



# 如何用 Claude Code 提升开发效率

谢威宇

Madsys & 趋境科技

2025 年 10 月



# 目录

---

1. 开场白
2. 基础概念：什么是终端
3. **Claude Code** 是什么
4. 核心能力概览
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**

## 开场白

---

# 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

## 核心价值

让我有能力实现各种想法，效率提升 5-10 倍

## 基础概念：什么是终端

---

# 终端是什么?

---

## 通俗理解

- 就像手机上的“文件管理器”
- 用文字命令控制电脑
- 程序员的“工作台”
- 比图形界面更强大、更快

## 日常类比

- 图形界面 = 用遥控器看电视
- 终端 = 直接对电视说话

## 终端能做什么

- 创建、移动、删除文件
- 运行程序
- 查看系统信息
- 批量处理任务

## 示例

```
cd project - 进入项目文件夹  
ls - 列出所有文件  
python app.py - 运行程序
```

# 为什么 Claude Code 基于终端?

---

## 1. 信息密度高

- 终端能看到文件结构、代码内容、运行结果
- 一个屏幕包含海量信息
- AI 可以快速理解整个项目状态

## 2. 操作能力强

- 可以直接修改文件、运行代码
- 不需要鼠标点来点去
- 一条命令完成复杂操作

## 3. 程序员的自然环境

- 程序员本来就在终端工作
- AI 和人在同一个“工作台”上协作
- 真正站在同一信息起跑线

## Claude Code 是什么

---



# 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

# 核心优势

---

## 最优协作模式

- AI: 处理海量背景信息
- 人类: 提供反馈和创意
- 基于文件系统, 上下文获取能力极强

## 真正能干活

- 不只是提建议, 直接动手
- 自动拆解多步骤任务

## 工具集成

- 编辑器、终端
- Git、包管理器
- 构建、测试工具
- 无缝切换

## 本质

AI 和人站在同一信息起跑线, 充分发挥 AI 实力

## 核心能力概览

---

# Claude Code 能做什么

---

## 代码开发

- 理解项目结构和代码
- 智能补全和重构
- 诊断和修复 bug
- 性能优化

## 项目管理

- Git 操作（提交、分支、PR）
- 依赖和构建管理
- 自动化测试

## 任务协作

- 自动拆解任务
- 追踪执行进度
- 并行处理优化

## 重点

具体能力通过真实案例来展示更直观

## 真实案例

---

# 案例 1：代码分析 - AWQ 量化格式研究

## 任务

在 sglang 代码库里找到 AWQ 量化的 swizzle 格式排列方式

## 挑战

- 陌生代码库
- 非标准排布方式
- 论文描述不清

## 效率

- Claude Code: 几分钟
- 传统方式: 几小时

## 价值

快速理解陌生代码，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
- 高维矩阵切分
- AVX 指令优化

效率提升

2 周 → 2 天

### 协作

- AI：写测试、生成框架
- 人：调试关键指令

启示

复杂系统级开发也能高效完成

## 案例 3：运维 - GPU 监控中心部署

---

### 任务

- 多 GPU 集群管理
- DCGM + Grafana
- Docker 容器化
- 批量远程部署

### 效率

4 小时 vs 2-3 天

### 价值

- 运维自动化
- 快速部署



## 案例 4：文档工程 - 立项文档自动化

### 场景

KTransformers 与华为合作立项文档编写

#### 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

#### Claude Code workflow

- 背景资料 + 模板 → MD
- AI 生成初稿
- 在 MD 里调整
- 复制到 Word

### 效果

文档编写又快又轻松，专注内容而不是格式

## 最佳实践

---

# 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 workflow

---

## 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

## 挑战与解决方案

---

# 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

- 单模型无限长上下文不合理
- 即使能实现，成本也极高
- 多 agent + 总结 token 是更优解
- 信息检索效率也会降低

## 未来趋势

多 agent 系统会成为主流 (Haiku 4.5 已采用)

