创始人手册：打造 AI 原生创业公司

Claude

目录

第 1 章

2026 年重启的创业生命周期

创业者手册：构建 AI 原生创业公司

> The Founder's Playbook: Building an AI-Native Startup 原文：Anthropic, 2026 | 中文翻译版

目录

第一章 | 重启 2026：创业公司生命周期的重构

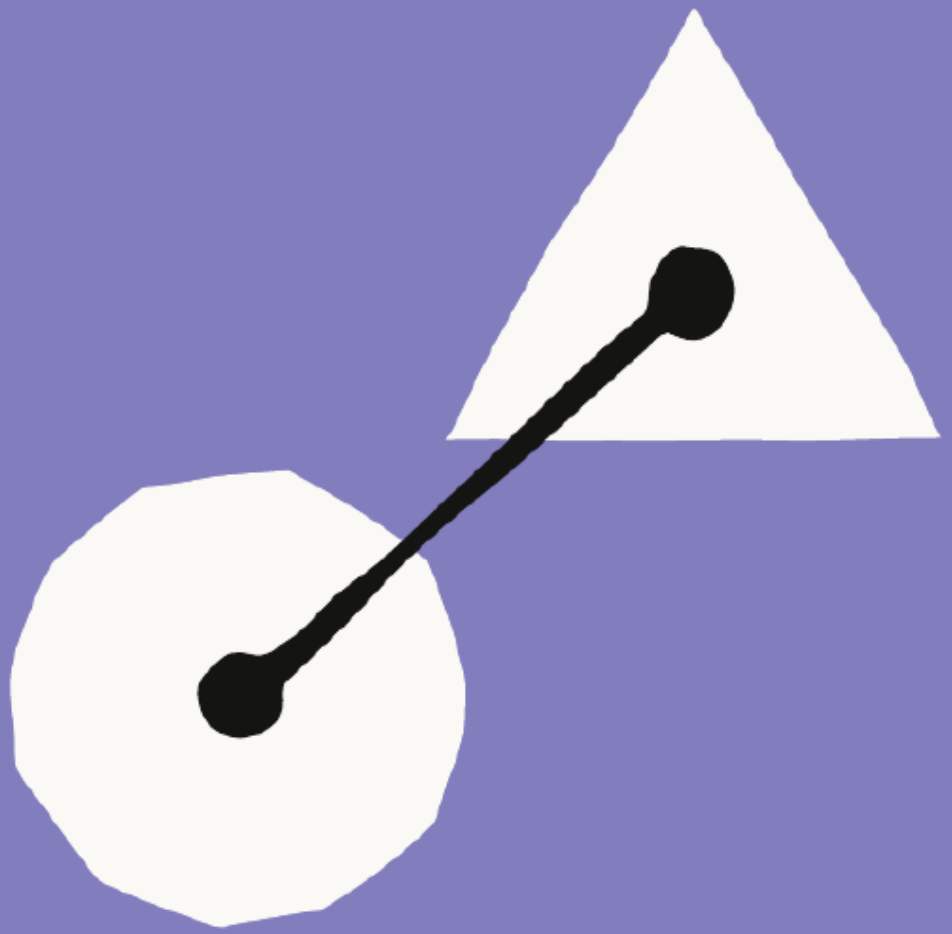
AI 正在重塑创业公司的构建方式。从未写过一行代码的创始人，如今也在交付生产环境应用；而精简的 "10 人独角兽"，从那种胆大包天的逆袭故事，演化成了一种刻意规划的行动方案。

在 2026 年，AI 可以编写生产代码、做市场调研、综合竞争格局、起草投资人材料、自动化运营工作流。它消除了那些哪怕是经验丰富的技术创始人都曾望而生畏的陡峭学习曲线（整合各种工具、平台和系统将创意落地）——更重要的是，它把"谁能创办创业公司、谁能造出产品"这件事的门槛彻底拉平了。

在 2026 年，一个好点子能把创始人推得比以往任何时候都远。Agentic Coding（智能体编码）把过去需要一整支工程团队才能完成的工作，压缩成了创始人自己就能交付的产出。

传统的创业增长曲线假定：从 idea 到 scale 的路径是 验证 → 融资 → 招人 → 构建 → 再融资 → 增长 → 再招人 → 循环。如今，AI 抹掉了"每进入一个新阶段都要更大团队、不同技能、新一轮融资"这种预期。

本手册依据这些新现实，重新绘制了创业旅程的四个核心阶段（Idea、MVP、Launch、Scale）的地图。我们逐一探讨：当 AI 成为你的技术与组织发展的核心时，每个阶段是什么样子；每个阶段对的工具是哪些；以及那些用上这些工具的创始人，正在如何压缩时间线。如果你已准备好绘制从 idea 到 exit 的最短路径，请继续读下去。



第 2 章

创始人的定义正在改变

第二章 ｜ "创始人"的定义正在改变

过去，创始人由他们"能做什么"来定义：技术型创始人写代码，非技术型创始人做业务运营、谈生意。但 2026 年创始人能够使用的模型、系统和 AI 智能体，已经溶解了"能造东西的人"与"有值得造的好点子的人"之间的那堵墙。

AI 原生创业公司从根本上改变了"创始人"的含义。如今，没有任何工程背景的人也能构建出把自己想法变成现实的生产级软件；而具备技术能力却对商业知之甚少的创始人，也能轻松产出一份 go-to-market 战略、财务模型、和高度打磨过的投资人 pitch deck。

历史上，创始人的大部分时间花在执行模式：写代码、管人、处理日常运营。在 AI 原生创业公司里，创始人的角色更少地是"个人贡献者"，更多地是"智能体的编排者"——这些专门的 AI 助手能够读文件、运行命令、执行代码，甚至浏览网页。创始人的注意力上移到更高阶的工作：产生想法，并指挥执行这些想法的系统（AI 智能体、工具，以及现存的小团队）。

不过，AI 作为核心基础设施最具革命性的成果，是为有领域专长的非技术创始人解锁能力。当创始人池从工程背景之外扩展开来，你会得到一批由"经历完全不同的人"创办的创业公司——他们解决传统技术创始人通道从未优先（甚至未曾注意到）的真实问题。

精简型创业公司的 AI 工具能力

传统创业模型假定：你需要雇工程师来构建，雇销售来卖东西，雇运营来管理业务。员工数量被当成组织势头与产品成熟度的标志。

2026 年的早期创业公司则截然不同。他们刻意保持极致精简，常常只是创始人一人，或加上少数几个伙伴。通过把技术和组织发展都建立在"AI 作为基础设施"之上，他们可以在扩大团队之前就达成产品验证、早期收入、甚至盈利。在三个方面，AI 帮助一个创业公司像一个更大的组织那样运转：研究、智能体编码、关键业务运营的工作流自动化。

对话式智能与研究

类比：每个领域的 on-call 专家

想想创始人在第一年中几乎肯定不知道、却必须了解的事：怎么搭薪资发放？怎么规划产品开发 sprint？怎么起草一份精炼的投资备忘录？

这类早期创业问题过去通通只有一个答案：**找一个懂的人**。对于自筹资金或种子前的创始人来说，这要么消耗本来该用于构建的时间，要么烧掉一大笔早期资本去请顾问。现在，他们手里有 AI 充当几乎所有领域的随叫随到的专家：

- **深度研究**：竞争分析、市场规模、财务模型
- **文档起草**：pitch deck、案例研究、投资人备忘录、PRD

- **战略思考伙伴**：唱反调式分析、事前剖检（pre-mortem）、情境规划、路线图优化

Agentic Coding（智能体编码）

类比：永远在线、从不阻塞的工程师

过去要构建软件，你需要一位技术联合创始人、一家外包开发公司，或者足够长的现金跑道，才能在写出第一行生产代码之前先雇一支工程团队。

如今的智能体编码工具让每一位有志的创始人都可以用自然语言描述他们想造什么，并指挥 AI 去生成、测试、调试、重构一份生产级代码库——以一整个工程团队的速度和规模。从"我有一个想法"到"我有一个产品"的时间线被压缩了。创始人的角色现在聚焦在"该造什么、为什么造"上，AI 则负责处理为真实用户准备好的真实基础设施的实际构建。

工作流自动化

类比：随叫随到、自动化的运营团队

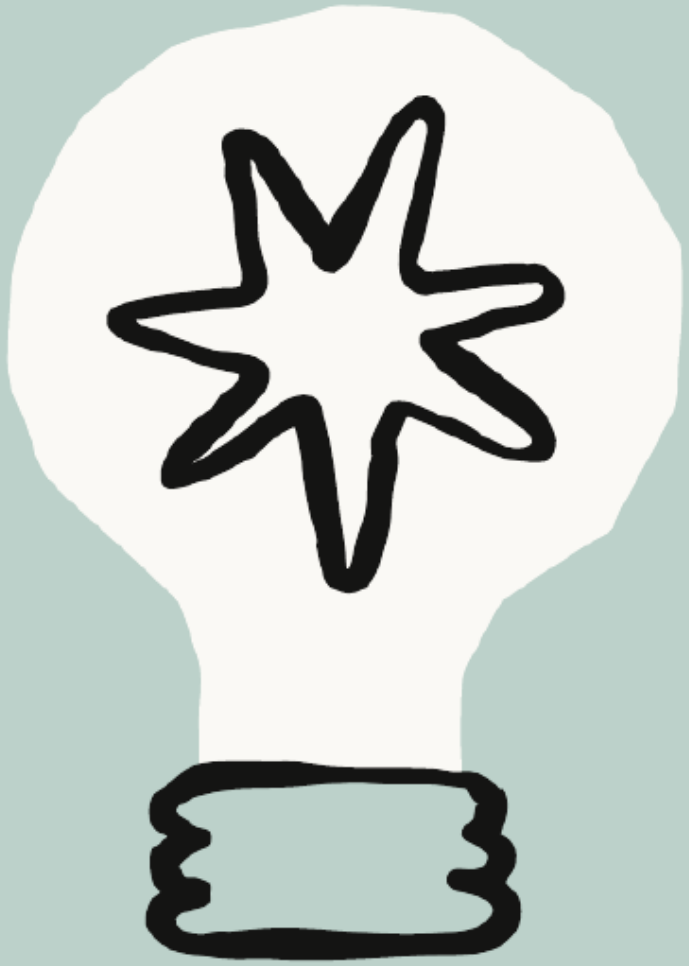
哪怕一位创始人能像顾问一样做研究、像工程团队一样构建产品，仍然有一整类工作既不属于战略规划，也不属于产品开发，但必须完成。日程安排、更新 CRM、汇总周报、维护文档、发布内容、跟踪合规要求、管理公司各种工具与系统之间的连接——这些事都得做。在精简型创业公司里，这部分负担主要落在创始人头上——而它对那些本该用于更高阶决策的时间和注意力，是相当严重的税。

