一文读懂 Skills｜从概念到实操的完整指南

字节跳动技术团队 2026年1月22日 17:39 北京

以下文章来源于TRAE.ai ，作者TRAE

TRAE.ai

TRAE，The Real AI Engineer｜字节跳动旗下的AI编程产品，你的专属AI开发工程师...

本文作者：咸鱼，TRAE 开发者用户

Agent 正在经历从“聊天机器人”到“得力干将”的进化，而 **Skills** 正是这场进化的关键催化剂。

你是否曾被 Agent 的“不听话”、“执行乱”和“工具荒”搞得焦头烂额?

本文将带你一文弄懂 **Skills** ——这个让 Agent 变得可靠、可控、可复用的“高级技能包”。

我们将从 Skills 是什么、如何工作，一路聊到怎样写好一个 Skills，并为你推荐实用的社区资源，带领大家在 TRAE 中实际使用 Skills 落地一个场景。

无论你是开发者还是普通用户，都能在这里找到让你的 Agent “开窍”的秘诀。

你是否也经历过或者正在经历这样的“ Agent 调教”崩溃时刻?

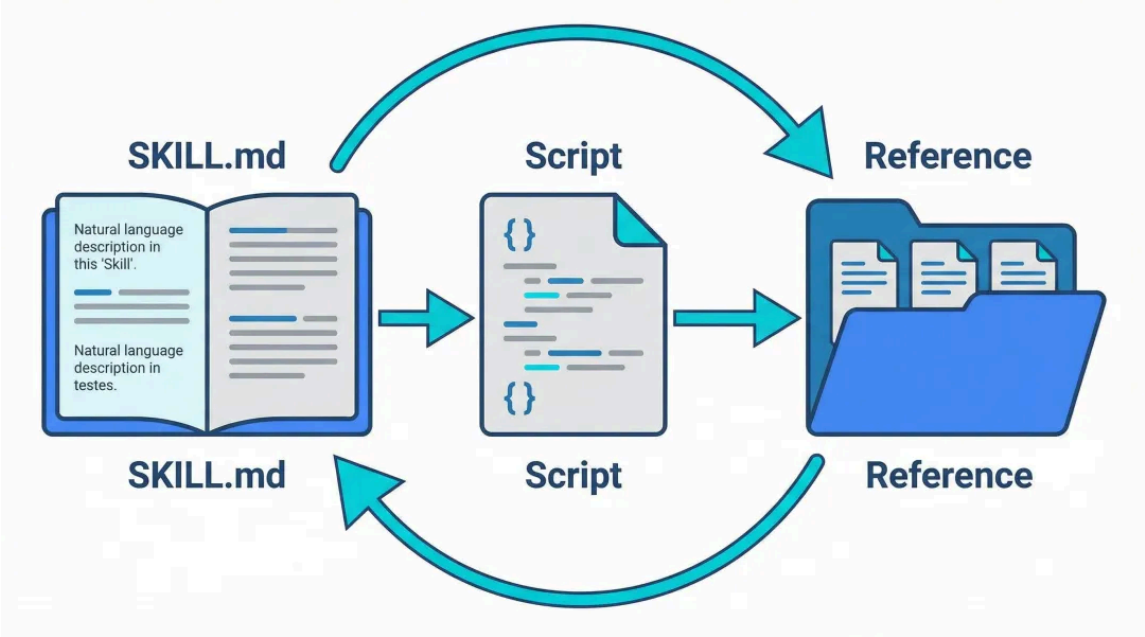
- **规则失效：**在 Agent.md 里写下千言万语，Agent 却视若无睹，完全“已读不回”。
- **执行失控：**精心打磨了无数 Prompt，Agent 执行起来依旧像无头苍蝇，混乱无序。
- **工具迷失：**明明集成了强大的 MCP 工具库，Agent 却两手一摊说“没工具”，让人摸不着头脑。

如果这些场景让你感同身受，别急着放弃。**终结这场混乱的答案，可能就是 Skills。**

01

什么是 Skills

“Skills”这个概念最早由 Anthropic 公司提出，作为其大模型 Claude 的一种能力扩展机制。简单来说，它允许用户为 Claude 添加自定义的功能和工具。随着这套做法越来越成熟，并被社区广泛接受，Skills 如今已成为大多数 Agent 开发工具和 IDE 都支持的一种标准扩展规范。



一个 Skills 通常以一个文件夹的形式存在，里面主要装着三样东西：**一份说明书（SKILL.md）、一堆操作脚本（Script）、以及一些参考资料（Reference）。**

