Claude Skills 不就是把提示词存个文件夹吗？

原创 刘小排 刘小排r 2025年11月14日 11:19 北京

哈喽，大家好，我是刘小排。

最近这段时间，我在反复研究和使用 Claude 新发布的 *Skills* 功能。

很多朋友问第一眼看过去会觉得：

> 这不就是把提示词存成一个文件夹吗？

再用一用，会觉得：

> 这和MCP有啥区别？

估计 Anthropic 自己也被问懵了，于是官方干脆写了一篇长文：《Skills explained: How Skills compares to prompts, Projects, MCP, and subagents》，专门来解释这几个名字听上去就很反人类的概念。

https://claude.com/blog/skills-explained

这篇文章来得非常及时。对我们这些做 AI 产品的人来说，它其实在讲一件更大的事：

> **有区别，而且区别非常大。**
>
> **通俗一点讲：Skills 是把「提示词工程」，升级成了「流程工程」。**

下面这篇是我站在「AI 产品创业者」的视角，把官方内容全部捋一遍，再加上我自己的理解。

提前说一句：**很长，但是值得耐心看完。**

接下来我会按这个顺序来讲：

1. Claude 生态里到底有几块「积木」？
2. Skills：让 Claude 真正「学会干活」
3. Prompts：依然是主角，但天生是一次性的
4. Projects：给每一个重要主题一个「专属上下文空间」

5. Subagents：给每个子任务配一个「专职 AI 同事」

6. MCP：把所有外部系统接成一张网

7. 一个完整的「研究 Agent」案例：这几块积木怎么拼在一起？

8. 官方 FAQ 里的几个关键信息

9. 实际上手：不同类型用户怎么用 Skills

0. 作为 AI 产品创业者，我自己的一个判断

下面的文字很长！再次提醒，做好准备！

一、Claude 生态里到底有几块「积木」？

先把名字捋顺，否则后面全是雾

- **Prompts**：你在对话框里敲给 Claude 的那一段话
- **Skills**：一个个「能力文件夹」，里面是可复用的流程、脚本和资源
- **Projects**：带自己知识库和历史记录的「项目空间」
- **Subagents**：专门干某件事的小助手，像「子 AI」
- **MCP（Model Context Protocol）**：把 Claude 接到你各种外部工具和数据源上的「通用连接层」

如果用一句人话总结：

> Prompts 是“当场吩咐一句”，
>
> Skills 是“把做事的方法写进操作手册”，
>
> Projects 是“给 AI 搭一个项目档案室”，
>
> Subagents 是“请来一堆专职的 AI 同事”，
>
> MCP 是“打通所有外部系统的总线”。

下面我们一个个拆。

二、Skills：让 Claude 真正「学会干活」

1. Skills 是什么?

官方定义是：

> Skills 是一些文件夹，里面放着指令、脚本和资源，当 Claude 觉得当前任务需要它时，就会动态加载。Claude

你可以把它想象成：
给 Claude 写的一本本“岗位说明书 + SOP + 工具包”。

比如一个「品牌规范 Skill」，里面可以写清楚：

- 品牌主色、辅色、渐变怎么用
- 标题字体、正文字体分别是什么
- PPT 的版式有哪些固定模板
- LOGO 在任何地方出现的尺寸和留白规则
- 不允许出现的低级审美错误

以后你再让 Claude「帮我写一份路演 PPT」，它会自动套用这套规范，**不需要你每次重新科普一遍品牌手册**。

2. Skills 在后台是怎么工作的?

这里有个很有意思的设计：**渐进披露（progressive disclosure）**。

大致流程是这样的：

1. **先读“封面简介”**
   Claude 会先扫描所有可用 Skills 的「元数据」——几句描述，大约 100 tokens 左右。
   目的只是判断：**这个 Skill 跟当前任务有没有关系。**

2. **觉得相关，再读“说明书正文”**
   一旦判断相关，它才会加载整个 Skill 的详细说明（SKILL.md），官网提到上限大约是 5k tokens，这里面通常是：
   - 步骤、流程
   - 注意事项
   - 输出格式要求
   - 风格偏好等

3. **真的需要代码时，才加载脚本和文件**
   有些 Skill 还会带脚本或参考文件（比如模板、示例）。
   只有在真正需要执行相关操作时，Claude 才会把这些东西「拎进上下文」。

这个设计的意义在于：

> 你可以给 Claude 装很多 Skills，
> 但不会一上来就把上下文撑爆，
> 它只会在需要的时候，把需要的那一本“手册”翻开。

3. 什么时候应该用 Skills?

