2026年4月 · 第1版

Obsidian + Claude Code
用AI重建你的第二大脑

从零搭建AI驱动的个人知识管理系统

Obsidian + Claude Code — Rebuild Your Second Brain with AI

信息来源：Obsidian官方文档 · kepano/obsidian-skills · Karpathy LLM Wiki · 实践验证

文档版本：v1.0.0

发布时间：2026-04-11 (build #1)

花叔

花叔

> 知识星球「AI编程：从入门到精通」专属内容

本手册基于Obsidian官方文档、kepano（Obsidian CEO）的obsidian-skills项目、Andrej Karpathy的LLM Wiki实践及花叔个人知识管理经验编写。AI工具迭代极快，请结合官方文档验证。

本手册持续更新，获取最新版本请关注公众号「花叔」。

目录

Table of Contents

自序 我的笔记从来不是我整理的

Preface

写这本书之前，我犹豫了一下。因为我自己还没用过Obsidian。

这听起来很荒谬。但你看完这个自序就会明白，为什么一个没用过Obsidian的人反而可能是最合适的作者。

先说一个数字：我现在的「写作」文件夹里有超过2000个Markdown文件。公众号文章、视频脚本、调研笔记、橙皮书手稿、培训课件、读书摘录、Prompt模板，全在一个Git仓库里。每天有十几个文件被创建或修改。

这些文件，我几乎从不手动整理。

不是因为我懒（好吧，也有一点），而是因为我有一个全职的「知识管理员」：Claude Code。它帮我分类、建索引、加标签、归档旧项目。我只负责往里丢东西，它负责让这些东西能被找到。

这套系统不是设计出来的，是长出来的。

从一个CLAUDE.md开始

2025年初，我开始全职用Claude Code做内容创作。当时写作文件夹里就几个散落的Markdown文件，没什么结构。Claude Code每次启动都像一个空降的新同事，什么都要重新介绍一遍。

于是我写了第一版CLAUDE.md，就几行字：我是谁，我在做什么，文件夹里有什么。

效果立竿见影。Claude Code不再问「你想把文章存在哪」之类的蠢问题了。它知道文章存在01-公众号写作/，视频脚本存在03-视频创作/，调研笔记存在07-调研与分析/。

然后我就上瘾了。每次Claude Code犯一个错——比如把草稿存错了地方，比如用了我讨厌的破折号，比如忘了搜索最新信息就开始写——我就在CLAUDE.md里加一条规则。半年下来，CLAUDE.md从几行字长成了一个路由系统：根目录的CLAUDE.md根据关键词把任务分发到不同的子目录，每个子目录有自己的CLAUDE.md定义具体规则。

到现在，我有9个CLAUDE.md文件，分布在不同的工作区。Claude Code进入任何一个目录，都知道在这里该怎么干活。

55个Skills

CLAUDE.md解决了「规则」的问题，Skills解决了「能力」的问题。

Skills是Claude Code的能力扩展机制，你可以把常用的工作流封装成一个可复用的技能。我现在有55个Skills，大致分三类：

第一类是人物视角。我把二十多个对我影响很大的思想者（费曼、芒格、纳瓦尔、塔勒布、Karpathy、Paul Graham）的思维方式和表达风格蒸馏成Skills。写文章卡住的时候，调一个perspective skill，用那个人的视角过一遍我的论点，往往能找到新角度。这个项目叫「女娲」，在GitHub上有6000多颗星。

第二类是内容创作。从选题、调研、写初稿、三审三校、配图、排版，到发布公众号、转X内容、生成小红书图文，每个环节都有对应的skill。我写一篇公众号文章的完整流程，大约要调用七八个skills。

第三类是工具类。PDF生成、视频素材分析、弹幕生成、飞书文档操作、图片上传图床。这些是日常运营的基础设施。

这就是我整个内容运营的「操作系统」。一个人做自媒体，靠这套系统撑起来。

拼接方案的天花板

但我必须承认，这套系统有一个问题：它是拼接出来的。

我的日常工作环境是Cursor的终端。左边是文件树，用来定位自己在知识库的哪个角落；右边是终端，Claude Code在里面跑。两个工具拼在一起，勉强能用。

但这个拼接方案有几个明显的短板。

第一，没有双向链接。我的调研笔记里提到了Karpathy，但我另一篇文章里也引用了Karpathy的观点，这两条笔记之间没有任何关联。除非我用Claude Code手动grep搜索「Karpathy」，否则这些散落的碎片永远不会自动连接。

第二，没有图谱。我知识库里2000多个文件之间的关系，只存在于我的脑子里和CLAUDE.md的文字描述里。如果有一天我忘了某个主题之前写过什么，只能靠搜索碰运气。

第三，导航效率低。在Cursor的文件树里找一个文件，要不停展开文件夹、滚动、点击。没有快速跳转，没有模糊搜索，没有收藏夹。

这些问题，Obsidian恰好都能解决。

[[双向链接]]让知识自动关联。图谱视图让我看到2000个文件之间的关系。Cmd+O模糊搜索，打几个字就跳到任何一条笔记。

所以这本书，其实记录的是我自己的一次迁移。从「Cursor文件树 + Claude Code + 裸Markdown文件夹」的拼接方案，迁移到「Obsidian + Claude Code」的原生方案。

你不需要成为笔记达人

在研究Obsidian社区的时候，我发现一个有趣的现象：很多重度Obsidian用户花了大量时间设计精美的模板、复杂的标签系统、层层嵌套的文件夹结构。他们的vault像一件艺术品。

但他们的痛苦和不用Obsidian的人一模一样：维护这套系统本身就成了一项沉重的工作。标签体系建了又塌，模板改了又改，笔记间的链接总是不完整。

AI改变了这个游戏。

你不需要成为一个自律的笔记达人。你不需要维护完美的标签系统。你甚至不需要手动建双向链接。你只需要做一件事：**把你的知识存成AI能读懂的格式，然后让AI来做剩下的事。**

Obsidian提供了最好的容器，本地的Markdown文件夹。Claude Code提供了最好的管理员，一个能读、能写、能搜索、能关联的AI Agent。这两者的组合，就是2026年个人知识管理的最优解。

这本书不会教你怎么成为Obsidian的高级用户。市面上已经有太多Obsidian教程了。这本书教你的是：**怎么搭建一个AI能够理解和操作的知识系统，然后让AI帮你把它越养越好。**

如果你觉得自己不是那种「每天整理笔记」的人，那恰好，这本书就是写给你的。

花叔

2026年4月

于深圳

§01 信息坟场

The Information Graveyard

你的笔记库像一个管理员失踪的图书馆。书全在，但没人知道哪本在哪。AI就是那个管理员。

你的笔记软件里有多少僵尸

回忆一下你上次打开笔记软件是什么时候。不是打开来记东西——那个动作人人都会——而是打开来找东西、复用东西。

大概率你找不到。

你可能有一个印象笔记账号，里面躺着2018年收藏的文章，你已经忘了为什么收藏。你可能有一个Notion工作区，里面有几十个「待整理」的数据库，从来没整理过。你可能有一个备忘录，里面塞满了随手记的想法，但这些想法之间没有任何关联。

这些笔记不是知识。它们是信息的坟场。

不是你不努力。每一条笔记在记录的那一刻都是有价值的：一个好想法、一段有用的信息、一个值得回顾的经验。但从「记下来」到「用起来」之间，有一道巨大的鸿沟：整理、关联、检索、复用。

这道鸿沟，人类天生不擅长跨越。

三个做不到

人类的大脑是一个极好的联想机器，但极差的索引系统。

做不到持续整理。标签系统很美好，但你坚持不了三个月。建标签需要纪律，而纪律是一种消耗品。第一周你给每条笔记都打上精心设计的标签；第三周你开始偷懒，直接扔进一个叫「待整理」的文件夹；第三个月「待整理」已经有200条笔记了，你再也不想打开它。

做不到全局关联。你上个月读的一本书和你今天遇到的一个问题之间可能有深刻的关联，但你想不到。因为人脑的工作记忆只能同时处理大约7个信息块。当你的知识库有几百上千条笔记时，你根本不可能记住每一条都说了什么。

做不到高效检索。你记得你「好像在哪里看过一个关于某某的分析」，但你不记得具体是在哪个文件夹、哪条笔记、用了什么关键词。搜索引擎能帮你在互联网上找东西，但在你自己的笔记库里，搜索质量取决于你当初记录时用了什么词。

这三个做不到叠加在一起，结果就是：**你的知识库越大，它的有效利用率越低。**

最讽刺的情况是：你明明记过一个很有用的东西，但因为找不到，你去网上重新搜了一遍。你的笔记软件变成了一个只进不出的黑洞。

工具们试过什么

笔记工具一直在试图解决这个问题。每一代方案都比上一代好一点，但都没解决根本矛盾。

文件夹是最早的方案。把笔记按主题分到不同的文件夹里。问题是：一条笔记经常属于多个主题。你读的一篇关于「AI在医疗领域的应用」的文章，到底放「AI」文件夹还是「医疗」文件夹？无论放哪个，你在另一个文件夹里都找不到它。

标签是对文件夹的修正。一条笔记可以打多个标签，不受文件夹的限制。但标签系统的维护成本极高。你需要记住你用过哪些标签、各个标签代表什么含义、新笔记该用哪些标签。标签体系不一致（「AI」vs「人工智能」vs「机器学习」）比没有标签更糟糕。

双向链接是2020年Roam Research带火的。你在A笔记里提到B，B自动知道「A提过我」。比文件夹和标签都好，因为关联是写的时候自然产生的。

但双向链接有一个前提：你得记得去链接。你写日记提到「今天读了芒格的演讲」，如果你不主动输入[[芒格]]，这条日记和你之前写的芒格读书笔记之间就没有关系。链接的质量完全取决于你的勤勉程度。

文件夹、标签、双链，这三种方案的共同问题是：**它们都把整理的责任放在了人身上。**

AI改变了什么

人不擅长做索引和关联，AI擅长。人擅长产生想法和做判断，AI不擅长。那就让各自做各自擅长的事。

人负责：记录、思考、决策。

AI负责：整理、关联、检索、维护。

这已经在发生了。

Stefan Imhoff是一个德国开发者，他用Obsidian记日记。每天的日记里会提到人名、地名、书名。以前他每天要花10-15分钟手动给这些名字添加双向链接：搜索vault里有没有对应的笔记，没有就新建，然后把[[链接]]插入日记。

后来他用了Claude Code。

Claude Code读取他的日记，自动识别出所有的人名、地名、书名。搜索vault里是否已有对应的实体笔记，没有就新建一个。然后把[[wikilinks]]插回日记原文。整个过程几秒钟。10-15分钟的手工活，变成了一条命令。

另一个案例。MakeUseOf的一位作者，积累了5年的笔记，散落在各种文件夹里，没有标签，没有链接，一团乱麻。他试过好几次手动整理，每次打开那个文件夹就感到绝望，然后关上。

他用Claude Code + Obsidian，90分钟整理完了。

Claude Code读取所有笔记，按主题自动分类，给每条笔记添加frontmatter（标题、标签、摘要），建立笔记间的双向链接，生成索引文件。5年的积累，从混乱变得有序，耗时不到两小时。

还有一个案例更有意思。一个产品经理分享了他用Obsidian + Claude Code的四个日常工作流：调研新领域时让AI自动汇总笔记、做产品决策前让AI搜索vault里的历史讨论、写需求文档时让AI基于已有知识生成初稿、每

周回顾时让AI从日记中提炼洞察。

注意这四个工作流的共同点：**人在做决策，AI在做整理和检索**。各自做各自擅长的事。

但AI不是魔法

那把笔记丢给ChatGPT不就行了？

没那么简单。AI能帮你整理笔记，前提是它能碰到你的笔记。

如果你的笔记存在Apple Notes里，AI摸不到。它是一个封闭的系统，没有对外的文件接口。

如果你的笔记存在Notion里，AI只能通过API间接访问，权限受限、速度受限、操作受限。你不能让AI自由地读写和搜索。

如果你的笔记存在印象笔记里，情况更糟。格式私有，导出困难，AI要先做格式转换才能理解内容。

如果你的笔记存在一个文件夹里，全是Markdown格式，那AI可以直接读、直接写、直接搜索、直接重新组织。零摩擦。

这就是为什么笔记格式很重要。**你选择什么格式存储你的知识，决定了AI能帮你到什么程度。**

选择Markdown，AI可以做你的全职知识管理员。选择私有格式，AI只能隔着一层玻璃看你的笔记。

管理员回来了

想象你的笔记库是一个图书馆。书一直在进，但管理员几年前就失踪了。没有索引卡片，没有分类架，书堆在地上，越堆越高。你偶尔需要找一本书，翻半天找不到，最后放弃，上网重新买一本。

