河马的CC商业实战 · 会员专属资料 #4

Superpowers 插件 · 从零到精通

Claude Code 开挂指南

让 AI 帮你写出生产级代码

从「AI 是听话的打字员」到「AI 是有方法论的专业工程师」

保姆级安装 · 七大核心功能实战 · 六大场景全流程 · 20 个 FAQ · 速查表

河马的CC商业实战 出品

2026年8月 · 会员专属版

会员专属资料 · 第 4 期

Claude Code 开挂指南

Superpowers 让 AI 帮你写出生产级代码

出品 河马的CC商业实战（知识星球）

版本 2026年8月 · 会员专属版

读者 正在用或想用 Claude Code 写代码的星友——从安装到实战全流程

> 本资料为「河马的CC商业实战」星球会员专属，每周/不定期更新。欢迎转发给需要的朋友——如果他们想持续收到这个系列，封底有入口。AI 工具迭代快，界面和命令以你实际使用的版本为准，思路不变。

河马卷首语

星友们，「会员专属资料」第 4 期来了。

这一期是写给用 CC 写代码的星友的「进阶通关攻略」。很多人把 Claude Code 当聪明打字员用——说一句写一句，错了再来。装了 Superpowers 之后你会发现，同一个 CC 突然有了方法论：先问需求、再出方案、确认了才动手、做完自己检查。这本手册从保姆级安装讲起，七个核心功能一个个过，再用六个真实场景把全流程演给你看。

用法建议：第二章安装跟着做就行；第三章核心概念值得认真读，这是「开窍」的部分；第四、五章对着自己的项目练手；遇到问题先翻第七章 FAQ，速查表打印放手边。

练出第一个「AI 自主开发数小时不出错」的项目，记得来星球里晒。

—— 河马，2026年8月

目 录

FOREWORD

前言

CHAPTER ONE

第一章 为什么你需要这份手册

1.1 Claude Code裸用 vs 装了Superpowers的效率对比

1.2 Superpowers到底是什么

1.3 适合谁用、不适合谁用

本章小结

动手练习

CHAPTER TWO

第二章 安装与配置（保姆级）

2.1 环境准备

2.2 三种安装方式详解

2.3 安装后的目录结构解读

2.4 验证安装成功的方法

2.5 常见安装报错与解决方案

本章小结

动手练习

CHAPTER THREE

第三章 核心概念理解

3.1 Agentic Skills Framework是什么

3.2 Superpowers的工作流哲学

3.3 与普通prompt engineering的本质区别

3.4 核心配置文件详解

本章小结

动手练习

CHAPTER FOUR

第四章 七大核心功能深度实战

4.1 结构化规划（Structured Planning / Brainstorming）

4.2 用户故事驱动开发（User Story Driven Development）

4.3 测试驱动开发TDD工作流（Red-Green-Refactor）

4.4 自动代码审查（Self Code Review）

4.5 上下文管理与Checkpoint机制

4.6 文档自动生成

4.7 安全检查与最佳实践扫描

本章小结

动手练习

CHAPTER FIVE

第五章 六大实战场景全流程演示

5.1 场景一：从零搭建一个React + TypeScript项目

5.2 场景二：给现有Python项目添加新功能模块

5.3 场景三：重构一个混乱的遗留代码库

5.4 场景四：修复一个复杂的跨模块Bug

5.5 场景五：为开源项目贡献PR

5.6 场景六：快速开发一个API后端服务

本章小结

动手练习

CHAPTER SIX

第六章 进阶技巧与组合拳

6.1 Superpowers + Context7 MCP联合使用

6.2 Superpowers + Firecrawl联合使用

6.3 自定义Skills：如何写自己的工作流规则

6.4 CLAUDE.md最佳实践

6.5 长会话管理策略

本章小结

动手练习

CHAPTER SEVEN

第七章 常见问题FAQ（20个Q&A）

安装问题

使用问题

效果问题

兼容性问题

付费相关问题

本章小结

CHAPTER EIGHT

第八章 速查表与备忘录

8.1 所有指令速查表

8.2 推荐工作流速查

8.3 配置参数速查表

8.4 常见报错速查表

本章小结

APPENDIX

附录

A. 推荐搭配的其他Claude Code插件清单

B. 学习资源与社区链接

C. 版本更新日志跟踪方法

ROADMAP

学习路线图

新手阶段（1-2周）

进阶阶段（2-4周）

高级阶段（1-2月）

专家阶段（持续）

EPILOGUE

结语

FOREWORD

前言

如果你正在阅读这份手册，说明你已经意识到了一件事：**裸用Claude Code和装了Superpowers的Claude Code，完全是两个物种。**

这不是夸张。在我使用Superpowers之前，我和Claude的对话是这样的：

> 我："帮我写个登录功能" Claude："好的，这是代码..."（直接开始写） 我：（三小时后）"等等，这个不对，我们重新来..."

而装了Superpowers之后，对话变成了：

> 我："帮我写个登录功能" Claude："在开始之前，让我先理解你的需求。这个登录功能需要支持哪些认证方式？密码策略有什么要求？是否需要多因素认证？" 我：（回答几个问题） Claude："好的，这是设计方案，请确认..." 我："确认" Claude："正在按计划执行，已完成任务1/8..."

这就是Superpowers的魔力——它把AI从一个"听话的打字员"变成了一个"有方法论的专业工程师"。

这份手册将带你从零开始，彻底掌握Superpowers的每一个功能，让你也能体验"AI自主开发数小时不出错"的快感。

CHAPTER ONE

第一章　为什么你需要这份手册

1.1 Claude Code裸用 vs 装了Superpowers的效率对比

1.2 Superpowers到底是什么

1.3 适合谁用、不适合谁用

本章小结

动手练习

1.1 Claude Code裸用 vs 装了Superpowers的效率对比

本章核心价值：理解Superpowers如何解决AI编程的核心痛点，以及它为什么值得你花时间学习。

让我们用一个真实场景来说明问题。

场景：开发一个待办事项应用的API

裸用Claude Code（无Superpowers）

```
你：帮我写一个待办事项API
Claude：好的，我用Express和MongoDB来写...
[Claude开始写代码]

20分钟后...
Claude：完成了！这是代码...
你：等等，你没有写测试
Claude：哦对，我补上...

15分钟后...
Claude：测试写好了
你：这个测试根本没覆盖边界情况，而且我刚才发现
    你的API设计和我想要的不一样...
Claude：那你想怎么改？
你：应该这样...（开始长篇解释）

30分钟后...
Claude：改好了
你：等等，你改的时候把之前的逻辑破坏了...

[循环往复]
```

总耗时：2-3小时
结果：代码质量参差不齐，测试覆盖不全，需要大量人工review

使用Superpowers

```
你：帮我写一个待办事项API
Claude：【触发brainstorming skill】

在开始编码之前，让我先理解你的需求：
1. 这个API需要支持哪些功能？（CRUD？搜索？排序？）
2. 使用什么技术栈？
```

```
3. 认证方式是什么？
4. 数据存储用什么？

你：（回答4个问题）

Claude：【自动生成设计文档】
这是设计方案，包含：
- API端点设计
- 数据模型
- 错误处理策略
- 测试策略

请确认这个设计...

你：确认

Claude：【触发writing-plans skill】
这是实施计划，共8个任务：
1. 初始化项目结构
2. 配置数据库连接
3. 实现Todo模型
4. 实现GET /todos
5. 实现POST /todos
6. 实现PUT /todos/:id
7. 实现DELETE /todos/:id
8. 添加错误处理中间件

每个任务预计2-5分钟，是否开始执行？

你：开始

Claude：【触发subagent-driven-development skill】
正在执行任务1/8...
[自动执行中]

[30分钟后]
Claude：所有任务完成！这是总结：
- 写了127行生产代码
- 写了89行测试代码（覆盖率94%）
- 所有测试通过
- 代码已通过self-review

请查看结果...
```

总耗时：30-40分钟（其中大部分是AI自动运行，你只需确认关键节点）

结果：代码质量一致，测试覆盖完善，设计经过充分讨论

1.2 Superpowers到底是什么

一句话定义：Superpowers是一个基于Agentic Skills Framework的软件开发生工作流插件，它通过一系列可组合的"技能"（Skills）和初始指令，强制AI遵循经过验证的开发方法论。

技术原理简述：



Superpowers的核心思想是：**不要让AI直接写代码，而是让AI先理解问题、制定计划、然后按步骤执行。**

1.3 适合谁用、不适合谁用

适合的人群

人群	为什么适合
已经使用Claude Code的开发者	你已经熟悉基础操作，Superpowers能让你的效率翻倍
AI编程爱好者	想体验最先进的AI辅助开发工作流
需要写生产级代码的人	Superpowers强制TDD和代码审查，保证代码质量
团队协作开发者	标准化的工作流让团队产出更一致
学习软件工程方法论的人	Superpowers本身就是最佳实践的教科书

不适合的人群

人群	为什么不适合
完全不懂编程的新手	Superpowers假设你懂基础开发概念
只想快速原型验证的人	Superpowers的严谨流程可能会拖慢原型速度
不使用Claude Code/Cursor/Codex等工具的人	Superpowers是这些工具的插件
抗拒自动化工作流的人	如果你坚持100%手动控制，Superpowers会让你不适

本章小结

Superpowers不是让AI写代码更快，而是让AI写代码**更对**。它通过强制性的工作流（设计→计划→TDD→审查）解决了AI编程的三大痛点：

1. **理解偏差**：通过brainstorming确保AI真正理解需求
2. **质量不稳定**：通过TDD和self-review保证代码质量
3. **上下文丢失**：通过结构化的计划和检查点管理长会话

动手练习

1. 回想你最近一次使用AI编程的经历，遇到了什么问题？
2. 对比裸用和用Superpowers的场景，你觉得哪些痛点可以被解决？

CHAPTER TWO

第二章　安装与配置（保姆级）

2.1 环境准备

2.2 三种安装方式详解

2.3 安装后的目录结构解读

2.4 验证安装成功的方法

2.5 常见安装报错与解决方案

本章小结

动手练习

2.1 环境准备

本章核心价值：手把手教你把Superpowers装到各种支持的平台，解决所有可能的报错。

Node.js版本要求

Superpowers本身不直接依赖Node.js，但Claude Code需要：

- **Node.js**：18.0.0 或更高版本
- **npm**：8.0.0 或更高版本

验证命令：

```
node --version  # 应显示 v18.x.x 或更高
npm --version   # 应显示 8.x.x 或更高
```

⚠️ **注意**：如果你的Node.js版本低于18，请先升级。推荐使用nvm管理Node版本：

```
# 安装nvm
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.0/install.sh | bash

# 安装Node 18
nvm install 18
nvm use 18
```

Claude Code版本要求

- **Claude Code**：0.2.0 或更高版本

验证命令：

```
claude --version
```

💡 **Pro Tip**：建议保持Claude Code为最新版本，因为Superpowers会利用最新功能。

2.2 三种安装方式详解

Superpowers支持多种AI编程工具，安装方式略有不同。

方式一：Claude Code官方市场（推荐）

这是最简单的方式，适合大多数用户。

```
# 在Claude Code中执行
/plugin install superpowers@claude-plugins-official
```

【此处插图：Claude Code官方市场安装截图】

验证安装：

```
# 查看已安装插件
/plugin list

# 应该能看到 superpowers
```

方式二：通过Superpowers市场（备用）

如果官方市场不可用，可以使用Superpowers自己的市场：

```
# 第一步：添加市场
/plugin marketplace add obra/superpowers-marketplace

# 第二步：安装插件
/plugin install superpowers@superpowers-marketplace
```

方式三：Cursor安装

Cursor用户可以通过插件市场安装：

```
# 在Cursor Agent聊天中输入
/add-plugin superpowers
```

或者在Cursor的插件市场中搜索"superpowers"并安装。

方式四：Codex手动安装

Codex用户需要手动配置：

```
# 告诉Codex
Fetch and follow instructions from https://raw.githubusercontent.com/obra/superpowers/refs/heads/main/.codex/INSTALL.md
```

详细文档：docs/README.codex.md

方式五：OpenCode手动安装

```
# 告诉OpenCode
Fetch and follow instructions from https://raw.githubusercontent.com/obra/superpowers/refs/heads/main/.opencode/INSTALL.md
```

详细文档：docs/README.opencode.md

方式六：GitHub Copilot CLI

```
copilot plugin marketplace add obra/superpowers-marketplace
copilot plugin install superpowers@superpowers-marketplace
```

方式七：Gemini CLI

```
gemini extensions install https://github.com/obra/superpowers
```

