么让它们跑通"。

大白话

打个比方：过去 IBM 是卖砖头的，砖头卖不动了。它没有去拼命把砖头做得更便宜更漂亮，而是改行当包工头——你别管砖头哪家的，把盖房子这整件麻烦事全包给我。砖头利润薄了没关系，盖房子的活儿又多又长久。

于是 IBM 把公司重心从"卖机器"搬到"卖服务与解决方案"：帮企业设计、集成、运维整套系统。这等于亲手承认自己最赚钱的硬件正在贬值，主动把重心挪到附加值更高的那一层。后来它一路往软件、咨询、云再迁，甚至把赚钱的 PC 业务卖给了联想。郭士纳有本书就叫《谁说大象不能跳舞》——重点不在大象会跳舞，而在**它敢先把脚下那块最熟悉的地板拆掉。**

苹果：承认桌面战争已经输了，去别处赢

1997 年的 苹果 Apple 离破产只剩几个月现金。它在干一件今天看很悲壮的事：在个人电脑这块战场上，和 Windows 死磕桌面份额（这场仗为什么必输，第四章已经讲透——生态飞轮转不起来）。磕了十几年，越磕越少。

乔布斯回归后的动作，粗看是止血：砍掉大量乱七八糟的产品线，只留几款；甚至接受了老对手**微软 1.5 亿美元的投资**——当众和宿敌握手，就为了活下去。但真正决定苹果命运的，是他悄悄放弃的那个执念：

> 他不再纠缠"怎么在桌面上打赢 Windows"。桌面这场仗，苹果输了，认了。要赢，得换一张全新的牌桌。

于是苹果跳到了一个当时还不存在的价值层：随身的数字生活。2001 年 iPod + iTunes，2007 年 iPhone，再到 App Store。注意——苹果并没有发明什么全新的护城河机制，它用的还是第四章、第十一章那套**开发者越多越好用、越好用开发者越多**的生态飞轮，只不过这一次，是在移动这个全新范式里从头搭。旧战场认输，新战场重开——这才是苹果活过来的真相。

大白话

好比一个在拳击台上被打惨的选手，没有继续在拳击规则里死撑，而是转身跑去了刚发明出来的综合格斗擂台——那里的规则对他有利，对方的经验全部作废。

微软：亲手拆掉自己最神圣的护城河

微软 Microsoft 的护城河，几十年就是四个字：**Windows 至上**。整个公司的一切都要先问"这对 Windows 好不好"。这条护城河太深、太赚钱，深到成了枷锁——正因为一切都要护着 Windows，它错过了搜索、错过了移动（第九章讲过，这不是眼瞎，是被自己最赚钱的业务困住的"创新者的窘境"）。

2014 年 萨提亚·纳德拉 Satya Nadella 接任 CEO，做了件在老微软看来近乎叛教的事：公开放下 Windows 的执念。他把战略重心整个搬到两处新地基——

- **云（Azure，微软的云计算平台）。** 第十三章讲过，云是比操作系统更深的地基。微软押上全力去做出租算力的生意。
- **订阅（Office 365，把 Office 从买断改成按月租）。** 正是第十二章那一跳。

最能说明问题的是几个"背叛旧护城河"的动作：微软开始拥抱**开源**（曾经把开源骂成"癌症"的公司，转头买下了全球最大的开源代码仓库 GitHub）；还把 Office 认认真真做到了苹果的 iOS 和谷歌的 Android 上——等于承认"用户在哪台设备上，我就伺候到哪，哪怕那台设备不是 Windows"。

> 这是主动把自己最神圣、最赚钱的护城河拆开一个口子，换到"云"这块更深、更长久的地基上。不是 Windows 不重要了，是纳德拉看清了：死守它，会输掉未来的全部。

Adobe：主动砸烂自己几十年的赚钱结构

第十二章已经细讲过 Adobe（Photoshop 那家）的这一跳，这里只从"重生"的角度点一句。2013 年，它把王牌软件全部从"一次买断、永久归你"改成"只租不卖、按月订阅"（Creative Cloud）。

这一跳的狠，在于它砸烂的是一个**已经赚了几十年、当下还在赚钱的收入结构**。转型头两年账面难看、老用户骂街，需要极大定力硬扛。但它换来的是更稳、可复利、还把用户数据牢牢沉淀在云端的新护城河。**用几年的短痛，换十年的长稳。**

