《Codex 蓝皮书：从入门到架构大师》

主理人: Hunk Wu (X: @ai_pmer)

🌐 English PDF Version | 🌐 English Online Version

🧭 目录 (Table of Contents)

第一部分：AI-Native 时代的产品心智与搭建

- Ch.01 告别手写代码：Vibe Coding 时代的产品心智
- Ch.02 跨端掌控：Codex 多端生产力矩阵搭建
- Ch.03 破局云端孤岛：沙盒调试与本地环境深度穿透

第二部分：架构工程与智能体约束

- Ch.04 目标驱动：用“边界与断言”驾驭推理型智能体
- Ch.05 制定 CAP 协议：构建项目专属的 AGENTS.md 规则层
- Ch.06 思维纠偏：如何像技术总监一样透视 CoT 推理链

第三部分：高级多端编排与巡检

- Ch.07 视觉闭环：Desktop Computer Use 自动巡检与设计还原
- Ch.08 移动看护工作流：全天候离线编排实战
- Ch.09 架构复苏：混乱遗留系统的全景解析与渐进式解耦

第四部分：一人公司的商业闭环

- Ch.10 商业实战：2小时跑通 Next.js + Stripe 商业级 MVP
- Ch.11 触角延伸：Expo 跨端原生 App 开发与云端打包
- Ch.12 终局思考：独立开发者如何打造自动化商业飞轮

第五部分：前沿瞭望与版本迁移

- Ch.13 前沿瞭望：2026 Codex 生态全景升级

🔌 关联开源项目

- **飞书助理 (Codex Feishu Sentinel)**：蓝皮书 Ch.08 官方参考工程。专为 Codex 开发者打造的飞书助理，支持日报自动汇总推送、CI 熔断移动端警报与手机端一键审批。
- **Codex Switch (多供应商无缝切换)**：macOS Codex 多供应商一键切换利器。支持 OpenAI 官方 / DeepSeek / Kimi Code 秒级平滑切换，历史会话跨供应商无缝续聊，彻底解决单模型配额耗尽与 Rate Limit 限流难题。

Ch.01 告别手写代码：Vibe Coding 时代的产品心智

> 🎯 ***具体工程麻烦***：*开发者把 AI 当作高级自动补全，手动写样板代码累死累活，或者任由 AI 自由发挥导致架构完全失控。*
> 💡 ***可运行实战代码与落地收益***：*掌握 3 阶段演进心智模型；带走「边界与验收标准 PRD 控制模板」，彻底告别低效打字。*
> ⚡ ***社交传播 / 截图金句***：*“人类不应该再手写一行无意义的模板代码。你的手应该用来握紧方向盘，而不是推手推车。”*

在“实战产品说”与很多读者交流时，我发现大家最容易陷入一个思维陷阱：拼命去背各种 AI 编程工具的快捷键和底层语法，像背 API 手册一样去学习 AI 编程。

醒醒吧！在最新的 AI 智能体时代（以 OpenAI Codex 和 GPT-5.6 世代为标志），手写代码本身的门槛已经被拉低到几乎为零。这就是所谓的 **“Vibe Coding”（氛围感编程）** 时代——你只需要专注在产品的核心业务逻辑、商业闭环与用户真实体验上，剩下的工程苦活与脏活全部交给智能体。

本章将帮你重构心智，理清从传统的“代码打字员”向“AI 编排架构师”跃升的底层逻辑。

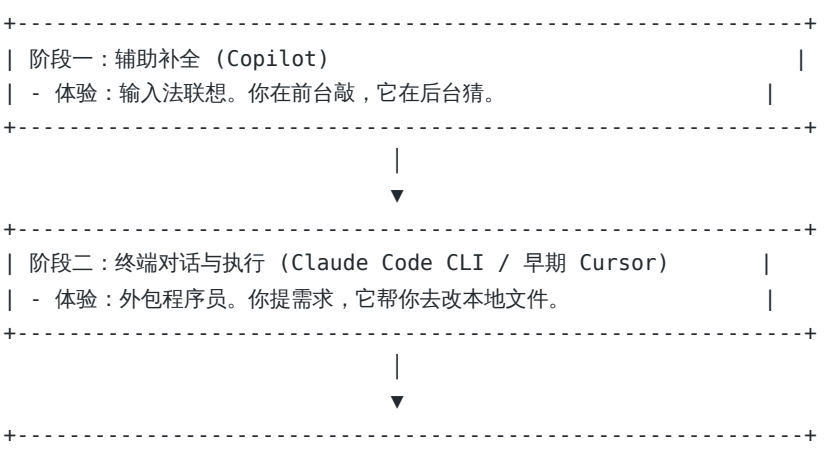
🎯 生活化直觉隐喻：告别砖瓦工，做工地总建筑师

如果你第一次接触 AI 编程，不要把它想成一个冷冰冰的代码生成器。用生活中的场景来理解：

```
【传统手写代码】 ──> 你自己当泥瓦匠，一砖一瓦搬砖砌墙，稍不留神墙角砌歪了，整栋楼推倒重来。
【传统 AI 补全】 ──> 你在前面砌砖，AI 在后面帮你递砖头（你敲一行，它猜半行，手脚依然停不下来）。
【Codex 编排时代】──> 你是「总承包建筑师」，Codex 是「高素质施工队」：
                        - 你画蓝图（定义目标 Goal）
                        - 你立施工安全围挡（设定沙盒与红线 Constraints）
                        - 你用水平仪验收房子（自动化测试 Validation）
                        - 施工队（Codex）自己去算力学结构、买水泥、砌墙、抹灰。
```

当心态从“我怎么敲出这段逻辑”变成“我怎么向施工队下发一份不会跑偏的施工单”时，你才算真正跨入了 AI 原生研发的大门。

1.1 编程浪潮的演进：我们在哪里？



```
| 阶段三：自主多端协同智能体 (OpenAI Codex 生态)                    |
| - 体验：数字化全栈研发团队。目标驱动、沙盒运行、跨端协同。        |
+-------------------------------------------------------------+
```

很多人仍把时间消耗在第一或第二阶段，但 2026 年的 Codex 生态已经确立了第三阶段标准：

- **跨端生产力矩阵**：本地 CLI 自动化执行，ChatGPT 桌面端（代码模式）负责复杂审查与 Sites 原位预览，手机端全天候离线看护与 Guardian 审批，辅以 Desktop Computer Use 自主操作浏览器完成视觉走查。
- **强推理与前沿智能体内核**：从 2026 年 9 月发布的最新前沿旗舰 **GPT-6 Astra**（具备 105 万超大上下文与原生 Computer Use 计算机操作能力）到成熟的 **GPT-5.6 家族（Sol / Terra / Luna）**，智能体具备了超长链深度规划、跨应用调度与视觉自愈能力，不再仅仅停留在样板代码修补。

1.2 核心心智转变：从“过程控制”到“边界控制”

做产品经理（PM）最深的心得往往是：给研发下需求，最忌讳说“第一步点这个按钮，第二步去数据库查那张表”——那是典型外行指导内行。优秀的做法是给出标准 **PRD（产品需求文档）**：讲清业务目标、输入输出规格、性能边界与验收测试用例。

指挥 Codex 完全同理。过去你用 Prompt 教 AI 怎么写 for 循环（过程驱动）；现在你要切换为架构师角色，只抓好三件事：

1. **定义终态 (Goal)**：这个功能要替终端用户解决什么具体问题？
2. **确立红线 (Constraints)**：哪些文件不可修改？安全密钥如何隔离？沙盒等级如何圈定？
3. **自动化验证 (Validation)**：提供可运行的测试命令或断言，让 Codex 自主运行并证明其产出的可靠性。

> *[!IMPORTANT] **主理人硬核公式***
> `稳定交付 = 明确的终态 (Goal) + 严苛的红线 (Constraints) + 自动化的断言 (Validation)`

🚀 新手极速上手 3 步走（无痛起步）

刚进入项目的新手，切忌一上来就给 AI 丢一段长达千字的模糊想法。按以下 3 步执行最简单的闭环：

1. **步骤一：起草 50 字极简需求卡**
   明确三要素：做哪个页面（目标）、基于现有哪个框架（约束）、用什么命令验证成功（断言）。
2. **步骤二：给 Codex 划定安全活动范围**
   在终端中使用沙盒模式执行（如 `codex exec --sandbox workspace-write`），禁止它越权访问外部系统。
3. **步骤三：让 AI 运行测试自证清白**
   要求 Codex：“修改完成后自行运行 `npm run build`，直到终端退出码为 0 再交付”。

🛡️ 翻车自救与避坑速查表

常见踩坑现象	致命原因	翻车自救锦囊
把 AI 当百度搜索问概念	对话几小时全是文字理论，没有落地一行代码	立即终止闲聊，直接给出一个目标：“在 `src/` 下创建登录组件并跑通测试”
一次性喂入上万行无关代码	上下文被噪音撑爆，AI 发生注意力涣散并胡编滥造	明确指出关心的 1-2 个具体文件路径，让 AI 局部精准重构
不看测试只信 AI 嘴上说“已完成”	AI 以为自己改对了，实际项目根本跑不起来	在 Prompt 强制追加验收断言：“修改后必须在终端执行构建验证，输出通过日志”

1.3 你的新护城河：编排力与商业认知

如果手写语法不再是技术壁垒，那么独立开发者与技术产品经理的真正护城河是什么？

在 pmer.cn 上我反复强调：**壁垒在于你对业务边界的定义，以及你对用户同理心的理解。**

你不再需要花三天去调试一个 CSS 弹性盒居中，或者翻遍 StackOverflow 解决打包构建报错；你的核心任务，是运用 Codex 这套多端协同矩阵，快速组装出可落地的商业 MVP，把产品推向真实市场去验证用户付费意愿。

在接下来的章节中，我们将正式开启这套“跨端编排”的落地之旅。

Ch.02 跨端掌控：Codex 多端生产力矩阵搭建

> 🎯 ***具体工程麻烦***：*新手急着和 AI 对话写代码，本地 Node 依赖错乱、权限没开对、客户端版本过旧，导致终端疯狂报错、白白烧掉几万 Token。*
> 💡 ***可运行实战代码与落地收益***：*提供基于 CLI 0.14x 的极速安装方案，ChatGPT 桌面端代码模式（Code Mode）与 Computer Use 权限核验命令，多端联调 checklist。*
> ⚡ ***社交传播 / 截图金句***：*“AI 编程第一步，先配稳你的指挥舱。别让本地依赖报错吃光你的宝贵 Token。”*

工欲善其事，必先利其器。在“实战产品说”中，我常强调一个原则：**AI Native 开发的第一步，是把你的“指挥舱”配置得足够稳定。** 很多新手急着去跟 AI 聊天写代码，结果因为本地环境不匹配、权限没开对，导致 AI 在终端疯狂报错、浪费 Token。

本章将带你一步步配置 Codex 的多端生产力矩阵，包括 CLI 0.14x 客户端、ChatGPT 桌面端代码模式以及 ChatGPT 移动端的联动桥接。

🎯 生活化直觉隐喻：打造你的“三端作战指挥舱”

不要把各个端当成孤立的软件。想象你正在指挥一支太空探险队：

```
【Codex CLI】              —> 主机舱的「动力引擎」：挂载在终端底层，负责高并发、纯代码批处理、跑测试与 CI 构建。
【ChatGPT 桌面端代码模式】—> 舰桥的「全景战术大屏」：多仓库 Diff 审查、Sites 页面原位预览、Computer Use 视觉走查。
【手机 ChatGPT App】      —> 舰长的「随身呼机」：离开工位时监控构建状态，高危操作手机一键批准（Guardian 自动分流）。
```

三端各司其职，你就不必死守在工位前按键盘，实现全天候随时随地编排交付。

🚀 新手极速上手 3 步走（无痛起步）

如果你是第一天上手，不需要做复杂的网络穿透或高级脚本配置，按以下 3 步即可 5 分钟跑通首个任务：

1. **步骤一：安装 CLI 并完成账号鉴权**
   确保本地已安装 Node.js 20+，在终端运行：

   ```
   npm install -g @openai/codex@latest
   codex
   ```

   浏览器将自动弹出 ChatGPT 登录窗口，登录你的账号完成快速绑定。
2. **步骤二：写入极简防爆配置文件**
   在终端执行以下命令，创建 `~/.codex/config.toml`，绑定 GPT-5.6 Terra 主力模型并防止上下文膨胀：

   ```
   mkdir -p ~/.codex
   cat << 'EOF' > ~/.codex/config.toml
   model = "gpt-5.6-terra"
   tool_output_token_limit = 12000
   model_auto_compact_token_limit = 64000

   # 前沿高阶架构与多端视觉走查可切换至最新旗舰
   # model = "gpt-6-astra"
   ```

```
[profiles.guardian]
model = "gpt-5.6-luna"
EOF
```

3. **步骤三：用安全沙盒模式跑通第一个任务**
   进入你的项目根目录，运行：

```
codex exec --sandbox workspace-write "检查当前目录代码规范并输出修复建议"
```

2.1 CLI 客户端（0.14x 时代）安装与避坑

Codex CLI 核心由高效的 Rust 编写（codex-rs），但 OpenAI 通过 npm 提供了官方封装分发，**作为用户你完全不需要安装 Rust 工具链**，只需满足 Node.js 20+ 环境。

1. 基础环境检查与安装

在终端运行：

```
# 1. 检查 Node.js 版本（要求 20+）
node --version

# 2. 全局安装最新稳定版 CLI (0.147.0+)
npm install -g @openai/codex@latest

# 或使用官方一键脚本（macOS / Linux）
curl -fsSL https://chatgpt.com/codex/install.sh | sh
```

Windows 用户可通过 PowerShell 安装：

```
powershell -ExecutionPolicy ByPass -c "irm https://chatgpt.com/codex/install.ps1 | iex"
```

安装完成后验证版本：

```
codex --version
```

2. 认证方式选择：ChatGPT 账号 vs API Key

Codex CLI 提供两种认证方式，**官方默认推荐 ChatGPT 账号登录**：

- **方式 A（推荐 / 默认）**：直接在终端运行 `codex`，浏览器会自动弹出 ChatGPT 登录页面。**ChatGPT Plus（$20/月）、Pro、Team、Business、Edu、Enterprise 套餐都已经包含 Codex 用量额度**，对独立开发者是性价比最高的选择。
- **方式 B（按量计费 / CI 场景）**：使用 OpenAI API Key。适合 CI/CD 自动化流水线或没有浏览器交互的服务器。

```
# 在你的 ~/.zshrc 或 ~/.bashrc 中写入
export OPENAI_API_KEY="sk-proj-xxxxxx..."
```

