Skill工程实战：从想法到第一个AI助手

树懒老K（拙一）


个人网站


个人微信


公众号

Skill工程实战：从想法到第一个AI助手

树懒老K（拙一）

作者：树懒老K（拙一）

出版：树懒老K

年份：2026

目 录

前言

第1章 你不是需要更多的AI工具，你需要一个自己的AI助手

1.1 工具收集者的困境

1.2 从"用工具"到"搭工具"

1.3 一个真实的故事

1.4 什么是Skill

1.5 这本书的Skill观

本章小结

第2章 Skill不是技术概念，是一种设计思维

2.1 用非技术语言定义Skill

2.2 Skill的五个要素

2.3 好的Skill vs 坏的Skill

2.4 你不是在编程，你是在设计

2.5 Skill自检清单

本章小结

第3章 什么样的工作适合做成Skill

3.1 可Skill化的四类工作

3.2 评估框架：频率×复杂度×确定性

3.3 三个岗位的Skill机会扫描

3.4 不适合做Skill的四个信号

3.5 你的第一个Skill选什么

本章小结

第4章 工作流拆解法

4.1 拆解之前：先做一次时间审计

4.2 五步拆解法

4.3 一个完整的拆解案例

4.4 常见错误

本章小结

第5章 Skill的五层设计框架

5.1 为什么要分层

5.2 第一层：输入层

5.3 第二层：工具层

5.4 第三层：执行层

5.5 第四层：输出层

5.6 第五层：反馈层

5.7 五层设计检查表

本章小结

第6章 提示词不是写出来的，是设计出来的

6.1 三个层次

6.2 从prompt到instruction的进化

6.3 结构化instruction模板

你是谁

你要做的任务

输入信息

工作流程

输出要求

关于反馈

6.4 案例：同一个任务的prompt v1到v5

6.5 好的instruction长什么样

本章小结

第7章 第一个Skill：「日报助手」

7.1 需求定义

7.2 第一步：写你的Skill说明书

你是谁

你要做的任务

输入信息

工作流程

输出要求

关于反馈

7.3 第二步：首次使用

7.4 第三步：检查和调整

7.5 第四步：迭代（第二周）

7.6 一张路线图

本章小结

第8章 让Skill能干"真活"：工具与MCP

8.1 为什么只有对话能力不够

8.2 工具的三个层次

8.3 MCP协议是什么（最短的篇幅讲清楚）

8.4 接入工具的三种方式

8.5 安全与权限：什么能让AI干，什么不能

8.6 什么时候该加工具，什么时候不该

本章小结

第9章 让Skill"记住"你：记忆与上下文

9.1 没有记忆的Skill，每次都是新同事

9.2 三层记忆体系

9.3 怎么建立Skill的长期记忆

固定信息

9.4 知识库的设计原则

9.5 Skill学习曲线的设计

本章小结

第10章 测试与迭代：Skill不是一次写好的

10.1 怎么验证一个Skill"好用"

10.2 常见的6种翻车模式

10.3 迭代节奏

10.4 什么时候该改，什么时候该重写

10.5 从v1到v10：一个Skill的进化日志

本章小结

第11章 多Skill协作：搭一个"助手矩阵"

11.1 什么时候需要多个Skill

11.2 Skill之间的配合模式

11.3 案例：一个项目经理的Skill矩阵

11.4 避免"Skill爆炸"

第11章小结

第12章 把Skill分享给别人

12.1 从个人Skill到团队Skill

12.2 Skill怎么封装

配置项（使用前请修改）

输出格式选择

如何使用这个Skill

12.3 Skill文档怎么写

12.4 分享的三种方式

12.5 权限设计：使用、修改、分发

本章小结

第13章 企业级Skill的特殊挑战

13.1 个人Skill和企业级Skill的区别

13.2 数据安全与合规

13.3 多用户场景下的权限治理

13.4 Skill的生命周期管理

13.5 四个常见的企业级陷阱

本章小结

附录

附录A：Skill模板库

固定信息

任务描述

输入信息
工作流程
输出格式
质量标准
固定信息
任务描述
输入信息
工作流程
输出格式
注意事项
附录B：翻车记录30条
附录C：工具链速查

前言

> *你不是需要更多的AI工具，你需要一个自己的AI助手。*

过去两年，我见了很多想用AI但不知道从哪开始的人。

他们分两种。

第一种：**在等”那个对的工具”出现。**

他们关注了20个AI公众号，试了30个工具，每个用了一两次就放下了。不是工具不好——是他们始终在用别人设计好的工具，做别人预设好的事。工具的通用性越强，离自己的具体场景就越远。他们一直在找，一直找不到。

第二种：**觉得自己”不会编程”所以做不了。**

他们觉得”搭AI助手”是程序员的事。自己虽然能看出来工作中哪些环节可以用AI优化，但没有能力把它实现。他们一直在等，一直等不到。

这本书是写给第二种人的——也是写给第一种人里那些愿意从”找工具”切换到”搭工具”的人。

你不需要会写代码。

你不需要懂AI原理。

你只需要一件事：**了解你自己每天在做的事。**

这本书的核心方法——我叫它”Skill工程”——不是一项编程技术，是一种设计思维。

它是把你每天重复的、费时的、有规律的工作拆解出来，用AI重新设计一个”关于这件事的助手”。

你可以叫它Skill，也可以叫它助手、Agent、自动化流程——叫什么不重要，重要的是：**它是你亲手为自己搭的，不是别人做好了卖给所有人的。**

这本书会带你走完完整的路径：

认知篇（第1–3章） — 帮你判断”什么值得做成Skill”，建立Skill思维

方法篇（第4–6章） — 教你Skill工程的核心方法论：拆解、设计、提示词

实战篇（第7–10章） — 从0到1走完一个完整案例，再扩展到多种场景

进阶篇（第11–13章） — 从一个人用到一个团队用，解决真实世界的复杂问题

每一章都可以停下来做一件事。书里的案例来自真实的工作场景——运营、分析、销售、项目管理——这些案例里的”主人公”也都不懂编程。

如果他们在自己的岗位上能搭出好用的AI助手，你也可以。

那么，开始吧。

—— 树懒老K（拙一），2026年春

第1章 你不是需要更多的AI工具，你需要一个自己的AI助手

> *你收集的每一个工具，都在替你解决别人定义的问题。你需要的，是一个只解决你问题的东西。*

1.1 工具收集者的困境

你手机里装了多少AI相关的App？你的书签栏里收藏了多少AI工具？你关注了多少个”XX AI更新了”的公众号？

我不认识你，但我可以猜一个数字：**两位数起步，往三位数走。**

这不是你的问题。过去两年，AI工具以每周几十个的速度涌现。每一个都说”重新定义工作方式”——然后你用了一周，发现它要么不贴合你的场景，要么需要你改变自己的工作习惯去适应它。

这是一个深层困境：**通用工具永远离你的具体场景有距离。**

ChatGPT能帮你写邮件，但不知道你老板喜欢什么语气、你的客户在意什么细节。Copilot能帮你写代码，但不知道你的项目架构、你的团队规范。Notion AI能帮你整理笔记，但不知道你脑子里那团乱麻的最终目的地。

不是AI不够强，是**通用工具不可能知道你的具体工作**——因为你的具体工作，只有你自己知道。

1.2 从”用工具”到”搭工具”

有一个思维转变，卡住了绝大部分人。

这个思维转变是：**从”找一个工具解决我的问题”到”搭一个助手解决我的问题”。**

前者叫”工具使用者”。后者叫”工具设计者”。

工具使用者看到一个问题，想的是：”有没有一个工具能解决这个问题？”然后花大量时间搜索、试用、比较。最后要么找不到合适的，要么找到一个凑合用的。

工具设计者看到同一个问题，想的是：”这个问题的哪些部分可以交给AI来做？怎么做？”然后花少量时间设计，花更少时间搭建——一个属于自己的AI助手就诞生了。

区别在哪？

工具使用者把AI当”产品”来消费。工具设计者把AI当”能力”来组合。

前者永远在等别人做出更好的产品。后者自己就是那个”做产品的人”——只不过他的产品只服务一个人：他自己。

这本书的目标，就是帮从”使用者”变成”设计者”。

1.3 一个真实的故事

讲一个真实的事。

我认识一个运营经理，叫他C。C每天花两小时做一件事：整理前一天的项目进度。

他需要从五个不同的群里提取消息、从飞书文档里摘录更新、从Excel里抄数据——然后汇总成一份日报。

他试了很多工具。试过各种写作助手，但它们不知道他的项目叫什么、他的团队有哪些人、他的进度标准是什么。试过自动化工具，但配置太复杂，学了半天放弃了。

有一天他做了一个很小的改变。

他花了一个下午，梳理了自己的日报流程——不是写代码，只是在纸上画了画每天是怎么做的。然后他写了一段”给AI的说明书”（其实就是一套清晰的提示词），告诉它： – 日报需要包含哪几个部分 – 每个部分的信息来源在哪 – 每个部分的格式是什么 – 什么信息他需要自己确认，什么可以直接用

然后他每天只需要做三件事： 1. 把各个来源的信息丢给AI 2. AI按他给的格式生成日报初稿 3. 他花10分钟检查确认

两小时的活儿变成了20分钟。

他没有做一个”产品”，他只是给自己搭了一个**关于日报的助手**。这个助手只有他能用——因为只有他知道自己的流程长什么样。但恰恰因为只有他能用，它才真正好用。

C后来管这个东西叫”我的日报Skill”。

1.4 什么是Skill

你现在大概明白了：Skill不是某个具体的AI工具。它是**你用AI为你的某项工作设计的专属助手**。

用一个简单的公式来定义：

Skill = 你的一项工作 + AI的能力 + 你的设计

三项缺一不可： – 没有你的工作，Skill是无源之水（你不需要它） – 没有AI的能力，Skill就是一张流程图（它不会干活） – 没有你的设计，Skill就是另一个通用工具（它不懂你）

这本书要教你的，就是这个”你的设计”的部分。

1.5 这本书的Skill观

市面上有很多关于Agent、AI助手、自动化流程的讨论。它们各有各的定义，这本书不做学术争辩。

在这本书里，Skill就是**你为自己搭的一个小助手，帮你做一件具体的事**。

它不一定需要多复杂的架构。它不一定需要联网、不需要数据库、不需要多Agent协作。它可以简单到只是一套精心设计的提示词——只要它真正帮你省了时间、提了质量，它就是好Skill。

这本书不教你写框架，不教你部署服务，不教你调模型参数。它只教你一件事：**怎么把你自己每天做的工作，变成一个AI可以参与的系统。**

这个能力，比任何具体的AI工具都值钱——因为工具会过时，但这个思维不会。

本章小结

1. **通用工具永远离你的具体场景有距离** — 这是”工具收集者”永远解决不了的困境
2. **从”用工具”到”搭工具”** — 你不是在消费AI产品，你是在组合AI能力
3. **一个真实案例** — 运营经理C用一下午搭了一个日报Skill，每天省1小时40分钟
4. **Skill的定义** — 你的工作 + AI的能力 + 你的设计
5. **你的任务** — 写下你现在工作中最想改进的一件事（不用完美，一个就够），这就是你读这本书的起点

📚 延伸阅读

- 公众号文章《你不是需要更多AI工具，你需要一个自己的AI助手》— 树懒老K（本章浓缩版）
- 《深度工作》— 卡尔·纽波特。为什么"自己的助手"比"通用工具"更有价值？因为深度的前提是"贴着你的场景走"

第2章 Skill不是技术概念，是一种设计思维

> *会搭AI助手的人，不是程序员，是设计者。*

2.1 用非技术语言定义Skill

先忘掉所有技术术语。

Skill不是什么高大上的东西。它就是**一个”说明书 + 工具包”**——你给AI一份说明书，告诉它你的工作是什么、怎么干、用什么工具、产出什么结果、标准是什么。AI读了你的说明书，就能帮你干这件事。

