从不写 到不看

Vibe Coding 时代，我们如何为"失控"的代码质量上锁?

2026 . 04 . 15 • @WAFFLE

SECTION 01

源起：两个极端挑战

CHALLENGE 01

不写

完全放弃手写代码。所有的逻辑实现、重构、测试全部由 AI 完成。人类只负责意图表达与架构决策。

已达成 (2026 至今)

CHALLENGE 02

不看

放弃代码级审核。人类不再逐行阅读 AI 生成的代码，而是通过一套“系统”来确保结果的正确性。

正在实践：从“信任”转向“验证”

SECTION 02

我的 2026：零行代码，零线上 Bug

0

手写代码行数

0

线上严重 BUG

4 个月+

连续 AI 编码实践

> 当开发速度提升 10 倍时，传统的“人肉审核”必然崩溃。我之所以能做到不写代码且不产生 Bug，是因为我将精力从 写代码 转移到了 构建质量确定性系统。

SECTION 03

确定性系统：六道质量防线

01

CLAUDE.md

项目规范与约束

02

Plan 模式

结构化开发流程

03

上下文压缩

保持 AI 性能

04

本地门禁

Pre-commit 检查

05

CI/CD 守卫

自动化质量门禁

06

灰度监控

渐进发布与反馈

SECTION 04

防线 1: CLAUDE.md / AGENTS.md

AI 没有项目上下文，会从训练数据中"猜测"最佳实践。**CLAUDE.md 是项目的宪法**，确保 AI 每次会话都能对齐规范。

- 项目结构
- 开发环境
- 代码规范
- 测试流程
- 提交指南
- DANGER 清单

MARKDOWN

```
# Project Guide for AI Assistants

## Commands
- Build: `npm run build`
- Test: `npm test <file_path>`
- Lint: `npm run lint`

## Coding Style
- Use Functional Components & Hooks
- Tailwind for all styling
- Error Handling: Use 'handleError' util

## DANGER
- ❌ No 'any' types
- ❌ No inline styles
- ❌ No commits without passing tests
```

SECTION 05

防线 2: Plan 模式 (思维对齐)