3. 账单熔断防爆配置

很多新手最担心的就是 AI 暴走刷爆信用卡。Codex 在 0.14x 中提供了多道保险防线：

1. **OpenAI 后台硬上限（最关键）**：在 OpenAI 开发者后台为该 API Key 设置月度 Usage Limit 硬限制（新手建议设为 $50）。即使 AI 陷入异常死循环，也能确保你的资金万无一失。
2. **`config.toml` 中的 Token 控制**：在 `~/.codex/config.toml` 写入输出上限与主动压缩阈值（如前述配置）。
3. **沙盒自治与 Guardian 自动审批**：废弃旧版 `--full-auto`，全面改用 `--sandbox workspace-write` 与 `--approve-for-me`（由 GPT-5.6 Luna 在后台基于策略自动审查，阻断高危系统操作）。

4. 多供应商无缝切换：突破单模型配额与限流（推荐开源项目 Codex Switch）

独立开发者在日常高强度全栈开发中，极易撞上 OpenAI 账号的 3 小时用量限额或 API Rate Limit 报错。频繁手动改环境变量或切换工作区不仅打断心流，更致命的是**会导致正在进行中的长会话上下文直接丢失**。

针对这一高频刚需，推荐使用配套开源利器 **Codex Switch**（由本书主理人打造）：

- **秒级多供应商一键切换**：在 macOS 上一键平滑切换 OpenAI 官方、DeepSeek、Kimi Code 等主流模型供应商，突破单一平台的速率瓶颈。
- **跨供应商无缝续聊**：攻克了第三方模型推理字段差异、本地加密校验与 `thread_history.sqlite` 数据库字节偏移等复杂校验机制，切换模型供应商后仍能**无缝继承当前历史会话上下文**，无需从零重新交代项目背景。
- **开箱即用快速上手**：

```
# 克隆并安装 Codex Switch
git clone https://github.com/aipmer/codex-switch.git
cd codex-switch && ./install.sh

# 一键切换当前供应商至 DeepSeek
codex-switch --to deepseek

# 查看当前绑定的供应商状态
codex-switch --status
```

2.2 桌面端：ChatGPT 桌面客户端「Codex 代码模式」

> ⚠️ ***重要生态演进**：2026 年下半年，原独立的 Codex 桌面应用正式退役，所有桌面端能力已完全并入 **ChatGPT 桌面客户端**。*

1. 安装与进入代码模式

```
# macOS：直接下载 ChatGPT.dmg，拖入 Applications
# Windows：使用 winget 一键安装
winget install OpenAI.ChatGPT
```

启动 ChatGPT 客户端并登录后，点击左侧边栏的 **Codex** 标签页，即进入「Codex 代码模式（Code Mode）」。

新增核心生产力特性：

- **Sites 原位托管**：在客户端内一键创建、部署和预览 Web 项目，配合 Annotations 批注实现“指着界面改代码”。
- **多仓库审查（Multi-repo Review）**：大型微服务或多包工程可在单一视图下统一审查 Diff。
- **内置浏览器升级**：直接检索浏览历史，并可引用当前 Chrome 打开的标签页进行上下文分析。

2. Computer Use 视觉测试权限配置

让 AI 自主操作桌面屏幕（Computer Use）**目前主要支持 macOS**，需在 macOS「系统设置 → 隐私与安全性」中授予 ChatGPT 两项核心权限：

```
[macOS 系统设置] -> [隐私与安全性]
  ├─ 辅助功能 (Accessibility)  ──> 勾选 [ChatGPT] (允许模拟鼠标点击/按键)
  └─ 屏幕录制 (Screen Recording) ──> 勾选 [ChatGPT] (允许截屏并由 Vision 模型分析)
```

安全边界控制：

- 首次操作某个第三方应用（如 Google Chrome）时，会弹窗询问你是否授权该应用。
- ChatGPT 无法自动操作终端本身、自身界面或系统级管理员授权弹窗（例如 `sudo` 提权输入密码）。

2.3 手机端（ChatGPT App）全天候看护桥接

人在户外或离岗时，无需死守工位：

1. **官方原生任务同步**：当你使用 ChatGPT 账号登录 Codex 时，终端与云端的任务都会实时同步到手机 ChatGPT App 中。你可以掏出手机查看长任务的执行进度、随时发送补充指令。
2. **Guardian 审批与 Webhook 提醒**：结合 Ch.08 的移动看护网关，当遇到需要人工确认的临界高危操作（如线上部署、数据库变更）时，通知将精准推送到飞书或微信，手机端回复数字即可一键放行或驳回。

🛡️ 翻车自救与避坑速查表

常见报错 / 翻车现象	核心原因	极速自救指南
`error: unknown option '--full-auto'`	CLI 升级到 0.14x 后废弃了旧参数	替换为新沙盒参数：`--sandbox workspace-write`
`SyntaxError: Unexpected token ...` (Node 报错)	本地 Node.js 版本低于 20	运行 `node -v` 检查，使用 `nvm use 20` 或 `nvm install 20` 升级 Node
`Permission denied: Screen Recording`	macOS 未开启屏幕录制权限	打开「系统设置 → 隐私与安全性 → 屏幕录制」，将 ChatGPT / Terminal 勾选开启并重启客户端
`Model not found: gpt-5.4`	旧模型已正式退役下线	检查 `~/.codex/config.toml`，将 `model` 升级为 `gpt-5.6-terra`

Ch.03 破局云端孤岛：沙盒调试与本地环境深度穿透

> 🎯 ***具体工程麻烦***：*AI 在隔离沙盒跑测试时连不上本地 Docker 的 PostgreSQL/Redis，报错 `Connection refused`，智能体抓瞎。*
> 💡 ***可运行实战代码与落地收益***：*提供基于 CLI 0.14x 沙盒级别解析、SSH / Ngrok 端口反向映射脚本与 3 步网络排错流程。*
> ⚡ ***社交传播 / 截图金句***：*“AI 报错连不上 localhost？不是代码写错了，是你没给隔离沙盒开网络通道。”*

使用 Codex 运行数据库测试时，新手最常碰到的灵异报错是：明明本地 Docker 里的 PostgreSQL 跑得好好的，AI 却在终端狂喊 `Connection refused to localhost:5432`。

这就是 **沙盒隔离（Sandbox Isolation）** 带来的典型壁垒。本章教你如何穿透这层隔离屏障，打通沙盒与宿主机的安全通道。

🎯 生活化直觉隐喻：防爆隔离仓与专用生命导管

把 Codex 的执行环境想象成医院里的「高等级无菌防爆隔离仓」：

【你的宿主机 Mac/PC】 ──> 外部大世界：装着你真实的本地数据库、私钥配置、个人微信和文件系统。
【Codex 运行沙盒】 ──> 无菌隔离仓：AI 所有的代码编译、文件生成都在这个密闭盒子里运行，就算把代码写崩了，也绝不会删掉你的系统文件。
【端口穿透与反向隧道】──> 密封手套与气体导管：当隔离仓里的 AI 需要连外界的本地数据库时，你必须主动在舱壁接一根“专用导管”（把端口转接到仓内）。

如果不接这根导管，AI 在隔离仓里自言自语喊 `localhost`，找到的只是空无一人的仓壁，自然就会报错 `Connection refused`。

🚀 新手极速上手 3 步走（无痛起步）

如果你需要让沙盒连上本地数据库，按以下 3 步建立通道：

1. **步骤一：确认本地服务端口正处于监听状态**
   在终端确认 Docker 或本地服务正常运行（如 Postgres 在 5432 端口）：

   ```
   lsof -i :5432  # 查看端口是否有进程正在监听
   ```

2. **步骤二：启动极简端口穿透（使用 ngrok 或 SSH）**
   运行一条命令为 5432 端口打通公网安全临时通道：

   ```
   ngrok tcp 5432
   # 获得转发地址，形如：tcp://0.tcp.ngrok.io:12345
   ```

3. **步骤三：将穿透连接串注入 Codex 沙盒执行任务**
   在执行时将地址作为环境变量传入，让 AI 执行连接测试：

   ```
   export DATABASE_URL="postgresql://postgres:password@0.tcp.ngrok.io:12345/dev_db"
   codex exec --sandbox workspace-write "运行数据库连通性自测脚本 npm run test:db"
   ```

3.1 沙盒网络壁垒：为什么 `localhost` 连不上？

默认状态下，沙盒与本地宿主机处于安全网络阻断状态：

```
+-----------------------------------+                  +-----------------------------------+
|      云端沙盒 (Sandbox)      |                  |     宿主机 (Local Mac)      |
| - App Code                 |    (隔离屏障)    | - Docker container         |
| - localhost:5432 (空无一人) ---[ X ]----------> | - PostgreSQL (port: 5432) |
+-----------------------------------+                  +-----------------------------------+
```

在 2026 年的 CLI 0.14x 中，Codex 强化了沙盒边界规范：

- `--sandbox read-only`：只读模式，智能体只分析代码，不具备写权限。
- `--sandbox workspace-write`（推荐常用）：智能体可以修改当前项目目录，但不可随意跨出目录或篡改全局系统。

要让沙盒内的代码读写宿主机上的数据库或微服务，必须进行**端口映射与反向隧道穿透 (Reverse Tunneling)**。

3.2 实战：使用 SSH / Ngrok 建立反向隧道

方法一：使用 SSH 反向端口转发（推荐长效方案）

如果你有公网 VPS 服务器，SSH 反向转发是最安全且免费的方案：

```
# 将本地的 5432 (Postgres) 转发到公网中转机的 54320 端口
ssh -R 54320:localhost:5432 user@your-public-vps.com -N
```

接着在环境中配置：

```
export DATABASE_URL="postgresql://postgres:password@your-public-vps.com:54320/dev_db"
```

方法二：使用 Ngrok 极速穿透（零服务器方案）

宿主机终端直接运行：

```
ngrok tcp 5432
```

终端输出类似：`Forwarding tcp://0.tcp.ngrok.io:12345 -> localhost:5432`。

将该临时地址写入环境变量或注入 Codex 会话即可。

3.3 目录映射与环境变量同步规范

1. 密钥安全硬隔离

严禁让 AI 自动把包含真实生产密钥的 `.env` 提交到仓库或云端。在 `.gitignore` 与项目 AGENTS.md 中添加硬约束：

```
## 🛑 Hard Constraints
- Never sync, copy, or commit files matching *.env.
- Always use `src/config.ts` or environment variables for secret injection.
```

2. 沙盒临时缓存清理

为防止 node_modules 缓存或 build 缓存导致的幽灵构建故障，在 AGENTS.md 中写入标准化清理脚本：

```
## 💻 Developer Commands
- **Clean Run**: `rm -rf node_modules/.cache .next/cache && npm run test`
```

🛡️ 翻车自救与避坑速查表

常见报错 / 翻车现象	致命原因	极速排查与自救指南
`Connection refused to 127.0.0.1:5432`	沙盒以为本地有数据库，实际隔离舱内空无一物	检查反向穿透隧道是否开启，严禁用 `localhost`，必须使用穿透地址
`ngrok session expired / reconnecting`	免费版 ngrok 隧道中断或地址重置	重启 ngrok 复制新地址，或改用项目自带的 `scripts/codex-watchdog` 反向穿透工具
`EACCES: permission denied`	沙盒权限受限，试图写入项目外的受保护系统目录	检查命令是否使用了正确的 `--sandbox workspace-write`，并约束操作在项目根目录内

Ch.04 目标驱动：用“边界与断言”驾驭推理型智能体

> 🎯 ***具体工程麻烦***：*写太长 Prompt 像教大厨切土豆丝导致模型受限；一句话太模糊又导致 AI 胡编乱造改出 10 个 Bug。*
> 💡 ***可运行实战代码与落地收益***：*目标驱动 Specs 标准模板（目标 + 前提 + 验收断言 + 禁区红线）；传统 Prompt 与 Specs 真实测试对比。*
> ⚡ ***社交传播 / 截图金句***：*“教大厨切土豆丝只会把菜炒糊。给 AI 下指令，定死输入输出与验收断言才是专业做法。”*

在以 **GPT-5.6 Terra** 等强推理模型为核心的 Codex 时代，传统流水账式的 Prompt 技巧已经过时。推理模型具备内生规划（Internal Planning）空间，死板的过程式指令反而会捆绑模型的思考手脚。

本章教你如何用专业产品 Specs 的方式，精准指挥 Codex。

🎯 生活化直觉隐喻：米其林主厨与点餐下单单

很多人给 AI 派任务，就像在米其林餐厅吃饭时，冲进后厨对大厨指手画脚：

【保姆式命令（过程驱动）】 —> ❌ “大厨，你左手拿刀，把土豆切成 2mm 细丝，锅烧到 180 度倒 15ml 油，翻炒 3 分钟，最后放 3g 盐。”

（大厨觉得你在侮辱他的厨艺，而且一旦煤气火候微变，菜必糊无疑。）

【架构师下单（目标驱动）】 —> ✅ “大厨，我要一份香煎土豆做牛排配菜。

- 目标（Goal）：口感香脆，作为晚宴配菜；
- 红线（Constraints）：热量低于 200 卡，严禁使用黄油与花生油（客人过敏）；
- 验收（Validation）：15 分钟内出菜，中心温度达到 75℃。”

Codex 就是精通算法和架构的“米其林主厨”。你要做的是告诉它吃什么、什么不能碰、怎么算合格，具体的刀工和火候完全交给它自己规划。

🚀 新手极速上手 3 步走（无痛起步）

起草你的第一份目标驱动 Specs：

1. **步骤一：定义终态 (Goal)**
   用一句话说清最终交付物，如：“在 `/api/checkout` 实现 Stripe 支付会话创建接口”。
2. **步骤二：划定绝对红线 (Constraints)**
   写明 2-3 条不能触碰的规则，如：“严禁明文打印 Secret Key，禁止修改全局中间件”。
3. **步骤三：提供自动校验断言 (Validation)**
   给出一个可运行的命令，如：“运行 `npm test -- checkout.test.ts` 必须一次性全绿通过”。

4.1 目标驱动的 Markdown 结构规范

在终端或 ChatGPT 桌面端中向 Codex 发起任务时，建议使用标准 Markdown 结构：

```
# 🎯 Goal
[描述希望达到的最终业务状态。例如：实现一个支持 GitHub OAuth 登录并能持久化存储偏好的路由。]

# 🛑 Constraints (红线约束)
- [安全约束：绝对不能将密钥明文写入代码或日志中。]
- [选型约束：必须沿用现有 Tailwind，禁止额外引入新的 UI 库。]
- [防腐约束：严禁改动 src/legacy 目录下的任何历史文件。]

# 🧪 Validation Specs (验收断言)
- [自动化测试：运行 npm run test:unit 测试套件必须 100% 通过。]
- [边界异常：当输入字段缺失时，接口必须返回 400 Bad Request 并附带标准 JSON 错误说明。]
```

