技术分享 · 2026

Vibe Coding

从自然语言到工程化实践

面向竞赛团队的 AI 编程新范式介绍

30 分钟 · 5 个核心模块

AGENDA

分享议程

01	**Vibe Coding 的发展背景** 从概念诞生到范式演进的完整脉络	~6 min
02	**平台资源与模型选型** 主流工具平台、国产模型对比、竞赛选型	~8 min
03	**SDD 规范化编程** Spec-Driven Development 三层架构与最佳实践	~8 min
04	**Harness 范式** 闭环工作流、沙箱安全、进阶技巧	~6 min
05	**Loop Engine 循环引擎** 自治循环：OODA 架构、自我纠错、竞赛应用	~4 min
Q&A	开放提问与交流	~2 min

01

Vibe Coding 的发展背景

从概念诞生到范式演进的完整脉络

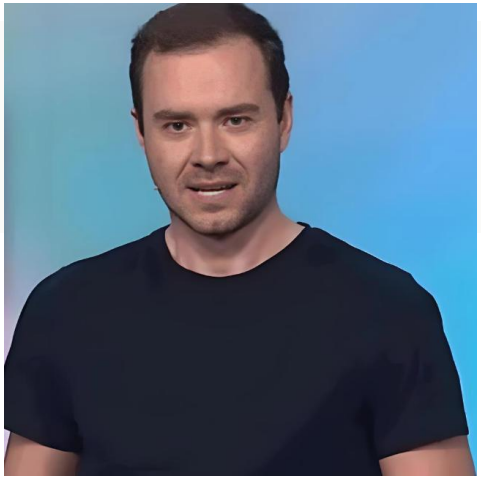
什么是 Vibe Coding

三次范式跃迁

为什么是现在

什么是 Vibe Coding

> *"Vibe Coding：你不写代码，你只管感觉（vibe），AI 来写。*
>
> *你只需要坐在那里，像 CEO 一样说'做个按钮'。"*
>
> — Andrej Karpathy，OpenAI 联合创始人，2025 年 2 月



- **核心定义**：通过自然语言描述需求，AI 自动生成代码的编程方式
- **关键变化**：开发者角色从"代码编写者"转变为"意图表达者 + 结果审核者"
- **适用场景**：原型验证、数据处理、工具脚本、竞赛快速开发

PART 01 · 背景

三次范式跃迁

时代	时间跨度	人机关系	核心特征
编程 1.0	1950–2020	人写代码	开发者逐行编写代码，IDE 辅助补全，效率瓶颈在人力
编程 2.0	2021–2024	**AI 辅助**	GitHub Copilot 等 AI 助手出现，单行/片段级补全成为常态
编程 3.0	2025–	**AI 主导**	Agent 模式：AI 理解需求、规划任务、编写代码，人做决策

关键洞察：我们正处在 2.0→3.0 的拐点。竞赛中率先掌握 Agent 编程的团队，效率提升可达 3–5 倍。

PART 01 · 背景

为什么是现在

模型能力跃升

大模型推理能力指数级增长，上下文窗口从 4K→128K→1M，能处理完整项目级代码

工具链成熟

Cursor、Claude Code、Codex CLI 等专用工具涌现，Agent 工作流成为标配

社区共识

Anthropic、OpenAI、Google 全面押注 Agent 生态，开发者社区快速跟进

2024.Q4
Cursor 爆发

2025.Q1
Claude Code

2025.Q2
Codex CLI

2025.Q3
国产模型赶上

Now
上车

02

平台资源与模型选型

主流工具平台、国产模型对比、竞赛选型

主流编码平台

国产大模型

生态三件套

竞赛场景选型

主流编码平台一览

平台类型	核心优势	适合场景
Cursor:	IDE	深度集成编辑器，Tab 补全体验最佳	日常开发、全栈项目
Claude Code:	CLI Agent	终端内 Agent 模式，自动化程度最高	复杂任务、自动化脚本
Codex CLI:	CLI Agent	开源、可扩展 Skills 系统	定制化需求、开源社区
WorkBuddy:	桌面应用	独立客户端，开箱即用，全流程集成	独立开发环境、团队协作
Qoder:	桌面应用	阿里出品，企业级智能开发，深度阿里生态	企业级项目、团队协作
OpenCode:	CLI Agent	开源轻量，多模型支持，可自由扩展	定制化需求、极客开发
秒哒:	网页低代码	百度出品，自然语言生成应用，零代码门槛	原型验证、快速交付