如果你做过饭，这就很好理解：

- AI是灶台——它有火（算力）、有锅（模型能力）
- 你的说明书是菜谱——告诉它放什么料、什么顺序、什么火候
- 你提供的材料是食材——也就是你工作中的原始信息
- 最终出品的菜，就是Skill的产出

没有菜谱，再好的灶台也做不出一桌菜。没有你的说明书，再强的AI也不知道你想让它做什么。

Skill = 给AI的一份”关于你的某项工作的说明书”

就这么简单。

2.2 Skill的五个要素

一本好的菜谱需要包含食材清单、步骤、时间控制、调味技巧。一份好的”Skill说明书”也需要包含五个基本要素：

要素一：输入

你要告诉AI：干这个活需要知道什么。

- 信息来源：数据在哪？文档在哪？人在哪？
- 输入格式：是文本、表格、图片、还是对话记录？
- 输入范围：全部信息都给，还是只给关键部分？

要素二：工具

你要告诉AI：你有什么工具可以用。

- 能访问什么：搜索引擎、数据库、API、本地文件？
- 能操作什么：发邮件、写文档、更新系统、发通知？
- 不能碰什么：什么数据不能碰、什么操作不能做？

要素三：规则

你要告诉AI：这个活按什么标准干。

- 怎么判断对错：什么情况是”对的”，什么情况是”错的”？
- 如何处理例外：超出预设范围时怎么办？直接干还是问人？
- 优先级：什么重要、什么次要、什么可以跳过？

要素四：输出

你要告诉AI：干完之后要给我什么。

- 格式要求：文档、表格、报告、还是直接执行？
- 质量标准：什么算”做好了”？什么算”要返工”？
- 交付方式：放在哪？发给谁？通知谁？

要素五：反馈

你要告诉AI——也告诉你的自己——怎么让这个Skill越来越好。

- 满意度信号：你怎么判断它做得好不好？
- 纠错机制：做错了怎么让它修正？
- 迭代方向：用多了之后，怎么调整升级？

这就是Skill的五个要素。你不需要一次性把五个都写完美。能从两三个开始，用起来，再迭代。

2.3 好的Skill vs 坏的Skill

好的Skill： – 输入明确：AI清楚地知道该接收什么信息 – 边界清晰：AI知道自己能干什么、不能干什么 – 可验证：你一眼就能判断输出对不对 – 有纠错：做错了你能告诉它，它能记住

坏的Skill： – 什么都想做：说明书太长、目标太宽，AI无法聚焦 – 没有工具：只让AI”想”，不让AI”做”，效率有限 – 依赖模糊判断：”写得专业一点”“分析深入一些”——AI不知道什么是”专业” – 没有反馈机制：每次都是从头开始，没有进步

一个对比：

坏的需求：”帮我管好项目进度。”

好的需求——拆解成Skill： – 输入：每周各成员的进度更新 – 工具：项目看板（只读）、团队通讯录 – 规则：超过3天未更新的任务标黄；超过7天标红；标红的通知我 – 输出：每周进度报告——按项目维度，包括风险项列表 – 反馈：我每周一确认，告诉我哪些判断对了、哪些错了

同样一件事，”帮我管好项目进度”这不是一个Skill——这是一个愿望。拆解之后的那五条，才是一个Skill。

2.4 你不是在编程，你是在设计

很多人听到”给AI写说明书”以为是要写代码。不是。

你做的每一件事——定义输入、选择工具、制定规则、明确输出、建立反馈——都是**设计工作**，不是编程工作。

设计师和程序员的区别：

- 程序员关心：”怎么实现？”
- 设计师关心：”怎么才更好用？”

Skill工程本质上是设计工作。你在设计一个”你+AI”的协作方式——不是写代码，是设计协作流程。

所以，会不会编程不重要。重要的是你能不能清晰地描述自己的工作——拆解它、定义它、再重新组合它。这是一种思维能力，不是技术能力。

2.5 Skill自检清单

在你开始搭建第一个Skill之前，先用这个清单给自己打分。每一项你能清晰回答，就说明你已经具备设计Skill的能力了。

1. [ ] 我能清楚地描述我要解决的具体问题是什么
2. [ ] 我知道完成这件事需要什么信息
3. [ ] 我知道怎么判断”做对了”和”做错了”
4. [ ] 我知道我希望产出什么格式的结果
5. [ ] 我能接受AI做得不完美——允许迭代

如果你现在一条都勾不上——没关系。这本书接下来会帮你一项一项建立这个能力。

如果你现在已经能勾上两三条——很好，你已经可以开始动手了。

本章小结

1. **Skill不是技术概念** — 它就是”给AI的一份关于你的某项工作的说明书”
2. **五个要素** — 输入、工具、规则、输出、反馈
3. **好的Skill是聚焦的、可验证的、有反馈的** — 坏的Skill相反
4. **你不是在编程，你是在设计** — 设计”你+AI”的协作方式

5. **记住：** 你的第一个Skill不需要完美。从两三个要素开始，跑起来，再迭代

📚 延伸阅读

- 《设计的觉醒》— 原研哉。好的设计不是”更好看”，是”更好用”。Skill设计的本质就是让”人+AI”更好用。
- 公众号文章《Skill不是技术概念，是一种设计思维》— 树懒老K

第3章 什么样的工作适合做成Skill

> *不是所有工作都适合做成Skill。学会判断，比学会做更值钱。*

3.1 可Skill化的四类工作

不是所有工作都适合做成一个AI助手——有些工作太简单（不需要AI），有些太复杂（AI还做不了），有些太依赖人的判断（不应该交给AI）。

根据我的经验，适合做成Skill的工作有四类。

第一类：信息处理类

核心模式：从A处拿到信息，整理成B处的格式。

- 接收各种来源的信息（邮件、文档、群消息、报表）
- 按固定规则进行筛选、分类、汇总
- 输出为结构化的报告或摘要

典型场景：日报周报、竞品监测、会议纪要、数据汇总

可Skill化的判断标准：**信息来源多样但格式固定，处理规则明确。**

第二类：判断辅助类

核心模式：AI帮你列出选项和依据，你做最终判断。

- 面对一个需要判断的问题

- AI按你给定的框架做分析
- 你基于AI的分析做决定

典型场景：方案评估、风险分析、优先级排序、供应商对比

可Skill化的判断标准：**判断有固定维度，但结论需要人确认。**

第三类：流程执行类

核心模式：你告诉AI”做这件事”，AI按固定步骤执行到底。

- 有明确的操作步骤
- 每一步的输入输出清晰
- 异常情况可以预判

典型场景：数据录入、流程审批跟进、工单处理、定期报告生成

可Skill化的判断标准：**步骤明确，操作可重复，异常可预判。**

第四类：知识沉淀类

核心模式：AI记录和整理你产生的知识，让它们可以被复用。

- 你在工作中产生的经验、判断、决策
- AI帮你结构化存储
- 以后遇到类似情况时，AI自动关联历史经验

典型场景：客户信息管理、项目复盘、案例整理、经验库建设

可Skill化的判断标准：**你经常产生值得记录的内容，但缺乏时间和方法去整理。**

3.2 评估框架：频率×复杂度×确定性

现在你知道了四类适合做成Skill的工作。但同一个类别里，不同的工作可Skill化的程度也不一样。

这里有一个简单的评估框架，帮你在具体任务上做判断。

三个维度：

维度一：频率

这个任务你多久做一次？ – 每天/每周 → 3分 – 每月 → 2分 – 每季度或更少 → 1分

频率越高，做Skill的价值越大——省一次是省，省一百次是革命。

维度二：复杂度

这个任务的复杂程度如何？ – 低（3–5个步骤，规则清晰）→ 3分（最容易做Skill） – 中（5–10个步骤，需要一些判断）→ 2分 – 高（10个以上步骤，需要大量判断）→ 1分（先拆解再做）

复杂度适中最好。太简单你不需要AI，太复杂AI做不好。

维度三：确定性

这个任务的规则是否明确？ – 高度确定（能写清楚"如果A就B"）→ 3分 – 部分确定（大部分情况有规则，偶尔有例外）→ 2分 – 低度确定（每件事都不一样）→ 1分（不适合做Skill）

确定性决定AI能不能稳定产出。

总分判断：

- 7–9分：**优先做** — 高价值，高成功率
- 5–6分：**可以做** — 值得试一试，可能需要调整
- 3–4分：**暂缓** — 可能太简单或太复杂，换个任务试试

3.3 三个岗位的Skill机会扫描

岗位一：运营经理

她每天要做的事： – 整理项目进度 ✅ 频率3×复杂度3×确定性3 = 9分 → 优先做 – 回复常见客户问题 ✅ 频率3×复杂度2×确定性3 = 8分 → 优先做 – 竞品动态监测 ✅ 频率2×复杂度2×确定性2 = 6分 → 可以做 – 新方案设计 ❌ 频率1×复杂度1×确定性1 = 3分 → 暂缓

岗位二：销售代表

他每天要做的事： – 写客户跟进记录 ✅ 频率3×复杂度3×确定性3 = 9分 → 优先做 – 做客户背景调研 ✅ 频率2×复杂度2×确定性3 = 7分 → 优先做 – 准备报价方案 ✅ 频率2×复杂度2×确定

性2 = 6分 → 可以做 – 客户谈判策略 ❌ 频率1×复杂度1×确定性1 = 3分 → 暂缓

岗位三：数据分析师

她每天要做的事： – 出常规数据报表 ✅ 频率3×复杂度3×确定性3 = 9分 → 优先做 – 回答重复性的数据问题 ✅ 频率3×复杂度2×确定性3 = 8分 → 优先做 – 做专题分析 ❌ 频率1×复杂度1×确定性1 = 3分 → 暂缓 – 数据质量检查 ✅ 频率2×复杂度2×确定性2 = 6分 → 可以做

3.4 不适合做Skill的四个信号

有些工作看起来适合，实际不适合。注意这四个信号：

信号一：你描述不清楚这个工作

如果你花了一小时还写不清楚”AI需要做什么”——这个工作可能连你自己都没完全理解。先理清流程，再考虑做成Skill。

信号二：判断标准经常变

如果每次做这件事的标准都不一样——今天按A规则，明天按B规则——AI没法稳定输出。先把标准定了，再动手。

信号三：你不敢让AI犯错

如果这件事错一点都不行（比如发给客户的合同、上报给监管的数据）——那这件事没必要做成全自动Skill。可以考虑”AI辅助”模式：AI做初稿，你终审。

信号四：你一年只做一两次

每年一次的年度总结——不值得做成Skill。每次重写也花不了多少时间。把时间花在每天/每周都在做的事上，回报率最高。

3.5 你的第一个Skill选什么

根据上面的框架，我给你一个非常简单直接的建议：

你的第一个Skill，选一个你每周都要做、你觉得烦、但做起来有固定步骤的事。

不是最具战略价值的，不是最展示技术水平的——是你最想”甩掉”的。

因为你要的不是证明”我会搭AI助手”，你要的是**省下时间**。第一个Skill省下的每一分钟，都会让你更有动力去做第二个。

这就是”最小可行Skill”的思路——小到你觉得”这也能算个Skill？”——对，就它了。

本章小结

1. **四类可Skill化的工作** — 信息处理、判断辅助、流程执行、知识沉淀
2. **评估框架** — 频率×复杂度×确定性，7–9分优先做
3. **不适合的四个信号** — 描述不清、标准常变、不敢犯错、频次太低
4. **第一个Skill选什么** — 你最想”甩掉”的那个。不是最重要的，是让你最烦的
5. **你的任务** — 用评估框架给你的工作打分，选出分数最高的那个。它就是你要搭建的第一个Skill的候选。

