— 技术 · 实践

CURSOR

不只是编码 agent

官网卖的是 AI 编辑器，它其实是一台元工具——代码之外的调研、写作、数据，也能在一个窗口里干完。

目录.

引子 · 把它当工作台，不是编辑器.

你为 Cursor Pro 付了钱，用的是 Tab 补全和在 Chat 里问两句代码。官网把它卖成「AI 代码编辑器」，你也就当编辑器用。但它内置 agent、集成终端、能读整个项目、能接 MCP —— 这套组合的边界，不是「它能写什么代码」，是「一台电脑上的活，有多少能在一个窗口里干完」。这本书讲的就是编辑器之外的那部分。

你大概是这样用 Cursor 的：写代码时按 Tab 接受补全，卡住了在 Chat 里问一句，要改某段就 Cmd+K。一天下来，它对你来说就是一个补全更准、能聊两句的 VS Code。这没错，但这是它最小的样子。官网把 Cursor 卖成「AI 代码编辑器」，你也就当编辑器用 —— 而同一个 Cursor 里还装着一个能自己跑命令的 agent、一个集成终端、一份读过你整个项目的索引，以及一个能接外部数据和工具的 MCP 接口。把这几样合起来看，它的边界就不再是「能写什么代码」，而是**「一台电脑上的活，有多少能在这一个窗口里干完」**。这本书讲的，就是编辑器之外的那部分。

– I

你只用了它最小的样子.

你为它付了 Pro，用的却是任何 AI 编辑器都有的那三样。截至 2026-07-10，一份 Pro 是每月 $20。[1] 这 $20 大多数人只花在 Tab 补全、Cmd+K 行内改、Chat 问答上 —— 这层好用，但它是 Cursor 和别家拉不开差距的地方，也是九成用户停下的地方。真正让它不一样的东西在更下面：一个会自己改多个文件、自己在终端跑命令、看结果再调整的 Agent，一份把你整个代码库切成向量、能按意思搜的索引，以及把外部系统接进来的 MCP。[2] 所以广度不是你没用上的福利，是一笔你已经付了、却没提的款。这本书想做的，是替你把这笔款里真正划算的那部分取出来 —— 不是把功能列全，是替你筛掉低价值的那些。

Cursor 是	Cursor 不是
一个 agent-native 的软件开发环境	VS Code 快捷键教程
代码、资料、命令、工具和 agent 的工作台	一份 Cursor docs 翻译
减少上下文搬运的元工具	万能工作台，或同类工具的黑稿
需要 Git、测试、权限和 review 兜底的执行环境	安全接管你整台电脑的 OS-level agent

工作面	它省什么	边界
Edit / Tab	低摩擦补全和局部连续编辑	跟随你的方向，不替你定方向
Cmd+K / Inline Edit	选区内改写和小片段生成	选区就是天花板
Chat / Ask	解释、问答、局部方案	理解，不执行
Agent	多步骤执行和多文件修改	省执行，增加 review 责任
Terminal	离开编辑器跑命令的切换	命令能破坏真实文件
Index / @	手工找文件和搬上下文	索引不是完美记忆，@ 要指准
Rules / AGENTS.md	重复说明偏好和流程	规则会腐烂，越多越稀释
MCP	外部数据和工具接入	权限边界扩大
Cloud / CLI / Bugbot	等待、本机占用和机械 review	远程环境、成本和验收责任

> **Cursor 最省的不是敲代码时间，而是上下文搬运成本。**
>
> 本书的第一条判断

· · ·

— II

元工具，不是编辑器.

把 agent、终端、索引、MCP 摞在一起，得到的不是编辑器，是一台元工具。所谓元工具，是一台能装下别的工具的工具。Agent 会用的工具里，有执行终端命令、跨文件编辑、语义搜索、控制浏览器、调 MCP、网页搜索、生成图像。[2] 凡是命令行能做的、外部接口能取的，它都够得着 —— 这意味着它能干的远不止写代码：用网页搜索加 MCP 调研一个主题、产出一份带引用的笔记；在终端里清洗合并一堆 CSV；批量重命名、转换几百个文件；设一条按时间或事件触发的自动化。一个装满资料的文件夹，对它来说就是一个能动手的项目。说它是元工具，不是说它万能。它在每一类活上都有强项和短板：写长文，它不如一个顺手的写作软件流畅；做数据可视化，它不如一个 notebook 直接。这本书的纪律是 —— 每讲一个用法，都告诉你**它省什么、什么活用它、什么时候不如一个专用工具**。不喊「什么都能干」，只给你一把量自己活的尺子。

· · ·

— III

唯一那条边界.

它干不了的那类活，有一条清楚的线：替你去点别的 app。2026 年 2 月起，Cursor 的云端 agent 也能「用电脑」了 —— 但用的是它**自己沙箱里**那台机器：每个 agent 有一台隔离 VM，在自己的桌面和浏览器里构建、点击、验证它刚改的东西，再交一份 merge-ready 的 PR。[3] 这条线要划清

楚：它操作的是它自己造出来待验收的软件，不是你本机上你登录着的那些 app。真正「替你点别的 GUI app」—— 用你的账号发邮件、在你的浏览器里跨站填表、操作你打开着的设计稿 —— 那是另一类工具的事（Claude in Chrome、Operator 这类）。这本书会反复回到这条边界。还有一条分工要先记下：Cursor 是编辑器侧的元工具，Warp 是终端侧的元工具。两者覆盖的活有重叠，选谁、什么时候两个都开，倒数第二章会摆开讲 —— 这本只写 Cursor。

> — 补充
>
> 立场说在前面：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；不接 affiliate，推荐一样东西会告诉你为什么，也会告诉你它的代价。它也不是所有人的最优解 —— 何时该换 Claude Code、Copilot 或 Warp，倒数第二章会照实摆。价格、档位、模型 Cursor 改得很勤 —— 书里每个数字都标了日期，过期了请以官网为准。

动手之前先照一次镜子 —— 看看你平时把多少活反射性地切出了 Cursor：

01 **翻两周，挑 5 件「开了别的 app」才做完的事**

不限于写代码：查资料、整理一批文件、清一份数据、写一段东西、改一堆配置 —— 挑出 5 件你当时离开 Cursor 去别处做的事。

02 **给每件标一列：它的 agent + 终端 + MCP 够得着吗**

凭直觉填「能 / 不确定 / 不能」。先别查，记下你此刻的判断。

03 **数一数「能」和「不确定」有几件**

那几件就是你这本书的起点：本可以不切走的活。读完回来改这张表 —— 它也是你的对照组。

参考.

01 Cursor · 套餐与价格 —— Hobby 免费、Pro $20/月、Pro+ $60/月、Ultra $200/月、Teams $40/座·月；各档含一份模型用量额度，超出按 API 价走 on-demand。截至 2026-07-10。Cursor · Pricing

02 Cursor Docs · Agent —— Agent 可执行终端命令并读输出、跨文件改代码、对索引过的代码库做语义搜索、控制浏览器截图与验证、用 MCP 工具，还有网页搜索与图像生成。Cursor Docs · Agent

03 Cursor Blog · Agents can now control their own computers（2026-02-24）—— 云端 agent 各自有一台隔离 VM，在自己的桌面与浏览器里构建并验证产出，再交 merge-ready PR；用户可远程接管这台沙箱机器，而不是 Cursor 控制你本机的 app。Cursor Blog · Agent computer use

04 Cursor · Changelog —— 版本迭代很快，当前 3.11（2026-07-10）；近一个月新增 Customize 页（plugins / skills / subagents / hooks 统一管理）、iOS app（public beta）、Team MCPs 分发、/automate、Side Chats。每次引用前重新核对版本与日期。Cursor · Changelog

---

编辑器是它最小的样子，
不是它的上限.

底座 · Tab、Cmd+K、Chat 到此为止.

Tab 替你补全，Cmd+K 改选中的几行，Chat 回答你贴进去的问题 —— 这三样好用，好用到你以为 Cursor 就是这些。九成用户停在这一层。这一章先把它讲清楚：每样省什么、各自的天花板在哪，以及为什么停在这里，你就只把它当了个更聪明的编辑器。

Tab 替你补全，Cmd+K 改选中的几行，Chat 回答你贴进去的问题。这三样是任何一个 AI 编辑器都有的底座，也是九成 Cursor 用户唯一用到的层。这一章不教你怎么用这三样 —— 你早会了。它要做的是给每样画一条天花板：它强在哪、到哪儿就不够了。认出那条线，你才知道什么时候该往下一层、把活交给 agent，而不是继续在底座上手动较劲。

入口	先用它，当你…	不要用它，当你…
Tab	已经知道下一行大概是什么，只是不想敲	还没决定设计、接口或边界
Cmd+K / Inline Edit	能选中一个明确范围，让它局部改写	改动会影响调用方、测试或多个文件
Chat / Ask	需要解释、定位、比较方案，暂时不改文件	已经知道要改什么，只差执行
Agent	有目标、范围、测试命令、验收标准	只是想改三行、问一句、或让它猜你的意图

— 要点

省额度规则：能局部就不全局；能指文件就不让它猜；能用 Tab / Cmd+K / Ask 解决，就先别开 Agent。Agent 不是更高级的默认入口，它是有 review 成本的执行入口。

提示词

Cmd+K / Inline Edit

```
只改选中范围：
- 目标：<一句话说明局部改动>
- 保持现有接口和命名
- 不新增依赖
- 不改选区外文件
给我可直接接受的 diff。
```

```
只改选中范围：
- 目标：把这段错误处理改成早返回，避免嵌套 if
- 保持现有接口和命名
- 不新增依赖
- 不改选区外文件
给我可直接接受的 diff。
```

— I

Tab 省的是手，不是脑.

Tab 替你打字，不替你定方向。它依据你最近的编辑、光标周围的代码和 linter 报错，预测下一处改动 —— 不只是补全当前这行，还会顺着你的改动方向往下接，跨行、跨多步。[1] 它最值钱的场景是「我知道要写什么、只是懒得敲」：重命名扩散、补样板、顺着已有模式接下一段。这类连续动作，它替你省掉的是肌肉记忆。天花板也在这里：它跟随，不引领。它接得住你已经定下的方向，却不会替你决定该不该这么写、这块要不要重构、这个设计对不对。一旦你需要的是「想清楚怎么改」而不是「快点敲出来」，Tab 就帮不上了 —— 那是后面 agent 的活。

— II

Cmd+K 的上限就是你的选区.

选中多少，它就改多少 —— 范围就是它的能力边界。Cmd+K(Inline Edit)对你选中的那段代码做定向修改，原地应用、不用开聊天面板。[2] 优点是快、准、不跑题：你框定了范围，它就不会去动别处。「把这个函数改成 async」「这段加上错误处理」—— 这类「我知道改哪、就这几行」的活，Cmd+K 比开一轮对话快得多。它的上限就是你框的那个选区。它只看你给的那段，不做项目级推理 —— 不会去查别的文件怎么调用它、不会顺手改掉受影响的调用方。要跨文件、要先理解整块再动，就别用 Cmd+K 硬撑，切到 Agent。

• • •

— III

Chat / Ask 解释，但不动手.

Ask 把代码库读给你听，但一个字都不改 —— 它的产物是理解，不是改动。Ask 模式只读地搜索你的代码库、回答问题，不碰文件。[3] 它最适合「改之前先看懂」：这块逻辑在哪、为什么这么写、改了会牵动谁。和底座另两样不同，它给的是判断的材料，不是改好的代码。但它不动手 —— 看懂之后，粘贴、运行、应用还是你自己来。从 Ask 跳到 Agent，就是从「它告诉我」跳到「它替我做完」：Shift+Tab 在 Ask / Agent / Plan / Debug 之间轮换。[3] 那一跳，是这本书剩下的全部。

> — 实践提示
>
> 一个自检：如果你用 Cursor 的每一步都还在「自己粘、自己跑、自己改」，你就只买到了一个更聪明的编辑器。底座是给「我知道怎么做、只想快点」的活；要它「替我把活做完」，从下一章的 Agent 开始。还有一条贯穿全书的习惯：把话说准 —— 指名文件、符号、报错行、原因 —— 一次到位比任何模型或设置的调整都更省时间和额度。

把这三样在一件真实任务里走一遍，记下你在哪一步「不够用了」：

- [ ] 挑一件你今天真要做的小任务，先只用 Tab + Cmd+K + Ask 完成它。
- [ ] 记下每一步你用了哪样、它省了什么。
- [ ] 标出第一处你想「要是它能自己跨文件改 / 自己跑一下就好了」的地方。
- [ ] 那一处，就是你该切到 Agent 的信号 —— 下一章接住它。

— 引用与参考

参考.