借助 AI 工具进行工作流自动化能卸下这些税。可以让那些重复性的运营任务自动发生：deal 一推进 CRM 自动更新、周报自动生成、产品文档随产品变化同步更新。而且，关键的是，**Claude Cowork** 可以集成创业公司运转所依赖的互联系统——你的项目管理工具、你的通讯栈、你的数据源——而无需有人去构建和维护这些集成。在 Day Zero 创业公司里，这个"有人"几乎永远是创始人本人。

时机与编排才是关键

凡是能有效驾驭 AI 在研究、自动化、智能体编码上的能力的创始人，可以构建一家"杠杆远超人头数"的创业公司。他们也得以把绝大部分时间和带宽，专门投在真正重要的工作上。

不过这些工作不会自动发生：编排这些 AI 工具的创始人需要知道如何（以及何时）应用它们。本手册剩余的篇幅，就在探讨创始人沿着 AI 原生创业之路前进时会遇到的目标和挑战，以及如何在每一个阶段有效地应用 AI 工具。



第 3 章

构想阶段

第三章 ｜ 想法阶段（Idea Stage）

每位创业者都从同一个地方出发：一个让他们魂牵梦绕的问题。这是创业公司的一个阶段——想法在这里与现实相遇：**2026 年要在创业上成功，就需要在证据足以支撑之前，做到不动手构建的纪律性**。

这个阶段的工作是研究、客户发现、竞争分析，以及对反向证据的诚实评估——这一切都得在你让 Claude Code 生成第一行生产代码之前完成。

Idea 阶段的目标

在 Idea 阶段，创始人的主要目标是**以研究为导向的验证**：在投入资源去构建之前，先把"真实问题确实存在（而你的方案能有效解决它）"的证据扎实地拼凑起来。

实操层面，Idea 阶段就是创始人需要按大致顺序回答的一系列问题：

- 这个问题是真实的、具体的、足够频繁，值得围绕它构建吗？
- 是谁真的有这个问题，他们能否构成一个市场？
- 还有别人在解决它吗？如果有，方式和效果如何？
- 一个真正能解决这个问题的方案，需要做到什么？我的想法做到了吗？

这些询问的结果加起来回答一个终极问题：**这值不值得做？**

这意味着在动起来之前先具体化。"人们苦于报销系统"是一个观察；"中型市场公司的财务经理每周花 4 小时以上对账，因为他们当前的工具不能与会计软件集成"则是一个**可被检验的假设**。

Idea 阶段的退出标准

Idea 阶段的退出条件是**找到 problem-solution fit（问题 - 方案契合）**。你已经积累了定性证据——主要来自真实的人类对话——表明你正在为真实的人解决真实的问题，然后再去开始构建解决它的东西。

当你能对以下三个问题都回答"是"时，你就准备好离开 Idea 阶段了：

1. **问题是真实且具体的吗？** 此处要肯定回答，意味着你能说出究竟是谁经历这个问题、他们多久遇到一次、问题对他们影响有多严重、他们目前是怎么应对的。
2. **你的方案是在解决真正的问题吗？** 不是你原本以为的那个问题，而是验证过程揭示出来的那个问题。有时这两者是同一件事，但并不总是。
3. **你掌握的信号是否足以支撑开始构建？** 在此阶段你永远不会有 100% 的确定性，而等待确定性本身就是一种失败模式。但你需要足够多的定性证据，让承诺造一个 MVP 是一项有理有据的决策，而不是一个信仰行为。

Idea 阶段的挑战

Idea 阶段是创业旅程中最重要的工作发生之处，因为那些最具后果的错误也在这里造成：现在做错某事，会迅速把你刚萌芽的创业拖出轨道。

不过，绝大多数 ideation 阶段的挑战都涉及"以超出你理解所能支撑的速度往前冲"——所以那些以审慎和深思熟虑前行的创始人，会获得稳定的进展。

把"构建"误当成"验证"

挑战：当技术阻碍被消除，一个充满激情的创始人很容易跳过创业旅程中最重要的工作：验证他们的想法是否真的是用户需要并会使用的方案。

哪怕在当下这场智能体编码浪潮之前，42% 的创业公司之所以倒闭，是因为他们造出了没人想要的东西。而现在，像 Claude Code 这样的智能体编码方案极大压缩了"我有一个想法"和"我有一个产品"之间的距离——这个失败率只会向上爬。

虽然今天是当一名拥有令人惊艳好点子的创始人最好的时代，但快速、轻松地拼出一个看起来像产品的原型，反而给 AI 原生创业带来了一种"反直觉但却真实存在的"重大生存风险。

直到不久前，"构建"还需要真实的开发时间和预算，把一个最基础的原型攒起来通常要好几个月。如今技术开发的门槛大体不存在了，AI 让创始人非常容易就直接跳进构建，而完全不验证它在现实世界中的实用性。

要达到 problem-solution fit，必须先验证假设再去构建。但许多首次（甚至有经验的）创始人都误以为 AI 可以走捷径绕开这条要求，让流程变成：**有想法 → 立即构建原型 → 把"原型存在"本身当作验证**。原型变成了"假设一直都是对的"的理由，而事实上没人真正测试过这假设到底是否成立。

一个能跑的原型很容易被误以为是"你在解决真实问题"的硬证据——但它不是。原型应该被当作和潜在用户对话时的**压力测试道具**。这些对话本身，才是真正的证据。

过早扩张（Premature Scaling）

挑战：当"构建"是不费力且即时的时候，你的执行就可能远远跑在业务需要之前。

过早扩张意味着在你真正验证一条产品路径值得承诺之前，就把自己锁定到了这条路径上。

这一直是创业公司的杀手。但 AI 让创始人远比从前更容易"在毫无察觉间掉入这个陷阱"。智能体编码助手太强大了，你可以让执行远远超过对 problem-solution fit 的验证而你自己都没意识到偏航。

它会以同样的热情，围绕一个根本性错误的前提去生成、测试、调试和重构代码库，仿佛它正在为一个好点子工作。系统中真正的智慧是**你**。这个阶段的"第一原则"就是：让你的判断走在你的构建之前，尤其当构建变得如此迅速又如此轻松时。

客观性的丧失

挑战：你让 AI 给你一些证据来支持你已相信的东西，它一定能给你找到。**确认偏误（confirmation bias）现在配上了一个研究引擎。**

确认偏误一直是创业公司的职业病：创始人本性上对自己的想法充满激情。如今，AI 工具给了确认偏误一个巨大的 power up。让 AI 验证你的创业想法，它就会去找支持它的证据；让它估算你的潜在市场规模，它就会去找让你的 TAM（总可达市场）看起来值得融资的那个数字。

AI 沿着你给的方向前进——这意味着一个不发出尖锐问题的创始人，现在可以为一个糟糕的想法构造出一份精致、看起来有研究依据的论证，比以往任何时候都更快，同时还感觉自己已经在做尽职调查。

解药就是同一个工具，只是把它指向相反的方向：让 AI 像验证一个想法那样，去对它做压力测试。当研究与结构化的对抗性思考揭示了你的想法需要修订的证据时，那就是**该 pivot（转向）的信号**。

Claude 如何帮助 Idea 阶段的创始人

让你的 AI 原生创业概念走过 Idea 阶段，会让人感觉好像永远没有尽头。你是一个创始人，你只想动手造点东西。但这一开局之关键的阶段，本质上是一项**研究与验证练习**，意味着要去拿那些能帮你"在压上代码之前更严谨思考"的工具。下面是如何在 Claude 各个产品面（Chat、Claude Cowork、Claude Code）使用它，以在做完该有的尽职调查的同时，尽快走完 Idea 阶段。

Chat、Claude Cowork、Claude Code：选哪个 Claude 面

AI 让创业创始人更容易快速出货、自动化乏味工作、以规模化方式运转，但你用哪个"面"也很重要。下面是根据手头任务该选哪一个 Claude 面：

- **Chat**：用于不离开你当前所在 app 的快速交流。适合公司运营中那些层出不穷的小任务：从一份密集投资人备忘里抽出那一句关键 take-away、在董事会前对某个论断做完备性核查、或在一长串 Slack 团队对话里厘清头绪。
- **Claude Cowork**：用于真正需要花时间的知识工作：从多个来源中抽取、综合、产出一份"成品"，比如一份文档、deck 或电子表格。例如把一文件夹的客户访谈转录变成下次产品评审用的主题化发现文档；融资前以十几家供应商的网站为输入构建一份竞争格局图；或一个每周一早上的常驻任务，从你的所有连接工具中汇集指标，把每周 KPI 简报放进一个共享文件夹。
- **Claude Code**：面向你团队中的工程师的智能体编码环境：直接访问代码库、Plan Mode、Git 集成，以及本地 / IDE / 沙箱式云环境。这是精简团队在一个不断成长的代码库上交付特性、把 MVP 时期的遗留代码迁移过来、从原型走到生产而无需等待更多人力的地方。