AI就是那个回来的管理员。而且是一个不用睡觉、不会偷懒、能同时处理所有书的管理员。

但管理员要能干活，有一个前提：它得能碰到你的书。如果你的书锁在玻璃柜里（云端私有格式），管理员只能隔着玻璃看。如果书就摆在开放的书架上（本地Markdown文件），管理员拿起来就能分类、贴标签、建索引。

这引出了两个问题：什么样的容器对AI最友好？什么样的AI最适合管理知识？

§02 为什么是Obsidian

Why Obsidian

三个完全独立的十亿级项目，不约而同选了同一种方案存储AI的记忆。这不是巧合，是必然。而Obsidian恰好站在这个必然的正中央。

一个文件夹，就是一个vault

先说一件最简单的事：Obsidian的vault是什么？

一个文件夹。

不是数据库，不是云端服务，不是私有格式。就是你电脑上的一个文件夹，里面全是.md文件。你用Finder打开它，看到的就是一堆Markdown文件和子文件夹。你用VS Code打开它，也能正常编辑。你把它拖进终端，Claude Code可以直接读写。

Obsidian做的事情，是给这个文件夹加了一层UI：双向链接、图谱视图、搜索、插件。但底层的文件，从头到尾都是标准的Markdown。你任何时候都可以不用Obsidian，换成任何能读Markdown的工具，你的笔记不会丢失一个字。

这个设计决策看起来平淡，但在AI时代，它变成了Obsidian最大的竞争优势。

Markdown是LLM的母语

这不是比喻，是技术事实。LLM的训练数据里，Markdown是最主要的格式之一。GitHub上几十亿个README.md，技术文档、博客文章、教程，LLM全读过。它对##标题、**加粗**、- 列表的理解，像你对中文标点一样自然。

具体来说，三个特性：

Token效率高。同样的信息，用Markdown表示比用JSON或XML少30-50%的token。这意味着在相同的上下文窗口里，LLM能一次「看到」更多的笔记内容。Claude的1M token上下文窗口，如果你的笔记是Markdown格式，大约能装下200-300万字的内容，足够覆盖大多数人的整个知识库。

结构天然清晰。##是标题，>是引用，```是代码块。这些标记就是天然的结构分隔符。你不需要告诉AI「这是标题，这是正文」，Markdown语法本身就在说。

AI的默认输出格式。你让ChatGPT或Claude回答问题，回复就是Markdown。AI读Markdown最自然，写Markdown也最自然。你的知识库用Markdown，AI进来就像回家。

本地文件 = AI可直接读写

Obsidian的另一个关键决策：数据完全存在本地。

过去这被视为劣势：多设备怎么同步？没有云端备份丢了怎么办？但在AI Agent时代，「数据在本地」变成了决定性的优势。

Claude Code的工作方式是：在终端里运行，直接操作本地文件系统。它可以读文件、写文件、搜索文件、移动文件、重命名文件。所有操作都是对本地文件系统的直接访问，不需要API，不需要认证，不需要网络请求。

把Claude Code指向一个Obsidian vault，它立刻就能工作：

推荐

Obsidian（本地Markdown）：
Claude Code cd 进vault目录 → 直接读写所有笔记 → 零配置、零延迟、完整权限

不推荐

Notion（云端私有格式）：
需要配置API → 受权限和速率限制 → 无法自由搜索和批量操作 → 格式转换有损

这个差异不是「体验好一点vs差一点」，是「能做vs不能做」的区别。在Obsidian vault里，Claude Code是有完整权限的管理员。在Notion里，Claude Code是只能通过API窗口传话的外人。

殊途同归

如果只是理论分析，你可能觉得「好吧，有道理，但也就那样」。但当你看到三个完全独立的十亿级项目做出了一模一样的选择，你就不得不认真对待了。

Manus。一家AI Agent公司，2026年被Meta以20亿美元收购。他们的Agent在执行长任务时，用什么存储记忆？task_plan.md和notes.md。Markdown文件。

OpenClaw。开源AI Agent框架，GitHub上355,000+颗星。Agent的知识存在哪里？MEMORY.md。Agent的人格定义存在哪里？SOUL.md。全是Markdown文件。

Claude Code。Anthropic的AI编程工具。项目上下文存在哪里？CLAUDE.md。用户偏好和长期记忆存在哪里？memory/目录下的Markdown文件。

三个不同的团队，解决不同的问题，服务不同的用户。在没有互相参考的情况下，做出了同一个架构决策：**用Markdown文件作为AI Agent的记忆层。**

不是向量数据库。不是SQL。不是JSON。不是任何花哨的技术方案。就是.md文件。

为什么？

维度	Markdown文件	向量数据库
延迟	近零（本地文件读取）	需要网络调用 + embedding计算
成本	$0（本地存储）	$50-200/GB/月
可读性	文本编辑器就能查看	黑箱，需要专门工具
版本控制	Git原生支持	需要额外方案
人工干预	直接打开编辑	需要专门工具
可移植性	复制文件即可	供应商锁定

向量数据库在百万级数据的语义搜索场景下有优势。但对于个人知识管理，几千到几万条笔记的规模，Markdown文件在每一个维度上都更优。它更快、更便宜、更透明、更可控、更持久。

而Obsidian vault，恰好就是一堆Markdown文件。你在Obsidian里积累的每一条笔记、每一个链接、每一个标签，对AI来说都是可以直接读取和操作的结构化数据。

官方认可的方向

Obsidian的CEO叫Steph Ango，网名kepano。他在Obsidian社区里非常活跃，对产品方向有很强的个人影响力。

2026年1月，kepano在GitHub上发布了一个仓库：obsidian-skills。

这不是一个社区爱好者的项目。这是Obsidian的CEO亲自做的，教Claude Code如何正确处理Obsidian专有格式的官方Skills。截至写这本书时，这个仓库已经有20,000+颗星。

obsidian-skills里包含几个Skill文件：

- **obsidian-markdown**：教Claude Code正确处理.md文件，包括[[wikilinks]]、callouts、frontmatter等Obsidian独有的语法
- **obsidian-bases**：教Claude Code处理.base文件（Obsidian的数据库功能）
- **json-canvas**：教Claude Code处理.canvas文件（Obsidian的空间笔记地图）
- **defuddle**：网页内容清洗工具，在把网页内容存入vault前去除广告、导航栏和页面装饰，节省token

kepano在X上说：

> I'm starting a set of Claude Skills for Obsidian... so far they're centered around helping Claude Code edit .md, .base, and .canvas files.

这个信号很明确：Obsidian官方认为AI Agent操作vault是一个核心使用场景，并且亲自下场提供支持。

作为对比，Notion的AI是内置的封闭系统，你只能用Notion自己的AI，不能选择模型，不能让外部Agent直接操作你的数据。LogSeq虽然也是本地Markdown，但AI插件生态和社区规模远不如Obsidian。

Obsidian vs Notion vs LogSeq

直接上对比。

维度	Obsidian	Notion	LogSeq
数据存储	本地Markdown文件	云端私有格式	本地Markdown/EDN
AI Agent可达性	极高（文件系统直读写）	低（需API）	高（本地文件）
AI集成方式	开放：任选模型和工具	封闭：内置Notion AI	社区插件
数据所有权	完全属于你	在Notion的服务器上	完全属于你
团队协作	弱（需付费Sync）	强（原生实时协作）	弱
学习曲线	中等	平缓	陡峭
社区规模	最大（Reddit 10万+）	大	小
插件生态	1000+社区插件	集成市场	200+插件
价格	免费	$10/月起	免费开源
开源	否（免费但闭源）	否	是

先说结论：**团队协作选Notion，个人知识管理 + AI选Obsidian。**

这不是说Notion不好。Notion 3.0的AI Agent能力很强，可以自主编辑文档、操作数据库、构建表单。如果你的核心需求是团队协作、项目管理、知识共享，Notion仍然是最好的选择。

但如果你的核心需求是个人知识管理，而且你想让AI深度参与管理过程，Obsidian的架构优势是决定性的：

第一，Agent可达性。Claude Code进入Obsidian vault就像进入自己家，想看什么看什么，想改什么改什么。进入Notion就像去别人公司，要先刷卡、登记、等审批，而且只能在指定区域活动。

第二，开放生态。今天你用Claude Code，明天可能有更好的AI工具出来。Obsidian不锁定你的AI选择，vault就是文件夹，任何AI都能操作。Notion的AI是内置的，你只能用它给你的。在AI快速迭代的时代，开放架构意味着你永远能用上最新最好的工具。

第三，数据持久性。Obsidian如果明天倒闭了（虽然不太可能），你的笔记一个字都不会丢，它们就是你硬盘上的文件。Notion如果关停服务，你需要导出数据，而导出过程中格式和结构一定会有损失。

LogSeq是最接近Obsidian的竞品，也是本地优先、Markdown存储、完全开源。如果你在意开源，LogSeq值得考虑。但它的AI插件生态和社区活跃度和Obsidian差了一个数量级。知识管理工具的价值很大程度上来自社区：模板、插件、教程、最佳实践。这方面Obsidian的优势非常明显。

Obsidian额外给你的东西

如果只是要一个文件夹存Markdown，用Finder就行了。为什么要装Obsidian?

因为Obsidian在文件夹的基础上，做了几件关键的事。

[[双向链接]]。在任何一条笔记里输入[[，Obsidian会弹出你所有笔记的列表，选一个就建立了链接。被链接的笔记自动显示「谁链接了我」。这意味着知识之间的关联是双向的、自动追踪的。你不需要维护一个索引，链接自己就是索引。

图谱视图。点开Graph View，你看到的是整个vault的知识网络。每条笔记是一个节点，链接是节点之间的连线。孤立的笔记在边缘飘，链接密集的笔记在中心聚集。这个视图不只是好看，它帮你发现知识结构中的空白和集群。

快速搜索和跳转。Cmd+O打开快速切换器，输入几个字就能跳到任何一条笔记。Cmd+Shift+F全文搜索，速度极快。对比在文件树里翻文件夹，效率高一个量级。

社区插件。Obsidian有超过1000个社区插件，覆盖从日历、看板、数据视图到AI集成的各种场景。后面第8章会详细讲哪些值得装。

这些功能的共同特点是：它们增强了知识之间的连接和发现。文件夹让你存东西，Obsidian让你在存的基础上建立网络。而这个网络，对AI来说也极有价值。Claude Code可以沿着双向链接遍历关联知识，而不是孤立地处理单个文件。

花叔的视角：我为什么决定迁移

在写这本书之前，我的知识管理工具是Cursor的文件树 + Claude Code。

自序里提过，我有一套CLAUDE.md + Skills的体系，公众号、视频、橙皮书各有各的工作区和规则。写文章的时候调七八个skills，从调研到配图到发布都能跑通。它是能用的。

但当我研究Obsidian + Claude Code的时候，我意识到：我用拼接方案手动实现的很多东西，在Obsidian里是原生的。

我的拼接方案	Obsidian原生方案
CLAUDE.md路由器（根据关键词分发任务）	MOC (Map of Content)，支持[[链接]]一键跳转
每个子目录的CLAUDE.md	每个文件夹的index.md，功能一致
_knowledge_base/INDEX.md索引	原生搜索 + 图谱视图 + 语义搜索插件
Cursor文件树导航	Cmd+O模糊搜索 + 收藏夹 + 最近文件
Claude Code grep搜索	原生全文搜索 + Smart Connections语义搜索
没有	[[双向链接]] + backlinks
没有	图谱视图
没有	1000+社区插件

前五行说明我的方法论是对的。在不知道Obsidian社区最佳实践的情况下，我独立演化出了几乎一样的组织模式。后三行说明工具层面有很大的提升空间。

双向链接是最让我心动的。我的调研笔记里经常提到同一个人物、同一个概念，但这些笔记之间没有自动关联。在Obsidian里，只要我在任何地方输入[[Karpathy]]，所有提到Karpathy的笔记就自动形成一个网络。Claude Code可以沿着这个网络遍历，而不是靠grep碰运气。

图谱视图是第二个。我知识库里2000多个文件的关系只存在于我的脑子和CLAUDE.md的文字描述里。在Obsidian里，这些关系是可视化的，一眼就能看到哪些话题关联紧密、哪些领域是孤岛。

所以，迁移是必然的。

> **给犹豫的人：**如果你和我一样，已经有了一个Markdown文件夹在管理知识，迁移到Obsidian几乎是零成本。打开Obsidian，选你的文件夹作为vault，完事。你已有的所有文件一个不会丢，Obsidian只是给它们加了一层UI。如果你还没有任何知识管理系统，那就更好了，直接从Obsidian开始。

我觉得五年后回头看，2025-2026年会是个人知识管理的分水岭。不是因为出了什么新概念，而是因为AI第一次真的能帮你打理知识了。而Obsidian恰好是那个让AI伸得进手的容器。

