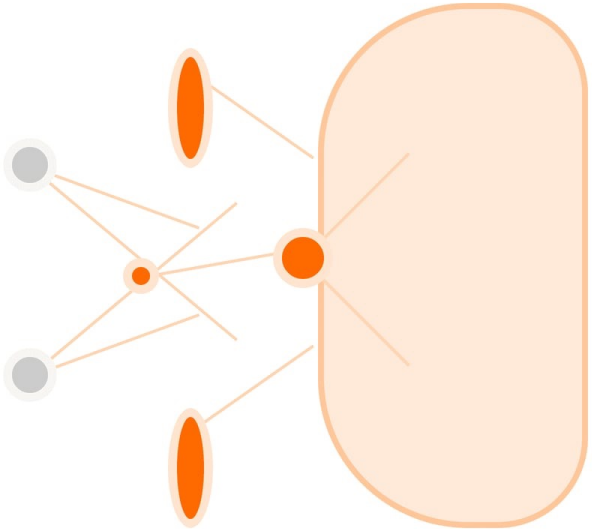
阿里云

AI 时代，我都用 AI 帮我做了什么

——日常篇&Agent开发篇



2026 · 阿里云 AI 实践分享——邓鑫怀

PART 01

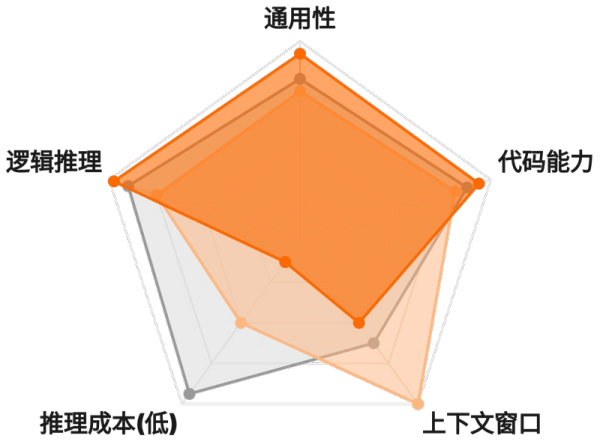
AI 时代，用哪些模型？

不同的模型适合做什么？

2026 主流大模型特性图谱

阿里云

模型与版本	输入侧价格	输出侧价格	系统编排定位
Claude 4.6 Opus	$5.00	$25.00	极致复杂架构从零推演
Claude 4.6 Sonnet	$3.00	$15.00	代码重构与执行主力编排
GPT-4o (Standard)	$2.50	$10.00	工具调用与全能基座
GPT-4o Mini	$0.15	$0.60	轻量级路由底座
Gemini 3.1 Pro	$1.25~3.50	$5.00~10.50	海量代码库极长上下文分析
Qwen 3.5 Plus	$0.26	$1.56	极速吞吐性价比、重度并发
DeepSeek V3/R1	**$0.28**	**$0.42**	**自动化推理闭环与后台运行**



Key Points

- **GPT-5.x:** 跨领域通用性与智能体编排标杆
- **Claude 4.6 Sonnet:** 代码重构与严谨推理的统治者
- **Gemini 3.1 Pro:** 200万+ 上下文，超大代码库的"显微镜"
- **DeepSeek V3/R1:** 节省 85%+ 日常开销，自动化推理闭环基石
- **真实教训:** 使用了 Opus，一晚上烧掉 ¥100,000 API 费用

阿里云

PART 02

介绍我平时都用哪些工具，如何使用

—— Coding 主题篇

人类掌舵，智能体执行——2026 年软件开发范式的彻底蜕变

阿里云

1. 最高效模式：**Human steers. Agent executes.** (人类掌舵，智能体执行)

2. 极限挑战：构建百万行代码产品，人类不手写一行代码

3. Agent 已能自主完成复杂任务，进入"自我迭代、自我验证修复"阶段

4. 速度越快，方向偏差代价越大——方向盘必须牢牢握在人类手里

5. 人类角色提升至**系统编排者**：设计环境、明确意图、保证轨道正确

> ***"AI 是那辆不知疲倦的高性能赛车，而你，是那位实时掌舵的车手。"***
>
> ***不同人之间指导AI生成的代码质量，可能比不同人之间生成的还高。***

阿里云

PART 02 - 1

使用面介绍

—— AI 编码工具实践与架构

AI 编码工具推荐与阵营对比

AI 原生编码工具

Agentic IDEs

- **Cursor** — cursor.com
- **Qoder (通义灵码)** — qoder.com
- **Antigravity** — antigravity.google
- **Kiro** — kiro.dev

