第 9 章：AI 项目开发提示词

用一套连贯 prompt 推动选题、规划、实现与交付

这一章看什么

- 为什么“会问模型”本身也可以成为一套项目开发方法
- 如何把一组 prompt 串成一条完整的项目开发流程
- 每个 prompt 分别解决什么问题
- 怎么从选题一路走到设计、实现、调试和验收

先给这章一个总判断

- 这组 prompt 不是“随便问问 AI”
- 它更像一套轻量的 AI 项目开发工作流
- 它的核心价值不是替你写代码
- 而是逼你把问题定义清楚、把计划拆清楚、把证据讲清楚

这套 workflow 的主线

1. `prompt/0-office-hours`：先判断你到底在做什么项目
2. `prompt/1-brainstorming`：把模糊想法变成清晰设计
3. `prompt/2-plan-ceo-review`：从价值和范围上重新挑战
4. `prompt/3-writing-plans`：把确认过的设计拆成执行计划
5. `prompt/4-plan-eng-review`：判断这份计划能不能安全开工
6. `prompt/7-front-end-design`：如果项目面向用户界面，先把前端审美方向和实现要求说清楚
7. `prompt/6-test-driven-development`：先定义测试，再写实现
8. `prompt/8-bug-workflow`：出问题后走完整个闭环：调查、定因、修复原则与 fresh verification

为什么这组 prompt 有价值

- 很多项目失败，不是因为代码太难
- 而是因为问题没问清、范围没收住、验证没做好
- 这组 prompt 把项目开发中最容易偷懒的地方都单独拉出来了
- 它其实是在训练一种更成熟的工程思维

第一阶段：先判断你到底在做什么

对应文件：`prompt/0-office-hours.md`

- 先区分：
  - 创业项目
  - 内部项目
  - 课程 / hackathon
  - 研究探索
  - 学习练手
  - 兴趣项目
- 因为不同场景，最重要的问题完全不同
- 这一轮的目标不是实现，而是重构问题

0-office-hours 真正在做什么

- 如果是创业 / 内部项目，就追问真实需求和最窄切口
- 如果是课程 / 学习 / 研究项目，就追问亮点、展示效果和最小版本
- 最终产出第一版设计文档

一句话说：

- 它解决的是“我们是不是在做对的事”

第二阶段：把模糊 idea 变成清晰设计

对应文件：`prompt/1-brainstorming.md`

- 这一轮不讨论实现
- 只讨论：
  - 用户目标
  - 关键场景
  - 范围边界
  - 不做什么
  - 成功标准

它解决的是：

- “这个想法到底要长成什么样”

第三阶段：CEO 视角重新审查

对应文件：`prompt/2-plan-ceo-review.md`

- 不是默认接受当前方案
- 而是主动挑战：
  - 范围是不是太散
  - 方向是不是太弱
  - 有没有更高价值的切口
- 它要求你重新看：
  - 核心问题
  - 关键前提
  - 机会点
  - 风险点

为什么要专门做一轮 CEO review

- 很多同学容易一开始就把项目做小、做碎、做成“功能拼盘”
- 这一步是为了强迫你回答：
  - 这个项目最值得被记住的点是什么
  - 什么值得扩大
  - 什么必须砍掉

它解决的是：

- “这个项目值不值得这样做”

第四阶段：开始写 实现计划

对应文件：`3-writing-plans.md`

- 到这一步，方向已经大体成立
- 现在不能再停留在“做一个系统”“做一个平台”这种空话
- 必须开始拆：
  - 目标
  - 约束
  - 涉及文件 / 模块
  - 任务拆分
  - 验证方法

为什么 实现计划 要拆得很细

- 因为 AI 最擅长执行“清楚的小任务”
- 最怕执行“模糊的大目标”
- 所以这一步不是为了写文档而写文档
- 而是为了把任务拆到足够可执行、可验证、可交接

第五阶段：做工程评审

对应文件：`4-plan-eng-review.md`

- 前一轮已经有 plan 了
- 这一轮要问的是：
  - 架构是不是合理
  - 数据流是不是清楚
  - 状态转移有没有漏洞
  - failure mode 有没有想过
  - 测试策略够不够

3-writing-plans 和 4-plan-eng-review 的区别

- 3-writing-plans
  - 把事情拆出来
  - 目标是“可执行”
- 4-plan-eng-review
  - 把计划拿来审
  - 目标是“可安全开工”

一句话说：

- 一个负责写计划
- 一个负责判断计划能不能真的落地

到这里，项目才算真的可以开始写代码

- 很多团队的问题是：
  - brainstorming 一做完就开始写
- 但更稳的顺序应该是：
  - idea
  - design
  - CEO review
  - 实现计划
  - engineering review
  - 如果是 UI 项目，先做 front-end design
  - 再开始实现

插一个常被忽略、但非常关键的阶段：前端设计

对应文件：`7-front-end-design.md`

- 不是让模型“随便美化一下界面”
- 而是要求先明确：
  - 这个界面的目的是什么
  - 它面向谁
  - 应该走什么审美方向
  - 哪些字体、配色、动效、空间构成和背景细节才真的匹配这个项目
- 它特别强调：
  - 不要落回通用化、模板化、千篇一律的 AI 界面风格
  - 要有明确的设计立场
  - 实现复杂度要和审美愿景匹配