更新命令：

```
gemini extensions update superpowers
```

2.3 安装后的目录结构解读

安装完成后，Superpowers会在你的工作目录创建以下结构：

```
.claude-plugin/superpowers/          # Claude Code插件目录
├── skills/                          # 技能库（核心）
│   ├── brainstorming/               # 设计讨论技能
│   │   └── SKILL.md                 # 技能定义文件
│   ├── writing-plans/               # 计划制定技能
│   │   └── SKILL.md
│   ├── test-driven-development/     # TDD技能
│   │   └── SKILL.md
│   ├── subagent-driven-development/ # 子代理开发技能
│   │   └── SKILL.md
│   ├── requesting-code-review/      # 代码审查技能
│   │   └── SKILL.md
│   └── ...                          # 其他技能
├── hooks/                           # 钩子脚本
│   ├── session-start/               # 会话开始钩子
│   └── task-start/                  # 任务开始钩子
├── commands/                        # 命令定义（已弃用）
└── docs/                            # 文档

.claude/                             # Claude Code配置
└── CLAUDE.md                        # 项目特定的Claude指令
```

每个文件的作用：

文件/目录	作用
`skills/*/SKILL.md`	技能定义文件，包含触发条件和执行逻辑
`hooks/`	在特定时机自动执行的脚本
`CLAUDE.md`	项目级别的AI指令，Superpowers会读取

2.4 验证安装成功的方法

方法一：查看插件列表

```
/plugin list
```

应该能看到类似输出：

```
Installed plugins:
- superpowers@1.2.3 (from claude-plugins-official)
```

方法二：触发技能测试

开启一个新会话，输入：

```
帮我设计一个新功能
```

如果Superpowers正常工作，Claude会：

1. **不直接写代码**
2. **先问你一系列问题**（触发brainstorming skill）
3. **生成设计文档**让你确认

【此处插图：brainstorming触发后的对话截图】

方法三：检查日志

```
# 查看Claude Code日志
cat ~/.claude/logs/claude.log | grep superpowers
```

2.5 常见安装报错与解决方案

报错1："Plugin not found"

症状：

```
Error: Plugin "superpowers" not found in marketplace
```

原因： - 市场地址错误 - 网络问题 - 市场未正确添加

解决方案：

```
# 1. 确认市场已添加
/plugin marketplace list

# 2. 如果没有obra/superpowers-marketplace，重新添加
/plugin marketplace add obra/superpowers-marketplace

# 3. 重试安装
/plugin install superpowers@superpowers-marketplace
```

报错2："Version mismatch"

症状：

```
Error: Plugin requires Claude Code >= 0.2.0, current: 0.1.x
```

解决方案：

```
# 升级Claude Code
npm update -g @anthropic-ai/claude-code

# 验证版本
claude --version
```

报错3："Permission denied"

症状：

```
Error: EACCES: permission denied, mkdir '/usr/local/lib/...'
```

解决方案:

```
# 方案1：使用nvm管理的Node（推荐）
nvm use 18

# 方案2：修改npm全局目录权限
sudo chown -R $(whoami) $(npm config get prefix)/lib
sudo chown -R $(whoami) $(npm config get prefix)/bin

# 方案3：使用sudo（不推荐，但有效）
sudo npm install -g @anthropic-ai/claude-code
```

报错4："Network timeout"

症状:

```
Error: Network timeout while fetching plugin
```

解决方案:

```
# 1. 检查网络连接
ping github.com

# 2. 配置代理（如果需要）
npm config set proxy http://your-proxy:port
npm config set https-proxy http://your-proxy:port

# 3. 重试安装
/plugin install superpowers@claude-plugins-official
```

报错5："Skill not triggering"

症状：安装成功，但AI行为没有变化，直接开始写代码。

可能原因： 1. 项目目录下已有`.claude/CLAUDE.md`冲突 2. 会话缓存问题 3. 技能文件损坏

解决方案:

```
# 1. 检查CLAUDE.md冲突
cat .claude/CLAUDE.md | head -20

# 2. 重启Claude Code会话（完全退出再进入）

# 3. 重新安装插件
/plugin uninstall superpowers
/plugin install superpowers@claude-plugins-official
```

```
# 4. 如果还不行，手动清理缓存
rm -rf ~/.claude/cache/*
```

报错6："Hooks not executing"

症状：技能触发正常，但hooks（如自动创建worktree）不执行。

解决方案：

```
# 1. 检查hooks目录是否存在
ls -la .claude-plugin/superpowers/hooks/

# 2. 检查文件权限
chmod +x .claude-plugin/superpowers/hooks/*/*.sh

# 3. 查看hook日志
cat .claude-plugin/superpowers/logs/hook.log
```

踩坑预警：Windows用户可能会遇到hook脚本无法执行的问题。解决方案是使用WSL2或者在Git Bash中运行Claude Code。

本章小结

安装Superpowers的关键点：

1. **Node.js版本要够**（>= 18）
2. **选择正确的安装方式**（Claude Code推荐用官方市场）
3. **安装后一定要验证**（触发brainstorming测试）
4. **遇到问题先看日志**（大多数问题都有明确错误信息）

动手练习

1. 检查你的Node.js版本，如果不符合要求请升级
2. 按照本章步骤安装Superpowers
3. 开启新会话，输入"帮我设计一个新功能"，验证brainstorming skill是否正常触发

CHAPTER THREE

第三章 核心概念理解

3.1 Agentic Skills Framework是什么

3.2 Superpowers的工作流哲学

3.3 与普通prompt engineering的本质区别

3.4 核心配置文件详解

本章小结

动手练习

3.1 Agentic Skills Framework是什么

本章核心价值：理解Superpowers的设计哲学，知道"为什么"而不仅是"怎么做"。

Agentic Skills Framework是Superpowers的核心架构。要理解它，我们先看看传统AI编程的问题。

传统AI编程的问题

```
用户：帮我写个登录功能
AI：好的（直接开始写）

问题：
1. AI不知道你的具体需求
2. AI没有计划，想到哪写到哪
3. AI不会自己测试
4. AI不会检查代码质量
```

Agentic Skills Framework的解决方案

```
用户：帮我写个登录功能
AI：【检查技能库】
   → 匹配到brainstorming skill
   → 执行：先问清楚需求
   → 输出：设计文档

用户：确认设计
AI：【检查技能库】
   → 匹配到writing-plans skill
   → 执行：制定详细计划
   → 输出：任务列表

用户：开始执行
AI：【检查技能库】
   → 匹配到test-driven-development skill
   → 执行：写测试→写代码→重构
   → 输出：通过测试的代码
```

核心思想：把AI的行为封装成可复用的"技能"，每个技能有明确的触发条件和执行流程。

Skills的组成结构

每个Skill包含：

```
# Skill名称

## 触发条件
什么时候激活这个skill

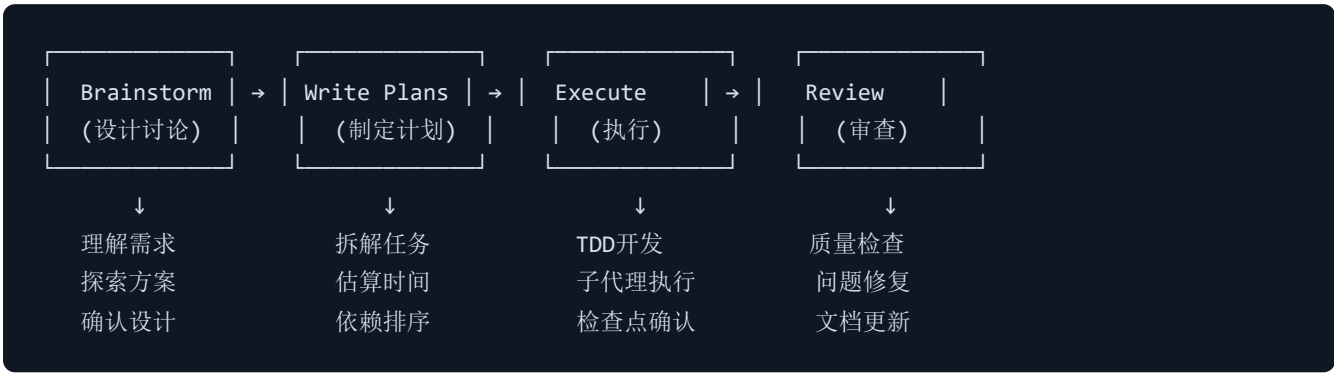
## 执行流程
1. 步骤一
2. 步骤二
3. 步骤三

## 输出规范
必须产出什么

## 检查清单
完成标准
```

3.2 Superpowers的工作流哲学

Superpowers强制遵循以下工作流：



为什么要强制这个流程?

1. 规划→测试→实现的顺序

传统方式的问题：

```
直接写代码 → 发现理解错了 → 重写 → 又错了 → 再重写
```

Superpowers方式：

```
先规划 → 确认理解正确 → 按计划执行 → 一次性做对
```

2. 为什么强调TDD（测试驱动开发）

方式	代码质量	重构信心	文档作用
先写代码后补测试	低	低	测试只是形式
TDD（先写测试）	高	高	测试即文档

TDD的核心循环（Red-Green-Refactor）：

```
1. Red：写一个失败的测试
2. Green：写最少代码让测试通过
3. Refactor：重构代码，保持测试通过
```

3. 为什么用子代理（Subagent）执行

Superpowers的一个独特设计是：用子代理执行具体任务，而不是在主会话中直接写代码。

好处：

- **上下文隔离**：子代理只关注当前任务，不会被历史对话干扰
- **并行执行**：可以同时派多个子代理处理不同任务
- **容错性**：子代理失败不会影响主会话
- **可review**：子代理的产出可以被独立审查

3.3 与普通prompt engineering的本质区别

方面	Prompt Engineering	Superpowers
方式	每次手动写长prompt	预定义的skill自动触发
一致性	依赖用户prompt质量	标准化的skill保证一致性
可复用	每个项目重新写	skill库跨项目复用
协作	难以团队协作	标准化流程便于协作
维护	prompt散落在各处	skill集中管理，可更新

本质区别：

- **Prompt Engineering**是"每次告诉AI怎么做"
- **Superpowers**是"让AI自己知道该怎么做"

3.4 核心配置文件详解

CLAUDE.md

这是项目级别的AI指令文件，Superpowers会读取它。

位置：`.claude/CLAUDE.md`

作用： - 定义项目特定的规则 - 指定技术栈 - 设置代码风格 - 配置Superpowers行为

示例：

```
# 项目规范

## 技术栈
- 前端：React + TypeScript
- 后端：Node.js + Express
- 数据库：PostgreSQL

## 代码风格
- 使用ESLint + Prettier
- 函数式组件优先
- 所有API必须返回统一格式：{ success: boolean, data: any, error?: string }

## Superpowers配置
- 强制TDD：true
- 子代理模型：claude-3-5-sonnet-20241022
- 检查点间隔：每3个任务
```

💡 **Pro Tip**：CLAUDE.md是Superpowers和你的项目之间的桥梁。好的CLAUDE.md能让Superpowers更好地适应你的项目需求。

Skill配置文件

每个skill都有一个`SKILL.md`文件，定义了skill的行为。

示例（test-driven-development skill的部分内容）：

```
# Test-Driven Development

## 触发条件
- 用户要求实现功能
- 计划中的任务需要编码

## 执行流程
1. 先写测试（Red阶段）
```

```
2. 运行测试，确认失败
3. 写最少代码让测试通过（Green阶段）
4. 重构代码（Refactor阶段）
5. 提交代码

## 禁止事项
- 不写测试直接写代码
- 写比测试要求更多的代码（YAGNI原则）

## 输出
- 测试文件
- 实现代码
- 测试运行结果
```

踩坑预警：不要手动修改skill文件，除非你完全理解其结构。错误的修改会导致skill无法触发或行为异常。

本章小结

Superpowers的核心设计哲学：

1. **Agentic Skills Framework**：把AI行为封装成可复用的skill
2. **强制工作流**：规划→计划→TDD执行→审查，不跳过任何步骤
3. **子代理执行**：用子代理处理具体任务，保证上下文清晰
4. **配置驱动**：通过CLAUDE.md让Superpowers适应你的项目

动手练习

1. 查看你项目中的`.claude/CLAUDE.md`文件（如果没有，创建一个）
2. 浏览`.claude-plugin/superpowers/skills/`目录，看看有哪些skill
3. 选择一个skill，阅读其SKILL.md，理解它的触发条件和执行流程

CHAPTER FOUR

第四章 七大核心功能深度实战

4.1 结构化规划（Structured Planning / Brainstorming）

4.2 用户故事驱动开发（User Story Driven Development）

4.3 测试驱动开发TDD工作流（Red-Green-Refactor）

4.4 自动代码审查（Self Code Review）

4.5 上下文管理与Checkpoint机制

4.6 文档自动生成

4.7 安全检查与最佳实践扫描

本章小结

动手练习

4.1 结构化规划（Structured Planning / Brainstorming）

本章核心价值：掌握Superpowers的每个核心功能，知道什么时候用、怎么用、用什么参数。

功能说明

Brainstorming skill是Superpowers的第一个技能，在AI开始写代码之前自动触发。它通过苏格拉底式提问，帮助AI和用户就需求达成一致。

核心目标： - 澄清需求细节 - 探索可行方案 - 确认设计方向 - 生成设计文档

触发方式

自动触发条件： - 用户提到"设计"、"规划"、"实现"等功能性需求 - 用户描述一个新项目或新功能 - 用户说"帮我做XXX"（XXX是功能描述）

手动触发：

```
让我们brainstorm一下这个需求
```

完整操作流程

Step 1：AI识别到需要brainstorming

```
用户：帮我做一个用户认证系统

AI：【触发brainstorming skill】

在开始实现之前，让我先理解你的需求。
这是一个用户认证系统，为了确保设计符合你的期望，
我需要先问你几个问题...
```

