Claude Autopilot v2 — 基于 Claude Code 4.6 原生能力的方案

0. 关键发现：不需要自己造轮子

之前设计的消息总线、Agent 进程管理、通信协议，Claude Code 4.6 **全部原生支持了**。

Opus 4.6 + Agent Teams 已经提供的能力

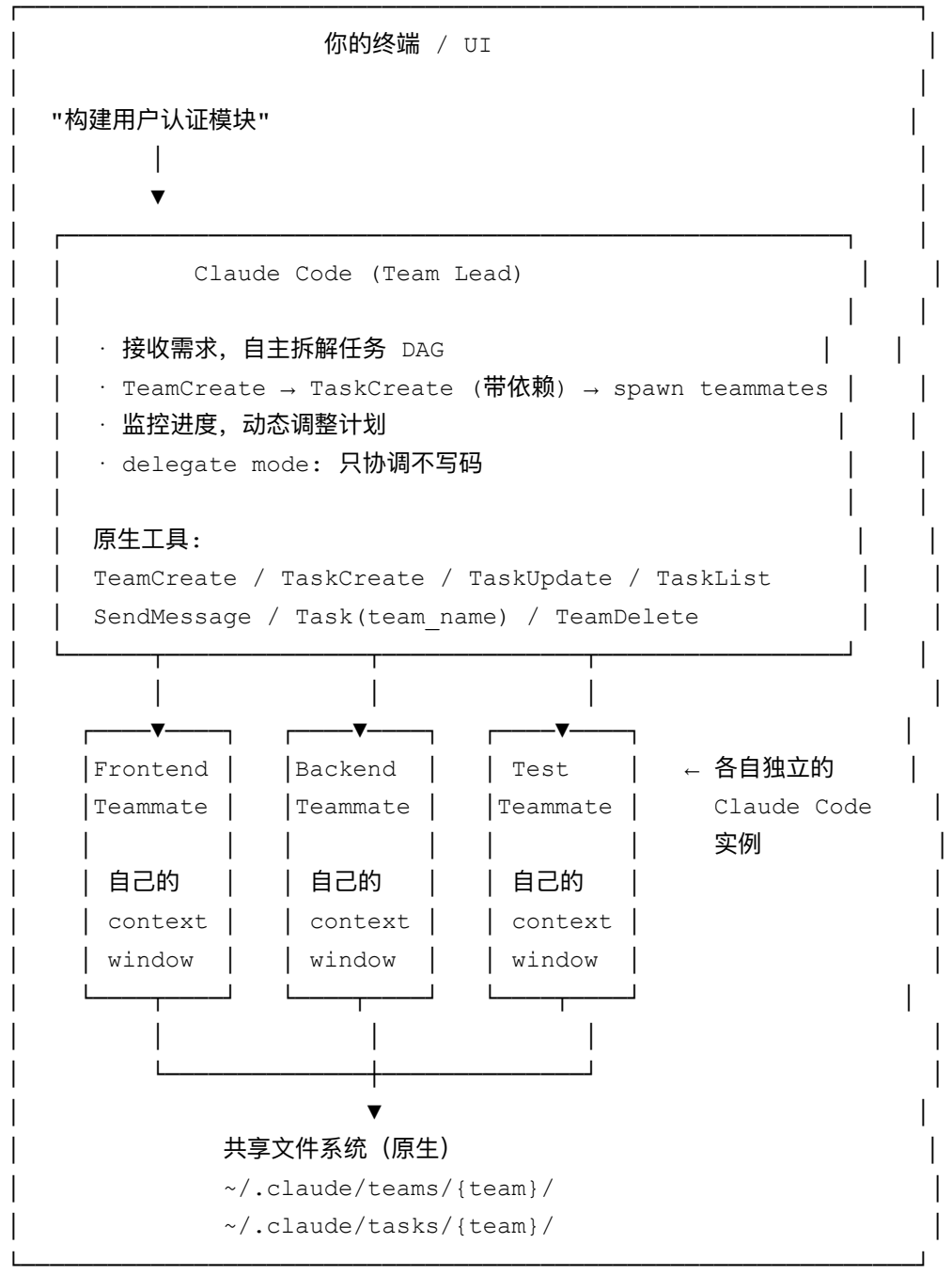
我们之前要自己造的	Claude Code 4.6 原生支持
消息总线	SendMessage — Teammate 间直接通信，无需经过 Lead
任务队列	TaskCreate/TaskList/TaskUpdate — 共享任务列表，JSON 文件存储
依赖管理	blockedBy/blocks 字段 — 任务自动阻塞和解除
Agent 进程管理	TeamCreate + Task(team_name) — 自动 spawn 独立 Claude Code 实例
Agent 间协调	Teammate 自己 claim 任务、互发消息、自主协调
冲突防护	文件锁机制防止多 Agent 同时 claim 同一任务
生命周期管理	requestShutdown/cleanup — 优雅退出

