2026年4月 · 橙皮书系列

Agent Skills
让AI记住你的工作方式

从概念到实战，从安装到创作——AI时代的技能系统完全指南

Agent Skills — Teach Your AI How You Work

定位：面向AI工具用户的实战手册

前置要求：使用过Claude Code / Cursor / Codex等任一AI编程工具

篇幅：~80页 · 10章 + 附录

实战案例：基于作者亲手打造的27个Skills

花叔

公众号「花叔」· B站「AI进化论-花生」

> 知识星球「AI编程：从入门到精通」专属内容

本手册基于Agent Skills开放标准（agentskills.io）编写。Skills生态发展迅速，部分平台数据和安装方式可能随时间变化，建议配合官方文档使用。

目录

Contents

E 术语表

01 AI工具的下一个进化

The Next Evolution of AI Tools

你有没有这种感觉：每次打开AI工具，都要把同样的事情重新解释一遍？

我用AI写了两年多的公众号。每次写文章，我都要跟AI说一遍：先给我3个选题方向，每个要有标题、大纲和优劣分析。然后我要说：去搜一下最新信息，重点看官方博客和英文媒体。然后再说：初稿写完了，帮我审校一遍，重点降AI味，把「说白了」「简单来说」这些套话去掉。然后是配图、然后是转成X平台的短内容……

每一步我都知道该怎么做。问题是，**AI不记得**。

每次新对话，AI都是一张白纸。我积累了两年的工作流程、审美偏好、写作禁忌，全部要从零开始教。这就像你招了一个超级聪明的实习生，但他每天早上醒来都失忆了。

Agent Skills解决的就是这个问题。

四次进化 Four Evolutions

要理解Skills，需要先看看AI工具是怎么一步步走到今天的。

2022年底，ChatGPT发布，AI工具进入**Chat时代**。你问它问题，它回答你。很神奇，但它只能说，不能做。你让它帮你改一个文件，它只能把修改后的内容贴给你，你自己复制粘贴。

2023年，Tool Use（函数调用）出现了。AI不再只是聊天，它可以**调用工具**了。搜索网页、执行代码、读写文件。这一步跨度很大，AI从「顾问」变成了「助手」。但每个AI产品都要自己实现这些工具接口，标准不统一，生态碎片化。

2024年底，Anthropic发布了**MCP（Model Context Protocol）**。这是一个开放协议，让AI工具可以用统一的方式连接外部服务。接上MCP Server，AI就能操作数据库、发Slack消息、管理GitHub仓库。MCP给了AI「手」，让它能触及真实世界。

2025年，**Agent Skills**出现了。

> 如果前三次进化解决的是「AI能做什么」，那Skills解决的是一个更本质的问题：**AI怎么按你的方式做**。

这四次进化的关系，可以用一个类比来理解：

进化阶段	类比	解决的问题
Chat	一个能说会道的人	AI能理解自然语言
Tool Use	给他一套工具	AI能执行操作
MCP	工具有了统一标准	AI能连接任何服务
Skills	给他一本工作手册	AI能按你的方式工作

注意最后一行。MCP让AI有了「手」，但光有手还不够。你新招了一个员工，给了他电脑、给了他所有系统的权限。但他还是不知道你们公司的周报怎么写、代码review用什么标准、客户邮件用什么语气。

Skills就是那本新员工手册。这也是Anthropic官方博客用的类比：Skills是给AI Agent的「入职指南」，把领域知识打包成可发现、模块化的能力。

一个操作系统的类比 The Operating System Analogy

还有一个更直观的方式来理解整个体系。

把AI Agent想象成一台电脑。操作系统（比如macOS）提供了基础能力。MCP就像是USB接口和驱动程序，让电脑能连接打印机、摄像头、外接硬盘。但你买了一台电脑，装了所有驱动，它还是一台空电脑。

你需要装App。

Skills就是AI Agent的App。每个Skill是一个独立的应用程序，教AI完成一项特定的工作。就像你的手机上有备忘录、相机、地图、微信一样，你的AI Agent上可以装选题生成Skill、审校Skill、配图Skill、数据分析Skill。

区别在于：手机App是代码写的，Skill是自然语言写的。你不需要会编程就能创建一个Skill。这一点非常重要，后面会展开讲。

从理论到实际：我的27个Skills 27 Skills in Practice

说这么多概念，不如看看真实场景。

过去一年多，我陆续给自己的AI工作流创建了27个Skills。它们覆盖了我作为内容创作者的几乎全部工作：

- 写公众号文章：从选题、调研、素材搜索、初稿、审校、配图到社交媒体分发
- 做视频：大纲生成、脚本口语化、封面标题检查、弹幕生成
- 做演示文稿：17种视觉风格可选，AI插画和HTML两条路径
- 写书：调研、多Agent并行写作、HTML构建、版本管理
- 数据分析：多专家并行分析，HTML报告自动生成

这些不是玩具。我每天都在用它们生产真实内容。这本橙皮书本身，就是用我的huashu-book-pdf这个Skill来组织和构建的。

下面用一个具体的例子来展示Skills怎么串联工作的。

一篇公众号文章的诞生 How a WeChat Article Is Born

假设我要写一篇关于「GLM-5V-Turbo发布」的公众号文章。在没有Skills之前，我需要手动指挥AI完成每一步。现在，整个流程变成了这样：

1. **选题生成（huashu-topic-gen）**

   我只需要说「GLM-5V-Turbo发布了，给几个选题」。Skill自动生成3-4个方向，每个包含标题、核心角度、大纲和优劣分析。我不用解释输出格式，不用解释什么是好的选题，Skill都记着。

2. **深度调研（huashu-research）**

   选定方向后，调研Skill接管。它知道要先搜官方博客，再搜英文科技媒体，然后是中文社区讨论。每搜到一批信息就增量保存到文件里，防止会话中断丢失调研成果。最后整理成结构化简报：关键事实、信息源、待确认问题。

3. **个人素材（huashu-material-search）**

   这个Skill会在我的1800多条即刻动态里搜索相关内容。比如我之前发过关于GLM系列模型的吐槽、使用体验、对比测试。它把这些真实的个人素材找出来，改写成适合长文的段落。这是降AI味最有效的方式之一，因为这些经历是真实发生过的。

4. **撰写初稿**

   这一步不需要专门的Skill。但因为前三步的产出（选题方向、调研简报、个人素材）都已经保存在文件里了，写作时AI可以直接引用，信息密度自然很高。

5. **三遍审校（huashu-proofreading）**

   初稿完成后，审校Skill上场。它知道花生的写作禁忌：不用「说白了」「简单来说」等套话，破折号全篇最多1-2处，「」引号不过度使用。它会做三遍审校：第一遍查事实，第二遍降AI味（有一份6大类AI腔识别清单），第三遍打磨细节。目标是把AI检测率降到30%以下。

6. **配图（huashu-wechat-image）**

   配图Skill知道我的审美偏好：不用赛博霓虹风格、不用深蓝色底、封面不加个人水印。它先提出3个配图方向让我选，确认后用AI生成图片，上传到图床，生成的Markdown链接直接内联到文章里。零placeholder，所有图片都是可访问的网络链接。

7. **社交分发（huashu-article-to-x）**

   文章完成后，这个Skill把3000-5000字的长文浓缩成200-500字的X平台内容。它知道要保持口语化、用具体数字、结尾加互动引导。提供金句开头、数据冲击、价值直给三种风格让我选。

整个过程中，**我没有解释过一次「输出格式是什么」「审校标准是什么」「配图风格是什么」**。因为这些都写在Skills里了。AI记住了我的工作方式。

> **这就是Skills的价值。**不是让AI变得更聪明，而是让AI变得更懂你。它不再是一个什么都能做但什么都要你教的通用助手，而是一个了解你的工作习惯、审美偏好、质量标准的专属助手。

20多个产品已经支持 Industry Adoption

Agent Skills不只是Anthropic一家的东西。2025年底，Anthropic把Skills发布为开放标准（agentskills.io），到2026年4月，已经有20多个AI产品采纳了这个标准。

名单包括你能想到的几乎所有主流AI编程工具：Claude Code、Cursor、GitHub Copilot、VS Code、OpenAI Codex、Gemini CLI、JetBrains Junie、Windsurf、Roo Code，还有字节跳动的TRAE、Databricks、Snowflake、Spring AI……

这意味着什么？意味着你写的一个SKILL.md文件，可以在所有这些产品上运行。你不需要为每个工具重复配置你的工作流程。学一次，到处用。

这让Skills成为了一种真正的「可移植的工作经验」。

这本书会教你什么 What You Will Learn

这本橙皮书分为四个部分：

概念篇（第1-3章）：你正在读的这部分。理解Skills是什么、为什么有效、背后的技术原理。不会太学术，但会帮你建立正确的心智模型，这样后面用起来才不会懵。

实战篇（第4-6章）：从生态全景到安装到实际使用。跟着做完这部分，你就已经能用Skills提升日常工作效率了。

创作篇（第7-9章）：这是我认为最有价值的部分。教你从零开始创建自己的Skill，用官方的skill-creator工具自动生成和优化，以及5种高级设计模式。创建Skill的门槛比你想象的低得多，但设计一个好的Skill需要一些技巧。

展望篇（第10章）：Skills会怎么发展？对个人和团队意味着什么？

全书会持续用我实际使用的Skills作为案例。这不是一本教你「理论上可以做什么」的书，而是一本「我已经这样做了、你也可以这样做」的书。

> **核心建议**
> 如果你已经在用Claude Code、Cursor或其他AI编程工具，但还没接触过Skills，建议从第4章开始快速上手，遇到概念问题再回头翻前三章。如果你是完全的新手，按顺序读效果最好。

02 Skills的本质

What Skills Really Are

一份改变AI行为的文档，仅此而已。

上一章我用了不少类比来解释Skills：新员工手册、手机上的App、操作系统的应用程序。这些类比帮你建立直觉，但如果要真正用好Skills，你需要知道它到底是什么。

答案比你想的简单得多。

一个Skill，本质上就是一个叫SKILL.md的文本文件。Markdown格式，用自然语言写成。没有代码，没有编译，没有API调用。你用记事本就能创建一个Skill。

但它和你随手写的Prompt有三个本质区别：**模块化、可触发、可分享**。这三个特性让一个普通的文本文件变成了一种全新的东西。

拆解一个最简单的Skill Anatomy of a Simple Skill

先看一个最小的例子。假设你每周都要写团队周报，格式固定，但每次都要跟AI解释一遍。你可以把它做成一个Skill：

```
---
name: weekly-report
description: |
  生成团队周报。当用户提到「周报」「weekly report」「本周总结」时使用此skill。
---

# 周报生成

1. 询问本周完成的主要工作
2. 按「完成/进行中/计划」三栏整理
3. 每条用一句话概括，不超过15字
4. 总字数控制在300字以内
```

就这么多。十几行，一分钟能看完。但它包含了一个Skill的全部结构。

两层结构 Two Layers

每个SKILL.md都由两层组成：

1. **Frontmatter（YAML元信息）**

   就是顶部两个 `---` 之间的部分。必填字段只有两个： `name` （名字）和 `description` （描述）。description里不光写这个Skill干什么，还要写什么时候该触发它。比如上面的周报Skill，description里就写了「当用户提到周报、weekly report、本周总结时使用」。Agent框架通过读这段描述来判断要不要加载这个Skill。

   除了name和description，还有一些可选字段： `allowed-tools` （允许使用哪些工具）、 `context` （需要加载的上下文文件）、 `user-invocable` （是否可被用户直接调用）等。大多数情况下你只需要name和description就够了。

2. **执行流程（Markdown正文）**

   这是Skill的核心内容，用普通的Markdown写成。它告诉AI具体该怎么做：先问什么、再做什么、输出什么格式、有什么约束。你怎么给同事写操作手册，就怎么写这部分。

两层结构，各司其职。Frontmatter让系统发现你、知道何时加载你，执行流程让AI知道该怎么做。触发条件不是独立的一层，它就写在description里，是元信息的一部分。

它和System Prompt有什么区别 Skills vs System Prompt

你可能会想：这不就是一个System Prompt吗？写一堆指令让AI遵守，有什么新鲜的？

区别很大。

System Prompt是全局的、永久加载的。你在CLAUDE.md里写的所有规则，每一轮对话AI都会看到。它像是公司的规章制度，所有员工都要遵守，不管你是做财务还是做市场。

Skill是模块化的、按需加载的。它像是某个岗位的操作手册，只在需要的时候翻开。

推荐

Skill（模块化）

你说「写周报」→ 加载周报Skill
你说「审校一下」→ 加载审校Skill
你说「配个图」→ 加载配图Skill
不说 → 不加载，不占上下文

不推荐

System Prompt（全局）

周报规则、审校规则、配图规则...
全部写在一个文件里
每次对话全部加载
上下文越来越臃肿

这个区别在实际使用中影响巨大。我有27个Skills，如果全部写进System Prompt，那就是上万字的指令，每次对话都要加载。AI的上下文窗口是有限的，你塞进去的指令越多，留给实际工作的空间就越少。

