

终点不是转写稿

一段口述怎么长成作品——VoiceDrop 的 740 次提交

王建硕

费曼式写法 · 由多个 AI 代理撰写与互相审校

导读

一段口述怎么长成作品——VoiceDrop 的 740 次提交

先别想 AI，先想一件更土的事：你刚才在路上想通了一件事，掏出手机说了三分钟。

现在这三分钟在哪儿？

按今天所有语音产品的设计，它会变成一份**转写稿**——可能还附一段摘要、几条待办。这份东西的用处是「回头你还找得到」。它是素材。要变成能给别人看的东西，你还得再坐下来一小时。

这本书讲的是另一个选择：让那三分钟直接变成一篇**能发出去的文章**——带你自己的笔法，带你说话时随手拍的那张照片，带题图和摘要，一按就躺进公众号草稿箱。中间的转写、挖矿、配图、封面，一个都不出现在界面上。你看到的只有：一个红键，和一篇文章。

这个差别听上去只是「多做了一步」。它不是。**终点在哪儿，形态就长成什么样**。把终点从素材挪到成品，会逼出一连串在原来那条路上根本不会出现的东西：一份会随你一起长大的个人文风、一套能让你对着文章说「把第三行改口语一点」而它真的改对第三行的协议、一台专门为了能说一句真话而架在东京的服务器、一个把每一次调用的成本明码标给你看的计费系统、以及——一句话就能长成一本书、并且直接出现在公开书架上的流水线。

你现在读的这本，就是那条流水线跑出来的。

大白话

贯穿全书的问题只有一个：**为什么几乎所有语音笔记产品的终点都是一份转写稿，而这一个的终点是一件作品？**读完你应该能自己回答，并且知道这个选择要付出什么代价。

材料是硬的。2026 年 6 月 18 日到 9 月 1 日，76 天，740 次提交，从一个 `chore: init` 长到 App Store 上的 1.13 版。每一章都从这些提交里来——包括那些做完又整个删掉的，那些上线当天就回滚的，那些查了一整天最后发现假设本身很无聊的。

它不是一本产品宣传册。第十四章会老老实实告诉你，这里面有哪些东西任何对手一周就能抄走，以及这条护城河最脆弱的地方在哪儿。

从一个只有一个动作的入口开始。

第一章 · 一个只有一个动作的入口

2026 年 6 月 18 日上午 11 点 31 分，仓库被 `chore: init voicedrop repo`（`20f7121`）建了起来；八分钟后，第一行产品代码落地（`760f573`）。跟它一起进仓库的设计文档只有一节叫「一句话」：

打开 iPhone App 即开始录音，点停止即自动把录音上传。结束。

另一节叫「不做」，列着列表、播放、剪辑、波形、账号、设置页。这一章讲的是，为了让仅剩的这一个动作真的可靠，后面付了多少代价。

为什么入口只能有一个动作

口述有个物理特性：**想法的保鲜期只有几秒**。掏出手机还要解锁、找 App、等加载、点进某页、再找录音键，那个想法通常在第三步就凉了——录下来的不是它，是它的追述版。所以入口的目标是**把「想录」到「正在录」的步数压到最少**。第一版压到了 0：一启动就录，黑底屏幕上只有一个计时器和一个停止钮。

两天后它被推翻（`023c5a2`）：改成点红键才录，状态机新增一个 `.idle` 相位，App 打开后停在准备屏等你按。

0 步比 1 步好，这在纸上显然，在口袋里不是。一打开就录意味着**每次误触都生成一条录音**：你只想看看昨天那条转写成什么样，结果又录了 8 秒口袋噪音，而它照样上传、转写、被送去让模型判断能不能成文。省下的那一步是从用户身上省的，垃圾却是全流水线一起付账。

大白话

把一个动作省到零，代价是把「用户根本没打算做这件事」也一起省掉了。第 1 步不是摩擦，是意图声明。

文件名即元数据

第一版文件名是 `rec-2026-06-18-114023.m4a`，够排序、够唯一，仅此而已。47 分钟后（`02abf12`）它变成：

```
VoiceDrop-2026-06-18-143052-0m33s-Thu-Afternoon-Shanghai-Xuhui.m4a
```

前缀、开始时间、时长 33 秒、周四、下午、上海徐汇。全 ASCII，无空格无中文无标点，穿过 URL、对象存储的 key、`curl` 都不用转义。末尾两段地名来自 CoreLocation（iOS 定位框架）加一次反向地理编码（把经纬度换成人能读的地名），而这一步有条硬约束写在代码里：**3 秒超时，绝不阻塞录音**。定位慢、没信号、没权限，文件名就少这两段。地名是锦上添花，锦不能等花。

它真正的价值在于，这是**三方共用的同一份契约**，而三方谁也不认识谁：

- **iOS App**：`RecordingName.make()` 生成、`parse()` 读回，全 App 只有这一处知道字段顺序。
- **服务端挖矿程序**：一个正则从名字里抠出时长，**不下载音频就知道这段有多长**，直接拿去算超时、算算力预估；看图写文那条路径还读出星期、时段、地点当写作素材。
- **Mac 端的挖矿 skill**：只认 `VoiceDrop-` 前缀加 `.m4a` 后缀，桶里其它文件一律不碰。

大白话

没有任何一张数据库表记着「这段是周四下午在徐汇录的、33 秒」。这些信息就写在名字上。名字本身就是那张表。

代价是文件名成了公共接口：改一个字段的位置，三个程序同时错——所以解析收进唯一一个 `parse()`：契约可以脆，脆点必须只有一个。

同一个目录，既是录音棚又是发货区

第一版还有个漂亮的简化：**Documents 目录本身就是上传队列**——不要索引、不要状态表，文件还在盘上就是还没传。以及一个同样漂亮的决定：正在录的文件一开始就叫正式名

字，注释写得明明白白——万一录到一半 App 被杀，它已经是合法的队列项了。为崩溃留后路，听上去无懈可击。

两天后，246d9a2 记下后果：Documents 目录同时充当「正在录音的位置」和「上传队列」，而队列的过滤器匹配的正是此刻正被写入的那个文件。App 启动时并发做两件事——传积压文件，和开始录音——于是队列扫到那条**正在写**的录音，把它 PUT 上服务器，再按「传完就删」的规矩，从还开着的录音器脚底下把文件删了。

传上去的东西是坏的。m4a（iPhone 录音的文件格式）里有一小块叫 moov atom 的索引，记着音频多长、每一帧在文件的什么位置，而它**录完、关闭文件那一刻才写进去**。录到一半的文件有音频数据却没有目录页，播放器打开只会说一句 `moov atom not found`。

大白话

一本书正在印，装订工把还没印目录的半成品发了货，顺手把车间里的原稿也销毁了。

让它不可能，而不是让它不容易

这个 bug 有一堆看起来合理的修法：给队列加锁、录音后延迟几秒再扫、加一个「正在写」的标记位。共同点是——**都在管理这场竞争，而不是取消它**。加了锁，你就得永远保证以后每条新路径都记得拿锁。

实际的修法是改名字：正在录的文件叫 `recording-<时间戳>.m4a`，队列只认 `VoiceDrop-` 开头的；录音器关掉文件之后才改成正式富名。commit message 里那句话是这本书的第一个思想高地：

Makes the race structurally impossible.（让这场竞态在结构上不可能发生。）

队列不是「决定不去碰」正在录的文件，是**根本看不见它**。这条修法不要求任何人遵守纪律：以后新加的路径无论怎么写，都扫不到一个不匹配前缀的文件。**修 bug 的最高级别，是让这个 bug 没有存在的位置。**

同一个 commit 还加了两层，凑成三重防守：**暂存名隔离**让竞态不可能；**启动清理残片**把上次进程被杀留下的半截文件挡在队列外；**isUploadable()** 在每次 PUT 前实扫文件，0 字节的、找不到 **moov** 的一律拒传并踢出队列。

一个坏文件能跑多远

竞态修好那天下午 4 点整（f93a9ca），另一个 commit 在服务端善后。现象是挖矿脚本单次要跑 30 多分钟，因果链：坏文件解码出 0 字节音频 → 把 0 字节喂给流式语音识别的 WebSocket → 那个接收循环**挂死约 15 分钟**才崩掉 → 四个这样的文件就是一小时。一个 iPhone 上的并发时序问题，跨过网络、跨过语言，变成了服务器一小时空转和真金白银的账单。

修法两条：解码出来不足 0.5 秒的音频直接以退出码 3 结束，**连那个注定挂死的连接都不去建**；再给子进程套一个「2 倍时长 + 2 分钟」的硬超时。**快速失败比顽强重试值钱**——顽强重试遇到不可能成功的输入，会把「不可能」变成「很慢」，而很慢是最难查的故障，因为它看上去还活着。

44.1 kHz 是纯浪费

第一版录音参数抄的是默认值：44.1 kHz 采样、64 kbps 码率——来自 CD 音质，来自「音频就该这么录」的惯性。6 月 28 日（64b73a2）改成 16 kHz / 32 kbps，理由在注释里只有一句：这段音频只喂两个消费者——App 里的回放，和豆包 ASR（自动语音识别，把声音转成文字），**而后者原生就工作在 16 kHz**。多出来的那一半频段，编码器花码率存下来，识别引擎第一步就重采样扔掉。5 分钟录音从 2.4 MB 降到 1.2 MB，上传时间对半砍，准确率不变。

大白话

参数不该按「这个领域默认多少」定，该按「下游到底要什么」定。没人消费的精度不是质量，是账单。

同一条逻辑一个月后又用了一次（be090ea）：**不足 4 秒的录音直接不上传**，本地删掉，屏幕上说太短。4 秒的口误产不出文章，传上去只会白走一遍流水线——转写、扣算力、让模

型判断、写一个「没检测到语音」的标记——最后给用户一个空欢喜。拦截收在唯一一处（`RecordingPromoter.promote`），停止、中断、退出、社区回复四条路径全经过它。

导航范式反转：谁是根

6 月 22 日傍晚（`7d98fca`），界面结构被整个翻过来。翻之前**录音页是根**，文章列表是从底下拉起的 sheet——完全符合「打开就录」；翻之后**列表是根**，底部悬浮一颗纯红圆键，按下去全屏录音页接管。

为什么反转是对的？因为「打开就录」优化的是**第一次**使用，「打开看看」才是**第二天以后每一次**的使用：录音只占几十秒，读、改、发文章占掉剩下的全部时间。而红键让反转没有代价——它浮在列表之上，永远在拇指够得着的地方，一步。一小时后又跟一个 commit（`81e8e0b`）把它下面的背景板也删了。**入口是一个按钮，不是一块区域。**

后台录音，和它的账单

第一天下午（`0c28b3a`）有个只改两行配置的 commit：声明 `UIBackgroundModes: audio`，告诉 iOS 这个 App 要在后台处理音频，切走或锁屏录音便不会被挂起。这是必需品：你不能要求人在说话的 25 分钟里一直盯着屏幕。

账单两周多以后寄到：App Store 审核以 2.5.4 条款拒绝，理由是「plays no audible content」——这权限是给**播放**音频的 App 用的，而你不播放任何声音。规则没错，只是它假设了后台音频等于播放。回应不是删权限（删了产品就废了），而是录一段真机屏幕录像，证明回主屏幕后录音还在走。**平台规则按典型用例写；用例不典型时，你付的不是代码，是举证。**

最里面那一层

7 月 8 日凌晨（`69e737a`）加了 AI 采访，麦克风的数据流第一次有了第二个消费者。底层换成 `AVAudioEngine`，每来一小段音频都回调一个叫 `tap` 的闭包（相当于在水管上开的取样口）。闭包里第一行的注释只有两个词：

```
self.write(buffer) // SACRED: file first
```

先写文件，再考虑要不要分一份给 AI；AI 那一路挂了也不影响已经写进文件的字节。**架构的本质是排序：把不可丢的放在最里面。**（这条原则第 7 章会展开。）

「打开就录」是一句口号，兑现它花掉的是上面这一整章。**入口只有一个动作，是因为它背后有很多个动作，只是没有一个落在用户身上。**

第二章 · 终点决定形态

2026 年 9 月，我把这赛道能找到的 25 个产品排了一张表：核心输出物、定价、学不学你的文风、能不能改稿、能不能发布、有没有社区。排完发现第一列整齐得反常。

Otter 输出转写稿加摘要加待办事项。飞书妙记输出转写稿加 AI 纪要。AudioPen 输出一段整洁的短笔记。Superwhisper 把字粘到光标处。

25 个产品，终点几乎是同一样东西：**一份你还得再动手一次的材料**。

这不是因为别人不够聪明——Otter 2025 年 3 月过了一亿美元的 ARR（年度经常性收入，「这生意一年稳定收多少钱」），Granola 2026 年 3 月按 15 亿美元估值融了 1.25 亿。他们停在那儿，是因为有四股力量摠着。

三个族群，三个终点

会议纪要族（Otter、Granola、飞书妙记、通义听悟、腾讯会议 AI 小助手）的终点是**一份可检索的记录**。Otter 2025 年底的新闻稿把自己定位成「企业的会议知识库」——知识库这词已说全：存进去，搜得到。

个人笔记族（AudioPen、Voicenotes、Whisper Memos）做的是个人版的同一件事：把你嘴里那段颠三倒四的话变成一段整洁笔记，终点是**三个月后你自己还看得懂**。AudioPen 做得相当好，是表里唯一沾到「学你」边的一家。但产物仍是笔记，要发出去还得再写。

「更快的键盘」族（Superwhisper、Wispr Flow、Aqua Voice、MacWhisper）最诚实——连摘要都不做，明说自己就是个更快的输入法：你说话，字出现在光标处，**「这段字要变成什么」它主动不管**。资本很认这条线（Wispr Flow 2026 年 2.6 亿美元 B 轮，总融资 3.15 亿），但内容形态被整个让出去了。

大白话

翻译成成人话：一族帮你记住开了什么会，一族帮你记住你想过什么，一族帮你打字快一点。三族都停在你的动作**之前**，没有一族走到「这东西可以给别人看了」。

四股力量

这四条不是并列词条，是一条因果链的四个环。

第一环：转写有对错，好文章没有

转写有客观正确答案。业内用 WER（词错误率，把机器转的文本跟人工标准答案逐词比，错一个算一个）打分：一个数字，可测、可比、可以写进合同。摘要模糊些，但也有大致共识。

「一篇好文章」没有这种东西。你没法给它定 SLA（服务等级协议，白纸黑字承诺「我至少做到这个水平」，做不到赔钱），产品经理也拿不出可做 A/B 测试的指标。于是**停在摘要，是整条链路上责任最轻的位置**——再往前一步就得为「写得好不好」负责，而这事没有裁判。绝大多数公司在这儿本能刹车，完全理性。

第二环：付钱的是企业，不是作者

看价格结构就明白了。Otter Business 每座位每月 30 美元（年付折合 19.99），腾讯会议专业版按年折合每月每账号 82.34 元。**钱在 B 端，按座位收。**

企业买它是为了合规存档和事后检索，不需要有个人味道的文章——恰恰相反，它要的是中性、无个性、谁读都一样理解的文本。产品会长成付钱那个人的形状：为按座位付费的组织优化，必然长成一个纪要工具。这不是选择，是重力。

第三环：通用工具的诅咒

要服务所有人的工具只能输出所有人都能接受的文本，而那必然是中性的、没口音的、谁写都差不多的。

假设一个通用 SaaS 的产品经理提议：我们学每个用户的文风吧。这功能对新用户毫无价值（没样本），对不写作的用户——绝大多数会议参与者——永远没有价值，还让产出不可预测：同一段录音，两个人拿到的纪要不一样。

在通用 SaaS 的逻辑里，「学某一个人」是减分项。只有当你的用户就是「一个要署自己名字发东西的人」时，它才翻转成加分项，而且是最大的那个。

这解释了整张表最刺眼的一列：学文风一栏几乎全是「否」，剩下的是预设选择器——Superwhisper 的 mode、AudioPen 的简明／学术／莎士比亚。**从下拉框里挑一个别人的风格，和学会你自己的风格，是两件根本不同的事。**

第四环：发布是脏活

把一篇文章真送进公众号草稿箱，要做这些事：申请开发者资质、走 IP 白名单、拿 access token 并在过期前刷新、把 Markdown 转成微信编辑器认的 HTML（它会把列表里的裸文本节点吃成空标签）、每张图先传成素材换 media_id、摸清审核规则。

这堆活**不可复用**（换个平台全部重来）、**有地域性**（只对中国有意义）、**纯苦力**（毫无技术优雅，也不会出现在任何一页融资材料上）。

结果是：没有一家美国公司愿意做，中国的会议工具公司又不认为这是自己该做的。整张表里唯一直连公众号的，是壹伴、135 这样的 AI 编辑器——但它们的入口是关键词，不是你的口述，产出是一眼可辨的通用 AI 腔。

四股力指向同一个方向，互相加固。

市场数据在说同一件事

有意思的是，钱正往相反方向流。语音转文字 API 市场，Grand View Research 的数字是 2024 年 38.135 亿美元、2030 年 85.694 亿，CAGR（复合年增长率，把多年增长摊成「平均每年涨百分之几」）14.4%。而 AI 会议助手（也就是「转写之后的处理」）Research and Markets 给的是 2026 年 39.1 亿到 2030 年 93.3 亿，CAGR 24.3%，各家机构的估计集中在 24%-28%。

下游的增速差不多是上游的两倍。转写正在变成大宗商品：讯飞听见机器快转 0.33 元一分钟，通义听悟的 ASR 0.6 元一小时——当一件事按元每小时论斤卖，它就不再是生意的中心了。那下游最尽头是什么？是成品，是发表。而那个位置**现在没人站**。

另一个数据是 Plaud。这家做录音硬件的公司，据媒体口径 2024 年营收约 5600 万美元、2025 年约 2.5 亿、估值约 20 亿，且没拿过 VC，客户不是企业座位而是一个一个的个体。它证明了**个人内容场景有真金白银**——只是所有人都把这笔钱拿去卖硬件和卖转写，没人拿去卖成品。

素材，还是成品

素材是你拿到手之后**还得再做一次工作**的东西。转写稿是素材：有口水话、有重复、没结构。摘要也是素材——它抹掉了细节和语气，你得往回填。Sudowrite、Novelcrafter 这类 AI 写作台的长稿也是素材，定位就是「给你继续改的工作台」。

成品是你拿到手之后**可以直接交出去**的东西。

两者的距离不是质量差 20%，是**差一整个人的一小时**。而那一小时恰恰是绝大多数灵感真正死掉的地方——不是死在没记下来，是死在记下来之后没人把它变成东西。

停在素材，产品的责任是「转得准不准」。走到成品，责任变成「这东西能不能署你的名字发出去」。这一步跨过去，所有设计取舍的源头就都换了。

把终点挪到哪儿

VoiceDrop 的终点是**一篇现在就可以发出去的成品**。

带你自己的文风，不是下拉框里的第七个选项。带你说话时顺手拍的照片，插在正文里描述那个场景的地方。带题图和摘要。一按，躺进公众号草稿箱。再往前一步，一句话能长成一整本书，直接出现在公开书架上。

而中间的一切——语音识别、把转写稿挖成文章、配图、选封面、写摘要、算钱——**一律不出现在界面上**。用户看得到的只有两样：一个红键，和一会儿之后出现的一篇文章。

这不是审美偏好，是终点定义的直接后果：终点在素材，中间产物必须露给你看，因为你要接手；终点在成品，中间产物露出来就是噪音。

查过一遍的三处空白，两处半空白

反着查一遍这张表：如果这条路真通，为什么没人走？

- **从一段口述直达可发表成品**——没找到第二家。最逼近的是国内公众号 AI 编辑器，但入口是关键词不是口述；AI 写作台产出长稿，却明确是待改的工作台。真空白。

- **用语音指令改一篇已写完的稿子，还能定位到某一行、某一张图**——本次调研未查到第二家。Aqua Voice 有「说话改文字」，但它改的是还在输入流里、你刚说出口的那段；AudioPen 能 rework，但那是按钮而且整篇重来；会议工具只允许「对纪要提问」，不允许「改纪要」。最干净的一处空白。
- **一句话生成整本书并公开发布**——未查到任何产品在做。现有 AI 写书工具是给专业小说作者的长稿工作台：人搭大纲、逐章生成、导出后自己去平台自出版。真空白。

还有两处只能算**半个**空白：

- **学用户本人的文风**——AudioPen 沾边，它的自定义 style 允许上传自己过去的写作样本（官网这页需登录，未能直接核实），但停在「一次性喂样本 → 一个风格槽位」，不是一份随你不断写作而进化的资产。
- **成本明码标价**——credits、积分、算力在中国的 AI 消费品里早是成熟惯例：Notion 的 Custom Agents 每月 10 美元换 1000 credits，腾讯 ima 的 copilot 干脆就叫「算力」。VoiceDrop 的 23 算力 = 1 元只是把它做得更实。**单位一点不新，新的只是用在语音笔记这个赛道**——赛道内部清一色「Pro 版无限用」的包月制，按量透明计费为零。

