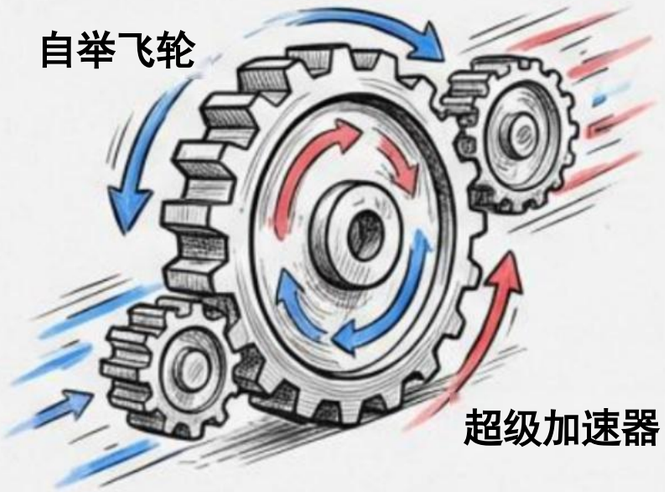
Anthropic 为什么成为迭代最快的AI团队



把 AI当作研发超级加速器的自举飞轮

- 核心判断是：它把自己造的AI，尤其是Claude Code，当成了内部研发引擎，而不是仅仅当成对外售卖的产品。
- 因 此Anthropic的速度不是单点快，而是模型、工具、组织、评测与治理一起快。

@清新研究|2026年3月

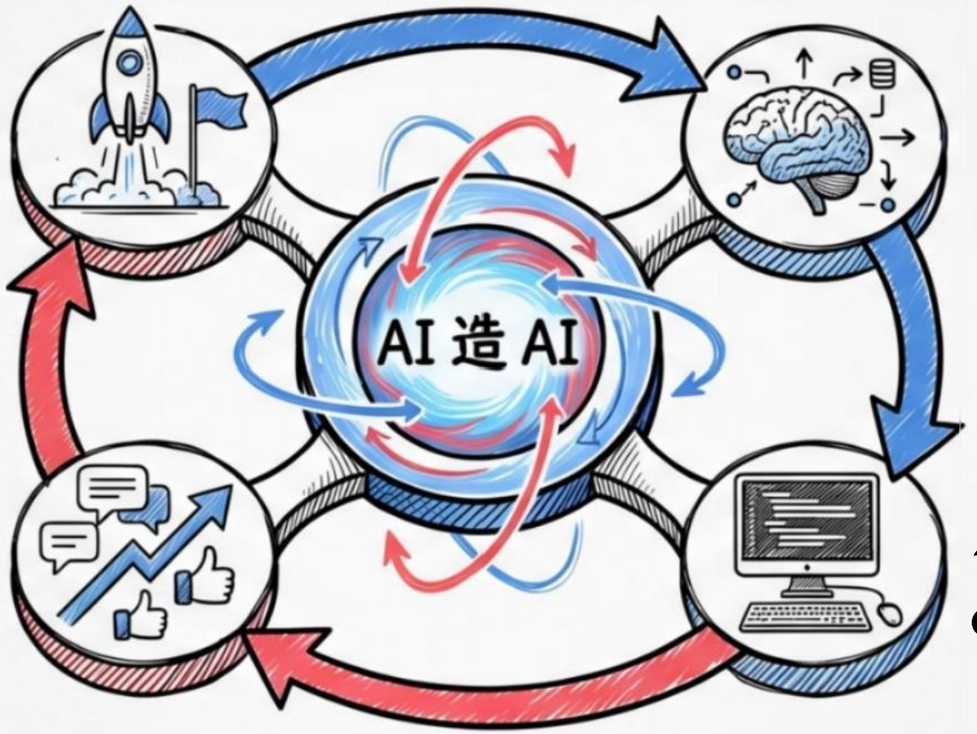
核心答案

发布

复利速度

模型

自举飞轮



用户反馈

内部高频 Dogfooding

代码

Claude Code 生产引擎

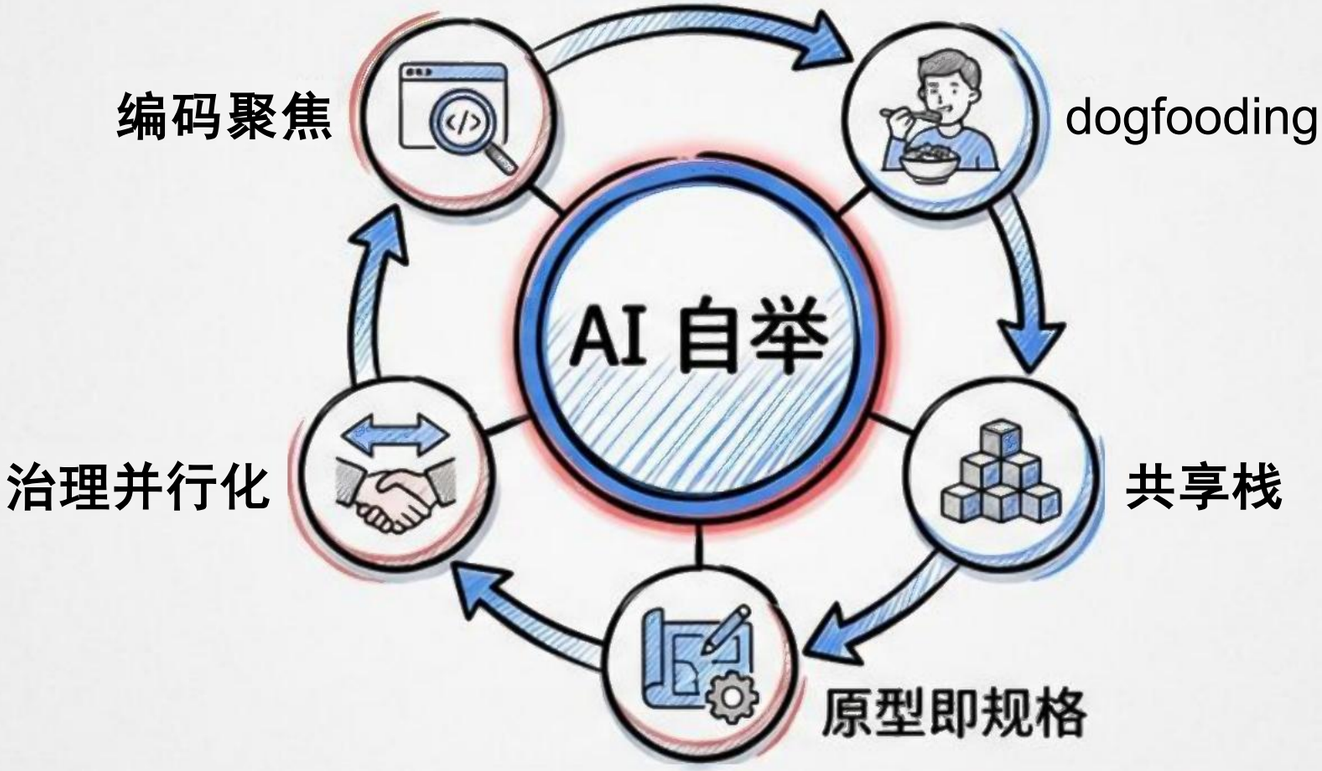
Anthropic 之所以快，核心不是“人类团队更拼”，而是“AI 正在帮助它更快地造AI”。

- Claude Code 在Anthropic 内部不仅是编码助手，更是把想法转成可运行原型、把原型转成可发布功能的生产引擎。
- 一旦内部高频dogfooding 与外部产品化共用同一套模型、agent loop 与上下文管理，速度就会形成复利。
- 本报告因此把“自举飞轮”视为一号原因，把组织、文化、治理、聚焦战略视为飞轮得以持续运转的放大器。

@清新研究|2026年3月

一个核心原因，五个放大器

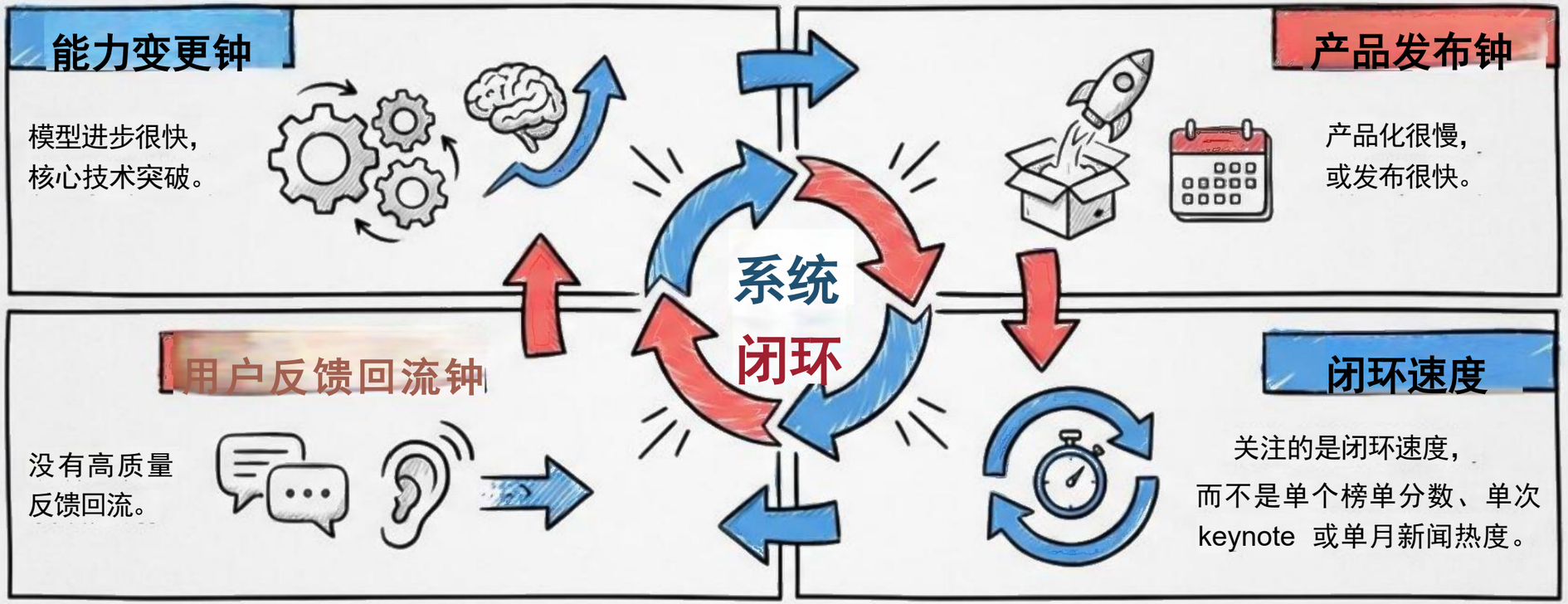
一个 原因是AI自举；五个放大器是dogfooding、共享栈、原型即规格、治理并行化、编码聚焦。



本报告如何定义“迭代最快”

不是谁发公告最多，而是谁最能把能力进步压缩成真实发布与真实使用。

我们把“迭代速度”拆成三只钟：能力变更钟、产品发布钟、用户反馈回流钟。



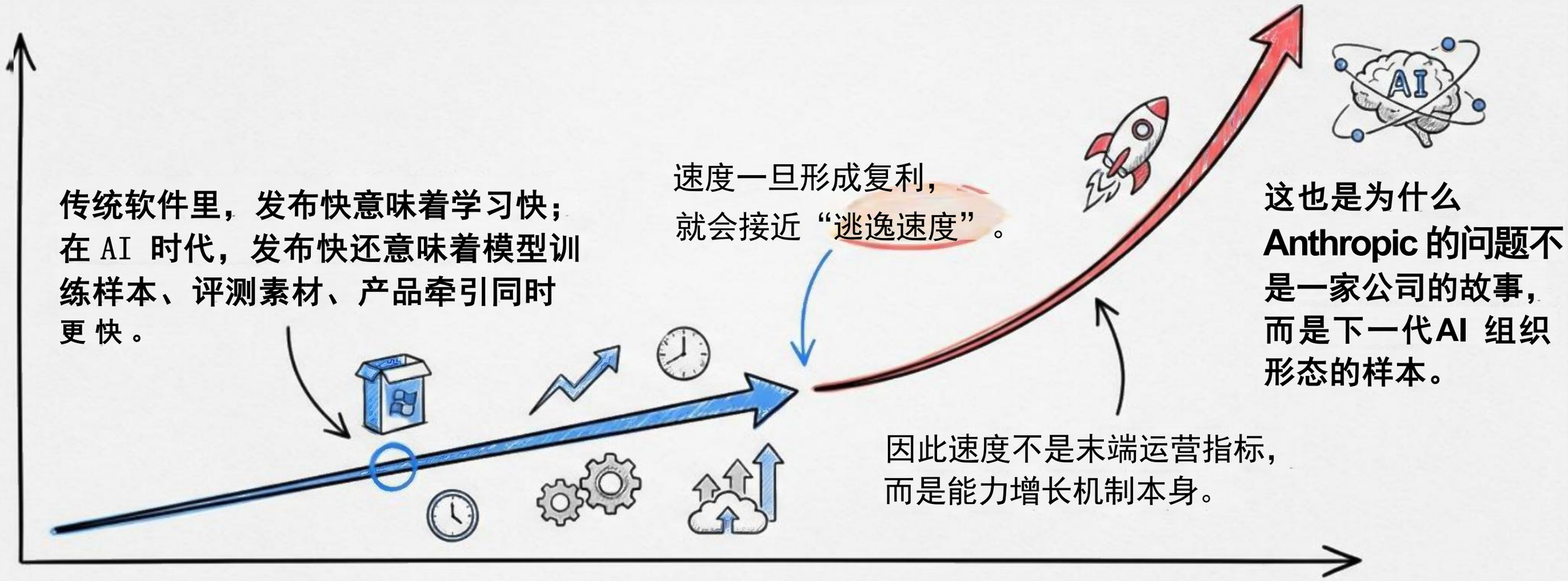
如果一个团队模型进步很快，但产品化很慢，或发布很快但没有高质量反馈回流，它都不算真正的“最快”。

因此本报告关注的是闭环速度，而不是单个榜单分数、单次keynote 或单月新闻热度。

@清新研究 | 2026年3月

在 AI 时代，速度会复利

谁更快把模型进步变成可验证工作流，谁就更容易拉开后续差距。



传统软件里，发布快意味着学习快；
在 AI 时代，发布快还意味着模型训练样本、评测素材、产品牵引同时更快。

速度一旦形成复利，
就会接近“逃逸速度”。

因此速度不是末端运营指标，
而是能力增长机制本身。

这也是为什么 Anthropic 的问题不是一家公司的故事，而是下一代AI 组织形态的样本。

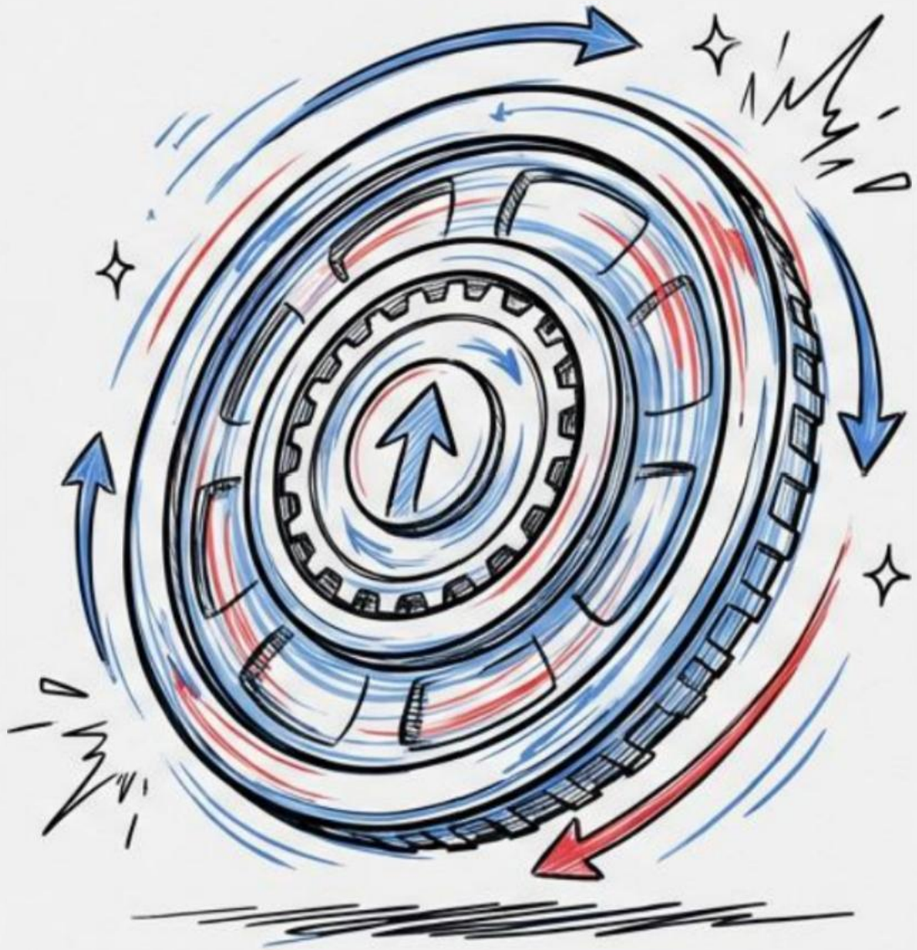
西

01

第一部分

公开证据链：为什么我们有理由说 Anthropic 处在迭代速度第一梯队

先看可公开核验的节奏证据，再进入为什么它能快的机制解释。



1

6.

Claude Code不是边缘项目，而是增长主轴

> 当内部加速器变成外部现金流，速度就从成本中心变 成利润中心。

99

- Anthropic 官 方 披 露，Claude Code于 2025 年 5 月对公众开放， 6 个 月达 到10亿 美 元run-rate revenue。

●到2026年2月，Anthropic又披露Claude Code的run-rate revenue已超过25亿美元，而且自2026年初以来又翻倍式增长。

Aha!