Skills的模块化设计，本质上是在有限的上下文窗口里做资源调度。用到什么加载什么，不用的就不加载。这个思路和操作系统管理内存是一样的。

它和MCP有什么区别 Skills vs MCP

另一个常见的困惑：Skills和MCP是什么关系？

一句话讲清楚：**MCP是能力接口，Skill是知识和流程**。

MCP告诉AI「你可以发飞书消息」。但AI不知道发飞书消息的时候应该先创建文档、再写入内容、再添加权限、最后把文档所有权转给用户。这些流程和经验，写在Skill里。

MCP = 我能连什么（连飞书、连GitHub、连数据库）。
Skill = 我该怎么做（发飞书时的完整流程、写代码时的审查标准）。

回到操作系统的类比：MCP是驱动程序和接口，Skill是App。驱动程序让电脑能连打印机，但你还需要一个排版软件来告诉电脑怎么打印一份好看的文档。两者配合才能完成真正的工作。

> **核心建议**
> 很多时候一个好的Skill会同时用到MCP。比如我的飞书文档Skill，流程里就会调用飞书MCP提供的API。Skill负责决策（先做什么后做什么），MCP负责执行（调接口发请求）。

和Cursor Rules的关系 The Naming Convention

如果你用过Cursor，你可能听过 `.cursorrules` 这个东西。如果你用GitHub Copilot，也可能见过类似的配置文件。它们和Skills是什么关系？

本质上是同一种东西的不同名称。

2024年，各家AI编程工具各自发明了自己的配置格式：Cursor有 `.cursorrules` ，Claude Code有 `CLAUDE.md` ，Copilot有自己的配置方案。用户很痛苦，同样的工作流规则要为每个工具写一遍。

2025年底，Anthropic把Agent Skills发布为开放标准，推动整个行业统一到SKILL.md格式。截至2026年4月，已有20多个AI产品采纳了这个标准。Cursor、Copilot、Codex、Gemini CLI、JetBrains Junie……你能想到的主流AI工具基本都在名单上。

这意味着你写的一个SKILL.md，可以在所有这些工具上运行。写一次，到处用。

真实案例：三遍审校Skill Real Case: Proofreading Skill

刚才的周报Skill只有15行，是最简单的例子。真实世界的Skill会复杂得多。来看看我日常使用频率最高的一个：**huashu-proofreading**，三遍审校Skill。

这个Skill的任务是：把一篇AI辅助写出来的文章，改到「不像AI写的」。目标是把AI检测率降到30%以下。

先看它的Frontmatter：

```
---
name: huashu-proofreading
description: |
  三遍审校流程：降低AI检测率、去AI腔、增加人味。包含6大类AI腔识别清单和花生风格改写规则。
  当用户写完文章/初稿后需要润色时触发。即使用户只是说"看看这篇文章"、"帮我改改"、"润色一下"、"polish"也应触发
  明确触发词：审校、降AI味、太AI了、AI检测率、没人味、像AI写的、自然一些、改一改、过一遍。
---
```

注意description字段。触发条件直接写在描述里。我不需要精确地说「请用审校Skill」，只要说「帮我改改」「润色一下」「太AI了」，Agent读到description里的触发词，就会自动把这个Skill加载进来。这让Skill的使用变得无感。

再看执行流程的核心部分。这个Skill设计了三遍审校：

1. **第一遍：内容审校**

   检查事实准确性、逻辑连贯性、信息完整性。数据对不对？产品名称有没有写错？前后有没有矛盾？这一遍解决「对不对」的问题。

2. **第二遍：风格审校（降AI味）**

   这是最核心的一遍。Skill里内置了一份6大类AI腔识别清单：套话连篇、AI句式、书面词汇、结构机械、态度中立、细节缺失。每一类都有具体的❌反面案例和✅改写方向。比如把「在当今AI技术飞速发展的时代」改成「Claude Code出了。我用了两周」。

3. **第三遍：细节打磨**

   检查句子长度（不超过30字）、段落长度（手机屏幕3-5行）、标点节奏、加粗频率。这一遍解决「读起来舒不舒服」的问题。

整个SKILL.md大约300行，包含了大量的改写示例。比如：

不推荐

AI味写法

「经过一段时间的使用，我发现Claude Code能够有效提升开发效率。」

推荐

改写后

「用了两周。最明显的变化是，之前要3天的项目，现在1天半就能搞定。特别是重构老代码，Claude Code能一次性找出所有关联文件，不用我一个个翻。」

为什么要在Skill里写这么多示例？因为AI是通过示例来理解你要什么的。你光说「去掉AI味」，AI不知道你指的是哪种AI味。但你给了6大类识别清单和改写示例，AI就能精确地知道该改什么、怎么改。

这也是设计Skill的一个核心原则：**与其给AI抽象的指令，不如给它具体的示例**。后面第8章会展开讲这个。

一份文档为什么能改变AI的行为 Why It Works

到这里你已经知道Skill是什么了。一个有特定格式的Markdown文件，frontmatter里写元信息和触发条件，正文里写执行流程。

但你可能还是觉得不太对劲：凭什么一个文本文件就能改变AI的行为？AI又不是程序，你给它一份文档它就会照做？

这个问题很好。答案涉及LLM的一些底层特性。下一章就来讲这个。

03 Skills背后的原理

How Skills Actually Work

为什么一个文本文件就能改变AI的行为？因为它恰好踩中了LLM最擅长的三件事。

上一章结尾留了一个问题：凭什么一个Markdown文件就能让AI按照你的方式工作？

这不是魔法。Skills之所以有效，是因为它利用了大语言模型（LLM）的三个底层特性：**指令遵循、上下文学习、条件触发**。理解这三个特性，你就能设计出更好的Skill。不理解，你也能用，但容易踩坑。

指令遵循：AI为什么会「听话」 Instruction Following

现代LLM都经过了大量的指令微调（instruction tuning）。通俗地说，就是研究人员用海量的「指令-回应」对来训练模型，让它学会理解并执行人类的指令。

这个过程的结果是：当你给AI一段结构化的指令，它会非常认真地遵循。而SKILL.md的格式恰好是LLM最擅长理解的格式。YAML的键值对结构清晰，Markdown的标题和列表层次分明。这不是巧合，Agent Skills标准选择这种格式，就是因为LLM对它的理解最准确。

举个例子。你在Skill里写：

```
3．每条用一句话概括，不超过15字
```

AI会真的数字数。你写「简洁一些」，AI不知道你要多简洁。你写「不超过15字」，它就知道了。这就是结构化指令的威力。

但指令遵循有一个重要前提：指令必须在AI的上下文窗口里。AI看不到的东西，它不会遵循。这就引出了第二个特性。

上下文窗口与Skill加载 Context Window and Skill Loading

上下文窗口是LLM的「工作记忆」。你和AI的对话内容、系统指令、加载的文档，全部要放在这个窗口里。窗口大小有限，即使是Claude最新的模型也只有100万token的上下文窗口。听起来很大，但如果你同时加载27个Skill，每个2000字，那就是5.4万字，大约7万token。光Skill就占了7%的上下文。

所以Skill不能全部加载，必须按需加载。这也是为什么Skills不是System Prompt的原因。

Anthropic在设计Agent Skills系统时，用了一个很聪明的架构：渐进式披露（Progressive Disclosure）。分三个层级：

L1 **预加载：只读元数据（约100词/Skill）**

Agent启动时，只读取每个Skill的Frontmatter，也就是name和description。一个Skill的元数据大约只有几十个token。27个Skill加起来也不过一两千token，几乎不占空间。这个阶段的目的是让系统知道「有哪些Skill可以用、什么时候该用」。

L2 **按需加载：完整SKILL.md正文（建议500行以内）**

当用户说了触发词，或者Agent判断当前任务需要某个Skill时，才把完整的SKILL.md正文加载到上下文里。比如你说「帮我审校」，审校Skill的全部300行才会被加载。

L3 **深度加载：Bundled Resources（脚本/引用文件/素材）**

有些Skill会附带外部资源：scripts/目录下的脚本、references/目录下的参考文件、assets/目录下的素材。比如我的审校Skill引用了个人素材库的路径。只有在AI真正需要用到这些资源的时候，才会去读取。

这个三层架构聪明的地方在于：用最小的token开销实现最大的能力覆盖。你可以装100个Skill，但平时只有元数据在上下文里，不到5000 token。只有用到的Skill才会完整加载。

> 想象你有一面巨大的书架，上面放着100本操作手册。你不会把所有手册都搬到桌上，那桌子放不下。你只是扫一眼书脊上的标题（Level 1），需要哪本就抽哪本到桌上翻开（Level 2），翻到某一页需要参考附录再去查附录（Level 3）。

条件触发：Agent怎么知道该用哪个Skill Conditional Triggering

你装了27个Skill，说了一句话，Agent怎么知道该加载哪个？

触发机制有三种，优先级从高到低：

第一种：用户显式调用。你直接说 `/weekly-report` 或者 `/huashu-proofreading` ，Agent就加载对应的Skill。这是最明确的方式，零歧义。

第二种：关键词匹配。基于description里写的触发词。你说「帮我写个周报」，里面有「周报」这个词，命中了weekly-report Skill的description里声明的触发条件。这种匹配速度快，准确率高。

第三种：语义匹配。你没说触发词，但你的意图和某个Skill的description语义接近。比如你说「这篇文章太生硬了」，没有触发「审校」「降AI味」这些关键词，但Agent通过理解你的意图，判断应该加载审校Skill。这种匹配更智能，但偶尔会判断错。

类比一下：就像手机根据你的动作自动打开对应App。你举起手机对准东西，它打开相机；你扫到一个二维码，它打开微信。你不需要先退回桌面、找到相机图标、点击打开。Agent的触发机制也是这样，它让Skill的使用变得无感。

Skill的执行模型 Execution Model

触发之后，Skill是怎么被执行的？整个流程是这样的：

几个关键细节：

理解指令阶段，AI会解析整个SKILL.md，建立对任务的整体认知。它不是逐行执行的机器人，而是先理解你的意图和约束，再规划行动。这也是为什么Skill里的示例很重要。示例帮AI建立「什么是好的输出」的标准。

规划步骤阶段，AI会把Skill里的流程拆解成可执行的步骤。如果Skill写了「先搜索、再分析、最后输出」，AI就会按这个顺序来。如果某一步需要调用MCP工具（比如搜索网页），它会在这个阶段规划好。

输出验证阶段，AI会检查自己的输出是否满足Skill里的约束。比如「不超过15字」「三栏格式」「总字数300字以内」。这个阶段不是所有Skill都需要，但对质量要求高的任务很有用。

一个反直觉的发现：好Skill不是越详细越好 The Goldilocks Zone

写了27个Skill之后，我发现一个反直觉的规律。

太短的Skill没用。你写「帮我写周报，要简洁」，AI不知道你的「简洁」是什么标准。100字算简洁还是300字算简洁？三栏式还是流水账式？没有足够的细节，AI只能猜。

太长的Skill也有问题。我曾经写过一个超过5000字的Skill，把所有边界情况都考虑到了。结果呢？AI反而执行得不好。因为关键信息被淹没在大量细节里了。上下文窗口是有限的，你塞进去的内容越多，每条信息的「权重」就越低。

最佳区间在哪里？我的经验是：**500到2000字**。

这个区间够你把核心流程写清楚，加上几个关键示例，但不会多到让AI抓不住重点。我用的最多的几个Skill，比如审校（约2000字核心内容）、选题生成（约800字）、配图（约1200字），都在这个范围内。

> 好的菜谱不会告诉你怎么握刀、怎么开火。它假设你有基本的厨房常识。它会告诉你的是：中火翻炒3分钟、盐半勺、出锅前加葱花。好的Skill也是一样：不需要教AI怎么写文章，只需要告诉它你的标准和偏好。

Token经济学 Token Economics

聊点实际的。一个Skill占多少token？多个Skill同时加载会不会打架？

先说体量。1000字的中文大约是1500-2000 token。一个典型的Skill（1000-2000字）大约占2000-4000 token。Claude的上下文窗口是100万token，单个Skill只占0.2%-0.4%。看起来微不足道，对吧？

但实际使用中你的上下文里不只有Skill。还有对话历史、系统指令、加载的文件内容。一个复杂的编程任务，光代码文件可能就占了几十万token。这时候Skill的体量就不能忽略了。

多个Skill同时加载怎么办？大多数情况下，一次对话只会触发1-2个Skill。但偶尔会有冲突的情况。比如你同时触发了两个Skill，一个说「输出简洁，300字以内」，另一个说「输出详尽，覆盖所有细节」。AI会懵。

