The Founder's Playbook
创始人行动手册

Claude

Anthropic · 2026 年 5 月

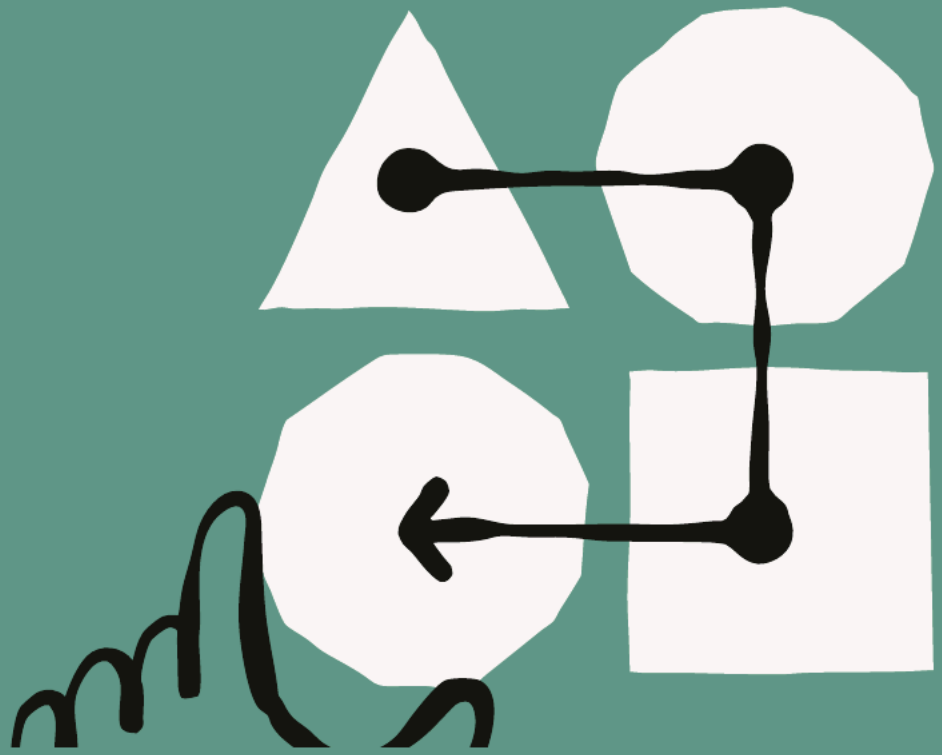
Building an AI-Native Startup

花叔 译

横版 36 页中文译本

仅供个人学习与内部研究

目录



Chapter 1

创业生命周期，为 2026 重新启动

Chapter 1

创业生命周期，为 2026 重新启动

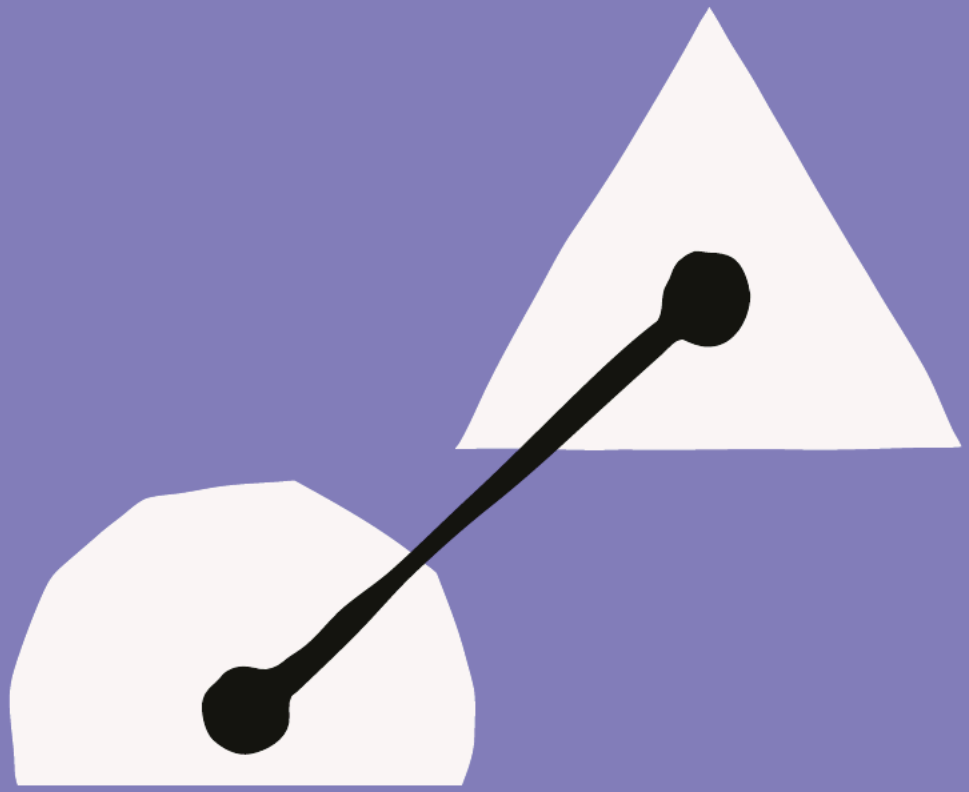
AI 正在重塑创业公司的搭建方式。今天，从未写过一行代码的创始人正在把生产级应用交付给用户，「10 人独角兽」也从草根逆袭故事变成了一份可以认真执行的行动方案。

到 2026 年，AI 可以写生产环境代码、做市场调研、梳理竞争格局、起草投资人材料，并把运营流程自动化跑起来。过去那条很陡的学习曲线被它抹平了。即便是经验丰富的技术创始人，以前也得花大量时间去整合工具、平台和系统，才能把想法落地。AI 最大的贡献，是把「谁能开一家创业公司、谁能做出一款产品」这件事拉到了同一条起跑线上。

2026 年，一个好想法能把创始人带得比以往任何时候都更远。Agentic 编程（agentic coding）把过去需要一整支工程团队才能完成的活，压缩成了创始人一个人就能交付的体量。

传统的创业增长曲线，默认路径是：验证 → 融资 → 招人 → 搭建 → 再融资 → 增长 → 继续招人 → 循环往复。现在，AI 已经打破了一个默认预期：创业生命周期里每进入一个新阶段，就必须配上更大的团队、不同的技能组合和新一轮融资。

这本手册会按照这些新现实，重新画出创业旅程的四个核心阶段：想法、MVP、发布、规模化。我们会逐阶段来看：当 AI 成为技术和组织发展的核心基础设施时，每个阶段长什么样、该用哪些工具，以及走在前面的创始人如何用这些工具把时间线压短。如果你想规划从想法到 exit（退出，指被收购或上市）的最短路径，请继续往下读。



Chapter 2

「创始人」这件事正在改变

Chapter 2

「创始人」这件事正在改变

过去，创始人是被「能做什么」定义的：技术型创始人写代码，非技术型创始人跑业务、谈合同。但 2026 年创始人手上的模型、系统和 AI agent，已经把「能造东西的人」和「有想法值得造的人」之间那堵墙拆掉了。

AI-Native 创业公司正在从根上改变「当创始人」这件事的含义。现在，一个完全没有工程背景的人，能做出真正能跑起来的生产级软件，把自己的想法落地；而一个技术很强但商业上经验不多的创始人，也能轻松产出 GTM 策略、财务模型和高度打磨的 pitch deck（融资演示文稿）。

过去，创始人大部分时间都在执行模式里：写代码、管人、处理日常运营。在 AI-Native 创业公司里，创始人不再只是个人贡献者，而更像是一群 agent 的编排者（orchestrator）。这些 agent 是专门化的 AI 助手，能读文件、跑命令、执行代码，甚至浏览网页。创始人的注意力开始往上走，转向更高阶的工作：产生想法，然后指挥这些 AI agent、工具和小团队把想法跑出来。

AI 作为核心基础设施最革命性的影响，是把那些有领域专长但没有工程背景的创始人也解锁了出来。当创始人这个群体不再只来自工程背景，你会看到由完全不同人生经验的人创办的公司，去解决传统技术创始人通道从未优先关注，甚至从未注意到的真实问题。

AI 为精益创业带来的能力

传统创业模型假定你必须招工程师来造、招销售来卖、招运营来跑公司。「人头数」被当作组织势能和产品成熟度的信号。

2026 年的早期创业公司则完全不一样。它们从设计上就极度精益，往往只有创始人一个人，或者再加上少数几位伙伴。把技术和组织发展都建立在「AI 即基础设施」之上，它们可以在扩张团队之前，就跑通产品验证、做出早期收入，甚至实现盈利。AI 在三个地方能让一家创业公司像大得多的组织一样运转：研究、Agentic 编程，以及把关键业务运营做成工作流自动化。

对话式智能与研究

类比：任何领域随叫随到的专家

想一下，一个创始人在头一年里几乎肯定不懂但又必须搞清楚的那些事：工资系统怎么搭？产品开发的冲刺怎么排？一份紧凑的投资人备忘录怎么写？

早期创业的这类问题，过去答案几乎都一样：找个懂的人。对一个自筹资金或种子轮前的创始人来说，这意味着把本该用来搭东西的时间花在打听上，或者把早期资金烧一截给顾问。现在，他们手里就有一个几乎覆盖所有领域、随时在岗的专家：AI。

- **深度研究：**竞品分析、市场规模测算、财务建模
- **文档起草：**pitch deck、案例研究、投资人备忘录、PRD（产品需求文档）
- **战略思考伙伴：**反方代言人分析、事前剖析（pre-mortem）、情境推演、路线图优化

Agentic 编程

类比：那位永远在线、永不被卡住的工程师

过去要做出软件，要么得有个技术合伙人，要么外包给开发公司，要么得有足够长的跑道，先把一支工程团队招齐，才能写下第一行生产代码。

Agentic 编程工具现在让每一个有想法的创始人，都能用大白话描述自己想做什么，再指挥 AI 去生成、测试、调试、重构一份生产级代码库，速度和体量都不输一支完整的工程团队。