把四家并排：重生者的四条共同机制

四个故事讲完，真正值钱的部分来了。把它们和前面那一长队死掉的公司对照，"重生"其实是同一套动作，可以拆成四条清清楚楚的机制。

一、在旧护城河还没干涸时就跳

IBM 是被逼到亏损八十亿才跳，其实已经算晚的；但更多重生者，是在旧生意还在赚钱、外人看不出问题时就动手了——Adobe 砸的是正在赚钱的买断业务，纳德拉动手时 Windows 还如日中天。**等护城河真干了再跳，往往已经没水行船。** 对照第九、十一章：黑莓、诺基亚、Yahoo 不是没看见新趋势，是舍不得在旧河还满的时候离岸，非要等船搁浅才动，那时肌肉已经僵了。

二、愿意自我蚕食

这是最反人性的一条。自我蚕食 self-cannibalization——说白了就是**亲手抢自己现在的饭碗**：主动削减、甚至砍掉自己此刻正在赚钱的业务，去喂一个将来才赚钱的新业务。

> 大白话
>
> 就像一家生意火爆的老馆子，在还天天满座的时候，自己开一家会抢走老店客人的新店。多数老板下不了这个手——"我干嘛砸自己招牌？"于是等隔壁那家新店把客人抢光，才追悔莫及。

Adobe 砍掉买断收入喂订阅，微软把 Office 送到对手的手机上抢自己 Windows 的独占性，IBM 卖掉 PC 业务、削自己的硬件——全是亲手抢自己饭碗。而崩掉的公司，恰恰是**舍不得**：Yahoo 舍不得门户流量的老广告，黑莓舍不得那块引以为傲的实体键盘。舍不得，就是重力。

三、认清价值转移到了哪一层，然后把公司搬过去

这四家的转身，路线其实是同一个句式——**价值从这一层流走了，就把公司整个搬到它流去的那一层**：

- IBM：硬件 → 服务
- 苹果：桌面 → 移动
- Adobe：买断 → 订阅
- 微软：软件 → 云

关键词是**"搬过去"，而不是"在旧层里做到更优"**。这正是死掉那批公司的共同幻觉：它们以为只要把旧东西做到极致就能翻盘——把桌面软件做到功能最全（第五章的 Lotus、WordPerfect），把实体键盘手机做到手感最好（第十一章的黑莓）。可价值已经不在那一层了，你在一层空气里做到最优，也还是空气。

四、常常需要换一次领导层，才能违背组织自身的重力

最后一条很微妙。一个组织有它自己的"重力"：全公司的奖金、晋升、荣誉，都是围着旧护城河建起来的，让所有人本能地去护它。要一个亲手建了这条河的人自己下令拆河，太难了。

所以你会发现，四次转身背后几乎都站着一个"换过的脑子"：IBM 请来个完全不懂技术的外人郭士纳，苹果请回被自己赶走的乔布斯，微软换上纳德拉。**常常需要一次领导层或心智的更换，才有人敢做违背组织重力的那个决定。**

收进一句话：护城河不是用来守的，是用来换的

走完这一圈，全书那条主线可以收束掉一半了。前面十三章反复证明：没有一条护城河能永不干涸——标准会被绕过，平台会被更替，商业模式会被改写，脚下的地基自己会移动。既然如此，把宝押在"挖一条永远不干的沟"上，本身就是错的方向。

> 真正的能力，不是挖一条永不干涸的沟（那不存在），而是在旧沟干涸之前，有勇气、有判断、也有那个敢拆自己的领导者，把自己搬到下一条河边。**护城河不是用来"守"的，是用来"换"的。**

死掉的公司，把护城河当成了要死守的城墙；活过来的公司，把它当成了一块可以随时拆掉、搬去下一处的临时营地。这就是重生者和殉葬者之间，那道最深的分水岭。

那么问题来了：当我们把这套"起、崩、生"的规律都看清之后，面对眼前这场最新、也最凶猛的浪潮——AI，谁会登顶、谁会塌方？下一章，我们用全书攒下的这把尺子，去量一量新的战场（第十五章）。