解决方法是设计好触发条件，让Skill之间的边界清晰。就像你不会把前端代码和后端代码写在同一个文件里一样，你也不应该让两个Skill的触发条件重叠太多。

> **核心建议**
>
> 如果你发现AI执行Skill的时候行为异常，第一件事应该检查：是不是同时加载了多个Skill，而且它们的指令有冲突。用 `/skills` 命令可以查看当前加载了哪些Skill。

Skill互相打架怎么办 Handling Conflicts

Skill冲突是个真实问题。我自己就遇到过。

有一次我在写文章，同时触发了「内容创作Skill」和「审校Skill」。创作Skill告诉AI「先写完整篇，不要中途停下来修改」。审校Skill告诉AI「逐段检查，发现问题立刻改」。两个指令矛盾了，AI的输出变得很奇怪，写一段改一段，既没写完整也没改彻底。

这类问题有几种解决思路：

设计不重叠的触发条件。让每个Skill在明确的场景下触发。「写文章」触发创作Skill，「帮我改改」触发审校Skill。不要让同一个关键词同时触发两个Skill。

在Skill里声明优先级。你可以在SKILL.md里写一条规则：「如果正在执行其他写作任务，审校Skill不自动加载，等用户显式调用」。AI理解自然语言，这种约束它能遵守。

把流程拆成阶段。先完成创作阶段，再进入审校阶段。在Skill的触发条件里加上阶段判断。

原理总结 Putting It Together

回顾一下这一章讲的内容。Skills有效，是因为三个LLM特性的组合：

特性	作用	对应Skill的哪部分
指令遵循	AI会认真执行结构化指令	执行流程（Markdown正文）
上下文学习	AI通过示例理解你的标准	示例和约束条件
条件触发	在对的时间加载对的Skill	description中的触发条件

理解原理的好处是什么？当你知道AI是通过「指令遵循」来执行Skill的，你就会明白为什么具体的指令比模糊的指令效果好。当你知道「上下文窗口」是有限的，你就会控制Skill的长度。当你理解「触发机制」的三个层

级，你就能设计出不打架的触发条件。

概念篇到这里就结束了。你现在对Skills有了完整的认知：它是什么（第2章）、为什么有效（第3章）。接下来进入实战篇，从生态全景开始，到安装你的第一个Skill。

04 Skills生态全景

The Skills Ecosystem

2026年，Skills生态已经相当热闹了。但质量参差不齐，知道去哪找、怎么选，比找到多少更重要。

在写这章之前，我花了一个下午把市面上所有Skills平台都逛了一遍。结论是：**Skills的数量已经不是问题了，质量才是。**全网加起来超过100万个Skills，但真正好用的可能不到1%。

所以这一章不是给你列一份平台清单让你自己去淘金。我会告诉你每个平台的特点、适合什么场景，以及最重要的——怎么判断一个Skill值不值得装。

开放标准：agentskills.io The Open Standard

先说标准本身。2025年12月，Anthropic把Agent Skills发布为开放标准，网站是agentskills.io，官方仓库在github.com/anthropics/skills。到2026年4月，已经有20多个AI产品采纳了这个标准。

标准的核心其实非常简单：一个Skill就是一个SKILL.md文件。标准定义了三件事：

- **格式**：YAML frontmatter + Markdown正文
- **触发机制**：什么时候激活这个Skill（when字段或description匹配）
- **加载方式**：放在哪个目录、怎么被AI工具发现

就这些。没有复杂的API，没有SDK，没有编译步骤。你用记事本就能写一个Skill。这个低门槛是Skills生态能爆发的根本原因。

> 开放标准的意义在于：你写的一个SKILL.md，可以在Claude Code、Cursor、Codex、Gemini CLI等20多个产品上运行。学一次，到处用。这和MCP的思路一样——不是让每个产品自己造轮子，而是大家用同一套规范。

七大Skills平台 Seven Platforms

目前值得关注的Skills来源有七个。我按「从官方到社区」的顺序介绍。

1. Anthropic官方Skills仓库（github.com/anthropics/skills）

Anthropic自己维护的示例Skills。数量不多，但每一个都是标杆级的质量。如果你刚开始接触Skills，从这里挑两三个装上体验，是最稳的起步方式。官方仓库里的Skill，触发条件写得清晰、执行步骤可验证、上下文占用合理。它们不只是能用，而且是「这就是Skill该怎么写」的参考答案。

2. skills.sh（Vercel Labs推出）

Vercel推出的开源Agent Skills目录（The Agent Skills Directory），用 `npx skills add` 安装。界面做得很干净，搜索体验不错。注意这不是Anthropic官方的，是Vercel Labs的开源项目。

3. AgentSkill.sh

106,000+个Skills的社区市场。安装方式很方便：在Claude Code里直接输入 `/learn @owner/skill-name` 就行。社区驱动意味着更新快、覆盖面广，但也意味着质量参差不齐。有些Skill写得很精心，有些就是随手糊的一个Markdown。用之前最好先看一眼SKILL.md的内容，别无脑装。

4. SkillsMP（skillsmp.com）

目前最大的Skills市场，700,000+个Skills。支持一键安装和SKILL.md复制。数量碾压其他平台，但大量Skills是自动生成的，质量参差不齐。适合有明确需求的时候去搜索，不适合随便逛。

5. SkillHub（skillhub.club）

7,000+个Skills，数量不大但有个亮点：**AI评审机制**。每个上传的Skill都会经过自动质量评估，包括触发条件是否清晰、执行步骤是否完整、上下文长度是否合理。这让SkillHub的平均质量明显高于其他社区平台。如果你不想花时间筛选，SkillHub是个省心的选择。

6. 腾讯SkillHub中国版

1.3万+个Skills，国内最大的Skills聚合平台。它不只是翻译海外的Skills，还做了安全审计和本地化适配。对国内开发者来说，最大的价值是与腾讯系AI产品的深度集成。如果你的工作环境主要在国内，这是首选平台。

我主要用的还是手动安装——把SKILL.md直接丢进 `.claude/skills/` 目录，简单粗暴但从来没出过问题。市场上那些平台我基本是当搜索引擎用，看到好的Skill就把内容复制下来自己改改，比直接装更放心。

7. 字节跳动生态

字节的路径比较特别：不是做一个独立平台，而是把Skills能力嵌入到自己的产品矩阵里。Coze 2.0支持Agent Skills标准，ArkClaw适配飞书工作流，TRAE IDE内置了Skills管理。如果你是飞书重度用户或者在用Coze做Agent，字节生态的Skills会和你的工作环境衔接得最顺滑。

横向对比 Platform Comparison

平台	数量	质量保证	安装方式	适用产品
Anthropic官方	数十个	官方维护，最高	GitHub clone	所有支持标准的产品
skills.sh	—	Vercel维护	npx skills add	通用
AgentSkill.sh	106K+	社区自治	/learn命令	Claude Code为主
SkillsMP	700K+	无审核	一键安装/复制	通用
SkillHub	7K+	AI评审	一键安装	通用
腾讯SkillHub	1.3万+	安全审计	平台内安装	腾讯系产品
字节生态	未公开	平台审核	产品内集成	Coze/飞书/TRAE

选Skill的5个标准 Five Selection Criteria

平台再多，最终你还是要做一个判断：这个Skill值不值得装？我用了大半年Skills，总结出5个标准。按重要性排序：

1. **触发条件清晰**

   好的Skill在frontmatter的when字段或description里写得很明确：什么时候该被激活。比如「当用户提到审校、降AI味、润色时触发」。坏的Skill写一句模糊的「帮助用户写代码」，这种Skill会在不该出现的时候出现，干扰正常对话。

2. **执行步骤可验证**

   Skill里写的每一步，你应该能看到AI确实在执行。好的Skill会说「第一步搜索官方文档，第二步对比竞品数据，第三步生成对比表格」。坏的Skill写「深入分析并给出专业建议」，这种模糊指令等于没说。

3. **上下文占用合理**

   每个被加载的Skill都会占用AI的上下文窗口。**500到2000字是最佳范围**。低于500字，Skill可能太简单，直接在对话里说一句就够了，不需要专门做成Skill。超过2000字，要考虑是不是塞了太多东西，能不能拆成两个Skill。我见过5000多字的Skill，加载后直接吃掉一大块上下文，得不偿失。

4. **有用户确认环节**

   好的Skill在关键决策点会暂停，等你确认后再继续。比如选题Skill生成3个方向后让你选，而不是自己替你选一个直接开写。特别是涉及文件操作、发送消息这类不可逆的动作，没有确认环节的Skill不要用。

5 **维护活跃**

看最后更新时间和issue响应速度。AI工具的迭代非常快，一个半年没更新的Skill很可能已经和最新版的AI工具不兼容了。在AgentSkill.sh上，star数和最近更新时间是两个最有参考价值的指标。

> **核心建议**
>
> 新手最容易犯的错误是装太多Skill。记住：**每个Skill都有认知成本**。装10个质量参差的Skill，不如精选3个真正好用的。先从官方仓库和SkillHub选，用熟了再去社区淘金。

一门免费课程 A Free Course

如果你想系统地入门Agent Skills，推荐DeepLearning.AI在2026年1月发布的「Agent Skills with Anthropic」课程。免费，2小时，由Anthropic的Elie Schoppik主讲。

这门课覆盖了Skills的基本概念、创建方法和最佳实践。它不会教你做很复杂的东西，但能帮你建立正确的心智模型。对于刚接触Skills的人，这2小时的投入非常值得。

讲完了去哪找，接下来讲怎么装。

05 安装你的第一个Skill

Install Your First Skill

安装Skill比你想象的简单。最简单的方式就是把一个文件放到正确的目录。

很多人一听到「安装」就觉得要折腾环境、装依赖、跑命令。Skills不是这样的。它的本质是一个Markdown文件。你把这个文件放到指定目录，AI工具就能读到它。没有编译，没有构建，没有依赖。

但「放到哪个目录」有讲究。放错了位置，Skill不会生效；放对了位置，效果也因层级不同而不同。

Skill的四个层级 Four Levels of Skills

Skills有一个层级体系。从最广到最窄：

1. **Enterprise级（公司统一管理）**
   公司IT部门统一下发的Skills，所有员工的AI工具自动加载。比如公司统一的代码规范Skill、安全审查Skill。个人用户一般接触不到这个层级。

2. **Personal级（个人全局）**
   存放在 `~/.claude/skills/` 目录。这里的Skills对你所有项目生效。适合放通用型Skill：代码review标准、周报格式、写作审校规则。我的27个Skills里，大概有10个放在这个层级。

3. **Project级（项目专属）**
   存放在项目根目录的 `.claude/skills/` 里。只在这个项目中生效。适合放项目特定的东西：这个项目的部署流程、这个项目的API规范、这个项目的测试标准。团队协作时特别有用，把Skills跟项目代码一起提交到Git，新人clone项目就自动获得了团队的工作方式。

4. **Plugin级（IDE插件提供）**
   由IDE插件自动注入的Skills。比如某些VS Code扩展会自带一些Skills。这个层级你一般不需要手动管理。

理解层级关系的关键是优先级：**Enterprise级优先级最高，Personal次之，Project最低。**也就是说，公司统一规范会覆盖个人偏好，个人偏好会覆盖项目设置。这个设计是有道理的——如果公司规定了代码安全审查标准，你不能在项目里把它关掉。Plugin级通过命名空间隔离，不和其他层级冲突。

实际使用中，大多数人接触到的就是Personal和Project两级。我自己的经验是，通用的写作风格、代码review标准放Personal，项目特有的部署脚本、API规范放Project，基本不会打架。

用文件树来看，整个结构长这样：

```
~/.claude/
└── skills/                    # Personal级（全局）
    ├── code-review/
    │   └── SKILL.md
    ├── weekly-report/
    │   └── SKILL.md
    └── proofreading/
        └── SKILL.md

~/my-project/
└── .claude/
    └── skills/                # Project级（项目专属）
        ├── deploy-flow/
        │   └── SKILL.md
        └── api-spec/
            └── SKILL.md
```

Claude Code安装方式 Installing in Claude Code

Claude Code是目前Skills支持最完善的工具，也是我主要使用的工具。三种安装方式，从最基础到最方便：

方式一：手动安装

最笨但最可靠的方式。创建目录，把SKILL.md放进去。

```
# 项目级安装
mkdir -p .claude/skills/my-skill
# 然后把SKILL.md文件放到这个目录里

# 全局安装
mkdir -p ~/.claude/skills/my-skill
# 同样把SKILL.md放进去
```

