别让 Claude Code 越用越变笨,一招保持“天才模式”

如果你使用Claude Code 或 Codex 的频率比较高的话，你一定遇到过这种情况:

刚开始用 Claude Code 的时候,它简直像个天才程序员,一句话就懂你要什么,代码写得又快又准。

但聊了半小时之后,画风突变——开始犯低级错误,甚至忘记你最开始说的需求。

你怀疑是不是模型抽风了?

不,是你的上下文炸了。

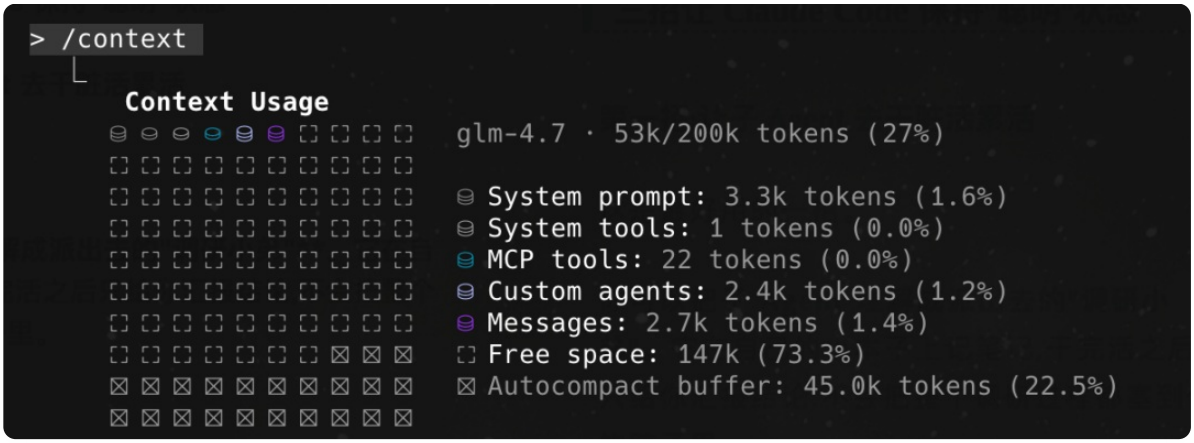
上下文就是 AI 的"工作记忆"

咱们先说清楚一个概念:**上下文 (Context) 就是 AI 能记住的全部对话内容**。

Claude Code 默认使用的模型,总上下文容量是 ** 20-25 万 token**(大约相当于一本中篇小说的字数)，但给官方也声称给到一些商业渠道可以到1M，但普通渠道基本上最大的上下文也就是20-25w了。

听起来很多?

但你要知道,**Claude Code 自带的系统提示词和工具定义,开局就吃掉了 10% 的容量**。还没开始干活,你的"内存条"就少了一块。



context

更要命的是:**当上下文超过 25 万 token 之后,模型会明显变笨**。

这不是玄学,是实测出来的，很多社区都有人做个类似的测试并得到验证。超过这个阈值,Claude 4.5 的回答质量会断崖式下跌,就像人的大脑超负荷运转一样,开始反应迟钝、频繁出错。

context 消耗进度条

所以,**最优策略是默认只有 25 万 token 可用,省着点花**。

为什么你的上下文总是不够用?

很多人用 Claude Code 的姿势是这样的:

让它调研一个大型项目 → 搜索代码、读文件、分析结构 → 所有中间结果、代码片段全塞进上下文。

等调研完,**上下文可能就烧掉了 10 万,差不多消耗一半了**。

还没开始写代码呢,AI 已经开始"失忆"了。

更糟糕的是,很多人遇到 bug 之后的第一反应是:**把报错信息复制给 Claude,让它继续改**。

这就像你已经累得不行了,还要继续加班,能不出错吗?

上下文已经很长的情况下,模型处于"不聪明"状态,继续聊只会 bug 越改越多。

三招让 Claude Code 保持"聪明"状态

第一招:让子 Agent 去干脏活累活

这是最好用的一招。

你可以把**子 Agent 理解成派出去的"调研小弟"**。它在自己的小本子上记笔记,干完活之后只给你汇报结论,不会把整个调研过程都塞到你的脑子里。

举个例子:

你让 Claude Code 调研一个项目,它启动一个子 Agent 去搜索代码、读文档、分析结构。

子 agent 启动

子 Agent 工作时可能用了 **8.4 万 token**,但它只告诉主 Agent:"这是一个 xxx 项目,文件结构是 yyy,有问题的代码是 zzz。"

