— 技术 · 实践

CLAUDE CODE

从写代码到指挥 agent

官方的定位还是「编码工具」。它其实是你指挥并验收一支 agent 队的编排台 —— 写代码只是它最小的样子。

目录.

引子 · 不是聊天框，是编排台.

你为 Claude Code 付了 Max，却把它当成一个会改文件的聊天框：问一句、看它写、粘走。官方给它的定位也还是「编码工具」。但计划模式、CLAUDE.md、Skills、subagents、MCP、hooks 拼在一起，它的边界不是「能写什么代码」，而是「一台电脑上的活，有多少能由你派出去、再由你验收」。这一章讲清楚这本书的立场：Claude Code 最小的样子是编辑器，最大的样子是你一个人指挥一支 agent 队的编排台 —— 以及为什么停在聊天框那层，等于把付费买到的后一半白扔。

你大概是这样用 Claude Code 的：开一个终端，敲一句要求，看它改文件，扫两眼 diff，合并。偶尔它自己跑了个测试，你觉得挺神。一天下来，它对你来说是一个会动文件的聊天框 —— 比网页版多省下的，只是复制粘贴。这没错，但这是它最小的一层。你的 Pro 或 Max 订阅本来就含它，额度和网页版共用一个池子[1] —— 也就是说，你从没为「聊天框」单独付费，你付的钱里装着一台大得多的机器。这本书的立场一句话说完：Claude Code 最小的样子是一个会改文件的编辑器，最大的样子，是你一个人指挥并验收一支 agent 队的**编排台**。停在聊天框那层，等于把后一半白扔。

— I

名字里的「Code」，已经装不下它.

官方文档给它的一句话定位是「agentic coding tool」：读你的代码库、改文件、跑命令；入口有终端、IDE、桌面应用和浏览器四个，连的是同一台引擎。[2] 这个定位没有错 —— 只是太谦虚。往同一页文档底下翻：用 MCP 接 Google Drive、Jira、Slack；用 Routines 把活挂到日程上定时跑；用 Channels 把 Telegram、Discord 的消息接成任务；开 agent teams 让几个 agent 并行分工；用 `claude -p` 把它嵌进脚本和 CI。[2] 这些能力没有一个长得像「编码工具」—— 它们长得像一张派活的台子。这本书不列全部功能。它按「一台编排台需要什么」挑出核心的六件，每件一章：[3]

部件	它替你做什么	哪章
计划模式	动手前先交方案,你批准了它才改	第 2 章
CLAUDE.md	项目的长期记忆,每次开工自动加载	第 3 章
Skills	把重复的流程收编成资产,用到才加载	第 4 章
Subagents	独立上下文的分身,干完只交回结论	第 5 章
MCP	把 issue、工单、数据库接进上下文	第 6 章
Hooks	固定时机自动执行的规矩,不靠模型自觉	第 7 章

单看,每一件都不新奇 —— 别的工具也有记忆、也有插件。拼在一起才变质:计划模式让你从「看它写每一行」退到「审方案、验结果」[4];CLAUDE.md 和 Skills 让你不用每次重新交代;subagents 让活离开你的上下文,headless 让活离开你的终端(第 8 章);hooks 把「希望它记得」换成「到点必然发生」[3]。每一件都在回答同一个问题:**怎么让你少盯一点,而不失控。**

· · ·

— II

边界重画:能派多少,验得动多少.

> **Claude Code 最省的不是敲代码的时间,而是你从「自己动手」切到「派活 + 验收」的整个工作方式。**
>
> 本书的第一条判断

所以量它的尺子得换。旧尺子问「它能写什么代码」—— 按这把尺子,它和一打同类挤在同一条跑道上。新尺子问「一台电脑上的活,有多少能由你派出去、再由你验收」—— 按这把尺子,写代码只是它顺手的一类活:排查一个线上报错、迁移两千个文件、盯着 CI 修失败、把一夜的日志读成三行结论,都在射程里。注意新尺子的后半句。派活是容易的一半;难的一半是确认它真做对了。官方最佳实践排在最前面的一条,不是「写更好的 prompt」,而是给它一个**它能自己跑的检查** —— 测试、构建、一张能对比的截图 —— 因为能自动验的活,你才敢真放手。[5] 这本书把「验收」单独设为第 9 章,也是全书的书眼:别的教程教你怎么派活,这本书还要你带着凭据收活。

> — 补充
>
> 立场照旧说在前面:我是付费用它的人,不是它的产品经理。官网自己能写的话,这里不写。Claude Code 以周为单位在变 —— 书里每个事实都标了核对日期,过期请以官方文档为准。它也不是所有活的最优解:哪些活该留给 Cursor、Codex 或 Warp,第 10 章照实摆。

• • •

— III

先量一次你的活.

往下读之前,先给自己的活拍一张底片:

01

翻两周,挑 5 件你亲手做完的活

不限写代码:排查过的一个报错、迁移过的一批文件、清洗过的一份数据、写过的一份发布说明 —— 挑 5 件当时从头到尾自己动手的。

02

给每件写一条「它能自己跑的检查」

想象把这件活派出去:你能不能写出一条测试、一条命令、一个可对比的产出,来判断它做没做对?写得出、写不出、拿不准,各自标上。

03

数「写得出」的有几件

那几件是你现在就能派出去的活;「拿不准」的部分,是这本书接下来九章要补的。读完回来重标一遍 —— 这张表就是你的前后对照。

参考.

01 Claude 帮助中心 · Use Claude Code with your Pro or Max plan —— Pro 与 Max 订阅均直接包含 Claude Code:一份订阅同时覆盖 Claude 应用(网页 / 桌面 / 移动)与终端里的 Claude Code,两边用量共享同一个额度池。截至 2026-07-17。Claude Help · Claude Code on Pro/Max

02 Claude Code Docs · Overview —— 官方一句话定位:「agentic coding tool」,读你的代码库、改文件、跑命令;入口有终端、IDE、桌面应用与浏览器,连着同一台引擎(CLAUDE.md、设置、MCP 跨入口通用)。同一页列出的能力还有:MCP 接 Google Drive / Jira / Slack、Routines 定时任务、Channels 接 Telegram / Discord 等消息渠道、agent teams 多 agent 协作、headless(claude -p)与 Agent SDK。截至 2026-07-17。Claude Code Docs · Overview

03 Claude Code Docs · Extend Claude Code —— 扩展机制总览:CLAUDE.md 是每次会话都加载的持久上下文;Skills 是按需加载的知识与流程;subagents 在隔离上下文里干活、只交回摘要;MCP 接外部服务;hooks 在生命周期事件上必然触发。并明确划线:写在 CLAUDE.md 或 skill 里的指令「是请求,不是保证」—— 必须每次都发生的规矩,要写成 hook。截至 2026-07-17。Claude Code Docs · Extend Claude Code

04 Claude Code Docs · Permission modes —— 计划模式:让 Claude 调研、提方案,但不动手改你的源码 —— 它会读文件、跑只读的探索命令、写出一份计划;在你批准计划之前,编辑一律被挡。Shift+Tab 或 /plan 进入。截至 2026-07-17。Claude Code Docs · Permission modes

05 Claude Code Docs · Best practices —— 验证一节的第一条:给 Claude 一个它能自己跑的检查 —— 测试、构建、一张能对比的截图;没有检查,「看起来做完了」就是唯一信号,你就是人肉验证回路。并要它交证据(测试输出、跑过的命令与返回)而不是宣称成功。截至 2026-07-17。Claude Code Docs · Best practices

---

写代码是它最小的样子,
指挥一支 agent 队才是最大的.

循环 · 计划、动手、验收.

多数人对着 Claude Code 一句接一句地聊，让它边想边改 —— 然后奇怪它为什么跑偏。它真正的用法是一个循环：先在计划模式里让它把打算怎么做讲给你听、你批准，再放它动手改文件、跑命令，最后你审那份 diff。权限模式决定它哪些能自己做、哪些要问你；checkpoint 让你随时倒回。这一章把这个「计划 → 动手 → 验收」循环讲透 —— 它是你从「自己写」过渡到「派活」的第一步，也是全书其余章节的地基。

多数人对着 Claude Code 一句接一句地聊：说一个想法，看它边想边改，不对再补一句。三十分钟后 diff 大得没法看，你也说不清它是在哪一步拐错的。问题不在模型，在没有节奏 —— 它真正的用法是一个循环：先计划，你批准；再动手，权限定自由度；最后验收，不对就倒回。这一章把这个循环装进你的手里。

— I

先计划：方案没批，一个字不许改.

计划模式是循环的第一格。进入之后，Claude 只做三件事：读文件、跑只读的探索命令、写出一份「我打算怎么做」—— 在你批准这份计划之前，任何编辑都被挡在门外。[1] 进入方式有三种：会话里按 Shift+Tab 循环切换，或在单条 prompt 前加 `/plan`，或启动时 `claude --permission-mode plan`。计划出来后别急着点头：按 Ctrl+G 能把它整份丢进你的编辑器，删掉过度设计的那段、补上它漏掉的边界情况，再让它照改好的版本动手。[1] 这份被你改过、批过的计划，不只是开工许可 —— 它是第 9 章验收时的**基线**：活交回来，diff 就对着它逐条核。所以计划请求本身值得写好：说清目标，更要说清验收标准和禁区。

提示词

计划请求模板

> /plan 我要做：<一句话目标>。
> 先读 <相关目录或文件>，把打算怎么做写成计划，包括：要改哪些文件、为什么。
> 验收标准：<能检验的标准，做完要能通过>。
> 禁区：<不许动的目录 / 不许引的依赖 / 不许做的重构>。
> 拿不准的地方列成问题问我，别自己猜。
>
> /plan 我要做：给登录接口加限流。
> 先读 src/middleware/ 和 src/auth/，把打算怎么做写成计划，包括：要改哪些文件、为什么。
> 验收标准：同一 IP 一分钟内第 6 次请求返回 429，已有登录测试全部仍然通过。
> 禁区：不动数据库 schema，不新引限流库，用项目里已有的 Redis 客户端。
> 拿不准的地方列成问题问我，别自己猜。

计划也有成本，别处处开。官方的判断尺很好用：方案不确定、改动跨多个文件、代码不熟 —— 值得计划；**一句话能说清那个 diff 的活，直接干**。[2] 改个错字还要走审批，是把仪式当成了纪律。

• • •

— II

再动手：权限模式定自由度.