## 核心思考

---

# 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



# 成本与价值

---

## 官方订阅

- 月费：200 美元
- 效率提升：5-10 倍
- 学习成本低

## 性价比

收益远超成本

## 实际挑战

- 个人负担较重
- 支付手段受限
- 可考虑拼单

## 建议

公司投资 / 团队分摊 / 用替代方案

# 经济替代方案

---

## GLM-4.6

- 60 元/季度
- Function call 榜单第一
- 日常使用足够

## Kimi K2

- 性能更强
- 价格适中

## 团队分摊

- 5 人：40 美元/月
- Claude Relay
- 需运维能力

## 推荐

预算有限优先 GLM-4.6，团队可考虑分摊方案

Claude Relay Service: <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

# 多 Agent 架构成为必然

## 为什么需要多 Agent?

### 1. 上下文污染问题

- 一个模型处理所有任务
- 不同任务的上下文混杂
- 影响推理质量

### 2. 性价比优化

- 小模型：快速阅读
- 大模型：深度思考
- 针对性使用资源

## 核心

上下文隔离 + 成本优化 = 多 Agent 必然趋势

## Claude Code 的实践

- Explore Agent：快速搜索
- Main Agent：任务编排
- Sub Agents：专项处理
- 各 Agent 独立上下文

## 对基础设施的要求

- 多模型并发调度
- 高效的上下文切换
- Agent 间通信机制
- 统一的资源管理
- 上下文总结与压缩

## 市场影响

# 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

# 商业化思考：从 Killer App 到生态

## Claude Code 的历史地位

第一个真正意义上的 AI Agent Killer App

### 成功的证明

- 程序员领域：代码开发
- 办公领域：文档处理
- 真正能干活的 AI
- 效率提升 5-10 倍

### 必然的扩散

- 各行各业都有类似需求
- 医疗、法律、金融、教育、设计...
- 具体形态还不确定
- 但趋势已经清晰

### 核心逻辑

如果在这些领域能成为 Killer App，其他领域也可以

### 商业机会

其他领域会出现类似应用，具体形态还在探索中

# 市场扩张: Token 需求爆炸性增长

## 对话 vs Agent 应用

- 对话应用: 几百到几千 tokens
- Agent 应用: 几万到几十万 tokens
- 持续的上下文处理
- 大量文件读写操作

## 市场扩大

每次交互 token 消耗是对话应用的  
10-100 倍

## 规模效应

- 垂直领域扩散  $\times$  Token 倍增
- AI Infra 市场指数级扩张

## 用户行为差异

- 倾向使用原始 session
- 不频繁新开 session
- 上下文持续累积

## 系统压力

- 平均上下文长度显著增加
- 显存占用大幅提升
- KV Cache 管理复杂度增加

## 基础设施挑战

长上下文场景成为常态, 不是例外

# SLO 目标的根本性变化

## 对话应用的 SLO

- 关注 Prefill 延迟
- 关注 Decode TPS
- 用户盯着屏幕等输出
- 逐字显示很重要

## 优化目标

首 token 时间 + 每秒输出 tokens 数

## Agent 应用的 SLO

- 用户不等在屏幕前
- 只关心最终结果
- Prefill/Decode TPS 概念淡化
- 任务完成时间才是核心

## 优化目标

端到端完成时间 (End-to-End Latency)

## 架构影响

- 可以接受更高的 Prefill 延迟
- 换取更好的资源利用率
- 批处理优化空间更大



# Agent 应用：爆发前夜

## Claude Code 的启示

- 第一个 Killer App
- 处于爆发前夜
- 主要用户：程序员

## 更广阔的需求

- 其他人也需要智能 agent
- 知识库管理
- 专业领域应用
- 各行各业的效率工具

## 市场机会

Agent 应用爆发 → AI Infra 需求爆发式增长

## 基础设施需求

- 多模型调度平台
- 高性能推理引擎
- 成本优化方案
- 企业级部署

## 趋境科技

在 AI Infra 领域占有一席之地

# 总结与展望

---

## 核心价值

- 全新协作范式
- 效率提升 5-10 倍
- 适用信息密集型工作

变革正在发生

从开发、运维到管理，工作方式正在改变

## 未来方向

- 多 agent 系统成熟
- 降低技术门槛
- 重塑团队结构

愿景

解放更多人的生产力

**Q&A**

---

# Q&A

提问与讨论

感谢聆听！

# 谢谢！

本演示文稿使用 Claude Code 制作