就这样。没有第三步。下次启动Claude Code，它会自动扫描这些目录，发现新的Skill并加载。

方式二：从市场安装

AgentSkill.sh支持一条命令安装：

```
/learn @owner/skill-name
```

比如安装一个前端设计Skill：

```
/learn @anthropics/frontend-design
```

这条命令会自动把SKILL.md下载到正确的目录。省去了手动创建文件夹的步骤。

方式三：从GitHub安装

如果Skill是一个GitHub仓库，直接clone到skills目录：

```
# 项目级
git clone https://github.com/xxx/skill-name .claude/skills/skill-name

# 全局
git clone https://github.com/xxx/skill-name ~/.claude/skills/skill-name
```

这种方式的好处是可以用 `git pull` 来更新Skill。当Skill的作者发布了新版本，你拉一下代码就是最新的了。

> **全局还是项目级？** 判断标准很简单：如果这个Skill跟你的身份有关（你的写作风格、你的review标准），放全局。如果跟特定项目有关（这个项目的部署脚本、这个项目的测试规范），放项目级。拿不准的话，先放项目级，用了一段时间发现到处在用，再提升到全局。

Cursor安装 Installing in Cursor

Cursor从一开始就有 `.cursorrules` 文件的概念，现在已经纳入了Agent Skills标准。安装方式和Claude Code几乎一样：

```
# 在项目根目录
mkdir -p .cursor/skills/my-skill
# 放入SKILL.md
```

Cursor同时也兼容项目根目录的SKILL.md文件。如果你的Skill很简单（只有一个文件），直接放在 `.cursor/skills/` 目录下也行，不需要再嵌套一层子文件夹。

一个实际的好处是：如果你的项目同时有 `.claude/skills/` 和 `.cursor/skills/` ，两个工具各取所需，互不干扰。当然，如果Skill内容是一样的，你也可以用符号链接避免重复。

其他工具 Other Tools

Agent Skills标准已经被广泛采纳，以下是几个主流工具的安装方式：

工具	Skills目录	备注
OpenAI Codex	`.codex/skills/`	支持Agent Skills标准
Gemini CLI	`.gemini/skills/`	Google的CLI工具，完整支持标准
VS Code + Copilot	`.github/skills/`	通过GitHub Copilot加载
JetBrains Junie	`.junie/skills/`	原生支持，IDE内管理
TRAE（字节）	`.trae/skills/`	支持标准 + Coze集成

你会发现规律：目录结构都是 `.[工具名]/skills/skill-name/SKILL.md` 。记住这个模式，遇到新工具也能猜对路径。

安装后验证 Verify Installation

装完了怎么确认Skill真的在工作？最简单的办法：**直接说触发词，看AI是否按Skill的流程来。**

比如你装了一个审校Skill，触发词是「审校」「降AI味」。你对AI说「帮我审校一下这篇文章」，如果AI开始按三遍审校流程走（而不是笼统地给你修改建议），说明Skill已经生效了。

在Claude Code中，你还可以看到更直接的证据。当Skill被加载时，日志里会显示加载了哪些Skills。如果你不确定，可以直接问AI：「你现在加载了哪些Skills？」它会告诉你。

跟着装三个Skill Hands-on: Install Three Skills

理论够了，动手吧。下面跟着安装三个Skill，从官方到社区，体验完整的安装流程。

1. **安装官方的frontend-design Skill**

   这是AgentSkill.sh上最热门的Skill之一，教AI生成高质量的前端界面。打开Claude Code，输入：

   ```
   /learn @anthropics/frontend-design
   ```

   安装完成后试一下：说「帮我做一个登录页面」，观察AI是否按照frontend-design Skill的设计规范来生成代码，而不是给你一个普通的HTML模板。

2 **从AgentSkill.sh安装一个实用Skill**

去agentskill.sh浏览一下，找一个跟你工作相关的Skill。比如如果你经常写文档，可以试试：

```
/learn @community/doc-coauthoring
```

安装前记得点进去看看SKILL.md的内容，确认它符合前面说的5个标准。特别是触发条件和执行步骤是否清晰。

3 **手动创建一个最简单的Skill**

这一步的目的不是安装别人的Skill，而是让你感受一下Skill到底有多简单。在你的项目目录里执行：

```
mkdir -p .claude/skills/hello-skill
```

然后创建一个SKILL.md文件，内容就写两行：

```
---
description: "当用户说「打个招呼」时触发"
---
# Hello Skill
用中文跟用户打招呼，然后告诉他今天是星期几。
```

保存，重新打开Claude Code，说「打个招呼」。如果AI用中文跟你打招呼并且告诉你星期几，恭喜，你的第一个手写Skill已经在工作了。

> **核心建议**
>
> 这个Hello Skill当然没什么实际用处。但它演示了一件关键的事：**创建一个Skill的门槛低到令人意外**。就是一个Markdown文件，加一个YAML头部。后面的章节会教你怎么把这个能力用到真正有价值的场景上。

常见安装问题排查 Troubleshooting

装了Skill但不生效，90%的原因出在以下三个地方：

问题一：Skill不生效

先检查文件路径是否正确。最常见的错误是把SKILL.md直接扔在 `.claude/skills/` 下面，没有创建子文件夹。正确的结构是 `.claude/skills/skill-name/SKILL.md` 。另外确认文件名——标准推荐使用**SKILL.md**（大写），虽然有些工具对大小写不敏感，但用大写是最稳的选择。

问题二：触发不了

Skill加载了，但你说了触发词它也不按Skill的流程走。检查SKILL.md里的description或when字段，看看触发条件是否和你说的话匹配。比如你的触发条件写的是「当用户要求代码review时」，但你说的是「帮我看看这段

代码」，AI可能没把这两者关联起来。把触发条件写得更具体、覆盖更多同义表达会有帮助。

问题三：Skill之间冲突

装了两个Skill，触发条件有重叠，AI不知道该用哪个。比如你有一个「写作审校」Skill和一个「降AI味」Skill，用户说「帮我审校」时两个都被激活了。解决方法是让触发条件更精确，或者把功能相近的Skill合并成一个。

> **注意**
>
> 一个容易被忽略的问题：SKILL.md的YAML frontmatter格式必须严格正确。三个短横线开头，三个短横线结尾，中间是合法的YAML。少一个短横线或者缩进不对，整个Skill都不会被识别。如果一切看起来都对但就是不生效，把frontmatter单独拿出来用YAML验证器检查一下。

装好了Skill，下一步就是真正用起来。第6章会讲Skills在实际工作中的使用方法和技巧。

06 用好Skills的实战技巧

Skills in Action

安装Skill只需要一行命令，但让它真正融入你的工作流，需要一些技巧。

我见过不少人装了一堆Skills，然后跟我说「没什么用啊」。聊完才发现，他们从来没成功触发过其中大部分Skill。这就像你手机装了50个App，但每天只用微信和相机。

Skill装上了只是开始。**会触发、会组合、会排错**，才算真正用起来了。

三种触发方式 How to Trigger Skills

触发一个Skill有三种方式，从显式到隐式：

1. **强制触发：斜杠命令**
   最直接的方式。在对话里输入 `/skill-name`，AI就会加载对应的Skill并执行。比如我输入 `/huashu-proofreading`，审校Skill立刻启动，不管当前对话在聊什么。

2. **显式触发：说触发词**
   每个Skill在SKILL.md里都定义了触发条件（写在description字段里）。比如我的审校Skill写的是「当用户提到审校、降AI味、太AI了、润色」时触发。所以我只要在对话里说「帮我审校一下这篇文章」，AI就会自动调用它。不需要记命令名，用自然语言就行。

3. **隐式触发：AI自动判断**
   这是最优雅的方式。你不需要说任何触发词，AI根据对话上下文自己判断该用哪个Skill。比如我刚写完一篇初稿说「看看这篇文章」，AI看到上下文里有一篇刚完成的草稿，就自动启动审校流程。这需要Skill的触发条件写得足够精准。

实际使用中，我80%的时间用显式触发，15%用强制触发，只有5%靠隐式触发。不是隐式不好用，而是**显式触发的确定性最高**。你说了触发词，Skill一定会启动。隐式触发有时候灵，有时候不灵，取决于AI对上下文的理解。

Skill不生效？四个常见原因 Why Skills Fail to Trigger

「我明明装了这个Skill，为什么没反应？」这是我被问得最多的问题。排查下来，原因通常就这几个：

触发词不匹配。你说的是「润色」，但Skill的description里写的是「审校」。虽然意思差不多，但AI是按照SKILL.md里的描述来匹配的。解决办法很简单：打开SKILL.md，在description的触发条件里多加几个同义

词。我的审校Skill就列了十几个触发词：审校、降AI味、太AI了、AI检测率、没人味、像AI写的、自然一些、润色、polish。宁可多写，不可漏掉。

多个Skill抢同一个触发词。你装了两个写作相关的Skill，都把「写文章」设为触发条件。AI不知道该用哪个，可能随机选一个，也可能两个都不用。解决办法是让触发条件更具体：一个改成「写公众号文章」，另一个改成「写小红书笔记」。

上下文太长，Skill被挤出去了。这个比较隐蔽。AI的上下文窗口是有限的。如果你在一个很长的对话里装了几十个Skill，SKILL.md的内容会占据大量上下文空间，到后面AI可能「忘了」某些Skill的存在。解决办法：把不常用的Skill从全局配置移到项目级配置，只在需要的项目里加载。

Skill本身写得不够好。有些Skill的指令太模糊，AI理解了触发词，但不知道具体该做什么。这个就需要回到Skill的编写质量上去优化了，第7章会详细讲。

> **核心建议**
>
> 排查Skill不生效时，最快的方法是先用斜杠命令强制触发。如果强制触发能正常工作，说明Skill本身没问题，是触发条件的匹配出了问题。如果强制触发也不行，那就是Skill的指令有问题。

组合使用：让Skills像流水线一样串联 Chaining Skills Like a Pipeline

Skills之间不是孤立的。我日常使用中，最强大的模式是把多个Skills串联起来，前一个的产出作为后一个的输入。

关键在于：**用文件系统作为Skills之间的传送带。**

我的每一个Skill都会把产出写入文件。选题Skill把方向写进一个md文件，调研Skill把简报追加到调研文件里，写作完成后初稿存为草稿文件，审校Skill从草稿文件读取、修改后覆盖写回。整个过程中，文件就是Skills之间的接口。

这个设计不是偶然的。AI对话是有上下文限制的，但文件不会丢失。即使对话中断了、换了一个新会话，只要文件还在，下一个Skill就能从断点继续。

这种模式还有一个好处：**每一步都有存档。**如果审校后觉得改得不好，我可以回退到审校前的版本。如果调研Skill搜到的信息有误，我可以单独修改调研文件，不影响其他步骤的产出。

实战案例：审校环节拆解 Case Study: Proofreading in Action

第一章介绍了一篇公众号文章从选题到分发的7步流程。这里我想展开讲其中最关键的一步：审校。

为什么说审校最关键？因为它是「AI写的内容」和「花生写的内容」之间的最后一道关卡。初稿再好，如果AI味没降下来，读者一眼就能看出来。

还记得第2章详细介绍的三遍审校流程吗？查事实、降AI味、磨细节，加上那份6大类AI腔识别清单。这里不重复了，直接看它在实际工作中的效果。

审校前，AI写出来的段落是这样的：

> **审校前（AI味重）：**
> Claude Code是一个非常强大的AI编程工具。它能够帮助开发者高效地完成各种编程任务，从简单的代码补全到复杂的架构设计，都能够轻松应对。总的来说，它代表了AI辅助编程的未来方向。

问题在哪？「非常强大」「高效地」「各种」「轻松应对」「总的来说」「未来方向」，每一个词都是AI味的标志。没有任何具体细节，换成任何一个AI工具都成立。

经过审校Skill处理后：

> **审校后（花生风格）：**
> Claude Code刚出来的时候我没太当回事，又一个AI编程工具嘛。直到有一天我让它帮我重构一个React组件，它不光改了代码，还顺手把我写的屎山测试用例也重写了。那一刻我意识到，这玩意儿不是代码补全，是真的在理解代码。

变化在哪？有了时间线（「刚出来的时候」「有一天」），有了具体场景（「重构一个React组件」），有了真实反应（「没太当回事」「那一刻我意识到」），有了口语化表达（「这玩意儿」「屎山」）。**这就是Skill记住了花生的写作方式后，能做到的事。**

实战案例：用Slides Skill做演示文稿 Case Study: Making Slides

我的huashu-slides Skill支持17种视觉风格，从极简主义到瑞士国际主义，从杂志编排到日式侘寂。但这些风格不是让你从17个里盲选一个的。

实际使用流程是这样的：我跟AI说「帮我把这份内容做成slides」，Skill会先分析内容类型和场景（是技术分享还是商业路演？是线下培训还是线上传播？），然后**推荐3个最合适的设计方向**，每个方向有名字、视觉描述和适用理由。我选一个方向后，Skill才开始制作。