我们只需要建的部分

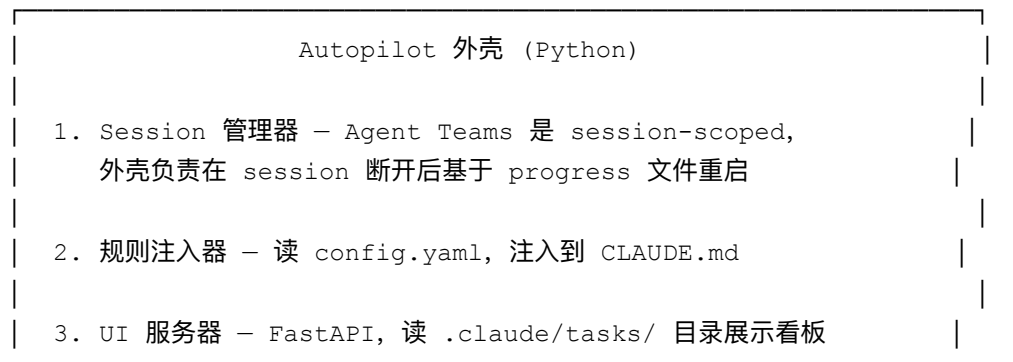
1. **CLAUDE.md + agents.yaml** — 定义团队结构、规则约束、项目上下文
2. **UI 看板** — 可视化 `.claude/tasks/` 和 `.claude/teams/` 目录的状态
3. **测试集成** — 测试 Tab 的逻辑（Playwright/API 测试）
4. **长时间运行的外层循环** — Agent Teams 是 session-scoped，需要外层编排器在 session 断开后重启

---

1. 架构（精简版）


你的终端 / UI
"构建用户认证模块"
Claude Code (Team Lead)
· 接收需求，自主拆解任务 DAG
· TeamCreate → TaskCreate（带依赖）→ spawn teammates
· 监控进度，动态调整计划
· delegate mode：只协调不写码
原生工具：
TeamCreate / TaskCreate / TaskUpdate / TaskList
SendMessage / Task(team_name) / TeamDelete
Frontend Teammate
Backend Teammate
Test Teammate
自己的 context window
← 各自独立的 Claude Code 实例
共享文件系统（原生）
~/.claude/teams/{team}/
~/.claude/tasks/{team}/

外层（我们建的部分）：


Autopilot 外壳 (Python)
1. Session 管理器 — Agent Teams 是 session-scoped，外壳负责在 session 断开后基于 progress 文件重启
2. 规则注入器 — 读 config.yaml，注入到 CLAUDE.md
3. UI 服务器 — FastAPI，读 .claude/tasks/ 目录展示看板

```
|     WebSocket 监听文件变化推送实时状态                                  |
|                                                                        |
|  4. 测试执行器 — Playwright / httpx                                    |
|                                                                        |
└────────────────────────────────────────────────────────────────────────┘
```

---

2. 项目配置

2.1 启用 Agent Teams

```
# 一次性设置
claude settings set env.CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS 1

# 或在 settings.json 中
{
  "env": {
    "CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS": "1"
  }
}
```