第十五章 · AI 平台战争：新一轮标准与护城河之争

前面十四章，我们一直在做同一件事：解剖已经发生的事故。谁登顶了、谁塌方了、谁又活过来了——都是回头看，答案早就写在结局里。这一章不一样。

2022 年底，一个叫 ChatGPT 的聊天框上线，几天内涌进上亿人。所有人突然意识到：又一次平台级的地震开始了。而这一次，我们还站在震中，不知道尘埃会落在哪里。

所以这一章要老实说在前头：**下面全是推演，不是事实。**我不知道谁会赢。我要做的，是把前面十四章提炼出来的那几把尺子，一把把架到今天的 AI 战场上，看它们量出什么。就像老水手不靠预知风暴，而靠一张读了一辈子的旧海图，来判断眼前这片没走过的水域深浅。

> 大白话
>
> 说人话：这不是算命，是"用旧地图找新坐标"。历史不会重复，但会押韵。我们盯的不是"谁赢"，而是"该盯哪几个变量"——盯对了，你自己就能判断谁在挖护城河、谁的护城河在被抽干。可能押错，但方法是清楚的。

先认识几个新名词，都讲成人话

这场战争的载体有几样东西，先各用一句大白话讲清楚：

- 大语言模型 (LLM)：吃下海量文本训练出来的 AI，能对话、能写代码、能总结。它是这一轮新应用真正的"发动机"。
- API (应用程序接口)：一个软件对外开的"插座"。别的程序不用懂模型内部怎么运转，插上这个口，付点钱，就能调用模型的能力。今天成千上万的 AI 应用，底下都是在往某个模型的 API 上插电。
- GPU (图形处理器)：一种原本用来渲染游戏画面、特别擅长同时算海量简单运算的芯片。训练和运行大模型，靠的就是它。
- CUDA：Nvidia 给自家 GPU 配的一套软件工具。全世界搞 AI 的人，多年积累的代码几乎都是照着 CUDA 写的——换别家芯片，这些代码就得推倒重来。

名词认全了，我们开始把旧尺子一把把架上去。

主线一：模型是不是新的"操作系统"？

回到第三章那条最硬的规律——**谁掌握标准，谁收税**。当年真正躺赢的不是造 PC 的 IBM，而是握住操作系统这个"所有软件都得来对接的接口"的微软。全世界的软件都写给 Windows，于是全世界都得给微软交过路费。

今天把"操作系统"四个字换成"大模型"，句子几乎原样成立。一个做 AI 客服、AI 写作、AI 编程助手的创业公司，它自己不训练模型，而是调用某家的 API——就像当年的软件公司不自己写操作系统，而是把程序架在 Windows 上。**谁的模型成了开发者"默认伸手就调"的那一个，谁就坐上了第三章那把收税的椅子。**

而决定这件事的，是第四章那条规律：**开发者才是真正的用户。**普通人用的是聊天框，但真正决定平台生死的，是那群把模型缝进自己产品里的工程师。他们一旦习惯了某家的 API、某套调用方式、某种"提示词该怎么写"的手感，迁移就变得很烦——这就是第四章讲的飞轮：调用的人越多 → 教程、工具、踩过的坑越多 → 新来的人越省事 → 越多人调用。

> 大白话
>
> 打个比方：模型公司真正想抢的不是"让你我多聊两句天"，而是"让全世界的程序员，写新功能时下意识地先想到调用我"。这跟当年抢"软件都写给我的系统"是同一件事。聊天框是门面，API 才是收税口。

所以第一个该盯的变量是：**API 调用量和开发者心智，正在往谁那儿聚。**不是看谁家模型这周测评分高——那个变量太吵、天天翻盘；要看谁正在变成"默认调用的那一个"。

主线二：算力才是最稳的收税口——卖铲子的人

但这里有个反直觉的推论。回想第七章和第十三章反复出现的一句话：**占住别人绕不开的基础设施位置，比做那个最耀眼的产品更赚钱、更稳。**

模型公司们打得头破血流，可它们全都得买同一样东西——GPU。而做 AI 训练、运行必需的 GPU，截至本书写作时，主要由一家公司主导：Nvidia。这就是那个古老的比喻：**淘金**

热里最稳赚的，往往不是淘金的人，而是卖铲子、卖牛仔裤的人。金子谁淘到不一定，但只要还有人在挖，铲子就一直有人买。

更狠的是 CUDA。这正是第七章"协议之王"和第三章"标准即收税"的合体：全世界的 AI 代码都是照着 CUDA 写的，换芯片就得重写一遍。这是一道用别人多年心血砌起来的锁——和当年"软件都写给 Windows、所以离不开 Windows"一模一样，只是这次绑的是芯片。