内容	作用
SKILL.md	通过自然语言描述清楚（使用场景、使用方式、使用步骤、及注意事项等上下文补充信息）
Script 脚本	Agent 可以执行的具体脚本代码
Reference 引用	参考文档，引用的模板，相关关联上下文的文件信息

你可以把一个 Skill 想象成一个打包好的“技能包”。它把完成某个特定任务所需的**领域知识、操作流程、要用到的工具、以及最佳实践**全都封装在了一起。当 AI 面对相应请求时，就能像一位经验丰富的专家那样，有条不紊地自主执行。

一句话总结：要是把 Agent 比作一个有很大潜力的大脑，那 **Skills 就像是给这个大脑的一套套能反复用的“高级武功秘籍”。**有了它，Agent 能从一个“什么都略知一二”的通才，

变成在特定领域“什么都擅长”的专家。

02

Skill 原理介绍

> 官方解释：
>
> https://platform.claude.com/docs/en/agents-and-tools/agent-skills/overview

Skill 的架构原理：渐进式加载

Skill 的设计很巧妙，它运行在一个沙盒环境里，这个环境允许大模型访问文件系统和执行 ***bash*** 命令（可以理解为一种电脑操作指令）。在这个环境里，一个个 Skill 就像一个个文件夹。Agent 就像一个熟悉电脑操作的人，通过命令行来读取文件、执行脚本，然后利用结果去完成你交代的任务。这种“按需取用”的架构，让 Skill 成为一个既强大又高效的“工具箱”。

为了平衡效果和效率，Skill 设计了一套聪明的三层分级加载机制：

级别	加载时机	Token 消耗	内容
1 级：元数据	始终（启动时）	每个技能约 100 Token	YAML 前置元数据中的名称和描述
2 级：说明文档	技能触发时	低于 5000 Token	SKILL.md 正文（包含操作指南和指导说明）
3 级及以上：资源	按需	几乎无限制	通过 bash 执行的捆绑文件（内容不加载到上下文）

Level 1：元数据（始终加载）

元数据就像是 Skill 的“名片”，里面有名称（***name***）和描述（***description***），是用 YAML 格式来定义的。Claude 在启动的时候，会把所有已经安装的 Skill 的元数据都加载进来，这样它就能知道每个 Skill 有什么用、什么时候该用。因为元数据很轻量，所以你可以安装很多 Skill，不用担心把上下文占满。

Level 2：说明文档（触发时加载）

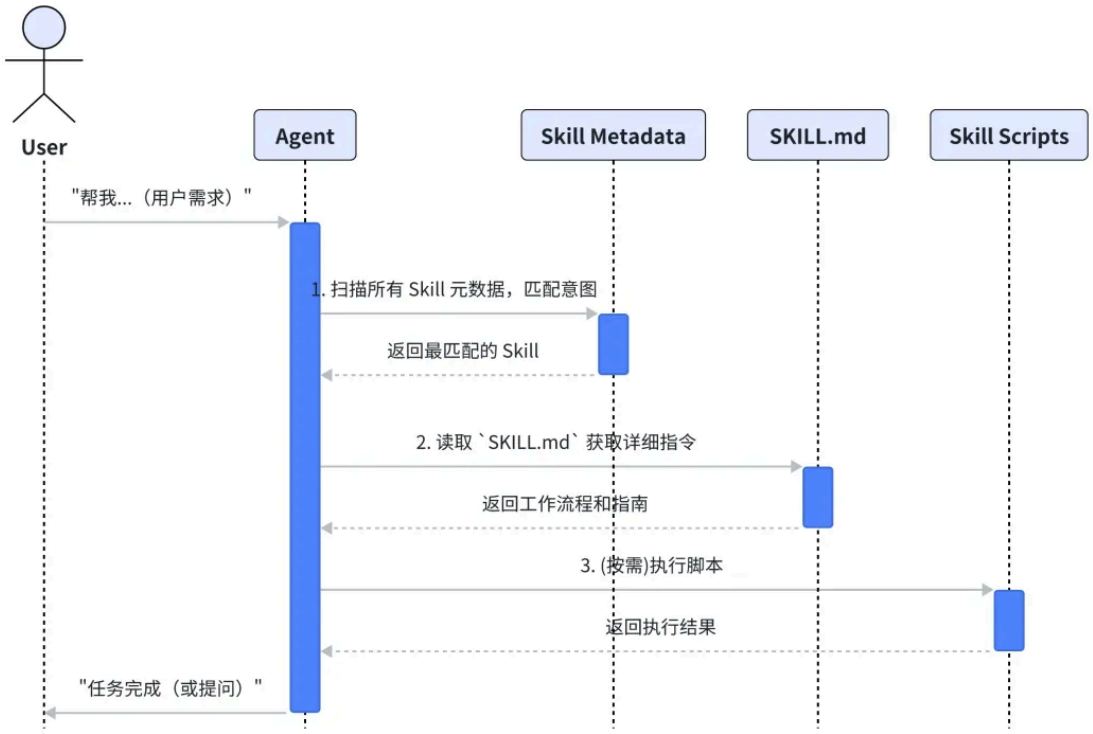
SKILL.md 文件的正文就是说明文档，里面有工作流程、最佳实践和操作指南。只有用户的请求和 Skills 元数据里的描述相符时，Claude 才会用 ***bash*** 指令读取这份文档，把内容加载到上下文里。这种“触发式加载”能保证只有相关的详细指令才会消耗 Token。