官方给了三个典型场景：

- **组织级工作流**
  - 品牌规范
  - 法务／合规流程
  - 各种标准化文档模板、
- **某个专业领域的「经验总结」**
  - Excel 公式、常用数据分析套路
  - PDF 处理的流程
  - 安全审计、代码 Review 的标准
- **个人偏好 & 习惯**
  - 你的笔记结构
  - 你的代码风格
  - 你的研究方法

一句话：

> **任何你不想一遍遍重新解释的东西，都可以长久地写进 Skill。**

三、Prompts：依然是日常交互的主角，但不适合作为「长期记忆」

1. Prompts 是什么?

这个大家都熟：

> Prompts 就是你在对话里用自然语言给 Claude 下的那些指令，是实时的、对话式的、一次性的。Claude

比如：

- 「帮我总结这篇文章」
- 「把刚才那段话的语气改得更专业一点」
- 「帮我分析一下这份数据，看有什么趋势」
- 「用项目符号重新排版一下」

甚至可以是非常完整的一段复杂 prompt，比如官方举的「请你对这段代码做一个完整的安全审计」，后面跟了详细的检查项、严重程度定义、修复建议要求等等。

2. Prompts 的局限在哪里?

Claude 官方直接说了：

> Prompt 是你和模型交互的主要方式，但它不会在不同对话之间自动保留。

也就是说：

- 你今天费心写了一个很长的「代码安全审计」提示词
- 明天开新对话，还得重新粘一遍
- 换个项目、换个窗口，又得重来

于是他们给出一个很自然的建议：

> **如果你发现自己在多个对话里反复敲同一类 Prompt，那就该把它升级成 Skill 了。**

比如这些典型句型：

- 「请按照 OWASP 标准对这段代码做安全审计」
- 「请总是给出'高层摘要 + 关键发现 + 建议'这三个结构」

这类东西，适合写进 Skill，变成「永远的工作方式」，而不是「今天一时想起来的提示词」。

官方也推荐你先看他们的 prompt library、最佳实践、以及一个「智能 Prompt 生成器」，这个就不展开了。

四、Projects：给每一个重要主题一个「专属上下文空间」

1. Projects 是什么?

在 Claude 的付费方案里，**Projects 是一个个独立的工作区**：

- 有自己的聊天记录
- 有自己的知识库
- 有自己的「项目级」指令

每个 Project 有一个大上下文窗口（官方说是 200K tokens 级别），你可以往里面上传各种文档、资料，让 Claude 在这个空间下工作。

当知识量很多的时候，Claude 会自动切换成类似 RAG 的模式，把项目知识进行检索，整体可扩到原来上下文的 10 倍左右。

2. 什么时候适合用 Projects?

官方建议：

- 需要长期存在的背景知识
  - 某个产品线
  - 某个大客户
  - 某个长期课题
- 需要把不同工作分“项目隔离”
  - Q4 产品发布
  - 某一场活动运营
  - 某一轮融资材料
- 团队协作（Team / Enterprise）
  - 共享历史对话
  - 共享知识库
- 项目级的自定义指令
  - 这个项目里，所有输出都要偏「To B、专业、严谨」
  - 另一个项目里可以更轻松一点