01 Cursor Docs · Tab —— AI 自动补全，依据你最近的编辑、周围代码和 linter 报错预测下一处改动，可跨行、可跨文件接受多步。截至 2026-07-10。Cursor Docs · Tab

02 Cursor Docs · Inline Edit —— Cmd/Ctrl+K 对选中的代码做小范围定向修改，不离开当前位置、不开聊天面板；要跨文件时 Cmd+L 把选区带进 Agent。截至 2026-07-10。Cursor Docs · Inline Edit

03 Cursor Docs · Agent mode —— 模式分工：Ask 只读(read-only)地回答、不改文件；Agent 动手；Plan 先出可审阅的计划；Debug 拿运行时证据抓疑难 bug。Shift+Tab 轮换模式，各模式各用一份独立上下文。截至 2026-07-10。Cursor Docs · Agent mode

---

补全省的是手，
省不了你定方向.

发动机 · Agent 自己把活干完.

Chat 给你一段代码，你再去粘、去跑、去改。Agent 不是 —— 它自己改多个文件、自己在终端跑命令、看结果、再调整，直到这件事做完。这是 Cursor 从「编辑器」变成「工作台」的那台发动机。这一章讲怎么给它定任务、它默认用哪个模型、什么时候该换前沿模型或开 Max，以及别让它漫无目的地跑。

Chat 给你一段代码，你再去粘、去跑、去改。Agent 不是这样 —— 它自己改多个文件、自己在终端跑命令、看结果、再调整，直到这件事做完。这是 Cursor 从「编辑器」变成「工作台」的那台发动机，也是你多付那笔钱真正买到的东西。前一章的底座替你打字，这一章的 Agent 替你把一件事跑完 —— 这本书后面所有的面（终端、索引、MCP、云端），都是给这台发动机加燃料。

— I

它的循环：改、跑、看、再改.

Agent 不是给你代码，是替你把一件事跑完。给它一个任务（Cmd+I），它会跨文件改代码、在终端跑命令、读回输出、对索引过的代码库做语义搜索、必要时开浏览器验证、调你接的 MCP 工具 —— 然后看结果不对就自己调整，直到达成。[1] 这一圈「改 → 跑 → 看 → 再改」是它和 Chat 的本质区别：Chat 把一段代码丢给你，闭环在你手上；Agent 把闭环收在它自己手里。你的工作因此换了形状：从「逐行写」变成「定义任务、审结果」。这不是说你不看代码了 —— 是你看的从「每一行怎么敲」挪到「这个 diff 对不对」。会用 Agent 的人，省下的不是打字，是把注意力从执行挪到判断。

• • •

— II

先 Plan，别让它瞎跑.

把验收标准定在前面，它就不容易跑偏；不定，它会很自信地做错。复杂、要动很多文件、有多种做法的任务，先按 Shift+Tab 进 Plan 模式：它会提澄清问题、研究你的代码库，生成一份可编辑的实现计划，你审完、改完再让它动手。[2] 计划默认存在 home，可以 Save to workspace 给团队看。最值得记的一条：如果它做偏了，回到计划改，比在执行里一处处纠错更干净。尺子很简单：一句话能说清的小改动、你做过很多遍的活，直接 Agent；要动很多文件、有多种做法、或你自己都没想清楚的，先 Plan。它最强的活是「有明确验收标准」的 —— 先写测试再让它实现，跑得过测试就是做完。别在 Agent 里跑漫无目的的长会话，它会越改越远。会话卫生很简单：一会话一任务、任务尽量控制在三五个文件；一看它在报错里越钻越深，就按 ESC 停、回到计划，别由着它继续挖。岔出去的问题 —— 查个用法、比较个替代方案 —— 别塞进主会话：3.11 起可以用 /side 开一个 side chat，它带着主线的上下文、默认只查只答不动手，答案要用再 @ 回主线。[4] 主会话保持一件事。

任务大小	适合入口	例子	验收
S	Cmd+K / Agent	改一个函数、补一个错误分支	肉眼 review + 单测
M	Agent	跨 2–5 个文件修 bug、补测试、改文档	明确测试命令 + diff review
L	Plan → Agent	重构一块模块、改 API 契约、迁移配置	先审计划，再分阶段跑测试
XL	拆成多个 Agent / 多个 PR	优化整个项目、重做架构、迁移框架	先写 spec，不直接跑

阶段	你给什么	Agent 做什么	你审什么
Plan	目标、范围、禁止事项、验收标准	读代码、提方案、列文件和测试	是否越界、是否漏了验收
Edit	批准后的计划	改文件、保持小步	是否碰了不该碰的文件
Run	允许跑的命令类型	跑测试、lint、脚本、读输出	命令是否可逆、输出是否可信
Test	测试命令和通过标准	修到通过或解释阻塞	是否只修了症状
Review	你自己的判断	解释 diff、列风险	设计意图、安全、成本、回滚

提示词

Cursor Agent

目标：`<一句话说清要做成什么>`
验收：
- `<可检验的条件，最好能跑测试>`
- `<另一个条件>`

约束：用 `<某库 / 某风格>`，别碰 `<某目录 / 文件>`
先给计划，我确认后再动手。

```
目标：给 /api/orders 加分页，支持 page 和 pageSize 两个查询参数
验收：
- tests/orders.test.ts 全绿，新增一条「翻到第二页拿到下一批」的用例
- pageSize 上限 100，超出返回 400
约束：用现有的 zod 校验，别碰 lib/db.ts 的连接逻辑
先给计划，我确认后再动手。
```

- [ ] 禁止把「优化一下项目」「提升质量」「整理一下」这种模糊任务直接交给 Agent。
- [ ] 禁止让它无监督处理生产数据库、真实支付、密钥、云控制台、删除类批处理。
- [ ] 禁止把你自己都没有验收标准的任务交给 Cloud Agents 长跑。

- [ ] 禁止把 Bugbot 或 Agent 的解释当成人工 code review 结论。

· · ·

— III

它背后用哪个模型：默认 Auto.

日常活让 Auto 选，难的那一成才手动换前沿模型。Auto 在智能、成本、稳定之间替你路由，配上 Cursor 自家的 Composer 2.5（专为 agentic 编码训练、又快又省 —— 现在连 Bugbot 也由它驱动），覆盖你大多数的活；这套走的是更宽的「第一方模型池」（Auto、Composer 2.5、Grok 4.5 共用）。[3] 遇到难推理、长链路、它反复卡住的题，再手动切到前沿模型（Claude / GPT / Gemini 这些，走 API 池）；不想自己挑，还有 Premium 路由替你直接挑最强的、按该模型 API 价计费；上下文实在塞不下时开 Max（最大上下文窗口，按模型 API 价计费，烧得快）。判断：默认就让 Auto 跑，别每次手动挑模型 —— 那是在替路由器干活，还更费额度。模型不是越贵越好，是越贵越该留给真难的题。计费的细账（两套额度池、Max 怎么算）放到第 10 章一起说。

> — 实践提示
>
> 让它先写测试、再实现 —— 验收标准定死，它就不容易跑偏；把项目规范和常用命令写进 AGENTS.md（第 6 章），省得每次重新交代。任务大就先 Plan，宁可在计划上多花五分钟，不在错的执行上耗半小时。

- [ ] Diff 是否只动了任务范围内的文件？
- [ ] 测试是新增真实断言，还是只改了快照 / 跳过失败？
- [ ] 有没有新依赖、网络调用、权限扩大、配置改动？
- [ ] 失败路径、回滚路径、日志是否还说得通？
- [ ] 它解释的风险，和你自己看到的风险是否一致？

把这章用到一个你最近手动做过的改动上 —— 编码或非编码都行：

01 **挑一个跨 3 个以上文件的小任务**

可以是代码改动，也可以是非代码的（整理一个笔记文件夹并生成索引、按规则改一批文件）。

02 **先 Plan，把验收标准写清楚，再让 Agent 动手**

你只定义任务和「做完算什么样」，让它去改，你不逐行写。

03 **审它的 diff，记下分工**

记下哪一步它明显比你快、哪一步你还得自己来。这条分工就是你之后该不该开 Agent 的依据。

参考.

01 Cursor Docs · Agent —— Agent(Cmd+I)会跨文件改代码、执行终端命令并读输出、对索引过的代码库做语义搜索、控制浏览器验证、调 MCP 工具，看结果后自我调整直到任务完成。截至 2026-07-10。Cursor Docs · Agent

02 Cursor Docs · Plan Mode —— Shift+Tab 进入；Agent 先提澄清问题、研究代码库，生成一份可编辑、可审阅的实现计划(默认存 home，可 Save to workspace)，批准后再动手；结果偏了「回到计划」比中途纠错更干净。Cursor Docs · Plan Mode

03 Cursor Docs · Models —— 个人档两套额度池：「第一方模型池」(Auto、Composer 2.5、Grok 4.5，included 用量宽得多)与 API 池(手动选 Claude / GPT / Gemini 等前沿模型，按该模型 API 价)。Auto 在智能/成本/稳定间路由；另有 Premium 路由，替你挑最强模型、按所选模型 API 价计费；Max(最大上下文)按模型 API 价。Composer 2.5 是 Cursor 自家 agentic 模型，官方 changelog(2026-06-10)称其现在也驱动 Bugbot。截至 2026-07-10。Cursor Docs · Models

04 Cursor Changelog · 3.11(2026-07-10)—— Side chats：/side 或 /btw 在主会话旁开一个带上下文的支线会话，默认偏向只读的查证与回答，可事后 @ 回主线；Agents 窗口新增会话转录搜索(Cmd+K)，会话内 Cmd+F。Cursor Changelog · Side chats

---

你从写每一行，
变成定义任务、审结果.

动手 · 终端把它变成能干活的环境.

一个能写代码的聊天框，和一个能在你机器上跑命令的环境，是两回事。Cursor 的集成终端让 agent 真的动手：装依赖、跑脚本、批量改文件、清一批数据。这一步也是它能干非编码活的关键 —— 凡是命令行能做的，它都能替你做。这一章讲哪些活该交给它的终端，哪些你该自己盯着，以及 auto-run 那条线划在哪。

一个能写代码的聊天框，和一个能在你机器上跑命令的环境，是两回事。Cursor 的集成终端让 agent 真的动手：装依赖、跑脚本、批量改文件、清一批数据。这一步也是它能干非编码活的关键 —— 凡是命令行能做的，它都够得着。编辑器层让它改文本，终端层让它对结果动手；少了终端，它就退回成一个「会写、不会做」的助手。这一章讲哪些活该交给它的终端、哪些你该自己盯着，以及 auto-run 那条线划在哪。

— I

能跑命令，才算能干活.

会建议的工具很多，能动手的才算一个环境。Agent 执行终端命令并读回输出 —— 装、跑、测、build 在同一个窗口里闭合，跑错了它看着报错自己改。[1] 这就是它比纯聊天多出来的那一截：聊天告诉你「跑 npm test」，Cursor 跑给你看、再读结果接着干。它默认运行在受限环境里：命令限制在工作区内、阻断未授权的文件访问与网络活动，是个有护栏的沙地，不是直接拿你整台机器开闸。这条「能动手」的能力，正是元工具论点的支点。它不止把代码写对，它把「让代码跑起来、让一批文件变成你要的样子、让一份数据清成你要的格式」这类需要执行的活揽了进来。下面两节，一节讲怎么放手、一节讲放到哪儿为止。

— II

auto-run 那条线：省审批，别全放.