这意味着Claude Code 已经不是实验性工具，而是可以反过来为更快迭代提供资金、用 户和战略优先级的主业务。

2026年1月，Anthropic一个月发布超过30个产品与功能

高频发布不是感受，而是官方自己披露的公司级节奏。

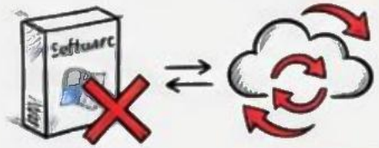
季度平均发布(旧节奏)

1月单月发布(新节奏)

1月密集发布刻度

Anthropic在2026年2月的融资公告中写明，仅2026年1月就发布了30多个产品与功能。

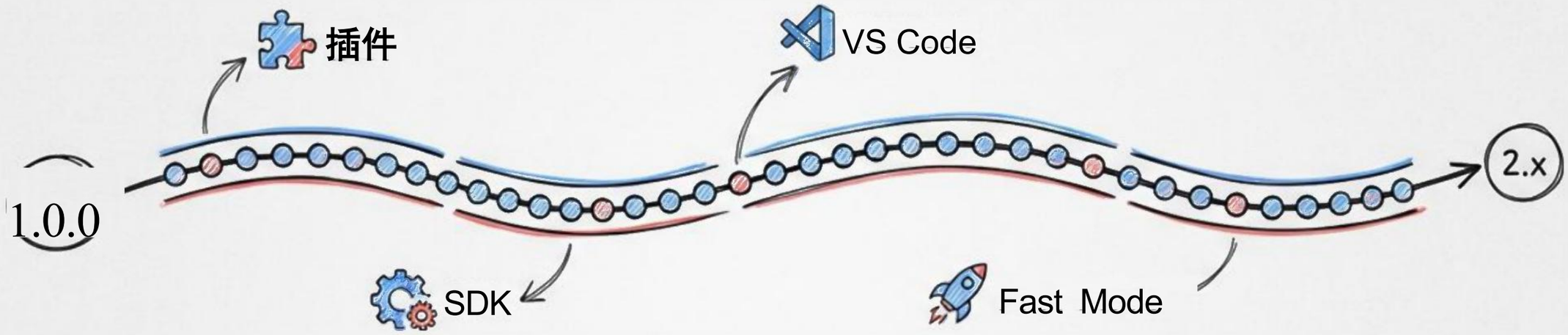
这说明它的外部节奏已经不是“季度级大版本”，而是更接近连续发布的操作系统。



当月超过30次的公开节奏，与内部研究里生产率和自主度的上升相互印证。

@清新研究 |2026年3月

Claude Code的更新模式是高频小步快跑



●从公开release notes看 ，Claude Code不是靠偶发大版本，而是靠连续的小步迭代。

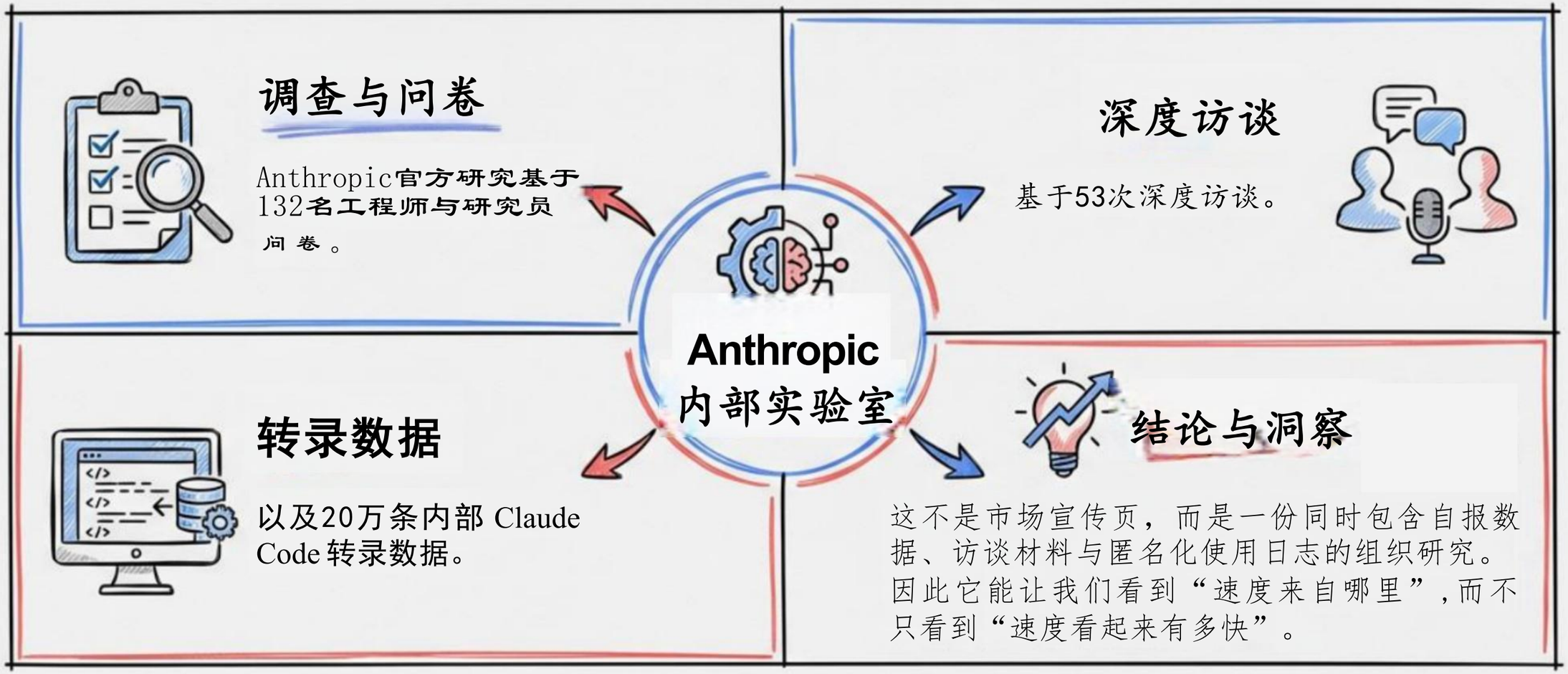
• Claude Code的发布说明展示出大量1.x、2.x的连续点版本更新，内容涵盖UI、插 件、MCP、权限、模型选择与性能优化。

• 这种节奏意味着团队在做非常密集的修正、扩展与试验，而不是等一个完美版本再统一上线。

●高频小步快跑的价值，在于把风险拆小，把反馈拉近，把产品和内部使用同步校正。

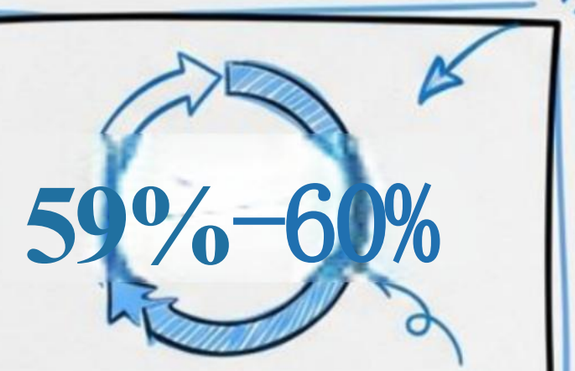
Anthropic用自己做了一次“内部劳动力实验”

这份研究之所以重要，是因为它直接观察了最早一批深度使用者如何工作。



内部工程 throughput 已经出现结构性跃升

速度飞轮首先表现在工程输出，而不是抽象愿景。



59%–60%

Anthropic 报告称，员工如今在约59%到60%的工作中使用 Claude

（工作占比）

+50%

平均自报生产率提升约50%

（生产率）

+67%

同一份研究还说，这大致与工程组织里每位工程师每天 merged pull requests提升67%的现象相互印证。

(merged PR)

如果这些增益能持续存在，团队的自然发布时间尺度就会从“周或月”向“天或周”下压。

@清新研究|2026年3月

AI 不只加速旧任务，还放大“本来做不了的任务”

真正改变组织速度的，往往不是节省几分钟，而是把原先不会排进计划表的工作做出来。

Anthropic研究显示，27%的 Claude 辅助工作属于“如果没有Claude 就不会做”的任务。

当一个组织能把边角需求、探索性修补和小型自动化都吞下去，它的累计进化速度会显著上升。

新增工作量

扩展项目

制作原本不值得人人工投入的小工具，以及更多探索性工作。

补齐工具

补齐边角需求和小型自动化工具，解决长尾问题。

探索性工作

进行更多前瞻性、试验性的探索和创新尝试。

质量修复

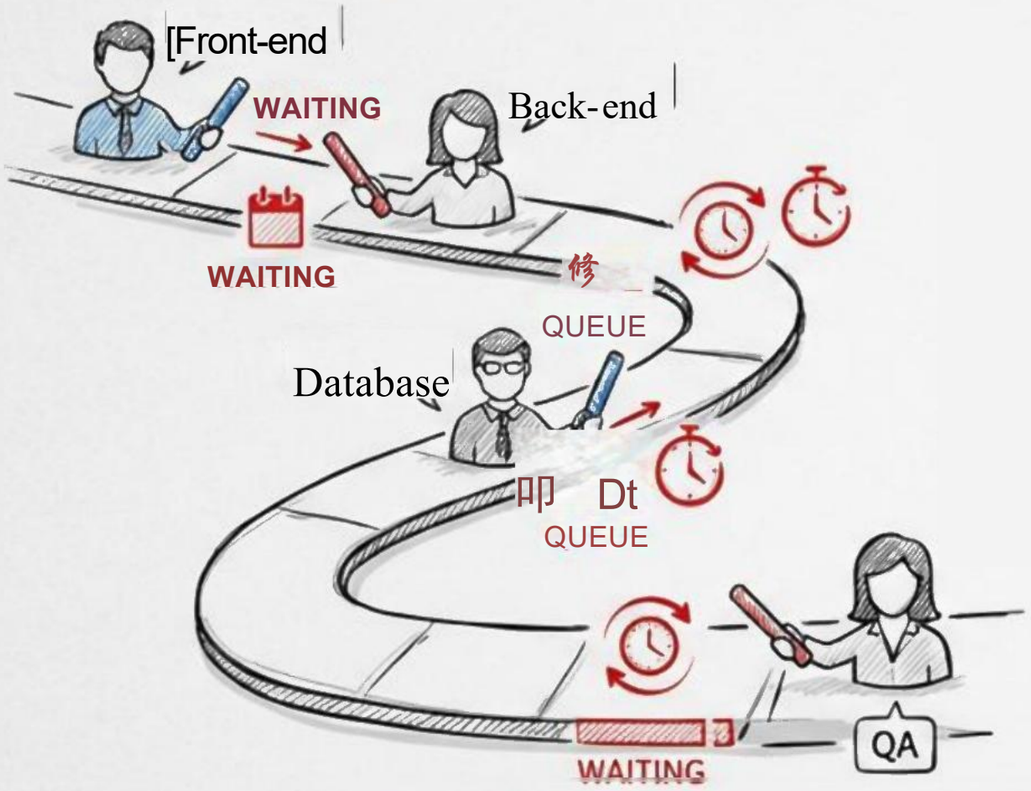
修复被搁置的非关键缺陷，提升整体系统稳定性。

这类任务包括扩展项目范围、制作原本不值得人工投入的小工具，以及更多探索性工作。

@清新研究|2026年3月

工程师的角色正在从“专才”转向“更广义的full-stack”

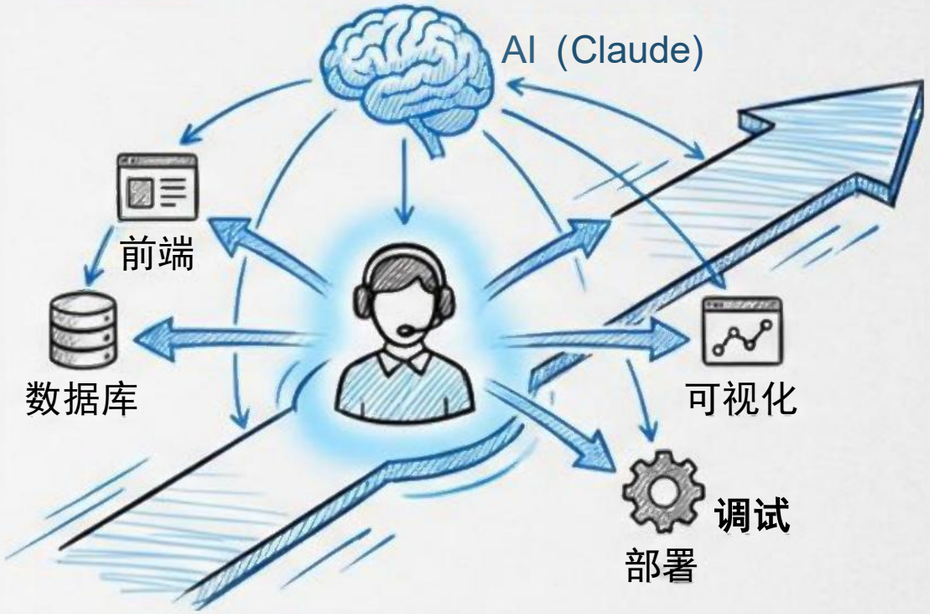
传统专才接力



◎ 这意味着一个人不再必须等另一个专门岗位排期，很多跨栈等待被Claude 压缩掉了。

个工等待被压缩后

AI辅助下单人full-stack 路径



◎ Anthropic 的速度，不只是同样的人干得更快，而是同样的人能覆盖更宽宽工作面。

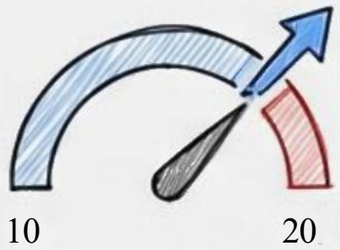
② Anthropic 访谈显示，不少工程师开始跨越前端、数据库、可视化、调试等原本不熟悉的领域完成任务。

◎ 速度因此不只是token 更快，而是组织排队时间显著减少。

Claude Code正在处理更复杂、更长链路的任务

迭代快的关键，不只是回答快，而是agent 能独立完成更多连续动作。

连续动作数
(Consecutive Actions)



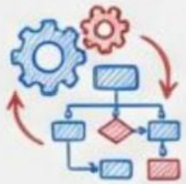
10→20 actions

Anthropic 研究显示，Claude Code 六个月前平均能连续完成约10个动作，如今大约能连续完成20个动作。

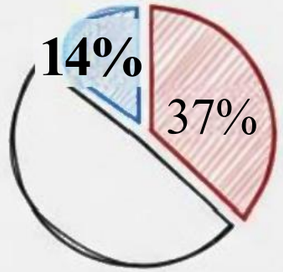
任务复杂度
(Task Complexity)

3.2→3.8 complexity

同一研究还显示，任务复杂度平均从3.2提升到3.8，复杂任务中“实现新功能”的占比从14%升到37%。

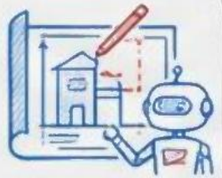


复杂任务中“实现新功能”占比
(New Feature Implementation %)



14%→37% feature work

当 agent 可以连续完成更长链路任务，组织才有机会把“原型就是规格”真正跑起来。

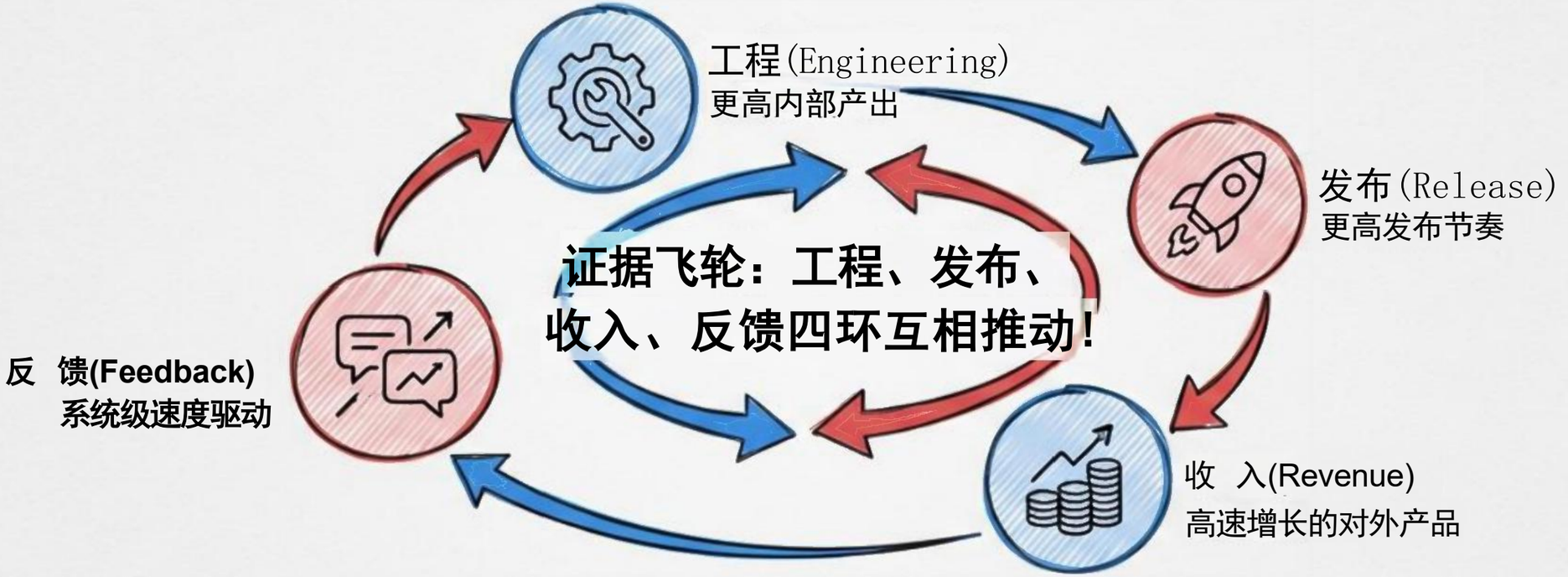


自主度提升(Increased Autonomy):Claude Code的自主能力显著增强，推动快速迭代和更复杂功能的实现。

@清新研究|2026年3月

公开证据已经足够说明：Anthropic 的速度不是营销幻觉

它的快，已经同时表现在工程 throughput、 发布频率、用户牵引与收入牵引上。

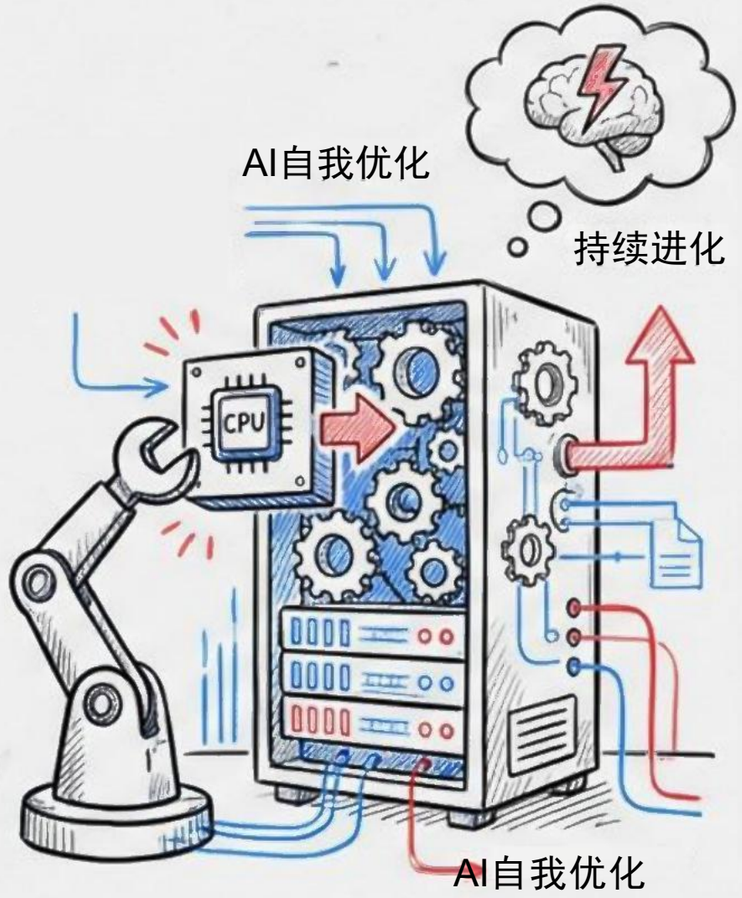


- 如果一个团队既有更高内部产出，又有更高发布节奏，还把一个内部工具做成了高速增长的对外产品，我们就有理由把它视为速度领先样本。
- √ 这并不意味着它在每个单项榜单都绝对第一，但足以支持 “Anthropic是当前最强速度样本之一” 的判断。
- 接下来真正要解释的，是它为什么能形成这种系统级速度。