下一章，30分钟，从零搭一个可用的vault。

§03 30分钟上手Obsidian

Get Started in 30 Minutes

前两章讲了为什么。这一章开始动手。目标很明确：30分钟后，你有一个能用的vault，写了第一条笔记，知道三个最重要的功能怎么用。

下载和安装

去 obsidian.md，下载安装包。全平台免费，没有注册，没有账号，没有试用期。

打开之后Obsidian只问你一件事：选一个文件夹作为你的vault。

如果你已经有一个装满Markdown文件的文件夹，直接选它。Obsidian不会改动你的任何文件，只是给这个文件夹加一层UI。打开Finder看，里面的文件一个都没变，只多了一个.obsidian/配置目录。

从零开始的话，新建一个空文件夹就行。

写第一条笔记

打开vault后，左上角有个「新建笔记」按钮（或者按Cmd+N / Ctrl+N）。点一下，一条空白笔记出现了。

写什么？

随便写。今天的想法、正在做的事、刚读到的一段话。

我知道你想问：要不要先设计一个分类系统？要不要先建几个文件夹？

不要。

推荐

好的做法：
先写100条笔记，然后根据实际内容决定怎么组织

不推荐

不好的做法：
还没写第一条笔记就花3小时设计标签体系和文件夹结构

新手最常见的陷阱就是「系统先行」。你花一个下午研究各种方法论，精心设计了一套完美的分类体系。一周后发现实际笔记根本不符合你的预设，整套系统推翻重来。

笔记系统应该从笔记里长出来，不是从设计图纸里长出来。

这本书的第5章会详细讲vault架构设计，但那是在你有了100条笔记之后的事。现在，写就行了。

三个真正有用的功能

Obsidian有很多功能，但真正让它和普通Markdown编辑器拉开差距的，是三个。

功能一：双向链接 [[]]

在任何一条笔记里输入两个左方括号 [[，Obsidian会弹出一个搜索框，列出你vault里所有的笔记。选一个，按回车，链接就建好了。

比如你在今天的日记里写「今天读了[[穷查理宝典]]」，这个[[穷查理宝典]]就变成了一个可点击的链接。点一下，直接跳到你之前写的穷查理宝典读书笔记。

如果vault里还没有「穷查理宝典」这条笔记呢？没关系，Obsidian会创建一个空白笔记等着你。链接先建上，内容以后再填。

这是前半段，后半段更有意思。

打开你的「穷查理宝典」笔记，往下看，有一个「Backlinks」面板。它告诉你：vault里有哪些笔记链接到了这里。你的日记链接了它，你的「投资原则」笔记也链接了它，你的「芒格语录」笔记也链接了它。

你不需要维护任何索引。链接关系是自动追踪的。你只管在写的时候顺手加个[[]]，Obsidian帮你记录谁和谁有关系。

> **一个对比：**在我之前的方案里，知识之间的关联全靠Claude Code用grep搜索碰运气。有了[[双链]]，关联是明确的、可追踪的。Claude Code沿着链接遍历知识，比grep高效一个量级。

功能二：标签 #tag

在笔记的任何位置输入#，后面跟一个词，就建立了一个标签。比如 #读书笔记 #投资 #芒格。

一条笔记只能存在一个文件夹里，但可以打任意多个标签。那篇「AI在医疗领域的应用」，放「AI」文件夹还是「医疗」文件夹？用标签就不纠结：同时打上 #AI 和 #医疗，两个维度都能找到。

标签不需要提前定义。想到什么就打什么，Obsidian自动收集所有出现过的标签。

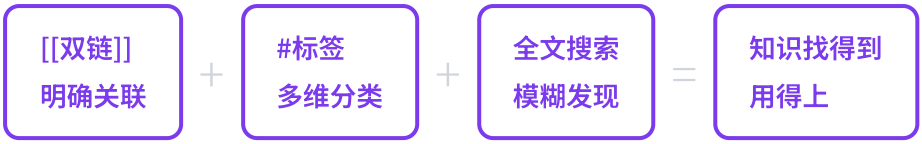
功能三：全文搜索

Cmd+Shift+F（Windows是Ctrl+Shift+F），输入关键词，搜索整个vault的所有内容。

几千条笔记瞬间出结果。支持正则表达式，支持按标签、文件夹、时间范围筛选。

你记得「好像在哪里看过关于XX的分析」，但记不清在哪条笔记里。全文搜索让你不需要精确记忆，模糊搜就能找到。

三个功能加在一起，形成了一个完整的知识发现体系：



图谱视图：看见你的知识网络

Cmd+P输入「graph」，点开。每条笔记是一个点，每个[[双向链接]]是连接两个点的线。

刚开始只有几个散落的点。别急。积累了几十条笔记后，图谱开始变得有意思：

某些笔记被很多笔记链接，它们变成网络的中心节点，这通常是你最核心的概念。

某些笔记在边缘独自飘着，没有任何链接，这些孤岛提醒你：它们应该和某些知识产生关联，只是你还没建立链接。

某些笔记形成了你没预料到的集群，这帮助你发现知识之间隐藏的联系。

图谱视图不是必需品。但vault规模大了之后，它是一个很好的「知识体检工具」，帮你看清自己的知识结构长什么样。

起步插件：只装两个

Obsidian有1000多个社区插件。先不管它们。

现在只装两个核心插件。打开 设置 → 第三方插件 → 浏览，搜索安装。

Daily Notes（日记）

开启后，每天自动创建一条以日期命名的笔记。你打开Obsidian，日记已经在那等你了。

日记是整个笔记系统中摩擦力最低的入口。不需要想标题，不需要选文件夹，不需要决定这条笔记属于哪个分类。今天想到什么就往日记里写，写完了。整理的事以后再说（第7章会讲怎么让AI帮你从日记里提炼洞察）。

Templates（模板）

创建笔记模板，新建笔记时一键套用。比如你可以做一个「读书笔记模板」：

字段	内容
书名	（填写）
作者	（填写）
核心观点	（填写）
我的想法	（填写）
相关笔记	[[]]

模板的好处是保持笔记结构一致。结构一致的笔记对AI非常友好，Claude Code读取时知道每条笔记的信息在什么位置。这个优势在第4章会体现出来。

> **关于插件的原则：**新手最大的错误是一上来装二十个插件。每个插件都有学习成本，装太多反而什么都用不好。先用两个月，等你明确知道自己缺什么功能的时候，再去找对应的插件。第8章会给你一份精选推荐清单。

对Cursor/VSCode用户说

如果你平时用Cursor或VS Code，Obsidian的界面你会觉得很眼熟。

功能	VS Code / Cursor	Obsidian
文件浏览	左侧文件树	左侧文件树
多文件编辑	标签页	标签页
快速跳转	Cmd+P	Cmd+O（几乎一样）
命令面板	Cmd+Shift+P	Cmd+P
分屏编辑	支持	支持
快捷键自定义	支持	支持
插件生态	VS Code Extensions	Community Plugins

操作方式几乎一样，切换过来没有学习成本。

本质区别在设计目标。IDE为代码而生，Obsidian为知识而生。长得像，解决的问题完全不同。

一个有意思的玩法是两个一起用。Cursor里跑Claude Code操作vault文件，Obsidian里浏览知识图谱和backlinks。代码工具做重活，知识工具做导航。

新手最常犯的四个错误

最后说说「别做什么」。

错误一：花太多时间设计系统。笔记还没写够50条，就开始研究各种方法论。每种都试三天就换，结果笔记没几条，系统倒换了五六套。先写后整理，永远是对的顺序。

错误二：装太多插件。社区插件列表就像一个糖果店，每个都想尝尝。装了二十个插件，侧边栏挤满了不认识的图标，快捷键冲突，启动变慢。两星期后全关了，回到原点。起步阶段只需要Daily Notes和Templates，真的。

错误三：试图完美分类。每条笔记都要想好放哪个文件夹、打什么标签、链接到哪里。这种完美主义让记笔记变成一件有压力的事，最终你选择不记。记住：链接比文件夹更重要，以后再整理比现在就完美更重要。而且第4章之后，整理的活可以交给AI。

错误四：从别人的复杂vault开始。网上有很多人分享自己用了三年的vault模板，几十个文件夹，上百个模板。看着很壮观，但那是别人积累了几千条笔记后演化出来的结构，直接搬过来只会水土不服。从空vault开始，让系统自然生长。

30分钟检查清单

到这里，如果你跟着做了，你应该有：

这就够了。

Obsidian还有很多功能你没看到。Canvas画布、Bases数据库、Publish发布。不急，以后慢慢探索。现在最重要的事是开始写笔记，积累内容。

下一章，真正的主角登场。

§04 Claude Code进场

Enter Claude Code

上一章你花30分钟搭好了vault。现在，把AI请进来。整个过程不超过10秒：打开终端，cd到vault目录，输入claude。没了。

4.1 最简连接：cd进去就能用

很多人一听「AI + Obsidian」，第一反应是要装插件、配API、搞MCP。

都不用。打开终端，两行命令：

```
cd ~/Documents/my-vault
claude
```

Claude Code启动后，自动获得整个vault目录的读写权限。读笔记、新建文件、搜索内容、重命名、移动、删除，全都可以。不需要任何配置。

为什么？因为vault就是一个文件夹，里面全是Markdown文件。Claude Code操作本地文件系统。两者天然兼容，不需要翻译层。

你还可以装一个Obsidian-Shell插件，在Obsidian内部打开终端。默认工作目录就是vault根目录，连cd都省了。

推荐

Claude Code + Obsidian vault：
cd进目录 → 直接读写所有.md文件 → 零配置、零延迟

不推荐

其他AI + 笔记工具：
配置API Key → 安装集成插件 → 权限受限 → 只能通过接口间接操作

第一次启动Claude Code的时候，它会扫描当前目录。如果vault里已经有几十上百条笔记，你可以试着问一句：「我的vault里有什么？帮我概览一下。」看看它的反应。这是一个很好的冒烟测试，确认连接没问题。

4.2 CLAUDE.md：给AI的员工手册

连接上了，但还不够。

你新招了一个助理，第一天上班，你指着满屋子的文件柜说「都在这里了，开始干活吧」。他要么无从下手，要么乱翻一气。缺了一份员工手册。

CLAUDE.md就是这份员工手册。

在vault根目录创建一个叫CLAUDE.md的文件。Claude Code每次启动时会自动读取它。你在里面写什么，AI就按什么规矩办事。

一个知识管理场景的CLAUDE.md，至少要包含四块内容：

第一块：你是谁。身份、关注领域、偏好。三五行就够。

第二块：vault怎么组织。文件夹结构、命名规范、各区域用途。AI读到就知道东西该存哪，不会乱放。

第三块：行为边界。什么能动什么不能动。比如「可以自由添加标签和链接，但不要删除任何已有内容」。

第四块：笔记模板。新笔记长什么样，frontmatter有哪些字段。有了模板，AI创建的笔记风格和你手写的就一致。

花叔的CLAUDE.md路由器

我自己的CLAUDE.md比上面说的要复杂得多。

「写作/」目录下有9个工作区：

```
写作/
├── CLAUDE.md                ← 路由器：根据关键词把任务分发到子目录
├── 01-公众号写作/
│   ├── CLAUDE.md            ← 公众号写作的专属规则
│   ├── 项目/                ← 进行中的文章项目
│   ├── _knowledge_base/     ← 知识库，有INDEX.md索引
│   └── _archive/            ← 已发布的文章
├── 03-视频创作/
│   ├── CLAUDE.md            ← 视频创作的专属规则
│   └── ...
├── 10-橙皮书/
│   ├── CLAUDE.md            ← 橙皮书制作的专属规则
│   └── ...
├── .claude/skills/          ← 55个Skills
└── tools/                   ← 工具脚本
```

根目录的CLAUDE.md是一个路由器。它里面有一张路由表：

关键词	工作区	读取文件
写文章、公众号	公众号写作	/01-公众号写作/CLAUDE.md
视频脚本	视频创作	/03-视频创作/CLAUDE.md
橙皮书、epub	橙皮书	/10-橙皮书/CLAUDE.md
调研、竞品分析	调研与分析	/07-调研与分析/

当我说「帮我写一篇公众号文章」，Claude Code读到根目录CLAUDE.md里的路由表，自动跳转去读/01-公众号写作/CLAUDE.md。那个文件里定义了公众号写作的完整规则：风格要求、审校标准、配图规范、发布流程。