让它自动跑读类命令，给写和删留一道闸。在 Settings > Agents > Approvals & Execution 里选 Run Mode，管 shell、MCP、Fetch 各类调用何时直接跑、何时问你；Auto-review(3.6 起的推荐默认)放行 allowlist 内的、能沙箱的命令进沙箱、其余交分类器判断。[2] 你的规矩可以用自然语言写进 permissions.json —— 一句「所有 AWS CLI 命令都要先过我」就够，甚至让 agent 替你写。但 Cursor 自己把这些防护称为「尽力护栏，不是硬安全边界」，分类器会犯错 [3] —— 省事归省事，别一键全放。尺子：可逆、限在工作区内的命令(读文件、跑测试、查状态)，放手让它跑；不可逆或要出网的(rm、数据库迁移、git push、装全局包、调外部 API)，留一道人工闸。自动化省的是你点「同意」的次数，省不了你为一条破坏性命令负的责。真要让它完全无人值守地连跑，放进一个沙箱 / VM，别直接对着你的主仓库。

提示词 Cursor Agent

```
终端任务安全约束：
- 工作目录限制在 <path>
- 先 dry-run / list，不直接写入
- 禁止：rm -rf、sudo、git push、生产数据库、读取 .env*
- 每条会写文件的命令先解释影响范围，等我批准
- 完成后输出：执行过的命令、生成/修改的文件、回滚办法。
```

```
终端任务安全约束：
- 工作目录限制在 ./data/invoices
- 先 dry-run / list，不直接写入
- 禁止：rm -rf、sudo、git push、生产数据库、读取 .env*
- 每条会写文件的命令先解释影响范围，等我批准
- 完成后输出：执行过的命令、生成/修改的文件、回滚办法。
```

- [ ] 命令是否限制在当前 workspace 或明确子目录？
- [ ] 有没有 dry-run、备份、git diff 或临时输出目录？
- [ ] 有没有出网、读密钥、写生产、删文件、改权限？
- [ ] 失败后能不能回滚？不能回滚就不要自动跑。

- [ ] 不要让 Agent 自动跑：`rm -rf`、`sudo`、`chmod -R`、`chown -R`。
- [ ] 不要让 Agent 自动跑：生产数据库迁移、线上写入脚本、云资源删除。
- [ ] 不要让 Agent 自动跑：`git push --force`、发布、部署、真实付款、发邮件。
- [ ] 不要让 Agent 自动读取：`.env*`、私钥、密码库、未授权客户数据。

• • •

— III

终端是非编码活的入口.

凡是命令行能做的，它都能替你做 —— 这就把它从「写代码」推到了「干电脑上的活」。一旦接受「描述结果、它在终端里跑出来」这个模式，非编码的活就涌进来了：批量重命名、转换格式几百个文件；用 python / pandas / csvkit 清洗、合并、去重一堆数据；从一批文档里抽字段、统计、归档。你不必会这些命令 —— 你说要什么结果，它写命令、它跑、读输出再修。这一类「机械但繁琐」的活，是 Cursor 当工作台最立竿见影的地方（第 9 章会把这条线铺满）。诚实的边界：要反复看图调参的交互式数据探索，一个 notebook 更直接；要它操作没有命令行接口、纯 GUI 的软件，它碰不到 —— 那是 Computer Use，第 10 章那条线。终端能覆盖的是「命令行说得清」的活，覆盖不了「只能用鼠标点」的活。

任务	Cursor integrated terminal	Warp
就着代码跑测试、lint、构建	首选：Agent 能读 diff 和输出继续改	可用，但缺少同一份代码上下文
批量文件处理、数据清洗	适合一次性、项目内、可 review 的批处理	适合长期 shell workflow 和反复命令流
系统层运维、跨项目命令	只在需要项目上下文时用	更顺：终端原生、shell 状态清楚
高风险命令	保留人工审批，别全自动	你自己手动跑更可控

> — 注意
>
> 放开 auto-run 之前，先确认 agent 跑的是你这个工作区，而不是一个你不想被它动的目录。受限环境是默认的安全网，别为图快把它整个关掉；真要放宽，按命令类型放，别按「全部允许」放。

挑一件你平时在别的 app 里做、或纯手工做的杂活，交给它的终端：

01 **选一件命令行能做的非编码杂活**

例如把一个文件夹里几十个文件批量改名 / 转格式，或把几份 CSV 合并去重。

02 **只描述结果，让 agent 在集成终端里做**

不写命令，看它写什么命令、怎么跑、读输出后怎么改。

03 **记下你放行了哪些、拦下了哪些**

哪些命令你放心 auto-run，哪一条你按了停。这条线就是你之后配 Run Mode 的依据。

参考.

01 Cursor Docs · Terminal —— Agent 可执行终端命令并读回输出；沙箱在受限环境里跑命令、阻断未授权的文件访问与网络活动，何时直接跑、何时问你由 Run Mode 决定。截至 2026-07-10。Cursor Docs · Terminal

02 Cursor Docs · Run Modes —— 在 Settings > Agents > Approvals & Execution 选模式：Auto-review(推荐)放行 allowlist 内的调用、能沙箱的 shell 命令进沙箱、其余交分类器审；Allowlist 只放白名单；Run Everything 全放。allow / block 规则可用自然语言写进 permissions.json(用户级 ~/.cursor/ 或项目级 .cursor/)。截至 2026-07-10。Cursor Docs · Run Modes

03 Cursor Docs · Agent Security / Run Modes —— Auto-review 自 3.6(2026-05-29)起是推荐默认；官方将 Run Modes 称为「best-effort guardrails rather than a hard security boundary」，并明说「Auto-review is not a security boundary」—— 分类器会犯错。.cursorignore 用来挡 agent 对指定文件的访问。截至 2026-07-10。Cursor Docs · Agent Security

---

会建议的很多，
能动手的才算环境.

读懂 · 索引、@ 引用与非代码资料.

agent 聪明不聪明，一半看模型，一半看它读到了什么。Cursor 把你整个项目切成向量索引，能按意思搜，不只按文件名；@ 让你把某个文件、文件夹、一份文档、甚至一段网页精确喂给它。它读的不必是代码 —— 一个装满笔记的文件夹、一批 CSV、一份手册都行。这一章讲怎么喂得准，以及索引上云那条隐私线。

---

agent 聪明不聪明，一半看模型，一半看它读到了什么。Cursor 把你整个项目切成向量索引，能按意思搜，不只按文件名；@ 让你把某个文件、文件夹、一份文档、甚至一段网页精确喂给它。它读的不必是代码 —— 一个装满笔记的文件夹、一批 CSV、一份手册都行。这一章讲两件事：索引怎么让它「读懂」你的项目，以及你怎么用 @ 喂得准、连非代码资料也喂进去。喂对了上下文，前面那台 agent 才真的有的放矢。

— I

索引让它按意思找，不按文件名.

语义索引是 Cursor 拉开差距的护城河，不是它的 UI。打开工作区，Cursor 就自动把代码切成块、转成向量嵌入，存进向量库，之后约每 5 分钟增量同步、只处理变过的部分；大仓库要等索引到 ~80% 语义搜索才可用。[1] [4] 好处是它能按意思找：你问「我们在哪儿处理鉴权」，它能定位到相关文件，哪怕那段代码里根本没出现「鉴权」这个词。语义搜索叠加传统的 grep，比单用 grep 答得更准 —— Cursor 自家评测给的数字是准 12.5%。[4] 这就是为什么同一个模型，在 Cursor 里比在一个干聊天框里更有用 —— 不是模型更强，是它读过你整个项目。底座层的补全谁都有，这层「它认识你的代码库」才是你为它付钱的核心理由之一。

• • •

— II

@ 是你精确喂料的手.

索引让它能找，@ 让你确保它找对。与其写一长段背景，不如 @ 准那两样东西。常用的几个：[2]

`@Files & Folders`	把指定文件或整个文件夹塞进上下文 —— 最常用，比让它去索引里猜更稳。
`@Docs`	搜索已索引的文档，包括你自己加的 —— 能读非代码资料。
`@Terminals`	把终端输出当上下文 —— 让它就着报错接着改。
`@Commit / @Branch`	把未提交改动或整条分支相对 main 的 diff 给它 —— review、写 commit message 都用得上。
`@Browser`	把内置浏览器里的内容附进来 —— 读网页、文档站。

边界：@ 得太多是噪声，张口就 @ 整个代码库更是 —— 那等于让它做一次全库语义搜索，又贵又容易跑偏。先 @ 准那一两个文件，不够再放宽；把最相关的三五样给它，不是把半个项目都贴进去。喂料的本事不是「喂得多」，是「喂得准」。

你手里有什么	先用	不要一上来用
明确文件 / 组件 / 章节	@file	@codebase
一组同目录资料	@folder	整仓检索
外部文档站或团队手册	@Docs	复制一大段网页
报错、测试输出、日志	@Terminals	口述报错
当前改动需要 review	@Commit / @Branch	让它猜你改了什么
不确定相关文件	先让 Agent search，再要求列来源	一次塞半个项目

• • •

– III

它读的不必是代码.

把一个装满资料的文件夹当项目打开，索引和 @ 照样工作 —— 这是它干非编码活的上料口。指向一个装满笔记的文件夹、一批 CSV、一份手册，@Docs 还能索引外部文档站；agent 就能按意思在你的资料里检索、回答、产出。调研、整理、写作这类活之所以能在 Cursor 里做，正是因为它对待你的资料和对待代码是同一套机制：建索引、按意思找、@ 精确取。

```
project/
  AGENTS.md                # 口吻、引用、文件命名、禁止事项
  docs/
    specs/                 # 任务说明、产品需求、章节大纲
    decisions/             # 已做决策，按日期命名
  research/
    sources/               # 原始来源摘录，只放可追溯材料
    notes/                 # agent 生成或你整理的中间笔记
  data/
    raw/                   # 原始数据，只读
    working/               # dry-run / 中间产物
    output/                # 可交付结果
  prompts/
    templates.md           # 复用 PromptBox / 工作流
```

非代码项目资料结构模板

提示词

Cursor Agent

先读资料，再行动：
- 先阅读 @ <资料文件或文件夹>
- 输出一份资料地图：关键文件、每个文件讲什么、缺口是什么
- 在我确认前，不要改文件
- 确认后只改 <目标路径>
- 每条事实保留来源文件路径。

先读资料，再行动：
- 先阅读 @research/sources 和 @docs/specs/cursor-outline.md
- 输出一份资料地图：关键文件、每个文件讲什么、缺口是什么
- 在我确认前，不要改文件
- 确认后只改 content/books/cursor/chapters/05-context.mdx
- 每条事实保留来源文件路径。

隐私这条线要讲清：官方口径是「Cloud Agents 是唯一需要 Cursor 存储代码的功能」—— 索引流程存的是向量嵌入和加密后的路径，不需要长期存你的源码。[5] [4] 但索引确实上云 —— 真正敏感的仓库或资料，自己掂量，用 .cursorignore 把不想上传的圈掉；它挡索引，也挡 agent 对这些文件的访问，但护栏不是保险箱：真正的密钥别只靠它，放进环境变量或密钥管理。[3] 能被它读懂的资料，才用得上它的脑子；但不是所有资料都该交给它读。

> — 实践提示
>
> 想知道它到底读了什么，去 Cursor Settings > Indexing & Docs 看 included files；该排除的（密钥、私有数据、超大产物）写进 .cursorignore。大仓库索引慢，就用它把生成物、依赖、lockfile 排掉 —— 索引小了、答得也准。

- [ ] 答案像在猜：先 @ 准文件，不要继续追问同一个模糊问题。
- [ ] 找不到新文件：检查索引状态，必要时 Reindex。
- [ ] 索引很慢：把生成物、依赖、二进制、大 CSV 写进 .cursorignore。
- [ ] 引用来源混乱：要求它列出使用过的文件路径和行号，再继续执行。
- [ ] 涉及敏感资料：先决定是否该进 Cursor；不是所有可读材料都该被索引。

拿一份非代码资料试它的上下文：

- [ ] 把一个非代码文件夹（调研笔记 / 一份手册 / 一批数据）当项目打开，让它建索引。
- [ ] 问一个「按意思」的问题，看它能否找到没出现关键词的那段。
- [ ] 用 @ 精确指一份文件纠正它一次，对比前后回答。
- [ ] 去 Settings 确认索引范围，把不该上传的写进 .cursorignore。

参考.