从「我有一个想法」到「我有一个产品」之间的时间线被压短了。创始人现在聚焦的是「该做什么、为什么做」，而 AI 负责把面向真实用户的那部分基础设施真正搭起来。

工作流自动化

类比：一支随叫随到的自动化运营团队

就算创始人能像顾问一样做研究、像工程团队一样写代码，战略规划和产品开发之外还有一整类活必须有人来做：排期、更新 CRM、拉周报、维护文档、发布内容、跟进合规要求、把公司赖以运转的工具和系统之间的连接管起来。这些事一样要发生。在精益创业公司里，这些活主要落在创始人自己头上，会大量吃掉本该用于高阶判断的时间和注意力。

AI 工具的工作流自动化能把这笔税卸掉。重复性的运营任务可以配置成自动跑：deal 状态一变，CRM 就自动更新；周报自己汇总；产品文档跟着产品改动同步更新。关键的一点是，Claude Cowork 能直接集成创业公司用的那一整套互联系统，比如项目管理工具、沟通栈、数据源，不需要再有人专门去搭和维护这些集成。在第零天创业公司里，那个人几乎总是创始人本人。

时机和编排，是一切

能把 AI 的研究、自动化和 agentic 编程能力真正用起来的创始人，可以用远超团队规模所暗示的杠杆来跑一家公司。他们也能把绝大部分时间和带宽，留给那些真正重要的工作。

但这套东西不会自己跑。负责编排这些 AI 工具的创始人，得知道怎么用、什么时候用。本手册接下来的部分，就是逐阶段拆解 AI–Native 创业路径上的目标和挑战，以及在每个阶段如何把 AI 工具用到位。

Chapter 3

想法阶段

Chapter 3

想法阶段

每一位创始人都从同一个地方出发：一个让你停不下来去想的问题。这是创业里想法撞上现实的阶段。2026 年的创业成功，要求你具备一种纪律：证据不充分之前，不动手造。

这一阶段的工作是研究、客户发现、竞品分析，以及对反证（disconfirming evidence）的诚实评估。所有这些都要发生在你让 Claude Code 写下第一行生产环境代码之前。

想法阶段·目标

在想法阶段，创始人的核心目标是以研究为导向的验证：在投入资源开始造之前，攒起扎实证据，证明一个真实问题确实存在，并且你提出的方案确实能解决它。

实务上，想法阶段就是一系列问题，大致按这个顺序回答：

- 这个问题真实、具体、足够高频，值得围绕它做一家公司吗?
- 到底谁有这个问题？这些人能构成一个市场吗?
- 有没有人在解决？如果有，他们怎么做的，做得多好?
- 一个真正解决这个问题的方案需要做到什么？我的想法做到了吗?

这些问询加起来，回答一个终极问题：这值不值得做?

所以说，先具体，再行动。「大家做报销很头痛」是观察；「中型公司的财务经理每周要花 4 个小时以上核对报销提交，因为现有工具不和会计软件打通」才是一个可被测试的假设。

想法阶段·退出标准

想法阶段的退出条件，是找到问题–方案匹配（problem–solution fit）。你要在开始造解决方案之前，建立起质性证据，主要来自真实的人类对话，证明你正在为真实的人解决真实的问题。

当你能对以下三个问题都回答「是」，就可以离开想法阶段了：

1. **这个问题真实且具体吗?** 你必须能精确说出谁经历这个问题、多频繁、影响多严重、目前他们怎么处理。
2. **你的方案对应的是真实问题吗?** 不是你最初假设的问题，而是验证过程里浮现出来的那一个。两者有时相同，但并不总是。
3. **你有足够信号去开干吗?** 这个阶段你永远不会有 100% 确定，而一味等待确定本身就是一种失败模式。但你需要足够多的质性证据，让投入做 MVP 看起来是有理有据的决定，而不是一次信仰行动。

想法阶段·挑战

想法阶段是创业旅程里最重要的工作发生的地方，因为最致命的错误就在这里酿成。现在搞错一点，能很快把刚起步的事业带偏。多数想法阶段的挑战，归根到底都是前进速度超过了理解所能支撑的范围。带着思考和审慎前行的创始人，才能稳步推进。

把「造」误当作「验证」

挑战：当技术障碍被移除，激情冲昏头的创始人很可能跳过创业旅程里最重要的工作：验证自己的想法是不是人们真正需要、也真正会用的方案。

即便在当下的 agentic 编程（agentic coding）时代之前，也有 42% 的创业公司失败，是因为做了没人想要的东西。如今像 Claude Code 这样的 agentic 编程方案已经把「我有一个想法」到「我有一个产品」之间的距离大幅压缩，这个失败率只会继续往上走。

现在确实是怀揣绝妙点子的创业者最好的时代。但一个看起来像产品的原型可以如此快速、轻松地搭起来，反直觉地，这恰恰给 AI-Native 创业公司带来了真正危险的存在性风险。

直到不久前，造东西仍需要真金白银的开发时间和预算，搭个最基本的原型通常都要几个月。如今技术开发门槛基本消失，AI 让创始人太容易直接跳进建造，完全跳过验证它在真实世界里是否有用。

抵达问题-方案匹配，必须先验证假设，再开始造。但许多首次创业者，甚至有经验的创业者，都错误地以为 AI 能让这一步短路，把流程改成：有想法 → 立刻造原型 → 把「原型存在」本身当作验证。原型变成了「我的假设一开始就对」的理由，而那个假设到底是不是真的，从未被检验。

一个能跑的原型很容易被误当作「我正在解决真实问题」的具体证据，但它不是。原型只是你和潜在用户对话时一个有用的压力测试道具。对话本身，才是真正的证据。

过早规模化

挑战：当造东西既不费力又即时，你的执行速度可以远远超过业务真正需要的水平。

过早规模化的意思是：在还没有真正验证一条产品路径值得投入之前，你就已经把自己锁在这条路径上。

这一直是创业杀手，但 AI 让创始人更容易在毫无察觉时掉进这个陷阱。Agentic 编程助手太强大了，执行很容易跑在问题-方案匹配验证之前，而你根本没意识到自己已经偏航。

它会围绕一个根本错误的前提，带着同样的热情去生成、测试、调试、重构代码库。系统里的智能是你的。这一阶段的最高指令是：让你的判断力始终走在建造之前，尤其是在建造如此快速、感觉如此轻松的时候。

客观性丧失

挑战：让 AI 工具去找支持你既有看法的证据，它一定能找到。确认偏误（confirmation bias）现在配上了一台研究引擎。

确认偏误一直是创业里的职业病：创始人天然对自己的想法充满热情。现在 AI 工具给确认偏误加了很强的外挂。让 AI 验证你的创业想法，它能找出佐证；让它估算潜在市场，它能找到让 TAM（总市场/可服务市场/可获得市场）看起来值得融资的数字。

AI 沿着你的方向走。这意味着一个不问难题的创始人，现在能比以往任何时候都更快构建出一套精致、看似研究充分的坏点子论据，自己还觉得是在做尽调。解药还是同一个工具，只是方向反过来：AI 验证一个想法有多彻底，压力测试（pressure-test）一个想法就有多彻底。当研究和结构化对抗性思考浮现出反方证据、提示想法需要修正时，这就是 pivot（调整方向）的信号。

Claude 如何帮助想法阶段的创始人

把你的 AI-Native 创业概念推过想法阶段，常常让人觉得漫长得没头。你是创始人，你只想动手造。但这个至关重要的启动阶段，本质上是一次研究和验证练习，意思是先用那些帮你想得更严谨的工具，不要急着写代码。下面是如何跨 Claude 的三个产品面（Chat、Claude Cowork、Claude Code），尽快又尽职地走完想法阶段。

Chat、Claude Cowork 还是 Claude Code：怎么选合适的 Claude 产品面

AI 让创业者更快交付、自动化繁琐工作流、在规模上运作。但你用哪一面，取决于手头任务。

Chat 适合无需离开当前 app 就能完成的快速交流：从一份密集的投资人备忘里抓一句话要点、在董事会前快速核查一个论断、读懂团队一条很长的 Slack 串。

Claude Cowork 适合真正需要时间的知识工作：从多个来源拉资料、整理之后产出完成品，比如文档、deck 或电子表格。比如把一文件夹的客户访谈纪要变成下次产品评审用的主题发现文档；融资前把十几家友商网站汇成一张竞争格局图；或者一个周一早上的常驻任务，从你的工具里拉指标，把周度 KPI 简报丢进共享文件夹。

Claude Code 是给团队工程师用的 agentic 编程环境：直接接入代码库、Plan Mode、git 集成、本地/IDE/沙盒云环境。精益团队在不断成长的代码库里交付功能、迁移 MVP 时期的遗留代码、把原型推到生产，都在这里完成，不用等更多人头到位。

如果你要做的是…	用	为什么这样选
一个问题、一次改写、一次快速头脑风暴	Chat	快、对话式、无需配置
研究、分析，或基于你的文件和系统产出一份成型文档	Claude Cowork	文件夹访问、连接器、Skills、计划任务
写、测，或交付软件	Claude Code	代码库访问、diff、git、开发环境

三者底层是同一个 Claude；变的是它周围的工作空间。

定义并压力测试问题假设

你的领域专长和前期研究已经产生了一个假设。第一项工作，是把它打磨到真正可被测试。Claude 在这里特别有用，它会逼你具体：究竟谁有这个问题？多频繁？多严重？他们目前怎么处理？任何不能精确回答这些问题的问题陈述，都还没准备好被验证。

