匠人学院
JR ACADEMY

2026 实战版

告诉 Cascade 你要什么，它帮你写代码、跑命令、修 bug

Windsurf Agentic IDE 中文上手手册

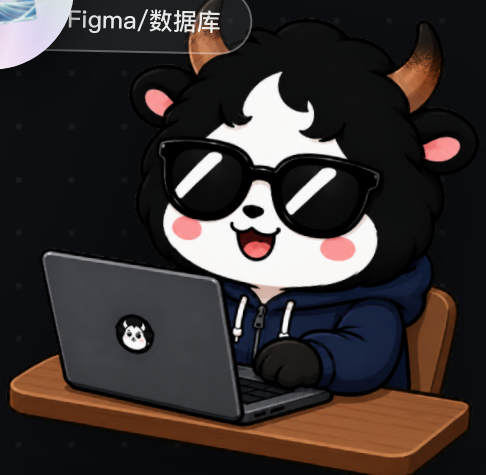
Codeium 出品、Cognition 收购的 AI 原生代码编辑器。Cascade Agent + Flow State 实时追踪 + 无限 Tab 补全——免费版就能用。8 章从安装到全栈实战，.windsurfrules 配置 / MCP 接外部工具 / Cascade 调 Bug 全在这。

Free 版含无限 Tab 补全

Cascade 自动跨文件改代码

Figma/数据库

一个人干整个项目，让 Cascade 跑腿



jiangren.com.au · 在线版随官网更新 /wiki/windsurf-guide

AI 来匠人 ✦ Windsurf Agentic IDE 手册 ✦ JIANGREN.COM.AU ✦ Cascade 帮你写代

CONTENTS

目录（8 章）

📖 这本手册怎么读

这本书和官网 wiki《Windsurf AI IDE 实战指南》同源（jiangren.com.au/wiki/windsurf-guide，免费，不用注册，官网那份持续更新）。

完全没碰过 Windsurf 的，第 1、2 章搞清楚概念，跑通第一个 Todo App；用 Cursor 转过来的，第 4 章的 .windsurfrules 是重点，对比 .cursorrules 一分钟就懂；想用 Cascade 调 bug 的，直接跳第 6 章，有五个可直接复制的调试 prompt 模板；想搭全栈项目的，第 8 章有完整从零搭建记录。代码全可复制，配置直接用。

💬 装好 Windsurf 想进 AI 开发群？扫这里

群里每周分享 AI IDE 选型、Cascade 配置技巧和真实项目案例。

CHAPTER 01

🌊 Windsurf 是什么：第一个 Agentic IDE

Windsurf 一句话介绍

Windsurf 是由 Codeium 团队打造的 AI 原生代码编辑器，2024 年底发布，2025 年 12 月被 Cognition AI（Devin 的母公司）收购。它基于 VS Code 内核，但把 AI 做成了编辑器的底层能力，而不是插件。

核心卖点就一个词：**Cascade**——一个能理解你整个代码库、跨文件推理、自主执行多步任务的 AI Agent。你告诉它"给这个项目加一个登录页面"，它会自己规划、创建文件、写代码、跑命令，一条龙搞定。

技术架构

Windsurf 不只是"VS Code 加了个 AI 侧边栏"。它在 VS Code 内核之上加了三层自研基础设施：

```
┌──────────────────────────────────┐
│         Cascade Agent            │  ← 多步推理 + 代码执行
├──────────────────────────────────┤
│     Flow State Engine            │  ← 实时追踪编辑 / 终端 / 剪贴板
├──────────────────────────────────┤
│   Codebase Indexing（本地向量库）  │  ← 语义搜索全项目
├──────────────────────────────────┤
│        VS Code 内核               │  ← 编辑器 / 插件 / 终端
└──────────────────────────────────┘
```

- **Codebase Indexing**：项目打开后自动建索引。Cascade 回答问题时不是靠你手动 `@` 文件，它能语义搜索整个代码库找到相关代码。
- **Flow State Engine**：持续监听你在编辑器里的行为——改了哪行、跑了什么命令、复制了什么报错。Cascade 随时知道你"在干嘛"。
- **Cascade Agent**：拿到上下文后，拆任务 → 改代码 → 跑命令 → 验证结果，循环执行直到任务完成。

Cognition AI 收购 Windsurf 后，Cascade 的 Agent 能力在持续增强——长期方向是让 Windsurf 接近一个"IDE 里的 Devin"。

和 Cursor、Claude Code 有什么不同

这三个工具代表了 AI 编程的三种思路：

特性	Windsurf	Cursor	Claude Code
本质	AI 原生 IDE	VS Code 魔改 + AI	终端 AI Agent
核心 AI	Cascade Agent	Composer Agent	Claude CLI
自动补全	Supercomplete（不耗 credit）	Tab 补全	无
上下文感知	Flow State 实时追踪	手动 @ 引用	自动读代码库
学习曲线	低，VS Code 用户直接上手	低	中等（需要终端基础）
免费额度	有，含无限 Tab 补全	有，2K 补全	无免费额度

实际使用中的差别：

- **Windsurf 的 Flow State** 会追踪你的每一个操作——编辑、终端命令、剪贴板——然后 Cascade 能直接"接住"你的意图，不用反复解释上下文。你复制了一段报错，直接说"修这个"就行。
- **Cursor 的 Composer** 在处理明确任务时速度更快，但大项目里上下文管理不如 Windsurf 自动化。Cursor 的 `.cursorrules` 生态更成熟，社区更大。
- **Claude Code** 在复杂重构和代码质量上碾压，但它是终端工具，没有可视化界面和自动补全。适合大型代码库的架构级改动。

```
# 选择思路：
# 日常写代码 + 需要自动补全 → Windsurf 或 Cursor
# 大型重构 + 安全审计 + CI/CD 自动化 → Claude Code
# 最佳组合：IDE 里用 Windsurf 写业务代码，终端开 Claude Code 做重构和审查
```

谁适合用 Windsurf

- **前端开发者**：Web Preview 功能可以直接在 IDE 里预览网页，点击元素让 Cascade 改样式，比 alt-tab 浏览器高效一个档次
- **全栈独立开发者**：一个人干整个项目，Cascade 的多文件编辑 + 终端命令执行能省大量时间
- **从 VS Code 迁移的人**：设置、插件、快捷键都能一键导入，零切换成本
- **预算有限的开发者**：免费版的 Tab 补全不限量，对学生和个人开发者友好
- **设计转开发的人**：Figma MCP 可以读取设计稿直接生成代码，降低实现门槛

不太适合的场景：纯后端 / 系统编程（Rust、C++、Go），这类场景 Claude Code 的终端模式更顺手；团队已经重度依赖 Cursor 生态（大量 `.cursorrules` 积累）的情况下迁移成本较高。

CHAPTER 02

🚀 安装与第一个项目

下载安装

Windsurf 支持 macOS、Windows、Linux 三个平台，去官网下载对应安装包就行：

```
# macOS（也可以直接下载 .dmg）
brew install --cask windsurf

# Linux（Debian/Ubuntu）
# 去 windsurf.com/download 下载 .deb，然后：
sudo dpkg -i windsurf_*.deb

# Windows
# 直接下载 .exe 安装包，双击运行
```

安装后第一次启动，Windsurf 会问你要不要导入 VS Code 或 Cursor 的设置。**强烈建议导入**——主题、快捷键、插件都能保留，体验几乎一模一样。

注册与登录

打开 Windsurf 后右侧会出现 Cascade 面板，点击登录。支持 Google、GitHub 账号直接注册，免费账户就能开始用。

注册完成后你会获得：

免费额度	说明
Tab 自动补全	**无限量**，不消耗任何 credit
Cascade 对话	每月 25 次 prompt credit
模型	默认使用 Codeium 基础模型

25 次 Cascade 够你完整体验功能了。觉得好用再考虑升级 Pro（$15/月，500 credit）。

第一个项目：用 Cascade 生成一个 Todo App

打开一个空文件夹，然后在 Cascade 面板里输入：

```
帮我创建一个 React + TypeScript 的 Todo 应用，要求：
- 用 Vite 脚手架
- 支持添加、删除、标记完成
- 用 Tailwind CSS 做样式
- 数据存 localStorage
```

Cascade 会开始规划任务，你能看到它的执行步骤：

1. 运行 `npm create vite@latest` 初始化项目
2. 安装 Tailwind CSS 依赖并配置
3. 创建 `TodoApp.tsx`、`TodoItem.tsx` 组件