01 Cursor Docs · Codebase Indexing —— 打开项目即自动扫描建索引，约每 5 分钟周期同步；状态栏看进度，命令面板可 Reindex；大仓库把生成物、依赖写进 .cursorignore 提速。截至 2026-07-10。Cursor Docs · Codebase Indexing

02 Cursor Docs · @ Symbols —— 用 @ 精确引用上下文：@Files & Folders、@Docs（含自建文档，可读非代码）、@Terminals（终端输出）、@Past Chats、@Commit / @Branch（git diff）、@Browser（内置浏览器内容）。截至 2026-07-10。Cursor Docs · @ Symbols

03 Cursor Docs · Agent Security —— .cursorignore 用来挡 agent 对指定文件的访问；读文件与搜代码不需审批，会暴露敏感数据的动作需要明确批准。真正的密钥应放环境变量 / 密钥管理，不该只靠 ignore 文件。截至 2026-07-10。Cursor Docs · Agent Security

04 Cursor Blog · Securely indexing large codebases（2026-01-27）—— 索引把代码切块转成向量嵌入；大仓库要等索引完成至少 80% 语义搜索才可用；用 Merkle 树只同步变过的部分；路径以加密形式存储；Cursor 自家评测称语义搜索叠加 grep 比单用 grep 答代码库问题准 12.5%。Cursor Blog · Secure indexing

05 Cursor Docs · Privacy & Data Governance —— 官方口径：「Cloud Agents 是唯一需要 Cursor 存储代码的功能」—— 索引流程与 LLM 请求都不需要长期存你的代码；企业档还有 CMEK（用你自己的密钥加密嵌入）。截至 2026-07-10。Cursor Docs · Privacy

---

模型决定它多聪明，
上下文决定它多有用.

固化 · 用 Rules 和 AGENTS.md 钉住偏好.

同一句要求你跟它说了十遍：用这个库、按这个风格、别碰那个目录。模型不跨会话记事，但你可以把这些钉进文件。Rules（.cursor/rules 里的 .mdc）和 AGENTS.md 就是项目的长期记忆 —— 每次开工自动加载。这一章讲什么该写进去、什么不该，四种触发怎么选，以及固化的落点如何从 Rules 扩到 skills、subagents、hooks 与 Customize 面板。

同一句要求你跟它说了十遍：用这个库、按这个风格、别碰那个目录。模型不跨会话记事，但你可以把这些钉进文件。Rules（.cursor/rules 里的 .mdc）和 AGENTS.md 就是项目的长期记忆 —— 每次开工自动加载，不用你再交代一遍。这件事在非代码项目里同样成立：一个写作或调研文件夹里放一份 AGENTS.md，写清口吻、引用格式、文件命名，agent 的产出就按你的规矩来。这一章讲什么该钉、用哪种触发钉、以及为什么钉多了反而没用。

— I

Rules 是项目的长期记忆.

说十遍不如钉一遍。三层各管一摊：Project Rules 放在 .cursor/rules 下、是 .mdc 文件、随仓库走、团队共享；User Rules 是你的全局个人偏好（口吻、习惯）；Team Rules 从 dashboard 下发、是组织标准、优先级最高。[1] 它们都在 prompt 层提供持久上下文，每次开工自动注入 —— 这就是你不用每次重新交代「这个项目用 pnpm 不用 npm」的原因。一个容易踩的点：User Rules 不作用于 Cmd+K 的行内编辑，只对 Agent 生效。所以你那条「永远用中文注释」写成 User Rule，Cmd+K 不一定听 —— 想让它在所有地方生效，写进项目里。

— II

四种触发，别全设成 always.

rule 设得越多、always 设得越多，它越读不过来。每条 rule 的触发由 frontmatter 决定，四种：`alwaysApply: true`(每次都加进上下文，且会让同一条里的 globs / description 失效)、`description`(让 agent 按描述智能判断相不相关)、`globs`(按文件路径，如 `src/components/**/*.tsx`)、手动(只在 @ 提到时进)。[1] 判断很简单：一条 rule 只挑一种触发 —— 全局铁律才 always，其余按需或按文件。下面是一条按文件触发的项目 rule：

要保存什么	放哪里	原因
跨工具、跨 agent 的项目说明	AGENTS.md	简单 markdown，Cursor / Codex / Claude 都容易读
Cursor Agent 的自动触发规范	.cursor/rules/*.mdc	支持 globs、description、alwaysApply
给人看的背景、架构、安装说明	README.md / docs	不是每次 agent 都该注入全文
只对这一次任务成立的限制	Prompt	不要写进长期规则污染未来任务

```
---
description: React 组件约定
globs: src/components/**/*.tsx
alwaysApply: false
---

- 组件用函数式 + hooks，不用 class。
- 样式走 Tailwind，不写内联 style。
- 每个组件配一个同名 .test.tsx。
```

.CURSOR/RULES/COMPONENTS.MDC —— 只在改组件文件时注入

把所有 rule 都设成 always，等于把整本规范每次全塞进上下文 —— 又占额度，又稀释重点，它反而抓不住真正要紧的那几条。always 留给「永远成立」的三五条，其它交给触发条件。

• • •

— III

AGENTS.md：简单到该先写.

复杂的 .mdc 之前，先写一份 AGENTS.md —— 一份纯 markdown，门槛最低。AGENTS.md 就是项目里一份简单的 markdown，写明 agent 该守的规矩；它可以按子目录嵌套，子目录里的就近覆盖父级。[2] 什么时候上 .mdc：你需要按文件 glob、或按描述智能触发，而不是「永远生效」时。多数项目，一份根目录的 AGENTS.md 加一两条按文件的 .mdc，就够了。

```
# Project Agent Guide

## Scope
- This repository is <what this project is>.
- Before editing, read <canonical docs / architecture file>.

## Commands
- Install: pnpm install
- Test: pnpm test
- Lint: pnpm lint
- Typecheck: pnpm typecheck

## Style
- Follow existing patterns before introducing abstractions.
- Keep changes scoped to the requested files and nearby tests.
- Prefer small diffs and explain trade-offs.

## Safety
- Do not read .env* or private keys.
- Do not run destructive shell commands without explicit approval.
- Do not push, deploy, or change production resources.

## Done Means
- Relevant tests pass.
- Diff is explained.
- Risks and follow-ups are listed.
```

AGENTS.MD 起手式 —— 放项目根目录，AGENT 每次开工先读

坏规则

写高质量代码，注意性能，风格要好，尽量不要出 bug。

好规则

修改 `src/api/**` 时必须新增或更新同路径 `*.test.ts`；如果不加测试，解释具体原因。不要改 `lib/db.ts` 的连接生命周期。

诚实的边界：rules 会腐烂。项目变了、约定改了，旧 rule 还在误导 agent —— 它比没有 rule 更糟，因为你信它。定期删一条过期的，比一直加新的更重要。钉得太多又没人读，等于没钉。

• • •

— IV

固化的落点，从「说法」扩到「做法」.

Rule 钉的是「怎么说」；现在可钉的还有「怎么做、谁去做、什么绝不许做」。2026 年年中起，Cursor 把可固化的东西扩成了一族原语，并给了它们一个统一的家：Customize 页（3.9 起）把 plugins、skills、MCPs、subagents、rules、commands、hooks 收进一个面板，按 user / team / workspace 三层管理，还能从 Marketplace 一键装、看团队排行榜。[3] 别被名词吓到 —— 它们都是同一件事的延伸：把你不想重复交代的东西写成文件。区别只在钉的是什么：

Skill — 钉一类活怎么做	一个装着 SKILL.md 的文件夹，可带脚本和模板，按需加载、/ 一下调用。[4] 你反复贴的那段长 prompt ——「按这个步骤出报告」「照这套规范迁移」—— 升格成 skill，流程本身成了资产。
Subagent — 钉谁去做	给一类活配一个专职角色：自己的 prompt、工具、模型，独立上下文、可并行。[5] 内置的 Explore / Bash / Browser 就是三个官配 —— 把搜代码、跑命令、开浏览器这些吵闹的中间过程隔离在主会话之外。
Hook — 钉铁律	hooks.json 里的脚本挂在 agent loop 的节点上，能观察、拦截、改写：改完自动跑 formatter、扫密钥、把危险写入挡在闸前。[6] rule 是请求，模型可能没听；hook 是强制，代码替你把门。
Command — 钉常用的那句话	一个 markdown 文件、/ 调用，装一段复用 prompt。[3] 比 skill 轻，够用就别升级。

升级路径还是老的：先一份 AGENTS.md；同一句话说了十遍，写成 rule；同一套带脚本的流程重复到第三遍，升成 skill；同一类活需要固定角色、要并行跑，配 subagent；只有「每次都必须、违反就出事」的硬约束，才值得上 hook。Customize 和 Marketplace 解决的是「放哪儿、怎么装」，不替你回答「该不该钉」—— 上一节那条纪律原样适用：钉得越多，读得越少。

> — 实践提示
>
> 让 agent 自己起草：做完一个任务后，让它把刚才反复纠正它的那几点固化下来 —— 内置的 /create-rule、/create-skill、/create-subagent 就是干这个的，它起草、你审一遍再留下。比你凭空想「该写哪些规矩」更准 —— 它知道自己在哪儿踩了坑。

- [ ] 这条规则是否可执行、可检查，而不是愿望？
- [ ] 它是否只说一件事？如果有三件，拆三条。
- [ ] 它是否有正确触发方式，而不是无脑 alwaysApply？
- [ ] 它是否引用现有文件，而不是复制一大段会过期的内容？
- [ ] 它是否含密钥、个人偏好、临时任务限制？这些不该提交。

把你最常重复的那句话钉下来：

01 **找出你跟 Cursor 重复最多的一条要求**

可能是某个库、某种风格、某个别碰的目录，或某种写作口吻。

02 **写成一条 rule 或一行 AGENTS.md，选对触发**

全局就 always / User Rule，按文件就 globs，偶尔才用就 description 或手动。

03 **跑一个任务验证它被遵守，再删一条过期的**

确认你没再重复那句话；顺手清掉一条已经不成立的旧 rule。

参考.

01 Cursor Docs · Rules —— 模型不跨会话记忆；Rules 在 prompt 层提供持久、可复用的上下文。Project Rules 放 .cursor/rules 下的 .mdc（随仓库版本化），User Rules 全局，Team Rules 走 dashboard、优先级最高。触发由 frontmatter 控制：alwaysApply / description（智能判断）/ globs（按文件）/ 手动 @。截至 2026-07-10。Cursor Docs · Rules

02 Cursor Docs · Rules / AGENTS.md —— AGENTS.md 是放在项目里的一份简单 markdown，定义 agent 指令，支持项目根目录与子目录；是相对 .mdc 更轻量的写法。注意：User Rules 不作用于 Inline Edit（Cmd/Ctrl+K），只对 Agent 生效。截至 2026-07-10。Cursor Docs · AGENTS.md

03 Cursor Docs · Customize Cursor —— Customize 页（3.9，2026-06-22 起）把 plugins、skills、MCPs、subagents、rules、commands、hooks 收进一个面板，按 user / team / workspace 三层管理；可从 Cursor Marketplace 一键安装，并有团队最常用插件的排行榜。截至 2026-07-10。Cursor Docs · Customize

04 Cursor Docs · Agent Skills —— skill 是一个装着 SKILL.md 的文件夹（可带脚本、模板、参考资料），从 .cursor/skills / .agents/skills（项目级）与 ~/.cursor/skills 等（用户级）自动发现，兼容 .claude/skills / .codex/skills；agent 按相关性自动选用，也可用 / 手动调用；内置 /automate、/babysit、/review、/create-rule、/create-skill、/create-subagent 等。截至 2026-07-10。Cursor Docs · Skills

05 Cursor Docs · Subagents —— subagent 是可委派的专职 agent：独立上下文窗口、可并行、可前台或后台跑，能配自己的 prompt、工具与模型；内置 Explore / Bash / Browser 三个；自定义放 .cursor/agents/（兼容 .claude/agents / .codex/agents）；编辑器、CLI、Cloud Agents 里都能用。截至 2026-07-10。Cursor Docs · Subagents

06 Cursor Docs · Hooks —— 在 hooks.json（项目或用户级，也可随插件装）里定义脚本，挂在 agent loop 的节点上，可观察、拦截、改写行为：改完跑 formatter、扫 PII / 密钥、把危险操作（如 SQL 写入）挡在闸前；cloud agents 也会执行仓库里 command 型的 hooks。截至 2026-07-10。Cursor Docs · Hooks

说十遍不如钉一遍，
钉多了又没人读.