📚 延伸阅读

- 《重来》— 贾森·弗里德。这本书的核心观点：”做少，但做精。”选Skill也是——先挑一个最小最有用的，做好它，再想下一个。
- 公众号文章《什么样的工作适合做成AI助手》— 树懒老K（本章的评估框架原版）

第4章 工作流拆解法

> *拆解不是”把工作分成步骤”，是”找到AI能插手的缝隙”。*

4.1 拆解之前：先做一次时间审计

在开始拆解之前，你需要先回答一个问题：你的时间到底花在哪了？

这不是一个你”以为”的问题——它是一个你需要”记录”的问题。

做一次时间审计：

花一周时间，记录你每天的工作内容。不用太精细，每半天记一次就行。记完之后，把所有工作按”价值×重复度”放进四个象限：

	高重复度	低重复度
高价值	象限A（AI甜蜜区） 日报、周报、数据汇总 定期报告、例行分析	象限C（需要专注） 方案设计、客户沟通 团队协调、问题诊断
低价值	象限B（直接自动化） 数据录入、格式整理 常规邮件回复	象限D（考虑是否要做） 某些审批流程 不必要的汇报

象限A（高价值+高重复度）——这就是你第一个Skill应该瞄准的战场。 **象限B**（低价值+高重复度）——用自动化工具或脚本直接解决，不一定需要做成Skill。 **象限C**（高价值+低重复度）——AI可以辅助，但不宜全自动，适合”判断辅助类”Skill。 **象限D**（低价值+低重复度）——不是AI的问题，是”这件事根本不该你做”。

大多数人每天在象限A上花30–40%的时间。这意味着你每周至少有12–16小时在做”应该让AI帮你做的事”。

先找到象限A里的任务，我们再往下走。

4.2 五步拆解法

找到目标任务之后，按五个步骤拆解它。

第一步：还原

把这件事从头到尾写下来。不是”写周报”，是： – 打开Excel – 从系统A导出上周数据 – 从系统B导出项目进度 – 对照模板填入数据 – 写三点分析 – 发给老板

写得越细越好。你没有遗漏的关键细节，就是AI做不到的细节。

第二步：标记

在每一个步骤旁边标记：这一步需要什么？ – 🧠 判断（需要人的经验和直觉） – 🤖 执行（有明确规则，可以交给AI） – 🔗 信息获取（需要从某个来源拿数据） – ✅ 确认（需要人看一眼确认）

第三步：替换

把标记为”🤖 执行”和”🔗 信息获取”的步骤，标注为”可由AI接管”。

重点：**不要追求一步到位全部替换。** 先从替换2–3个步骤开始。

第四步：设计

被替换的步骤，重新设计成”人+AI”的协作模式： – AI做的事：明确的输入→规则→输出 – 人做的事：确认、补充判断、处理异常 – 交接点：什么时候给人看、给人看什么格式

第五步：验证

搭出来之后，用两到三次真实任务来验证： – AI的产出质量是否可用？ – 交接点是否清晰？ – 异常情况怎么处理？

发现问题→调整→再验证。

4.3 一个完整的拆解案例

现在让我们跟着一个真实的案例走一遍这五个步骤。

任务： 运营主管小林每周一做的”竞品分析周报”

第一步：还原

小林把整个过程写了下来： 1. 打开6个竞品网站和公众号 → 逐个查看更新 2. 把重要更新复制到Excel → 按”产品/市场/技术”分类 3. 在Excel里标注”对我方的影响”（高/中/低） 4. 写一段本周重点观察 5. 排版、检查、发送给团队

第二步：标记

1. 打开竞品网站查看更新 → 🔗 信息获取 + 🧠 判断哪些重要
2. 复制到Excel分类 → 🤖 执行（分类标准是固定的）
3. 标注影响程度 → 🧠 判断
4. 写重点观察 → 🧠 判断 + 🤖 辅助
5. 排版检查发送 → 🤖 执行（格式固定）

第三步：替换

可替换为AI的部分： – 步骤1的信息采集 → AI自动抓取6个信息源的新内容 – 步骤2的分类整理 → AI按小林的历史分类标准自动归入 – 步骤5的排版发送 → AI生成报告初稿

需要人做的部分： – 判断哪些更新”重要”（保留） – 标注影响程度（保留） – 写重点观察（保留——用AI辅助）

第四步：设计

新流程： 1. 🔗 AI抓取：周一早上，AI自动抓取6个信息源过去一周的更新 → 小林花5分钟浏览确认无遗漏 2. 🤖 AI分类：AI按预设标准自动分类 → 小林花5分钟调整分类 3. 🧠 小林判断：小林阅读分类后的内容，标注影响程度 4. 🧠+🤖 小林写+AI辅助：小林写重点观察，AI提供数据支撑 5. 🤖 AI生成：AI按模板生成周报 → 小林10分钟确认修改

第五步：验证

小林先用这个方法试用了两周，发现： – 第一周：AI的信息抓取漏了一个信息源 → 调整配置 – 第二周：分类标准需要补两个分类 → 更新规则 – 第三周：稳定运行

时间从120分钟降到45分钟，而且小林把主要精力从”信息搬运”转到了”分析和判断”上。

4.4 常见错误

错误一：想一次让AI完成全部工作

最常见的错误。把整条链条交给AI，让AI”一步到位”——然后发现AI的产出没法直接用，于是放弃。

正确的做法：**AI只做”环节”而不是”链条”。** 你把一个工作拆成多个环节，AI只做其中几个环节，你做剩下的。

错误二：流程设计好了就不调整

第一版通常不是最好的。你的工作习惯在变，AI的能力也在变。每两到四周回头看一次：有没有新的环节可以交给AI？原来的交接点是不是最优的？

错误三：忽略了异常处理

你做了100次这个工作，可能99次是正常的，1次是异常。你习惯了自己处理那1次。但AI不知道那1次怎么处理——因为你没告诉它。

在设计阶段就想好：出现异常情况时，AI是停下来等你处理，还是按默认规则继续？这两种选择都有代价，你需要提前想清楚。

本章小结

1. **先做时间审计** — 找到你象限A里的任务，那是Skill的战场
2. **五步拆解法** — 还原→标记→替换→设计→验证
3. **不要一次全交给AI** — 只替换2–3个环节，逐步扩大
4. **流程是活的** — 每两到四周迭代一次
5. **提前想异常** — 异常不是”不会发生”，是”你忘了告诉AI怎么处理”
6. **你的任务** — 选一个象限A里的任务，用五步拆解法走一遍，把拆解结果写下来

📚 延伸阅读

- 《精益创业》— Eric Ries。五步拆解法的核心思想就是”Build-Measure-Learn”——做一个小版本，验证它，改进它。
- 《高效能人士的七个习惯》— 史蒂芬·柯维。”要事第一”：时间审计的本质就是把精力从”不重要的事”转移到”重要的事”。
- 公众号文章《工作流拆解：让AI帮你省时间的正确方式》— 树懒老K

第5章 Skill的五层设计框架

> *同一个需求，有的人搭的Skill好用，有的人搭的不好用。区别不在AI，在设计。*

5.1 为什么要分层

你有没有遇到过这种情况：你觉得自己已经写得很清楚了，但AI还是做不对？

问题往往不是AI不够聪明——是你的设计不够精细。

好的Skill设计，是逐层展开的。就像建筑设计师不会直接把一堆材料堆在一起——他先画结构图、再画管线图、再画装修图。每一层解决一层的问题。

Skill设计也一样。我把它分成五层，从底层到顶层依次是：输入层、工具层、执行层、输出层、反馈层。

每一层都有自己需要回答的问题。上一层的设计决定下一层的选择空间。

5.2 第一层：输入层

输入层回答的问题是：**AI需要知道什么？**

很多人在这个环节就出问题了——要么给得太少（AI信息不足），要么给得太多（AI不知道什么重要）。

给AI的信息分三类：

固定信息： 不会变的基本设定 – 你的名字、岗位、公司 – 你的团队结构、项目名称 – 常用的术语和缩写 – 你偏好的风格和标准

每次信息： 每次使用Skill时需要提供的内容 – 这次要处理的具体数据 – 这次的上下文和特殊情况 – 这次的优先级和偏好

参考信息： 需要时可以查阅的知识 – 历史记录和过往案例 – 标准操作流程文档 – 数据字典和定义

设计原则：

固定信息直接写进Skill的说明书里。每次信息由你在每次使用时提供。参考信息放在Skill可以访问的地方，有需要时告知AI去查阅。

一个反例：

“帮我整理一下这周的项目进度。”——这句话信息太少了。AI不知道你的项目有哪些、不知道进度标准是什么、不知道你关注什么。

一个正解：

“我是[项目名]的项目经理。这周各成员的进度更新如下：[粘贴]。我们的进度标准是：按计划=绿，延迟3天内=黄，延迟3天以上=红。请按这个格式整理一份进度报告：[格式模板]。重点关注延迟的任务和下周到期任务。”

5.3 第二层：工具层

工具层回答的问题是：**AI能用什么工具来实现?**

只让AI”想”不让AI”做”，效率非常有限。你需要给AI配备工具——就像给人配备办公设备一样。

常见的工具类型：

- 信息查询：搜索引擎、数据库查询、文档检索、API调用
- 信息处理：计算、翻译、格式化、转换
- 操作执行：发送邮件、更新系统、创建文档、发送通知
- 记忆存储：记录历史、记住偏好、积累知识

设计原则：

只给AI它需要的工具。不是工具越多越好——工具越多，AI选择错误的概率越大。

最小工具集原则： 先从一个工具开始，确认AI能正确使用后再增加。

特别提醒：权限边界

每个工具都要明确告诉AI： – 能做什么：这个工具可以操作的范围 – 不能做什么：这个工具的禁区 – 什么情况下要问你：遇到不确定的操作时停下来

5.4 第三层：执行层

执行层回答的问题是：**AI具体怎么做?**

这是大多数人理解的”提示词工程”的部分。但我建议你不要只把它理解为”写prompt”——它是在设计一套”AI的思考流程”。

三种执行模式：

模式一：直接执行

规则清晰、输入明确、输出固定。

适合场景：数据提取、格式转换、定期报告。 设计要点：给明确的规则和输出模板。

模式二：分步执行

多步骤任务，每一步依赖上一步的结果。

适合场景：复杂分析、多源信息汇总。 设计要点：把步骤拆开，每一步都说明输入和输出。

模式三：判断执行

AI需要根据条件判断走哪条路径。

适合场景：异常处理、分类决策、优先级排序。 设计要点：给判断标准+各路径的执行方案。

一个对比：

不分步："分析这份数据，告诉我结论。" → AI不知道从哪开始分析，不知道什么维度重要。

分步："第一步：列出数据中的Top 5异常值。第二步：对每个异常值找出可能的原因。第三步：将原因按影响程度排序。第四步：给出Top 3建议。" → AI知道每一步该做什么，产出质量可控。

5.5 第四层：输出层

输出层回答的问题是：**AI要交付什么?**

很多人只告诉AI"做什么"，没告诉AI"做成什么样"。这是Skill质量不稳定的主要原因。

输出设计的三个要素：

格式： 结构、模板、样式 – "按以下格式输出：问题–原因–建议" – "用表格呈现，第一列是客户名，第二列是跟进状态" – "先给摘要，再给详细分析"

标准： 什么算"好"，什么算"合格" – "每条建议不少于50字" – "每个判断必须给出依据" – "不确定的地方标注'需确认'"

交付方式： 放哪、给谁、什么时候 – “生成后放到共享文件夹” – “生成后直接在对话框中呈现” – “生成后通知我，由我决定谁接收”

