我如何用一台笔记本和一部手机搭建了一个会自己进化的工作系统

一个软件工程师的个人系统架构全解析

xinyun2020 | 2026 年 4 月 11 日 | 墨尔本

问题：知识散落一地，每天都在重新发明轮子

你有没有这种经历?

上周做了一个重要的技术决策，今天完全想不起来为什么。三个月前研究过一个工具，现在又从头 Google。工作笔记在 Notion 里，代码在 GitHub，待办事项在脑子里，每天早上都要花半小时"回忆昨天做到哪了"。

我也一样。直到有一天我决定：不再容忍了。

我花了几周时间，把所有工具、笔记、代码、自动化串成了一个系统。这个系统有五层，像一栋楼。最底层是知识，最顶层是我从手机连进去工作。中间三层让一切自动运转。

* * *

这栋楼长什么样？五层架构一张图

想象一栋五层楼:

```
第5层：接入层 —— 我怎么连进系统
第4层：执行层 —— 工作在哪里发生
第3层：自动化层 —— 系统自己干活
第2层：持久层 —— 状态怎么存活
第1层：知识层 —— 理解怎么积累
```

从下往上，每一层都依赖下面的层。知识是地基，接入是屋顶。我们一层一层看。

* * *

第1层：知识层 —— 像图书馆一样管理你知道的一切

核心工具：**Obsidian**——一个纯文本笔记软件。

什么是 Obsidian?

简单说，它就是一个文件夹里的一堆文本文件。没有云端，没有数据库，文件就在你电脑上。但它有一个杀手级功能：**双向链接**。你可以像维基百科一样，在任何笔记里链接到另一个笔记。

我怎么用它?

我用一个叫 **Zettelkasten**（卡片盒）的方法来管理知识：

- **闪念笔记**：脑子里冒出什么，先记下来，不纠结格式
- **文献笔记**：读了一篇文章、看了一个视频，把理解写下来
- **永久笔记**：把前两种提炼成一个个独立的概念，每个概念一张卡片

比如"如何防止程序内存溢出"就是一张永久笔记。里面记录了：问题是什么、为什么发生、怎么解决、什么时候需要重新评估。

> **核心原则：一个事实只存一个地方。**
> 如果同一个信息存在两个地方，迟早有一个会过期。工具的使用方法存在工具自己的笔记页，架构决策存在架构文档，日记只放链接不复制内容。任何信息只需要更新一个地方，全系统自动保持一致。

还有一个重要理念：**被动消费不等于学习**。 AI 可以每天给你生成一份技术新闻摘要，但如果你只是看一眼就划过去，那不是你的知识，那是噪音。日记里只放你真正参与思考过的内容。

* * *

第2层：持久层 —— 让状态活过今天

核心工具：**Git + GitHub**（版本管理）、**CLAUDE.md**（AI 行为规则）。

我用 Claude Code——一个终端里的 AI 编程助手来写代码和管理系统。但 AI 没有记忆，每次开新对话都从零开始。

解决方案是三个文件：

- **CLAUDE.md**：写给 AI 的"员工手册"，规定它怎么写代码、怎么提交、怎么记录
- **MEMORY.md**：AI 的跨会话记忆，记住"用户喜欢简洁的回复"、"上次那个 bug 的根因是 X"
- **技能模板**：把重复的工作流程写成模板，AI 按模板执行就不会偏

所有这些文件都存在 Obsidian 里，通过 Git 同步到 GitHub。笔记本坏了？换一台，一个命令，所有配置和知识都回来了。

> **配置管理的关键：一个入口，不重复。**
> 所有 AI 配置都从 Obsidian 的一个文件夹管理，通过符号链接（类似快捷方式）映射到系统目录。改一个地方，所有地方生效。版本可控，可以回滚。

* * *

第3层：自动化层 —— 系统在你睡觉的时候干活

这是最有趣的一层。我设置了几个定时任务：

- **凌晨 1 点**：系统自我改进——自动研究最新技术趋势，跟现有系统对比，安全的改动直接应用
- **凌晨 3 点**：笔记库健康检查——找出过长的文件、过期的待办、断掉的链接
- **早上 7:42**：每日技术简报——扫描科技热点，只推送跟我当前工作相关的

摩擦检测机制

每次使用某个工具或流程时遇到不顺，我会记一条"摩擦日志"。系统自我改进的时候会读这些日志，优先修复反复出现的问题。

比如：某个技能模板的指令不清晰，导致 AI 每次都理解错。记录摩擦，下次自我改进时自动修复模板。**每次运行都让下次运行更顺畅**——这就是系统复利。

安全约束

系统可以自动改笔记、改配置、改代码，但有红线：永远不会自动推送代码到 GitHub（需要我确认），永远不会修改工作代码库（只改个人项目），结构性变化只标记不执行（需要审批）。结果通过 Telegram 发到手机，我可以在地铁上审核。