为什么这一步应该放在实现前

- 很多同学做 UI 项目时，最大的错误不是“不会写前端”
- 而是：
  - 设计方向没定
  - 页面气质没定
  - 组件一边写一边想
  - 最后做成一个能用、但完全记不住的默认界面

所以 `7-front-end-design` 真正解决的是：

- “如果这个项目要被人看到，它到底应该长成什么样”

7-front-end-design 和前面几轮是什么关系

- 0 到 2
  - 先判断项目值不值得做、应该做成什么
- 3 到 4
  - 再判断工程上能不能安全开工
- 7
  - 如果这是一个面向用户的产品、页面或 demo，就进一步明确视觉方向与交互表达

一句话说：

- 前几轮解决“做什么”
- 7 解决“让用户看到的那一层，应该怎么做得像样而且有辨识度”

第六阶段：进入 TDD

对应文件：`6-test-driven-development.md`

- 这一轮明确要求：
  - 先定义行为
  - 先写失败测试
  - 再做最小实现
  - 最后再重构

这一步最关键的价值是：

- 把“我觉得应该对”变成“测试真正证明它对”

为什么这一步特别重要

- AI 很容易快速生成实现
- 但也很容易快速生成错的实现
- 如果没有测试兜底，就会进入：
  - 看起来写完了
  - 实际上并不知道对不对

所以这一步是在给后面的实现加护栏。

第七阶段：出问题时，走完整条 bug workflow

对应文件：`8-bug-workflow.md`

- 测试失败了
- demo 不对了
- 行为和预期不一致了
- 修完之后还不确定能不能宣称完成

这时最危险的做法是：

- 一边猜原因，一边乱修
- 最后再凭感觉说“应该好了”

这份合并后的 prompt 要求一次走完整条闭环：

- 现象
- 证据
- 假设

为什么把调查、调试和验证合成一份更合理

- 因为这三步在真实工程里本来就是一条连续流程
- 如果拆成三份，学生很容易只做前半段，或者只记住“怎么修”
- 合成之后，逻辑会更完整：
  - 先调查
  - 再定因
  - 再修复
  - 最后再验证

这一步真正解决的是什么问题

- 防止没有证据就开始乱猜
- 防止 root cause 还没确定就开始大改
- 防止修完之后没有 fresh evidence 就说“完成了”

一句话说：

- 它把 bug 处理过程变成了一条可以审查的工作流
- 但如果 root cause 还没找准，修复往往只是碰运气

分开之后，流程会更干净：

1. 先调查
2. 再定因
3. 再谈修复

第九阶段：完成前必须 fresh verification

对应文件：`10-verification-before-completion.md`

- “理论上应该好了”不算
- “上次跑过”不算
- “我觉得没问题了”不算

真正能支撑“完成”的，只能是：

- 当前版本
- 当前这次
- 新鲜跑出来的验证证据

这一步为什么经常被忽略

- 因为项目到后面，大家最想听到的是“做完了”
- 但真正负责任的说法应该是：
  - 跑了哪些验证
  - 覆盖了哪些路径
  - 还有哪些没覆盖

这一步解决的是：

- “我们能不能负责任地说项目已经完成”

把这组 prompt 串起来看

- `0` 到 `2`：判断问题、收敛方向
- `3` 到 `4`：把方向变成能开工的工程计划
- `7`：如果项目有界面，再把视觉和交互层的设计方向说清楚
- `6`：把实现建立在测试之上
- `8`：把“调查 -> 定因 -> 修复 -> fresh verification”合成一个完整闭环

这其实已经是一条完整的项目开发链路。

这套方法最适合什么项目

- 课程项目
- hackathon 项目
- 实验室内部小系统
- 研究型原型
- 个人 side project

因为这些项目最容易出现的问题不是“完全不会写代码”

- 而是：
  - 不知道做什么
  - 边界不清
  - 写着写着就乱了

这套方法不该被误解成什么

- 不是让 AI 替你做所有决定
- 不是把开发变成“复制 prompt”
- 不是每轮都必须机械使用

更准确地说：

- 它是一套项目推进脚手架
- 帮你在不同阶段问对问题

这章你该带走什么

- 项目开发可以被拆成一组更清楚的 AI 协作阶段
- 不同 prompt 解决的是不同阶段的问题
- 好的项目推进，不是一路猛写，而是不断澄清、收敛、验证
- 会用 prompt，不等于会开发；但会把 prompt 串成 workflow，才更接近真正的 AI 协作开发

现在就可以开始做的事

- 先拿你自己的一个项目 idea，跑一遍 `0 -> 1 -> 2`
- 再把结果交给 3 和 4
- 如果要做 Web 页面、app 界面或演示原型，再补一轮 7
- 真正开始写代码前，先用 6
- 一旦出问题，用 `8 -> 9`
- 准备交付前，用 `10`

第 9 章收束

- 这一章讲的不是某一个新框架
- 而是一套可以直接拿来带项目的 AI workflow
- 当你把这些 prompt 串起来，AI 就不再只是“回答问题”
- 而开始成为项目开发过程中的协作伙伴