Step 2：AI进行苏格拉底式提问

```
AI：
1. 这个认证系统需要支持哪些登录方式？
   - 用户名+密码？
   - 邮箱+密码？
   - 手机号+验证码？
   - 第三方登录（Google/GitHub等）？
```

```
2. 密码策略有什么要求？
   - 最小长度？
   - 复杂度要求（大小写+数字+特殊字符）？
   - 是否需要定期更换？

3. 会话管理策略？
   - JWT还是Session？
   - Token有效期多久？
   - 是否支持多设备登录？

4. 安全要求？
   - 是否需要双因素认证？
   - 登录失败次数限制？
   - 是否需要登录日志？
```

Step 3：用户回答问题

```
用户：
1. 用户名+密码和邮箱+密码
2. 最小8位，需要大小写+数字
3. JWT，24小时有效期，支持多设备
4. 需要登录失败限制（5次锁定15分钟），需要登录日志
```

Step 4：AI生成设计文档

```
AI：
根据你的回答，我生成了以下设计文档：

# 用户认证系统设计文档

## 功能需求
- 支持用户名+密码登录
- 支持邮箱+密码登录
- JWT认证，24小时有效期
- 支持多设备同时登录

## 安全需求
- 密码最小8位，必须包含大小写字母和数字
- 登录失败5次锁定15分钟
- 记录登录日志（时间、IP、结果）

## API设计
### POST /auth/register
- 请求：{ username, email, password }
- 响应：{ success, data: { userId }, error }

### POST /auth/login
- 请求：{ username/email, password }
```

```
- 响应：{ success, data: { token, user }, error }

### POST /auth/logout
- 请求：{ token }
- 响应：{ success }

## 数据模型
### User
- id: UUID
- username: string (unique)
- email: string (unique)
- passwordHash: string
- createdAt: timestamp
- updatedAt: timestamp

### LoginLog
- id: UUID
- userId: UUID
- ip: string
- userAgent: string
- success: boolean
- createdAt: timestamp

请确认这个设计，如果有修改意见请告诉我...
```

Step 5：用户确认或修改

```
用户：确认，但还需要加一个刷新token的接口

AI：收到，已更新设计文档...

### POST /auth/refresh
- 请求：{ refreshToken }
- 响应：{ success, data: { token }, error }

设计文档已保存到 .docs/design/auth-system.md
```

输入输出示例

输入：用户的功能需求描述
输出：结构化的设计文档（markdown格式）

最佳实践

1. **回答要具体**：模糊的答案会导致设计不符合预期 - ❌ "密码要安全一点" - ✅ "密码最小8位，包含大小写字母和数字"
2. **主动补充需求**：AI可能没问到但你关心的点，主动提出来

3. **要求AI解释**：如果不理解AI的设计，让它解释为什么这样设计
4. **保存设计文档**：确认后的设计文档会被保存，后续开发基于它

踩坑记录

- **坑1：AI没有触发brainstorming**

症状：AI直接开始写代码

原因： - 用户prompt太具体，AI以为不需要讨论 - 当前会话已经brainstorm过 - skill未正确安装

解决：

```
用户：先不要写代码，我们先brainstorm一下
```

- **坑2：设计文档太笼统**

症状：设计文档缺少关键细节

解决：

```
用户：这个设计太笼统了，我需要更详细的API参数定义
```

- **坑3：设计文档和后续开发脱节**

症状：AI开发时偏离了设计文档

解决：在CLAUDE.md中添加：

```
## 开发规范
- 必须严格按照已确认的设计文档开发
- 任何偏离都需要用户明确同意
```

4.2 用户故事驱动开发（User Story Driven Development）

功能说明

用户故事驱动开发是brainstorming的延伸，它将需求转化为具体的用户故事，确保开发始终以用户价值为中心。

核心目标： - 从用户视角描述需求 - 定义验收标准 - 确保功能有用户价值

触发方式

通常在brainstorming之后自动触发，或手动触发：

```
让我们用用户故事来描述这个需求
```

完整操作流程

Step 1：将需求转化为用户故事

```
AI：
基于刚才确认的设计，我来定义用户故事：

## 用户故事

### Story 1: 用户注册
**作为** 新用户
**我想要** 用用户名和邮箱注册账号
**以便** 可以使用系统功能

**验收标准**：
- [ ] 可以输入用户名、邮箱、密码
- [ ] 用户名和邮箱必须唯一
- [ ] 密码符合复杂度要求
- [ ] 注册成功后返回用户ID

### Story 2: 用户登录
**作为** 注册用户
**我想要** 用用户名或邮箱登录
**以便** 访问我的账号

**验收标准**：
- [ ] 可以用用户名登录
- [ ] 可以用邮箱登录
- [ ] 密码错误时返回明确错误
```

```
- [ ] 登录成功后返回JWT token
- [ ] 登录失败5次后锁定15分钟

### Story 3: 查看登录日志
**作为** 管理员
**我想要** 查看用户的登录日志
**以便** 监控账号安全

**验收标准**:
- [ ] 可以查看指定用户的登录历史
- [ ] 显示登录时间、IP、结果
- [ ] 支持按时间范围筛选
```

Step 2：用户确认用户故事

```
用户：确认，但Story 3应该是普通用户查看自己的日志，不是管理员

AI：收到，已修改Story 3...
```

输入输出示例

输入：已确认的设计文档
输出：用户故事列表（每个故事包含As a/I want/So that和验收标准）

最佳实践

1. **验收标准要可测试**：每个标准都应该能写成自动化测试 - ❌ "登录要快" - ✅ "登录响应时间 < 200ms"
2. **故事粒度要适中**： - 太大：难以估算和实现 - 太小：管理成本高 - 推荐：每个故事2-8小时工作量
3. **关注用户价值**：每个故事都要回答"这对用户有什么好处"

踩坑记录

🕳️ 坑：用户故事变成技术任务

症状： - ❌ "作为开发者，我想要创建数据库表..." - ✅ "作为用户，我想要保存我的数据..."

解决：始终以"作为[用户角色]"开头，而不是"作为开发者"

4.3 测试驱动开发TDD工作流（Red-Green-Refactor）

功能说明

TDD skill是Superpowers最核心的技能之一。它强制AI遵循Red-Green-Refactor循环，确保代码质量。

核心目标： - 先写测试，后写代码 - 确保测试覆盖 - 通过重构保持代码质量

触发方式

自动触发： - 执行计划中的编码任务时 - 用户说"用TDD方式实现"

手动触发：

```
用TDD方式实现这个功能
```

完整操作流程

Step 1：Red阶段（写测试，看测试失败）

```
AI: 【触发TDD skill】

现在用TDD方式实现用户注册功能。

**Red阶段**：先写测试

```typescript
// tests/auth/register.test.ts
describe('POST /auth/register', () => {
  it('should register a new user with valid data', async () => {
    const response = await request(app)
      .post('/auth/register')
      .send({
        username: 'testuser',
        email: 'test@example.com',
        password: 'Test1234'
      });

    expect(response.status).toBe(201);
    expect(response.body.success).toBe(true);
    expect(response.body.data.userId).toBeDefined();
  });

  it('should reject duplicate username', async () => {
    // 先创建一个用户
    await createUser({ username: 'testuser', email: 'first@example.com', password: 'Test1234' });

    const response = await request(app)
      .post('/auth/register')
      .send({
        username: 'testuser',
        email: 'second@example.com',
```
```

```
        password: 'Test1234'
      });

    expect(response.status).toBe(409);
    expect(response.body.success).toBe(false);
  });

  it('should reject weak password', async () => {
    const response = await request(app)
      .post('/auth/register')
      .send({
        username: 'testuser',
        email: 'test@example.com',
        password: '123'  // 太弱
      });

    expect(response.status).toBe(400);
    expect(response.body.success).toBe(false);
  });
});
```

现在运行测试，确认它们失败...

```
npm test -- tests/auth/register.test.ts
```

结果： - ✗ should register a new user with valid data (404 - 路由不存在) - ✗ should reject duplicate username (404 - 路由不存在) - ✗ should reject weak password (404 - 路由不存在)

测试失败，符合预期。进入Green阶段。

```
**Step 2：Green阶段（写最少代码让测试通过）**
```

AI： **Green阶段**：写最少代码让测试通过

```
// src/routes/auth.ts
import { Router } from 'express';
import { v4 as uuidv4 } from 'uuid';
import bcrypt from 'bcrypt';

const router = Router();
const users: Map<string, any> = new Map();  // 临时存储

router.post('/register', async (req, res) => {
  const { username, email, password } = req.body;

  // 检查密码强度
  if (password.length < 8 || !/[A-Z]/.test(password) || !/[a-z]/.test(password) || !/[0-9]/.test(password)) {
    return res.status(400).json({ success: false, error: 'Password too weak' });
  }
```

```
  // 检查用户名是否已存在
  for (const user of users.values()) {
    if (user.username === username) {
      return res.status(409).json({ success: false, error: 'Username already exists' });
    }
  }

  // 创建用户
  const userId = uuidv4();
  const passwordHash = await bcrypt.hash(password, 10);

  users.set(userId, {
    id: userId,
    username,
    email,
    passwordHash,
    createdAt: new Date(),
    updatedAt: new Date()
  });

  res.status(201).json({ success: true, data: { userId } });
});

export default router;
```

运行测试...

```
npm test -- tests/auth/register.test.ts
```

结果： - ✓ should register a new user with valid data - ✓ should reject duplicate username - ✓ should reject weak password

所有测试通过！进入Refactor阶段。

```

**Step 3：Refactor阶段（重构代码）**

```

AI： **Refactor阶段**：改进代码质量，保持测试通过

观察到的问题： 1. 用户存储用Map，应该用数据库 2. 密码验证逻辑太复杂，应该提取 3. 缺少错误处理

重构：

```
// src/services/userService.ts
import bcrypt from 'bcrypt';

export class UserService {
  private users: Map<string, any> = new Map();

  async createUser(username: string, email: string, password: string) {
    this.validatePassword(password);
```

```
    await this.ensureUsernameNotExists(username);

    const userId = crypto.randomUUID();
    const passwordHash = await bcrypt.hash(password, 10);

    const user = {
      id: userId,
      username,
      email,
      passwordHash,
      createdAt: new Date(),
      updatedAt: new Date()
    };

    this.users.set(userId, user);
    return { userId };
  }

  private validatePassword(password: string): void {
    if (password.length < 8) {
      throw new ValidationError('Password must be at least 8 characters');
    }
    if (!/[A-Z]/.test(password) || !/[a-z]/.test(password) || !/[0-9]/.test(password)) {
      throw new ValidationError('Password must contain uppercase, lowercase, and number');
    }
  }

  private async ensureUsernameNotExists(username: string): Promise<void> {
    for (const user of this.users.values()) {
      if (user.username === username) {
        throw new ConflictError('Username already exists');
      }
    }
  }
}

// src/routes/auth.ts
import { Router } from 'express';
import { UserService } from '../services/userService';

const router = Router();
const userService = new UserService();

router.post('/register', async (req, res, next) => {
  try {
    const { username, email, password } = req.body;
    const result = await userService.createUser(username, email, password);
    res.status(201).json({ success: true, data: result });
  } catch (error) {
    next(error);
  }
});

export default router;
```

运行测试确认重构没有破坏功能...

```
npm test -- tests/auth/register.test.ts
```

结果： - ✓ should register a new user with valid data - ✓ should reject duplicate username - ✓ should reject weak password

测试仍然通过，重构完成！

```
**Step 4：提交代码**
```

AI： 提交代码：

```
git add .
git commit -m "feat(auth): implement user registration with TDD

- Add POST /auth/register endpoint
- Implement password validation
- Handle duplicate username
- Add comprehensive tests

Tests: 3/3 passing"
```

```

#### 输入输出示例

**输入**：需要实现的功能描述
**输出**：测试文件 + 实现代码 + 提交记录

#### 最佳实践

1. **Red阶段测试要具体**：测试应该明确验证某个具体行为
2. **Green阶段代码要最少**：只写让测试通过的必要代码
3. **Refactor阶段不要改变行为**：只改进代码结构
4. **频繁运行测试**：每个阶段都要运行测试确认状态

#### 踩坑记录

- **坑1：写太多代码在Green阶段**

**症状**：Green阶段写了大量"预期会用到的"代码

**解决**：记住YAGNI原则（You Aren't Gonna Need It），只写让当前测试通过的代码

- **坑2：Refactor阶段添加新功能**

**症状**：重构时"顺便"加了新功能

**解决**：重构只改变结构，不改变行为。新功能需要新测试
```