02

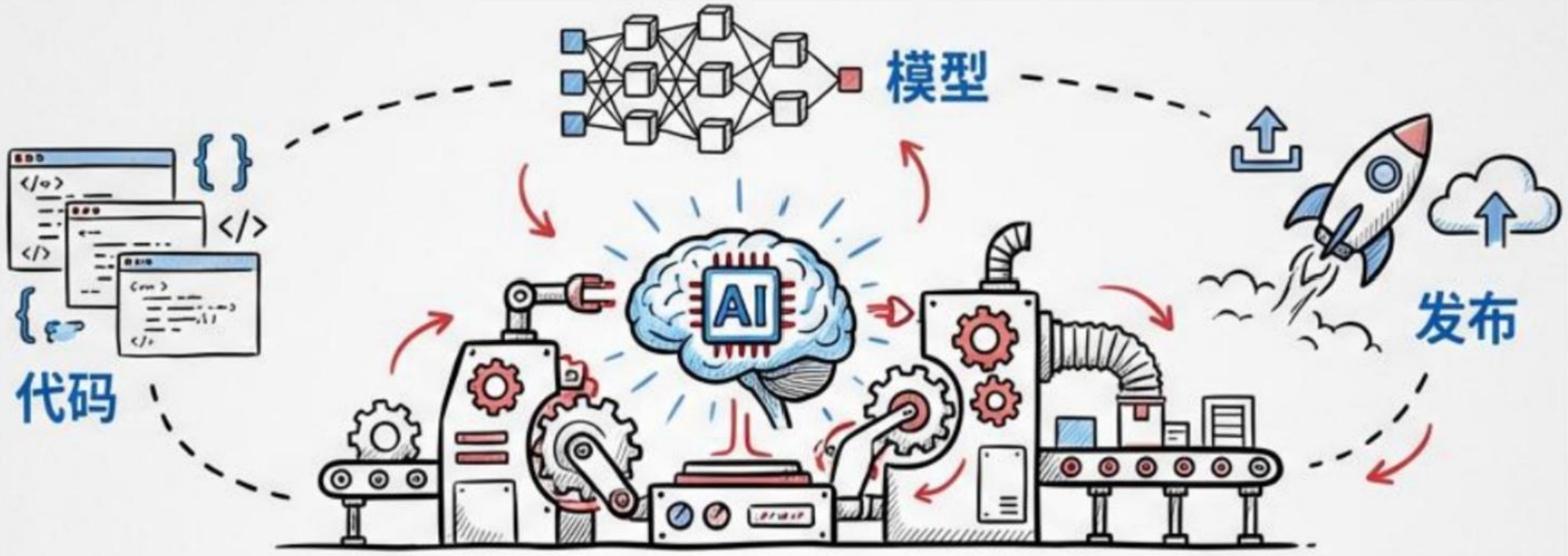
核心机制：为什么Anthropic会把速度优势越滚越大

从这里开始，我们不再只看“有多快”，而是分析“为什么能持续快”。



第二部分

原因：把 AI 自己变成“造AI 的机器”



“Claude Code是 Anthropic 的研发超级加速器，也是自举飞轮的起点。”

- 如果说传统研发像“人写代码、工具辅助”， Anthropic 的模式更接近“AI写大部分实现、人负责目标、验证与取舍”。
- 这不是抽象愿景，而是由公司级dogfooding、 内部研究和产品商业化共同支撑的组织现实。

所以本报告的核心答案非常直接：Anthropic快，是因为它已经把AI 变成了自己的研发操作系统。

Claude Code在Anthropic内部被高强度、跨职能使用

只有当工具在公司内部被当作默认工作面，飞轮才会真正出现。

数据基础设施

(Data Infrastructure)

- 维护数据管道，自动化任务。

产品开发

(Product Development)

- 快速选代，功能实现，代码调试。

安全工程

(Security Engineering)

- 威胁检测，代码安全审计。

company-wide

dogfooding

推理&数据科学

(Reasoning &Data Science)

- 模型分折，数据洞察，算法优化。

增长营销&产品设计

owth &Product Design)

- 内容生成，用户体验原型，市场策略。

法务&跨职能协作

(Legal &Cross-Functional Collab)

- 合同审查，政策合规，流程对齐。

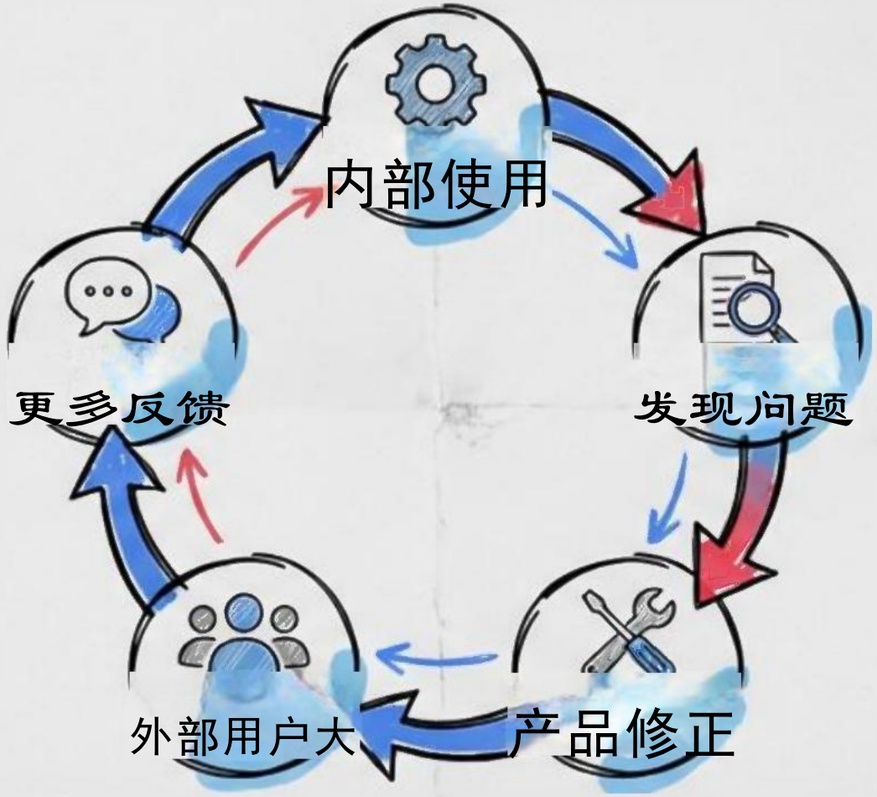
Anthropic 官方案例显示，数据基础设施、产品开发、安全工程、推、数据科学、增长营销、产品设计、法务等团队都**在使用Claude Code。**

这意味着Claude Code的内部反馈不是单一工程场景，而是覆盖从调试、理解代码、自动化到跨职能协作的真实任务集。

越多团队把它当默认工作面，它就越像组织底层runtime，而不是一款孤立插件。

Claude Code 既是产品，也是内部生产引擎

这类“双重身份”会极大压缩产品到组织的学习距离。

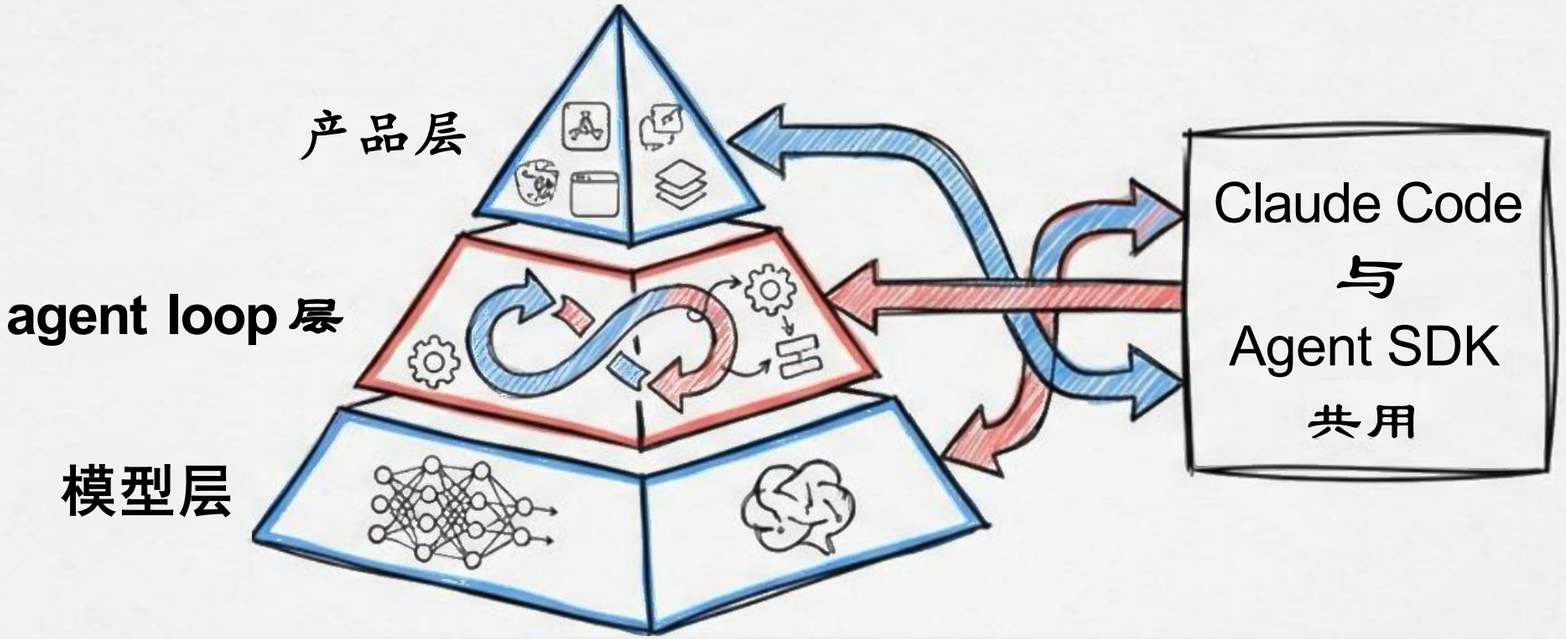


☑ 很多公司内部工具和对外产品是两套栈，内部收益无法快速转成市场验证；Anthropic 恰恰相反。

☑Claude Code的内部使用会直接暴露问题、提出需求、产生新工作流；这些发现又能快速沉淀到对外版本。

☑ 因此Anthropic的产品团队不是隔着用户猜需求，而是在用自己每天的工作流直接制造需求。

Agent SDK 让“内部工具”与“外部能力”共用一套底座



共享同一套agent loop，是 Anthropic 速度能够扩散的关键。

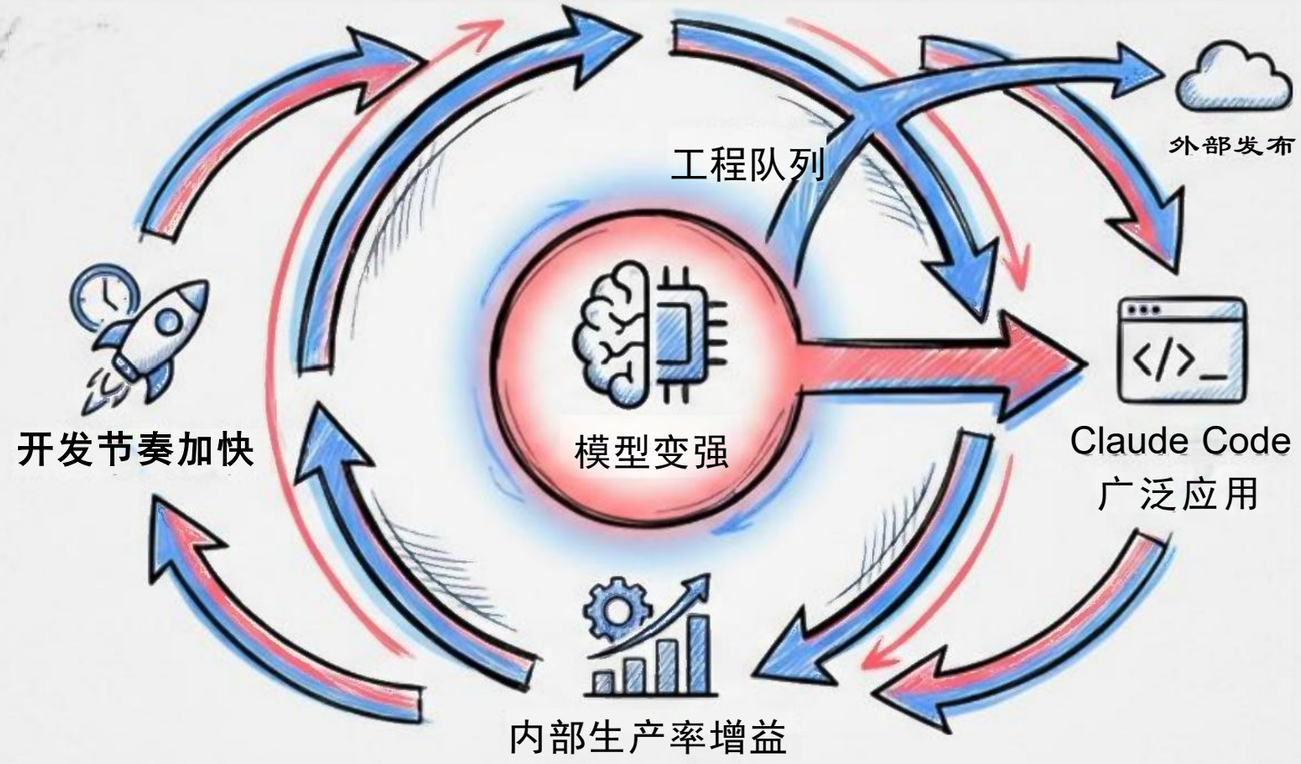
·Anthropic 文档明确写道，Agent SDK 提供的 tools、agent loop与 context management，正是驱动 Claude Code的同一套核心机制。

●这意味着内部已经证明有效的执行框架，可以非常快地变成可编程的外部能力，被客户和生态复用。

●当内部和外部共用底座，组织不会为“内部版”和“客户版”维护两套完全不同的系统。

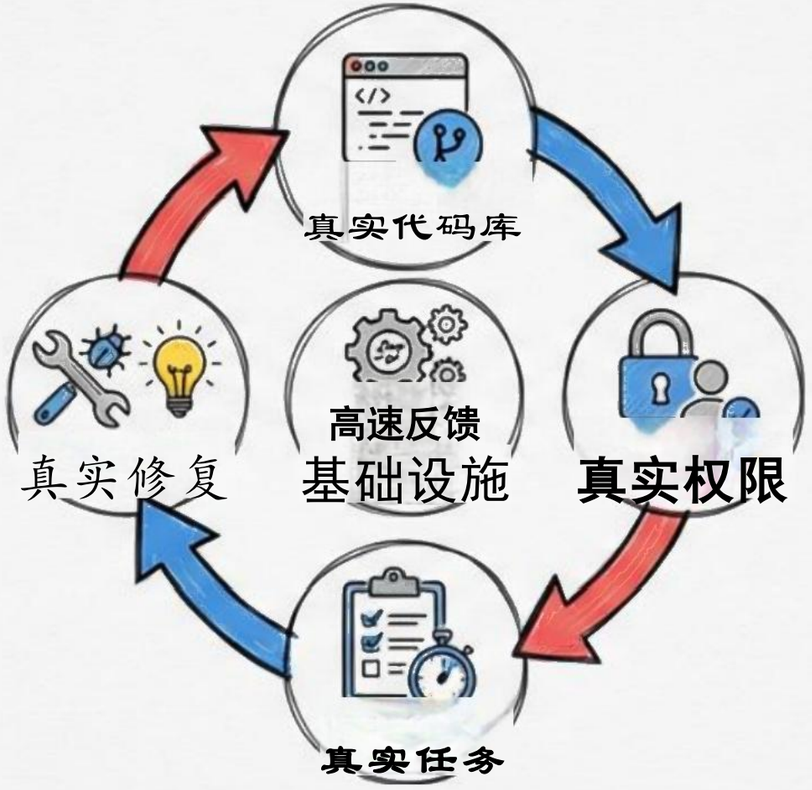
模型变强，不只提升用户体验，也直接提升Anthropic自身的开发速度

在 Anthropic这里，模型进步和工程速度是同向耦合的。



- 因为内部广泛使用Claude Code,任何模型在代码理解、调试、规划和长链路执行上的增强，都会立刻变成内部生产率增益。
- 这使Anthropic 的模型进步不需要经过漫长的“组织吸收期”,而是可以快速体现在开发节奏上。
- 当模型改进、工具改进与组织吸收同时发生，速度就会呈现出复利效应。

真实任务反馈



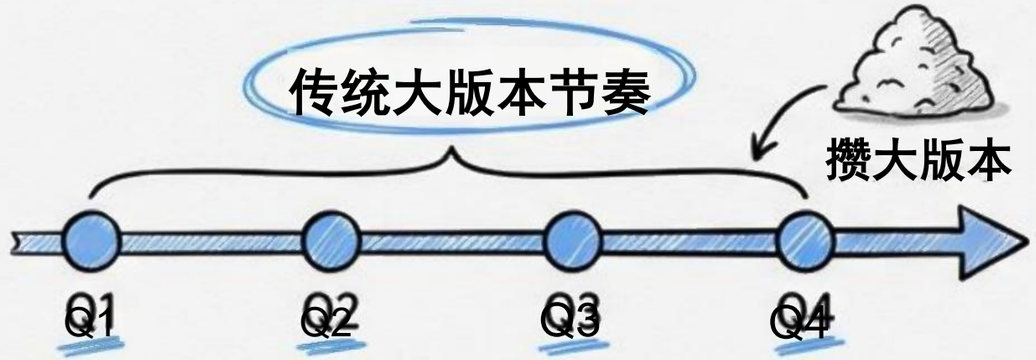
- 高频真实使用，让Anthropic 拿到的是“活数据”而不是“纸面反馈”
  这使它比只靠评测集和用户访谈的团队更快发现问题。
- 内部员工每天在真实代码库、真实权限、真实协作关系中使用Claude Code,这些反馈比实验室环境更接近生产。
- 当一个问题在内部被高频撞到，它通常会以更短路径进入产品修正或模型改进 。
- 所以Anthropic 的 dogfooding 不是文化口号，而是一种高速反馈基础设施。

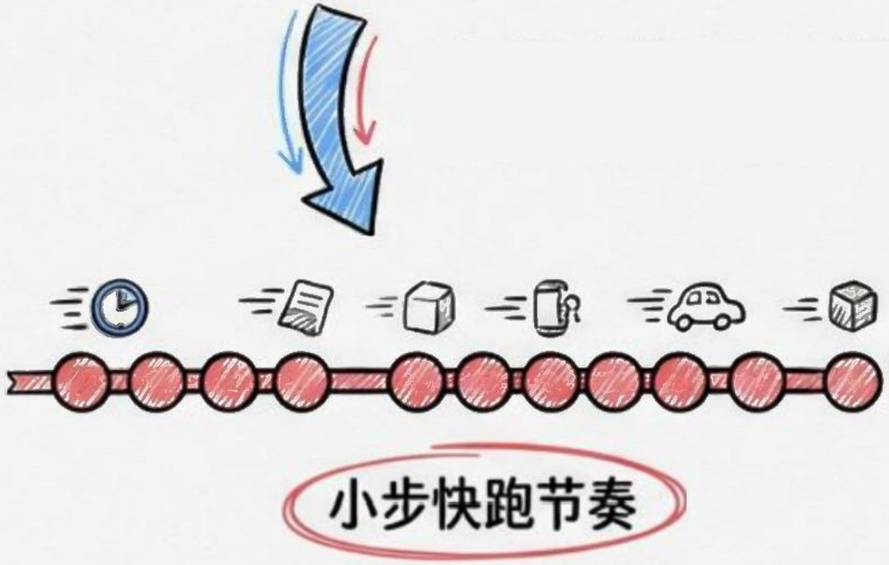
原型即规格

当 AI 可以在几小时内给出可运行版本，文档不再是唯一规格，
原型本身开始承担规格功能。

- ☑ 传统组织里，需求先变成长文档，再变成排期，再变成实现；Anthropic 的优势在于，这条链路被Claude Code强行压短。
- ☑ 因为原型产出成本骤降，讨论可以更早围绕可执行物展开，而不是围绕大量前置文字展开。
- ☑ 这并不意味着规格文档消失，而是意味着“原型先行、文档补全”的比例显著上升。

原型即规格，会把组织节奏改造成“小批量连续发布”