官方例子：

> 建一个「Q4 Product Launch」项目，把市场研究、竞品分析、产品规格都扔进去，以后在这个项目里的所有对话都会自动带着这些背景。

3.Projects 和 Skills 的区别

这一点很关键。官方一句话概括得非常好：

> **Projects 解决的是「你要知道什么」（背景知识）。**
> **Skills 解决的是「你要怎么做事」（流程方法）。**

换个比喻：

- Project 像「整个项目的档案室＋学习资料」
- Skill 像「公司内部的一份份标准操作手册」

Project 是局部的——只在这个项目空间里生效。
Skill 是全局可用——任何对话、任何项目，只要相关，都能调出来用。

五、Subagents：给每个子任务配一个「专职 AI 同事」

1. Subagents 是什么？

在 Claude Code 和 Claude Agent SDK 里，你可以创建很多「子代理（subagents）」。
它们具备：

- 自己的上下文窗口
- 自己的系统提示
- 自己的一组工具权限

你可以把它们理解成：

> **Subagents = 一个个岗位明确、权限有限、职责清晰的 AI 员工。**

2. Subagents 适合干什么？

官方给了 4 类典型用途：

- **任务专业化**
  - 专门做代码审查
  - 专门生成单元测试
  - 专门做安全审计
- **上下文的拆分**
  - 主对话保持干净
  - 把“重活”丢给 subagent 做
- **并行处理**
  - 一个 subagent 做市场调研
  - 另一个做技术分析
  - 再一个做文档整理
- **工具权限隔离**
  - 某些 subagent 只有只读权限
  - 它永远不能写入、不能删东西

例子：

> 建一个「代码审查 subagent」，
> 只给它 Read / Grep / Glob 权限，不给 Write / Edit。
> 每次代码有改动，Claude 会自动把审查任务丢给它，
> 这样就能保证有安全审查，而不会误改代码。

3. Subagents 和 Skills 怎么配合？

官方推荐是：

- **多对多**
  - 一个 subagent 可以使用多个 Skills（例如语言规范、领域 Best Practice）
  - 多个 subagent 也可以共享某些 Skills（比如统一的写作规范 Skill）

你可以这样理解：

> Skill 更像“知识＋流程”；
> Subagent 更像“带着这些知识/流程去执行任务的具体人”。

六、MCP：把所有外部系统接成一张网

1. MCP 是什么？

Model Context Protocol（MCP）是一个开放协议，用来把 AI 助手接到各种外部系统上。

简单理解：

> 你不用再给每个系统写一套单独的集成，
> 只要对接 MCP，就可以用统一方式连接各种数据源和工具。

官方举的典型连接对象：

- 外部数据源：Google Drive、Slack、GitHub、数据库等
- 业务工具：CRM、项目管理系统
- 开发环境：本地文件、IDE、版本控制
- 自研系统：你们自己公司的内部平台

你把这些系统包装成一个个 MCP server，Claude 作为 MCP client 去连它们。

2. MCP 和 Skills 怎么配合？

非常重要的一点是：

> **MCP 负责“接通数据和工具”，**
> **Skills 负责“告诉 Claude 要怎么用这些数据和工具”。**

比如：

- MCP：让 Claude 能访问你的数据库
- Skill：规定「查询时必须先按日期过滤」「查询结果要按某种格式输出」
- MCP：连接你的 Excel 文件
- Skill：规定「生成报表时必须使用哪些公式」「怎么排版」