练习：和 Claude 一起反复打磨你的问题陈述，直到它变成一个可被测试的假设。比如「合同审查耗时太长」没什么意义；但「中型公司的法务团队每个合同审查周期要花 3 天以上，因为修订意见散落在邮件串里，而不是一份带版本控制的文档」就非常可测试。

下一步，让 Claude 反过来论证你的想法，去找反证。这能浮现出负面市场信号、失败的竞品、用户行为模式，以及那些被「支持性综述」悄悄降权的结构性障碍。

目标是：在进入客户发现之前，你已经拿最强的反方论据把自己的假设压力测试过一遍。这样，用户访谈才会真正开放，而不是一次寻找确认的行动。

注：把 Claude 当作结构化的反方代言人来用，是 AI 创业生命周期每一个阶段都成立的核心用法。

市场研究与竞争格局梳理

给竞品大小定个位

创业公司有一种特有现象叫「竞品忽视」（competitor neglect）：你太专注自己的愿景和执行，于是系统性低估了同一赛道里别人在做什么。所幸 AI 给了解药：让 Claude 为这个赛道里的某个竞品写出最有说服力的论证，说明为什么它会成功而你不会。

Claude 可以分析为什么对方的方法其实更好，为什么客户会选他们，为什么你以为的差异化可能并没有那么守得住。

练习：让 Claude 按层级绘制你的竞争格局：直接竞品、间接竞品、潜在收购方，以及可能进入你所在赛道的相邻玩家。然后让它论证每一层为什么对你的成功构成真实威胁，而不是只列出最容易被你驳倒的那个版本。

市场研究

Claude Code 可以综合公开可得的客户反馈，浮现反复出现的抱怨和未满足的需求。附加价值是：这等于在白嫖竞品客户的质性研究。

练习：让 Claude Cowork 汇总关键来源里的竞品评论，找出现有方案仍未解决的高频抱怨。如果你的假设能解决其中一条或多条，这是问题–方案匹配的强信号；如果不能，也值得知道。

Claude Cowork 还能从密集的行业报告、分析师文档和市场研究材料里提取相关信息和数字。这些干净、合成后的输入，会成为 Claude 后续分析工作的理想上下文。

练习：基于公开数据构建 TAM/SAM/SOM 模型，并压力测试背后的假设。判断市场是在扩张、整合还是成熟；这个背景会影响你对时机和差异化的判断。再画出买方格局：谁掌握预算，谁影响决策，他们是不是同一个人。

趋势分析

最后，用 Claude 倾听早期信号，判断你是否在正确的时刻进入。追踪那些已经在讨论你这个问题的 subreddit 和 LinkedIn 群组，记下用户描述问题时使用的原话。让 Claude 找出曾经解决过类似问题的类比市场，提取哪些做法有效、哪些无效。浮现可能加速或威胁这个机会的监管、技术或人口趋势。

练习：让 Claude 找出三个可能在未来两年显著影响你市场的外部趋势（监管、技术或人口都行），并判断每一个分别是你具体假设的顺风还是逆风。

注：本节的市场研究与竞争格局梳理不是一次性练习。你会在 MVP 和发布阶段继续发现、继续演进思考，所以每当假设变化，都要重复这些练习。

规划并设计客户发现

你从潜在用户对话里学到什么，取决于两件事：你问的问题质量，以及你有没有把这些问题问给对的人。Claude 特别适合帮你设计客户发现，包括找谁聊、问什么，以及怎么理解听到的内容。

找谁聊

一份精确的目标画像，比一长串联系人更有价值。画像里要包括最可能强烈经历这个问题的具体职位、公司类型、团队结构和资历层级。接着，识别这些人实际能在哪里被触达：社区、活动、LinkedIn 群组、Slack 工作区。最后建立一个优先级框架，按他们离问题有多近，决定先触达谁。

问什么

目标人群定义好之后，用 Claude 搭访谈框架：正确的问题、正确的顺序，结构能浮现人们「实际做了什么」，而不是「以为自己会做什么」。新手创始人最常犯的错，是问一个泛泛的未来式开放问题，比如「你会用这样的东西吗？」而不是追问相关的过去，比如「跟我讲讲你上一次处理这个问题的过程」。

Claude 也能指出你草稿里哪些问题在诱导受访者、哪些太宽泛、哪些会产生噪音而非信号。它还能帮你设计追问，用来处理回避，或深入挖掘那些重要但含糊的答案。

如果你的假设涉及多个 persona（用户画像），Claude 也能为每一类设计不同的问题组。财务经理和 CFO 面对同一个问题的关系并不相同，单一访谈框架会把这种差异抹平。

练习：先手写访谈问题，再让 Claude 审。明确要它标出任何诱导性、面向未来、过宽，或容易引出「社会期许答案」而非真实答案的问题。然后让它为访谈中最可能出现回避的两三个时刻设计追问。

访谈后分析

每次对话后用 Claude 复盘：把笔记喂给它，让它指出哪些确认了你的假设、哪些挑战了它、哪些是真正出乎意料的。

收集一批访谈之后，把全部访谈笔记交给 Claude Cowork，让它浮现反复出现的主题、矛盾，以及正反两个方向最强的信号。再把合成结果带回 Claude，让它指出：你对数据的解读，哪里可能是在把信息凑成你想听到的样子，而不是数据真正呈现的样子。

练习：每完成五次访谈，就让 Claude Cowork 综合笔记并产出两张清单：支持你假设的证据，挑战你假设的证据。如果第一张清单明显更长，让 Claude 判断这种不对称是数据本身如此，还是你一开始就希望找到这些。

客户触达与排期

用 Claude Cowork 自动化「建立联系人列表、跑触达、安排用户访谈」这些运营负担。

Claude Cowork 可以基于你和 Claude 定义好的目标画像（包括职位、公司类型、资历层级），研究并整理出一份结构化的潜在客户名单和已验证的联系方式。随后批量起草个性化触达邮件，让每封都贴合对方的角色和处境。

回复进来之后，它通过 MCP 连接 Gmail 和 Google Calendar，管理邮件串、处理排期请求、把访谈放进日历。流程继续往后：Claude Cowork 按设定节奏生成跟进草稿（比如第七天跟进未回复联系人），并在每一步完成时更新追踪表，让你始终知道每个潜在对象在 pipeline 里的位置。

练习：把验证过的访谈目标画像交给 Claude Cowork，请它建立潜在客户列表、起草个性化触达序列，并建好一张追踪表（包含触达状态、跟进节奏、访谈完成情况）。然后让它负责协调，你专注准备对话本身。

设计你的最终方案概念

你已经做完了验证工作：问题真实存在，你知道谁有这个问题，也有一个被证据支持的方案概念。用 Claude 从各个角度去发展并挑战这个方案概念：缺口在哪里？有哪些替代方案？这个方案要规模化成立，需要哪些条件？这是一个重要的现实检查：这个设计真的对应了验证过程揭示的那个问题，还是仍在对应你一开始以为的问题?

练习：把你的方案概念交给 Claude，让它识别这个设计最依赖的三个假设。然后追问：每个假设要成立，必须哪些条件为真？如果任何一个假设不成立，后果是什么?

用 Claude Code 搭一个轻量原型

现在到好玩的部分：有了验证过的假设和经过压力测试的方案概念，你终于可以做点东西出来了。

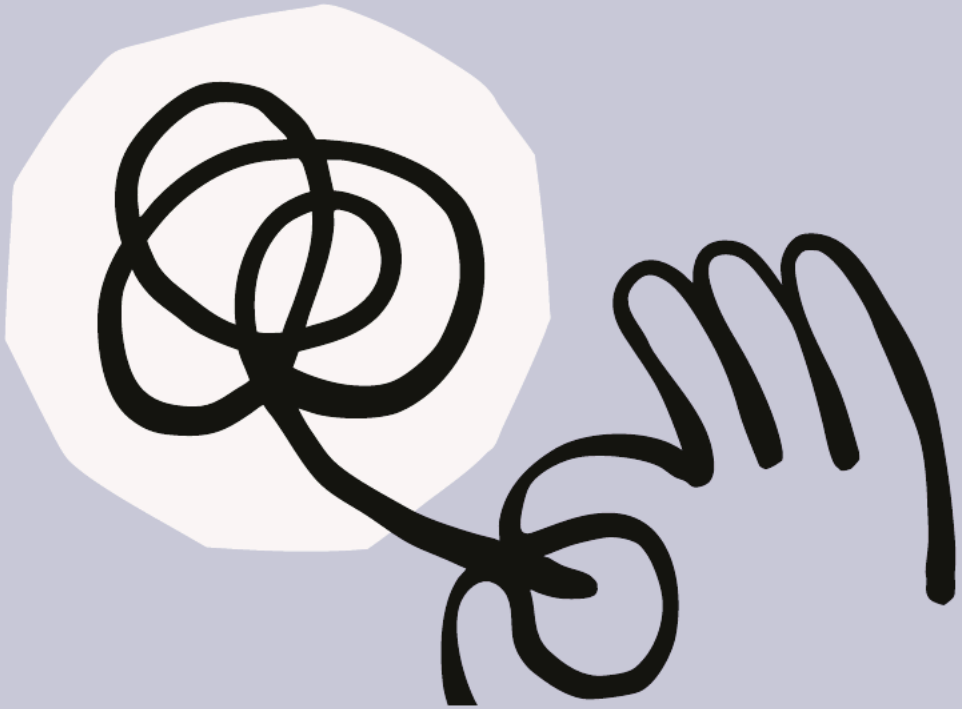
这就是想法阶段里 Claude Code 登场的时刻。哪怕你之前一直在小修小补，现在才是生成正式轻量原型的节点：用最小的产品面，把想法摆到真实的人面前，获得真实反应。

你还不是在造一个真实世界产品，而是在做一份功能样本，用于客户和投资人对话。真实用户触碰到一个能实际操作的东西，会告诉你十几次问题–方案访谈都告诉不了你的事。此前你是在证明这个问题真实存在；现在，你是在邀请潜在用户来反应拟议中的方案本身。