4. 编写 localStorage 读写逻辑
5. 运行 `npm run dev` 启动开发服务器

整个过程大概 2-3 分钟。Cascade 每一步都会展示 diff 预览，你可以点 Accept 或 Reject 逐步控制。

关键操作速查

快捷键	作用
`Ctrl/Cmd + L`	打开 Cascade 面板
`Ctrl/Cmd + I`	Inline Edit（选中代码后原地修改）
`Tab`	接受 Supercomplete 补全建议
`Esc`	拒绝当前补全建议
`Ctrl/Cmd + Shift + P`	命令面板（和 VS Code 一样）
`Ctrl/Cmd + .`	快速修复（Quick Fix）

Turbo 模式

Cascade 默认每一步都要你点 Accept。开启 Turbo 模式后，Cascade 可以自动执行终端命令，不再逐步确认。适合你信任 AI 输出、想让它一口气跑完的场景。

在 Cascade 面板顶部找到 Turbo 开关，或者在设置里搜索 `turbo`：

```
// settings.json
{
  "windsurf.cascade.turboMode": true
}
```

刚上手的时候建议关着 Turbo，等你熟悉 Cascade 的行为模式后再打开。跑 `rm`、`DROP TABLE` 这类危险命令时 Cascade 仍然会弹确认，不会盲执行。

索引与 .windsurfignore

第一次打开大项目时，Windsurf 会做代码库索引（Indexing）。这个过程会吃 CPU 和内存，大项目可能要等几分钟。**别急着开始问 Cascade 问题**，等右下角的索引进度条跑完再操作，否则 AI 回答的上下文会不完整。

在项目根目录创建 `.windsurfignore` 文件，排除不需要索引的目录：

```
node_modules/
dist/
build/
.git/
*.lock
coverage/
.next/
```

这样能把索引文件数量减少 90% 以上，Cascade 响应也会更快。

一个实测数据：10 万行代码的 Next.js monorepo，不加 `.windsurfignore` 索引要 4 分钟，排除 `node_modules` 和 `dist` 后 40 秒内完成。

Memories：让 AI 记住你的习惯

Windsurf 的 Memories 功能会在使用过程中学习你的编码偏好——比如你总是用 `const` 不用 `let`、API 错误处理总是返回 `{ success: false, error }` 格式。大约用两天后 Memories 开始生效，之后 Cascade 生成的代码会自动符合你的风格，不用每次在 prompt 里重复说明。

Memories 在设置里可以查看和清除：命令面板搜 `Windsurf: Manage Memories`。

CHAPTER 03

⚡ 三大核心功能深度解析

Cascade：你的 AI 编程搭档

Windsurf 的灵魂就是 Cascade。它不是一个聊天机器人，而是一个能动手干活的 Agent——读代码、改文件、跑命令、看报错、修 bug，全自动。

Code 模式 vs Chat 模式

Cascade 有两个模式，切换按钮在面板顶部：

模式	用途	会改文件吗
Code	写功能、改 bug、重构	会，直接修改代码
Chat	问问题、讨论架构、学习	不会，只给建议

日常开发 90% 的时间用 Code 模式。想讨论"这个 API 该怎么设计"或"这段代码为什么慢"时切 Chat 模式。

规划 + Flow State

给 Cascade 一个复杂任务，它会先生成执行计划。后台有 planning agent 优化长期方案，前台模型执行当前步骤：

```
// "给 Express API 加 JWT 认证" → Cascade 自动规划:
// Step 1: npm install jsonwebtoken bcryptjs
// Step 2: 创建 src/middleware/auth.ts
// Step 3: 创建 src/routes/auth.ts
// Step 4: 给现有路由加 authMiddleware
// Step 5: .env.example 添加 JWT_SECRET
```

每步都有 diff 预览和 checkpoint，搞砸了一键回滚。

Flow State 是 Windsurf 独有能力：Cascade 持续追踪你的编辑、终端命令、剪贴板内容。你直接说"修一下刚才那个报错"就行，不用复制粘贴错误信息。

实战：用 Cascade 重构 Express 认证 Middleware

这是最能体现 Cascade 多文件能力的场景：项目早期偷懒，把 token 校验逻辑直接写在每个路由里，后来发现要改逻辑时要改七八个地方。告诉 Cascade：

> "把散落在各路由里的 JWT 验证逻辑提取到 `src/middleware/auth.ts`，所有用到的路由改成 `router.use(authMiddleware)`。"

Cascade 的实际执行步骤：

第一步，它先扫描整个 `src/routes/` 目录，找出所有包含 `jwt.verify` 的文件，列出来让你确认范围。

第二步，新建 `src/middleware/auth.ts`：

```ts
// src/middleware/auth.ts
import { Request, Response, NextFunction } from 'express';
import jwt from 'jsonwebtoken';

export interface AuthRequest extends Request {
  user?: { id: string; role: string };
}

export function authMiddleware(
  req: AuthRequest,
  res: Response,
  next: NextFunction
) {
  const token = req.headers.authorization?.split(' ')[1];
  if (!token) {
    return res.status(401).json({ error: 'No token provided' });
  }
  try {
    const payload = jwt.verify(token, process.env.JWT_SECRET!) as {
      id: string;
      role: string;
    };
    req.user = payload;
    next();
  } catch {
    return res.status(401).json({ error: 'Invalid token' });
  }
}
```

第三步，同时修改 `src/routes/users.ts`、`src/routes/orders.ts`、`src/routes/admin.ts` 等七个文件——把各自内嵌的 `jwt.verify` 块删掉，在 router 顶部插入 `router.use(authMiddleware)`。整个过程 Cascade 自动处理，每个文件都生成 diff 预览。

第四步，跑 `npm run test`，有一个测试失败：测试里发的请求没带 token。Cascade 看到终端报错，直接定位到 `src/__tests__/orders.test.ts`，在 `beforeAll` 里加了生成测试 token 并注入 `Authorization` header 的逻辑，不用你动手。

这就是 Cascade 多文件感知的价值——七个路由文件 + 一个新 middleware + 一个测试文件，一次对话搞定，且改动是一致的，不会出现只改了五个路由漏掉两个的低级错误。

你发现一个细节：之前有的路由用 `req.userId`，有的用 `req.user.id`，现在统一了。Cascade 在改完后会主动提示："发现 3 处旧的 `req.userId` 引用，已一并替换为 `req.user?.id`。"这就是 Flow State 的作用——它记住了你在这个对话里的意图，不只是机械执行字面指令。

Supercomplete：不花钱的自动补全

Supercomplete 是 Windsurf 的 Tab 补全功能，**免费版也不限量**。它比普通补全更聪明，能根据你最近的编辑模式预测下一步操作。

举个例子，你刚在一个文件里把 `console.log` 换成了 `logger.info`，切到另一个文件按 Tab，Supercomplete 会自动建议同样的替换。

```
# 你在写一个 FastAPI 路由，刚敲完函数签名：
@app.get("/users/{user_id}")
async def get_user(user_id: int):
    # 按 Tab，Supercomplete 会补全：
    user = await db.users.find_one({"_id": user_id})
    if not user:
        raise HTTPException(status_code=404, detail="User not found")
    return user
```

和 GitHub Copilot 的区别：Supercomplete 对你当前项目的上下文理解更准，因为它基于 Windsurf 的索引引擎，了解你整个代码库的模式。但注意：**装了 Copilot 或 Tabnine 插件的话要先禁用**，否则两个补全会打架。

Supercomplete 在重复性改动时特别省力。比如你在给 API 加统一的错误处理，第一个路由手动写完之后，后面每个路由按一下 Tab 就补全了相同的 `try/catch` 结构——它认出了你的模式，不需要你说任何话。

Web Preview：在 IDE 里看效果

做前端开发时，Cascade 面板右上角有一个 Web Preview 按钮。点开后会在 IDE 内嵌一个浏览器窗口，实时显示你的网页。

最好用的地方：你可以直接**点击预览里的元素**，Cascade 会定位到对应代码。然后说"把这个按钮改成蓝色圆角"，它就能精准修改——因为它知道你点的是哪个 DOM 元素。