4.2 真实案例对比：传统 Prompt vs 目标驱动 Specs

假设我们要写一个 **“带 Redis 限流的 API 代理服务”**。

❌ 传统过程驱动型 Prompt

> *“请帮我用 Express 写一个 API 代理。首先引入 express 和 express-rate-limit。然后配置 rate-limit，设置 windowMs 为 15 分钟，max 为 100 次。接着写一个路由 `/api/proxy`，使用 axios 请求第三方 API `https://api.github.com`。如果请求成功就返回数据，如果失败就返回 500 错误。注意在请求头里带上 Authorization Bearer Token。”*

✅ 目标驱动 Specs

```
# 🎯 Goal
实现一个 Express API 代理路由，安全代理所有发往 GitHub API 的请求。

# 🛑 Constraints
- 必须使用 Redis 作为限流器的数据源，禁止使用内存型限流，以支持多容器横向扩展。
- 代理请求的超时时间（Timeout）必须硬性限制在 3000ms 以内，防止挂起 Node 事件循环。
- 严禁将 GitHub Token 写入代码或日志，必须从 `process.env.GH_TOKEN` 安全读取。

# 🧪 Validation Specs
- 压测模拟：1 分钟内发送超过 60 次请求时，代理必须返回 429 Too Many Requests。
- 容错断言：当上游发生超时或网络断开时，必须返回 504 Gateway Timeout 并包含结构化 JSON。
```

4.3 GPT-5.6 Terra 对 Specs 的推理链路分析

当你把这套规范交给 Codex 后，其内部推理过程会这样高效拆解：

```
[解析 Specs 目标] ──> 分析 Constraints (Redis限流 / 3s超时 / 密钥隔离)
                                  │
                                  ▼
[自主选型与架构规划] ──> 引入 ioredis + rate-limit-redis，配置 Axios 拦截器与超时断路器
                                  │
                                  ▼
[对齐 Validation Specs 编写断言自测] ──> 模拟 429 频控场景 + 模拟超时场景并验证
                                  │
                                  ▼
[自愈代码直至测试 100% 通过] ──> 提交最终干净的代码与测试报告
```

把逻辑规划权让渡给 AI，把验收标准牢牢攥在自己手里。 这就是 AI 时代最高效的人机协同法则。

🛡️ 翻车自救与避坑速查表

常见踩坑现象	致命原因	极速自救指南
步骤写太碎，AI 遇到异常直接卡死	过程命令剥夺了 GPT-5.6 的自愈规划空间	删掉具体实现步骤，只保留目标、红线与测试断言，放手让 AI 规划
AI 改完后产生连锁反应改坏其他模块	没有设定明确的代码防腐与修改范围 Constraints	在 Specs 中严厉声明红线：“仅允许修改 `src/modules/auth/`，严禁碰触其他目录”
AI 以为写完了，其实逻辑存在致命破绽	缺少自动执行的断言（Validation Specs）	要求它在终端自行运行 `npm run test` 或编写单元测试覆盖边界条件

Ch.05 制定 CAP 协议：构建项目专属的 AGENTS.md 规则层

> 🎯 ***具体工程麻烦***：*AI 修好 1 个 Bug 带出 3 个新 Bug；在同一报错上连续重试 10 次烧钱自旋；擅自安装杂乱外部包或手写假 TODO。*
> 💡 ***可运行实战代码与落地收益***：*生产级 `AGENTS.md` 规则层模板；失败 2 次强制熔断机制；禁止假代码占位与依赖失控红线。*
> ⚡ ***社交传播 / 截图金句***：*“没有规则约束的 AI 就是脱缰野马。一份 AGENTS.md，让智能体像十年老架构师一样遵守工程规范。”*

在实际做产品的过程中，最怕遇到的一种开发情况是：修好了一个 Bug，却顺手带出了三个新 Bug；或者新写了一个功能，结果把团队约定的代码风格破坏得一塌糊涂。

在人机协同开发中，Codex 如果没有边界约束，也会变成一个“破坏性极强”的勤奋员工。为了给它戴上缰绳，我们需要在项目根目录下建立 ** `AGENTS.md` **。

这就是我们的 **“智能体协作宪法” (Codex Collaboration Protocol, CAP)**。

🎯 生活化直觉隐喻：新员工入职交接手册与安全守则

把 Codex 想象成刚刚入职你公司的“天才实习生”：

```
【没给 AGENTS.md】 ──> 天才实习生第一天上班，没人告诉他规矩，他为了修一个前台样式，
                       顺手把你运行了三年的核心认证模块全重构成他喜欢的非标写法，上线直接炸库。
【配好 AGENTS.md】 ──> 天才实习生一进门，桌上放着打印好的《团队入职守则》：
                       - “这是 Next.js 15 项目，用 Tailwind”（明确技术栈）
                       - “编译运行 npm run build，别瞎改命令”（规范标准动作）
                       - “绝对禁止动 auth/ 鉴权目录，违者开除”（划定绝对红线）
                       实习生瞬间心领神会，效率翻倍且绝不闯祸。
```

`AGENTS.md` 就是这样一份免面试、零沟通成本的“AI 入职交接手册”。

🚀 新手极速上手 3 步走（无痛起步）

用 3 分钟为你的项目注入第一份智能体规约：

1. **步骤一：在项目根目录创建文件**
   在终端运行：

   ```
   touch AGENTS.md
   ```

2. **步骤二：填入 4 大核心板块极简模板**
   将以下内容存入 `AGENTS.md` ：

   ```
   # 🤖 Codex Collaboration Protocol

   ## 📌 Project Fingerprint
   - Stack: Next.js 15, TypeScript, Tailwind CSS

   ## 💻 Developer Commands
   - Build: `npm run build`
   - Test: `npm run test`

   ## 🛑 Hard Constraints
   - Never update package.json dependencies without human confirmation.
   - Never touch files inside `src/legacy/`.
   - Always run `npm run build` before completing the task.
   ```

3. **步骤三：启动 Codex 验证加载**
   运行 `codex` ，在 TUI 初始日志中确认已自动识别并挂载工作区规则。

5.1 为什么我们需要 `AGENTS.md` ？

`AGENTS.md` 是 OpenAI Codex 官方约定的指令文件名。Codex 在启动时会从当前工作目录开始，**逐级向上扫描**并合并以下文件，作为系统级上下文的一部分：

1. `~/.codex/AGENTS.override.md` （最高优先级，个人覆盖）

2. `~/.codex/AGENTS.md`（全局指令）
3. 项目根目录的 `AGENTS.md`（团队约定）
4. 当前工作目录的 `AGENTS.md`（最具体的局部约定）

它的核心价值在于：

1. **启动即恢复记忆**：每次 Codex 启动时，第一件事就是扫描 `AGENTS.md`，瞬间拾起整个项目的技术栈、文件结构和协作约束，不需要你在对话里反复唠叨。
2. **代码防腐层**：明确定义哪些文件是“只读/禁止触碰”的，引导 Codex 不去擅自重构核心安全模块（如登录鉴权、数据库 Schema）。
3. **约束命令执行**：规定只能使用特定的测试和部署命令，降低 AI 误跑破坏性脚本的概率。

5.2 `AGENTS.md` 的核心四大版块

```
# 项目指纹 (Project Fingerprint)
- 告诉 AI 这是一个什么样的项目，核心技术栈是什么。

# 开发常用命令 (Commands)
- 明确指出编译、测试、迁移数据库的命令，不要让 AI 瞎猜。

# 架构与编码规范 (Styles & Patterns)
- 规定文件存放目录、大小限制及必用的设计模式。

# 智能体安全红线 (Hard Rules)
- 绝对的禁区。一旦触碰，Codex 必须立刻中止并请求人工确认。
```

5.3 实战：一个生产级的 `AGENTS.md` 模板

以下是一个典型的全栈 SaaS 项目的 `AGENTS.md` 规范：

```
# 🤖 Project: Aurora SaaS Core (AGENTS.md)

## 🧬 Project Fingerprint
- **Stack**: Next.js 15 (App Router), TypeScript, Prisma ORM, TailwindCSS.
- **Database**: PostgreSQL on Supabase.
- **Auth**: Next-Auth v5.

## 💻 Developer Commands
- **Install**: `npm install` (Only run if package.json has changed)
- **Dev Server**: `npm run dev`
- **Lint Code**: `npm run lint`
- **Run Tests**: `npm run test`
- **DB Migration**: `npx prisma migrate dev` (Never run in production branch)

## 🎨 Styles & Architecture Patterns
- **Directory Structure**:
  - Components: Keep UI components inside `@/components/ui/` (Shadcn styled).
  - Business Logic: Custom React hooks must go to `@/hooks/`.
  - API Routes: Next.js Route Handlers go to `src/app/api/.../route.ts`.
- **Formatting**:
  - Keep components modular. If a component exceeds 200 lines, decompose it.
  - All API routes must implement Zod schema validation for request body.
  - Return HTTP 400 for validation errors, 500 for internal uncaught errors.
```

```
## 🛑 Agent Boundary & Hard Rules
- **READ-ONLY Directories**:
  - Never modify files inside `src/app/api/auth/[...nextauth]` (OAuth Core).
  - Never alter `prisma/schema.prisma` without explicit human confirmation.
- **PR Rules**:
  - Before declaring a feature complete, run `npm run test` and `npm run lint`.
  - If tests fail, rollback the change immediately and report the error logs.
- **Security Check**:
  - Never commit raw `.env` files or API Keys. Use environment variables.
```

5.4 软约束与硬约束的组合防线

需要特别厘清的一点是：** `AGENTS.md` 里写的规则本质上是“软约束”**——它是注入到模型上下文里的自然语言指令，GPT-5.6 Terra 等模型会**高优先级遵守**，但并不等同于操作系统的只读挂载。

如果你要实现“物理级绝不可越界”的拦截，必须搭配 Codex 0.14x 的三层运行时硬性控制：

1. **沙盒模式 (Sandbox Mode)**：使用 `--sandbox workspace-write` 从 OS 容器层面收紧文件写入范围，阻断越界篡改。
2. **Guardian 审批策略 (Approval Policy)**：使用 `--approve-for-me`，当操作涉及全局依赖安装、系统级文件写出时触发 Guardian 自动评审或中断等待。
3. **Hooks 拦截引擎**：在 `config.toml` 中配置 `[hooks]`（0.14x 转正），在 `session_start` 或执行写入前自动触发安全检查脚本。

```toml
# ~/.codex/config.toml
model = "gpt-5.6-terra"

[hooks]
stop = "npm run lint --silent"
```

“AGENTS.md 业务软规约 + 沙盒与 Hooks 物理硬围栏”，这就是让大团队和新手都能高枕无忧的防腐护城河。

🛡️ 翻车自救与避坑速查表

常见踩坑现象	致命原因	极速排查与自救指南
规则写得密密麻麻，AI 还是偶尔犯规	单个文件字数超标或存在相互矛盾的条目	精简规则，优先写“负向禁止（Never do X）”；关键安全规则升级为沙盒/Hooks 硬拦截
AI 陷入同一 Bug 连续自我修改 5 次死循环	缺少 Anti-Loop 防死锁机制	在 AGENTS.md 写入：“连续尝试修复同一错误超过 2 次必须立即停下，输出根因并等待指令”
AI 擅自安装各种乱七八糟的 npm 依赖	允许其随意执行包管理器命令	在红线中严令：“禁止在未获批准的情况下运行 npm install 添加新 package”

Ch.06 思维纠偏：如何像技术总监一样透视推理过程

> 🎯 ***具体工程麻烦***：*面对强推理模型只能干等；当 AI 第一步假设走偏时，它会顺着错误连续深陷，搞乱代码库。*
> 💡 ***可运行实战代码与落地收益***：*终端 TUI 实时透视 CoT 思考步骤方法；3 种死循环特征识别表；配套 Chrome 扩展实战工程（`examples/ch06-chrome-extension`）。*
> ⚡ ***社交传播 / 截图金句***：*“别让 AI 蒙头狂奔 20 分钟才告诉你走偏了。看懂思考日志，在它跑偏的第一步一键拽回。”*

在传统开发中，管理初级程序员时最让人头疼的情景，莫过于他闷头闭门造车一周，最后交付了一堆与业务方向南辕北辙的代码，甚至把主干分支改崩。

在使用 **GPT-5.6 Terra** 深度推理模型驱动的 Codex 时，虽然 AI 的代码能力极强，但一旦前置假设出错，它就会顺着错误的假设自圆其说、一路狂奔，甚至陷入自我纠错的“无限自旋”。

本章教你如何穿透 Codex 的**推理过程**，在它刚偏离航线时，像一个资深技术总监一样精准介入、一键拽回。

🎯 生活化直觉隐喻：在 AI 脑海里装一个“监考透视窗”

不要把 AI 推理当成一个黑盒魔术：

```
【被动盲等模式】 ──> 就像期末考试，你坐在考场外干等 2 小时，交卷后才发现学生从第一道题就把公式背错了，整张试卷全判零分。
【透视纠偏模式】 ──> 就像你在考场里站在学生身后，看着他面前的「草稿纸」（Reasoning Summary）：
                        - 他刚在草稿纸写下：“假设要重写整个数据库 Schema……”
                        - 你立刻轻轻拍拍他肩膀：“别动 Schema，只改当前的查询索引！”
                        - 学生瞬间在草稿纸划掉错误思路，重新回到正轨。
```

看懂 Reasoning 摘要，就是拿到了这张草稿纸的实时透视权。

🚀 新手极速上手 3 步走（无痛起步）

用 3 步学会像技术总监一样监督并纠正 AI：

1. **步骤一：启动交互式 TUI 并锁定 Reasoning 面板**
   直接在终端运行 `codex`，在分栏界面中实时观察滚动的思维日志。
2. **步骤二：发现假定走偏立即按下 `Ctrl + C`**
   一旦在思考流中看到它打算引入陌生外部依赖或重构核心非目标文件，立即按下 `Ctrl + C` 中断。
3. **步骤三：一句话精准纠偏并恢复会话**
   直接输入修正提示词：“禁止重构已有数据表，改用内存缓存解决”，AI 将清空错误假设重新规划。

6.1 为什么要看模型的推理过程？

强推理模型（如 GPT-5.6 Terra）与传统模型最大的区别在于：它在输出最终代码前，会先在内部进行深度的假设验证与自我模拟。Codex TUI 会在交互界面上**以推理摘要（Reasoning Summary）的形式**实时展示这个过程。

```
[用户需求] ──> 1. 解析目标与限制 ──> 2. 规划步骤 ──> 3. 运行测试 ──> 4. 自我修正 ──> [最终输出]
                  └──(在 TUI 中显示为 Reasoning Summary，即你的“监考视窗”)──┘
```

