



如何用 Claude Code 提升开发效率

谢威宇 Madsys

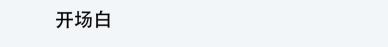
2025年10月





目录

- 1. 开场白
- 2. Claude Code 是什么
- 3. 基础功能
- 4. 高级功能
- **5.** 真实案例
- 6. 最佳实践
- 7. 挑战与解决方案
- 8. 核心思考
- 9. Q&A



我的 Claude Code 使用体验

今年最幸运的一件事

用上 Claude Code 是我今年最幸运的事。它给我的震撼不亚于、甚至超过了当初初次使用 ChatGPT 3.5 时的感受。

使用时间:从8月至今约2个月

完成的项目:

- 系统级代码开发
- 运维自动化
- 各类小工具
- 项目管理工作

全新的协作范式

AI 终于能和人站在同一信息起跑线上,充分发挥现代 AI 的真正实力

性价比问题

虽然每月 200 美元, 但是仍然觉得值得



Claude Code 定位

核心功能

- AI 驱动的编程助手
- 命令行界面操作
- 深度集成文件系统
- 自主执行任务

和其他工具的区别

• vs IDE: 理解更智能

• vs Copilot: 能完成整个任务

• vs ChatGPT: 直接操作代码

最大优势

基于命令行和文件系统,能拿到海量的上下文信息

人机协作的本质

效率瓶颈在哪

• 人类短板: 打字和说话的速度远比不上 AI

• 人类优势: 强化学习(RL)效率远超 AI

• AI 优势: 处理海量背景信息

最优协作模式

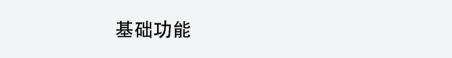
AI 负责处理大量背景信息,人类根据实际情况提供新反馈和想法

Claude Code 的优势

基于命令行和文件系统工作,获取上下文信息的能力极强

三大核心优势

- 1. 全新的协作范式
 - AI 和人终于站在同一信息起跑线上
 - 充分发挥现代 AI 的真正实力
- 2. 真正能干活
 - 不只是提建议,而是直接动手
 - 自动拆解多步骤任务
- 3. 工具集成能力强
 - 编辑器、终端、Git、包管理器等一应俱全
 - 在不同工具间无缝切换



快速理解大项目

- 自动分析项目结构
- 找出关键组件
- 理清代码依赖关系

智能搜索

- 语义化搜索代码
- 跨文件关联分析
- 快速定位具体实现

- 智能补全
 - 基于上下文给出精准建议
 - 理解项目的代码风格
- 重构和优化
 - 自动发现可优化的地方
 - 安全地重构代码
- 自动修 bug
 - 检测和诊断错误
 - 给出修复方案



项目管理

Git 操作

- 自动提交
- 管理分支
- 创建 PR

依赖管理

- 装包
- 版本控制
- 解决冲突

构建测试

- 自动构建
- 跑测试
- 配置 CI/CD

Todo 列表

自动创建和追踪任务,确保不漏掉任何步骤

拆解任务

把复杂任务拆成一个个小步骤,逐步搞定

并行处理

识别可以同时做的任务,提高效率

调试和优化

1. 诊断和修复

- 快速找到错误源头
- 理解错误的来龙去脉
- 给出修复建议

2. 性能优化

- 找出性能瓶颈
- 提供优化方案
- 实施并验证改进效果

3. 代码质量

- 检查代码规范
- 建议最佳实践
- 审查安全性



案例 1: 快速看懂代码 - AWQ 量化格式研究

任务

在 sglang 代码库里找到 AWQ 量化的 swizzle 格式是怎么排列的

技术背景

- AWQ: 4bit 量化格式
- 不按行主序/列主序排
- 用 Swizzle 顺序排列权重
- 论文里说得不清楚

效率对比

- 用 Claude Code: 几分钟
- 传统方式: 几小时
- 对代码库完全不熟的情况下
- 需要深入理解底层实现

核心价值

快速看懂陌生代码库,精准定位关键实现

案例 2: 系统级开发 - KTransformers 支持 AWQ

任务

实现 group 量化 (group size = 64)

技术难点

- K 维度需要分块
- 高维矩阵切分和 pack
- Per-channel 改 Per-group
- AVX 指令优化

协作流程

- 1. AI 先写测试用例
- 2. 参考原方案改代码
- 3. 生成基本框架
- 4. 人工调 AVX 指令
- 5. 成功跑通

效率提升

项目周期从 2 周 缩短到 2 天

案例 3: 运维 - GPU 监控中心部署

背景

多个 GPU 集群要管理,还得解决 GPU 抢占和资源分配的问题

技术方案

- Node Exporter (系统监控)
- NVIDIA DCGM Exporter (GPU 监控)
- Grafana (可视化)
- 自己写的 Dashboard