未来比较理想的状态是：

> 每接入一个新系统（MCP），
> 最好配一套相应的使用说明和流程（Skill）。

七、【重点】这些东西是怎么拼在一起的？——一个「研究 Agent」的完整例子

如果你只想知道「这个东西怎么用在真实工作里」，下面这个"研究 Agent"的例子是最值得耐心看完的一段。

官方给了一个很完整的例子：

> 构建一个用于竞品研究的综合 Agent，
> 同时用到 Projects、MCP、Skills 和 Subagents。

我们按步骤拆：

第一步：建一个 Project——「竞争情报」
把下面这些东西都扔进去：

- 行业报告、市场分析
- 竞争对手的产品文档
- CRM 里的用户反馈
- 你们之前写过的研究总结

并且加一段项目级指令：

> 分析竞品时要站在我们自家产品战略的视角，
> 尤其关注差异化机会和新兴趋势，
> 给出的结论要带证据、要可执行建议。Claude

第二步：用 MCP 接数据源

打开几个 MCP server：

- Google Drive：访问共享研究文档
- GitHub：看竞品的开源仓库
- Web 搜索：查实时的市场信息

第三步：创建一个「竞争分析 Skill」

比如叫 `competitive-analysis`，里面可以包含：

- 公司内部 GDrive 的目录结构
- 搜索时的最佳实践
  - 先从哪个目录下手
  - 优先看最近 6 个月的文档
  - 最终确认「权威版本」的方式
- 一个标准化的研究工作流：
  1. 明确研究主题
  2. 在对应的目录里组合关键词搜索
  3. 选 3–5 个最新 / 最关键文档
  4. 和战略文档交叉引用
  5. 输出时标注来源文件名和日期

这就是一个非常典型的「流程型 Skill」。

第四步：配置 Subagents（在 Claude Code / SDK 里）

比如两个子代理：

1. `market-researcher`
   - 优先使用权威来源（Gartner/Forrester 等）
   - 关注市场份额、增长率、融资情况
   - 需要给出带引用和置信度的结论
   - 负责：市场趋势、行业报告、竞品定位
   - 工具：Read、Grep、Web-search
   - 系统提示里写清楚：
2. `technical-analyst`
   - 分析技术栈、架构模式
   - 评估可扩展性和性能
   - 找出技术优势和短板，并给出对你有用的启示
   - 负责：技术架构、实现方式、工程决策
   - 工具：Read、Bash、Grep
   - 系统提示：

第五步：调用这个 Agent

你现在问 Claude：

> 「帮我分析一下我们前三个主要竞品最近发布的 AI 功能，它们是怎么定位的？我们有哪些可利用的空档？」

背后到底会发生什么？（重点来了）

1. **Project 上下文加载**：
   - Claude 拿到你之前上传的研究文档、战略文档
2. **MCP 联通数据**：
   - 去 GDrive 里找最新竞品材料
   - 去 GitHub 拉开源仓库
   - 用 Web 搜索补实时信息
3. **Skill 启用**：
   - 用 `competitive-analysis` 这个 Skill 提供的工作流框架来组织分析
4. **Subagents 并行工作**：
   - `market-researcher` 去研究市场定位
   - `technical-analyst` 看技术实现
5. **你通过 Prompt 微调方向**：
   - 比如补一句：「重点关注医疗行业的企业客户」

最终，你拿到的是一份：

- 有来源
- 有结构
- 有可执行建议
- 又符合你战略视角的竞品研报。

这就是「几块积木组合起来」的威力。

八、官方 FAQ 里的关键点

最后，官方在文末做了一段 FAQ，总结得很好，我给你浓缩一下：

1. Skills 是怎么保持「高效」的？

靠的就是前面说的「渐进披露」：

- 先扫 metadata
- 再按需加载完整说明
- 有代码和文件再按需加载

所以你可以放心地给 Claude 装很多 Skills，它不会一开始就被上下文压垮。

2. Skills vs Subagents：什么时候用哪个？

- 用 **Skills**：
  - 当你希望「任何 Claude 实例」都能加载某种能力，比如安全审计流程、数据分析方法
- 用 **Subagents**：
  - 当你需要一个完整的「小代理」，自己带上下文、带工具权限，能独立跑完整工作流

最推荐的姿势是：

> Subagent + Skills 组合使用。
> 让一个专职"代码审查 subagent"去调用「语言特定 best practice Skill」，
> 相当于给这个小同事配一堆教材。

3. Skills vs Prompts：什么时候该升级？

- 用 **Prompts**：
  - 一次性指令、即时交互、补充上下文
- 用 **Skills**：
  - 当你有一套流程／专业经验，需要反复使用
  - 当你希望 Claude 能自己判断「什么时候应该套这套流程」

比较理想的模式是：

> 用 Skills 打地基，用 Prompts 在每次任务上做具体微调。

4. Skills vs Projects：核心差别是什么？

官方原话的对比非常精炼：

- Projects：
  - 「这是你需要知道的东西」（知识、文档、背景）
  - 总是在项目里加载
- Skills：
  - 「这是你应该怎么做事」（流程、代码、方法）
  - 动态按需加载，节省上下文