> ⚠️ ***注意**：完整的思维链（Chain of Thought）为安全防护内部流，TUI 中看到的是模型自主生成的**高保真推理摘要**——这已经完全足够用来判断其技术选型与规划方向。*

6.2 如何在 TUI 中观察并解读推理过程

1. 交互模式（TUI）

直接运行 `codex` 进入 TUI，主面板会实时滚动推理摘要。常用审查与控制指令：

```
# TUI 交互快捷命令
/diff       # 查看当前会话生成的真实代码变动
/review     # 唤起子智能体自动审查最近的代码变更
/copy       # 快速复制最后一条响应代码
```

2. 非交互模式（脚本化分析）

在后台运行长任务时，可通过 JSONL 流进行监控：

```
# 非交互模式，输出结构化 JSONL 流
codex exec --json "重构认证中间件" > task.jsonl
```

实时推理摘要示范

当 Codex 收到“修复 Redis 限流器连接超时”时，Reasoning 面板的健康思考流通常如下：

```
[Reasoning Summary - 正常流]
- User wants to fix Redis rate limiter connection timeout.
- Checking existing implementation in src/lib/redis.ts...
- Found `redis = new Redis()` without retryStrategy.
- If Redis is unreachable, this hangs the Node process, violating the 3000ms SLA in AGENTS.md.
- Action Plan:
  1. Add `maxRetriesPerRequest: null` and explicit connectTimeout: 2000.
  2. Implement custom retryStrategy up to 3 attempts.
  3. Run `npm run test:redis` to verify behavior.
```

6.3 识别 AI 陷入的典型“死循环”

在日常开发中，必须对以下两类死循环保持敏锐：

1. 依赖狂躁循环 (The Dependency Loop)

- **特征**：AI 尝试使用一个未经测试的新库，安装报错后，在推理流中尝试更换 3 个不同的版本或换成另一个更陌生的第三方库。
- **信号**：终端连续出现 `npm install --legacy-peer-deps` 超过 2 次。

2. 补丁打地鼠循环 (The Regression Loop)

- **特征**：修改 A 文件导致测试用例 B 挂掉；它去改 B，结果 C 模块报错；它回头去改 C，A 又坏了。
- **信号**：测试通过率在 80% 和 90% 之间反复横跳，且反复修改同一批文件。

6.4 介入三部曲：打断、修正与接管

第一步：果断打断（`Ctrl + C`）

按下 `Ctrl + C`，立即终止当前任务，阻止 Token 损耗。

第二步：点对点纠偏（直接对话）

打断后直接告诉它盲区所在：

```
你刚才试图引入 axios-retry，但本项目严禁使用第三方 HTTP 重试库。请使用原生的 AbortController 实现超时，重新规划。
```

第三步：人手接管与 Git 回滚

如果代码已经被改乱，使用 Git 撤销并追加规约：

```
git checkout -- src/lib/redis.ts
```

在 AGENTS.md 中追加一条红线：“严禁引入外部重试依赖”。

6.5 章节实操配套：Codex Web Copilot (Chrome 扩展沙盒样例)

为了让读者直观体验如何使用 Codex 思考链引导与 Anti-Loop 护栏进行浏览器插件开发，本项目配套提供了开箱即用的轻量开源样例：

👉 **源码沙盒目录**：examples/ch06-chrome-extension

核心亮点：

1. **纯原生 Manifest V3**：零打包依赖，直接在 Chrome 浏览器中「加载已解压的扩展程序」即可 1 分钟开箱体验；
2. **严格 CSP 护栏**：在 `AGENTS.md` 中严禁内联脚本与 `eval()`，展示 AI 智能体如何在最严苛的浏览器安全沙盒下编写高可用代码；
3. **自动化测试守卫**：执行 `npm test` 自动验证 MV3 规范与脚本语法。

🛡️ 翻车自救与避坑速查表

常见踩坑现象	致命原因	极速排查与自救指南
AI 陷入自我修改死循环停不下来	提示词缺少明确的重试上限阈值	立即 `Ctrl + C` 中断，在对话中追加指令：“已触发重试阈值，停止自旋，输出排查结论”
打断后重新提问，AI 忘记了前面的背景	会话上下文丢失	使用 `codex resume` 恢复原有会话线程，保留先前的思考上下文
AI 改坏了多个历史核心文件	未在修改前做 Git 干净分支隔离	运行 `git checkout .` 快速还原，并重新使用 `--sandbox workspace-write` 约束活动范围

Ch.07 视觉闭环：Desktop Computer Use 自动巡检与设计还原

> 🎯 ***具体工程麻烦**：传统前端 UI 还原依赖肉眼查像素，响应式漏看弹窗遮挡；纯命令行测试无法覆盖真实浏览器渲染与点击。*
> 💡 ***可运行实战代码与落地收益**：ChatGPT 桌面端代码模式操作框配置；Figma 设计稿与本地网页自动截图比对走查流程；UI 视觉巡检真实录屏复现。*
> ⚡ ***社交传播 / 截图金句**：“写完前端还在肉眼查像素？让 AI 自己打开浏览器量尺寸、点按钮，截图标注哪里不合规。”*

在传统的 UI 还原度走查中，最耗费产品经理和前端时间的是“像素眼”校对：

“这个按钮好像往左偏了 4 像素。”
“这个弹窗在移动端尺寸下会被软键盘遮挡。”

在 2026 年的 Codex 生态中，通过 **ChatGPT 桌面客户端（Codex 代码模式）** 与最新前沿模型 **GPT-6 Astra**（OpenAI 专为 Computer Use 打造的原生计算机操作员模型）的结合，智能体不仅能编写代码，还能“动用眼和手”直接操作你的 macOS 桌面，打开浏览器、切换开发者工具分辨率并进行高保真视觉校对。

本章教你如何操纵 Computer Use 自动化完成前端 UI 的设计还原。

🎯 生活化直觉隐喻：24 小时不知疲倦的“像素质检员”

把 Computer Use 想象成你工位旁雇佣的专职体验质检员：

```
【传统人工走查】 ──> 你一手拿 Figma 设计稿，一手拿手机或切换 Chrome 标签页，
                     肉眼眯着看字体大小对不对，手动缩放窗口查断点，改完代码再手动刷一次页面（低效枯燥）。
```

```
【Computer Use】 ──> 你给质检员下发任务卡：“对照设计稿，走查 /auth/login”。
                     质检员戴上防蓝光眼镜（截屏分析），右手握住鼠标（模拟点击），
                     量出距离相差 16px，直接在编辑器改好 Tailwind 类名，重新刷页面截屏交差。
```

你不必再当肉眼人肉比对机，把精力完全释放到业务逻辑交互设计上。

🚀 新手极速上手 3 步走（无痛起步）

用 3 步跑通你的首次 AI 视觉走查：

1. **步骤一：确认 ChatGPT 桌面端权限已授予**
   在 macOS「系统设置 → 隐私与安全性」中，确认已勾选 **ChatGPT** 的「辅助功能」与「屏幕录制」权限。
2. **步骤二：在本地启动待测前端服务**
   在终端启动项目开发服务器（确保能在浏览器打开）：

   ```
   npm run dev
   # 确认 http://localhost:3000 可访问
   ```

3. **步骤三：在代码模式中唤起 `@Chrome` 发送走查任务**
   在 ChatGPT 桌面端 Codex 模式下输入：

   ```
   @Chrome 请打开 http://localhost:3000 登录页，截取主卡片区域，检查是否有超出视口的横向溢出，并调整 padding。
   ```

7.1 安全第一：沙盒边界与屏幕操作白名单

让 AI 操作你的物理屏幕必须建立严密的安全围栏。为防止 AI 因误识别误点个人隐私聊天软件或改动系统文件，必须建立**应用层面的白名单控制**。

1. GUI 权限白名单

在 ChatGPT 桌面客户端中：

- 首次使用 Computer Use 操作某个特定应用（如 Google Chrome）时，会触发系统弹窗询问；
- 可选择 **“Just this once”**（仅本次允许）或 **“Always allow”**（永久放行）；
- 严禁将非必要应用（如微信、邮件、终端本身）加入白名单；
- **出厂硬限制**：Codex 无法自动操作终端、客户端自身或系统级管理员提权密码弹窗（`sudo`），这是操作系统级的硬隔离。

2. 坐标定位与视觉识别机制

Computer Use 遵循高可靠的感知闭环：

```
[全屏/视口截屏] ──> [GPT-5.6 视觉模型识别] ──> [计算目标元素像素坐标 (x:450, y:230)] ──> [执行鼠标点击/滚动]
```

结合 2026 年新增的 **Appshots** 能力（快捷双击），可瞬间将前台焦点窗口的视觉截图与文本上下文直接压入当前会话，免去人工截图粘贴。

7.2 视觉驱动的 UI 走查实战：Figma 还原对比

这是最实用的自动化场景：**让 Codex 自主比对设计图与本地网页渲染，自动微调样式。**

🎯 目标 (Goal)

对比本地网页 `/auth/login` 与设计稿截图 `figma_login_mockup.png`，消除视觉间距差异。

🛑 约束 (Constraints)

- 仅允许调整 `src/app/login/page.tsx` 的 Tailwind 工具类。
- 禁止改动原有 DOM 树与语义化标签。

🧪 自动化执行 Specs

在 ChatGPT 客户端中发送：

```
@Chrome 请按以下步骤完成 UI 视觉走查：

# 🎯 Goal
Compare and align browser rendering with figma_login_mockup.png.

# 🛑 Constraints
- Only use Tailwind utility classes in src/app/login/page.tsx.
- Do not change the DOM structure.

# 🚀 Execution Steps
1. Open Google Chrome and navigate to http://localhost:3000/auth/login.
2. Capture screenshot of the login form container.
3. Compare against assets/figma_login_mockup.png and identify padding discrepancies.
4. Update Tailwind classes in src/app/login/page.tsx to match spacing.
5. Reload page and confirm visual alignment within 2% delta.
```

7.3 Codex 自动走查的概念流程（示意）

```
> Running visual review for /auth/login
> Step 1: Opening Google Chrome on http://localhost:3000/auth/login...
> Step 2: Taking screenshot. Saved to /tmp/screenshot_v1.png
> Step 3: Calling vision model for image comparison.
    Analysis: "Login card padding-top is 16px (pt-4), but mockup requires 32px (pt-8). Font size
is text-base, needs text-xl."
> Step 4: Updating src/app/login/page.tsx via apply_patch...
> Step 5: Reloading Chrome tab and taking validation screenshot...
> Step 6: Vision check: "Visual delta is within 1.2% tolerance. Perfectly aligned."
> Task completed successfully.
```

7.4 进阶：Sites 原位预览与响应式多端巡检

借助 2026 ChatGPT 桌面端内置的 **Sites** 功能与移动端断点走查：

```
@Chrome
# 📱 Mobile Viewport Inspection
1. Open Chrome DevTools.
2. Toggle Device Toolbar and select iPhone 15 Pro (393 x 852).
3. Verify the submit button does not drop below the first screen fold.
4. If obscured, reduce hero section padding to keep the CTA button immediately clickable.
```

独立开发者无需手动来回缩放浏览器窗口，让智能体完成 90% 的样式走查脏活。

7.5 跨界延伸：从 Web 走查到 3D 软件自主操作（Blender 联动）

随着 2026 年具备原生空间几何直觉的模型 **GPT-6 Astra** 上线（并在 BenchCAD 基准中拿下 95.9% 的 CAD 空间重建成绩），Computer Use 的舞台已经不仅限于浏览器和前端页面。

在桌面环境中，开发者甚至可以让智能体自主打开专业 3D 创作软件 **Blender**：

- **视图操作与视口检查**：智能体通过 Computer Use 自动将 Blender 视口切换至 Shading 材质预览模式或 Rendered 渲染模式；
- **排查模型着色与破损**：智能体直接对 3D 视口进行截图比对，检查材质反射是否合理、法线是否翻转；
- **空间联动工作流**：配合 Ch.13 中详述的 **Tripo3D + Blender + Astra** 3D 自动化管线，实现从提示词生成几何体到桌面级 DCC 软件渲染验证的全自动化。

🛡️ 翻车自救与避坑速查表

常见踩坑现象	致命原因	极速自救指南
`Computer Use is not supported on this platform`	当前系统非 macOS 或位于未开放区域	确认使用 macOS 客户端，或改用 Headless Puppeteer 纯脚本方案作为替代
`Failed to capture window: Permission denied`	macOS 屏幕录制权限未正确勾选或需要刷新	打开系统设置，重新开关一次 ChatGPT 的「屏幕录制」权限并彻底重启客户端
AI 鼠标在屏幕上乱点，点不到目标按钮	浏览器缩放比例不是 100% 或多屏幕 DPI 换算异常	将 Chrome 缩放比例重置为标准 100%，并将待测窗口放置于主显示器正中

Ch.08 移动看护工作流：全天候离线编排实战

> 🎯 ***具体工程麻烦***：*开发者被困在工位看滚动编译日志；离开工位后 CI 挂了或高危发布卡住，整个团队进度中断。*
> 💡 ***可运行实战代码与落地收益***：*Guardian 自动审批（`--approve-for-me`）与移动看护网关双层分流；飞书 Webhook 告警卡片；手机远程一键审批部署。*
> ⚡ ***社交传播 / 截图金句***：*“下班不盯屏幕，任务照常推进。常规风险 AI 自审，高危发布手机一键批复。”*

独立开发者与产品经理的核心追求除了极致效能，还有高维的时间自由。整天死盯在终端前看几千行编译日志并不是 Vibe Coding 的初衷。

本章我们将搭建一套 **全天候移动看护工作流**：在 2026 年结合 **Guardian 智能自动审批** 与 **移动看护网关**，实现低风险操作 AI 自动放行、高危生产发布推送到手机飞书或微信群，人在户外随时随地一键批复。

🎯 生活化直觉隐喻：现代无人农场的智能巡视与中央呼机

把离线编排想象成经营一座现代无人农场：

```
【传统低效盯梢】 ──> 你一天 24 小时搬个小板凳坐在大棚里，死盯着灌溉水管会不会漏水（枯燥且无法脱身）。
【Guardian + 移动网关】──> 农场引入了全自动机械犬（由 GPT-5.6 Luna 驱动的 Guardian）：
                            - 水管轻微漏水？机械犬自己拧紧阀门，顺畅放行（Guardian 策略自审）；
                            - 遇到总水闸开关切换或高压电网变更？机械犬不敢妄动，
                              立刻往农场主的手机飞书发送一张带高清照片的确认卡；
                            - 你在地铁上喝着咖啡，手机点个「确认」，农场继续自动化运作。
```