2.2 CLAUDE.md（项目根目录）

这个文件是 Claude Code 每次启动自动读取的全局指令。 我们把团队结构、规则约束、项目上下文全部写在这里。

```
# CLAUDE.md

## 项目信息
- 项目名：Kortex 智能办公平台
- 架构：Spring Cloud 微服务 + Vue 3 前端
- 前端目录：./kortex-frontend
- 后端目录：./kortex-backend

## Agent Teams 工作规范

### 你是 Team Lead
当收到功能需求时，你应该：

1. **分析需求**，拆解为具体任务，构建依赖 DAG
2. **创建团队**：TeamCreate
3. **创建任务**：TaskCreate（设置好 blocks/blockedBy）
4. **Spawn Teammates**：按需创建 frontend/backend/test 三个角色
```

5. **启用 delegate mode**：你只协调，不直接写代码
6. **监控进度**：定期 TaskList 检查状态
7. **动态调整**：如果 teammate 报告阻塞，重新规划

Teammate 角色定义

frontend-dev
- 工作目录：./kortex-frontend
- 技能：Vue 3 Composition API, TypeScript, Pinia, Vue Router
- 规则：只修改 src/, tests/, public/ 目录
- 遇到需要后端 API 但还没有 → 先用 Mock 数据，同时 SendMessage 通知 backend-dev

backend-dev
- 工作目录：./kortex-backend
- 技能：Spring Boot, MyBatis-Plus, PostgreSQL, Redis
- 规则：只修改 src/main/, src/test/ 目录
- API 设计完成后 → 立即 SendMessage 通知 frontend-dev 接口规格

test-engineer
- 工作目录：可访问两个项目
- 技能：Playwright E2E, JUnit, API 测试
- 规则：某个功能完成后主动介入测试，不等安排
- 发现 bug → SendMessage 给 team-lead，附截图路径

Teammate Spawn 模板

Task({ team_name: “kortex-feature”, name: “frontend-dev”, subagent_type: “general-purpose”, prompt: `你是 Kortex 项目的前端工程师。 技术栈: Vue 3 + TypeScript + Pinia + Vue Router 工作目录: ./kortex-frontend

工作流程：
1. TaskList 查看可用任务
2. 找到 pending 且无 owner 的前端任务
3. TaskUpdate claim 任务
4. 实现功能，每步 git commit（格式：[TASK-ID] type: description）
5. 完成后 TaskUpdate 标记 completed
6. SendMessage 通知 team-lead 完成情况
7. 如果需要后端 API，SendMessage 给 backend-dev
8. 循环直到无更多任务

编码规范：
- 使用 Composition API + <script setup>
- TypeScript 严格模式，不用 any
- 样式用 scoped CSS
- 组件 props 必须有类型定义

```
禁止：
- 修改 .env / .git / node_modules
- git push --force
- npm publish`,
```

run_in_background: true })

```
### 任务排期原则

你（Team Lead）在排期时遵循以下原则：

1. **关键路径优先**：识别 DAG 中的关键路径，优先安排
2. **最大并行度**：无依赖关系的任务尽量并行
3. **Mock 解耦**：前端可以用 Mock 数据先做，不要死等后端
4. **Test 尽早介入**：有功能完成就安排测试，不要等全部做完
5. **Bug 热修复**：Test 报 bug 后立即创建修复任务，优先级最高
6. **Schema 变更广播**：数据模型变更时 SendMessage 广播所有 teammate

### Git 规则
- 工作分支：ai/autopilot
- 保护分支：main, master（禁止操作）
- commit message: [TASK-ID] type: description
- 禁止 force push

### 禁止命令
rm -rf /, npm publish, docker push, DROP DATABASE, git push --force
```

2.3 Teammate Hooks（可选增强）

Claude Code 支持 hooks，可以在特定事件触发时执行脚本：

```
// .claude/settings.json
{
  "hooks": {
    "TaskCompleted": [{
      "type": "command",
      "command": "python .autopilot/hooks/on_task_complete.py $TASK_ID",
      "timeout": 10000
    }],
    "TeammateIdle": [{
      "type": "command",
      "command": "python .autopilot/hooks/on_teammate_idle.py",
      "timeout": 5000
```

```
    }]
  }
}
```

