写出能让大模型替你构建软件的提示词。

BY MARC GLOOR · 第 1 页(共 2 页)——九条要义与参考模板

模型能否把一个想法翻译成可运行的代码,几乎完全取决于请求本身的清晰度与结构。需求工程与代码生成方面的近期研究都指向同一个结论:一行的提示词,只能换来一行的软件。下面是一份提炼后的实践清单,来源是经同行评议的研究与一线实践指南,目的只有一个——把需求转化成大语言模型真正能执行的提示词。

九条要义

从需求到代码,从终端到浏览器——将产出从原型级别推向生产级别的实践。

1 先设定角色与上下文

开头先设角色——「请以面向服务架构中的资深后端工程师的身份回答」——并用一段话写明项目名称、约束条件以及必须遵循的标准。一个角色会隐式地引入优先级(安全性、可扩展性、惯用风格),否则模型只能去猜。

2 使用肯定且具体的语言

把「应当」「可以」换成「必须」「需要」。含糊的动词只会换来含糊的代码。直接点名新代码必须遵循的文件、类、方法和现有模式,而不是抽象地描述它们。

> ✗ 「写一个处理用户数据的函数。」
> ✓ 「在 `UserProcessor.js` 中新增 `transformUser`,风格仿照 `transformPaymentData`。」

3 钉死技术环境

明确写出语言及精确版本、框架、允许(和禁止)的库、运行时、目标平台以及编码规范。否则模型会回退到训练数据中占多数的模式,而这往往与你当前的代码库相冲突。

4 把需求写成编号列表

把软件该做什么拆成离散、编号、可测试的条目。相比连续文字,编号列表被处理得更可靠,也逼你在模型之前先面对其中的歧义。功能性、非功能性和接口需求要分组列出。

5 定义输入、输出与边界情况

写明 I/O 格式、前置与后置条件、错误模式,并给出至少两到三个具体示例——其中一个要是边界情况。少样本示例稳定地优于抽象规格,尤其是对数据形态敏感的任务。

6 采用渐进式提示

凡是超出单个函数的工作,都不要让模型一次性写完。带它走一条阶梯式的链路:先细化需求,再推导设计,再写测试,最后写代码。每一步的输出就是下一步的输入。这与瀑布模型类似,并在每个环节都留出评审面。

7 把验证写进提示词里

让模型在实现之前——或者与实现同时——写好测试,然后对照你的验收标准自查结果,列出所做假设和未处理的边界情况。测试是你需求的可执行形式;没有测试,就无法判断代码是否满足需求。

8 按风险匹配严谨度

一次性脚本可以容忍「懒惰」的提示词,支付流程则不行。把投入校准到风险上:低风险的原型可以用宽松、探索式的提示;高风险功能则值得走一遍完整的「先规约,后写码」流程,并把需求、设计与测试沉淀下来。

9 指明界面范式——TUI 还是 GUI

明确写出界面类型并钉住它的约定;否则模型会选默认值,而那些默认值很少匹配你的平台。

> **TUI** 库(textual / curses / prompt_toolkit);按键方案(vi / emacs / 具名);最小宽度;--json 用于管道;默认单色。
> **GUI** 设计系统;状态(空、加载、错误、成功);可访问性(WCAG AA);对比度;键盘可达;在提示词中给出文案。

参考模板 · 一个可复用的提示词骨架

在大多数项目里都站得住的八段结构。

ROLE	你是一名资深 {语言} 工程师。优先级:清晰、可测试、安全。
CONTEXT	项目:{名称,一句话目的}。代码风格:{要遵循的模式与文件}。
STACK	语言 {X.Y}。框架 {X.Y}。允许的库:{…}。禁止的库:{…}。
UI / UX	范式:{TUI | GUI}。 TUI—库 {textual/curses/prompt_toolkit};按键 {vi/emacs/具名};宽度 ≥ {N} 列;--json 用于管道;默认单色。 GUI—设计系统 {…};状态 空/加载/错误/成功;可访问性 {WCAG AA};对比度 ≥ 4.5:1;此处写文案。
REQUIREMENTS	1. {必须} 2. {必须} 3. {禁止} …(编号、肯定、可测试)。
I/O & EDGES	输入:{类型、格式}。输出:{类型、格式}。边界情况:{…}。示例:{…}。
PROCESS	第 1 步—提出函数签名 + 文档字符串;等我批准。 第 2 步—为上述需求写失败的测试。 第 3 步—实现;然后对照需求自查并列出假设。
OUTPUT	每段代码前给出完整文件路径。生产可用,不截断,不留占位符。