批准计划后它开始动手，而「动手时哪些事要问你」由权限模式决定。六档从紧到松：

模式	不问就能做	适合
default	只读操作	敏感代码、刚上手
acceptEdits	文件编辑 + 常见文件系统命令	你事后看 diff 的日常活
plan	只读操作	动手前的调研
auto	几乎全部,分类器在后台拦越界	方向可信的长任务
dontAsk	只有预批准的工具	锁死的 CI 与脚本
bypassPermissions	一切	只在隔离容器 / VM 里用

日常的甜点位是 acceptEdits:文件随它改,你看最终 diff —— 逐个批编辑的第十次点击之后,你其实已经不在审了。要走得更远用 auto:不是全放行,而是有一个独立的分类器在每个动作前把关,拦下越权范围、生产部署、危险删除这类动作,常规工作不再弹窗。[1] 原则一句话:**信方向,就松档;信不过的那几个点,用规矩钉死** —— 怎么钉,第 7 章的 hooks。

• • •

— III

随时倒回:checkpoint 是你敢放手的底气.

循环的最后一格是验收,而验收敢严,是因为错了能倒回。你每发一条 prompt,Claude Code 就自动存一个 checkpoint(会话保留最近 100 个);按两次 Esc 或输入 /rewind 打开菜单,可以只恢复对话、只恢复代码、或两者一起倒回任一节点,还能把跑偏的那段对话压成摘要腾出上下文。[3] 这改变的不只是安全感,是策略:一个激进方案,试错成本从「手工收拾残局」降到「按两下 Esc」,你就敢让它先试再说。

> — 注意
>
> checkpoint 的边界要记牢:它只追踪 Claude 用文件编辑工具做的改动 —— Bash 命令删掉的文件、你自己在编辑器里的修改,都倒不回来;官方原话是它「补充而非替代」版本控制。[3] git 该提交还是提交,checkpoint 是本地撤销,不是历史。

倒回之外还有两个更轻的纠偏动作:Esc 随时打断它、补一句再继续;同一个问题纠正两次还不对,说明上下文里已经堆满失败尝试 —— `/clear` 清场,把学到的教训写进新的第一句,几乎总比在旧会话里继续拉扯更快。[2]

• • •

— IV

循环连起来,就是最小的派活单元.

把三格连起来看:计划,是你在**定义这件活和它的验收标准**;权限模式,是你在决定盯多紧;diff 加倒带,是你在收活。这正是第 1 章说的那个切换的最小版本 —— 你已经不是在写代码,是在派一件活、收一件活。后面的章节只是把这个循环放大:记忆让你不用每次重新交代(第 3 章),subagent 让循环并行开好几个(第 5 章),headless 让循环在没有你的地方转(第 8 章)。顺带一个第 10 章的预告:`opusplan` 模型档专为这个循环而设 —— 计划阶段用 Opus 想,执行阶段换 Sonnet 干。[4]

动手 · 用一件真实的活跑一遍完整循环:

01

用 /plan 起手,亲手改一次计划

挑一件跨几个文件的中等改动,用上面的模板请求计划。计划出来按 Ctrl+G,至少删掉或改写其中一条 —— 感受「审方案」和「审代码」的差别。

02

切到 acceptEdits,数你被打断几次

批准计划后用 acceptEdits 放它做完,只在结尾看 diff。数一数这一轮你被问了几次 —— 那几次就是该进第 7 章 hooks 或权限规则的候选。

03

故意 /rewind 一次

随便挑一个节点倒回代码再重做,确认这条退路真的存在。敢试的底气,来自亲手验证过的退路。

参考.

01 Claude Code Docs · Permission modes —— 计划模式:让 Claude 调研、提方案,但不动手改源码;读文件、跑只读探索命令、写出计划,批准前编辑一律被挡。Shift+Tab 或 /plan 进入,Ctrl+G 在编辑器里直接改计划;批准时可选「手动逐个批」或切自动模式。模式表:default(只读免批)、acceptEdits(文件编辑与常见文件系统命令免批)、plan(只读)、auto(分类器盯着放行一切、拦越界)、dontAsk(只跑预批准项)、bypassPermissions(全放行,仅限隔离环境)。截至 2026-07-17。Claude Code Docs · Permission modes

02 Claude Code Docs · Best practices —— 推荐工作流:探索、计划、实现、提交四步;计划在方案不确定、改动跨多文件、代码不熟时最值,「如果一句话能说清那个 diff,跳过计划」。跑偏时尽早纠:Esc 打断、同一问题纠正两次仍不对就 /clear 带着教训重开。截至 2026-07-17。Claude Code Docs · Best practices

03 Claude Code Docs · Checkpointing —— 每条用户 prompt 自动创建一个 checkpoint,会话保留最近 100 个,checkpoint 随会话保存、续开后仍可倒回。Esc Esc 或 /rewind 打开菜单:恢复代码与对话、只恢复对话、只恢复代码、从此处起摘要、摘要至此处。界限写得明白:只追踪 Claude 文件编辑工具的改动,Bash 命令与外部修改不在内,「补充而非替代版本控制」。截至 2026-07-17。Claude Code Docs · Checkpointing

04 Claude Code Docs · Model configuration —— opusplan 模型别名:计划模式用 Opus,退出计划、进入执行切到 Sonnet。截至 2026-07-17。Claude Code Docs · Model configuration

---

先批方案,再收结果,
中间的键盘留给它.

记忆 · CLAUDE.md 是项目的长期记忆.

同一件事你跟它说了十遍：用这个包管理器、按这个风格、别碰那个目录、跑测试用这条命令。模型不跨会话记事 —— 但 CLAUDE.md 会。它是每次开工自动加载的项目长期记忆，分项目级、用户级、企业级三层，还能用 @import 把别的文件拼进来。这一章讲什么该钉进去、什么不该（它不是文档垃圾场），三层怎么分工，以及为什么一份写得好的 CLAUDE.md，比你换一个更强的模型更能改善它干活的质量。

用这个包管理器，不是那个。测试要跑 npm test 不是 npx jest。migrations 目录别碰。—— 这些话你跟它说过十遍，因为模型不跨会话记事，每个新会话都是一张白纸。CLAUDE.md 就是治这个的：一份每次开工自动进入它脑子的文件。这一章讲什么该钉进去、钉在哪一层，以及一条容易被高估的边界：记忆是请求，不是规矩。

— I

放哪：四层记忆，各管一摊.

CLAUDE.md 是普通的 markdown，特殊在于加载时机：每个会话开始，Claude 自动读它 —— 不用你贴、不用你提。[1] 起点不用手写：跑一次 `/init`，它会分析你的代码库，把探出来的构建命令、测试方式、项目惯例生成一份初稿；已经有 CLAUDE.md 的话，它提改进而不是覆盖。[1] 之后的问题只剩一个：哪句话放哪一层。

层级	位置	放什么
组织	managed policy(IT 下发)	全公司的编码与合规规范
用户	`~/.claude/CLAUDE.md`	你个人的偏好,跟你走所有项目
项目	`./CLAUDE.md`	团队共享:命令、架构、惯例,进版本库
项目 · 个人	`CLAUDE.local.md`	你的沙盒地址、测试数据,进 .gitignore

四层叠加生效,不互相覆盖。[1] 文件长了还有两个泄压阀:`@path` 语法把别的文件拼进来(比如 `@docs/git-instructions.md`,最多递归 4 层);`.claude/rules/` 目录把规则拆成多个主题文件,还能用 `paths` 限定 —— 只在 Claude 碰到匹配文件时才加载,不白占上下文。[1]

· · ·

— II

写什么:一行一个否决问题.

CLAUDE.md 的敌人不是太短,是太长:它整份进入每次会话的上下文,臃肿的文件会让真正要紧的规则被淹没 —— 官方直说,写得太满,Claude 反而开始忽略你的指令。[2] 纪律是对每一行问同一个问题:**删掉这行,它会不会因此犯错?**不会,就删。[2]

值得一行	不值得
它猜不到的构建 / 测试命令	读代码就能推断的结构
和默认不同的风格规则	语言的通行惯例
禁区目录、仓库礼仪、环境怪癖	经常变动的信息
踩过的坑和非直觉行为	逐文件的代码库导览

什么时候写?用第 1 章那张触发表里的第一条:**同一个错误出现第二次,就是写一行的时候**。[3] 什么时候搬走?一段记忆长成了多步流程,它就不该住在这里 —— 那是 skill 的形状(第 4 章);只对某个目录有效的规则,搬进带 `paths` 的 rules 文件。[1] 200 行是个好的警戒线。[1]

· · ·

– III

记忆是请求,不是规矩.

把预期校准准确:文档对 CLAUDE.md 的定性是「上下文,不是强制配置」—— 它读了、大概率照做,但没有任何机制保证。[1] 所以「提交前必须跑 lint」写在这里是叮嘱,写成 hook 才是闸门(第 7 章);「绝不许碰 .env」想要硬拦,用 PreToolUse hook,不是加粗字体。分层记忆负责让它**懂你的项目**,hooks 负责让它**越不过线** —— 两件事,别用一个工具硬扛。还有两个容易踩的时间差。其一:CLAUDE.md 在会话开始读入内存,**中途改了不生效** —— 新内容要等下次 `/clear`、`/compact` 或重启才加载。[4] 改完发现它还按旧规矩走,不是它叛逆,是你还没翻页。其二:除了你写的记忆,它也在自己记 —— auto memory 默认开启,构建命令、调试心得这类「下次会用上」的东西被它写进 `~/.claude/projects/` 下的项目记忆目录,每次会话带上索引。[1] 用 `/memory` 能看到两套记忆的全部文件:定期翻一翻,删掉过时的 —— 记忆也会腐烂,烂记忆比没记忆更误事。

> — 实践提示
>
> 维护也可以派出去:直接说「把这条加进 CLAUDE.md」,它会替你写;说「记住这个」,它会存进 auto memory。你要做的只是当编辑,不是当录入员。

动手 · 给你的项目铺第一层记忆:

- [ ] 跑一次 /init;已有 CLAUDE.md 就让它提改进,逐条批。
- [ ] 回想最近两周你重复交代过两次以上的话,各压成一行写进去。
- [ ] 用「删掉会不会出错」的尺子删掉三行 —— 减法和加法一样重要。

参考.