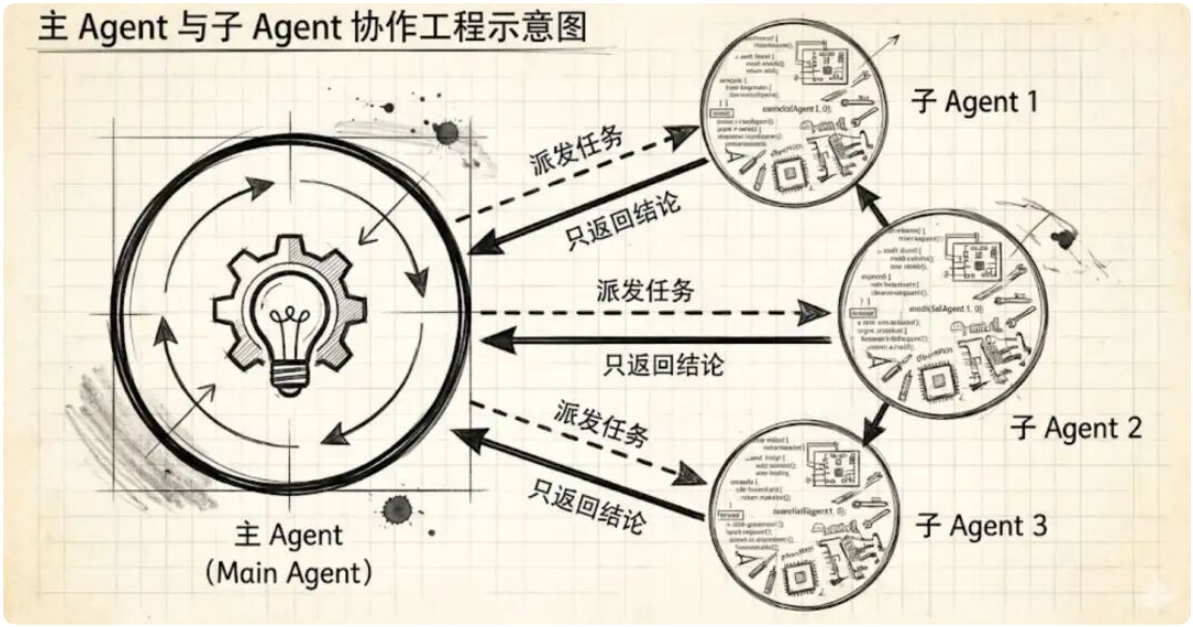
主对话的上下文只增加了一点点,但已经拿到了想要的信息。

所以,**凡是"查资料"性质的工作,都让子 Agent 去干**:

- 搜索代码
- 阅读文档
- 学习库的用法
- 调研项目结构

这些任务的中间过程都是"上下文垃圾",不该留在主对话里。

而且子 Agent 可以并发。找 5 个功能的代码?同时启动 5 个子 Agent,比主 Agent 一个个串行快得多。



主子 agent 交互

第二招:约束 AI 的输出字数

Claude Code 默认会输出很详细的解释,但**如果你已经懂了,这些解释就是浪费**。

解决方法很简单:在项目根目录的 `CLAUDE.md` 文件里加一句:

这一句话能显著延长主对话的有效寿命。

第三招:多用 ESC 回退,少用 /compact 压缩

当 Claude Code 写出 bug 的时候,**最好的办法不是把报错发给它,而是直接按 ESC 回退到修改之前,重新跑一遍**。

为什么?

因为上下文已经很长了,模型处于"不聪明"的状态,继续聊可能 bug 越改越多。

你可能会问:**回退之后重新跑,不也是一样的代码吗?**

不一样。**大模型输出有随机性**,上一轮走进死胡同,这一轮可能换条路就通了。

而且回退的时候,你可以在输入里补一句:"刚才试过 xx 方案,有 xx 问题,别走这条路。"

相当于提前帮模型排雷,成功率会更高。

那 `/compact` 命令呢?
很多人上下文用满 25 万之后,第一反应是用 `/compact` 压缩。

但压缩后,**AI 对之前的内容记得不全**。如果它需要用到之前的信息,还得重新读一遍原代码或原文档。

那既然都要重新读,**压缩的那段总结在上下文里就是废话了**。

不如直接新开一个窗口,干净利落。

一个窗口干完一件事就新开

这是最简单但最有效的习惯。

修好了一个 bug,或者加完了一个新功能,马上做两件事:

1. **提交代码**
2. **新开窗口**

不要在同一个窗口里干多件事。**上下文越聊越长,模型越来越笨,效率会断崖式下降**。

保持窗口"新鲜",就是保持模型"聪明"。

官方也在优化,但主动管理更重要

Anthropic 自己也知道上下文管理很重要,所以官方一直在做优化,比如:

- 读过的文件,模型觉得不再需要后会自动"卸载",等再需要的时候重新读
- 启动 plan 模式的时候,模型会自己派子 Agent 去查资料,而不是把调研过程塞进主对话

官方这些优化确实有效,但它们是被动的。

等官方帮你压缩的时候,模型已经变笨了。

所以,**主动管理上下文,才能让模型一直保持在最聪明的状态**。

说到底,用 AI 编程工具就像开车——你不能只踩油门,还得学会换挡、刹车。

懂不懂上下文管理,用的完全是两个 Claude Code。

现在你知道怎么让它保持"天才模式"了吧?

Agent Skills 编写葵花宝典

Agent Skills 到底是个啥？为什么你的 AI 智能体需要它？

Anthropic 放了个大招：Agent Skills 开源，企业 AI 的玩法真的要变了

公众号排版神器：免费 + 单文件 + 本地运行

别问skill怎么写，生产级skill 直接喂到你嘴里

Claude Code 悄悄放大招：一个人指挥一支 AI 军团

Claude Code 内一键调用 Gemini CLI，我把多模态彻底打通（超长上下文 + 免费额度）

如果觉得这篇文章对你有启发，请一键三连，分享给更多正在学习 AI 的朋友。

我是AIGC 胶囊，在这个快节奏的AI时代，我想陪你走得稳一点。不分享我没用过的，不推荐我没验证的。我把踩过的坑填平，把验证过的路铺好，只为了让你在应用AI时，少走弯路，多拿结果。