制作时有两条路径可以选：

AI插画路径：每一页的背景和插图都用AI生成。优点是视觉质感好，看着像设计师做的。缺点是生成时间长，而且后期修改不方便，想换个文字就得重新生成整张图。

HTML路径：用HTML+CSS构建slides，Playwright截图导出为图片。优点是完全可编辑，改个字重新截图就行。缺点是视觉表现力受限于CSS能做到的程度。

我个人的偏好是：**重要场合用AI插画路径，日常分享用HTML路径。**这个偏好也写在了Skill里，所以AI每次都会提醒我选路径，而不是自己默默选一条。

> **注意**
>
> Slides Skill有一个重要规则：**不能混用两条路径。**要么全部AI插画，要么全部HTML。混用会导致风格不统一，一眼假。这个是我踩了几次坑之后加进Skill的硬性约束。

实战案例：用Data Pro做数据分析 Case Study: Data Analysis

huashu-data-pro是我做内容创作之外用得最多的Skill。它的特点是**多专家并行分析**。

什么意思？当你丢给它一份CSV或Excel文件时，它不是用一个视角去分析，而是同时启动多个「专家角色」：一个看数据趋势，一个看异常值，一个做对比分析，一个提业务建议。这些分析并行进行，最后整合成一份HTML报告。

我用它最多的场景是**B站UP主数据分析**。把90天的视频数据导出为CSV，丢给这个Skill，它会告诉我：哪些选题的完播率最高、哪个时间段发布互动最好、评论区的高频关键词是什么、跟上一个周期比涨了还是跌了。

另一个场景是**投放ROI分析**。把广告投放数据扔进去，它会算出每个渠道的获客成本、转化漏斗的流失点、以及跟行业平均水平的对比。输出不只是数字和图表，还有一段文字版的分析摘要，可以直接复制到周报里。

这个Skill让我明确感受到了一件事：**Skill的价值不在于AI做了你做不了的事，而在于AI用你习惯的方式做了你不想重复做的事。**每次手动分析B站数据，我也能做，但要花两三个小时。有了Skill，十分钟出报告。

常见问题排查清单 Troubleshooting Checklist

安装层面的问题（路径不对、YAML格式错误）第5章已经讲过了。这里聚焦的是**使用中的问题**：Skill装好了但没触发、触发了但效果不对。

用了一段时间Skills之后，你大概率会遇到下面这些问题。我把排查方法整理在这里，遇到了直接对照：

问题	症状	排查方法
Skill不生效	说了触发词但没反应	1. 用 `/skill-name` 强制触发测试 2. 检查SKILL.md的description是否包含你用的触发词 3. 检查文件路径是否正确
多Skill冲突	触发了错误的Skill	1. 让两个Skill的触发条件更具体，减少重叠 2. 用强制触发绕过自动匹配
上下文溢出	对话后期Skill不工作了	1. 把不常用的Skill从全局移到项目级 2. 开新对话，不要在超长对话里连续用 3. 精简SKILL.md内容，去掉冗余描述
输出不稳定	同一个Skill每次产出不一样	1. 在Skill中加入更具体的格式要求和示例 2. 用「必须」「始终」等强约束词 3. 提供一个输出模板，让AI照着填
产出质量低	Skill能触发但结果不满意	1. 检查Skill里的指令是否足够具体 2. 是否缺少示例（AI看到好例子才知道什么算好） 3. 考虑拆成多个步骤，而不是一步到位

> **核心建议**
>
> 如果你只记一条排查原则，记这个：**先确认Skill能被触发，再优化Skill的产出质量。**这两个是完全不同的问题。触发问题改description里的触发条件，质量问题改指令内容。不要混在一起排查。

到这里，你已经知道怎么触发Skills、怎么串联使用、怎么排查问题了。但我们一直在用别人写好的Skill。接下来的三章，我们进入最有意思的部分：**自己创建Skill。**

门槛比你想象的低得多。毕竟，Skill是用自然语言写的。你能写一段话教别人做事，你就能写一个Skill教AI做事。

07 创建你的第一个Skill

Build Your First Skill

把重复的工作变成技能，不需要写一行代码。

我认识不少人，每天用AI干活，但每次都在重复同样的指令。写周报要说一遍格式，做会议纪要要说一遍模板，审校文章要重新解释一遍禁忌词。他们不是不会用AI，而是没意识到一件事：**你脑子里那套「我每次都这样做」的流程，是可以固化下来的。**

创建Skill的本质，就是把你的隐性知识变成显性文档。你不需要会编程，不需要懂YAML语法，你只需要能把自己的工作流程讲清楚。有一个很好的判断标准：如果你要教一个刚入职的新人做这件事，你会怎么说？那些话，就是Skill的内容。

发现你的隐性知识 Discover Your Tacit Knowledge

花3分钟想想，你每天重复做的事情里，哪些是有固定模式的？

比如你每周五写周报。你可能觉得这没什么模式，不就是把这周干的事情列一列嘛。但仔细想想：你是不是总会先翻一遍聊天记录和日历？是不是会按项目分类而不是按时间排列？是不是结尾总会写下周计划？是不是老板特别在意数据指标所以你每次都会把关键数字加粗？

这些「是不是」，就是你的隐性知识。你做了上百次，已经变成了肌肉记忆，自己都意识不到了。但AI不知道这些。每次你不说，它就按自己的理解来，然后你又要改。

Skill做的事情，就是把这些肌肉记忆写下来，让AI也能拥有。

SKILL.md完整结构详解 Anatomy of a SKILL.md

我们直接看一个完整的例子。假设你经常需要把会议录音整理成纪要：

```
---
name: meeting-notes
description: |
  将会议录音/文字转为结构化会议纪要。
  当用户提到「会议纪要」「会议记录」「meeting notes」,
  或分享了会议录音、转写文本时使用此Skill。
---

# 会议纪要生成

## 执行步骤

1. 确认输入：录音文件还是文字转写?
2. 提取关键信息：
   - 会议主题和日期
   - 参会人员
   - 讨论的核心议题（不超过5个）
   - 每个议题的结论
   - Action items（谁、做什么、deadline）
3. 按模板整理：
   - 标题：【会议纪要】主题 - 日期
   - 正文：议题 → 结论 → Action Items
   - 总字数控制在原文的20%以内
4. 写入文件并确认

## 注意事项
- 不要遗漏任何action item
- 人名使用全名
- deadline用具体日期，不用「尽快」「下周」
```

这就是一个完整的SKILL.md。拆解一下每个部分：

name：简短英文名，用于识别。就像文件名一样，简洁明了就好。

description：一句话说清这个Skill做什么。这行非常重要，因为AI会用它来判断是否需要加载这个Skill。写得太模糊，AI可能在该触发的时候没触发；写得太窄，又可能漏掉一些场景。

description中的触发条件：注意我们把触发条件直接写在了description里，而不是用单独的when字段。这是因为AI用description来判断是否加载这个Skill，把触发条件和功能描述放在一起，匹配更准确。触发条件越具体越好，不要写「用户需要整理文档时」，要写「用户提到会议纪要、meeting notes，或分享了录音文件」。

正文：Markdown格式的执行指令。这是Skill的核心，写你希望AI怎么一步步完成这个任务。

注意事项：边界条件和禁区。那些你被坑过的经验教训，在这里写下来。比如「人名用全名」这条，大概率是因为之前AI把「王总」写成了纪要里的称呼，导致发给其他部门的人看不懂。

从0写一个Skill：手把手教程 Step-by-Step Tutorial

概念讲完了，我们实际动手。以「周报生成」为例，走完整个流程。

1. **回忆你的流程（花3分钟）**

   闭上眼想想你上周五是怎么写周报的。从打开文档的那一刻开始。你先看了什么？然后做了什么？格式有什么要求？老板最关心什么？有没有什么坑你踩过？把这些全部记下来，不需要整理，流水账就行。

2. **把流程写成自然语言步骤**

   整理你的流水账。你会发现大概可以分成几个阶段：收集信息、分类整理、撰写正文、检查提交。把每个阶段拆成具体步骤，用「先...然后...最后...」的句式。不用管格式，能看懂就行。

3. **补充格式要求和注意事项**

   你的周报有固定格式吗？「本周完成」「下周计划」「需要协调」这种分区？有字数限制吗？有没有什么雷区，比如不能提还没确认的项目？把这些补上。

4. **填写frontmatter**

   给Skill起个名字，写一句话描述，列出触发条件。触发条件想想你平时怎么跟AI说的：「帮我写周报」「整理一下这周的工作」「周报」。

5. **保存到正确位置**

   把文件保存到 `.claude/skills/weekly-report/SKILL.md` 。注意路径结构： `.claude/skills/` 是固定前缀， `weekly-report/` 是你的Skill文件夹名， `SKILL.md` 是固定文件名。

6. **测试触发**

   打开一个新对话，说「帮我写周报」。如果Skill被正确加载，AI会按你定义的流程来执行，而不是自由发挥。如果没触发，检查一下description里的触发条件是不是太窄了。

整个过程不超过15分钟。你花15分钟写一次，以后每周都能省15分钟。一年下来就是12个小时。而且你写的Skill可以分享给团队里的其他人，他们也不需要重新踩你踩过的坑。

5个设计原则 Five Design Principles

写一个能用的Skill不难。写一个好用的Skill，需要一些经验。我从自己的27个Skill里总结了5个原则，都是踩过坑之后才明白的。

原则1：先确认再动手 Confirm Before Acting

我的选题生成Skill，每次都会先给出3-4个方向，等我选了一个才继续往下走。这不是多此一举，这是血的教训。

早期我写过一个版本，AI分析完brief直接就开始写初稿了。写了2000字我才发现角度完全不对。那2000字全白费了，而且AI的语境已经被「错误的方向」污染了，即使纠正也很难写出好东西。

推荐

生成3个选题方向 → 用户选择 → 开始写

不推荐

AI自己挑了一个方向 → 直接写了2000字 → 用户说不对

重要决策必须等用户点头。这条原则写进Skill里，AI就不会自作主张了。

原则2：边做边存 Save As You Go

长会话随时可能截断。网络波动、Token用完、手滑关了浏览器，都有可能。如果所有产出都只存在对话里，一断就全没了。

我的调研Skill有一条规则：每搜到一批信息就追加写入文件。不是搜完了一起写，是边搜边存。这样即使会话中断，已经搜到的内容都还在文件里，下次可以接着来。

> **核心建议**
>
> 在Skill里加一条「每完成一个阶段就保存到文件」。这条规则看起来简单，但能帮你避免无数次心态崩溃。

原则3：模块化可组合 Modular and Composable

一个Skill只做一件事。

我早期犯过一个错误，把「选题+调研+初稿+审校+配图」全塞进一个Skill里。结果那个SKILL.md有4000多字，AI加载后上下文就很拥挤了，执行效果反而变差。而且我只想做调研不想写初稿的时候，还得去跟AI解释「跳过后面的步骤」。

后来拆成了5个独立Skill，反而灵活得多。需要哪个用哪个，也可以串联起来用。就像Unix的管道哲学：每个工具做好一件事，通过组合实现复杂功能。

推荐

「选题生成」「深度调研」「审校」分成三个Skill，各司其职

不推荐

一个巨大的「写文章」Skill包揽一切，臃肿且不灵活

原则4：给选择不给答案 Offer Choices, Not Answers

提供3个方案让用户选，而不是直接给1个。

这不只是用户体验的问题，这也是降AI味的关键。当你参与了决策过程，最终的产出就不完全是AI的，而是你和AI共同的。你选了标题B而不是A，你决定用对比的角度而不是教程的角度，这些选择让最终的文章带上了你的判断。

用户有最终决策权，Skill是顾问不是老板。这个定位很重要。

原则5：放大你，而不是替代你 Amplify, Don't Replace

Skill是放大器，不是替代者。

我的审校Skill会标记所有可能有问题的地方，给出修改建议，但最终改不改由我决定。它可能标记了15处，我看完觉得其中8处确实该改，另外7处是AI过度敏感了。如果让它自动改，那7处就会被「改坏」。

好的Skill让你做得更好更快，但关键判断始终在你手里。这也是为什么Skill不是「自动化脚本」。自动化脚本是：输入→处理→输出，人不参与。Skill是：输入→提案→人决策→执行→人确认。人在每个关键节点都在场。

创建后的维护 Maintenance

Skill不是写完就一劳永逸的。用了一段时间，你一定会发现某些步骤是多余的，或者缺了什么。

我的审校Skill到现在已经迭代了十几个版本。最早只有一遍审校，后来发现一遍不够，拆成了三遍。再后来增加了AI腔识别清单，因为发现有些AI味很重的表达光靠「感觉」是抓不全的。又后来加了传播力审查，因为文章写得好不等于传播得好。

建议用git跟踪Skill的变化。这样你可以看到一个Skill是怎么从粗糙变成精细的，也可以在某个版本效果不好时回退。

> **核心建议**
> 当一个Skill超过3000字时，考虑拆分成两个。Skill太长，AI的执行准确率会下降。短而精的Skill比长而全的Skill好用得多。