Level 3：资源与代码（按需加载）

Skills 还能打包一些更深入的资源，比如更详细的说明文档（***FORMS.md***）、可执行脚本（***.py***）或者参考资料（像 API 文档、数据库结构等）。Claude 只有在需要的时候，才会通过 ***bash*** 去读取或执行这些文件，而且脚本代码本身不会进入上下文。这样一来，Skills 就能捆绑大量信息，几乎不会增加额外的上下文成本。

Skills 的调用逻辑：从理解意图到稳定执行

那么，Agent 是如何智能地选择并执行一个 Skill 的呢？整个过程就像一位经验丰富的助理在处理工作：



1. **意图匹配（找到对的人）：** Agent 首先聆听你的需求，然后快速扫一眼自己手头所有 Skill 的“名片夹”（元数据），寻找最匹配的那一张。
2. **读取手册（看懂怎么干）：** 找到合适的 Skills 后，Agent 会像模像样地翻开它的“操作手册”（*SKILL.md*），仔细研究详细的执行步骤和注意事项。
3. **按需执行（动手开干）：** 根据手册的指引，Agent 开始工作。如果需要，它会随时从“工具箱”里拿出脚本或工具来完成具体操作。
4. **反馈结果（事毕复命）：** 任务完成后，Agent 向你汇报最终结果，或者在遇到困难时，及时向你请教。

03

Skills vs. 其他概念的区别

为了更清晰地理解 Skills 的独特价值，我们不妨把它和另外两个容易混淆的概念——**快捷指令（Command）**和**原子工具（MCP）**——放在一起做个对比。用一个厨房的例子就很好懂了：

概念	好比是.....	核心特点	适用场景
斜杠命令 Command	微波炉上的**“指定按钮”**（如：`/解冻`、`/爆米花`）	由**用户主动触发**的、固定的、单一功能的命令。简单直接，但不够智能和灵活。	作为主动复用 prompt 的快捷方式，让用户快速复用模板 prompt
模型上下文协议 MCP	厨房里的**“单一厨具”**（如：一把刀、一个烤箱）	为 AI 提供调用**外部 API** 或脚本的基础能力。模型知道有这个工具，但不知道具体在什么菜谱里、什么步骤下使用，以及怎么更好的使用。	通过标准协议链接外部系统获取信息或完成任务
技能 Skills	一本完整的**“菜谱”**（如：“如何制作法式焗蜗牛”）	由**模型根据任务自动匹配和驱动**。它定义了制作这道菜（完成这个任务）需要用到哪些厨具（原子工具）、遵循哪些步骤、注意什么火候（最佳实践），是**流程化、场景化**的。	由模型驱动意图理解，通过渐进式加载补充上下文，并根据 SOP 完成指定任务

我们也列举了几个大家容易混淆的其他功能，一起来对比看看。

概念	核心特点	适用场景	易混淆点
rules 规则	通常通过 agent.md 或者 project_rules 文件维护，内容基本与项目绑定	主要专注于和项目强相关的场景	在无 Skills 的时候部分场景通过，在 rules 文件手动维护索引和文件 summary 的方式实现项目里的不同场景复用不同规则。
Agent 智能体	通常搭配相关 prompt 和对应的 tools use 打包使用，普遍用于多 agent 系统中	专用型场景使用 agent 可以更加符合场景需要	在 Skills + sub agent 的场景中也能实现多 agent 调度的的逻辑
Memory 记忆	由模型主动或被动从用户对话中提炼关键上下文以进行记录	日常开发过程中潜移默化的记录，对用户干扰性不大	Memory 生成的内容可能类似于 rules 或者 Skills 通常明确了使用场景的 memory 可以被手动迁移到 rules 或者 Skills 中