然后云厂商（第十三章）在上面再收一道租：它们把成堆的 GPU 买来，打包成"按小时租用的云端算力"卖给创业公司。创业公司不用自己砸钱建机房，付月租就行——这正是第十三章的"把资本开支变成别人的生意"。于是一层算力，被收了两道税：芯片一道，云一道。

> 大白话
>
> 说人话：如果历史规律成立，那么这一轮真正"旱涝保收"的，可能不是那些名字天天上头条的模型公司，而是站在它们身后卖铲子的芯片商，和把铲子出租的云厂商。模型可以被超越、被开源抽底，但只要战争还在打，铲子和场地就一直有人付钱。

当然，铲子也有风险：如果哪天出现一种"绕开 CUDA、换芯片不用重写"的通用标准，这道锁就会松。所以第二个该盯的变量是：**算力这个收税口，是被一家越锁越死，还是正在被某种开放标准撬松。**

主线三："拥抱、扩展、消灭"会不会重演？

第六章那三步棋——**拥抱、扩展、消灭**——是本书最阴冷的一课：网景做出一个爆款功能，微软把它变成操作系统免费附送的零件，网景就被"白送"抽干了呼吸。

今天几乎能看到同一个剧本的影子。微软大额投资并深度绑定了 OpenAI，然后把模型能力缝进了 Windows、Office、Azure——冠名"Copilot"。你在写文档的地方，AI 助手就已经在那儿了，不用另外找、另外下载、另外付费。这不正是第六章的翻版吗？**上游平台把一个爆款功能，吸进自己现成的分发渠道。**

于是无数独立 AI 应用头上都悬着同一把剑，业内叫它"套壳焦虑"：你辛辛苦苦做出一个 AI 功能，可它随时可能被某个大平台免费附送进它已有的产品里——就像当年网景那个浏览器，随时会变成 Windows 自带的一个零件。你不是被打败，是被"平台顺手白送"这件事本身抽干。

大白话

但第六章还留了两条活路，今天依然管用：一是**监管会不会出手**拦住那只手（美国政府诉微软案给业界立过红线）；二是**跳到平台够不着的那一层**——别只做一个会被吸走的"功能"，去做平台复制不了的东西：一个只有你有的专业数据源、一群离不开你的行业用户、一套别人绕不开的工作流。

所以第三个该盯的变量是：**哪些 AI 功能正在被大平台"缝进渠道"白送掉。**一个独立应用的产品越是"大厂顺手就能加的一个按钮"，它的处境就越像网景。

主线四：开源模型会不会重演第八章？

第八章讲过最痛的一刀：**当"白送还能协作改进"遇上边际成本为零，付费授权的地基就被抽走了。**Linux 白送，抽掉了 Unix 巨头脚下的地基。

今天有 开放权重模型（把训练好的模型参数公开、任何人可下载自建，以 Meta 的 Llama 系列为代表）。这会不会像 Linux 抽掉 Unix 那样，抽掉"闭源模型靠卖 API 收钱"的地基？两面都得推：

像的一面：如果一个足够好用的模型白送、还能被全世界一起改进，那"为调用闭源模型付费"这件事的地基就会松动。很多本来要买 API 的场景，会改用免费自建——正如很多本来要买 Unix 授权的场景，改用了免费的 Linux。开源一旦"够用"，付费方就得不断解释"我凭什么更贵"。

不像的一面：这里有个第八章没有的新变量——**第十三章的资本护城河**。当年开源软件，一台旧电脑、几个志愿者就能编译运行；而训练一个顶尖大模型，要烧掉天文数字的算力和电费，是普通人和小团队根本砸不起的门槛。所以"模型开放权重"不完全等于"人人能造模型"：配方也许公开了，但那口能炼出配方的炉子贵得吓人。这意味着开源可能抽掉"卖现成模型 API"这层地基，却抽不掉"谁掏得起钱训练下一代最强模型"这层——后者反而更集中在少数巨头手里。

大白话

说人话：如果历史规律成立，那么开源模型很可能像 Linux 一样，把"卖标准化模型"这门生意的利润薄薄地刮平——够用的能力会越来越便宜、越来越白菜价。但它未必能撼动"谁烧得起钱训练最前沿模型"这个由资本堆出来的高地。开源吃掉的是中间层，不是最顶端。