Claude Code的连续版本迭代与Anthropic 在2026年1月超过30个产品与功能发布，符合小批量连续发布的模式。

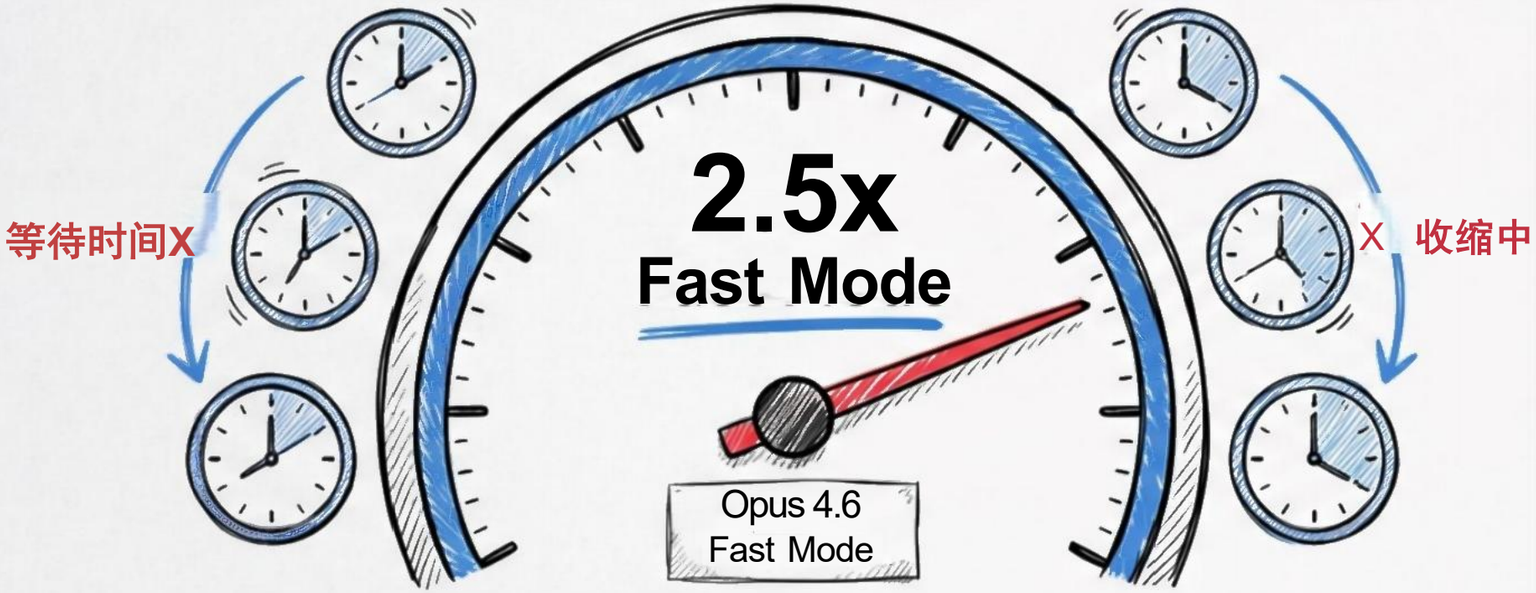
- 这种模式的好处是每次改动更小、解释成本更低、回滚更容易，组织更愿意加速。

发布本身因此从“重大仪式”转成“日常动作”。

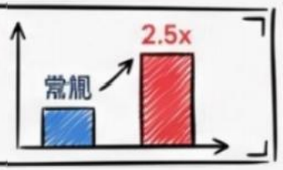
@清新研究|2026年3月

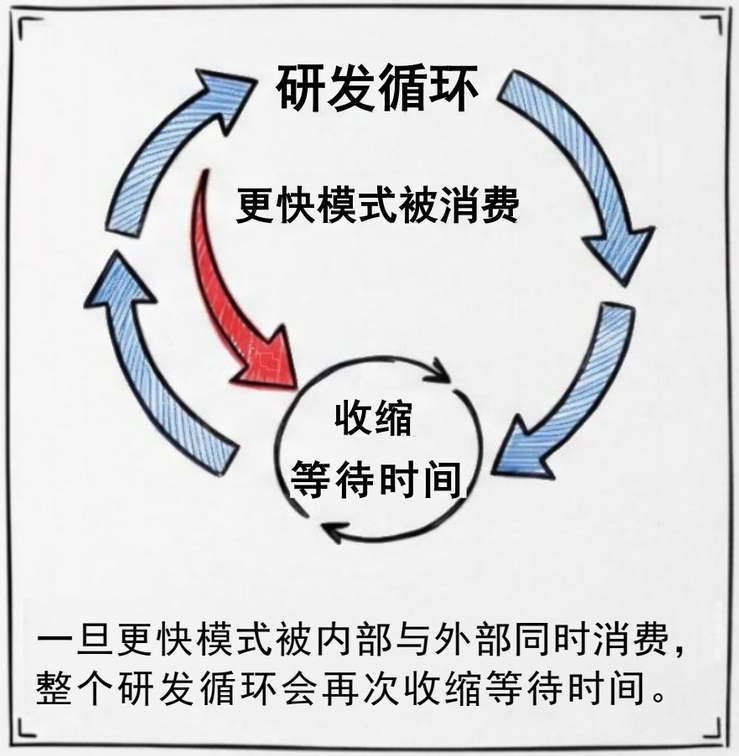
Anthropic 连“加速器的速度”都在继续加速

这说明它不仅在用AI开发，还在持续优化AI开发的runtime。



Anthropic 在研究中举例称，Opus 4.6的 Fast Mode 输出速度是常规模式的2.5倍。

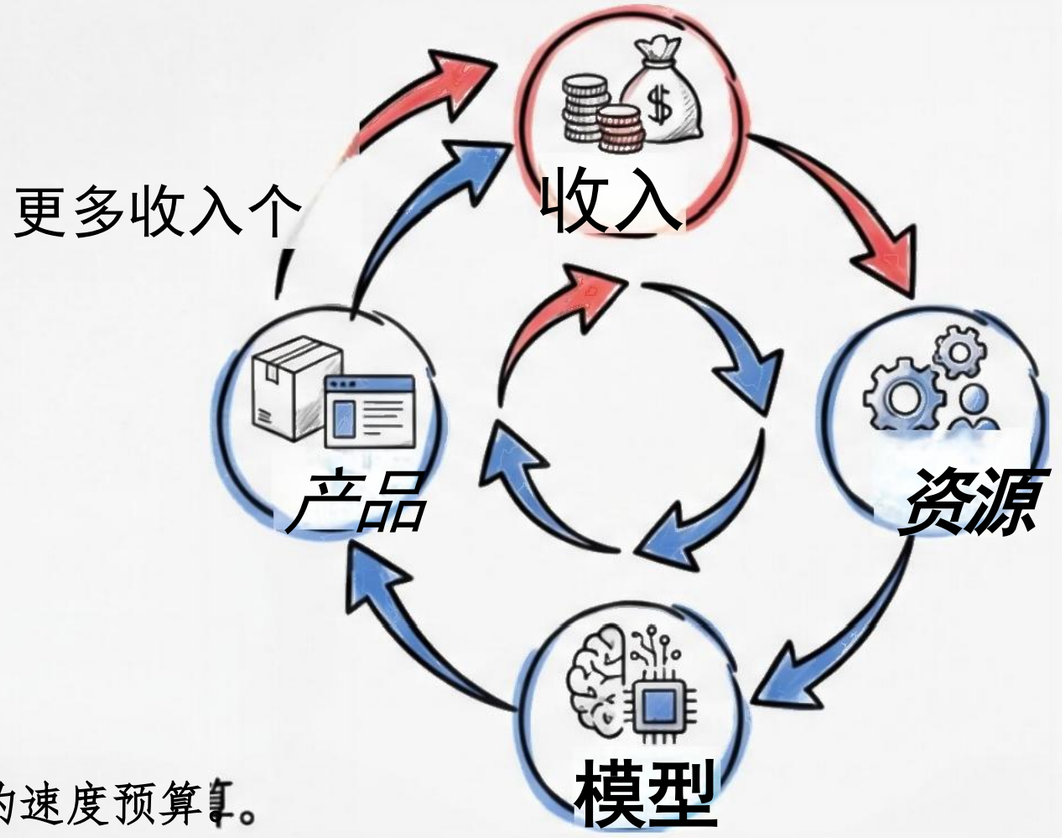




一旦更快模式被内部与外部同时消费，整个研发循环会再次收缩等待时间。

也就是说，Anthropic 正在优化的不只是模型能力，还有模型作为开发引擎时的吞吐表现。

当编码产品赚钱，飞轮会从“研发自救”变成“研发扩张”



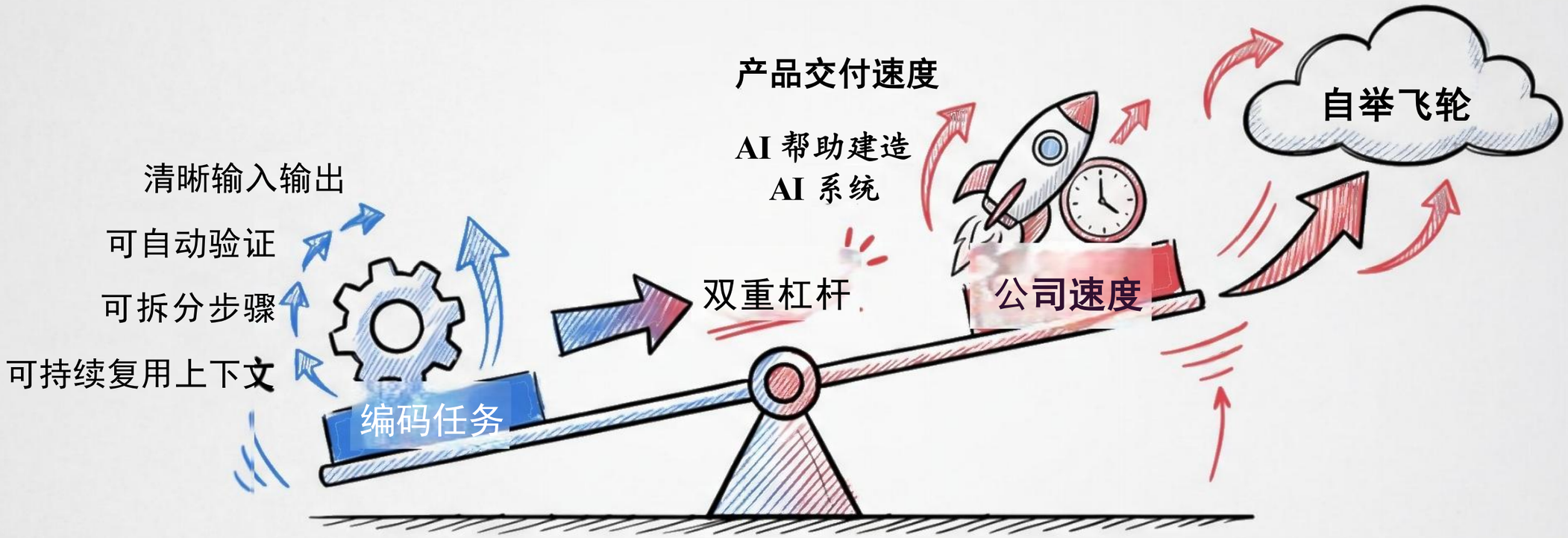
●收入反过来购买更大的速度预算。

- Claude Code达到十亿美元、再到二十五亿美元以上的run-rate revenue,意味着它已经能反向争取资源、人才与组织优先级。

●这种状态下，继续给Claude Code投入并不是“研究预算”,而是“增长预算”。

- 于是产品增长、模型改进与工程效率，会共同争夺更多内部权重，进一步强化速度优势。

Anthropic把火力集中在编码与agent，不是偶然，而是最高杠杆选择



因为编码是最容易形成自举飞轮的AI 场景。

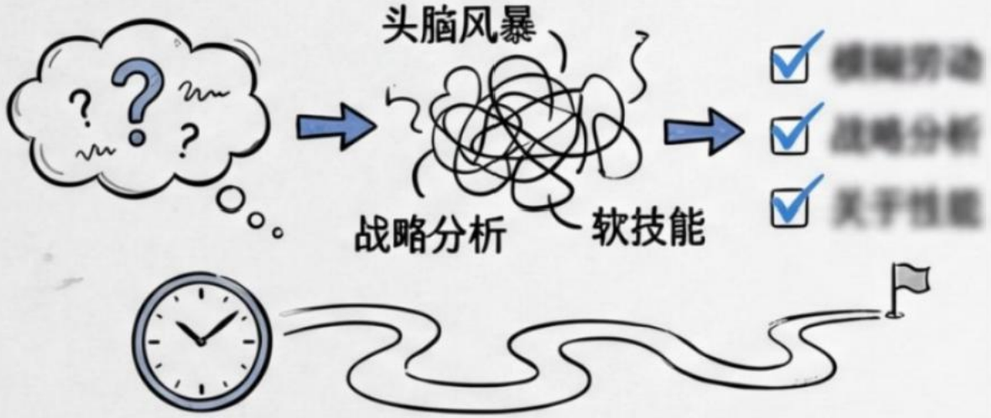
- 编码任务有清晰输入输出、可自动验证、可拆分步骤、可持续复用上下文，这些特征天然适合 agent 化 。
- 更重要的是，编码又直接决定AI 公司的产品交付速度，因此在Anthropic内部具有双重杠杆。
- 谁先把编码做成高效agent，谁就更容易把整家公司变成“AI帮助建造AI”的系统。

编码为什么适合AI

与多数白领任务相比，编码更适合形成高可信闭环

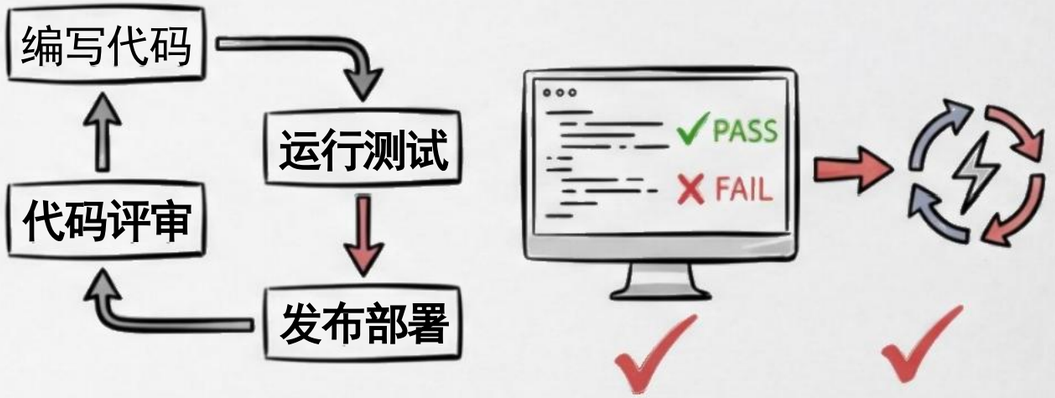
可验证性，是Anthropic 速度飞轮成立的重要前提。

模糊知识工作



难以验证的知识劳动，
常常需要长周期判断与含糊标准

可验证编码工作



编码任务，可以通过测试、运行结果、
diff 与代码评审快速反馈

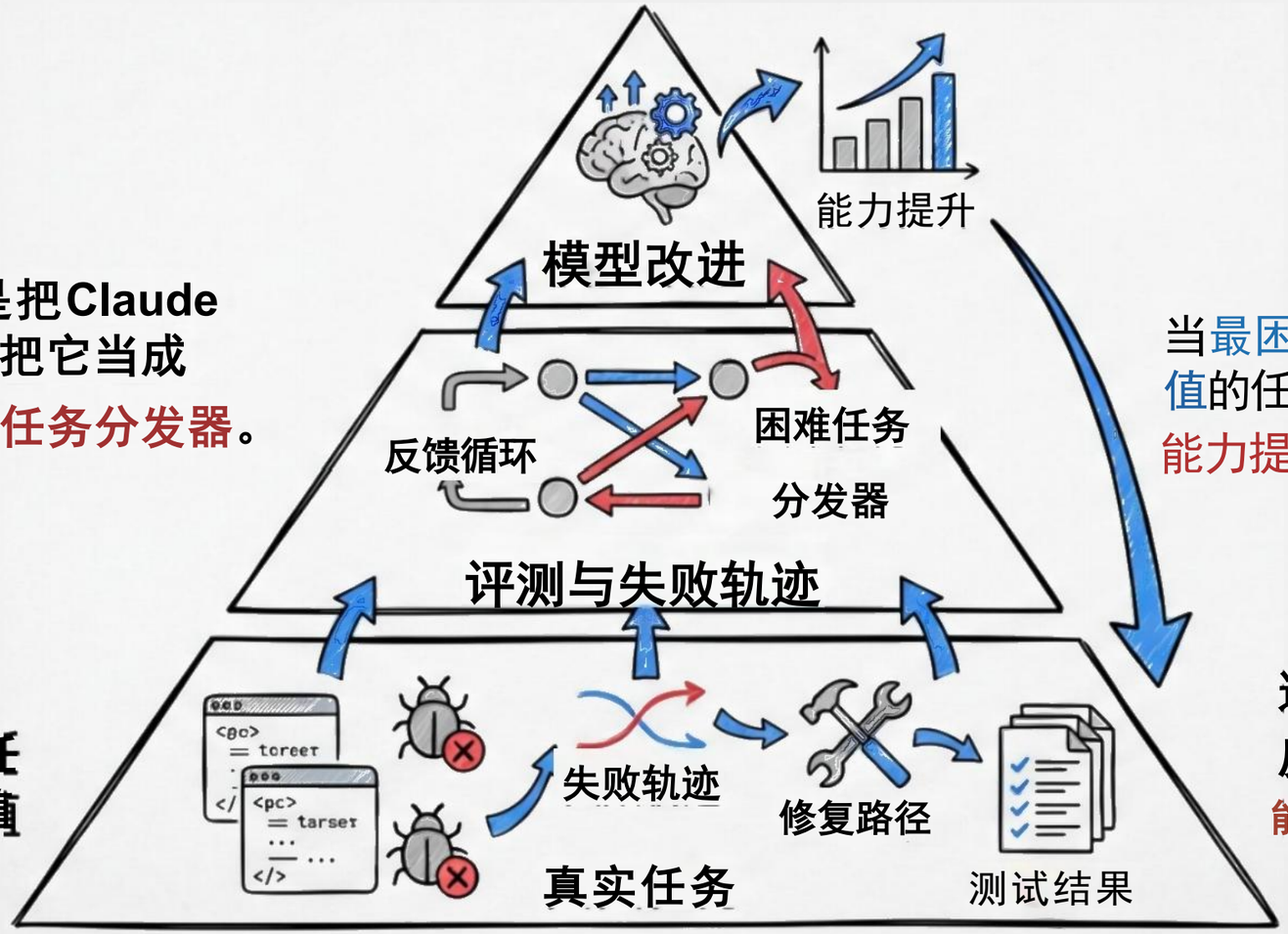
越容易验证，组织越敢把更多任务交给AI; 越敢交给AI, 闭环就越快。

这也是为什么Anthropic 先在编码上形成突破，再把同样的agent 基础设施扩展到其他知识工作。

@清新研究 | 2026年3月

编码如何反哺模型

编码不只是应用场景，还是模型改进的高质量训练与评测来源



因此Anthropic不只是把Claude Code 当营收产品，也把它当成最密成最稠密的困难任务分发器。

真实代码库里的bug、失败轨迹、修复路径、测试结果和长上下文任务，本身就是极高价值的模型改进素材。

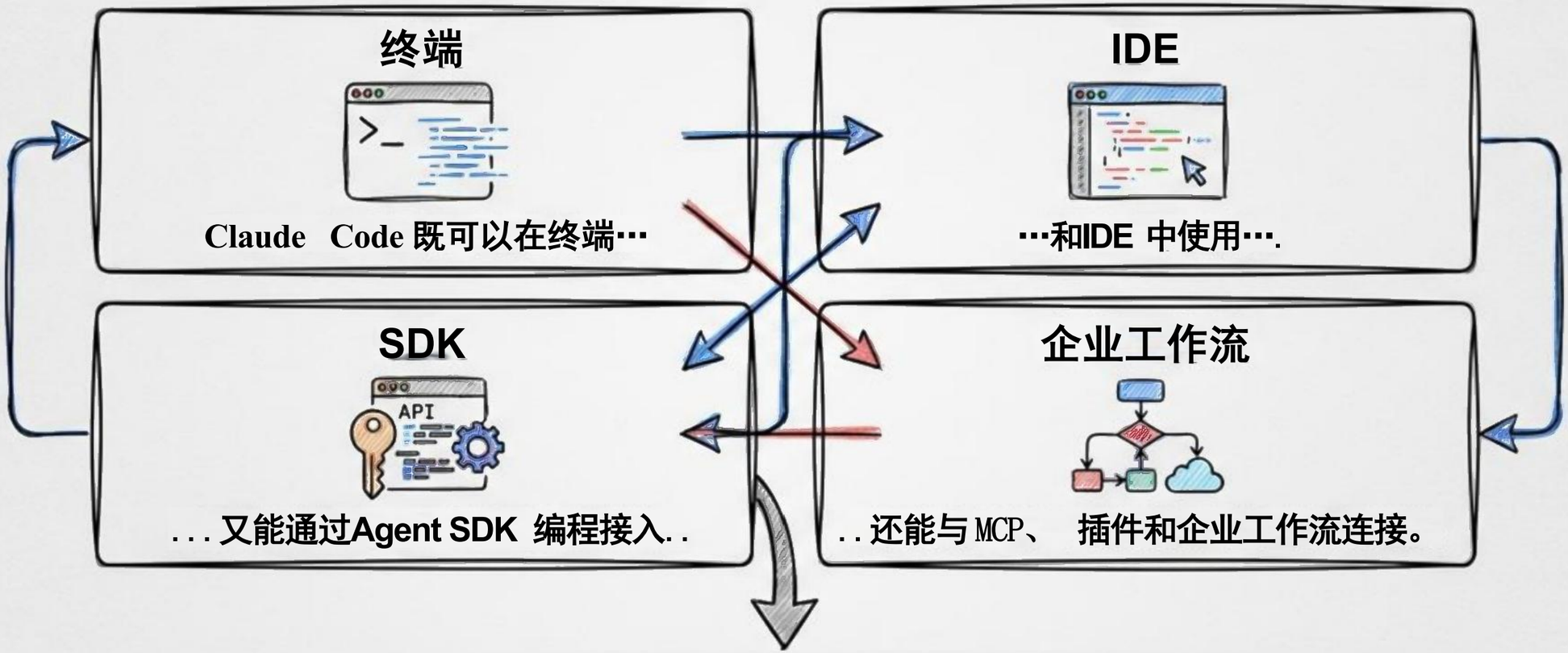
当最困难、最可验证、最有价值的任务不断流回模型团队时，能力提升会更贴近真实工作。