> 官方博客解释：
>
> https://claude.com/blog/skills-explained

04

什么是好的 Skills：从“能用”到“好用”

Good Skills vs Bad Skills

评判维度	Good Skills	Bad Skills
单一职责原则	**每个 Skill 只做一件事，且把它做好。**例如，可以分解为三个独立的 Skill：`query_data`、`generate_chart`、`send_email`。	一个 Skill 试图做太多事，比如“既负责数据查询，又负责图表生成，还负责邮件发送”。
描述清晰度	**描述清晰、具体，使用自然语言**，明确说明输入、输出和核心功能。例如：“根据用户提供的城市名和日期范围，查询并返回该城市的天气数据。”	描述模糊，充满技术术语，智能体难以理解。例如：“一个用于数据处理的工具。”
参数设计	参数精简、命名语义化（如`city_name`，`date_range`），并为每个参数提供清晰的描述和示例。明确使用 Skill 需要的参数如何获取，以及参数如何使用。	参数过多、命名不规范（如`arg1`，`p2`），缺少详细的注释说明。
可组合性	设计时就考虑到了**可组合性**，其输出可以作为其他 Skill 的输入，方便构建更复杂的任务流（Workflow），可以尝试通过单一职责完成原子 Skill 的开发，并通过某项具体任务 SOP Skill 完成协调。	设计上是“一锤子买卖”，难以与其他 Skill 联动。

如何写好 Skills

1. **原子性（Atomicity）：**坚持单一职责，让每个 Skill 都像一块积木，小而美，专注于解决一个具体问题，便于日后的复用和组合。
2. **给例子（Few-Shot Prompting）：这是最关键的一点**，与其费尽口舌解释，不如直接给出几个清晰的输入输出示例。榜样的力量是无穷的，模型能通过具体例子，秒懂你想要的格式、风格和行为。
3. **立规矩（Structured Instructions）：**
    - **1) 定角色：**给它一个明确的专家人设，比如“你现在是一个资深的市场分析师”。
    - **2) 拆步骤：**把任务流程拆解成一步步的具体指令，引导它“思考”。
    - **3) 画红线：**明确告诉它“不能做什么”，防止它天马行空地“幻觉”
4. **造接口（Interface Design）：**像设计软件 API 一样，明确定义 Skill 的输入参数和输出格式（比如固定输出 JSON 或 Markdown）。这让你的 Skill 可以被其他程序稳定调用和集成。
5. **勤复盘（Iterative Refinement）：**把 Skills 当作一个产品来迭代。在实际使用中留心那些不尽如人意的“Bad Case”，然后把它们变成新的规则或反例，补充到你的 Skills 定义里，让它持续进化，越来越聪明、越来越靠谱。

> 📚 一些官方最佳实践指南
>
> https://platform.claude.com/docs/zh-CN/agents-and-tools/agent-skills/best-practices

05

社区热门 Skills 推荐

刚开始接触 Skills，不知从何下手？不妨从社区沉淀的这些热门 Skills 开始，寻找灵感，或直接在你的工作流中复用它们。

Claude 官方提供的 Skills

> 官方 Skills 仓库
> https://github.com/anthropics/skills

学习 Claude 官方的 Skills 仓库可以帮助我们最快的了解 Skills 的最佳实践，便于我们沉淀出自己的 Skills。

> **如何快速使用官方 Skills?**
>
> 大多数官方 Skills 都能直接下载，或者通过 Git 克隆到本地。在 TRAE 等工具里，一般只需把这些 Skills 的文件夹放到指定的 ***Skills*** 目录，接着重启或刷新 Agent，它就会自动识别并加载这些新能力。具体操作可参考工具的使用文档。
>
> 更多细节可参考下面这部分内容：**如何在 TRAE 里快速用起来**

Claude 官方提供的 Skills 列表

社区其他最佳实践

如何在 TRAE 里快速使用

理论说再多，不如亲手一试。我们先讲一下如何在 TRAE SOLO 中创建并应用一个 Skill 并以基于飞书文档的 Spec Coding 为例讲解一下如何利用 Skills 快速解决一个实际问题。