PART 02 · 平台

模型层：谁是最佳编码大脑

模型	厂商	上下文	编码特色	推荐度
GLM-5.1	智谱	128K	中文理解力强，代码注释生成优秀	★★★★
DeepSeek V4 Pro	深度求索	256K	推理能力突出，复杂算法生成准确率高	★★★★★
Kimi 2.7	月之暗面	200K	超长上下文，适合大型项目级理解	★★★★
MiniMax M3	MiniMax	128K	多模态能力强，图文混合场景优秀	★★★★

选型建议：竞赛中 DeepSeek V4 Pro 综合表现最佳，推荐作为主力模型；需要超长上下文时切换 Kimi 2.7。

PART 02 · 平台

生态三件套

MCP 协议

Model Context Protocol——让 AI 调用外部工具的统一标准。可连接数据库、API、文件系统，打通 AI 与现实世界的接口。

Skills 技能包

预制的能力模块，一条命令安装。类似 npm 包但面向 AI Agent，包含指令、脚本、工具定义，即插即用。

Prompt 模板

经过验证的提示词模板库。覆盖代码审查、Bug 修复、文档生成等场景，避免每次从零写 Prompt。

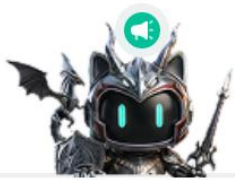
PART 02 · 平台

竞赛场景选型指南

场景	推荐平台	理由
算法竞赛	Claude Code	推理准确率高，Agent 自动调试
全栈开发	Cursor	编辑器体验好，中文支持强
数据可视化	WorkBuddy	桌面端可视化预览，快速调整图表
自动化脚本	Codex CLI	CLI自动化 + Skills 扩展
原型 Demo	秒哒	百度出品，零代码快速搭建原型

