470万阅读续篇！三大功能让 Claude Code 脱胎换骨

> 你好啊，我是阿杰，一个探索普通人 AI 破局之路的程序员，欢迎来到我的进化日常。

前一天写了那篇 470 万阅读的 Claude Code 教程后，作者 Eyad 又放出了第二波内容。

这次主要讲三大高级功能：**Skills、Subagents 和 MCP**，我们这篇文章就来简单说下。

上下文窗口上限不是都一样

你可能以为所有 AI 编程工具的上下文处理都差不多，但实际上差别很大。

根据 Qodo 工程团队的详细对比，Claude Code 提供"更可靠、明确的 200K token 上下文窗口"，而 Cursor 的"实际使用往往达不到理论 200K 上限"——因为它会为了性能或成本管理做内部截断。

如果你在处理大型代码库，**对于大型项目，相比 Cursor 作者更建议直接用 Claude Code**，它更能稳定交付。

不过，这里说的是相对的，我觉得真正的开发过程中模型的好坏也占据重要原因，好模型不降智就已经很好了。

Skills：把你的工作流教给 Claude

Skill 就是一个 Markdown 文件，用来教 Claude 怎么做你特定的工作。

当你的问题匹配某个 Skill 的用途时，Claude 会自动应用它。

关于 Skill 可以看我这些文章：

- **加上这篇，终于敢说把 Claude Skills 彻底讲清楚了**
- **史无前例！Claude Code一周更新连发5个版本，快来看看更新了什么?**
- **我把全网最火的标题公式封装成了 Skill，告别标题焦虑**
- **我在 Claude Code 里装了12个 AI 专家 Skills，小白也能写出专业级提示词！**

Skill 关键架构原则是**渐进式披露**。

Claude 启动时只预加载每个 Skill 的名称和描述（大约 100 个 token）。

完整指令只在 Claude 判定相关时才加载，这意味着你可以有几十个 Skill 而不会膨胀上下文。

Skill 不限于代码。作者见过的工程师用它做过：

- 特定数据库模式的查询模式
- 公司的 API 文档格式
- 会议纪要模板
- 甚至个人工作流比如餐饮规划或旅行预订

任何你发现自己需要反复向 Claude 解释的上下文或偏好，都可以做成 Skill。

Subagents：独立上下文的并行处理

Subagent 是一个独立的 Claude 实例，有自己的上下文窗口、系统提示词和工具权限。

当 Claude 委托给 subagent 时，该 subagent 会独立运行，然后把结果摘要返回主对话。

Subagents 让你把复杂研究或实现任务卸载到全新上下文，只带回相关结果——你的主对话保持干净。

Claude Code 有三个内置 subagent：

- **Explore**：快速只读 agent，用于搜索和分析代码库。Claude 需要理解代码但不做改动时会委托到这里。正确使用时会指定 thoroughness：quick、medium 或 very thorough。
- **Plan**：规划模式中使用的研究 agent，在展示计划前收集上下文。它调查你的代码库并返回发现，让 Claude 做出明智的架构决策。
- **General-purpose**：能干的 agent，用于需要探索和行动的复杂多步骤任务。任务需要多个依赖步骤或复杂推理时，Claude 会委托到这里。

创建自定义 Subagent

作者强烈建议创建自己的自定义 subagent，运行 `/agents` 查看所有可用的并创建新的。

手动创建的话，在 `~/.claude/agents/`（用户级，所有项目可用）或 `.claude/agents/`（项目级，团队共享）添加一个 markdown 文件：

```
---
name: security-reviewer
description: 审查代码安全漏洞。检查认证问题、注入风险或数据暴露时调用。
tools: Read, Grep, Glob
---

你是一个专注安全的代码审查员。分析代码时：
1. 检查认证和授权缺口
2. 查找注入漏洞（SQL、命令、XSS）
3. 识别敏感数据暴露风险
4. 标记不安全的依赖

为每个发现提供具体的文件和行引用。按严重性分类：critical、high、medium、low。
```

- `tools` 字段控制 subagent 能做什么。
- 只读审查员就限制在 read、grep 和 glob 命令。
- 实现 agent 就包括 write、edit 和 bash 命令。