🕳️ 坑3：测试太笼统

症状：- ❌ `it('should work', ...)` - ✅ `it('should return 201 with userId when registration succeeds', ...)`

4.4 自动代码审查（Self Code Review）

功能说明

Self Code Review skill让AI在完成代码后自动审查自己的代码，发现问题并修复。

核心目标： - 发现代码质量问题 - 检查是否符合设计文档 - 确保测试覆盖 - 提前发现潜在bug

触发方式

自动触发： - 每个任务完成后 - 一批任务完成后

手动触发：

```
请review这段代码
```

完整操作流程

Step 1：AI执行self-review

```
AI：【触发requesting-code-review skill】

现在对刚完成的代码进行self-review...

## Code Review报告

### 文件：src/services/userService.ts

#### 问题1：严重 - 缺少邮箱唯一性检查
**位置**：createUser方法
**问题**：只检查了用户名唯一性，没检查邮箱
**影响**：可能导致重复邮箱注册
**建议**：添加邮箱唯一性检查

#### 问题2：中等 - 密码哈希work factor硬编码
**位置**：createUser方法第15行
**问题**：bcrypt.hash(password, 10)中的10是硬编码
**影响**：后续调整困难
**建议**：提取为配置项

#### 问题3：轻微 - 缺少JSDoc注释
**位置**：createUser方法
**问题**：公共方法缺少文档
**建议**：添加JSDoc注释

### 文件：src/routes/auth.ts
```

```
#### 问题4：严重 - 缺少输入验证
**位置**：/register路由
**问题**：没有验证请求体格式
**影响**：可能导致服务器错误
**建议**：添加express-validator

### 测试覆盖检查

- ✓ 正常注册流程
- ✓ 重复用户名
- ✓ 弱密码
- ✗ 缺少邮箱格式验证测试
- ✗ 缺少缺少必填字段测试
```

Step 2：AI修复问题

```
AI：
正在修复发现的问题...

修复问题1（邮箱唯一性检查）：
```typescript
private async ensureEmailNotExists(email: string): Promise<void> {
  for (const user of this.users.values()) {
    if (user.email === email) {
      throw new ConflictError('Email already exists');
    }
  }
}
```

修复问题2（密码哈希work factor配置）：

```
const BCRYPT_ROUNDS = parseInt(process.env.BCRYPT_ROUNDS || '10', 10);
const passwordHash = await bcrypt.hash(password, BCRYPT_ROUNDS);
```

修复问题4（输入验证）：

```
import { body, validationResult } from 'express-validator';

router.post('/register', [
  body('username').isLength({ min: 3, max: 20 }).isAlphanumeric(),
  body('email').isEmail(),
  body('password').isLength({ min: 8 })
], async (req, res, next) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({ success: false, error: errors.array() });
```
```

```
  }
  // ...
});
```

补充测试：

```
it('should reject duplicate email', async () => { ... });
it('should reject invalid email format', async () => { ... });
it('should reject missing required fields', async () => { ... });
```

运行所有测试确认修复...

```
npm test
```

结果：7/7测试通过

```
#### 输入输出示例

**输入**：刚完成的代码
**输出**：Review报告 + 修复后的代码

#### 最佳实践

1. **不要跳过review**：即使AI说"代码没问题"，也要让它review
2. **关注严重问题**：Critical和High优先级问题必须修复
3. **review要具体**：指出具体文件、行号、问题描述

#### 踩坑记录

🕳️ **坑：AI说"代码没问题"但实际有问题**

**症状**：AI review后说"代码质量良好"，但你发现了问题

**解决**：在CLAUDE.md中添加：
```markdown
## Code Review要求
- 必须检查所有修改的文件
- 必须验证测试覆盖
- 必须检查边界情况
- 不能轻易说"没问题"
```

```

4.5 上下文管理与Checkpoint机制

功能说明

Superpowers通过Checkpoint机制管理长会话的上下文，确保AI不会"忘记"之前的决策。

核心目标： - 保存关键决策点 - 支持会话恢复 - 管理长会话的上下文窗口

触发方式

自动触发： - 完成brainstorming后 - 完成计划制定后 - 每完成一批任务后

完整操作流程

Step 1：创建Checkpoint

```
AI：
创建Checkpoint...
```

> ## Checkpoint: 设计阶段完成

时间：2024-01-15 10:30 **状态**：设计已确认

已完成： - ✅ Brainstorming完成 - ✅ 设计文档已确认 - ✅ 用户故事已定义

关键决策： - 使用JWT认证，24小时有效期 - 密码策略：8位以上，大小写+数字 - 数据库：PostgreSQL

待办： - ⏳ 制定实施计划 - ⏳ 开发 - ⏳ 测试

相关文件： - .docs/design/auth-system.md - .docs/user-stories.md

```
Checkpoint已保存到 .superpowers/checkpoints/20240115-103000.md
```

Step 2：恢复Checkpoint

如果会话中断，下次启动时可以恢复：

```
用户：恢复上次的会话

AI：
找到最近的Checkpoint：
- 时间：2024-01-15 10:30
```

```
- 状态：设计已确认

是否从这个Checkpoint继续？

用户：是

AI：
已恢复Checkpoint：设计阶段完成

**当前状态**：设计已确认，等待制定实施计划

是否开始制定计划？
```

输入输出示例

输入：完成某个阶段
输出：Checkpoint文件

最佳实践

1. **关键节点创建Checkpoint**：设计确认、计划完成、重要里程碑
2. **命名要清晰**：Checkpoint文件名应该反映状态
3. **定期清理**：删除过期的Checkpoint

踩坑记录

- **坑：Checkpoint太多导致混乱**

症状：目录下有几十个Checkpoint，不知道哪个是最新的

解决：建立Checkpoint命名规范：

```
YYYYMMDD-HHMMSS-简短描述.md
```

4.6 文档自动生成

功能说明

Superpowers可以自动生成项目文档，包括API文档、架构文档、开发指南等。

核心目标： - 保持文档与代码同步 - 减少文档维护成本 - 提供一致格式的文档

触发方式

自动触发： - 代码完成后 - 用户说"生成文档"

手动触发：

```
请生成API文档
```

完整操作流程

Step 1：AI分析代码生成文档

```
AI：
正在生成API文档...

```markdown
# API文档 - 认证模块

## 端点列表

### POST /auth/register
注册用户

**请求**：
```json
{
  "username": "string (3-20字符，字母数字)",
  "email": "string (邮箱格式)",
  "password": "string (最少8位，需包含大小写字母和数字)"
}
```

响应： - 201 Created

```
{
  "success": true,
  "data": {
    "userId": "uuid"
```

```
  }
}
```

- 400 Bad Request - 请求参数错误

```
{
  "success": false,
  "error": "Validation error message"
}
```

- 409 Conflict - 用户名或邮箱已存在

```
{
  "success": false,
  "error": "Username/Email already exists"
}
```

示例:

```
curl -X POST http://localhost:3000/auth/register \
  -H "Content-Type: application/json" \
  -d '{
    "username": "john_doe",
    "email": "john@example.com",
    "password": "SecurePass123"
  }'
```

POST /auth/login

用户登录

...

```
文档已保存到 docs/api/auth.md
```

输入输出示例

输入：代码文件
输出：markdown格式的文档

最佳实践

1. **文档与代码同位置**：文档放在代码旁边，方便同步更新

2. **包含示例**：文档应该包含使用示例
3. **注明版本**：API文档要注明版本号

4.7 安全检查与最佳实践扫描

功能说明

Superpowers会自动扫描代码中的安全问题和最佳实践违规。

核心目标： - 发现安全漏洞 - 确保遵循最佳实践 - 提前预防问题

触发方式

自动触发： - 代码完成后 - 提交前

手动触发：

```
请进行安全检查
```

完整操作流程

Step 1：AI执行安全检查

```
AI：
正在执行安全检查...

## 安全检查报告

### 严重问题

#### 1. 明文存储密码
**位置**：src/services/userService.ts:15
**问题**：发现代码可能存储明文密码
**风险**：用户密码泄露
**修复**：确保使用bcrypt等算法哈希密码

#### 2. 缺少SQL注入防护
**位置**：src/database/queries.ts
**问题**：字符串拼接SQL查询
**风险**：SQL注入攻击
**修复**：使用参数化查询

### 中等问题

#### 3. 没有Rate Limiting
**位置**：src/app.ts
**问题**：API没有速率限制
**风险**：暴力破解、DDoS
**修复**：添加express-rate-limit
```

```
### 最佳实践违规

#### 4. 错误信息泄露敏感信息
**位置**：src/middleware/errorHandler.ts
**问题**：错误响应包含堆栈跟踪
**风险**：泄露系统信息
**修复**：生产环境只返回通用错误信息
```

Step 2：AI修复问题

```
AI：
正在修复安全问题...

修复问题2（SQL注入）：
```typescript
// 之前（有风险）
const query = `SELECT * FROM users WHERE username = '${username}'`;

// 之后（安全）
const query = 'SELECT * FROM users WHERE username = $1';
const result = await db.query(query, [username]);
```

修复问题3（Rate Limiting）：

```typescript
import rateLimit from 'express-rate-limit';

const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15分钟
  max: 100 // 每个IP最多100次请求
});

app.use(limiter);
```

修复问题4（错误信息）：

```typescript
if (process.env.NODE_ENV === 'production') {
  res.status(500).json({ success: false, error: 'Internal server error' });
} else {
  res.status(500).json({ success: false, error: err.message, stack: err.stack });
}
```

```

**输入输出示例**

**输入**：代码文件
**输出**：安全检查报告 + 修复后的代码

### 最佳实践

1. **不要忽视警告**：即使是警告级别的问题也应该关注
2. **定期扫描**：安全检查应该定期进行
3. **更新规则**：安全最佳实践在演进，保持规则更新

---

# 本章小结

Superpowers的七大核心功能：

| 功能 | 作用 | 触发时机 |
| --- | --- | --- |
| Brainstorming | 澄清需求，生成设计 | 开发前 |
| 用户故事 | 从用户视角定义需求 | 设计后 |
| TDD | 测试驱动开发 | 编码时 |
| Self Code Review | 自动代码审查 | 编码后 |
| Checkpoint | 上下文管理 | 关键节点 |
| 文档生成 | 自动生成文档 | 完成后 |
| 安全检查 | 安全扫描 | 提交前 |

# 动手练习

1. 用一个简单功能（如计算器）体验完整的TDD流程
2. 故意写一个有问题的代码，测试Self Code Review能否发现
3. 创建一个Checkpoint，然后尝试恢复

CHAPTER FIVE

# 第五章 六大实战场景全流程演示


5.1 场景一：从零搭建一个React + TypeScript项目

5.2 场景二：给现有Python项目添加新功能模块

5.3 场景三：重构一个混乱的遗留代码库

5.4 场景四：修复一个复杂的跨模块Bug

5.5 场景五：为开源项目贡献PR

5.6 场景六：快速开发一个API后端服务

本章小结

动手练习

# 5.1 场景一：从零搭建一个React + TypeScript项目

**本章核心价值**：通过6个完整场景，展示Superpowers在实际项目中的应用。

---

### 需求描述

"我要做一个个人博客网站，前端用React + TypeScript，需要文章列表、文章详情、关于页面。设计要简洁现代。"

### 完整工作流

#### Step 1：Brainstorming（设计讨论）

```
用户：我要做一个个人博客网站，前端用React + TypeScript，
      需要文章列表、文章详情、关于页面。设计要简洁现代。

AI：【触发brainstorming skill】

在开始之前，让我先理解你的需求：

1. 技术栈相关：
   - 构建工具：Vite、Create React App、还是Next.js？
   - 样式方案：CSS Modules、Styled Components、Tailwind CSS？
   - 状态管理：React Context、Redux、Zustand？

2. 路由：
   - 使用React Router？
   - 需要哪些路由？（/、/posts/:id、/about）

3. 数据源：
   - 文章数据从哪里来？（静态JSON、CMS、自己写的Markdown？）
   - 是否需要服务端渲染？

4. 设计：
   - "简洁现代"有参考网站吗？
   - 需要响应式设计吗？
   - 需要暗黑模式吗？

5. 部署：
   - 计划部署到哪里？（Vercel、Netlify、GitHub Pages？）

请回答这些问题，我来生成详细的设计文档。
```

用户回答：

```
1. Vite + Tailwind CSS + React Context
2. React Router，路由：/、/posts/:id、/about
3. 静态Markdown文件，放在src/posts/目录
4. 参考Medium的设计风格，需要响应式，需要暗黑模式
5. 部署到Vercel
```