哪些不是差异化

「打开就是一个红键，松手自动上传」是交互层的东西，Voicenotes 和 AudioPen 早就这么干了。第一章讲它，是因为它是这条流水线的入口，不是因为它稀缺。「按算力明码标价」同理，对手一周就能跟进。

真正抄起来痛的，是把文章结构化到行和图的粒度、把一句口语指令解析成一处定点修改、再配上版本回滚——那是工程纵深，不是一段提示词。以及那份越写越像你的文风资产：**护城河不在代码里，在用户身上，而用户走的时候带不走。**

这条要展开得等第 14 章。本章只需立住那个问题：

所有人都停在转写稿，是因为再往前一步没有裁判、没有企业客户、会牺牲通用性，而且全是脏活。**那如果只服务一个要署名发东西的人，这四条约束是不是就全都不成立了？**

接下来的十几章，都在回答这个问题的具体后果。

第三章 · 挖矿

一段二十分钟的口述里，往往埋着三件不相干的事：昨天那场会为什么开砸、孩子学琴该不该继续、一个突然冒出来的点子。它们唯一的共同点是共用了一条录音。

大多数语音产品把它还成一整块转写稿。VoiceDrop 要做的是把三个话题分别挖出来各自成篇——矿石不是一整块金子，是石头里散着几处矿脉。

一个没有数据库的数据库

链子不复杂：App 录完音，文件落到云存储 R2 里你自己的目录；服务端把音频送去做 ASR（automatic speech recognition，把声音变成文字）；文字连同你的文风交给大模型；挖出来的文章写回同一目录，成为 `articles/<stem>.json`。

关键这里是**没有后端数据库**。没有一张表记着「这条录音处理过了吗」，判断依据只有一条：目录里有没有对应的文章 JSON，或者一个叫 `.empty` 的空标记文件（a8b86ba）。

大白话

翻译成人话：不设台账，用「东西在不在货架上」当台账。

好处是全系统只有一份真相，而它就是用户能直接下载走的那些文件；代价是所有状态都得表达成「一个文件在不在」——后面的麻烦都在为它交学费。

一录多篇：挖矿这个词才成立

第一版一段录音出一篇。跳跃在 v2（f4922f0）：用 ASR 返回的 utterance（一句话的起止时间）生成真正的 SRT（带时间码的字幕，每句标着从第几秒到第几秒）；同时要求模型按**明显不同的主题**拆成多篇独立文章。输出单位从此不是「录音」而是「话题」。

事件驱动 + 兜底定时

触发方式改了四次：每小时 → 每两小时 → 改回每小时 → 上传完成立刻触发 (a58e62a) → 上传触发加每 6 小时兜底 (5568fbe)。纯定时的问题是延迟；纯事件驱动的问题是漏——那次触发万一失败，录音就永远躺着，没有第二个事件叫醒它。**事件负责快，定时负责不漏。**而这组合能成立，恰恰因为兜底扫描不必知道哪条失败过，只重新问一遍「谁的目录里还缺文章」。**幂等，是重试敢存在的前提。**

让模型吐出 JSON 的三次尝试

第一版靠 prompt 求「请只输出 JSON」。模型照做了，但习惯性把它包在 Markdown 代码围栏里 (``json 开头)。老解析器认不出，走了 fallback，而那个 fallback 把模型的原始回复当正文存了下来——用户打开 App，看到自己的正文是 ``json 开头的一坨 (4171ea0)。

第二次用 assistant prefill：替模型把回复的第一个字符写成 {，它就只能顺着写 JSON。模型直接拒绝：400 conversation must end with a user message (ff751e4)。第三次才对——用 structured outputs (在 API 层给回复挂一张 schema，让不合法的输出根本生不出来) 从根上约束 (748822b)。

比方案更值钱的是那条铁律：**永远不要把模型的原话当成文章存起来**，解析失败宁可什么都不存、下轮重试。「没有结果」可恢复，「错误的结果被写进了正确的位置」不可恢复——系统会认为它处理完了，再不回头看。

空录音的几种死法

.empty 的 reason 攒出五种：corrupt (解不开)、silent (不到 1 秒)、no-speech (ASR 返回空)、too-short (转写不足 20 字，跳过大模型不花那个钱)、no-article (模型认为不值得成文)。后来又添了第六种 asr-error:<码>，下面会说。

第一个坑是**永久重试的无底洞** (5ea87e1、298c7b9)。`{"articles": []}` 是合法 JSON，意思是「读完了，没有值得写的」；JSON 解析失败则是「没读懂模型说了什么」。早期代码把两者混成同一个 `RuntimeError`，录音永远 FAILED，每轮重跑、重付一次 ASR 和大模型的钱。

修法是拆开：前者写 `.empty` 标记，从此算处理完。**确定性的失败和暂时性的失败必须分开**——前者留墓碑，后者才该重试。这条原则在 ASR 上原样重演（79ad2c7 → da0b213）：业务错误码送一百次还是那个码，exit 4 写 `.empty(reason=asr-error:<code>)` 不再重试；网络超时和限流 exit 1，下轮再来。

第二个坑是**我们在骗用户**（736ade3）。这么多原因，详情页却一律显示「没检测到语音」：有人认真说了 49 秒，只是模型觉得不够成篇，App 却说「没听到你说话」——系统把自己的判断伪装成了设备故障。修法是按 reason 分开写文案，各说各的实话。

高光事故：风格卡变成了质量闸门

用户可以上传一份「文风卡」，挖矿时拼在系统提示词后面。结构化输出留了个逃生口：内容不够成文时输出 `{articles:[]}`。prompt 里还写着：每篇 800-1000 字、不许编造。

三件事单看都对，凑一起出了事：一段 33 个字的真实想法，既写不到 800 字又不许编内容来凑，模型于是走了逃生口，用户看到「无语音」——他明明说了话。

更深一层：那份详细文风卡被模型当成了**「够不够格写」的判据**，而不是**「怎么写」的指南**。你越认真描述自己的写作标准，它就越严格地拿这标准审判你随口说的话。

修法两步（c6108c9、c7aef96）：去掉字数硬指标，让长度随素材走；再改成两段式——先带文风跑一遍，若判空，用一个毫无风格约束的极简 prompt 重跑，两次都空才写 `.empty`。

风格规则管的是 HOW，不是 WHETHER。

你给模型的每一句约束，都可能被当成打分标准而不是执行指南。

静音的录音教会我们什么叫「完成信号」

ASR 是异步的：提交任务，轮询问「好了吗」。早期的依据是返回结果里 `result.text` 非空。这在 99% 的情况都对，直到有人上传一秒钟的静音——静音的正确转写结果**本来就是空文本**，循环永远等不到非空，每条都死等 600 秒（2c233f0）。连改四版全在猜，最后抓

实时响应体才看明白：真正的完成信号是 `audio_info` ——处理中是 `{}`，完成时带上 `duration`，哪怕全程静音。

大白话

翻译成成人话：别拿「有没有内容」当「有没有做完」。**空结果也是结果。**

完成信号必须跟结果内容正交——「做完必然有产出」这个假设，总会被一个合法的空结果打脸。

换掉方案，比修那个方案更省事

最早的 ASR 走流式 WebSocket：服务端先下载 m4a、转成 PCM、再一帧帧推过去。连锁麻烦里最滑稽的是「ubuntu-latest 到底预装没预装 ffmpeg」——一天之内改了五次（3dbec95 → f71fba7）。终结它的不是第六次修改，是换方案（41e681d）：改用火山的「录音文件识别」异步 API，给一个 R2 预签名 URL，让服务商**自己去把文件拉走。在同一个地方连改五次还没修好，往往不是手笨，而是这条路径背了太多不属于它的责任。**

搬家，以及删掉 1025 行

挖矿最早跑在 GitHub Actions 上，白嫖一台构建机——它能用，但它是「构建 CI」不是「运行时」：启动慢、并发难控、状态没地方放。后来整个搬进 Cloudflare Worker 的一个 Durable Object（DO，可理解成「全球唯一、自带一小块持久内存、请求都路由到同一实例的对象」）：`idFromName("miner")` 让所有触发落到同一个实例，配上 `alarm()` 定时回调，天然把并发变成串行（b039914、3a793a7）。

大白话

翻译成成人话：不用加锁，因为压根只有一个人在干活。

搬完顺手删掉 1025 行的 `mine.py`，但没有偷偷删——立了块碑 `mining/REMOVED.md`，写清怎么从 git 恢复（3640e2d）。**删代码是最好的架构变更**，留块墓碑是基本礼貌。

云存储的列表会说谎

搬完之后出了件有意思的事 (56cdd72)：第 318 次挖矿写了 10 篇文章；12 秒后，上传触发的第 322 次挖矿去列目录，**看不见那 10 个文件**，把 10 篇全重挖一遍。原因是对象存储的两种一致性不一样：`list()` 是最终一致的，写进去的过一会儿才出现在列表里；`HEAD`（问「这个 key 在不在」）是强一致的。修法不是加锁，而是每条录音开挖前**再问一次那个说真话的接口**。

看图挖矿：iOS 一行没改

从别的 App 分享一张图片进来，它给你写一篇文章。做法很讨巧 (727aa91、1c51d1e)：客户端伪造一段**静音的占位 m4a**，加一个同时间戳的照片目录——这形状和「录了段静音、期间拍了照」一模一样，于是天然走完整条既有管线。服务端只加一处分支：**无语音、有照片 → 看图写文**。

这是复用的最高形态：不是抽个公共函数让两条路径都调，而是**把新东西伪装成旧东西该有的样子**。前提是数据形态够朴素——当初若塞过一个「这是不是真录音」的字段，这招就没了。

追问：问题存在哪，决定了它会漏到哪

2026 年 7 月上线的「追问」：文章挖出来后，AI 针对每篇提 1 到 3 个**只有作者本人知道答案**的细节问题，你按住说话回答，答案被织进正文。值得说的不是问题怎么生成，而是**问题存在哪里**。最初评审过一版：写在正文末尾加一节「——追问——」。否掉的理由很直白——正文是要出门的：它会被推到公众号草稿、分享页、社区、小红书图文包，只要问题在正文里就会从**每一个出口**漏出去。最终形态是存成文档顶层的 `sidecar` 字段 `questions`，跟正文彻底分家，于是所有出口**天然**不带它。**与其在每个门口安检，不如别让它进大厅**。

顺带一条硬知识：Anthropic 的结构化输出 schema **不支持数组的 `maxItems`**，写了线上直接 400。上限只能靠 prompt 加代码截断——**能写进 schema 的才是硬保证，写不进的只是祈祷**。

还有一道闸：挖矿动手前要过 `meteredMineGate`，卡在「已处理就跳过」之后、ASR 之前，余额不够就写个**非终态的** `.blocked` 标记停下，余额够了删掉标记就能重挖。算力怎

么记账是第八章的事——挖矿是唯一会自己花掉用户钱的动作，必须有一道能拦住也能解开的闸。

矿工的手艺不在于挖得多快，而在于分得清哪块是矿、哪块是石头，以及——挖到石头的时候，老老实实说这是石头。

第四章 · 文风是一份资产

下拉菜单和资产，是两种东西

市面上的语音笔记产品几乎都给你一个下拉菜单：正式、随意、学术、莎士比亚。选一个，它按那个腔调写一段。逻辑是「挑件衣服穿」——挑完就完了，下次再挑。

VoiceDrop 走另一条路：你的文风是一份**资产**。它有版本号、有历史、有一个不断长大的语料库，还会被盖到每一篇文章上。不是你每次挑的衣服，是你的骨相。预设风格存在一个变量里；个人文风需要一整套数据结构。

起点：一个 Markdown 文件

2026 年 6 月 20 日，00750ea。做法土得可爱：给每个用户在云上存一个 CLAUDE.md——纯文本文件，写两段：「我的名字」和「我的文风」。设置页填的直接落进去，挖矿时把它**追加到 system prompt 后面**。

大白话

system prompt 就是每次请求模型前先塞给它的一段固定说明书，告诉它「你是谁、按什么规矩干活」。用户的文风就贴在这份说明书的末尾。

目标只有一句：**让每个人的文章是自己的味道**。

它立刻露出一个副作用：风格卡越详细，越容易被模型当成「够不够格写」的判据。那是第 3 章的事故，这里只留一句：**文风是「怎么写」，不是「写不写」**。

为什么一段描述不够用

纯文本字段的问题不在于存不下，在于它**只有一个当前值**。人的文风不是一次定稿的：今天调得更口语，明天觉得过头想退回去。一个字段只记得住最后一次覆盖，也说不清「这篇当初是按哪一版写的」。

6 月 29 日，4ea8e43：CLAUDE.md 升级成 CLAUDE.json。关键不在换后缀，在于它用的是**和文章一模一样的版本信封**：

```
{schema, head, versions:[{v, savedAt, source, style}], createdAt,
updatedAt}
```

大白话

schema 是「这份数据长什么样」的版本号，告诉程序按哪一代格式解析，这里是第 3 代。

versions 是只增不减的数组，每存一次多一条；head 是个指针，只存一个数字，指着「现在生效的是第几版」——把 versions 想成一排编号的抽屉，head 就是贴上面的「当前使用」便签。

这个设计带来两条省事的规矩：

- **切版本 = 挪便签，不铸新抽屉**（7aa1046）。从 v7 切回 v3，只是把 head 从 7 改成 3。早期恰恰做错了——切一次存一版，切三次就多三个重复版本。
- **只在真改动时存新版本**（66d0db6）。打开编辑器看一眼又关掉，不该留痕。

为什么复用文章那套版本机制？因为**同一个通用机制用两次，比造两套省得多**。文章已经有版本、有 undo/redo（撤销重做，本质就是把 head 往前后挪），服务端的 `style-store.js` 干脆就是 `article-store.js` 的镜像。后来的「多风格对比」也因此几乎没有新结构（它已整体下线，见第 12 章）。

加法式兼容：让老文件自然退休

换格式最容易翻车的是存量。常规答案是写迁移脚本挑个凌晨全转一遍，VoiceDrop 没这么干（2026-06-29），规则只有两条：**写方只写 CLAUDE.json；读方优先读**

CLAUDE.json，读不到就回落解析老 CLAUDE.md 的「# 我的文风」段。

于是老文件在用户**第一次保存文风的那一刻自然退休**——保存新文件就有了，回落分支从此走不到。不用停机，不用担心脚本半路挂掉。这叫**加法式兼容**：让新旧共存到旧的那份自然死亡。

顺带一件诚实的事：文风搬家的那一刻，名字被留在了老 `CLAUDE.md` 里——只搬了文风没搬名字，而社区署名等几处还在用那个文件的正则抠名字，`STATE.md` 就老实写着「名字的**永久归宿 TBD**」。它后来才落进 `CLAUDE.json` 的一个不参与版本的字段里，老文件退化回落。技术债不可耻，记在明处才是还得起的债。

语料从哪来：任何 App 里的分享按钮

让用户手写「我的文风」，写出来的多半是「简洁、有力」这类正确的废话。真正的文风藏在他**写过的东西**里。

入口做成了 iOS 的 Share Extension——你在任何 App 里点「分享」时弹出的面板里多出来的那个 VoiceDrop 图标。在微信里翻到自己三年前写的长文，点分享选它，那篇就成了语料。

服务端三条路由撑着：`POST style/collect` 收语料，`GET style/dataset` 列出攒了多少（只返标题、字数、来源），`DELETE style/dataset` 清空。对应界面是「风格数据集」（4c4f868）。

这里有个干脆的决定：**内容类型即路由**（727aa91）——**不让用户选用途**：音频和图片去挖文章，文字、链接、PDF/docx 去当语料。答案本来就唯一。多问一句「你想拿它做什么」，是把系统的犹豫转嫁给用户。

提取风格：从「调个接口」改成「走主干」

攒够语料，点「提取文章风格」，模型把样本蒸馏成一版新文风。第一版实现是调一个专门的异步接口，问题是这条支线太细——同步调用让分享页干等着转圈；改成 `waitUntil`（响应返回后继续跑的后台任务）又**静默失败**：用户点完按钮页面关了，什么也没发生，连个错都看不到。

7 月 2 日，5d72482 换了思路：**把「提取风格」当成一条录音，走同一条挖矿流水线**。做法是传一段 960 字节的静音占位音频，任务类型烧进文件名——前缀 `VoiceDrop-style-`、尾部 token `TaskStyleExtract`，服务端 `classifyKey` 看一眼就认出来。不加配套文件，不加数据库表。

最大的收益是**白捡的**：提取风格现在是「我的录音」里的一条，待处理 → 处理中 → 已完成，进度看得见，能重试。**复用主干比新开支线更可靠**——支线上每段路都得自己修。

还有条小补丁：语料不足 300 字，「提取」按钮直接禁用（54d4c70）——真有用户只分享了一个书名就点提取。前置拦截比事后解释便宜。

那行隐形注释，和错位一行的事故

文章得记住它按哪一版文风写的，界面上才能显示风格 chip。最初做法很讨巧：正文里藏一行看不见的 HTML 注释 `<!-- style: 风格 v7 -->`。讨巧在于**通用**——不动数据结构，任何东西都能这么标一下。

然后就出事了。iOS 渲染前会先剥掉注释再给每行编号；服务端的 `linenum.js`（给正文标行号的模块）**不剥**。用户看到的「第 3 行」和模型看到的差了整整一行。用户说「改第 3 行」，改的是第 4 行。

大白话

一行人看不见的注释，把人眼里的世界和模型眼里的世界错开了一格。而这两个世界必须逐行对齐，「说话改稿」才成立——那是第 5 章的主题。

修法是把注释**搬进字段**（10c3583，2026-07-03）：`articles[i].style`，整数，per-article，随版本走——撤销重做之后 chip 还是对的。存量迁移的账很清楚：382 个文档扫描、50 个迁移（连历史版本一起改）、二次空跑 0、线上残留 0，迁移前做了全量备份。

留下来的那条铁律

所有重建 `doc.articles` 的写路径，一律「继承 + 覆盖」`{...a, title, body}`，**绝不允许**回退成 `{title, body}` 的白名单重建。

大白话：改一篇文章要**先把旧对象整个摊开、再盖上新的标题和正文**，而不是新造一个只有标题正文的干净对象。

差别在哪？前者，未知字段（style、微信素材 id wechatMediaId、将来任何还没发明的字段）**自动存活**；后者每编辑一次就静默丢一批，等发现时已经丢了几百篇。

wechatMediaId 就是靠每处写路径手动搬运才活着的，漏一处就完蛋。

可扩展的契约不是「支持所有字段」，是「不认识的东西照样原样传下去」。

配套还加了防御：linenum.js 现在编号前剥掉一切注释（镜像 iOS），残留的老注释在下次编辑回写时自动消失——自清洁。

chip、版本名，和「免费切换」

- **版本名 = 文风首行，≤12 字**（ee23577）。不让用户额外起名——蒸馏出的风格卡第一行本来就是风格名。少一个输入框，少一次犹豫。
- **风格 chip + 换风格重写**（d29dd31）。文章信息行挂个 chip 显示当前风格，点它可换。这里藏着省钱的判断：**这篇已经有那个风格的版本就直接切过去，不花算力也不用等；确实没有，才真去重写。**

它能做到，正因为文章本身是版本化的——同一篇的多个风格版本本来就并排躺在

versions[] 里。**好的数据结构会在你没预料到的地方省钱。**

还有个补丁（2026-07-03）：合并出来的、看图挖出来的文章没有录音，transcript（转写稿）是空的，重写曾硬失败 422。修法是回退用**正文当挖矿来源**。注意一个克制的细节：新文档的 transcript 仍留空，**不伪造**——别为了让下游代码好过就往数据里填假话。

回头看：和市面的差距在哪

整个赛道唯一沾边的是 AudioPen 的 custom style——可上传自己过去的写作样本来定义一个风格（第三方资料如此描述；官网的 styles 页需登录，未能直接核实）。其余从 Superwhisper 的 mode 到 Granola 的模板，**全是预设选择器**。

所以差距不在「有没有学文风」这个功能格子上——那一格 AudioPen 也能打勾。差距在**它是不是一份资产**：有没有版本能退回上一版；有没有历史知道这篇当初按哪一版写的；有没有数据集能持续加语料重蒸馏；能不能继承到每一篇文章上。

一次性喂样本换一个风格槽位是功能；一份随你一起长大的个人资产，是**用户走了带不走的東西**。而让它成为资产的全是些不性感的工程决定：复用已有的版本信封、让 head 只是个