所以第四个该盯的变量是：**开源模型把地基抽到了哪条线。**"够用的能力"被抽成免费白菜的那条线，会一年年往上抬——它抬到哪，闭源卖 API 的生意就得退到哪之上。

主线五：谁会掉进创新者的窘境？谁在主动自我蚕食？

第九章那条最让人后背发凉的规律：**杀死巨头的不是愚蠢，是一次算得清清楚楚的、正确的计算。**高利润的老业务，会让在位者理性地舍不得革自己的命。

今天最标准的一个"窘境样本"，是搜索广告。截至本书写作时，Google 的现金牛仍是搜索广告——你搜一个东西，它给你一串链接，链接旁边是广告，你点了它就赚钱。可 AI 直接把答案讲给你听，你还需要点那串链接吗？**AI 直接回答，恰好侵蚀的正是"点链接、看广告"这个动作本身。**

这就是教科书级的窘境：一家公司手握最强的 AI 能力（Google 有 Gemini），却也手握一头靠"你必须点链接"吃饭的现金牛。全力推 AI 直答，等于亲手拆自己最赚钱的收费站。用它今天手里那把尺子量，"慢一点、别太狠地革自己命"是财务上最理性的决定——而第九章告诉我们，正是这种理性，最容易让在位者掉进坑里。

反过来，第十四章讲的"重生者"，做对的恰恰是相反的事：**主动自我蚕食，放弃保护旧护城河，把自己迁到新的价值层。**谁敢在老业务还在赚钱时，就亲手用新东西去替换它——哪怕短期难看——谁就更像那批活过来的公司。这一轮里，越是把 AI 直接摆到会冲击自己老收入的位置上、宁可先革自己命的玩家，越值得高看一眼。

大白话

说人话：这里有一个残酷的对称。手里现金牛越肥的在位者，越理性地下不去手革自己的命（第九章）；而恰恰是敢自断一臂、先替自己造好下一条命的公司，才可能重生（第十四章）。所以别只看谁 AI 做得炫，要看**谁敢拿 AI 去砸自己现在最赚钱的那口锅。**

所以第五个该盯的变量是：**谁的老现金牛正好被 AI 顶在刀口上，以及他敢不敢自己动这一刀。**下不去手的，可能正在重演第九章；先动刀的，可能正在走第十四章的路。

落点：一副看懂这场战争的仪表盘

这一章我不会告诉你谁会赢——诚实地说，我也不知道。历史规律能帮你缩小范围、看清结构，但它给的是概率，不是名单。任何拍着胸脯说"某某一定登顶"的人，要么在卖故事，要么忘了前十四章里死掉的那些"看起来必胜"的公司。

我能给你的，是一副仪表盘。盯住这五个表针，你就能自己判断谁在挖护城河、谁的护城河在被抽干：

1. **开发者默认调用的是谁**（标准即收税 + 开发者飞轮）——API 调用和开发者心智往谁那儿聚。
2. **算力这个收税口锁在谁手里**（卖铲子 + 资本护城河）——芯片和云的锁，是越锁越死还是被开放标准撬松。
3. **哪些 AI 功能正被大平台缝进渠道白送**（拥抱扩展消灭）——你的产品越像"大厂顺手一个按钮"，越危险。
4. **开源模型把地基抽到了哪条线**（开源吃商业软件 + 资本护城河）——"够用即免费"那条线每年抬多高。
5. **谁的现金牛正被 AI 顶在刀口上，他敢不敢自己动刀**（创新者窘境 vs 主动重生）。

再说一遍：以上全是用旧海图对新水域做的推演，可能错。但方法本身是可靠的——它是从十四场真实的沉船和重生里捞出来的。风暴的样子每次都新，可让船沉、让船浮的那几条水文规律，一直没变。

> 看懂一场平台战争，靠的不是猜谁赢，而是认得那几个决定输赢的变量——谁握住了开发者、谁占住了收税口、谁敢自我蚕食、开源把地基抽到了哪、平台又把哪些功能吸走了。认得这几样，你就有了自己的判断，而不必等结局揭晓。