接外 · 用 MCP 把数据和工具接进来.

你的上下文大多不在项目里 —— 在 Linear 的工单、Figma 的稿、数据库的表、线上的日志里。MCP 让 agent 直接读它们、调它们，不用你来回粘。这是 Cursor 从「写代码」扩到「几乎做任何事」的接口。这一章讲 mcp.json 怎么配、哪些服务器值得接，以及为什么「能接」不等于「该接」。

你的上下文大多不在项目里 —— 在 Linear 的工单、Figma 的稿、数据库的表、线上的日志里。MCP 让 agent 直接读它们、调它们，不用你来回粘。这是 Cursor 从「写代码」扩到「几乎做任何事」的接口，也是这本书「元工具」论点最关键的那块：代码库内的活靠索引，代码库外的活靠 MCP。这一章讲 mcp.json 怎么配、哪些服务器值得接、以及为什么「能接」不等于「该接」。

— I

mcp.json 一配，工具就在它手里.

接一个 MCP 服务器，就等于给 agent 多发一套它能调的工具。先说省事的路：官方或社区的服务器，直接在 Customize 页 / Marketplace 里一键装，OAuth 当场走完，不用碰 JSON。[1] 手配则放两处：项目级 `.cursor/mcp.json`（随仓库走），全局 `~/.cursor/mcp.json`。传输有三种：stdio 跑本地命令、SSE 和 Streamable HTTP 接远程（带 OAuth）。服务器把能力暴露成 Tools（可调用的函数）、Prompts、Resources，agent 在 Chat 和 Plan 里按需调用。配一次，它以后自己去取，不用你再贴。下面是一个本地服务器加一个远程服务器：

```
{
  "mcpServers": {
    "local-tools": {
      "command": "python",
      "args": ["mcp-server.py"],
      "env": { "API_KEY": "${env:MY_API_KEY}" }
    },
    "remote-service": {
      "url": "https://example.com/mcp",
      "headers": { "Authorization": "Bearer ${env:TOKEN}" }
    }
  }
}
```

注意密钥那一行用的是 ${env:MY_API_KEY}，不是明文 —— 把真实凭证写进 mcp.json 再提交，等于把钥匙挂在仓库里。用环境变量引用，远程的尽量走 OAuth。

• • •

— II

它是「几乎做任何事」的接口.

代码库内的活靠索引，代码库外的活靠 MCP。实际接法：Linear（读写工单）、Figma（取设计稿）、数据库（查表、跑只读查询）、Sentry（拉线上错误）、各种网页 / 文档服务器。接上之后，agent 不再需要你把这些粘进来 —— 它自己去工单里读需求、去数据库里核对字段、去错误平台里定位堆栈。对非编码的活，MCP 是它的数据臂膀：调研时拉一手数据、运维时查状态、写作时取资料。这正是「在一个窗口里干完」能成立的原因之一 —— 不是 Cursor 内置了这些功能，是它能通过 MCP 把你已经在用的系统接进同一个 agent。

• • •

— III

能接，不等于该接.

每接一个服务器，都同时多了一份攻击面、一截上下文膨胀、一个可能误触的工具。[2] 默认每次调用都要你审批，可在 permissions.json 预授权一个 allowlist；但权限给得越宽，风险越大。一个 MCP 服务器是在你机器上、用你的凭证跑的外部代码 —— 它能读什么、能写什么，连之前就要看清，给最小的那一份。来路不明、你没法读源码的服务器，别接。

风险级别	例子	默认策略
只读	读文档、读 issue、查只读数据库视图	可接，仍保留首次审批
草拟	创建草稿 ticket、生成 PR 评论草稿	可接，但要求人发布
写入	改 Linear 状态、写 Notion、更新测试环境数据	按工具逐个审批，最小权限
高风险	生产数据库、支付、云控制台、密码管理器、钱包	默认不接；隔离环境 + 人工确认才考虑

选择	适合	不适合
MCP	反复使用、需要实时外部系统、可定义权限的工具	一次性小材料、来路不明工具
直接上传 / @ 文件	稳定文档、离线资料、可版本化材料	需要实时状态或写回系统
浏览器	需要人工登录、视觉检查、一次性网页	需要可重复自动化
API 脚本	可测试、可审计、固定流程	探索性、多工具、多上下文任务

判断：只接你这周真用得上的两三个，别把 MCP 目录当装机清单。接得越多，agent 越容易在一堆工具里挑错那个、跑得也越慢 —— MCP 让它能做更多，但不是接得越多越强。团队里，这个判断从「配置」升级成「治理」：3.10 起，管理员把 Team MCP 配一次、放进团队 marketplace，就能分发到 cloud agents、Agents 窗口、IDE、CLI —— 成员一键装已审批的集成，不用各自折腾 server 和凭证。[3] 谁来审、放行哪些、给多宽的权限，这些个人版里你替自己做的决定，团队版里得有人明确地替所有人做。

> – 注意
>
> 凭证别写进 mcp.json 的明文里 —— 用 ${env:VAR} 引用，远程服务器走 OAuth。提交前检查 .cursor/mcp.json 像检查一段基础设施配置：它能动你的真实系统，写错一行的代价不只是报错。

- [ ] 来源可信、能读源码或来自官方 marketplace。
- [ ] 凭证最小权限、可撤销、不过期也能轮换。
- [ ] 第一次只开只读工具，跑一次真实任务后再考虑写入。
- [ ] 工具描述清楚，不会让 Agent 误以为能做高风险操作。
- [ ] permissions.json / Run Mode 没有把陌生写入工具全自动放行。

- [ ] 不要接钱包、交易、支付、生产数据库写权限。
- [ ] 不要接能删除云资源或改 IAM / 权限的云控制台工具。
- [ ] 不要接你无法审计源码、权限说明模糊、要求过宽 token 的服务器。

接一个你真用得上的服务器，再删一个你「以防万一」接的：

- [ ] 挑一个你本周真会用的服务器（文档 / 数据库 / Linear 之类），配进 mcp.json。
- [ ] 让 agent 做一件必须用到它的任务，确认它先请求了审批。
- [ ] 检查它给这台服务器开了哪些权限，收到最小；凭证用 ${env:VAR}，不写明文。
- [ ] 删掉一个你接了却没真用过的服务器。

— 引用与参考

参考.

01 Cursor Docs · MCP —— 两条接入路：从 Customize 页 / Marketplace 一键安装（OAuth 直接走完），或用 JSON 手配：项目级 .cursor/mcp.json、全局 ~/.cursor/mcp.json；支持 stdio / SSE / Streamable HTTP 三种传输。服务器暴露 Tools / Prompts / Resources 等；每次调用默认需审批，可用 MCP allowlist 预授权。截至 2026-07-10。Cursor Docs · MCP

02 Model Context Protocol —— MCP 是连接 AI 应用与外部工具 / 数据的开放标准，由 Anthropic 提出，已被多家客户端采用；Cursor 的连接器即基于它。modelcontextprotocol.io

03 Cursor Docs · Plugins / Team marketplaces —— 3.10（2026-06-30）起：管理员把已配置给 Cloud Agents 的 Team MCP servers 放进团队 marketplace，成员即可在 Agents 窗口、IDE、CLI 一键安装已审批的集成，不必各自配 server；加进 marketplace 不等于替所有人安装，管理员仍控制访问与安装模式。截至 2026-07-10。Cursor Docs · Team marketplaces

上下文大多不在项目里，
MCP 是去取它的那只手.

派出去 · Cloud Agents、CLI 与自动化.

有些活不用你守着：交给云端 agent，它在自己的机器上跑，你去忙别的；放进 CLI，它在终端和 CI 里无人值守地跑；设条自动化，它按时间或事件自己触发。Cursor 把同一个 agent 铺到了网页、iOS、Slack、GitHub 上，还给了一套 SDK 让你编排它。这一章讲哪类活值得派出去（包括 Bugbot 那种自动 PR 评审），哪类还是留在你眼前。

有些活不用你守着：交给云端 agent，它在自己的机器上跑，你去忙别的；放进 CLI，它在终端和 CI 里无人值守地跑；设条自动化，它按时间或事件自己触发。Cursor 把同一个 agent 铺到了网页、手机、Slack、GitHub 上。前面几章的 agent 是「陪你做」的，这一章讲「派出去做」的那一面 —— 哪类活值得交出去（包括 Bugbot 那种自动 PR 评审），哪类还是留在你眼前。

— I

Cloud Agents：派出去，它在自己机器上跑.

本地 agent 占着你的机器和注意力；云端 agent 不占。云端 agent（原 Background Agents）各自有一台隔离 VM，可以并行跑好几个，还带 computer use 沙箱去验证自己的产出。[1] 发起入口铺得很开：网页、iOS app、桌面、Slack / Linear 里 @cursor、GitHub / Bitbucket 的 PR 评论里 @cursor，也能把本地或 CLI 上跑着的活 handoff 到云端继续。它克隆你的仓库、在自己机器上改完、交一个 PR 回来。「派出去」也不非得整单外包：3.7 起，本地会话里 /in-cloud 就能把下一个任务丢给一个云端 subagent，它在自己的 VM 和分支上跑，你的本地工作区不被占用；/babysit 则让一个云端 agent 盯住某个 PR，回评论、解冲突、修挂掉的检查，直到它能合并。[1] 手机那头也接上了：iOS app（beta，官方公告称 public beta，各付费档可用）能从手机发起、跟进、直接合并 agent 的 PR；Remote Control 让你人在外面、继续指挥跑在你电脑上的那个 agent；Live Activities 把进度放在锁屏上。[5] 判断不变：手机适合查进度、批结果、给下一步指令，不适合当主工作台。判断：陪你做的活留在眼前，做完通知的活派出去。它适合可并行、定义清楚、跑错能回滚的任

务；不适合探索性的、你自己都没想清楚的改动 —— 那种你不在场，它容易产出一堆你没有心智模型、又得硬着头皮审的代码。它的 computer use 只在自己沙箱里(验证它造的东西)，不碰你本机，这条线第 10 章再收。

提示词

Cloud Agent

```
Cloud Agent 任务:
目标: <做成什么>
仓库/分支: <repo + base branch>
范围: 只改 <paths>
环境: 运行 <setup/test commands>
验收:
- <测试命令必须通过>
- <PR 描述必须列出风险和回滚>
禁止: 生产操作、读取 .env*、改权限、无测试大改。
完成后开 PR，并附日志 / 截图 / artifact。
```

```
Cloud Agent 任务:
目标: 把所有日期格式从 MM/DD/YYYY 统一改成 ISO 8601(YYYY-MM-DD)
仓库/分支: acme/billing-service，base 分支 main
范围: 只改 src/utils/date.ts 和 src/components/**
环境: 运行 npm ci && npm run test:date
验收:
- npm run test:date 必须全绿
- PR 描述必须列出风险和回滚(指出哪些展示位会变)
禁止: 生产操作、读取 .env*、改权限、无测试大改。
完成后开 PR，并附日志 / 截图 / artifact。
```

选择	适合	不适合
本地 Agent	你要边看边改、随时打断、审每一步	长时间占机、可并行的任务
Cloud Agents	明确、可回滚、跑完给 PR 的任务	敏感账号操作、无测试大改、探索性设计
Cursor CLI	脚本、CI、headless review、非 IDE 场景	需要大量交互式 UI 判断的工作
Cursor SDK	把 agent 编成你产品或流水线的部件：CI 修复机器人、批量任务、自定义工具	一次性的活、CLI 一行就能解决的活

• • •

— II

CLI 与自动化：无人值守地跑.