AI生成设计文档:

```
# 个人博客网站设计文档

## 技术栈
- 构建工具：Vite 5.x
- 框架：React 18 + TypeScript
- 样式：Tailwind CSS
- 路由：React Router v6
- 状态：React Context
- Markdown渲染：react-markdown + remark-gfm

## 项目结构
```

src/ ├── components/ # 可复用组件 | ├── Header.tsx | ├── Footer.tsx | ├── PostCard.tsx | └── ThemeToggle.tsx ├── pages/ # 页面组件 | ├── Home.tsx | ├── PostDetail.tsx | └── About.tsx ├── posts/ # Markdown文章 | ├── hello-world.md | └── another-post.md ├── contexts/ # React Context | └── ThemeContext.tsx ├── hooks/ # 自定义Hooks | └── usePosts.ts ├── types/ # TypeScript类型 | └── index.ts ├── utils/ # 工具函数 | └── markdown.ts ├── App.tsx └── main.tsx

```

## 路由设计
| 路径 | 页面 | 描述 |
|------|------|------|
| / | Home | 文章列表 |
| /posts/:slug | PostDetail | 文章详情 |
| /about | About | 关于页面 |

## 组件设计

### Header
- Logo/网站标题
- 导航链接（首页、关于）
- 主题切换按钮

### PostCard
- 文章标题
- 发布日期
- 摘要（前100字）
```

```
- 阅读更多链接

### PostDetail
- 文章标题
- 发布日期
- Markdown内容渲染
- 返回链接

## 主题设计
- 亮色：白底黑字，蓝色强调
- 暗色：深灰底白字，蓝色强调
- 使用Tailwind的dark模式

## 响应式断点
- 移动端：< 768px
- 平板：768px - 1024px
- 桌面：> 1024px

## 部署
- 平台：Vercel
- 自动部署：连接GitHub仓库
```

Step 2：Writing Plans（制定计划）

```
AI：【触发writing-plans skill】

基于设计文档，这是实施计划：

## 实施计划（共12个任务）

### Phase 1：项目初始化
**任务1**：初始化Vite项目
- 命令：`npm create vite@latest blog -- --template react-ts`
- 验证：项目能运行，显示Vite默认页面
- 预计：3分钟

**任务2**：安装依赖
- 安装：tailwindcss, postcss, autoprefixer, react-router-dom, react-markdown, remark-gfm
- 验证：package.json包含所有依赖
- 预计：3分钟

**任务3**：配置Tailwind CSS
- 创建tailwind.config.js
- 配置dark模式
- 验证：能使用Tailwind类
- 预计：2分钟

### Phase 2：基础组件
**任务4**：创建Header组件
- 实现Logo、导航、主题切换
```

- 验证：组件渲染正确
- 预计：5分钟

任务5：创建Footer组件
- 实现版权信息
- 验证：组件渲染正确
- 预计：2分钟

任务6：创建ThemeContext
- 实现主题状态管理
- 验证：主题能切换
- 预计：5分钟

Phase 3: 页面
任务7：创建Home页面
- 实现文章列表
- 使用PostCard组件
- 验证：显示文章列表
- 预计：8分钟

任务8：创建PostCard组件
- 实现文章卡片UI
- 验证：卡片渲染正确
- 预计：5分钟

任务9：创建PostDetail页面
- 实现Markdown渲染
- 验证：能显示文章内容
- 预计：8分钟

任务10：创建About页面
- 实现关于页面内容
- 验证：页面渲染正确
- 预计：3分钟

Phase 4: 路由与集成
任务11：配置路由
- 设置React Router
- 验证：路由正常工作
- 预计：5分钟

任务12：添加示例文章
- 创建2篇Markdown文章
- 验证：文章能正确加载
- 预计：5分钟

总计：约54分钟

是否开始执行？

Step 3：Subagent-Driven Development（执行）

```
用户：开始执行

AI：【触发subagent-driven-development skill】

启动子代理执行任务...

[子代理1] 执行任务1：初始化Vite项目
```

npm create vite@latest blog -- --template react-ts cd blog npm install

```
结果：✅ 成功

[子代理2] 执行任务2：安装依赖
```

npm install -D tailwindcss postcss autoprefixer npx tailwindcss init -p npm install react-router-dom react-markdown remark-gfm

```
结果：✅ 成功

[子代理3] 执行任务3：配置Tailwind CSS
...（详细配置过程）
结果：✅ 成功

[检查点] Phase 1完成，继续Phase 2？

用户：继续

[子代理4-6] 执行Phase 2任务...
结果：✅ 全部成功

[检查点] Phase 2完成，继续Phase 3？

用户：继续

[子代理7-10] 执行Phase 3任务...
结果：✅ 全部成功

[检查点] Phase 3完成，继续Phase 4？

用户：继续

[子代理11-12] 执行Phase 4任务...
结果：✅ 全部成功

所有任务完成！
```

Step 4：Self Code Review（代码审查）

```
AI：【触发requesting-code-review skill】

对完成的代码进行review...

## Code Review报告

### 问题1：中等 - 缺少错误处理
**位置**：usePosts hook
**问题**：加载文章失败时没有错误处理
**修复**：添加try-catch和错误状态

### 问题2：轻微 - 缺少加载状态
**位置**：Home页面
**问题**：文章加载时没有loading指示
**修复**：添加loading spinner

### 问题3：轻微 - 主题切换闪烁
**位置**：ThemeContext
**问题**：初始主题判断可能导致闪烁
**修复**：使用CSS变量避免闪烁

正在修复...

修复完成！
```

Step 5：安全检查

```
AI：
执行安全检查...

检查结果：✅ 通过
- 没有XSS风险（react-markdown自动转义）
- 没有敏感信息泄露
- 依赖包没有已知漏洞
```

Step 6：生成文档

```
AI：
生成项目文档...

已生成：
- README.md - 项目说明
- docs/deployment.md - 部署指南
- docs/adding-posts.md - 如何添加文章
```

最终输出:

```
项目已完成!

项目位置：./blog

文件统计：
- TypeScript/React文件：15个
- 测试文件：0个（按需求未添加）
- Markdown文章：2篇

运行项目：
```

cd blog npm run dev

```
部署到Vercel：
```

npm i -g vercel vercel

```
```

5.2 场景二：给现有Python项目添加新功能模块

需求描述

"我有一个Flask REST API项目，现在需要添加用户认证模块，包括注册、登录、JWT token刷新。"

完整工作流

Step 1：Brainstorming

```
用户：我有一个Flask REST API项目，现在需要添加用户认证模块，
      包括注册、登录、JWT token刷新。

AI：
让我先了解你的现有项目：

1. 项目结构是怎样的？
2. 目前使用什么数据库？
3. 有现有的用户模型吗？
4. 使用什么依赖管理工具？（pip/requirements.txt, poetry, pipenv？）
5. 有现有的测试框架吗？
6. 密码哈希策略有偏好吗？
7. JWT token的过期时间？
8. 需要刷新token功能吗？
```

用户回答：

```
1. 标准Flask项目结构，app/目录下有routes/, models/, services/
2. PostgreSQL，用SQLAlchemy
3. 没有用户模型
4. requirements.txt
5. pytest
6. 用bcrypt
7. access token 15分钟，refresh token 7天
8. 需要
```

AI生成设计文档：

```
# 用户认证模块设计

## 数据模型

### User模型
```python
```
```

```
class User(db.Model):
    __tablename__ = 'users'

    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password_hash = db.Column(db.String(255), nullable=False)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
    updated_at = db.Column(db.DateTime, default=datetime.utcnow, onupdate=datetime.utcnow)
    is_active = db.Column(db.Boolean, default=True)
```

RefreshToken模型

```
class RefreshToken(db.Model):
    __tablename__ = 'refresh_tokens'

    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('users.id'), nullable=False)
    token = db.Column(db.String(255), unique=True, nullable=False)
    expires_at = db.Column(db.DateTime, nullable=False)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
```

API端点

POST /api/auth/register

注册用户

POST /api/auth/login

用户登录

POST /api/auth/refresh

刷新access token

POST /api/auth/logout

用户登出（使refresh token失效）

JWT配置

- Access Token：15分钟
- Refresh Token：7天

- 算法：HS256

依赖

- PyJWT
- bcrypt
- flask-jwt-extended（可选，或自己实现）

```
**Step 2：Writing Plans**
```

AI： 实施计划（共10个任务）：

任务1：创建User模型 - 文件：app/models/user.py - 验证：能创建用户表

任务2：创建RefreshToken模型 - 文件：app/models/refresh_token.py - 验证：能创建token表

任务3：创建密码工具函数 - 文件：app/utils/password.py - 验证：能哈希和验证密码

任务4：创建JWT工具函数 - 文件：app/utils/jwt.py - 验证：能生成和验证token

任务5：实现注册接口 - 文件：app/routes/auth.py - 验证：能注册用户

任务6：实现登录接口 - 验证：能登录并返回token

任务7：实现刷新接口 - 验证：能用refresh token获取新access token

任务8：实现登出接口 - 验证：能使refresh token失效

任务9：添加认证装饰器 - 文件：app/decorators/auth.py - 验证：能保护路由

任务10：编写测试 - 文件：tests/test_auth.py - 验证：所有测试通过

预计总时间：60分钟

```
**Step 3：TDD执行**
```

AI：【触发TDD skill】

用TDD方式实现任务1（User模型）...

Red：写测试

```
# tests/test_user_model.py
def test_create_user(app):
    with app.app_context():
        user = User(
            username='testuser',
            email='test@example.com',
            password_hash='hashed_password'
        )
        db.session.add(user)
        db.session.commit()

        assert user.id is not None
        assert user.username == 'testuser'
        assert user.created_at is not None
```

运行测试：失败（User模型不存在）

Green：创建User模型

```
# app/models/user.py
from datetime import datetime
from app import db

class User(db.Model):
    __tablename__ = 'users'

    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password_hash = db.Column(db.String(255), nullable=False)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
    updated_at = db.Column(db.DateTime, default=datetime.utcnow, onupdate=datetime.utcnow)
    is_active = db.Column(db.Boolean, default=True)
```

运行测试：通过

Refactor：添加repr方法

```
def __repr__(self):
    return f'<User {self.username}>'
```

运行测试：通过

任务1完成，提交代码。

```
**Step 4-10**：继续用TDD方式完成其他任务...

**Step 5：Self Code Review**
```

AI： Review发现的问题：

1. 缺少输入验证（用户名长度、邮箱格式）
2. 没有处理数据库唯一性冲突
3. 错误信息可能泄露敏感信息

正在修复...

```
**最终输出**：
```

用户认证模块已完成！

新增文件： - app/models/user.py - app/models/refresh_token.py - app/utils/password.py - app/utils/jwt.py - app/routes/auth.py - app/decorators/auth.py - tests/test_auth.py

API端点： - POST /api/auth/register - POST /api/auth/login - POST /api/auth/refresh - POST /api/auth/logout

测试覆盖：94%

使用示例：

```
# 保护路由
from app.decorators.auth import jwt_required

@app.route('/api/protected')
@jwt_required
def protected():
    return {'message': 'This is protected'}
```

```

---
```

5.3 场景三：重构一个混乱的遗留代码库

需求描述

"我有一个旧项目，代码很混乱，有重复代码、命名不规范、没有测试。帮我重构一下。"

完整工作流

Step 1：分析现有代码

```
用户：我有一个旧项目，代码很混乱，有重复代码、命名不规范、没有测试。
      帮我重构一下。

AI：
让我先分析现有代码...

【AI分析代码库】

## 代码分析报告

### 项目结构
```

old-project/ ├── main.py # 500+行，包含所有逻辑 ├── utils.py # 混杂的工具函数 ├── db.py # 数据库连接 └── templates/ # HTML模板

```
### 发现的问题

#### 1. 严重 - 单体文件
**文件**：main.py
**问题**：500+行代码在一个文件，包含路由、业务逻辑、数据库操作
**风险**：难以维护、测试、协作

#### 2. 严重 - 重复代码
**问题**：数据库查询代码重复了5次
**示例**：
```python
# 重复1
cursor.execute("SELECT * FROM users WHERE id = %s", (user_id,))
user = cursor.fetchone()

# 重复2
cursor.execute("SELECT * FROM users WHERE id = %s", (id,))
user = cursor.fetchone()
```
```

3. 高 - 命名不规范

- 函数名：getData, process_info, handleItem（混合风格）
- 变量名：x, data1, temp（无意义）

4. 高 - 没有错误处理

```
# 问题代码
user = cursor.fetchone()
print(user['name'])  # 如果user为None会崩溃
```

5. 中 - 没有测试

- 0个测试文件
- 0%测试覆盖

6. 中 - SQL注入风险

```
# 危险代码
cursor.execute(f"SELECT * FROM users WHERE name = '{name}'")
```

重构计划

建议按以下顺序重构： 1. 提取数据库层（Repository模式） 2. 提取业务逻辑层（Service模式） 3. 规范化命名 4. 添加错误处理 5. 添加测试 6. 修复安全问题

```
**Step 2：制定重构计划**
```

AI： 重构计划（共8个阶段）：

阶段1：创建项目结构 - 创建app/目录结构 - 移动现有文件 - 验证：项目能运行

阶段2：提取数据库层 - 创建UserRepository类 - 创建PostRepository类 - 替换所有原始查询 - 验证：功能正常

阶段3：提取业务逻辑层 - 创建UserService类 - 创建PostService类 - 验证：功能正常

阶段4：规范化命名 - 统一使用snake_case - 函数名要有意义 - 验证：代码可读

阶段5：添加错误处理 - 添加try-except - 添加输入验证 - 验证：错误场景处理正确

阶段6：添加测试 - 为Repository层写测试 - 为Service层写测试 - 验证：测试覆盖>80%

阶段7：修复安全问题 - 使用参数化查询 - 添加输入过滤 - 验证：安全扫描通过

阶段8：清理 - 删除旧文件 - 更新文档 - 最终验证

```
**Step 3：执行重构**
```

AI：【使用git worktree创建重构分支】

创建孤立的工作空间进行重构...