Skill 创建

方式一：设置中直接创建

TRAE 支持在设置页面可以快速创建一个 Skill

按下快捷键 Cmd +/ Ctrl + 通过快捷键打开设置面板。

在设置面板左侧找到「规则技能」选项

找到技能板块，点击右侧的「创建」按钮。

你会看到一个简洁的创建界面，包含三要素：**Skill 名称、Skill 描述、Skill 主体**。我们以创建一个“按规范提交 git commit”的 Skill 为例，填入相应内容后点击「确认」即可。

填入我们需要的内容「确认」即可

方式二：直接解析 SKILL.md

在当前项目目录下，新增目录.***trae/Skills/xxx*** 导入你需要文件夹，和 TRAE 进行对话，即可使用。

可以在「设置 – 规则技能」中看到已经成功导入

方式三：在对话中创建

目前 TRAE 中内置了 Skills-creator Skills，你可以在对话中直接和 TRAE 要求创建需要的 Skills

Skill 使用

在 TRAE 里使用技能很容易，你加载好需要的技能后，只需在对话框中用日常语言说明你的需求就行。

- 例如，输入“帮我设计一个有科技感的登录页面”，系统就会自动调用“frontend-design”技能。
- 例如，输入“帮我提取这个 PDF 里的所有表格”，系统会自动调用“document-Skills/pdf”技能。
- 例如，输入“帮我把这片技术文档转为飞书文档”，系统会自动调用“using-feishu-doc”技能。

系统会自动分析你的需求，加载技能文档，还会一步步指导你完成任务！

实践场景举例

还记得引言里提到的那些问题吗？比如说，项目规则文件（***project_rules***）有字符数量的限制；又或者，就算你在根规则文件里明确写好了“在什么情况下读取哪个文件”，Agent 在执行任务时也不会按照要求来做。

这些问题的根本原因是，**规则（Rules）对于 Agent 而言是固定不变的**，它会在任务开始时就把所有规则一次性加载到上下文中，这样既占用空间，又不够灵活。而 **技能（Skill）采用的是“逐步加载”的动态方式**，刚好可以解决这个问题。所以，我们可以把之前那些复杂的规则场景，重新拆分成一个个独立的技能。

接下来，我们通过一个基于飞书文档的“Spec Coding”简单流程，来实际操作一下如何用技能解决问题。

什么是 Spec Coding?

Spec Coding 提倡“先思考后行动”，也就是通过详细定义可以执行的需求规范（Specification）来推动 AI 开发。它的流程包含“需求分析、技术设计、任务拆解”的文档编写过程，最后让 AI 根

据规范来完成编码。这种一步步的工作流程能保证每一步都有依据，实现从需求到代码的准确转化。

让我来分析一下这个场景

上面提到将开发过程划分为四个关键阶段，所以要完成“需求分析、技术设计、任务拆解”的飞书文档撰写，还有最终的代码实现。为此，我们需要不同的技能来满足不同场景下的文档编写需求，并且要教会 Agent 如何使用飞书工具进行创作协同。

下面我们就一起完成上面提到的 Skills 的设计实现。

多角色专家 Skills

通过实现多角色 Skills 通过创建多个交付物过程文档，约束后续的编码，为编码提供足够且明确的上下文，每个Skill 专注完成一件事

- **下面让我们进一步详细设计**

按照上述的表格我们就可以大致明确我们需要的 Skills 该如何实现了。

- 本次只作为一个例子大家可以参考上面创建 Skill 的教程自己完成一下这个多角色 Skills 的创建和调试，当然正如上面所述好的 Skill 需要在实践中逐渐优化并通过场景调用不断进行优化的

飞书文档使用 Skill

飞书文档的格式是 markdown 的超集，**我们 Skill 的目的则是教会 Agent 飞书文档的语法**，便于 Agent 写出符合格式的 md 文件。并通过约束 Agent 行为，**充分利用飞书文档的评论的读写完成多人协作审阅的过程**，用户通过在飞书文档评论完成相关建议的提出，Agent 重新阅读文档和评论，根据建议进一步优化文档，**实现文档协作工作流。**

Spec Coding Skill

上面我们实现了多个角色 Skills 和一个功能 Skill，但实际使用时，还需要有一个能统筹全局的技能，来实现分工协作。把上述多个技能组合起来，告诉智能体（agent）整体的规格编码（spec coding）流程，完成工具技能和角色技能的组合与调度。