到这里，你已经具备了创建Skill的全部知识。但你可能还是觉得，从零开始写有点没方向。下一章我们介绍一个官方工具，它能帮你快速起步，甚至自动优化你的Skill。

08 用skill-creator自动创建Skills

The Skill Creator Tool

用Skill来创建Skill，这件事本身就很meta。但它真的好用。

上一章我们手写了一个Skill。手写的好处是你对每一行都有控制权，坏处是你可能不知道自己写的东西好不好。触发条件够精确吗？执行步骤有遗漏吗？注意事项覆盖了边界情况吗？

Anthropic官方的skill-creator解决的就是这个问题。它是一个meta-skill，用Skill来创建、评估和改进Skill。2025年发布，2026年3月有过一次重大更新。核心理念是：**把软件工程的严谨性引入Skill创作。**

写代码有单元测试、有Code Review、有CI/CD。写Skill为什么不能有？skill-creator就是Skill的质量工程体系。

四种模式 Four Modes

skill-creator有四种工作模式，分别解决Skill生命周期中的不同问题。

Create模式：从描述到成品 Create Mode

你用自然语言描述需求，skill-creator帮你生成完整的SKILL.md。

但它不是拿到描述就直接生成。它会先问你几个澄清问题，确保理解你真正想要的。这个过程很像一个好的产品经理在做需求澄清。

比如你说：「我需要一个Skill来做代码review。」

skill-creator可能会追问：

- 你的代码是什么语言为主？
- review的重点是什么？性能、安全、可读性，还是全都要？
- 有没有团队的编码规范需要遵守？
- review的产出是什么格式？内联注释还是独立报告？
- 有没有什么绝对不能做的事？比如自动修改代码？

这些问题帮你厘清了很多你可能没想到的细节。回答完之后，它生成的SKILL.md会比你自己从零写的更完整、更严谨。

> Create模式特别适合一种场景：你知道自己需要什么Skill，但不知道从哪里开始写。让skill-creator给你一个起点，然后在这个基础上修改，比从空白文件开始高效得多。

Eval模式：给你的Skill打分 Eval Mode

这是我觉得最有价值的模式。

Eval模式的做法是：生成测试用例，然后分别在「有这个Skill」和「没有这个Skill」两种情况下让AI执行，对比产出差异。用assertions来评分，看Skill到底带来了多少提升。

我第一次用Eval模式去评估我自认为「已经很好了」的审校Skill时，结果很打脸。有些测试用例里，有Skill和没Skill的产出差异很小，说明Skill的指令不够具体，AI光靠自己的通识能力也能做到差不多的效果。这就逼着你去想：我这个Skill到底教了AI什么它原本不会的东西？

自己写的东西自己很难看出问题，这和写文章一个道理。Eval模式相当于一个严格的评审者，帮你发现盲点。

Improve模式：自动改进 Improve Mode

Eval告诉你哪里不好，Improve帮你改好。

它读取Eval的评估结果，针对薄弱环节进行定向改进。改完之后你可以再运行一次Eval，看产出质量是否真的提升了。这就形成了一个「评估 → 改进 → 再评估」的循环。

通常2-3轮循环后，Skill的质量会有明显提升。但要注意，不是所有维度的分数都需要追到满分。有些Skill天然就是简单的检查清单，不需要复杂的执行流程。追求不必要的复杂度反而会降低Skill的可用性。

Benchmark模式：统计验证 Benchmark Mode

Benchmark模式解决的是另一个问题：**Skill的触发精度。**

它的做法是优化Skill的description字段。具体来说，它会生成各种触发场景的查询语句，测试Skill是否在该触发的时候触发、不该触发的时候保持沉默。通过反复调整description的措辞，提升触发的精准度。

对于大多数个人用户来说，Benchmark模式不是必须的。但如果你发现自己的Skill经常在不该触发的时候乱入，或者该触发的时候没反应，跑一轮Benchmark能帮你把description调到比较理想的状态。

实操：完整流程 Full Walkthrough

把四种模式串起来，看一个完整的工作流。假设你要创建一个「PR描述生成」的Skill。

1. **描述需求**

   告诉skill-creator：「我需要一个Skill，在我提交Pull Request的时候自动生成PR描述。要包含改动摘要、影响范围、测试方案。我们团队用的是TypeScript，PR描述用英文。」

2 **回答澄清问题**

skill-creator追问了：PR模板有固定格式吗？要不要自动关联Jira ticket？描述的详细程度怎么把控？你逐一回答，帮它理解你的真实需求。

3 **生成初版SKILL.md**

基于你的回答，skill-creator生成了一份完整的SKILL.md。包括frontmatter、执行步骤、输出模板、注意事项。你快速浏览一遍，看看有没有明显的偏差。

4 **运行Eval**

对生成的Skill运行评估。它生成了几个测试用例，对比有Skill和没Skill时的产出。结果发现：触发精度不错，但有些步骤太笼统导致产出和baseline差异不大，而且没考虑到monorepo的情况。

5 **运行Improve**

针对评估中发现的薄弱环节自动改进。执行步骤被拆得更细了，增加了monorepo场景的处理逻辑，边界情况补上了「如果改动太大怎么办」「如果只改了配置文件怎么办」。

6 **再次Eval验证**

重新评估：这次有Skill和没Skill的产出差异明显拉大了，monorepo场景也能正确处理了。满意了，保存使用。

整个过程大概10-15分钟。你得到的是一个经过「创建 → 评估 → 改进 → 验证」闭环的Skill，质量比纯手写高了一个档次。

自动化优化：hill-climbing Automated Hill-Climbing

如果你觉得手动跑Eval→Improve循环还是太麻烦，可以更进一步：让整个循环自动跑。

我受Karpathy的autoresearch启发，自己做了一个叫huashu-auto-skill-optimizer的Skill。注意这不是官方skill-creator的功能，是我单独做的工具。它和skill-creator的Eval模式思路不同：它用一套7维度的rubric（触发精度、步骤完整性、边界覆盖、产出格式等）给Skill打分，然后自动改进低分维度，改完再打分，如果分数上去了就保留，没上去就回退。

这本质上是一个hill-climbing（爬山算法）优化过程。每一步都尝试往更好的方向走，走不动了就停。

实际效果：有些Skill经过3-4轮自动优化后，评分从6-7分提升到了8-9分。改进主要集中在边界处理和执行步骤的精确度上，这些地方确实是人容易忽略但对执行质量影响很大的。

> **注意**
>
> 自动优化不是万能的。它擅长改进结构性的问题（步骤遗漏、边界缺失），但不擅长改进「品味」层面的东西（比如审校Skill里「什么算AI味」这种判断标准）。品味需要人来定义。

什么时候手写、什么时候用skill-creator When to Use Which

不是所有场景都需要skill-creator。我的经验是：

场景	推荐方式	原因
简单的检查清单型Skill	手写	10分钟就能写完，用skill-creator反而更慢
复杂的多阶段流水线	skill-creator + 手动调优	自动生成框架，人工调整细节
已有Skill的优化	Eval模式必用	发现自己看不到的盲点
完全没思路的新Skill	Create模式起步	让它帮你破冰，比盯着空白文件强
团队共享的关键Skill	完整闭环（Create→Eval→Improve→Benchmark）	影响多人，值得花时间做质量保证

我个人的习惯是：新Skill先手写一个粗糙版本，用几天，积累了实际使用中的问题后，再用Eval模式评估，用Improve模式改进。因为很多问题是你坐在那里想不到的，得实际用了才会暴露。

> **skill-creator最大的价值不是帮你写Skill，而是帮你看到自己的Skill哪里有问题。**Eval模式对我来说是使用频率最高的模式，远超Create和Improve。

一个思维框架 A Mental Model

skill-creator让我意识到一件事：Skill的创作过程和软件开发非常像。

写Skill就像写代码。Create模式是scaffold（脚手架），帮你生成项目骨架。Eval模式是测试，告诉你哪里有bug。Improve模式是重构，让代码更健壮。Benchmark模式是触发精度调优，确保Skill在该出现的时候出现。

区别在于，Skill用的是自然语言而不是编程语言。但「写清楚、测仔细、改到位」这套方法论是通用的。

如果你是开发者，你已经具备了写好Skill的底层能力，只是需要把「对代码的严谨」迁移到「对自然语言指令的严谨」上。如果你不是开发者，skill-creator就是你的辅助轮，帮你跳过最难的冷启动阶段。

> **核心建议**
> 下一章我们会进入更高级的领域：5种Skill设计模式。这些模式来自对大量真实Skill的抽象和总结，能帮你在更复杂的场景下设计出更强大的Skill。

09 高级Skills设计模式

Advanced Design Patterns

建筑有哥特式、巴洛克式，代码有单例、观察者，Skill也有自己的设计模式。

写了27个Skills、观察了社区里大量别人的Skills之后，我发现一件事：**所有好用的Skill都能归入5种基本模式。**就像乐高积木只有几种基础形状，但你可以拼出任何东西。

这一章不是教你照搬模板。模式的价值在于，当你面对一个新需求时，不用从零开始想「这个Skill该怎么写」，而是能快速判断：这属于哪种模式？然后在成熟的结构上做调整。

5种模式分别是：检查清单型、多方案选择型、多阶段流水线型、外部API集成型、多Agent协作型。大多数实际的Skill是其中两三种模式的组合。先掌握单一模式，组合就是自然的事。

模式1：检查清单型 Checklist Pattern

这是最直觉的模式。你有一份清单，AI逐项检查，标记问题，给出修改建议。审校、代码review、合规检查、质量验证，都是这个模式。

拿我的huashu-proofreading（三遍审校Skill）来说。第2章已经完整介绍过它的6大类AI腔识别清单和三遍审校流程，这里不重复了。关键是看它作为检查清单型Skill的设计特点。

这份清单的颗粒度是关键。它不是写「检查语言是否自然」这种正确的废话，而是直接列出具体要删除的词汇和要标记的句式。**清单条目越具体，AI执行得越准确。**

另一个关键设计：分三遍执行，每遍聚焦不同维度。为什么要分三遍？因为如果把所有任务塞进一次检查，AI会顾此失彼，重要问题淹没在细枝末节里。

检查清单型的设计技巧

1. **条目要具体到可执行**

   「不要用套话」是无效指令。「删除以下词汇：说白了、简单来说、换句话说、不可否认、毫无疑问」才是有效指令。AI需要明确的判断标准，不是模糊的方向。

2. **分优先级，先查致命问题**

   事实错误比用词不当严重十倍。如果一篇文章里把发布日期写错了，语言再优美也是废品。让AI先处理最重要的问题，再处理锦上添花的部分。

3 **检查结果要带动作**

只标记「这里有问题」不够，还要给出修改建议。比如不只是说「这句话AI味重」，而是直接给出改写后的版本。AI的工作不是挑毛病，是帮你解决问题。

模式2：多方案选择型 Options Pattern

这个模式的核心思想是：**AI不做决策，AI做分析，决策权在你手上。**

适用场景很广：选题方向、设计风格、命名方案、策略选择。任何需要「从几个选项中挑一个」的时刻，都适合这个模式。

我的huashu-topic-gen（选题生成Skill）是个典型案例。每次生成3-4个选题方向，每个方向包含：标题、核心角度、大纲、工作量评估（用⭐评级）、优劣分析。它还会覆盖4种不同类型：深度评测型、实战教程型、洞察观点型、案例拆解型。

关键在「优劣分析要诚实」这一点。早期我的Skill会把每个选题都夸一遍，「这个角度新颖，容易引发讨论」「这个实用性强，转化率高」。后来我加了一条规则：**每个方案必须有至少一条明确的劣势**。比如「这个选题写起来需要大量实测数据，预计工作量是其他方案的2倍」或「这个角度已经有3篇同类文章，差异化空间有限」。

只有提供真实的差异对比，选择才有意义。如果每个方案都说「效果好」，那等于什么都没说。

多方案选择型的设计技巧

1 **方案数量3-4个最佳**

2个太少没选择感，5个以上会产生决策疲劳。心理学上有个说法，人在面对太多选项时反而会拒绝选择。3-4个是最舒服的区间。

2 **方案之间要有真正的差异**

不是同一个想法的三种不同表述，而是三个真正不同的方向。一个偏深度分析、一个偏实战教程、一个偏观点输出，让用户在「类型」层面做选择，而不是在「措辞」层面做选择。

3 **加上现实约束的评估**

工作量评估、时间预估、资源需求。有些方案理论上很好但执行成本太高，帮用户在理想和现实之间做平衡，这是AI能提供的真正价值。

模式3：多阶段流水线型 Pipeline Pattern

这是5种模式中最复杂的一种，也是处理端到端工作流的唯一靠谱方式。

核心结构很直白：把大任务拆成多个阶段，每个阶段有明确的输入和输出，阶段之间用检查点分隔。用户确认后才进入下一阶段。

我的huashu-book-pdf（书籍级PDF制作Skill）是这个模式的极端案例。你现在正在读的这本橙皮书，就是用它做的。整个流水线分5个阶段：

第一阶段创建PROJECT.md、确定章节结构。第二阶段多Agent并行写作，每章一个HTML片段文件。第三阶段用build.js把所有片段合并，CSS渲染样式。第四阶段管理version.json和CHANGELOG.md。第五阶段输出PDF、EPUB和微信读书格式。