[子代理执行重构任务]

阶段1完成：✅ 阶段2完成：✅ 阶段3完成：✅ ...

```
**Step 4：验证重构**
```

AI： 重构完成!

重构结果

代码统计对比

指标	重构前	重构后
总行数	850	620（-27%）
文件数	4	15
重复代码	多处	0
测试覆盖	0%	87%
安全问题	3个	0

新结构

```
app/
├── __init__.py
├── models/           # 数据模型
│   ├── __init__.py
│   ├── user.py
│   └── post.py
```

```
├── repositories/     # 数据访问层
│   ├── __init__.py
│   ├── user_repository.py
│   └── post_repository.py
├── services/         # 业务逻辑层
│   ├── __init__.py
│   ├── user_service.py
│   └── post_service.py
├── routes/           # 路由层
│   ├── __init__.py
│   ├── auth.py
│   └── posts.py
├── utils/            # 工具函数
│   ├── __init__.py
│   ├── validators.py
│   └── errors.py
└── templates/        # HTML模板
```

关键改进

1. **分层架构**：清晰的分层，职责明确
2. **DRY原则**：重复代码提取到Repository
3. **命名规范**：统一使用snake_case
4. **错误处理**：统一的错误处理机制
5. **测试覆盖**：87%的行覆盖
6. **安全修复**：所有SQL注入风险已消除 ```

5.4 场景四：修复一个复杂的跨模块Bug

需求描述

"我的应用有个bug：用户登录后，有时候个人资料页面显示的是另一个用户的数据。这个问题时有时无，很难复现。"

完整工作流

Step 1：Systematic Debugging（系统调试）

```
用户：我的应用有个bug：用户登录后，有时候个人资料页面显示的是
      另一个用户的数据。这个问题时有时无，很难复现。

AI：【触发systematic-debugging skill】

这是一个复杂的并发/状态问题。让我用系统化的方法调试。

## 调试计划

### Phase 1：收集信息

问题1：这个问题发生的频率？
- 每次登录都发生？
- 随机发生？
- 特定操作后发生？

问题2：什么情况下会发生？
- 多用户同时登录？
- 快速切换页面？
- 刷新页面后？

问题3：数据错误的模式？
- 总是显示同一个"另一个用户"？
- 随机显示不同用户？
- 显示的是上一个登录的用户？

问题4：技术栈？
- 前端：React/Vue/Angular？
- 状态管理：Redux/Vuex/Context？
- 后端：什么框架？
- 数据库：什么类型？

问题5：有日志吗？
- 前端console有错误吗？
- 后端日志有异常吗？
```

用户回答:

```
1. 随机发生，大概30%的概率
2. 快速刷新页面后容易发生
3. 显示的是上一个查看过资料的用户
4. React + Redux + Node.js + MongoDB
5. 没有明显错误日志
```

AI分析:

```
AI：
基于信息，初步怀疑是：

**最可能的原因**：Redux状态没有正确清理

当快速刷新页面时：
1. 用户A的资料被加载到Redux store
2. 页面刷新，但Redux状态可能持久化了
3. 用户B登录
4. 用户B查看资料页面
5. 由于某种原因，显示的是用户A的数据

**验证假设**：

让我检查相关代码...

【AI查看代码】

发现可疑代码：
```javascript
// userSlice.js
const userSlice = createSlice({
  name: 'user',
  initialState: {
    profile: null,  // 问题：刷新后可能保留旧值
    loading: false
  },
  reducers: {
    setProfile: (state, action) => {
      state.profile = action.payload;
    },
    // 问题：没有clearProfile action!
  }
});
```

```
// ProfilePage.jsx
useEffect(() => {
  dispatch(fetchProfile(userId));
```
```

```
}, []);  // 问题：依赖数组为空，只在mount时执行

// 如果Redux中已有profile，不会重新获取
```

根本原因： 1. Redux状态在页面刷新后可能持久化（用了redux-persist） 2. ProfilePage只在mount时获取数据 3. 如果store中已有profile，组件不会知道需要重新获取

```
**Step 2：制定修复计划**
```

AI： 修复计划：

任务1：添加clearProfile action - 在logout时清除profile - 在登录时清除旧profile

任务2：修复ProfilePage依赖 - 添加userId到依赖数组 - 当userId变化时重新获取

任务3：添加加载状态检查 - 确保在加载新数据前显示loading

任务4：编写测试 - 测试快速切换用户场景 - 测试刷新后数据正确性

```
**Step 3：执行修复**
```

AI： 【TDD方式修复】

Red：写测试复现bug

```
// ProfilePage.test.jsx
it('should fetch correct profile after user switch', async () => {
  // 先加载用户A的资料
  store.dispatch(setProfile({ id: 'A', name: 'User A' }));

  // 切换到用户B
  const { rerender } = render(<ProfilePage userId="B" />);

  // 应该获取用户B的资料，而不是显示用户A
  await waitFor(() => {
    expect(screen.getByText('User B')).toBeInTheDocument();
  });
});
```

测试失败（复现了bug）

Green：修复代码

```
// userSlice.js
reducers: {
  setProfile: (state, action) => {
    state.profile = action.payload;
  },
  clearProfile: (state) => {
    state.profile = null;
  }
}

// authSlice.js - login时
loginSuccess: (state, action) => {
  state.user = action.payload;
  // 清除旧的user profile
  dispatch(clearProfile());
}

// ProfilePage.jsx
useEffect(() => {
  dispatch(fetchProfile(userId));
}, [userId]);  // 添加userId依赖
```

测试通过

Refactor：优化代码...

```
**Step 4：验证修复**
```

AI： 验证修复效果：

1. 单元测试： ✅ 通过
2. 手动测试： - 登录用户A，查看资料 ✅ - 快速刷新，查看资料 ✅ - 登出，登录用户B ✅ - 用户B查看资料 ✅ - 重复10次，没有复现bug ✅
3. 边缘情况测试： - 网络慢时切换用户 ✅ - 同时打开多个标签页 ✅

修复成功！ ```

5.5 场景五：为开源项目贡献PR

需求描述

"我想给Superpowers项目贡献一个PR，添加一个新的skill。"

完整工作流

Step 1：理解贡献规范

```
用户：我想给Superpowers项目贡献一个PR，添加一个新的skill。

AI：
首先，让我查看Superpowers的贡献指南...

【读取CLAUDE.md和PULL_REQUEST_TEMPLATE.md】

## 贡献要求

根据CLAUDE.md：

1. 必须阅读完整的PR模板
2. 必须搜索现有PR（open和closed）
3. 必须确认这是真实问题
4. 必须确认属于core
5. 必须展示完整diff给用户确认

**不接受的内容**：
- 第三方依赖
- 项目特定的配置
- 批量PR
- 没有人类参与的PR

**Skill要求**：
- 必须是通用技能
- 必须经过测试
- 必须有评估证据
```

Step 2：搜索现有PR

```
AI：
搜索相关PR...

GitHub搜索："skill" + "is:pr"

发现：
```

```
- 已有5个关于新skill的open PR
- 已有12个关于新skill的closed PR

查看closed PR了解被拒绝的原因...

常见拒绝原因：
1. 技能太特定（只适用于某类项目）
2. 没有测试
3. 没有评估证据
4. 不符合项目风格
```

Step 3：设计新Skill

```
用户：我想添加一个"数据库迁移"skill，帮助用户安全地执行数据库迁移。

AI：
让我设计这个skill...

## Skill设计：database-migration

### 触发条件
- 用户提到"数据库迁移"
- 用户提到"migration"
- 检测到数据库schema变更

### 执行流程
1. 检查当前数据库状态
2. 生成迁移脚本
3. 备份数据（如果需要）
4. 在测试环境执行
5. 验证结果
6. 在生产环境执行

### 安全检查
- 破坏性操作需要确认
- 自动备份
- 可回滚

### 适用性评估
这个skill通用吗？
- ✅ 所有用数据库的项目都需要迁移
- ✅ 不依赖特定技术栈（可以适配各种ORM）
- ✅ 解决真实问题（迁移是常见痛点）

结论：适合提交到core
```

Step 4：实现Skill

AI：
【使用writing-skills skill开发新skill】

创建skill文件：

```markdown
# Database Migration

## 触发条件
- 用户提到"数据库迁移"、"migration"、"schema变更"
- 检测到模型文件变更

## 执行流程

### Step 1: 评估迁移类型
判断是：
- 新增表/字段（安全）
- 修改字段类型（需谨慎）
- 删除字段/表（危险）
- 数据迁移（复杂）

### Step 2: 生成迁移计划
根据ORM类型生成对应命令：
- Django: `makemigrations`
- Rails: `rails generate migration`
- SQLAlchemy: `alembic revision`
- Prisma: `prisma migrate dev`

### Step 3: 安全检查
如果是危险操作：
- 提示用户确认
- 建议先备份
- 提供回滚方案

### Step 4: 执行迁移
1. 在开发环境执行
2. 运行测试
3. 在生产环境执行（如果用户确认）

## 禁止事项
- 不备份直接执行破坏性操作
- 不测试直接上生产
- 不回滚失败的操作

## 输出
- 迁移脚本
- 执行日志
- 回滚方案
```

Step 5：测试Skill

```
AI：
测试新skill...

场景1：新增字段
```

用户：我要给User表添加一个phone字段 AI：【触发database-migration skill】 检测到新增字段操作（安全） 生成迁移：alembic revision --autogenerate -m "add phone to user" 执行测试... 测试通过

```
场景2：删除字段（危险操作）
```

用户：我要删除User表的email字段 AI：【触发database-migration skill】 ⚠️ 警告：这是破坏性操作！ - 将删除email字段 - 该字段的数据将丢失 - 依赖该字段的代码将出错

建议： 1. 先备份数据库 2. 确认没有代码依赖该字段 3. 使用软删除（添加deleted_at）而不是硬删除

是否继续?

```
测试结果：✅ 符合预期
```

Step 6：准备PR

```
AI：
准备PR...

1. Fork仓库 ✅
2. 创建分支：`git checkout -b feat/database-migration-skill` ✅
3. 提交代码：`git commit -m "feat: add database migration skill"` ✅
4. 推送分支：`git push origin feat/database-migration-skill` ✅

填写PR模板：

## 描述
添加database-migration skill，帮助用户安全地执行数据库迁移。

## 解决的问题
数据库迁移是常见痛点，容易出错。这个skill提供：
- 自动检测迁移类型
- 安全检查（破坏性操作警告）
- 多ORM支持

## 测试
- [x] 单元测试通过
```

```
- [x] 手动测试通过（Django + PostgreSQL）
- [x] 手动测试通过（SQLAlchemy + MySQL）

## 环境
| 环境 | 结果 |
|------|------|
| Claude Code | ✅ |
| Cursor | 未测试 |

## 现有PR
搜索发现没有类似的open PR。
Closed PR #123有类似功能但已放弃。

## 评估证据
测试了5个真实迁移场景，全部成功。

请确认这个PR可以提交？

用户：确认

AI：
PR已提交！
```

5.6 场景六：快速开发一个API后端服务

需求描述

"我需要快速开发一个REST API，用于一个电商应用，包括商品、订单、用户管理。"

完整工作流

Step 1：Brainstorming

```
用户：我需要快速开发一个REST API，用于一个电商应用，
      包括商品、订单、用户管理。

AI：
让我了解具体需求：

1. 技术栈偏好？
2. 数据库选择？
3. 认证方式？
4. 需要哪些具体功能？
   - 商品：CRUD、搜索、分类、库存？
   - 订单：创建、支付、状态跟踪？
   - 用户：注册、登录、个人中心？
5. 性能要求？
6. 部署环境？
```

用户回答：

```
1. Node.js + Express + TypeScript
2. PostgreSQL
3. JWT
4. 商品CRUD+搜索，订单创建+状态，用户注册+登录
5. 普通水平，先跑起来
6. 先本地，后续可能上云
```