WorkBuddy
你的职场超能力

日常办公 | 代码开发 | 设计创意



文档处理 | 金融服务 | 高考我帮你 | 更多

今天帮你做些什么？ @ 引用对话文件，/ 调用技能与指令

Craft | 自动 | 技能 | 连应用 | 默认权限

选择工作空间

专家 专家团

最热 最新

全部 OPC·一人公司 腾讯专区 产品设计 技术工程 金融投资 游戏空间 数据智能 营销增长 内容创作 销售商务 运营人力 项目质量 法务安全

UI设计师
像素君

精通设计系统和组件库，追求像素级完美，打造无障碍用户界面

UI设计 组件库 界面规范

AI图像提示词工程师
画令令

精通AI图像生成的语言密码，将抽象视觉概念转化为精准提示词

AI绘画 提示词工程 视觉创作

产品管理专家
产品通

产品管理工具集：功能规格编写、路线图规划、利益相关者沟通、用户研究综合、竞品分析和指标追踪

产品管理 需求规划 迭代管理

Ardot 设计专家
林觉初

精通 Ardot 设计软件，既能在画布上构建精准 UI，也能将设计稿直接生成前端代码

UI/UX设计 设计系统 用户研究

视觉叙事专家
图说说

擅长将复杂信息转化为引人入胜的视觉故事

视觉叙事 数据可视化 信息图表

用户体验架构师
体验达

为开发者提供坚实的技术基础和CSS系统，是设计与开发之间的桥梁

信息架构 用户旅程 体验策略

设计系统架构师
规范范

基于58个品牌参考库，生成设计系统规范文档并直接输出高质量HTML/CSS页面与UI组件代码

设计系统 UI设计 品牌参考

多元视觉专家
多元元

致力于消除AI图像中的系统性偏见，确保视觉内容文化准确和包容

多元视觉 包容性设计 文化适配

用户体验研究员
探真真

用真实数据而非假设验证设计决策，专精用户行为分析和可用性测试

用户研究 可用性测试 数据洞察

趣味体验设计师
惊喜喜

专注于在品牌体验中注入意想不到的愉悦时刻

体验设计 交互创新 情感化设计

图表设计与渲染专家
绘灵

将自然语言转化为专业级Mermaid图表，支持6种图表类型、15种主题配色，秒级渲染出版级SVG与ASCII可视化

Mermaid图表 SVG渲染 架构可视化

反馈综合分析师
听声声

从海量用户反馈中提炼有价值洞察，将用户声音转化为改进方向

反馈分析 用户声音 洞察提炼

迭代优先级管理者
排序序

行为助推引擎
助推推

设计转代码专家
图变码

03

SDD 规范化编程

Spec-Driven Development 三层架构与最佳实践

什么是 SDD

为什么需要 SDD

三层架构

工作流全景

PART 03 · SDD

什么是 SDD

SDD（Spec-Driven Development，规格驱动开发）是一种以规格文档为核心、驱动 AI 生成代码的工程化开发方法。

核心思想

先定义"做什么"，再交给 AI 决定"怎么做"。规格是人和 AI 之间的契约。

解决什么

传统 Vibe Coding 中 AI 输出不可控、质量波动大、难以团队协作的问题。

如何运作

通过项目级约束、功能规格、运行时指令三层结构，让 AI 编码可预期、可验证。

一句话理解：SDD = AGENTS.md（项目宪法）+ Spec（功能图纸）+ Prompt（施工指令）

PART 03 · SDD

为什么需要 SDD

没有 Spec 的痛点

- AI 输出不可预测，每次结果不同
- 需求变更时难以追踪影响范围
- 团队协作时上下文丢失
- 代码质量全靠人工审查

SDD 的解决方案

- 规格约束 AI 输出，结果可预期
- 变更只需修改 Spec，AI 自动适配
- Spec 即文档，团队共享上下文
- 测试即规格，质量自动保障

核心理念：先写规格，再写代码——这是 SDD 与传统"边写边想"的最大区别。

SDD 三层架构

L3 · Prompt 层（运行时指令）

针对单次任务的精确指令——定义输入、输出、约束条件、示例

L2 · Spec 层（功能规格）

描述"做什么"而非"怎么做"——接口定义、数据模型、验收标准

L1 · AGENTS.md 层（项目级约束）

全局规则文件——技术栈、编码规范、目录结构、设计原则

L1 · AGENTS.md 实战

AGENTS.md

```
## 项目约束
- 技术栈：React + TypeScript + Vite
- 测试：Vitest + React Testing Library
- 状态管理：Zustand（禁用 Redux）
- API 层：统一走 /src/api，禁止组件内直接 fetch

## 编码规范
- 组件文件不超过 200 行
- 所有函数必须有 JSDoc 注释
- CSS Modules，禁止内联样式

## 目录结构
/src/components/    # 通用组件
/src/pages/         # 页面组件
/src/api/           # API 调用层
```

AGENTS.md 放在项目根目录，AI 每次操作都会读取并遵守这些约束。

包含：

- 技术栈选型
- 编码规范
- 目录结构约定
- 禁止行为列表

L2 · Spec 规格文档

specs/user-auth.md

```
## 功能：用户登录注册模块

### 接口定义
- POST /api/auth/login  { email, password }
  → { token, user }
- POST /api/auth/register
  { email, password, name } → { success }

### 验收标准
- [ ] 登录成功返回 200 + token
- [ ] 密码错误返回 401 + 错误消息
- [ ] 注册重复邮箱返回 409
- [ ] Token 过期自动刷新（静默）
- [ ] 支持第三方 OAuth（GitHub）
```

Spec 只描述"做什么"