特别提醒：输出可验证性

你要确保自己能快速判断AI的输出质量。如果AI的输出你需要花很长时间才能验证——说明输出层的设计有问题。

好的输出是：你看一眼就知道”这个可以”或”这个不行”。

5.6 第五层：反馈层

反馈层回答的问题是：**怎么让Skill越来越好?**

这是五层中最容易被忽略的——但也是让好Skill和普通Skill拉开差距的关键。

反馈的三种形式：

纠正： AI做错了，你告诉它怎么改 – “这个分类不对，应该是A类不是B类” – “这个判断过于保守，实际情况是……” – “这个格式我不喜欢，改成这样……”

确认： AI做对了，你告诉它继续 – “这个分析很到位” – “这个格式很好，以后都用这个” – “这个判断标准正确”

升级： 场景变了，需要更新Skill – “现在多了一个数据源，需要加入” – “报告模板更新了，新模板是……” – “团队的关注点变了，分析维度调整一下”

设计原则：设计反馈机制，而不是依赖反馈意愿

不要指望自己每次都会主动给反馈。把反馈机制嵌入到使用流程中： – 每次输出后加一个问题:”这个结果可用吗？第几点需要调整？” – 每周做一次回顾:”过去一周的Skill输出质量如何？” – 记下你每次手动调整AI输出的地方——那是下一次迭代的信号。

5.7 五层设计检查表

每次设计完一个Skill，用这个表检查：

- [ ] 输入层：我知道AI需要什么信息，信息分类清楚
- [ ] 工具层：AI有必要的工具，权限边界明确
- [ ] 执行层：我设计了执行步骤，不是一句笼统的指令
- [ ] 输出层：我知道AI交付什么，能快速验证质量
- [ ] 反馈层：我有办法让Skill越用越好

五个✅全勾上，你的Skill基本不会差。差一两个也没关系——先跑起来，再迭代补上。

本章小结

1. **五层设计框架** — 输入层→工具层→执行层→输出层→反馈层
2. **输入层**：分清楚固定信息、每次信息、参考信息
3. **工具层**：最小工具集原则，只给AI需要的工具
4. **执行层**：三种执行模式，不要一句话指令，要分步走
5. **输出层**：格式+标准+交付方式，让输出可验证
6. **反馈层**：设计反馈机制，让Skill能进化
7. **记住**：五个✅全勾上当然最好，但勾上三四个就可以动手了。迭代比完美重要。

📚 延伸阅读

- 《设计思维》— 蒂姆·布朗。五层设计框架本身就是一种设计思维——从需求出发，逐层展开。
- 《写作法宝》— 威廉·津瑟。好的输出层设计，本质上就是好的写作。简洁、清晰、有结构。
- 公众号文章《Skill的五层设计框架》— 树懒老K。

第6章 提示词不是写出来的，是设计出来的

> *好的提示词不是”写”出来的，是”设计”出来的。区别在于：前者靠灵感和修辞，后者靠结构和迭代。*

6.1 三个层次

大多数人对提示词的理解停留在第一层。

第一层：指令级

“帮我把这个翻译成英文。”

这是最简单的提示词——一句话告诉AI做什么。适合一次性、简单的任务。但对于日常使用的Skill来说，远远不够。指令级提示词的输出质量高度依赖AI的”临场发挥”，不稳定。

第二层：角色级

“你是一个资深项目经理，有10年经验，擅长做项目复盘。请根据以下信息……”

给AI一个角色，帮助它理解”以什么身份回答”。这比指令级好——输出质量更稳定，风格更一致。但对于需要复杂流程的Skill来说，还是不够。

第三层：系统级

这是Skill需要的层次。系统级的提示词是一套完整的”工作说明书”——不只是告诉AI”做什么”和”扮演谁”，而是告诉它：

- 你是谁（Skill的使用者）
- 你的上下文（工作背景、偏好、约束）
- 你的标准（什么算对、什么算好）
- 工作流程（先做什么、后做什么）
- 工具接口（能用什么、不能做什么）
- 输出规范（格式、标准、交付方式）
- 异常处理（什么情况怎么办）

系统级提示词不是”写”出来的——它是**设计**出来的。

6.2 从prompt到instruction的进化

我建议你把自己的Skill提示词叫”instruction”而不是”prompt”。

为什么？

“Prompt”暗示这是一次性的、即兴的——你给AI一个提示，它给你一个回应。“Instruction”暗示这是持续的、系统的——你给AI一套操作说明，它按说明执行。

这个区别决定了你的态度：你是每次随便写一句，还是认真设计一套可复用的系统。

一个Skill的instruction通常包含五到八个模块：

```
模块一：你是谁（Skill的使用者描述）
       — 让AI知道它在为谁工作

模块二：你要做什么（Skill任务描述）
       — 一句话说清楚这个Skill是干什么的

模块三：上下文和背景（环境设定）
       — 项目信息、团队信息、行业信息、术语定义

模块四：工作流程（执行步骤）
       — 按顺序列出AI该怎么做

模块五：输出规范（交付标准）
       — 格式、内容要求、质量标准

模块六：工具和权限（可选）
       — AI能用什么工具，不能用什么

模块七：异常处理（可选）
       — 遇到不确定的情况怎么办

模块八：反馈机制（可选）
       — 怎么让AI知道你满意或不满意
```

你不需要每个Skill都包含全部八个模块。根据任务复杂度，选四到六个就够了。

6.3 结构化instruction模板

下面是一个通用的Skill instruction模板。你可以把它当作起点，根据你的任务调整。

```
<span style="font-size:0.5pt;">§TOC6-4§</span>
## 你是谁
我是[你的岗位]，负责[你的工作范围]。
我的项目/团队：[一句话背景描述]

<span style="font-size:0.5pt;">§TOC6-5§</span>
## 你要做的任务
帮我做[任务名称]。
这个任务我需要[频率]完成一次。

<span style="font-size:0.5pt;">§TOC6-6§</span>
## 输入信息
我需要你给我以下信息，我才能开始：
1. [信息A] — [说明]
2. [信息B] — [说明]

<span style="font-size:0.5pt;">§TOC6-7§</span>
## 工作流程
请按以下步骤执行：
1. 第一步：[做什么]
   输入：[需要什么信息]
   产出：[输出什么]
   标准：[怎么算做好]

2. 第二步：[做什么]
   ……

<span style="font-size:0.5pt;">§TOC6-8§</span>
## 输出要求
请按以下格式输出：
[格式描述或模板]

质量标准：
- [质量标准1]
```

```
- [质量标准2]
- 不确定的地方请标注"需确认"

<span style="font-size:0.5pt;">§TOC6-9§</span>
## 关于反馈
每次输出后，我会告诉你这个结果是否可用。
如果我调整了你的输出，请记住这个调整，下次按调整后的方式做。
```

这个模板不需要一次性写完美。先用起来，每次用的时候觉得”这里应该加一句”“那里应该改一下”——那就改。一个月后，你的instruction会比你第一版好十倍。

6.4 案例：同一个任务的prompt v1到v5

现在让我们看一个真实案例——一个”客户跟进记录助手”的instruction是如何演化的。

v1（指令级）

“帮我写客户跟进记录。”

效果：AI写了一篇很通用但没用的话——不知道客户是谁、不知道什么信息重要。

v2（角色级）

“你是一个销售助理。帮我整理客户跟进的记录。”

效果：语气更专业了，但还是不知道具体信息。

v3（增加了输入和流程）

“你是一个销售助理。以下是今天我跟客户沟通的内容：[粘贴对话记录]。请帮我整理成跟进记录：1. 提取关键信息；2. 列出待办事项；3. 标注需要升级处理的问题。”

效果：好多了。AI能提取出对话里的关键信息。但格式不统一，有时候长有时候短。

v4（增加了输出规范）

“你是一个销售助理。以下是今天我跟客户沟通的内容：[粘贴对话记录]。请按以下格式整理跟进记录——客户名、沟通日期、沟通方式、关键信息（3–5点）、待办事项（按Priority排序）、需要升级的问题（如有）。每条待办必须有负责人和截止日期。不确定的信息标注’待确认’。”

效果：稳定了。每次输出格式一致，该有的信息都有了。用了两周，发现几个问题：AI面对复杂对话有时候会遗漏重要信息，而且不会记住之前跟这个客户的沟通历史。

v5（增加了记忆和工具）

“你是一个销售助理。现在是[日期]。以下是今天我跟[客户名]沟通的内容：[粘贴对话记录]。请先查阅这个客户的历史跟进记录（如果有），然后按格式整理。格式——客户名、沟通日期、沟通方式、本轮重点（3–5点）、对比上次的变化（如有）、待办事项（按Priority排序，每条必须有负责人+截止日期）、需要升级的问

题（如有）。注意事项：1. 如果客户提到竞争对手，请标红。2. 如果有客户明确表示不满，请在开头用⚠️标注。3. 不确定的信息标注'待确认'。"

效果：这就是一个合格的Skill了。v5比v1长了10倍，但输出质量稳定、可预期，而且越用越好。

这一个案例告诉你的不是"要写长prompt"——而是"要迭代prompt"。

v1到v5的每一步，都是因为你发现了问题才改的。如果你一开始就想要一个完美的v5，你永远写不出来。因为你不知道你的具体场景会让AI在哪犯错。

先用起来，犯错，改进。这是唯一的路径。

6.5 好的instruction长什么样

作为本章的总结，记住这几个特征：

好instruction的特征：

1. **具体，不抽象** — 不说"专业一点"，说"每条建议不少于50字，必须有数据支撑"
2. **分步，不笼统** — 不说"分析一下"，说"第一提取异常，第二分析原因，第三给出建议"
3. **有标准，没废话** — 每一条指令都有"怎么判断做对了"的标准

4. **可迭代，不追求完美** — v1能用就行，v2在v1的基础上改，不是重写

这四点做到了，你的instruction就是好的。不管它是一页纸还是三页纸。

本章小结

1. **提示词的三个层次** — 指令级→角色级→系统级。Skill需要系统级
2. **从prompt到instruction** — 从”一次性提示”到”可复用说明书”
3. **八个模块** — 不需要全用，但结构化的框架比灵光一现更可靠
4. **迭代才是王道** — v1能用就行，在真实使用中改进
5. **好instruction的四个特征** — 具体、分步、有标准、可迭代
6. **你的任务** — 选第3章评估出的候选任务，按本章的模板写一个v1的instruction

📚 延伸阅读

- 《风格感觉》— 史蒂芬·平克。好的写作风格来自清晰的思路——不是修辞，是结构。提示词设计同理。
- 《Reprompting》— 一篇关于提示词迭代的研究论文。核心结论：最好的提示词不是写出来的，是迭代出来的。
- 公众号文章《提示词不是写出来的，是设计出来的》— 树懒老K