核心优势

以 AI 为核心的开发体验，支持**全局上下文感知**、多文件联合重构、自然语言直接生成可执行代码与环境配置。

最佳场景

适合从零构建项目、快速功能迭代、全栈代码生成与系统编排。

传统经典 IDE

Classic IDEs

- **IntelliJ IDEA** — jetbrains.com/idea
- **PyCharm** — jetbrains.com/pycharm

核心优势

极其强大的**静态代码分析**、深度调试器（Debugger）、复杂的性能分析（Profiling）工具。

最佳场景

适合**人类开发者**进行底层逻辑排查、复杂内存泄漏分析，以及对老旧庞大系统的深度调试。

结论：AI 工具负责 "高效生成与系统编排" ，传统 IDE 负责 "人类深度排查与精细调试" 。

"Vibe Coding"的狂热与"熵增"地狱

快速生成 → 调试地狱 → 失控 → 技术债务

- **"Vibe Coding"：** 只靠自然语言，AI 分分钟生成上万行代码
- **灾难现实：** 10,000 行 AI 生成代码 vs 1 行 Error
- **调试地狱：** 代码完全陌生，大海捞针找那 1 行错误
- 初期节省的时间，被"调试地狱"连本带利吞噬
- **核心教训：** 无架构约束 → 代码库迅速陷入不可读的"熵增" → 掌控者彻底失控

如何解决——8大核心实践

阿里云

1 规划先行 (Kick-off)

在动手编码前，先和 Cursor/Claude 进行深度对话。讨论技术栈、第三方库，明确 MVP 的核心功能。不要盲从 AI，手动验证最新文档。

2 人类主导分步执行

让 AI 生成极其详尽的步骤计划，但不要一次性丢给 Agent 执行。你才是“指挥家”，一次只分配一个明确、独立的任务。

3 快速 MVP 与迭代

拒绝“一键生成复杂应用”的幻想。快速实现一个最简陋但能跑通的版本，然后在一个可工作的基础上，一次增加一个功能。

4 记录会话与主动清空

建立工作日志，记录每次会话的有效方法。当任务告一段落，尽早使用 /clear 清空会话，防止上下文超限导致 AI “变笨”。

5 TDD with Agents (测试驱动)

不要直接让 AI 写业务代码。先写测试 → 确认失败 → 让 Agent 实现逻辑 → 验证通过。这是确保 AI 代码不偏离轨道的最佳护栏。

6 自我纠错循环 (Self-Correcting Loops)

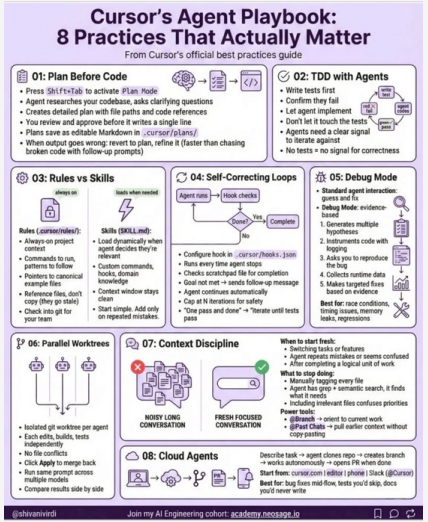
赋予 Agent 运行测试和查看日志的权限。当报错时，让它自主阅读堆栈信息、分析原因并自我修复，形成“执行-反馈-修正”的闭环。

7 借助“外援”处理疑难杂症

遇到棘手 Bug 时，使用 Repo Prompt MCP 让轻量级模型收集上下文，然后打包丢给最强大的推理模型（如 o3 或 Claude 3.5 Sonnet）降维打击。

8 多实例并行 (Parallelization)

不要让一个 Agent 干所有事。使用独立的实例分别进行编码、代码审查 (Review) 和测试，通过多 Agent 协作解决复杂问题。



从"写代码"到"编排系统"——智能体工程师的能力跃迁

阿里云

1. 2026 年行业共识：**Agent Harness**全面兴起
2. 工程师工作重心转移：具体编码 → **宏观系统、架构设计、杠杆作用**
3. 人类通过 Prompt 交互：拆解目标 → 规则转化为代码 → 建立架构护栏

维度	传统工程师 (2020)	AI 编码工程师 (2026)
核心工作	编写业务代码	**设计意图与架构护栏**
主要产出	功能代码	**Prompt、文档、Linter 规则**
调试方式	手动排查	**赋予智能体可观测性工具**
价值体现	代码质量	**系统编排能力与技术品味**

一：当 AI 进展缓慢时，解法绝不是"人类回去写代码"，而是追问："还需要什么能力让智能体清晰可执行？"