当我说「做一本橙皮书」，它就去读/10-橙皮书/CLAUDE.md，里面是橙皮书的专属流程：章节Fragment怎么写、EPUB怎么构建、微信读书怎么上架。

一个人管这么多工作区，靠的不是效率高，是Claude Code读完CLAUDE.md之后知道每一件事该怎么做，不需要我每次都从头解释。

你不需要一开始就搞这么复杂。我这套系统是一年多慢慢长出来的。每次Claude Code犯一个错，我就在CLAUDE.md里加一条规则。一开始只有几行字，慢慢长成了路由系统。

> **CLAUDE.md的生长是自然的。**不要试图一上来就写一个完美的员工手册。先写最基础的：你是谁、文件夹怎么组织。然后每次AI做错了什么，就加一条规则。三个月后你回头看，会发现它已经变成了一份很详尽的指南。

知识管理场景的CLAUDE.md模板

如果你的vault主要用来做个人知识管理，这个模板可以直接用：

```
# 我的知识库

## 关于我
- [你的身份/职业]
- 关注领域：[列出3-5个]
- 偏好：用中文、不要过度格式化

## Vault结构
- daily/：每日记录，按YYYY-MM-DD.md命名
- notes/：永久笔记，按「主题-标题.md」命名
- projects/：进行中的项目，每个项目一个文件夹
- raw/：待整理的原始素材
- archive/：已完成/不再活跃的内容

## 笔记模板
新建笔记时使用以下frontmatter：
```yaml
---
title: 笔记标题
tags: []
created: YYYY-MM-DD
type: permanent
summary: 一句话摘要
---
```

行为规则
- 可以：添加标签、创建[[双向链接]]、生成摘要、整理和分类
- 不可以：删除已有笔记内容、修改我的原始记录
- 创建新笔记时必须遵循上面的frontmatter模板
- 每次整理后更新相关文件夹的index.md

标签体系
- 按领域：#tech #business #reading #life
- 按状态：#todo #in-progress #done
- 按类型：#idea #reference #project
```

复制这个模板，根据自己的情况改一改，放到vault根目录。这就是你的起点。

4.3 index.md：做了80%工作的导航文件

CLAUDE.md告诉AI整个vault的全局规则。但vault里有十个文件夹，每个文件夹里有几十个文件。AI怎么知道具体去哪找东西？

Michael Crist在实践中发现了一个简单到离谱的技巧：**在每个主要文件夹里放一个index.md。**

index.md不需要复杂。三五行就够：

```
# 读书笔记

这个文件夹存放所有读书笔记，按「作者-书名.md」命名。

## 关键文件
- [[卡尼曼-思考快与慢]] - 认知偏差的经典
- [[芒格-穷查理宝典]] - 跨学科思维模型
- [[塔勒布-反脆弱]] - 不确定性中获益
```

就这么点东西，效果却很显著。没有index.md的时候，Claude Code进入一个文件夹，看到50个文件，需要逐个扫描。有了index.md，它先读这个文件，3秒钟就知道这个文件夹的用途和重要文件在哪。

有意思的是，我在不知道他的做法时独立搞了一模一样的东西。这说明这个模式是自然演化出来的，不是谁发明的。

> **核心建议**
> 你不需要手动给每个文件夹都写index.md。等把Claude Code接入vault之后，直接让它帮你生成。它会扫描文件夹内容，自动写出描述和关键文件列表。这件事本身就可以作为你的「第一次协作」练习。

4.4 安装kepano/obsidian-skills

第2章提到过，Obsidian的CEO kepano在GitHub上发布了obsidian-skills。现在把它装上。

操作很简单。在vault目录下创建.claude/skills/文件夹（如果还没有的话），然后把obsidian-skills里的Skill文件复制进去：

```
# 在vault目录下执行
mkdir -p .claude/skills
# 把obsidian-skills的文件复制到这里
```

Claude Code下次启动时会自动加载这些Skill文件。

为什么需要它？Obsidian的Markdown有些独有语法，Claude Code不一定懂：

Obsidian语法	标准Markdown	差异
`[[笔记名]]`	`[文本](url)`	Wikilinks，Obsidian特有的双向链接语法
`> [!note] 标题`	`> 引用文本`	Callout blocks，带类型和折叠的引用块
`.canvas文件`	无对应	JSON Canvas，空间笔记地图
`![[图片.png]]`	`![alt](path)`	嵌入语法，可以嵌入笔记、图片、PDF

装了之后，Claude Code知道在vault里要用[[]]而不是[text](url)，知道callout语法，知道.canvas文件格式。一次性操作，30秒搞定。

4.5 第一次协作：让AI整理笔记

连接做好了，CLAUDE.md写好了，Skills装好了。来干点实事。

找一批散乱的笔记丢进vault。可以是你以前在备忘录里记的东西，可以是微信收藏的文章摘要，可以是读书时随手记的几段话。不管格式多乱都行，先扔进来。

然后在终端里告诉Claude Code：

```
帮我整理vault里的散乱笔记。
读取所有没有frontmatter的文件，按主题分类，
给每条笔记加上frontmatter，建立双向链接，
最后生成index.md。
```

接下来你会看到Claude Code开始工作：

读取 → 扫描vault里的所有文件，理解每条笔记的内容　**分类** → 按主题把笔记移到对应的文件夹

标注 → 给每条笔记添加frontmatter（标题、标签、摘要）　**链接** → 发现笔记间的关联，插入[[双向链接]]

索引 → 为每个文件夹生成index.md

五分钟。也许更快。

整理完成后，打开Obsidian的图谱视图。你会看到原来散落的点开始形成网络。笔记之间开始出现连线。一条关于「注意力」的笔记连向了你的卡尼曼读书笔记，也连向了一条关于社交媒体的思考。这些关联以前只在你脑子里模糊地存在着，现在变成了可见的、可点击的链接。

你只是把东西扔进来，AI把它变成了一个有结构的知识库。**人负责记录和思考，AI负责整理和关联。**

> **一个小建议：**第一次让AI整理后，花几分钟检查一下结果。看看分类是否合理，标签是否准确，链接是否有意义。如果有不对的地方，在CLAUDE.md里加一条规则。比如「技术类笔记的标签必须包含具体技术名称，不能只打一个#tech」。这就是CLAUDE.md自然生长的过程。

不到20分钟，你的vault从一个被动的文件夹，变成了一个有AI管理员驻场的知识系统。

但目前的架构还比较基础。下一章讲怎么设计vault的结构，让AI和你的配合效率再上一个台阶。

§05 设计你的Vault架构

Design Your Vault

上一章你用不到20分钟把Claude Code接入了vault。能用了，但「能用」和「好用」之间差了一个架构设计。这一章讲的是：怎么组织你的vault，让AI能高效地理解和操作它。

六条设计原则

这不是理论，是和Claude Code协作过程中反复踩坑换来的。违反其中几条，AI的表现就会明显下降。

原则一：Markdown是唯一格式

不要在vault里存PDF、Docx、富文本。

任何你想让AI处理的信息，先转成Markdown再放进vault。读了一本书，笔记用.md。收到PDF报告，关键内容摘录成.md。开会录了音，AI转写后存.md。

原始文件可以保留，但放在vault之外。vault里只存Markdown，这是底线。

推荐

vault里：

reading-notes/芒格的智慧.md（你的摘录和思考）

meeting/2026-04-11-产品评审.md（会议纪要）

不推荐

不要放进vault：

芒格的智慧.pdf（原始PDF扫描件）

会议录音-0411.m4a（原始音频）

原则二：保持用词一致

你在一条笔记里写「RAG」，另一条写「检索增强生成」，第三条写「retrieval augmented generation」。对你来说是同一个东西。但Claude Code搜索「RAG」时只能命中第一条。

解决方法：在vault里统一术语。选一个你最常用的写法，始终用它。

统一用	不要混用
RAG	检索增强生成 / retrieval augmented generation
Claude Code	CC / claude-code / Anthropic的编程工具
LLM	大模型 / 大语言模型 / foundation model
frontmatter	元数据 / YAML头 / 文件头信息

你可以在vault根目录放一个 glossary.md（术语表），列出你选定的标准用词。Claude Code在处理笔记时会参考它。

原则三：扁平优先

文件夹不要超过三层嵌套。

「学习/AI/LLM/RAG/论文/」五层嵌套，分类精确，看起来很整洁。但对AI来说，每一层嵌套都在消耗路径token，深层文件夹很容易被遍历遗漏。

更好的方式：用标签和链接做分类，文件夹只管生命周期（进行中/已完成/归档）。

推荐

扁平结构（推荐）：

notes/rag-retrieval-strategies.md（标签：#AI #RAG）

notes/rag-chunking-methods.md（标签：#AI #RAG）

notes/transformer-attention.md（标签：#AI #LLM）

不推荐

深层嵌套（不推荐）：

学习/AI/LLM/RAG/检索策略/笔记.md

学习/AI/LLM/RAG/分块方法/笔记.md

学习/AI/LLM/基础/注意力机制/笔记.md

经验法则：如果你的文件路径长度超过3个斜杠（根目录/一级/二级/文件.md），就该考虑扁平化了。

原则四：每条笔记要有summary

在frontmatter里加一个summary字段，用一句话说清楚这条笔记讲什么。Claude Code扫描vault时不需要读全文，看summary就能判断这条笔记和当前任务有没有关系。

```
---
summary: 对比了RAG的三种检索策略（向量/关键词/混合），结论是混合检索在大多数场景下效果最好
---
```

写summary的技巧：假设你的朋友问你「这条笔记讲啥」，你会怎么用一句话回答？那就是summary。

原则五：frontmatter五字段

五个推荐字段：

```
---
title: RAG检索策略对比
tags: [AI, RAG, 技术笔记]
created: 2026-04-11
type: permanent
summary: 对比了三种检索策略，混合检索在大多数场景下效果最好
---
```

字段	作用	对AI的意义
title	笔记标题	语义搜索的关键信号
tags	主题分类	批量筛选特定领域的笔记
created	创建日期	判断信息时效性（AI领域3个月前的信息可能已过时）
type	笔记类型	区分处理策略（permanent要细读，fleeting可以跳过）
summary	一句话摘要	快速判断相关性，不需要读全文

type字段的三个值借鉴了卡片盒笔记法：

- **fleeting**（闪念笔记）：随手记的想法、待整理的素材。AI可以跳过或者帮你整理
- **literature**（文献笔记）：读书/读文章后的笔记。AI可以帮你提取要点、建立关联
- **permanent**（永久笔记）：经过思考和整理的、你自己的观点和总结。AI在生成内容时优先参考这些

你不需要从第一天就把所有笔记都加上完整的frontmatter。先养成习惯，新笔记都加上这五个字段。老笔记可以让Claude Code批量补上——第7章会讲具体怎么做。

原则六：区分人的输入和AI的产出

这条原则容易被忽略，但很重要。

你写的原始笔记，和AI生成的内容（摘要、关联分析、自动整理的索引），应该分开存放。最简单的方式是用文件夹区分：你的笔记放 notes/，AI生成的内容放 ai-output/ 或者在文件名前加 _ai- 前缀。

为什么要分开？两个原因。

信任度不同。你自己写的笔记，你知道它的来源和可信度。AI生成的内容可能有幻觉、可能过时、可能理解错了你的意思。混在一起，时间长了你分不清哪些是你的原始记录、哪些是AI的产物。

处理策略不同。AI在处理你的原始笔记时会更谨慎（这是源数据，不能随意修改）。处理AI自己生成的内容时可以更大胆（这是衍生数据，可以随时重新生成）。分开存放让Claude Code知道该用哪种策略。

> **核心建议**
> 一个实用的做法：在CLAUDE.md里写清楚哪些文件夹是人工产出、哪些是AI产出。比如「notes/下的文件是我的原始笔记，未经确认不要修改。ai-output/下的文件是AI生成的，可以自由更新。」

三种架构模板

原则讲完了，现在来看具体怎么组织文件夹。这里提供三种模板，覆盖从极简到复杂的不同需求。你不用纠结选哪个，先从模板A开始，用不够了再往B或C升级。

模板A：极简型

适合刚开始用Obsidian的人，或者笔记数量在200条以下的场景。

```
my-vault/
├── daily/ # 每日笔记
├── notes/ # 所有笔记（扁平存放）
├── projects/ # 进行中的项目
├── archive/ # 已完成的项目和过时笔记
└── CLAUDE.md # AI的操作手册
```

四个文件夹，加一个CLAUDE.md，结束了。

daily/ 放每日笔记（Obsidian有Daily Notes核心插件，一键生成当天的日记）。notes/ 放所有的知识笔记，不分子文件夹，靠标签和链接做分类。projects/ 放正在进行的项目相关文件。archive/ 放完成的东西。

CLAUDE.md是关键。它告诉Claude Code：这个vault的结构是什么、每个文件夹放什么、你的偏好是什么。没有它，Claude Code进来就像走进一个没有标识的仓库，只能盲目搜索。