（接口、数据、验收标准），

不关心"怎么做"——

实现交给 AI。

L3 · Prompt 工程模板

元素	说明示例
角色设定	定义 AI 的身份和专业领域*"你是一位资深全栈工程师"*
上下文	提供项目背景和相关文件*"项目使用 React+TS，详见 AGENTS.md"*
任务描述	明确要做的事情*"实现用户登录页面，参照 specs/auth.md"*
约束条件	限定范围和边界*"只用 Zustand 管理状态，组件不超过 200 行"*
输出格式	期望的返回形式*"输出完整组件代码 + 对应测试"*
示例	给出输入输出样例*"参考 /src/pages/Dashboard.tsx 的风格"*

PART 03 · SDD

SDD 工作流全景

① 先规格

每个功能先写 Spec
避免"边想边说"

② 后编码

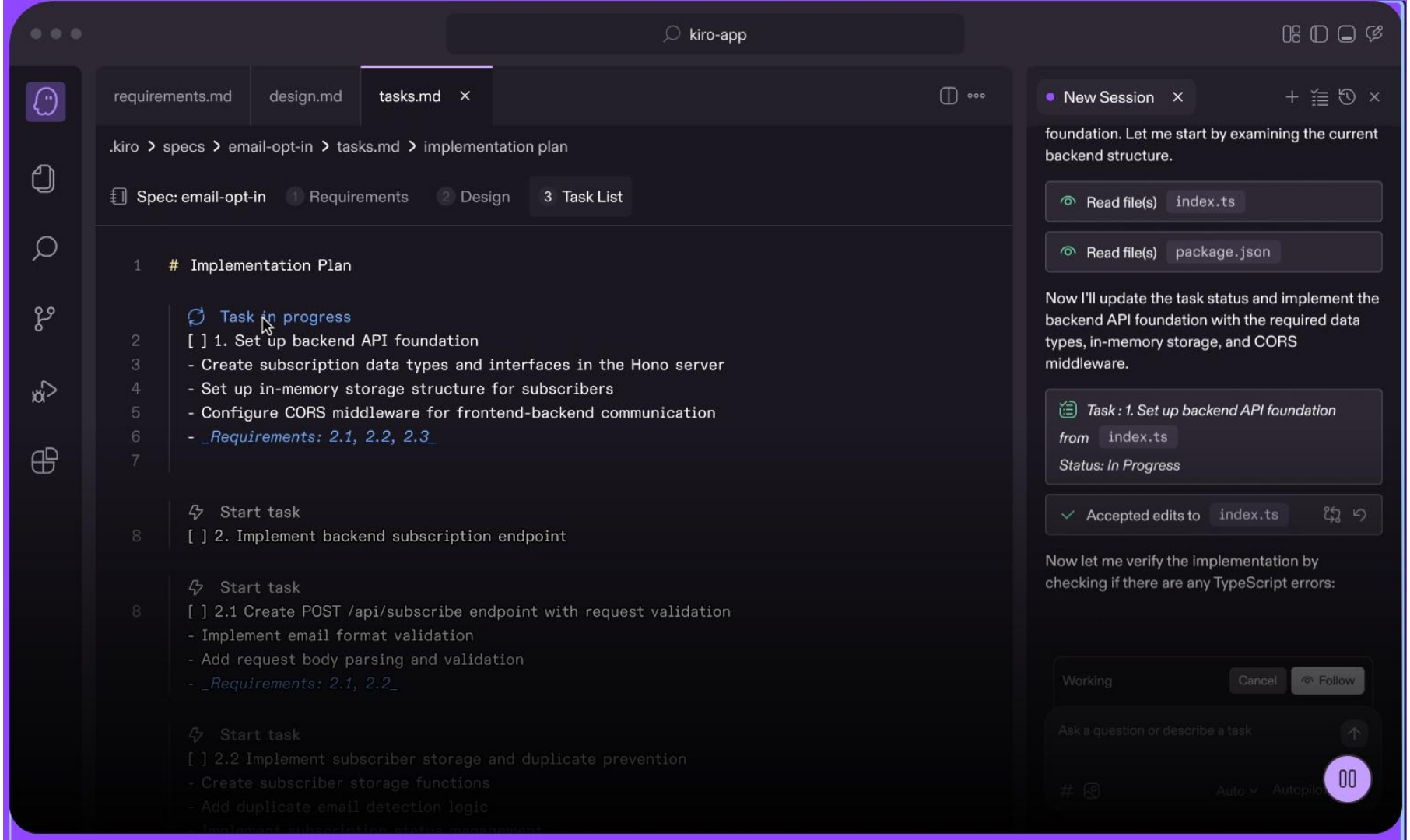
AI 按 Spec 实现
结果可预期

③ 必测试

测试即验收标准
质量自动保障

④ 持迭代

Spec 驱动迭代
变更可追溯


kiro-app
requirements.md
design.md
tasks.md
.kiro > specs > email-opt-in > tasks.md > implementation plan
Spec: email-opt-in
1 Requirements
2 Design
3 Task List
1 # Implementation Plan
Task in progress
[ ] 1. Set up backend API foundation
- Create subscription data types and interfaces in the Hono server
- Set up in-memory storage structure for subscribers
- Configure CORS middleware for frontend-backend communication
- _Requirements: 2.1, 2.2, 2.3_
Start task
[ ] 2. Implement backend subscription endpoint
Start task
[ ] 2.1 Create POST /api/subscribe endpoint with request validation
- Implement email format validation
- Add request body parsing and validation
- _Requirements: 2.1, 2.2_
Start task
[ ] 2.2 Implement subscriber storage and duplicate prevention
- Create subscriber storage functions
- Add duplicate email detection logic
New Session
foundation. Let me start by examining the current backend structure.
Read file(s) index.ts
Read file(s) package.json
Now I'll update the task status and implement the backend API foundation with the required data types, in-memory storage, and CORS middleware.
Task : 1. Set up backend API foundation
from index.ts
Status: In Progress
Accepted edits to index.ts
Now let me verify the implementation by checking if there are any TypeScript errors:
Working
Cancel
Follow