这里面有一个关键设计：**每个阶段的产出都必须写入文件**。第7章讲过「边做边存」的教训，流水线型Skill把这个原则发挥到了极致。每一步都写入文件，即使会话中断了，你随时可以说「上次做到第3阶段了，从那里继续」，AI读取文件就能恢复上下文。

流水线型的设计技巧

1. **每个阶段必须有文件产出**
   调研阶段输出调研笔记md，写作阶段输出HTML片段，构建阶段输出合并后的完整文件。不要让任何重要信息只存在于对话上下文中。

2. **阶段之间用检查点分隔**
   完成一个阶段后向用户汇报进度，等确认后才继续。这不是为了走流程，而是因为越早发现方向偏差，修正成本越低。写了5章发现大纲有问题，比写了1章就发现要痛苦得多。

3. **允许从任意阶段恢复**
   Skill的开头应该检测当前状态：「项目文件夹是否已存在？写到第几章了？」然后从断点继续，而不是每次都从头开始。这需要在Skill里明确写清楚状态判断的逻辑。

4. **进度可视化**
   「已完成3/5阶段」「当前：第7章写作中（共10章）」。让用户随时知道整体进度在哪里。对于大型任务，心理上的掌控感和实际效率一样重要。

模式4：外部API集成型 Integration Pattern

有些Skill需要调用外部服务：发消息到飞书、上传图片到图床、调用AI生图API。这类Skill的挑战不在于业务逻辑，而在于**错误处理和边界情况**。

我的huashu-feishu（飞书集成Skill）覆盖了发消息、创建文档、写入知识库、上传文件、图片嵌入这些功能。听起来很直接，但实际做的时候发现到处是坑。

举个例子：往飞书文档里插入图片。你可能以为是一步操作，实际上需要三步。第一步，创建一个空的image block；第二步，上传图片文件拿到file token；第三步，用PATCH请求把image block里的图片替换成上传的那

张。少了任何一步都不行，而且这三步的调用顺序不能乱。

> **注意**
>
> 我把这个「图片三步法」写进了Skill里，因为如果不写，AI每次都会尝试用一步搞定，然后报错，然后我解释一遍，然后它才做对。这种重复性的坑，正是Skill最该记录的东西。

另一个关键设计：**认证信息从环境变量读取，永远不硬编码。**这不只是安全考虑。Skill文件是可能分享给别人的，如果里面写死了你的API密钥，分享出去就是泄露。环境变量让Skill保持通用，每个人配自己的密钥就行。

集成型的设计技巧

1. **密钥永远从环境变量读取**

   在Skill里写清楚需要哪些环境变量（比如FEISHU_APP_ID、FEISHU_APP_SECRET），以及在哪里获取它们。用户第一次用的时候照着配就行。

2. **大操作拆成小步骤**

   不要指望一个API调用完成所有事情。「创建文档并写入内容并添加权限并转移所有权」应该拆成四个独立步骤，每步都有成功/失败的反馈。哪步失败了，就从哪步重试。

3. **记录已知的坑**

   每个API都有自己的怪癖。飞书的图片三步法、ImgBB的文件大小限制、某些API必须unset代理才能正常调用。这些坑踩过一次就记进Skill里，不要让自己（或别人）踩第二次。

4. **每步都要有用户反馈**

   「正在上传图片...上传成功，链接：https://...」「创建文档成功，正在写入内容...」。API调用是黑盒操作，用户看不到过程，只有通过反馈才能知道进度是否正常。

模式5：多Agent协作型 Swarm Pattern

这是最「高级」的模式，但不一定是最常用的。只有当任务可以真正并行、且规模足够大时，才值得用。

核心思路：把一个大任务拆成多个独立子任务，分配给多个AI Agent并行执行，最后合并结果。

我的huashu-agent-swarm（蜂群开发Skill）用的是一个比较极端的方案：**纯git自组织，没有master agent。**每个Agent在独立的git worktree中工作，互不干扰。通过git branch和merge来协调。没有一个「总管」Agent在中间分配任务和汇总结果。

为什么不要master agent？因为在实际使用中，master agent会成为瓶颈。它要理解所有子任务的状态，要处理冲突，要做调度决策。这些额外的认知负担会让它犯错。相比之下，用git这种「笨」方法反而更可靠：每个Agent只管自己的分支，做完了merge，冲突了人工resolve。

这个模式在写这本橙皮书的时候用过。10个章节，每章一个Agent，并行写作。每个Agent拿到相同的全书大纲和风格指南，然后独立完成各自的章节。写完之后我逐章review，调整章节之间的衔接和重复内容。

> **核心建议**
>
> 多Agent并行听起来很酷，但有个前提：**子任务之间必须真正独立**。如果第5章要引用第3章的结论，它们就不能完全并行。在拆分任务时，先画依赖图，只有没有箭头连接的任务才能并行。

多Agent协作型的设计技巧

1. **任务之间要真正独立**

   有依赖关系的任务不能并行。如果硬要并行，就会出现一个Agent等另一个Agent的结果，整体效率反而更低。宁可把有依赖的部分串行执行。

2. **用文件系统或git做协调**

   不要让Agent之间直接通信。Agent之间的「对话」会指数级增加复杂度。用文件和git分支这种「共享存储」的方式协调，简单粗暴但有效。

3. **合并时需要人工review**

   并行产出的质量是参差不齐的。有的Agent发挥好，有的会跑偏。合并不是简单地把所有产出拼在一起，而是需要人看过、调整过、确认了才算完成。

4. **不是所有任务都该并行**

   启动多个Agent有成本：拆分任务、分配上下文、合并结果。对于一个30分钟就能做完的任务，花20分钟拆分和10分钟合并，总耗时反而更长。**并行的收益只在任务规模足够大时才显现。**

模式组合：真实Skill是怎么设计的 Combining Patterns

真实世界里，很少有Skill只用一种模式。大多数好用的Skill是两三种模式的组合。

拿我的huashu-wechat-image（公众号配图Skill）举例。它同时用了三种模式：

首先是**多方案选择型**：提出3个配图方向（比如AI生成的概念图、信息图、氛围图），每个方向有预览描述和优劣分析，等我选一个。

然后是**外部API集成型**：选定方向后，调用AI生图API生成图片，上传到ImgBB图床拿到永久链接，把链接内联到Markdown文件里。

最后是**检查清单型**：生成完图片后，检查一遍质量。有没有文字畸变？有没有用了禁忌风格（赛博霓虹、深蓝色底）？封面有没有个人水印？分辨率是否合格？

三种模式串联起来，形成一个完整的端到端配图流程。你也可以说它整体上是一个流水线型，只不过流水线的每个阶段内部用了不同的模式。

这就是设计模式的价值：它给你积木，你自己搭。

5种模式速查 Pattern Reference

模式	适用场景	核心结构	代表案例
检查清单型	审校、review、质量验证	输入→逐项检查→标记问题→修改建议	huashu-proofreading
多方案选择型	选题、设计方向、策略选择	需求→生成N个方案→展示优劣→等待选择	huashu-topic-gen
流水线型	端到端工作流、复杂任务	阶段1→检查点→阶段2→...→产出	huashu-book-pdf
API集成型	调用外部服务	需求→调用API→处理结果→格式化输出	huashu-feishu
多Agent协作型	大规模并行任务	拆分→分配→并行执行→合并	huashu-agent-swarm

掌握这5种模式后，面对任何新需求，你的思考过程会变成：**「这个需求最接近哪种模式？需要组合哪几种？」**而不是对着空白文件发呆。这就是模式的力量，它把「从0到1的创造」变成了「从0.5到1的调整」。

10 Skills的未来

The Future of Skills

Skills不只是一个技术特性。它代表了一种新的可能：把你的工作经验变成别人能直接拿去用的东西。

写到这里，我们已经从理解Skill是什么，走到了亲手创建和优化Skill。最后一章，聊聊我对Skills未来的判断。有些是已经在发生的趋势，有些纯粹是我的猜测，我会尽量区分清楚。不一定对，但至少是我真实的判断。

从个人工具到团队知识 From Personal Tool to Team Knowledge

想象一个场景：新员工入职第一天，拿到电脑，装上团队的10个Skills。写周报的Skill告诉AI用什么格式、老板关心什么指标；做代码review的Skill定义了团队的编码规范和review标准；写技术方案的Skill包含了公司惯用的模板和审批要求。

这个新人不需要翻wiki，不需要问同事「咱们周报一般怎么写」，不需要看三个月才能摸清团队的不成文规矩。**装上Skills，AI大概率就能按团队标准来干活了。**当然，实际效果取决于Skill写得好不好，但方向是对的。

这比传统知识管理强在哪？传统方式是写文档放在Notion或飞书里。问题是，文档写了没人看，看了记不住，记住了执行走样。Skills不一样，它不是给人看的文档，它是AI会主动执行的活知识。你不需要记住规范，AI替你记着，而且每次都按规范来。

我自己的体验是：27个Skills就是我这个「一人公司」的全部SOP。选题怎么做、调研怎么做、审校什么标准、配图什么风格，全在里面。如果明天我要招一个助理，我不需要花两周手把手教，我只需要让他用我的Skills就行。

当团队每个人都有自己的Skills，再把个人级的提炼成团队级的，**知识就不再锁在某个人脑子里了**。那个最懂业务流程的老员工离职了？没关系，他的经验已经沉淀在Skills里。

Skills经济：创作者生态 The Skills Economy

Skills市场已经不是概念了，它正在发生。

前面提到过的一些数据：像frontend-design这样的热门Skill安装量非常可观；AgentSkill.sh平台上有超过10万个Skills；SkillsMP上超过70万个。这说明市场需求是真实存在的，有大量的人在找现成的Skill来用。

这让我想起几个类似的生态演进。WordPress的插件生态：从免费插件开始，后来出现了付费高级版，再后来有了插件订阅制。Chrome扩展生态也类似。npm包生态更不用说了，整个现代Web开发都建立在成千上万的开源包之上。

Skills很可能走同样的路。现在大多数Skill是免费的，但已经能看到一些苗头：

- 免费基础版 + 付费高级版（更多功能、更精细的流程）

- 订阅制Skills套件（比如「内容创作者套件」包含10个Skill，每月更新）
- 企业定制（针对特定业务流程创建的专属Skills）

一个有意思的可能性：从「开源Prompt」到「Skills市场」的演进。两年前大家分享Prompt，但Prompt太碎片化了，没有结构，没有版本管理，换个模型可能就不好使了。Skill解决了这些问题，它有标准格式、有元数据、有触发条件、可以渐进式加载。**Skill就是Prompt的升级版，有了固定格式、有了版本管理、有了标准。**

Skills + MCP = 完整的AI Agent能力栈 Skills + MCP = Complete Agent Stack

前面第一章讲过，MCP给了AI「手」，Skills给了AI「工作手册」。这两者正在越来越紧密地绑定。

举个具体的例子。现在如果你想让AI帮你操作飞书，你需要做两件事：装一个飞书MCP Server（让AI能调用飞书API），再装一组飞书Skills（告诉AI怎么用这些API完成具体任务）。MCP提供能力，Skills提供知识。

我猜测未来的趋势是：**MCP Server和Skill会成套出现。**你装一个「飞书套件」，里面既有MCP Server也有配套的Skills。装完就能完整操作飞书，不需要自己摸索怎么用。就像你装一个App，它自带使用说明。

更远一点看，Skills + MCP构成了AI Agent的完整能力栈：

层级	提供的是	类比
基座模型	通用智能	人的大脑
MCP	连接能力（能操作什么）	手和工具
Skills	领域知识（怎么操作）	工作手册

三层缺一不可。模型再聪明，没有MCP就碰不到外部世界；有了MCP能碰到了，没有Skills就不知道该怎么碰。

Agent自己创建Skills Agents Creating Their Own Skills

Anthropic在博客里提到了一个未来愿景：Agent在工作中自动总结出新的Skill。

想象这个场景：你连续三天让AI帮你做类似的数据清洗工作。第三天，AI说：「我注意到你每次清洗数据都按同样的步骤做——先去重，再处理空值，最后标准化日期格式。要不要我把这个流程保存为一个Skill？以后你只需要说一句『清洗这个数据集』我就知道怎么做了。」

这个场景离我们其实没那么远。Claude Code的Memory功能已经在做类似的事情了，它会自动记录你的偏好和习惯。从自动记录到自动提炼成结构化的Skill，技术上不是什么大的跨越。