Chat、Claude Cowork 与 Claude Code：如何选对 Claude 形态

AI 让创业者能更快交付、自动化繁琐流程、规模化运营，但你用哪种形态很关键。下面按手头的任务给出选用建议。

Chat：不离开当前应用就能完成的快速互动。适合公司日常运营中的小任务：从冗长的投资人备忘录里提炼一句要点；董事会前快速核查一个事实；或把团队 Slack 上的长讨论梳理清楚。

Claude Cowork：真正费时间的知识工作——跨多源整理、综合分析，并产出成品（文档/演示/表格）。例如把一堆客户访谈记录整理成产品评审用的主题发现文档，融资前从十几个供应商网站搭出竞品图谱，或周一固定从工具拉数据并把 KPI 简报放进共享文件夹。

Claude Code：给团队工程师用的智能体编码环境：直接连代码库、Plan Mode、Git 集成，支持本地、IDE 或沙箱云环境。让精简团队在不断扩大的代码库里交付新功能、迁移 MVP 时期的遗留代码、把原型推到生产环境，不必等到扩招。

如果任务是……	选择	原因
一个提问、一次改写、一次快速头脑风暴	Chat	快、对话式、零配置
调研、分析，或基于你的文件和系统产出成品文档	Claude Cowork	文件夹访问、连接器、Skills、定时任务
写代码、测试、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；变化的只是它所处的工作空间。

如果任务是…	选用	为什么
一个问题、一个改写、一次快速头脑风暴	**Chat**	快速、对话式，无需配置
研究、分析，或从你的文件和系统构建出的成品文档	**Claude Cowork**	文件夹访问、connector、skill、定时运行
编写、测试或交付软件	**Claude Code**	代码库访问、diff、Git、开发环境

底层是同一个 Claude；变的只是它周围的工作空间。

定义并压力测试你的"问题假设"

你自己的领域专长加上前期的研究，已经产生了一个假设。第一项工作是把它**磨锋利到真正可被检验为止**。Claude 在这里尤其有用，因为它能逼你具体化：**究竟是谁有这个问题、多频繁、多严重、他们目前怎么应对？** 如果一个问题陈述无法精确回答这些问题，那它就还没准备好被验证。

- **练习**：和 Claude 一起把你的问题陈述打磨到它本身就是一个可检验的假设。例如，"合同审查耗时太长"并不能被有意义地检验。但"中型市场公司的内部法务团队，每个合同审查周期要花 3 天以上，因为红线（redline）分散在多个邮件线程而不是单一版本化文档里"——这就**非常可检验**。

下一步是让 Claude 反过来与你的想法对抗，去找否定你假设的**反向证据**。这能浮现出市场层面的负面信号、失败的竞品、客户行为模式、以及结构性障碍——一份只挑支持性证据的综合分析会悄悄把它们打入冷宫。

目标是：在进入客户发现之前，就已经用最强的反方意见对你的假设做过应力测试，这样"信息性的用户访谈"才真正是开放式的，而不是寻找确认的过程。

> 提示：把 Claude 用作有结构的"反方辩手"，在 AI 原生创业生命周期的每一个阶段都是核心用例。

市场调研和绘制竞争格局

评估你的竞争对手

有个创业公司特有的现象叫**竞争忽视（competitor neglect）**：极度专注于自己的愿景和执行，从而系统性地低估了其他人在同一空间里在做什么的倾向。所幸 AI 提供了解药：让 Claude 给出"为什么这一方案空间里的某个竞品会成功、而你不会"的最具说服力的论证。

Claude 可以分析为什么他们的做法实际上更好、为什么客户会选他们、为什么你认为的差异化在防御性上可能没你想的那么牢固。

- **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、以及可能切入你空间的邻接玩家。然后让它论证每个层级**为什么真的对你的成功构成威胁**——而不是那种最容易被反驳掉的版本。

市场调研

Claude Code 可以综合公开可得的客户反馈，浮现出反复出现的抱怨和未被满足的需求。Bonus：这等于是对你竞品客户的免费定性研究。

- **练习**：让 Claude Cowork 跨你的关键来源综合竞品评测，识别出现有方案尚未解决的 top complaints。如果你的假设恰好解决了其中一个或多个，这就是 problem-solution fit 的有力证据。如果没有，那也值得知道。

Claude Cowork 还能从密集的行业报告、分析师文件、市场研究文档中抽取相关信息和数据；接着，这些干净、综合过的输入对 Claude 的分析工作而言就是理想的上下文。

- **练习**：基于公开数据构建 TAM/SAM/SOM 模型，并对它们背后的假设做压力测试。识别市场是在扩张、整合还是已成熟；这个 context 会影响你对时机和差异化的思考。绘制买方格局：谁掌握预算？谁影响决策？这两者是不是同一个人？

趋势分析

最后，用 Claude 去监听那些告诉你"是否正在合适时机进入市场"的早期信号。跟踪那些已经在讨论你的问题的 subreddit 和 LinkedIn 群组，以及用户在描述他们问题时**实际用到的语言**。让 Claude 识别出"类似问题已被解决过的相似市场"，并提取哪些奏效、哪些不奏效。浮现出可能加速或威胁这个机会的监管、技术、人口结构趋势。

- **练习**：让 Claude 找出三个外部趋势——监管、技术或人口结构——它们可能在未来两年显著影响你的市场，并评估它们对你的具体假设来说，是顺风还是逆风。

> 提示：本节的市场调研与竞争映射不是一次性练习。你会在 MVP 和 Launch 阶段继续不断有所发现并演化你的思考，所以每当假设进化时都要重复这些练习。

规划与设计客户发现

你通过与潜在用户交谈所学到的东西的质量，由两件事决定：(1) 你问的问题的质量，以及 (2) 你是否问对了人。Claude 在做客户发现方面特别有帮助，包括跟谁谈、问什么、以及怎么解读你听到的。

跟谁谈

一份精确的目标画像比一份冗长的联系人清单价值高得多，包括具体的岗位头衔、公司类型、团队结构、最有可能强烈体验到这个问题的资历层级。在此基础上，识别出这些人**真实可触达**的地方——社区、活动、LinkedIn 群组、他们聚集的 Slack 工作区——并基于"他们与问题的距离"建立一个优先级框架。

问什么

定义好目标后，用 Claude 来构建访谈框架本身：合适的问题、合适的顺序，结构化地浮现"人们实际做了什么"，而不是"他们以为自己会做什么"。一个新手创始人的错误是问关于未来的、通用的、开放性问题（"你会用这种东西吗？"），而不是去具体询问相关的过去（"告诉我你最近一次遇到这个问题是什么时候"）。

Claude 还能标出你拟的问题中哪些在带路（leading）、过宽、或者更可能产生噪声而非信号的。Claude 还能帮你设计**追问式问题**，去探问回避或者对关键问题深挖含糊的回答。

如果你的假设涉及不止一种 persona，Claude 也能为每一种设计不同的问题集。一位 finance manager 和一位 CFO 与同一个问题之间的关系是不同的，单一访谈框架会抹平这种区别。

- **练习**：先手动起草访谈问题，再让 Claude 审计它们。具体地让它标出任何带路的、面向未来的、太宽泛的、或更可能产生"社会期望性回答"而非诚实回答的问题。然后让它针对访谈中最可能产生回避的两到三个时刻，建议一个追问。

访谈后分析

每次对话之后，用 Claude 来做 debrief：把你的笔记喂给它，让它识别什么确认了你的假设、什么挑战了它、什么是真正令人意外的。一旦你积累了一批访谈，把完整的访谈笔记集合丢进 Claude Cowork，让它浮现出反复出现的主题、矛盾、以及两个方向上最强的信号。然后再把综合后的输出交回 Claude，让它指出"你对数据的解读是否在向你希望听到的、而非真实存在的内容做模式匹配"。

- **练习**：每做完 5 次访谈，让 Claude Cowork 综合你的笔记，产出两张清单：支持你假设的证据和挑战你假设的证据。如果第一张比第二张明显更长，问 Claude：这种不对称是数据中真实存在的，还是你**当时希望找到**的？

客户触达与排期

用 Claude Cowork 来自动化"建联系人清单、做外联、安排用户访谈"周围的运营工作量。

Claude Cowork 可以使用你与 Claude 一起定义好的目标画像（包含岗位头衔、公司类型、资历层级），去调研并整理出一份结构化的潜在客户清单，附带已验证的联系方式。然后它会大规模起草个性化的外联邮件，每一封都根据对方的角色和上下文做剪裁。

当回复进来时，它通过 MCP 连接 Gmail 和 Google Calendar，去管理邮件线程、处理排期请求、把访谈塞进日历。这个工作流接着继续：Claude Cowork 按设定的节奏生成跟进草稿（例如对未回复的联系人 day-seven 的跟进），并随每一步完成而更新你的追踪表，让你随时知道每位潜在客户在 pipeline 中的位置。

- **练习**：把你已验证的访谈目标画像交给 Claude Cowork，让它构建一份潜在客户清单、起草一段个性化外联序列，并搭建一张追踪表，列包括外联状态、跟进节奏、访谈完成度。然后让它去跑协调工作，你专心准备对话本身。

设计你最终的方案概念

你已经做完了验证工作：问题是真实的，你知道是谁有这个问题，并且证据支持一个方案概念。用 Claude 从各个角度去发展和挑战你的方案概念：差距在哪？还有什么替代方案？为了让这个方案在规模上奏效，必须有什么前提成立？