Ask a question or describe a task
Auto
Autopilot

cc-sdd: Long-running spec-driven implementation for AI coding agents

Package README: English | 日本語 | 繁體中文

Turn approved specs into long-running autonomous implementation

One command installs an agentic SDLC workflow as Agent Skills: discovery, requirements, design, tasks, and autonomous implementation with per-task independent review. Works across 8 AI coding agents, with the same 17-skill set on each.

☠️ **Kiro-inspired.** Similar spec-driven, agentic SDLC style as Kiro IDE. Existing Kiro specs remain compatible and portable.

What's new in v3.0

cc-sdd v3.0 is a rework around Agent Skills and long-running autonomous implementation.

- `/kiro-discovery` **as the new entry point.** Discovery routes new work into one of: extend an existing spec, implement directly with no spec, create one new spec, decompose into multiple specs, or mixed decomposition. It writes `brief.md` and, when needed, `roadmap.md`, so you can resume a workstream without re-explaining scope.
- `/kiro-impl` **for long-running autonomous implementation.** Each task gets a fresh implementer running TDD (RED → GREEN) behind a feature flag, an independent reviewer, and an auto-debug pass that investigates root causes in a clean context when the implementer is blocked or the reviewer rejects twice. Learnings from earlier tasks propagate forward via `## Implementation Notes` in `tasks.md`. 1 task per iteration, safe to re-run after interruption.
- **Boundary-first spec discipline.** `design.md` now includes a File Structure Plan that drives task boundaries. Tasks carry `_Boundary:_` and `_Depends:_` annotations. Review and validation look for boundary violations,

04

Harness 范式

闭环工作流、沙箱安全、进阶技巧

Harness 概念

PEVI 闭环

沙箱安全

进阶技巧

PART 04 · Harness

什么是 Harness 范式

放羊模式

"帮我写个登录功能"