指针、让老文件自然退休、把类型标签烧进文件名，以及那条「继承 + 覆盖，绝不白名单重建」的铁律。

资产之所以是资产，不在于它多聪明，在于它不会丢。

第五章 · 第 3 行到底是哪一行

一篇文章已经写好，摆在手机屏幕上，有段落，有配图。你按住屏幕下方那根条说一句：「把第 3 行改口语一点。」

要让这句话生效，系统必须保证一件事：**你嘴里那个「第 3 行」，和模型脑子里那个「第 3 行」，是同一行。**

这句话听起来像废话。做起来是这一整章。

先说这件事有多不寻常

2026 年 9 月那次竞品调研，25 个产品挨个查「能不能改稿」：

- 所有会议纪要工具（Otter、飞书妙记、通义听悟、腾讯会议、Granola）都提供「对纪要提问」，**没有一个提供「改纪要」**。你能问「刚才谁负责这件事」，不能说「第二段太长了，砍一半」。
- AudioPen 能 rework，但那是个按钮，而且整篇重来。
- Aqua Voice 是唯一带「语音改写」概念的产品，但它改的是**你刚说出来、还漂在输入流里的那句话**（「删掉上一句」）。它是一支更聪明的笔，管的是正在流出来的墨水。

对一篇**已完成、有段落有配图、已经落盘的文章**下语音指令做定点编辑——一家都没查到。这是整张表上最干净的一块空白。

大白话

翻译成成人话：别人做的是「说话时能改口」，这里做的是「文章写完了，对着它说话就能改」。差一个字，差一整套工程。

它抄不走，因为它不是一句提示词：你得把文章结构化到「行」和「图」的粒度，把一句自然语言解析成一处定点修改，还要有版本回滚接住改坏的情况。这是纵深，不是点子。

闭环长什么样

按住说话，声音进 ASR（自动语音识别）变成一句中文，通过 WebSocket（建立后一直开着的双向通道，不像普通请求问一句答一句就挂断）打到 `/agent/edit`。接住它的是一个 Durable Object（Cloudflare 上的一种小服务：有身份、有持久存储，同一篇文章的请求都路由到同一个实例，所以它能记住事情）。它读出文章、读出你的个人文风、调模型重写、写回存储，再把整篇推回 App 就地刷新。这一次改稿花多少算力是第 8 章的事，这里只当它是一道闸门。

写回时字段是「继承加覆盖」而非「按白名单重建」，所以像 `wechatMediaId` 这种模型压根不知道的字段能活下来——它记着这篇对应公众号后台哪条草稿。于是你语音改完再发公众号，是**原地更新那条草稿**，不是又发一篇新的。

说话可以连着说，改稿必须排队

人说话是连续的：说完「第 3 行改口语一点」，两秒后又想起「第 5 行那个词也换掉」。麦克风若被「正在重写」锁住，体验就断了。但模型编辑必须有因果顺序——第二条若基于旧正文算出来，会改错地方，甚至覆盖掉第一条的成果。

于是把两者拆开：**麦克风永远不锁，指令严格排队**。连说三句都收下、都堆成气泡，但真正跑起来的时候严格串行。最初这个串行是客户端做的——上一条的 `updated` 回来了才发下一条；后来队列的真源搬到了服务端（下一节），客户端变成有话就发，由服务端按序号一条一条地跑。不管由谁来排，不变量始终是同一个：每次修改都基于上一次的**结果**，不是上一次的**请求**。嘴上可以抢话，手上必须排队。

队最初排在手机里，而手机里的队有个致命问题：App 被系统杀掉，队就没了——你说了三句，走进电梯，出来只改了一句，还不知道另两句蒸发了。后来真源搬到服务端：DO 里一张 SQLite 持久队列表加一个自驱的排空循环，稳定 id 做幂等（同一件事做一次和做十次结果一样，重连重发认得出是同一条），重连时对快照两边对账，而 **App 只落盘 pending、绝不清队列**——清空的权力在服务端。断网、切后台、进程被杀都不丢。**客户端瘦，真源在服务端。**

锚点协议：本章的核心

一、给用户一个可以指的东西

屏幕上没有「第 3 行」这个东西，用户就没法说「第 3 行」。所以有了**定位浮标**：一按住说话，正文左边空白里淡入一列小小的「第 N 行」，每张图左上角淡入一个「图 N」，松手淡出。

这里有个不起眼但很硬的约束：**文字一个像素都不许动**。浮标绝对定位在左边距

(`.overlay(alignment:.topLeading)` 加 `offset`)，不占布局空间，正文不重排。要是行号一冒出来正文就挪一下，这功能从第一秒起就是错的——你想指的那一行会在你按下去的瞬间跑掉。

二、编号必须连续，图片也占一行

最初的「第 N 行」只数文字段落，图片被跳过——逻辑上说得通，图片又不是文字。结果是：文章前面有两张图，图后面那段屏幕上显示「第 5 行」，正文里的真实位置是第 7 行。用户说「改第 5 行」，模型老老实实去改第 5 行——改到了别的地方。

改法是把编号变成**一个连续的计数器**：段落算一行，图片标记也算一行。于是一张图同时显示两个东西——左边距上的「第 N 行」（在正文里排第几）和角上的「图 M」（第几张图）。顺带一提，正文里那行看不见的注释也曾把编号顶歪一格，那是第 4 章的故事：一个隐形注释怎么让行号差 1。

三、模型读行号，永远不数行号

这是全章最重要的一句话。

前两步做完，服务端发给模型的还是整篇正文，指望它自己数到第 3 行。这条路能跑通，但**脆**：文章一长就容易数错，带图的更容易——模型得自己判断哪个是段落、空行算不算、那个 `[[photo:xxx]]` 标记该不该占一行。每个判断都可能和 iOS 那边不一样，错一格就足够改错地方。

所以 2026 年 6 月 28 日改成：**不让模型数，把数好的结果递给它**。服务端多了个文件 `agent/src/linenum.js`，用和 **iOS 完全相同的算法**给正文编号，生成一张「行号对照

表」注进发给模型的消息里，形如 第3行：那天下午.....、第4行 = 图2：[[photo:...]]。提示词里明写：按这张表定位，**不要自己数**，输出时也不要带行号。

大白话

翻译成成人话：别让模型做算术，算术让代码做。它擅长「理解这句话想干什么」，不擅长「这是第几行」。把它不擅长的那部分从它手里拿走。

由此产生的一条契约

代价必须写在墙上：linenum.js 和 iOS 里的 bodyRows() / segments 是**同一套算法的两份实现，必须成对修改**。改一边不改另一边，模型的编号就和用户看到的漂移，而且漂移是无声的——不报错，只改错行。所以 linenum.js 有自己的单元测试。

连带一条：**iOS 新版和 Worker 新版必须一起上线**。老 App 的行号只数文字，新 Worker 的对照表含图片，用户看到「第 5 行」，模型收到的表里第 5 行是另一段。这不是崩溃，是安静地改错地方，比崩溃难查得多。

多篇文章同理：一条录音常挖出好几篇，App 每切一篇都从「第 1 行」重新编号，所以每条指令带 articleIndex，一路传进持久队列的 article_index 列，Worker 拿它决定给**哪一篇**编号。老版本不带，默认第 0 篇。

v2：不光传行号，还传那行原文

当你用长按菜单选中目标时，App 知道你精确指的是什么，那就别让这个信息退化成一句自然语言。它把目标**结构化**地传上去：图片是 {type:"image", key}，段落是 {type:"line", line, text} ——注意 text，整行原文。

为什么带原文？因为**行号会漂移，文本内容不会**。你长按第 7 行时，队列里可能还压着一条没执行完的指令，它一执行第 7 行就成了第 8 行。服务端拿整行原文回正文里做唯一匹配，能把行号自己修正回来；匹配不上就丢弃这个锚点——宁可退回「模型自己理解」，也不要拿一个错锚点去精确地改错地方。副作用是提示词反而变简单：不必再用占位符往里塞坐标，回归纯自然语言，坐标走另一条管子。

长按菜单：新入口，不新开管道

长按一张已出图的配图，弹出「图片风格」：卡通、广告、水彩、素描、油画、胶片。长按一个段落，弹出「改写这段」：更简洁、更口语、更书面、扩写一点，还有「插入图片 → 公众号题图」。

关键在于这些点选**没有新开一条路**：点一下等于把一条配置好的指令塞进**同一个语音编辑通道**——同一个 `enqueue`、同一个持久队列、同一个「正在改」指示器、同一套占位图。一个入口修好的 bug，另一个入口自动也好了。视觉上倒是放弃了系统自带的

`contextMenu`：它改不了颜色和圆角，做不出设计稿要的暖纸调子，于是改成自绘覆盖层。菜单发出的指令还带 `itemId`，服务端出图时把这条提示词的「魔法数字」写进图片的 XMP 元数据（文件里一块给机器读、肉眼看不见、跟着文件走的说明区）——每张 AI 配图都能追溯是哪条提示词生成的。

不该经过模型的，就别经过模型

长按菜单里还有「编辑」，调出键盘就地改字。这条路**完全不经过大模型**，复用通用的文章写入接口 `PUT /files/api/articles/<stem>`，白拿版本记录和撤销重做。但省事只是顺带的，真正的理由是：把「改掉这个错别字」包成自然语言喂给 AI，AI **顺手把别处也润色一遍**的概率不是零——你只想动三个字，回来一看整段语气变了。用户手敲进去的字就该原样落盘，中间别站着一个有创作欲的东西。写回时另有一条纪律：只 `merge articles` 这一个 key，免得冲掉模型没建模的字段。

两个事故

「说话和没说一样」

语音听写从 Apple 端上识别换成火山流式 ASR，中文更准。凭证不能放 App 里，所以服务端做**哑中继**：客户端持全部协议、服务端持密钥，收到帧后剥掉客户端自己的 `Authorization` 头、换上火山凭证转出去。App 永远接触不到第三方凭证。

上线之后，完全不工作。说话，没反应。

根因在一行转发代码里。Cloudflare Workers 把二进制帧交给你时是个 Blob 对象，而 `send(Blob)` 会把它隐式 `toString` 成字符串 `"[object Blob]"`——**13 个字节**。每一

帧音频，原来多少 KB 都一样，到达服务商的都是这 13 个字节的文字。人家当然不转写：你给它的根本不是声音。修法很短：两端 socket 都设 `binaryType="arraybuffer"`，转回 `ArrayBuffer` 保序转发。

但真正的教训在于：**这个坑两次上线都没被发现**。因为冒烟测试是这么写的——连上去，发一个配置帧，看有没有字节回来。有字节回来（就是那 13 个字节），测试通过。它一直在验证一个被强转出来的字符串，还把它当成「有效回包」。

大白话

翻译成人话：只验「有没有回音」的测试，不能证明「说的话被听懂了」。合格的冒烟测试必须流真实音频进去，断言解码出来的文字是对的。测「通了没有」是最容易写的测试，也是最容易骗自己的测试。

「UI 停在旧正文」

第二个事故在弱网下：你说完一句，气泡消了，正文没变。原因是服务端改完广播 `updated`，而弱网下 `WebSocket` 每十几二十秒断一次，广播落在一条已经死掉的连接上。App 于是永远显示旧内容——它在等一条永远不会到的消息。

两处修法。一是 25 秒心跳，保活空闲连接，顺便让死连接提前暴露立刻重连。二是更根本的：App 不再依赖任何一条推送来收敛，每条指令走到终态（`updated` 回来、报错、或重连对账发现它已完事）都**主动发一次普通 HTTP 请求，把权威文档重拉一遍**覆盖本地状态。

推送是优化，拉取是保证。凡是「只要那条消息送到一切就对了」的设计，在真实网络里都会有一部分用户永远看到旧东西，而且他们不报 bug，只觉得这功能坏了。（同期做过四版「实时预览」最后整体撤掉，那是第 12 章讲减法的内容。）

最后：让改动看得见

所有语音修改落地后，App 把新旧正文做一次 `diff`（逐行比对，找出哪些行变了），变动的那几行涂上淡黄底色，3.5 秒后淡出。你说了一句「把第 3 行改口语一点」，不该再去通读全文找哪里不一样——**那一行自己会亮一下告诉你：是我，改的是我。**

用户标的那一行，模型改的那一行，屏幕上亮起来的那一行。从头到尾，同一行。

第六章 · 一张照片的旅程

你在工地上一边说话一边拍了四张照片，然后揣起手机忘了这件事。第二天早上打开 App，文章挖好了，四张照片各自躺在它被说到的那一段旁边。

这条路听上去很简单——拍下来、传上去、插进去。但每一步都出过错，每次修法都指向同一个方向：**让正确性来自结构，而不是来自时序运气。**

起点：拍照不能打断你说话

前提只有一条：拍照那几秒，录音不能断。这在 iOS 上不自动成立——相机默认接管音频，录音当场被掐。做法是相机的 `AVCaptureSession` 只开视频通道、不碰麦克风（09298d1），录音器不知道你刚才拍了张照。

入口做得极淡：一个很轻的相机图标加「拍照」小字，用 `.overlay` 挂在停止键上而不是并排放——它不参与布局，停止键仍整屏居中。相机打开后不关闭：连拍，每张落进胶片条、可逐张点 × 删掉，按「完成」统一上传（be6bc79）。**删除于是是纯本地移除**，云上不留孤儿。

大白话

翻译成成人话：能在本地解决的犹豫，别送上天。

相机方向：一个不能同步读的值

一桩悬案：前置摄像头拍出来的照片是横的。第一版把旋转角写死 90° ——错，你可能横着拿手机。第二版在翻转摄像头之后补一次修正——也错，翻转会重建拍照连接并重置几何，「事后修正」和重建在赛跑。

正确做法是问系统：用 `RotationCoordinator`，它知道手机此刻的物理朝向。真正的坑在于它**异步解析**方向，同步读一次拿到的是上一次的旧值，必须用 KVO（key-value observing：不是我去问值，而是它变了通知我）监听（c37fab2）。另一条铁律：**预览的左右镜像交给系统**，手动镜像会把旋转的方向感弄反。

异步解析的值只能订阅，不能顺手读一次——你读到的不是「还没算好」，而是「上一次的答案」，它伪装得像真的。

名字：两次死在时间戳上

第一次，照片和录音对不上。照片目录名单独取了一次当前时间，音频文件名用的却是录音器记的开始时刻。两次取时间隔了几毫秒——一旦跨过整秒边界，两个名字就差 1 秒，那批照片没人认领。修法：目录名一律用录音器的起始时刻，**同源**（c2c5752）。

第二次，同一秒的照片互相覆盖。相册批量导入最多 9 张，回调几乎同时触发、各取一次秒级时间戳——几乎必撞。同名文件在云存储上就是**静默覆盖**：不报错，照片少几张（7666075）。

新名字是 `<offset>-<rand>`。`offset` 是这张照片距录音开始的第几秒——绝对时刻没丢，它等于「录音开始 + offset」，原来那个绝对时间戳本就冗余；`rand` 是 3 位 base36（用 0-9 加 a-z 共 36 个字符记数）随机尾，46656 种组合，9 张同秒相撞概率约 0.08%。**唯一性不该靠时钟，靠结构加一点熵**——时钟只会告诉你「现在」，而「现在」是很多人共享的。

正文里的记号：从序号到地址

模型写文章时会在照片该出现的位置插一个记号。老设计是 `[[photo:3]]`——「照片数组里的第 3 张」。它很脆：每个记号都耦合在数组顺序上，语音编辑插一张、删一张，后面全部错位，用户看到的是「记号变了但图没变」（12ebf64）。

新设计是 `[[photo:photos/<录音时刻>/<offset-rand>.jpg]]`——**记号本身就是这张图的地址**。插入、删除、重排都不影响别的记号，它们之间根本没有关系。

大白话

翻译成成人话：别写「第三排那个人」，写他的名字。

更妙的是后来那件事：服务端和网页把这个记号当成不透明字符串（opaque string，只搬运、不解读、不假设格式的一串字），所以照片名字改成 offset-rand 时它们**一行代码都不**

用改。不透明契约是最省事的契约：你没承诺过格式，就没人能依赖它。同时 `photos` 数组不再控制显示（3a1b868），后来服务端干脆不再写它，消费方直接从正文里抠出被引用的地址，老数组只读，仅用于翻译老的 `[[photo:3]]`。

一个端点服务所有人

照片要显示在四处：App 内社区、作者详情页、公开分享页、导出的 HTML。一个短命的「按帖发临时凭证」设计被极简的一条取代：`GET /files/api/photo/<完整 key>`，公开、无鉴权，直接从照片原本的位置吐出来，连作者看自己的也走这条。安全只靠两条：文件类型白名单、禁止路径出现 `..` ——它永远吐不出文章或凭据；而完整地址不可猜（哈希过的用户前缀加时间加随机尾）。**一个入口比五个入口安全，因为你只需要审一个。**

三层保证：一张都不能丢

用户报了个含糊的现象：录音时拍的照片，有时候没进正文。查下来是三个独立的丢失点，于是有三层。

第一层，写盘前重新列一次。此前全靠模型自觉插记号，漏插就静默少一张。现在写文章前重新（带游标翻页地）列一遍照片目录——是重新列，不用跑批开始时的快照，因为照片可能在语音识别期间才传完。正文没引用到的地址按顺序补一个记号到末尾。**模型的自觉不能当作不变量。**

第二层，晚到照片的服务端补写，堵的是第一层堵不住的死窗口：文章已经写完了，照片才到。做法是照片落盘后带时间戳去戳这个用户的挖矿实例；实例记个待办（60 秒宽限期合并猛拍，失败重试 3 次），在下次闹钟里**先于挖矿**执行补写，分三种情况：

1. 已成文 → 把所有历史版本引用过的地址取并集当作 `everSeen`，缺的补到末尾，铸一个 `photo-backfill` 版本。**出现过又被删掉的，永不复活**——那是用户的意志，不是丢失。
2. 被判成「没语音」的空录音 → 删掉那个标记，同一次闹钟里改走「看图挖矿」（语音识别有检查点，不会二次扣费）。
3. 两者都不在 → **什么都不做。**

第三种情况值得停一下。什么都不做凭什么对？因为文章只可能在这个实例的串行闹钟里诞生；既然串行，未来那次挖矿的「重新列一遍」必然晚于这次检查，也就必然列到这张照

片。不是「大概率被兜住」，是「不可能不被兜住」。闹钟取「照片的 +60 秒」与「挖矿的 +500 毫秒」中较早者——补照片绝不许拖慢挖矿。

第三层，客户端落盘队列。此前拍完是一次性上传，一失败就永久丢失。现在先写进本地 `pending-photos` 目录，**传成功才删**；启动、联网恢复、入队、音频上传前、前台刷新五个时机各 `drain`（把队列里积压的东西倒空）一次（1c47681）。

取舍：照片不再阻塞音频

有用户抱怨上传特别慢，查下来真凶不在线路上：音频上传前，代码强制串行地等标签边车写完、等照片队列逐张传完，每张失败还退避重试 3 次（这桩谜案的完整破法在第七章）。

改法是把顺序拆开（9f3884b）：音频立刻上传；照片并发 3 张、每张每次只试一次、失败留盘等下一次 `drain`。敢这么改，是因为第二层兜住了「照片晚到」。**顺序上的「稳妥」，常常是在拿最重要的东西去等最不重要的东西。**

画质与尺寸：过期的旧答案

画质走了一条摇摆的路：上传前中心裁成 1:1 方图（2b73862）→ 横竖图被剪坏，改成按真实比例显示（80a7419）→ 相册导入也保住长宽比（b07ad15）。可用户说比例还是没了——上传口里还藏着一道二次裁方，在最后一刻剪掉（d008edc）。那行代码上方写着一句注释：「photos are square everywhere」，它曾经是对的。**注释是会过期的事实**：代码写错会报错，注释变错只会静静地骗下一个人。另一件：相册里存的一直是模糊小方图，而拍照回调里的原片本就是 4032×3024，只是存相册时拿错了张（2ca0724）——**很多「做不到」其实是「拿错了」。**

同一张照片有三个尺寸：云上的原图永不改动；**给模型看的缩到 320×320**（c2c5752）——模型只需看懂「这是一堆钢筋」，不必数清根数，一张约 137 个 `token`（模型计量的最小单位），省约 10 倍；第三个是 App 里的缩略图，教训出在这儿：分享扩展只有约 120MB 内存，而 `UIImage(data:)` 会把整张原图解码进内存，分享大照片时当场被系统 `OOM`（out of memory，内存超限被强制杀掉）干掉（f09b68c）；缩略图那条路用的是另一个 API `UIImage(contentsOfFile:)`，同样全分辨率解码（12MP 约 46MB 一张）且每一格常驻，6 张就越过上限（1494c06）。两处的修法是同一个：