练习：定义你的方案所依赖的那一个核心交互。让 Claude Code 只做这一件事。做出来之后，把它放到五个来自已验证目标画像的人面前，请他们试用。这五次对话里学到的东西，将决定你是继续往前造，还是回到画板。

走到想法阶段的尽头，是 AI 创业赛跑里的一次大跃迁。因为你不再是在赌一个直觉，而是在对着证据执行。

接下来进入 MVP 阶段。创始人的指导问题从「这值不值得做？」转向「到底先造什么？」而 AI 的主要角色，也从研究伙伴切换为施工队。



Chapter 4

MVP 阶段

Chapter 4

MVP 阶段

不少创始人把 MVP 阶段当成「开工建设」，但它本质上还是一次收集证据的过程。区别在于，这一阶段你收集的是关于解决方案的证据，不再是问题本身：一群真实、可识别的人，是否觉得这个产品值得用、值得回头再用、值得付费，甚至值得推荐给别人。

MVP 阶段·目标

作为 AI-Native 创业公司的创始人，你的目标是把已经验证过的问题，转化成一款真实用户愿意使用的可运行产品。它不是路线图上所有功能都齐全的完整版，而是你想法里最小、最聚焦的那一次迭代：把真实方案摆到真实用户面前，跑出 PMF（产品-市场匹配）的真实证据。

与此同时，你现在怎么做，决定了未来能做什么。所以 MVP 阶段还有第二个同等重要的目标：跑得快，但不要背上那种会复利、等真实用户大规模涌入时反过来缠住你的技术债（technical debt）。

第三，从第一天起就投入精力建设持久上下文，才能让 AI 持续做你的放大器，而不是混乱的来源。在 AI-Native 创业公司里，你的代码库是你和 AI 一次次会话共同协作的对象，可读性是地基。那些跳过规格文档（spec）、架构决策和 CLAUDE.md 这类上下文文件的创始人，迟早会撞上一堵可预见的墙：每次新会话都要重新解释一遍代码库，AI 生成的改动也会一点点偏离最初的愿景。

MVP 阶段·退出标准

MVP 阶段的退出条件，是 PMF 的真实证据出现：一个具体、可识别的用户群体觉得产品有价值，愿意回头再用（留存）、愿意付费（收入），或者愿意告诉别人（推荐）。

MVP 阶段·挑战

在 MVP 阶段，创始人最关键的两件事是速度和判断力。挑战在于：你能不能用对的方式做对的产品、做得足够快，同时不抄那些日后会让你付代价的近路。

Agentic 技术债（agentic technical debt）

挑战：AI 几乎抹平了过去那些控制代码进入生产环境的天然瓶颈，速度因此被保证了。但当速度成为创始人在 MVP 阶段唯一考虑的变量，他们就会积累起很难偿还的技术债。

有些技术债在 MVP 阶段是合理的，前提是规模化之前必须把它管住。这种债会逐步累积，可以慢慢还，也可以专门安排一个冲刺（sprint）清掉。但 Agentic 技术债不一样，它会复利。

如果没把规格和架构约束写在 AI 能读到的地方，每次会话都得从零推导一遍基础决策，决策也会一次次发生架构漂移（architectural drift）。最后你会得到一个缺乏一致心智模型的代码库。不是因为某一块代码写得糟糕，而是这些零件从一开始就没被设计成能拼到一起。这是真问题，而且往往要等到很晚才暴露出来。

误把假 PMF 当成真 PMF

挑战：AI 工具能跑出亮眼的早期数据，但这些数字并不保证市场需要你的产品。

早期势头是创始人能经历的最强心理体验之一。经过数周甚至数月的验证和克制开发之后，把产品发出去，会让人觉得自己从一开始就是对的。

Agentic 编程工具能让你比以往更快抵达这个时刻，但早期牵引（traction）不等于 PMF。发布期的热度可能来自一些短暂因素：创始人的朋友、投资人其他被投公司的潜在买家，或者 Hacker News 上一个标题带来的流量尖峰。可惜，这些都不能可靠预测第六周、第十二周，初始助推消退之后会发生什么。

零摩擦的范围蔓延（scope creep）

挑战：当开发几乎不费力、近乎免费，总有一个很酷的功能可以加，或一个边缘情况想处理。这种范围蔓延弊大于利。

范围蔓延一直是创业的风险。现在的区别是，过去抑制它的「强制函数」（也就是工程时间的真实成本），已经不再以同样方式存在。加一个功能不再是一个冲刺，而是一个下午。

难点在于，每一项单独的新增看起来都合理。产品当然应该处理那个边缘情况，用户当然会想要那个工作流。由于 agentic 编程让每一项都很省力，当下并不像范围蔓延。但当产品越过原始边界开始摊大饼，你会失去方向和动量。

解药是在动手前先写下范围定义：产品做什么、明确不做什么，以及来自真实用户的哪种具体证据才足以证明应该加新东西。这样，决策点就从「要不要做这个？」变成「是不是已经有足够多用户告诉我们：没有它就拿不到价值？」

因经验不足而不安全

挑战：创始人用 AI 工具匆忙把应用推向市场，却没有先搞懂基本安全原则，最终会让用户暴露在本可避免的风险中。

硬道理是：agentic 编程工具生成的是能运行的代码，不是天然安全的代码。功能代码很好判断——要么能跑，要么不能。但安全漏洞在被利用之前是隐形的，没有天然反馈环（feedback loop）会提醒首次创业者哪里出了问题。把上线的 MVP 交给真实用户，意味着真实数据、真实暴露和真实后果。

轻视安全并不是 AI-Native 项目才有的新问题。每个时代的自筹资金创业公司都常常把安全考量拖到很晚，有时甚至拖到生产发布前夕。在任何用户接触你的 app 或方案之前先做一次安全审查，是把最小可行产品送到世界上所需的最低责任门槛。

Claude 如何帮助 MVP 阶段创始人

动手之前先定架构

在 Claude Code 写下第一行生产环境代码之前，用 Claude 定义并记录本阶段所有开发都要遵守的架构决策：遵循哪些模式、避开哪些依赖、做了哪些取舍以及为什么。这份输出会成为聚焦的架构上下文文档，建立 Claude Code 要在其中运行的护栏。

没有这份上下文，每次会话都从零开始，Claude Code 只能自行推断结构性假设。让 Claude Code 在没有护栏的情况下开发，会产出功能可用但结构不一致的代码库。在这样的代码库上迭代和扩张，本质是在浪费时间和 token。迟早有一天，代码会不可避免地坍塌，迫使你从头重建。

练习：打开 Claude Code 之前，先打开 Claude，描述你要做的事：它解决的核心问题、服务的用户，以及未来六个月你现实预期的规模。让它帮你定义 MVP 开发应遵守的架构原则、在你的约束下应避开的依赖，以及这个阶段你有意识接受的取舍。

接着，把这份输出保存为 CLAUDE.md 文件。这就是你的架构上下文文档：开发过程的第一个产物，也是后续每次会话都依赖的对象。CLAUDE.md 是 Claude Code 的项目级指令，为特定代码库提供上下文和说明，Agent SDK 在该目录运行时会自动读取。功能上，它就是项目的持久记忆。

定义并执行 MVP 范围

零摩擦的范围蔓延，是 AI 时代 MVP 的典型失败模式之一。就像你定义并记录产品的应用架构一样，在任何功能动工之前，也要先定义 MVP 的范围。

Claude 可以帮你写一份范围文档，描述 MVP 做什么、明确不做什么，以及功能修订标准：来自真实用户的哪种具体证据，才足以证明此刻应该加新东西。

当新的功能想法冒出来（它们一定会冒出来），用 Claude 做压力测试：这是用户的真实信号，还是披着产品思考外衣的创始人热情?

用 Claude Code 做 MVP

架构和范围定义清楚之后，Claude Code 就是主要的 MVP 开发工具。用它生成、测试、调试、迭代代码库，但要把每次会话当成对你已做出的产品决策的执行，而不是趁机塞进新想法的机会。

每次 Claude Code 会话开始时，先重读范围文档，并给模型提供 CLAUDE.md 架构上下文文档。每次会话结束时，把会话中浮现的决策更新进去。目标是一个你能解释其结构的代码库，而不只是一个能跑起来的代码库。

练习：给 Claude Code 工作建一个简单的会话模板，包含架构上下文文档、本次具体任务，以及要遵守的约束或模式。每次会话结束时，在上下文文档里加一条简短日志，记录做了什么、定了哪些决策、引入了哪些假设。每次五分钟的文档记录，是防止架构漂移复利成无法管理的代码库的便宜保险。

任何用户上手之前先做安全审查

作为 AI-Native 创业公司的创始人，你有责任知道代码库里有什么、理解潜在的暴露路径，不要把明显的漏洞交付给那些信任你处理他们数据的真实用户。

Claude 可以对 AI 生成的代码做一轮有用的初步安全审查，帮你识别常见漏洞。这是发布前应该养成的好习惯。但它不能替代安全工具，在更高风险的场景下也不能替代人类审查者。把它当成替代品的创始人，最后往往会出现在事故新闻里。

Claude Code Security 更进一步：它会扫描代码库里的安全漏洞，并为人类审查提出定向补丁建议，浮现传统方法容易漏掉的问题。

注：截至本电子书出版时，Claude Code Security 还是受限 beta 版本。加入工作流之前请先确认当前可用状态。

练习：部署给任何真实用户之前，用一份明确的 brief 把核心应用代码交给 Claude 审查：认证和会话处理、API 响应中的数据暴露、输入校验和注入风险，以及存在已知漏洞的依赖。认真对待每个发现，判断是否需要修复；凡是涉及认证、密钥或数据处理的部分，都要走人类审查。

发布前先把度量框架建好

那些把早期牵引误判成 PMF 的创始人，通常也是发布之后才开始追踪数据的人，而且挑选的指标往往是为了证明什么有效，而不是浮现什么无效。解药是在第一个用户出现之前，就建立度量框架。

用 Claude 定义你的具体产品应该看哪些指标、基准是多少、数据里哪些模式构成真 PMF、哪些只是好看的噪音。具体来说，发布 MVP 前先设定留存基准、激活标准，以及第 7 日和第 30 日目标。

接着，定义对你的产品而言什么是假阳性：有注册但无激活、有收入但无留存、初始热情但没有重复使用等等。数据到来时，让 Claude 替怀疑者发言：一个怀疑者会怎么解读这些数字?