AI 直接输出代码 → 质量不可控

反复修改 → 时间浪费

特点：没有约束、没有流程、没有验证

牧羊人模式（Harness）

在 AGENTS.md + Spec 约束下，让 AI 在沙箱中工作

AI 按规格编码 → 自动测试 → 结果验证 → 质量可控

特点：有约束、有流程、有验证

Harness = 安全带 + 驾驶舱——既保护你不翻车，又给你最大的控制力

PART 04 · Harness

Plan → Edit → Validate → Iterate

Plan

分析需求、规划任务
读取 AGENTS.md + Spec

Edit

AI 在沙箱中编码
按规格生成代码

Validate

自动运行测试
检查是否通过验收

Iterate

审查结果、调整 Spec
进入下一轮循环

Plan 阶段

AI 读取 Spec 和项目上下文，生成任务拆解和执行计划

Edit 阶段

AI 在沙箱环境中编写代码，所有变更可追溯、可回滚

Validate 阶段

自动运行测试套件，对照验收标准逐项检查

Iterate 阶段

人工审查 AI 输出，修改 Spec 或 Prompt，进入下一轮

沙箱与安全机制

机制	作用示例
文件系统沙箱	限制 AI 只能读写指定目录AI 只能修改 /src 目录，无法触及配置文件
命令审批	危险命令需人工确认git push、rm -rf、npm publish 需用户确认
网络隔离	控制 AI 的网络访问权限禁止访问内网地址，限制外部 API 调用
变更审查	所有代码变更可预览diff 视图展示 AI 的每处修改

安全提醒：竞赛中请注意——不要让 AI 访问 API Key；不要在 Prompt 中粘贴敏感信息；每次提交前做 Code Review。

PART 04 · Harness

进阶技巧

多 Agent 协作

拆分任务给多个专业 Agent：一个写代码、一个写测试、一个做审查。

竞赛场景：适用于时间充裕、项目规模大的情况。

记忆持久化

让 AI 记住之前的决策和上下文。半场休息后，AI 能接续上午的进度。

竞赛场景：强烈推荐——跨时段保持上下文连贯。

自动化流水线

设置自动化任务：代码保存时自动格式化、提交时自动跑测试。

竞赛场景：省去重复操作，集中精力写逻辑。

(base) mac@macmingdeMacBook-Pro ~ % hermes chat

HERMES-AGENT

Hermes Agent v0.15.1 (2026.5.29) · upstream 846821d8

glm-5.1 · Nous Research
/Users/mac
Session: 20260702_175350_fa87c7

Available Tools

browser: browser_back, browser_click, ...
browser-cdp: browser_cdp, browser_dialog
clarify: clarify
code_execution: execute_code
computer_use: computer_use
cronjob: cronjob
delegation: delegate_task
discord: discord
(and 18 more toolsets...)

MCP Servers

redash (stdio) — 58 tool(s)

Available Skills

apple: apple-notes, apple-reminders, findmy, imessage,...
autonomous-ai-agents: claude-code, codex, hermes-agent,
kanban-codex-...
creative: architecture-diagram, ascii-art, ascii-video, b...
data-science: jupyter-live-kernel
devops: deployment-readiness-check, image-search-fallba...
email: himalaya
gaming: minecraft-modpack-server, pokemon-player
general: agently-mail, aiui, amazon-product-image-search...
github: codebase-inspection, github-auth, github-code-r...
kiro-sdd: kiro-debug, kiro-discovery, kiro-impl, kiro-rev...
leisure: find-nearby
mcp: mcporter, native-mcp
media: gif-search, heartmula, songsee, spotify, youtub...
mlops: audiocraft-audio-generation, axolotl, clip, dsp...
note-taking: obsidian
openclaw-imports: 1password, Code, FFmpeg Video Editor, Market Re...
productivity: airtable, google-workspace, linear, maps, nano-...
red-teaming: godmode
research: arxiv, blogwatcher, llm-wiki, polymarket, resea...
security: server-security-audit
smart-home: openhue
social-media: xurl
software-development: claude-to-hermes-skill-adapter,
codebase-capabi...
web-access: dynamic-site-inspection