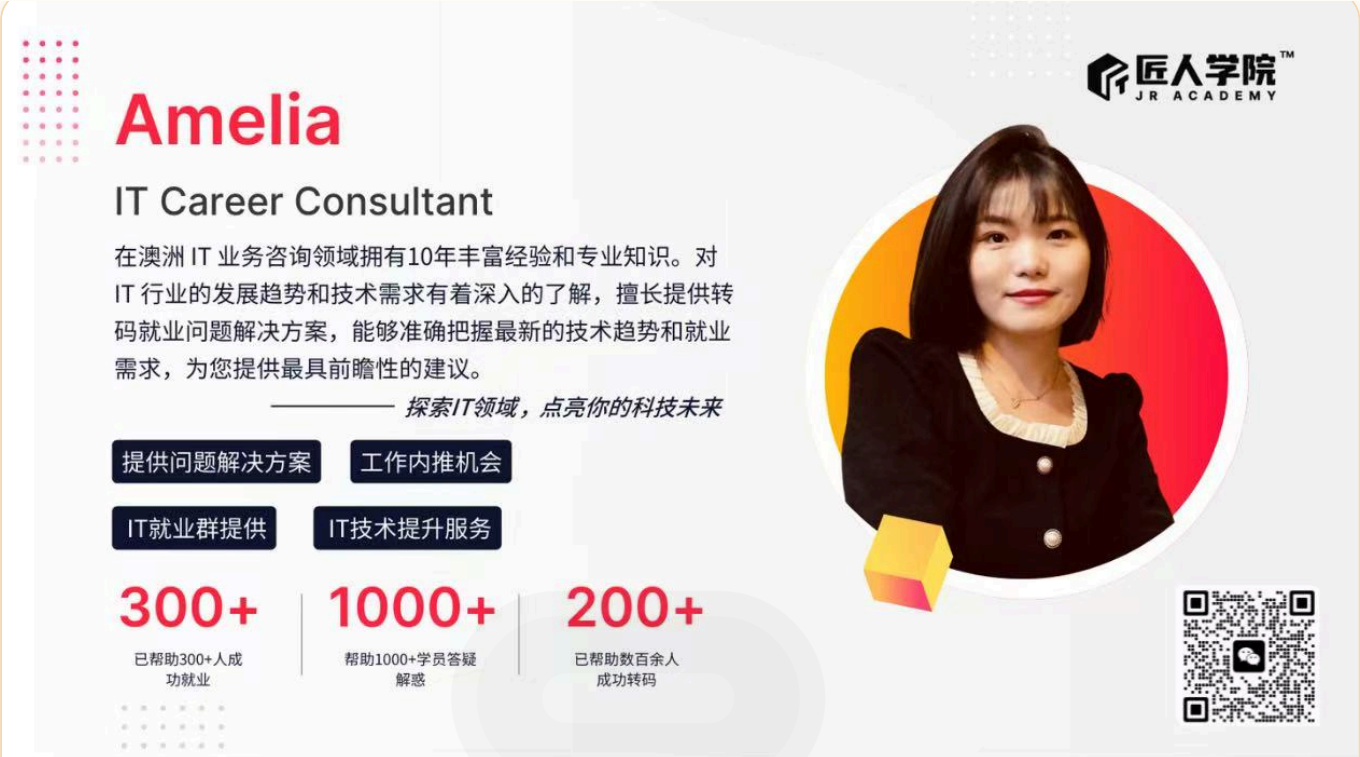
这个工作流比"改代码 → 切浏览器 → 刷新 → 切回来"高效太多，特别是调 CSS 的时候。Web Preview 适合轻量前端项目（Vite / Next.js dev server），重型应用建议还是开独立浏览器，IDE 内嵌渲染性能有上限。

三个功能的协作关系

Cascade、Supercomplete、Web Preview 不是孤立的三个工具，日常开发里会同时用到：

- 写新功能时：用 Cascade Code 模式起步，拿到可跑的骨架
- 补细节时：Supercomplete 接管，Tab 键飞速填充重复逻辑
- 前端调样式时：Web Preview 点哪改哪，Cascade 精准定位代码

理解这三个功能的分工，是从"偶尔用 AI 辅助"升级到"以 AI 为主力"的关键一步。

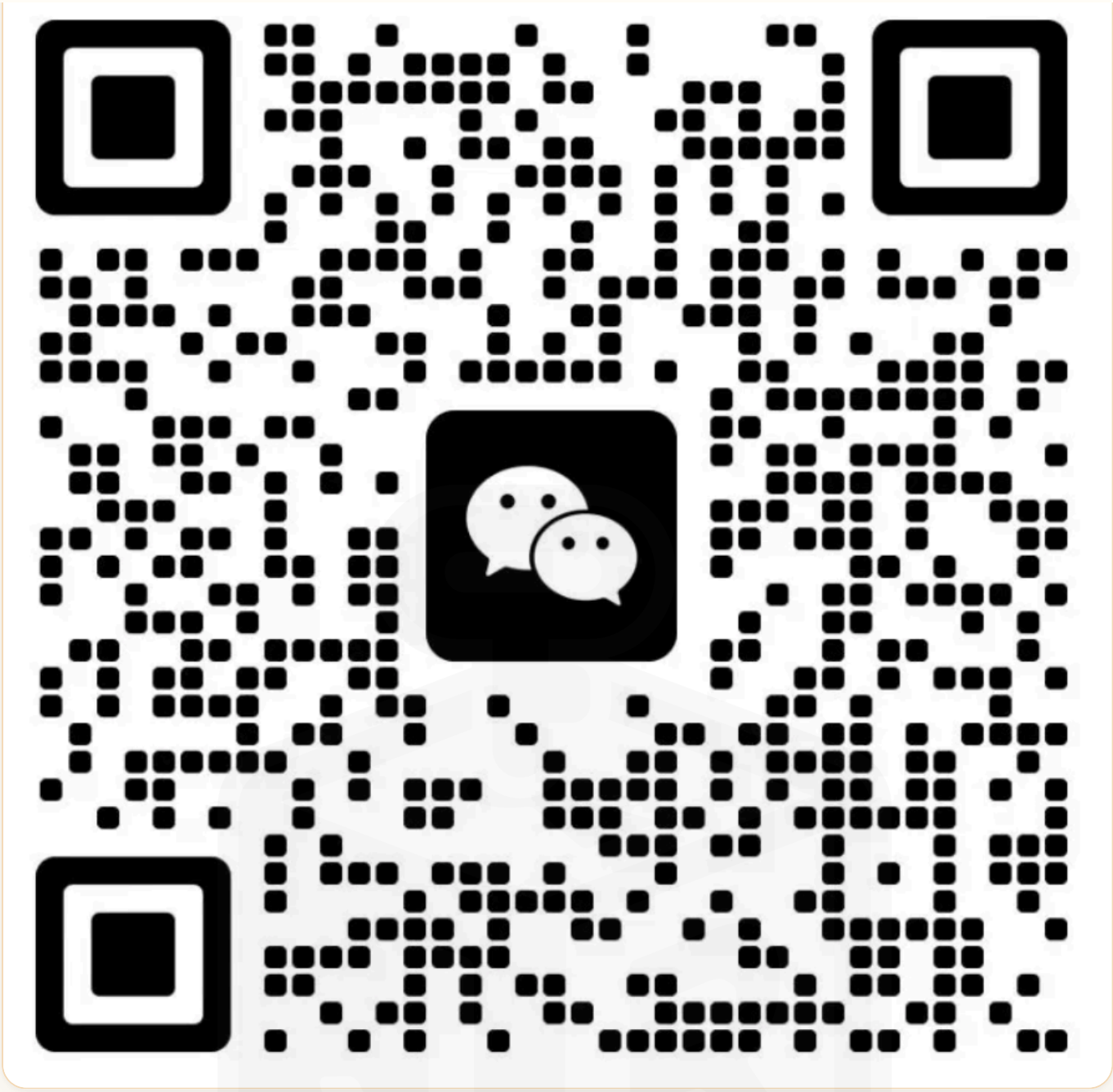


Amelia

匠人学院 IT 顾问

装好了还不知道怎么用 Cascade？或者想知道哪个 AI IDE 最适合你现在的项目？扫码加我，15 分钟聊清楚。

jiangren.com.au



CHAPTER 04

⚙️ 进阶技巧与实战配置

.windsurfrules：让 AI 按你的规矩写代码

Windsurf 最高 ROI 的配置就是 `.windsurfrules` 文件。在项目根目录创建它，Cascade 每次生成代码都会遵守里面的规则。

规则文件有两个位置：

- **全局规则** — 存在 `~/global_rules.md`，对所有项目生效，适合个人习惯（如"永远写中文注释"）
- **项目规则** — 存在项目根目录的 `.windsurfrules`，只对当前项目生效，适合团队约定