这种“低级风险自主消化，核心关卡人工兜底”的机制，才是现代一人公司的终极形态。

🚀 新手极速上手 3 步走（无痛起步）

用 3 步搭建你的移动告警通知通道：

1. **步骤一：创建一个飞书 / 企微自定义群机器人**
   在群设置中添加机器人，复制其获得的 Webhook URL。
2. **步骤二：在终端发送一条测试告警卡片**
   运行 curl 测试手机能否收到推送：

   ```
   curl -X POST -H "Content-Type: application/json" \
     -d '{"msg_type":"text","content":{"text":"🔔 Codex 移动看护上线：终端任务正在安全运行！"}}' \
     https://open.feishu.cn/open-apis/bot/v2/hook/YOUR-WEBHOOK-TOKEN
   ```

3. **步骤三：用 Guardian + 沙盒模式启动你的离线任务**
   工位离席前，输入以下命令即可安心离开：

   ```
   codex exec --sandbox workspace-write --approve-for-me "执行全仓回归测试并重构陈旧类型定义"
   ```

8.1 双层看护分流架构

在 2026 年最新的架构中，移动端不再被琐碎的确认弹窗轰炸：

```
[Codex 长时任务] ──> 触发敏感动作（改依赖/写文件/调网络）
                                  │
                                  ▼
                  [第一道：Guardian 自动策略评审] ──(低风险)──> 自动放行继续执行
                                  │ (高危/生产部署)
                                  ▼
                  [第二道：移动看护网关] ──> [手机飞书/企微推送卡片] ──> [手机回复 1 放行]
```

配套参考开源工程：

- 蓝皮书官方飞书助理：plugins-codex-feishu
- 本机极简穿透网关：scripts/codex-watchdog

8.2 实战：GitHub Actions 失败推送与 Webhook 配置

在项目根目录下编写 `.github/workflows/codex-watchdog.yml`，当远程构建失败时，精准提炼失败摘要推送到手机：

```yaml
name: Codex Agent Watchdog

on:
  push:
    branches: [ main ]
  workflow_dispatch:

jobs:
  agent-build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout Code
        uses: actions/checkout@v4
```

```yaml
      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: 20

      - name: Run Codex Validation (exec mode)
        id: run_agent
        env:
          OPENAI_API_KEY: ${{ secrets.OPENAI_API_KEY }}
        run: |
          npm ci
          npm run test || echo "STATUS=failed" >> $GITHUB_ENV

      - name: Push Fail Notice to Mobile
        if: env.STATUS == 'failed'
        run: |
          curl -X POST -H "Content-Type: application/json" \
            -d '{
              "event": "build_failed",
              "repo": "${{ github.repository }}",
              "commit": "${{ github.sha }}",
              "message": "Codex sandbox testing failed. Attention required on phone."
            }' \
            ${{ secrets.MOBILE_WEBHOOK_URL }}
```

8.3 户外移动端双向交互与审批

1. 场景：生产环境部署审批

当 Codex 跑通所有测试，准备将代码发布到 Vercel 时，它会暂停并向飞书群发送审批卡片：

```
🚨 [Codex Auth Requested]
Project: pmer-cn-saas
Action: Deploy to production (Vercel)
Change summary: Implemented Stripe subscription webhook in /api/stripe.
Tests: 12 passed, 0 failed.
[回执指令]: 回复 "1" 批准发布，回复 "0" 打断并回滚。
```

2. 服务器端极简中转脚本 (Node.js)

网关服务解析手机回复，并通过信号文件或 socket 联动 Codex：

```javascript
// File: gateway.js
const express = require('express');
const { exec } = require('child_process');
const app = express();
app.use(express.json());

app.post('/api/mobile-reply', (req, res) => {
  const { userMessage, user } = req.body;
  if (user !== 'hunkwu') {
    return res.status(403).json({ error: 'Unauthorized' });
  }

  if (userMessage === '1') {
```

```
    exec('echo "approved" > /tmp/codex_deploy_signal', (err) => {
      if (err) return res.status(500).send('Error');
      res.json({ reply: '🚀 部署已批准，生产环境正在上线！' });
    });
  } else if (userMessage === '0') {
    exec('pkill -f codex && git checkout -- .', (err) => {
      res.json({ reply: '🛑 部署已中止，代码已安全回滚！' });
    });
  } else {
    res.json({ reply: '⚠️ 无效指令，请回复 1 (批准) 或 0 (中止)' });
  }
});

app.listen(8080, () => console.log('Mobile gateway listening on port 8080'));
```

8.4 官方参考实现：飞书助理 (Codex Feishu Sentinel)

为了避免开发者重复手写 Webhook 网关与长连接轮询逻辑，本项目配套开源了开箱即用的官方参考工程：**plugins-codex-feishu（飞书助理）**。

它将上述架构全面产品化，具备三大核心形态：

1. **日常巡检与值班日报（极速上手）**：
   - 自动聚合本地 Git 提交与工作区动态，生成图文并茂的进展日报，免去每天下班写进展报告的痛点。
   - 默认通过飞书 Bot 仅推送到开发者个人私聊，3 分钟即可完成从零到一的验证。
2. **移动端警报与双向审批（离线编排）**：
   - 当云端/本地 Codex 执行自动化测试遇挫、或遇到需要人工授权的高危部署时，飞书第一时间震动弹窗通知卡片。
   - 手机端直接点击或打字回复指令，即可远程触发「批准发布」或「一键回滚中断」。
3. **沉淀与进阶知识库（团队协同）**：
   - 支持将结构化项目报告写回飞书 Docx 云文档、同步至 Bitable 多维表格，实现代码进展向团队业务知识库的无缝沉淀。

> 💡 ***3 分钟极速配置**：该工程内置了飞书官方应用清单 `feishu_app_manifest.json`，在飞书开放平台直接点击「导入应用清单」即可自动开通所有必要权限与长连接，免去繁琐的手工勾选。*

🛡️ 翻车自救与避坑速查表

常见踩坑现象	致命原因	极速排查与自救指南
手机接收不到 Webhook 消息	机器人安全设置中未配置自定义关键词或 IP 白名单	检查飞书机器人设置中的“安全设置”，设置包含关键词（如 `Codex`）或签名校验
手机回复了“1”但本地没有任何动静	中转网关未能与执行机器建立长连接或文件信号未监听	运行 `node scripts/codex-watchdog` 检查隧道穿透连通性与信号文件状态
手机每隔 1 分钟被审批卡片轰炸	每一个细微改动都未做分流直接请求人工审批	开启 `--approve-for-me`，将非破坏性操作委派给 Guardian 自动审批

8.5 实战产品说心法：把自由留给自己

很多同行在用 AI 编程时，把自己活成了一个“人体测试机”和“Git 提交工具人”：AI 改完了，人去点刷新；AI 报错了，人复制报错发给 AI。

移动看护工作流的本质，是把人从“即时等待”中抽离出来。

通过将测试断言（Validation Specs）托付给 GitHub Actions，将异常通知托付给移动 Webhook，将最终签字权（Deploy Approval）掌握在手机端。你才能真正做到“人在喝咖啡，产品在自动进化”。

Ch.09 架构复苏：混乱遗留系统的全景解析与渐进式解耦

> 🎯 ***具体工程麻烦***：*接手几万行无文档、无测试的屎山系统，改一行崩三处，重构不敢碰，盲目重写又交不了差。*
> 💡 ***可运行实战代码与落地收益***：*基于 GPT-5.6 Terra 超长上下文的全景拓扑生成 Prompt；接口行为快照基准测试；微创解耦三步法。*
> ⚡ ***社交传播 / 截图金句***：*“面对几十万行没文档的遗留系统，别冲动推倒重来。先让 AI 画出地图、补上防崩测试，再做微创手术。”*

在独立开发或承接存量项目时，最让人胆战心惊的莫过于接手前人留下、毫无文档且零单测覆盖的“屎山”工程。任何微小的改动，都可能引爆隐藏在深处的连环地雷。

面对这类遗留系统，千万不要冲动推倒重写。借助 2026 年 **GPT-5.6 Terra 的超长上下文理解能力** 与沙盒隔离测试，我们可以实施教科书级别的“渐进式微创手术”。

🎯 生活化直觉隐喻：在高速公路上给飞驰的老卡车换轮胎

重构正在跑着的遗留老系统，就像这样一种高危操作：

```
【莽撞全面重写】 ──> ❌ “这台老卡车太破了，我要在路边重新手搓一台新车！”
                        （结果新车造了 6 个月没造出来，业务早断粮饿死了。）
【渐进式微创解耦】──> ✅ 老卡车继续在高速上拉货（核心业务不中断）：
                        1. 先拿无人机拍底盘（让 AI 扫描全仓生成拓扑地图）；
                        2. 装上安全防侧翻支架（为关键接口编写基准行为快照）；
                        3. 每次只拆卸一颗生锈的螺丝，换好测跑没问题再动下一颗。
```

Codex 就是带毫米级激光仪的特种维修臂，只要你给它戴上测试护栏，它就能精准切下坏疽而不伤及健康肌肉。

🚀 新手极速上手 3 步走（无痛起步）

接手一个陌生混乱仓库的第一天，按这 3 步稳健破局：

1. **步骤一：让 AI 输出全仓拓扑地图（只读探索）**
   在终端运行只读沙盒分析，生成 Mermaid 依赖图：
   ```
   codex exec --sandbox read-only "分析项目整体架构与数据库模型，在 docs/topology.md 中输出 Mermaid 依赖图"
   ```
2. **步骤二：为要修改的接口拍一张“行为快照”（录制基准）**
   要求 Codex：“不要动 `src/api/` 的业务代码，只为现有接口补齐 3 个成功与失败的单元测试”。
3. **步骤三：单点切除并用测试断言守门**
   下发微创重构 Specs，要求：“每次仅抽离一个函数，且修改后自动执行 `npm test`，测试必须全绿”。

9.1 第一步：全景逆向，建立代码拓扑图

接手混乱项目的首要任务是**画出全景地图**。利用 GPT-5.6 Terra 的长上下文扫描能力直接逆向出 Mermaid 实体关系图（ERD）与路由拓扑：

```
# 🎯 Goal
分析当前项目的数据库配置与路由层，生成核心表结构外键关联的 Mermaid ERD 图。
```

```
# 🛑 Constraints
- 只分析 prisma/schema.prisma 或 src/db/models/。
- 排除临时缓存表与第三方集成日志表。

# 🧪 Validation Specs
- 在 docs/database_topology.md 中输出语法正确的 Mermaid 渲染图。
```

Codex 几秒钟即可梳理出核心实体之间的多对多与级联关联，效率远胜人肉翻代码。

9.2 第二步：安全红线，编写行为基准测试

重构遗留代码的绝对铁律是：**动手术前，必须先拉起生命体征监护仪（基准测试）。**

```
# 🎯 Goal
为 src/pages/api/checkout.ts 接口编写基准单元测试，捕获当前真实的入参与返回快照。

# 🛑 Constraints
- 严禁修改 src/pages/api/checkout.ts 本身的任何逻辑代码。
- 使用本地 Jest / Vitest 执行测试套件。

# 🧪 Validation Specs
- 覆盖三种核心分支：正常结算、库存告罄报错、未认证拦截。
- 确保基准测试在当前旧代码上 100% 通过。
```

9.3 第三步：微创手术，实施渐进式解耦

1. 拆解胖控制器 (Fat Controllers)

将混杂在几百行单文件里的鉴权、打折、库存扣减逐个剔除为纯函数：

```
# 🎯 Goal
将 src/pages/api/checkout.ts 中的“折扣计算逻辑”抽离为独立服务 src/services/discountService.ts。

# 🛑 Constraints
- 外部 API 接口的 Request / Response JSON 格式必须保持严格向下兼容。
- 严禁影响 checkout.ts 的其他核心逻辑分支。

# 🧪 Validation Specs
- 运行 npm run test，确保之前编写的结算基准快照测试 100% 全绿通过。
- 运行 npm run lint 无任何类型错误。
```

2. 局部验证与安全回滚

在 `--sandbox workspace-write` 保护下，Codex 会在修改后立即运行验证断言。一旦发现行为破坏，依靠 Git 干净分支立即恢复：

```
# 遇到重构偏差时一键还原
git checkout -- src/pages/api/checkout.ts
```

🛡️ 翻车自救与避坑速查表

常见踩坑现象	致命原因	极速排查与自救指南
一句话让 AI “重构整个项目”，导致系统彻底瘫痪	步子迈太大，AI 在大规模文件修改中出现幻觉与类型断裂	严格约束重构粒度，每次 Prompt 仅允许改动 1 个特定服务或函数
重构后旧功能莫名其妙丢失或报错	缺少重构前的行为基准快照测试	坚持“测试先行”，在修改前必须先让 AI 针对旧代码写出 100% 通过的基准单测
AI 在重构中擅自把底层框架给升级了	约束中未锁死 package.json	在 AGENTS.md 与 Constraints 中强调：“严禁变更 package.json 内的任何依赖版本”

Ch.10 商业实战：2小时跑通 Next.js + Stripe 商业级 MVP

> 🎯 ***具体工程麻烦***：*想做独立付费产品，在登录鉴权、数据库模型、Stripe Webhook 验签和部署上折腾两周，热情耗尽还没上线。*
> 💡 ***可运行实战代码与落地收益***：*完整可运行代码库* `examples/ch10-saas-mvp` *（Next.js 15 + Supabase + Stripe 订阅）；Stripe CLI 本地支付闭环调试命令。*
> ⚡ ***社交传播 / 截图金句***：*“独立开发最核心的里程碑不是架构多完美，而是收到第一笔付款。2 小时搞定全套付费闭环。”*

作为独立开发者（Indie Hacker）或微型创业团队，你最核心的里程碑不是“完美架构”，应该是“收到第一笔付款”。很多人把时间浪费在了反复配置脚手架上，迟迟无法上线。

本章我们以极客速战速决的风格，教你如何指挥 Codex，在 2 小时内利用 `Next.js 15 (App Router) + Supabase (PostgreSQL) + Stripe` 搓出一个具有完整支付与会员权限闭环的 SaaS MVP。

> 📦 ***配套实战源码***：*examples/ch10-saas-mvp —— 完整可运行的订阅制 AI 翻译工具（TransFlow），自带 CAP 协议* `AGENTS.md`，`npm install && npm run build` *已验证通过。*

🎯 生活化直觉隐喻：开出能收银的“流动煎饼摊”