这让Anthropic的产品反馈几乎天然可转成能力反馈。

@清新研究|2026年3月

编码如何反哺产品

编码产品还有一个额外优势：它天然连接 IDE、终端、API 与企业流程
因此一项能力改进可以快速跨多个入口释放价值。



这种分发面让单次改进不必等待单一客户端发布，而是可以同时作用在多个产品界面上。
发布价值的扩散速度越快，团队越愿意继续采用高频发布。

组织形态

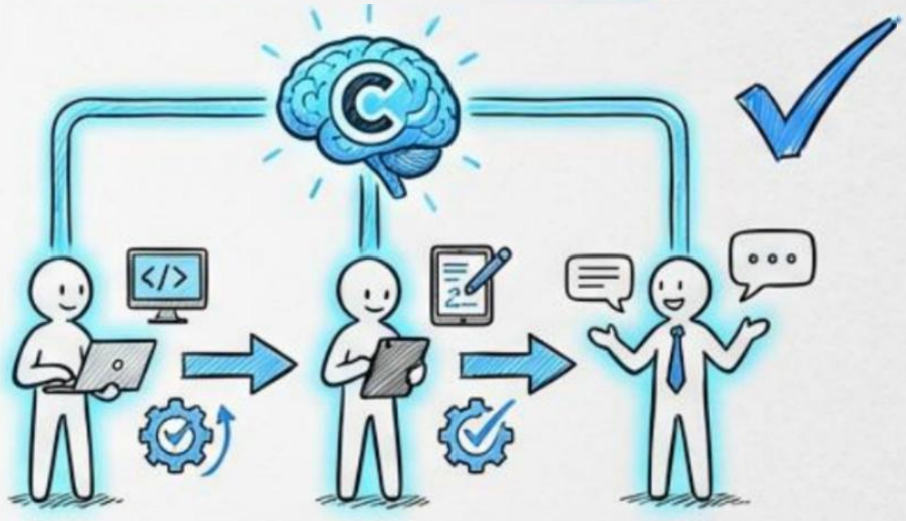
小团队、短链路、强工具，会把组织从“排队制”改成“并行制”

传统排队制

Anthropic的速度不需要每一步都经过重文档和长排期。

当个体被Claude 放大后， 一个小团队就能承担原本需要更多角色接力的工作，于是协调成本下降。

并行制(AI 放大)



这会让组织更容易形成短链路试错：先做、先用、再 修 ，而不是先协商完整、再执行。

因 此Anthropic 的速度既来自模型，也来自“更少排队”的组织结构。

稳定的组织味觉，会放大速度收益

“速度不只是权限，更是长期一致的产品判断。 ”

●公开资料显示， Anthropic 七位联合创始人仍在公司，并持续把安全与产品能力作为共同主线。

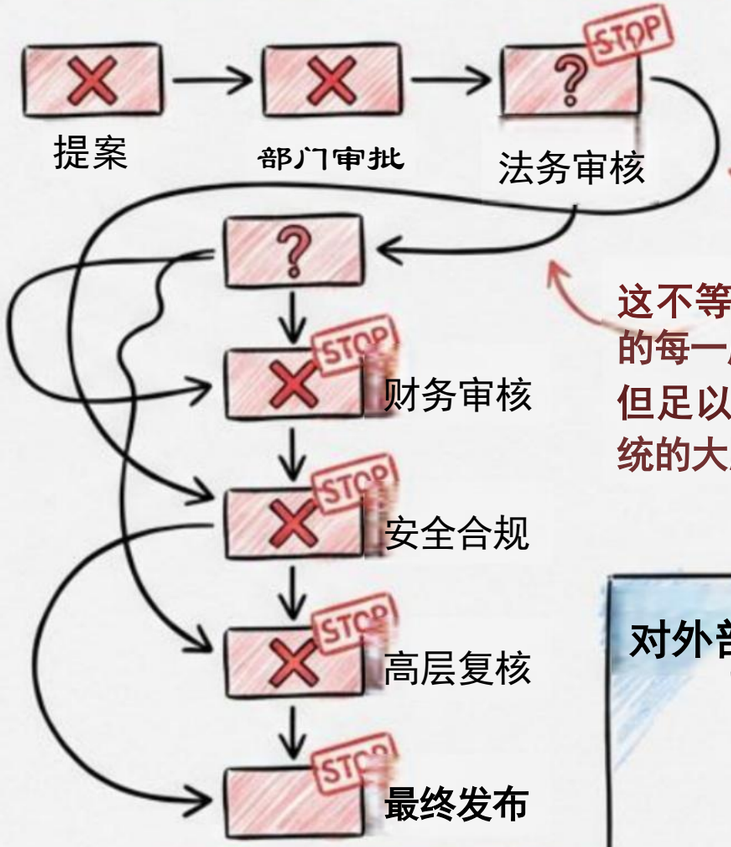
●这种连续性的重要性在于：当工具、模型与政策持续演进时，组织仍保持比较稳定的判断基线。

● 速度要持续，不只要快做，还要少走反复推翻的弯路。

@清新研究 |2026年3月

短决策链的推论

左边长审批链

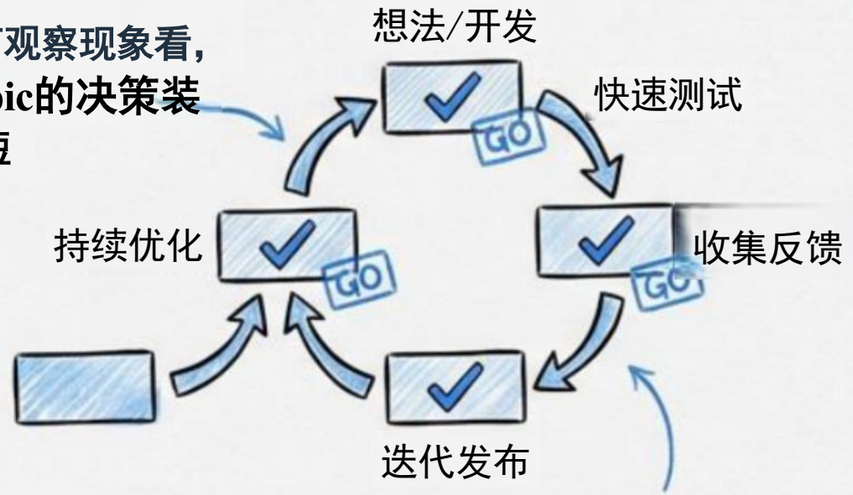


这不等于我们掌握了它的每一层审批机制，
但足以说明它没有被传统的大版本门槛困住。

本页为推论，不是内部纪实

从外部可观察现象看，
Anthropic的决策装明显更短

右边短链路试错



官方披露的密集发布、连续release notes和快速商业化，都更符合短链路试错而非重审批组织。

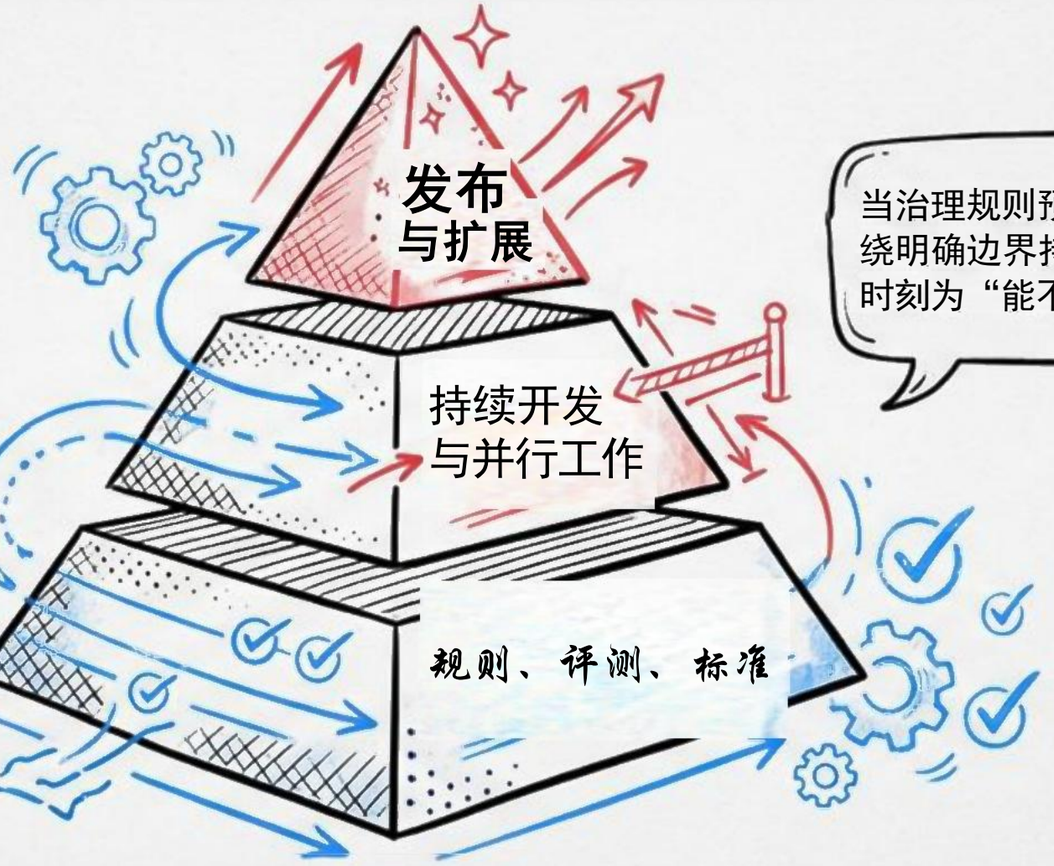
对外部团队来说，这一页的重点不是猜它内部怎么批，
而是看到“快”一定对应更低的协调摩擦。

我们把这作为基于发布节奏的推论，
而不是未经证实的内部事实。

@清新研究|2026年3月

治理并没有成为Anthropic 的刹车，而是被工程化成了护栏

这就是“治理并行化”的核心。



当治理规则预先清晰，团队就可以围绕明确边界持践开发，而不是在最后时刻为“能不能发”展开重复博弈。

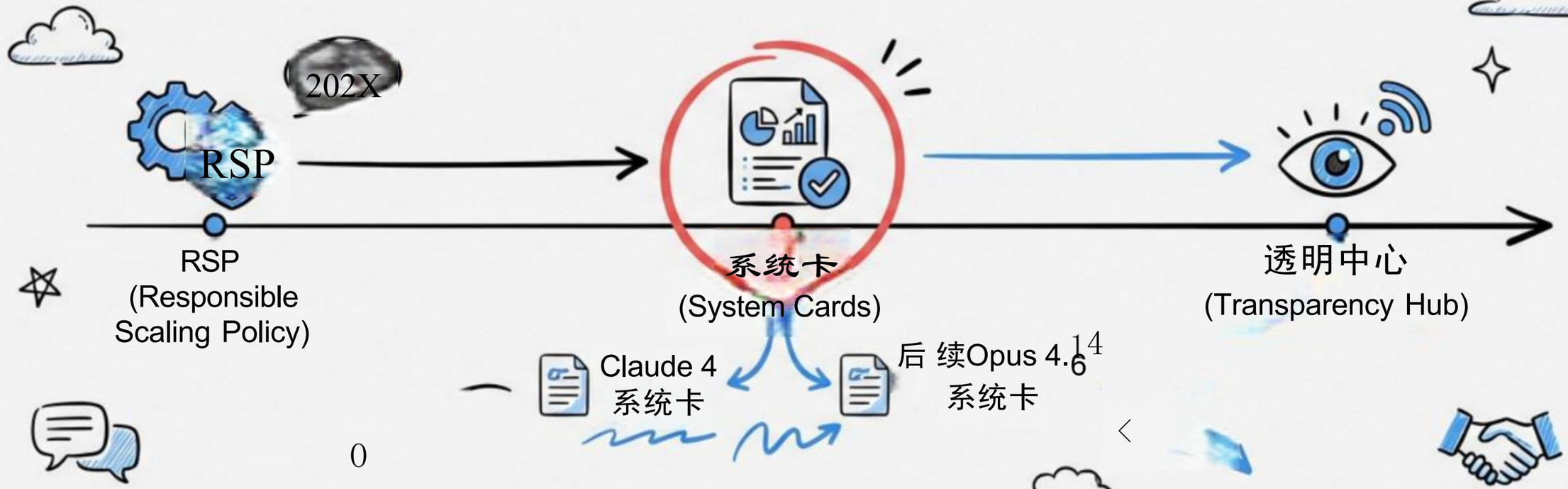
Anthropic 的 RSP不是上线前临时拉起的审批会，而是一套预先声明的能力闻值、评测要求与发布标准。

对于高能力模型团队来说，这种前置治理比事后争论更能保护速度。

@清新研究 | 2026年3月

系统卡、风险报告与透明机制，减少了“最后一分钟政策扯皮”

把风险信息结构化公开，本身就是一种加速方式。



14

0

- Claude 4系统卡与后续Opus 4.6系统卡，把能力评测、风险判断和部署决定以文档化方式对外说明。

- 这种文档化的价值，不只是对外沟通，更是让内部在发布时有可复用模板和共同语言。

- 当治理信息可以复用，组织就不必每次从零开始争论 。

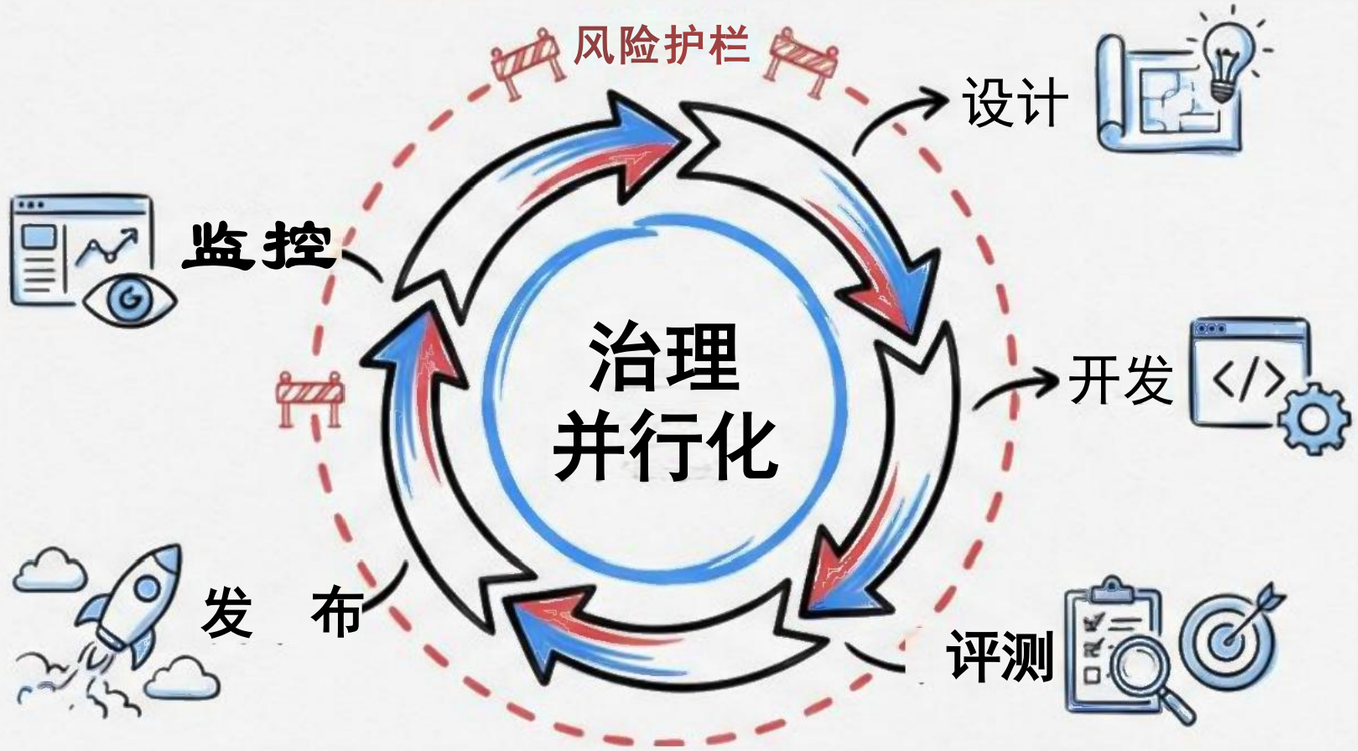
@清新研究 | 2026年3月

真正快的组织，会把治理做成运行时，而不是末端审批

“Anthropic的独特点在于：安全不是速度的对立面，而是速度的边界条件。”

“NIST 的 AI RMF 与 GenAl Profile**都强调，风险管理应嵌入设计、开发、使用与评测全过程，而**不是末端附加项。”