两者上限各 6000 字符，合计 12000 字符。超了要截断。

几个要点：

- **规则改了之后重启 Windsurf** 才生效，不重启改了等于没改
- 新版 Windsurf 也支持 `.windsurf/rules/rules.md` 格式，两者同时存在时新格式优先
- 规则文件可以 commit 进仓库，团队所有人都共享同一套约束

最常见的失效写法是"写干净的代码"这类空洞指令——AI 看不懂"干净"的边界，照样乱写。有效规则必须是**可执行的约束**，比如"不用 any 类型"、"所有 API 调用用 TanStack Query 封装"。

前端项目 .windsurfrules 完整范例

以下是一个 Next.js 14+ App Router 项目的真实规则文件。按照类别分块，每块职责清晰：

```
# .windsurfrules — Next.js 前端项目

## 技术栈约定
- Framework: Next.js 15 App Router，不用 Pages Router
- 语言: TypeScript 5.x，strictMode 开启，禁止 any
- 样式: Tailwind CSS 4 + CSS Modules（复杂动画）
- 组件库: shadcn/ui，不要引入其他 UI 库
- 状态: Zustand（全局）+ React Context（局部）
- 数据获取: TanStack Query v5，禁止在 component 里裸写 useEffect + fetch
- 表单: React Hook Form + Zod schema 校验
- 图片: 只用 next/image，不用 <img>

## 文件命名
- React 组件文件: PascalCase（UserCard.tsx）
- 普通 util 文件: kebab-case（format-date.ts）
- 目录名: kebab-case
- 测试文件: *.test.tsx 放在组件同目录

## 组件规范
- 全用函数式组件，禁止 class component
- 用具名导出（export function），不用 default export
- props 类型用 interface 定义，放在组件文件顶部
- 超过 200 行的组件必须拆分

## API 调用规范
- API base URL 从 process.env.NEXT_PUBLIC_API_URL 读取，禁止硬编码
- 所有 HTTP 请求封装在 /lib/api/ 目录，不要在 component 文件里直接调用 fetch
- 错误边界用 React Error Boundary 处理，不要在每个 component 写 try-catch

## 禁止事项
- 禁止 CSS-in-JS（styled-components / emotion）
- 禁止在 Server Component 里使用 useState / useEffect
- 禁止把 API key 写进前端代码
- 禁止用 lodash（用原生 ES2024 代替）
```

这份规则覆盖了新人最容易犯的 10+ 个错误——技术栈选型、命名、组件粒度、API 封装、环境变量管理。Cascade 拿到这份规则后，生成的代码基本可以直接过 code review。

后端项目 .windsurfrules 完整范例

Python FastAPI 项目的规则风格略有不同，更侧重类型标注和接口设计：

```
# .windsurfrules — FastAPI 后端项目

## 技术栈
- Python 3.12+，强制类型标注
- 框架：FastAPI 0.115+，不用 Flask
- ORM：SQLAlchemy 2.x async，不用同步 session
- 数据校验：Pydantic v2
- DB：PostgreSQL 17，连接池用 asyncpg
- 认证：JWT（jose 库），不用 Session
- 测试：pytest + httpx.AsyncClient

## 项目结构
- app/routers/  — 路由层，只处理 HTTP 逻辑
- app/services/ — 业务逻辑，不直接操作 DB
- app/repositories/ — 数据访问层，唯一直接写 SQL 的地方
- app/schemas/  — Pydantic 输入输出 schema

## 接口规范
- 所有端点必须写 OpenAPI 注释（summary + description）
- 响应 schema 用 response_model 标注
- 错误统一用 HTTPException，不要直接 raise 500
- 路由命名用 kebab-case（/user-profiles，不用 /userProfiles）

## 安全
- 禁止把密钥写进代码或 .env 提交到仓库
- 所有用户输入通过 Pydantic 校验，不做手动 sanitize
- SQL 参数化查询，绝不拼接字符串

## 异步规范
- DB 操作全部 async/await，禁止混用同步 session
- background task 用 FastAPI BackgroundTasks，不要自己起线程
```

这套规则配合 PostgreSQL MCP Server（后面会讲），Cascade 可以读取数据库 schema 后直接生成 CRUD 接口，不需要你手动告诉它表结构。

规则写法：有效 vs. 无效

写规则的常见坑是指令太模糊，AI 不知道该怎么执行：

无效写法	有效写法
写干净的代码	函数不超过 40 行，超过就拆
注意安全性	所有用户输入必须过 Zod/Pydantic 校验
合理处理错误	用 Result 类型，不用 try-catch 直接 throw
保持代码可读	变量名不用缩写（dt → dateTime，usr → user）
用好 TypeScript	禁止 any，禁止 as 类型断言，禁止 @ts-ignore

规则文件不是越长越好，超过 4000 字符后规则之间开始互相干扰。实测下来，**50 条清晰的具体规则**比 200 条模糊描述效果好得多。

Cascade Memory：跨会话持久记忆

Rules 是你手动写的约束，Memory 是 Cascade 自动学习并记下来的东西。

每次会话结束后，Cascade 会把它认为"有用的上下文"提取成记忆条目，存在本地文件：

```
~/.codeium/windsurf/memories/
├── {workspace-hash}/
│   ├── memories.json
│   └── ...
```

下次打开同一个工作区，Cascade 会自动加载这些记忆，不需要你重新解释项目背景。

Cascade 会自动记住什么

- 你纠正过的错误（"你之前用 axios，我们这个项目用 ky"）
- 项目里不在代码里但重要的约定（"部署目标是 AWS Lambda，不是 EC2"）
- 你明确要求记住的信息

手动创建记忆

不需要等 Cascade 自动识别，你可以直接告诉它：

```
记住：我们的后端跑在 Kubernetes 上，端口配置在 k8s/config.yaml，不要在代码里硬编码端口号
```

Cascade 收到指令后会创建记忆条目。你也可以在 Cascade 面板里管理已有记忆，看到它记了什么，随时删改。

记忆不会跨工作区共享

这是个容易踩的坑：在项目 A 里告诉 Cascade "我们用 pnpm 不用 npm"，切换到项目 B 后 Cascade 不会知道这件事。全局习惯用 `global_rules.md` 写，项目约定用 `.windsurfrules` 写，临时上下文才用 Memory。

Memory vs. Rules：怎么选

特征	Memory	Rules
谁写的	Cascade 自动 / 你口头要求	你手动维护
持久性	本地，不提交 git	文件，可提交
团队共享	❌ 仅本机	✅ 提交后团队共享
适合场景	临时上下文、当前任务进展	代码规范、技术栈约定
可控性	低（AI 自动总结）	高（你写什么就是什么）

结论：**重要的团队约定写进 `.windsurfrules` 提交 git**，不要依赖 Memory。Memory 只用于会话内的临时状态。

MCP Server：给 Cascade 接外部能力

MCP（Model Context Protocol）让 Cascade 能调用外部工具。Windsurf 内置了 MCP Marketplace，一键就能装。

最实用的几个 MCP Server：

MCP Server	能干什么
Figma	读取 Figma 设计稿，直接生成对应 UI 代码
PostgreSQL	查询数据库 schema，生成 SQL，验证查询结果
Playwright	跑端到端测试，截图验证 UI
Slack	在 IDE 里读/发 Slack 消息

配置方式：打开 Cascade 面板右上角的 MCP 图标，或者手动编辑配置文件：

```
// mcp_config.json
{
  "mcpServers": {
    "postgres": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-postgres"],
      "env": {
        "POSTGRES_CONNECTION_STRING": "postgresql://user:pass@localhost:5432/mydb"
      }
    },
    "figma": {
      "command": "npx",
      "args": ["-y", "@anthropic-ai/mcp-server-figma"],
      "env": {
        "FIGMA_ACCESS_TOKEN": "your-token-here"
      }
    }
  }
}
```

配好 Figma MCP 后，跟 Cascade 说"按照 Figma 里的 Login Page 写代码"，它会自己读设计稿、提取布局颜色、生成组件。

配好 PostgreSQL MCP 后，Cascade 能主动查 schema，你只需要说"给 users 表加一个 email 唯一索引"，它知道当前表长什么样，直接生成正确的 migration。

模型选择与 Credit 管理

Pro 用户可在 Cascade 面板底部切换模型：

```
# 日常写代码 → 默认模型，速度快省 credit
# 复杂架构 → Claude Sonnet，推理强
# 快速原型 → GPT-4o，响应最快
```