能进脚本的 agent，才能在你睡觉时干活。agent 把 agent 带进终端：`-p` 非交互、`--output-format json` 给结构化输出，可以塞进脚本和 CI / GitHub Actions；消息前加 &，任务就推给云端接着跑。[2] Automations 则让云端 agent 按计划或事件自己触发 —— 事件面很宽：GitHub / GitLab / Bitbucket 的 PR 与评论、CI 跑完、Slack 消息（甚至指定一个表情当扳机）、webhooks、Linear。[3] 建一条也不再需要你会配：3.8 起在本地会话里 /automate，一句话描述要自动化什么，它替你配好触发器、指令和工具；自动化拉起的云端 agent 默认带 computer use，你可以直接在指令里要求它交一段 demo 录屏当作业。[3] 装上很简单，跑一个非交互任务也就一行：

```
curl https://cursor.com/install -fsS | bash

agent -p "把 ./reports 下所有 csv 合并成 summary.csv 并去重" --output-format json
```

提示词

Cursor CLI

```
agent -p "<任务>"
--model <model-or-auto>
--mode agent
--output-format json

约束：
- 只改 <paths>
- 不提交、不 push、不发 PR
- 所有写入先生成 diff
- 输出 JSON 包含 changedFiles、commandsRun、tests、risks。


agent -p "把 src/api 下所有 console.log 换成 logger.debug，并在文件顶部补上
logger 的 import"
--model auto
--mode agent
--output-format json

约束：
- 只改 src/api/**
- 不提交、不 push、不发 PR
- 所有写入先生成 diff
- 输出 JSON 包含 changedFiles、commandsRun、tests、risks。
```

- [ ] CI 里用 `CURSOR_API_KEY`，不要把 token 写进 workflow。
- [ ] 把 git commit / push / PR 评论放在确定性步骤，不一定交给 Agent。
- [ ] 用权限配置限制读写路径和 shell 命令。
- [ ] 失败要有日志、artifact、通知；成功也要有人 review。

边界：无人值守适合可验证、低风险、跑错能回滚的活 —— 定时整理数据、跑一份报告、批量处理文件。改动大、要判断、出错代价高的，别全自动；自动化的前提是「做完算什么样」你已经定死，机器只是去够这个标准。

- [ ] 无人值守禁止：生产部署、数据库迁移、账单 / 支付 / 钱包动作。

- [ ] 无人值守禁止：读取密钥、改 IAM、删除云资源、批量删除用户数据。
- [ ] 无人值守禁止：没有测试命令、没有日志、没有回滚办法的代码大改。

• • •

— III

Bugbot：把机械评审那半交出去.

PR review 里最累的是机械检查，这部分该交出去。[4] Bugbot 在 GitHub / GitLab / Bitbucket 的 PR 上自动评审 diff，留下解释和修复建议，会读已有评论避免重复；效果分 Default / High / Custom 三档，usage-based 计费，也能用 `cursor review` / `bugbot run` 手动触发。[4] 它现在由 Cursor 自家的 Composer 2.5 驱动，官方称平均一次评审约 90 秒；推送前还能在本地 `/review` 先跑一遍 —— 开 PR 时 Bugbot 认得这份已审过的 diff，不再重复评审。[4] 它是「派出去」的一种形态：你不必盯着每个 PR 做那一遍机械扫描。诚实的边界：它抓的是机械问题（bug、安全、规范），不替你判断设计意图和取舍。它替你看「这改得对不对」，你还得看「该不该这么设计」—— 意图层的 review 仍然是你的。把它当成给你省下第一遍粗筛的工具，不是替你拍板的评审者。

> — 实践提示
>
> 判断一件活该不该派出去，问一句：它有没有明确的「做完算什么样」、跑错了能不能回滚。两样都有，派；缺一样，留在眼前自己盯。并行派多个云端 agent 之前先想清楚你审不审得过来 —— 派出去的代码也得有人读。

• • •

— IV

SDK：从用 agent 到编排 agent.

CLI 把 agent 放进脚本，SDK 把 agent 放进程序。@cursor/sdk（TypeScript / Python）让你在自己的代码里调用同一个 agent：local runtime 跑在你的进程里、文件走本地磁盘，cloud runtime 跑在隔离 VM 里，一把 API key 通用。[6] 比 CLI 多出来的是「编排」的那层：把你自己的函数注册成它能调的工具（custom tools）；headless 跑时不必在「全放行」和「跑不动」之间二选一 —— 接 auto-review 分类器，用自然语言写哪些调用放行、哪些拦下；subagent 还能嵌套，审稿的可以再派一个写测试的。官方给的起点很说明定位：CI 自动修复机器人、bug 分诊、code review、嵌进你自己产品里的 agent。连云端 agent 的会话本身都能挂 hook（3.11 起有 prompt 提交前、响应后、subagent 启动等节点），拿来搭自纠错的循环。判断：个人 builder 大多停在 CLI 就够 —— 一行命令、一条 automation，能覆盖绝大多数「派出去」的活。SDK 值得上的时刻只有一个：agent 不再是你手里的工具，而要成为你产品或流水线里的一个部件。到那一步，这一章讲的验收纪律不变，只是从「你审它的 PR」变成「你的程序审它的输出」。把一件长跑或定时的活派出去一次：

01 **挑一件「做完通知我」的活**

一个跨度较长、验收标准清楚的任务，编码或非编码都行（如定时整理一批数据 / 跑一份报告）。

02 **交给 cloud agent，或写成一条 headless CLI 命令**

然后离开 —— 去网页 / Slack 看它的进度，而不是守着它。

03 **回收结果，决定它值不值得变成自动化**

如果它跑得稳、你以后还会重复，就设条 Automation 让它按时间或事件自己跑。

参考.

01 Cursor Docs · Cloud Agents —— 原 Background Agents；每个 agent 一台隔离 VM、装完整开发环境，可随便并行，能在自己的桌面 / 浏览器里验证产出；发起入口：iOS app、cursor.com/agents 网页（Android 装 PWA）、桌面、Slack / Linear @cursor、GitHub / Bitbucket 评论 @cursor、API；本地会话里 /in-cloud 把下一个任务丢给云端 subagent，/babysit 让它盯 PR 到能合并。云端一律跑 Max Mode。截至 2026-07-10。Cursor Docs · Cloud Agents

02 Cursor Docs · CLI —— agent 把 agent 带到终端；-p 非交互，--output-format json / stream-json 结构化输出，可进脚本与 CI / GitHub Actions；支持 Plan / Ask 模式；消息前缀 & 即把任务推给 Cloud Agent 继续跑。截至 2026-07-10。Cursor Docs · CLI

03 Cursor Docs · Automations —— 让 Cloud Agents 按计划或事件运行，事件源含 GitHub、GitLab、Bitbucket、Slack（含表情触发）、webhooks、Linear 等；本地会话里用 /automate 一句话建自动化，Cursor 替你配触发器、指令与工具；computer use 对每条自动化默认开启，可要求它交 demo / artifact；自动化一律以 Max Mode 计费。截至 2026-07-10。Cursor Docs · Automations

04 Cursor Docs · Bugbot —— 在 GitHub / GitLab / Bitbucket 的 PR 上自动评审 diff，留下解释与修复建议；会读已有评论避免重复；效果档 Default / High / Custom；usage-based 计费；可用 cursor review / bugbot run 手动触发，推送前可在本地 /review 先跑。官方 changelog（2026-06-10）称 Bugbot 现由 Composer 2.5 驱动、平均评审约 90 秒。截至 2026-07-10。Cursor Docs · Bugbot

05 Cursor Docs · Cursor for iOS —— iOS app 处于 beta（changelog 称 public beta，2026-06-29 起），各付费档可用：从手机发起、跟进、合并云端 agent 的 PR，语音描述任务；Remote Control 从手机继续指挥跑在你电脑上的 agent（Teams / Enterprise 需管理员开启）；Live Activities 锁屏看进度；iPhone（iOS 26+）、英文界面，Android 计划中。截至 2026-07-10。Cursor Docs · iOS app

06 Cursor Docs · SDK（TypeScript / Python）—— @cursor/sdk 让你在自己的代码里调用 Cursor 的 agent：同一个 agent，local（在你的进程里、文件走本地磁盘）或 cloud（隔离 VM）两种 runtime，一把 CURSOR_API_KEY 通用；可把自己的函数注册成工具（custom tools）、headless 跑时接 auto-review 分类器管权限、subagent 可

嵌套；官方给的起点是 CI 自动修复、bug 分诊、code review、嵌入产品的 agent 与编排器。截至 2026-07-10。Cursor Docs · SDK

陪你做的活留在眼前，
做完通知的活派出去.

干别的 · 调研、写作、数据都在它里面.

把前面几章拼起来，会得出一个官网不会明说的结论：agent + 终端 + 上下文 + MCP，凑成的不是编辑器，是一台什么活都能接的工作台。一个装满资料的文件夹当项目，它能替你调研出一份带引用的笔记、清洗合并一堆 CSV、批量重命名几百个文件、就着 Canvas 写长文。这一章给三个非编码的真实场景，也诚实说清：哪一步它不如一个专用工具。

把前面几章拼起来，会得出一个官网不会明说的结论：agent + 终端 + 上下文 + MCP，凑成的不是编辑器，是一台什么活都能接的工作台。一个装满资料的文件夹当项目，它能替你调研出一份带引用的笔记、清洗合并一堆 CSV、批量改几百个文件、就着 markdown 写长文。这一章给三个非编码的真实场景，也诚实说清：哪一步它不如一个专用工具。这是整本书的落点 —— 如果元工具论点只在一处兑现，就是这里。

非代码任务	留在 Cursor 的信号	切走的信号
技术调研 / 写笔记	资料能文件化、来源可追溯、输出是 markdown	需要对抗式多源查证和深度网页研究
写作项目	有资料、风格指南、版本控制，重点是结构和改写	需要心流、精细排版、协作文档批注
数据清洗	规则清楚、可 dry-run、输出可 diff	需要交互式可视化、统计建模、数据库级规模
文件批处理	路径明确、可备份、能脚本化	只能通过 GUI / 登录账号操作
Prompt / rules / specs 管理	文件化、可 review、可复用	需要产品管理系统或知识库权限流

— I

调研：搜、取、写成带引用的笔记.

它能查、能取、能写，三步在一个窗口里闭合。用 agent 的网页搜索拉来源、用 MCP 取你自己系统里的数据、用 @Docs 把一份文档站索引进来，[1] 然后让它边读边把结论写进一份 markdown 笔记、每条带出处，资料就在你这个文件夹里、随时可复查。[2] 它和聊天框查资料的区别在于：结论落进了文件，不是停在一段对话里、关掉就没了。同一套机制也让它当「第二大脑」：把多年的笔记建成索引，按意思找回那条你早忘了文件名的记录。诚实的边界：要十几个来源交叉核对、对抗式查证的深度报告，一个专门的 deep-research 工具走得更远。Cursor 的甜区是「就着你已有的资料 + 几次搜索」产出可用、可复查的笔记，不是替代严肃的多源调研。要它出报告，先把来源喂准，别指望它替你决定该信谁。

提示词

Cursor Agent

就 `<主题>` 查证并记笔记：
- 先用网页搜索找 3–5 个一手来源
- 每条结论后用方括号标出处链接
- 拿不准的标「待核」，别编
写进 notes/`<主题>`.md。

就「欧盟 AI 法案对开源模型的豁免」查证并记笔记：
- 先用网页搜索找 3–5 个一手来源（法案原文、官方问答优先）
- 每条结论后用方括号标出处链接
- 拿不准的标「待核」，别编
写进 notes/eu-ai-act-open-source.md。

• • •

— II

数据与文件：在终端里清、合、批处理.