CGImageSourceCreateThumbnailAtIndex，在解码那一刻就限制尺寸。上传路径先修过一次，缩略图那条路漏了、后来原样复发：同一个错误会在另一条路上再长一次。

开头那个病还犯过一次：AI 配图永远显示「正在制作中」，Safari 打开同一地址却好好的（bd205d3）。系统网络层无视服务端写的「不要缓存」，把窗口期里那次失败的响应钉死在进程缓存里，图片真落盘后也永不自愈。你以为在问外面的世界，其实在读一份缓存。

回头看这条路

- 照片与录音的关联，不靠「两次取时间差不多」，靠同源；名字的唯一，不靠「同一秒不会来两张」，靠随机尾。
- 正文里的记号，不靠「数组顺序不会变」，靠地址；照片不丢，不靠「上传通常会成功」，靠落盘队列和两层服务端兜底。
- 那个「什么都不做」的分支，不靠概率，靠串行闹钟这个结构本身。

这些旧假设单看都合理，也确实在多数时候成立。共同问题是：都是对时序的祈祷——祈祷在演示时永远灵验，在一万个用户手上必然失效。一张随手拍的照片能走到文章里正确的那一段旁边，靠的不是每一步都恰好赶上，而是每一步都不需要赶上。

第七章 · 永不丢失

唯一不能重来的那 3 分钟

这个 App 里几乎所有东西都可以重做：文章重挖一遍，照片重拍一张，文风把版本切回去。只有一样东西**不可再生**——你刚才说过的那 3 分钟话，说完就没了。这一章讲的所有事，都在回答同一个问题：**围绕这个唯一不可再生的东西，得一层层建起什么？**

磁盘就是队列

先说最反直觉的决定：**没有上传队列**——没有队列数据库，没有 `status` 字段，没有三态机。规则只有一句：Documents 目录里还躺着一个 `VoiceDrop-*.m4a`，就等于「这条还没传成功」。传成功了文件就被删掉，文件还在就显示「正在上传」。

大白话

翻译成成人话：文件在不在，就是状态本身——不是拿一条记录去描述文件，文件就是那条记录。

它的价值在最糟的时刻才看得出来。进程被杀、崩溃、删了重装、手机关机——**状态不可能和现实不一致**，因为它没有第二份副本可以对不上。扫一遍目录就知道还欠谁。这个动作叫 drain（把积压的东西一个个排空，像给浴缸拔塞子），触发点撒得到处都是：启动、切回前台、联网恢复、下次录音。

代价是它的第一个 bug——扫描器分不清「录完在等上传的」和「正在录、还在往里写的」，于是把只写了一半的 m4a 传上去，还顺手从录音机脚下把它删了。修法是给正在录的文件起一个队列永远匹配不上的暂存名（完整故事在第 1 章）。**用文件当状态，就必须让「哪些文件算数」这条边界绝对清晰。**

第一桩谜案：214 秒

2026 年 8 月 1 日，苏州，蜂窝弱网。实测录音上传的中位耗时是 **214 秒**。有一个细节让所有「网太慢」的解释当场破产：**214 秒与文件大小无关**——不管录的是一分钟还是十分钟，都是这个数；而同一条线路实测能跑 1.7MB/s。一个不随大小变化的延迟，说明堵的不是管子，是管子前面的排队。

真凶在 `upload()` 的开头。它上来串行 `await`（等一件事做完才做下一件）了两样东西：标签边车文件的 `PUT`（把一份文件整个放到云上某个地址），和照片队列的完整 `drain`——每张照片失败还要 3 次退避重试（等一会儿再试，每次等得更久）。

也就是说：**那段唯一不可再生的音频，在排队等一堆远不如它重要的东西**——照片可以重拍，标签只有几百字节，它俩却把音频堵在门后 3 分半。

修法是重新排队：音频立刻 `PUT`，不等任何人；照片 `drain` 改成并行三张、发出去不管结果，每张每轮只试一次，失败就留在盘上等下一个触发点；标签边车另起并行任务。

第二桩谜案：延迟恰好等于你下次打开 App 的时间

照片解开之后延迟并没消失。8 月 1 日到 3 日，每条录音的「录完到落地」延迟在 22 分钟到 2 小时之间——仍然与文件大小无关，与网速无关，却精确地等于另一个数：**用户下次打开 App 的时间间隔**。而同一次录音里的 4 张照片，录完当秒就传完了——线路是无辜的。

根因是前台的 `URLSession`（App 自己发起的网络请求）只有大约 30 秒的后台缓冲。录完立刻锁屏、把手机揣兜里，第一枪还没打完进程就被冻住——而冻住之后，**再没有任何人重试过**。文件只能干等到用户下次打开 App。

修法不是把重试写得更结实，而是**把重试整个交出去**：改用 `background URLSession`——一个由系统守护进程接管的传输会话，你把任务交出去，就算 App 被杀掉，系统也会替你传完，完成事件在进程复活后回放给你。录完立刻锁屏、切走、被杀，传输照样完成。

于是进程内那套「3 次尝试 + 退避 sleep」被**删掉了**——系统会在 6 小时的资源窗口里自己等网、自己重试。只有真正的失败（服务器 4xx/5xx，或窗口耗尽）才让文件留在盘上，等下一个 `drain` 触发点重新入队。**永不丢失，是因为最坏情况也只是回到「文件还在盘上」这个原点**。由此带来两个约束：

- 收尾（删本地文件、清边车、更新列表那一行、写埋点）必须集中在**一个函数**里，并且幂等——做一遍和做十遍结果一样。因为「进程一直活着」和「死过一次、隔世重启后回放」是两条完全不同的路，必须收敛到同一个结果。收尾入口只能有一个，`Uploader` 因此被改成单例。
- 任务元数据只能存在系统给的 `taskDescription` 字段里，而且必须是 **Documents 的相对路径**——容器的绝对路径每次安装都会变，存绝对路径等于给隔世重启埋一颗雷。

一条录音，两篇文章

8 月 28 日，一段 2 分 49 秒的录音在服务端变成了两个文档、两篇文章：相差两秒，一个带地名，一个不带。根因是一句**写在代码里的旧注释**——「改名会让在飞的上传失败」。

流程是这样的：录完先落一个不带地名的正式文件名（让它立刻进入 drain 的扫描范围，免得等待期间被杀成孤儿），再等地理编码（把 GPS 坐标反查成「中央区新川」这样的地名）回来，改成带地名的富文件名。改名会让在飞的传输失败——旧名那一枪打空没关系，富名那份会重传。

这句话曾经**完全正确**。但换成 `background URLSession` 之后，系统在**建任务的那一刻就把文件拷走了**。改名毫无影响，旧名字照样传成功；而改好名的那份，又被下一次 drain 当成一条全新的录音传了一遍。一段话，两篇文章。

修法很轻：一个内存里的 hold 集合，在等地名的窗口期让 drain 跳过它，地名回来就放行。用内存态是有意的——窗口期里进程被杀，集合蒸发，下次启动按盘上现名老实上传，**最坏只丢一个地名**；录音不丢，不双传。

注释是会过期的事实。一句在当时完全正确的断言，会在底层实现换掉之后安静地变成一个错误的前提，而且看起来仍然可信。

索引也是会过期的事实

同一种病换个壳又犯过一次。首页列表有个循环写成 `for i in indices`，循环体里有 `await`；挂起的那一瞬间用户左滑删掉一条，数组变短，恢复执行时旧下标越界，崩。修法

是快照一份稳定标识（文件的 stem），每次 await 回来重新找下标。**在异步循环里，索引也是会过期的事实**：时刻 A 记下的东西到时刻 B 已经不成立，代码却默认它还成立。

另一个更短：心跳的 `sendPing` 回调可能被调两次，`CheckedContinuation` 第二次 resume 直接崩。修法是一个一次性门闩。**凡是「保证只调一次」的回调，都要按「可能被调两次」来写。**

神圣主路径

后来做 AI 语音采访员——录音时有个 AI 在旁边追问你——架构上的第一个决定就是本章命题的正面表达：所有录音统一走同一个音频引擎，在音频采集回调里先**把 m4a 写下去**；这是主路径，任何开关都碰不到它。写完之后，才把同一份 PCM（未经压缩的原始音频数据）分给 AI。

于是「AI 采访」被降级成录音过程中**一条纯旁路**：随时可开可关，失败就降级成一次普通录音。另外留了一扇逃生门——「经典录音引擎」开关，新引擎万一出问题，整套 AI 都绕得过去。

架构的本质是排序：把不可丢的放最里面，把可失败的放外面。

克制也是一种保护

还有一个反方向的决定：**不足 4 秒的录音直接不传**。4 秒说不出一篇文章，传上去白花一次算力，最后还回给用户一句「没检测到语音」——而在那之前，界面上已经给了他一个**虚假的期待**。拦截点放在最早那一步：把录音提升为正式文件的那个函数里，不足 4 秒直接删掉暂存文件、返回空，四条调用路径一条也绕不过。**知道什么不值得进入这套保障体系，也是保障体系的一部分。**

没有数据，就只能靠猜

214 秒这个数字最初是**反推**出来的——拿云存储上 947 条录音的落地时间戳减去文件名里的时间——App 根本没记过上传耗时。排查完第一件事就是补埋点：耗时、排队时长、文件 KB、网络类型。更值得记的是另一个细节：**换口径时给埋点打了版本号**——解除照片串行那

次叫「口径 2」，换后台会话那次叫「口径 3」。因为「耗时」的含义变了：前台会话下它指「PUT 本身花了多久」，后台会话下指「入队到系统送达」，**包含进程被冻住的那段**，也就是用户体感的「录完到传完」。两个不同的量，只是碰巧同名。

换了度量口径，就要给它一个新名字。否则你会把两套数据混在一起算平均值，然后对着一个不存在的数字做决策。

一句话

别指望时序，别指望进程活着，别指望自己会重试。把状态放在磁盘上，把重试交给系统，把最重要的东西放在最里层。

第八章 · 把成本明码标价

写一本书的那个页面上，最显眼的不是「开始」按钮，是一个价签：琥珀底、一个闪电符号、一个数字。右边是实时余额，够了绿、不够红。按钮上印着价格，钱不够时它变灰，并告诉你还差多少。

在 AI 产品里这是反常的。反常到得先说说别人怎么收钱。

这个赛道怎么收钱

2026 年 9 月那次竞品调研把二十几家的价目表抄了一遍，结论很整齐：

- **主流是包月订阅**：个人档 8-17 美元/月，企业档 14-35 美元/座位/月，年付普遍打四五折（Otter Pro 月付 16.99，年付摊到 8.33）。
- **免费额度用「时间」做闸门**：Otter 300 分钟/月，Voicenotes 100 分钟/周，飞书妙记 300 分钟/人/月。
- **买断制只存在于本地工具**：MacWhisper 59 欧元终身、Superwhisper 249.99 美元终身——模型跑在你自己电脑上，卖完不再花钱。**凡是自己烧 API 的云端产品，一个敢卖买断的都没有。**
- **按量透明计费只活在云 API 那一侧**：通义听悟 ASR 0.6 元/小时、讯飞听见机器快转 0.33 元/分钟，是给开发者看的价目表，不是给写作者看的。

摞在一起是一块干净的空白：**在语音笔记赛道内部，面向 C 端的按量透明计费为零。**所有人都卖「Pro 版无限使用」。

可「无限」是句假话。云端每一次调用都在真实烧钱，它的真实意思是「我们赌你用不了那么多」。成本没消失，只是塞进包月价里藏起来了。

大白话

翻译成成人话：自助餐门口写着「随便吃」，但后厨每盘菜的成本老板门儿清。区别只在于他告不告诉你。

单位本身倒不新鲜：Notion 的 Custom Agents 是 10 美元换 1000 credits，Sudowrite 卖 credits，腾讯 ima 的 copilot 干脆就叫「算力」。新的不是单位，是把它用在一个人人都卖「无限」的赛道里。

算力是成本穿了件马甲

先钉死一个数：**23 算力 = 1 元**，从上线那天没动过。

但底层账本里根本没有「算力」。账本记的是微元——百万分之一元——而且只记整数。算力只是把微元乘以 23 之后的显示投影，是给人看的那层皮。汇率写死 $FX=7.3$ ，一律 `Math.ceil` 向上取整，保证「只会多收一丝，绝不亏」；没登记过价格的模型成本记 0，宁可漏收不可乱收。

手感是这样的：一段典型录音挖成一篇文章约 **2 算力**，便宜模型改一刀约 **1.4**、贵模型约 **4**；新用户一次性送 **200 算力**（2026-07-19 从 500 下调，只影响新账号）。200 算力约合 8 块 7，够挖上百篇——**单篇成本小到几乎可以忽略**。记住这一点，后面那个反直觉的选择才讲得通。

为什么是整数微元，不是小数

计算机里的小数是浮点数，为了表示极大极小而牺牲精度的近似格式，最有名的毛病是 0.1 加 0.2 等于 0.30000000000000004。单看是尘埃，但账本是**累加**的，几千次调用几千颗尘埃各偏各的；更糟的是对不上账——你按流水重算和存的余额差一点点，不知道是算错了还是被人动过。所有跟钱打交道的系统答案都一样：**换个小到不用小数的单位，然后只用整数**。银行记「分」不记「元」，这里记「微元」。

大白话

翻译成人话：别用「三分之一的披萨」记账，改用「切成一百万份，你欠我 333333 份」。数字丑，但永远算不错。

一句定义挡掉四件事

算力的另一半定义是三个「不」：**无现金价值、不可提现、不可退款**。

看着像用户协议里的套话，实际是这个项目里投入产出比最高的一行字：它一次性把充值、退款、提现、记账全挡在项目之外，而那四件事各自都要带上支付通道、法务和财务合规，都比整个计费系统本身更重。关键是换了个会计身份——**账上挂的不是用户的钱，是一笔营销负债**，跟超市积分、航空里程同类，不是预付款。将来真要「充值收钱」，那是另一个带支付带法务的项目。没写一行代码，砍掉半个系统。

计费必须长在花钱的地方

所有 ASR 和大模型调用都在同一个 agent worker 里，计费就落在那儿——**计费和成本发生地之间每多一跳，就多一个对不上账的理由**。数据库另起一个**独立的**，不寄生在推荐服务的库上：推荐数据随时可以丢掉重算，账本不行。

两张表：**账户**（余额、累计获赠、累计消费，全是微元整数）+ **流水**（ledger，流水账，**只追加从不修改**，每行记赠送还是消费、原因、明细，外加当时的余额快照）。账户懒创建，两个请求同时发现「没账户」都去建的竞态用 `INSERT OR IGNORE`（已存在就当没发生）挡住；扣费和赠送走原子批处理。

一个统计口径值得单说：「编辑次数」数的是**真实编辑次数**（按每轮对话的 id 去重），不是模型调用次数——一次编辑在实现上是个循环，模型可能来回调好几次工具。用户说的是「我改了三次」，不是「我触发了十一次调用」。**度量要对齐用户的心智，不是对齐实现**。这在限额上尤其要命：拿调用数去卡「上限 100 次」，用户改到第三十次就被挡住，还不知道自己错在哪。

fail-open：账本挂了，放行

fail-open 意思是「出故障时打开闸门放行」，反义是 fail-close（出故障时锁死）。门锁一律 fail-close，计费按直觉也该如此：扣不了费就别让用。这里反过来——计费数据库挂了就**放行**，同时记一条漏计。所有扣费都被 `try/catch` 包住、尽力而为：**扣费失败，绝不中断挖文章**。理由写进了设计文档：成本本来就小，产品体验大于这几分钱。上面那个「一篇 2 算力」的手感就是依据——**能算出来的账，才敢这么赌**。

闸门放在哪儿

挖矿前有个判断，三个出口：录音超过 3 小时 → 挡住，**终态**，不再重试；余额 ≤ 0 → 挡住，但**非终态**，下次余额够了删掉标记重新挖；否则放行。

这个区分是唯一要想一下的地方：三小时的录音明天还是三小时，重试一万次没用；余额不足是暂时的，用户充上钱那段录音还得给他挖出来。挡住时写一个 `.blocked` 标记，App 显示「余额不足」或「录音过长」徽标——**被挡住的人必须知道自己为什么被挡住**。语音编辑的闸门则放在**调模型之前**：不足就直接拒绝，绝不空花一次钱再说没钱，错误通过队列广播回 App。另有两条独立于余额的防滥用硬闸：单条录音 ≤ 3 小时、单篇编辑 ≤ 100 次，只拦极端。

本章最好的故事：认证即计费

写书 **320 算力**（约 14 元），修书 **40 算力**。价格高到必须被看见，于是有了开头那个价签。（2026-09-01 写书降价到 160。）

这个功能最初的门槛是「调用者得是个合法用户」，听着天经地义。用户当场戳穿：**匿名 token 是客户端自己生成的随机数，服务端对任何格式正确的 token 都会返回一个身份**。那个查身份的接口只回答「这次操作记在谁账上」，不回答「这个人有没有资格」——**whoami 只做归因，不是门槛**。任何人写个循环生成随机 token 就能无限白嫖。

先打了三个补丁（「该账号必须已有成文文章」之类），都是在给一道假门加锁。最终答案是最朴素的那个：**收钱**。新账号送 200，写书要 320，伪造出来的账号天生不够，被一个 402（HTTP 里的「需要付费」）天然挡住。原来的「每天 2 本／全局单本／须有成文文章」三道限制全部拆掉——**算力成了唯一的闸门，防刷门槛是附赠的**。

这是整章的缩影：**把成本摆上桌，很多本来要单独解决的问题会自己消失**。身份验证、防滥用、限流，本质都是「怎么防止有人白拿」的变体；东西一旦明码标价，白拿在定义上就不成立了。

订阅是怎么长出来的

一、一个诚实的占位。算力页上有「包月订阅」按钮，点下去弹「即将上线」——不假装能扣费，不做一个点了转圈然后报错的假按钮。那条提交记录里有一节标题就叫 "Honest

scope (not faked)". **设计稿画了，不等于系统里有。**

二、接上 StoreKit 2（苹果的应用内购买框架），¥19.9/月 → 200 算力：监听交易更新，外加每次启动把未认领的交易逐笔认领——服务端幂等，重放多少次都不会多发。

三、售卖开关放服务端配置，改一个 JSON 就能启停，不用发版。这一步在苹果生态里怎么强调都不过分：发版要等审核，改字段是一分钟。

四、加高档：¥199/月 → 2000 算力，同倍率。冷知识：中国区没有 ¥199.99 这个价格点，只好取 ¥199。

档位设计里的一个非直觉细节

新档的 groupLevel（苹果订阅组里的档位序号，数字越小越高级）设为 1，旧主档降为 2。为什么要动旧档？因为按苹果规则，**只有从低级别换到高级别才算「升级」，才立即生效并按比例补差价**，同级切换得等下个周期。两档同级的话，用户点了升档、钱付了，算力这个月却不到账——而他想升档，恰恰是因为算力不够用。

什么时候该劝人升档

时机被抽成一个纯函数，9 条单测钉死。两个：①**订着主档且当月烧光** → 推高档，因为再买同一档苹果只会回「你已订阅」，升档是唯一还能花钱的路；②**还没订、且缺口大于主档给的 200**（写书要 320）→ 越级直推高档，别让人订完才发现还是写不成。三个否决条件：售卖开关关着、已在顶档、商品从苹果拉不到。**绝不摆一个点了必然失败的按钮。**

一个漏掉的档位

用户报：「算力不够时以前有购买入口，现在没了。」查下来**代码一行没删**，是那个布尔条件——「开关开着**且**没订阅」——漏了一档：**订着、而且当月额度已经烧光**的人。他们既不满足「没订阅」，又确实没算力，付费的路整条消失。线上取证：状态接口返回「订阅中，本月算力 0」，余额 188.2，书价 320，正好卡死在洞里。

最贵的 bug 常常不是写错的代码，是没想到的那一档。布尔条件的可怕正在于此：它看着只有真假两种，背后其实是四个象限，而你只想到了对角线上那两个。

那条提交记录里还写了一句很少有人愿意写的话：**这补的是「不再是死胡同」，不是「现在就能买到算力」**——商品表里只有自动续期订阅，没有一次性加油包，已订阅的人不能重复买同一个订阅。把自己没做完的缺口写进提交记录，保证三个月后的自己不会误以为这里已经完事了。

三条攒算力的来路

写书页算力不够时，摆三条路：①**请朋友给你的文章投币加油**；②**邀请朋友装 VoiceDrop**——作者 12 币、新人 6 币，按币记价，入账那一刻按实时汇率折算，作者每天封顶 30；③**订阅包月算力**。