做了什么

- 自动化部署脚本
- Docker 容器化
- 批量远程部署
- 聚合监控数据
- 多机器 GPU 看板

完全 Video Coding

4 小时搞定 vs 传统方式 2-3 天

主要时间花在解决 Docker 源和依赖问题上

案例 4: 文档工程 - 立项文档自动化

场景

KTransformers 和华为合作的立项文档

传统方式的痛点

- 疯狂复制粘贴
- 反复调格式
- 信息整合麻烦
- 版本同步困难

用 Claude Code 的流程

- 1. 准备背景资料和模板
- 2. 转成 Markdown
- 3. AI 生成初稿
- 4. 在 MD 里调整
- 5. 复制到 Word 里

效果

写文档变得又快又轻松,专注内容而不是格式

案例 5: 管理自动化 - 项目进度管理系统

需求

多个项目要对接同步,还得向上汇报、追踪进度

解决方案

- 自动化进度更新
- 多项目信息聚合
- 自动生成日报
- 自动生成周报
- 数据可视化

带来的价值

- 减少重复劳动
- 汇报更准确
- 节省管理时间
- 方便追踪进度
- 管理效率大幅提升

关键洞察

把管理工作系统化、自动化,腾出时间来思考



怎么问问题

1. 清楚地描述

- 说清楚想要什么结果
- 描述现在是什么状态
- 贴上错误信息

2. 给足上下文

- 指出相关文件
- 说明用的什么技术栈
- 讲清楚有什么限制

3. 一步步来

- 把复杂任务拆开
- 每步都验证结果
- 及时给反馈

优化工作流

工具集成

- Git
- IDE
- 终端
- CI/CD

团队协作

- 代码规范
- 文档同步
- 知识共享
- 最佳实践

提效技巧

- 快捷命令
- 用模板
- 批量操作
- 自动化脚本



上下文限制问题

现状

Claude Code 的上下文一般在几百 K 级别,开发大项目时可能一个特性就用完

解决方案

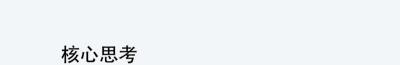
- 让 AI 写文档记录状态
- 定期重构项目结构
- 模块化开发
- 分阶段处理任务

为什么不能简单扩展

- 长上下文成本太高
- 信息检索效率降低
- 多 agent 是更优解

未来趋势

多 agent 系统会成为主流(Haiku 4.5 已采用)



Claude Code 的本质

信息处理加速器

任何需要快速处理和输出信息的工作都可以交给它

优化流程的思维

人负责思考整个流程哪里可以优化

解放生产力

从重复的信息处理工作中解放出来,专注创造性思考

性价比分析

成本

• 月费: 200 美元

• 学习成本: 很低

• 适应时间: 很短

总投入

每月 200 美元 + 一点学习时间

收益

• 效率提升: 5-10 倍

• 能力扩展: 显著

• 项目交付: 更快

• 收入潜力: 大幅提升

性价比

超值,简直是捡了大便宜

价格问题与解决方案

价格确实是个问题

每月 200 美元对个人用户确实是不小的负担

国内使用的实际问题

- 缺乏合适的支付手段
- 需要跨境支付机构
- 可以考虑和同学拼单

对不同用户的价值

• 个人: 需权衡投入产出

• 公司: 效率提升值得投资

• 学生: 可以拼单降低成本

更经济的替代方案

Claude Code 支持多种模型

除了 Anthropic 官方模型,还可接入其他模型

GLM-4.6

- 开源模型
- 价格: 60 元/季度
- 在 BFCL 等 function call 榜单第一
- 日常使用基本没问题

Kimi K2

- 模型尺寸更大
- 性能更强
- 价格介于两者之间

推荐

如果觉得 Claude 价格太贵, 可以考虑用 GLM-4.6

团队成本分摊方案 - Claude Relay Service

核心价值

- 自托管 API 中继服务
- 团队共担订阅费用
- 多账户统一管理
- 使用统计透明化

适用场景

- 学生拼单降低成本
- 团队协作开发
- 绕过地理访问限制
- 保护数据隐私

成本示例

5 人团队: 每人 40 美元/月

技术栈

Node.js + Redis + Docker

注意

需一定运维能力,可能与 Anthropic 服务条款冲突

开源项目: https://github.com/Wei-Shaw/claude-relay-service

AI 能力的本质思考

框架也是能力的一部分

Claude Code 证明了 AI 的能力不完全由基础模型决定

传统观点

- AI 能力 = 模型能力
- 更大的模型 = 更强的能力
- 追求参数规模

新的认知

- 工作范式很重要
- 框架设计是核心
- 多 agent 协作

实证

Haiku 4.5 发布后,看代码功能完全用子 agent 实现

对程序员角色的影响

技能广度 vs 技能深度

Claude Code 显著增加了程序员的技能广度

- 1. 分工方式的变化
 - 从技能出发 → 从需求出发
 - 更接近产品经理的工作方式
- 2. FAE 角色越来越重要
 - 现场应用工程师 (Field Application Engineer)
 - 理解客户需求
 - 转化成 AI 能处理的形式
- 3. 变革正在发生
 - 从一个人用到许多人用
 - 当共识形成,下一步变革就会开始

未来展望

Claude Code 的进化

- 理解能力更强
- 集成更多工具
- 协作体验更好
- 任务执行更智能
- 多 agent 系统成熟

AI 协作范式的影响

- 改变开发方式
- 提升生产力水平
- 降低技术门槛
- 加速创新速度
- 重塑团队结构

愿景:解放更多人的生产力

- 1. Claude Code 带来了全新的协作范式, AI 和人终于站在同一信息起跑线上
- 2. 最优协作模式: AI 处理背景信息 + 人类提供反馈和想法
- 3. 从多个真实案例看:效率提升 5-10 倍是常态
- 4. 月费 200 美元的投资带来巨大回报
- 5. 任何需要快速处理和输出信息的工作都可以交给 Claude Code

感谢 Claude Code, 让我有能力实现我的各种想法,希望未来它会带来进一步的变革。



Q&A

提问与讨论

感谢聆听!

谢谢!

本演示文稿使用 Claude Code 制作