用途：

- TaskCompleted → 自动触发测试、通知 UI 更新
- TeammateIdle → 检查是否还有任务，没有就触发 shutdown

---

3. 实际执行流程（具体示例）

3.1 你输入需求

```
创建一个 agent team 来构建用户认证模块。

需要三个 teammate：
- frontend-dev：负责登录/注册页面
- backend-dev：负责认证 API 和 JWT
- test-engineer：负责 E2E 和接口测试

请先分析需求拆解任务，设好依赖关系，然后 spawn teammates 开始工作。
你用 delegate mode 只做协调。
```

3.2 Team Lead 自主执行

```
[Team Lead 的思考过程]

1. 分析需求 → 识别出 7 个任务
2. 构建依赖 DAG：

   TASK-001 (DB Schema) → TASK-002 (User API) ─┐
                        → TASK-003 (Auth API)  ─┤→ TASK-005 (联调) → TASK-007 (E2E
                        → TASK-004 (登录页面) ─┘
                                                → TASK-006 (API 测试)

3. 执行：


// Lead 自动执行的操作序列：

// Step 1: 创建团队
TeamCreate({ team_name: "kortex-auth", description: "用户认证模块开发" })
```

```
// Step 2: 创建任务（带依赖关系）
TaskCreate({
  subject: "设计用户表 Schema",
  description: "PostgreSQL user 表设计，含 email/password/phone/avatar/role 字段",
  activeForm: "设计数据库..."
})
// → 自动分配 ID #1

TaskCreate({
  subject: "用户 CRUD API",
  description: "POST/GET/PUT/DELETE /api/users，Spring Boot RESTful",
  activeForm: "开发用户 API..."
})
TaskUpdate({ taskId: "2", addBlockedBy: ["1"] })

TaskCreate({
  subject: "认证 API（登录/注册/刷新token)",
  description: "POST /api/auth/login, /register, /refresh。JWT + Redis 存储",
  activeForm: "开发认证 API..."
})
TaskUpdate({ taskId: "3", addBlockedBy: ["1"] })

TaskCreate({
  subject: "登录注册页面",
  description: "Vue 3 页面，含表单验证、记住我、响应式布局。先用 Mock 数据",
  activeForm: "开发前端页面..."
})
TaskUpdate({ taskId: "4", addBlockedBy: ["1"] })
// 注意：前端不强依赖后端，可以 Mock

TaskCreate({
  subject: "前后端联调",
  description: "替换 Mock 为真实 API，处理错误状态",
  activeForm: "联调中..."
})
TaskUpdate({ taskId: "5", addBlockedBy: ["3", "4"] })

TaskCreate({
  subject: "API 接口测试",
  description: "测试所有 auth 端点，覆盖正常/异常场景",
  activeForm: "测试 API..."
})
TaskUpdate({ taskId: "6", addBlockedBy: ["3"] })

TaskCreate({
  subject: "E2E 测试",
```

```
  description: "Playwright 测试完整登录注册流程",
  activeForm: "E2E 测试..."
})
TaskUpdate({ taskId: "7", addBlockedBy: ["5"] })

// Step 3: Spawn Teammates
Task({
  team_name: "kortex-auth",
  name: "frontend-dev",
  subagent_type: "general-purpose",
  prompt: "你是前端工程师...（见 CLAUDE.md 模板）",
  run_in_background: true
})

Task({
  team_name: "kortex-auth",
  name: "backend-dev",
  subagent_type: "general-purpose",
  prompt: "你是后端工程师...（见 CLAUDE.md 模板）",
  run_in_background: true
})

Task({
  team_name: "kortex-auth",
  name: "test-engineer",
  subagent_type: "general-purpose",
  prompt: "你是测试工程师...（见 CLAUDE.md 模板）",
  run_in_background: true
})
```