一条铁律：**现价拿不到就说通用文案，绝不编数字**。「加油一次大约能得多少」是实时从服务端拉的，拉不到就退回一句不带数字的话——不能一边标榜透明一边写个假数。（2026-08-30 还上过一个「一生一次领 320 算力」的活动，当天整体回滚，那是另一章的故事。）

点赞绝不碰钱

更早的一份增长设计稿里写死了一条边界：**点赞只做质量信号，永远不碰钱；铸币走另一条管子**。理由一句话：

点赞一旦能换钱就会被刷、就不再是诚实的质量信号。把钱走另一条管子，点赞回归纯净。

这是 古德哈特定律（一项指标一旦成为目标，就不再是个好指标）的一次工程化落地：你想用点赞衡量内容质量，就千万别让它变成收入来源。

同一份稿子里还有个早期增长模型，叫空头支票：**面值巨大且免费，真实成本只发生在算力被烧掉、真去调 API 的那一刻**——而那一刻正好是「用户产出了内容」的那一刻，那笔钱本来就要花。剩下的靠沉淀率（breakage，发出去的券最终没人用掉的那部分）：绝大多数人用不完，真实成本远小于名义面值；再用日封顶防挤兑。跟商场发代金券是同一道算术，区别是商场的券兑现时要贴钱，这里的「兑现」等于用户在用产品。

最大的受益者是产品自己

「Pro 版无限使用」是一件麻醉品。它让你永远不必回答那个最难的问题：**这一次调用，到底值不值？**反正打包卖了，多调一次少调一次都在包月价里——重试一次、再重试一次、模型链再串一层，反正用户看不见。成本被摊平之后，浪费就没有了名字。

而当每次挖矿标着 2 算力、每本书标着 320 算力，这个问题每天追着你跑。你会认真挑模型，认真想「这一步能不能省掉」，会为「一次编辑到底算几次调用」这种口径较真。**用户看得见的价签，反过来成了内部的一把尺子。**

算力不是一个收费机制，它是把「成本」从财务报表挪到产品界面上的动作——而所有被挪到界面上的东西，都会开始被认真对待。

第九章 · 读者的回应也是一篇作品

评论区是内容的坟场

任何内容产品的评论区里，文字写的时候都是认真的，命运却是被折叠、被翻页、被时间冲走。评论不属于写它的人，属于那篇文章的下方——**好东西掉进去就出不来了。**

VoiceDrop 的社区（App 里叫 VD 社区）从一开始就想反过来。这一章讲这件事在合规、架构、激励三层上各付了什么代价。

上线当天就被换掉的功能

最早做的是**语音评论**：按住麦克风说一句，端上转写成文字发出去。理由很硬——读者想回应，不想打字。这个功能**上线当天就被替换掉了**，换成的叫「回应文章」，形态借自博客时代的 trackback（一篇文章引用另一篇时，被引用的那篇下面自动冒出一条「有人写了篇东西说到我」）。

差别不在界面，在管道。读者的回应**走的是和原创录音完全同一条挖矿流水线**：转写、成文、配图，进他自己的文章列表。它是他名下的一篇作品，能改、能分享、能被再回应，只不过多挂一个标：**↩ 回应《原文》**。

大白话

翻译成人话：回应不再是挂在别人文章屁股后面的一句话，而是你自己的一篇文章，顺手指了一下它冲谁说的。

代价是更重、更慢、更贵——一条本来一秒发出去的评论，现在要过一遍完整的模型流水线，要花算力。收益是**每一次回应都在给回应者自己积累资产**，他不是在给别人的地里施肥。

存指针，不存快照

一条帖子存的是什么？直觉答案是内容的一份拷贝。这里存的是一个 schema-2 live pointer（活指针）——只记「哪篇文章、谁的、什么时候第一次分享」，**正文一个字都不复制**，标题和正文每次请求都从原文实时读。

大白话

翻译成成人话：快照是拍张照片存起来，原件改了照片不变；指针是记个门牌号，每次都跑去原地看一眼。

好处直给：作者把文章改顺了，社区里立刻就是新的；重新分享时最初那个分享时间还留着，帖子不会靠反复分享往前插队。坏处也直给：门牌号背后的房子可能被拆。用户删掉底层录音，社区里就留下一条空的、没标题的僵尸帖。

修法讲究在**它分了三情况，而不是两种**：

- 原文还在 → 正常显示。
- 原文没了，**音频也没了** → 整条录音被删，把这条指针**收割**掉。
- 原文没了，**但音频还在** → 这是「重新生成」的中间态：旧文章刚删、新文章还没挖出来。保留指针，只是暂时藏起来，等重挖完自己回来。

第三条是重点。「删掉了」和「正在重做」在存储里长得一模一样——都是文章不见了。不区分它俩，用户每重新生成一次已分享的文章，系统就静默地帮他取消一次分享。

一个盒子，两把钥匙

社区能存在的前提是有个可追责的主体。而核心录音流程是**零登录**的——打开就能录，身份是一把躺在 iCloud 钥匙串里的匿名密钥。

被否掉的方案是「迁移」：登录后把匿名账号的数据搬进新账号。盒子里躺着几百 MB 录音，**每次登录做一次批量复制，又慢又贵又容易搬串**。采用的方案是：登录**不搬数据，只往既有的数据盒上挂一条绑定记录**。盒子还是那个盒子，多了一把钥匙——别名是瞬时的，搬家不是。

副产品很漂亮：Apple 会话解析出来的身份**本身就是那个匿名身份**，所以旁路服务（比如下面的推荐服务）不需要签名密钥，只凭匿名令牌就能认全所有用户。

坑也埋在这里，而且是一刀切必然会掉进去的那种。App 默认永远发匿名令牌，服务端却只认会话凭证当「登录证明」——那么已登录用户开分享仍然会 403，App 好心拉起登录、重试，送的还是匿名令牌，**登录等于白登**。所以在把匿名令牌升为全局默认的同一刻，就得给分享这两个写操作留一条单一真源：可追责令牌 = 会话凭证，没有才用匿名令牌。

古德哈特定律在这里不是抽象道理。点赞一旦能换钱，互赞群和小号会在一周内出现；排序靠的正是这个信号，信号一脏，推荐就废了。把钱走另一条管子，点赞才留得住它的诚实。

Apple Guideline 1.2 的四件套

只要有跨用户的 UGC（用户自己生产的内容），苹果审核的 1.2 条一定会来敲门。它要四样东西：

1. **过滤**：在**分享的那一刻**扫标题和正文里的明确违规词（中文按子串，英文按词边界），命中就 403。**零成本、零大模型**，词表放云端一个 JSON，改词不用发版。
2. **举报即下架**：任何登录用户举报，帖子**立刻**从列表消失（待审），管理员后台复核决定移除还是恢复。24 小时的处理承诺写在文案里。
3. **屏蔽用户**：只存本机，**永不上报服务端**，客户端把被屏蔽作者滤掉——四件里最反直觉的一件，服务端甚至不需要知道。
4. **公约 + 联系方式**：第一次打开「VD 社区可见」必须同意社区公约。

第一条有段历史：更早的版本是在**生成的时候**用小模型审一遍，后来去掉了——那会把用户压根没打算给人看的私人文章也审一遍。**审核该发生在内容变成公共物品的那一刻，不是它诞生的那一刻。**

还有一条容易漏：被举报的文章，连它的**公开分享页**也返回 404「已不可用」。列表里藏起来不够，出口要一起堵。

推荐排序做成一个能随时拔掉的旁车

纯时间序刷不出好内容。但「上推荐排序」通常意味着核心接口被重写、排序长进主干、再也拆不下来。这里把它做成一个 sidecar（旁车，摩托车侧面挂的那个斗——拆掉，车照

跑)：一个完全独立的服务。

- 核心的取列表接口**零改动**，仍按时间倒序。App 拿到列表后再去问推荐服务要个顺序。
- **推荐服务挂了、或超过 2 秒不回，App 回退时间序，feed 照常出。**
- 它自带一张互动表——浏览、读完、点赞、举报，每用户去重。分数是 $(1 + \text{浏览} \cdot 1 + \text{读完} \cdot 4 + \text{点赞} \cdot 3 + \text{回应} \cdot 5 + \text{举报} \cdot (-9)) / (\text{小时数} + 2)^{1.5}$ ，再做作者打散。
- 它**完全独立**：不碰主存储、不反向调用核心、不共享任何密钥。删掉它，产品少一个排序，不少一个功能。

最值得抄的是**举报被设计成一次带负权重的互动**：-9 分，约等于三个负点赞。量级是刻意挑的——一个举报足以把一条刚发出来、还没什么互动的帖子压成负分沉底；但一两个举报**压不倒一篇真有互动的热帖**。

大白话

翻译成人话：防滥用靠权重的量级，不靠一票否决。

点赞绝不碰钱

VoiceDrop 有一套算力经济（第 8 章），于是最自然的增长设计就是点赞铸币。这条被明确否掉了，理由只有一句：**「点赞一旦能换钱就会被刷、就不再是诚实的质量信号。」**这是 古德哈特定律（一项指标一旦成为目标，它就不再是个好指标）的工程化落地。点赞只给排序提供质量信号，永不铸币；要给钱，走另一条管子。

连显示都跟着这个原则：**点赞不显示计数**，只反映「我赞过没有」。没有数字就没有攀比，也没有刷的动力。

有个小 bug 值得当标本：点了 ♥ 只发了上报，忘了同步本地状态；屏幕上心是红的，退出再进变回空心——重进时是拿本地记录重播状态的。乐观更新（先改界面、后台再确认）要更新的是本地那份真源，不只是屏幕上那个像素。

投币：读者的钱直接变成作者的算力

钱走的那条管子叫**投币**。社区详情页给作者投一枚币：作者得 2 币，投币者自己也得 0.5 币，即时到账。配套改动看着像小事：到账要**推送**给作者——在这之前奖励是静默进桶的，一份没人知道的礼物等于没送。

这条管子把经济系统接进了社交系统。点赞同声誉，投币管钱，两件事从一开始就没混过。

列表页的演化

单列列表 → **双排瀑布流**：有图的帖子按原图宽高比错落，纯文字帖按帖子 ID 哈希定色，生成一张三色暖调渐变封面——**同一篇永远是同一个颜色**。后端从「先取列表、再问排序」两跳合并成**一次索引查询**，一趟带回列表、排序、红心和回应数。默认排序后来从「推荐」**改回「最新」**：打开社区先看到新东西，比先看到好东西更重要；同时加了搜索。（另有一次「提示词」独立 tab 的试水，两天后整体撤回，账记在第 12 章。）

书帖转正，和一台叫「清理」的绞肉机

书是后长出来的第二条产品线（第 10 章）。它最初进 feed 的方式是**读取侧混入**：每次有人刷社区，后端就去拉一份要 9 秒才回来的书架 JSON 临时混进列表，外面套两层缓存掩盖那 9 秒。而且混进来的书是二等公民——它不在帖子表里，永远显示 0 赞、没有回应，在推荐排序里是**零信号**。

改法是翻到另一侧：**写入侧登记**——写完书、改完书的收尾直接把这本书登记进社区帖表。从此书帖的赞、回应、推荐排序与普通帖**同权**，慢源和两层缓存一起废掉，存量 98 本一次回填。

上线当天踩了个能当模板用的坑。系统里有个例行扫描负责清理漂移数据——它以云存储里的指针为真源，比对帖子表，把对不上的行删掉。书帖**没有对应的指针**，于是 98 行被整批当垃圾清掉。修法是让删除扫描豁免书帖，但真正该记住的是那条规律：

「以 A 为真源清理 B」这条规则，在 B 里出现了不来自 A 的行时，会立刻变成一台绞肉机。

它出错不是因为它变了，是因为世界变了——B 有了第二个来源。任何「对账即删除」的逻辑，都必须在加新来源的同一次改动里被重新申一遍。

还有个细节值得抄：书卡的**版本闸门放在服务端**，按客户端上报的版本号过滤，旧版本 App 根本看不到书卡——**不存在过渡期的报错，因为旧客户端压根不知道有这个东西存在。**

回到那个坟场

回应走同一条流水线，读者的话就变成读者自己的作品；存活指针不存快照，作品永远是最新的样子；点赞不铸币、投币另开一管，质量信号才诚实、钱才干净；排序是可以拔掉的旁车，社区不会因为「算法」的死活而停摆。

一个社区长成什么样，不取决于它的界面，取决于它**奖励什么**。评论区之所以是坟场，是因为在那套规则里，写评论的人什么也留不下。改掉这一条，剩下的都是工程。

第十章 · 一句话长成一本书

一个语音备忘录，为什么会长出一台印书机

先把这件事的位置摆清楚。市面上确实有 AI 写书工具——Sudowrite、Novelcrafter 是最像样的两个。但它们是**给职业小说家的重工作台**：你自己搭大纲、自己配人物设定、一章一章生成，最后导出稿子拿去自出版平台。它们卖的是「一间更好用的工作室」。

而「说一句话 → 得到一本完整的书 → 这本书直接出现在一个公开书架上」，2026 年 9 月做的竞品调研里，没找到第二家在做。**这一格是空的。**

那为什么是一个语音备忘录长出这个东西？因为它从第一天起赌的就是同一件事：终点不是转写稿。把一段声音变成一篇能发出去的文章，是把终点往前推一步；把一句话变成一本书，是把同一根杠杆按到底。

先做成一个藏起来的按钮

2026 年 8 月 10 日，写书第一次出现的位置是「**关于**」页最下面、一个叫「**实验功能**」的**分组里的一行字**。当天它被挪去设置页并改成扣算力。第二天，8 月 11 日，它从设置页里被拎出来，变成了 App 的**第三个 tab**。

这个 tab 长得像一个实体书架：两本一排，下面垫一层木色搁板。**第一格固定是「写书」入口**——一个虚线画的空书壳，中间一个红加号，和真书一样高。后面才是书：有封面图的铺封面图，没有的用布面缺省封面，宋体印上主副标题，颜色由这本书的 slug（书的英文短名，也是它在网址里的那一截）哈希出来。点一本书原地推进阅读器，返回时还站在原来的位置。

接下来一周，阅读器（8 月 11 日）、修书对话线（8 月 15 日）、书的微信原生分享卡片（8 月 16 日）接连长齐；书作为一等社区帖进入信息流，还要再等半个月，到 8 月 27 日（第九章）。**一条产品线的诞生方式是：先做成一个藏起来的按钮，能用了再给它一个 tab。**

交互只有一个输入框

写书页上要填的东西只有一样，叫「中心思想」：一个词，一句话，或者一整篇你已经写好的文章——文章详情页的 … 菜单里有「扩展成一本书」，把那篇文章当种子扔进来。

价签明码写在最上面：一口价 320 算力（2026 年 9 月 1 日降到 160，见第八章）。点下去之后，**你可以直接关掉 App**。这是一次 fire-and-forget（发射后不管——把请求扔出去就走人，不守着结果）：扣费成功立刻返回，剩下的活在服务器后台自己跑，跑完推送通知你，链接**直达那本书**。写失败则不推送——**不要用失败去打扰人**。

四个角色

提交之后跑起来的是一条流水线，四个角色。它们不是四个函数，是四个各自独立的代理。

- **建筑师**：只有一个，把种子拆成一本书——书名、slug、切入角度、8 到 20 章的清单（每章一句 brief）、配色、一句导读钩子。硬要求是章节必须**递进**，不是并列词条：每一章解决上一章留下的疑问；还要先定一个**贯穿全书的问题**——读者读完能回答的那一个，整份大纲围着它推进。
- **写手**：每章一个，**并行**跑。每个写手拿到的是本章的 brief 加上全书大纲——它得知道别人正在写什么，才不会重复。
- **评审**：每章一个，**必须是另一个独立的代理**。喂给它的只有两样：这一章的成品，和全书大纲。**写手的思路一个字都不给它看**，让它从零判断，存疑处自己去查。三个维度打分——准确度、新颖性、有趣度；准确度必须 ≥ 4 分，另两项 ≥ 3 ，且不留任何未解决的事实硬伤，否则打回。
- **发布**：把过审的章节组装成页面，增量发出去。

两条铁律钉在这套编排上：**写手和评审绝不共享 上下文**（一个代理这次能看到的全部材料）——共享了就不叫独立评审，只叫自己批改自己的作业。第二条：不过就**换一个新写手**照评审列的 `must_fix` 重写，不复用原写手的上下文；最多 3 轮，还不过就降级这一章、记下短板放行。**宁可放一章弱的，也不许死循环**。

边写边发

这条流水线的技术核心是：它**不等全书写完再发**。

建筑师交出大纲的那一刻，骨架就上线了：目录在，每章标着「待写」。之后每完成一章重发一次。一章过审，立刻落盘、发布、刷新目录——三件事合并成一次原子操作（要么全做完，要么当没发生）。读者当场能读到新章，目录上的进度往前走一格。

大白话

翻译成成人话：这本书是当着读者的面长出来的，不是憋完了才端上桌。

理由不是为了好看。正文一旦落到磁盘并且发布过，这一章就**永久在盘上、且已经上线**——之后会话被切断、进程被回收，都不影响已发的章。**能保证的从来不是「进程不被切」（那做不到），而是「切了也无损」**：把一个几小时的长任务，切成每章一个能在单个回合内落地的持久 检查点（存档点，坏了从这里接着来，不用从头）。

磁盘就是检查点。这 and 第 7 章里「磁盘就是队列」是同一个思想的两次使用。

被打断之后回来怎么办？规矩写死在流程里：先挨个探一遍线上每章在不在（200 还是 404），再和本地稿子对账，**只补没做的**——线上有的绝不重发，本地有稿线上没有的直接补发，两边都没有的才重走写→评→发。**绝不从头重来。**

规格文档里还写了一条反直觉的选择：**不首选后台任务**。后台任务听上去正合适——派出去让它慢慢跑。问题是它的命绑在当前会话进程上，会话一超时、容器一回收，正在跑的东西就成了孤儿，连一条完成记录都留不下。默认路线因此是最土的那条：主循环里并行派发，每章过审在前台立刻发掉。

修书：一条永不结束的对话线

2026 年 8 月 15 日之后，书写完不再是终点。阅读页的 … 菜单里多了「修改这本书」：主人继续下指令，每次 40 算力。每本书都有**一条永远在那儿的对话线**——开书的那句种子、之后每一条指令、代理改完写的说明，按时间排在一起。书从一锤子买卖变成一段可持续的对话。

关键设计有三处，都是同一个考虑的不同侧面：

- **修书是全新的会话，不接着旧的跑。**正文本来就不在主对话里（是各个写手子代理写的），背一整段历史只多花钱不多给信息。

- **文件即真源。**工作目录从临时目录挪到耐久盘上——临时目录服务一重启就没了。每次发布顺手把源稿镜像回云存储，工作目录丢了可以整个拉回来重建；连镜像都没有的老书，就去抓已渲染的网页，把正文从 HTML 里抠回来。
- **一切以文件为准，不以记忆为准。**这句话是给「一个随时可能被中断的代理」立的规矩：你以为你记得的东西不算数，盘上有什么才算数。

还有一条克制的约束：只改与指令相关的文件，**无关章节一个字都不动**。大改仍要走完整的「写 → 独立评审 → 不过重写」。

把所有权写进作品本身

「谁能改这本书」最初记在服务器本地的一个登记簿里：一台机器上的一个文件夹，一本书一个 JSON。能用，但它是单点——机器换了、盘丢了，就没人知道这些书是谁的了。8 月 23 日改掉：**产权的真源变成书自己的 book.json 里的 owner 字段**，跟着书走、跟着每次发布走；对话线也挪去了和书放在一起的云存储目录。服务器退回成执行者。

把所有权写进作品本身，而不是写进某台机器的账本。

引擎是可替换件

写书用的模型引擎两周内换了好几次腿：从一家换到另一家，又做成两条腿可切换，再到按时间窗口配额分流——每 6 小时的窗口里前 N 本走一条腿，之后走另一条。全部靠改一个环境变量加一次重启。**接口、扣费、登记一次都没变**：对 App 来说，这些换腿从未发生过。

中间踩过一个非常怪的坑。换到某家模型服务之后，请求开始被拒，理由是「高风险」。二分了一整天才定位到元凶：编程助手会自动把项目的记忆笔记注入每次请求，而那个工作目录的笔记里恰好带着某本书的内容——是它触发了对方的风控。跟文件名无关，跟目录名无关，关掉常规开关也压不住。

修法是让写书这条腿跑在一个**零记忆的独立工作目录**里，每接一单开跑前清空。**上下文污染是新一类的生产事故**：出问题的不是你写的代码，是别人替你塞进请求里的那几段字。