这是一个重要的"现实校验"检查点：你设计的东西是否真的在回应**验证过程揭示出的问题**，而不是你最初带着进来的那个假设问题？

- **练习**：把你的方案概念呈给 Claude，让它找出你的设计最依赖的三个假设。然后问每个假设要成立必须有什么前提，以及任意一个假设不成立时的后果是什么。

用 Claude Code 构建一个轻量原型

现在到了好玩的部分：有了一个已验证的假设，并经过应力测试的方案概念，你终于可以动手造东西了。

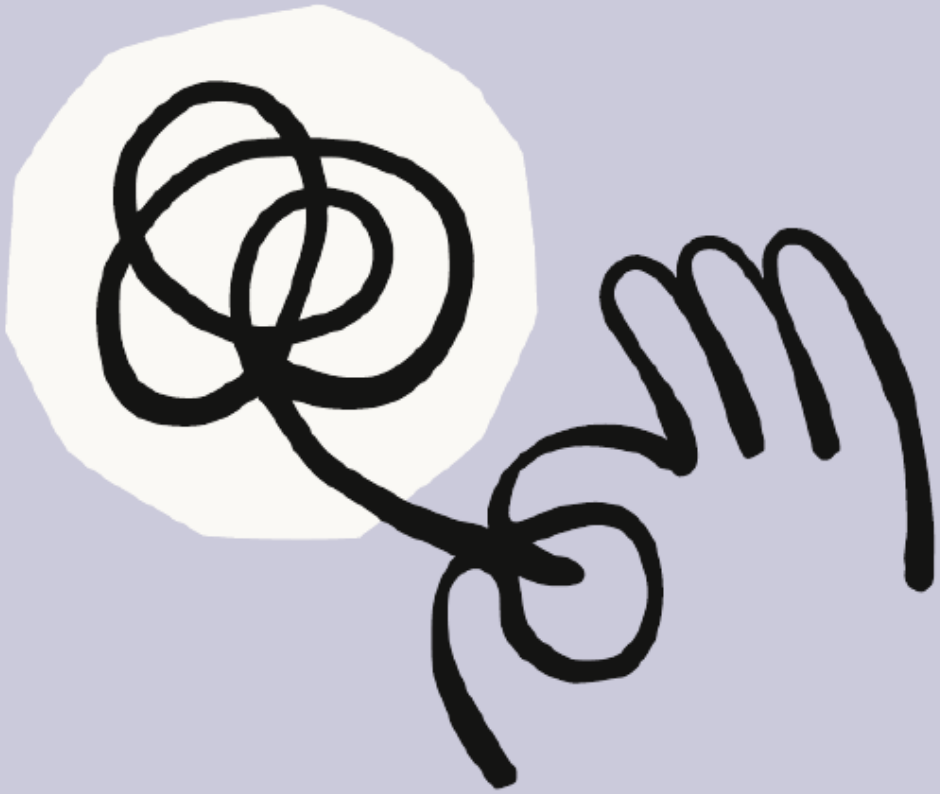
这就是 Idea 阶段里 Claude Code 登场的时刻。哪怕你一直在私下捣鼓，现在才是你产出"正式轻量原型"的时刻：把你的想法摆到一个真人面前并拿到真实反应所需的**最小表面积**。

你还没在造一个真实世界的产品（暂时还没）；你在构造一个能用于"客户和投资人对话"的、你想法的功能性样品。**真实用户对一个他们能真正触摸的东西的反应，会告诉你十几次"问题发现访谈"都告诉不了你的事**。之前，你在确认你要解决的问题是真实的；现在，你是在请潜在用户与你提议的方案进行互动。

- **练习**：定义你的方案最依赖的那个核心交互。让 Claude Code 只构建那一个。当你拿到它后，把它放在 5 位来自你已验证目标画像的人面前，让他们试用。这 5 次对话中你学到的东西，决定你是继续构建，还是回到画板上重来。

Idea 阶段的尽头是 AI 创业马拉松中的一次大跨越，因为现在你不是在赌一个直觉，你是在依据证据执行。

现在进入 MVP 阶段，在这里创始人的指引性问题从"这值不值得做"变成"我们应该先具体造什么"，而 AI 的主要角色也从研究伙伴转变为**施工队**。



第 4 章

MVP 阶段

第四章 | MVP 阶段

很多创始人把 MVP 阶段当成一个施工阶段，但 MVP 阶段从根本上仍然是一项证据采集练习。不同之处在于，你现在采集的是关于**方案**的证据，而不是问题空间的证据；具体来说：一组真实且可识别的人是否觉得它有足够价值，从而**使用它、回到它、为它付费、并/或把它告诉别人**。

MVP 阶段的目标

作为 AI 原生创业公司的创始人，你的目标是**把已验证的问题转译成一个真实用户实际会用的可运行产品**。这不是一个把所有 roadmap 特性都做齐的完整版本，而是把"真实方案"摆到"真实用户"面前、生成关于 product-market fit 真实证据的、最小、最聚焦的版本。

同时，你**现在如何构建，决定了将来什么是可能的**。这意味着 MVP 阶段还有第二个同等重要的目标：**在不积累那种会复利的技术债务的前提下快速行动**——一旦真实用户开始大量到来，这种技术债会随时回来纠缠你。

最后，从第零天开始就投资**持久化的上下文**，是让 AI 保持为力量倍增器、而不是熵的来源的关键。在 AI 原生创业公司里，你的代码库是你一次又一次会话地与 AI 协作的对象，所以**可读性是基础**。那些跳过规约（spec）、架构决策、上下文文件（如 `CLAUDE.md`）的创始人会撞上可预期的墙：每个新会话都要重新解释一遍代码库，AI 生成的修改也渐渐偏离原始愿景。

MVP 阶段的退出标准

MVP 阶段的退出条件是 **product-market fit 的真凭实据**：证明一组具体、可识别的用户认为这个产品有足够价值，因而**回到它（留存）、为它付费（收入）、或把它告诉别人（推荐）**。

MVP 阶段的挑战

在 MVP 阶段，创始人的首要指令是**速度与判断**。这里的挑战集中在：你是否能以足够快的速度造出对的东西、用对的方式，又不偷那种之后让你付出代价的捷径。

智能体式技术债

挑战：因为 AI 基本上消除了过去控制"什么能进生产"的每一个天然瓶颈，速度被保证了。但当**速度成了创始人对 MVP 构建唯一变量**时，他们就会积累很难还清的技术债。

在 MVP 阶段，一定的技术债是合适的，前提是理解它必须在 scale 之前管理掉。它会逐步累积，可以一段时间慢慢清理，或在专门的 sprint 里清掉。**而 AI 技术债会复利**。

如果规约和架构约束没有写在 AI 能读到的地方，每一次会话都会从零重新推导基础决策，这些决策就开始漂移。最终你得到一个**没有任何连贯心智模型**的代码库——不是因为任何单个部分糟糕，而是因为这些部分从未被设计成能拼到一起。这是真问题，并且往往后期才浮现。

落入虚假的 product-market fit

挑战：AI 工具能产生很漂亮的早期数字，但这些不是市场需要你产品的保证。

早期势头是创始人能拥有的最具心理冲击力的体验之一。在做了几周或几月的验证工作和谨慎、有纪律的构建之后，发布一个产品感觉就像在确认"你一直是对的"。

智能体编码工具能让你比以往任何时候都更快达到这一刻——但**早期 traction 不等于 product-market fit**。Launch 时的能量来自一些短暂的力量，比如你创始人圈子里的朋友、你投资人 portfolio 中其他公司里的潜在买家、或者一条把流量推高的 Hacker News 头条。不幸的是，这些都不能可靠地预测：在最初的爆发褪去之后，第 6 周或第 12 周会发生什么。

零摩擦的范围蔓延（Zero-friction Scope Creep）

挑战：当构建感觉毫不费力且几乎免费，永远会有一个"还想加上去的酷特性"或者"还想处理的一个边界情况"。这种范围蔓延弊大于利。

范围蔓延一直是创业风险。不同的是，传统上对它的强制约束力——工程时间的真实成本——在加一个特性只要一个下午（而不是一个 sprint）时已经不复存在。

这里的挑战是：每一个单独的添加都"说得过去"。产品当然应该处理这个边界情况；用户当然会想要那个工作流。借助智能体编码，每一项添加做起来都那么轻松，因此在当下不会被感受为范围蔓延——但随着产品膨胀超出最初的边界，你就会失去方向感和势头。

解药是：**在开始构建之前**就写下一份明确的范围定义，描述这个产品**做什么、刻意不做什么、以及来自真实用户的什么具体证据才会成为加新东西的理由**。这把决策点从"我们应该构建它吗？"变成"是否有关键多数的用户告诉我们：没有这个他们就无法从产品中得到价值？"

因经验不足而不安全

挑战：那些用 AI 工具把应用赶上市、却不先理解基本安全原则的创始人，最终把他们的用户暴露在可预防的风险中。

残酷的事实是：智能体编码工具生成的是**能工作的代码**，不是**天然安全的代码**。功能性代码很容易，因为要么特性能工作要么不能。安全漏洞是不可见的，直到它们被利用为止——这意味着没有自然的反馈回路提醒一个新手创始人"出问题了"。但把一个真实 MVP 推给真实用户，意味着真实数据、真实暴露、和真实后果。

忽视安全并不是 AI 原生项目独有的。每个时代的自筹资金创业公司都常常把安全考虑拖到构建后期，有时甚至拖到生产发布的边缘。**在任何用户碰你的 app 或方案之前做一次安全审查，是把一个 MVP 放进现实世界的最低责任门槛**。

Claude 如何帮助 MVP 阶段的创始人

在你开始构建之前定义架构

在 Claude Code 写下一行生产代码之前，使用 Claude 来**定义和记录**那些将治理本阶段所有构建工作的架构决策：要遵循的模式、要避免的依赖、所做的权衡及其理由。这一输出将成为一份聚焦的**架构上下文文档**，并建立 Claude Code 在其中运作的护栏。

没有这层 context，每个会话都从零开始，Claude Code 不得不去推断它自己的结构假设。让 Claude Code 在没有护栏的情况下构建，会产生一个能工作但结构上不连贯的代码库——而在不连贯的代码库上迭代和扩展，最终是浪费时间和 token。迟早会有一个点，代码不可避免地崩塌，迫使你从零重建。

- **练习**：打开 Claude Code 之前，先打开 Claude，描述你在构建什么：它解决的核心问题、它服务的用户、以及你在接下来 6 个月内现实预期的规模。让它帮你定义应该治理你这次 MVP 构建的架构原则、考虑到你的约束应该避免的依赖、以及你在这一阶段有意接受的权衡。