很多人做 SaaS 产品，总想着造一座五星级大酒店：

```
【空想五星级酒店】 ──> 花 6 个月设计华丽大堂、采购高档地毯、招几十个服务员（过度工程），
                       结果开业第一天发现没人愿意来吃饭，直接倒闭。
【流动煎饼果子小推车】──> ✅ 你只需要 3 件核心装备：
                       1. 煎饼炉子（Next.js 核心业务页面：提供核心价值）；
                       2. 储物箱（Supabase / PostgreSQL：存用户和订单数据）；
                       3. 扫码收款码（Stripe 订阅：能把钱结结实实收进账户）。
```

只要客人扫码付了 10 块钱，你把热腾腾的煎饼交到他手上，商业闭环就算真正跑通了！

🚀 新手极速上手 3 步走（无痛起步）

用 3 步快速打通本地 Stripe 支付联调闭环：

1. **步骤一：获取 Stripe 测试密钥并在本地声明**
   登录 Stripe 开发者控制台，获取测试公私钥并写入 `.env.local`：

   ```
   STRIPE_SECRET_KEY="sk_test_..."
   NEXT_PUBLIC_STRIPE_PUBLISHABLE_KEY="pk_test_..."
   ```

2. **步骤二：启动 Stripe CLI 本地 Webhook 转发**
   在终端运行监听命令，打通本地隧道：

```
stripe listen --forward-to localhost:3000/api/webhooks/stripe
# 终端将输出专属签名秘钥：whsec_...
```

3. **步骤三：让 Codex 沙盒执行并验证 Webhook 签名逻辑**
   将获得的 `whsec_...` 注入环境，并命令 Codex："编写测试用例，模拟触发 `checkout.session.completed`，必须返回 200 且用户权限状态变更为 ACTIVE"。

10.1 初始化骨架与数据库 Schema 设计

我们的目标是做一款订阅制的 AI 翻译工具。首先使用 Next.js 初始化项目，并配置好基础约束。接着，让 Codex 生成核心的数据库模型。

1. 编写 Prisma Schema (Prisma 实体建模)

向 Codex 下达目标 Specs：

```
# 🎯 Goal
编写 Prisma 数据库模型，支持用户表（User）、订阅表（Subscription）与翻译记录表（TranslationRecord）。

# 🛑 Constraints
- 数据库驱动使用 PostgreSQL（连接 Supabase）。
- 订阅状态必须是 Enum 类型，包含 ACTIVE, CANCELED, EXPIRED。

# 🧪 Validation Specs
- 运行 `npx prisma validate` 无错误提示。
```

Codex 会自动输出标准的 `prisma/schema.prisma` 模型，包含级联删除与外键索引：

```
datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

generator client {
  provider = "prisma-client-js"
}

enum SubscriptionStatus {
  ACTIVE
  CANCELED
  EXPIRED
}

model User {
  id             String              @id @default(uuid())
  email          String              @unique
  createdAt      DateTime            @default(now())
  subscription   Subscription?
  records        TranslationRecord[]
}

model Subscription {
  id             String              @id @default(uuid())
  userId         String              @unique
```

```
  stripeSubId    String               @unique
  status         SubscriptionStatus
  priceId        String
  currentPeriodEnd DateTime
  user           User                 @relation(fields: [userId], references: [id], onDelete:
Cascade)
}

model TranslationRecord {
  id             String               @id @default(uuid())
  userId         String
  sourceText     String
  translatedText String
  createdAt      DateTime             @default(now())
  user           User                 @relation(fields: [userId], references: [id], onDelete:
Cascade)
}
```

10.2 集成 Stripe 订阅与 Webhook 监听

支付系统的核心是**回调安全**。当用户支付成功后，Stripe 的服务器会向你的 Next.js 服务发送一个 Webhook 请求。

自动化 Specs：向 Codex 下达路由编写指令

```
# 🎯 Goal
实现 Next.js 15 App Router 风格的 Stripe Webhook 路由处理程序 `/api/webhooks/stripe`。

# 🛑 Constraints
- 必须使用 `stripe.webhooks.constructEvent` 验证请求签名的真实性，防止伪造攻击。
- 当接收到 `checkout.session.completed` 或 `invoice.payment_succeeded` 事件时，更新用户的订阅状态。
- 严禁对响应体进行普通 JSON 解析，Stripe 验证要求使用原始 req.text() 字符串。

# 🧪 Validation Specs
- 编写单元测试，验证签名失败时返回 400，成功时返回 200。
- 真实订阅周期必须从 Stripe 响应中读取，不得硬编码。
```

Codex 生成的标准实现：

```ts
// File: src/app/api/webhooks/stripe/route.ts
import { NextResponse } from 'next/server';
import Stripe from 'stripe';
import { prisma } from '@/lib/prisma';

const stripe = new Stripe(process.env.STRIPE_SECRET_KEY!, {
  apiVersion: '2026-04-22.dahlia' as any,
});

export async function POST(req: Request) {
  const body = await req.text();
  const signature = req.headers.get('stripe-signature')!;

  let event: Stripe.Event;

  try {
```

```
    event = stripe.webhooks.constructEvent(
      body,
      signature,
      process.env.STRIPE_WEBHOOK_SECRET!
    );
  } catch (err: any) {
    return NextResponse.json({ error: `Webhook Error: ${err.message}` }, { status: 400 });
  }

  if (event.type === 'checkout.session.completed') {
    const session = event.data.object as Stripe.Checkout.Session;
    const stripeSubId = session.subscription as string;
    const customerEmail = session.customer_details?.email!;

    const subscription = await stripe.subscriptions.retrieve(stripeSubId);

    await prisma.subscription.upsert({
      where: { stripeSubId },
      update: {
        status: 'ACTIVE',
        currentPeriodEnd: new Date(subscription.current_period_end * 1000),
      },
      create: {
        stripeSubId,
        status: 'ACTIVE',
        priceId: subscription.items.data[0]?.price.id || 'default',
        currentPeriodEnd: new Date(subscription.current_period_end * 1000),
        user: { connect: { email: customerEmail } }
      }
    });
  }

  return NextResponse.json({ received: true });
}
```

10.3 本地联调：用 Stripe CLI 跑通支付闭环

1. 在本地运行 Stripe 转发：

   ```
   stripe listen --forward-to localhost:3000/api/webhooks/stripe
   ```

2. 触发一次真实的结账测试：

   ```
   stripe trigger checkout.session.completed
   ```

3. 观察终端是否输出 `200 OK`，并使用 Prisma Studio 查验数据库：

   ```
   npx prisma studio
   ```

商业 MVP 的价值在于上线速度。用最严密的边界规约约束 AI，换取最极致的交付体验。

🛡️ 翻车自救与避坑速查表

常见踩坑现象	致命原因	极速排查与自救指南
`Webhook Error: No signatures found matching the expected signature`	使用了 `await req.json()` 解析 Body，破坏了加密签名原貌	严格使用 `await req.text()` 获取 Raw String，再传入 `constructEvent`
测试信用卡支付成功后，数据库无任何新增记录	Webhook URL 配置错误或本地没有运行 `stripe listen` 转发	启动 `stripe listen`，确认监听终端打出 `200 OK [POST /api/webhooks/stripe]`
`DATABASE_URL` 连不上 Supabase	忘记开启 Supabase 连接池（Session / Transaction 端口混淆）	检查数据库连接串，确保在 Serverless 环境使用 6543 连接池端口

Ch.11 触角延伸：Expo 跨端原生 App 开发与云端打包

> 🎯 ***具体工程麻烦***：*做完 Web 想做 App，但 Xcode 证书、Android Gradle、Cocoapods 依赖地狱让人崩溃，环境配三天打不出包。*
> 💡 ***可运行实战代码与落地收益***：*完整可运行代码库 `examples/ch11-expo-mobile`（Expo SDK 57 + Expo Router + NativeWind）；零本地配置云端打包 `eas build`。*
> ⚡ ***社交传播 / 截图金句***：*“不用配本地 Xcode 和 Android Studio，借助云端打包与 AI 报错自愈，一个人也能轻松交付双端原生应用。”*

做完网页版 SaaS 后，很多独立开发者希望能将触角延伸到移动端。但在传统的原生开发（React Native / Flutter）中，最耗费时间的往往是复杂的本地开发环境配置：iOS 证书管理、Android Gradle 报错、Cocoapods 冲突，这些环境地狱常常让人望而却步。

我坚信 **“云端开发与打包（EAS）是独立开发者做原生 App 的唯一解”**。结合 Codex 的智能编译报错排查，你可以完全跳过本地 Xcode/Android Studio 的繁琐配置，直接打包出可以上架的原生 App。

> 📦 ***配套实战源码***：*examples/ch11-expo-mobile —— 完整可运行的 Expo SDK 57 工程（Expo Router `src/app` 路由 + NativeWind + EAS 三档打包配置），自带 CAP 协议 `AGENTS.md`，已通过 `npx expo lint`（零错误）与 `npx expo-doctor`（20/20）验证。*

🎯 生活化直觉隐喻：写好剧本，让“云端工坊”为你定制戏服

跨端开发不需要把你的电脑变成重型加工厂：

```
【传统原生开发】 ──> 你在家里自己买炼钢炉（装几十 G 的 Xcode 和 Android Studio）、
                     买织布机（配各种 Gradle/Cocoapods 运行环境），动不动断电报错，三天造不出一只鞋。
【Expo + EAS 模式】──> ✅ 你只负责写剧本（写 React Native / TypeScript 核心代码）：
                     - 剧本写好后，一键发给「云端特种裁缝工坊」（EAS Build）；
                     - 云端自动为你剪裁出 iOS 的 IPA 和 Android 的 APK 安装包；
                     - 手机直接扫码就能把做好的衣服套在身上跑！
```

Codex 就是你在工坊里的剧本精修助理，负责在你写漏了依赖或样式语法时，自动对齐版本。

🚀 新手极速上手 3 步走（无痛起步）

用 3 步在手机真机上看到你的第一个跨端 App：

1. **步骤一：手机下载安装 Expo Go 客户端**
   在 App Store 或各大安卓应用商店搜索并安装「Expo Go」。
2. **步骤二：在项目终端启动本地开发服务器**
   进入项目目录，运行开发命令：

```
npx expo start
# 终端将打出巨大的二维码矩阵
```

3. **步骤三：用手机相机扫码即可秒开实时热更新**
   手机扫码后，App 瞬间加载完成。无论你在本地或让 Codex 调整任何页面代码，手机屏幕都会瞬间热重载呈现！

11.1 Expo 项目极速初始化与规范设定

为了能使用 EAS 进行云端免配置打包，我们首先初始化一个标准的 Expo 项目。

1. 编写 App 初始化 Specs

给 Codex 下达 Specs 指令：

```
# 🎯 Goal
初始化一个使用 TypeScript 的 React Native Expo 项目。

# 🛑 Constraints
- 使用最新稳定的 Expo SDK 57，搭配 Expo Router 实现基于文件系统的路由。
- 引入 NativeWind 作为 Tailwind 风格的样式方案。
- 目录约定使用 src/ 前缀（即 src/app/ 作为路由根）。

# 🧪 Validation Specs
- 运行 `npx expo lint` 无错误。
- 运行 `npx expo-doctor` 检查依赖版本对齐（必须 20/20 通过）。
```

目录结构规范：

```
src/
├── app/
│   ├── index.tsx          # APP 首页
│   └── _layout.tsx        # 全局路由导航布局
├── components/
└── hooks/
app.json                   # Expo 核心配置文件
package.json
```

11.2 解决移动端顽疾：原生依赖冲突

React Native 开发最忌讳使用普通 `npm install` 安装带底层原生代码的第三方包（如相机、传感器、手势库）。

铁律：使用 `npx expo install` 对齐版本

如果依赖版本不匹配，在 TUI 中输入纠偏指令：

```
请改用 `npx expo install react-native-reanimated` 重新安装该依赖，它会自动适配当前的 Expo SDK 版本。在本项目中，严禁使用普通的 `npm install` 安装任何原生依赖包。请将该铁律写入 AGENTS.md。
```

> 💡 ***主理人心法：*** *`npx expo install` 是官方认证的版本锁。任何时候只要原生模块报错，强制使用该命令覆盖安装，90% 的依赖地狱都会被自动抹平。*

11.3 EAS Build 云端打包与证书自动化

在传统流程中，申请苹果开发者证书往往会卡住新手几天。现在通过 EAS，全流程可以在云端自动化解决。

1. 配置 `eas.json`

```
{
  "cli": {
    "version": ">= 9.0.0"
  },
  "build": {
    "development": {
      "developmentClient": true,
      "distribution": "internal"
    },
    "preview": {
      "distribution": "internal"
    },
    "production": {
      "ios": {
        "simulator": false
      }
    }
  }
}
```

2. 让 Codex 监考云端构建日志

```
# 启动 EAS iOS 打包任务，把日志写入本地
eas build --platform ios --profile production --non-interactive 2>&1 | tee eas-build.log
```

如果打包遭遇异常，直接将日志转交给 Codex 分析：

```
codex exec --sandbox read-only "分析 eas-build.log，定位失败原因并给出修复命令"
```

最终 EAS 会返回一个安装二维码，你只需用手机扫码即可直接安装测试版 App。

🛡️ 翻车自救与避坑速查表

常见踩坑现象	致命原因	极速排查与自救指南
手机扫码显示“Could not connect to development server”	手机与电脑不在同一个 Wi-Fi 局域网，或电脑开启了路由器防火墙	运行 `npx expo start --tunnel`，强制使用公网穿透隧道模式，零网络门槛
`Invariant Violation: "main" has not been registered`	入口路由文件丢失或 `app.json` 中配置的 entry 路径错误	检查 `package.json` 中的 `"main": "expo-router/entry"` 是否被意外篡改
安装新组件后终端直接报红红屏	误用了 `npm install` 安装了不兼容当前 SDK 版本的包	运行 `npx expo install <package-name>`，或运行 `npx expo-doctor` 诊断修复

Ch.12 终局思考：独立开发者如何打造自动化商业飞轮

> 🎯 ***具体工程麻烦***：*花 99% 精力写代码，只花 1% 精力找客户，产品上线无人问津；一人微型创业无法兼顾开发与获客。*
> 💡 ***可运行实战代码与落地收益***：*自动化营销脚本（业务事件触发日报推送、社媒动态分发）；核心数据看板模板；一人公司商业运转流水线。*
> ⚡ ***社交传播 / 截图金句***：*“代码写得再漂亮，没人用就是精美的垃圾。让 AI 不仅帮你写代码，更帮你转动商业获客的轮盘。”*

在“实战产品说”微信公众号和 pmer.cn 上，我写过很多关于“独立开发与副业变现”的文章。很多开发者最容易走入的死胡同是：**把 99% 的精力用来打磨代码语法，却只花 1% 的精力去寻找真实用户和痛点。**