二：当前最先进的Agent Harness做到了什么地步？

PART 02 - 2

OpenAI——Codex的Agent harness实践

https://openai.com/zh-Hans-CN/index/harness-engineering/

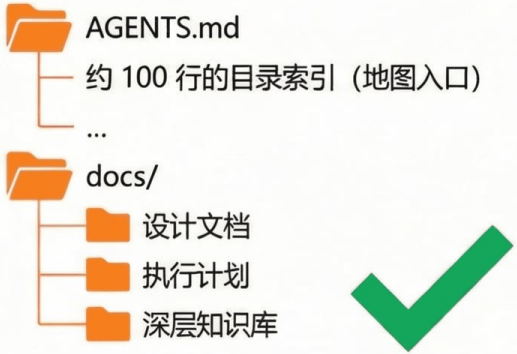
代码仓库即“记录系统” —— 渐进式披露架构

旧方式

信息过载 · 指导腐烂

- **旧方式**：巨大的 .md 文件导致信息过载与指导腐烂。

新最佳实践

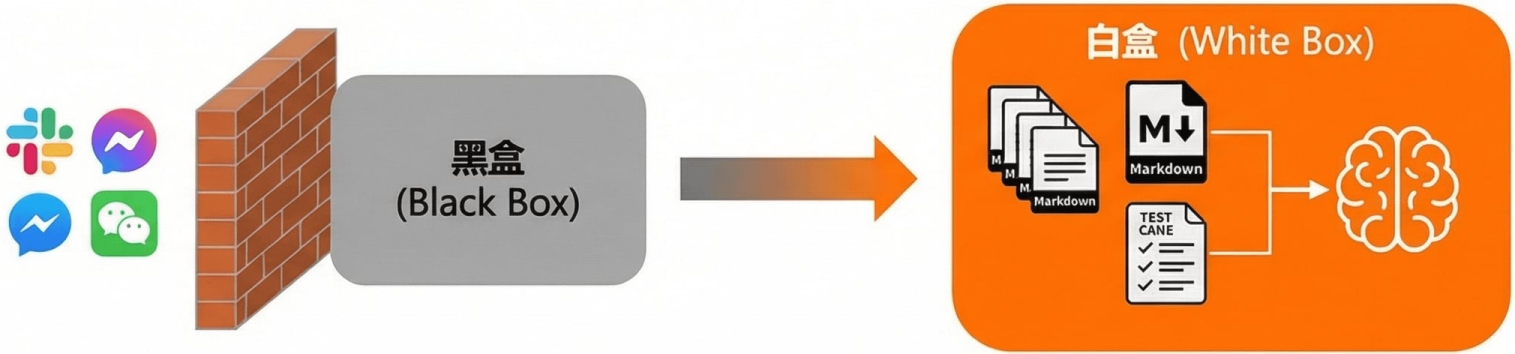


渐进式披露

- **新最佳实践**：给智能体一张“地图”，而非一本 1000 页的说明书。
- AGENTS.md：约 100 行的目录索引（地图入口）。
- docs/ 目录：存放设计文档、执行计划等深层知识库。
- **渐进式披露**：按需动态检索，避免信息淹没。

智能体可读性与证据驱动调试——从黑盒到白盒

- **新目标：**智能体可读性（Agent Readability）优先于人类可读性
- **原则：**不在代码仓库里的内容，对智能体来说不存在



闭环可观测性：接入 Chrome DevTools + LogQL/PromQL 日志查询

Bad Prompt：“还是报错”

→ 黑盒，盲目试错

Good Prompt：“记录日志，查看日志信息，看哪里出了问题”

→ 白盒证据链，快速定位

Agent-First Engineering 三层架构全景蓝图

阿里云

人类掌舵层（Human Steering Layer）

定义意图与规范 → 架构决策 → 品味与反馈编码化

智能体执行层（Agent Execution Layer）

AI 编码智能体为中枢

左翼 Chrome DevTools

右翼可观测性栈

底部自动化 CI/CD Linter

代码仓库层（System of Record）

AGENTS.md 目录索引 + docs/ 结构化知识库

核心原则：渐进式披露——给智能体一张地图，而非一本说明书

Agent-First Engineering 三层架构全景

阿里云

第一层：人类掌舵层 (Human Steering Layer)

定义意图与规范 → 架构决策 → 品味与反馈编码化

第二层：智能体执行层 (Agent Execution Layer)

AI 编码智能体中枢

左翼	底部守卫	右翼
Chrome DevTools (UI驱动/截图/录屏)	Custom Linter + GC Agent (架构强制执行与技术债清理)	可观测性栈 (LogQL/PromQL/TraceQL)

第三层：代码仓库层 (Repository = System of Record)

- **AGENTS.md** (~100行目录索引) → **docs/** 结构化知识库
- 强制Agent分层架构（减少冗余）：Types → Config → Repo → Service → Runtime → UI
- Providers 统一处理横切关注点（认证 / 遥测 / 功能标志）

其他任何内容都不被允许，并将通过自动化方式强制执行。

反馈闭环 (Feedback Loop)

文档更新

↓

Linter 规则

↓

品味编码化

持续回流至人类决策层
形成自我进化的系统

我们当前最棘手的挑战集中在【设计环境、反馈回路和控制系统】方面，帮助智能体实现我们的目标：大规模构建和维护复杂、可靠的软件。

个人 Agent 开发实践：闭环迭代与人机协同

从通用 Loop 到高度定制化的垂直领域 Agent

Agent Loop 的底层逻辑是通用的，真正的差异化竞争力在于 Tool 与 Prompt 的深度定制，以及 完备的基建与自动化评测闭环。

01

定制核心

基于通用的 Agent Loop 框架

深度开发专属 Tool 与 System Prompt

增加自动注入逻辑，显著增强工具的执行效果

02

基建与文档

梳理并沉淀当前架构与调优方式

核心法则：文档越全，天花板越高

确保代码与配置完全解耦，实现轻松定制

03

自动评测

构建测试集，对比 Gold 标准差异

自动导出 LLM 友好的 Markdown 文档

清晰呈现 Input、Output 与 Diff 的结构化数据

04

人机协同

Agent 基于 Diff 文档自动生成修改建议

人工介入：深度审核、思考与修订建议

Agent 根据最终修订结果自动执行代码修改

持续对齐与迭代优化

阿里云

PART 02 - 2

原理面介绍

—— 深入 AI 编码背后的核心机制

TTS (Test-Time Scaling)：测试时计算扩展

阿里云

什么是 TTS?