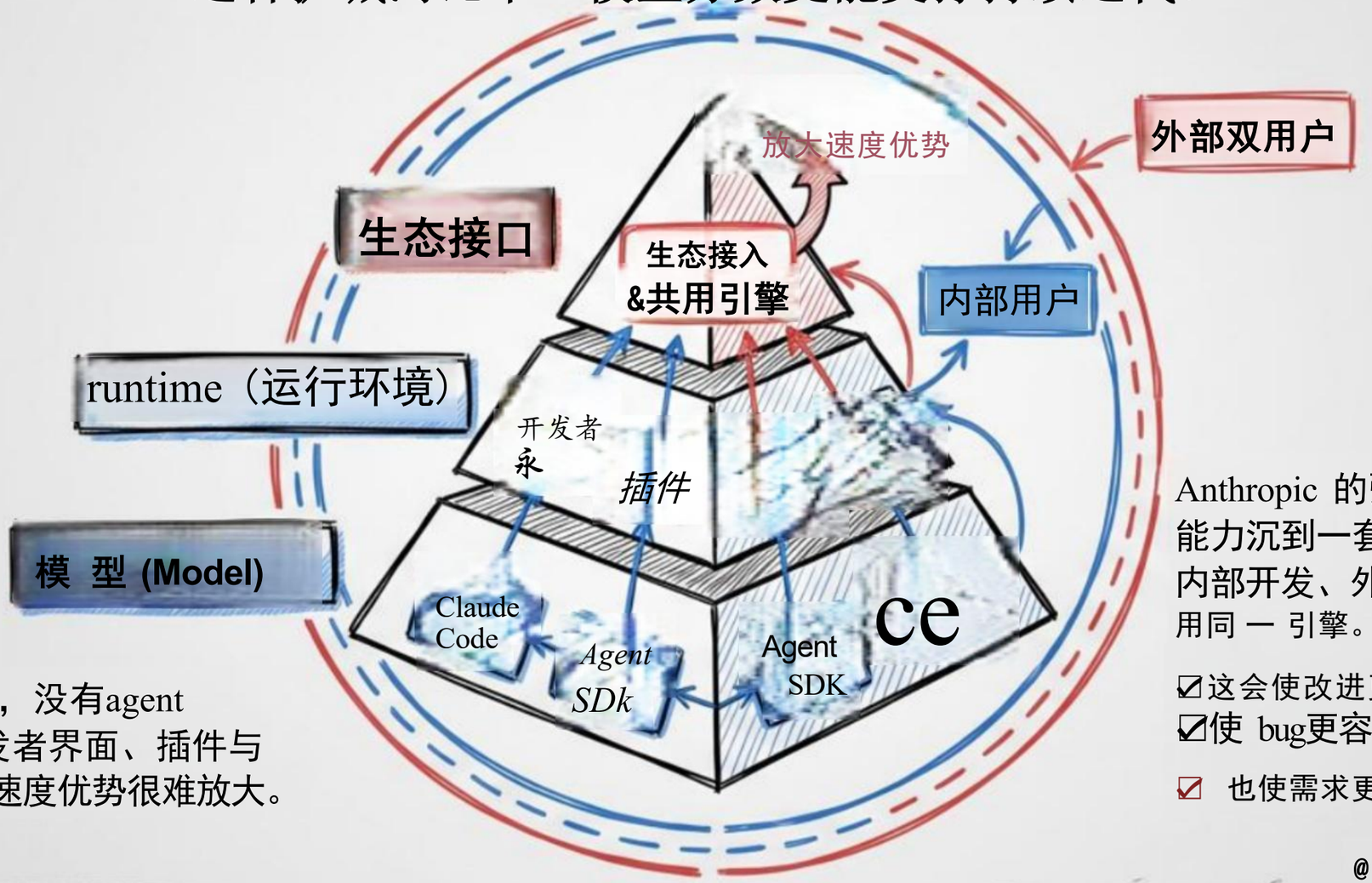
“Anthropic的RSP 与系统卡实践，正好是把这类原则落成内部可运行机制的例子。 ”



因此它的速度不是“绕过治理”，而是“让治理更少阻塞”。

Claude Code +Agent SDK+MCP 构成的是一整套工具化护城河

这种护城河比单一模型分数更能支撑持续迭代。



如果只有模型，没有agent runtime、 开发者界面、插件与上下文管理，速度优势很难放大。

Anthropic 的强点在于，它把模型能力沉到一套可复用工具链里，让内部开发、外部使用和生态接入共用同 一 引擎。

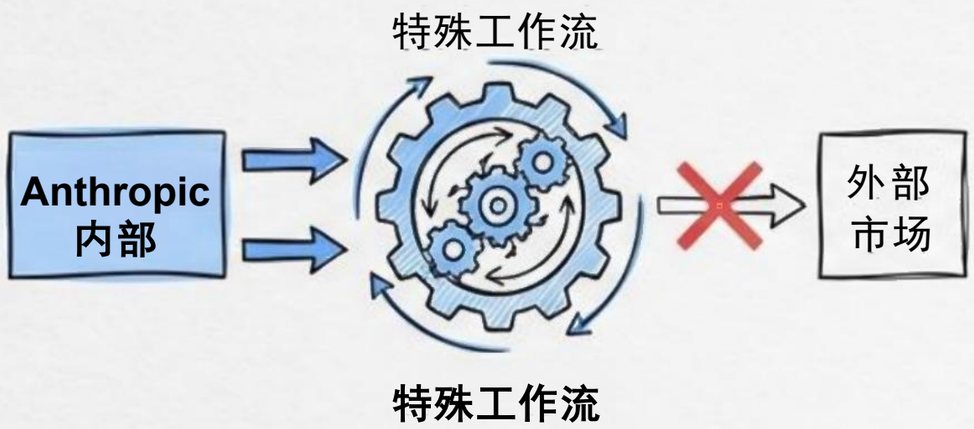
☑这会使改进更容易扩散，
☑使 bug更容易复现，
☑ 也使需求更容易沉淀为平台能力。

@清新研究 | 2026年3月

什么样的能力能在Anthropic内部先跑通，再向外迁移

可迁移性决定了内部飞轮能否外化成市场优势。

内部专用能力



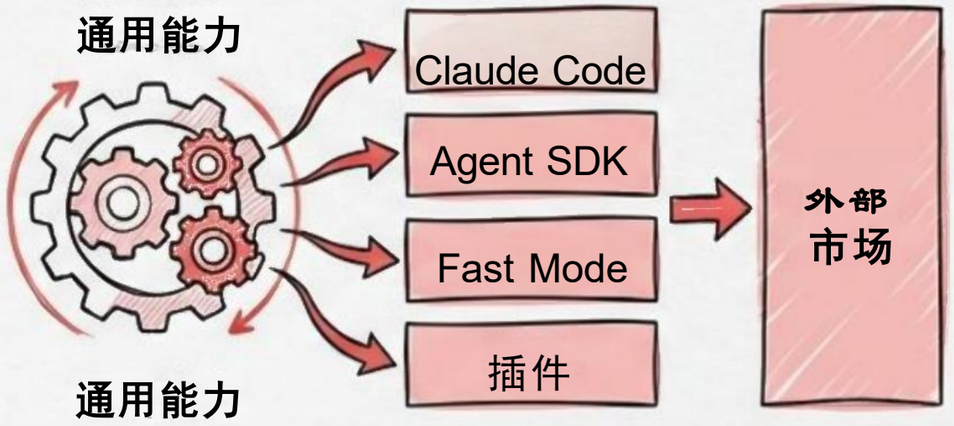
左侧是只能在内部成立的特殊工作流

内部投资不会快蚀死

Anthropic 的优势在于，它开发出来的很多“内功”正好具有可产品化潜力，因此内部投资不会被锁死。

内外部可迁移性

内外同构能力



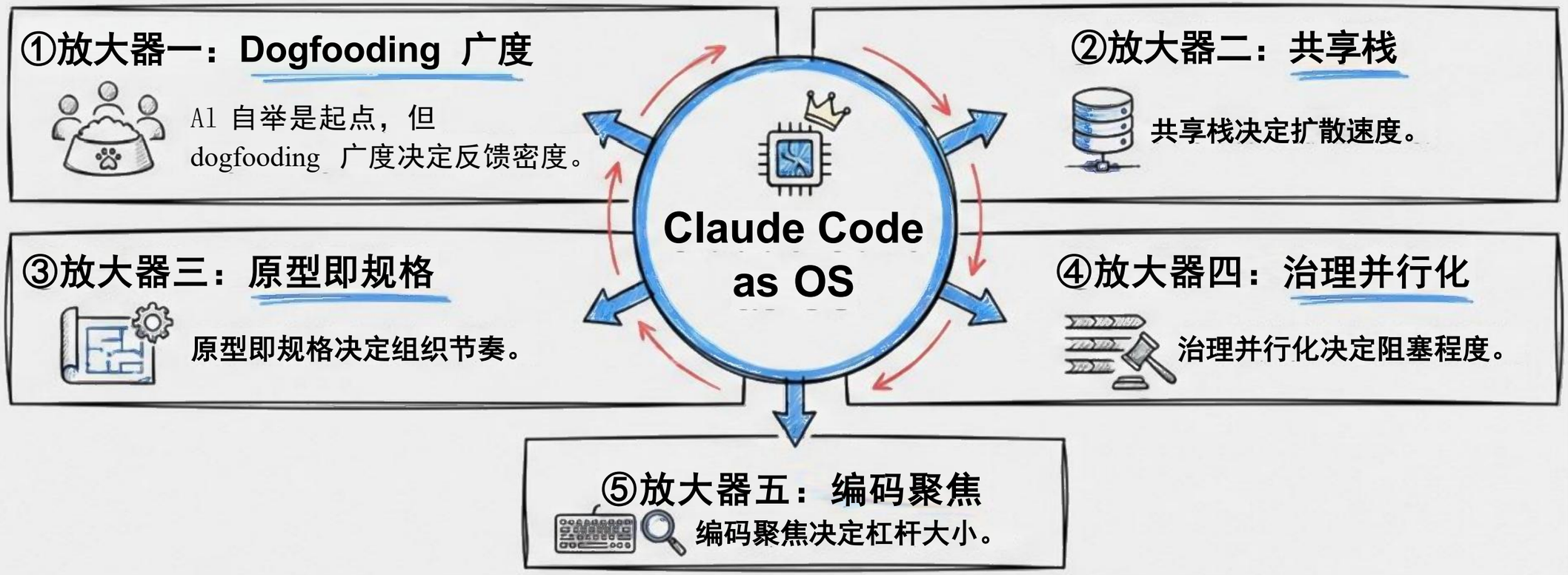
右侧是可以包装成 Claude Code、Agent SDK、Fast Mode或插件的通用能力。

市场优势

这进一步解释了为什么它的速度会不断被市场验证，而不是只在内部自嗨。

为什么“一个核心原因”仍然需要五个放大器

没有放大器，Claude Code只是工具；有了放大器，它才变成组织操作系统。



这五个条件共同作用，Anthropic 才会从“做出一个好工具”迈向“整家公司变成更快的系统”。

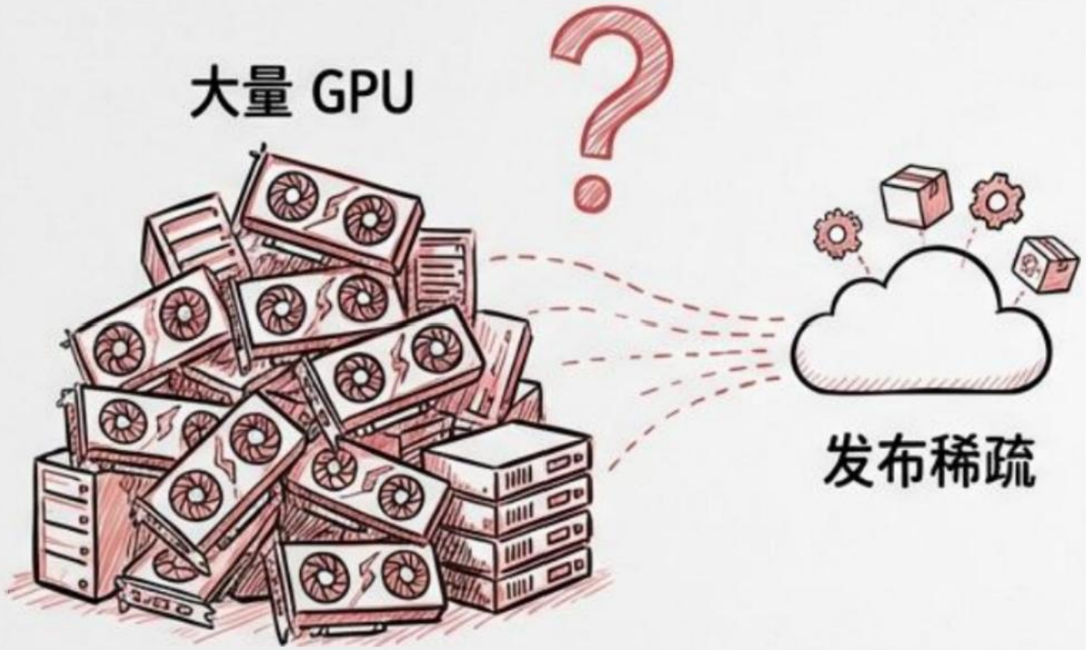
因此外界看到的不是单个产品成功，而是一种新的AI组织范式。

@清新研究|2026年3月

第三部分

边界、反证与弱证据：我们相信什么，我们不采用什么？

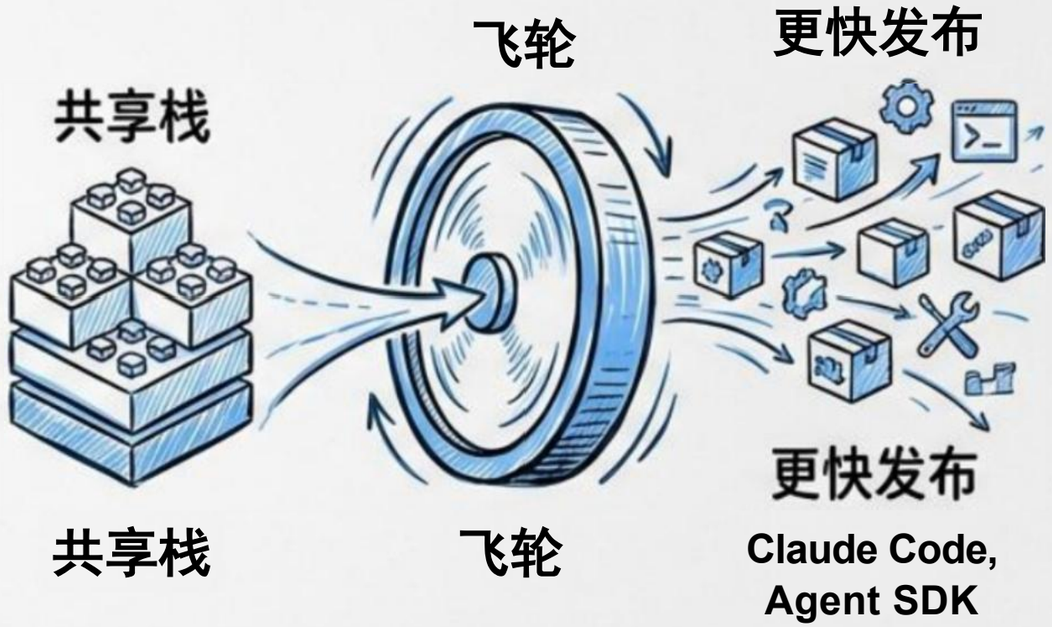
✗ Anthropic的速度优势，不足以由“算力更多”单独解释



算力重要，但它更像燃料，不像传动系统。

很 多AI 实验室都能获得大量算力，但并不是所有团队都把算力差异转成同样密集的产品节奏与工具迭代。

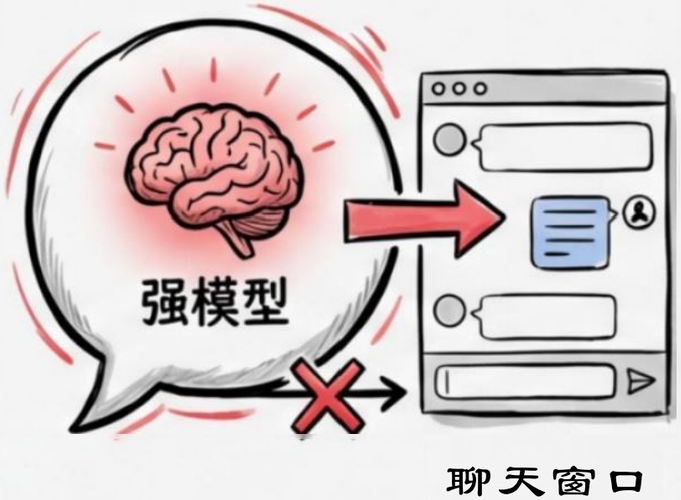
✓ Anthropic 的独特之处，是把模型能力通过Claude Code 和 Agent SDK 变成了组织速度，而不只是benchmark 成绩。



所以“更多GPU” 是背景变量，不是本报告的核心解释变量。

模型强并不自动等于组织快

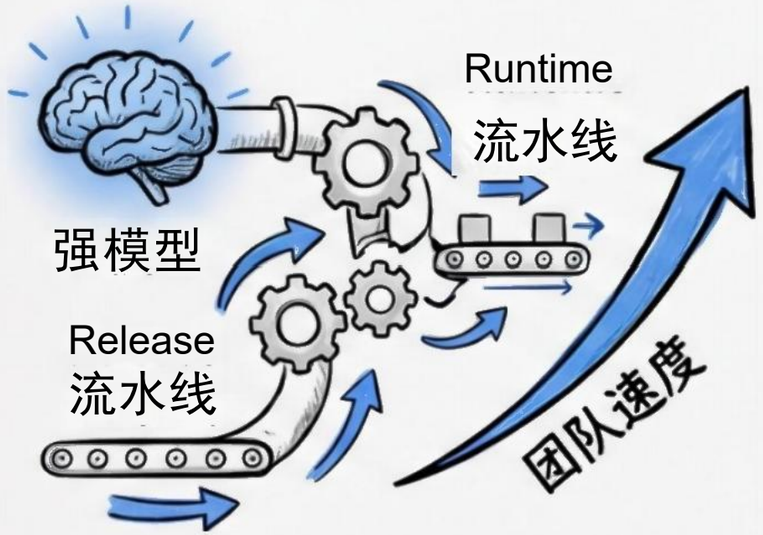
模型能力 ----------> 团队速度



如果模型强但内部吸收慢，能力不会自动变成速度。

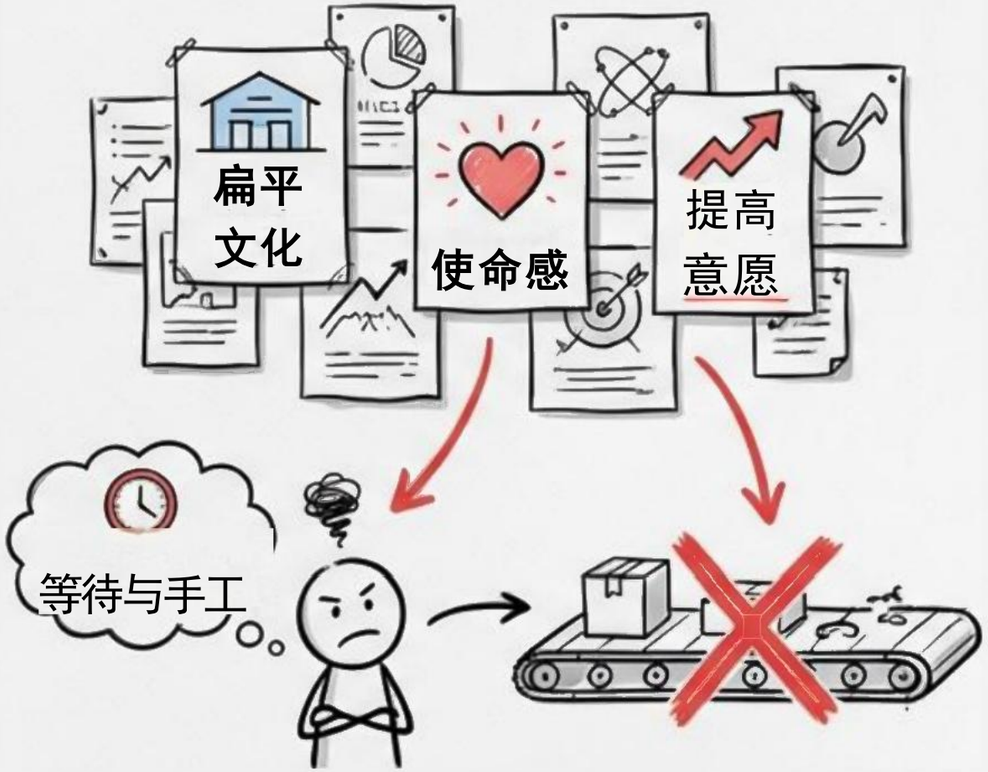
Anthropic自己的内部研究证明，速度提升依赖使用占比、任务委派方式、可验证性与工具链配置，而不只是模型本身更聪明。