代码写得再优雅、架构配置得再完美，只要没人用，它就是一个精美的摆设。在 AI 原生时代，我们不仅要让 Codex 帮我们“生产产品”，更要让它帮我们“转动商业轮盘”。

🎯 生活化直觉隐喻：造一台日夜自转的“水利灌溉水车”

商业飞轮不是靠你每天手动人肉挑水：

```
【人力挑水型创业】 —> 你每天写代码到半夜，第二天还要四处发传单拉客户，
                      只要你生病歇一天，产品断更、客户归零（极其脆弱）。
【自动水车商业飞轮】—> ✅ 你在小河边组装好这台水利水车：
                      1. 导流槽（SEO 与自动化 Sitemap：持续吸附长尾搜索引擎流量）；
                      2. 碾米轮叶（核心 SaaS 功能：为用户解决实际问题）；
                      3. 出米漏斗（Stripe 自动结账与订阅扣款：持续产生现金流）；
                      4. 水位仪表盘（每日飞书数据战报：监控转化率与活跃指标）。
```

一旦把这套流水线组装完毕，无论你是在睡觉、吃饭还是旅行，水车日夜自转，源源不断为你创造价值。

🚀 新手极速上手 3 步走（无痛起步）

用 3 步迈出商业化闭环的第一步：

1. **步骤一：部署自动化 Sitemap 生成脚本**
   在项目部署流水线中自动执行 `scripts/generate-sitemap.js`，确保每次新增 Markdown 博客或页面自动被 Google 收录。
2. **步骤二：挂载每日早 8 点营收推送**
   配置定时脚本，每天准时往手机推送新增注册与付费活跃订阅数，培养对商业数字的敏感度。
3. **步骤三：在社区抛出你的“一句话价值提案”**
   在 X (Twitter)、即刻、V2EX 或微信朋友圈发布你的产品 MVP 链接，验证首批种子用户的真实付费反馈。

12.1 “一人 SaaS 公司”的自动化流量管道

一个健康的独立项目，其流量获取（SEO、长尾关键词、社交媒体）应该和它的代码库一样，是一套自动运转的流水线。

实战：让 Codex 自主维护 SEO 博客与 Sitemap

```
// File: scripts/generate-sitemap.js
const fs = require('fs');
const path = require('path');

const BASE_URL = 'https://pmer.cn';
const blogDir = path.join(__dirname, '../content/blog');

function getBlogSlugs() {
  if (!fs.existsSync(blogDir)) return [];
  return fs.readdirSync(blogDir)
    .filter(file => file.endsWith('.md'))
```

```
    .map(file => `/blog/${file.replace('.md', '')}`);
}

function generate() {
  const staticPages = ['/', '/auth/login', '/features'];
  const blogPages = getBlogSlugs();
  const allUrls = [...staticPages, ...blogPages];

  const xml = `<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  ${allUrls.map(url => `
  <url>
    <loc>${BASE_URL}${url}</loc>
    <changefreq>daily</changefreq>
    <priority>${url === '/' ? '1.0' : '0.8'}</priority>
  </url>`).join('')}
</urlset>`;

  fs.writeFileSync(path.join(__dirname, '../public/sitemap.xml'), xml);
  console.log('✅ Sitemap.xml generated successfully!');
}

generate();
```

12.2 对接业务数据，获取每日商业战报

为了让你对钱的动向足够敏感，每天早上拉取 Stripe 收益和用户增长，直接通过 Webhook 发送到手机：

```
// File: scripts/daily-report.js
const { PrismaClient } = require('@prisma/client');
const prisma = new PrismaClient();
const axios = require('axios');

async function sendReport() {
  const yesterday = new Date(Date.now() - 24 * 60 * 60 * 1000);
  const userCount = await prisma.user.count({
    where: { createdAt: { gte: yesterday } }
  });

  const activeSubs = await prisma.subscription.count({
    where: { status: 'ACTIVE' }
  });

  const reportText = `📊 【实战产品战报】\n昨日新增注册用户: ${userCount} 人\n当前总活跃订阅:
${activeSubs} 个\n—— 继续加油！`;

  await axios.post(process.env.MOBILE_WEBHOOK_URL, {
    msg_type: 'text',
    content: { text: reportText }
  });
}

sendReport();
```

12.3 终局心法：AI 原生时代的真正壁垒

当任何人都可以在几个小时内写出几万行代码、打包跨端原生 App、搭建起支付管道时，**纯粹的代码编写已经彻底商品化了**。在这个“代码极大充裕”的新时代，独立开发者和产品经理真正的终局壁垒在于：

1. **你对用户真实痛点深刻的同理心（User Empathy）**。
2. **你在具体行业中沉淀多年的业务认知（Domain Knowledge）**。
3. **你将 AI 智能体编排为生产力、快速验证商业闭环的执行力。**

🛡️ 翻车自救与避坑速查表

常见踩坑现象	致命原因	极速自救指南
产品上线后连续 1 个月零访问	闭门造车，未在早期接入 SEO 与社媒种子流量	立即在 Twitter/小红书/社区发帖分享产品开发故事，并用脚本输出多语种长尾博文
沉迷于重构优化性能，迟迟不敢发推	完美主义陷阱，恐惧被外界挑刺	记住：只要核心收单流程没坏，其他小瑕疵直接发，在真实用户反馈中迭代
为了微小的样式调整浪费大半天	没有把视觉微调全权委托给 Codex	用 Ch.07 学习的 Computer Use 自动巡检，把时间抢回来去跟付费用户聊天

Ch.13 前沿瞭望：2026 Codex 生态全景升级

> 🎯 ***具体工程麻烦***：*工具迭代极快，旧命令作废（Codex 桌面端退役并入 ChatGPT 代码模式）、模型换代（GPT-5.6）、CLI 变更不知所措。*
> 💡 ***可运行实战代码与落地收益***：*2026 迁移命令作战清单（winget/brew 安装、CLI 0.14x 变更配置、Plugins 插件系统对接）；版本兼容检查脚本。*
> ⚡ ***社交传播 / 截图金句***：*“工具每三个月换一轮血，心智才是护城河。这里没有空洞新闻，只有告诉你哪条命令已作废的作战地图。”*

前面十二章建立的，是一套不依赖具体版本的编排方法论。但方法论要落地，就必须踩在真实的工具地面上。2026 年的 Codex 生态发生了四次结构性地震：**桌面端大合并、模型大换代、插件生态成型、安全能力独立成军**。本章逐条拆解这些变化，并给出具体的迁移命令与配置。

🎯 生活化直觉隐喻：超音速客机的新一代航电换装

不要对频繁的版本迭代感到恐慌。用飞行员的视角来理解这次升级：

```
【驾驶心智】 ──> 你的飞行手册没有变：依然是目标驱动（起飞与降落目标）、边界约束（航线与安全高度）。
【动力引擎】 ──> 动力系统换装为前沿旗舰「GPT-6 Astra」（具备超视距雷达与全自主航电控制的顶级核心）、
                 日常主力发动机「GPT-5.6 Terra」以及高频巡航/Guardian 自动审批发动机「GPT-5.6 Luna」。
【驾驶舱屏】 ──> 原先外挂的副屏（独立 Codex App）正式集成到主驾驶台（ChatGPT 桌面端代码模式），
                 并新增 Sites 原位实景雷达与 Annotations 触控批注。
【自动副驾】 ──> 过去的简单定速巡航（--full-auto）升级为具备安全航线自检的「智能副驾」（--sandbox + Guardian）。
```

只要心智在握，新工具只会让你飞得更快、更稳。

🚀 新手极速上手 3 步走（无痛起步）

用 3 步完成你的 2026 工具链体检与迁移：

1. **步骤一：一键全面排查过时的旧模型代号**
   在项目根目录运行 grep 扫描：

   ```
   grep -rn "gpt-5.4" ~/.codex/config.toml .
   ```

2. **步骤二：全局升级至 CLI 0.147.0+ 稳定版**
   运行 npm 全局更新命令：

   ```
   npm install -g @openai/codex@latest
   codex --version
   ```

3. **步骤三：淘汰旧的 `--full-auto` 废弃指令**
   在所有本地别名（aliases）与 CI 脚本中，将 `--full-auto` 替换为：

   ```
   codex exec --sandbox workspace-write "<task>"
   ```

13.1 桌面端大合并：Codex App 并入 ChatGPT 客户端

2026 年 7 月，独立的 Codex 桌面应用正式退役，全部能力并入 **ChatGPT 桌面客户端**，以「Codex 代码模式（Code Mode）」的形态存在。免费、Plus 与企业版用户均可使用。

迁移动作：

```
# macOS：直接下载 ChatGPT.dmg，拖入 Applications
# Windows：使用 winget 安装
winget install OpenAI.ChatGPT
```

登录后点击左侧边栏的 **Codex** 标签页即可进入代码模式。需要特别注意的两点：

1. **账号 vs API Key**：使用 ChatGPT 账号登录可获得完整能力（云端任务、跨端同步、Sites 托管、Computer Use）；使用 API Key 登录仅有本地基础编码能力。
2. **独立组件不受影响**：Codex CLI、VS Code 插件、Codex Cloud 均为独立产品，继续正常演进。本书 Ch.02 的多端矩阵依然成立，只是「桌面 App」这一格换成了 ChatGPT 客户端。

新增能力速览：

- 内置浏览器升级：地址栏直接搜索浏览历史，Chrome 扩展可引用当前打开的标签页。
- **多仓库审查（Multi-repo Review）**：多文件夹项目可在一个视图中查看所有仓库的变更行数并逐一审查 Diff，不再需要来回切换。
- **Sites**：在客户端内直接创建、部署、管理托管 Web 项目，配合 Annotations 实现「指哪改哪」的原位编辑。

13.2 模型大换代：GPT-6 Astra 领衔与 2026 模型全景演进

2026 年下半年，OpenAI 的模型版图迎来了结构性质变：告别了过往单模型修修补补的策略，一方面正式推出了定位前沿自主智能体的最新旗舰 **GPT-6 Astra**，另一方面将 **GPT-5.6 家族** 固化为清晰的持久化三层阶梯体系（Sol / Terra / Luna）。

1. 新一代前沿旗舰登场：GPT-6 Astra（2026年9月3日发布）

OpenAI 于 2026 年 9 月 3 日正式发布最新一代旗舰模型 **GPT-6 Astra**（API 代号：`gpt-6-astra`）。这是 OpenAI 历史上智能化程度与对齐水准最高的模型，核心参数与特性包括：

- **超长上下文与超大输出**：原生支持 **105 万 (1.05M) Token 上下文窗口** 与 **128,000 输出 Token**，彻底打破超大型单体仓库与多应用全量代码注入的上下文天花板。

- **原生计算机操作员（Native Computer Operator）**：专为 **Computer Use** 设计，模型不再仅是文本生成器，而是具备像人类程序员一样直接识别屏幕像素、定位交互控件、跨多个桌面软件端到端执行研发与视觉校对的强大能力。
- **顶尖基准突破**：在 SWE-bench 软件工程基准与 FrontierMath 数学推理评测中刷新业界最高纪录。
- **安全分级认证**：在 OpenAI 预备框架（Preparedness Framework）中被评定为具备“关键（Critical）”级别的网络安全攻防与逆向推理能力。

2. 2026 在役主流模型矩阵决策表

面对多款在役模型，工程选型切忌“凡事盲目上最贵”，要按需分流：

模型代号	官方命名	定位与核心场景	优势与选型建议
`gpt-6-astra`	**GPT-6 Astra**	前沿自主智能体 / Computer Use 计算机操作员	跨多个桌面软件联动、超长全栈架构设计、高保真视觉走查与像素自愈
`gpt-5.6-sol` / `gpt-5.6`	**GPT-5.6 Sol**	顶级推理与架构旗舰 (Daybreak Blue)	大规模遗留系统重构、高难度算法攻坚、系统安全防御
`gpt-5.6-terra`	**GPT-5.6 Terra**	均衡型日常研发主力工作母机	团队日常主力编码，兼顾深度推理思考与响应速度，性价比极高
`gpt-5.6-luna`	**GPT-5.6 Luna**	轻量极速低延迟与自动化守卫	高并发轻量改动、测试路由分发、Guardian (`--approve-for-me`) 自动化审批

3. 2026 OpenAI 模型上新与退役下架路线图（Deprecation & Sunset Schedule）

随着新模型迭代，OpenAI 在 2026 年执行了密集严谨的清理与换代计划。开发者必须定期排查自己的依赖配置，避免生产突发中断：

时间节点	动作类型	涉及模型 / 服务	影响与官方替代方案
2026年3月26日	彻底关闭 (Shutdown)	`gpt-4-0314`, `gpt-4-1106-preview`, `gpt-4-0125-preview`	早期 GPT-4 快照停服，全面迁移至 `gpt-5.6-terra`
2026年5月12日	API 下架 (Shutdown)	DALL-E 2 / DALL-E 3	图像 API 下架，迁移至 `gpt-image-1` / `gpt-image-2`
2026年6月	界面退役 (Retired)	`o3`, `gpt-4.5`, 早期 `gpt-5` 测试快照	从 ChatGPT 界面移除，全面收敛至 GPT-5.6 / 6 体系
2026年8月26日	协议下线 (Shutdown)	老版 Assistants API	废弃旧架构，全面过渡为 Responses API 与原生 Agent 范式
2026年8月31日	正式退役 (Retired)	`gpt-5.4`, `gpt-5.4-mini`	从 Codex 登录态移除，全面由 `gpt-5.6-terra` 与 `luna` 接管
2026年9月3日	正式发布 (Launch)	**GPT-6 Astra (`gpt-6-astra`)**	全新前沿旗舰上线，支持 105 万上下文与原生 Computer Use
2026年10月23日	批量关闭 (Scheduled)	残存各版本 Preview 快照模型	官方统一清理预览版，强制要求锁定长期稳定模型 ID
2026年12月1日	图像整合 (Scheduled)	`gpt-image-1-mini`, `gpt-image-1.5`, `chatgpt-image-latest`	图像能力统一整合收敛至 `gpt-image-2`

配置升级推荐：

```
# ~/.codex/config.toml
```

```toml
# 默认日常主力研发模型（推荐平衡深度与速度的 Terra）
model = "gpt-5.6-terra"

# 开启前沿自主操作与超长上下文（按需启用最新旗舰 Astra）
# model = "gpt-6-astra"

# 后台自动化审批与审查守卫（极速低成本 Luna）
[profiles.guardian]
model = "gpt-5.6-luna"

# 高阶全栈架构与安全性深度审计（顶级推理 Sol）
[profiles.architect]
model = "gpt-5.6-sol"
```