Test-Time Scaling（也称推理时扩展）是指在模型推理（回答问题）阶段，投入更多的计算资源，让模型“思考得更久”。代表作包括 OpenAI 的 o1/o3 系列和 DeepSeek-R1。

核心机制：思维链与自我纠错

传统的 LLM 是“快思考”（System 1），直接吐出预测的下一个词。而支持 TTS 的模型采用“慢思考”（System 2），在后台生成隐式的思维链（Chain of Thought），它会**尝试不同策略、发现死胡同并自我回溯纠错**。

对 AI 编码的革命性影响

过去，我们需要外部 Agent 框架（如 AutoGPT）来反复调用模型进行 Debug。现在，TTS 将这种“尝试-验证-修复”的循环内化到了模型自身的单次推理过程中，大幅提升了复杂算法和架构设计的准确率。

Pass@k：评估 AI 编码能力的核心指标

阿里云

什么是 Pass@k?

在 HumanEval 或 SWE-bench 等代码评测基准中，Pass@k 是最常用的指标。它表示：让模型为同一个问题生成 **k** 个不同的代码解决方案，只要其中**至少有 1 个**能够通过所有单元测试，即视为通过。

Pass@1 vs Pass@10 vs Pass@100

数据现实

很多模型在 **Pass@1**（一次写对）的成功率可能只有 40%，但如果允许它生成 100 次（**Pass@100**），成功率往往能飙升到 80% 甚至 90% 以上。

为什么它对智能体工程至关重要?

Pass@k 的巨大差异，正是 **Agentic Workflow（智能体工作流）** 能够超越单纯大模型的原因。

人类很难有耐心去审查 AI 生成的 100 份代码。但是，如果我们为 AI 提供一个**自动化测试环境（编译器、Linter、单元测试）**，让它自己生成、自己测试、自己筛选呢?

结论：通过构建自动化的验证闭环，我们可以用系统工程的方法，将模型在实际应用中的表现从拉胯的 Pass@1 强行提升到接近 Pass@100 的极限水平。

PART 03

Guide Learning & Quiz

—— 学习主题篇

Guide Learning & Quiz： AI 伴读与知识内化

阿里云

核心功能：NotebookLM & Gemini

将海量多源文档（PDF、网页、代码）一键导入，生成专属知识库。AI 不仅能总结内容，还能生成 **Audio Overview（双人播客式音频概览）** 和 **Quiz（自动测验）**。

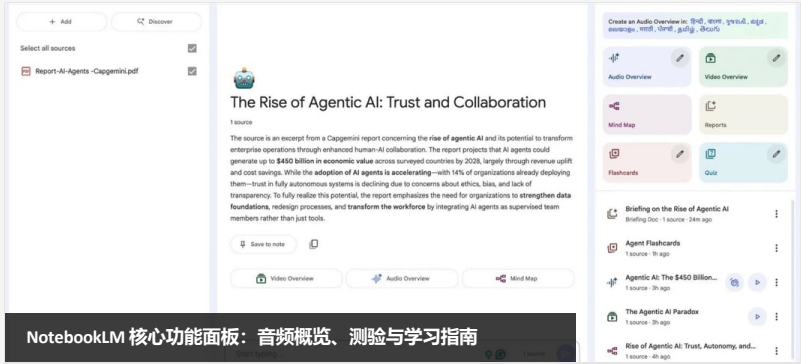
使用场景：面对全新复杂领域

当你需要快速学习一个全新且复杂的领域时，让 AI 充当你的**“专属伴读”**。它能帮你过滤冗余信息，提炼核心概念，并随时解答你在阅读过程中产生的任何疑惑。

背后的思路与小技巧

1. 听觉辅助的费曼学习法： 开启 Audio Overview，听两名 AI 用大白话互相讨论深奥概念，利用碎片时间完成初步认知。

2. 主动检索式学习： 放弃传统的“从头读到尾”，改为带着问题向知识库提问，并利用 Quiz 功能不断验证和巩固自己的掌握程度。



NotebookLM 核心功能面板：音频概览、测验与学习指南

Audio Overview：将枯燥文档转化为生动的双人播客

PART 04

Writing

—— 写作主题篇

Writing： Deep Research 深度研究与重度整合

阿里云

核心功能：自主规划的深度研究

以 **OpenAI Deep Research**、**Gemini** 以及国内的 **秘塔（Metaso）** 为代表。AI 能够自主规划搜索路径，并行阅读数百个网页，最终合成带有严谨引用的万字级长篇研究报告。