* * *

第4层：执行层 —— 工作真正发生的地方

核心工具：**Claude Code**（AI 编程助手）。

内存管理：一个血的教训

有一天 Claude Code 直接被系统杀掉了——占了 4.5GB 内存。原因？同一个插件在三个地方都配置了（全局、项目、插件系统），每次开会话都启动一份，从不清理。光浏览器调试插件就开了 5 个进程，吃掉近 1GB。

解决方案：**去重**。每个插件只在一个地方配置，零重复。外加养成好习惯——定期清理上下文、切换任务时重开会话。

云端 vs 本地 AI

不是所有任务都需要最强的 AI。我的划分：

- **高风险任务**（写代码、架构决策、复杂分析）→ 用云端 Claude API
- **低风险任务**（笔记库检查、文件分类）→ 用本地模型（免费、离线）

既省钱，又不牺牲重要工作的质量。

* * *

第5层：接入层 —— 从手机连进整个系统

这一层解决一个问题：**我不在电脑前，怎么继续工作？**

Tailscale：隐形的安全隧道

Tailscale 是一个免费的 VPN。装在笔记本和手机上之后，两个设备就像在同一个局域网里，无论你在咖啡店、在家、还是在国外。

安全特性：零信任模型——笔记本没有开放端口，路由器不需要端口转发。WireGuard 加密，跟企业 VPN 同一标准。流量点对点直连，不经过第三方服务器。

mosh：网络切换无感知

普通的远程连接用 TCP 协议。问题是：手机从 WiFi 切到 4G 时，IP 地址变了，连接就断了。终端卡住，你得重新连。

mosh 用 UDP 协议。UDP 没有"连接"的概念，所以 IP 变了它根本不在乎。你从 WiFi 走到地铁切到 4G，终端纹丝不动。而且它只发送屏幕的差异，不发完整画面。丢失的数据包直接跳过——反正旧画面已经过时了。

tmux：断网也不丢进度

如果手机彻底断网了呢？笔记本上的 tmux 会保持会话。AI 继续在笔记本上运行，不受影响。手机重新连上后，一个命令就能恢复——AI 可能已经把活干完了。

> **三层安全叠加：**
>
> ① Tailscale WireGuard 加密隧道 → ② mosh AES-128 加密 → ③ SSH 密钥认证。手机上不存储任何代码或凭证，只是一个显示终端文字的窗口。手机丢了？攻击者需要同时突破三道独立的锁。

流量消耗：用手机 4G 工作一小时，大约消耗 **10-30MB**。因为所有重活都在笔记本上通过家里宽带完成，手机只传输终端文字。1GB 的流量套餐可以用几十个小时。

* * *

额外设计：把个人系统当公司来管

我给系统设置了三个"高管"角色：

- **CTO（首席技术官）**：管理技术栈、自动化、配置
- **CFO（首席财务官）**：管理投资组合，整合股票和外汇

· **CEO（首席执行官）**：管理目标和决策，做季度回顾和决策日记

为什么？因为个人事务跟公司一样，需要不同视角。技术决策、财务决策、人生决策，混在一起只会互相干扰。分角色之后，每个角色有自己的仪表盘和工作流。

* * *

总结：这个系统的设计哲学

回头看这五层架构，其实就是几个简单的原则：

1. **一个事实只存一个地方** —— 信息不重复，就不会过期
2. **被动消费不等于学习** —— AI 生成的内容，你没参与思考，就不是你的
3. **摩擦就是改进信号** —— 记录每次不顺，让系统自动修复
4. **安全不是一道墙，是多道锁** —— 每一层都有自己的保护
5. **系统在你睡觉的时候进化** —— 自动化不是省事，是让改进变成习惯

这个系统不是一天建成的。它从一个笔记软件开始，慢慢长出了五层。每一层都是因为遇到了真实的问题才加上去的。

你不需要照搬这个系统。但如果你也有"每天都在重新发明轮子"的感觉，也许可以从最简单的一步开始：**今天做了一个决策？写下来。写清楚为什么。** 下次你就不用再想一遍了。

写于 2026 年 4 月 11 日。作者是一个相信好记性不如烂笔头的软件工程师。

GitHub: github.com/xinyun2020 | 原文首发于作者个人知识库