01 Claude Code Docs · Memory —— CLAUDE.md 每次会话开始加载;位置分四级:组织(managed policy,IT 统一下发)、用户(~/.claude/CLAUDE.md,你的全部项目)、项目(./CLAUDE.md 或 ./.claude/CLAUDE.md,随版本库共享)、项目个人(CLAUDE.local.md,加进 .gitignore);@path 导入其他文件,最多递归 4 层;.claude/rules/ 可按路径限定加载;建议单文件 200 行以内,过长反而降低遵循度;/init 自动生成起点;明确一句「context, not enforced configuration」—— 要硬拦用 PreToolUse hook。auto memory:Claude 自己跨会话记笔记,存于 ~/.claude/projects/<项目>/memory/,MEMORY.md 索引每次加载前 200 行或 25KB,/memory 可查看、编辑、关闭。截至 2026-07-17。Claude Code Docs · Memory

02 Claude Code Docs · Best practices —— 写 CLAUDE.md 的取舍表:收录它猜不到的构建命令、和默认不同的风格规则、测试跑法、仓库礼仪、项目特有的架构决定与坑;排除读代码就能推断的、语言通识、频繁变动的信息、逐文件的代码库描述。每一行问一句「删掉它会不会导致出错」,不会就删;臃肿的 CLAUDE.md 会让真正的规则被忽略。截至 2026-07-17。Claude Code Docs · Best practices

03 Claude Code Docs · Extend Claude Code —— 各扩展机制的触发时机:同一个约定或命令错第二次,写进 CLAUDE.md;反复粘贴的多步流程,做成 skill;必须每次发生的,写 hook。截至 2026-07-17。Claude Code Docs · Extend Claude Code

04 Claude Code Docs · How Claude Code uses prompt caching —— 项目根与用户级 CLAUDE.md 在会话开始读入内存,中途编辑不废缓存、但也不生效,下次 /clear、/compact 或重启才加载新内容。截至 2026-07-17。Claude Code Docs · Prompt caching

---

同一句话说第三遍之前,
把它变成一行记忆.

收编 · Skills 与输出风格.

你有一套反复输入的提示：同样的审查清单、同样的发布步骤、同样的「用这个口吻写」。每次重打一遍，既慢又不稳。Claude Code 给了两个收编的去处：重复的流程连脚本带资源打包成 Skill，一个 /名字 唤起、用到才加载，不挤占上下文 —— 从前的自定义 /命令 也已并入这里；要改的若是它整场对话的角色和口吻，那是输出风格的事，在设置里切换。这一章讲两者怎么分工：哪些活沉成 skill、哪些交给风格、哪些根本不值得固化。

你有几段每周都要重打的提示:发版前那套检查、审 PR 的固定清单、「用我们团队的口吻写 changelog」。每次凭记忆重打,慢,而且每次都略有不同 —— 于是结果也略有不同。这一章讲收编:把你的重复劳动变成两类资产,流程沉成 Skill,说话方式定成输出风格。收编过的东西不再依赖你的记性,也不再吃你的手速。

— I

Skills:把流程存成一个词.

一个 skill 就是一个目录加一份 `SKILL.md`:frontmatter 里一句 description 说清「什么时候该用我」,正文写做法,旁边还能放脚本、模板、示例文件。存好之后有两条入口:你打 /名字 直接唤起,或者 Claude 自己判断相关就加载。[1] 关键的经济学在加载方式:描述常驻、**正文用到才进上下文** —— 一份几百行的发版手册,平时几乎不占地方。[1] 这也是它和 CLAUDE.md 的分工:每次都要的一句事实住第 3 章,偶尔才用的一套流程住这里。如果你记得「自定义 slash 命令」:它们已经并入 skills —— `.claude/commands/deploy.md` 和 `.claude/skills/deploy/SKILL.md` 生成同一个 `/deploy`,旧文件继续可用,skills 只是多了支撑文件、调用控制这些新能力。[1] 一份真实形状的 skill 长这样:

.claude/skills/release/SKILL.md

```
---
description: 发布一个新版本。用户说「发版」「release」时使用。
disable-model-invocation: true
argument-hint: "[patch|minor|major]"
---

## 当前状态

!`git status --short && git log --oneline -5`

## 步骤

发布一个 $ARGUMENTS 版本:
1. 跑 npm test,全绿才继续
2. 按 $ARGUMENTS 升版本号,更新 CHANGELOG
3. 提交、打 tag、推送,贴出每一步的输出
```

一个发版 SKILL:动态注入 + 参数 + 只许手动触发

三个 frontmatter 字段值得先认识。`disable-model-invocation: true`:只许你手动触发 —— 发版、发消息这类有副作用的流程,别让它「觉得代码差不多了」就自己跑。[1] `allowed-tools`:这一轮预批准的工具,让 `/release` 跑 git 命令不用逐条弹窗。[1] `context: fork`:整个 skill 丢进 subagent 的独立上下文里执行,重活不弄脏主对话(第 5 章的机制)。[1] 正文里那行 `!`git status`` 是动态注入:调用瞬间先执行命令、把输出内联进内容 —— 你的 skill 拿到的是活数据,不是你写死的假设。[1]

• • •

— II

输出风格:换的不是知识,是口吻.

第二根柱子管的是另一类重复:「解释详细点」「别铺垫直接干」「像导师一样带我」—— 这些不是流程,是**整场对话的角色设定**。输出风格直接改 system prompt 里的角色、口吻与输出格式,一次设

定,每一轮生效。[2] 文档的界定句很准:改变它怎么回应,不改变它知道什么。[2] 内置四款够覆盖多数场景:Default 是原生工程师;Proactive 更敢直接动手、少停下来问;Explanatory 边干活边给你讲原理;Learning 最有意思 —— 它会在代码里留 TODO(human),把关键的一小段留给你亲手写,拿它学新代码库比纯看快得多。[2] 切换走 /config 里的 Output style 选项,或直接在设置文件里写 outputStyle(独立的 /output-style 命令已经移除)。[2] 自定义也简单:markdown 文件放进 .claude/output-styles/,用 keep-coding-instructions 决定保不保留原生编码指令 —— 保留,它是个换了口吻的工程师;去掉,它可以彻底变成写作助手或数据分析员。[2] 两条边界记住:风格只作用于主对话,派出去的 subagent 不受影响;会话中途换风格不生效,下个会话才加载。[2]

• • •

— III

什么收、收到哪、什么不收.

到这里,固化的去处一共四个。判断题每次只有一道:这段重复,是流程、是事实、是口吻,还是铁律?

这段重复是	收到	性质
多步流程、带脚本的做法	Skill(本章)	按需加载,模型执行
每次都要的一句事实	CLAUDE.md(第 3 章)	常驻上下文
整场对话的角色与口吻	输出风格(本章)	改 system prompt
必须每次都发生的动作	Hook(第 7 章)	harness 执行,不经模型

头两行的分界线,官方给的触发律是「第三次」:同一段提示重打到第三次、同一套做法第三次贴进对话,就存成 skill。[3] 末行的分界线更硬:skill 是模型理解后执行的,结果可能有出入;hook 在事件

上必然执行 —— 「希望它跑 lint」是 skill 的事,「必须跑过 lint 才许提交」是 hook 的事。[3] 反过来,三类东西不值得收:只做一次的活,写清楚 prompt 就够;还在每周变的流程,固化了就是提前腐烂;以及模型本来就会做对的事 —— 给它立规矩,和给它写说明书一样,都有维护成本。动手 · 本周就收编三样:

01

把重打过两次的 prompt 存成第一个 skill

找出你本周重打过的那段提示,原样放进 .claude/skills/<名字>/SKILL.md 的正文,description 写清触发时机。下次用 /名字 唤起。

02

给一个有副作用的流程上锁

发版、发通知、清数据 —— 挑一个,加 disable-model-invocation: true,让触发权只在你手里。

03

试一周 Learning 或 Explanatory 风格

在你最近接手的陌生代码库里开一周,再决定回不回 Default —— 风格是可逆的,试错成本几乎为零。

参考.

01 Claude Code Docs · Skills —— 一个 skill 是一个目录加一份 SKILL.md:frontmatter 写清何时该用,正文是做法;描述常驻上下文,正文只在被调用或被判定相关时加载;/名字 直接唤起。「自定义命令已并入 skills」—— .claude/commands/deploy.md 与 .claude/skills/deploy/SKILL.md 同样生成 /deploy,旧命令文件继续可用。常用 frontmatter:disable-model-invocation(只许你手动触发)、user-invocable: false(只许它自己用)、allowed-tools(该轮免批工具)、context: fork(丢进 subagent 跑)。正文里还能用 $ARGUMENTS 参数替换、!`命令` 动态注入(执行后把输出内联进内容)。位置:企业 / 个人 ~/.claude/skills/ / 项目 .claude/skills/ / 插件;SKILL.md 建议 500 行内,细料放支撑文件。截至 2026-07-17。Claude Code Docs · Skills

02 Claude Code Docs · Output styles —— 「输出风格改变 Claude 怎么回应,不改变它知道什么」:修改 system prompt 里的角色、口吻与输出格式,作用于整场对话。内置四款:Default、Proactive(先动手少停顿)、Explanatory(边做边讲解)、Learning(留 TODO(human) 让你亲手写一段);自定义放 .claude/output-styles/,keep-coding-instructions 决定保不保留原生编码指令。经 /config 选择或直接改 outputStyle 设置(独立的 /output-style 命令已移除);只作用于主对话,subagent 不受影响。截至 2026-07-17。Claude Code Docs · Output styles

03 Claude Code Docs · Extend Claude Code —— 收编的触发时机:同一段提示重打到第三次、或同一套多步流程第三次贴进对话,存成 skill;hook 与 skill 的分界:hook 在事件上必然执行,skill 由模型理解并执行、结果可能有出入。截至 2026-07-17。Claude Code Docs · Extend Claude Code

---

重打第三遍的提示,
是一个还没存档的 skill.

派活 · Subagents、并行与后台.

一个上下文装不下一件大事：读遍二十个文件找一处调用、同时审五个维度、跑一个 migration。Claude Code 让你把活派给 subagent —— 独立上下文、干完只把结论交回来；可以一次并行铺开许多个，也可以丢到后台让它自己跑、跑完叫你。这是「编排台」这个词的实体：你不再自己钻进每个文件，而是拆活、派活、收活。这一章讲什么活值得派、怎么并行、后台任务什么时候用，以及自定义 agent 类型怎么让每支队伍各司其职。

一个上下文装不下一件大事。读二十个文件找一处调用，找到时上下文已经塞满了搜索残渣；同时审安全、性能、风格三个维度，审到第三个，第一个的标准已经被挤得模糊。到目前为止，这本书里干活的还是「一个 Claude」—— 这一章把它变成复数。subagent 是编排台这个词的实体：你不再钻进每个文件，你拆活、派活、收活。

— I

为什么派：上下文是你最贵的资源.

subagent 的定义一句话：在**自己的上下文窗口**里跑的一个 Claude，带自己的系统提示和工具权限，干完只把结论交回主对话。[1] 它解决的问题在第 2 章末尾就埋下了：主对话的上下文是你最贵的资源，而调研恰恰是最脏的活 —— 官方的建议直白：用 subagents 做调研，探索的几十次文件读取都发生在别处，你的主对话只收到三段结论。[2] 其实你已经见过它干活：Claude 平时自己就会把搜索派给内置的 Explore（只读快查），计划模式的调研派给 Plan。[1] 这一章教你的，是主动派、成建制地派。什么活值得派？三个信号：过程你不需要看，只要结论（调研、找调用点）；量大而平行（一百个文件各查一遍）；或者你想要一个不带偏见的新脑子（第 9 章的复审）。反例也清楚：三轮以内你要来回指挥的活，派出去的沟通成本比省下的上下文还贵。

· · ·

— II

建队:一个 markdown 文件一个兵种.