模板A的优点是没有决策负担。新笔记往notes/扔就行，不需要思考该放哪个文件夹。缺点是笔记多了以后，notes/会变得很大，但用标签和搜索完全能管理。

模板B：PARA型

适合同时管理多个项目、多个领域的人。PARA是Tiago Forte提出的个人知识管理框架，把所有信息分成四个维度：Projects（项目）、Areas（领域）、Resources（资源）、Archive（归档）。

```
my-vault/
├── 1-Projects/ # 有明确截止日期的项目
├── 2-Areas/ # 持续关注的领域（健康、财务、技能...）
├── 3-Resources/ # 感兴趣的主题资料
├── 4-Archive/ # 已完成/不再活跃的内容
├── daily/ # 每日笔记
└── CLAUDE.md # AI的操作手册
```

PARA的核心思想是按「行动力」分级：Projects最需要行动（有deadline），Areas需要持续关注，Resources是备用参考，Archive是存档。

我自己的写作体系本质上就是PARA的变体，虽然我没有刻意按PARA来设计。独立演化出几乎一样的结构，说明PARA对应的是一种很自然的信息组织方式。

模板B的优点是结构清晰，每条笔记的归属很明确。缺点是你需要经常做「移动」操作，一个项目完成了要从1-Projects移到4-Archive。不过这个操作可以交给Claude Code自动完成。

模板C：Karpathy Wiki型

适合做研究的人、知识密集型工作者、或者想把vault当成个人wiki来运营的人。

```
my-vault/
├── raw/ # 原始输入（文章摘录、会议记录、灵感）
├── wiki/ # 经过整理的知识条目（一个概念一个文件）
├── output/ # 基于知识库产出的内容（文章、报告、脚本）
├── SCHEMA.md # 知识条目的格式规范
└── CLAUDE.md # AI的操作手册
```

这个模板的灵感来自Andrej Karpathy。他在一次访谈中提到，他维护一个个人wiki，每个概念一个页面，像Wikipedia一样组织知识。

模板C的核心是 raw → wiki → output 的信息流。raw/ 是输入层，什么都往里扔，不需要整理。wiki/ 是知识层，每个概念一个文件，经过提炼和结构化。output/ 是输出层，基于wiki里的知识产出内容。

SCHEMA.md 定义了wiki条目的标准格式。比如：

```
## wiki条目规范

每个wiki条目必须包含：
- frontmatter：title, tags, created, summary
- 定义：用一两句话解释这个概念
- 要点：3-5个核心要点
- 关联：[[链接]]到相关概念
- 来源：原始信息来自哪里
```

有了SCHEMA.md，Claude Code就能按照统一格式生成新的wiki条目。你把一篇论文的PDF摘录扔进raw/，Claude Code读取后在wiki/里生成一条格式标准的知识条目。

模板C的优点是知识复用率最高——wiki层的内容是经过整理的，可以直接引用。缺点是维护成本高，不过这个成本主要由AI承担。下一章会详细讲Claude Code怎么自动从raw生成wiki条目。

> **怎么选？** 笔记少于200条，或者刚入门，选模板A。多个项目并行管理，选模板B。研究型工作、需要高度结构化的知识库，选模板C。不确定？先A，用两周后再决定。

frontmatter实操细节

一个完整的frontmatter长这样：

```
---
title: 笔记标题
tags: [AI, knowledge-management]
created: 2026-04-11
type: permanent
summary: 一句话摘要
---

正文从这里开始。
```

几个注意事项：

tags用英文。虽然Obsidian支持中文标签，但英文标签在Claude Code的grep搜索中表现更稳定。你可以用 #AI 而不是 #人工智能，用 #reading-notes 而不是 #读书笔记。

created用ISO格式。2026-04-11，不是「4月11日」或「April 11, 2026」。ISO格式（YYYY-MM-DD）是唯一没有歧义的日期写法，AI和人类都能一眼看懂。

type只用三个值。fleeting、literature、permanent。不要发明更多类型。分类越细，维护成本越高，你越容易放弃。三个值够用了。

summary控制在一句话。30字到80字之间。写不出一句话摘要的笔记，可能本身就需要拆分。

> **核心建议**
> Obsidian有一个模板功能（核心插件Templates），可以预设frontmatter模板。新建笔记时一键插入，不用每次手打。设置路径：设置 → 核心插件 → 模板 → 指定模板文件夹。

双链 vs 标签 vs 文件夹

Obsidian给了你三种组织笔记的工具：[[双向链接]]、#标签、文件夹。很多人纠结该用哪个。答案是三个都用，但各管各的事。

文件夹管生命周期

文件夹回答的问题是：「这个东西处于什么阶段？」

进行中的项目放 projects/。完成了移到 archive/。每日记录放 daily/。知识条目放 notes/ 或 wiki/。

文件夹不应该用来做主题分类。「AI」文件夹、「投资」文件夹、「健康」文件夹，不要这样做。一条笔记可能同时涉及AI和投资（比如「用AI做量化交易」），放哪个文件夹都不对。

#标签管主题

标签回答的问题是：「这个东西属于什么领域？」

一条笔记可以打多个标签。#AI #投资 #量化 三个标签同时存在，完美解决了「放哪个文件夹」的问题。Claude Code搜索某个主题的笔记时，grep标签比遍历文件夹高效得多。

标签系统的关键是保持克制。不要超过20个一级标签。嵌套标签（#AI/RAG）可以用，但不要超过两层。

[[双链]]管关系

双向链接回答的问题是：「这个概念和哪个概念有直接关系？」

当你在一条笔记里写到某个概念，而vault里恰好有一条专门讲这个概念的笔记时，用[[链接]]把它们连起来。这种链接是语义层面的，比标签更精确。标签说「这两条笔记都关于AI」，双链说「这条笔记引用了那条笔记里的一个具体观点」。

Claude Code可以沿着双向链接遍历关联知识。你让它写一篇关于RAG的文章，它看到你的RAG笔记里链接了[[向量数据库]]和[[embedding]]，就会自动去读这两条笔记，获得更完整的上下文。

三者协作的例子

假设你读了一篇关于RAG优化的论文：

```
文件位置：notes/rag-optimization-2026.md    ← 文件夹管生命周期（notes = 知识笔记）
标签：#AI #RAG #论文                          ← 标签管主题
正文链接：[[向量数据库]] [[embedding]] [[chunking]]  ← 双链管关系
```

三种工具各司其职，没有冲突。

> **注意**
> 最常见的错误是只用一种工具。只用文件夹→一条笔记无法属于多个主题。只用标签→没有笔记间的直接关联。只用双链→没有全局分类视图。三个一起用，才能发挥Obsidian的全部能力。

你的CLAUDE.md应该写什么

结合上面所有原则，一个vault用的CLAUDE.md模板：

```
# Vault操作手册

## 结构说明
- daily/：每日笔记，按YYYY-MM-DD命名
- notes/：知识笔记，扁平存放，靠标签和链接分类
- projects/：进行中的项目，一个项目一个子文件夹
- archive/：已完成的项目和过时笔记

## frontmatter规范
每条笔记必须包含：title, tags, created, type, summary
type取值：fleeting / literature / permanent

## 术语表
- 统一用「RAG」，不用「检索增强生成」
- 统一用「LLM」，不用「大模型」
- 统一用「vault」，不用「知识库」

## 操作规则
- notes/下的文件是我的原始笔记，未经确认不要修改内容
- 可以修改frontmatter（补充缺失的tags、summary等）
- 新建笔记时必须包含完整frontmatter
- 项目完成后移入archive/
```

不长，但信息密度很高。先写个基础版，用的过程中发现AI做错了什么，就加一条规则。

好的vault架构，就是让AI用最少的token理解最多的信息。设计完了，下一章开始真正干活。

§06 让AI维护你的知识库

Let AI Maintain Your Knowledge Base

与其建RAG让AI检索你的笔记，不如让AI直接维护你的知识库。这不是我说的，是Karpathy说的。而且1600万人看到了。

编译器，不是检索器

2026年4月，Andrej Karpathy发了一条推文。

内容不长，大意是：他在用LLM维护一个个人wiki。不是让AI搜索他的笔记然后回答问题，而是让AI阅读原始素材后直接写出结构化的wiki文章，这些文章会被保留、迭代、交叉引用。

这条推文获得了1600万+浏览。

为什么引爆了？因为它戳中了一个很多人隐约感觉到但没说清楚的问题：**RAG的路子可能走偏了。**

RAG是目前最主流的「让AI使用你的数据」的方式。笔记丢进向量数据库，AI收到问题后先搜相关片段，塞进prompt，生成回答。

听起来合理。但用过就知道问题在哪。

每次你问AI一个问题，它都在从零开始。搜索、匹配、拼接、生成。下次问一个相关的问题，同样的流程再走一遍。AI没有记忆，没有积累。今天帮你理清了「RAG和Fine-tuning的区别」，明天你再问，它又要重新搜一遍、重新想一遍。

就像一个每天早上失忆的实习生。查资料写报告写得不错，但第二天忘了昨天写过什么，同样的话题从头来。

Karpathy的洞察很简单：**让这个实习生把报告存下来。**

让AI把理解到的知识写成wiki文章。这些文章被保留在知识库里，可以被后续的问题引用，被新信息更新，和其他文章建立交叉引用。

AI不是在检索，是在编译。

检索器每次都在做重复劳动。编译器做一次工作，产出可复用的成果。你的原始笔记被AI编译成wiki文章之后，下次用到这个知识时，直接读wiki就行，不需要重新处理原始素材。

RAG的思路：每次提问 → 搜索原始素材 → 拼接生成回答 → 回答用完即弃

LLM Wiki的思路：AI阅读原始素材 → 写成wiki文章 → 文章被保留和迭代 → 后续提问基于wiki回答

RAG的产出用完就扔。LLM Wiki的产出越用越厚。这是一个正反馈循环。

三层架构

Karpathy的方案落地到Obsidian vault里，是一个三层结构。

第一层：raw/（源材料）
你丢进来的一切原始素材。文章、论文、会议记录、读书笔记、网页剪藏、聊天截图、随手记的想法。

规则：只增不改。这是source of truth。不管AI后面怎么加工，原始素材永远保留原貌。就像做菜，买回来的食材不要动，加工在厨房里进行。

第二层：wiki/（结构化知识）
AI阅读raw/里的素材后写出的结构化文章。按类型分三个子目录：
- concepts/：概念页。比如「RAG」「向量数据库」「Agent记忆」
- entities/：实体页。比如「Karpathy」「Obsidian」「Claude Code」
- topics/：主题页。比如「AI知识管理工具对比」「2026年大模型趋势」

规则：主要由AI写入和维护，人可以修正。这一层是整个系统的核心价值所在。

第三层：output/（查询产物）
基于wiki生成的报告、分析、回答。比如你让AI「对比一下Obsidian和Notion在AI时代的竞争力」，它基于wiki里已有的概念页和实体页生成一篇分析报告，存在output/里。

好的output可以反哺wiki。如果分析报告里产生了新的洞察，AI可以把它更新回wiki的对应文章。

完整的vault结构长这样：

```
vault/
├── raw/           # 第一层：源材料（不可变）
│                  # 你丢进来的一切
│                  # 规则：只增不改
│
├── wiki/          # 第二层：结构化知识（AI维护）
│   ├── INDEX.md   # 全局索引
│   ├── concepts/  # 概念页
│   ├── entities/  # 实体页
│   └── topics/    # 主题页
│
├── output/        # 第三层：查询产物
│                  # 报告、分析、回答
│
├── SCHEMA.md      # wiki的「宪法」
└── CLAUDE.md      # AI的员工手册
```

三层之间的数据流向很清晰：raw → wiki → output。原始素材被编译成结构化知识，结构化知识被用来生成产出。信息只在一个方向流动，每一层都有明确的职责。

你可能注意到了两个特殊文件：SCHEMA.md和CLAUDE.md。CLAUDE.md我们在第4章讲过，它是给AI的员工手册。SCHEMA.md是新东西，专门服务于wiki这一层。