从「人教AI」到「AI自己学」，这个转变的意义在于：**它极大降低了创建Skill的门槛。**不是每个人都愿意坐下来把自己的工作流程写成文档。但如果AI能观察你的工作模式，替你写出来，你只需要审核确认，那就完全不同了。

不过AI自动提炼的Skill质量还得人把关。它能发现重复模式，但这个模式值不值得固化，还得靠人判断。我自己的感觉是，AI提炼出来的Skill大概有六七成是有用的，剩下的要么太碎（不值得做成Skill），要么太笼统（做

成Skill也没用）。

我的判断 My Take

我自己的感觉是，Skills会成为AI时代的「工作SOP」，但不是所有工作都适合skill化。还不好说它最终能覆盖多大比例的工作，但方向我是看好的。

适合的：有固定流程的工作、需要重复执行的任务、有明确质量标准的输出。比如周报撰写、代码审查、内容审校、数据报告。这类工作的特点是，做法相对稳定，好坏有标准，重复频次高。

不太适合的：需要高度创造性的工作、每次情况都完全不同的决策、需要深度人际互动的场景。比如写一首诗、处理一次棘手的客户投诉、做一个从未做过的战略决策。这些工作的核心不是流程，而是判断力和创造力。

> 一个可能的未来：你的简历上不再只写「精通Python」「熟悉React」，还会写「创建了50个生产级Skills，覆盖内容创作全流程」。Skills让「经验」变成了可验证、可转移的数字资产。你不用再说「我有十年经验」，你直接拿出Skills，别人一看就知道你的工作方法论是什么水平。

最后的话 One Last Thing

回顾一下这本书走过的路。第一章理解了Skills在AI进化中的位置，第二、三章搞清楚了它的本质和设计原理，第四到六章学会了怎么找、怎么装、怎么用，第七到九章亲手创建了自己的Skill并掌握了高级技巧。

如果你还没有开始，我的建议是：**从一个小的、具体的Skill开始。**

不要一上来就想做一个覆盖全流程的大Skill。想想你这周重复做了什么，找一个最简单的，花15分钟写一个SKILL.md，放到 `.claude/skills/` 目录下，然后用起来。

它不需要完美。第八章讲过，Skill是迭代出来的，不是一步到位的。你会在使用中发现哪里不够好，然后改进它。每改一次，它就更懂你一点。

其实，**Skills的真正价值不完全在技术本身。**它在于，当你试着把自己的工作流程写成Skill的时候，你会被迫认真审视自己的工作方式。你会发现有些步骤是多余的，有些是关键的，有些你以为很重要其实可有可无，有些你一直忽略的才是核心。

这个过程本身就有价值，即使你最终没有写出那个Skill。

> **核心建议**
>
> 这本书的所有内容和我实际使用的Skills，都会持续更新。关注公众号「花叔」或访问我的B站频道「AI进化论-花生」，获取最新版本和更多实战案例。

附录 参考资料

Appendix

附录A：推荐Skills清单 Recommended Skills

按使用场景分类的推荐Skills。来源标注「官方」的是Anthropic维护的，标注「社区」的来自AgentSkill.sh、skills.sh（Vercel）等平台。

场景	Skill名称	来源	说明
前端开发	frontend-design	官方	最热门的Skill之一，创建高质量前端界面
代码审查	code-review	社区	自动化代码review，支持多语言
文档撰写	doc-coauthoring	官方	结构化共写文档，三阶段流程
PDF制作	pdf	官方	PDF读写、合并拆分与制作
演示文稿	pptx	官方	PPT创建与编辑
内容创作	content-creator	社区	内容策划全流程，含SEO和分发
SEO优化	seo-audit	社区	技术SEO审计，识别问题并给修复建议
数据分析	xlsx	官方	电子表格处理，支持公式和图表
MCP开发	mcp-builder	官方	创建MCP Server，Python和TypeScript
Skill创建	skill-creator	官方	创建和优化Skills，含评估框架

> **核心建议**
>
> 这个清单会随着生态发展不断变化。获取最新推荐，可以访问 skills.sh（Vercel开源目录）或 agentskill.sh（社区市场）浏览按分类排列的Skills目录。

附录B：SKILL.md模板 SKILL.md Templates

两个模板，一个起步用，一个进阶用。直接复制到你的 `.claude/skills/your-skill-name/SKILL.md` 开始写。

起步模板

```
---
name: your-skill-name
description: |
  一句话描述这个skill做什么。
  当用户提到「关键词1」「关键词2」时使用此skill。
  即使用户只是说「口语化的表达」也应触发。
---

# Skill名称

## 执行步骤

1. 第一步：确认输入和需求
2. 第二步：执行核心操作
3. 第三步：整理输出
4. 第四步：确认结果

## 输出格式

描述期望的输出格式和结构。

## 注意事项

- 不要做什么（边界）
- 质量标准是什么
- 特殊情况怎么处理
```

进阶模板

```
---
name: your-skill-name
description: |
  一句话描述这个skill做什么（简洁、准确，AI根据这个决定是否加载）。
  当用户提到「关键词1」「关键词2」「关键词3」时使用此skill。
  当用户正在做某类工作时也应触发。
  即使用户只是说「某种口语化表达」也应触发。
---

# Skill名称

## 前置检查

执行前先确认：
- [ ] 输入是否完整？缺什么先问用户
- [ ] 是否有相关的历史文件可以参考？
- [ ] 工作目录是否正确？

## 执行步骤

### 阶段一：准备（用户确认后再进入阶段二）

1. 分析输入，理解需求
2. 列出执行计划，等用户确认

### 阶段二：执行

3. 核心操作步骤A
4. 核心操作步骤B
5. 中间产物保存到文件（防止会话中断丢失）

### 阶段三：输出

6. 整理最终输出
7. 质量自检（对照下方检查清单）
8. 交付并确认

## 输出格式

- 格式要求（Markdown/表格/文件）
- 长度要求
- 命名规范

## 质量检查清单

- [ ] 检查项1
- [ ] 检查项2
- [ ] 检查项3

## 注意事项
```

- 绝对不要做的事情
- 容易出错的地方
- 和其他Skill的协作方式

参考资源

- 相关文件路径
- 外部参考链接

> **核心建议**
>
> 起步模板足够应对80%的场景。当你的Skill需要多阶段执行、用户确认点、或和其他Skill协作时，再升级到进阶模板。不要一开始就写太复杂，迭代着来。

附录C：花生的27个Skills索引 Huashu's 27 Skills Index

这是我目前实际在用的全部Skills。按功能领域分组，每个标注了核心功能和设计亮点。如果你在做类似的工作，可以参考它们的设计思路。

内容创作核心（8个）

名称	功能	设计亮点
huashu-topic-gen	公众号选题生成，3-4个方向含标题、大纲和优劣分析	强制先讨论再动笔，避免直接写跑偏
huashu-article-edit	标准化文章编辑流程，增量编辑实时保存	防会话截断丢失，先列修改项再执行
huashu-article-to-x	长文浓缩为200-500字X平台内容	提供3种开头风格让用户选
huashu-research	结构化深度调研，多轮搜索增量保存	每搜一轮就写入文件，防止断点丢调研成果
huashu-info-search	写作中嵌入式信息查证和知识管理	区别于独立调研，是写作过程中的即时验证
huashu-material-search	搜索1800+条即刻动态中的真实经历	降AI味的杀手锏，真实素材比任何技巧都有效
huashu-proofreading	三遍审校：事实、AI味、细节	6大类AI腔识别清单 + 花生风格改写规则
huashu-prompt-save	Prompt自动分类保存和索引管理	5大类自动识别，无需手动归档

视频生态（8个）

名称	功能	设计亮点
huashu-douyin-script	抖音爆款脚本：竞品拆解到脚本生成	Gemini视频分析 + 爆款公式提炼
huashu-video-outline	视频大纲生成，2-3个方案含时长预估	5种标题对比公式，强制先选方案再写脚本
huashu-video-check	视频封标与内容承接检查	基于MrBeast策略的三维度评估
huashu-script-polish	脚本三遍审校：准确性、口语化、演示标注	让脚本适合大声说出来
huashu-video-material	视频素材批量分析与剪辑指导	Gemini API分析视频，输出素材清单和剪辑脚本
huashu-danmaku-gen	根据SRT字幕生成B站弹幕列表	UP主弹幕 + 模拟观众弹幕两种类型
huashu-speech-coach	基于MIT方法论的演讲教练	幻灯片七宗罪检查 + 模拟观众视角
huashu-slides	AI演示文稿全流程，17种视觉风格	AI插画和HTML两条路径，用户先选再做

视觉配图（4个）

名称	功能	设计亮点
huashu-wechat-image	公众号配图：封面+正文插图+信息图	AI生成为主，精确数据用HTML兜底
huashu-xhs-image	小红书配图：封面+套图+富文本长图	智能选择AI/HTML路径，支持长文转分页图
huashu-image-upload	图片上传ImgBB图床生成Markdown链接	端到端：分析需求 → 生成/获取 → 上传 → 插入
huashu-design	设计哲学顾问，20种风格中推荐3个方向	并行生成视觉Demo，自动评审打分

文档出版（3个）

名称	功能	设计亮点
huashu-book-pdf	100页+书籍级PDF手册制作	模块化HTML片段 + 语义化版本 + 多Agent并行写作
huashu-md-to-pdf	Markdown转苹果设计风格PDF	自动封面、双列目录、页眉页脚
huashu-weread	微信读书电子书发布全流程	简介撰写 + 封面设计 + EPUB构建 + 上传检查

系统基础设施（4个）

名称	功能	设计亮点
huashu-agent-swarm	多Agent并行协作开发	纯git自组织，无master agent，蜂群模式
huashu-auto-skill-optimizer	自动评估和优化SKILL.md	7维度评分 + 爬山算法迭代 + git版本控制
huashu-feishu	飞书API集成：消息、文档、知识库	图片三步法 + 加粗支持 + 权限自动处理
huashu-data-pro	数据分析全能助手，多专家并行	HTML报告/PPT自动生成，端到端数据工作流

附录D：常用资源链接 Useful Links

标准与文档

资源	链接
Agent Skills开放标准（Anthropic推出）	agentskills.io
Anthropic官方Skills仓库	github.com/anthropics/skills
Claude Code Skills文档	code.claude.com/docs/en/skills
Agent Skills API文档	platform.claude.com/docs/en/agents-and-tools/agent-skills/overview

Skills市场

平台	链接	规模
skills.sh（Vercel推出，开源）	skills.sh	开源Skills目录
AgentSkill.sh	agentskill.sh	10万+ Skills
SkillsMP	skillsmp.com	70万+ Skills

学习资源

资源	链接
DeepLearning.AI课程	learn.deeplearning.ai/courses/agent-skills-with-anthropic

花生的频道

平台	链接
公众号	微信搜索「花叔」
B站	AI进化论-花生（space.bilibili.com/14097567）
X/Twitter	x.com/AlchainHust
YouTube	youtube.com/@Alchain

附录E：术语表 Glossary

术语	英文	解释
Agent	Agent	能够自主执行任务的AI系统。不只是聊天，而是能调用工具、读写文件、做出决策并执行操作。
MCP	Model Context Protocol	Anthropic发布的开放协议，让AI工具可以用统一的方式连接外部服务。给AI提供了「手」。
Skill	Agent Skill	用自然语言编写的指令文档，教AI按照特定方式完成特定任务。给AI提供了「工作手册」。
SKILL.md	—	Skill的标准文件格式。一个Markdown文件，包含元数据（frontmatter）和指令正文。
Frontmatter	Frontmatter	SKILL.md文件顶部用 --- 包裹的YAML格式元数据区域，定义Skill的名称、描述和触发条件。
触发条件	When / Trigger	定义在什么情况下自动加载这个Skill。写在frontmatter的description字段中，用自然语言描述触发场景。
渐进式披露	Progressive Disclosure	一种Skill设计模式：不一次性加载所有内容，而是按需分阶段加载，节省上下文窗口。
上下文窗口	Context Window	AI模型一次对话中能处理的最大文本量。Skills需要占用上下文窗口，所以设计时要控制大小。
Token	Token	AI模型处理文本的基本单位。大约1个中文字 = 1-2个token，1个英文单词 = 1个token。
Prompt	Prompt	给AI的指令或提问。Skill可以理解为结构化的、可复用的、自动触发的Prompt。
skill-creator	—	Anthropic官方提供的Skill，用来辅助创建和优化其他Skills。类似「用Skill来造Skill」。
Agent Skills开放标准	Agent Skills Open Standard	Anthropic发布的开放标准，定义了Skills的格式和行为规范，已被20多个AI产品采纳。
Worktree	Git Worktree	Git的多工作目录功能，允许同一仓库同时检出多个分支。多Agent并行协作时常用。