省 credit 四招：

1. **大任务先 Chat 讨论再 Code 执行** — Chat 模式消耗的 credit 比 Agent 模式低很多，先想清楚再动手
2. **拆分任务粒度** — 一次做一个小改动，比让 Cascade 自己想"下一步"省很多 token
3. **用 Inline Edit（`Ctrl+I`）** — 只改当前光标位置的代码，不启动完整 Cascade 流程
4. **能 Tab 补全解决的别开 Cascade** — Copilot 级别的自动补全不消耗 Cascade credit

另外，如果你用公司账号有 Team 计划，Credit 是按 seat 分配的。把 `.windsurfrules` 写好，减少 Cascade 来回确认的次数，是最实际的降本手段。

实战技巧汇总

走完上面的配置，几个高频操作值得单独记：

快速生成 Rules 文件：直接跟 Cascade 说"根据这个项目的代码，帮我生成一份 `.windsurfrules`"，它会扫描现有文件总结规律，你再审核调整比从零写快很多。

调试 Rules 是否生效：在 Cascade 里问"你对这个项目有哪些规则约束"，如果它能复述你 `.windsurfrules` 的主要内容，说明规则已被加载。

团队共享记忆：Memory 本身不能共享，但可以把重要的记忆内容手动转写成 `.windsurfrules` 条目，提交到仓库。或者维护一份 `AGENTS.md`，描述 AI 在这个项目里应该知道的背景信息。

MCP 连不上的排查：先在终端手动跑 MCP server 命令确认没有依赖缺失，再检查 Windsurf 的 MCP 配置文件路径是否正确——Mac 上在 `~/.codeium/windsurf/mcp_config.json`，Windows 在 `%APPDATA%\Codeium\windsurf\mcp_config.json`。

CHAPTER 05

❓ 常见问题、定价与选型建议

常见问题

Cascade 转圈不动怎么办?

Windsurf 里最常见的问题。通常是这几个原因:

1. **免费额度用完了**——检查右下角的 credit 余额
2. **请求超时**——大项目里上下文太长，Cascade 处理不过来。缩小任务范围，比如指定"只改 `src/components/Header.tsx`"
3. **网络问题**——Cascade 需要稳定的网络连接。如果你在国内，考虑配代理
4. **索引没跑完**——右下角有索引进度条，等它跑完再操作

```
# 如果 Cascade 彻底卡死，终极解决方案:
# 1. Ctrl+Shift+P 打开命令面板
# 2. 搜索 "Windsurf: Clear Cascade History"
# 3. 重启 Windsurf
# 4. 等索引完成后再试
```

Cascade 改错了代码怎么办?

别慌。Windsurf 有 checkpoint 机制，每次 Cascade 修改代码前都会自动保存一个快照。

在 Cascade 面板里找到对应的操作步骤，点 **Revert** 就能回滚到那一步之前的状态。这比手动 `git stash` 或 `Ctrl+Z` 可靠得多。

更稳妥的做法是在做大改动前先提交一次 git:

```
git add -A && git commit -m "wip: before cascade refactor"
```

这样即便 checkpoint 也不能救你，`git reset --hard HEAD` 一条命令回到原点。

插件兼容性 & 中文支持

兼容绝大部分 VS Code 插件，但 **Copilot/Tabnine 必须禁用**（和 Supercomplete 冲突），Remote SSH 不支持。Cascade 完全支持中文对话、中文注释，日常中文指令 + 英文代码体验流畅。

Windsurf Memories 是什么?

Memories 是 Windsurf 2025 年下半年加入的长期记忆功能——Cascade 会从对话里自动提取你的偏好（比如"用 TypeScript strict 模式"、"测试框架用 Vitest"），存入一个持久化的文本文件，下次打开新对话时自动载入。

你可以在 Settings → Cascade → Memories 里手动编辑或删除记忆条目。建议定期审查一下，防止旧的错误偏好影响新项目。

```
# 典型的 Memories 条目示例
- Always use pnpm, never npm or yarn
- Prefer async/await over .then() chains
- Database: PostgreSQL with Drizzle ORM
- Test framework: Vitest + Testing Library
```

为什么 Cascade 没有用我选的模型?

免费版只能用基础模型，选 Claude Sonnet/GPT-4o 不会生效。另一个原因是该模型当前处于高负载——Windsurf 会自动降级到可用模型。如果你付了 Pro，可以在设置里把首选模型固定住，避免自动降级。

一个"交互（interaction）"到底消耗多少?

2026 年 3 月切换到 quota 制之后，计量单位从 credit 变成了 interaction。规则是：**一次 Cascade 提问 = 一次 interaction**，不管 Cascade 在背后读了多少文件、改了多少行代码。Tab 补全不消耗 interaction，完全免费。

所以如果你的问题描述得越清楚，Cascade 一次搞定，就越省 interaction。拆成 5 次小问题反而消耗更多。

定价详解（2026 年）

当前套餐

方案	月费	AI 交互额度	Tab 补全	可用模型
Free	$0	5 次/天	无限	基础模型
Pro	$20	50 次/天	无限	Claude 3.7 / GPT-4o / Gemini 1.5 Pro
Pro Plus	$35	250 次/天	无限	旗舰模型优先排队
Teams	$30/人	50 次/人/天	无限	全部模型 + 管理后台
Enterprise	定制	定制	无限	全部模型 + SSO + 审计日志

> **注意**：2026 年 3 月 19 日，Windsurf 把 Pro 从 $15 涨到 $20，并将计费方式从 credit 池改成每日 quota。老用户在取消或切换套餐前仍保留旧制。

额度用完了怎么办?

每天额度在 UTC 午夜重置。如果当天用完了，有两个选项：

1. **等第二天重置**——对大多数人够用
2. **购买 Pro Plus**——每天 250 次，重度用户的选择

对比一下：Cursor Pro 同样是 $20/月，但用的是 credit 池而不是每日 quota，重度 Agent 使用者可能一周内就打光 credit，然后要按使用量付费。Windsurf 的每日 quota 对突发性高峰更友好——比如某天密集重构，明天又恢复正常。

Windsurf vs Cursor vs Claude Code 深度选型对比

这三款工具经常被放一起比，但它们的定位其实不完全重叠。

核心维度对比

维度	Windsurf	Cursor	Claude Code
形态	IDE（VS Code fork）	IDE（VS Code fork）	终端 CLI
月费	$20（Pro）	$20（Pro）	API 计费，Max plan $100
Tab 补全	无限（含 Free）	无限（含 Free）	无
Agent 额度	50 次/天（Pro）	credit 池（$20/月约 500 次，重度下降快）	按 token 计费，无次数上限
上下文处理	Flow State 自动追踪多文件	Composer 手动添加上下文	整个项目 git tree 实时可见
规则文件	`.windsurfrules`	`.cursorrules` / `.mdc`	`CLAUDE.md`
内置 Web Preview	✅	❌（需外部浏览器）	❌
离线使用	❌ 需要联网	❌ 需要联网	❌ 需要联网
Git 操作	基础 diff / checkpoint	基础 diff	完整 git 操作（commit/PR/rebase）
多文件重构	Cascade 自动识别影响范围	Composer 需手动指定文件	精准，但需明确指令
学习曲线	低（IDE 习惯即可）	低（VS Code 用户无缝切换）	中（需熟悉 CLI 工作流）
中文支持	✅ 对话/注释	✅ 对话/注释	✅ 完整中文

按场景选

写新功能、日常开发：Windsurf 或 Cursor 都行。区别在于 Windsurf 的 Tab 补全在 Free 版就是无限，Cursor 的 Tab 补全在超出阈值后也会限速。如果你的主要需求是补全 + 偶尔问 AI，Windsurf Free 就够。

大型项目重构、跨多个文件改动：Windsurf Cascade 的 Flow State 胜出——它会主动追踪哪些文件和你的改动相关，无需手动 `@file` 添加。Cursor Composer 在项目大了之后需要频繁手动指定上下文，费力气。

精确控制、脚本化任务：Claude Code。它是 CLI，可以嵌入 CI 流程、shell 脚本，做自动化；而且整个项目的 git 历史对它都是透明的，不需要手动喂上下文。Cascade 再聪明，也是个 GUI，没法被脚本调用。

预算敏感：

- 只写前端/全栈、每天有几十次 AI 对话 → Windsurf Pro $20
- 偶尔用 AI 辅助、主要靠 Tab 补全 → Windsurf Free，完全够
- 需要处理复杂架构决策或长篇重构 → Claude Code Max plan $100，按月付，不用担心 token 超支