SCHEMA.md：把通用AI变成专业wiki管理员

Claude Code是通用AI。你让它「帮我整理笔记」，它会整理，但整理的方式可能每次都不一样。今天标签叫#ai-tools，明天叫#AI工具。

SCHEMA.md解决这个问题。它定义了wiki的命名规范、标签体系、wikilink规则、文章模板。有了它，通用AI变成了严格执行规范的wiki管理员。

给你看一个完整的SCHEMA.md示例：

````
# Wiki Schema

## 命名规范
- 概念页：`concepts/概念名.md`（用小写英文，多词用连字符）
  - 例：`concepts/retrieval-augmented-generation.md`
- 实体页：`entities/实体名.md`（人名用英文全名）
  - 例：`entities/andrej-karpathy.md`
- 主题页：`topics/主题描述.md`
  - 例：`topics/ai-knowledge-management-tools.md`

## Frontmatter 模板
每篇wiki文章必须包含：
```yaml
---
title: 文章标题
type: concept | entity | topic
tags: [标签1, 标签2]
sources: [raw/中的源文件路径]
created: YYYY-MM-DD
updated: YYYY-MM-DD
summary: 一句话摘要
---
```


## 标签体系
- 领域标签：#ai, #programming, #product, #business
- 状态标签：#stub（存根，需扩充）, #mature（成熟）
- 不要自创新的顶级标签，如有需要先更新本文件

## Wikilink 规则
- 首次提到某个已有wiki页面的概念时，用[[链接]]
- 同一篇文章中同一个概念只链接第一次
- 如果提到的概念没有wiki页面，创建一个stub

## 文章结构
概念页：定义 → 核心要点 → 和其他概念的关系 → 参考源
实体页：简介 → 关键贡献 → 相关概念/实体 → 参考源
主题页：概述 → 多角度分析 → 结论 → 参考源

## 全局索引
每次新增或修改wiki文章后，更新 wiki/INDEX.md
INDEX.md 按分类列出所有wiki页面及一句话摘要
````

不到50行，但统一了命名、格式、标签、链接四件事。AI不会再纠结「这篇文章叫RAG.md还是retrieval-augmented-generation.md」，也不会出现#ai和#AI并存的混乱。

你可以根据自己的需求修改。做投资研究的加status字段（看多/看空/中性），做学术的加citation格式。SCHEMA.md的核心价值不在于具体写什么，而在于「有一份明确的规范，AI严格执行」。

实操：从零构建一个知识Wiki

说了这么多概念，来动手做一个。

第一步：准备raw/素材。

找5-10篇你最近读过的文章、笔记或者文档，丢进vault的raw/目录。不需要整理，不需要改名，原封不动地放进去。这些就是你的源材料。

比如你最近在关注AI知识管理这个话题，你可能有这些素材：

- Karpathy的LLM Wiki推文截图和讨论
- 一篇关于RAG原理的技术博客
- Obsidian和Notion的对比文章
- 你自己之前记的关于向量数据库的笔记
- 一次关于知识管理工具的播客笔记

第二步：写SCHEMA.md。

把上面那个模板复制到vault根目录，根据你的需求微调。刚开始不用改太多，先用默认的跑起来，后面再迭代。

第三步：让Claude Code阅读素材并编写wiki文章。

在终端里cd到vault目录，启动Claude Code，告诉它：

```
请阅读 raw/ 目录下的所有文件，按照 SCHEMA.md 的规范，
为其中涉及的核心概念、人物和主题创建 wiki 文章。
每篇文章都要建立和其他文章的 [[交叉引用]]。
完成后更新 wiki/INDEX.md。
```

然后看着它工作。

Claude Code会读完你的素材，提取出关键概念（比如RAG、向量数据库、LLM Wiki）、关键人物（比如Karpathy）和关键主题（比如AI知识管理工具对比）。然后按照SCHEMA.md的格式，逐一创建wiki文章。

它写RAG的概念页时，会自动链接到[[向量数据库]]。写Karpathy的实体页时，会链接到[[LLM Wiki]]和[[RAG]]。写主题页时，会引用多个概念页和实体页。知识网络就这样自动生长出来了。

第四步：打开Obsidian图谱视图。

点开Graph View，你会看到刚才还空空如也的vault里出现了一个小型知识网络。节点是wiki文章，连线是[[wikilinks]]。概念相关的文章聚集在一起，形成集群。

这个网络现在还很小。但随着你不断往raw/丢新素材，让AI编译成wiki文章，网络会越来越稠密，知识之间的连接会越来越丰富。

第五步：向AI提问。

现在试一下效果。问Claude Code一个问题：

```
RAG和LLM Wiki这两种方案，分别适合什么场景？
```

它不需要重新去搜索原始素材。wiki里已经有了RAG的概念页和LLM Wiki的概念页，两者之间还有交叉引用。AI读完这些已经编译好的知识，直接给你一个高质量的对比分析。

这就是编译的威力：第一次花时间把知识整理好，后面每次使用都是即时的。

规模和边界

这套方案能撑多大？

根据我的实际测试和社区的反馈，**40万字规模以内，完全不需要向量数据库**。Claude Code直接读取Markdown文件就够了。wiki/目录下几十篇结构化文章，加上INDEX.md做导航，AI找到相关知识的速度和准确度都很高。

为什么40万字就够了？因为Claude的上下文窗口足够大。200K token的标准窗口大约能装50-80万字的Markdown内容。如果你用1M token的扩展窗口，那就是200-300万字。对绝大多数个人知识库来说，这个容量绰绰有余。

AI不需要一次读完你所有的wiki。它先读INDEX.md，知道有哪些文章，然后根据当前的问题有针对性地读取相关的几篇。这和人查百科全书一样，先看目录，再翻到对应的页面。

但超过百万字级别的知识库，纯文本检索的效率会下降。这时候可以考虑加一层语义搜索，第8章推荐的Smart Connections插件就能做这件事。它用AI embedding为vault里的每篇笔记建立语义索引，搜索时不只匹配关键词，还匹配语义相似度。

知识库规模	推荐方案	是否需要向量数据库
10万字以内	wiki/ + INDEX.md + Claude Code直读	不需要
10-40万字	同上，wiki目录按主题细分	不需要
40-100万字	加Smart Connections语义搜索辅助	可选（插件内置）
100万字以上	必须配合RAG做语义检索	需要

这个方案依赖大上下文窗口。2026年，200K token已经是主流模型的标配，1M也不罕见。随着窗口继续增大，纯文本wiki方案能支撑的规模也会跟着增长。Karpathy说的「不需要向量数据库」在大多数场景下已经成立了。

花叔的_knowledge_base/就是wiki的雏形

当我第一次看到Karpathy那条推文时，我的反应是：等等，这不就是我一直在做的事吗？

我的_knowledge_base/目录里有五十多个文件，按主题分类，有INDEX.md索引。结构上和wiki/几乎一样。

维度	花叔的_knowledge_base/	Karpathy的wiki/
索引	INDEX.md（表格形式）	INDEX.md（表格形式）
分类	按主题文件夹（技术/人物/产品）	按类型文件夹（concepts/entities/topics）
格式	Markdown，有简介	Markdown，有frontmatter
交叉引用	少量手动链接	系统性[[wikilinks]]
维护方式	手动整理 + AI辅助	AI全权维护
规范	隐含在CLAUDE.md里	独立的SCHEMA.md

核心理念是一样的。差异在两个地方：交叉引用的密度，和自动化程度。

我的知识库里的文件之间很少互相链接，Karpathy方案里每篇wiki文章通过[[wikilinks]]紧密关联。AI可以沿着链接遍历相关知识，而不是只看单一文件。

我的知识库主要靠手动整理，Karpathy方案把这个过程完全自动化：SCHEMA.md定义规范，AI按规范创建文章、更新索引、建立交叉引用，全程不需要人参与。

从手动维护的知识仓库，到自己会长的知识网络。工作量不大，效果是质变。

> **给已有知识库的人：**你不需要从零开始。把你已有的笔记目录当作raw/，让Claude Code按照SCHEMA.md把它们编译成wiki文章。已有的积累一点都不浪费，只是换了一种更高效的组织方式。

AI不是你的搜索引擎，是你的知识编译器。

下一章，7个真实工作流，全是可以直接抄的实操。

§07 7个真实工作流

7 Real Workflows

不讲理论了。这一章全是可以直接抄的操作。每个工作流我都会给出具体步骤和prompt，你看完就能在自己的vault里跑起来。

7.1 日记 + AI周报

这是最低门槛的工作流。如果你只打算尝试一件事，就从这个开始。

Obsidian有一个核心插件叫Daily Notes。打开它之后，每天你按一个快捷键（Cmd+D或点左侧栏的日历图标），Obsidian自动创建一个以日期命名的文件，比如2026-04-11.md。你不需要自己创建文件、不需要想文件名、不需要决定存在哪。一个快捷键，直接开始写。

每天只写几行就够了。不要试图写长篇日记，那坚持不了两周。三五行就好：今天做了什么、想了什么、看到什么有意思的东西。格式随意，不需要结构化。你写给未来的自己和AI看，不是写给别人。

比如这样：

```
## 2026-04-11

- 上午写了橙皮书第7章，Obsidian工作流那章
- 看到一篇文章说Cursor要出Obsidian插件了，存到raw/了
- 下午和出版社开会，讨论纸质书排版，决定放弃纸质书这条路
- 晚上试了一下Gemini 3 Pro的图片生成，质量比上个月好很多
- 想法：AI工具的更新速度已经快到「写书」这个形式跟不上了
```

就这样。一天五分钟。

坚持一周之后，有意思的事情来了。你让Claude Code读取这一周的日记，生成一份周报摘要。

1. **设置Daily Notes插件**
   Settings → Core plugins → 打开Daily notes → 设置模板和存储路径（建议放在 daily/ 文件夹下）

2. **每天写几行**
   Cmd+D打开当天日记，随手记录。不需要格式，不需要完整句子。关键是降低记录的心理门槛。

3. **周末让Claude Code生成周报**
   在终端里进入vault目录，让Claude Code读取这周的日记并生成摘要。prompt可以这样写：

```
读取 daily/ 目录下本周（4月7日到4月11日）的所有日记。
帮我生成一份周报摘要，包括：
1. 本周主要做了什么（按项目分类）
2. 值得记住的想法和洞察
3. 下周可以跟进的事情
保存到 weekly/2026-W15.md
```

Claude Code会扫描你的日记，把零散的碎片整理成有结构的周报。

年度Wrapped更好玩。年底的时候，让Claude Code读取整年的周报（或直接读365天的日记），生成一份年度回顾：你这一年做了多少个项目、读了几本书、最频繁思考的话题是什么、哪些想法反复出现。

Spotify Wrapped让你看到自己一年听了什么歌。Obsidian + Claude Code让你看到自己一年想了什么。

> **核心建议**
> 日记最大的价值不在于回顾，而在于积累。你今天随手写的一个想法，可能在三个月后写文章时被Claude Code搜到，成为一段精彩的素材。但前提是你得先记下来。

7.2 读书笔记 + AI提炼

读书最大的浪费是什么？读完了但什么都没留下。

你划了线、写了批注、当时觉得「这段太精彩了」。三个月后你想引用这本书的某个观点，发现你只记得「好像说过一个什么什么」，具体是什么完全想不起来。

解决方案很简单：把划线和批注丢进vault的raw/文件夹，让Claude Code帮你生成结构化的文献笔记。

1. **读书时随便划**
   在Kindle、微信读书或纸质书上正常阅读。划线、写批注，不需要在意格式。

2. **读完后把原始标注存入vault**
   把所有划线和批注复制到一个文件里，存为 raw/反脆弱-原始标注.md。不用整理，原样粘贴即可。

3. **让Claude Code生成文献笔记**
   prompt可以这样写：

```
读取 raw/反脆弱-原始标注.md，这是我读《反脆弱》的划线和批注。
帮我生成一份结构化的读书笔记，包括：
1. 这本书的核心论点（3-5个）
2. 每个论点的关键论据和例子
3. 我在批注中表达的反应和想法（保留原味，不要改写）
4. 这本书和vault中已有的哪些笔记有关联（搜索一下vault）
保存到 books/反脆弱.md
同时，在笔记中用[[双向链接]]链接到vault中已存在的相关概念。
```

最后一步是关键。Claude Code不只是帮你整理一本书的笔记，它还会搜索你的vault，找到已有的相关笔记并建立链接。比如你之前写过关于[[塔勒布]]的笔记，或者在某篇日记里提到了[[黑天鹅]]，Claude Code会自动把这些关联建立起来。

每一本书不再是一个孤岛。它被编织进了你整个知识网络。

> **核心建议**
> 进阶选项：如果你用Kindle，可以装Readwise插件（第三方服务），它能自动把Kindle划线同步到Obsidian vault。同步之后再让Claude Code处理，连「复制粘贴划线」这步都省了。

7.3 调研 + 知识积累

这是我自己用得最多的工作流。每次写公众号文章之前，我都要做调研。看官方文档、读科技媒体的报道、翻X上的讨论、试用产品。这些调研成果以前散落在浏览器标签页和临时文件里，文章写完就没了。

现在我把所有调研成果存进vault。

1. **调研时把原始信息存入raw/**
   搜索到有用的文章、产品文档、用户评价，把关键内容存入 raw/ 文件夹。不需要整理，原始信息优先保存，格式不重要。