管理用户发现和反馈的运营层

真实用户进入产品之后，运营层会迅速膨胀。Claude Cowork 可以承接重要但琐碎的工作：建立和维护用户联系人列表、跑触达序列、安排反馈会、给 bug 报告做分诊（triage）、追踪迭代周期。想法阶段那套管理用户发现后勤的 MCP 集成，在这里同样适用。

对细腻的用户反馈探索，要让真人留在收集环里。用户说「这很好，但我希望它还能……」时，需要解释：这是核心需求还是锦上添花？是这一个客户特有的，还是代表某个细分人群？缺失功能才是真问题，还是新用户引导（onboarding）上游出了问题？没有工具能替你回答。

练习：配置 Claude Cowork 跑你的 MVP 阶段反馈环：给早期用户列表起草触达、安排反馈会、为 bug 和功能请求设计结构化收集流程、每周写一份综合摘要。你先自己读摘要；之后再让 Claude 分析信息，捕捉你可能漏掉的重要点。

朝证据迭代，而不是朝「做完」迭代

MVP 阶段结束的标志，是你拿到了 PMF 的真实证据，而不是产品看起来有多「完成」。宣布达到 PMF、准备从 MVP 阶段进入发布阶段，本质上是一项判断练习，需要把创始人直觉和已收集的证据结合起来。不过，有几个有用的试金石：

- **Sean Ellis 测试：**问你的活跃用户：「如果再也不能用这个产品，你会有什么感觉？」如果超过 40% 的人回答「非常失望」，就是一个有意义的 PMF 指标。
- **努力测试（effort test）：**PMF 之前，留存需要持续干预：频繁触达、激励、个人跟进，加上创始人那种英雄式的精力维持用户活跃。PMF 之后，产品开始自己承担这项工作。当事情开始「被拉」而不是「被推」时，这种付出感的变化，是真东西发生的最清晰信号之一。

最终，没有哪个单一数据点能确认 PMF，因为它必须在多个迭代周期里持续成立，你才能确认。

数据要求 pivot 时，就要 pivot

如果投入这么多工作之后，仍然摸不到 PMF 怎么办？结果没有印证你最初的方向，并不是失败，而是系统在正常工作：MVP 阶段的设计目的，就是在你对错误答案过度投入之前浮现这些信息。

当数据不支持当前产品时，用 Claude 梳理这些数据到底在告诉你什么。

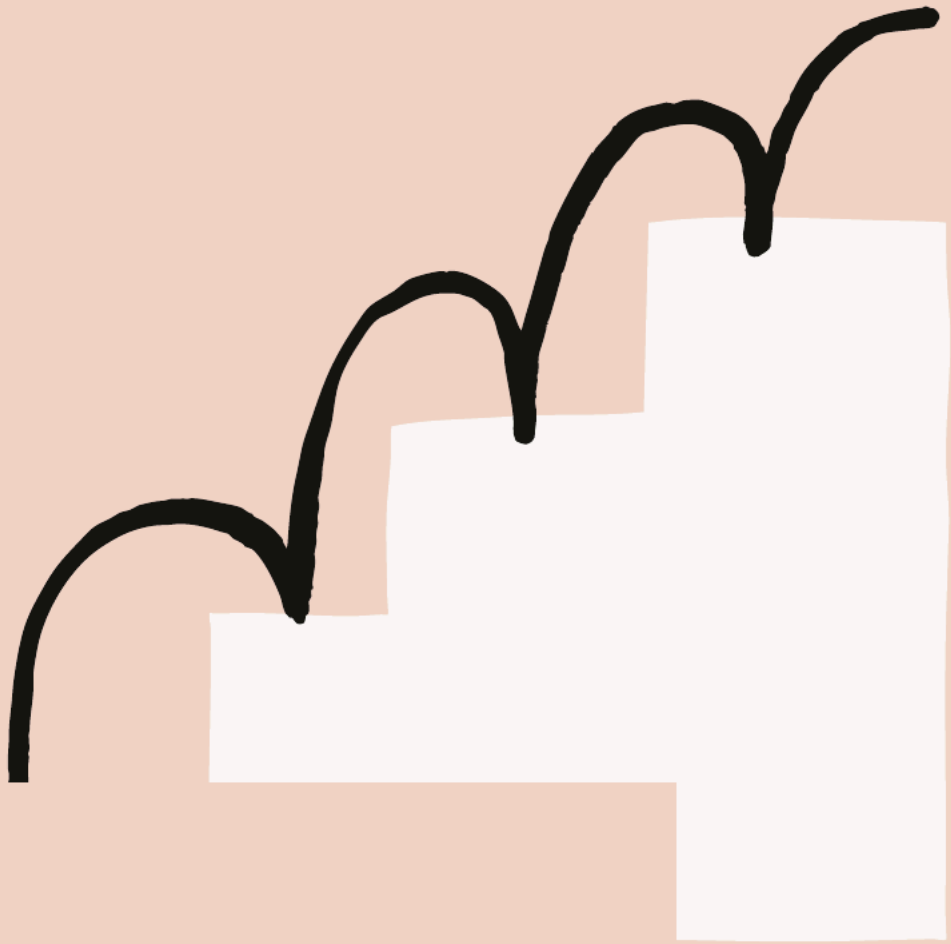
- **探索替代客户细分。**也许那些没转化的用户，从一开始就不是合适的目标。很多时候，对的受众已经在你的数据里，只是权重被低估了。
- **调整产品的价值主张（value prop）。**也许受众是对的，但 MVP 没有和用户产生共鸣。调整新用户引导、信息表达或核心功能侧重，可能不用改你已经造好的东西就能修复问题。

同时也要保持开放：设计价值和体验价值之间的脱节，可能深到需要一次更根本的改变。

练习：如果你已经完成三轮以上迭代周期，仍然没有朝 PMF 基准的实质推进，让 Claude 在你决定下一步之前先跑一次诊断。把留存数据、用户反馈和最初的问题假设都交给它，问三个问题：

- 数据里是不是存在某一段用户的反应方式和其他人不同?
- 设计价值与体验价值之间的落差，是定位问题还是产品问题?
- 当前产品要找到真正的 PMF，需要哪些条件？以你观察到的现象，那个情境现实吗?

让答案决定你是微调、pivot（调整方向），还是退回想法阶段。



Chapter 5

发布阶段

Chapter 5

发布阶段

如果说 MVP 阶段是要证明你的产品值得存在，那么发布阶段就是要证明你的生意值得长大。

发布阶段·目标

在发布阶段，创业者必须把早期势头转化成一台可重复、可持续的增长引擎。除了让产品具备生产就绪状态，你还得同时加固产品底下的基础设施——也就是围绕产品真正搭起一家公司。

创业公司在想法和 MVP 阶段天然以创始人为中心，因为你需要完整的处境感知和紧密的反馈环。但到了发布阶段，那些仍想把每一根线握在自己手里的创始人，会变成创始人瓶颈。目标不是把自己从公司里拿掉，而是搭建运营系统，把注意力释放出来，去做那些只有创始人能做的决定。

发布阶段·退出标准

发布阶段的退出条件包含三项：

1. **增长是可重复、由渠道驱动的。**你不只是在留住用户，而是通过具体渠道、用清晰的单位经济模型（unit economics）可预测地获取用户。CAC（获客成本）、LTV（用户生命周期价值）、回收周期，都是你心里有数、说得清楚的数字。
2. **产品扛得住生产工作负载。**基础设施已加固，安全与合规到位，可靠性在真实生产条件下也站得住，而不只是在你测试过的条件下。
3. **运营在没有创始人瓶颈的情况下也能跑。**流程已就位，自动化已部署。你不再是亲自处理客服、分诊、冲刺规划或报表的人。

发布阶段·挑战

找到 PMF 是早期创业生命周期里最难的问题。现在，创始人的挑战变成了把它保住。发布阶段是这样一个地方：那些已经找到真实产品牵引的公司，仍可能在包围和支撑产品的组织跟不上时散掉。下面是要警惕的失败模式。

技术债到期

挑战：那个为速度和验证而搭的 MVP 代码库，勉强够证明产品能跑。但生产流量、新功能和持续增长的复杂度，正在把当初的捷径一一暴露出来。

在 MVP 阶段，积累一些技术债是用速度换来的合理代价。到了发布阶段，那笔债开始算利息，拖得越久，修起来越贵。

解决方案是一次系统的架构审查（architectural audit）：找出结构性弱点，对最严重的几处做定向重构（refactoring），并实质性地扩展测试覆盖率，确保下一轮功能开发不会重新引入同样的问题。

创始人成了瓶颈

挑战：MVP 阶段，创始人参与每个环节是一种资产。到了发布阶段，随着支持工单上涨、产品决策堆积、运营复杂度倍增，同样的本能会变成约束。

从亲自做事，到设计能把事情做完的系统，是创业生命周期里最难的转变之一。因为很少有一个清晰时刻宣告它已经发生，风险就在于你完全错过它，继续停留在「建造者模式」里，而组织在你身边停滞。明显信号包括：本该一小时完成的决策，因为等你处理变成一周；支持请求越堆越多，因为只有你知道答案；运营任务只有在你亲自想起来时才发生。