下一章，也就是全书最后一章，我们把这十五章里反复出现的那几条机制，收束成一张可以随身带走的"因果地图"——回答我们从第一页就一直在问的那个问题：护城河到底从哪里

来，又为什么会塌。

第十六章 · 起伏的底层规律：一张软件兴衰的因果地图

走到这里，我们已经翻过十五章：从 IBM 把软件当赠品，到微软握住标准收税，到 Netscape 一夜蒸发，到黑莓输给一块玻璃，再到苹果和微软死而复生。故事各不相同，但如果把这些故事像地图一样叠在一起，透过纸张看，会发现底下是同一批线条在重复走。这一章不讲新公司，只做一件事：**把散落在全书的机制，收成几条你能带走、能自己拿去分析任何一家软件公司的定律。**

贯穿全书的问题只有一个：**护城河从哪来，又为什么会塌？**现在我们正面回答它。

第一部分 · 护城河从哪来：登顶的六种力量

所谓护城河，就是让对手即使做出一样好的产品也抢不走你客户的那道结构性障碍。软件行业的护城河，来来去去就是下面这六种。每一种我们只用一句话定义、点一个前文的例子、说清它什么时候成立。

一、成本结构：边际成本近零，天生赢家通吃

软件写一份和写一千万份，几乎一个价（第一章）。这意味着一旦你领先，你能把成本摊到最大的分母上，别人永远比你贵。这是所有其它护城河的地基——赢家通吃不是谁狠，是成本曲线注定的。**适用条件**：产品能无损复制、分发近乎免费。

二、控制标准与接口：这是收税权

谁定义了别人必须对接的那个接口，谁就在整条价值链上收税。微软握住 PC 操作系统标准，造机器的 IBM 反而给它打工（第三章）；今天 Nvidia 用 CUDA 把 AI 算力的接口攥在手里（第十五章）。**适用条件**：存在一个大家都必须穿过的"卡口"，而你拥有它。

三、网络效应与开发者飞轮：越多人用越好用

网络效应就是每多一个用户，产品对其他所有人都更值钱。软件里它常以"开发者飞轮"出现：应用越多平台越好用，越好用越多人写应用（第四章的 Windows、第十一章的 App

Store）。飞轮一旦转起来，对手要追的不是你的产品，是你整个生态。**适用条件**：用户之间、或用户与开发者之间存在互相吸引。

四、规模与资本：把门槛堆到别人进不来

有些护城河靠的是别人烧不起的钱和攒不出的数据。搜索用得越多，数据越多，结果越准，于是更多人用（第十章的 Google 数据飞轮）；云需要天文数字的资本开支才能建起来（第十三章）。**适用条件**：规模本身直接转化成产品优势或成本优势。

五、切换成本与锁定：走不动比留下来更贵

当客户把身家绑在你身上——数据、流程、习惯都长在你的系统里——离开的代价高到他宁可忍。Oracle 数据库一旦装进核心业务就很难拔（第七章），深度使用的云同理（第十三章）。**适用条件**：迁移要付出高昂的时间、风险或重写成本。

六、商业模式的地基：钱从哪来决定你能守多久

护城河也可以建在收钱的方式上。Google 用"免费产品 + 广告拍卖"把竞争对手的收费软件变成没人愿意付钱的东西（第十章）；SaaS 用订阅把一锤子买卖变成年复一年的经常性收入，让公司的现金流稳如租金（第十二章）。**适用条件**：你的赚钱方式让对手要么补贴不起，要么估值跟不上。

大白话

说人话：登顶的六条路，其实是同一句话的六个版本——**让别人即使追平了产品，也追不平你所处的结构**。要么你更便宜，要么你在卡口上，要么你身后站着整个生态，要么客户走不掉，要么你收钱的姿势别人学不来。

第二部分 · 护城河为什么会塌：崩塌的五种机制

诡异的是，让公司登顶的力量，往往正是让它塌方的力量。护城河不会因为"对手更努力"而消失——那种你能看见、能应战。真正要命的塌方，都来自你防不住的方向。

一、被上游平台吸收：你的产品变成人家的免费附件

当你的整个业务，在更大的平台眼里只是"一个功能"，它就会把这个功能免费塞进系统里送掉。Netscape 靠免费浏览器登顶，又被微软"操作系统里白送 IE"绞死（第六章）。你不是败给了更好的浏览器，你是败给了"浏览器不该单独收钱"这件事。**信号**：你的核心产品正在变成别人产品的一个勾选项。

