

GitHub热门Harness项目解析

架构拆解、机制对比与选型指南

目录

01  Agent Harness的概念与本质

02  智能编程Harness主流方案

03  个人助理Harness典型框架

04  GitHub热门Harness开源项目

05  Harness的选型指南与发展趋势

模型能力趋同后， Harness 决定 Agent 上限

大模型能力趋同后，真正拉开差距的是什么？

闭源与开源模型的智能差距快速收窄，
同一模型在不同 Agent 框架下的表现差异，[远超模型本身的代际升级](#)。

数据支撑：同一模型，更换 Harness 后的变化

GLM-5.2

23% → 52%

模型不变，仅更换Harness
在SWE-bench Pro 上的pass@1 准确率+29 个百分点

GPT-5.6 Sol

13.3% → 38.3%

在ARC-AGI-3 基准上，“保留推理 + 上下文压缩” 两项
Harness 优化把GPT-5.6 Sol得分增加 25 个百分点，同时输出token 减少了 6 倍。

答案是：[Harness](#)

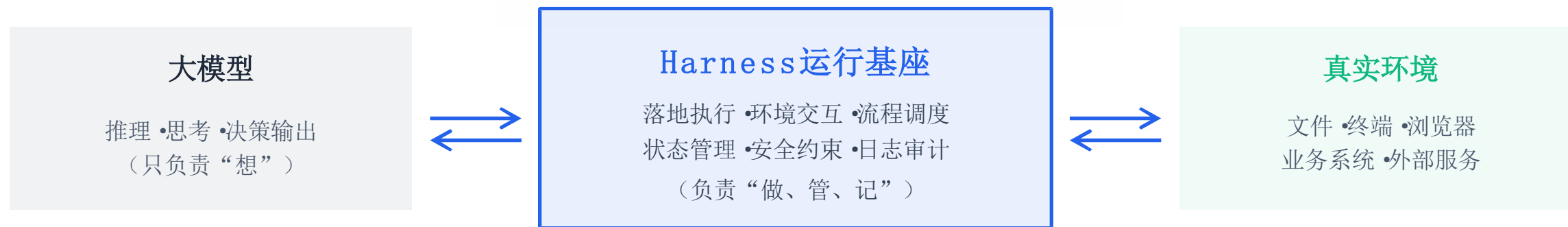
Harness定义了Agent可以使用哪些工具、哪些操作需要审批、每轮能看到什么上下文、崩溃后如何恢复。

模型是引擎，Harness是整车——引擎再强，没有好的底盘也跑不远。

什么是Agent Harness

Harness是AI智能体的工程化运行基座，位于「大模型」与「真实环境」之间的中间执行层。不少AI智能体产品即便搭载相同的大模型，实际落地效果却截然不同。大模型只承担推理与决策，而任务执行、状态管理、安全管控、异常处理等整套落地保障能力，都依赖配套的运行体系支撑，也就是Harness。模型决定了智能体的智商上限，而Harness决定了智能体在真实世界中能否稳定、安全、高效地工作。