接着，把这个输出存为 `CLAUDE.md` markdown 文件。这就是你的架构上下文文档：你这次构建的第一个产物，也是后续每一个会话所依赖的对象。`CLAUDE.md` 文件作为 Claude Code 的项目级指令，提供项目特定的 context 与指引；当 Agent SDK 在某个目录下运行时会自动读取它们。功能上，它们是你项目的**持久化"记忆"**。

定义并执行你的 MVP 范围

无摩擦的范围蔓延是 AI 时代 MVP 的定义性失败模式之一。就像你定义并记录了产品的应用架构一样，你也要在写下第一个特性之前**先定义你的 MVP 范围**。

Claude 能帮你创建一份 scope 文档，描述你的 MVP 产品**做什么、刻意不做什么、以及"特性修改标准"——即来自真实用户的什么具体证据才会让此时加东西变得合理**。

当新特性想法浮现时（一定会），你用 Claude 去压力测试：这究竟是用户的真实信号，还是创始人热情伪装成的产品思维？

用 Claude Code 构建你的 MVP

一旦架构和范围都已定义，Claude Code 就成为 MVP 的主要构建工具。用它来生成、测试、调试、迭代你的代码库——但把每个会话当作**对你已经做出的产品决策的执行**，而不是**临时加新东西的机会**。

每个 Claude Code 会话开始时：(1) 先回顾你的 scope 文档；(2) 把你的 `CLAUDE.md` 架构上下文文档提供给模型。每个会话结束时，用本次会话浮现的任何决策更新它。**目标是一份你能解释其结构的代码库，而不仅仅是一份能跑的代码库。**

- **练习**：为 Claude Code 工作创建一份简单的会话模板，包括：架构上下文文档、本次会话的具体任务、要遵守的任何约束或模式。每次会话结束时，给上下文文档加一条简短的日志条目，详述本次构建了什么、做了什么决定、引入了什么假设。每会话花 5 分钟做文档化，是对抗"复利成不可管理代码库的架构漂移"的**便宜保险**。

任何用户接触之前先做安全审查

作为一名 AI 原生创业创始人，你的责任是知道你代码库里有什么、理解任何潜在的暴露向量、并不向真实用户（他们把数据交给你）发布显而易见的漏洞。

Claude 能对 AI 生成的代码做一次有用的初轮安全审查，并能识别常见漏洞。这是发布前要养成的好习惯。但它**不能替代专门的安全工具**，更不能在更高风险下替代人类评审者——把它当成替代品的创始人，就是那些会上数据泄露新闻的人。

Claude Code Security 更进一步：它扫描代码库以发现安全漏洞，并为人类评审建议针对性补丁，揭示传统方法可能错过的问题。

> 注：本电子书发布时，Claude Code Security 处于有限 beta 阶段，请在把它嵌入工作流前查看当前可用性。

- **练习**：在部署到任何真实用户之前，让 Claude 对你的核心应用代码做一次专门简报式审查：审查认证与会话处理、API 响应中的数据暴露、输入校验与注入风险、以及已知有漏洞的依赖。把每一项发现都认真对待，评估是否需要修复；**对任何涉及认证、密钥、或数据处理的部分，必须有人工评审。**

在发布前构建你的度量框架

那些**把早期 traction 误判为 product-market fit** 的创始人，通常也是那些**在发布之后才开始追踪数据**的人——他们挑的指标用于评估"什么正在工作"，而不是用于暴露"什么没在工作"。解药是：**在第一个用户出现之前**就建立你的度量框架。

用 Claude 来定义哪些指标对你这具体产品重要，benchmark 是什么，以及哪些数据模式才构成真实的 product-market fit，又有哪些只是迷人的噪声。具体地：在发布 MVP 之前就设定你的留存基准、激活标准、以及 Day 7 和 Day 30 的目标。

接下来，定义对于你这个具体产品来说"假阳性"长什么样：例如，没有激活的注册、没有留存的收入、没有重复使用的初始热情。当数据进来时，让 Claude 对你自己的 traction 提出反方论证：怀疑者会怎么解读这些数字？

管理发现与用户反馈的运营

一旦真实用户进入产品，运营层就会迅速扩张。Claude Cowork 处理那种"重要但乏味"的工作，比如构建并维护用户联系名单、跑外联序列、安排反馈会、归类 bug 报告、追踪迭代周期。Idea 阶段用于管理发现 logistics 的同一批 MCP 集成在这里同样适用。

保持人类在收集回路中，以便对用户反馈做细微的探索。比如有用户说"这很好，但我希望它还能……"，这就需要解读：它是一个核心需求还是 nice-to-have？是这位客户特有的，还是某个细分群体的代表？缺失的特性才是真问题，还是 onboarding 上游里有别的问题？没有工具能回答这些问题。

- **练习**：配置 Claude Cowork 跑你的 MVP 阶段反馈回路：起草给早期用户清单的外联、安排反馈会、为 bug 报告和功能请求设计结构化的收件流程、并撰写每周的输入综合。先你自己看 synthesis；之后你可以让 Claude 分析这些信息，帮你抓住任何被你忽略的重要点。

朝着"证据"迭代，而不是朝着"完工"迭代

MVP 阶段结束的时机是：你拥有 **product-market fit 的真凭实据**，无论这个产品在感觉上"多完工"。宣布你已达成 product-market fit 并准备从 MVP 阶段迈入 Launch 阶段，最终是一项把创始人直觉与已采集证据结合起来的判断练习。不过有几条有用的试金石：

- **Sean Ellis 测试**：问你的活跃用户："如果你不再能用这个产品，你会作何感受？" 如果超过 40% 回答"非常失望"，那是一个有意义的 PMF 指标。
- **努力程度测试**：在 product-market fit 之前，留存要靠持续干预，包括频繁外联、激励、人盯人的跟进，以及创始人耗尽英雄式精力让用户保持参与。在 product-market fit 之后，产品开始**自己做这件事**。当事情从"推"变成"拉"，这种努力程度的转变就是"某种真实变化已发生"最清晰的信号之一。

最终，没有任何单一数据点能确认 product-market fit，因为它是一种必须**跨多个迭代周期都成立**的模式，你才能明确地这么说。

当证据要求时就 pivot

如果你做了这么多工作，还是看不到 product-market fit，怎么办？你的结果没确认你最初的方向，这**不是失败**，这是**系统在工作**：MVP 阶段就是被设计来在你**把错误答案投资过深之前**浮现这种信息的。

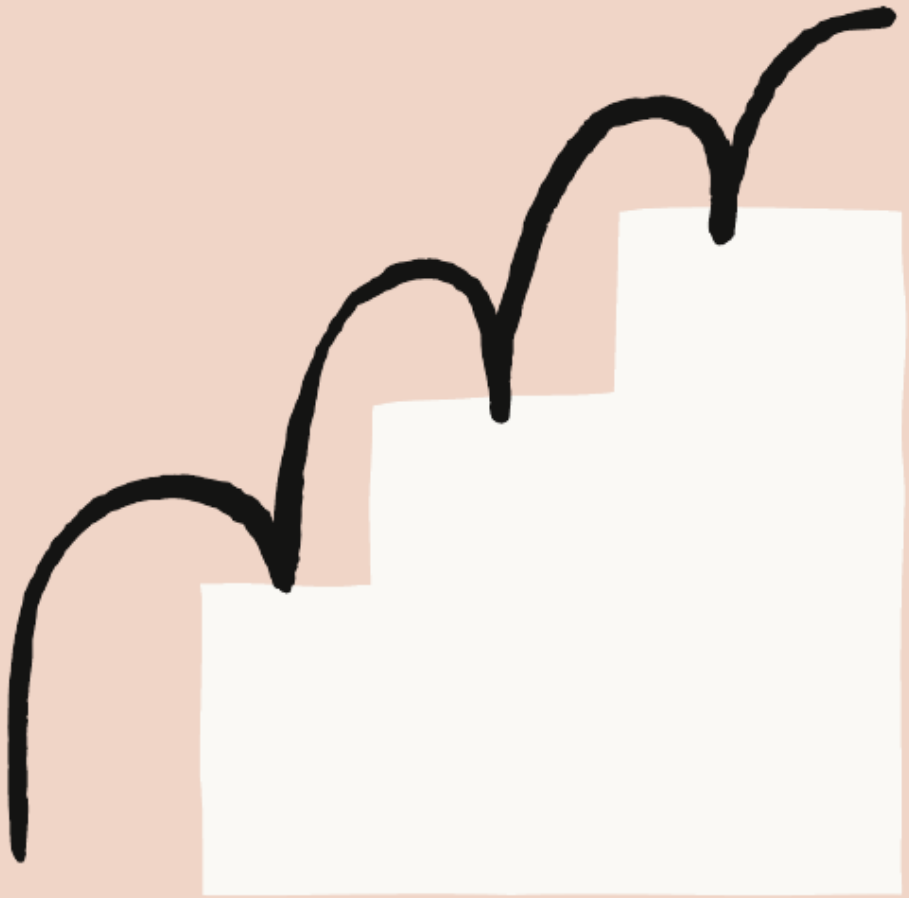
当数据不支持你当前的产品时，用 Claude 来梳理数据在告诉你什么。

- **探索不同的客户细分**：也许那些没在转化的用户从一开始就不是合适的目标。常常合适的受众其实就在你的数据里，只是权重被低估。
- **调整产品的价值主张**：也许你有正确的受众，但 MVP 没和用户产生共鸣。对 onboarding、文案、或核心特性的强调做调整，可能就能修这个问题，而无需改你已造的东西。

也要对一种可能性保持开放：脱节深到需要更根本的改变。

- **练习**：如果你已经完成了 3 个或以上迭代周期，而朝 product-market fit 基准的实质性推进很有限，使用 Claude 在决定下一步之前先做一次诊断。把你的留存数据、用户反馈和最初的问题假设都喂给它，问它三个问题：
  - 这些数据里是否存在一个细分群体，其响应与其他人显著不同？
  - 设计价值与体验价值之间的差距，是定位问题，还是产品问题？
  - 当前产品要找到真正的 PMF，必须发生什么前提条件？基于你所看到的，这个情景现实吗？