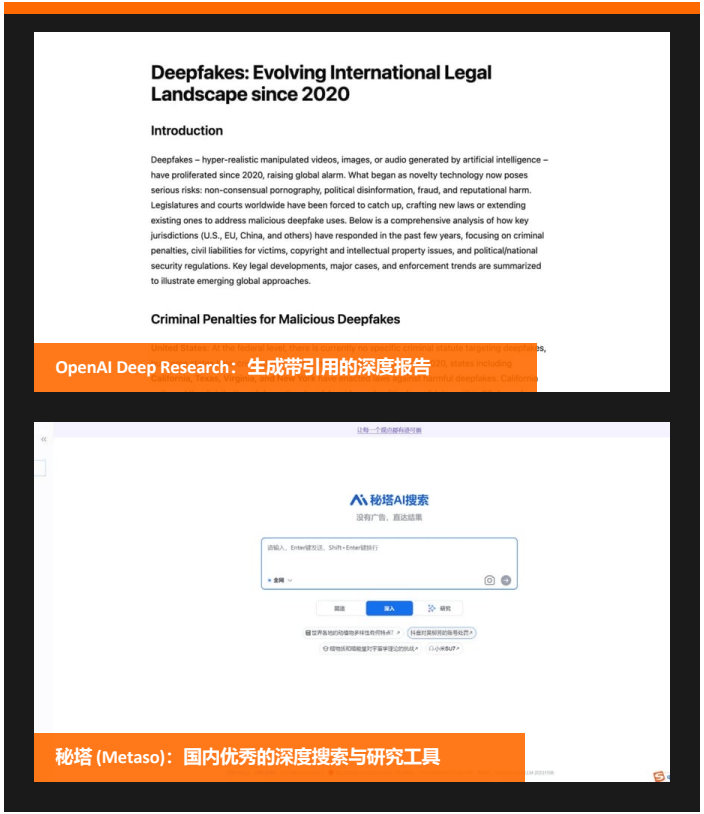
使用场景：告别手动搜集资料

在撰写深度文章、行业分析报告或进行重度信息整合时，不再需要人工逐个点开网页、复制粘贴。AI 一次性为你提供结构化的详实素材库。

背后的思路与小技巧

1. **角色转换：从“写作者”到“主编”。** 你的核心工作变为定义研究边界和核心论点，把搜集、阅读、归纳的“脏活累活”全部交给 AI。

2. **渐进式追问：** 拿到初版报告后，不要直接使用。针对报告中的盲点或有趣的分支，继续使用 Deep Research 进行二次下钻，最终拼接成具有深度的独家内容。



PART 06

Life Beyond Work

—— 工作之外主题篇

Life Beyond Work： AI 作为人生导师与破局者

“能和 AI 聊天就不要和人类聊天... 千万不要沾染任何老登气息，时间宝贵！全力拥抱未来！”

—— 孙宇晨 (X/Twitter 发文)

核心定位：从工具到“赛博导师”

1. 无压力的心灵导师： AI 没有情绪包袱，不会评判你。它能提供 24/7 的无条件倾听与逻辑分析，即使你知道这是算法计算出的共情，依然能提供极大的情绪价值。

2. 绝对理性的困境破局者： 当你陷入思维死胡同或人际冲突时，AI 能够以绝对理性和第三方的“上帝视角”，帮你拆解问题，提供跳出局外的破局思路。

高阶使用思路：建立“私人智囊团”

不要只把 AI 当作搜索引擎。你可以设定不同的 Persona（例如：马斯克、心理学大师阿德勒、冷酷的商业分析师），让他们针对你的同一个人生困境进行**多视角辩论**。

通过这种“虚拟圆桌会议”，你可以跳出自身认知局限，获得前所未有的启发。

阿里云

NEXT CHAPTER

Agent 开发篇 (技术向)

Agent= Model + Agent Harness

模型是驾驶者 → Harness 是载具

Tools (工具)

给模型一双手。提供文件读写、Shell 执行、浏览器控制等与外界交互的能力。

Knowledge (知识)

给模型领域专长。按需动态加载项目文档、代码规范和业务逻辑，避免幻觉。

Observation (观察)

让模型感知环境状态。例如执行命令后的 stdout、Git diff 结果、报错日志等。

Action (行动)

模型对环境施加影响的标准化接口，确保每一次动作都是原子化且可追踪的。

Permissions (权限)

沙箱隔离和审批边界。决定哪些操作可以自动执行，哪些需要人类干预（Human-in-the-loop）。

The Agent Loop：30 行代码，一个智能体

CORE_LOOP.PY

```python
def agent_loop(messages):
    while True:
        response = client.messages.create(
            model=MODEL, system=SYSTEM,
            messages=messages, tools=TOOLS,
        )
        messages.append({"role": "assistant", "content": response.content})

        # 退出条件：模型不再需要调用工具
        if response.stop_reason != "tool_use":
            return

        # 执行工具调用并收集结果
        results = []
        for block in response.content:
            if block.type == "tool_use":
                output = TOOL_HANDLERS[block.name](**block.input)
                results.append({
                    "type": "tool_result",
                    "tool_use_id": block.id,
                    "content": output,
                })
        messages.append({"role": "user", "content": results})
```

模型驱动循环

模型决定何时调用工具、何时停止。代码只是忠实地执行模型的要求，并将结果反馈给模型。

stop_reason 控制退出

当模型认为任务已完成，不再输出工具调用请求时，循环自然终止。没有复杂的硬编码路由。

tool_use 驱动执行

解析模型输出的工具调用块，执行本地函数，并将结果作为 user 角色反馈，形成闭环。

"One loop & Bash is all you need."