第7章 第一个Skill：「日报助手」

> *这是你亲手搭建的第一个AI助手。它不会很复杂*
>
> —— 但会很实用。

这是全书最重要的一章。我们不谈理论了，直接动手。

你将会花大概一个下午的时间，搭建一个能帮你省时间的日报Skill。它不完美——但能用。

7.1 需求定义

我们的目标：搭一个能做日报整理的Skill。

为什么选日报？ – 频率高（每天/每周），省时间效果明显 – 流程固定，适合第一次练手 – 做完了你立刻能感受到效果

Skill要做的事： 接收你一天的工作碎片信息（会议记录、工作笔记、群消息等），按你指定的格式整理成一份日报。

不需要做的事： – 不需要自动从各处收集信息（你手动给AI） – 不需要帮你判断什么重要（你告诉AI什么重要） – 不需要跟其他系统对接（只用AI的对话能力）

记住：第一个Skill追求的是”能用”，不是”完美”。

7.2 第一步：写你的Skill说明书

用第6章的结构化模板：

```
<span style="font-size:0.5pt;">§TOC7-3§</span>
## 你是谁
我是[你的名字]，[你的岗位]。

<span style="font-size:0.5pt;">§TOC7-4§</span>
## 你要做的任务
帮我整理每日工作日报。
这个任务我每天需要完成一次。

<span style="font-size:0.5pt;">§TOC7-5§</span>
## 输入信息
我需要给你以下信息：
1. 今天的日期
2. 今天的主要工作事项（简要列出）
3. 今天的重要会议记录（如有）
4. 今天的关键进展和问题（如有）
输入格式不限——可以是条目式的笔记，也可以是零散的对话记录。

<span style="font-size:0.5pt;">§TOC7-6§</span>
## 工作流程
请按以下步骤执行：
1. 浏览我提供的所有信息，识别出今天的关键事项
2. 按"今日完成工作/今日关键进展/遇到的问题/明日计划"四个模块归类
3. 检查归类是否有遗漏或错误
4. 按输出要求整理成日报

<span style="font-size:0.5pt;">§TOC7-7§</span>
## 输出要求
请按以下格式输出：

【日报】[日期]

一、今日完成工作
- [事项1]：[简要说明]
```

```
- [事项2]：[简要说明]

二、今日关键进展
- [进展1]：[说明，如有数据支撑更好]
- [进展2]：[说明]

三、遇到的问题
- [问题1]：[说明] | 状态：[已解决/处理中/待定]
- [问题2]：[说明] | 状态：[已解决/处理中/待定]

四、明日计划
- [计划1]
- [计划2]

质量标准：
- 每一个事项不超过两行
- 问题必须有状态标注
- 不确定的信息标注"需确认"
- 如果输入信息太少，请告诉我"信息不足以生成完整日报，需要补充什么"

<span style="font-size:0.5pt;">§TOC7-8§</span>
## 关于反馈
每次输出后，我会告诉你是否需要调整。
如果你记住了我的偏好（比如某些分类方式、某些固定的信息项），下次输出时
```

把这份说明书保存好。这是你第一个Skill的核心文件。

7.3 第二步：首次使用

找一个工作日，在下班前，花5分钟收集你今天的工作信息，然后交给AI。

你提供给AI的信息示例：

```
日期：2026年5月18日
今天的工作记录：
1. 上午开了项目周会——讨论了A项目的进度，B模块延迟3天，原因是第三方接
2. 下午处理了客户反馈——客户说产品手册的第三章不清楚，需要重写
3. 跟设计团队沟通了新版首页的修改方案，预计本周五出设计稿
4. 明天下午2点有跨部门会议，准备材料
```

AI会产出什么：

```
【日报】2026年5月18日

一、今日完成工作
- 参加项目周会，跟进A项目进度
- 处理客户产品手册反馈
- 沟通新版首页设计方案

二、今日关键进展
- A项目B模块延迟3天（第三方接口未完成测试）
- 客户手册反馈已接收，待重写第三章

三、遇到的问题
- 第三方接口测试未完成，B模块交付延迟 | 状态：处理中
- 产品手册第三章需重写，需要协调文档资源 | 状态：待定

四、明日计划
- 准备跨部门会议材料
- 跟进第三方接口测试进度
- 启动手册第三章重写

---

请确认这份日报是否可用？如果某个部分需要调整，请告诉我。
```

7.4 第三步：检查和调整

看完AI的输出，你可能会有一些调整：

- “B模块延迟3天这个说得太重了，应该是’预计延迟3天’——注意措辞”

- “明日计划加一条：跟文档团队确认重写排期”
- “整体格式没问题，以后都用这个”

把这些反馈告诉AI。它会在下一次输出时记住你的偏好。

这就是反馈层的第一次应用。**你不用等到Skill”完美”才开始用——用了才能发现问题，发现问题才能改进。**

7.5 第四步：迭代（第二周）

用了一周之后，你可能会发现新的需求：

- “我每天的输入格式不太统一，有时候写得乱——能不能我先随便写，你帮我梳理清楚再生成日报？”
- “周一和周二的日报模板不一样——周一需要写上周总结，周二不用”
- “有些信息我需要你自动从其他地方获取”

把新需求加进你的Skill说明书里，这就是v2。

v2可能长这样（只改了一处）：

在”工作流程”的第一步后面加一行：

```
1.5 如果输入信息比较零散或顺序混乱，请先按时间顺序或逻辑关系重新组织
```

Skill就是这样一点一点变好的。没有魔法，只有迭代。

7.6 一张路线图

你的第一个Skill完成之后，接下来的路线图：

第一个月：稳定使用 – 每天用，每天给反馈 – 定期检查AI的输出质量 – 积累3–5次修改后，把共性问题补进说明书

第二个月：增加复杂度 – 增加对多源信息的处理能力 – 增加对固定信息（项目名、团队成员等）的记忆 – 开始尝试”判断辅助”模式——让AI帮你做第一轮分析

第三个月：分享 – 你的日报Skill已经稳定运行了两个月 – 把它分享给团队里的同事 – 开始想第二个Skill

如果你走完了这三个月的路线图，你已经不是一个”会用AI的人”了——你是一个”会设计AI助手的人”。

这是这本书最重要的跨越。

本章小结

1. **第一个Skill选日报** — 频率高、流程固定、效果立竿见影
2. **四步走** — 写说明书→首次使用→检查调整→迭代
3. **不要追求完美** — v1能用就行，在真实使用中改进
4. **三个月路线图** — 稳定使用→增加复杂度→分享
5. **最重要的跨越** — 从”会用AI”到”会设计AI助手”

在你开始下一个Skill之前，先把这个日报Skill稳定使用两周。这两周的经历，比读完整本书都值钱。

📚 延伸阅读

- 《原子习惯》— 詹姆斯·克利尔。每天进步1%，90天后就是2.5倍。Skill的迭代就是这个道理。
- 公众号文章《我的第一个AI助手：日报Skill》— 树懒老K（包含可复制的完整Skill模板）

第8章 让Skill能干"真活"：工具与MCP

> *只会说话的AI助手，只能帮你写东西。能动手的AI助手，才能真正帮你省时间。*

8.1 为什么只有对话能力不够

到目前为止，你的日报Skill只能做一件事：你给它信息，它整理成日报。这已经很好了，但它有一个局限——**它不能主动去拿信息，只能等你给它。**

如果它能自己去项目管理系统里查进度、自己去邮件里找更新、自己把日报发到团队群里——那才是真正省时间的Skill。

这一章就是教你怎么让Skill从”只会说”变成”还能做”。

不过我先说清楚：**你的第一个Skill不需要工具。** 只用对话能力也能做出很好用的助手。等你觉得”如果它能自动做XX就好了”的时候，再回来看这一章。

8.2 工具的三个层次

给Skill配备工具，按难度和效果分为三个层次。

第一层：知识工具（最简单，推荐先做）

Skill可以访问你提供的参考信息，而不是只靠你每次输入。

- 你的常用模板库
- 你的项目字典和术语表
- 历史案例和过往报告
- 操作手册和流程文档

怎么做：把这些资料整理成文档，告诉Skill”你需要的时候可以查阅这个文件”。

效果：Skill的准确性和一致性明显提升。

第二层：信息工具（中等难度）

Skill可以去外部获取最新的信息，而不是只靠你给的。

- 搜索网络上的最新信息
- 查询数据库里的数据
- 读取某个文档或网页的内容
- 接入API获取实时信息

怎么做：通过MCP协议或类似的接口，让Skill能调用外部数据源。

效果：Skill从”被动接收”变成”主动获取”。

第三层：行动工具（较复杂，进阶）

Skill可以做操作、改数据、触发流程。

- 创建文档、编辑表格
- 发送消息、更新状态
- 触发通知、启动流程

- 执行批准或转发

怎么做：通过MCP协议的工具接口，授权Skill执行具体操作。

效果：Skill从”建议者”变成”执行者”。

8.3 MCP协议是什么（最短的篇幅讲清楚）

MCP（Model Context Protocol）是一个开放协议——你可以把它理解为”AI工具的USB接口”。

USB接口的意义是什么？不管你插的是键盘、鼠标、还是U盘，接口标准是一样的，插上就能用。

MCP做的是同样的事：它定义了一套标准，让AI能够”插上”各种外部工具——数据库、API、搜索引擎、办公软件。

对你来说，关键信息只有三条：

1. MCP是一个开放标准，不是某个公司的私有协议。越来越多的AI工具和平台支持它。
2. 有了MCP，你不需要为每一个AI工具写不同的”连接代码”——写一次MCP接口，所有支持MCP的AI都能用。
3. 你不需要深入理解MCP的技术细节。你需要知道的是：你的Skill可以通过MCP去”做”事情，不只是”说”事情。

你什么时候需要关心MCP： – 当你的Skill需要从外部获取实时信息时 – 当你的Skill需要操作其他系统时 – 当你想让你的Skill”自动完成”而不是”建议你去做”时

你现在不需要关心MCP： – 如果你的Skill只靠你给的信息就能工作 – 如果你的Skill的目标是帮你整理和分析信息，不是替你操作

8.4 接入工具的三种方式

方式一：知识库接入（推荐初学者）

把常用的参考信息整理成文档，作为Skill的”参考资料”。

具体做法： 1. 把所有参考信息放在一个固定的位置（比如一个文件夹） 2. 在Skill说明书的工具模块里告诉AI：你的参考资料在[路径]，需要时可以查阅 3. 当Skill觉得需要参考信息时，它会自动去读取

适用场景：模板、术语表、标准操作流程、历史案例。

方式二：搜索接入（进阶）

允许Skill在需要时搜索网络或内部知识库获取最新信息。

具体做法： 1. 确认你的AI平台支持搜索功能 2. 在Skill说明书里授权：需要最新信息时可以先搜索再回答 3. 设定搜索范围：全网搜索还是仅限指定来源

适用场景：竞品动态、市场信息、最新的数据和政策。

方式三：API接入（高阶）

通过API接口，让Skill直接操作系统里的数据。

具体做法： 1. 确定要接入的系统（项目管理工具、CRM、邮件系统等） 2. 通过MCP或平台提供的接口建立连接 3. 设置权限——Skill只能读还是可以写 4. 在Skill说明书里定义清楚：什么操作可以做，什么操作需要你确认

适用场景：自动更新项目状态、自动创建任务、自动发送消息。

8.5 安全与权限：什么能让AI干，什么不能

让Skill动手做事之前，必须先想清楚一个更重要的问题：**什么能让AI干，什么不能？**

绝对不能碰的禁区： – 涉及公司核心机密的敏感数据 – 需要法律效力的操作（签合同、付款、审批） – 涉及用户隐私的个人信息 – 一旦出错后果严重的关键操作

建议谨慎对待的区域： – 可以读取但不能修改的数据 – 修改前需要你确认的操作 – 有撤销机制的操作（可以先试）

基本原则：

先读后写。 先让Skill只读数据，确认它理解正确后，再考虑开放写权限。

最小权限。 只给Skill完成工作所需的最少权限。不需要写，就不给写权限。

人要在环里。 特别是刚开始的阶段，任何"写"操作都先经过你确认。

8.6 什么时候该加工具，什么时候不该

该加的信号： – 你经常需要手动查同样的信息（频率高） – 这些信息有固定来源可以自动获取（可自动化） – 信息获取占用了你大量时间（时间成本高）

不该加的信号： – 你的Skill只用对话能力已经很好用了（没必要） – 信息获取本身只需要几秒钟（不值得做） – 接入工具的成本大于它省下的时间（ROI为负）

记住：工具是锦上添花，不是必要条件。 这本书里的大部分Skill，只用对话能力就已经能省下大量时间了。

本章小结

1. **工具的三个层次** — 知识工具→信息工具→行动工具

2. **MCP是"AI工具的USB接口"** — 你不需要深入技术细节，只需要知道它能做什么
3. **接入工具的三种方式** — 知识库（推荐初学者）、搜索（进阶）、API（高阶）
4. **安全第一** — 先读后写、最小权限、人要在环里
5. **不是所有Skill都需要工具** — 先用好对话能力，有刚需再加
6. **你的任务** — 在你目前的Skill（日报助手）上，尝试加入一个知识工具：把你的常用模板和术语表整理成文档，告诉AI可以查阅