反复派同一类活,就该把那类工人定义下来。自定义 subagent 是一个带 frontmatter 的 markdown 文件,项目级放 `.claude/agents/`(进版本库,全队共用),个人级放 `~/.claude/agents/`。[1] 必填只有两项:name,和决定命运的 **description** —— Claude 靠它判断什么任务该派给谁,写得越清楚,派工越准。[1]

`.claude/agents/dep-auditor.md`

```
---
name: dep-auditor
description: 审计依赖升级的破坏性变更。升级依赖、审 lockfile 变更时使用。
tools: Read, Grep, Glob, Bash
model: haiku
memory: project
---

你是依赖审计员。对每个被升级的包:查它的 changelog 与 breaking
changes,对照本仓库的实际用法,只报会影响我们的条目,给出
文件:行 与迁移建议。查不到 changelog 的包要明说,不要猜。
```

一个依赖审计员:收权、降档、带记忆

frontmatter 的三个旋钮对应三种控制。`tools` 收权:审计员只该读和查,不该写 —— 少给的每一样工具都是少一分风险。`model` 降档:机械的批量活派给 haiku,便宜且快,难题才留给主对话的大模型(选型逻辑在第 10 章)。`memory` 让它跨会话攒经验:这个审计员下个月还记得上次哪个包坑过你。[1] 还有一个后面会用到的:`isolation: worktree` 让它在临时 git worktree 里干活 —— 这是并行改文件不打架的钥匙。[1] 文件不用手写:描述清楚你要的兵种,让 Claude 替你写(创建向导已移除,文件就是接口)。[1]

– III

铺开与后台:活不必排队,也不必等你.

并行不需要仪式,一句话就够:「分别用 subagent 并行调研认证、数据库和 API 三个模块」—— 三个独立上下文同时开工,三份结论先后落回你的对话。[2] 而且默认就是**后台**跑的:子任务转它的,你和主对话继续聊你的,只有 Claude 立刻要用结果时才前台等;一个正跑的任务按 Ctrl+B 也能随手丢去后台,/tasks 看全部进度。[1] 两个让它可控的细节:后台 subagent 要权限时,弹窗仍回你的主会话,该批的还是你批;派出去的活干完不是终点 —— 用名字就能把它叫回来续跑,之前的全部上下文都在。[1] 嵌套也允许:subagent 自己还能再派 subagent(比如一个审查员给每条发现派一个核实员),最深五层。[1] 并行的硬边界在写文件:两个 agent 同时改同一份检出,就是打架。出路按重量排:只读的并行随便铺;要改文件的,给 subagent 加 `isolation: worktree`,各自在临时 worktree 里改自己的副本[1];要更彻底的隔离,开多个会话配 git worktrees —— 官方对多会话的第一条建议,就是一个会话占一个独立检出,互不碰撞。[2]

提示词

派活单模板:范围 + 交回物 + 验收

用 `<subagent 名字或 use a subagent>` 做:`<一句话任务>`。
范围:只看 `<目录 / 文件>`,别展开到范围外。
交回:`<结论的形状 —— 几条发现 / 一张表 / 一份 diff>`,不要过程。
验收标准:`<能检验的标准>`(交回时对着它自查一遍)。

用 dep-auditor 做:审计这次 lockfile 变更里所有主版本升级。
范围:只看 package.json、pnpm-lock.yaml 和 src/ 里的实际用法。
交回:每个包一条 —— 破坏性变更、影响到的 文件:行、迁移建议;没影响的包归成一行列表。
验收标准:每条结论都有出处链接或代码位置(交回时对着它自查一遍)。

— IV

队伍的上限:什么时候 subagent 不够用.

subagent 的模型是「工人向工头汇报」:它们不互相说话,你的主对话是唯一的汇合点。要的就是这个的话 —— 绝大多数派活场景都是 —— 它是成本最低的形态。当任务真的需要多个工人**互相讨论**(竞争假设的调研、各占一块的大型开发),那是 agent teams 的地界:多个独立会话共享任务列表、互发消息,token 成本也按整会话计 —— 目前是实验特性,默认关闭。[3] 先用足 subagent,再考虑编队;多数「需要 team」的直觉,其实是「派活单没写清」。动手 · 这周把三类派活各试一次:

- [ ] 下次想「查一下 X 是怎么实现的」,改成让 subagent 去查,只收结论 —— 对比一下你主对话省下的篇幅。
- [ ] 按第 II 节的样子,给你最常重复的那类活建第一个自定义 subagent,进 .claude/agents/ 提交给团队。
- [ ] 把一个要跑十分钟的活派出去后按 Ctrl+B,继续手头的事,用 /tasks 看它跑完 —— 体会「活在转,你没停」。

参考.

01 Claude Code Docs · Subagents —— 每个 subagent 在自己的上下文窗口里运行:独立系统提示、独立工具权限,干完只把结果交回主对话。内置 Explore(只读快查)、Plan(计划模式的调研)、general-purpose(全能)。自建:.claude/agents/*.md(项目)或 ~/.claude/agents/(个人),frontmatter 必填 name 与 description(description 决定它何时被派),可配 tools、model(sonnet/opus/haiku/fable/inherit)、memory(跨会话记忆)、isolation: worktree(临时 git worktree 里跑)、maxTurns 等;/agents 创建向导已移除,直接让 Claude 写文件。运行:默认在后台跑,要结果才前台等;Ctrl+B 手动丢后台;权限弹窗仍回主会话;/tasks 看清单;subagent 可再派 subagent,最深 5 层;完成的 subagent 可用 SendMessage 按名字续跑,历史全在。截至 2026-07-17。Claude Code Docs · Subagents

02 Claude Code Docs · Best practices —— 上下文是根本约束,「用 subagents 做调研」:探索在独立上下文里发生,主对话只收结论;并行的几条路:git worktrees 里开多个 CLI 会话互不打架、桌面端多会话、云端会话;fan out:循环调用 claude -p 批量分发。截至 2026-07-17。Claude Code Docs · Best practices

03 Claude Code Docs · Extend Claude Code —— subagent 与 agent team 的分界:subagent 在你的会话内干活、只向主 agent 汇报,适合「只要结论」的聚焦任务;agent teams 是多个互相通信、共享任务列表的独立会话,token 成本更高,适合要互相讨论的复杂协作 —— 目前为实验特性,默认关闭。截至 2026-07-17。Claude Code Docs · Extend Claude Code

---

你不必钻进每个文件 ——
拆活,派活,收活.

接线 · MCP 把你的世界接进来.

你的上下文大半不在代码里 —— 在 GitHub 的 issue、Linear 的工单、数据库的表、Sentry 的报错、Notion 的文档里。MCP 让 Claude Code 直接读它们、调它们，不用你来回复制粘贴。这是它从「改这个项目」扩到「接管你整套工作」的接口。这一章讲 MCP 怎么配、哪些服务器值得接、本地 vs 远程的取舍，以及一条比「能不能接」更重要的线：每接一个外部工具，你就把它能读到、能动到的范围扩大了一圈 —— 什么时候「能接」不等于「该接」。

你的上下文大半不在代码里。需求在 Jira 的工单里,报错在 Sentry 里,数据在 PostgreSQL 里,设计稿在 Figma 里,讨论在 Slack 里 —— 于是你的日常变成了搬运:开网页,复制,粘进对话,再解释一遍这是什么。MCP 把搬运换成接线:让 Claude Code 直接读它们、动它们。这一章讲怎么接、接在哪一层,以及一条比「能不能接」更要紧的线 —— 每接一个,它的读写半径就大一圈。

— I

接线:一条命令,两种形态.

MCP(Model Context Protocol)是接外部工具的开放标准,几百个现成服务器可用。接上之后的样子,文档自己举的例子最直观:「实现 JIRA 工单 ENG-4521 描述的功能并在 GitHub 开 PR」「按 Slack 里新发的 Figma 设计稿改邮件模板」「从我们的 PostgreSQL 里找 10 个用过这个功能的用户」。[1] 接入只是一条命令,分两种形态:远程服务用 `--transport http`(或 sse),本地进程用 stdio、-- 后面跟启动命令:

```
# 远程服务(HTTP)
claude mcp add --transport http notion https://mcp.notion.com/mcp

# 本地进程(stdio):-- 之后是服务器自己的启动命令
claude mcp add --env AIRTABLE_API_KEY=YOUR_KEY --transport stdio airtable \
  -- npx -y airtable-mcp-server

# 看连接状态;需要 OAuth 的远程服务器也在这里登录
/mcp
```

三条接线:远程服务、带密钥的本地进程、查看与登录

担心几十个服务器把上下文挤爆?这一点默认已被解决:tool search 只把工具名先注册进来,完整定义等用到那一刻才加载 —— 闲置的接线几乎不占地方。[1] 顺带一条省心的对照:如果那个系统有好用的 CLI(比如 GitHub 的 gh),装上让它直接用,常常比接 MCP 更省 —— CLI 是最上下文经济的外部接口,不熟的它还能靠 `--help` 现学。[2]

· · ·

— II

放哪一层:私用、随仓库、跟着你.

和 CLAUDE.md、subagent 一样,MCP 配置也分层,答案取决于「这条线属于谁」。[1]

作用域	存在哪	适合
local(默认)	~/.claude.json,只属于你 + 这个项目	实验性接线、带私人凭据的服务器
project	.mcp.json,进版本库	全队共用的工单、数据库接线
user	你的所有项目	你走到哪用到哪的个人工具

project 层是团队分发的正道:一份 `.mcp.json` 提交进仓库,全队 clone 下来就有同一套接线。配套的安全阀值得注意:来自 `.mcp.json` 的服务器**首次使用前要你亲自批准** —— 仓库不能替你决定接什么;批错了,`claude mcp reset-project-choices` 重来。[1]

• • •

– III

能接,不等于该接.

现在说那条更要紧的线。每接一个服务器,你都在扩大两样东西:它**能读到**的范围(这些数据从此可能进入对话与上下文),和它**能动到**的范围(工具是能执行动作的)。文档把丑话说在前面:先确认你信任每个服务器再连接;而会抓取外部内容的服务器,还可能带来 prompt injection 风险 —— 它替你读回来的网页或工单里,可能藏着写给模型看的指令。[1] 接线前过三问,比接完再后悔便宜:这条数据真需要直读吗 —— 一次性的贴进来更快,反复要的才值得接;只读够不够 —— 能查库和能改库是两个半径,从只读开始;写操作有没有闸 —— 危险动作配 ask 级权限规则或第 7 章的 PreToolUse hook,让「能写」不等于「随时会写」。

– 注意

一条底线判断:**别接你不敢让它写的系统**,除非你已经把写路径锁死。生产数据库若非接不可,接只读副本;真要写权限的场景,把那几个写动作单独设成必须弹窗。信任是按服务器给的,不是按协议给的。

• • •

– IV

什么时候接:第三次复制的时候.