工具系统：给 Agent 打造一双手

原子化　可组合　描述清晰

FILESYSTEM

提供对本地文件系统的读写和修改能力，是代码编辑的基础。

read_file, write_file, edit_file

SYSTEM

执行终端命令，运行测试、安装依赖、启动服务。

bash_execute

SEARCH

在庞大的代码库中快速定位文件和代码片段。

glob, grep, *semantic_search

AGENT

派生子智能体处理特定任务，或进行任务板的管理。

Subagent（Agent as task）

NETWORK

发起 HTTP 请求，获取外部 API 数据或查阅在线文档。

search_page, fetch_page

MCP

Model Context Protocol 集成，无缝接入第三方工具生态。

态。

mcp_xxx


Claude Code Tools Reference · 27 Tools
FILE OPERATIONS
Read file_path offset limit pages
Edit file_path old_string new_string replace_all
Write file_path content
NotebookEdit notebook_path new_source cell_id cell_type edit_mode
Glob pattern path
Grep pattern path glob type output_mode -i -A -B multiline
EXECUTION
Bash command timeout description run_in_background
Agent description prompt subagent_type model isolation
Skill skill args
TASK MANAGEMENT
TaskCreate subject description activeForm metadata
TaskGet taskId
TaskUpdate taskId subject status owner addBlocks addBlockedBy
TaskList 无参数
TaskOutput task_id block timeout
TaskStop task_id
WEB
WebFetch url prompt
WebSearch query allowed_domains blocked_domains
USER INTERACTION
AskUserQuestion questions answers annotations metadata
PLANNING
EnterPlanMode 无参数
ExitPlanMode allowedPrompts
GIT WORKTREE
EnterWorktree name
ExitWorktree action discard_changes
SCHEDULING
CronCreate cron prompt recurring
CronDelete id
CronList 无参数
IDE INTEGRATION (MCP)
getDiagnostics uri
executeCode code
Required Optional

3-24版本


Claude Code Tools Reference · 18 Tools
Based on Claude Code Proxy Logs · Jan 2026 · qwen3-coder-plus
FILE OPERATIONS
Read file_path offset limit
Edit file_path old_string new_string replace_all
Write file_path content
NotebookEdit notebook_path new_source cell_id cell_type edit_mode
Glob pattern path
Grep pattern path glob type output_mode -i -A -B -C -n head_limit offset multiline
EXECUTION
Bash command timeout description run_in_background
Task description prompt subagent_type model resume run_in_background
TaskOutput task_id block timeout
KillShell shell_id
WEB
WebFetch url prompt
WebSearch query allowed_domains blocked_domains
TASK MANAGEMENT
TodoWrite todos[]
USER INTERACTION
AskUserQuestion questions[] answers
SKILLS
Skill skill args
PLANNING
EnterPlanMode 无参数
ExitPlanMode 无参数
IDE INTEGRATION (LSP)
LSP operation filePath line character
Required Optional | Source: claude-code-proxy logs 2026-01-12

1-12版本

Agent Hooks：生命周期注入与跨框架对照

22 Hook Events × 7 阶段 × 4 种 Handler

Claude Code · Agent Hooks 生命周期

docs.anthropic.com

① SESSION INIT

- ⭐ SessionStart（环境初始化/注入ENV）
- InstructionsLoaded（加载 CLAUDE.md 规则）

② USER INPUT

- ⭐ UserPromptSubmit（用户提交后/处理前）

③ TOOL EXECUTION LOOP

- ⭐ PreToolUse（工具调用前 allow/deny）
- PermissionRequest（权限弹窗审批）
- ⭐ PostToolUse（工具执行成功后）
- PostToolUseFailure（工具执行失败后）

🔁可多轮循环

④ SUBAGENT

- ⭐ SubagentStart（子Agent启动/handoff）
- SubagentStop（子Agent完成）

⑤ COMPLETION

- ⭐ Stop（Agent完成响应/可block）
- StopFailure（API错误终止）
- Notification（发送通知）

⑥ TEAM & TASK

- TeammateIdle（队友空闲）
- TaskCompleted（任务标记完成）

⑦ SYSTEM EVENTS（贯穿全程）

- ConfigChange（配置变更）
- WorktreeCreate/Remove（Worktree 管理）
- Pre/PostCompact（对话压缩前后）
- Elicitation / Result（MCP 用户交互）
- ⭐ SessionEnd（会话结束/cleanup）

跨框架通用核心注入点

Claude Code	OpenAI SDK	Kiro	LangChain
SessionStart	on_agent_start	—	on_chain_start
UserPromptSubmit	—	Prompt Submit	on_chat_model_start
PreToolUse	on_tool_start	Pre Tool Use	on_tool_start
PostToolUse	on_tool_end	Post Tool Use	on_tool_end
SubagentStart	on_handoff	—	on_agent_action
Stop	on_agent_end	Agent Stop	on_agent_finish
SessionEnd	—	—	on_chain_end

4 种 Handler 类型

Command　HTTP　Prompt　Agent

上下文工程：守护模型的思维清晰度

子智能体隔离

大任务拆小，父子拥有独立的 **messages[]**。子 Agent 的冗长历史直接丢弃，不污染主对话，避免 **Needle in a Haystack** 效应。