解药是把你亲自在做的事彻底审视一遍，从最小的任务到最关键的决策，识别哪些能系统化、哪些能委派出去、哪些确实仍值得创始人的时间和注意力。

安全与合规不能再往后拖

挑战：MVP 阶段，把安全与合规措施做得简单还能接受。但现在，有真实用户、真实数据，甚至桌上可能摆着企业合同，它会变成负债。

MVP 阶段只有少量 beta 用户、生产环境里没有敏感数据时，安全漏洞还是理论风险。但产品一旦进入生产、有真实用户依赖它，假设就会变成非常现实的暴露风险。再往前一步，那些原型期可以忽略的合规要求，会在你处理客户数据、处理支付或卖给受监管行业的那一刻立刻适用。

解药是在生产规模到来之前（不是之后），做一次系统的安全与合规审查。凡是浮现出来的问题，都要当成下一波用户到来前必须修复的事项，而不是建议。

还没准备好就扩张

挑战：新市场和融资机会看起来都像增长机会。它们也可能是 PMF 死掉的地方。

你建立的初始牵引是真实的，但它也具体绑定在早期受众上。过早扩张到一个跟原市场差异很大的市场，会引入新的用户行为、合规要求、支付基础设施和基线预期，而你的产品并不是围绕这些设计的。一下子变量太多，你失去了清晰解读自己数据的能力。你还可能一边追逐新的、未经验证的受众，一边把原始用户群晾在一边。

Claude 如何帮助发布阶段创始人

发布阶段会完整用到 Claude 的三种形式，它们彼此支持：每个工具的输出都会变成另外两个工具的输入。结果会自然复利，一个同时使用三者的创始人，得到的不只是简单相加的总和。

这也是超精益创业模型在结构上可行的原因。当 Claude Code 构建产品、Claude Cowork 构建产品周围的公司、Claude 帮助把产品和组织知识运营化，一个小团队就能像大得多的公司那样运行。

在技术债复利之前清算

你的 MVP 代码库能跑，但它也需要一次系统化的修复扫描，找出任何可能变成结构性负债的技术债。

第一步，用 Claude Code 跑一次完整架构审查：识别代码库哪里脆弱、哪些捷径会变得维护昂贵、测试覆盖在哪里薄到下一轮功能开发会重新引入同样的问题。

把 Claude Code 的审查发现喂回 Claude，让它分诊并排序修复工作：什么必须在下一次发布前修、什么可以等一个冲刺、什么在当前阶段属于可接受的持续债务。这也是把 MVP 阶段那些只装在你脑子里的架构决策写下来的时机。把它们放进 CLAUDE.md，能确保未来每次 Claude Code 会话都从同一份理解开始。

练习：让 Claude Code 审查 MVP 代码库，产出结构性弱点、测试覆盖缺口和重构候选项的优先级清单。然后把清单交给 Claude，让它把修复工作排进几个冲刺：哪些重大问题要先处理、哪些可以和功能开发并行、哪些可以等等再说。

搭起替代创始人注意力的系统

要搭建能释放你注意力的运营系统，让你专注处理只有创始人能承担的责任，第一步是搞清楚你的注意力到底花在哪里。用 Claude Cowork 对当前运营负荷做一次结构化审计，记录每个重复任务、每个落到你桌上的决策、每个只因为你亲自记得才会发生的工作流。然后让 Claude Cowork 把清单分成三类：能完全自动化的、需要人但不一定需要你的、确实需要创始人判断的。

审计完成后，用 Claude Cowork 设计自动化候选项的工作流逻辑：每个工作流由什么触发、决策规则是什么、输出长什么样、完成后流向哪里。

把安全与合规做成一条产品工作流

用 Claude Code 浮现代码层面的问题：那些常出现在 SOC 2、GDPR、HIPAA 审计以及目标市场要求标准中的问题。它会同时浮现漏洞和合规缺口。把发现交给 Claude，帮你排修复优先级，并设计企业买家签约前会要求的控制项、审计日志和访问管理。

注：AI 扫描是一种辅助，不能替代有资质的合规审查。

接下来，把合规工作流并入开发周期，而不是当成一次性项目。合规文档需要持续维护和更新。对正在接近企业合同或国际市场的创始人来说，这也是 Claude Code 安全扫描帮你准备独立安全评估的时机。

练习：用 Claude Code 跑一次代码级安全审查，方向对准目标市场所需的框架。把输出喂给 Claude，请它产出两样东西：优先级排序的安全修复序列，以及为了通过潜在企业买家的合规审查、你需要准备的文档和控制清单。

把一直跳过的产品管理流程立起来

发布阶段需要一套轻量、可重复的流程，不需要创始人介入触发或维持也能运行。用 Claude 设计：产品时间线和工作周期如何组织，一份 spec 在 Claude Code 触碰功能前必须包含什么，bug 报告如何分诊和路由，每周指标简报覆盖什么、如何分发。

流程设计好之后，用 Claude Cowork 搭建并运行运营层：安排冲刺仪式、把新进 bug 报告路由到正确位置、从连接的数据源汇总每周指标、维护让用户信号持续流入产品决策的反馈环。

练习：请 Claude 设计一套轻量的产品管理操作系统：明确的冲刺节奏、最低规格文档模板、bug 分诊决策树，以及从真实数据源拉取的每周指标简报。然后让 Claude Cowork 执行并运行系统里的重复运营环节（例如排期、路由和报告汇总），让它们按计划发生，而不需要你出手。

Chapter 6

规模化阶段

Chapter 6

规模化阶段

在规模化阶段，创始人的角色重新校准，从建造者转向面向公众的高管。产品仍然是核心，但你的日常工作越来越多地落在公司本身。你必须把注意力扩展到规模化阶段的新活动，比如分析师沟通会和 IPO 路演，同时尽量守住那份以 AI 为中心的精益结构性优势。

规模化阶段·目标

扩张技术基础设施的工作不会停，现在又加上一项：扩张组织本身，把它催熟为一门真正的生意。

到了规模化阶段，你要从几千用户走向几百万，从单一市场走向多个市场。前面每个阶段，增长都是你能摸着走的事：贴近用户、依靠紧密反馈环的数据，再加上一点健康的创始人直觉调整方向。现在的目标，是搭建由成熟组织运营支撑的系统化增长。

对一家 AI-Native 创业公司来说，你的目标应该是用累积深度建出一条防御性护城河（defensible moat）。这种深度来自三块：你已经构建进产品里的专业知识、产品与用户所依赖的其他工具和平台之间的深度整合，以及专有系统数据和工作流。那些一直朝同一方向、在同一基础设施上持续构建的创始人，手里现在握着真正难以复制的东西。

到了这个阶段，公开市场投资人、分析师、监管者、企业采购团队和收购方都会施加更大压力，也带着更深的怀疑，因为赌注变高了。你的产品和组织都必须经得起外部审视：不只是已经造出来的能力，还有围绕它的治理、合规姿态、财务控制和战略叙事。

规模化阶段·退出标准

规模化阶段的退出条件不再是单一里程碑，而是一个阈值事件：即使创始人越来越少直接管日常运营，公司也能可持续运转。你已经展示了系统化增长；建好了能满足最苛刻外部审查者的组织治理和合规基础设施；并且能扎实回答这个问题：「如果一个资金充足的在位者今天复制了你的产品，你的用户会留下来吗？」

实际中，这个阈值通常以三种形式之一出现：不再需要外部资本的规模化可持续盈利、IPO 就绪，或被收购。三者都要求你的增长系统化且可审计，产品护城河经得起推敲，组织在运营上成熟可持续。

当这些都成立时，恭喜你：你的创业公司已经从一场赌注，变成了一门生意。

规模化阶段·挑战

把运营层放手出去

挑战：规模化阶段的运营系统必须可靠、可持续地运行，不能靠人盯着。对一个从第一天起就亲力亲为的创始人来说，这种转变既是结构挑战，也是心理挑战。

发布阶段的工作是把系统创建出来；规模化阶段的工作则是让这些系统成熟到完全可信，然后真的信任它们。

这比听起来更难。即使你是一个擅长授权的创始人，也未必每次都清楚什么该交出去、什么该留在自己手里。交得太多、太快（尤其是交给 AI 自动化系统），关键决策可能会在缺少创始人独有上下文的情况下被做出来。但抓得太久，你又会变成瓶颈。

这里的根本挑战，是识别那些只存在于创始人脑子里、或藏在未成文工作流里的机构知识，把它编码进有文档、可审计、可交接的系统中。

扩张技术运营

挑战：客户不再只评估你的产品，他们还想知道你的组织能不能成为可靠的基础设施伙伴。

创业前三个阶段的技术挑战都围绕代码库：构建对的方案而不积累技术债，再为真实用户加固安全和合规。进入规模化阶段，挑战变成围绕代码库的一切：搭建支持基础设施、文档和可靠性保证，向外界传递成熟信号。

签多年合同的大客户和机构买家，签约前会要求看到这些，签约后也会按这些标准约束你。好在带你走到这里的同一套 AI 基础设施，也能帮你建立专门的支持职能：明确的响应时间、新客户工程团队真正用得上的文档。

扩张组织职能

挑战：规模化阶段的公司通常需要招聘、薪资、会计和法务运营等组织基础设施，不管真正在跑公司的人有多少。

发布阶段，系统化运营意味着把那些消耗创始人注意力的工作流自动化。规模化阶段的创业公司现在需要扩展一组更广、某些方面也更要紧的运营职能，比如财务报告、合规监控、合同管理和客户支持。