让答案决定你是调整、pivot，还是回到 Idea 阶段。



第 5 章

发布阶段

第五章 | 发布阶段（Launch Stage）

如果 MVP 阶段是要证明你的**产品**值得存在，那么 Launch 阶段是要证明你的**业务**值得增长。

Launch 阶段的目标

在 Launch 阶段，创业创始人必须把早期 traction 转化为一个**可复制、可持续的增长引擎**。除了让你的产品做到生产就绪，你还必须把它下面的基础设施变硬，同时**围绕产品真正建立一家公司**。

创业公司在 Idea 与 MVP 阶段天然以创始人为中心，因为你需要完整的态势感知与紧致的反馈回路。但到 Launch 阶段，仍然事事亲自抓的创始人会变成瓶颈。目标不是让你从公司里抽身，而是构建那些**释放你注意力、把它留给只有创始人能做的决策**的运营系统。

Launch 阶段的退出标准

Launch 阶段的退出条件有三个元素：

1. **增长是可复制且由渠道驱动的**。你不是只在留存用户，而是通过明确的渠道在**可预测地获取**他们，单位经济学清晰：CAC、LTV、payback period 都是你能说得出、并能为之辩护的数字。
2. **产品能承受生产负载**。基础设施已加固，安全与合规就绪，可靠性在**真实生产条件**下成立（而不仅仅是你测过的条件）。
3. **运营无须创始人瓶颈**：流程已存在、自动化已就位。你不再是个人在处理客户支持、问题分类、sprint 规划或汇报的那个人。

Launch 阶段的挑战

找到 product-market fit 是早期创业生命周期中最难的问题。现在，创始人的挑战变成了**守住它**。Launch 阶段是这样的地方：找到了真实产品 traction 的公司，如果围绕产品的组织跟不上，也可能在此分崩离析。下面是要警惕的几种失败模式。

技术债到期

挑战：MVP 时期为速度和验证而造的代码库，跑得够好以证明产品是可行的。但生产流量、新特性和不断增长的复杂度，正在把那些当年的捷径暴露出来。

在 MVP 阶段，积累一些技术债换取速度是合理的权衡。在 Launch 阶段，那笔债开始**计息**，越是不处理，修起来越贵。

解决方案包括：系统性的架构审计来识别结构弱点；针对性重构来修最糟糕的部分；以及有意义地扩展测试覆盖，确保下一轮特性工作不会重新引入同样的问题。

创始人变成瓶颈

挑战：在 MVP 阶段，创始人在每一个回路里都是资产。在 Launch 阶段，随着支持量增长、产品决策堆积、运营复杂度成倍上升，同样的本能就变成了约束。

从"自己做这件工作"过渡到"设计做这件工作的系统"，是创业生命周期中最难的转换之一。因为很少有一个清晰的时刻标志着这一切的发生，**风险是你完全错过这个转换，留在 builder 模式里，而周围的组织停下了脚步**。征兆包括：本该 1 小时完成的决定现在要等 1 周才能轮到；只有你知道答案，所以支持工单越积越多；只有你亲自记得做的运营任务才能完成。

解药是做一次"全开"式审计，从最微小的任务到最高风险的决策，盘点你亲自处理的所有事情，识别什么可以系统化、什么可以委派、什么仍然真正值得创始人时间和注意力。

安全与合规不再能延后

挑战：把安全与合规的措施保持简单，对 MVP 来说还可以。但当真实用户、真实数据、可能还有摆在桌上的企业合同时，这就变成了**负债**。

在 MVP 阶段，beta 用户寥寥、生产中没有敏感数据，安全漏洞还只是理论风险。但一旦你的产品进入生产、有真实用户依赖它，假设就变成了**非常真实**的暴露风险。此外，对原型不适用的合规要求，一旦你在处理客户数据、处理付款、或卖给受监管行业，**立刻**就适用。

解药是：在生产规模到达之前——而不是之后——做一次系统性的安全与合规审查，并把任何浮现的问题当作**强制修复**——不是"建议"——必须在下一波用户到来之前完成。

在你还没准备好的时候扩张

挑战：新市场和融资机会看起来像增长机会。它们也可能是 product-market fit 的死亡之地。

你已积累的初始 traction 是真实的，但它**只针对你早期的受众**。过早地扩展到一个与你最初市场显著不同的市场，引入了新的用户行为、合规要求、支付基础设施、以及基线预期——这些都不是你产品被设计为应对的。突然有太多新变量，你就失去了清晰解读自己数据的能力。你也面临这样的风险：在追逐一个新的、未经验证的受众时，**忽视了你最初的用户基础**。

Claude 如何帮助 Launch 阶段的创始人

在 Launch 阶段，Claude 的三种形式都在全力使用，并且相互支撑：每个工具产出的输出会成为另外两个的输入。结果有机地复利——同时使用所有三个工具的创始人，得到的远超三者之和。

这正是**超精简创业模型在结构上成为可能**的原因。当 Claude Code 构建产品、Claude Cowork 围绕它构建公司、Claude 帮助把产品与组织知识运营化时，一个小团队可以以"远超自身规模"的样子运转。

在技术债复利之前处理它

你的 MVP 代码库能跑，但它也需要一次系统性的处理 pass，找出任何可能成为结构性负债的技术债。

首先，用 Claude Code 跑一次完整的架构审计：识别代码库哪里脆弱、哪些捷径会变得维护昂贵、哪里测试覆盖薄到下一轮特性工作会重新引入相同问题。

把 Claude Code 的审计发现交回 Claude，去分诊并安排修复工作的顺序：哪些必须在下一次发布前修、哪些可以等一个 sprint、哪些在你当前阶段是可以接受的持续技术债。

这也是把你在 MVP 阶段做过的架构决策**正式落档**的时刻（那些一直在你脑袋里、当时没时间写下来的决策）。现在把它们写进 `CLAUDE.md`，可以确保未来每一次 Claude Code 会话都从对**系统如何被设计以及为什么**的共同理解出发。

- **练习**：让 Claude Code 审计你的 MVP 代码库，产出一份按优先级排序的结构弱点、测试覆盖差距、和重构候选清单。然后把这份清单交给 Claude，让它在你接下来的若干 sprint 中给修复工作排序：必须首先处理的重大问题、可以与特性开发并行处理的、可以等待的。

构建那些代替创始人注意力的系统

要构建那些**释放你注意力以处理只有创始人能处理的责任**的运营系统，前提是知道你的注意力究竟去了哪里。用 Claude Cowork 对你当前的运营负载做一次结构化审计，记录每一项重复任务、每一项落到你桌上的决策、以及每一个只因你亲自记得才发生的工作流。然后让 Claude Cowork 把这份清单归类为：可完全自动化的、需要人但不一定是你的、真正需要创始人判断的。

审计完成后，使用 Claude Cowork 为自动化候选设计工作流逻辑：什么触发每个工作流、决策规则是什么、输出长什么样、完成后去哪里。

把安全与合规变成产品工作流

用 Claude Code 浮现出代码层级的问题，这些问题常出现在 SOC 2、GDPR、HIPAA 审计以及你目标市场要求的标准里。这会同时浮现出漏洞和合规差距。把这些发现交给 Claude，让它帮你给修复工作排序，并设计企业买家在签字前会要求的控制、审计日志、和访问管理。

> 注：AI 扫描是辅助，不能替代有资质的合规评审。

接着，把合规工作流嵌入到你的开发周期里，而不是把它当作一次性项目跑——合规文档需要持续维护和更新。对那些正在接洽企业合同或国际市场的创始人，这也是 Claude Code 的安全扫描能帮你为独立安全评估做准备的时刻。

- **练习**：用 Claude Code 跑一次面向你目标市场所要求框架的代码级安全审查。把输出交给 Claude，让它产出两样东西：一份按优先级排序的安全修复序列；以及一份"你需要产出的文档与控制清单"，以满足来自潜在企业买家的合规评审。

把你一直在跳过的产品管理流程立起来

Launch 阶段需要一组**轻量、可重复**的流程，能在没有创始人干预的情况下被触发或运转。用 Claude 来设计：你的产品时间线和工作周期如何组织、一份规约（spec）在 Claude Code 接触特性之前需要包含什么、bug 报告如何归类与路由、每周指标报告涵盖什么以及如何分发。

流程设计完成后，用 Claude Cowork 来构建并运行运营层：安排 sprint 仪式、把进来的 bug 报告路由到合适的地方、从你连接的数据源编排周度指标、并维护把用户信号送进产品决策的反馈回路。

- **练习**：让 Claude 设计一套轻量产品管理操作系统：明确的 sprint 节奏、最小规约模板、bug 分诊决策树、以及从你真实数据源汇总的周指标简报。然后设置 Claude Cowork 实现并运行这套系统的循环性运营元素（如排期、路由、报告编排），按计划自动发生，无需你出手。

第 6 章

规模化阶段

第六章 ｜ 规模化阶段（Scale Stage）

在 Scale 阶段，创始人的角色从 builder 重新回到**公开面向的高管**。产品仍然居中，但你个人日常工作越来越多地是关于公司本身。即使你仍力图维持精简、以 AI 为中心的结构性优势，你的注意力也必须向新的 Scale 阶段活动扩张，比如分析师 briefing 和 IPO 路演。

Scale 阶段的目标

技术基础设施的规模化工作仍在继续，现在又加入了**把组织本身规模化、并让它成熟为一家企业**的工作。

在 Scale 阶段，你面对的是从数千用户走到数百万，从一个市场走到多个市场。在之前的每个阶段，增长是你可以"用感觉走"的：贴近用户、根据紧致反馈回路加上一剂健康的创始人直觉来调整方向。而现在，目标是**构建由成熟组织运营所支撑的系统化增长**。