团队场景：Windsurf Teams $30/人，有集中管理后台、使用量统计、billing 控制。Cursor Business $40/人，功能类似但贵 $10。Claude Code 团队版需要用 Anthropic API，按实际消耗计费，适合用量不均衡的团队。

价格综合对比

```
个人开发者（中等使用量）：
  Windsurf Pro:   $20/月  →  50 次/天 Agent + 无限 Tab
  Cursor Pro:     $20/月  →  credit 池（重度 Agent 可能不够）
  Claude Code:    $20/月  →  Claude Pro API，轻量任务够用

个人开发者（重度 Agent）：
  Windsurf Pro Plus:  $35/月  →  250 次/天
  Cursor Pro:         $20+超量费
  Claude Code Max:    $100/月 →  token 无限

5 人团队：
  Windsurf Teams:  $150/月 ($30×5)
  Cursor Business: $200/月 ($40×5)
  Claude Code API: 视用量，$100–$300/月
```

我的实战建议

大多数全栈开发者用下来，最有效率的组合是：**Windsurf（日常写代码） + Claude Code（大型重构/自动化任务）**。

Windsurf 处理 80% 的时间——打开项目，Tab 补全加速，遇到需要多文件改的功能让 Cascade 跑一遍，速度快、上下文不用手动管。

Claude Code 处理剩下 20% 的硬骨头——需要读整个 git 历史的 bug 追查、需要脚本化的批量任务、需要精确控制每一步的重构。

Cursor 作为第三选择，在你已经熟悉 `.cursorrules` 生态且团队里有大量 Cursor 配置积累时，换过来的迁移成本太高，就留在 Cursor 里。新手从 Windsurf 进入比 Cursor 更平滑——免费版就给无限 Tab 补全，不用一开始就想清楚要不要付费。

最后一个选型原则：**不要在同一台机器上同时运行 Windsurf + Cursor**，索引服务会互抢资源，两个都变慢。选一个主力 IDE，另一个按需打开。

CHAPTER 06

🪲 用 Cascade 调试和修复 Bug

调试是 Cascade 最能体现 Agent 能力的场景。传统调试你得自己看报错 → 猜原因 → 找代码 → 试修复 → 再跑。Cascade 能一条龙把这个循环自动化。

报错直接甩给 Cascade

终端里跑 `npm run dev` 爆红了，不用复制粘贴报错信息。Windsurf 的 Flow State 已经捕获了终端输出，直接在 Cascade 面板里说：

```
修一下刚才终端里的报错
```

Cascade 会自动读取报错内容，定位到出错的文件和行号，分析根因，然后给出修复。整个过程你不需要提供任何额外上下文。

如果你想更精确地定位问题：

```
终端里报了 TypeError: Cannot read properties of undefined (reading 'map')
看一下 src/components/UserList.tsx 里的数据流，找到 undefined 是哪来的
```

Cascade 会从组件出发，往上追溯 props 传递链、API 调用、状态管理，找到数据在哪一步变成了 `undefined`。

跨文件追踪 Bug

真实项目里的 bug 很少只涉及一个文件。比如"用户登录后跳转到空白页"，问题可能出在路由配置、Auth 状态管理、或者 API 返回的 token 格式。

给 Cascade 的 prompt 这么写：

```
用户登录成功后跳转到 /dashboard 显示空白页。
请检查以下链路：
1. 登录 API 的 response 格式
2. token 存储逻辑（localStorage 还是 cookie）
3. 路由守卫的鉴权判断
4. Dashboard 组件的数据加载
```

Cascade 会按链路逐个检查，跨 5–10 个文件追踪数据流向，最后定位到具体问题。

用 Chat 模式做诊断

不确定 bug 在哪时，先用 Chat 模式（不改文件）做诊断：

```
Chat 模式下的 prompt：
"这个项目的 API 请求为什么有时会返回 401？
帮我分析 src/lib/api.ts 里的 token 刷新逻辑有没有竞态条件"
```

Chat 模式只分析不动代码，适合先搞清楚问题再决定怎么改。确认诊断结果后切回 Code 模式让 Cascade 执行修复。

TypeScript 类型错误批量修复

项目里一堆 TypeScript 类型错误是最适合 Cascade 的场景之一：

```
# 先让 TypeScript 编译器输出所有错误
npx tsc --noEmit 2>&1 | head -50
```

然后告诉 Cascade：

```
跑 tsc --noEmit 有 23 个类型错误，帮我逐个修复。
优先处理 src/types/ 目录下的类型定义问题，
然后再修使用侧的类型不匹配。
```

Cascade 会按依赖顺序修复：先修类型定义，再修使用端，避免改了 A 导致 B 报新错的连锁反应。

完整 Bug Fix 走查：真实 case

下面是一个真实场景的完整调试过程，展示 Cascade 是怎么一步步推进的。

场景：Next.js 项目，用户偶发性看到"购物车商品突然清空"。复现概率大约 20%，纯前端行为，后端日志干净。

第一步：告诉 Cascade 现象和怀疑范围

```
购物车偶发清空，概率约 20%，后端日志没有异常，怀疑是前端状态管理问题。
项目用 Zustand 管购物车状态，持久化到 localStorage。
请检查 src/store/cartStore.ts 和 src/hooks/useCart.ts，
看看有没有竞态或者 hydration 时机问题。
```

Cascade 读完两个文件后发现：`useCart` 在组件 mount 时调用了 `initCart()`，而 `initCart()` 会从 localStorage 读数据覆盖当前 state——如果用户在快速切换页面时，两个实例同时 mount 就会互相覆盖。

第二步：让 Cascade 验证推断

```
你的推断是 initCart 被并发调用，能帮我在 cartStore.ts 里
加一个 initialized 标志位，确认一下 initCart 有没有重入问题？
先加 console.log 跑一遍，别直接改逻辑。
```

Cascade 在 `initCart` 前后加了 `console.log('[cart] init start')` 和 `console.log('[cart] init end')`，然后指导你复现：快速点击导航 3 次，观察控制台。结果出现了 `init start → init start → init end → init end`，证实了重入。

第三步：让 Cascade 修复

```
确认是重入问题了，帮我修：
- 加 initialized 标志位，第二次调用直接 return
- 修复后跑一遍 tsc --noEmit 确认没有类型问题
```

```
// cartStore.ts 修复后的关键片段
let initialized = false;

const initCart = () => {
  if (initialized) return;
  initialized = true;

  const stored = localStorage.getItem('cart');
  if (stored) {
    try {
      useCartStore.setState({ items: JSON.parse(stored) });
    } catch {
      localStorage.removeItem('cart');
    }
  }
};
```

第四步：回归验证

```
修完了，帮我写一个快速 smoke test：
模拟 initCart 被连续调用 3 次，验证 state 只被设置一次
```

Cascade 写出了对应的 Vitest 测试用例，跑通后整个 debug 流程结束。

从发现问题到修复 + 测试，全程大约 15 分钟，期间你主要在引导和验证，Cascade 在跨文件读代码和修改。

高效调试 Prompt 模板

下面 5 个模板可以直接复制使用，按场景套。

模板 1：异步 / 竞态条件

```
[现象] {描述偶发性 / 顺序相关的 bug}
[怀疑] 可能是 {Promise 竞态 / useEffect 依赖 / 事件监听未清理}
请检查 {文件路径} 里的异步逻辑，
重点看：1) Promise 是否有 race condition；2) 组件卸载时是否 cleanup；
3) 订阅是否幂等。
先 Chat 模式分析，确认后再改代码。
```

模板 2：性能回归定位

```
[现象] {页面/接口} 比上周慢了 {X} 倍，CI benchmark 报告附在下面。
[已知] 最近合并的 PR 是 {PR 列表}。
帮我分析 git diff {commit A}..{commit B} 里有没有明显的性能问题：
- N+1 查询
- 同步阻塞操作移到了热路径
- 缓存 key 变化导致命中率下降
输出可疑点，我逐个确认。
```

模板 3：网络请求失败 / 4xx 5xx

```
POST {路径} 返回 {状态码}，服务端日志如下：
---
{粘贴日志}
---
请检查 {客户端 fetch 文件} 和 {服务端 handler 文件}，
对比请求参数格式（Content-Type / body schema / auth header），
找出客户端发送的和服务端期望的有什么不一致。
```