搭一个 GTM 职能

挑战：自然增长有天花板，多数规模化阶段的创始人在真正搭过 GTM 职能之前，就会撞上它。

想法、MVP 和发布阶段的增长，往往来自创始人亲自卖：从一篇恰到好处的 Product Hunt 帖子，到与早期客户的私人关系。这样的自然增长只能走到某个点，多数创业公司会在规模化阶段撞到这个上限。信号包括用户曲线变平、获客成本上升，以及销售管道只有在创始人亲自介入时才会推进。

规模化阶段的增长，需要搭建一台专门的增长引擎，让产品触达更新、更广的人群。但多数创业公司创始人，可能从来没真正运行过市场、销售、分析师关系这类项目。一个像样的 GTM 打法（motion）不只是建立新系统和流程，还要打造一种品牌声音和产品故事，说明你想怎么谈论自己的产品。因为在生命周期的这个阶段，你需要触达的不只是单个新用户，还包括投资人、企业买家等完整目标人群。

好消息是，GTM 职能不必做得很大，也可以有效。构建产品的同一套 AI 基础设施，也能帮你把产品带向市场。

Claude 如何帮助规模化阶段创始人

早期阶段，Claude 是产品本身的基础设施：验证想法的研究伙伴、设计并构建原型的工程团队，以及让单人创业公司得以成立的 AI 运营层。走到规模化阶段的 AI-Native 创始人，现在可以继续用 Claude、Claude Code 和 Claude Cowork，沿着同样的方式扩张。

把日常任务交给 Claude Cowork

用清醒的视角开启规模化阶段：你现在最需要把时间和注意力投在哪里？对第一次创业、从未真正搭过一门生意的创始人来说，这一步并不容易。Claude 可以帮你列出在这个阶段只有你能做的事，比如产品叙事决策、董事会关系、企业大单、创始人之间的对话。清单之外的，都是可以委派或交给 Claude Cowork 自动化的候选项。

练习：用 Claude 绘制当前运营层的瓶颈地图：所有目前经过你的工作流、决策和审批。然后让 Claude 推演：如果你一周不在，每一项会发生什么？那些会停下来的工作流，就是你还抓得太紧、紧到足以阻碍进展的地方。

这些瓶颈和 Claude 帮你列出的创始人优先事项与责任清单怎么对应？

接下来，压力测试一下你已经搭好的系统：它们真的能随业务增长一起扩张吗？

练习：用 Claude 绘制当前工作流，再追问：如果你一周不在，每一项会发生什么？那些会停滞的工作流，说明交接标准、升级路径或异常处理还需要收紧。Claude 可以分析失败点并提出修复建议，让你按需更新或替换 Claude Cowork 自动化。

把技术运营升级为企业级基础设施（enterprise-grade infrastructure）

扩张时，买家需要确信你的产品和组织可以被当作长期基础设施来信任。代码库内部的技术工作照常继续，但现在代码库周围的技术工作也必须开始做。

第一步，是把机构知识转换成能扩张的系统。用 Claude 起草并维护企业采购方期望看到的书面基础设施，包括产品文档、支持 playbook 和 SLA。

与此同时，让 Claude Code 按企业合同要求的可靠性和安全标准对代码库做审查与加固，并构建那些 Discord 社区支持从来不需要提供的技术支持基础设施：日志、监控、事故响应工具，以及让 SLA 真正能被执行的可观测性层。

接下来，Claude Cowork 运行企业支持本身的运营层：工单路由、升级工作流、由产品变更触发的文档更新、续约追踪，以及企业客户成功依赖的报告节奏。三者加在一起，让一个小团队拥有大组织级别的支持姿态，而这正是签下多年期企业合同前你必须证明的能力。

练习：挑出最苛刻的三个潜在客户，或者列出你最想签下的三个理想客户。让 Claude 做一次差距分析：这些客户的企业采购团队在签多年期合同前，会期待看到哪些文档、SLA 和支持基础设施？你目前哪里还不到位？把分析结果用来安排 Claude Code 和 Claude Cowork 的技术与文档工作。

搭一个真正的 GTM 职能

创始人的 hustle 把你带到了这里，但要把公司继续扩张，你需要做出并执行一套真正的 GTM 策略。AI 可以帮你搭建、运行一整套 GTM 引擎。

Claude 可以从零搭起基础 GTM 资源：市场细分、信息架构、分析师关系策略、销售 playbook，以及当你开始面对公开市场投资人、企业买家和华尔街分析师时关键的投资人指标叙事。每类受众都有自己的语言，并用自己的标准评估你；Claude 的工作，是把产品的价值主张翻译成对每个受众都成立的产品营销方式。

这时候，Claude Cowork 可以成为你的战术执行层：内容管线、外呼节奏、分析师沟通会后勤、新闻和 PR 节奏、CRM 维护、销售管道报告，以及把 GTM 策略转化成实际商业动作的大量重复性周期。

当 GTM 打法需要产品营销基础设施（比如交互式演示环境、集成文档、沙盒租户、API 参考、技术 one-pager），Claude Code 可以为你构建。买家期待从技术层面评估你的产品。在规模化阶段，一段 Loom 视频加一份销售 deck 已经不够看。这也是让 GTM 打法能异步跑起来的基础设施：一个做得到位的 demo 环境，可以在你开董事会的同时帮你成交。

把领域专长（subject matter expertise）和机构知识转化为 AI 上下文

许多超精益的创业公司创始人，正在为某个自己亲身经历或一手观察到的行业真实问题，构建高度具体的 app 或工具。Agentic AI 让从未写过一行代码的创始人，也能用自己的领域专长做出解决复杂问题的产品。Claude、Claude Code 和 Claude Cowork 各自分工，把创始人知识转化成会复利的产品具体性。

用 Claude 捕捉、整理并打磨创始人知识，就是把领域专长放到产品能够触达的地方。通过长对话、Projects 和记忆，创始人可以把自己知道的一切（行业黑话、监管坑、边缘情况、痛点、为什么那些显而易见的答案并不管用）放进结构化、可搜索的上下文里。Skills 可以把重复工作流编码成可复用的例程（比如「我如何审查商业租约」「我如何分诊一份患者入院表」），让 Claude 每次都按同样的方式运行。几个月之后，这会沉淀成一层专有知识基底，通用 AI 无法匹敌。

把领域知识外化到 Claude 里，对于把行业特有的边缘情况编码进产品非常有价值。比如说，一个通用医疗计费工具可能会在 340B 药品项目索赔上栽跟头，而你的工具有专门的逻辑处理它。Claude Code 帮你把同行业其他专业人士的常见痛点，转译成验证逻辑、prompt 优化，或者与某个竞争对手压根没听过的小众行业系统的 MCP 集成。结果是，你的 app 或工具在深度和广度上持续复利，竞争对手没法简单复制。

练习：找出一个通用竞品在你的垂直领域一定会做错的边缘情况。基于你真实见过的场景，让 Claude Code 帮你为它写一个专门的测试用例（不是单元测试）。每当类似的边缘情况再次出现，就把它加进去。你的测试套件会逐渐长成一张护城河地图。

让累积的用户数据复利成防御性优势

用户在与产品交互时，会生成行为信号（比如他们接受哪些输出、拒绝哪些输出），这些信号会反过来影响产品路线图。时间一长，你会摸清自己用户群的具体模式、偏好和边缘情况。这就是复利价值（compounding value）：每一次改进都让产品更有用，带来更多使用，激发更多反馈，再推动下一轮改进。

这些数据带着时间锁定和具体上下文，没办法被抄袭者重建：你买不到数千名用户在你产品里打磨工作流之后留下的行为指纹。

Claude 可以帮你审计已经收集到的用户交互数据，识别其中信号最强的行为模式，并设计一条把持续使用转化为系统性模型改进的反馈环。

练习：给 Claude 一份产品交互数据摘要：你收集了什么、收集了多久、你对用户随时间的使用方式了解多少。让它识别三个信号最强的行为模式，并为每一个设计一条能转化为系统性模型改进的反馈环。然后请它帮你起草一页护城河叙事，用于产品营销：你的数据飞轮（data flywheel）怎么运转、转了多久、为什么一个资源充足的竞争对手即使今天开干，也没法在两年内复制。

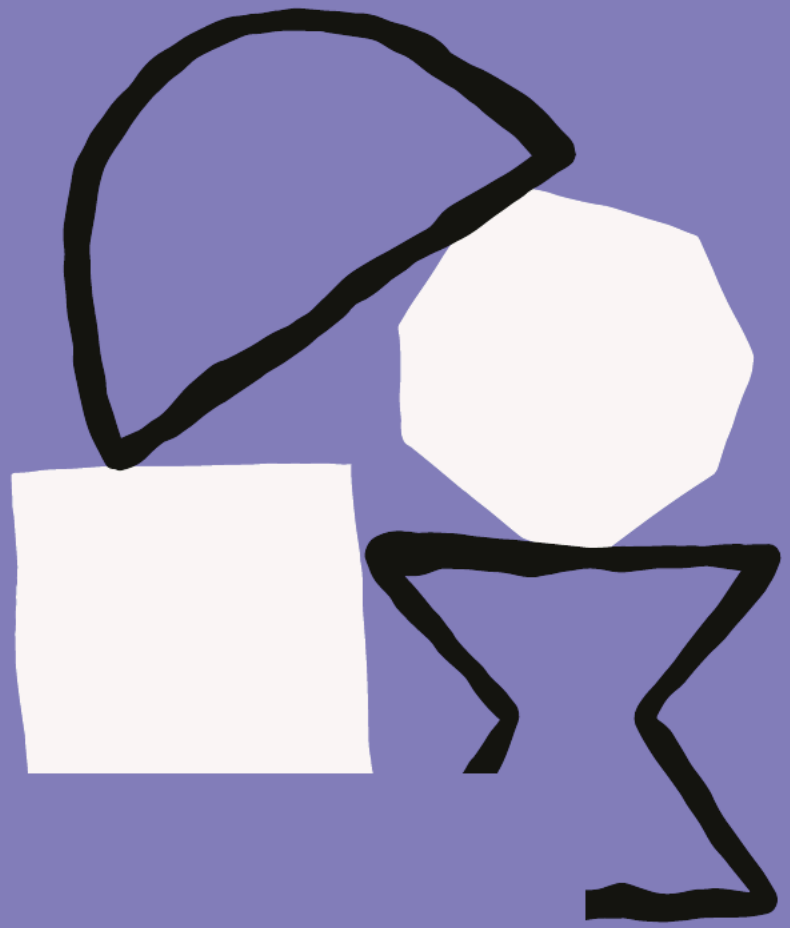
制造工作流锁定（workflow lock-in）

数据网络效应（network effects）的复利会让产品更难被复制，而用户工作流的锁定（lock-in）会让产品更难被抛弃。用户在日常运营中跑你的产品越久，它就越深地嵌入他们真实工作的方式。他们在上面搭了自动化，训练团队使用它，把它接到数据源和其他工具。那些 prompt、被打磨过的工作流、被标准化的输出，都是围着你的产品「能做什么、怎么做」长出来的。到了这一步，切换就不再是一个产品决策，而是一项完整规模的运营工程。