回到触发律,和 skill 的「第三次重打」同构:当你第三次从同一个系统复制数据贴进对话 —— 工单、报错、面板读数 —— 那个系统就该接进来了。[3] 一次性的资料,贴进来仍然最快;每周都在搬的,才值得一条线。这也是「编排台」的接线逻辑:第 5 章给了你人手,这一章给他们素材 —— 派出去的 subagent 能自己去读工单、查报错,你连转述都省了。动手 · 接第一条线,按信任梯度来:

01 **挑你每周复制最多的那个系统**

工单、报错监控或数据库,选一个。有顺手 CLI 的先考虑 CLI;没有,claude mcp add 接上,/mcp 确认连通。

02 **只读用一周,再谈写**

让它查、让它读,观察它把数据用得对不对;一周后再决定要不要放开写动作,放开的每一个都配上弹窗。

03 **值得全队的,升到 project 层**

把这条线用 --scope project 写进 .mcp.json 提交 —— 每个队友 clone 即得,首次使用时各自批准。

参考.

01 Claude Code Docs · MCP —— MCP 是开放标准,把外部工具与数据源接给 Claude Code: 实现 Jira issue 并开 PR、查 PostgreSQL、按 Slack 里的 Figma 稿改模板、起草 Gmail 邀请等都是文档自己举的例子。接入: claude mcp add,--transport http / sse(远程服务)或 stdio(本地进程,-- 后接启动命令);/mcp 面板看连接状态、给需要 OAuth 2.0 的远程服务器登录。作用域: local(默认,存 ~/.claude.json,只属于你和当前项目)、project(写 .mcp.json 进版本库,团队共享,首次使用前要你批准,claude mcp reset-project-choices 可重置)、user(你的所有项目)。tool search 默认开启: 工具名先注册、完整定义用到才加载,几十个服务器也不挤占上下文。安全原话: 「先确认你信任每个服务器再连接」,会抓取外部内容的服务器可能带来 prompt injection 风险。截至 2026-07-17。Claude Code Docs · MCP

02 Claude Code Docs · Best practices —— 与外部服务打交道时,CLI 是最省上下文的方式: 装了 gh,Claude 就会用它建 issue、开 PR、读评论;不熟的 CLI 也能靠 --help 现学。截至 2026-07-17。Claude Code Docs · Best practices

03 Claude Code Docs · Extend Claude Code —— 接 MCP 的触发时机: 当你反复从另一个系统的页面把数据复制进对话(工单、监控面板),就该把那个系统接成 MCP 服务器。截至 2026-07-17。Claude Code Docs · Extend Claude Code

---

上下文不该靠你搬运,
该自己长进来.

自动化 · Hooks 让它按规矩来.

有些事你不想靠「希望模型记得」：每次改完自动跑 formatter、提交前必须过 lint、绝不允许它碰生产库、每次收尾发个通知。这些不能是建议，得是规矩。Hooks 就是规矩：它们是你在特定时机（工具调用前后、会话结束时）自动执行的脚本 —— 由 harness 执行，不由模型自觉，所以它跳不过。这一章讲 hooks 能钉住哪些行为、几个真正值得设的（安全闸、质量门、通知），以及它和 CLAUDE.md「叮嘱」的根本区别：一个是请求，一个是硬约束。

有些事你不想靠「希望模型记得」：每次改完文件跑格式化，提交前必须过 lint，生产配置绝对不许碰，收工时给你发条通知。写进 CLAUDE.md，它十次里九次照做 —— 而这四件事你要的是十次里十次。差的那一次，不能靠更大的字体解决。这一章讲 hooks：不求模型自觉、由 harness 到点必然执行的规矩。

— I

请求与规矩的分界线.

这本书已经两次踩到这条线：第 3 章说记忆是「上下文，不是强制配置」[3]，第 4 章说 skill 由模型理解执行、结果可能有出入。hooks 站在线的另一侧 —— 定义原话：「用户定义的 shell 命令、HTTP 端点或 LLM 提示，在 Claude Code 生命周期的特定时点**自动执行**」。[2] 关键词是自动：hook 由 harness 在事件上触发，不经过模型的判断，它想跳也跳不过。官方把分工说得很干脆：必须每次发生、零例外的动作，用 hooks；CLAUDE.md 是建议，hooks 是确定。[4]

— II

装第一条:通知 + 格式化.

hooks 住在 settings 文件里(~/.claude/settings.json 归你,.claude/settings.json 随仓库),结构三层:事件名 → matcher 筛工具 → 要跑的命令。下面这份配置装了两条最常用的:改完文件自动 prettier,Claude 等你输入时发桌面通知 —— 后者治好的病叫「盯终端」:

.claude/settings.json

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "jq -r '.tool_input.file_path' | xargs npx prettier --write"
          }
        ]
      }
    ],
    "Notification": [
      {
        "matcher": "",
        "hooks": [
          {
            "type": "command",
            "command": "osascript -e 'display notification \"Claude 在等你\" with
   title \"Claude Code\"'"
          }
        ]
      }
    ]
  }
}
```

两条入门 HOOK:POSTTOOLUSE 格式化、NOTIFICATION 桌面提醒(MACOS)

读法:PostToolUse 是「工具跑完之后」这个时点;matcher Edit|Write 表示只在编辑或写文件后触发;命令从 stdin 收到这次事件的 JSON,用 jq 取出文件路径喂给 prettier。[1] 不想手写 JSON

也行,这活可以派:直接说「写一个每次编辑后跑 eslint 的 hook」,让 Claude 自己给自己立规矩。[4] 装好后用 /hooks 面板核对 —— 它是只读的浏览器,改动仍走 settings 文件。[1]

• • •

— III

时点地图与否决权.

生命周期上有约三十个可挂的事件,不用全记 —— 先认四个:PreToolUse,动手之前,唯一能**拦下**动作的时点;PostToolUse,动完之后,放质量动作;Stop,它想收工时,放验收闸;SessionEnd,会话结束,放通知和清理。[2] hook 和 harness 的对话协议是退出码:exit 0 放行;**exit 2 否决** —— 动作被拦下,你写进 stderr 的理由会交还给 Claude,它读了会调整做法。[1] 要更细的裁决(允许、询问、改写输入),exit 0 加一段 stdout JSON。[1] PreToolUse 有一个值得单独记住的性质:它跑在权限模式检查**之前** —— 一条返回 deny 的 hook,连 bypassPermissions 模式也拦得住。[1] 这意味着 hooks 是比权限模式更硬的一层:模式管的是「问不问你」,hook 管的是「许不许做」。第 2 章那句「信不过的点用规矩钉死」,钉子就是它。

• • •

— IV

真值得设的三类:安全闸、质量门、通知.

别一口气挂三十个事件。多数人长期真正在用的是三类:

类型	挂在	典型规矩
安全闸	PreToolUse	碰 .env、migrations、生产配置 → exit 2 拦下
质量门	PostToolUse / Stop	改完跑 lint;检查不过不许收工
通知	Notification / SessionEnd	等输入、收工时提醒你,人离开终端

安全闸的写法在文档里就有现成样:一个脚本核对目标路径,命中保护清单就 exit 2 并在 stderr 里说明为什么 —— Claude 收到理由,换路走。[1] 质量门的顶配是 Stop hook,第 9 章已经预告过:收工前跑你的检查,不过不放行;也记得它的边界 —— 连续拦 8 次后 harness 会放行,它是门,不是牢,兜底仍在测试和 CI。[4]

> — 补充
>
> 反向纪律:别把该是判断的事塞进硬规矩。「代码要优雅」做不成 exit 2;需要裁量的检查,用 prompt 型 hook(让一个模型按你的标准评估)或干脆交给第 9 章的 reviewer。hook 擅长的是黑白分明的事 —— 路径、命令、退出码。灰色地带塞进闸门,得到的不是纪律,是误拦。

动手 · 一晚上装齐三类:

- [ ] 装上面的通知 hook,从此让终端叫你,而不是你守终端。
- [ ] 把你仓库里最不可碰的路径(.env、migrations)写成 PreToolUse 安全闸,亲手触发一次确认它真拦。
- [ ] 把第 9 章那条验收检查升级成 Stop hook —— 从「你记得看」变成「不过不收工」。

参考.

01 Claude Code Docs · Hooks guide —— hooks 提供确定性控制,「保证某些动作必然发生,而不是依赖 LLM 选择去做」。配置放 settings 文件的 hooks 块:事件名 → matcher(如 Edit|Write)→ type: command 的命令;/hooks 面板只读浏览,增改要编辑 settings 或让 Claude 代写。协议:事件数据以 JSON 进 stdin;exit 0 放行,exit 2 拦下且 stderr 作为反馈交还给 Claude 让它调整;更复杂的决定用 exit 0 + stdout JSON(如 PreToolUse 的 permissionDecision: deny)。PreToolUse 在权限模式检查之前执行 —— 返回 deny 的 hook 连 bypassPermissions 也拦得住。示例即文档所载:PostToolUse 用 jq 取 file_path 交给 prettier;PreToolUse 脚本对 .env、.git/ 等保护路径 exit 2。截至 2026-07-17。Claude Code Docs · Hooks guide

02 Claude Code Docs · Hooks reference —— 定义原话:「hooks 是用户定义的 shell 命令、HTTP 端点或 LLM 提示,在 Claude Code 生命周期的特定时点自动执行」;事件约三十个,含 PreToolUse、PostToolUse、Stop、SessionStart、SessionEnd、Notification、UserPromptSubmit、SubagentStop 等;能力包括拦下工具调用、改写工具输入输出、注入上下文、发通知。截至 2026-07-17。Claude Code Docs · Hooks reference

03 Claude Code Docs · Memory —— CLAUDE.md 与 auto memory 是「上下文,不是强制配置」;要不论模型怎么决定都拦住某个动作,用 PreToolUse hook。截至 2026-07-17。Claude Code Docs · Memory

04 Claude Code Docs · Best practices —— 「hooks 用于必须每次发生、零例外的动作」,CLAUDE.md 的指令是建议性的,hooks 才是确定性的;可以直接让 Claude 替你写 hook;Stop hook 把检查设为收工硬门,连续拦 8 次后放行。截至 2026-07-17。Claude Code Docs · Best practices

---

叮嘱是请求,
hook 是规矩.

无人值守 · Headless、SDK 与 CI.

到这里，Claude Code 还都在你眼前的终端里。但它也能离开终端：headless 模式（claude -p）让它无人值守地跑一次；Agent SDK 让你把它嵌进自己的脚本和服务；GitHub Actions 让它在每个 PR 上自动跑一遍。这是「派活」的极限形态 —— 活不在你这台机器、不在你盯着的时候完成。这一章讲哪类活适合无人值守（批处理、CI 检查、定时任务），怎么用 SDK 把它编排进流水线，以及无人时最该守住的那条：它跑得越自动，你越要在它动手的边界上设死规矩。

到这里,Claude Code 还都在你眼前:终端开着,活跑着,你在场。这一章拆掉最后一个前提。headless 让它无人值守跑一次,SDK 让它长进你的程序,GitHub Actions 让它守在每个 PR 上,Routines 让它半夜也上班 —— 派活的极限形态:活不在你这台机器、不在你醒着的时候完成。也因此,这一章的另一半讲规矩:它跑得越自动,边界越要在出发前写死。

— I

headless:一条命令,进出都是管道.

把交互会话变成一条 Unix 命令,只需要一个 `-p`:`claude -p "提示"` 跑完就退,stdin 能灌数据进去,stdout 能接进下一个程序。[1] 它把 Claude Code 从「一个你打开的应用」变成「一个脚本里的工序」:

```
# 把构建日志灌进去,要一句根因
cat build-error.txt | claude -p "一句话解释这次构建失败的根因"

# 当一个只报错字的 linter 用
git diff main | claude -p "你是错字检查器:每个错字报 文件:行 和问题,别的什么都不说"

# 结构化输出,给脚本吃
claude -p "列出 src/api 里的全部路由" \
  --output-format json \
  --json-schema '{"type":"object","properties":{"routes":{"type":"array","items":
   {"type":"string"}}},"required":["routes"]}' \
  | jq '.structured_output.routes'
```

三种管道姿势:灌日志、当 LINTER、出结构化结果

第三条是无人化的关键零件:`--output-format json` 让结果带上元数据,`--json-schema` 更进一步 —— 按你给的 schema 约束输出,脚本直接消费 `structured_output` 字段,不用解析散文。[1] 第 9 章说的「回归集用 `claude -p` 重跑、拿 JSON 对一对」,零件就是这两个 flag。脚本化调用再加一个 `--bare`:跳过 hooks、skills、MCP、CLAUDE.md 的自动发现,保证同一条命令在每台机器行为一致 —— CI 里尤其该加。[1]

• • •

— II

SDK:同一台引擎,做成库.

脚本再长,也只是把命令串起来;要把它嵌进你自己的服务 —— 一个自动分诊 bug 的机器人、一个替客服读日志的后台 —— 用 Agent SDK。官方的定义一句话:给你的是**驱动 Claude Code 的同一套工具、agent 循环与上下文管理**,以 Python 和 TypeScript 可编程。[2] 包名 `claude-agent-sdk`(Python)和 `@anthropic-ai/claude-agent-sdk`(TypeScript),一个 `query(prompt, options)` 就是一个带全套工具的 agent;其他语言不用等移植,`claude -p --output-format json` 就是它的通用接口。[2] CLI 和 SDK 怎么分工,官方表格说得干脆:交互开发、一次

性任务用 CLI;CI/CD、定制应用、生产自动化用 SDK。[2] 这本书讲到这条线为止 —— 再往里,是《Agent 实战》整本书的地界。

• • •

— III

长在流水线里:每个 PR、每个整点.

第三种形态不属于任何一台机器。装上 GitHub Actions 集成(交互式安装:`/install-github-app`),在任何 PR 或 issue 里 @claude,它就地分析代码、实现功能、修 bug,开出完整的 PR;`anthropics/claude-code-action@v1` 也能不等召唤、按你的 workflow 定时或按事件跑。[3] 记两笔账:CI 里它烧的是 GitHub runner 分钟加 API token —— 认证用的是你放进仓库 secrets 的 API key,按 token 计费。[3] 这套走的是 Console 的账,和 Pro/Max 订阅额度是两码事,自动化越勤越要单独盯。要「每个 PR 无须触发的自动底审」,那是另一条产品线(GitHub Code Review),配好即每 PR 一审。[3] 定时的活则交给 Routines:跑在 Anthropic 托管的基础设施上,你的电脑关了它照跑,CLI 里 `/schedule` 就能建 —— 夜间依赖巡检、晨会前的 PR 摘要,都是它的班次。[5]

• • •

— IV

无人时的规矩:边界先于出发.

交互会话里,权限弹窗是最后的安全网;无人值守时没有这张网 —— 所以边界要前置。两种写法:`--allowedTools` 白名单,精确到 `"Bash(git diff *)"` 这样的规则,名单之外一律不许[1];或者 `--permission-mode dontAsk`,把「没预批就拒绝」定为总则,专为锁死的 CI 而设。[4] 想要

更多自主,auto 模式的分类器也能在无人时压阵 —— 但记住它的失败姿态:交互会话里连续拦截会退回问你,-p 里没有你,它直接中止。[4] 这是无人化的正确性质:**宁可停,不越界**。最后一层是第 7 章的 hooks:headless 照样触发,PreToolUse 安全闸在没有观众的地方一样否决。三层叠起来再出发 —— 活可以离开你,规矩不行。动手 · 把一件活送出你的视线:

01

把一个重复检查做成一行脚本

挑你每周手动做的一个检查(错字、日志异常、路由清单),照第 I 节写成 claude -p 一行,挂进 package.json 或 Makefile。

02

给团队仓库装上 @claude

跑 /install-github-app,然后在一个真实 issue 里 @claude 试一件小活 —— 看它从 issue 到 PR 走完全程。

03

设一个夜班 Routine

用 /schedule 给明早设一个巡检(昨日 CI 失败摘要、依赖告警),明天读它的班报,再决定要不要长期雇佣。

参考.

01 Claude Code Docs · Run Claude Code programmatically —— claude -p(--print)非交互跑一次:读 stdin、可管道进出;--output-format 选 text / json / stream-json,--json-schema 按 JSON Schema 约束结构化输出(结果在 structured_output 字段);--allowedTools 按权限规则语法预批工具;--bare 跳过 hooks / skills / MCP / CLAUDE.md 的自动发现,保证脚本在每台机器行为一致,是脚本化调用的推荐模式;--no-session-persistence 不留会话;文档场景即 CI、构建脚本与管道。截至 2026-07-17。Claude Code Docs · Run programmatically

02 Claude Code Docs · Agent SDK overview —— 「Agent SDK 给你的,是驱动 Claude Code 的同一套工具、agent 循环与上下文管理,以 Python 和 TypeScript 可编程」;包名 claude-agent-sdk(Python)与 @anthropic-ai/claude-agent-sdk(TS,自带 Claude Code 二进制);query(prompt, options) 即起一个 agent;其他语言走 claude -p --output-format json。官方分工表:交互开发与一次性任务用 CLI,CI/CD、定制应用与生产自动化用 SDK。截至 2026-07-17。Claude Code Docs · Agent SDK

03 Claude Code Docs · GitHub Actions —— 在 PR 或 issue 里 @claude,它分析代码、实现功能、修 bug、开出完整 PR;动作为 anthropics/claude-code-action@v1,交互式安装跑 /install-github-app;API key 放仓库 secrets;成本两笔:GitHub runner 分钟 + API token。每个 PR 无须触发的自动审查是另一条产品线(GitHub Code Review)。截至 2026-07-17。Claude Code Docs · GitHub Actions

04 Claude Code Docs · Permission modes —— 无人场景的两档:dontAsk 自动拒绝一切未预批的调用,适合锁死的 CI;auto 由分类器把关,但在 -p 非交互模式下被连续拦截会直接中止会话 —— 没有人可以回退询问。截至 2026-07-17。Claude Code Docs · Permission modes

05 Claude Code Docs · Overview —— 定时的活交给 Routines:跑在 Anthropic 托管的基础设施上,你的电脑关了也继续,可由日程、API 调用或 GitHub 事件触发,CLI 里 /schedule 创建;本机定时任务归桌面端。截至 2026-07-17。Claude Code Docs · Overview

活可以不在你眼前跑，
规矩必须先于它出发。

验收 · 怎么评测你委派的活.

派活容易，难的是确认它真做对了。这是编排时代唯一持续稀缺的技能 —— 会用 agent 的人过剩，能系统性验收 agent 的人不够。凭手感看两眼 diff 不算验收：你需要能复跑的检查（测试、类型、跑一遍看结果）、一套判断它是否跑偏的标准、以及把「这次错在哪」沉淀成下次能自动拦住的用例。这一章把你日常的「看一眼就合并」升级成一套真正的验收工作流，并接上《Agent 实战》里评测那一章 —— 这也是这本书和那本书真正合流的地方：Claude Code 是你派活的地方，评测是你确认没被坑的地方。

到第 5 章，你已经会把活派出去；到第 8 章，活甚至不在你眼前跑。于是只剩一个问题：交回来的东西，你怎么知道是对的？扫两眼 diff、看它说一句「测试通过了」、合并 —— 那不是验收，是祈祷。这一章讲编排时代唯一持续稀缺的技能：会派活的人已经太多，能系统性验收的人还太少。

— I

底线：一条能复跑的检查.

没有检查的时候，「看起来做完了」是它唯一的停机信号，而你是唯一的验证回路 —— 每个错误都躺在 diff 里，等你亲眼看见。所以 Anthropic 自己的最佳实践把这条排在第一：给它一个**它能自己跑的检查** —— 测试、构建、一张能对比的截图。有了能返回过/不过的信号，回路才闭合：它干活、跑检查、读结果、修到过为止。[1] 检查不必隆重。一条测试、构建的退出码、一个 linter、一个把输出和样例对拍的脚本、一张和设计稿并排的截图 —— 任何能压成过/不过的东西都算。再加一条纪律：**要证据，不要断言**。让它交测试输出、贴跑过的命令和返回，而不是一句「我搞定了」；看证据比你亲手重跑一遍快，也是你敢不盯着它的前提。[1]

提示词

派活时把验收一起派下去

这件活的验收标准：<一条可检验的标准>。
做完后跑 <检查命令>，把完整输出贴给我。
没验证过的部分明说「没验证」，不要写「应该可以」。
检查跑不过就修到跑过再交，别把失败的输出藏起来。
最后列出你改过的每个文件和一句为什么。

这件活的验收标准：登录超时后刷新 token 的用例全部通过。
做完后跑 npm test -- src/auth，把完整输出贴给我。
没验证过的部分明说「没验证」，不要写「应该可以」。
检查跑不过就修到跑过再交，别把失败的输出藏起来。
最后列出你改过的每个文件和一句为什么。

· · ·

— II

第二层：对着计划审，换个脑子审。

检查证明「它能跑」，证明不了「是你要的」。跑得过测试的代码，也可能顺手重构了你没让它动的模块、绕开了你点名的方案。第二层验收要有基线 —— 最现成的基线，是计划模式里你批准过的那份计划（第 2 章）：把 diff 对着计划逐条核，要求都实现了吗、点名的边界情况有测试吗、范围之外的东西动了没有。这层审最好换个脑子来。写代码的那个上下文装满了它自己的推理，官方文档说得直白：新上下文审得更好，因为它不会偏心自己刚写的代码。[1] subagent 恰好提供这种隔离 —— 独立上下文、独立提示，只看 diff 和你给的标准，把发现交回主对话[2]；内置的 /code-review 就是这么做的：在一个全新的 subagent 里审当前 diff。[1]