> ⚠️ ***注意**：所有定时任务（Automations）、工作区默认配置、自定义 Agent 都要全面排查更新，禁止在生产中使用已退役的 `gpt-5.4` 或临时 `preview` 模型。*

13.3 CLI 0.14x：必须知道的决定性变更

Codex CLI 已全面进入 `0.14x` 时代（最新稳定版 `0.147.0+`）。

```bash
npm install -g @openai/codex@latest
```

1. `--full-auto` 正式移除

```bash
# ❌ 旧写法（已报错）
codex exec --full-auto "修复所有 lint 错误"

# ✅ 新写法：用沙盒模式表达自治级别
codex exec --sandbox workspace-write "修复所有 lint 错误"
```

2. Hooks 引擎转正（Stable）

在 `config.toml` 中直接配置 `SessionStart` / `Stop` 钩子，并能观测 MCP 工具调用、`apply_patch` 与长时间运行的 Bash 会话：

```toml
# ~/.codex/config.toml
[hooks]
session_start = "bash scripts/bootstrap-env.sh"
stop = "npm run lint --silent"
```

3. Agent Plugins 与插件市场

支持本地、工作区与远程三方市场：

```bash
# 插件管理统一入口
codex plugin list
codex plugin marketplace add https://plugins.example.com/catalog.json
codex plugin install security-workbench
```

在对话中通过 `@plugin` 提及即可自动注入插件上下文。

4. 子智能体（Subagents）与多智能体编排

CLI 原生支持派生拥有独立上下文窗口的子智能体：

```
> 派生两个子智能体：一个为 src/api 补充集成测试，
  另一个并行重构 src/hooks 的状态管理，最后汇总 Diff 给我审查。
```

5. Guardian 自动审批与 `--approve-for-me`

高危操作不再只有“人肉点确认”一条路。新的 `--approve-for-me` 标志可将审批请求路由给 Guardian 子智能体（由 GPT-5.6 Luna 驱动）自动评审：

```
codex --approve-for-me "升级依赖并跑通测试"
```

13.4 Skills 生态：把重复流程沉淀为资产

如果说 AGENTS.md 是项目的“通用宪法”，**Skills 就是某一类任务的“专项工艺流程”**：

```
skills/
└── pr-review/
    └── SKILL.md      # 带 YAML frontmatter 的流程说明书
```

13.5 Codex Security 与 Daybreak：安全能力独立成军

面向企业防御与关键基础设施保障，OpenAI 推出了 **Daybreak** 攻防体系与官方安全插件：

- **Daybreak Blue**：通用工程防御与漏洞自愈，基于高阶推理模型 **GPT-5.6 Sol**，覆盖自动化代码安全补丁生成与配置加固。
- **Daybreak Red / Frontier Eval**：红队攻防与深度渗透评估，基于具备“关键（Critical）”网络安全评级的最新前沿旗舰 **GPT-6 Astra**（通过受控权限与严格安全合规门禁），提供端到端未知漏洞挖掘与自动化逆向分析。

```
codex plugin install codex-security
codex-security scan --deep --report sarif > results.sarif
```

13.6 空间智能与 3D 跨界狂飙：GPT-6 Astra 联动 Blender 与 Tripo3D 实战

随着大语言模型从传统的纯文本与前后端代码，进化到具备物理世界空间理解能力的具身智能，2026 年下半年掀起了一场席卷 3D 建模、空间计算与游戏开发的**跨界狂飙**。

1. 行业突围：从“二维代码”到“空间具身几何”

2026 年 9 月，OpenAI 官方在 X 平台发布实机演示（tweet #2100679992720142459）并公布了全新的 **BenchCAD** 评测基准——GPT-6 Astra 在“多视角视图逆向重构 CAD 几何代码”任务中拿下了惊人的 **95.9% 体素交并比 (mean voxel IoU)**。这意味着大模型长期为人诟病的“缺乏三维空间几何直觉”的短板被彻底攻破。

全球头部 3D 生成独角兽 **Tripo3D** 随后迅速官宣深度集成，上线了专有的 **GPT-6 Astra 专属 3D 提示词工程模型中心**。独立开发者如今可以用一套极简管线，完成过去需要数周的 3D 资产建模、灯光材质烘焙与端到端渲染。

2. “双引擎”协同体系心智模型

独立开发者必须破除一个常见误区：“指望大语言模型单凭文字直接生造出包含数万个多边形的复杂 3D 网格是不切实际的”。当前最高效的工业级实战范式，是 **GPT-6 Astra 与专业 3D 生成引擎（Tripo3D）及 DCC 软件（Blender）的优势互补**：

```
[ 创意设想与空间需求 ]
          |
          ▼
[ GPT-6 Astra (3D 架构与技术美术总监) ] ──> 结构化提示词扩写 (拓扑/PBR材质/比例约束)
          |
          ▼
[ Tripo3D (原生 3D 几何与纹理生成引擎) ] ──> 极速生成连续网格 (Quad 拓扑, PBR, UV) 导出 .glb
          |
          ▼
[ Blender + bpy / Blender MCP (Astra 自动化接管) ] ──> 自动重设原点、Studio三点布光、多角度巡检渲染
          |
          ▼
[ 前端 Web / 游戏引擎 (Three.js / React Three Fiber) ] ──> 零成本在线交互分发
```

- **Tripo3D（原生几何与材质引擎）**：拥有海量优质 3D 结构训练数据，负责从单张参考图或文本生成拓扑规整（Quad-dominant）、自带法线与高光贴图的基底资产。
- **GPT-6 Astra（技术总监与自动化流水线中枢）**：充当资深 3D 技术美术（TA）。它负责提炼 3D 专属 Prompt，并通过 Python 脚本（`bpy`）或 **Blender MCP** 自动化接管 Blender，完成模型居中、地面吸附、影棚布光、摄像机运镜并基于视觉能力自主走查修复。

3. 四步端到端落地操作演练

步骤一：使用 GPT-6 Astra 结构化提纯 3D 提示词

在 Codex 终端或桌面端中向 GPT-6 Astra 发送需求，要求其以 3D 技术总监角色输出适合 Tripo3D 的提示词：

```
你现在是资深 3D 技术美术总监。我要在 Web 项目中嵌入一个“赛博朋克风格的全息贩卖机”。
请输出一份专为 Tripo3D 优化的专业结构化 3D 生成提示词。

要求：
1. 突出主体轮廓（Silhouette）与几何对称性；
2. 强制指定拓扑质量标签（Clean quad topology, game-ready, low-poly）；
3. 声明 PBR 材质映射规范（Matte painted metal, glowing neon emissive panels）。
```

Astra 将输出可直接复制至 Tripo3D 控制台的标准 Prompt：

```
Cyberpunk holographic vending machine, freestanding rectangular kiosk with chamfered edges.
Center features transparent acrylic dispenser bay with internal glowing LED strip.
Materials: Brushed matte dark gray titanium alloy (roughness: 0.35, metallic: 0.85),
glowing cyan neon emissive trim lines (emission strength: 5.0), scratched hazard decals on base.
Topology tags: clean quad-dominant topology, watertight mesh, manifold geometry, PBR 4K
textures, game-ready asset.
```

步骤二：Tripo3D 快速生成并导出标准化 GLB

在 Tripo3D 官网平台 输入提纯后的提示词，平台将在数十秒内完成高精度网格构建与 UV 烘焙。点击导出为标准 **`vending_machine.glb`** 文件，存放于项目的 `public/models/` 目录中。

步骤三：GPT-6 Astra 编写 Blender 自动化管线脚本（`pipeline.py`）

无需人工打开 Blender 费时调整，让 Astra 编写并运行完全无头的 Python 自动化脚本：

```python
import bpy
import math

# 1. 初始化纯净环境（清空默认立方体与杂乱灯光）
```

```
bpy.ops.wm.read_factory_settings(use_empty=True)

# 2. 自动导入 Tripo3D 生成的 GLB 资产
asset_path = "./public/models/vending_machine.glb"
bpy.ops.import_scene.gltf(filepath=asset_path)

# 3. 选中导入网格，自动计算边界盒并吸附至地面 (Z=0)
imported_objs = [obj for obj in bpy.context.selected_objects if obj.type == 'MESH']
if imported_objs:
    bpy.ops.object.select_all(action='DESELECT')
    for obj in imported_objs:
        obj.select_set(True)
    bpy.context.view_layer.objects.active = imported_objs[0]
    bpy.ops.object.origin_set(type='ORIGIN_GEOMETRY', center='BOUNDS')

    # 获取世界坐标系下的最低点并补偿对齐
    min_z = min([v[2] for obj in imported_objs for v in [obj.matrix_world @ v.co for v in
obj.data.vertices]])
    for obj in imported_objs:
        obj.location.z -= min_z

# 4. 自动化标准影棚三点布光 (Studio 3-Point Lighting)
# 主光 (Key Light)
key_light_data = bpy.data.lights.new(name="Key_Light", type='AREA')
key_light_data.energy = 500
key_light_data.size = 2.0
key_light = bpy.data.objects.new(name="Key_Light", object_data=key_light_data)
key_light.location = (3.0, -3.0, 4.0)
bpy.context.collection.objects.link(key_light)

# 辅光 (Fill Light)
fill_light_data = bpy.data.lights.new(name="Fill_Light", type='AREA')
fill_light_data.energy = 200
fill_light_data.size = 3.0
fill_light = bpy.data.objects.new(name="Fill_Light", object_data=fill_light_data)
fill_light.location = (-3.0, -2.0, 2.0)
bpy.context.collection.objects.link(fill_light)

# 轮廓光 (Rim Light)
rim_light_data = bpy.data.lights.new(name="Rim_Light", type='SUN')
rim_light_data.energy = 3.0
rim_light = bpy.data.objects.new(name="Rim_Light", object_data=rim_light_data)
rim_light.location = (0.0, 4.0, 3.0)
rim_light.rotation_euler = (math.radians(-45), 0, math.radians(180))
bpy.context.collection.objects.link(rim_light)

# 5. 设置展示摄像机
cam_data = bpy.data.cameras.new("Product_Camera")
cam_obj = bpy.data.objects.new("Product_Camera", cam_data)
cam_obj.location = (0, -4.5, 2.0)
cam_obj.rotation_euler = (math.radians(72), 0, 0)
bpy.context.collection.objects.link(cam_obj)
bpy.context.scene.camera = cam_obj

# 6. 配置 Eevee 快速无头验证渲染
bpy.context.scene.render.engine = 'BLENDER_EEVEE_NEXT'
```

```
bpy.context.scene.render.resolution_x = 1280
bpy.context.scene.render.resolution_y = 720
bpy.context.scene.render.filepath = "./public/renders/validation_preview.png"

bpy.ops.render.render(write_still=True)
print("✅ Blender 自动化巡检渲染完成：./public/renders/validation_preview.png")
```

在终端以无头后台方式运行：

```
blender --background --python pipeline.py
```

渲染完成后，GPT-6 Astra 直接读取 `validation_preview.png`，运用 Vision 视觉感知核对有无模型破损、法线反转或光影过曝，自主迭代代码直到测试通过。

步骤四：一键嵌入 Web 端（React Three Fiber 商业化呈现）

最后，将该资产以零成本嵌入你的前端产品（如 Next.js 15 或 Vite 项目）中，供用户实时旋转查看：

```
// components/VendingMachineViewer.tsx
import { Canvas } from '@react-three/fiber';
import { useGLTF, OrbitControls, Environment } from '@react-three/drei';

export function VendingMachineViewer() {
  const { scene } = useGLTF('/models/vending_machine.glb');
  return (
    <div className="w-full h-[500px] bg-slate-900 rounded-xl overflow-hidden shadow-2xl">
      <Canvas camera={{ position: [0, 2, 4], fov: 45 }}>
        <ambientLight intensity={0.7} />
        <directionalLight position={[5, 10, 5]} intensity={1.2} />
        <primitive object={scene} position={[0, -1, 0]} />
        <Environment preset="city" />
        <OrbitControls autoRotate autoRotateSpeed={2.0} enableZoom={true} />
      </Canvas>
    </div>
  );
}
```

13.7 成本优化与弹性分流：Codex Switch 多供应商实战

在 2026 年多模型共存的生态下，高水准独立开发者绝不会被单一供应商锁死：

1. **跨级别模型弹性分流**：常规日常编码使用 `gpt-5.6-terra`，轻量审批使用 `gpt-5.6-luna`，只在需要全桌面级视觉走查、3D 空间计算与超长上下文攻坚时才调用 `gpt-6-astra`。
2. **多供应商与灾备切换**：当遇到 OpenAI 官方账号用量限流（Rate Limit）或额度见底时，借助开源利器 **Codex Switch**，**一秒切换至 DeepSeek 或 Kimi Code，其特有的跨供应商会话续聊机制**能够保持历史会话上下文无损延续，保障全天候研发闭环不间断。

13.8 本章迁移清单（TL;DR）

```
# ① 桌面端：改装 ChatGPT 客户端
winget install OpenAI.ChatGPT        # Windows

# ② CLI 升级到 0.147.0+
```

```
npm install -g @openai/codex@latest

# ③ 模型升级到 GPT-6 Astra / GPT-5.6 阶梯
grep -rn "gpt-5.4" ~/.codex/ .        # 全面排查退役模型并替换

# ④ 替换已移除的 --full-auto
codex exec --sandbox workspace-write "<task>"

# ⑤ 启用 Guardian 自动审批
codex --approve-for-me "<task>"

# ⑥ 多供应商无缝切换与防限流
git clone https://github.com/aipmer/codex-switch.git && cd codex-switch && ./install.sh
codex-switch --to deepseek

# ⑦ 安装插件市场与安全插件
codex plugin marketplace add <catalog-url>
codex plugin install codex-security
```

🛡️ 翻车自救与避坑速查表

常见踩坑现象	致命原因	极速排查与自救指南
`Unknown argument: --full-auto`	0.14x 彻底删除了该参数，脚本报错中断	替换为 `--sandbox workspace-write`
`API Error: Model 'gpt-5.4' is deprecated`	8月31日后模型已正式下线	编辑 `~/.codex/config.toml`，将模型修改为 `gpt-5.6-terra`
`Failed to install plugin: catalog not found`	插件市场源 URL 无法访问或网络被阻断	检查网络连通性，或使用本地路径安装插件：`codex plugin install ./my-plugin`

13.7 终局不变量

工具层面的结论只有一条：**凡是写死在脚本里的版本号、命令行标志、模型代号，都是技术债**。把它们集中到 `config.toml` 与 `AGENTS.md` 中管理，让迁移成本收敛到「改一行配置」。

而穿越所有版本周期仍然成立的，正是本书反复强调的三件事：目标驱动而非步骤驱动、边界先行而非事后追责、人机分工而非人被 AI 牵着走。工具会换血，心智不打折。