3.3 Teammates 自主工作

从这里开始，**完全不需要人干预**：

```
[时间线]

0:00  backend-dev: TaskList → claim #1 (Schema) → 开始设计
      frontend-dev: TaskList → 无可用任务（#4 被 #1 阻塞）→ 等待
      test-engineer: TaskList → 无可用任务 → 等待

0:08  backend-dev: #1 完成 → TaskUpdate(complete)
      → #2, #3, #4 自动解除阻塞

0:09  backend-dev: TaskList → claim #2 (User API)
      frontend-dev: TaskList → claim #4 (登录页面，用 Mock)
      test-engineer: TaskList → 无可用任务 → 等待（或 Lead 安排预备工作）
```

```
0:12  frontend-dev → SendMessage → backend-dev:
      "我需要 POST /api/auth/login 的接口，建议格式：
       request: { email, password }
       response: { code, data: { token, refreshToken, user } }"

0:13  backend-dev → SendMessage → frontend-dev:
      "收到，格式同意。我会在 /api/auth/login 实现这个规格"

0:25  backend-dev: #2 完成 → claim #3 (Auth API)
      frontend-dev: #4 完成（Mock 版）

0:35  backend-dev: #3 完成
      → #5 和 #6 自动解除阻塞

0:36  frontend-dev: claim #5 (联调)
      test-engineer: claim #6 (API 测试)

0:38  test-engineer → SendMessage → team-lead:
      "🐛 发现 bug: POST /api/auth/login 密码错误时返回 500 而不是 401"

0:39  team-lead: 创建热修复任务
      TaskCreate({ subject: "修复登录错误码", ... })
      → SendMessage → backend-dev: "紧急修复，密码错误应返回 401"

0:42  backend-dev: 修复完成
      frontend-dev: 联调完成
      test-engineer: API 测试通过 → claim #7 (E2E)

0:55  test-engineer: E2E 测试全部通过
      team-lead: 所有任务完成，清理团队
```

---

4. UI 看板（我们需要建的部分）

4.1 数据源

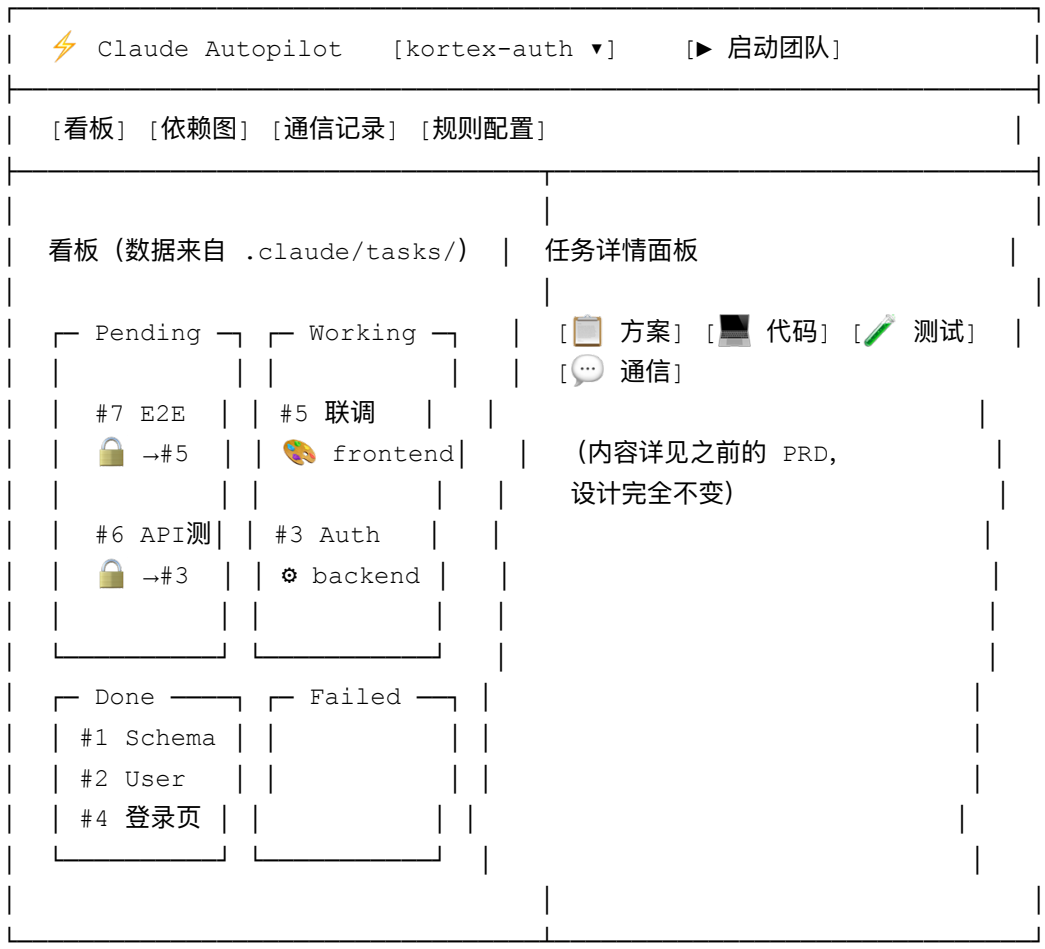
UI 不需要自己维护任务状态，直接**读取 Claude Code 原生的文件系统**：

```
~/.claude/teams/kortex-auth/config.json     → 团队信息
~/.claude/tasks/kortex-auth/1.json          → 任务 #1
~/.claude/tasks/kortex-auth/2.json          → 任务 #2
...
```