— 注意

一条反向纪律：被派去找碴的 reviewer，几乎总会交出几条碴 —— 哪怕活本身是好的，因为找碴就是它接到的任务。逐条照办会把你推向过度工程。所以给 reviewer 的标准里写明：只报影响正确性和既定要求的，其余当可选建议。[1] 这不是把审查放松，是把「找什么」说清楚。

— III

固化:从「你记得看」到「机器替你拦」.

同一套手动验收做到第三遍,就该问:哪部分能固化?Claude Code 给了三个台阶,约束力一级比一级硬。第一阶是 CLAUDE.md:把「提交前要跑 lint」写进去,它每次开工都读 —— 但文档说得明白,这是**上下文,不是强制配置**,它大概率照做,不保证。[3] 第二阶是 hooks:用户定义的命令,在生命周期的固定时点自动执行 —— `PostToolUse` 在每次改完文件后跑格式化,`PreToolUse` 在它碰危险路径之前直接拦下,`Stop` 在它想收工时跑你的检查、不过不放行。[4] hooks 由 harness 执行,不经过模型的自觉 —— 这是请求和规矩的分界,第 7 章展开。第三阶,是把犯过的错写成一条测试:从此那个错不归你拦,归测试拦。

它坑过你的方式	沉到哪里	约束力
忘了你交代过的规矩	CLAUDE.md / Skills	请求 —— 大概率照做,不保证
跳过了该跑的检查	Hooks(PreToolUse / Stop)	规矩 —— harness 执行,跳不过
犯过一次的逻辑错误	一条测试用例	回归 —— 从此自动拦截

最外圈,验收本身也离开你的手:`claude -p` 一条命令无人值守跑一遍检查,`--output-format json` 让脚本直接消费结果[5];接进 CI,每个 PR 自动过一遍审(第 8 章的地界)。你的角色从「亲自验每一件」退到「维护验收器 + 抽查」。顺带说清一个细节:Stop hook 连续拦 8 次后会放行[1] —— 它是一道门,不是一座牢;真正的兜底,永远是测试和 CI 里那些不会松口的检查。

— IV

一次对了不算数:接上评测.

单件活的验收到此闭环。但你迟早会发现自己在重复派同一类活 —— 每周的依赖升级、每个 PR 的安全审、每天的日志巡检。这时候该换单位了:单件问「这次对不对」,批量要问「这类活它多稳」。τ-bench 给这个直觉起了正式的名字:**pass@k** 量「k 次里至少对一次」,是能力上限;**pass^k** 量「k 次全对」,是可靠性。实测扎心:当时最强的 function calling agent(gpt-4o)任务成功率不到 50%,retail 域连续 8 次全对的 pass^8 掉到 25% 以下。[6] 跑对一次和次次跑对之间,隔着「你敢不敢把它设成定时任务」的全部距离。第二个坑在「改进」上。你换了模型、改了 CLAUDE.md、加了个 skill,感觉变好了 —— 感觉不算数。评测是实验,实验自带噪音:报成功率要带标准误;比较改动前后,用同一批任务的逐题配对差,而不是两个总分相减;想在 80% 功效下检出 3 个点的差距,大约要 1000 道题。[7] 日常尺度记一条就够:**小样本上的个位数点差,不构成「变好了」的结论** —— 它只构成「再跑几遍」的理由。落地不用自建 harness。把它坑过你的每个案例攒下来 —— 十几二十条,每条带上第 I 节那样的可检验标准,就是你的回归集;每次动配置,用 `claude -p` 把这批活重跑一遍、拿 JSON 结果对一对。[5] 这正是《Agent 实战》第 10 章那套 golden set + 回归的家用版,也是两本书真正合流的地方:那本书教你把评测建成产线,这本书把它接到你每天派活的台子上 —— Claude Code 是你派活的地方,评测是你确认没被坑的地方。动手 · 把这章装进你的下一次派活:

- [ ] 给你最近派出的一件活补一条能复跑的检查,并要它交证据(输出、命令、截图)—— 不收「搞定了」。
- [ ] 挑一份你本来打算直接合的 diff,先让一个全新上下文的 subagent 对着计划审一遍,数数它找出几条你没看到的。
- [ ] 把它最近一次坑你的错,按第 III 节的表沉淀掉:进 CLAUDE.md、变成一条 hook,或写成一条测试。

参考.

01 Claude Code Docs · Best practices —— 验证一节:给 Claude 一个它能自己跑的检查(测试、构建、一张能对比的截图);没有检查,「看起来做完了」是唯一信号,你就是人肉验证回路。闸门从轻到重:同一条 prompt 里跑检查、/goal 让独立评估器每轮复查、Stop hook 当硬门(连拦 8 次后放行)、fresh-context 的验证 subagent 试图推翻结论。另有:要证据不要断言;新上下文审代码更好,「不会偏心它自己刚写的代码」;内置 /code-review 在 fresh subagent 里审当前 diff;被派去找碴的 reviewer 即使活是好的也会报出问题 —— 要求它只报影响正确性与既定要求的。截至 2026-07-17。Claude Code Docs · Best practices

02 Claude Code Docs · Subagents —— 每个 subagent 在自己的上下文窗口里运行,带自己的系统提示与工具权限;独立干活,只把结果交回主对话。截至 2026-07-17。Claude Code Docs · Subagents

03 Claude Code Docs · Memory —— CLAUDE.md 与 auto memory 都是「上下文,不是强制配置」;要不论模型怎么决定都拦住某个动作,用 PreToolUse hook。截至 2026-07-17。Claude Code Docs · Memory

04 Claude Code Docs · Hooks reference —— hooks 是「用户定义的 shell 命令、HTTP 端点或 LLM 提示,在 Claude Code 生命周期的特定时点自动执行」;事件含 PreToolUse、PostToolUse、Stop、SessionEnd 等,可拦下工具调用、改写输入输出、注入上下文。截至 2026-07-17。Claude Code Docs · Hooks

05 Claude Code Docs · Run Claude Code programmatically —— claude -p 非交互跑一次;--output-format 可选 text / json / stream-json,--json-schema 约束结构化输出,--allowedTools 限定权限;文档给出的场景是脚本与 CI/CD。截至 2026-07-17。Claude Code Docs · Run programmatically

06 Yao et al. · 「τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains」(2024,arXiv:2406.12045)—— 提出 pass^k「度量 agent 行为在多次试验下的可靠性」(k 次全部成功的概率,区别于 pass@k 的至少一次);实测当时最强的 function calling agent(gpt-4o)任务成功率不到 50%,retail 域 pass^8 低于 25%。arXiv · τ-bench / pass^k (2406.12045)

07 Miller · 「Adding Error Bars to Evals: A Statistical Approach to Language Model Evaluations」(Anthropic,2024-11,arXiv:2411.00640)—— 把评测当实验:报成功率要带标准误;比较两个配置,用同一批题上的题级配对差做推断,而不是两个总分相减;动手前先算最小可检测效应(MDE)—— 论文算例:80% 功效下检出 3 个点的差距约需 1000 题。arXiv · Adding Error Bars to Evals (2411.00640)

派活的人已经过剩,
验收的人还稀缺.

节奏 · Pro/Max、选模型、成本、同类分工.

选档不是看价格，是看你撞哪堵墙：额度、模型、还是并发。这一章把 Pro、Max(5× / 20×)摆开，讲清 Opus / Sonnet / Haiku 各自的活该怎么派、什么时候开思考、prompt caching 怎么帮你省钱、/fast 什么时候值。再划一条同类分工的线：Claude Code 和 Cursor / Codex / Copilot / Warp 各自强在哪，一个 builder 的组合怎么搭 —— 哪些活留在 Claude Code，哪些活切走。选对节奏，比选贵的档更省钱。

选档的错误问法是「哪档划算」,正确问法是「我撞的是哪堵墙」。额度不够,是一堵墙;默认模型不够强,是另一堵;等不起响应速度,是第三堵 —— 三堵墙对应三个不同的开关,只有一个真需要花更多钱。这一章把档位、模型、力度、成本四个旋钮摆开,最后划一条和同类的分工线。

— I

档位与三堵墙.

先把价目摆平:Pro 月付 $20(年付折合 $17/月,一次预付 $200);Max 从 $100 起,按比 Pro 多 5× 还是 20× 用量分两档。[1] 两档都含 Claude Code,并且和 Claude 应用**共享同一个额度池** —— 你在网页里聊掉的,和在终端里跑掉的,是同一笔钱。[2]

你撞的墙	症状	开关
额度	下午就见底,活排着队	升 Max 档,或先做第 III 节的省额度纪律
模型	难题上明显吃力	Pro 默认 Sonnet 5,Max 默认 Opus 4.8 且 Opus 自动 1M 上下文
速度	交互时等不起	/fast(走 usage credits,另一本账)

第二行值得展开:两档的差距不只是量,还有默认起点 —— Pro 账号的 default 落在 Sonnet 5,Max 落在 Opus 4.8,而且 Max / Team / Enterprise 档的 Opus 自动升到 1M token 上下文,长会话、大仓库时这一条常常比用量倍数更值钱。[3] 判断顺序照表走:先确认撞的是哪堵墙,再动钱包 —— 三堵墙里只有额度墙必须用钱解决。

· · ·

— II

模型与力度:两个旋钮,一套分工.

模型的日常分工用别名说话:`sonnet`(当前解析到 Sonnet 5)管日常编码,`opus`(Opus 4.8)管难题与长程任务,`haiku` 管机械杂活 —— 最顺手的用法不是主对话换来换去,而是第 5 章那招:主对话坐大模型,批量杂活派给 `model: haiku` 的 subagent。两个组合别名替你省判断:`opusplan` 计划时用 Opus 想、执行时切 Sonnet 干,正是第 2 章那个循环的性价比形态;还有一个天花板 `fable`(Fable 5),任何档都不是默认,最难的长程活才值得 `/model fable` 主动请它出山。[3] 第二个旋钮是力度。effort 五档(low 到 max)控制它每一步想多深,Sonnet 5 与 Opus 4.8 默认都在 high:难题升 `/effort xhigh`,机械活降 low —— 同一个模型,力度对了比换模型便宜。[3] 「开思考」在这套体系里已经不是单独的开关:思考深度由 effort 主控,你只保留否决权 —— Option+T(macOS)/ Alt+T 会话内开关,`/config` 定全局,Ctrl+O 展开看它想了什么。[3]

· · ·

— III

成本三律:缓存、快慢、上下文.