📚 延伸阅读

- MCP协议官网 — modelcontextprotocol.io
- 《黑客与画家》— 保罗·格雷厄姆。"做好一件事的最好方法是把它做出来，但做出来之前先想清楚安全问题。"

第9章 让Skill"记住"你：记忆与上下文

> *最好的助手不需要你每次都交代一遍自己是谁。它记得。*

9.1 没有记忆的Skill，每次都是新同事

你有没有这个经历：你新入职一家公司，第一天同事跟你说一遍流程，第二天你又问了一遍，第三天又忘了？

Skill没有记忆的时候，就是这种感觉。每次你用同样的Skill，它都不记得上次做过什么、不记得你的偏好、不记得你纠正过什么。

这很浪费——因为你每次都要把同样的信息重复一遍。

“我叫小王，我是运营部的……这个项目叫飞翔……我们的进度标准是绿黄红……”

如果有记忆的Skill，你说一遍它就记住了。下次直接用。

这一章就是教你怎么让Skill拥有记忆——不是让它更聪明，是让它更懂你。

9.2 三层记忆体系

Skill的记忆需要三个层次。每一层解决不同的问题。

第一层：短期记忆

短期记忆让Skill”记得这次对话里发生了什么”。

- 你刚才说了什么（AI能引用你之前的内容）
- 你在这次对话中调整了什么
- 你这次提供的特殊信息

大部分AI工具本身就带短期记忆——你在这轮对话里说的，它都能引用。所以短期记忆不需要你特意设计，它自然就有。

第二层：长期记忆

长期记忆让Skill”记得你这个人”。

- 你叫什么名字、做什么工作
- 你的团队、项目、常用工具
- 你的偏好（格式、语言风格、详略程度）
- 你之前纠正过什么

长期记忆需要设计。它的主要载体就是你的Skill说明书——你放在”固定信息”里的内容，就是AI对你的长期记忆。

第三层：知识库

知识库让Skill”记得你所在的世界”。

- 你的项目文档和资料
- 你的历史案例和报告
- 你的行业信息和术语
- 你的标准操作流程

知识库和长期记忆的区别：长期记忆是关于”你这个人”的，知识库是关于”你的工作环境”的。

9.3 怎么建立Skill的长期记忆

方法一：写在说明书里（最可靠）

把你希望Skill记住的信息，直接写进Skill说明书的第一部分”固定信息”。

每次使用Skill时，AI都会读到这份说明书——所以它”记住”了。

比如说，你的日报Skill可以加上：

```
<span style="font-size:0.5pt;">§TOC9-4§</span>
## 固定信息
我的名字：小王
我的岗位：运营主管
我的项目：飞翔项目（项目周期2026.01-2026.12）
我的团队：5人（运营2人、设计1人、开发2人）
我的偏好：日报格式用"完成/进展/问题/计划"四个模块
我的关注重点：项目进度风险、团队问题、外部依赖
```

这些信息你只需要写一次。每次用Skill的时候，AI自动读到，不需要你重复。

这就是Skill记忆最简单、最可靠的方式。没有技术含量，但极其有效。

方法二：靠反馈积累（需要时间）

你每次使用Skill时给的反馈——“这里改一下”“这个格式好”“这个判断不对”——AI应该能记住。

实现方式是：在反馈模块里明确告诉AI”记住我的调整，下次沿用”。

效果取决于AI平台对对话上下文的支持程度。对于大多数主流AI工具，在一个持续使用场景中，它们确实能记住你之前的偏好。

方法三：建立知识库文件（适合大量信息）

如果你的Skill需要记忆大量信息（比如项目文档、产品手册、历史报告），写在说明书里就不太现实了——说明书会变得太长。

这时候，把参考信息整理成单独的文件，告诉AI”需要时查阅X文件”。

9.4 知识库的设计原则

知识库是Skill记忆中最有价值也最容易失控的部分。

做知识库的三个原则：

原则一：先有需求，再建知识库

不要提前建一个庞大的知识库。等你发现Skill”如果知道XXX就好了”的时候，再把XXX加进去。

知识库是长出来的，不是建出来的。

原则二：一件事一个文件

不要把所有的信息塞进一个文件。每类信息一个文件——模板一个文件、项目文档一个文件、术语表一个文件。

这样做的好处：你需要AI查阅A信息时，只让它看A文件，不用在巨大文件里搜索。

原则三：定期更新

知识库的最大敌人是过期。如果你三年前的流程还放在Skill的知识库里，AI会拿过时的信息给你做判断。

每季度检查一次知识库里的信息是否仍然有效。过期的要么删除，要么标注”已过期仅供参考”。

9.5 Skill学习曲线的设计

一个有记忆的Skill，使用时间越长越好用。这就是Skill的”学习曲线”。

前两周：学习期

- AI在适应你的工作方式
- 你每次使用后主动给反馈
- 输出质量70–80分（可用但需要调整）

第一个月：稳定期

- AI已经熟悉了你的偏好和常用场景

- 你给的反馈越来越少
- 输出质量85–90分（大多数情况不需要调整）

第三个月：信任期

- AI了解你的深层工作习惯
- 能主动提出你可能忽略的事项
- 输出质量90分以上（基本不需要修改）

如果到了第三个月Skill还是不好用，通常是因为：

- 你的说明书写得不够清楚（回头检查五层设计框架）
- 你没有坚持给反馈（反馈层是学习曲线的核心）
- 这个任务本身不适合做成Skill（回到第3章的评估框架）

本章小结

1. **没有记忆的Skill，每次都是新同事** — 你每次都重复同样的话，不值得
2. **三层记忆体系** — 短期记忆（对话内）、长期记忆（关于你）、知识库（关于你的工作）
3. **长期记忆最可靠的方式** — 写在Skill说明书里，不要依赖AI自己记住
4. **知识库** — 先有需求再建，一件事一个文件，定期更新
5. **学习曲线** — 两周学习期→一个月稳定期→三个月信任期。前提是你要坚持使用和反馈

6. **你的任务** — 回顾你第一版的日报Skill说明书，加上”固定信息”模块，写下至少5条你希望AI记住的关于你的信息

📚 延伸阅读

- 《如何打造一个记忆系统》— David Allen。GTD方法中的”参考资料”部分，跟Skill知识库的设计思路完全一致。
- 公众号文章《Skill越用越好用的秘密：三层记忆体系》— 树懒老K

第10章 测试与迭代：Skill不是一次写好的

> *没有产品是v1就完美的。Skill也一样。区别在于：好Skill是改出来的，坏Skill是"放"出来的。*

10.1 怎么验证一个Skill"好用"

在你投入大量精力优化一个Skill之前，你需要先回答：它现在到底好不好用？

很多人对Skill质量的判断方式——"感觉还行"或者"感觉不太好"——太模糊了。你需要一个可衡量的标准。

用三个指标来判断：

指标一：省时率

你用这个Skill之前和之后，做同一件事需要的时间对比。

计算方式：（原来时间 − 现在时间）/ 原来时间 × 100%

- 省时率<30%：效果不明显，检查哪个环节还是太慢
- 省时率30–50%：合格，值得继续用
- 省时率>50%：优秀，你已经找到了一个好的Skill场景

指标二：修改率

AI的输出，你每次需要修改的比例。

计算方式：每次平均修改的字符数 / 输出总字符数 × 100%

- 修改率>30%：说明Skill输出质量不稳定，需要优化
- 修改率10–30%：合格范围
- 修改率<10%：高质量输出，AI已经理解你的标准

指标三：满意度

你主观上对Skill的满意程度。

用1–5分打分，每次使用后记录： – 1分：根本不能用，完全重写 – 2分：需要大改 – 3分：能用，但需要一些调整 – 4分：基本满意，只需小改 – 5分：直接可用，无需修改

每周统计一次平均分。平均分在3.5分以上，说明Skill运行良好。

10.2 常见的6种翻车模式

当一个Skill出了问题，问题通常属于以下六类之一。

翻车模式一：输出格式不对

现象：AI不按你指定的格式输出。

原因：输出规范的描述不够具体。”用表格输出”和”用三列表格输出，第一列是日期，第二列是事项，第三列是状态”的区别。

修复：把输出要求从”建议”改成”规定”。明确说”必须按X格式，不按格式则重新生成”。

翻车模式二：判断不准确

现象：AI在某些判断上持续犯错。

原因：判断标准模糊。”标注重要事项”——AI不知道你的”重要”标准是什么。是影响项目进度的算”重要”？还是客户关注的算”重要”？

修复：把”重要”拆解成具体标准。”重要事项的定义：1. 影响项目关键路径的；2. 客户明确提及的；3. 涉及资源变化的。”

翻车模式三：信息遗漏

现象：AI忽略了你提供的某些关键信息。

原因：信息太多，AI没有优先级。你给了20条信息，AI只关注了前5条。

修复：按重要性排序输入信息。或者在说明书里明确”请完整处理所有信息，不要遗漏”。

翻车模式四：过度发挥

现象：AI补充了你没说的信息，而且补充错了。

原因：AI在”脑补”它不知道的信息。它以为自己知道，其实不知道。

修复：在说明书里要求”不确定的信息标注’需确认’，不要自行补充”。

翻车模式五：用着用着变差了

现象：刚开始还好，用了两周后质量下降。

原因：可能的原因——1. 你的工作内容变了但说明书没更新；2. 你给的反馈太多太杂，AI搞混了；3. AI平台有更新影响了输出方式。

修复：先检查说明书是否过时，再清空并重写说明书（保留核心内容，去掉临时调整）。

翻车模式六：反馈无效

现象：你告诉AI”这里不对”，但下一次它还是错的。

原因：反馈不够具体。说”这个不对”不够——要让AI知道”哪里不对、应该是什么、为什么这个更好”。

好的反馈：”第二点的时间写错了，应该是5月18日不是5月16日。原因是这个数据源更新了，以后请以[新数据源]为准。”

10.3 迭代节奏

不是所有的修改都值得马上做。你需要一个迭代节奏。

日常微调（每天）： – 在使用Skill时直接给反馈 – 让AI记住你的偏好 – 不需要更新说明书

周度检查（每周一次，10分钟）： – 统计本周的省时率和修改率 – 回顾这周遇到的问题 – 把共性问题补进说明书

月度升级（每月一次，30分钟）： – 全面检查Skill的五个设计层 – 评估是否需要增加新的功能 – 检查知识库是否需要更新

季度重做（每季度一次，1小时）： – 从0开始重新评估这个Skill是否仍然需要 – 你的工作有没有变化？Skill还适配吗？ – 有没有更好的方式？是改造还是重做？

10.4 什么时候该改，什么时候该重写

不是所有问题都需要改。有些问题意味着你应该重写。

该改的信号： – 大部分输出是好的，只是一个模块有问题 – 输入信息没有大的变化 – 你还能说清楚”哪里不对”

该重写的信号： – 大部分输出都不满意 – 你的工作流程发生了大的变化 – 你已经说不清”哪里不对”了——说明说明书跟你的工作已经脱节了

学会判断什么时候重写，跟学会设计Skill一样重要。 重写不是失败——意味着你对这个工作的理解更深了，能做出更好的设计。

10.5 从v1到v10：一个Skill的进化日志

这是一个真实Skill的迭代记录。

v1（第1天） — 基础版本，能用，但格式不统一 **v2（第3天）** — 加输出规范，格式稳定了 **v3（第1周末）** — 发现了遗漏问题，增加”请完整处理所有信息” **v4（第2周）** — 加入长期记忆