对一家 AI 原生创业公司而言，你的目标应当是通过**累积的深度**构建一条可防御的护城河：来自你嵌入产品的专业知识、你产品与用户依赖的其他工具与平台的集成深度、以及专有的系统数据与工作流。那些一直朝同一方向、在一致基础设施上持续构建的创始人，现在拥有了**真正难以复制**的东西。

到这个阶段，公开市场投资人、分析师、监管者、企业采购团队、收购方会施加更大压力——以及更大怀疑——因为现在赌注更大了。你的产品和组织必须经得起外部审视：不只是你造的东西的能力，还有围绕它的治理、合规姿态、财务控制、战略叙事。

Scale 阶段的退出标准

Scale 阶段的退出条件不再是一个单一里程碑，而是一个**阈值事件**：即使创始人越来越不直接运营日常事务，公司也是可持续的。你已展示出系统化增长；构建出能满足最苛刻外部审查者的组织治理与合规基础设施；并且对"如果一个资金雄厚的在位者今天复制你的产品，你的用户会留下吗？"这一问题有坚实的答案。

实践中，这一阈值通常以三种形式之一到来：在不再需要外部资本的规模上实现的可持续盈利、IPO 就绪、或被收购。**三种都要求**：你的增长系统化且可审计、你的产品护城河经得起审视、你的组织在运营上成熟可持续。

当这一切为真，恭喜的话该说了：你的创业从"一场赌注"变成了"一门生意"。

Scale 阶段的挑战

把运营层委派出去

挑战：Scale 阶段的运营系统必须**可靠且可持续地无人看管运行**。对从第一天起就亲力亲为的创始人来说，这个过渡既是结构性挑战，也是心理挑战。

你在 Launch 阶段的工作是把系统**创建**出来；在 Scale 阶段，工作变成了 (1) 让这些系统成熟到可完全信赖；(2) **真的去信任它们**。

这比听起来难。哪怕你是一个善于委派的创始人，**什么该 hand off、什么该留在自己盘子上**，也并不总是显而易见。把太多东西交得太快，特别是给 AI 自动化的系统，关键决策可能在缺少只有创始人能提供的关键 context 时被做出。但如果握得太久，你就成了瓶颈。

这里的根本挑战是：**识别那些只存在于创始人脑袋里、或未文档化工作流中的"机构知识"，然后把它编纂成有文档、可审计、可转交**的系统。

扩展技术运营

挑战：客户不再只评估你的产品本身；他们想知道你的组织能否成为一个**可靠的基础设施合作伙伴**。

前三个创业阶段的技术挑战围绕代码库本身：构建对的方案、不积累技术债、然后为真实用户加固安全与合规。到了 Scale 阶段，挑战变成了围绕代码库的一切：创建**支持基础设施、文档、可靠性保证**，这些都标示着成熟度。

签订多年合同的大客户和机构买家在签字之前会要求这些，并且签字之后也会要求你兑现。让你走到此处的同一套 AI 基础设施，现在帮你构建有明确响应时间的专门支持职能，以及一个新客户工程团队真的能用的文档。

扩展组织职能

挑战：处于 Scale 阶段的公司通常需要**组织基础设施**——招聘、薪资、会计、法务——而无论运营它的人数有多少。

在 Launch 阶段，"系统化运营"指的是把消耗创始人注意力的工作流自动化。Scale 阶段的创业公司现在需要发展更广、在某种程度上更具后果的一整套运营职能：财务报告、合规监控、合同管理、客户支持，等等。

构建 GTM 职能

挑战：**有机增长有天花板**，大多数 Scale 阶段的创始人在他们曾经构建过一个真正的 go-to-market 职能之前就先撞到了这个天花板。

Idea、MVP 和 Launch 阶段的增长通常来自创始人主导的销售，从一篇时机得当的 Product Hunt 帖子到与早期客户的个人关系。但这种有机增长只能走到某个点，大多数创业公司在 Scale 阶段就撞到这个上限。征兆包括：用户曲线变平、客户获取成本上升、以及一个**只有当创始人亲自介入时才会动的 pipeline**。

Scale 阶段的增长需要构建一个**专门的增长引擎**，去触达你产品的新的、更广泛的受众。但大多数创业创始人此前可能从未需要运营过 marketing、sales、和分析师关系等程序。一个正经的 GTM 动作不只需要建立新系统和流程，还需要打造一个**品牌声音和故事**，去讲述你想如何谈论你的产品。因为，到创业生命周期的这一阶段，你需要这样一个东西，去触达的不仅是个体新用户，还有整组目标受众，比如投资人和企业买家。

所幸，GTM 职能不需要规模化才有效；而且**构建产品的同一套 AI 基础设施可以把它推上市场**。

Claude 如何帮助 Scale 阶段的创始人

早期创业阶段把 Claude 当作产品本身的基础基础设施：验证想法的研究伙伴、设计与构建原型的工程团队、让单创始人创业成为可能的 AI 运营层。到达 Scale 阶段的 AI 原生创业创始人，现在可以使用 Claude、Claude Code 和 Claude Cowork **以他们一路构建的方式来继续扩张**。

把日常任务交给 Claude Cowork

带着一种清醒视角开始 Scale 阶段：你最需要把时间和注意力投在哪里？这对首次创业者——一群从未真正构建过一个企业的人——可能是个挑战。Claude 能通过构建一份**只有你应该在这阶段做的事**的清单来帮忙，可能包括产品叙事决策、董事会关系、企业级 deal、以及创始人对创始人的对话。**不在这份清单上的，都是被委派或被 Claude Cowork 自动化的候选**。

- **练习**：用 Claude 产出你当前运营层的**瓶颈地图**：当前所有路由经过你的工作流、决策和审批。然后让 Claude 推断当你一周不在时每一项会发生什么。那些会停滞的工作流，就是你仍然 hands-on 到能"卡住进度"的地方。

这些与你和 Claude 一起做的创始人优先级与责任清单，如何映射？

接下来，是时候压力测试你**已经构建好的系统是否真的能随你业务的增长而扩展**。

- **练习**：用 Claude 映射你当前的工作流，然后问它当你一周不在时每一个会怎么样。会停滞的工作流，就是 hand-off 标准、升级路径、异常处理仍需收紧的地方。Claude 能分析失败点并推荐合适的修复，让你按需更新或替换 Claude Cowork 的自动化。

把技术运营扩展为企业级基础设施

随着你扩张，买家需要确认**你的产品和组织作为长期基础设施是可信赖的**。代码库内部的技术工作一如既往地继续，但现在还有围绕代码库的技术工作要处理。

第一步是把机构知识转化为可扩展的系统。用 Claude 起草并维护企业采购期望看到的书面基础设施，包括产品文档、支持 playbook、SLA。

并行地，让 Claude Code 对照企业合同要求的具体可靠性与安全标准来审计并加固代码库，并构建那些 Discord 社区式支持从未需要提供的**技术支持基础设施**：日志、监控、事件响应工具，以及让 SLA 真正可执行的可观测层。

然后 Claude Cowork 运行企业支持本身的运营层：工单路由、升级流程、由产品变更触发的文档更新、续约跟踪、以及企业 customer success 所依赖的报告节奏。把这三者放在一起，让一个小团队拥有"远大于自身组织规模"的支持姿态——这正是签订多年企业合同所要求你证明的。

- **练习**：挑出你最严苛的三家潜在客户，或者识别出三家你希望签下的理想客户。让 Claude 产出一份**差距分析**：这三家中每一家的企业采购团队在签订多年合同前会期望看到什么样的文档、SLA、和支持基础设施？你目前在哪里有不足？用这份输出在 Claude Code 与 Claude Cowork 之间安排技术与文档工作的顺序。

构建一个真正的 GTM 职能

创始人的拼劲把你带到这里，但创业的扩张需要创建并实施一个**真正的 go-to-market 策略**。AI 能帮你构建并随后运行这个完整的 GTM 引擎。

Claude 能从零协助构建基础性 GTM 资源：市场细分、信息架构、分析师关系策略、销售 playbook，以及一旦你开始与公开市场投资人、企业买家、华尔街分析师对话时所需的"对投资人讲的指标叙事"。这些受众每一个都有自己的词汇，并以自己的标准评估你；**Claude 的工作是把你产品的价值主张翻译成对各个受众细分都贴切的产品营销路径**。

接下来，Claude Cowork 能成为你的战术执行层：内容流水线、出站序列、分析师 briefing 物流、新闻室和 PR 节奏、CRM 卫生、pipeline 报告，以及把 GTM 策略变成真实商业动作的诸多循环周期。

当 GTM 动作需要产品营销基础设施时——交互式 demo 环境、集成文档、沙箱租户、API 参考、技术 one-pager——Claude Code 能为你构建。买家期望从技术层面评估你的产品；在 Scale 阶段，一段 Loom 视频和一份销售 deck 已经不够了。**这也是让你的 GTM 动作能异步运转的基础设施**：你在开董事会时，一个构建得当的 demo 环境也在帮你 close deal。

把领域专长与机构知识变成 AI 上下文

许多超精简创业的创始人在为一个亲身经历或在某个垂直行业一手观察到的现实问题构建**高度具体的应用或工具**。智能体 AI 现在让那些从未写过一行代码的创始人也能利用他们的领域专长，去构建解决复杂问题的产品。Claude、Claude Code、Claude Cowork 各自在**把创始人知识转化为可复利的产品具体性**这件事上扮演角色。

用 Claude 来捕捉、整理、提炼创始人知识，把领域专长放到产品能触达的地方。通过延伸性的对话、项目、和记忆，创始人可以把他们所知的一切——行业行话、监管陷阱、边界情况、痛点、为什么这个问题的显而易见的答案行不通——分享进一个**结构化、可搜索的 context**。然后 Skills 可以把重复性的工作流（例如"我如何审计一份商业租约"、"我如何分诊一份病人接诊表"）编纂成可被