制造工作流锁定的第一步，是让 Claude 按集成深度给当前客户群画一张图。对每个客户细分，识别他们在你产品上搭建了哪些工作流、依赖哪些集成。这张图会告诉你产品在哪里真正粘住了，又在哪里还需要扎得更深。

你提供的集成越多，客户就越有空间构建依赖你产品的工作流。Claude Code 可以帮你快速搭起目标用户依赖的数据管线、项目管理工具和其他系统的原生集成。它还能构建 API、webhook 和 SDK，让客户不只是使用你的产品，而是在你的产品之上做开发，这才是最深一层的锁定。

练习：让 Claude 为你前十大客户做一次工作流集成审计。对每个客户，记录他们搭建的自动化、依赖的集成、跑在你产品上的团队工作流，以及你估算的切换成本。然后请 Claude 在这群客户里识别共同模式：哪些类型的集成对你这款具体产品锁定最深？你还能构建或开放什么，让目前只停留在表层的客户拥有更深的集成？



Chapter 7

工作没变，规则变了

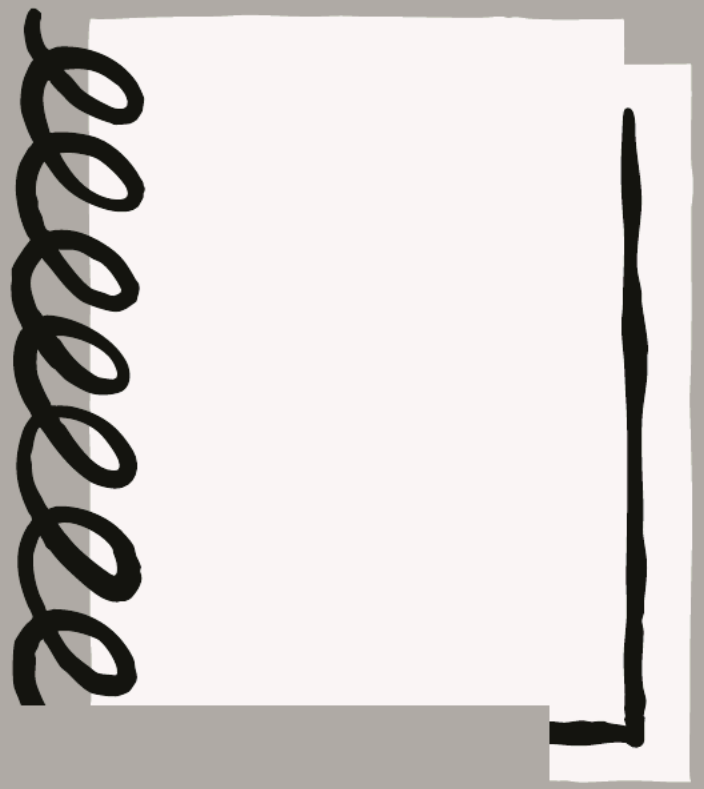
Chapter 7

工作没变，规则变了

在 AI 时代，创始人的工作并没有变：找到一个真问题，做出能解决它的东西，把它扩张成一家有意义的公司。变的是通往那里的路。穿过想法、MVP、发布、规模化这四个阶段，AI 把一个个季度压缩成了一个个星期。

那些过去要花几个月的验证周期，现在一个下午就够。一个能跑的原型，不再需要一位拥有合适技术栈的联合创始人；它只需要一个清晰的问题，加上几次专注的会话，再配上一个编程 agent。发布就绪从一段发布前的紧张冲刺，压缩成一条持续工作流。到了规模化阶段，过去把早期员工逼成救火队员的那份运营重量，越来越多可以交给 AI，让你的团队把注意力放在那些会变成护城河的判断决策上。

瓶颈不再是「你能造什么」，而是「你选择造什么」。



Resources

Resources

用 Claude 构建

- **Building AI Agents for Startups**：讲创业公司如何用 agent，让自己在扩张过程中不再过度依赖创始人本人。
- **Claude Code docs**：带构建者从初始安装一路走到进阶 agentic 工作流。小提示：先从「How Claude Code works」概览开始。
- **Claude Code best practices**：覆盖 Anthropic 内部和工程团队验证过的模式，包括上下文管理、权限、规划和验证工作流。
- **Using CLAUDE.md files**：讲解如何为特定代码库配置 Claude Code。MVP 阶段创始人搭建开发环境时的必读材料。
- **Claude Code power user tips**：来自 Claude Code 团队自己的工作流模式，包括并行会话和验证环。
- **Get started with Claude Cowork**：讲解团队如何搭建 Claude Cowork，并开始落地 Skills、插件等能放大影响的功能。
- **Tutorials**：claude.com/resources/tutorials 提供一份可搜索的、面向具体任务的实操教程清单。

创始人故事

- **How three YC startups built their companies with Claude Code**：看 HumanLayer (F24)、Ambral (W25)、Vulcan Technologies (S25) 如何用 Claude 让原型快速上市，并用 agentic 编程工作流扩张 AI 驱动平台。
- **GC AI**：创始人用领域专长打造了一个由 Claude 驱动的法律平台，贴合企业内部法务团队真实的工作方式：公司专属 playbook、跨职能利益相关者、可变的风险容忍阈值。
- **Carta Healthcare**：用 Claude 驱动其临床抽象平台，每年处理 22,000 例手术，数据抽象时间减少 66%。
- **Anything**：基于 Claude 和 Agent SDK 构建，已经帮助 150 万用户在不写代码的情况下把想法变成可运行的软件产品，其中包括一位非技术背景的创始人做出并已经在卖的完整招聘平台。Anything 的 AI agent 负责完整构建，让单干创业者（solopreneur）可以把精力 all in 在自己的领域专长上。
- **Cogent**：一家应用 AI 实验室，构建 agent 来自动化企业关键安全任务。Claude 是其 agent 的推理层，覆盖漏洞全生命周期的调查、优先级排序和修复。
- **Airtree**：把 Claude Cowork 当作运营基础设施的中心，统一过去散落在十几个工具和团队里的数据。现在，只要一个人用 Skills 搭出一个工作流自动化，组织里所有人都能用它完成那些长期没空做的事。
- **Duvo**：构建 AI agent，跨 ERP、供应商门户、电子表格、邮件甚至电话，运行采购、供应链和品类管理流程。Duvo 完全构建在 Claude 之上，并用 Agent SDK 跨工作流做编排。
- **Zingage**：面向居家护理机构（home-care agencies）的 24/7 自动化运营 AI agent 平台。它用 Claude 的结构化工具调用在 EMR 和多条沟通渠道之间编排，再用 Claude 的上下文推理，做出能因患者而异、有细腻判断的结果，而不是只对最常见的情况做模式匹配。

- **Kindora**：由一位非营利组织高管打造的 AI 驱动平台，用 Claude Sonnet 做出了一个急需的工具：智能匹配慈善组织与资助方。在把数千个匹配筛成少数值得追进的对象之后，Kindora 的 MCP 连接器让非营利机构可以直接在 Claude 里使用它的潜在资助方挖掘工具。
- **Wordsmith**：由一位转行做 CTO 的律师创办，为企业内部法务团队提供可靠的 AI 法律技术。Claude 是 Wordsmith 合同审查、协议起草和文档审阅能力背后的推理引擎，公司的工程团队则用 Claude Code 来构建并演进平台本身。

创业支持与机会

- **Anthropic Startups Program**：面向与 Anthropic 的 VC 合作伙伴有合作关系的创业公司，提供免费 API credits、公开可得的最高一档 rate limits，并邀请参加专属创始人活动和工作坊。
- **Claude community**：面向构建者的论坛和社区空间。
- **Live learning resources**：大会、网络研讨会、直播和回看录播。

译后记

这本《创始人行动手册》是 Anthropic 2026 年 5 月发布的官方手册，原版 36 页。我自己作为 AI Native Coder，看完之后觉得它把「AI 时代怎么创业」这件事讲得比许多公众号文章都更踏实，所以烧了几百万 tokens，让 Claude Code 替我把它译成中文，并按原版结构 1:1 重新排版。我自己一行代码、一行译文都没动。

本译本仅供个人学习与内部研究使用，不做商业发行。原版下载请到 claude.com/blog/the-founders-playbook。

如果你也在用 AI 做产品、做公司，欢迎在下面这些地方找到我：

B 站	花叔v · space.bilibili.com/14097567	**X**	@AlchainHust
YouTube	@Alchain	**小红书**	花叔
公众号	花叔	**官网**	huasheng.ai

花叔

2026.05 · huasheng.ai