2. **让Claude Code编译成wiki文章**
   调研差不多了，让Claude Code把raw/里的信息整合成一篇结构化的wiki文章：

```
读取 raw/ 目录下所有关于Obsidian的调研文件。
帮我整合成一篇wiki文章，保存到 knowledge/Obsidian.md。
要求：
1. 按主题组织（是什么、核心功能、和竞品的对比、社区生态、AI集成现状）
2. 标注每条信息的来源
3. 用[[双向链接]]链接到vault中已有的相关笔记
4. 列出调研中发现的、值得深入了解的问题
```

3. **下次调研同一话题时，AI已经有上下文**
   三个月后你又要写一篇和Obsidian相关的文章。Claude Code读到 knowledge/Obsidian.md，立刻知道你之前已经调研过什么，可以在已有基础上增量更新，而不是从头开始。

每一次调研不再是一次性消耗品，而是知识库的一次增量更新。你调研越多，vault里的上下文越厚。AI的起点也越高。

我写到第七篇Claude Code文章时，明显感觉到了差异。Claude Code已经知道我之前说了什么、测试过什么、观点是什么。初稿不再需要从零解释。

越写越厚，越厚越好写。正反馈循环。

7.4 写作工作流（从笔记到文章）

前面三个工作流（日记、读书笔记、调研）都是在往vault里「存」东西。这个工作流是「用」。

写文章的传统方式是打开一个空白文档，从零开始。想法是模糊的，初稿遥不可及。

有了vault，你不再面对空白页。

1. **告诉Claude Code你要写什么**
   不需要大纲，不需要结构，一句话描述你的选题就行。比如「我想写一篇文章，主题是AI工具的更新速度为什么让写书变得很难」。

2. **Claude Code先搜索你的vault**
   它会在你的日记、读书笔记、调研文件、之前的文章里搜索所有和这个话题相关的内容。然后给你一份「已有素材清单」，告诉你vault里已经有什么可以用。

3. **基于已有知识生成初稿**
   Claude Code不是凭空编一篇文章，而是基于你已有的笔记、观点和素材组织初稿。你曾经在日记里写过的一个观点、你在某本书的批注里表达的反应、你调研时发现的一个数据，都可能被编织进初稿。

初稿建立在你曾经思考和写过的一切之上。

AI生成的初稿不是终稿，你仍然要改。但你改的是一个已经包含了你自己想法和素材的草稿，不是空白页。

不推荐

没有vault的写作：

空白页 → 凭记忆想素材 → 去网上重新搜 → 拼凑初稿 → 反复修改

推荐

有vault的写作：

一句话选题 → AI搜索vault已有素材 → 基于你自己的知识生成初稿 → 聚焦修改

vault里的笔记越多，这个工作流就越强大。这也是为什么前三个工作流（日记、读书笔记、调研）很重要：它们是这个工作流的弹药库。

7.5 项目管理

Obsidian不是专业的项目管理工具，不要指望它替代Jira或Linear。但对于个人项目和小团队，Obsidian的项目管理能力完全够用，而且有一个独特优势：项目知识和项目管理在同一个地方。

方法很简单：每个项目一个文件夹，文件夹里放一个index.md。

```
projects/
├── 橙皮书-Obsidian/
│   ├── index.md          # 项目状态、待办事项、关键决策
│   ├── 调研/             # 调研笔记
│   ├── 草稿/             # 章节草稿
│   └── 素材/             # 参考资料
├── 公众号-Claude源码分析/
│   ├── index.md
│   ├── 调研.md
│   └── 草稿.md
└── archive/              # 已完成项目
```

index.md里放什么？项目状态、待办事项、关键决策记录。就这么简单。

Obsidian的优势在于：项目之间的知识可以通过[[双向链接]]互相引用。

比如你在写橙皮书时调研了Claude Code的CLAUDE.md机制，这个知识也和你的「公众号文章」项目有关。你在调研笔记里输入[[CLAUDE.md]]，这个概念就同时出现在两个项目的上下文里。你不需要维护一个跨项目的知识索引，链接自己就是索引。

项目完成后，整个文件夹移入archive/。不删除，只归档。archive/里的笔记仍然可以被搜索到、被链接到。一个已完成的项目，它的知识资产不会消失，只是从「活跃」变成了「可查阅」。

> **核心建议**
> 如果你需要看板视图（类似Trello），可以装Kanban插件。它把Markdown文件变成可拖拽的看板卡片，所有数据仍然是Markdown，Claude Code可以直接读写。

7.6 自动整理旧笔记

你有一堆散乱的旧笔记，知道里面有好东西，但每次打开都想关上。

MakeUseOf有人分享过：5年的笔记，没有标签，没有链接，手动整理了三次都放弃了。最后把整个文件夹指给Claude Code，90分钟整理完。

1. **把旧笔记放进vault的一个文件夹**
   比如 inbox/ 或 unsorted/。不管格式是什么，先扔进去。如果是txt或doc格式，先批量转成Markdown（Claude Code也能帮你做这件事）。

2. **让Claude Code扫描并整理**
   给Claude Code这样的prompt：

```
扫描 unsorted/ 目录下的所有Markdown文件。
为每个文件做以下处理：
1. 添加frontmatter（title、date、tags）
2. 根据内容主题，移动到合适的文件夹（books/、ideas/、projects/、reference/）
3. 搜索vault中其他笔记，添加相关的[[双向链接]]
4. 生成一份整理报告，保存到 unsorted/整理报告.md

注意：
- 不要删除任何内容，只添加结构
- 如果不确定分类，放到 inbox/ 并在报告中标注
- frontmatter中的tags根据内容自动生成，不超过5个
```

3. **审查整理报告**
   Claude Code会生成一份报告，列出它做了什么：移动了多少文件、添加了多少链接、有哪些不确定的分类。你快速过一遍，调整不对的地方。

整理完之后，旧笔记从「信息坟场」变成了有结构、有链接、可检索的知识资产。更重要的是，它们和你的新笔记在同一个网络里了。今天写的日记有可能通过双向链接和三年前的一条旧笔记产生关联。

> **实操建议：**不要一次性扔太多文件给Claude Code。如果你有上千个旧文件，分批处理（每次100-200个），每批处理完审查一下质量，再进行下一批。这样你可以在过程中修正Claude Code的分类逻辑，后面的批次会越来越准。

7.7 自动Backlinks（Agentic Note-Taking）

前面的工作流都需要你主动触发。这个可以做到半自动。

双向链接很好，但你得记得去建。写日记提到「今天和老王吃了个饭」，不手动输入[[老王]]，这条日记和之前关于老王的笔记就没有关系。

德国开发者Stefan Imhoff每天花10到15分钟手动添加[[wikilinks]]。直到他用了Claude Code。

他的工作流是这样的：

1. **正常写日记**
   不需要加任何链接，就像在记事本里随便写一样。提到人名、书名、概念都用纯文本。

2. **写完后运行Claude Code**
   让Claude Code扫描今天的日记，自动识别所有可以链接的实体。prompt类似这样：

```
读取今天的日记 daily/2026-04-11.md。
找出所有的人名、地名、书名、概念名词。
对于每一个实体：
1. 搜索vault中是否已有对应的笔记
2. 如果有，把日记中的纯文本替换为[[wikilink]]
3. 如果没有，创建一个新的实体笔记（包含基本信息），然后链接
修改后直接保存到原文件。
```

3 **日记自动变成链接网络**

你写的「今天和老王吃了个饭，聊了聊塔勒布的新书」，会被自动变成「今天和[[老王]]吃了个饭，聊了聊[[塔勒布]]的新书」。如果vault里没有「老王」的笔记，Claude Code还会帮你创建一个。

手动做这件事要10到15分钟。Claude Code做只要几秒钟。

而且Claude Code做得更彻底。你可能漏掉一个人名，它不会。你可能忘了vault里有一个关于某个概念的笔记，它会搜索到。

这就是Stefan Imhoff所说的「Agentic Note-Taking」。笔记的整理和关联工作交给AI Agent，你只负责写。你的每一条日记、每一条笔记，都会被自动编织进你的知识网络。

不推荐

手动链接：

写完日记 → 回头检查有没有可链接的实体 → 逐个搜索vault → 逐个添加[[链接]] → 每天10-15分钟

推荐

AI自动链接：

写完日记 → 一条命令 → Claude Code自动识别、搜索、链接 → 几秒钟完成

核心建议

进阶玩法：把这个自动链接做成一个Obsidian模板命令或CLAUDE.md里的自动规则。每次你保存日记，Claude Code自动触发链接流程。从「手动触发」变成「写完就链」。

一张总览表

七个工作流，从简单到复杂，总结如下：

工作流	你做的事	AI做的事	每天花多久
日记 + AI周报	每天写几行	每周生成摘要	5分钟
读书笔记	读书时划线批注	生成结构化笔记 + 建链接	读书时间（不额外增加）
调研积累	把原始信息存入raw/	编译成wiki + 增量更新	调研时间（不额外增加）
写作工作流	说一句选题	搜索vault + 生成初稿	0（触发即可）
项目管理	建文件夹 + 写index.md	跨项目链接知识	按需
整理旧笔记	把旧笔记扔进vault	分类、加标签、建链接	一次性（审查报告）
自动Backlinks	正常写笔记	自动识别实体、建链接	0（自动运行）

注意最右边那列。**这些工作流不给你增加额外负担。**你做的事情和以前一样。AI在后面把信息变成有结构、有关联的知识。

不需要七个全用。从日记开始，坚持两周，自然会想试其他的。

§08 插件和工具生态

Plugins and Tools

Obsidian的插件生态有1000多个选项，但你真正需要的可能只有4个。这一章帮你跳过试错。

8.1 必装：obsidian-skills

只推荐一个「必装」，因为它真的是唯一一个不装会亏的。

obsidian-skills是Obsidian CEO kepano亲手做的。不是传统插件，而是一组Claude Code Skills，教AI正确理解Obsidian的专有语法。

Obsidian的Markdown有自己的扩展语法：[[wikilinks]]双向链接、callout提示框、frontmatter属性、.canvas空间笔记、.base数据库文件。这些不是标准Markdown，Claude Code默认不认识，操作时可能搞乱格式。

obsidian-skills做的事情就是在CLAUDE.md里告诉Claude Code：遇到[[]]不要当Markdown链接处理，遇到callout不要拆散结构，遇到frontmatter要保留YAML格式。

安装只要30秒：

1. **打开终端，进入vault**

   ```
   cd /path/to/your/vault
   ```

2. **安装Skills**

   ```
   claude install kepano/obsidian-skills
   ```

装完之后你不需要做任何配置。Claude Code在操作vault时会自动读取这些Skills，知道怎么正确处理Obsidian的文件。

8.2 强烈推荐的三个

以下三个不是必须的，但每一个都能明显提升Obsidian + AI的体验。看你的使用习惯选装。

Claudian：在Obsidian里直接用Claude Code

你在Obsidian里看笔记，想让Claude Code处理点什么，得切到终端、跑命令、看结果、切回来。来回切窗口。

Claudian把Claude Code的对话界面嵌进Obsidian侧边栏。右侧输入指令，主编辑区实时看到笔记变化。读文件、写文件、搜索、分析图片，和终端里完全一样。就是少了一次Alt+Tab。

> **核心建议**
> Claudian需要你本地已经安装了Claude Code。它不是一个独立的AI，而是Obsidian和Claude Code之间的桥梁。

Smart Connections：语义搜索

Obsidian自带的搜索是关键词匹配。搜「投资」，只能找到写了「投资」两个字的笔记。「资产配置」「长期收益」？搜不到。

Smart Connections用AI embedding给vault建语义索引。搜「投资」，所有相关概念的笔记都能出来。

更实用的是打开任何一条笔记，侧边栏自动显示语义最接近的其他笔记。经常能发现你自己没意识到的关联。

支持本地模型跑embedding，笔记不需要上传云端。

Copilot for Obsidian：知识库问答

你想知道自己之前关于定价策略写过什么。或者芒格和费曼有哪些共同的思维方式。

Copilot在vault层面做了一个RAG系统。你直接提问，它搜索vault里的相关笔记，基于笔记内容生成回答，标注引用来源。

和问ChatGPT的区别：回答来自你自己的笔记，不是互联网通用信息。你问的是自己的知识库。

推荐

Copilot for Obsidian：
基于你的vault内容回答 → 每个观点都有笔记出处 → 是你自己知识的重新组织

不推荐

直接问ChatGPT：
基于互联网通用知识回答 → 没有你的个人积累 → 可能和你的实际情况不符

8.3 按需安装

以下插件解决的是更细分的场景，不是每个人都用得上。

插件	做什么	适合谁
Smart Composer	类似Cursor的AI写作助手，在编辑器里内联补全和改写	每天写大量文字的人
Text Generator	模板化文本生成，定义模板后批量产出	需要批量生成相似内容的人
Agent Client	统一界面接入Claude Code、Codex、Gemini CLI等多种AI工具	同时用多个AI工具的人
Templater	高级模板引擎，支持JavaScript脚本和动态内容	模板重度用户
Dataview	把vault当数据库查询，用类SQL语法筛选和展示笔记	想把vault做成结构化知识系统的人