你描述要什么结果，它写脚本、跑脚本、读输出再修。合并去重一堆 CSV、按规则批量重命名或转换几百个文件、从一批文档里抽字段做成一张表 —— 这些都在集成终端里用 python / pandas / shell 跑，你不必会这些命令。第 4 章讲过这条入口；这里是它最省事的用法：你说「把这二十个表按日期列合并、去掉重复行、空值填 0」，它写好脚本、跑给你看、有问题接着改，你审最后那张表就行。诚实的边界：要反复看图、调参的交互式数据探索，一个 notebook 更直接 —— Cursor 适合「一次性、规则清楚」的批处理，不适合「边看边想下一步」的探索。还有一条：它跑的是脚本，处理得了结构化数据和文件，处理不了「只能在某个 app 的界面里点」的数据 —— 那又回到 Computer Use 那条线。

提示词

Cursor Agent + Terminal

```
数据清洗 dry-run:
目标: <清洗后要得到什么>
输入: <raw files>
输出: 先写到 data/working, 不覆盖 raw
步骤:
1. 先列文件和字段概览
2. 写脚本但先 dry-run, 输出将改动的行数 / 样例
3. 我确认后再生成 data/output/ <file>
4. 输出校验: 行数、空值、重复值、字段类型
5. 最后给回滚办法。
```

```
数据清洗 dry-run:
目标: 把 12 个月的销售表合并成一张, 按订单号去重, 金额列空值填 0
输入: data/raw/sales-2025-*.csv
输出: 先写到 data/working, 不覆盖 raw
步骤:
1. 先列文件和字段概览
2. 写脚本但先 dry-run, 输出将改动的行数 / 样例
3. 我确认后再生成 data/output/sales-2025-merged.csv
4. 输出校验: 行数、空值、重复值、字段类型
5. 最后给回滚办法。
```

• • •

— III

写作: 在它里面写, 但知道何时切走.

它适合「就着资料结构化地写」, 不适合「需要心流的纯散文」。一套真实可跑的写作流: 文稿用 git 管的 markdown, 把你的风格指南和几篇旧文用 @Files 喂进去定调, 初稿让 agent 起、你用 Cmd+K 逐段收(「这段压缩 15%、换成主动语态、一句一个意思」), 写完 git push 发布; Canvas 给你一个画布式的排布和分享面。[3] 它强在「资料、索引、改写都在同一处」—— 你不必在浏览器、文档、笔记之间来回搬。

> **提示词** Cmd+K

> 就选中这段，按我的风格改：
> - 压到原长度的 85%
> - 主动语态，一句一个意思
> - 删填充词和套话
> - 术语和事实别动
>
> 只给改后的文本。
>
> 就选中这段产品发布说明，按我的风格改：
> - 压到原长度的 85%
> - 主动语态，一句一个意思
> - 删「赋能」「打造闭环」这类套话
> - 版本号和日期别动
>
> 只给改后的文本。

带引用的学术 / 长报告写作也吃这套：它长于改写、重排、套期刊或基金的格式，配合 Zotero、Pandoc 把引用和导出接上。一个真实缺口要先说 —— 它对 PDF 的原生支持有限，源文献多是 PDF 时，得先转成文本或 markdown 再 @Docs 拉进来，这也是社区最常提的不足。诚实的边界：要心流的纯写作，一个专门的写作软件更顺 —— Cursor 是编辑器底子，长在「改」不长在「流」。而且最关键的一条：它写完不能替你「点」发布到你的 CMS、不能替你登录后台贴上去 —— 那是 Computer Use（驱动你真实的 app），是另一类工具（Claude in Chrome / Operator）的事，下一章那条线。[4] 它能把资料写成稿，替你点「发布」不归它管。

> **— 实践提示**
>
> 非编码用法的共同形状：把一个文件夹当项目，让 agent 用「索引 + 终端 + MCP + 网页」围着你的真实资料干活。它强在「资料和动手在同一处」，弱在「需要专门交互界面」的活（画图、纯写作流、GUI 操作）。记住这条，你就知道一件活该留在 Cursor 还是切走。

看到这个信号	切到
需要浏览器登录、跨站点击、账号确认	浏览器 / Computer Use / 手动
要十几个来源互证、追踪新闻、判断可靠性	专用 Research 工具 + 人工核查
数据大到不能用文件夹脚本舒服处理	Notebook / database / BI
要精细版式、评论修订、多人实时协作	文档 / 设计 / 排版工具
长期 shell 流程、跨项目命令、系统层操作	Warp / cron / scripts

这一题是全书的试金石 —— 挑一件你平时绝不会想到用编辑器做的事：

01 **选一件非编码的真实活**

整理一批文件、清洗合并一堆 CSV、或调研一个主题并产出带引用的笔记 —— 选你这周真要做的那件。

02 **全程在 Cursor 的 agent + 终端 + MCP 里完成，不开别的 app**

把资料文件夹当项目打开，描述结果，让它搜、取、跑、写。

03 **记下三件事：哪些顺手、哪一步你不得不切走、为什么**

那「不得不切走」的一步，就是 Cursor 这台元工具的真实边界 —— 也是你下一章要对照的那条线。

参考.

01 Cursor Docs · Agent —— agent 的工具里含网页搜索、图像生成、终端执行、跨文件编辑、浏览器控制、MCP，这些不绑定「写代码」，构成它做非编码活的基础。截至 2026-07-10。Cursor Docs · Agent

02 Cursor Docs · Codebase Indexing —— 打开任何项目（文件夹）即自动建索引、可按语义检索，非代码资料同样适用；@Docs 可把外部文档站索引进来。截至 2026-07-10。Cursor Docs · Codebase Indexing

03 Cursor Docs · Canvases / Design Mode —— Canvas 让 agent 生成可交互的 artifact（仪表盘、分析、报告）在聊天旁渲染、可复开可迭代；Design Mode 在浏览器里点选元素、画标注、语音描述来改 UI。截至 2026-07-10。Cursor Docs · Canvas

04 Cursor Blog · Agents can now control their own computers（2026-02-24）—— Cursor 的 computer use 限于云端 agent 的沙箱，划出「项目内 / 沙箱」与「驱动你真实 app」之间那条边界。Cursor Blog · Agent computer use

---

它能干的远不止代码，
但不是替你点别的 app.

选型 · Pro / Pro+ / Ultra、同类与边界.

选档不是看价格，是看你撞的是哪堵墙：额度、模型、Cloud Agents、Bugbot，还是团队管控。这一章把 Pro、Pro+、Ultra、Teams、Enterprise 和计费方式（Auto + Composer 池 vs 前沿模型 API 池）摆开，再划三条线：Cursor 和 Claude Code / Codex / Copilot / Windsurf / Warp 的分工，个人 builder 的组合，以及最该记住的那条 —— Cursor 的 Computer Use boundary 不是通用电脑控制。

选档不是看价格，是看你撞的是哪堵墙：额度、模型、Cloud Agents、Bugbot，还是团队管控。这一章把 Pro、Pro+、Ultra、Teams、Enterprise 和计费方式摆开，再划三条线：Cursor 与 Claude Code / Codex / Copilot / Windsurf / Warp 的分工、个人 builder 的工具组合、以及最该记住的那条 —— Cursor 的 Computer Use boundary 不是通用电脑控制。这是全书的地图章：它在哪儿停、该交给谁接。

— I

先认清你撞的是哪堵墙.

每一档解开的不是笼统的「更高级」，是一堵具体的墙。

档位	价格(截至 2026-07-10)	它解开的那堵墙
Hobby	免费	够试用，Agent 与 Tab 很快见底
Pro	$20/月	全部功能 + $20 API 用量
Pro+	$60/月	更高 API 用量，适合日常 Agent 用户
Ultra	$200/月	最高个人用量，适合多 agent / 自动化重度用户
Teams Standard	$40/座·月	协作 + 管控：SSO、admin、团队市场、共享上下文
Teams Premium	$120/座·月	Teams Standard 的 5× Agent 用量
Enterprise	定制	池化用量、SCIM、访问控制、审计、模型 / MCP / 网络管控

计费的关键不在档位，在两套额度池：Auto、Composer 2.5、Grok 4.5 走「第一方模型池」(宽得多，但**不是无限**)，手动选 Claude / GPT / Gemini 这些前沿模型走 API 那套(烧得快)；额度用完转 on-demand，按模型 API 价走、质量速度不降；Max 模式按 token 算。[1][2] 所以判断很简单：大多数活让 Auto 跑，Pro 的池子就够；天天手动用前沿模型、或开 Max 处理大上下文，才升 Pro+ / Ultra；需要 SSO、admin、团队共享上下文，才上 Teams。Pro 到 Ultra 解开的是额度，不是能力 —— 功能 Pro 就全给了。别为「更高级」付钱，为你真撞的那堵墙付钱。

自测问题	如果是	如果不是
你每周是否多次撞到 included usage / on-demand?	看 Pro+ / Ultra	留在 Pro
你是否每天跑 Agent，多次开 Max 或指定前沿模型?	Pro+ 起步，重度看 Ultra	Auto + Composer 优先
你是否需要多人账单、SSO、隐私模式统一执行?	Teams	个人档即可
你是否需要 SCIM、池化用量、模型/MCP/网络限制、审计?	Enterprise	不要用 Enterprise 解决个人额度焦虑

- [ ] 你还没有稳定使用 Agent，只是 Tab / Cmd+K 多：不需要升级。
- [ ] 你多数任务能用 Auto / Composer 跑完：不需要升级。
- [ ] 你只是想偶尔试强模型：先用 on-demand 看账单。
- [ ] 你没有 review 产能审多个并行 agent：不要因为 Cloud Agents 升级。

- [ ] 你每天让 Agent 做多文件任务，且经常被额度打断。
- [ ] 你频繁用 Max / 前沿模型处理大上下文、复杂重构、长链路 debug。
- [ ] 你并行派 Cloud Agents，有稳定测试和 PR review 流程。
- [ ] 你把 CLI / automations 放进真实工作流，而不是偶尔玩一次。

所以「什么烧钱」值得记牢：同一个任务，agent 在内部要多次调模型（不是一问一答），一个 agent 任务因此比一句 chat 贵得多；长上下文也贵 —— 加价规则按模型而异（有的超过 200k 输入按 2× 计，有的到 1M 明确不加价），开 Max 前看一眼模型表；Max 按模型 API 价走、烧得最快，而 Cloud Agents 和 Automations 一律跑在 Max 上、没有关闭开关。[5] 反过来，Tab 补全在各付费档不限量、根本不吃这套额度池。[2]

> — 实践提示
>
> Max 的加价是历史遗留：当前个人档 Max 按模型 API 价计、没有额外加价，只有旧的 request-based 档才加 20%。省钱第一招不是降档，是写准 prompt、把任务拆小、别让一个会话拖太长 —— 烧的大头是 agent 的来回，不是档位。

• • •

— II

同类与配对：取舍，不踩.

它们不是谁好谁坏，是各自压在不同的轴上。给取舍轴，不拉踩：Cursor 压在「编辑器集成 + 索引 + Agent + MCP + Cloud Agents」这一轴上；Claude Code 压在 terminal-native / Claude-native 的工程闭环上；Codex 压在 OpenAI / ChatGPT / cloud tasks / PR review 生态上；Copilot 压在 GitHub / VS Code / Microsoft 企业生态上；Windsurf 和 Cursor 同在 IDE + agent 赛道；Warp 则是 AI terminal / shell 工作台。

工具	强项	什么时候选它
Cursor	IDE-native：代码、索引、终端、MCP、Agent、Cloud Agents 在一个工作台，外加 Marketplace 与 SDK	项目上下文和可审 diff 最重要
Claude Code	terminal-native / Claude-native 工程闭环	你主要在 shell 里工作，喜欢深度终端 agent 流
OpenAI Codex	OpenAI / ChatGPT / cloud task / PR review 生态	你已经把工作流压在 ChatGPT、OpenAI API 或 Codex cloud tasks 上
GitHub Copilot	GitHub / VS Code / Microsoft enterprise 生态	你要轻量补全、PR 内联、企业采购顺滑
Windsurf	同类 IDE + agent，产品取向更面向新手路径	你更喜欢它的交互节奏或团队已有席位
Warp	AI terminal / shell 工作台	工作主要是命令流、运维、跨项目 shell，而不是代码上下文

判断：要一个把项目上下文、编辑、终端、review 放在同一窗口的主力环境，Cursor；活主要在 shell 和命令流，Claude Code 或 Warp 更顺；工作流已经压在 GitHub / Microsoft，Copilot 管理成本更低；已经把任务分发、评审、云端修复压在 OpenAI 生态，Codex 更自然。这半年 Cursor 的轴在变宽 —— Marketplace 分发用法、SDK 编排 agent、手机接管云端 —— 它在从工作台往平台长；但选型的问题一个字没变：这件活的上下文、执行面和验收面在哪里。不要问谁替代谁。

> — 补充
>
> 个人 builder 的推荐组合很朴素：Cursor 当项目工作台；Claude Code / Codex 负责你更信任的特定工程闭环或云端任务；Warp 负责独立终端流；Research 工具负责深度查证。不要逼一个工具吞掉所有节奏。

> — 补充
>
> 配对那一半：Cursor 是编辑器侧的元工具，Warp 是终端侧的元工具。活主要在代码和项目里 —— Cursor 当主力；活主要在 shell、运维、跨命令的流程里 —— Warp 更顺。两个都开不冲突，但别让它们重复干同一件事。（Warp 这本另说，这里只画分工。）

• • •

— III

那条 Computer Use 线.