你可以把它记成一句话：

> **Project = 知识场景，Skill = 能力模组。**

5. Subagents 能不能用 Skills？

答案是：**可以，而且非常推荐。**

在 Claude Code 和 Agent SDK 里，Subagent 可以和主 Agent 一样使用 Skills。

比如：

- `python-developer` subagent
  - 用 `pandas-analysis` Skill 按你团队习惯来做数据处理
- `documentation-writer` subagent

- 用 `technical-writing` Skill 固定 API 文档的写法和格式

九、如何开始上手 Skills?

如果你是刘小排的读者，那你可以看这篇　技术圈吹爆的Claude Skills，小白用起来比程序员还爽 和这篇　用Claude Skills做PPT，真实演示

官方给了三类用户的路径，你可以按自己场景选：

1. 如果你是 Claude.ai 网页用户
- 在 Settings → Features 里把 Skills 打开
- 去 claude.ai/projects 创建你的第一个项目
- 在一个具体分析任务里尝试「Project + Skill」联合使用

2. 如果你是 API 开发者
- 去看文档里关于 Skills endpoint 的部分（支持通过 API 管理 Skills）
- 打开官方的「skills cookbook」仓库，看他们给的 Skill 示例

3. 如果你是 Claude Code 用户
- 通过插件市场安装 Skills
- 同样可以参考「skills cookbook」，照着抄一遍先跑起来

十、作为 AI 产品创业者，我的一个小结论

读完这篇官方文档，我脑子里冒出的最强烈一个念头是：

> **Prompt Engineering 只是上半场，**
> **真正的下半场，是「流程工程（Workflow / Skill Engineering）」。**

- Prompt 解决的是「怎么和模型说话」
- Skill/Project/Subagent/MCP 解决的是
  「模型在一个复杂环境下，怎么长期、稳定、可维护地为你干活」

对于做 AI 产品的人来说，这是几个非常现实的落地点：

1. 你的差异化，将越来越体现在 Skills 设计上

- 把领域经验、工作流、组织规范，沉淀成一套 Skills 库
- 这是你产品的护城河，而不仅仅是“调了哪个大模型”

2. **你的产品架构，会越来越像「Agent 积木组合」**
   - 某个功能 = 若干 Project + Skills + Subagents + MCP 的组合
   - 未来甚至可以对外开放自己的 Skill Store

3. **你的团队，迟早需要一个“Skill Engineer / AI Workflow Architect”角色**
   - 不再只是写 prompt 的人
   - 而是真正负责「把经验变成可复用的 AI 工作流」的人

现在很多人还沉迷在“写花式 Prompt”，但从 Skills / Projects / MCP / Subagents 这套组合来看，趋势已经非常明显了：

做 AI 产品，如果只停留在 Prompt 层，就是停留在 Demo 层；要往真正的“业务系统”走，就绕不开流程工程和 Skill 设计。

我自己现在已经用 Skills 做很多件事：

- 一个是「写公众号 Skill」，比如标题怎么写、导语怎么设计、配图比例怎么选，都写死在里面；
- 一个是「代码性能分析Skill」，性能涉及到很多方面，比如数据库设计（索引、事务等）、Redis和数据库的配合、代码中的算法和架构等等、缓存策略等等，单独靠MCP或Subagent是很难完成的，需要一整套流程。
- 一个是「产品更新日志 Skill」，我只管往里丢 changelog，它会自动帮我改成对用户友好的版本。
- 一个是「产品idea头脑风暴Skill」，我有新的idea的时候，不再是直接问ChatGPT，而是有一个特定的流程。
- 一个是「域名讨论Skill」，做新产品时，想域名是一个头疼的事，可以通过Skill来找到后选域名、查询是否可用
- ……

谢谢你看到这里！期待你的反馈。

Claude Code · 目录

上一篇 · 终于有国产模型支持「看图写代码」了：9块9元包月的Doubao-Seed-Code实测