WHY & WHEN / 为何与何时

打破 AI 的“直接编码”冲动。在涉及**跨文件重构、复杂逻辑、或架构变更**时，Plan 模式是防止逻辑漂移的唯一手段。

01 探索 Explore
只读模式，建立完整的上下文地图

02 规划 Plan
输出分步骤的实施清单，由人类确认

03 执行 Implement
严格按确认后的清单进行原子化修改

AGENT TRIGGER / 主流工具用法

Claude Code

CLI

`Shift` + `Tab`

会话中循环切换模式：
Normal → Auto-Accept → **Plan Mode**

Codex

IDE

`/plan` or `Shift` + `Tab`

SECTION 06

防线 3: 主动压缩上下文

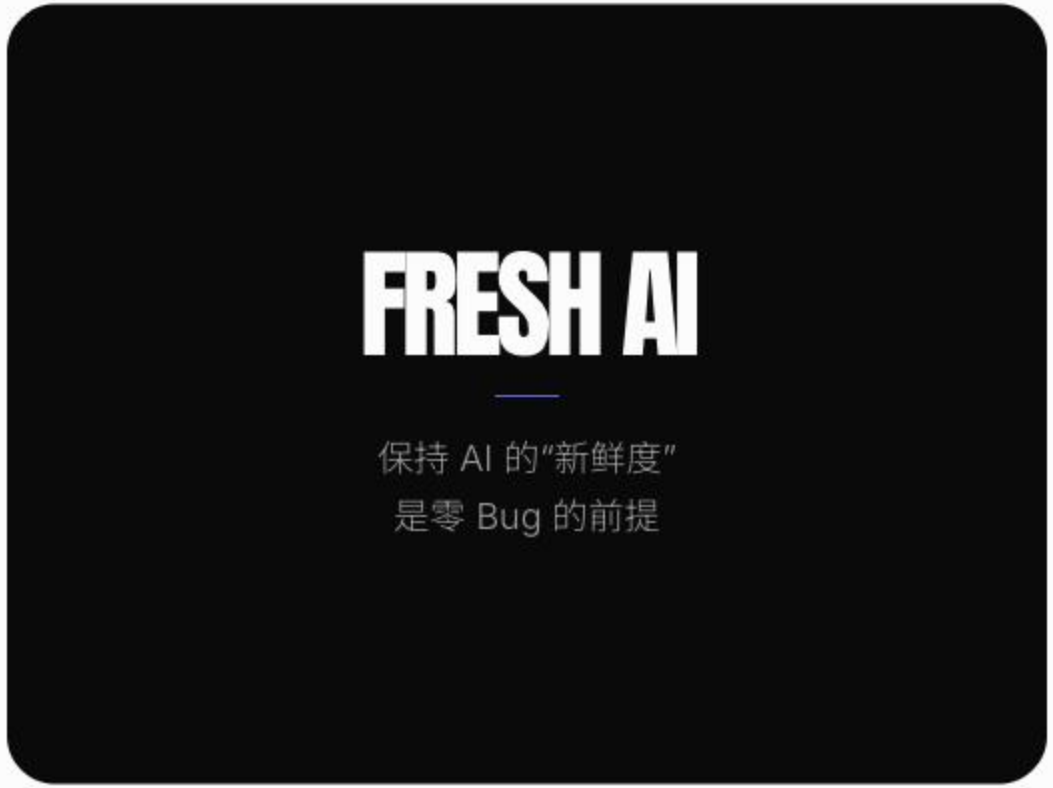
THE CHALLENGE / 核心挑战

Context Window 不是无限的。
随着对话增长，AI 的注意力会分散，导致“幻觉”和逻辑冲突。

/compact
手动触发摘要压缩，保留关键决策

/clear
任务完成后立即清空，防止逻辑干扰

单任务会话
一个会话只解决一个独立问题，物理隔离上下文



SECTION 07

防线 4: 本地门禁 (Pre-commit)

当我不看代码时，我依靠**自动化门禁**来拦截 90% 的低级错误。

- 类型检查：拦截逻辑不匹配
- Lint：强制执行项目风格
- Code Simplifier：自动化代码精简与重构
- 增量测试：确保新代码不破坏旧功能
- Agent 交叉 Review：多模型复核暂存更改

BASH

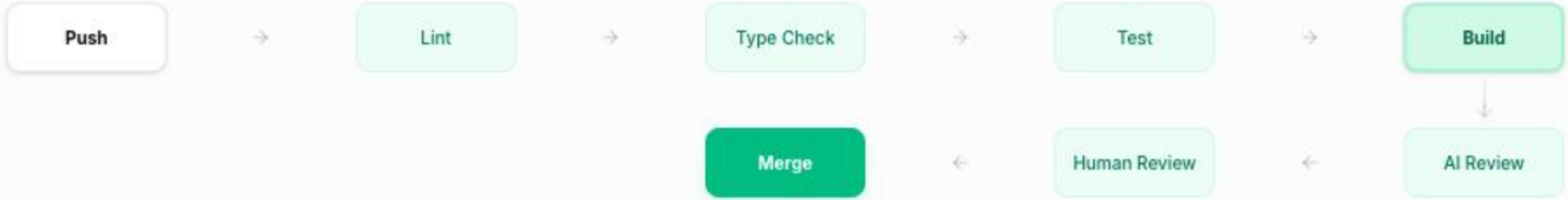
```
#!/bin/sh
# .git/hooks/pre-commit

# 1. 静态检查
npm run lint && npm run typecheck || exit 1

# 2. 代码精简 (Code Simplifier)
npx code-simplifier --staged || exit 1

# 3. 自动化测试
npm run test:changed || exit 1

# 4. 触发多 Agent 交叉审计
npx agent-review --staged || exit 1
```

SECTION 08

防线 5: CI/CD 守卫

THE TRUTH MACHINE / 最终真理

本地门禁追求**效率**，CI/CD 追求**绝对正确**。
在隔离环境中执行全量验证，是代码进入主干的最后关卡。



全量编译测试 (Build)

不同于本地的增量检查，CI 会在干净的容器中执行完整构建。确保没有"只在我的电脑上能运行"的代码。

AI + 人工双重审计

AI 负责扫描模式错误和安全漏洞，人类负责业务逻辑对齐。只有双重通过，代码才具备"确定性"。

SECTION 09

防线 6: 行为审计与反馈

BEHAVIORAL TRUTH / 行为即真相

这是“不看代码”的最后一道防线：**通过行为验证正确性。**
AI Agent 实时监控灰度版本，确保新功能符合预期且不破坏历史资产。

异常与日志审计

Agent 自动扫描 Error Logs，识别新引入的运行时异常，防止“静默失败”。

Error Rate	0.00%
New Exceptions	0

性能基准对齐

对比历史 P99 耗时与内存占用，拦截任何导致用户体验下降的性能退化。

P99 Latency	120ms
Memory	Stable

业务回归验证

验证新功能是否符合产品预期，并确保核心业务链路（如支付、登录）未受影响。

Success Rate	100%
Regressions	0

Agent 闭环反馈机制

一旦监控到行为偏离预期，Agent 将自动收集上下文（日志、堆栈、性能快照），触发回滚并生成改进MR，实现“监控-发现-修复”的自动化闭环。

STATUS
Auto-Optimizing

SECTION 010

Vibe Coding 不是"不看不写"

整体效果：更快 · 更稳 · 更可持续

战略层面（人类负责）

- 架构设计
- 标准制定
- 最终验证

战术层面（AI 负责）

- 代码实现
- 单元测试
- 快速迭代

Q & A

感谢聆听。让我们一起探讨 Vibe Coding 的未来。

2026 . 04 . 15 • @WAFFLE