一份清单换掉一次考古

还有一次很有代表性的性能返工。书架接口原本要「考古」：去每本书的 index.html 里把标题抠出来，再把整个文件夹列一遍数出有几章。99 本书刨一遍要八九秒。8 月 28 日改成每本书只读一个 `book.json` ——标题、作者、分类、隐不隐藏、归谁、创建时间、封面、章节数全在里面。8.5 秒降到约 5 秒，而 5 秒基本就是地板（99 本 × 一次读取 ÷ 6 路并发）。

值钱的不是那 3.5 秒，是那个动作：**把散落在渲染结果里的信息收回成一份结构化清单，比优化那次考古更值**。渲染结果是给人看的，不该反过来当数据库用。

不吵人的收尾

图画书这一类有个特殊待遇：创建时默认带「隐藏」标记，不上书架，主人验收完改一下才公开。通知照发给书主人，用的是那本书的直链——隐藏的书在书架上找不到，只有直链打得开。这不是大功能，但它表达一件事：**东西是你的，什么时候示人由你说了算**。

你正在读的这本书

这本书就是这条流水线跑出来的。大纲由建筑师切，每章一个写手写，另一个没参与写作的代理独立评审，过了才落盘、才上线。你现在读到的第 10 章，走的也是同一遍。

把这条产品线放回全书的主线上：它不是附加功能。它是「**终点不是转写稿**」这个命题的**极端形式**。从一段声音到一篇文章，是把终点往前推了一步；从一句话到一本书，是把终点推到了尽头。

第十一章 · 一条请求真实走过的路

先看一眼地图

你在地铁上按下红键，说了四十秒。三分钟后那一行翻到「已成文」。这四十秒的声音走过七个地方。先列出来，因为后面每讲一站，都要回答同一个问题——**没有它会怎样。**

- 一台 iPhone 上的 App。
- 三个跑在 Cloudflare 上的东西：一个 Pages Function（静态网站顺带挂的一小段服务端代码，管上传下载分享这些日常活）、一个 Worker（独立部署的服务端代码，干 Pages Function 干不了的事）、以及 Worker 里的 Durable Object（DO：有名字、有自己那份持久存储、且「叫同一个名字必被送到同一个实例」的小服务）。
- R2——对象存储，只会「按 key 存一坨字节、按 key 取回来」的大柜子，不会查询也不会排序；D1——一个真数据库，能写 SQL。
- 一台在东京的小 VPS，一核一 G。
- 腾讯 EdgeOne 的国内边缘节点（离用户更近的一圈机器，请求先落上面，再由它去够源站）。

七个部件，一个人维护。所以每一个的存在都不是「架构该长这样」，而是「不这么干就有件具体的事办不成」。

第一站：一堵自家的墙

音频由 Pages Function 接住写进 R2，然后它去敲挖矿的门。挖矿跑在 Worker 里，Worker 也挂在同一个域名上，最自然的写法是

```
fetch('https://jianshuo.dev/agent/...')。
```

它 405 了。Cloudflare 遇到「同域名内部的调用」会先查 Pages 自己的路由表，在 Pages 眼里 `/agent/...` 是个静态路径，对静态路径发 POST，答案就是 405。**Worker 压根没收到这个请求。**

翻译成成人话：你在自家院子里喊隔壁，声音先撞在自家的墙上弹回来了。修法是走大门——不打域名，改打 Worker 自己那个 `workers.dev` 子域，绕开 Pages 路由表。**同一个域名下的两个服务，可能根本看不见对方。**

第二站：为什么挖矿非得是 Durable Object

挖矿有三个入口：上传完立刻触发、六小时一次的定时兜底、手动触发。三个入口意味着**两次挖矿随时可能撞在一起**：都看到同一条没处理的录音，都跑一遍转写、都调一遍模型，产出两篇一样的文章，付两遍钱。

所以挖矿必须串行。而「串行」在一个天生并发、随时能起无数实例的云函数世界里，是要专门找地方安放的。DO 就是那个地方，三个性质刚好对上：**有身份**（用同一个名字 `miner` 去要，全世界拿到同一个实例）、**有持久存储**、**能给自己定闹钟**（不需要外部调度器）。于是三个入口谁来都一样，只是把那一个 Miner 的闹钟往前拨一点，活在闹钟回调里排队做完。三种 DO，对应三种「必须有身份」的场景：

- **Miner**：全局**一个**，因为要串行。
- **ArticleEditor**：每篇文章**一个**。你对一篇文章连说三句改稿指令，必须按顺序执行——第二句改的是第一句改完的结果。DO 自带的持久队列就是这个顺序的真源，手机只管说，不管排。
- **StatusHub**：每个用户**一个**。手机开一条长连接挂上去，挖矿每走一步就推一条消息，行上的字从「听录音」翻到「挖文章」再到「已成文」。没它就只能轮询。

第三站：柜子和账本

一开始，这个系统**没有数据库**。一切都是 R2 上的对象：音频、文章 JSON、文风、微信凭证。连「这条录音处理过了吗」都不用查表——`articles/<名字>.json` 在就是成文，`.empty` 在就是没听到人声。

翻译成成人话：不建账本，直接看柜子里有没有那个盒子。有就是办过。

省事到近乎作弊，但有代价：**小 JSON 的读写代价随规模上升**。索引、计数、指针这些本该一行 SQL 的东西，全变成「取回整个 JSON、改一个字段、写回去」，还得时不时把整个前缀列一遍对账。慢，并发写还会互相盖掉。

2026 年 8 月 3 日收口：十类状态数据搬进 D1，**大块二进制仍留在 R2**；先双写两周，稳了才停掉 R2 那一半。**存储选型随规模变，不是一次定死的**。当初「一切都是对象」在当时是对的，错的是不肯换。

第四站：为一句真话，建一台服务器

你点「发布公众号草稿」。微信的接口有条规矩：**只接受从公众号后台登记过的那个固定出口 IP 打来的请求**，别的一律 40164: invalid ip。而 Cloudflare 的 `fetch` 没法指定走某个代理出去——从哪个节点出去是它说了算。

于是东京那台 VPS 上跑了个极小的中继。它是**哑的**，哑得刻意：

- 只绑本机回环口。每次请求把现成的凭据和文章递进来，它调微信、原样递回结果。
- 它不持有对象存储的密钥，不写任何盘上的业务数据**——唯一落盘的是一份「这张图传过、URL 是什么」的草稿缓存，丢了就重传一次。
- 对外靠一条 Cloudflare Tunnel（隧道：机器主动向外建连，请求顺着这条连接倒灌进来）暴露，所以**一个入站端口都不用开**；入站鉴权只有一个共享密钥头。出站仍从这台机器自己的 IP 走，白名单不受影响。
- 没有兜底**。中继挂了就发布失败，不假装成功。这是故意的：失败必须浮出来。

为什么值得为它养一台机器？因为在它之前，App 上显示的是一句写死的话——「约 1 分钟后到草稿箱」。那不是承诺，是安慰：云函数够不到微信，谁也不知道成没成。有了中继，那一行变成真实结果——真实的 `media_id`，或者微信原样返回的错误码。**宁可为一句真话建一套基础设施。**

路通了，它就成了正规出海口：后来第三方平台授权的五类出站请求也统一走这里——但中继**不接受任意 URL**，每条路径只放行对应的最小字段，仍不落盘。这台机器上还长出了写书流水线：几十分钟的 agent 循环、读写工作目录、装模型订阅凭据——云函数一样干不了。

还有一条小而硬的规矩：**保存公众号凭据前先真验一次**。服务端从白名单 IP 真去换一次 token，换不到就报错、**不写盘**——存下来的错凭据，会在你真正想发文章的那一刻才暴露。**不要把「我以为对的凭据」存下来。**

第五站：国内那一跳

Cloudflare 的节点在境外，国内用户直连，每次开屏都要跨一趟境。解法是主机名分两套：中国区商店账号走 `voicedrop.cn`（腾讯 EdgeOne 的境内边缘做反代——用户打境内这个域名，边缘替他去够境外源站），其他区直连 Cloudflare。判定依据不是 IP，不是系统语言，是 **App Store 的商店区域**——StoreKit 给出的国家码，启动时取一次，持久化，分享扩展沿用。竞速探测选线做过，上线一周整体删掉：**能查的东西不要测。**

但不能一刀切，三件事恒定不变：

- **发给别人的东西写死 `voicedrop.cn`**：分享链接、邀请链接、微信里点开的链接。发出去的东西不能因人而异——你是海外账号，收的人可能在国内。
- **WebSocket 恒走 Cloudflare**：边缘对长连接的透传没验证过，不拿实时功能赌。
- **缩略图恒走 Cloudflare**：边缘截图是它的专有能力和能力，别处没有。

于是同一张照片，原图从国内边缘出（不跨境回源，微信里秒开），缩略图从 Cloudflare 出。

代价也有。邀请奖励靠访客 IP 判断「这人是谁请来的」，可请求经境内边缘回源，云函数侧读到的「访客 IP」恒等于边缘的出口 IP——**对微信分享来的流量，这一层归因 100% 失效**，所有人看起来都是同一个。（归因本来就有三层——先看邀请链接被点开时带回的标记，再看 IP 指纹，最后拿剪贴板兜底——所以坏的是中间那层。2026 年 7 月 16 日的修法是让落地页自己发一个「第一方信标」：访客的浏览器**直连** Cloudflare 那个域报到一次，绕开境内边缘，真实 IP 只有这条路拿得到。）顺带一条硬知识：**X-Real-IP 不可信**，边缘

会用它自己看到的连接方 IP 覆写这个头；X-Forwarded-For (XFF，记录请求一路经过哪些代理的头) 是**追加**不是覆写，链条还在，但你得知道该信第几段。

第六站：回到手机上

跨境往返的时延消不掉，所以不消它，**改成不等它**。全 App 用 stale-while-revalidate (SWR：先把上次落盘的旧快照画出来，同时后台去拉新的，回来了再原地覆盖)。开屏一瞬间就有内容，网络在背后追；**失败就不清屏**——你至少还看着上次那份，而不是空白加一个转圈。

这套东西一开始在五个页面各写一遍，最后收敛成一个三十五行的公用件。收敛的真正原因不是嫌重复，是**清空时机**：缓存必须挂在销号和换身份上一起清，否则换个身份进来，屏上还画着上一个身份的文章——**缓存的正确性问题，本质上是身份边界问题**。

还有一件小东西作用大得不成比例：**每个请求都带上客户端的版本号和构建号**，服务端按构建号决定给不给某类内容。新做的书卡只发给够新的 App，旧版本根本看不到——**不存在「点开失败」的过渡期**。这套头立好，以后任何灰度都复用它。

看不见的三件事

一个人维护的系统，最贵的成本不是坏，是**不知道坏了**。**留痕**：每次大模型调用原样记一份日志，三十天后自动清；每次挖矿写一份事件日志。**叫醒**：系统级故障（转写欠费、写书失败）直接推到作者手机上，同类问题按小时节流。**后悔**：R2 数据每晚增量镜像到备份桶，被删或被覆盖的旧版本留十四天——对象存储没有版本控制，覆盖即丢。

还有一次事故值得记：新接口带 token 返回 405，不带 token 返回 401。真相是部署命令按当前 git 分支名决定环境，那次在游离 HEAD 的工作树里执行，分支名成了 HEAD，于是**静默**部到了预览环境，生产纹丝不动。405 是老代码没这路由的兜底，401 是老代码自己的鉴权——**你以为上线了，其实没有**。修法只有一条：**必须显式指定分支**。

没有一件是为了好看

这条路看着像一份过度设计的技术栈清单。但每一件都能当场回答那个问题：

- DO，是因为挖矿必须串行，不然要付两遍钱。

- 东京那台 VPS，是因为微信认 IP，而云函数选不了从哪出去。
- 境内边缘，是因为跨境慢，而慢会让人不打开。
- D1，是因为柜子当账本用到一定规模就慢了。
- SWR，是因为时延消不掉，只能不等它。
- 版本头，是因为发版有窗口期，窗口期里的用户不该看到会失败的按钮。

把这几句倒过来读，就是这套基础设施的全部设计文档。**一个人维护的系统没有余粮**——任何一个答不上「没有它会怎样」的组件，早晚变成某个凌晨叫醒你的那条推送。

第十二章 · 减法

性格写在删掉的东西里

看一个产品加了什么，你只知道它想要什么；看它删了什么，你才知道它**愿意为什么付代价**。加功能几乎不需要理由——有人提、能做、看着挺好，就留下了。删功能一定有理由，而那个理由通常比功能本身更能说明它是什么。

740 次提交里有一批很特别：它们把已经做完、有的已经发到用户手机上的东西原样拿掉。这一章不重讲它们是什么（多风格对比在第 4 章、实时预览在第 5 章、竞速选线在第 11 章讲过），只讲**为什么砍，以及砍的时候留下了什么**。

确定性比自适应值钱

最锋利的一条，因为被砍掉的方案技术上明显更「聪明」。App 有两条线路：国内走 `voicedrop.cn`（境内边缘），海外走 `jianshuo.dev`（Cloudflare 直连）。8 月 19 日上线的判定方案是**竞速探测**：同时向两个域名发一个只要响应头的请求，谁先回话用谁；再加迟滞防抖（新赢家要快出 150 毫秒才允许换线，像空调设了温差，不到差值不启停）、单边失败的兜底和 6 条单元测试。

它活了一周。8 月 26 日整体删除，换成一行判断：读 App Store 账号注册在哪个国家，中国区走 `voicedrop.cn`，其他区走 `jianshuo.dev`。探测、防抖、埋点、那 6 条单测，全删。写进架构文档的理由是五个字：**判定简单可预期**。

翻译成人话：能查的东西不要测。测出来的答案每次都可能不一样，查出来的答案永远是同一个。

自适应的代价不在写起来难，而在**没人能说出它此刻在做什么**。用户报「我这儿连不上」，竞速方案下第一个问题是「那一刻两条线各自多快」——那一刻已经过去，查不到。商店区域方案下你只问「你账号是哪个区的」，答案确定、可复现。**一个会自己找最优解的系统，你放弃的是解释它的能力。**

同一条判据更早砍掉过一个多得多的方向。7 月初有条分支在做 SDUI (server-driven UI: 首页长什么样不写死在 App 里, 服务端发一份描述文件下来, App 照着渲染)。第一阶段的 iOS 渲染引擎已经做出来——页面模型、渲染器、测试 target, 10 个提交。用户拍板不要, 整条分支连同这 10 个提交一起删除, **一行都没进过主干**。它换来「不发版就能改界面」, 付出的是「没人能只看 App 的代码知道用户看到了什么」——和竞速选线同一笔账。

删除不等于烧掉

删分支前, 先在分支尖上打了一个永久 tag (给某次提交起个不会被回收的名字):

`archive/sdui-homepage`, 推到 GitHub。任何时候 checkout 它就回到删除那一刻的完整现场, 也能重建分支, 或者 cherry-pick (把某次提交单独摘出来贴到另一条线上) 任意一个。分支上唯一被证明有价值的东西——长按段落弹操作菜单——正是这么摘出来的, 今天还活在主干里。

手法反复出现。6 月 26 日删掉 1025 行的 `mine.py` (挖矿那半边早被 Worker 取代, 微信那半边内联进了中继), 一条提交净删 615 行, 删完立了块碑 `mining/REMOVED.md`: 它做什么、为什么删, 最后一行是能直接粘贴的恢复命令。

8 月 30 日回滚「一生一次领 320 算力写一本书」那次, 账记得更狠。落地页、深链 (deep link, 一条能从网页直接跳进 App 里某个具体页面的链接)、服务端「一人一次」的唯一索引、8 条单测, 一天做完、迭代两轮文案, 当天整体撤回。回滚提交没有只写「撤销」, 它多写一段: **一并撤掉的还有一个独立的真 bug 修复, 它搭在活动的提交里, 需要的话单独再提。**

回滚是把钝刀, 会连着切掉旁边不该切的; 切掉了就在案板上记一行。删除的成本从来不是「东西没了」, 是「没人记得它去哪了」。留 tag、留碑、留恢复命令、留账, 成本几乎为零, 却把不可逆变成可逆。**能撤回的删除, 才敢删得快。**

能力可以留下, 界面可以砍掉

7 月 12 日晚上撤掉的实时预览, 是最漂亮的一次「半砍」。它连做四版, 越做越漂亮: 底部打字机卡片、正文里原地打字、整篇长出来的幽灵稿。用户一句「在真机上引入多个难预料的布局 bug」, 当天全撤, 编辑和重写回到「一次性出结果」。

但撤的只是**看得见的那层**。服务端一行没动，大模型调用照旧走流式、预览增量照发，App 收到直接忽略；客户端的预览消息解析留着，回调没挂上就当没收到；**双路收尾**留着——长文重写超过 HTTP 超时时间，走预览通道的完成信号也能让它落定，超时同时从 120 秒上调到 300 秒。

为什么值得留？非流式调用会被大模型门口的网关在约 100 秒掐断，一段两小时十二分钟、四万字转写的录音生成要 168 秒，必死；改流式后一次成功，挖出了 13 篇文章。**这套被界面带出来的基建，后来成了修超时的根基。**

被砍掉的是界面，留下的是能力。动手砍之前先分清：你要砍的是「用户看到的东西」，还是「系统能做的事」。它们经常长在一起，但不是一回事。

「顺手改的」比「要改的」更容易毁掉一个功能

7 月 14 日撤掉的是段落就地编辑 v1——长按一段文字直接改。上了 TestFlight 真机一跑错误百出，整体撤销。根因不在功能本身，在三处**顺手改了不该改的**：

1. 为拿到手指按下去的坐标，把长按手势换了实现。副作用是菜单从「按住 0.35 秒就弹」变成「松手才弹」。
2. 给编辑中的段落加了描边、内边距、淡出。**排版一动用户就觉得坏了**，哪怕功能是对的。
3. 想让光标精确落在手指按的那个字上，那套坐标换算本身就是 bug 之源。

v2 用三条铁律重做，每条都是「不许动」：长按行为一行不改；排版零变化（编辑框与只读文字同字号、同行距、同色、零内边距）；光标一律落段尾，点哪儿改哪儿交给系统原生行为。

你只想加一个功能，但你顺手改了手势、改了排版、改了光标——用户感知到的是那三样，不是你的功能。

更小的一版：6 月下旬录音页的拍照按钮一天改了三次，revert 提交直说「**大意是：今天那三次拍照按钮改动越改越不好，按要求把录音 UI 还原到昨天。**」然后又从「隐藏的透明点击区」重做成「很淡的相机图标加拍照标签」。**回到昨天不是失败，是重新取一次基线**——在被搅浑的地基上接着改，第四次几乎注定也错。

有些功能的正确做法是不做

- **「重置身份」入口**：一点就和所有录音永久解绑、不可恢复。连确认弹窗一起删掉，理由只有「太危险」。**给不可逆的操作加二次确认，不如不提供它。**
- **分享扩展跳回主 App**：官方接口对分享扩展是空操作，能绕的路子都踩在私有 API 上，真机还不跳。三次尝试整块删掉。**宁可让用户自己去开 App，也不拿审核赌。**
- **追问的开关**：设置里本来能关掉成文后的追问，后来开关删了，默认永远开。**有些选择不该交给用户**——一个没人会动、动了也说不清后果的开关，只是把设计者的犹豫转嫁出去。
- **生成时的内容审核**：早期在成文那一刻用小模型审一遍，去掉的理由是它把用户压根没打算给人看的私人文章也审了。审核挪到**分享的那一刻**做关键词扫描，零成本、零大模型。**审查的时机比手段更重要。**
- **手搓 AWS 签名**：早上为给持续集成省两秒启动开销，把云存储签名手写一遍替掉现成的库，当天自己撤回：「这条路不该自己手写签名，维护风险大，省的约 2 秒不值。」

砍得快，是因为发得快

还有一批砍得更干脆，理由更朴素——不值得。主页曾经有个「提示词」tab，试了两天，认为不值得占一个 tab，撤回，入口退回设置页；社区列表里的「提示词」分段 tab 也去掉，理由一句：**提示词淹没了内容流。**

多风格对比在 7 月 24 日整体下线，两端一起删：开关、勾选界面、对比态版本栏全移除，挖矿只按当前文风挖一篇。留下的痕迹很说明手法——旧的用户数据字段留着不删，只是**再没有人读它**；选过别的文风的用户静默回到当前文风，这被写成「预期行为」。

注意这些事共同的时间尺度：试两天、活一周、当天回滚。这不是果断，是**发布成本足够低的副产品**。改动从想法到用户手上要走两周，没人舍得撤——沉没成本替你做决定，那功能会以「再优化一下」的名义活很久；只要一天，撤销就变成和「再改一版」同等重量的选项。**敢做减法的前提，是加法不贵。**让加法变便宜的机制是下一章的事。