每个任务 JSON 文件的结构：

```
{
  "id": "1",
  "subject": "设计用户表 Schema",
  "description": "...",
  "status": "completed",       // pending | in_progress | completed
  "owner": "backend-dev",
  "blockedBy": [],
  "blocks": ["2", "3", "4"],
  "activeForm": "设计数据库...",
  "completedAt": "2026-02-13T14:08:00Z"
}
```

4.2 UI 只做展示层 + 操作触发



4.3 UI 后端实现（极简）

```
# 核心就是一个文件监听器
```

```
from fastapi import FastAPI, WebSocket
from watchfiles import awatch
import json, glob

app = FastAPI()

TASKS_DIR = os.path.expanduser("~/.claude/tasks")
TEAMS_DIR = os.path.expanduser("~/.claude/teams")

@app.get("/api/teams")
def list_teams():
    """列出所有团队"""
    teams = []
    for config_file in glob.glob(f"{TEAMS_DIR}/*/config.json"):
        teams.append(json.loads(open(config_file).read()))
    return teams

@app.get("/api/teams/{team_name}/tasks")
def list_tasks(team_name: str):
    """列出团队的所有任务（直接读 Claude Code 原生文件）"""
    tasks = []
    task_dir = f"{TASKS_DIR}/{team_name}"
    for f in sorted(glob.glob(f"{task_dir}/*.json")):
        tasks.append(json.loads(open(f).read()))
    return tasks

@app.websocket("/ws")
async def ws_endpoint(websocket: WebSocket):
    """监听文件变化，实时推送到前端"""
    await websocket.accept()
    async for changes in awatch(TASKS_DIR, TEAMS_DIR):
        for change_type, path in changes:
            if path.endswith(".json"):
                data = json.loads(open(path).read())
                await websocket.send_json({
                    "type": "task_updated",
                    "data": data
                })
```

UI 后端非常薄 — 本质就是把 Claude Code 原生的 JSON 文件暴露为 REST API + WebSocket。

4.4 💬 通信 Tab

读取 teammate 的 inbox 文件：

```
~/.claude/teams/kortex-auth/inboxes/team-lead.json
~/.claude/teams/kortex-auth/inboxes/frontend-dev.json
~/.claude/teams/kortex-auth/inboxes/backend-dev.json
```