这正是很多公司有强模型却没有同等速度飞轮的原因。



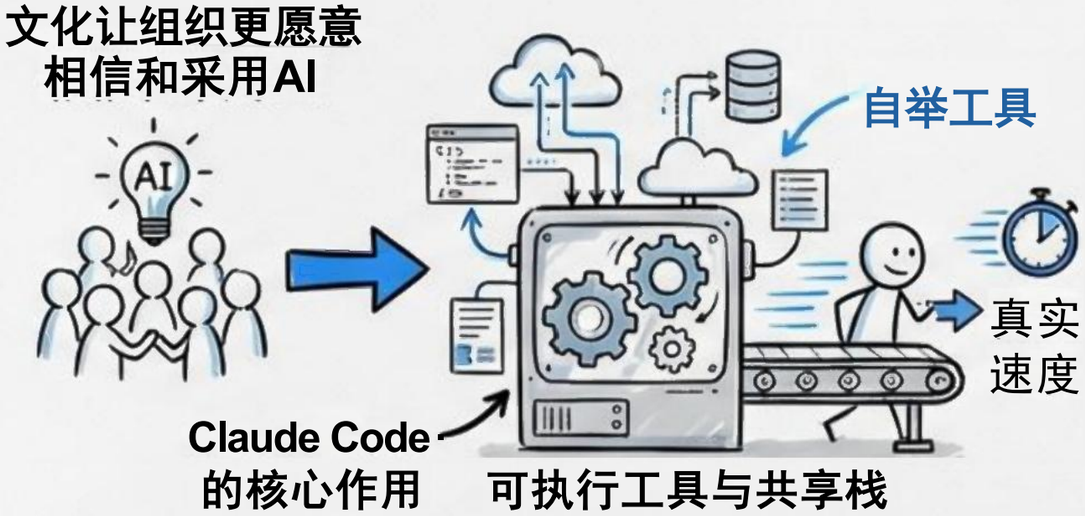
也就是说，模型能力是必要条件，但要变成团队速度，还需要被正确地嵌入工作流。

不是文化alone



- 扁平文化或使命感也不足以单独解释速度
- 如果没有可执行工具与共享栈，文化只能提高意愿，不能压缩周期。

文化+工具系统=真实速度



- 文化稳定性能减少内耗，但真正把周期压短的，还是能直接替代等待与手工实现的工具系统。
- 因此把Anthropic 的快仅仅解释成“更扁平”或“更有使命感”，会低估Claude Code的核心作用。
- 正确的理解应是：文化让组织更愿意相信和采用AI，自举工具让这种信任转成真实速度。

弱证据处理

这些流传很广的数字，我们刻意没有纳入主证据链

“严谨研究最重要的品质，
是知道哪些数字先不用。”

流传数字

例如“52天74个功能”“单月23个版本”“代码库90%由AI编写”等说法，目前缺少足够稳定的一手公开出处或统一口径。

证据级别

另一些表述来自Boris Cherny的公开言论与媒体转述，例如“两个月未手写代码”“两天提交22与27个PR”，这些更适合放在轶事证据层。

处理方式

因此本报告保留这些信息作为方向性信号，

是否纳入主结论

但不让它们承担主结论。

@清新研究|2026年3月

速度飞轮也有成本：
信任、验证、上下文债
与治理负担

Anthropic 内部研究也提到，
AI 使用会带来技能退化、协
作减少、监督变难等担忧。

所以真正成熟的速度系统，
不只追求更快，还要不断升
级校验、评测、权限与训练。

越快的系统，越需要及
时修复自己的副作用。

如果agent 速度继续上升而
验证机制没有同步增强，速
度会反而转化为质量债。

时间线从增益走向风险点，再回到护栏升级。

@清新研究 |2026年3月

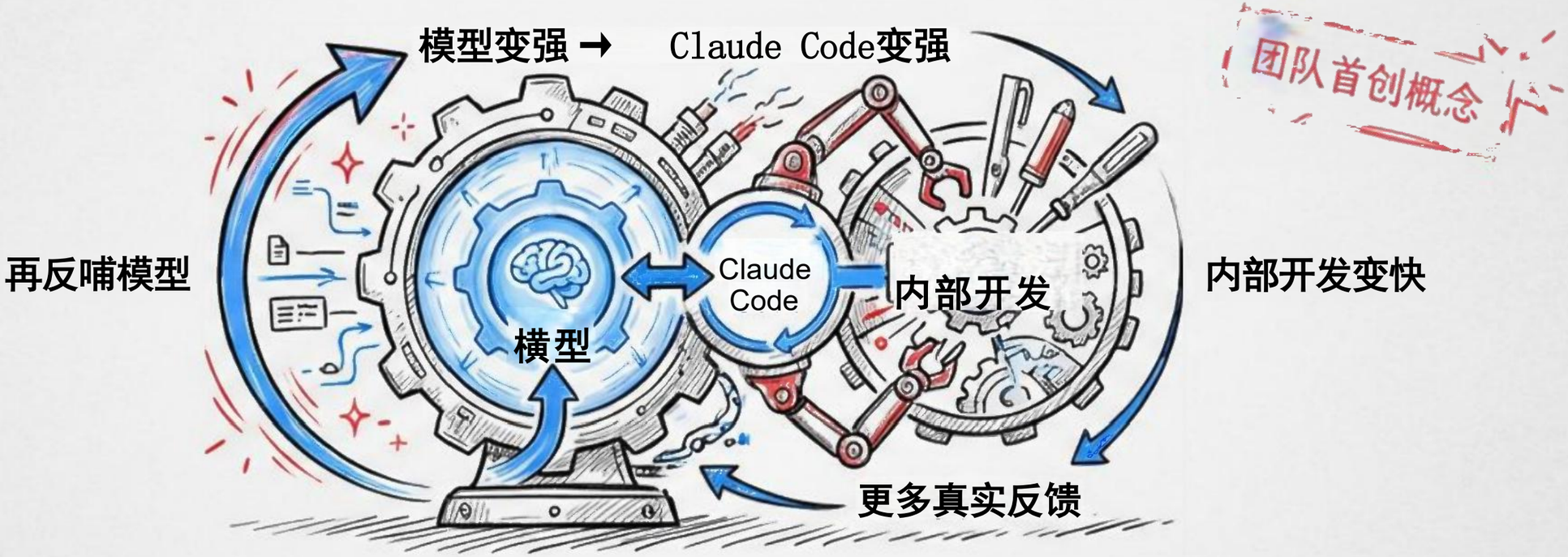
04

第四部分

原创概念

自举飞轮

自举飞轮=模型变强+ Claude Code变强+内部开发变快+更多真实反馈+再反哺模型



Anthropic 最关键的突破，不是先做出一个“好用工具”，而是让这个工具直接参与公司自己的研发。

这样一来，模型能力、工程效率和产品反馈不再是三条线，而是一个自我增强系统。

自举飞轮一旦跑起来，组织速度就会从线性提升变成复利提升。

@清新研究|2026年3月

“原型即规格”

原型即规格=可运行原型前置+长文档后置+协调成本下降

“当AI 把把想法变成可运行原型的成本降到足够低，最重要的规格不再只是文档，而是先跑起来的东西。”

- 这会把讨论从抽象争论，推向围绕真实交互、真实代码、真实结果的快速修正。
- 原型即规格不是不要文档，而是把文档从阻塞前置条件变成补全和沉淀装置。

@清新研究|2026年3月

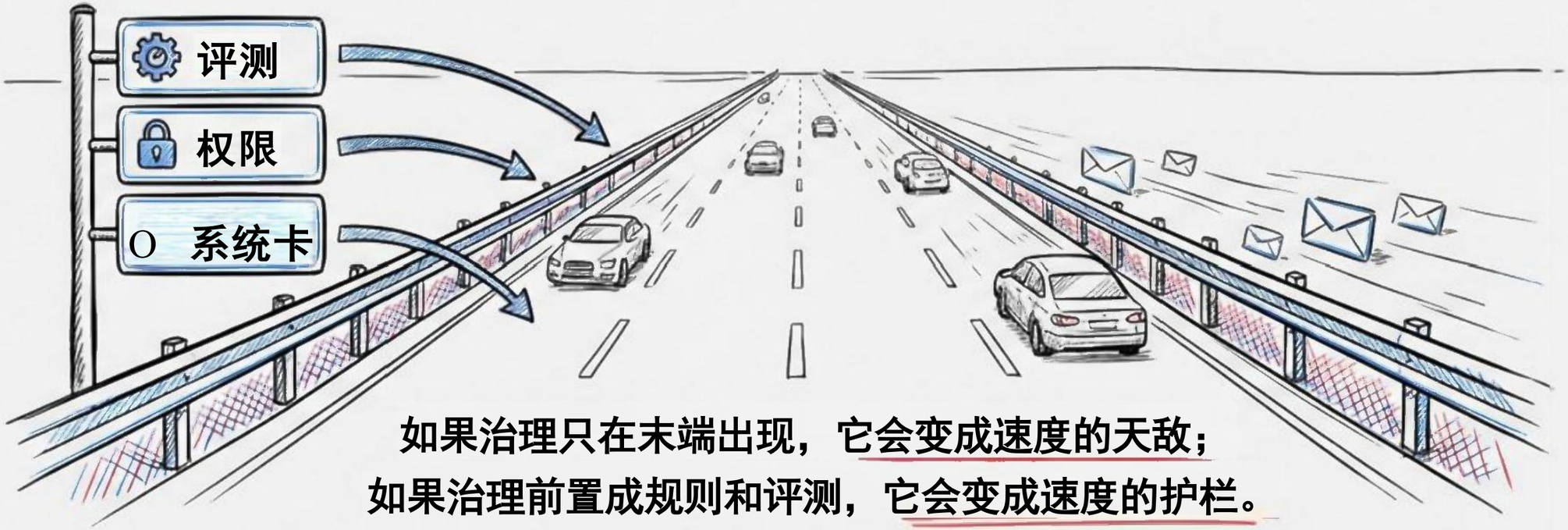
运行时型组织

运行时型组织=统一 agent runtime + 实时验证+边做边决策

- ☑ 传统组织像编译型组织，先写很多静态说明，再整体执行； Anthropic 更像运行时型组织，在统一 runtime 上不断试、改、发。
- ☑ Claude Code 与 Agent SDK 提供了共同的执行环境，使试验、实现和反馈能够在同一工作面上连续发生。
- ☑ 这类组织的最大特征，是大量协调发生在“运行中”，而不是“运行前”。

治理并行化

治理并行化=规则前置+评测内嵌+风险在开发中被持续处理



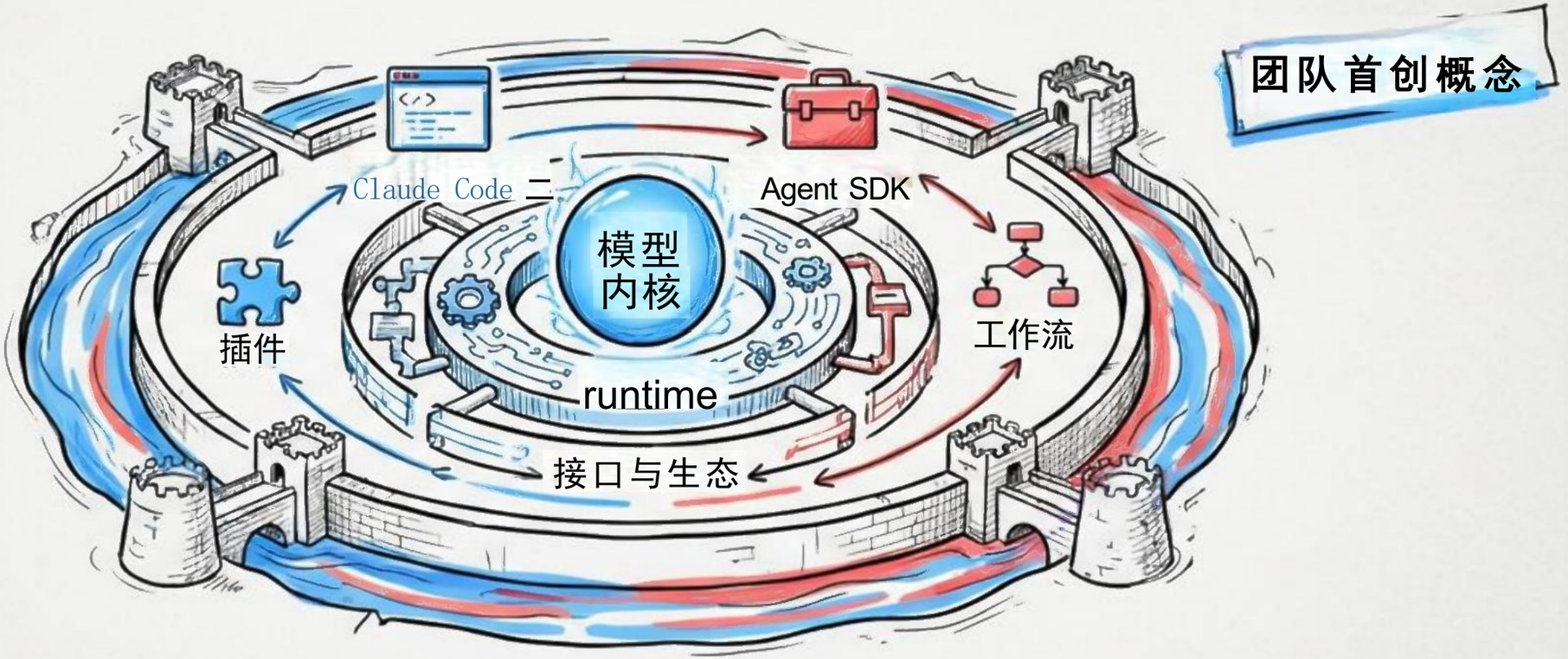
如果治理只在末端出现，它会变成速度的天敌；
如果治理前置成规则和评测，它会变成速度的护栏。

Anthropic的 RSP、 **系统卡与透明机制，展示了如何把高风险模型的治理工程化、模板化和并行化。**

治理并行化的本质，是把“能不能发”的摩擦，转成“按什么规则持续发”的制度。

“工具化护城河”

工具化护城河=模型能力沉到**runtime**、接口与生态里，形成持续迭代优势

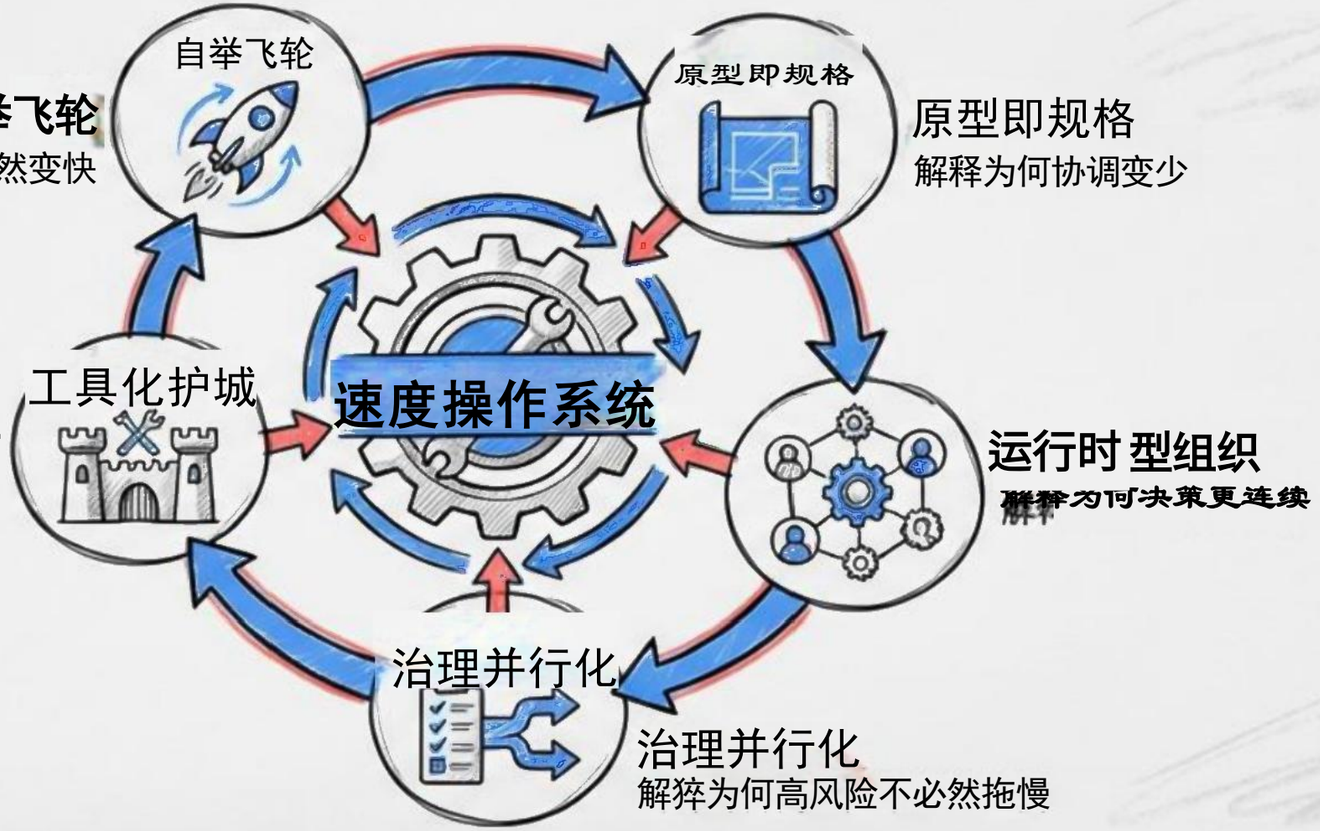


- 单一模型分数容易被追平，但被嵌入Claude Code、Agent SDK、插件与工作流的能力更难被复制。
- 因为这类能力不仅服务用户，也服务Anthropic自己的开发流程，所以每次改进都能得到双重验证。
- 工具化护城河因此比单点模型领先更耐久，也更能支撑连续发布。

@清新研究 |2026年3月

五个概念，其实描述的是同一台速度机器

它们分别解释起点、节奏、组织、治理与护城河。



自举飞轮
解释为何会突然变快

原型即规格
解释为何协调变少

工具化护城河
解释为何速度能持续

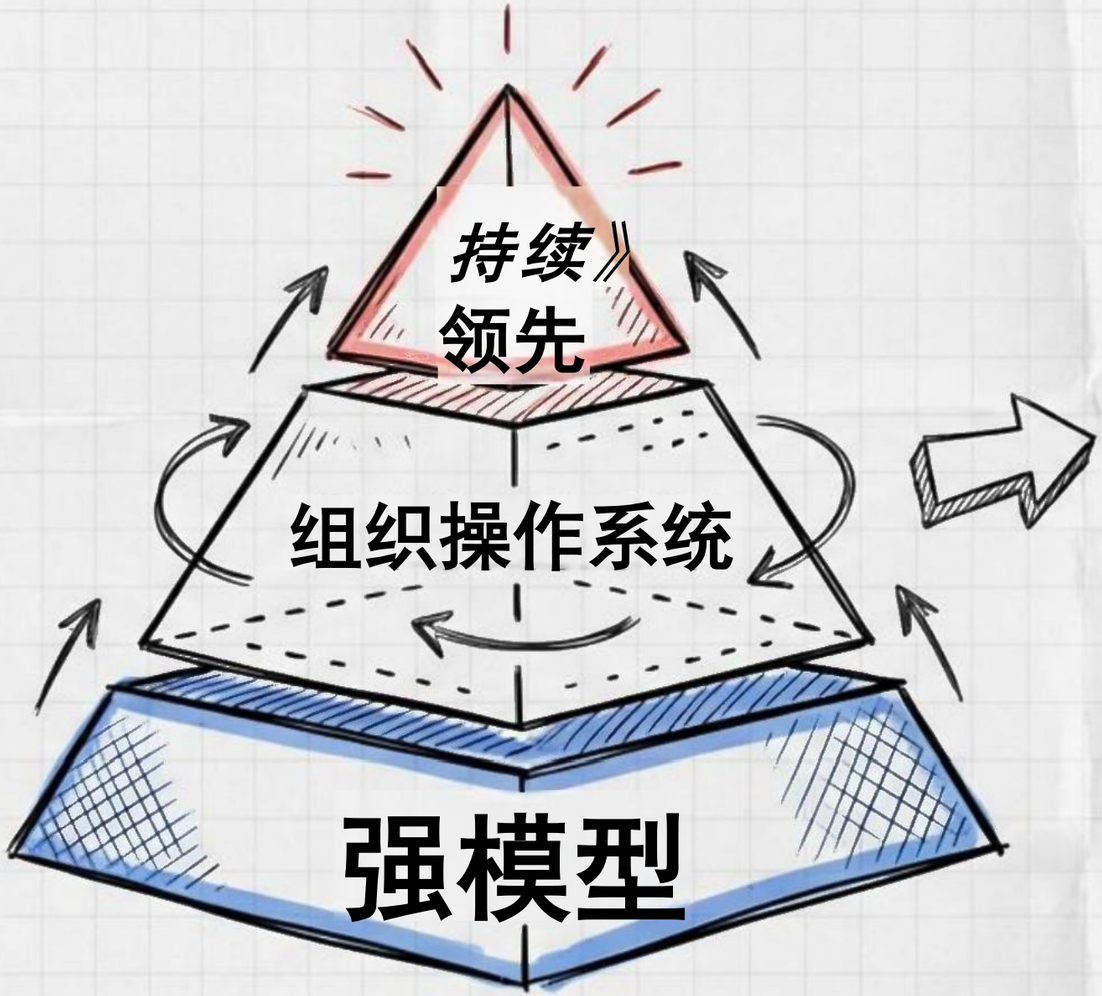
运行时 型组织
解释为何决策更连续

治理并行化
解猝为何高风险不必然拖慢

自举飞轮解释为何会突然变快，原型即规格解释为何协调变少，运行时型组织解释为何决策更连续，
治理并行化解释为何高风险不必然拖慢，工具化护城河解释为何速度能持续。
五个概念组合起来，就是Anthropic 的速度操作系统。