别一次装太多。先用基础功能跑一两周，哪里痛了再装对应的插件。插件装多了，注意力会从笔记转移到配置上。

8.4 MCP：大多数人不需要

MCP（Model Context Protocol）是Anthropic推出的协议，让AI工具和应用双向通信。在AI社区里经常看到这个词。

但在Obsidian场景下，大多数人不需要。Claude Code直接读写本地文件已经够了，不需要MCP做中间层。

什么时候MCP有用？两个场景：

- 你想让Claude Desktop（不是Claude Code）也能操作vault。Claude Desktop不能直接访问本地文件系统，需要MCP做桥梁。
- 你需要调用Obsidian插件的高级功能。比如让AI执行一个Dataview查询，拿到结果后做分析。Dataview的查询功能在Obsidian内部运行，Claude Code直接读文件做不到，需要MCP把Dataview的能力暴露给AI。

遇到「读文件解决不了」的需求时再考虑MCP。在那之前，忽略它。

8.5 Skills Marketplace

Claude Code有独立的Skills生态，和Obsidian插件互补。社区已有700,000+个skills，和知识管理相关的值得看看：

- **obsidian-second-brain**：专门为「第二大脑」场景设计的Skills集合，包含笔记整理、关联发现、定期回顾等自动化流程
- **claude-obsidian**：社区维护的Obsidian操作Skills，覆盖了比kepano官方Skills更多的使用场景

安装方式和obsidian-skills一样：

```
claude install [skill-name]
```

插件扩展Obsidian的UI和功能，Skills教会Claude Code新的操作能力。一个改工具，一个改AI。

选择原则：先装obsidian-skills，再根据最痛的需求选1-2个插件，最后按需探索Skills Marketplace。别贪多，每加一层都是认知负担。

§09 进阶

Advanced Patterns

基础搭好了，接下来让系统变成你自己的。版本控制、自定义Skills、本地AI、多Vault策略，以及从拼接方案迁移到Obsidian的真实经验。

9.1 用Git管理vault

Obsidian vault就是一个文件夹。Git是管理文件夹变化的最成熟工具。把vault用Git管理，你获得三样东西。

完整的改动历史。Claude Code每次修改笔记都留下记录。改了什么、改之前什么样，全部可追溯。最坏的情况就是git revert回去。

AI操作的透明性。让AI整理了一批笔记之后，`git diff`精确地看到它动了什么。比打开几十个文件逐个检查强得多。

放手的自信。「帮我重新组织这50个笔记的目录结构」，没有Git你会犹豫。有Git你知道随时能回退。

初始化非常简单：

1. **进入vault目录**

   `cd /path/to/your/vault`

2. **创建.gitignore**

   把`.obsidian/workspace.json`和`.obsidian/workspace-mobile.json`加进去。这两个文件记录窗口布局，每次打开Obsidian都会变，不需要追踪。

3. **初始化Git仓库**

   `git init && git add -A && git commit -m "初始化vault"`

之后的日常用法：

```
# 看AI改了什么
git diff

# 满意了就提交
git add -A && git commit -m "Claude整理了读书笔记"

# 不满意就回退
git checkout -- .
```

> **核心建议**
> 如果你不想手动操作Git，Obsidian有一个「Obsidian Git」插件，可以设置自动定时提交。比如每30分钟自动commit一次。

9.2 自定义Skills

obsidian-skills教会Claude Code怎么处理Obsidian文件。自定义Skills更进一步：教它执行你专属的工作流。

Skills就是CLAUDE.md里的一段自然语言指令。你写什么规则，它就按什么做。

几个实用的例子：

自动为日记生成backlinks

日记里提到了某本书、某个人、某个概念。手动加[[链接]]很烦。在CLAUDE.md里写一条规则：

> 当我让你「处理今天的日记」时，读取最新的daily note，识别其中提到的所有人名、书名、概念。检查vault里是否已有对应的笔记，如果有，把原文中的提及替换为[[wikilink]]。如果没有，在wiki/目录下创建一个空的占位笔记，然后再链接过去。

这一条规则就把第1章里Stefan Imhoff的整个工作流自动化了。

每周编译raw/到wiki/

如果你按照前面章节建立了raw/（原始素材）和wiki/（精炼知识）的双区结构，可以设一条规则让Claude Code定期把raw/里的笔记提炼成wiki/条目：

> 当我让你「编译本周笔记」时，读取raw/目录下最近7天修改过的文件。对每个文件，提炼出核心概念，检查wiki/里是否已有相关条目。已有的就追加新信息，没有的就创建新条目。编译完成后给我一个变更摘要。

遇到新概念自动创建wiki页面

在CLAUDE.md里加一条通用规则：

> 在操作vault的过程中，如果你在笔记里发现一个重要概念但vault里没有对应的wiki页面，主动在wiki/目录创建一个包含基本定义的页面，并在原始笔记中添加[[链接]]。

这样Claude Code在日常操作中就会持续充实你的wiki，不需要你手动触发。

Skill开发快速指南

写自定义Skill不需要会编程。它就是自然语言指令，写在CLAUDE.md里。几个要点：

- **触发条件要明确。**「当我说XX的时候」比「你觉得合适的时候」好。前者可预测，后者全看AI心情。
- **操作范围要限定。**「读取raw/目录下的文件」比「读取所有文件」好。限定范围能避免AI误操作你不想动的笔记。
- **输出格式要指定。**「创建的wiki页面用H1标题+一段定义+相关链接」比「创建一个页面」好。AI有了模板才能保持一致性。
- **先小范围测试。**写完一条Skill，先在几个文件上试试效果。满意了再推广到整个vault。

9.3 本地AI方案

这本书用的AI都是云端的。数据要发送到服务器处理。大多数人不在意，但vault里有公司机密、飞机上没网、或者你就是不想让数据出电脑时，需要本地方案。

Ollama可以在本地跑开源大模型（Llama 3、Mistral、Qwen）。下载一个模型，你电脑上运行，不发送到任何外部服务器。

Obsidian + 本地AI的组合方式：

- **Smart Connections + Ollama**：用本地模型生成embedding和语义搜索，完全离线
- **Copilot for Obsidian + Ollama**：用本地模型做vault级问答，数据不出电脑
- **Text Generator + Ollama**：用本地模型生成和改写文本

推荐

本地AI的优势：
数据完全私有 / 无网络也能用 / 没有API费用 / 不受速率限制

不推荐

本地AI的代价：
模型能力弱于云端（GPT-4/Claude级别的模型本地跑不动） / 需要较好的硬件（建议16GB+内存） / 首次下载模型需要网络

务实的建议：日常用云端AI（能力强、速度快），敏感内容切到本地AI。不是非此即彼，两个可以共存。

9.4 多Vault策略

所有笔记放一个vault，还是分多个？

一个vault的好处是简单：所有知识在一个地方，搜索和链接不断裂。但内容量大了，或者需要隔离不同性质的内容时，多vault就有意义。

两种常见分法：

工作 + 个人。工作vault放项目文档、会议记录、客户资料。个人vault放读书笔记、日记、生活记录。好处是边界清晰，换工作时工作vault留下（或带走），个人vault永远跟着你。也避免了公司机密和个人日记混在一起的尴尬。

公开 + 私有。如果你有写博客或做数字花园（Digital Garden）的习惯，可以把打算公开的笔记放一个vault，私人笔记放另一个。公开vault可以直接发布到网站，私有vault用更严格的备份策略。

多vault的注意事项：

- [[双向链接]]不能跨vault工作。两个vault之间的笔记无法互相引用。
- 每个vault有独立的.obsidian配置，插件和主题需要分别设置。
- Claude Code一次只能操作一个vault（一个工作目录）。要切换vault需要cd到另一个目录。
- 每个vault可以有自己的CLAUDE.md，定义不同的AI行为规则。工作vault里的AI可能更正式，个人vault里的AI可能更随意。

> **核心建议**
> 如果你刚开始用Obsidian，建议先用一个vault。等笔记积累到你觉得「这两类内容真的不该放在一起」的时候再拆分。过早拆分会增加管理成本。

9.5 从「拼接方案」迁移到Obsidian

这一节写我自己的迁移经历。踩过什么坑、保留什么、改了什么。

迁移之前，我用一个文件夹、Cursor文件树、Claude Code做所有读写和搜索。每个子目录有CLAUDE.md定义规则，根目录CLAUDE.md做路由。55个自定义Skills覆盖从调研到发布的全流程。

能用。用了大半年产出不少东西。但本质上是「拼接方案」，用文件系统+CLAUDE.md手动模拟知识管理，每一个本该工具提供的功能都得自己实现。

迁移步骤（极简版）

对，就这么简单。因为你的文件本来就是Markdown，Obsidian打开它们不需要任何转换。你现有的所有文件、所有目录结构、所有CLAUDE.md都原封不动。Obsidian只是在上面加了一层UI。

踩过的坑

CLAUDE.md路由器需要调整。我原来的CLAUDE.md里有大量「当收到XX任务时，先读取XX目录的CLAUDE.md」的路由规则。这些在Obsidian里仍然有效（Claude Code照样读CLAUDE.md），但有些路由逻辑可以被Obsidian原生功能替代。比如「搜索知识库」不再需要手动写grep规则，Smart Connections的语义搜索更好用。冗余的路由规则要逐步清理。

_knowledge_base/INDEX.md可以退役了。我之前手动维护了一个知识库索引文件，列出所有知识条目和位置。在Obsidian里，Cmd+O快速搜索和图谱视图比手动索引好用一百倍。INDEX.md留着当参考就行，不再需要维护。

.obsidian/目录会出现在文件树里。Obsidian在vault根目录创建一个.obsidian/目录存配置。vault用Git管理的话，记得在.gitignore里排除`.obsidian/workspace*.json`（布局文件会频繁变化）。其他配置文件（插件、主题）可以纳入Git，方便多台机器间同步。

值得保留的

CLAUDE.md体系完全保留。CLAUDE.md不是Cursor的概念，是Claude Code的。无论你用Cursor还是Obsidian还是纯终端，Claude Code都会读它。花在定义规则上的时间一秒没浪费。

目录结构完全保留。Obsidian不要求特定目录结构。9个工作区、三层结构，迁移后原样运行。

所有Skills完全保留。55个Skills独立于编辑器。换了UI，执行逻辑没有变化。

必须改的

养成用[[双向链接]]的习惯。这是Obsidian给你的最大新能力。写笔记时遇到重要概念、人名、项目名，随手输入[[链接过去。前几天觉得多了一个动作，一周后变成肌肉记忆。

定期看图谱视图。每周花5分钟打开Graph View扫一眼。哪些话题形成了集群？哪些笔记是孤岛？哪些概念连接了不同领域？这个全局视角，纯文件系统给不了你。

> 迁移的本质：不是换工具，是在已有基础上加一层更好的UI。文件、结构、规则、Skills全部保留。Obsidian加的是双向链接、图谱视图、插件生态这些CLAUDE.md实现不了的东西。成本接近零。

最后的话

这本书讲的所有东西，Obsidian、Claude Code、双向链接、自动整理，它们单独看都是工具和技巧。但它们指向一个更大的变化。

过去几十年，我们用的所有知识工具，从笔记本到Word到Notion，都有一个共同假设：整理是你的事。你负责分类、打标签、建索引、维护结构。工具提供容器，你提供秩序。

AI改变的是这个假设。你可以只管往里扔东西，整理交给AI。你写，它织网。

这听起来像一个小变化，但它的后果可能比我们想象的大。当「整理」不再是瓶颈，你的记录行为会改变。你会记录更多，因为不用担心「记了找不到」。你会更自由地联想，因为AI会帮你发现联系。你的笔记系统从一个需要维护的花园，变成一个自己生长的东西。

我不确定这最终会长成什么样。但我知道一件事：那些现在就开始积累的人，在AI能力继续提升的未来，会拥有别人没有的东西。不是工具，不是技巧。是数据，是你多年来思考和写下的一切。

工具会变。今天是Obsidian，明天可能是别的。但你vault里的那些Markdown文件，它们是你的。

Obsidian + Claude Code

用AI重建你的第二大脑

花叔

AI Native Coder · 独立开发者 · AI自媒体博主

代表作：小猫补光灯（App Store付费榜Top 1）

加入知识星球 →

B站：花叔 · 公众号：花叔 · X/Twitter · YouTube · 小红书 · 官网

Created by 花叔 · v1.0.0 · 2026年4月