---

5. 长时间运行：外层循环

Agent Teams 是 session-scoped，session 结束任务状态就没了。 所以需要一个轻量的外层来处理长时间运行：

```
# autopilot.py — 外层循环，极简

import subprocess, time, json
from pathlib import Path

def run_session(goal: str, team_name: str):
    """启动一个 Claude Code Agent Teams session"""

    # 读取上次进度（如果有）
    progress_file = Path(f".autopilot/progress/{team_name}.md")
    context = ""
    if progress_file.exists():
        context = f"\n\n## 上次进度\n{progress_file.read_text()}"

    prompt = f"""
{goal}

请使用 Agent Teams 来完成。团队名称：{team_name}

创建 frontend-dev、backend-dev、test-engineer 三个 teammate。
启用 delegate mode，你只做协调。
每完成一个里程碑，把进度摘要写入 .autopilot/progress/{team_name}.md

{context}
"""

    result = subprocess.run(
        ["claude", "-p", prompt, "--max-turns", "100"],
        capture_output=True, text=True,
        timeout=3600,  # 1小时超时
    )

    return result.returncode == 0
```

```
def main():
    goal = "构建用户认证模块，包含登录、注册、JWT token 管理"
    team_name = "kortex-auth"

    max_sessions = 5
    for i in range(max_sessions):
        print(f"=== Session {i+1}/{max_sessions} ===")
        success = run_session(goal, team_name)

        # 检查是否全部完成
        if is_all_done(team_name):
            print("✅ 全部任务完成！")
            break

        print(f"Session 结束，{10}秒后启动下一个...")
        time.sleep(10)
```

关键：每个 session 结束前，Team Lead 把进度写入 `.autopilot/progress/` 文件。 下一个 session 启动时读取进度，从断点继续。 这就是 Anthropic 那篇 “Effective harnesses for long-running agents” 文章的核心思路。

---

6. 与之前两份 PRD 的关系

内容	状态
第一份 PRD: 看板 UI + 方案/代码/测试三个 Tab	✅ 完全保留，UI 设计不变
第一份 PRD: 规则配置页面	✅ 保留，改为写入 CLAUDE.md
第二份方案: Agent 角色设计	✅ 保留，改用 CLAUDE.md 中的 prompt 模板
第二份方案: 消息总线	❌ 删除，用原生 SendMessage
第二份方案: 任务 DAG	❌ 删除，用原生 TaskCreate + blockedBy
第二份方案: Supervisor 进程管理	❌ 简化为外层循环脚本
第二份方案: Agent 自主决策	✅ 保留，通过 prompt engineering 实现
新增: Claude Code hooks	🆕 TaskCompleted/TeammateIdle 事件触发
新增: delegate mode	🆕 Team Lead 只协调不写码
新增: 原生 tmux 分屏	🆕 可视化各 Teammate 实时状态

7. 实施计划（精简后）

Phase 0: 环境准备（0.5 天）

- 启用 Agent Teams feature flag
- 编写 CLAUDE.md（团队结构 + 角色 prompt + 规则约束）
- 测试基本的 TeamCreate → TaskCreate → spawn teammate 流程

Phase 1: 外层循环 + 进度持久化（1 天）

- autopilot.py 外层脚本
- 进度文件读写
- Session 恢复逻辑
- hooks 配置（TaskCompleted 等）

Phase 2: UI 看板（2-3 天）

- FastAPI 后端 — 读 .claude/tasks/ 和 .claude/teams/
- WebSocket 文件监听 + 实时推送
- Vue 3 看板视图 + vuedraggable
- 依赖图视图（Mermaid/D3）

Phase 3: 任务详情面板（2-3 天）

- 📋 方案 Tab — Markdown 编辑/预览
- 💻 代码 Tab — git diff 解析 + diff2html 渲染 + 多项目分组
- 🧪 测试 Tab — Playwright / API 测试触发和结果展示
- 💬 通信 Tab — 读 inbox 文件展示 Agent 间消息

Phase 4: 规则配置（1 天）

- 表单式配置页面
- 写入 CLAUDE.md 和 config.yaml

- 热重载验证

总计: 7-8 天（比之前 10-14 天少了近一半，因为核心编排逻辑全用原生的）

---

8. 成本评估

Agent Teams 每个 teammate 都是独立的 Claude Code 实例，token 消耗是单 session 的 N 倍。

配置	预估 token	按 Opus 4.6 $5/$25 计算
单任务单 agent	~50K input + ~10K output	~$0.50
3 teammate 团队	~200K input + ~50K output	~$2.25
完整功能模块（30+ 任务）	~2M input + ~500K output	~$22.50

省钱建议:

- spawn teammate 时指定 `model: "sonnet"` 给简单任务用便宜模型
- 复杂决策（Planner）用 Opus，执行类任务用 Sonnet
- 利用 effort control 降低简单任务的 token 消耗