模板 4：React 渲染异常（闪烁 / 无限重渲染 / hydration mismatch）

```
组件 {ComponentName} 出现 {闪烁/无限重渲染/hydration error}。
React DevTools 显示 {Profiler 截图描述 / 控制台报错}。
请检查：
1. useEffect 依赖数组是否完整
2. state 更新是否在渲染期间触发（setState in render）
3. SSR/CSR 的初始 state 是否一致
先给我分析，不要动代码。
```

模板 5：数据库慢查询

```
{接口路径} P99 延迟 {Xms}，EXPLAIN ANALYZE 输出如下：
---
{粘贴 EXPLAIN 结果}
---
ORM 代码在 {文件路径}。
帮我分析：
1. 缺少哪些索引
2. ORM 生成的 SQL 有没有 N+1
3. 能否加 select 字段裁剪 payload
给出具体的 SQL 和 ORM 改法。
```

调试实战技巧

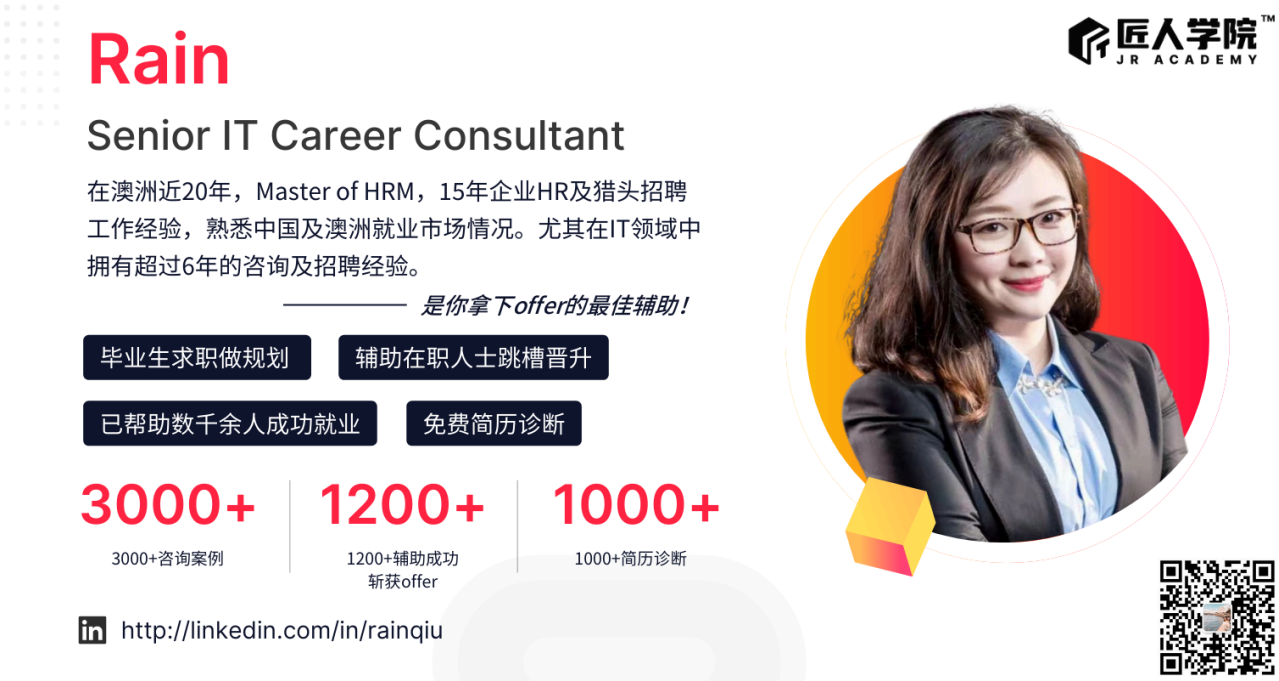
给 Cascade 看日志，而不是只描述现象。"接口报错了"不如"POST /api/users 返回 500，日志里显示 `UNIQUE constraint failed: users.email`"——后者 Cascade 能直接定位到数据库 schema 和插入逻辑。

用 Revert 大胆试。Cascade 每次改代码都有 checkpoint，修错了一键回滚。所以不要犹豫"要不要让它试试"，试完不行就 Revert，比你自己手动 debug 快得多。

```
// 一个通用的调试 prompt 结构：
// "这段代码的预期行为是 X，实际行为是 Y。
//  相关日志/报错是 Z。
//  请定位根因并修复。"
//
// 越具体，Cascade 越准。
```

搭配 `console.log` 定位。有时候 Cascade 的推理不够准确，你可以让它先加几个 `console.log` 跑一遍看输出，然后基于实际输出再修复——这比猜测更可靠。

让 Cascade 写验证用例。修完 bug 后，顺手让 Cascade 把刚才的 fix 逻辑写成单测或 E2E 测试步骤，防止回归。这一步很多人省掉，结果同样的 bug 两周后又出现。

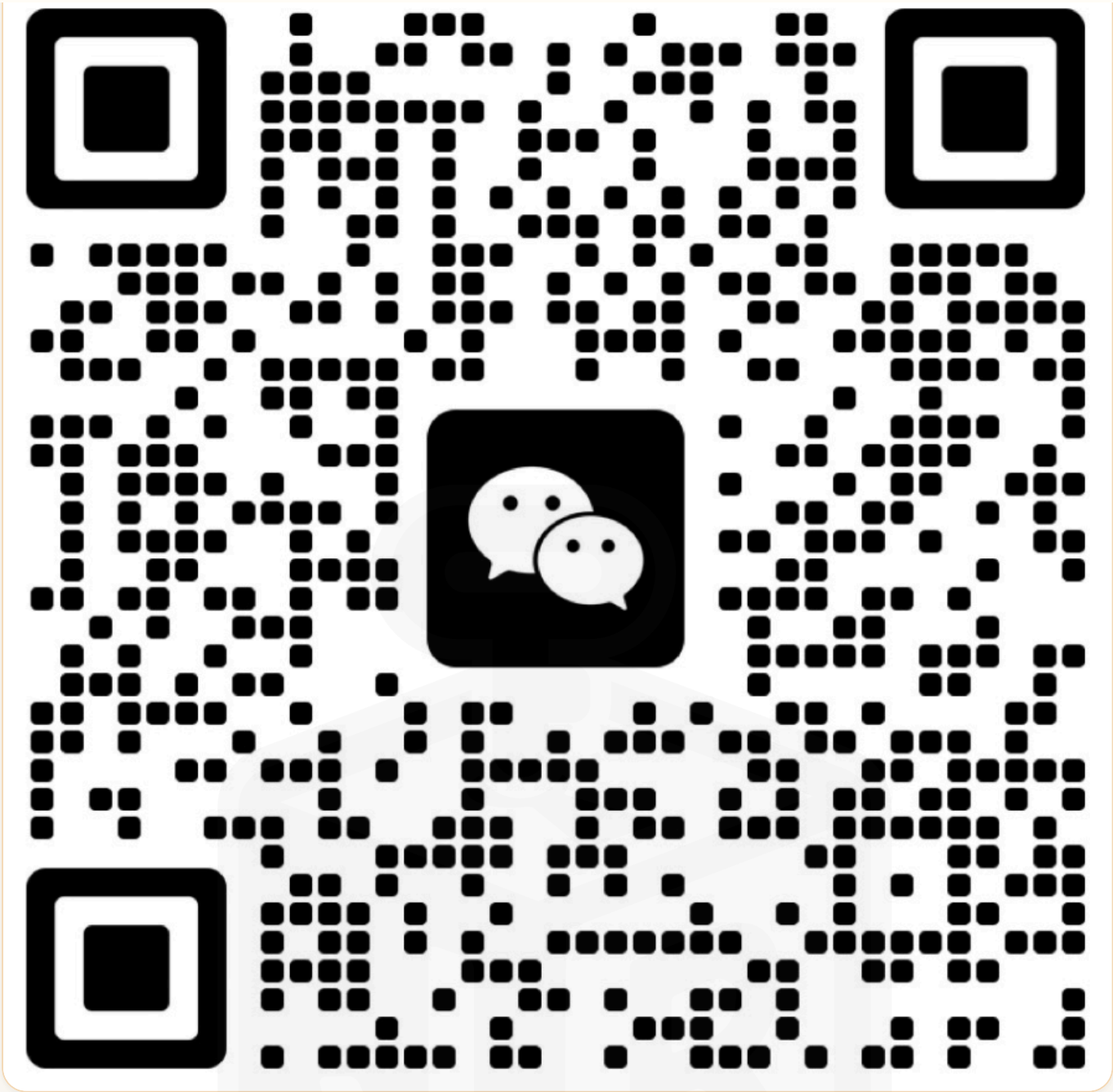


Rain

匠人学院课程顾问

调试这关过了，下一步是系统学 AI 开发？我们的 AI 工程师课程从 Vibe Coding 到生产部署都有，扫码了解。

jiangren.com.au



CHAPTER 07

🔀 Git 集成与团队协作

Windsurf 在 VS Code 的 Git 基础上加了一层 AI 能力。commit message、PR 描述、merge conflict 都可以让 Cascade 代劳。