Claude 每次以同样方式跑的可重用例程。几个月之后，这就成为一种**没有任何通用 AI 能匹敌的专有知识基底**。

用 Claude **把你的领域知识外化**，在把行业特定的边界情况编码进产品方面尤为宝贵：例如一个通用的 AI 医疗账单工具会在 340B 药品计划 claim 上崩溃，但你的工具针对它们有专门逻辑。Claude Code 帮你把你领域内其他从业者经历的常见痛点翻译成校验逻辑、prompt 改进、或与你竞品都没听说过的小众行业系统的 MCP 集成。结果是：你的应用工具的**深度与广度持续复利**，方式是竞品根本无法复制的。

- **练习**：识别一个通用竞品在你的垂直里"肯定会搞错"的边界情况。和 Claude Code 一起为它构建一个专门的 test case（不是单元测试），基于一个你真正见过的场景。每当类似的边界情况浮现，就加进去。**你的测试套件成为你护城河的地图**。

把累积的用户数据复利成可防御的优势

当用户与你的产品互动，他们生成行为信号（即他们接受哪些输出、拒绝哪些），这些信号告知产品 roadmap。久而久之，你会学到你这个特定用户群体的具体模式、偏好、边界情况。这就是我们说的**复利价值**：每次改进让产品更有用，进而带来更多使用，从而创造更多反馈，再驱动更多改进。

这种数据是**时间锁定的、与上下文相关的、抄袭者无法复刻**的：你根本买不到那些已经在你产品里反复打磨自己工作流的成千上万用户的**行为指纹**。

Claude 能帮你审计你已采集的任何用户交互数据，识别其中最高信号的行为模式，并设计**把持续使用转化为系统化模型改进**的反馈回路。

- **练习**：把你产品交互数据的概要喂给 Claude：你在采集什么、采集了多久、你对用户随时间与产品的互动方式了解什么。让它识别这些数据中三个最高信号的行为模式，并为每一个设计把它变成系统化模型改进的反馈回路。然后让它帮你起草一份给产品营销用的**一页护城河叙事**：你的数据飞轮怎么运转、它已经转了多久、为什么一个今天才起步的、资源充足的竞品在两年内都无法复刻。

创建工作流锁定

数据网络效应使你的产品更难复制；而**用户工作流锁定让你的产品更难离开**。用户在他们的日常运营中跑你的产品越久，它就越深地嵌进他们**实际工作的方式**里。他们在它之上建了自动化、训练了人去使用它、把它接到他们的数据源和其他工具上。他们开发的 prompt、他们改进的工作流、他们标准化的输出，全都围绕你产品做什么、以及怎么做来塑造。到这个点，切换从一个产品决策变成一个**全方位的运营项目**。

创建工作流锁定的第一步是请 Claude 按集成深度映射你当前的客户基础。对每个客户细分，识别他们在你产品之上构建了哪些工作流、依赖哪些集成。这显示了你的产品在哪里粘着，以及它需要往哪里更深扎根。

你提供的集成越多，客户构造**依赖于你产品的工作流**所拥有的表面积也越大。Claude Code 帮你快速搭建与你目标用户依赖的数据　　pipeline、项目管理工具、其他系统之间的原生集成。Claude Code 也能构建 API、webhook 和 SDK，让客户不仅使用你的产品，更**在它之上构建**——这是最深的锁定形式。

- **练习**：让 Claude 帮你为你的 top 10 客户做一次**工作流集成审计**。对每一位，记录他们构建的自动化、他们依赖的集成、跑过你产品的团队工作流、以及你对他们切换成本的估算。然后让 Claude 识别这群人中的共同模式：什么类型的集成对你这个具体产品创造了最深的锁定？你可以构建或开放什么，去为目前仍在表层的客户加深集成？

第 7 章

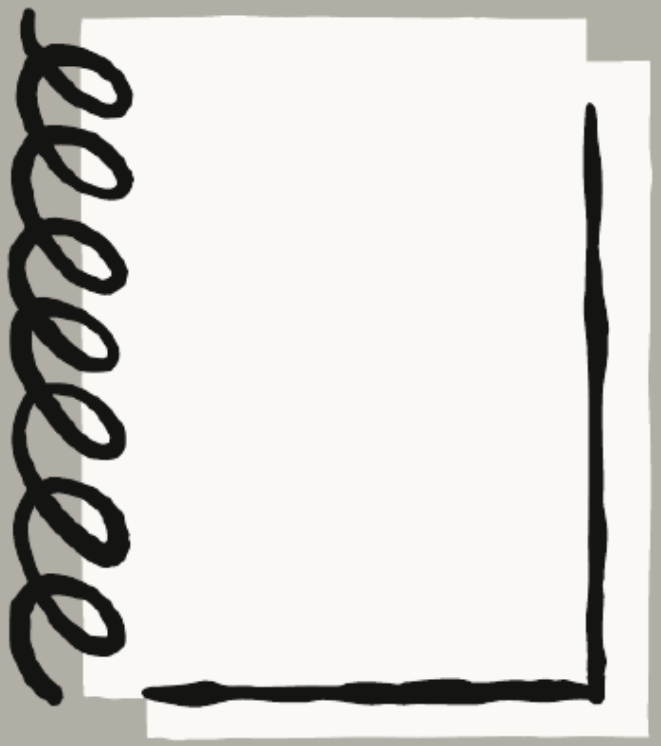
工作不变，规则变了

第七章 | 工作没变，规则全新

在 AI 时代，创始人的工作没有变：找一个真实问题，构建解决它的东西，把它扩展成一家有意义的公司。**变的是到达那里的路径**。横跨四个阶段——Idea、MVP、Launch、Scale——AI 把季度压缩成周。

过去要用几个月的验证周期，现在只要一个下午。一个能跑的原型不再需要一位有合适技术栈的联合创始人；它只需要一个清晰的问题，加上少数几次与编码 agent 的聚焦会话。Launch 就绪从"发布前的兵荒马乱"变成了一个**持续的工作流**。而在 scale 阶段，那种过去逼着早期招聘掉进救火角色里的运营负担，越来越能被交给 AI，把你团队的注意力**释放出来**去做那些会成为你护城河的判断性决策。

瓶颈不再是你能造什么，而是你选择造什么。



参考资源

资源

用 Claude 来构建

- **Building AI Agents for Startups**：分享创业公司如何使用 agents，让公司随规模化变得不那么"创始人依赖"。
- **Claude Code 文档**：从初始安装一直带你走到高级智能体工作流。建议从 "How Claude Code works" 概览入手。
- **Claude Code 最佳实践**：覆盖在 Anthropic 内部以及各工程团队中跑得通的模式——context 管理、权限、规划、验证流程。
- **使用 CLAUDE.md 文件**：手把手介绍如何为你具体的代码库配置 Claude Code。对要搭建开发环境的 MVP 阶段创始人是必读。
- **Claude Code 高级用户技巧**：精选来自 Claude Code 团队自身的工作流模式，包括并行会话和验证回路。
- **Get started with Claude Cowork**：分享团队如何搭建 Claude Cowork 并开始实施 skill、插件、和其他在你创业里放大其影响力的特性。
- **教程**：提供一份针对具体任务的可搜索动手 walkthrough 列表。

创始人故事

- **How three YC startups built their companies with Claude Code**：看 HumanLayer (F24)、Ambral (W25)、Vulcan Technologies (S25) 如何使用 Claude 把原型快速推向市场，并通过智能体编码工作流扩展 AI 驱动的平台。
- **GCAI** 的创始人用领域专长构建了一个响应式、由 Claude 驱动的法务平台，贴近内部团队真实工作方式：公司特定的 playbook、跨职能利益相关方、以及可变的风险容忍阈值。
- **Carta Healthcare** 使用 Claude 为其临床抽取平台提供支持，每年处理 22,000 例外科病例，将数据抽取时间缩短了 66%。
- **Anything**，由 Claude 与 Agent SDK 提供动力，已经帮助 150 万用户在不写代码的情况下把想法变成可工作的软件产品，包括一位非技术创始人，他构建并正在销售一整套招聘平台。Anything 的 AI agent 处理完整构建，让 solopreneur 把精力集中在他们的领域专长上。
- **Cogent** 是一家应用 AI 实验室，正在构建自动化关键企业安全任务的 agents。这家创业公司把 Claude 作为推理层，用于自动化跨整个漏洞生命周期的调查、优先级排序与修复。
- **Airtree** 使用 Claude Cowork 作为其运营基础设施的中心，把过去分散在十几种不同工具和团队中的数据统一起来。现在，当一个人用 skill 构建一个工作流自动化，整个组织里的每个人都能用它来做那些一直没完成的 to-do。

- **Duvo** 构建跨 ERP、供应商门户、电子表格、邮件甚至电话来跑采购、供应链和品类管理流程的 AI agents。Duvo 完全基于 Claude，使用 Agent SDK 来跨工作流编排。
- **Zingage** 是为家庭护理机构构建的 24/7 自动化运营的 AI agent 平台。该创业公司使用 Claude 的结构化工具调用来跨 EMR 和多个通讯通道编排，并使用 Claude 的上下文推理来构建可以给出**对患者细化定制**结果的 agents，而不是去匹配最常见的回应。
- **Kindora** 是一个 AI 驱动的平台，由一位非营利组织高管使用 Claude Sonnet 构建——一个非常急需的工具，用于把慈善机构和资助方做智能匹配。在把数千匹配筛到值得追的那几个之后，Kindora 的 MCP 连接器让非营利机构能直接在 Claude 里访问其潜在客户工具。
- **Wordsmith** 由一位"律师转 CTO"创立，为内部法务团队提供可靠的 AI 驱动法律技术。Claude 是 Wordsmith 在合同审查、协议起草和文档审查能力上的推理引擎，该创业公司的工程团队也使用 Claude Code 来构建并演化平台本身。

创业支持与机会

- **Anthropic Startups Program**：对于与 Anthropic 的 VC 伙伴合作的创业公司，本项目提供免费 API 额度、公开可用的最高一档 rate limit、以及创始人专属活动和工作坊的邀请。
- **Claude 社区**：构建者的论坛与社区空间。
- **直播学习资源**：会议、网络研讨会、直播和录像。

> 🌐 `claude.ai`