Subagents 怎么通信

Subagents 不直接共享上下文，因为它们在隔离中运行，通信通过**委托和返回模式**：

```
主 agent 识别适合委托的任务
↓
主 agent 用描述任务的特定提示词调用 subagent
↓
Subagent 在自己的上下文窗口中执行
↓
Subagent 把发现/操作的摘要返回给主 agent
↓
主 agent 整合摘要并继续
```

摘要才是关键，设计良好的 subagent 不会把整个上下文倒回来，这就是为什么 subagent 描述和系统提示词需要明确输出格式。

对于复杂工作流，你可以**链式调用 subagents**。主 agent 来编排，比如文章举的例子：

```
主 Agent
 |── 委托研究给 Explore subagent
 |    └── 返回："找到 3 个相关文件：auth.py, middleware.py, routes.py"
 |── 委托实现给自定义 implementer subagent
 |    └── 返回："添加了密码重置端点，更新了 2 个文件"
 └── 委托测试给自定义 test-runner subagent
      └── 返回："12 个测试全部通过，覆盖率 94%"
```

每个 subagent 为自己的特定任务获得全新上下文，主 agent 只持有摘要结果，无需完整的探索历史，防止了长会话的上下文污染。

一个重要约束：**subagents 不能生成其他 subagents**。这防止无限嵌套，保持架构可预测。

Subagent 实战

- **大型重构**：让主 agent 识别所有需要变更的文件，然后为每个逻辑组启动一个 subagent。每个 subagent 处理自己的范围并返回摘要，主 agent 从不需要同时持有每个文件的完整上下文。
- **代码审查流水线**：创建三个 subagent：style-checker、security-scanner、test-coverage，对一个 PR 并行运行。每个返回一致格式的发现 → 主 agent 合成为单个审查。
- **研究任务**：当你需要理解代码库的陌生部分时，用具体问题委托给 Explore。它返回相关文件和模式的精简梳理，让你的主上下文专注于实际实现工作。

MCP Connectors

MCP（Model Context Protocol）是 AI 模型通过统一接口调用外部工具和数据源的标准方式，不需要为每个工具做定制集成。

过去6个月 MCP 服务器为作者做过的一些事：

- 从 issue tracker 实现功能："添加 JIRA issue ENG-4521 中描述的功能"
- 查询数据库："从我们的 PostgreSQL 数据库找到上周注册的用户"
- 集成设计："基于新的 Figma 设计更新我们的邮件模板"
- 自动化工作流："创建 Gmail 草稿邀请这些用户参加反馈会话"
- 总结 Slack 线程："团队在 #engineering 频道关于 API 重新设计决定了什么？"

一个原本需要五次上下文切换的工作流（查 issue tracker、看设计、审查 Slack 讨论、实现代码、更新 ticket），通过 MCP 现在在一个连续会话中完成

组合效应

一个懂你代码库模式的 **Skill** + 一个处理测试的 **Subagent** + 连接你 issue tracker 的 **MCP** = 一个无敌的系统。

- Skill 编码你团队的规范，你不用担心上下文。
- Subagents 在处理复杂子任务时保持主对话干净。
- MCP 连接消除分散你注意力的上下文切换。

一开始可能不知道怎么结合做事情，但如果你不知道从哪里开始，就从一个创建 Skill 开始，针对某个你反复解释的东西。

总结

最后我们小小总结下：

- **Skills** 把具体固定的工作流做成Skill，不需要每次都解释
- **Subagents** 让复杂任务并行处理，独立上下文不污染主会话
- **MCP** 把你从频繁的上下文切换中解放出来，将不同工作流程服务串联成一个连续会话

但目前看，还是那句话，能用 Skill 的优先用 Skill 解决。

如果你是为了个人提效，Skill 也是最容易生成和使用的。

至于 SubAgents 和 MCP，未来这三者多多少少会有重叠的地方，但随着社区 Skills 的爆发和流程，未来 Skills 将会逐渐抹平差异。

推特原文：https://x.com/eyad_khrais/status/2010810802023141688

> □□□ 我是阿杰，一个 All in AI 的前端摸鱼工程师，专注分享普通人拿来即用的**AI 实战**与**副业变现**。 如果觉得文章有用，欢迎**「关注 + 星标」**，交个朋友，一起进化。