三层上下文压缩

上下文总会满，必须有办法腾地方。

1. Micro-compact (微压缩)

在每次 LLM 调用前，将较旧的工具执行结果替换为极简占位符（如 `[Previous: used grep]`），释放即时 Token。

2. Auto-compact (自动压缩)

当 Token 超过阈值时，将完整对话保存到磁盘，并让 LLM 生成当前状态的摘要，用摘要替换所有历史消息。

3. Manual compact (手动压缩)

提供 `compact` 工具，允许模型在认为必要时主动触发压缩，赋予模型对自身记忆的控制权。

任务系统：持久化 DAG，让目标跨越对话存活

没有计划的 Agent 走哪算哪。

任务系统是多 Agent 协作的骨架。

DAG 任务图结构

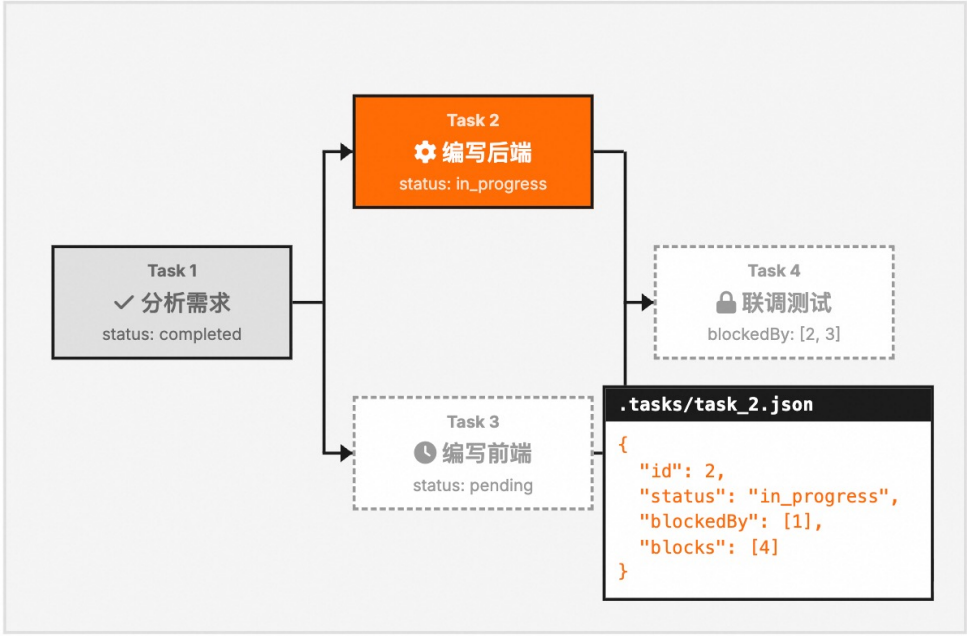
将大目标拆解为有向无环图（DAG）结构的小任务，明确执行顺序与状态（pending / in_progress / completed）。

依赖关系管理

通过 `blockedBy`（前置依赖）和 `blocks`（后置影响）控制任务流转，确保前置任务完成后才解锁后续任务。

磁盘持久化

每个任务保存为独立的 JSON 文件。即使会话重启或上下文被压缩，Agent 依然能从磁盘找回自己的目标。



Worktree 隔离与自治团队：并行不碰撞

Control Plane

任务板管「做什么」(.tasks/)

task_1.json

worktree: "auth-refactor"　in_progress

task_2.json

worktree: "ui-login"　pending

按 ID 绑定
双向同步

Execution Plane

Worktree 管「在哪做」(.worktrees/)

/auth-refactor/

branch: wt/auth-refactor　task_id: 1

/ui-login/

branch: wt/ui-login　task_id: 2

JSONL 异步邮箱与团队协议

当任务足够复杂时，单个 Agent 会遇到瓶颈。通过标准化的 Request-Response 模式（JSONL 邮箱），Agent 可以自主查看任务板，发现状态为 **pending** 且依赖已解除的任务时，**主动认领并执行**，无需中心化的调度器，实现真正的自治团队协作。

多模型协作：让对的模型做对的事

models.yaml

```
profiles:

- name: Claude Sonnet

modelName: claude-4-5-sonnet

- name: Claude Haiku

modelName: claude-4-5-haiku
```

ModelManager 动态路由

借鉴 Kode-Agent 架构，摒弃单一模型打天下的思路。通过 YAML 配置统一管理模型资产，利用**模型指针（Pointers）**将不同复杂度的任务精准路由到最合适的模型，实现成本与效率的极致平衡。