$$\text{Agent} = \text{大模型} + \text{Harness}$$



模型负责思考，Harness负责做事、管控、记录、循环执行

为什么需要Harness

原生大模型存在三大落地短板：

痛点一：无执行能力

大模型只会输出文本，无法独立操作：

- 本地文件、终端命令
- 浏览器、业务系统
- 第三方API服务

输出的代码和指令需要人工复制执行，无法自动闭环。

痛点二：无状态能力

没有会话持久化机制，无法支撑长期任务：

- 断点续跑、分支回放
- 上下文压缩管理
- 历史会话检索
- 工作区状态保存

对话中断后需要从头重新开始。

痛点三：无工程约束

缺乏企业级安全管控，无法用于生产环境：

- 权限分级审批
- 沙箱环境隔离
- 全链路审计日志
- 高危操作拦截

误操作可能直接影响生产系统安全。

Harness的核心价值：

为大模型补齐**工程化、安全性、可落地、可复用**的全套能力，让AI从对话模型升级为**可自主工作的智能体**。

Harness 的常见能力模块

01 Agent 循环规划

构建Agent的核心执行引擎，驱动任务自主拆解、优先级调度与闭环执行，是智能决策的中枢。

02 工具调用与协议扩展

打通模型与外部工具的交互通道，通过MCP扩展能力边界，实现实时数据对接与业务场景落地。

03 上下文工程构建

精准提炼与动态管理上下文信息，优化模型输入质量，确保决策的准确性、关联性与场景适配性。

04 记忆与状态管理

持久化存储交互历史与环境状态，支持长程记忆回溯与状态复用，打造连贯、一致的智能交互体验。

05 安全管控与沙箱

构建细粒度权限体系与隔离沙箱环境，严控操作边界，规避越权访问与执行风险，保障系统安全。

06 多智能体协同编排

实现多Agent的任务拆解、分工协作与全局调度，高效应对复杂业务场景下的分布式协同需求。

07 故障自愈与持久化

具备异常检测、自动重试与故障恢复能力，结合状态持久化机制，确保系统高可用与任务连续性。

08 全链路观测与评测

实时监控Agent运行指标与交互日志，量化评估性能与效果，为模型迭代与策略优化提供数据支撑。

Harness 行业演进现状

早期阶段

当前阶段

未来趋势

「模型+零散脚本」

- 无统一架构、无法复用；
- 不可控、不可审计；
- 每个项目从零搭建，缺乏工程化保障。

「Harness基座时代」

- 自研/开源Harness，如Codex、DeepSeek、Pi、Cline、Goose等；
- 标准化协议涌现；
- 生态快速成熟。

「差异化竞争时代」

- 模型能力趋同，Harness将成为护城河；
- 工程能力、生态能力、安全能力决定胜负；
- 向平台化、服务化演进。

行业共识

大模型提供智能与推理能力，Harness 提供运行与工程化能力。

未来智能体的差异化，不再只是模型能力，
还有Harness的工程能力、生态能力、安全能力。

目录

01  Agent Harness的概念与本质

02  智能编程Harness主流方案

03  个人助理Harness典型框架

04  GitHub热门Harness开源项目

05  Harness的选型指南与发展趋势

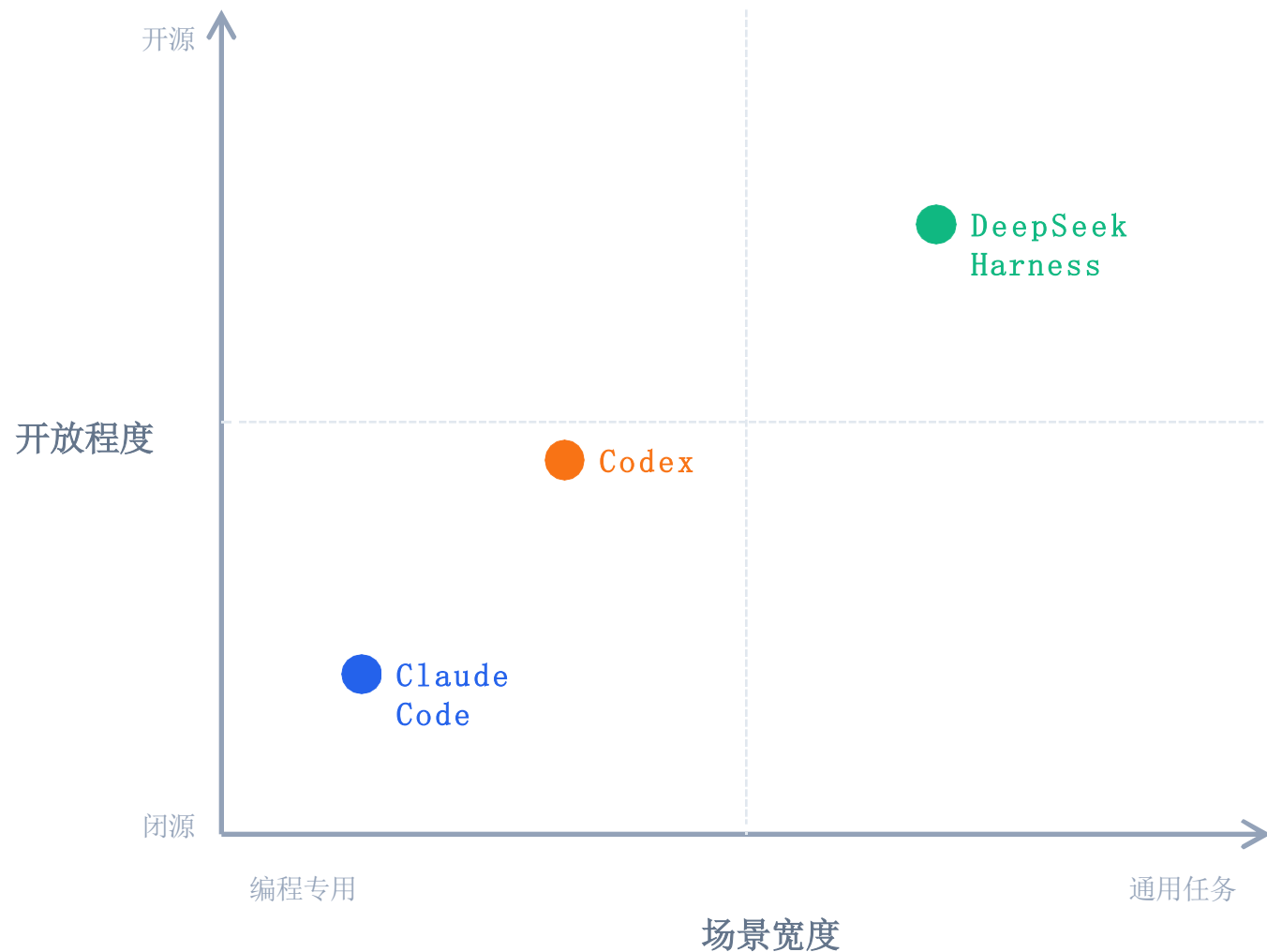
Agent Harness 项目总览

分类	项目	定位	关键特性	Github Stars
主流编程智能体	Claude Code	Anthropic 推出的终端 AI 编程 Agent	可自主读写代码、调用工具并持续执行复杂开发任务，深度绑定 Claude 模型	144k+
	Codex	OpenAI 推出的编程 Agent	强调结构化指令、沙箱执行与可追溯任务流程，可完成端到端编码任务	122k+