判据清单

1. **确定性 > 自适应**：能查的不要测。
2. **能力可以留下，界面可以砍掉。**

3. **删除要留碑**：tag、墓碑文件、恢复命令、回滚备注。
4. **「顺手改的」比「要改的」更容易毁掉一个功能。**
5. **有些功能的正确做法是不做。**
6. **砍得快，是因为发得快。**

这些被删掉的东西加在一起，画出一个固执的轮廓：**宁可笨一点也要可解释，宁可少一个选项也要少一个坑，宁可让用户多点一下也不拿审核赌，删的时候一定留一条回来的路。**这些话没写在任何产品文档里，它们写在 revert 提交的第一行。

第十三章 · 一个人和一群代理

一份可以核对的账

2026 年 6 月 18 日，仓库第一条提交是 `chore: init voicedrop repo`；到 9 月 1 日走到 738 条，如今 740 出头。中间 76 天。按天数一遍：最猛的是 6 月 23 日 70 条（去掉合并提交还有 55 条），其次 6 月 20 日 43 条、7 月 14 日 40 条，其间 43 次合并，以及十几次明确标着 `revert` 的回滚。产物也不止一个 iOS App：还有 Cloudflare 上的三套服务、一个排序旁车、东京 VPS 上的公众号中继、一套网页书架、一个给别的代理调用的接口服务。

这堆数字容易被读成「AI 让人写得快」。它的真实含义是另一件事——**当写代码不再是瓶颈，瓶颈会往上游跑。**

并行的前提是隔离

提交历史里反复出现 `Merge branch 'worktree-swift-jingling-pike'`，十几次。
worktree（工作树）是 git 的一个功能：同一个仓库同时签出到好几个目录，各占一个分支，互不打扰。因为同时开着好几个 AI 会话，共用一个目录就会互相踩脚——一个刚写完文件，另一个的编译还在读旧版本。

并行不是「多请几个人」，是「给每个人一张自己的桌子」。桌子不够，人越多越慢。

代码便宜了，规格反而贵了

740 条提交里有一套反复出现的节奏：`docs(spec)` 写清楚要什么、边界在哪 → `docs(plan)` 拆成一串带验收条件的任务 → 一串 `feat`，每个任务先写一个必然失败的测试，再写代码让它变绿（这就是 TDD）→ 评审打出的 `fix` → `docs(state)` 收口。仓库里躺着 27 份 spec、30 份计划与任务报告、41 条 `docs(state)`；算力计费、设备配对登录、持久编辑队列、接受分享、语音指令、追问、提示词注册表、邀请奖励、分享码，九个大功能全走完这套。

代码写得越快，不是越该「直接开干、错了再改」吗？恰恰相反：想清楚的成本没变，把它变成代码的成本掉到了零头，**规格于是成了整条流水线上最贵的一段。**

以前打字慢，含混的想法在敲键盘的过程里会被自己发现。现在打字不要钱了，含混会被原封不动地铸成产品——而且长得很像对的：变量名整齐、注释完整、测试全绿。

洞在计划里，不在代码里

最值钱的一条发现出现在 7 月 13 日。计划文档里照惯例给了一段「参考实现」，约束本身没错：标签不超过 40 字，整棵树不超过 200 个节点。然后对抗性评审上场——它的任务不是确认代码能跑，是想办法把它弄坏。三轮对抗评审连着打穿六处，挑三处说：

- 标签长度量的是去掉空格后的串，「`」.repeat(10000)+「A」` 就绕过了 40 字上限；
- 某类节点挂的子节点既不校验也不计数，`{ref:「sys_cartoon」, children:[500 条垃圾]}` 整包混进存储；
- 传一个空对象 `children:{}`，遍历它的循环直接抛异常，本该返回 400（「你请求错了」）的接口变成 500（「我崩了」）。

关键在于这几刀砍中的位置。三条修复提交的正文写着同一句自白：

这些洞是计划本身写错的（实现者是逐字照抄的）。

实现没走样。**你的实现者会逐字照抄你的计划，包括你计划里的错。**人照着一份有毛病的设计写代码，往往写着写着「觉得哪里怪怪的」就顺手改了；代理不会。

由此写死进计划两条规矩。**校验器必须是全函数**——对任何输入都得给出确定答复，「过」或「不过」，不能自己先炸；它一崩，400 就变成 500，而 500 意味着这个洞可以被利用。**输出是要落盘的，必须永远满足自己的校验器**——恢复默认值居然能产出一份连自己都会拒收的文档。

还有一个动作值得单说：每次修完代码都回头改计划文档，理由只有五个字——**「免得再被照抄一遍」**。评审的火力要往上游移：审代码只救这一次，审计划才救下一次。

测试从「保险」变成「契约」

CLAUDE.md 里有条铁律，6 月 24 日加的：任何改动前后各跑一遍单测。iOS 第一个测试 target 是 7 月才建的，从 0 涨到 125，如今 214。一批「决策逻辑」被刻意抠成纯函数再钉死：何时推荐升档订阅、何时弹评分请求、要不要问署名、走哪条线路、社区搜索怎么过滤。**测的是行为契约，不是实现细节**——实现细节明天就会被某个代理重写。

服务端的规矩叫「**向下兼容即契约**」：永远伺候老数据和老客户端，老契约钉绿才推新版本。最有仪式感的一条是**删掉一段遗留代码之前，先删它的测试**。顺序是关键：先删测试，你必须写下「我知道我在放弃这种老数据」；反过来就成了「被测试挡了路，把挡路的搬开」。**同样两行删除，一个是决策，一个是事故。**

「有响应」不等于「能用」

语音改稿换了听写后端，两次上线都通过了冒烟测试（最粗的一层验证：点着火看冒不冒烟），两次都完全不工作，用户反馈是「说话和没说一样」。根因是中转服务把二进制音频帧隐式转成了字符串，每帧都变成「[object Blob]」这十三个字符。而冒烟测试之所以说「通过」，是因为它只发配置帧、只检查「有没有字节回来」——回来的正是对这十三个垃圾字符的回应。**合格的冒烟测试必须流真实数据、断言解码出来的结果。**

当外部世界给你限流

苹果给每个 App 大约 20 次/24 小时的上传额度，高强度迭代日里「每次 push 就发一版」直接烧光，CI 成片返回 409。修法不是加重试，而是把默认值翻过来：默认只跑不签名的编译校验，提交信息带 [tf] 才真打包上传——整个历史里带这标记的只有 51 条。**但这条闸只活了 24 天。**8 月 2 日它被撤销了——恢复每次推送都自动发 TestFlight，理由写在提交里：苹果那约 20 包/24 小时的限额当然还在，撞上 409 就是额度到了，等窗口刷新就好，**但不要为此把默认值改成「默认不发布」**。这个反转比原来那条方法论更值得记：限流是外部世界的成本，而「每次推送都能出包」是你自己的节奏——**宁可偶尔吃一个 409，也不要让节奏迁就限额。**唯一没变的那句是：CI 绿不等于出包。另一条纪律踩了两次才写：某个版本号一过审，那趟「列车」就永久关闭，同号构建再也传不上去；第二次当天就加了守卫——构建时查在售版本，工程版本没高过它就就地加一。

一个新的失效模式：上下文污染

写书引擎接某家模型服务时，请求被风控判成「高风险」直接拒绝。二分了一整天才定位到元凶：编程助手会自动往上下文里注入项目记忆笔记，而那个目录的笔记里恰好带着某本书的内容——风控读到的是那段内容，不是你写的提示词。它跟文件名、目录名统统无关：搜代码搜不到，看日志看不到。修法是让这条腿跑在一个独立的零记忆工作目录里，每次开跑前清空。

代理的输入不只是你传的参数，还包括系统提示词、项目记忆、工具返回值、上一轮对话。

「我传了什么」和「它收到了什么」第一次成了两件事。

人在哪儿

这不是「AI 全自动」的故事。「用户当场指出」「用户拍板」「用户要求还原」是提交历史里的高频词，而且每次都落在关键位置：

- 写书功能拿一个身份查询接口当门槛，用户一眼看穿：**那个接口只做归因，不是门槛**——匿名令牌是客户端自造的随机数。三个补丁之后是最朴素的答案：收钱。新账号送 200 算力，一本书 320，伪造账号天然被余额挡住。
- 实时预览做了四版，用户一句「引入多个难预料的 bug」，当天全部下线。
- 线路选择做完六条单测的竞速探测，用户说不如「判定简单可预期」，一周后整体删除（详见「减法」那章）。**能查的东西不要测。**
- 拍照按钮一天改五版，用户说「越改越不好」，还原到昨天。

共同点：**全都不是「代码写错了」，而是「这个方向本身不对」**。代理能把任何方向执行得又快又漂亮，包括错的那个。人的角色于是从「写代码的」变成「定义什么是对的」。

给未来的代理留账

长期文档里，一条从来没真跑通过的路被诚实写成「待验证」，后来第一次拿两台真机跑当场挂了两次——**所有「待验证」本质上都是「已知会坏，只是还没坏给你看」**。一次「合并 → 回滚 → 一模一样重做」的曲折被专门写进文档，理由是「免得 git log 误导后人」。有条提交信息里专设一节叫 `Honest scope (not faked)`：照着设计稿做，但不照着设计稿撒谎。一次整体回滚里还写明「一并撤掉的还有一个独立的真 bug 修复」。**留账是这套工作方**

式的必要成本，不是美德——收益在三周后，下一个会话不管是人还是代理，读到的是「为什么」。

真正变了的三件事

76 天七百多次提交，速度是这故事里最不重要的部分。真正变了的是三件：**规格成了最贵的东西**——「你想清楚了没有」成了唯一的瓶颈；**评审的火力要往上游打**——实现者会逐字照抄你的计划，包括错的那部分；**人必须留在「什么是对的」这个位置上**——方向、取舍、什么时候该停、什么该整体删掉，没有一条能从提交数里读出来，也没有一条能外包。

第十四章 · 什么能抄，什么抄不走

前面十三章讲的都是「怎么做的」。这一章只回答一个问题：**那又怎样。**

做产品的人最爱谈护城河——就是别人想抢你生意时，得先跨过去的那道沟。谈的时候通常很含糊，说着说着就变成夸自己。所以先立一把尺子：

一个功能能不能被抄，不取决于它有多难想到，而取决于抄它需要多少「不能一次性买到」的东西。

钱能买到的：程序员、服务器、模型 API、一次产品会上的决定。钱买不到的：踩过的坑、用户在你这儿攒了两年的东西。用这把尺子把前面讲过的能力过一遍，会分成很清楚的三层。

第一层：一周就能抄的

这一层得说得难听一点，因为它们最容易被拿来当卖点吹。

一个红键 + 自动上传

打开就是一个大红键，按下去录，松手自动传走——这是第 1 章的入口。它是个好决定，但它是**交互层**的东西。Voicenotes、AudioPen 早就这么干了。任何一家想跟，一个迭代周期的事。

「算力」这个计费单位

第 8 章讲了账本怎么记、为什么用整数微元。那些工程细节确实是纵深，但**「按算力计费」这个单位本身一点都不稀奇**。腾讯 ima 的 copilot 就叫「算力」，新用户送 500、每天登录领 100；Notion 的 Custom Agents 是 10 美元换 1000 credits；Sudowrite 卖 credits；即梦、可灵全在用积分。**23 算力 = 1 元** 只是把「透明」这件事做得更实——把汇率钉死、把每一笔明细摊开给你看。它是个态度，不是个壁垒。对手一周就能跟进。

翻译成成人话：这不是发明了一种新货币，只是在一群写着「随便吃」的餐馆里，第一个把每道菜的成本贴在门口。贴出来需要勇气，不需要技术。

「发到公众号草稿箱」这个动作

「一按就进草稿箱」听着很独特，但**这个动作本身国内早有人做**——壹伴、135 这类公众号 AI 编辑器早就接了这套接口。VoiceDrop 稀缺的不是接了这个口，是**接口前面接的是一段口述**；口子本身不稀缺。

这三样的共同点很整齐：**它们是决定，不是积累**。一个决定可以在一次产品会上被完整复制，复制的成本就是开会的那一小时。凡是能被一句话说清、说清之后照做就能做出来的东西，都不是护城河。

第二层：抄起来很痛的

这一层不是想不到，是**做起来要流血**。

锚点协议

「你说改第 3 行，模型改的就是第 3 行。」这句话在竞品调研里**零竞品**——最接近的 Aqua Voice 改的是你刚说出口、还在输入流里的文本，AudioPen 能重写但是按钮式的整篇重来。这次调研没查到有谁在做「对着一篇已经写好的稿子，用嘴做定点修改」。

为什么没人做？因为它不是一个提示词，是一条贯穿全栈的**契约**。至少得有：

- 把文章结构化到**行和图**的粒度——图片自己也占一个行号，否则图片后面的段落号就会比真实位置小；
- 客户端和服务端跑**完全相同的编号算法**（一份 `linenum.js`，一份 Swift 的 `bodyRows`），改一个必须同时改另一个，而且**必须成对发版**——新服务端配旧客户端，行号就会漂；
- 不让模型自己去数行，而是把「第 3 行是什么」做成一张对照表塞给它读——模型数长文章会数错；

- 把语音指令解析成定点修改，配一整套版本记录和撤销重做；
- 多篇文章各自从第 1 行重新编号，所以每条指令还得带上「说的是哪一篇」。

任何一环偷懒，用户就会遇到「我说改第 3 行，它改了第 4 行」。而这种错的代价是不对称的：**它只要发生一次，用户就再也不会信任语音改稿了**。一个不被信任的功能，等于没有。

「永不丢失」那一整套

第 7 章讲的那些——磁盘即队列、上传交给系统重试、收尾做成幂等、照片的三层保证、晚到照片在服务端补写——有一个共同的、要命的性质：**用户永远看不见它们**。没人会在应用商店给「我的录音一次都没丢过」写五星好评。

所以也永远不会有人在产品会上提出要做它们。这些需求不是想出来的，是**被真实故障砸出来的**：苏州弱网上传中位数 214 秒的谜案（真凶不是文件大，是音频在排队等照片一张张重试）；分享图片时先传音频、结果服务端抢跑写下「没检测到语音」的双传事故；把空录音区分成损坏、静音、无语音、超时四种原因，才知道该重试哪一种。

大白话

翻译成成人话：这些不是设计出来的功能，是疤。没受过那些伤的人，连需求文档都写不出来——他不知道要防什么。

发布出口的脏活

为了让文章真能落进微信草稿箱：微信认固定出口 IP，就在东京专门架一台机器、开一条隧道、写一个什么逻辑都没有的哑中继；为了确认用户填的凭据是真的，保存前真去微信换一次 token，而不是只做格式校验；为了链接卡片有图有摘要，把每篇文章自己的第一张照片做成题图和 og:image，而不是甩一张通用封面。

这些活**没有任何技术上的优雅可言，只有「做没做」**。调研里那四条结构性原因的最后一条正是这个：发布是脏活，是不可复用的地域性苦力，美国公司不愿意做，中国的会议工具公司不认为自己该做。

这一层的共同点也很整齐：**它们是时间和事故的函数，不是聪明的函数**。再聪明的团队，也没法用聪明去替代「上线一年、被真实用户砸过一年」。

第三层：抄不走的

前两层说到底还是代码，代码总是能被重写的。真正抄不走的东西，一行都不在代码库里。

用户攒下的文风

竞品里唯一沾边的是 AudioPen 的 custom style——据第三方资料，你可以上传自己过去的文章当样本（官网那一页需登录，未能直接核实）。但它停在「一次性喂样本 → 一个 style 槽位」。第 4 章那套东西是：有版本（v1 到 v10 随你切）、有历史、有一份持续长大的语料数据集、能精确记到每一篇文章头上（这篇是用第几版文风写的，撤销重做之后标签还是对的）。

但关键不在技术。技术上，一个对手花两周就能做出一模一样的功能。关键在于**那是用户自己攒出来的**——他往里喂过的每一篇旧文章、每一次「不对，我不这么说话」的修正，都沉在里面。他带不走，对手也复制不来。功能可以一夜之间对齐，**时间不行**。

已经发布出去的东西

文章在社区里挂着，书在公开书架上摆着，读者投过币，订阅还在续。这些和文风资产是同一种性质的东西：**属于用户，不属于产品**。

一个人不会因为另一家的转写准了 0.5%，就把发过的一百篇文章和攒下的读者一起搬走。搬家的成本不在导出数据——数据能导出，很早就做了整包 ZIP 导出。搬不走的是**关系**。

三者叠起来的复利

复利就是利息也开始生利息——长出来的东西反过来喂增长本身。单看每一项都能被抄：文风学习两周，草稿箱接口一周，社区更简单。但把它们乘起来：

学了你三年写法的模型 × 一按就进草稿箱 × 读者在这儿给你投币

护城河不在代码里，**在用户积累的东西里**。这跟 Granola 在 2026 年 3 月用 15 亿美元估值买到的是同一种逻辑——买的是企业会议数据的沉淀——只是标的从「组织的会议」换成了「个人的声音」。

必须承认的脆弱

上面那套听着挺美，但它有致命的前提。把风险摊开：

- **规模是最大的软肋。**整条护城河建立在「用户攒了东西」上，可这要求用户**先来、并且留得足够久**。在还没有足够多用户之前，这条护城河只是一张设计图。更刺眼的是调研的结论：钱高度集中在企业会议场景（Otter 2025 年过 1 亿美元年经常性收入，Granola 一个季度营收涨 250%），个人创作场景几乎无人重注。这既是机会——没人跟你抢；也是警告——**可能是因为这门生意不好做。**
- **平台风险。**发布出口依赖微信的接口和规则，分发依赖 App Store 的审核。1.12 版初审就被拒了，理由是订阅类 App 的**元数据**里必须带最终用户许可协议链接（App 内有不算）。这类规则随时会变，而且变的时候不会跟你商量。
- **模型是别人的。**引擎换过好几家，接口和扣费能保持不变是好设计（第 12 章），但它同时也在说一件事：**核心能力是租来的**。真正属于自己的，只有围绕模型建起来的那些契约、数据和用户关系。
- **这是一个人的系统。**第 13 章那套方法论让一个人能维护这么多东西，但它的上限也就是**一个人**。
- **转写正在变成大宗商品。**大宗商品化就是一样东西谁做都差不多，最后只能拼价格——讯飞听见机器快转 0.33 元/分钟、通义听悟 ASR 0.6 元/小时，那条线已经打没了。市场数字也在说同一件事：语音转文字 API 市场的年复合增速 14.4%，只有会议助手赛道 24%-28% 的一半，**价值正在从「转写」往「转写之后」跑**。所以「更准的转写」绝不是该押的方向——这既是判断，也是约束。

这一章的一句话

把三层码在一起，结论其实很朴素：**能被一句话说清的，都能被抄；要靠时间和事故换的，抄起来很痛；长为用户身上的，抄不走。**

而更朴素的一层是：第三层的价值，完全建立在第一层和第二层持续可用之上。用户不会为了一条看不见的护城河留下来，他留下来是因为红键好按、改稿改得准、录音一次没丢过。**护城河是结果，不是策略。**

第十五章 · 下一步在哪

它已经不只是一个 App

这本书从头到尾在讲一个 App：你按下红键说话，一篇文章出来。但在写这本书的时候，同一套东西已经悄悄长出了第二副面孔——它同时是**一组任何 AI 助手都能调用的工具**。

入口是一个 MCP 服务（Model Context Protocol，一套公开的约定，让 AI 助手能去调用外面的工具，就像给助手插了一排 USB 口）。地址是 `voicedrop.cn/mcp`，上面挂着 **44 个工具**：读文章、写文章、切版本、看版本历史；读文风、写文风、喂样本、蒸馏文风；催挖矿、按指令重写；刷社区、读帖、发帖、投币；查算力余额和账单；生成分享链接、发公众号草稿、打小红书图文包；分享自己的提示词、收下别人的提示词；列书架、读整本书、读某一章、写新书、修书。

大白话

翻译成成人话：以前你要打开 App 才能做的每一件事，现在你的 AI 助手可以自己去做。

接上去的过程本身就很说明问题。手机「设置 → 账户」里有一个 6 位十六进制码，把它给助手，助手调一次 `login`；手机上立刻弹出一个 4 位数字，把数字报给助手，助手再调一次 `login`，拿到令牌。为什么非要两次？因为 4 位数字是服务端在第一步**才现生成的**——第一步之前它不存在。而且一个 6 位码可能同时匹配到多个账号（最多 10 个），服务端给每个候选推**不同**的 4 位数字。报对了，才同时证明了两件事：手机在你手上，以及你要的是哪个账号。省掉这一步，别人猜中一个 6 位码、再骗你在手机上误点一下，就能接管账号。这个握手只活 2 分钟。