模块，不再重复问基本信息 **v5（第3周）** — 加入异常处理”不确定的标注需确认” **v6（第5周）** — 加入第一个工具：查阅术语表 **v7（第2个月）** — 工作流程变了，重写说明书（不是改——是重写） **v8（第3个月）** — 分享给同事，加了使用说明 **v9（第4个月）** — 同事反馈的共性问题，加到了说明书中 **v10（第6个月）** — 大版本更新：从”被动接收”变成了”主动提醒”

从v1到v10用了半年。每个版本的变化都不大——有时候只是加了一句话。但半年下来，这个Skill从”偶尔能用”变成了”每天必用”。

这就是迭代的力量。

本章小结

1. **三个指标判断好坏** — 省时率、修改率、满意度（每周统计一次）
2. **六种翻车模式** — 输出格式不对、判断不准确、信息遗漏、过度发挥、越用越差、反馈无效
3. **迭代节奏** — 每日微调→每周检查→每月升级→季度重做
4. **改还是重写** — 能用就改，不能用了就重写。重写不是失败
5. **v1到v10** — 半年时间，每个版本进步一点点。不是奇迹，是坚持
6. **你的任务** — 用三个指标评估你现在的日报Skill，看看它处于什么水平

📚 延伸阅读

- 《精益创业》— Eric Ries。”Build-Measure-Learn”循环跟Skill的”做→用→改”循环是一回事。
- 公众号文章《我的Skill进化日志：从v1到v10》— 树懒老K

第11章 多Skill协作：搭一个"助手矩阵"

> *一个Skill能帮你省时间。但几个Skill配合起来，能帮你重新定义怎么工作。*

11.1 什么时候需要多个Skill

你已经有了一个日报Skill，每天帮你省了30分钟。很好。

现在你可能开始想：那我能不能再做一个周报Skill？再做一个项目进度跟踪Skill？再做一个客户反馈分析Skill？

可以。但先想清楚一个问题：**你需要的是更多的Skill，还是更好的Skill？**

以下信号告诉你**该做新Skill**：

- 你的第二个任务是完全不同类型的工作（比如日报是信息整理，第二个是数据分析——它们不重叠）
- 你现在的Skill说明书越来越长，超过了1000字（可能说明一个Skill里塞了太多事）
- 你的日报Skill很稳定，但你每周还有另一个固定工作需要手动做

以下信号告诉你**该优化现有Skill**：

- 你的日报Skill还没稳定运行两周（太早了，先跑稳）
- 你发现日报Skill里可以加一个模块来处理周报（合并而不是拆分）

- 你只是”觉得应该多做一些”而不是”真的需要一个新助手”

一个经验法则： 你的第一个Skill稳定使用一个月之后，再考虑第二个。

11.2 Skill之间的配合模式

当你有了两个以上的Skill，它们之间可以配合工作。常见的有三种模式。

模式一：串联模式

一个Skill的输出，是另一个Skill的输入。

案例： – 信息收集Skill → 整理日报Skill → 日报直接可用 – 竞品监测Skill → 竞品分析Skill → 周报自动生成

设计要点：明确前一个Skill的输出格式，让后一个Skill可以直接读取。

模式二：并联模式

多个Skill独立工作，产出汇总到一个人工节点。

案例： – 项目进度Skill（自动更新状态） – 团队反馈Skill（收集成员问题） – 风险预警Skill（识别潜在风险） – 三项产出汇总到你手里，由你写周报

设计要点：每个Skill的产出使用统一格式标准，方便你汇总。

模式三：主从模式

一个”主Skill”调度多个”子Skill”。

案例： – “项目总管Skill”是你的接口 – 你问它”项目怎么样了” – 它自动调用进度Skill、风险Skill、资源Skill – 汇总成一个整体的回答给你

设计要点：主Skill需要知道什么时候该调用哪个子Skill，以及怎么把多个结果综合起来。

11.3 案例：一个项目经理的Skill矩阵

让我们看一个真实案例——一个项目经理的Skill矩阵是如何演化的。

第一个月：只有一个日报Skill

她每天花30分钟写项目日报。第一个Skill帮她把这个时间缩到了10分钟。

第三个月：增加到三个Skill

1. 日报Skill（每天用，已稳定）
2. 周报Skill（每周一用，基于日报自动生成周报初稿）
3. 会议纪要Skill（开会后用，根据录音或笔记整理纪要）

这三个Skill配合的方式是串联： – 日报信息 → 写入项目日志 – 项目日志 + 周报Skill → 周报初稿 – 会议录音/笔记 + 会议纪要Skill → 会议纪要

第六个月：增加到五个Skill

新增了两个： 4. 风险预警Skill（每周扫描项目状态，识别潜在风险） 5. 资源调度Skill（辅助评估资源分配的合理性）

这五个Skill形成了一个矩阵——覆盖了她工作中80%的”高价值+高重复度”任务。

11.4 避免”Skill爆炸”

一个很自然的倾向：越做越多，越做越细，最后你有了20个Skill，每个只用了一次。

这不是Skill工程，这是另一种形式的”工具收集”。

避免Skill爆炸的三个原则：

原则一：一个Skill解决一个完整的问题

不是”一个任务一个Skill”——是一个完整的问题一个Skill。如果整理日报和整理周报是同一个问题（项目信息汇总），就不应该分成两个Skill。

原则二：Skill的数量控制在你记得住的范围

如果你需要一张清单才能记得自己有哪些Skill——太多了。建议上限是5–7个。超过了就考虑合并或淘汰。

原则三：定期淘汰

每季度检查一次你的Skill列表。那些过去三个月没用过的Skill——是”暂时不用”还是”再也不会用”？后者删掉。

第11章小结

1. **先确认是”需要新Skill”还是”需要优化现有Skill”** — 第一个Skill稳定使用一个月再想第二个
2. **三种协作模式** — 串联（一个接一个）、并联（独立工作后汇总）、主从（一个调度多个）
3. **真实案例** — 一个项目经理从1个Skill到5个Skill的六个月演化
4. **避免Skill爆炸** — 数量控制在5–7个，每季度淘汰不用的
5. **提醒** — Skill的目的是帮你省时间，不是为了让你管理一套复杂的系统。如果管理Skill本身变成了负担，那就违背了初衷。

📚 延伸阅读

- 《系统思考》— 丹尼斯·舍伍德。多个Skill配合的本质，是你在为自己设计一个”工作系统”。系统思维能帮你更好地设计Skill之间的关系。

第12章 把Skill分享给别人

> *你为自己搭了一个好用的助手。现在，让别人也能用上。*

12.1 从个人Skill到团队Skill

你独自使用Skill已经很好了。但有一个更大的价值：**你的同事也可以从中受益。**

问题是——你的Skill是用你的语言、你的习惯、你的工作流设计的。直接扔给同事，他们用不了。

分享Skill不是”复制粘贴”——是”翻译”。

你需要做三件事：

1. **把”你”的部分抽离出来** — 你的名字、你的项目、你的偏好——这些换成通用占位符
2. **加使用说明** — 不是每个人都知道怎么”用”一个Skill
3. **设权限边界** — 别人用的时候，哪些能改哪些不能改

这个过程叫”Skill的封装”。

12.2 Skill怎么封装

把一个个人用的Skill封装成团队适用的版本，通常需要改三处。

第一处：个人信息变配置项

原来你的Skill说明书里：

```
我的名字：小王
我的项目：飞翔项目
我的团队：5人
我的每周报告接收人：李总
```

封装后：

```
<span style="font-size:0.5pt;">§TOC12-3§</span>
## 配置项（使用前请修改）
你的名字：[填写]
你的项目：[填写]
你的团队人数：[填写]
你的报告接收人：[填写]
```

这样，任何人拿到这份说明书，改一下配置项就能用。

第二处：个人偏好变可选功能

原来你的偏好是固定的（”用这个格式”“按这个标准”）。

封装后，变成可选配置：

```
<span style="font-size:0.5pt;">§TOC12-4§</span>
## 输出格式选择
方案A：简洁版（日报用）
方案B：详细版（周报用）
方案C：自定义（请在配置项中填写格式模板）
```

第三处：加使用说明

在Skill文件的最前面或最后面，加一段使用说明：

```
<span style="font-size:0.5pt;">§TOC12-5§</span>
## 如何使用这个Skill
1. 复制本说明书的全部内容
2. 修改"配置项"部分为你的信息
3. 每天/每周使用时，提供你的工作信息
4. AI会按本说明书的要求输出
5. 如果输出不理想，请反馈给创建者（小王）
```

12.3 Skill文档怎么写

一个好的Skill文档应该包含四个部分：

第一部分：Skill卡片（一句话介绍）

“日报助手：帮你把零散的工作记录整理成结构化日报，支持简洁版和详细版。”

第二部分：配置说明（怎么让自己能用）

配置项列表 + 每一项怎么填。

第三部分：使用方式（怎么用）

一个典型的使用场景演示： – 输入什么 – 获得什么 – 怎么给反馈

第四部分：已知限制（什么做不到）

坦诚告诉使用者这个Skill的边界。比如”这个Skill不会自动收集信息——你需要把工作记录提供给AI”。

12.4 分享的三种方式

方式一：一对一分享（最简单）

你把封装好的Skill说明书直接发给同事，他看完就用。

适合场景：你的直属同事、有相同岗位的人。 优点：即时反馈，你能帮他们调整。 缺点：每个人都要单独配置。

方式二：共享文件夹（团队级）

你把封装好的Skill文档放在团队共享空间（飞书文档、Notion、共享文件夹）。

适合场景：5–10人的小团队。 优点：大家都能找到，你可以统一更新。 缺点：需要有人维护版本。

方式三：Skill库（组织级）

你建立一份”团队Skill目录”——列出所有可用的Skill、适用场景、负责人。

适合场景：10人以上的团队或部门。 优点：系统化，可扩展。 缺点：需要投入管理精力。

12.5 权限设计：使用、修改、分发

分享Skill必须想清楚三个权限问题：

使用权限：谁能用这个Skill？ – 公开（团队内所有人） – 指定（只有某些岗位能用） – 受限（需要培训或授权才能用）

修改权限：谁能改这个Skill？ – 只有创建者能改 – 使用者可以提修改建议但无权直接改 – 谁都可以改（适用于开源式分享）

分发权限：谁能把这个Skill再分享给别人？ – 只有创建者可以分享 – 使用者可以用但不能转分享 – 自由转发

建议： 第一次分享时，设置”使用者可提建议但不可直接修改”。等Skill稳定运行一段时间后，再根据情况调整权限。

本章小结

1. **分享不是复制粘贴** — 个人用Skill需要封装才能给团队用
2. **封装的三件事** — 个人信息变配置项、个人偏好变可选功能、加使用说明
3. **Skill文档四部分** — Skill卡片、配置说明、使用方式、已知限制
4. **三种分享方式** — 一对一、共享文件夹、Skill库，根据团队规模选择
5. **权限三要素** — 使用权限、修改权限、分发权限，第一次建议保守一点

6. **你的任务** — 把你的日报Skill封装成”配置项+使用说明”的版本，至少可以分享给一个同事试用

📚 延伸阅读

- 《知识迁移》— 戴维·珀金斯。把个人经验变成他人可用的工具，是一种高难度的”知识迁移”。封装Skill就是这种能力。

第13章 企业级Skill的特殊挑战

> *当你从"搭一个Skill给自己用"到"管理100个Skill给一个团队用"，问题不再是技术问题*
>
> —— 是治理问题。

13.1 个人Skill和企业级Skill的区别

这本书的大部分内容都在教你怎么搭一个”个人Skill”——你一个人用，你自己维护，出了问题你自己知道。

但在企业环境里，事情更复杂：

表 14–1

	个人Skill	企业级Skill
使用者	你一个人	多个人
数据	你的数据	公司数据
维护者	你自己	不确定
标准	你说了算	需要统一
风险	用错了只影响你	用错了可能影响业务
生命周期	想用就用，不用就删	需要管理