第一律:**别在会话中途乱切**。prompt caching 全自动打理,订阅账号还自动享受 1 小时的长缓存 —— 但缓存按模型、按 effort 分开存,会话中途 `/model` 或 `/effort` 一切,下一条回复就要无缓存重读全部历史,又慢又费;用选择器切换时 Claude Code 会弹确认框拦你,正是这个原因。官方建议照抄即可:模型和力度在会话开头定好,`/compact` 留给任务之间的自然断点;走错路想回头,`/rewind` 比 `/compact` 便宜 —— 它落回的是已有缓存。[5] 第二律:**/fast 是用钱换等待,想清楚再开**。fast mode(研究预览)让同一个 Opus 最快提到 2.5 倍速,代价是 $10 / $50 每百万 token 的高价 —— 订阅用户注意,它**不吃你的订阅额度**,只从 usage credits 扣,是单独的一本账。[4] 所以它的正确用法窄而清晰:赶工的交互时段开(快速迭代、现场调试),长的自动化任务与 CI 一律关;撞到 fast 的限流它会自动回落标准速度,不至于断档。[4] 第三律:**上下文纪律就是额度纪律** —— 换任务 `/clear`、调研派 subagent(第 5 章)、检查交给 hooks(第 7 章),每一条前面章节的习惯,在账单上都有回声。

· · ·

— IV

同类分工:入口跟着活走.

最后划分工线 —— 不是黑稿,是各自的主场。**Cursor**(个人档 Pro $20 起,顶到 Ultra $200)的主场在编辑器:你要「陪着改、盯着每一步 diff」的工作节奏时,编辑器侧的 agent 工作台仍是最顺的入口。[6] **Copilot**(Pro $10 起)的主场在 GitHub 本体:issue 直接派给它、收回一个 PR,甚至能把活转派给 Claude、Codex 这些第三方 agent —— 你的团队整个活在 GitHub 流程里时,它是那条流程自带的手。[7] **Codex** 是 OpenAI 的对位产品,CLI、IDE 扩展、云端、ChatGPT 应用四路入口 —— 已经

付着 ChatGPT 的人自然先试它。[8] **Warp** 自我定位是「面向 agentic coding 的现代终端」(Build $20/月):当你的活以终端会话本身为中心 —— 多面板、长命令、运维流 —— 它贴得更近。[9] 组合的原则一句话:**入口跟着活走,订阅别叠着买** —— 每多一份订阅都该对应一类你真在做的活;先把一份用满,再谈第二份。

> – 补充
>
> 本章全部价格、档位、模型名核对于 2026-07-17;这些是全书更换最勤的数字,读到时请以各家官网为准。判断框架 —— 三堵墙、两个旋钮、三条成本律 —— 比数字活得久。

动手 · 用一周数据代替直觉:

- [ ] 记录一周你每次「用不了」的原因:额度见底、模型吃力、还是嫌慢 —— 月底对着三堵墙的表决定动不动钱包。
- [ ] 把 subagent 的杂活降到 haiku、主对话保持默认,跑一周看额度曲线的变化。
- [ ] 只在赶工的那一小时开 /fast,完事就关 —— 月底看 usage credits 那本账,判断这个「快」值多少钱。

参考.

01 Claude · Pricing —— Pro: 月付 $20,年付折合每月 $17(一次预付 $200);Max: 从 $100 起,可选比 Pro 多 5× 或 20× 用量两档;Pro 与 Max 均标明含 Claude Code。截至 2026-07-17。Claude · Pricing

02 Claude 帮助中心 · Use Claude Code with your Pro or Max plan —— 一份订阅同时覆盖 Claude 应用与 Claude Code,两边用量共享同一个额度池。截至 2026-07-17。Claude Help · Claude Code on Pro/Max

03 Claude Code Docs · Model configuration —— 默认模型按账户:Pro(及 Team Standard、Enterprise 订阅席位)默认 Sonnet 5,Max(及 Team Premium、按量 Enterprise、API)默认 Opus 4.8;Max / Team / Enterprise 档的 Opus 自动升到 1M 上下文。别名:opus / sonnet / haiku / fable(Fable 5 任何档都非默认,需 /model fable 主动选)、opusplan(计划用 Opus、执行切 Sonnet)、sonnet[1m]。effort:Sonnet 5 与 Opus 4.8 支持 low–max 五档,默认 high,/effort 调;思考由 effort 主控,Option+T(macOS)/ Alt+T 会话内开关,/config 设全局,Ctrl+O 展开思考内容。会话中途用 /model 选择器切换会弹确认(直接给定名字则立即切)—— 无论哪种,下一条回复都要无缓存重读全部历史。截至 2026-07-17。Claude Code Docs · Model configuration

04 Claude Code Docs · Fast mode —— 研究预览:同一个 Opus、优先速度的配置,最快 2.5×,按 MTok 计价 $10 / $50(Opus 4.8);/fast 开关;订阅档(Pro/Max/Team/Enterprise)只能经 usage credits 使用,不占订阅额度、从第一个 token 起按 fast 价计;撞到 fast 限流自动回落标准速度。截至 2026-07-17。Claude Code Docs · Fast mode

05 Claude Code Docs · How Claude Code uses prompt caching —— 缓存全自动管理;订阅账号自动用 1 小时 TTL(超出套餐、转 usage credits 后自动降到 5 分钟)。会作废缓存、让下一条又慢又贵的动作:中途切模型(每个模型各有缓存)、改 effort、首次开 fast mode、/compact;/rewind 反而落回已有缓存。官方建议:模型和 effort 在会话开头定好,/compact 留给任务间的自然断点。截至 2026-07-17。Claude Code Docs · Prompt caching

06 Cursor · Pricing —— 个人档逐档(月付):Hobby 免费、Pro $20、Pro+ $60、Ultra $200;Teams $40/座·月,Enterprise 定制;页面默认的年付视图显示 Pro $16、Teams

$32(八折)。逐档价格取自页面自带的结构化数据(JSON-LD)与档位选择器;卖点为 Agent 请求、code review、cloud agents。截至 2026-07-17。Cursor · Pricing

07 GitHub Copilot · Plans —— Free / Pro $10 / Pro+ $39 / Max $100(每人·月),另有 Business 与 Enterprise;卖点原话:「把活派给 Copilot,它创建一个 pull request」,并可把任务转派给 Claude、Codex 等第三方编码 agent。截至 2026-07-17。GitHub Copilot · Plans

08 OpenAI · Codex 文档 —— OpenAI 的编码 agent,入口覆盖 ChatGPT 桌面端与网页、Codex CLI、IDE 扩展与 Codex cloud。截至 2026-07-17。OpenAI · Codex docs

09 Warp · Pricing —— 自我定位「面向 agentic coding 的现代终端」;Free / Build $20 / Max $200(月付),另有 Business 与 Enterprise。截至 2026-07-17。Warp · Pricing

---

选对节奏,
比选贵的档更省钱.

收束 · 一个人，一支 agent 队.

把整本书压成一句：Claude Code 的问题从来不是「它能不能写这段代码」，是「你能把多少活派出去、又能不能确认它做对了」。编辑器是它最小的样子；计划循环、记忆、Skills、subagents、MCP、hooks、headless 把它撑成一个人指挥一支 agent 队的编排台。剩下的，是知道那条边界画在哪 —— 哪些活自己盯、哪些活派出去、哪一步该切走 —— 然后让这支队伍替你把活干完，而你只做那件机器替不了的事：决定做什么，以及确认它真做成了。

把十章压成一句：Claude Code 的问题从来不是「它能不能写这段代码」，是「你能把多少活派出去 —— 又能不能确认它真做对了」。写代码只是它最小的样子；你付费买到的，是一个人指挥一支 agent 队的编排台。

– I

回扣：六件与两翼.

回扣一遍这台机器怎么拼起来的。循环给了节奏 —— 先批方案，再收结果（第 2 章）；记忆免了重复交代（第 3 章）；Skills 和输出风格把你的做法与口吻收编成资产（第 4 章）；subagents 给你人手，一句话铺开一支并行的队（第 5 章）；MCP 给这支队素材，工单、报错、数据自己长进上下文（第 6 章）；hooks 立规矩，到点必然执行、不看模型脸色（第 7 章）。再加两翼：headless 让活离开你的终端，在 CI 里、在夜里、在你不在的地方跑完（第 8 章）；验收让你敢收 —— 检查、复审、回归，凭据在手才算完（第 9 章）。四个入口，一台引擎，你的配置走到哪跟到哪。[1]

• • •

— II

你只做机器替不了的两件事.

队伍成形之后,你的工作只剩两件,恰好是机器替不了的两件:**决定做什么**,和**确认它真做成了**。前者是品味和取舍,后者是这本书反复回来的那句 —— 给它一个能自己跑的检查,要证据,不要断言。[2] 派活的人已经太多,这两件事上的功夫,才是编排时代里你真正的手艺。

• • •

— III

边界,和一把不过期的尺.

剩下的是画边界。哪些活自己盯:不可逆的、高风险的、要品味的。哪些活派出去:可验证的、可并行的、重复的。哪一步切走:该在编辑器里陪着改的、长在 GitHub 流程里的、以终端本身为中心的 —— 入口跟着活走(第 10 章)。最后说句实话:这本书会过期,而且很快 —— Claude Code 以周为单位在变,书里每个数字都标了核对日期,过期请以官方文档为准。但这把尺子不过期:拿到任何一个新特性,还是那三问 —— **它省什么,什么活用它,怎么验收**。问完这三问,你就不需要下一本教程了。

参考.

01 Claude Code Docs · Overview —— 终端、IDE、桌面应用与浏览器四个入口连着同一台引擎,CLAUDE.md、设置与 MCP 跨入口通用。截至 2026-07-17。Claude Code Docs · Overview

02 Claude Code Docs · Best practices —— 验证一节的第一条:给它一个它能自己跑的检查;要证据,不要断言。截至 2026-07-17。Claude Code Docs · Best practices

---

一个人,一支 agent 队:
你派活,你验收,其余交给它们.

CLAUDE CODE

从写代码到指挥 agent.

官方的定位还是「编码工具」。它其实是你指挥并验收一支 agent 队的编排台
—— 写代码只是它最小的样子。

扫码在线阅读

作者 · AKLMAN

Aklman · 文库 —— 写给已经订阅 AI 工具、却只用到一小部分的人；每本短书讲清一份订阅里该学、该跳过、该换入口的部分。写代码，也写字。

这是静态 PDF 快照；网页版阅读体验更佳，内容持续更新、数据更及时准确。

library.aklman.com · x.com/aklman2018 · github.com/aklmans

章节	字数	更新	版本
11	1.4 万字	2026.07	PDF 1.0