AI 生成 Commit Message

在 Source Control 面板里暂存文件后，点输入框旁边的 ✨ 按钮（或按快捷键），Windsurf 会根据 diff 内容自动生成 commit message：

```
# Windsurf 会分析你的 staged changes，生成类似这样的 commit message：
# feat: add user authentication with JWT
#
# - Create auth middleware for route protection
# - Add login/register endpoints with bcrypt password hashing
# - Store JWT tokens in httpOnly cookies
```

生成的 message 会遵循 Conventional Commits 格式。如果你的团队有其他约定，在 `.windsurfrules` 里写清楚：

```
# .windsurfrules
## Git 规范
- commit message 用中文
- 格式：<type>(<scope>): <描述>
- type 可选：feat / fix / refactor / docs / test / chore
```

分支管理

创建新分支时 Cascade 会基于你当前的工作上下文建议分支名。你也可以直接告诉它：

```
帮我创建一个分支，实现用户头像上传功能
```

Cascade 会建议 `feature/user-avatar-upload` 这样的分支名，并自动执行 `git checkout -b`。

Wave 13 之后 Windsurf 支持 **Git Worktree**：在一个窗口里同时打开多个分支的工作目录，配合 Cascade 的并行 Agent，你可以一边在 `feature-a` 分支写新功能，一边让另一个 Cascade 在 `fix-b` 分支修 bug。

Merge Conflict 解决

遇到合并冲突时，Cascade 能理解冲突双方的意图。直接说：

```
帮我解决当前的 merge conflict，保留两边的改动，合并逻辑
```

Cascade 不是简单地选"ours"或"theirs"——它会读懂两个分支分别改了什么，把两边的逻辑合到一起。比如一个分支加了字段验证，另一个分支改了错误提示文案，Cascade 会保留验证逻辑 + 新文案。

复杂冲突还是建议先用 Chat 模式分析，确认合并策略后再让 Code 模式执行。

AI 生成 PR 描述

写 PR 时让 Cascade 帮你生成描述：

```
帮我写一个 PR 描述，总结这个分支上所有的改动。
包括：改了什么、为什么改、怎么测试
```

Cascade 会读取分支上的所有 commit，分析代码变更，生成结构化的 PR 描述：

```
## What
- 新增用户头像上传功能，支持 JPG/PNG/WebP 格式
- 头像存储到 S3，URL 写入用户表

## Why
产品需求 #142：用户个人资料页需要展示头像

## How to test
1. 登录后进入 /profile 页面
2. 点击头像区域上传一张图片
3. 验证图片显示正确且刷新后持久化
```

团队协作配置

多人协作时，`.windsurfrules` 应该提交到仓库里，让团队成员共享 AI 编码规范：

```
# 把 rules 文件加入版本控制
git add .windsurfrules
git commit -m "chore: add windsurf rules for team coding standards"
```

建议团队统一的 rules 内容：

```
# .windsurfrules（团队共享）
## 代码规范
- 变量命名用 camelCase，组件用 PascalCase
- API 路由文件放 src/routes/，中间件放 src/middleware/
- 测试文件和源文件同目录，命名 *.test.ts

## Review 准则
- 新增 API 端点必须加入参校验（Zod schema）
- 数据库操作必须在 try-catch 里
- 敏感操作需要日志记录
```

每个人还可以在 `~/.windsurf/global_rules.md` 里配自己的个人偏好（比如用 Vim 快捷键、偏好 pnpm），这些不会影响团队成员。

CHAPTER 08

🏗️ 项目实战：用 Cascade 搭全栈 CRUD

光看功能介绍不直观，这一章完整演示用 Windsurf 从空目录到可运行的全栈应用的过程。项目是一个带用户认证的 Todo CRUD，技术栈 Next.js + Prisma + SQLite。

第一步：初始化项目

打开一个空文件夹，在 Cascade Code 模式里输入：

```
创建一个 Next.js 15 + TypeScript 项目，用 App Router。
集成 Prisma ORM + SQLite 做数据库。
不要用任何 UI 库，用 Tailwind CSS 手写样式。
```

Cascade 会自动执行：

```
npx create-next-app@latest todo-app --typescript --tailwind --app --src-dir
cd todo-app
npm install prisma @prisma/client
npx prisma init --datasource-provider sqlite
```

等它跑完你会看到一个标准的 Next.js 项目结构，`prisma/schema.prisma` 已经创建好。

第二步：定义数据模型

继续告诉 Cascade：

```
在 prisma/schema.prisma 里定义两个模型：
- User：id, email (unique), password, name, createdAt
- Todo：id, title, completed (默认 false), createdAt, userId (外键关联 User)
然后跑 prisma migrate
```

Cascade 会编辑 schema 文件并执行迁移：

```
model User {
  id        Int      @id @default(autoincrement())
  email     String   @unique
  password  String
  name      String
  createdAt DateTime @default(now())
  todos     Todo[]
}

model Todo {
  id        Int      @id @default(autoincrement())
  title     String
  completed Boolean  @default(false)
  createdAt DateTime @default(now())
  userId    Int
  user      User     @relation(fields: [userId], references: [id])
}
```

```
npx prisma migrate dev --name init
# 自动生成 prisma/migrations/ 目录和 SQLite 数据库文件
```

第三步：API 路由

```
创建 CRUD API 路由：
- POST /api/todos - 新建待办
- GET /api/todos - 获取当前用户的所有待办
- PATCH /api/todos/[id] - 更新待办状态
- DELETE /api/todos/[id] - 删除待办
用 NextResponse，数据库操作用 Prisma Client
```

Cascade 会在 `src/app/api/todos/` 下创建 `route.ts` 和 `[id]/route.ts`，每个端点大约 20–30 行代码。这里的关键是让 Cascade 一次性生成所有路由，而不是一个一个写——它能保持代码风格一致。

第四步：前端页面

```
创建 /todos 页面，功能：
- 顶部输入框 + 添加按钮
- 待办列表，每项有勾选和删除按钮
- 勾选后文字加删除线
- 用 Server Actions 或 fetch 调用 API
- 样式用 Tailwind，暗色主题
```

Cascade 会创建 `src/app/todos/page.tsx` 和相关的 Client Components。它一般会把列表项拆成独立组件（`TodoItem.tsx`），这是合理的拆分。

这时候用 **Web Preview** 功能实时查看效果：点 Cascade 面板右上角的预览按钮，边改边看，不用切浏览器。

第五步：收尾和优化

项目能跑了，让 Cascade 做收尾：

```
请检查并优化：
1. API 路由加入参校验（title 不能为空，不能超过 200 字符）
2. 加 loading 状态和错误提示
3. 空列表显示友好的提示文案
4. 添加成功后自动清空输入框并 focus
```

Cascade 会跨多个文件做这些优化，每步都有 diff 预览。

从这个项目学到什么

整个过程耗时大约 15–20 分钟（如果你手写，少说两个小时）。几个关键操作习惯：

- **大任务先给全貌**。第一个 prompt 就说清技术栈和项目结构，后面的 prompt 就不用反复解释上下文。
- **一次只做一层**。数据模型 → API → 前端 → 优化，每步让 Cascade 完成后检查一下再继续，别一个 prompt 塞所有需求。
- **善用 Web Preview**。前端改动实时看效果，比盲写然后刷浏览器高效得多。
- **Revert 不可怕**。Cascade 生成的代码不满意直接 Revert 重来，比手动改它的代码更快。

匠人学院
JR ACADEMY

Windsurf 上手了，接下来怎么把 AI 编程变成真正的技能？

这本手册的在线版在 jiangren.com.au/wiki/windsurf-guide，随官网更新。匠人学院还有 20+ 本同款 AI 工具指南（Cursor、Claude Code、n8n、Dify…）和 AI Engineer 方向的项目制课程——想知道怎么把 Windsurf + Cascade 变成简历里的实战经历，来聊聊。

扫码进社群，Windsurf / AI IDE 实战答疑

把这本 PDF 转给还在手写代码、没用上 AI IDE 的同学。

jiangren.com.au