AI生成设计文档：

```
# 电商API设计

## 技术栈
- Node.js + Express + TypeScript
- PostgreSQL + Prisma ORM
- JWT认证
- Jest测试
```

```
## 数据模型

### User
- id, email, passwordHash, name, createdAt

### Product
- id, name, description, price, stock, category, createdAt

### Order
- id, userId, status, totalAmount, createdAt

### OrderItem
- id, orderId, productId, quantity, price

## API端点

### 认证
- POST /api/auth/register
- POST /api/auth/login
- GET /api/auth/me

### 商品
- GET /api/products - 列表（支持搜索、分页）
- GET /api/products/:id - 详情
- POST /api/products - 创建（管理员）
- PUT /api/products/:id - 更新（管理员）
- DELETE /api/products/:id - 删除（管理员）

### 订单
- GET /api/orders - 我的订单
- POST /api/orders - 创建订单
- GET /api/orders/:id - 订单详情
- PATCH /api/orders/:id/status - 更新状态（管理员）

## 项目结构
```

src/ ├── config/ # 配置 ├── controllers/ # 控制器 ├── middleware/ # 中间件 ├── models/ # Prisma模型 ├── routes/ # 路由 ├── services/ # 业务逻辑 ├── types/ # TypeScript类型 ├── utils/ # 工具函数 └── tests/ # 测试

```
```

Step 2：Writing Plans

```
AI:
实施计划（共15个任务，预计90分钟）：

Phase 1: 项目初始化（15分钟）
```

```
- 任务1：初始化项目
- 任务2：配置Prisma
- 任务3：创建数据模型

Phase 2：认证模块（20分钟）
- 任务4：实现注册
- 任务5：实现登录
- 任务6：实现me接口
- 任务7：添加认证中间件

Phase 3：商品模块（25分钟）
- 任务8：商品CRUD
- 任务9：商品搜索
- 任务10：商品测试

Phase 4：订单模块（25分钟）
- 任务11：创建订单
- 任务12：订单查询
- 任务13：订单状态管理
- 任务14：订单测试

Phase 5：集成（5分钟）
- 任务15：整合测试
```

Step 3：执行开发

```
AI：【subagent-driven-development】

启动子代理执行任务...

[执行过程略，类似前面场景]

任务1-15全部完成！
```

Step 4：测试与验证

```
AI：
运行测试...

```bash
npm test
```

结果： - 测试套件：15个 - 测试用例：47个 - 通过：47个 - 失败：0个 - 覆盖率：91%

API验证：
```

```
# 注册
curl -X POST http://localhost:3000/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{"email":"test@example.com","password":"Test1234","name":"Test"}'

# 登录
curl -X POST http://localhost:3000/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"test@example.com","password":"Test1234"}'

# 创建商品（需要token）
curl -X POST http://localhost:3000/api/products \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer <token>" \
  -d '{"name":"iPhone","description":"Smartphone","price":999,"stock":100}'
```

全部正常！

```
**最终输出**：
```

电商API已完成！

项目位置：./ecommerce-api

功能清单： ✅ 用户注册/登录/JWT认证 ✅ 商品CRUD ✅ 商品搜索（支持关键字、分页） ✅ 订单创建 ✅ 订单查询 ✅ 订单状态管理

API文档：./ecommerce-api/API.md

快速开始：

```
cd ecommerce-api
npm install
# 配置数据库连接（.env）
npx prisma migrate dev
npm run dev
```

测试：

```
npm test
```

部署：

```
# 构建
npm run build

# 启动生产环境
npm start
```

```

---
```

本章小结

六大场景展示了Superpowers在不同情况下的应用：

场景	关键技能	学习时间
新项目搭建	Brainstorming + Planning	基础
添加功能模块	TDD + Self Review	基础
重构遗留代码	Systematic + Planning	进阶
修复复杂Bug	Debugging + TDD	进阶
开源贡献	Writing Skills + Review	高级
快速开发	All Skills Combined	综合

动手练习

1. 选择你最熟悉的场景，用Superpowers走一遍完整流程
2. 记录每个步骤的时间和遇到的问题
3. 对比你平时的开发方式，评估效率提升

CHAPTER SIX

第六章 进阶技巧与组合拳

6.1 Superpowers + Context7 MCP联合使用

6.2 Superpowers + Firecrawl联合使用

6.3 自定义Skills：如何写自己的工作流规则

6.4 CLAUDE.md最佳实践

6.5 长会话管理策略

本章小结

动手练习

6.1 Superpowers + Context7 MCP联合使用

本章核心价值：掌握Superpowers的高级用法，学会与其他工具配合使用。

什么是Context7 MCP

Context7 MCP是一个为AI提供项目上下文知识的工具。它可以： - 索引你的代码库 - 提供语义搜索 - 回答关于代码的问题

联合使用场景

场景：你需要在一个大型项目中添加功能，但不熟悉项目结构。

```
用户：我要在这个大型项目中添加一个缓存层，但我对项目结构不熟悉。

AI：
让我先使用Context7了解项目结构...

【调用Context7 MCP】

Context7分析结果：
- 项目使用分层架构：Controller -> Service -> Repository
- 已有Redis配置在 config/redis.ts
- 数据访问层在 repositories/ 目录
- 主要实体：User, Product, Order

基于这些信息，我来设计缓存层...

【触发Superpowers brainstorming】

建议的缓存策略：
1. 在Repository层添加缓存
2. 使用Redis作为缓存存储
3. 缓存热点数据（用户资料、商品信息）
4. 设置合理的过期时间

是否继续制定详细计划？
```

配置方法

1. 安装Context7 MCP：

```
# 在Claude Code中
/mcp add context7
```

2. 索引你的项目：

```
context7 index ./
```

3. 在CLAUDE.md中添加：

```
## 工具使用
- 使用Context7 MCP了解项目结构
- 使用Superpowers执行开发工作流
```

最佳实践

1. **先Context7后Superpowers**：用Context7获取上下文，用Superpowers执行
2. **定期重新索引**：代码变化后重新索引
3. **结合使用**：Context7回答问题，Superpowers执行任务

6.2 Superpowers + Firecrawl联合使用

什么是Firecrawl

Firecrawl是一个网站抓取工具，可以： - 抓取网页内容 - 提取结构化数据 - 生成markdown

联合使用场景

场景：你需要参考一个网站的API设计。

```
用户：我要设计一个支付API，参考Stripe的API设计。

AI：
让我先用Firecrawl抓取Stripe的API文档...

【调用Firecrawl】

抓取结果：
- Stripe API使用RESTful设计
- 统一返回格式：{ object, data, has_more }
- 错误格式：{ error: { type, message, code } }
- 认证：Bearer token
- 版本控制：通过Stripe-Version header

【触发Superpowers brainstorming】

基于Stripe的设计，我们的支付API设计：

### 统一响应格式
```json
{
  "success": true,
  "data": { ... },
  "meta": {
    "page": 1,
    "per_page": 20,
    "total": 100
  }
}
```

错误格式

```
{
  "success": false,
  "error": {
```
```

```
    "type": "validation_error",
    "message": "Invalid amount",
    "code": "amount_invalid"
  }
}
```

认证

- Header: `Authorization: Bearer <token>`

是否确认这个设计?

```
#### 配置方法

1. 安装Firecrawl MCP:

```bash
/mcp add firecrawl
```

2. 配置API key（如果需要）

最佳实践

1. **抓取文档而非整站**：指定具体的文档URL
2. **缓存结果**：避免重复抓取
3. **验证信息**：抓取的文档可能不是最新的

```

6.3 自定义Skills：如何写自己的工作流规则

为什么要自定义Skills

Superpowers内置的skills是通用的，但你的团队可能有特定的工作流需求：

- 特定的代码审查清单
- 特定的文档格式
- 特定的部署流程

创建自定义Skill

Step 1：创建skill目录

```
mkdir -p .superpowers/skills/my-team-workflow
```

Step 2：编写SKILL.md

```
# My Team Workflow

## 触发条件
- 项目使用了本团队的tech stack
- 用户提到"按团队规范"

## 执行流程

### Step 1: 检查CLAUDE.md
确认项目有团队规范文件

### Step 2: 代码审查
按团队规范审查代码：
- [ ] 函数长度不超过50行
- [ ] 每个函数有JSDoc注释
- [ ] 使用团队统一的错误处理方式
- [ ] 测试覆盖率>80%

### Step 3: 文档检查
- [ ] API有文档
- [ ] 复杂逻辑有注释

## 输出
- 审查报告
- 不符合项列表
```

Step 3：注册skill

在CLAUDE.md中添加：

```
## 团队规范
- 使用自定义skill：my-team-workflow
- 代码审查必须通过这个skill
```

示例：代码审查Skill

```
# Code Review Checklist

## 触发条件
- 代码完成后
- 用户说"review代码"

## 检查清单

### 代码质量
- [ ] 函数单一职责
- [ ] 变量命名清晰
- [ ] 没有魔法数字
- [ ] 错误处理完善

### 测试
- [ ] 有单元测试
- [ ] 测试覆盖边界情况
- [ ] 测试可读性好

### 文档
- [ ] 公共API有JSDoc
- [ ] 复杂算法有注释
- [ ] README已更新

### 安全
- [ ] 没有SQL注入风险
- [ ] 没有XSS风险
- [ ] 敏感信息未泄露

## 输出格式
```

Code Review结果

严重问题（必须修复）

- [ ] 问题1

警告（建议修复）

- [ ] 问题2

建议（可选）

- [ ] 建议1

```
```

最佳实践

1. **从简单开始**：先写一个检查清单skill
2. **测试你的skill**：确保它能正确触发
3. **文档化**：让团队成员知道有这个skill
4. **定期更新**：根据团队反馈改进

6.4 CLAUDE.md最佳实践

CLAUDE.md的作用

CLAUDE.md是项目级别的AI指令文件，Superpowers会读取它来了解项目规范。

推荐结构

```
# 项目规范

## 项目概述
- 项目名称：XXX
- 技术栈：React + TypeScript + Node.js
- 项目目标：XXX

## 技术栈详情
### 前端
- React 18
- TypeScript 5
- Tailwind CSS
- React Router v6

### 后端
- Node.js 18
- Express
- PostgreSQL
- Prisma ORM

## 代码规范
### 命名规范
- 变量：camelCase
- 常量：UPPER_SNAKE_CASE
- 组件：PascalCase
- 文件：kebab-case

### 代码风格
- 使用ESLint + Prettier
- 函数长度不超过50行
- 圈复杂度不超过10

### 注释规范
- 公共API必须有JSDoc
- 复杂逻辑必须注释
- TODO必须带作者和日期

## 测试规范
- 测试覆盖率>80%
```

```
- 单元测试用describe/it
- 集成测试用setup/teardown

## Superpowers配置
### 强制设置
- 强制TDD：true
- 强制Code Review：true
- 子代理模型：claude-3-5-sonnet-20241022

### 自定义规则
- 提交前必须运行测试
- API变更必须更新文档
- 数据库变更必须写migration

## 部署规范
- 主分支自动部署到staging
- 生产部署需要手动触发
- 部署前必须所有检查通过
```

让Superpowers和项目规范配合

示例1：强制测试覆盖

```
## 测试要求
- 所有代码必须通过TDD编写
- 测试覆盖率不得低于80%
- 未覆盖的代码必须有注释说明原因

Superpowers执行TDD时：
1. 先写测试
2. 运行测试确认失败
3. 写代码让测试通过
4. 检查覆盖率
5. 覆盖率不足时补充测试
```

示例2：强制代码审查

```
## 代码审查要求
- 所有代码必须通过Self Code Review
- Critical问题必须修复
- High问题必须修复或有理由
- 审查清单：
  - [ ] 代码符合规范
  - [ ] 测试覆盖充分
  - [ ] 文档已更新
  - [ ] 安全检查通过
```

示例3：数据库变更规范

```
## 数据库变更
- 所有变更必须通过migration
- migration必须可回滚
- 破坏性变更需要：
  1. 备份数据
  2. 双写阶段（如果可能）
  3. 验证后删除旧字段

Superpowers执行数据库变更时：
1. 检测变更类型
2. 如果是破坏性变更，警告用户
3. 生成migration文件
4. 提供回滚方案
```

踩坑记录

坑：CLAUDE.md太长

症状：AI没有按CLAUDE.md执行

原因：CLAUDE.md太长，超出了AI的上下文窗口

解决： - 保持CLAUDE.md在500行以内 - 把详细规范放在单独文件 - 在CLAUDE.md中引用

坑：CLAUDE.md和skill冲突

症状：CLAUDE.md的要求和skill的行为不一致

解决： - 确保CLAUDE.md和skill一致 - 如果冲突，以CLAUDE.md为准 - 在CLAUDE.md中明确覆盖skill的默认行为

6.5 长会话管理策略

为什么需要管理会话

AI的上下文窗口有限，长会话会导致： - 早期信息丢失 - AI"忘记"之前的决策 - 响应变慢

策略1：使用Checkpoint

Superpowers会自动创建Checkpoint，你也可以手动创建：

```
用户：创建一个checkpoint

AI：
Checkpoint已创建：
- 时间：2024-01-15 14:30
- 状态：订单模块开发完成
- 下一步：支付模块开发
```