这并不意味着你不能把个人Skill推广到团队。而是你要意识到——**从1到N需要的不仅仅是复制粘贴。**

13.2 数据安全与合规

这是企业环境里最先被问到的问题——通常不是你问，是你的IT部门或法务问。

三个必须回答的问题：

问题一：Skill处理的数据存在哪里？

- 如果用的是公有云AI服务：数据会上传到第三方服务器
- 如果用的是本地部署的AI：数据不离开公司网络
- 如果用的是混合方案：敏感数据本地处理，非敏感数据用云服务

你需要知道你公司的信息安全政策允许哪种方案。

问题二：哪些数据可以被Skill处理？

不是所有数据都适合让AI处理。适合的： – 公开信息、内部文档（非机密） – 脱敏后的业务数据 – 已授权公开的客户信息

不适合的： – 客户隐私数据（身份证号、联系方式） – 公司核心商业机密 – 涉及法律和合规敏感的信息

问题三：AI的产出结果归谁？

当AI基于公司数据给出了一个判断——这个”判断”的知识产权归谁？当AI的产出被用于业务决策时，谁为这个决策负责？

这些问题没有标准答案——每家公司不一样。你需要跟你的法务和合规团队确认。

对于个人实践者，两个”绝对不能”： – 绝对不要把自己的公司数据上传到不安全的第三方AI服务 – 绝对不要让AI做出未经你确认的业务决策

13.3 多用户场景下的权限治理

当你的Skill被多个同事使用时，你需要回答：

谁可以用？ – 按岗位：只有运营岗能用，还是所有人？ – 按场景：只有A项目能用，还是所有项目？ – 按等级：只有经理级别能用，还是全员？

谁可以改？ – 版本控制：如果你改了说明书，同事的Skill会自动更新吗？ – 自定义：同事能不能加自己的偏好？ – 分支：会不会出现一个Skill有多个”变体”？

谁负责维护？ – 如果Skill出了问题，谁负责修？ – 如果工作需要调整，谁负责更新Skill？ – 如果有用户投诉Skill不好用，谁处理？

一个建议：设定一个”Skill管理员”角色。

即使只是团队里的非正式角色——指定一个人负责Skill的质量、更新和用户支持。不需要全职，但需要有明确的归属。

13.4 Skill的生命周期管理

一个企业级Skill从生到死，会经历五个阶段：

阶段一：诞生

一个人为自己搭了一个Skill，觉得好用，开始想分享。

阶段二：试用

封装后给2–3个同事试用。收集反馈，确认它确实对团队有价值——而不仅仅对创建者自己有价值。

阶段三：推广

正式发布到团队。设立配置指南、使用说明、已知限制。安排一个维护者。

阶段四：运行

持续使用和维护。定期的质量检查。收集用户反馈。处理异常。做版本更新。

阶段五：退役

当Skill不再需要时（工作流程变了、更好的替代出现了、用户不再使用）——正式宣布退役，存档相关资料，清空不再使用的数据。

很多团队的问题： 只做了前三个阶段，不做第四和第五个阶段。结果就是一堆没人用也没人维护的”僵尸Skill”。

13.5 四个常见的企业级陷阱

陷阱一：Skill泛滥

每个人都在做自己的Skill，最后团队里有几十个”几乎一样但细节不同”的日报助手。

应对：推广前先统一命名规范和分类标准。做之前先查一下有没有类似的已经存在。

陷阱二：Skill越权

某个Skill获得了过多的权限，能读取它不应该读取的数据。

应对：严格执行”最小权限”原则。每次给Skill增加工具或数据访问权限前，先问”它真的需要这个吗”。

陷阱三：Skill依赖

团队过于依赖某个Skill，但这个Skill的唯一知情者（创建者）调岗或离职了，没人知道怎么维护。

应对：每个推广到团队的Skill，至少有两个以上的人理解它的工作原理。

陷阱四：Skill无法更新

Skill说明书写死了，当工作流程变化时没人去改它——于是它给出的建议越来越不靠谱，最终被弃用。

应对：设置定期的”Skill健康检查”。至少每季度检查一次推广中的Skill是否仍然适配当前的业务流程。

本章小结

1. **个人到企业级的差距** — 不是技术的差距，是治理的差距
2. **数据安全三问** — 数据存哪、什么能处理、产出归谁
3. **权限治理** — 使用权限、修改权限、维护责任
4. **生命周期五阶段** — 诞生→试用→推广→运行→退役
5. **四个陷阱** — 泛滥、越权、依赖、无法更新
6. **你的任务** — 如果你想把Skill分享给团队，先用本章的框架做一个风险评估——你的Skill适合分享吗？如果适合，怎么做？

📚 延伸阅读

- 《企业IT架构转型之道》— 钟华。企业级治理的核心问题，这本书讲得很清楚——不是技术问题，是”怎么把个人能力变成组织能力”的问题。
- 《赋能》— 斯坦利·麦克里斯特尔。从”一个人强”到”一群人强”，这本书讲的就是这个跃迁。

附录

附录A：Skill模板库

模板一：信息整理类Skill

适用场景： 把零散的工作记录整理成结构化报告

```
<span style="font-size:0.5pt;">§TOC14-2§</span>
## 固定信息
我的名字：[填写]
我的岗位：[填写]

<span style="font-size:0.5pt;">§TOC14-3§</span>
## 任务描述
帮我整理[报告名称]。

<span style="font-size:0.5pt;">§TOC14-4§</span>
## 输入信息
我每天/每周会给你以下信息：
[描述输入格式]

<span style="font-size:0.5pt;">§TOC14-5§</span>
## 工作流程
1. 阅读所有输入信息
2. 按以下分类整理：[分类列表]
3. 检查是否有遗漏或明显错误
4. 按输出格式生成报告

<span style="font-size:0.5pt;">§TOC14-6§</span>
## 输出格式
[粘贴你的模板]

<span style="font-size:0.5pt;">§TOC14-7§</span>
## 质量标准
- 分类准确
- 完整无遗漏
- 不确定的信息标注"需确认"
```

模板二：判断辅助类Skill

适用场景： 需要从多个选项中做出选择

```
<span style="font-size:0.5pt;">§TOC14-8§</span>
## 固定信息
[你的背景信息]

<span style="font-size:0.5pt;">§TOC14-9§</span>
## 任务描述
帮我分析[决策问题]，列出选项和依据。

<span style="font-size:0.5pt;">§TOC14-10§</span>
## 输入信息
我需要给你的信息：
1. 决策背景：[描述]
2. 可选方案：[列出]
3. 评估维度：[列出]

<span style="font-size:0.5pt;">§TOC14-11§</span>
## 工作流程
1. 按我给的评估维度，逐项分析每个方案
2. 列出每个方案的风险和收益
3. 给出你的推荐（标注置信度）

<span style="font-size:0.5pt;">§TOC14-12§</span>
## 输出格式
一、方案对比（表格形式）
二、各方案风险分析
三、推荐方案及理由

<span style="font-size:0.5pt;">§TOC14-13§</span>
## 注意事项
- 你的角色是提供分析，不是替我决策
- 不确定的地方请标注"需验证"
```

附录B：翻车记录30条

1–10条：说明书设计问题

1. 说明书太长（超过1500字），AI开始遗漏关键指令。→ **控制字数，或拆成多个Skill**
2. 只写”做什么”不写”怎么做”，AI自由发挥。→ **加执行步骤**
3. 输出标准模糊（”好一点”“专业一点”），AI无法量化。→ **给具体标准**
4. 输入信息交给AI时不整理，杂乱的输入影响产出质量。→ **输入前先做基本分类**
5. 忘了加”不确定的标注’需确认’”，AI在编造信息。→ **加入防幻术条款**
6. 说明书里用了太多行业术语，AI理解偏差。→ **术语第一次出现时加定义**
7. 想让一个Skill做太多事，每个都做不好。→ **拆成多个Skill**
8. 把AI当搜索引擎用，期待它知道它不知道的事情。→ **明确知识边界**
9. 规则写得像建议（”最好”“尽量”），AI当成可选项。→ **改成”必须”“不能”**
10. 反馈不够具体（”不对”“不行”），AI不知道怎么改。→ **说清楚哪里不对、应该是什么**

11–20条：使用习惯问题

1. 开头几周不坚持用，断断续续，Skill无法积累记忆。→ **至少坚持连续使用两周**

2. 每次给AI的信息格式不一样，AI需要时间适应。→ **建立信息输入的固定格式**
3. 期待AI一次就完美，错了就直接放弃。→ **接受v1能用就行**
4. 不给反馈，AI不知道自己做得怎么样。→ **每次使用后至少给一个反馈**
5. 给了反馈但不说原因（”这里改一下”）。→ **说明改动的原因**
6. 频繁切换AI平台，每次都要重新适应。→ **选一个平台稳定使用**
7. 在移动端和电脑端交替使用，上下文不连续。→ **尽量在同一设备上完成一个流程**
8. 用了两周发现场景变了，但没有更新说明书。→ **定期检查说明书是否匹配当前工作**
9. 过于依赖AI，自己不再学习新技能。→ **Skill是放大器，不是替代品**
10. 追求完美，不肯启动。→ **先做一个能用的版本，哪怕只有60分**

21–30条：分享和协作问题

1. 直接把自己的Skill扔给同事，同事用不了。→ **封装后再分享**
2. 分享时不写使用说明，同事不知道”怎么用”。→ **配使用说明**
3. 同事反馈问题后不更新，Skill越来越没人用。→ **定期收集反馈并更新**
4. 团队里出现了多个功能重复的Skill。→ **先查重，再创建**
5. 重要Skill只有一个人知道怎么维护。→ **至少两个人理解Skill的原理**

6. 把Skill推广到团队前没做安全评估。→ **先跟IT确认数据安全**
7. 给了AI过高的权限，出现了不该发生的操作。→ **严格遵循最小权限原则**
8. 没有人定期检查推广中的Skill是否仍然有效。→ **每季度做一次Skill健康检查**
9. 没有淘汰机制，”僵尸Skill”越来越多。→ **定期清理不再使用的Skill**
10. 以为Skill工程是一次性的工作，做完就不管了。→ **Skill是有生命周期的，需要持续维护**

附录C：工具链速查

当前推荐的工具组合（2026年）

如果你刚开始（个人）： – 通用AI助手（ChatGPT / Claude / 文心一言）— 直接用对话界面 – 不需要额外工具，先把第一个Skill跑起来

如果你需要更多功能（个人进阶）： – 通用AI助手 + 知识库（飞书文档 / Notion / Obsidian 作为知识库） – MCP或平台内置工具接入（查询网络、读取文档）

如果是团队使用： – AI平台 + 共享知识库 + 权限管理 – 建议选择支持Skill封装和分享的平台

常见问题

Q：我应该用ChatGPT还是Claude还是国内的？ 取决于你的数据安全要求和语言偏好。建议：先用一个，把Skill跑起来，再考虑换平台。换了也没关系——Skill的核心是”你的说明书”，说明书可以带到任何平台。

Q：需要学编程吗？ 不需要。这本书教的所有内容都不需要写代码。

Q：我的公司不让用公有云AI怎么办？ 看公司是否有本地部署的AI方案。如果有，同样的方法可以用。如果没有，你可以用Skill的方法论”在允许的范围内”——比如在内部知识库里做结构化整理。

Q：Skill能跟我的工作软件（飞书/钉钉/企微）集成吗？ 部分AI平台已经支持这种集成。具体情况取决于你使用的平台。建议：先用文字版把Skill跑通，再考虑集成的事。

树懒老K（拙一）

30年企业服务经验 · 专注AI智能体与组织变革

个人网站　　个人微信　　公众号

慢一点，深一度