二、地基被开源抽干：你没法跟 0 元报价竞争

软件边际成本本就近零，一旦有人把它白送、还越送越好，付费授权的地基就被抽走了。Linux 和 Apache 掀翻了卖 Unix 授权的巨头，Novell 守着标准却守不住价格（第八章）。**信号**：你卖的东西，有一个"够用的免费版"正在逼近。

三、平台转移：脚下的地基整块移走，旧优势瞬间归零

平台转移就是承载一切的底座换了一代（PC 换到手机）。黑莓和诺基亚握着旧世界最好的产品，却发现旧世界本身不见了（第十一章）。他们没做错产品，是站错了地基。**信号**：一个新品类正在重新定义"用户到底想要什么"，而你的所有优势都长在旧品类上。

四、创新者的窘境：现金牛让你"理性地"不转身

这是最反直觉的一条。新技术刚冒头时又小又差，转过去会伤害你正赚大钱的主业——于是每一个季度，最聪明的管理层都会做出"再等等"的理性决定，直到来不及（第九章）。杀死你的不是愚蠢，是**短期看完全正确的算账方式**。**信号**：你能说出十个"现在不做那个新东西"的好理由，而且每个都对。

五、护城河成了枷锁：越成功越舍不得自我蚕食

前四条最后都汇到这一条。护城河越深，你越不愿意跳出去，因为跳出去就等于亲手放干那条护得你舒舒服服的河（第九、十一章）。**护城河的本质，是让你不想离开——顺境里这是优点，地基移动时这就是致命伤。**

大白话

说人话：护城河塌，很少是被正面攻破的，多半是**你脚下的地基自己走了，而护城河让你舍不得跟着走**。同一样东西，先是你的靠山，后是你的锚。

第三部分 · 为什么有的能重生

既然塌方常因"舍不得离开"，那重生的秘密就呼之欲出了（第十四章）：**护城河是用来"换"的，不是用来"守"的。**

苹果、微软、IBM、Adobe 都崩过又活过来，他们做对的是同一件事——在旧沟还没干涸时，主动跳进新沟。把这些重生者摆在一起看，共同点很干净：

- **在旧护城河干涸之前主动跳**，而不是等它见底了才被迫动——那时已经没有资源跳了。
- **敢自我蚕食**：主动用新业务打自己的现金牛，Adobe 冒着营收暴跌的风险把买断改成订阅（第十二章）就是活教材。
- **认清价值转移到了哪一层**：微软从"卖 Windows 授权"迁到"卖云"（第十三章），是因为它看懂了收税口从操作系统挪到了算力。
- **常常要换脑子、甚至换人**：因为守旧护城河的那套本能，恰恰是转身最大的阻力，往往得靠新的领导者才能狠下心。

重生不是运气，是一种反本能的纪律：**在自己还强的时候，承认强的那一层正在贬值。**

第四部分 · 一张对照表，和一句总规律

把全书压成一张能带走的对照表：

1. **造就护城河的力量**：近零成本 → 控制标准 → 网络效应 → 规模资本 → 切换成本 → 商业模式。
2. **让护城河塌方的力量**：被平台吸收 → 被开源抽干 → 平台转移 → 创新者窘境 → 护城河变枷锁。

看懂这张表，你会发现左右两列其实咬在一起：**每一种护城河，都对应着一种特定的塌法。**靠标准的怕平台转移；靠付费授权的怕开源；靠现金牛的怕自己不肯转身。护城河越深，对

应的那种塌法就越致命——因为它让你越发舍不得离开。这就是书名"护城河与塌方"要说的那件事：**它们不是两样东西，是同一样东西的两面。**

如果只带走一句话，就带这句：

> 别问一家软件公司现在多强，问三件事——它的护城河建在哪一层，那一层的地基正在往哪移，以及它敢不敢在地基移动之前先跳。

前两问看清现状与趋势，第三问看清命运。因为软件公司的兴衰，说到底不是强弱之争，是**愿不愿意在还强的时候先放手**之争。护城河永远会有；地基永远会移；能穿越几代人的，从来不是护城河最深的那家，而是最舍得换河的那家。

地图叠完了。往后你再看任何一家软件公司——包括这一刻风头正劲的 AI 巨头——都可以把它放到这张图上：它站在哪条河边，河水正流向哪里，以及它握着的桨，是用来守，还是用来跳。