这也是本报告最希望留下的思想骨架。

为什么这些概念有用

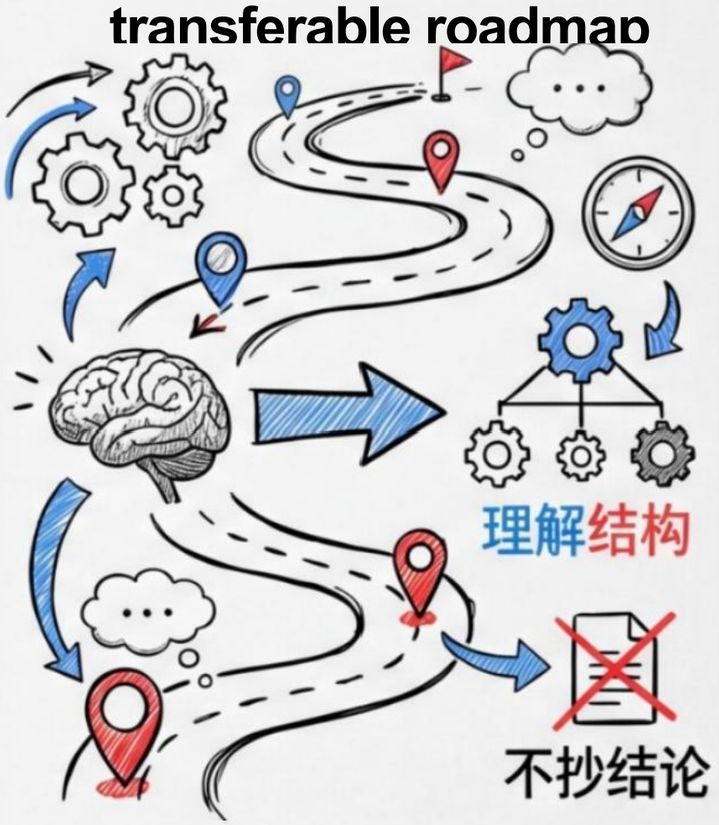


- 这些概念不只解释Anthropic，也能解释未来AI 团队的分化
- 未来领先团队，往往不是模型最强，而是速度系统最完整。
- 越来越多团队都会有强模型，但不是每个团队都能形成自举飞轮与治理并行化。
- 因此行业分化，很可能不是“谁先有模型”，而是“谁先把模型嵌入组织操作系统”。
- Anthropic的启发，在于它已经把这种分化提前显现出来。

第五部分

对外部团队的启示：如果想学快，应该学什么，不该学什么

不是抄它所有结论，而是理解它速度系统的结构。

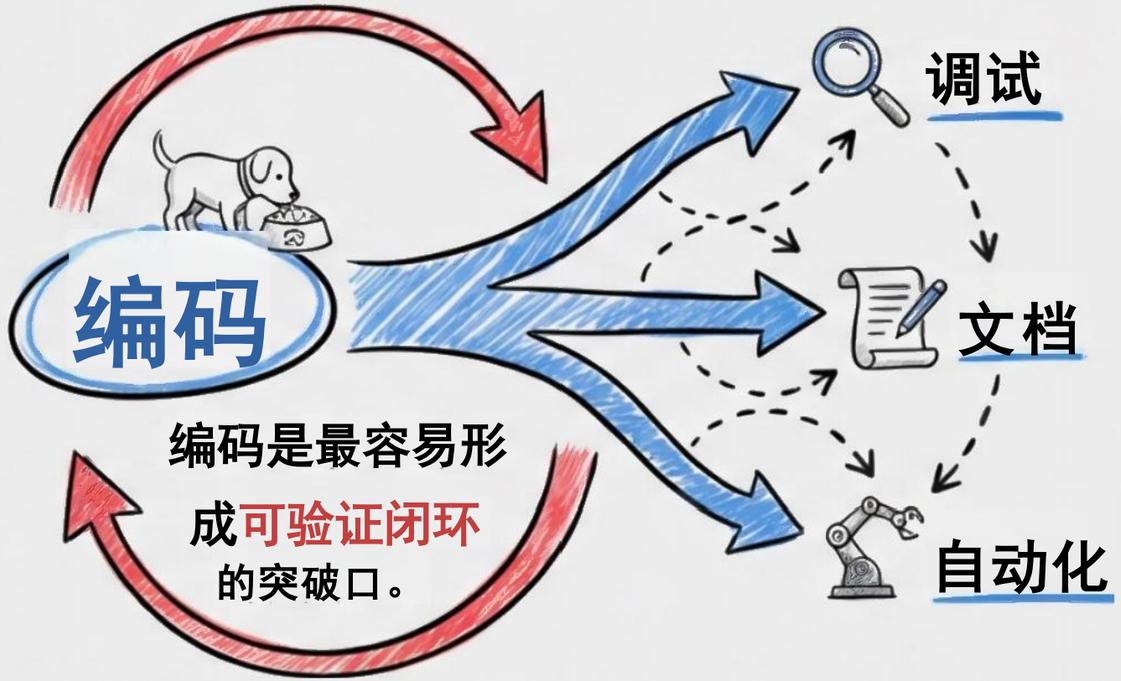


@清新研究|2026年3月

第一启示

先在编码 dogfooding 上形成突破，而不是一开始追求所有场景全面铺开

●如果你的团队也在做AI，最值得率先下注的内部场景往往仍是编码、调试、文档和自动化，因为这些任务更容易验证。



● 先在这些高杠杆、可验证任务上把AI变成默认工作面，比同时铺开所有知识工作更容易形成飞轮。

—Anthropic 的路径说明：先抓最容易复利的环节，而不是先求面面俱到。

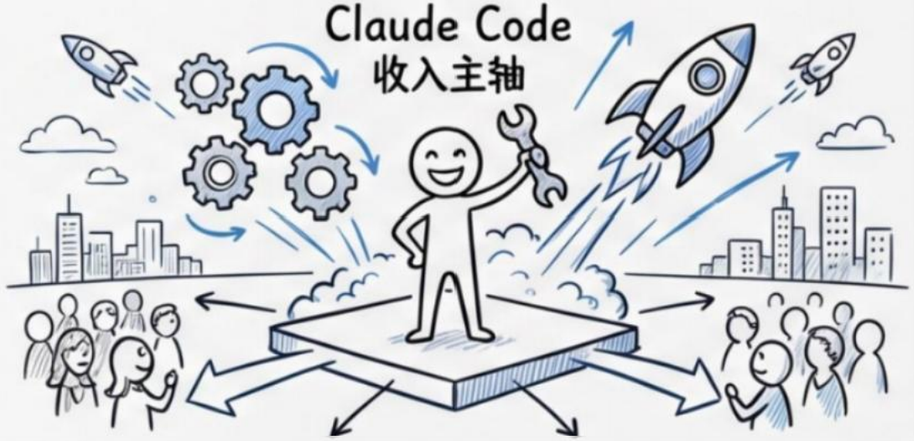
最强的内部工具，应该努力变成最强的外部产品

只有内外同构，内部投资才会不断得到市场强化。

内部工具孤岛

很多组织把内部提效工具做成“仅内部可用”，
于是其优先级总会被外部业务挤压。

内部工具成长为外部主力产品



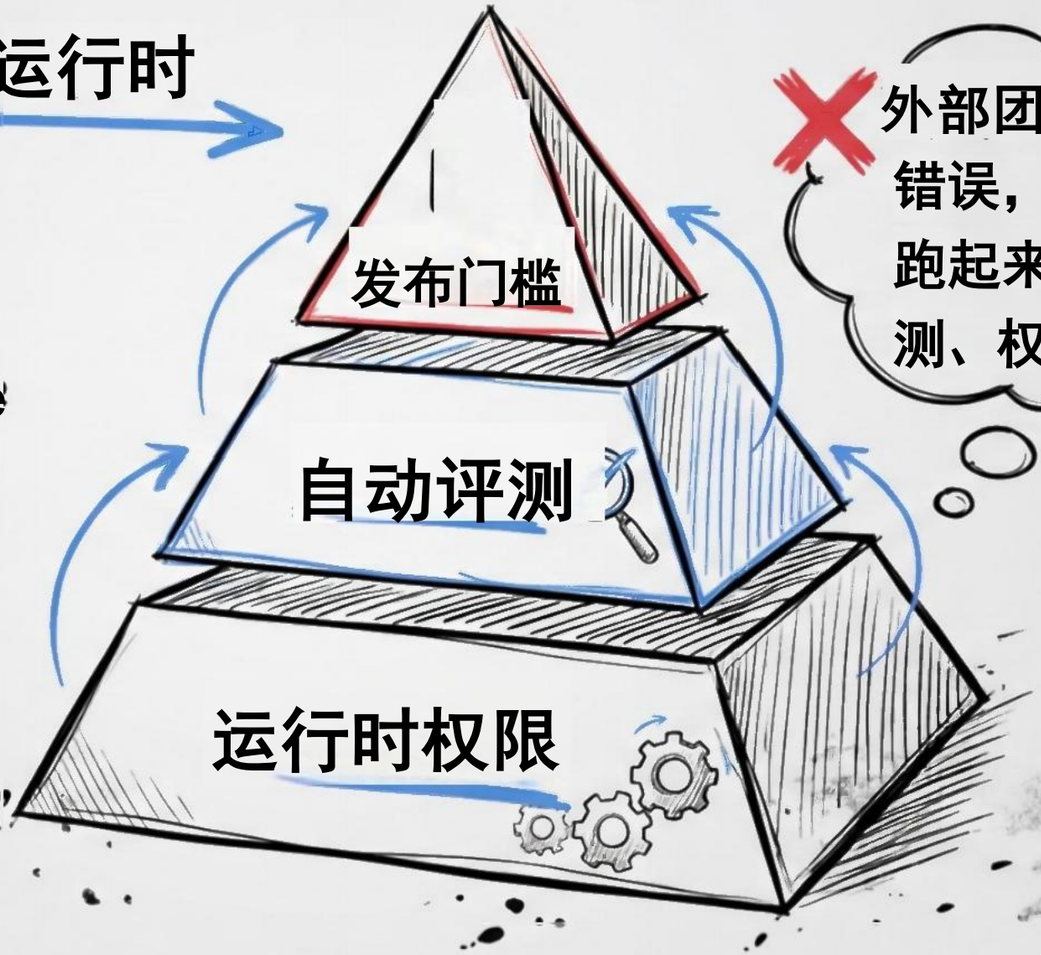
Anthropic 反过来把Claude Code做成收入主轴，
让内部提效工具与外部增长工具变成同一件事。

这类设计能让速度系统自己为自己争取预算。

第三启示

把评测、权限与治理嵌进运行时

- ☑ 越早做成系统，越不容易在速度上被治理拖住。
- ☑ NIST的 AI RMF、GenAI Profile 与 SSDF都强调，把风险管理内嵌到设计与开发流程中更有效。
- ☑ Anthropic 的 RSP 提供了一个前沿版本：用明确阈值与模板，把高风险治理做成日常工程。



外部团队最容易犯的错误，是先把 agent 跑起来，再临时补评测、权限和上线边界。

用“三只钟”测速度，而不是只看每周上线次数

真正的速度度量，需要同时看能力、发布与反馈。

能力变更钟

看模型或 agent 是否
真的能完成更难的任务

发布钟

看功能多快
进入可用状态

反馈钟

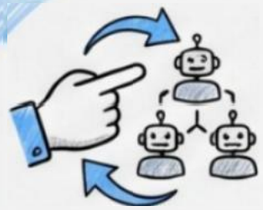
看真实使用多
快回流到改进

如果只看上线次数，团队很容易变成低质量快发；
如果只看benchmark，团队又会变成高分低产。

Anthropic 的价值就在于三只钟相互校验。

工程师会越来越像“Al主管”，组织要重做角色定义

第五启示



委派（Delegate）

工程师向Al agent分配任务，设定目标与边界。

验证（Verify）

检查AI输出的结果是否准确、符合要求，进行初步确认。

审校（Review）

对AI生成的内容进行深度审查、修改和优化，确保质量。

责任（Accountability）

为最终成果负责，承担风险，把握整体方向。

- 速度飞轮会改变岗位，而不是只改变工具。
- Anthropic研究里已经出现“工程师更像管理Al agent的人”的描述，这意味着角色重心在转向委派、验证、审校和最终责任。
- 外部团队如果还把Al 当作单纯的插件，而不改变岗位边界，很难吃到全部速度红利。
- 下一代高效团队，很可能是“少量专家+大量agent+ 强验证习惯”的组合。

@清新研究|2026年3月

第六启示

坚持小批量、连续发布，而不是攒功能包

快团队的核心美德，不是“大动作”，
而是“更短的学习半径”。

小批量发布能让错误更小、解释成本更低、回滚更快，
因此长期看比大版本冲刺更容易保持高速度。

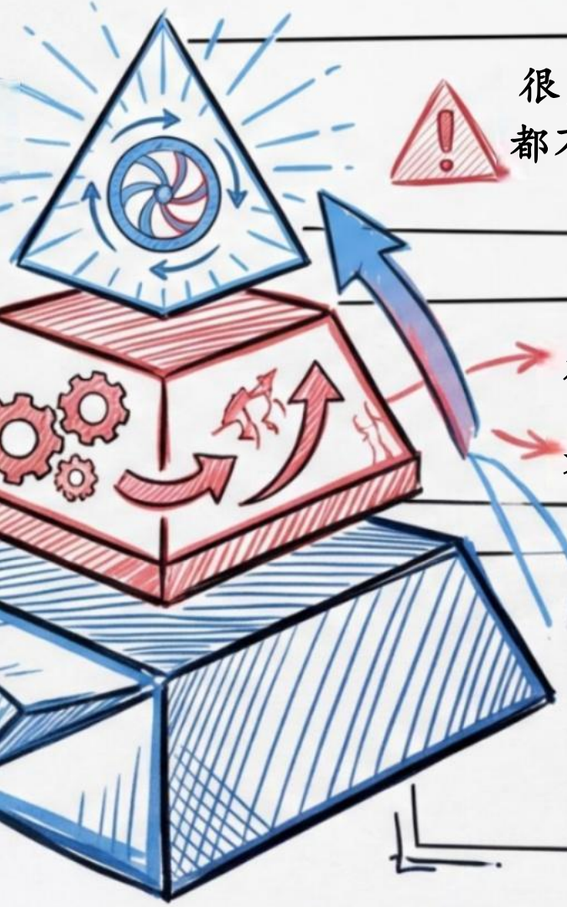
Anthropic的 release notes与2026年1月超过30项发布，
正是这一哲学的公开证据。

真正的速度，不是偶尔冲刺，而是连续形成肌肉记忆。

@清新研究|2026年3月

第七启示

金字塔底部是编码楔子，上层才是更广平台。



广阔平台与成功扩张

很多团队会过早扩张到所有业务线，结果每条线都不够深，最终没有形成任何一条真正的飞轮。

平台扩张与飞轮效应

在平台扩张前，先守住一个最高杠杆楔子

先有楔子，才有平台；先形成飞轮，才有资格扩张。

编码楔子与 Agent Runtime

Anthropic 的楔子不是通用办公，而是编码与 agent runtime。

Anthropic的优势之一，是把最强火力先集中在最容易自举、最容商业化、又最能反哺自身的编码场景。

@清新研究|2026年3月

如果今天要学Anthropic,前90天该做什么

先搭最小自举系统，而不是先追求宏大平台。

90天动作

第一个30天

第一个30天，选定最可验证的内部编码与自动化场景，建立统一使用规范和基线度量。

- √ 内部编码
- √ 自动化场景
- √ 统一使用规范
- √ 基线度量

第二个30天

第二个30天，把真实失败案例、修复路径与评测脚本沉淀到统一 runtime 或agent harness中 。

- 个
- 真实失败案例
- 评测脚本

第三个30天

第三个30天，把最成功的内部工作流包装成可复用产品能力或平台接口。

- ☆ 成功的内部工作流
- 可复用产品能力或平台接口

@清新研究|2026年3月

从“会用AI”到“让AI造AI”的一年升级路径

真正难的不是装一个工具，而是把工具变成组织基础设施。

前3个月

建立dogfooding 把内部成功场景产品度量化和商业化。

6到12个月

只有走到最后一步，速度系统才会从成本中心转成增长飞轮。

1月

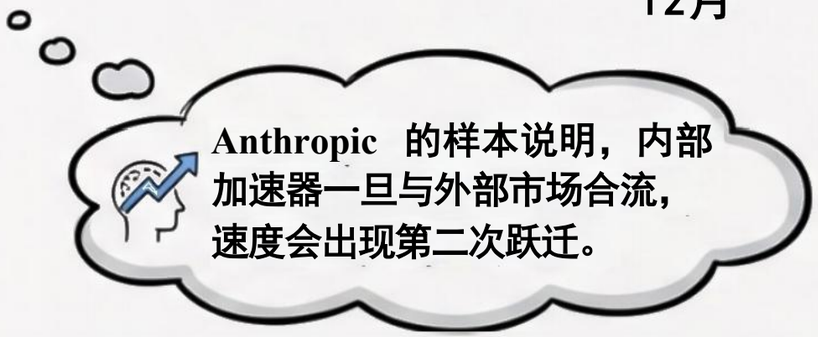
3月

6月

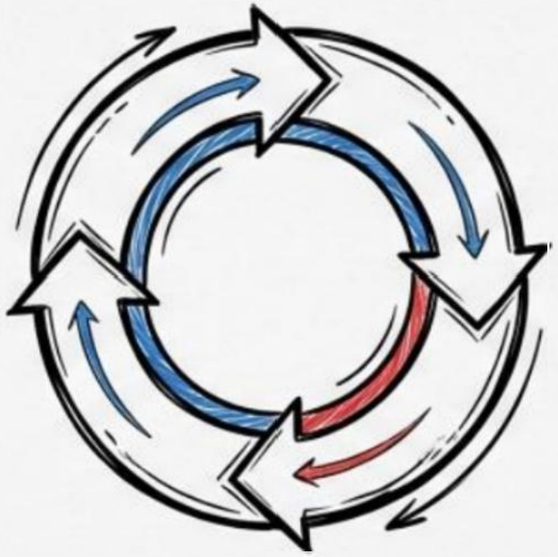
12月

3到6个月

完成内部 runtime、评测与权限体系



最后结论



Anthropic 之所以成为迭代最快的AI 团队，根本上是因为它先把自己改造成了 “AI 帮助建造AI” 的组织。

Claude Code 不是附属产品，而是Anthropic 的研发超级加速器；Agent SDK、发布节奏、治理规则和商业和商业化让这台加速器持续增压。

因此Anthropic的领先，不只是能力领先，更是速度系统领先。
只要这个自举飞轮继续转动，它在未来一段时间内仍大概率保持极强的执行优势。

@清新研究|2026年3月