如此我们就能快速搭建一个规格编码工作流程，完成基础开发。当然也可以参考上面的逻辑，用技能来重新复刻社区里的规格编码实践（如 SpecKit、OpenSpec 等）。

总结

上述场景提到了两种不同风格的 Skill（角色型，工具型），利用 **Skill 的动态加载机制**（取代固定规则的一次性加载方式），完成了复杂场景下的任务分解；通过 **不同角色技能的分工协作**（避免 Agent 什么都做导致执行混乱）；尝试借助**飞书文档形成协作闭环**（打通人机交互的最后一步），有效解决了 Agent“不听话、执行乱、工具少”的问题，让 AI 从“对话助手”真正转变为“可信赖的实干家”，实现从需求提出到代码产出的高效、精准、协作式交付。

Q & A｜一些常见问题

> **为什么我写的 Skills 不生效，或者效果不符合预期?**

那十有八九是你的“名片”（*Description*）没写好。

记住，Agent 是通过读取 Skills 的 ***Description*** 来判断“什么时候该用哪个 Skill”的。要是你的描述写得含糊不清、太专业或者太简单，Agent 就很难明白你的意思，自然在需要的时候就不会调用这个 Skill。所以，**用大白话写的清晰、准确的*Description*，对 Skill 能否起作用至关重要。**

> **使用 Skills 的效果，会受到我选择的大语言模型（LLM）的影响吗?**

会有影响，不过影响的方面不一样。

- **一个更强大的模型，主要影响“挑选”和“安排”技能的能力。** 它能更准确地明白你的真实想法，然后从一堆技能里挑出最适合的一个或几个来解决问题。它的优势体现在制定策略方面。

- **而技能本身，决定了具体任务执行的“最低水平”和“稳定性”。** 一旦某个技能被选中，它里面设定好的流程和代码是固定的，会稳定地运行。所以，技能编写得好不好，直接决定了具体任务能不能出色完成。。

Skills 是不是万能的？有什么它不擅长做的事情吗？

当然不是万能的。Skills 的主要优势是**处理那些流程明确、边界清晰的任务。**在下面这些情况中，它可能就不是最好的选择了：

- **需要高度创造力的任务：**像写一首饱含情感的诗，或者设计一个全新的品牌标志。这类工作更需要大模型本身的“灵感”。
- **需要实时、动态做决策的复杂策略游戏：**比如在变化极快的金融市场中做交易决策。
- **单纯的知识问答或开放式闲聊：**如果你只是想问“文艺复兴三杰是谁？”，直接问大模型就可以，不用动用 Skills 这个“大杀器”。

我发现一个社区的 Skills 很好用，但我可以修改它以适应我的特殊需求吗？

当然可以，我们强烈建议你这么做！

大多数共享的 Skill 都支持用户“Fork”（也就是“复制一份”）并进行二次开发。你可以把通用的 Skill 当作模板，在自己的工作空间里复制一份，然后修改里面的逻辑或参数，以适应你自己的业务需求。这对整个生态的共建和知识复用很重要。

结语｜让 Agent 成为你真正的“行动派”

Skill 的出现，为 AI 从“对话式助手”转变为“可信赖的执行者”搭建了关键的技术桥梁。它用结构化的方法把领域知识、操作流程和工具调用逻辑封装起来，解决了 Agent 规则失效、执行失控的混乱问题，让 AI 的能力输出变得可以控制、值得信赖且高效。

Skill 的核心价值在于：

- **精准实际痛点：**通过巧妙的三级加载机制（元数据→说明文档→资源）平衡上下文效率与功能深度，在功能深度和上下文效率之间找到了一个绝佳的平衡点，既避免了宝贵 Token 的浪费，又确保了任务执行的精准性，实现了 Agent 上下文的动态加载能力。
- **生态赋能，降低门槛：**无论是官方还是社区，都提供了丰富的资源（如 Claude 官方仓库、SkillsMP 市场等），让普通用户也能轻松站在巨人的肩膀上，快速复用各种成熟的能力。

虽然 Skill 不是万能的，但它在“确定性流程任务”上的优势无可替代。未来，随着 AI 模型能力的提升与 Skill 生态的进一步完善，我们有望看到更多跨领域、可组合的 Skill 出现——**让 AI 从“样样懂一点”的通才，真正进化为“事事做得好”的专家协作伙伴。**

不妨从今天开始，尝试创建你的第一个 Skill：将你最擅长的领域经验封装成可复用的能力，让 AI 成为你延伸专业价值的放大器。