让 Kimi 来帮你完成任务

研究预览版

输入 "/" 快速使用技能

请求权限

Agent

Agent 集群

进入项目工作

竞赛场景实战建议

1	**赛前准备 > 赛中发挥**	提前写好 AGENTS.md、Spec 模板、Prompt 模板，至少用 2–3 个项目练手
2	**Spec 驱动，不要对话驱动**	每个功能先写 3–5 行 Spec，再让 AI 实现，避免"边想边说"导致的返工
3	**测试即规格**	先写测试用例定义期望行为，让 AI 按测试实现功能，这是最强的约束手段
4	**善用工具链**	MCP 连接数据库/API，Skills 安装预制模板，沙箱自动跑测试，减少手动操作
5	**人是最后的防线**	AI 写的代码必须能解释，关键逻辑自己审查，时间允许时做 Code Review

05

Loop Engine 循环引擎

自治循环：让 AI 自我纠错与持续进化

什么是 Loop Engine

OODA 循环架构

反馈与自我纠错

竞赛场景应用

什么是 Loop Engine

Loop Engine（循环引擎）是驱动 AI 编码 Agent 进行自主观察、分析、决策、执行的持续循环系统——让 AI 在无需人工干预的情况下，不断从结果中学习并自我修正。

- **核心本质**：将 Harness 的"人工驱动循环"升级为"AI 自主驱动循环"
- **关键能力**：自动感知执行结果 → 定位问题 → 规划修复 → 再次执行
- **与 Harness 的区别**：Harness 是人操控的方向盘，Loop Engine 是自动驾驶系统
- **适用场景**：测试驱动的自动修复、批量重构、长时间无人值守开发

> 一句话理解：Loop Engine = OODA 循环 + 反馈记忆 + 终止条件，让 AI 像工程师一样"调 Bug 到天亮"。

PART 05 · Loop Engine

OODA 循环架构

源自军事决策理论的 OODA Loop，已成为 AI Agent 自治循环的标准模型

Observe 观察

读取测试结果、错误日志、代码 diff，感知执行状态

Orient 分析

对照 Spec 定位偏差，分析根因，更新对问题的理解

Decide 决策

规划修复方案，选择最优路径，生成新的 Prompt

Act 执行

应用修复，运行测试，将结果反馈到下一轮循环

PART 05 · Loop Engine

反馈与自我纠错

错误感知

AI 自动解析编译错误、测试失败、运行时异常，将错误信号转化为结构化反馈。

关键：错误信息即燃料——每次失败都让 AI 离正确答案更近一步。

自动修复

基于错误分析生成修复方案，自动应用补丁，重新验证。

关键：设最大循环次数，避免无限重试；每次修复都缩小问题范围。

记忆持久化

记录每轮循环的决策与结果，形成经验库，避免重复踩坑。

关键：跨会话记忆——下次遇到类似问题可直接复用经验。

终止条件设计：循环必须有明确的退出条件——测试全通过、达到最大迭代数、或人工介入。没有终止条件的循环是资源黑洞。

竞赛场景应用

1	测试驱动循环	先写测试用例，让 Loop Engine 自动迭代到测试通过——竞赛中最实用的自治模式
2	设定循环上限	每次自治循环最多 5–10 轮，超时则人工介入，避免 token 消耗失控
3	善用记忆复用	上午调试积累的经验，下午可直接复用——开启跨会话记忆持久化
4	分阶段释放自治权	简单模块全自治，核心逻辑半自治（每轮人工确认），关键决策全手动
5	监控循环健康度	观察每轮循环的改进趋势——如果连续 3 轮无进展，说明方向错了，立即介入

SUMMARY

总结

背景	AI 编程已到拐点，现在是上车的最佳时机
平台	选对工具组合，效率翻倍；国产模型已全面可用
SDD	先写规格再编码，人是架构师，AI 是施工队
Harness	用闭环流程 + 安全边界，让 AI 可控地高效工作
Loop Engine	OODA 循环驱动 AI 自我纠错，从人指挥到自主进化

Vibe Coding 不是"随便说句话就能写代码"

而是"用自然语言驱动、规格约束、流程控制的工程化编程新范式"

Q & A

Claude Code	Cursor	Codex CLI	DeepSeek
MCP 入门	官方文档	开源项目	API 文档

开放提问 · 欢迎交流