更值得注意的是：这不是后来加的。仓库 2026 年 6 月 18 日初始化，**6 月 19 日**设置页就有了「点按复制我的 ID、长按复制访问令牌」（42b78b8、efafed4）——长按拿到的就是可以直接 `curl` 自己数据的凭证；**6 月 21 日**加了一个叫「给 Agent 用」的链接（c110bbd），指向一份专门写给代理看的说明书。第一周就把「给代理用」摆在了设置页上。**这不是一个功能，是一个从第一周就在的设计意图。**

顺着这条线往下推，「个人内容操作系统」就不是修辞了。你的文章、你的文风数据集、你的算力账户、你的书架，全都变成了可以被程序访问的资源——你自己的代理可以读它们、改它们、花你的钱、替你发布。这条线上目前唯一硬性做不到的是**上传录音和照片**：MCP 服务跑在远端，它读不到你电脑里的文件，唯一的办法是把几兆的二进制塞进模型的上下文里，那是灾难。这是物理限制，不是没做。

终点还能往哪儿推：已经落地的那几步

全书的命题是「终点决定形态」。这半年里，终点被往前推过好几次，每次都长出新东西。

书之后是有声书。2026 年 8 月 15 日上线：书架的章节页上多了一个「听本章」。第一次点，服务端一边用语音合成把这一章念出来、一边把音频直接吐给你（实测第一个字 1.3 秒出声，8 分钟的音频 55 秒就合成完了），同时把完整音频写回存储；第二次点，直接播缓存好的 mp3，能拖进度条。**一章只花一次合成的钱，哪怕你听到一半关掉了**。从「能读」到「能听」。

书之后是 PDF。8 月 23 日上线：每本书的目录页注入一个「下载 PDF 版」。服务端拿一个真的浏览器去渲染一个专门的打印视图——封面页、导读、所有已完成的章节，没有导航件也没有播放器——渲染完存起来。改过书之后自动失效重渲。从「在线看」到「拿得走」。

内容要出现在人所在的地方。微信小程序（文章、社区帖、书三类分享路径，三端一字不差），好友分享发的是小程序大图卡片；安卓版（微信登录换令牌的后端跑在云函数里）；英文版（网页做了一份英文镜像，平时只改中文页，每天凌晨一个定时代理对比校验码、只把变动的页重译，iOS 那边用的是字符串目录——代码里正常写中文，中文字面量本身就当翻译的钥匙）。仓库里还有一份 `MULTIPLATFORM.md`，列着五个目标：小程序、安卓、手表、网页、Mac。**做出来的是前两个，后三个到今天还只是清单上的字**。

夜班编辑。这是最值得单独说的一个。2026 年 7 月 26 日起，每天凌晨 2 点，一个无人值守的会话通过 MCP 连上来，读建硕的文章，给其中几篇写候选修订版，再写一份晨报。它自己维护一本打法手册和一张编号实验台账，第二天早上你看到的是 diff，采纳不采纳你说了算。它的手法很讲究：直接写文章会把「当前版本」推到新版上去，所以它三步走——先记下当前是第几版，写完，再把当前版切回原来那版；候选留在历史里，不动线上。它的红

线不靠自觉，靠**把工具拿走**：发公众号、删文章、分享、投币、改文风，这些工具在启动参数里被直接掐掉，它夜里只能做一件事——加一个版本。北极星指标是采纳率。

这是「代理替你干活」的第一个真实形态：**不是你叫它，是它自己来**。它现在跑在一台 Mac 上，状态存在用户自己的存储前缀下（这个设计是刻意的——将来要按用户复制成产品功能）。但把它搬进服务端、变成每个用户都能开的付费功能，**还没做**。

同期还有几件各自把某一段做窄做深的事：无语音纯照片的记录走「观察→立意→写作→审稿」四阶段流水线；题图提示词有一个从线上拉现行版本、用真实文章出真图并排对比、一键上线的调优台；邀请奖励用了三层归因（点链接、IP 指纹、剪贴板兜底）；已发布的 78 本书全量镜像进了一个私有仓库。

几条有据可依的方向

下面这几条不是愿望，每条都挂着一个已经查证过的数字或者一次已经发生的事，并且标注了把握程度。

一、押整条流水线，不押更准的转写。依据是两条增长率的差。语音转文字 API 市场的 CAGR（年复合增长率，就是每年平均涨多少）是 14.4%，而会议助手——也就是「转写之后的那些处理」——是 24% 到 28%。**价值在往下游跑，而且跑得快一倍**。与此同时转写本身已经被打成了大宗商品（谁做的都差不多，只能拼价格，像卖钢筋和小麦）：讯飞听见机器快转 0.33 元一分钟，通义听悟的 ASR 0.6 元一小时。在这条线上比准确率，是去和按吨报价的对手拼刺刀。

二、个人创作场景可能被严重低估。依据是 Plaud：靠个人消费者（不是企业按座位买单）做到约 2.5 亿美元营收、约 20 亿美元估值，媒体口径说它没拿风投。这证明个人这一侧有真金白银。但它卖的是硬件和转写，不是成品。整条赛道的钱几乎全压在企业会议上——**这既是一块空地，也是一个警告：可能所有人都算过这笔账，只是算出来不划算**。

三、「一句话成书 + 公开书架」是这次调研没找到第二家的空白。现有的 AI 写书工具全是给职业小说家的重工作台：你得自己搭大纲、维护设定集、逐章生成，导出去自己发。「说一句话 → 一本书 → 直接出现在一个公开书架上」，调研里没找到第二家。所以这条线的机会不在「写得更好」，在**门槛更低 + 直接可发布**。

四、文风资产会想要被继承、被携带、被交易。这条是**推论，不是事实**，我标出来。逻辑是：如果文风真的是资产，它自然会想要版本（已经有了）、想要被分享、想要被别人「收下」。这条路已经被走通过一次——提示词分享码「魔法数字」：一串 7 位数字加一个短链，你把数字念给朋友听，他就能把你的提示词收进自己的 App；服务端出图时还会把这串数字写进图片的元数据里，配图从此可溯源。**提示词能这么走，文风为什么不能，这是个待验证的问句，不是结论。**

五、代理主动性。夜班编辑证明了「代理自己来找你」这个形态可行。往前推一步，最自然的下一站是让它**在你需要的时候来，而不是每天 2 点来**。定时是最笨的触发条件，它只是最容易实现的那个。

六、必须先解决的近处问题。诚实收尾：一次性加油包到今天还没做——商品表里只有自动续期订阅，当月把算力烧光的人现在仍然买不到算力，只能等下个周期或者升到高一档。上一版补的是「不再是死胡同」，不是「现在就能买到」。至于规模和平台依赖这两条更大的欠账，第十四章已经摊开过，这里不再重复。

终点在哪，形态就长成什么样

把终点从「一份转写稿」挪到「一件作品」，长出来的是文风资产、定位到行的锚点、公众号草稿箱这个出口，和一个会给作者投币的社区——这本书前十四章讲的就是这四样东西怎么被那一次挪动逼出来的。

如果继续往前挪，把终点定成「一个被人读到、听到、买到的作品」，会长出什么？**这本书答不了**。能答的只有下一批提交。

附录 · 完整功能清单

前面十五章讲的是「为什么这么做」。这一篇不讲道理，只列东西——把 740 次提交里长出来的每一件事，按模块摆出来。它更像一张零件清单：你可以从任何一行进去，回头翻到对应的章节看它为什么长成这样。

大白话

看法：粗体是这个模块的骨头，缩进的小字是它身上真实存在的细节。数字都是实数，不是约数。

一、录音与上传

- **一键录音**：打开即录，红键是全屏接管界面，点一下开始，不做「按住说话」。
- **暂存名 → 正式名**：录制中文件叫 `recording-<ts>.m4a`，录完 `finalize` 成功才改名成 `VoiceDrop-<时间>-<时长>-<星期>-<时段>[-<城市>-<区>].m4a`。改名这一步就是「这条录音完整」的凭证——半截文件永远拿不到正式名。
- **文件名即元数据**：时间、时长、星期、时段、城市区县全写进文件名，服务端不用查库就知道这条录音是什么。
- **为语音调过的编码**：单声道 AAC，标准档 16 kHz / 32 kbps、高质档 24 kHz / 64 kbps。原来是 44.1 kHz——但这段音频只喂给播放器和火山 ASR（原生就是 16 kHz），高采样率纯属浪费。改完 5 分钟录音从 2.4 MB 降到约 1.2 MB。
- **Documents 目录就是待传队列**：盘上还躺着 `VoiceDrop-*.m4a` = 还没传完。不另建队列表，因为任何独立的队列都可能和真实文件对不上。
- **后台传输**：走 `background URLSession`，录完立刻锁屏、切走、进程被杀，传输都由系统守护进程接着做完。完成事件在进程复活后回放，收尾逻辑幂等——「进程一直活着」和「隔世重启」两条路收敛到同一结果。
- **重灌触发点**：App 启动、网络恢复（`NWPathMonitor`）、新录音入队、前台刷新，任一发生就重扫盘上残留。

- **录音时拍照**：停止键右边一个极淡的相机图标，点开是不打断录音的取景器（只开视频通道）。方形取景、九宫格、连拍成胶片条、可逐张删、支持相册导入（≤9 张）、前后置翻转。旋转角度读手机物理朝向，不写死角度。

二、挖矿：一段声音怎么变成文章

- **火山 ASR**：中文转写走豆包，文件式接口 `volc.bigasr.auc`。
- **Claude 挖文章**：一段录音可以挖出多篇，不是一篇。结构化输出直接落成文章数组。
- **四种「没东西可写」**：`corrupt`（文件坏）、`silent`（没声音）、`no-speech`（有声音但没人说话）、`too-short`（太短），外加 `no-article`（说了但挖不出文章）和 `asr-error:<码>`。任一命中就写 `.empty` 标记——App 统一显示「无语音」，但服务端知道具体是哪一种。
- **「已处理」的定义**：`.json` 或 `.empty` 存在即已处理。音频永不自动删除，只有用户在 App 里删。
- **ASR 断点**：转写结果单独存 checkpoint，重挖不重复付 ASR 的钱。
- **请求级热词**：单一真源 `agent/src/asr-hotwords.js`，批量挖矿直接带 `corpus.context`，流式路径由 Worker 拦截注入。加个词只改数组。
- **触发路径三条**：上传落盘即派发、每 6 小时定时、手动 `POST /agent/mine/trigger`。
- **逐条日志**：每条录音的处理过程写 `minelogs/<日期>/`，每次 Claude 调用的原始请求响应写 `llmlogs/<日期>/`（30 天生命周期），都有管理页可看。

三、文风

- **文风是带版本的资产**：`CLAUDE.json`，schema-3 版本信封，和文章用同一套版本机制，走专用 `/files/api/style` 路由读写。
- **版本号跟着文章走**：`articles[i].style = N` 是 per-article 字段。曾经是正文里的隐形注释 `<!-- style: 风格 vN -->`，因为 iOS 剥注释、服务端不剥，导致「改第 3 行」错行 +1，一次性迁移掉了：扫 382 篇、迁 50 篇、线上残留 0。
- **蒸馏文风**：喂样本进风格数据集（`style/<id>.json`），从样本里提炼文风，而不是让你从预设 tone 里挑一个。
- **换风格复用旧版本**：换回用过的风格时匹配已有版本，不重新生成。

四、改稿

- **用嘴改稿**：长按红键对某篇文章说「把第三行改得口语一点」，一条 WebSocket 连到 `ArticleEditor` 这个 Durable Object。
- **行号由服务端算**：模型不会数数，所以行号是服务端编好递进去的（`linenum.js`），编号前把一切 HTML 注释剥掉。
- **严格串行**：多条指令在客户端排队，上一条返回 `updated` 才发下一条——每一刀都改在上一刀的结果上。
- **听写走火山流式**：不用 Apple 本地识别。服务端是哑中继，注入密钥、剥掉客户端 Authorization，App 永远碰不到火山凭证。
- **长按菜单**：长按配图 → 图片风格（宫崎骏 / 广告 / 水彩 / 素描 / 油画 / 胶片）；长按段落 → 改写这段（更简洁 / 更口语 / 更书面 / 扩写）、插入公众号题图、拷贝。菜单内容是服务端配置（`GET /agent/ui-config`），改 R2 就能改文案，零部署。
- **版本与撤销**：每次改动铸一个版本，可回看历史、可切回任一版。
- **编辑用 Haiku**：挖矿仍用 Opus。模型 id 写在 R2 `config/model.json`，每轮动态读，换模型零部署。

五、照片

- **标记即键**：正文里的 `[[photo:<相对 key>]]`，token 本身就是照片的 R2 键。早期用的是 `[[photo:N]]` 下标，改一张就把后面全改错位，现在自包含、零耦合。
- **键的形状**：`photos/<录音时间戳>/<偏移秒>-<3位随机>.jpg`。偏移秒就是「第几秒加进来的」；三位随机尾是为了同一秒内多张照片不互相覆盖——九张相册导入的回调几乎同时触发，旧方案会静默丢照片。
- **一个照片端点**：`GET /files/api/photo/<完整 key>`，公开无鉴权。社区、公开分享页、导出、甚至主人自己的详情页，全走这一个。
- **一张都不能丢，三层保证**：① 写盘前重新列一次照片目录（兜住照片在 ASR 期间才传完的竞态），正文没引用的按序补到末尾；② 晚到的照片由服务端补写——照片落盘 poke 一下 Miner，60 秒合并窗口后回头把它塞进已成文的文章，并铸一个 `photo-backfill` 版本；③ iOS 有落盘队列，成功才删。

- **删掉的不复活**：补写时算的是「所有版本里出现过的键」的并集——出现过又被你删掉的，是你的意志，永不复活。
- **AI 改图**：口述「把这张变成广告」→ 蒸馏 prompt → 调 paint 服务出图 → 回调写回。失败写原图副本、不扣费。

六、出口：文章往哪儿去

- **公开分享链接**：voicedrop.cn/<短码>，短码是 HMAC(key) 的前 10 位。
- **分享卡片元标签**：og:title、og:description、og:image 全部随 ?s=<第几段> 变——你分享的是哪一节，卡片就显示哪一节的标题、摘要和那一节自己的第一张照片。微信的抓取器只读 <meta name=description> 不读 og:description，所以两个都发。没照片的段落就不发 og:image，是一张干净的文字卡——不硬塞一张通用banner，因为每次分享都同一张图看起来像垃圾。
- **公众号草稿**：同步发布，等真实结果（created / updated 或真实 errcode），不是发完就不管。Markdown → 微信 HTML 的转换在东京那台 VPS 的 relay 上。十张共享题图按哈希分配。
- **小红书图文包**：xhs_pack，打成可直接发的图文。
- **微信小程序**：文章、社区、书三类分享路径，iOS 和安卓分享给好友时发的是小程序卡片。
- **导出**：带照片的 HTML 导出，标记在不能显示照片的地方一律剥掉。

七、VD社区

- **帖子是活指针**：community/<shareId>.json 只存 {shareId, owner, articleKey, author, firstSharedAt}，不拷贝正文。改原文，社区里立刻跟着变；删原录音，指针自动回收。
- **合规四件套**（Apple 审核指南 1.2）：举报、拉黑、内容过滤、24 小时内处理。被举报的帖子连公开分享页一起 404。
- **投币**：读者可以给作者投算力。点赞绝不碰钱——点赞是表态，投币是转账，混在一起两个都不成立。
- **推荐排序是可拔掉的旁车**：voicedrop-reco 单独一个服务，拔掉它社区照常工作，只是排序退回时间序。

- **举报当负权重**：举报信号进排序，不只是删帖开关。

八、算力

- **锚死的换算**：23 算力 = 1 元。单位无现金价值、不可提现、不可退款。
- **微元整数**：钱一律以 $1e-6$ 元的整数存，一律 `Math.ceil` ——只多收不少收，绝不亏。未知模型成本记 0。
- **手感**：一条录音挖一篇约 2 算力，Haiku 改一刀约 1.4，Opus 约 4，写一本书一口价 320。新用户送 200。
- **两张表**：`account`（余额 / 已赠 / 已花）+ `ledger`（只追加的流水，带余额快照）。用 `db.batch()` 原子化。
- **闸门在花钱之前**：挖矿在 ASR 之前判、编辑在调 Claude 之前判，不够就拒绝，不空花。
- **fail-open**：扣费失败绝不中断挖文章——宁可少收一次钱，不能让用户的录音卡住。
- **非终态与终态**：余额不足写 `.blocked` 标记，是非终态（下次够了删标记重挖）；录音超 3 小时是终态。
- **硬闸**：单条录音 ≤ 3 小时、单篇编辑 ≤ 100 次（数的是真实编辑次数，不是 Claude 调用次数）。
- **邀请奖励**：三层归因（链接 / IP / 剪贴板），按币记价的实时汇率入账。

九、写书与书架

- **四个角色**：建筑师切大纲、写手每章一个并行写、评审独立复核（绝不是本章写手、不喂写手思路）、发布器增量上线。
- **过审即发**：每章写完过审就立刻发布，绝不攒到最后。会话断了、任务崩了，已发的章一章不丢。
- **三种书**：科普（费曼式）、绘本（图为主，创建即隐藏，主人验收后上架）、小说。三种共用一套骨架，只换写作模块。
- **修书是一条永久对话线**：这本书的历史里，第一条是开书种子，之后每条是修改指令加回执。
- **有声书**：章节页「听本章」，首播火山流式边合成边播并写回 R2，二播直接放 mp3 且支持 Range 请求。

- **整本下载 PDF**: Browser Rendering 渲染一个专门的整本打印视图, 结果缓存到 R2; 修书后自动失效重渲。
- **封面**: 竖版 1024×1536, 文字由模型直接画进图里, 不用任何本地叠字工具。

十、MCP: 44 个工具

第十五章讲过它是什么, 这里只把清单列全: 文章读写与版本、文风读写与蒸馏、挖矿与重写、社区读写与投币、算力余额与账单、分享链接与公众号与小红书、书架读书与写书修书、提示词分享与导入。登录是 6 位码加 4 位码的两步握手, 握手只活 2 分钟。

十一、账号

- **匿名优先**: 一装 App 就有一个匿名能力令牌 (存 iCloud 钥匙串), 不登录也能全功能用。
- **Apple 登录只是绑定**: 把 Apple ID 绑到已有的匿名盒子上, 绑完 scope 还是 `users/anon-<哈希>/` ——匿名和登录解析到同一个人。
- **设备配对**: 新设备用 6+4 握手登录老账号。
- **安卓微信登录**: 换 token 在 Cloudflare Pages Function 上。
- **账号删除**: 一次调用删干净——用户目录全部对象、本人社区帖及其举报标记、指向本人的短链、Apple 绑定。算力流水保留 (成孤儿, 无害)。

十二、地基

- **按商店区域定线路**: 入口主机由 App Store 商店区域运行时决定, 竞速探测已经拆掉——确定性比自适应值钱。
- **国内边缘**: `voicedrop.cn` 走腾讯 EdgeOne 边缘函数反代, 照片在国内边缘缓存。
- **存储收口到 D1**: 小 JSON 从 R2 迁进 `voicedrop-core` (10 张表), 双写期结束后 R2 副本停写停读、D1 成唯一真源; 大文件 (音频、照片、书页) 仍在 R2。
- **每晚备份**: R2 数据由 Worker 加定时任务增量镜像, 删除和覆盖的旧版本保 14 天。
- **实时状态**: `StatusHub` 每个用户一个实例, 挖矿每个阶段推一次, 列表的徽标从「待处理 → 听录音 → 挖文章 → 已成文」自己走, 不轮询。
- **意见反馈**: 写 R2 存档 + APNs 直推管理员, 同一用户 60 秒只推一条, 存档不节流。

- **报警**：实时通道断了看 `close` 的 `source` 字段区分是墙还是配额，APNs 节流报警。
- **测试即契约**：服务端 `agent/test/` 覆盖文章仓储、API 路由、工具、循环；iOS 侧 125 条纯逻辑单测，不打网络。向后兼容是靠测试钉住的。

十三、砍掉的

清单不列砍掉的东西就不诚实。第十二章讲过为什么，这里只记账：**SDUI 自定义首页**（Phase 1 写完了，从未进主干，存档在一个 tag 里）、**多风格并排对比**、**挖矿实时预览**、**线路竞速探测**、**文章后处理轮询器**（整套拆除：定时器、状态、环境变量、skill 全删）、**加急处理按钮**（路由还在，按钮没了）、**转发沿用原码**（等有真实传播量再说）。

这份清单会过时——写下它的那一天之后还会有新的提交。但它记录的不是功能有多少，而是一件事：每一行都曾经是一个具体的坑，或者一个具体的人在具体一天说「这里不对」。