Cursor 的 agent 能「用电脑」，但用的是它自己沙箱里那台，不是你的。把线画死：Cursor 的 computer use 在云端 agent 的隔离 VM 里发生，用来点开它自己造的软件、验证它改的东西；企业版的 browser / network 控制也是管这个沙箱。[3] 3.8 起自动化拉起的云端 agent 默认就带这个能力、给你出 demo —— 还是同一条线：它点的是自己 VM 里的桌面。它不操作你本机登录着的 app；连 iOS 的 Remote Control 也是反着的 —— 那是你从手机指挥你电脑上的 agent，不是它操控你的手机。真正「替你点别的 GUI app」—— 用你的账号发邮件、在你的浏览器里跨站操作、动你打开着的设计稿 —— 是另一类工具：Claude in Chrome、Operator、Claude 的 computer use。所以这台元工具的边界是清楚的：项目里、沙箱里、命令行说得清的活，Cursor 替你做完；要驱动你真实的、登录着的 app，切到那一类工具。诚实补一句：这条线在移动，今天清楚、明天可能模糊，每次引用前重新核对官网。

任务	Cursor 能不能做	边界
改项目文件、跑测试、开 PR	能	需要 Git / 测试 / review 兜底
Cloud Agent 在 VM 里打开 app 点击验证	能	只在它自己的远程环境
用你的浏览器账号发邮件 / 改表单 / 操作后台	不要写成 Cursor 能力	切到浏览器 agent / Computer Use / 手动
控制桌面 app、设计软件、聊天软件	不是 Cursor Desktop 的官方通用能力	按账号、权限、审计风险另选工具

SpaceX / Anysphere 这类商业新闻，只能放在背景里：截至 2026-07-10，SEC 层面仍是 6 月那份 8-K —— SpaceX 与 Anysphere 签署 merger agreement，交易还受成交条件和监管批准约束，预计 2026 Q3 完成。[4] 产品侧倒是先看到了合作的痕迹：模型文档把 SpaceXAI 列为供应方，Grok 4.5 写明「由 Cursor 与 SpaceXAI 联合训练」、进了第一方池。[2] 这说明 Cursor 已经进入基础设施级竞争，不说明今天的 Cursor 多了某个功能，也不说明你该因为收购新闻升级。

提示词 Subscription self-check

```
Cursor 订阅自测：
1. 我上个月最常撞的墙是：<额度 / 模型 / Cloud Agents / 团队管控 / 没撞>
2. 我每周 Agent 任务数约：<n>
3. 我是否经常手动选前沿模型或 Max：<是/否>
4. 我是否有 review 多个并行 agent 的产能：<是/否>
5. 我是否需要 SSO / SCIM / 审计 / 模型管控：<是/否>
结论：<Hobby / Pro / Pro+ / Ultra / Teams / Enterprise / 不升级>，理由：<一句话>。
```

```
Cursor 订阅自测：
1. 我上个月最常撞的墙是：额度(每周三四次转 on-demand)
2. 我每周 Agent 任务数约：25
3. 我是否经常手动选前沿模型或 Max：否，基本让 Auto 跑
4. 我是否有 review 多个并行 agent 的产能：否，一次只盯一个
5. 我是否需要 SSO / SCIM / 审计 / 模型管控：否，个人单干
结论：Pro+，理由：天天用 Agent 又总被额度打断，但没到要并行多 agent 的程度。
```

把这章用到你自己的订阅和工作流上：

- [ ] 写下你上个月撞的墙：额度、模型，还是根本没撞。
- [ ] 判断你是 Auto 为主还是前沿模型为主 —— 据此看 Pro 的池够不够，还是要 Pro+ / Ultra。
- [ ] 决定你是 Cursor 单干、还是 + Warp（终端侧）、还是 + 一个能点你真实 app 的工具。
- [ ] 确认你没在为一档你从没撞到其上限的额度付钱。

参考.

01 Cursor · 套餐与价格 —— Hobby 免费、Pro $20/月、Pro+ $60/月、Ultra $200/月、Teams Standard $40/座·月、Teams Premium $120/座·月(Standard 的 5× Agent 用量)、Enterprise 定制。各档含一份模型用量额度(Pro 含 $20 API 用量、Pro+ $70、Ultra $400)。截至 2026-07-10,Cursor 改价很勤,引用前以官网为准。Cursor · Pricing

02 Cursor Docs · Models & Pricing —— 个人档两套额度池:「第一方模型池」(Auto、Composer 2.5、Grok 4.5 —— Grok 4.5 由 Cursor 与 SpaceXAI 联合训练;included 用量宽得多,但非无限)与 API 池(手动选的前沿模型按该模型 API 价);超出转 on-demand,官方明言「质量与速度不降级」;Max 按 token 的模型 API 价计费(旧 request 制套餐加 20%);长上下文加价按模型而异;付费个人档 Tab 补全不限量。截至 2026-07-10。Cursor Docs · Pricing

03 Cursor Blog · Agents can now control their own computers(2026-02-24)—— Cursor 的 computer use 发生在云端 agent 自己的隔离 VM / 沙箱里,用来验证它自己造的软件,不是控制你本机登录着的 app。Cursor Blog · Agent computer use

04 Space Exploration Technologies Corp. Form 8-K(2026-06-16)—— SpaceX 与 Anysphere, Inc. 签署 merger agreement,交易仍受成交条件和监管批准约束,预计 2026 Q3 完成;截至 2026-07-10 未见完成交割的后续披露。本文只把它作为市场背景,不写成产品能力。SEC · SpaceX Form 8-K

05 Cursor Docs · Cloud Agents —— Cloud Agents 使用一组精选模型、一律跑在 Max Mode,没有关闭开关;Automations 因为跑的就是 cloud agents,同样恒为 Max Mode 计费。截至 2026-07-10。Cursor Docs · Cloud Agents

---

贵的不是高档,
是买了用不上的额度.

收束 · 一台机器的活，一个窗口干完.

把整本书压成一句：Cursor 的问题从来不是「它能不能写这段代码」，是「这台电脑上的活，有多少能在这一个窗口里干完」。编辑器是它最小的样子；agent、终端、上下文、MCP 把它撑成一台元工具。剩下的，是知道那条边界画在哪 —— 然后尽量在一个窗口里把活做完。

把整本书压成一句：Cursor 的问题从来不是「它能不能写这段代码」，是「这台电脑上的活，有多少能在这一个窗口里干完」。编辑器是它最小的样子；agent、终端、上下文、MCP 把它撑成一台元工具。剩下的，是知道那条边界画在哪 —— 然后尽量在一个窗口里把活做完。

– I

编辑器是地板，不是天花板.

你停在的那层，是它的起点，不是它的上限。回扣一遍：Tab / Cmd+K / Chat 是底座；Agent 是发动机，终端让它动手，索引和 @ 让它读懂你的项目，Rules 钉住偏好，MCP 把外部系统接进来。[1] 这几样叠起来，才是你多付那笔钱买到的东西 —— 大多数人买了一台元工具，只用了它的编辑器。

• • •

— II

选面，就是选节奏.

同一个 agent，在编辑器里陪你做，在云端 / CLI 里替你跑完，在 Warp 里贴着终端干。把分工压成节奏：要陪着做、要审每一步，留在编辑器里；做完通知我，派给 cloud agent / CLI / 自动化；活在终端，切 Warp。问题不是「Cursor 能不能」，是「这件事该在哪个面、哪种节奏里做」—— 选对入口，比追更高的档位重要。

• • •

— III

边界让元工具可信.

知道它在哪儿停，比知道它能做什么更值钱。两条边界收尾：一是有些活专用工具更强 —— 深度多源调研、交互式数据、要心流的纯写作；二是替你点你真实的 app 是另一类工具（Computer Use）。[2] 承认这两条，「在一个窗口里干完」才是判断，不是口号。一个肯说自己边界的工具，比一个号称万能的工具可信。

30 天	只做这件事	留下什么资产
第 1 周	把 3 类高频任务留在 Cursor：小 bug、文档整理、文件批处理	一份留/切走清单
第 2 周	写 AGENTS.md 和 2 条 .cursor/rules	项目工作台说明书
第 3 周	接一个只读 MCP，做一次真实任务	MCP 白名单和风险记录
第 4 周	派出一个 Cloud Agent 或写一条 CLI automation	验收模板、日志、回滚办法

Starter workflow	步骤	成功标准
小型 bug 修复	Ask 定位 → Agent 写测试和修复 → terminal 跑测试 → 你 review diff	新增测试能失败再通过，diff 只碰相关文件
技术调研 + 写作	@Docs / @files 读资料 → 生成资料地图 → 写 markdown 草稿 → Cmd+K 逐段收	每条事实可追来源，稿件可 git diff
数据清洗 + 批处理	raw 只读 → dry-run 到 working → 确认 → 输出到 output → 校验行数/重复/空值	原始数据未覆盖，脚本可复跑，结果可回滚

留在 Cursor	切走
任务能文件化、可追踪、可 review	任务主要是登录账号和 GUI 点击
上下文在代码库、资料文件夹、终端输出、MCP 里	上下文在浏览器会话、设计软件、实时协作文档里
验收能写成测试、diff、清单、报告	验收主要靠视觉审美、业务拍板、多人审批
出错能回滚	出错会影响生产、付款、权限、真实用户

提示词 Stay or switch

```
这件任务该留在 Cursor 还是切走？
任务：<一句话>
上下文在哪里：<repo/files/terminal/MCP/browser/app>
执行方式：<edit/shell/MCP/cloud/GUI>
验收方式：<test/diff/checklist/human approval>
风险：<可回滚/不可回滚/敏感账号/生产>
结论：<留在 Cursor / 切到专用工具>，理由：<一句话>。
```

```
这件任务该留在 Cursor 还是切走？
任务：把官网的活动 banner 文案改成春促版，并发布上线
上下文在哪里：文案在 repo 里，但发布要登录 CMS 后台点「发布」
执行方式：改文件用 edit，上线要 GUI 点击
验收方式：文案改动可 diff，上线后要人工在浏览器里目检
风险：不可回滚（直接影响生产首页），动的是登录着的后台
结论：切到专用工具，理由：改文案留在 Cursor，但「点发布」得人工或浏览器 agent，不归 Cursor。
```

回到第 1 章你列的那张表，现在重新打分：

- [ ] 逐行重标：这件活现在你会留在 Cursor，还是切走。
- [ ] 数一数有几行从「切走」变成了「留在一个窗口里」。
- [ ] 圈出那一两件确实属于别处的活，写清为什么 —— 那才是你判断力的证据。

参考.

01 Cursor Docs · Agent —— agent + 终端 + 索引 + MCP 是 Cursor 作为「元工具」的核心组合：读项目、跑命令、调外部工具，自我迭代到完成。截至 2026-07-10。Cursor Docs · Agent

02 Cursor Blog · Agents can now control their own computers（2026-02-24）—— Cursor 的 computer use 限于云端 agent 的沙箱，划出了「项目内 / 沙箱」与「驱动你真实 app」之间那条边界。Cursor Blog · Agent computer use

---

问的不是它能不能，
是这活该不该留在一个窗口里.

CURSOR

不只是编码 agent.

官网卖的是 AI 编辑器，它其实是一台元工具 —— 代码之外的调研、写作、数据，也能在一个窗口里干完。

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	1.7 万字	2026.07	PDF 1.0