避免 **一句话的需求。** **在同一句话里把**「做什么」和「怎么做」混在一起。 **把技术栈** 交给模型自己发明。 **把第一稿当成** 终稿来用。 **没有测试,就没有验收。**

7×

当提示词被拆分为以下流程时,会浮现出更多虽小但关键的修正:**需求 → 设计 → 测试 → 代码** 而不是一次性发出。渐进式提示能在它们*进入*代码库之前就捕获它们。

「我们反对在基于大模型的软件开发中依赖一句话需求的做法。相反,我们强调要投入时间和精力,把高质量、详细的需求清楚地表达出来。」

ARORA ET AL. · REQUIREMENTS ARE ALL YOU NEED (2024)

参考资料 · White et al., *ChatGPT Prompt Patterns for Software Engineering* (arXiv 2303.07839) · Arora et al., *Requirements are All You Need: From Requirements to Code with LLMs* (arXiv 2406.10101) · *Prompt Engineering for Requirements Engineering: A Literature Review & Roadmap* (arXiv 2507.07682) · Ullrich, Koch & Vogelsang, *From Requirements to Code: Developer Practices in LLM-Assisted SE* (arXiv 2507.07548) · *Guidelines to Prompt LLMs for Code Generation* (arXiv 2601.13118) · Mosofsky, *Spec-Then-Code* (GitHub) · Palantir, *Prompt-Engineering Best Practices.*

示例 · 一次端到端的完整构建

构建一个文本编辑器。复制模板。填好花括号。

*下面有两段。**第一段**是第 1 页里那套裸的八段骨架——把它粘进你选择的模型,然后把每个 `{花括号}` 替换成你项目的具体内容。**第二段**是同一份骨架,已经为一个小而现实的目标填好了内容:一个浏览器中运行、单窗口的纯文本编辑器(GUI)。TUI 的替代方案,在裸模板的 UI / UX 段中已有描述。两段都可以当作示范来用,也可以直接复制后改写。*

BLOCK 1　裸模板——复制这一段

替换掉每一个 {花括号}

ROLE	你是一名资深 {语言} 工程师。优先级:清晰、可测试、{优先项}。
CONTEXT	项目:{名称 + 一句话目的}。代码风格:{要遵循的模式与文件}。
STACK	语言 {X.Y}。框架 {X.Y}。允许的库:{…}。禁止的库:{…}。
UI / UX	范式:{TUI | GUI}。 TUI——库 {textual/curses/prompt_toolkit};按键 {vi/emacs/具名};宽度 ≥ {N} 列;--json 用于管道;默认单色。 GUI——设计系统 {…};状态 空/加载/错误/成功;可访问性 {WCAG AA};对比度 ≥ 4.5:1;此处写文案。
REQUIREMENTS	1. {必须}　2. {必须}　3. {禁止}　…(编号、肯定、可测试)。
I/O & EDGES	输入:{类型、格式}。输出:{类型、格式}。边界情况:{…}。示例:{…}。
PROCESS	第 1 步——提出文件结构 + 类型;等待批准。 第 2 步——为需求写失败的测试。 第 3 步——实现;自查;列出假设。
OUTPUT	每段代码前给出完整文件路径。生产可用,不截断,不留占位符。