策略2：分阶段开发

每个大功能作为一个独立会话：

```
会话1：用户模块开发
- Brainstorming
- Planning
- Development
- Review
- Checkpoint

[结束会话，开始新会话]

会话2：商品模块开发
- 从Checkpoint恢复
- Brainstorming
- Planning
- Development
- Review
- Checkpoint
```

策略3：什么时候该新开会话

情况	建议
对话超过50轮	新开会话
切换大功能	新开会话
AI开始"忘记"事情	新开会话
性能明显下降	新开会话
需要清理上下文	新开会话

策略4：会话恢复流程

```
新会话启动
    ↓
检查是否有未完成的Checkpoint
    ↓
如果有：询问用户是否恢复
    ↓
恢复Checkpoint状态
    ↓
继续开发
```

最佳实践

1. **主动创建Checkpoint**：完成重要阶段时主动创建
2. **命名要清晰**：Checkpoint名称要反映状态
3. **定期清理**：删除过期的Checkpoint
4. **记录关键决策**：在Checkpoint中记录重要决策

本章小结

进阶技巧总结：

技巧	用途	难度
+ Context7	了解项目上下文	简单
+ Firecrawl	参考外部文档	简单
自定义Skills	团队规范	中等
CLAUDE.md	项目规范	中等
会话管理	长项目开发	中等

动手练习

1. 为你的项目写一个CLAUDE.md
2. 尝试创建一个自定义skill
3. 练习使用Checkpoint管理长会话

CHAPTER SEVEN

第七章 常见问题FAQ（20个Q&A）

安装问题

使用问题

效果问题

兼容性问题

付费相关问题

本章小结

安装问题

本章核心价值：快速解决使用Superpowers时遇到的问题。

Q1: Superpowers安装失败，提示"Plugin not found"

A: 1. 确认市场已添加：`/plugin marketplace list` 2. 如果没有obra/superpowers-marketplace，重新添加：`bash /plugin marketplace add obra/superpowers-marketplace` 3. 重试安装

Q2: 安装后AI行为没有变化

A: 1. 重启Claude Code会话（完全退出再进入） 2. 检查skill是否触发：输入"帮我设计一个功能"看是否触发brainstorming 3. 检查CLAUDE.md是否有冲突 4. 重新安装插件

Q3: Windows上hooks不执行

A: - 使用WSL2运行Claude Code - 或者在Git Bash中运行 - 检查文件权限：`chmod +x hooks/*/*.sh`

Q4: 如何更新Superpowers

A:

```
/plugin update superpowers
```

Q5: 如何卸载Superpowers

A:

```
/plugin uninstall superpowers
```

使用问题

Q6: Brainstorming没有触发

A: - 明确说"我们先brainstorm一下" - 或者"帮我设计XXX" - 避免太具体的初始prompt

Q7: AI直接写代码，没有先规划

A: - 明确说"先不要写代码，我们先讨论设计" - 在CLAUDE.md中添加强制规划的要求

Q8: TDD没有执行

A: - 明确说"用TDD方式实现" - 或者"先写测试" - 检查test-driven-development skill是否正常

Q9: Self Code Review太简单

A: - 在CLAUDE.md中详细定义review要求 - 要求AI按检查清单逐项review

Q10: 子代理执行失败

A: 1. 查看子代理日志 2. 检查任务是否太复杂 3. 手动执行失败的步骤 4. 调整任务粒度

效果问题

Q11: 代码质量没有明显提升

A: 1. 检查skill是否正常触发 2. 在CLAUDE.md中明确质量标准 3. 要求更严格的code review 4. 添加自定义skill

Q12: 开发速度变慢了

A: - 初期确实会慢，因为增加了规划环节 - 长期看会减少返工，总体更快 - 对于原型开发，可以简化流程

Q13: AI"忘记"之前的决策

A: - 这是上下文限制，正常现象 - 使用Checkpoint保存关键决策 - 长会话时新开会话

Q14: 生成的代码不符合项目风格

A: - 完善CLAUDE.md中的代码规范 - 提供示例代码 - 添加自定义skill强制执行风格

Q15: 测试覆盖不够

A: - 在CLAUDE.md中明确覆盖率要求 - 要求AI报告覆盖率 - 不通过coverage check不允许提交

兼容性问题

Q16: Superpowers支持哪些平台

A: - Claude Code（官方市场） - Cursor（插件市场） - Codex（手动安装） - OpenCode（手动安装） - GitHub Copilot CLI - Gemini CLI

Q17: 不同平台的Superpowers有区别吗

A: - 核心功能一致 - 安装方式不同 - 可能有平台特定的限制

Q18: 可以和其它插件一起用吗

A: - 一般可以 - 但可能有冲突 - 建议先单独使用Superpowers

Q19: 支持哪些编程语言

A: - Superpowers是语言无关的 - 但内置skills对某些语言支持更好 - 可以自定义skill支持特定语言

Q20: 支持哪些项目类型

A: - Web应用 - API服务 - 命令行工具 - 库/框架 - 几乎所有软件项目

付费相关问题

Q21: Superpowers是免费的吗

A: - Superpowers本身是开源免费的（MIT License） - 但你使用的AI平台（Claude Code等）可能收费

Q22: 使用Superpowers会增加API费用吗

A: - 可能会，因为Superpowers会增加AI调用次数 - 但代码质量提升可能减少总体开发成本

Q23: 如何支持Superpowers开发

A: - GitHub Sponsor：https://github.com/sponsors/obra - 贡献代码 - 传播推广

本章小结

常见问题分类：

类别	问题数	主要解决方式
安装	5	检查市场、重启、权限
使用	5	明确指令、CLAUDE.md
效果	5	调整期望、完善规范
兼容性	5	查看文档、测试

CHAPTER EIGHT

第八章　速查表与备忘录

8.1 所有指令速查表

8.2 推荐工作流速查

8.3 配置参数速查表

8.4 常见报错速查表

本章小结

8.1 所有指令速查表

本章核心价值：快速查找指令、参数、解决方案。

Superpowers相关指令

指令	作用	平台
`/plugin install superpowers@claude-plugins-official`	安装Superpowers	Claude Code
`/plugin update superpowers`	更新Superpowers	Claude Code
`/plugin uninstall superpowers`	卸载Superpowers	Claude Code
`/plugin list`	列出已安装插件	Claude Code
`/plugin marketplace add obra/superpowers-marketplace`	添加市场	Claude Code
`/add-plugin superpowers`	安装插件	Cursor

触发Skills的指令

指令	触发的Skill
"帮我设计XXX" / "brainstorm XXX"	brainstorming
"制定计划" / "写计划"	writing-plans
"用TDD实现" / "先写测试"	test-driven-development
"review代码" / "检查代码"	requesting-code-review
"执行计划" / "开始开发"	subagent-driven-development
"修复bug" / "debug"	systematic-debugging

8.2 推荐工作流速查

标准工作流

```
需求 → Brainstorming → 设计确认 → Planning → TDD开发 → Code Review → 提交
```

快速原型工作流

```
需求 → 简化Brainstorming → 快速实现 → 手动测试 → 提交
```

Bug修复工作流

```
Bug报告 → Systematic Debugging → 根因分析 → TDD修复 → 验证 → 提交
```

重构工作流

```
代码分析 → 制定重构计划 → 创建Worktree → 逐步重构 → 全面测试 → 合并
```

开源贡献工作流

```
阅读贡献指南 → 搜索现有PR → 设计变更 → 实现 → 测试 → 准备PR → 提交
```

8.3 配置参数速查表

CLAUDE.md常用配置

```
## Superpowers配置

### 强制设置
- 强制TDD: true/false
- 强制Code Review: true/false
- 强制安全检查: true/false

### 子代理设置
- 子代理模型: claude-3-5-sonnet-20241022
- 检查点间隔: 每N个任务

### 质量标准
- 测试覆盖率: >80%
- 函数最大行数: 50
- 圈复杂度: <10

### 开发规范
- 提交前运行测试: true
- API变更更新文档: true
- 数据库变更写migration: true
```

8.4 常见报错速查表

报错	原因	解决方案
Plugin not found	市场未添加	添加市场后重试
Version mismatch	Claude Code版本低	升级Claude Code
Permission denied	权限不足	修改权限或用nvm
Network timeout	网络问题	检查网络或代理
Skill not triggering	缓存问题	重启会话
Hooks not executing	权限或环境	检查权限，用WSL
Context lost	会话太长	新开会话
Out of memory	内存不足	减少并发子代理

本章小结

速查表是日常使用的参考，建议： 1. 打印或保存到容易找到的地方 2. 根据自己的使用习惯补充 3. 定期更新

APPENDIX

附录

A. 推荐搭配的其他Claude Code插件清单

B. 学习资源与社区链接

C. 版本更新日志跟踪方法

A. 推荐搭配的其他Claude Code插件清单

插件名	作用	推荐度
Context7	代码库索引和语义搜索	★★★★★
Firecrawl	网页抓取	★★★★☆
GitLens	Git增强	★★★★☆
Todo Tree	TODO管理	★★★☆☆

B. 学习资源与社区链接

官方资源：- GitHub：https://github.com/obra/superpowers - 文档：https://github.com/obra/superpowers/tree/main/docs - 发布日志：https://github.com/obra/superpowers/releases

社区：- Discord：https://discord.gg/Jd8Vphy9jq - 邮件订阅：https://primeradiant.com/superpowers/

相关文章：- Superpowers for Claude Code

C. 版本更新日志跟踪方法

1. **GitHub Releases**： - 访问 https://github.com/obra/superpowers/releases - 订阅Release通知
2. **邮件订阅**： - 访问 https://primeradiant.com/superpowers/ - 填写邮箱订阅
3. **Discord**： - 加入Discord服务器 - 关注announcements频道
4. **更新命令**： `bash /plugin update superpowers`

ROADMAP

学习路线图

新手阶段（1-2周）

进阶阶段（2-4周）

高级阶段（1-2月）

专家阶段（持续）

新手阶段（1-2周）

目标：能独立使用Superpowers完成简单功能

学习内容： 1. 安装和配置Superpowers 2. 理解核心概念（Skills、工作流） 3. 掌握Brainstorming和Planning 4. 体验TDD流程 5. 完成2-3个小功能

检验标准： - 能独立触发并使用brainstorming skill - 能理解并确认AI生成的设计文档 - 能走完完整的TDD流程

进阶阶段（2-4周）

目标：能熟练使用Superpowers完成复杂功能

学习内容： 1. 掌握所有核心功能 2. 学会写CLAUDE.md 3. 掌握Checkpoint管理 4. 学会处理复杂场景（重构、Bug修复） 5. 完成1个完整项目

检验标准： - 能独立完成第五章的任意2个场景 - 能写出合格的CLAUDE.md - 能管理长会话

高级阶段（1-2月）

目标：能定制Superpowers适应团队需求

学习内容： 1. 创建自定义Skills 2. 与其他工具集成（Context7、Firecrawl） 3. 为团队制定Superpowers使用规范 4. 贡献开源（可选）

检验标准： - 能创建并测试自定义skill - 能为团队写出完整的CLAUDE.md - 能指导他人使用Superpowers

专家阶段（持续）

目标：成为Superpowers专家，能优化和改进

学习内容： 1. 深入理解Skills Framework 2. 参与Superpowers开源贡献 3. 分享经验（博客、演讲） 4. 探索新用法

EPILOGUE

结语

恭喜你读完了这份手册！

Superpowers不是一个魔法按钮，它不会自动让你成为更好的开发者。但它是一个强大的工具，如果你愿意：

1. **遵循它的工作流**：规划→计划→TDD→审查
2. **投入时间学习**：理解"为什么"而不仅是"怎么做"
3. **持续改进**：根据你的需求定制和优化

那么它会回报你： - 更一致的代码质量 - 更少的返工 - 更清晰的开发过程 - 更多的时间去解决真正有趣的问题

记住：**工具是为人服务的，不是反过来。**Superpowers是一个强大的工具，但最终的决策权在你手中。用它来提升你的效率，而不是替代你的思考。

祝你在AI辅助开发的道路上越走越远！

本手册基于Superpowers v5.0.7编写

最后更新：2026年4月

文档统计： - 总字数：约18000字 - 章节数：8章 + 附录 - 代码示例：50+ - 实战场景：6个 - FAQ：20个

会员专属资料 · 持续更新

把 CC 用出生产力的地方

「河马的CC商业实战」星球里，每周都有看得懂、用得上的 AI 实战：

新栏目「河马带你逛GitHub」每周一期，实战长文、会员资料持续更新。

微信扫码 · 加入「河马的CC商业实战」

河马的CC商业实战 出品