main 指针

`Claude 4.6 Sonnet`

主会话模型。负责与用户进行高层次交互、核心逻辑规划、复杂任务拆解以及全局状态把控。

task 指针

`Claude 4.5 Haiku`

后台子任务模型。负责执行具体、明确的子任务（如批量文件修改、简单的代码生成），高频调用且成本低廉。

compact 指针

`Claude 4.5 Haiku`

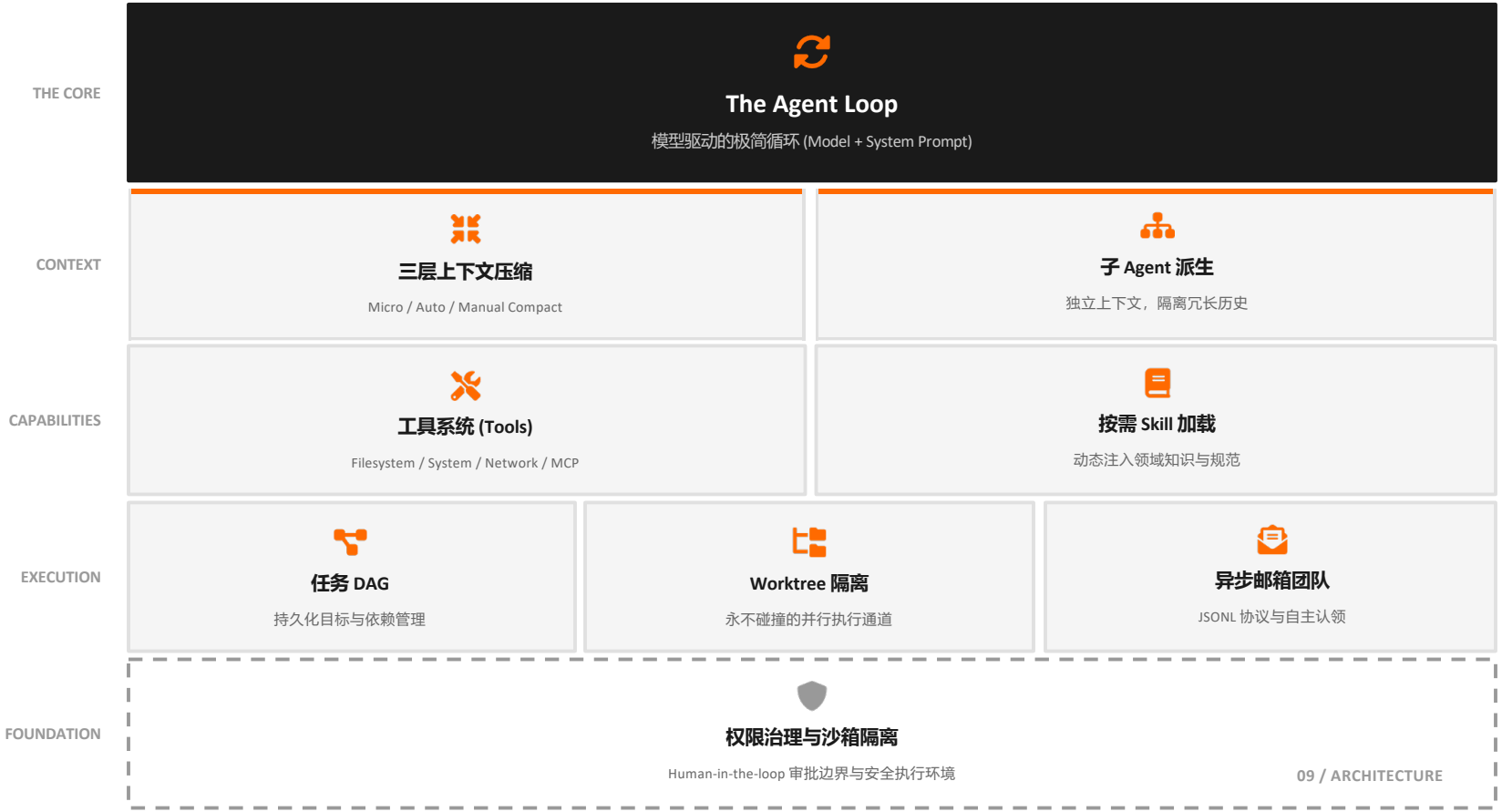
上下文压缩模型。专职负责在后台静默运行，对冗长的历史对话进行摘要和记忆整理，释放 Token 空间。

AskExpertModel

`o3 / opus`

专家召唤工具。当主 Agent 遇到复杂算法或深度推理瓶颈时，临时召唤慢思考模型，获取结果后继续主流程。

Claude Code 完整架构全景



Harness 工程师的自我修养

开发一个 Claude Code 级别的 Agent，本质上是一场极致的 Harness 工程实践。

造好 Harness。
Agent 会完成剩下的。

01 信任模型

不要试图用硬编码逻辑去代替模型做决策。

02 提供工具

构建原子化、健壮的 Action 接口。

03 管理记忆

通过子 Agent 和三层压缩策略保持上下文的纯净。

04 持久化目标

用磁盘上的任务图管理长程计划。

05 隔离执行

利用 Worktree 实现安全的并行协作。

阿里云

结语——向"高维编排者"跃迁，成为 AI 时代的真正大脑

人类掌舵，智能体执行

纪律依然重要，但它体现在支撑结构上，而非代码中。

- AI 时代并未终结工程学，而是根本重塑了它
- 基础系统实现的直接成本正逼近零度
- 纪律依然重要，但体现在支撑结构上，而非代码中
- 人类最终门槛：定义现实问题的能力 + 技术品味 + 系统架构想象力

阿里云

Thank You / 谢谢

阿里云 · 智能体工程实践

参考:

https://github.com/shareAI-lab/learn-claude-code

https://github.com/Yuyz0112/claude-code-reverse

https://openai.com/zh-Hans-CN/index/harness-engineering/