BLOCK 2　完整示例——一个基于浏览器的文本编辑器

复制并调整

ROLE	你是一名资深 TypeScript / React 工程师。优先级:清晰、可测试、可访问。宁选简单、惯用的写法,而非花哨的抽象。
CONTEXT	项目:**NotePad**——一个在浏览器里运行的单窗口纯文本编辑器。风格:小而聚焦的组件,文本操作用纯函数,不用全局状态。请仿照 `src/components/Toolbar.tsx` 中的写法。
STACK	TypeScript 5.4。React 18.3。Vite 5。CSS Modules。允许的库:仅限 `react` 与 `react-dom`。禁止:Redux、MobX、jQuery、任何富文本库(Quill、Slate、CKEditor)。编辑器必须是受控的 `<textarea>`,而不是 `contenteditable`。
UI / UX	范式:**GUI**(单窗口网页应用)。设计:极简——系统字体,中性配色,除了工具栏不留任何装饰。 状态:空文档→居中占位文字「开始输入,或打开一个文件」;加载→工具栏中显示加载图标;错误→顶部 toast,可关闭;成功(已保存)→「已保存」徽标显示 2 秒。 可访问性(WCAG AA):完整的键盘对等支持,不允许只有鼠标可达的操作;`<textarea>` 须打标签;工具栏按钮带 `aria-label`;可见的焦点环;对比度 ≥ 4.5:1;尊重 `prefers-reduced-motion`。 文案:保存按钮「保存 .txt」;状态栏「字数:{n} · 字符:{n}」;关闭提醒「有未保存的更改,要放弃吗?」
REQUIREMENTS	1. 文档在一个铺满窗口的 `<textarea>` 中显示。 2. 工具栏必须提供:新建、打开(.txt)、保存(下载 .txt)、字数。 3. 打开必须使用 FileReader 读入并替换当前文档。 4. 保存必须触发文档以 UTF-8 纯文本格式下载。 5. 字数必须在每次按键时更新,并显示在状态栏中。 6. 文档必须在每次变更时写入 localStorage,加载时恢复。 7. Ctrl/Cmd+S 必须触发保存;Ctrl/Cmd+O 必须打开文件选择框。 8. 关闭标签页时,若有未保存的修改,应用必须发出警告(beforeunload)。
I/O & EDGES	输入:按键事件;最大 5 MB 的 .txt 文件。 输出:下载得到的 .txt 文件,UTF-8。 边界情况:首次运行时为空文档;从 localStorage 恢复 4 MB 文件;粘贴二进制垃圾(不得崩溃);带空格的文件名。 示例:打开内容为「hello\nworld」的 `notes.txt` → textarea 显示两行,字数 = 2。
PROCESS	第 1 步——提出文件结构(组件、hook、工具函数)以及 AppState 的 TypeScript 类型,等我批准。 第 2 步——编写 Vitest 测试,覆盖:字数函数、保存文件名的推导、localStorage 往返、键盘快捷键处理。 第 3 步——实现;然后对照上面的八条需求自查,列出所有假设。
OUTPUT	每段代码前给出完整文件路径(例如 `src/App.tsx`)。生产可用,不截断,不写占位注释。修改已有文件时,请返回该文件的完整更新版本。

红线 = 裸模板。把它作为起点粘进去,然后替换每个 {花括号}。　　**绿线** = 一个完整填好的实例。同一份骨架,被具体化成一款小产品。

用法

1. 把第 1 段复制到对话里。 **2.** 把每个 `{花括号}` 替换成项目的具体内容——第 2 段展示了能跑通的细节程度。 **3.** 发送提示词,在**第 1 步**停下来,等模型给出文件结构方案之后,再批准第 2、3 步。 **4.** 把结果当草稿看待:复核假设清单,运行测试,然后再迭代。

与第 1 页配套使用 · 这套八段骨架依据的是与第 1 页相同的资料——主要包括 Arora 等(2024)关于渐进式提示的论文、Mosofsky 的 *Spec-Then-Code* 关于「严谨度对应风险」的校准方法,以及 Mastropaolo 等(2026)的经验性指南。「构建一个文本编辑器」只是一个演示目标;同样的结构同样适用于 CLI 工具、微服务或数据管道。