	DeepSeek Harness	“一切皆插件”的可组合 Agent 运行时	基于 Cordis 微内核，模型、工具、记忆、Agent Loop 等核心能力均可插件化替换	215k+
个人 AI 助理	OpenClaw	本地运行、支持多渠道接入的个人 AI 助理	Gateway-centric 架构，支持多聊天渠道、设备节点及本地优先运行	389k+
	Hermes Agent	可持续积累经验并自我改进的个人 AI 助理	可从任务经验中生成和改进 Skills，支持长期记忆检索与丰富工具调用	243k+
	OpenHuman	本地优先的个人 AI 与记忆管理平台	强调本地记忆、Agent 编排与个人数据整合，支持 Memory Tree 等能力	39.5k+
其他热门开源项目	OpenCode	开源 AI 编程智能体	面向终端开发工作流，支持多模型、多工具与可扩展 Agent 能力	205k+
	Pi	极简、高度可扩展的编程 Agent	核心保持轻量，通过扩展机制增加工具、子 Agent、工作流等能力	102k+
	OpenHands	面向软件开发任务的 AI Agent 平台	提供事件驱动架构、工具调用、沙箱环境与软件开发 Agent 能力	86.8k+
	Open Interpreter	可通过自然语言操作本地计算机的开源 Agent	支持执行代码、调用本地工具和控制计算机；新版进一步强化开放模型与轻量化运行	68.3k+
	Cline	IDE 原生的 AI 编程智能体	可自主规划、读写代码、调用终端与浏览器，并通过审批机制控制高风险操作	67.7k+
	Goose	本地优先的通用开源 AI Agent	支持 MCP 等扩展协议，可连接工具、服务与开发环境，在本地完成多步骤任务	54k+

主流编程Agent Harness 对比

维度	Claude Code	Codex	DeepSeek Harness
开发者	Anthropic	OpenAI	DeepSeek
核心定位	终端AI编程搭档	可嵌入 Agent Runtime	一切皆插件的可组合Agent运行时
设计理念	Agent-Centric 增强Agent本身	Agent + Platform 标准化 Runtime，可嵌入复用	Runtime-Centric 一切皆插件
核心语言	TypeScript	Rust+TypeScript	TypeScript
开源协议	闭源	Apache-2.0	MIT
模型绑定	以 Claude 系列为核心	默认 OpenAI 模型 可配置自定义 Model Provider	可替换（插件化适配器）
扩展方式	MCP 外部工具+子Agent委托	MCP 服务+自定义工具	一切皆插件
运行形态	终端CLI	CLI/SDK	CLI/SDK

三大Agent Harness的定位与开放程度



- **Claude Code**: 核心实现闭源、专有许可；公开仓库提供文档、插件及相关资源。
- **Codex**: CLI开源、云端闭源，编程为主，开源的一体化，支持本地模型。
- **DeepSeek Harness**: MIT全开源，编程起步但指向通用，采用插件化架构，特点是可改造性。

(1) Claude Code 设计理念与架构

设计理念：把模型当作一个「聪明的软件工程师」

核心思想：运行时越笨，架构越稳定；
框架只做确定性的事，所有智能下沉给模型。不为模型预设复杂流程，而是配齐工具，让模型自己决定怎么做。

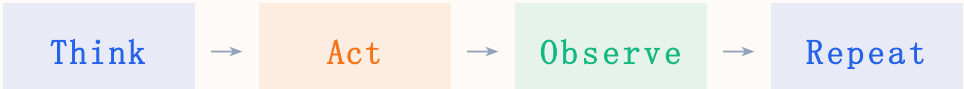
可将其核心工具能力抽象为四类



Bash作为通用适配器：不单独封装git、npm、docker等工具，而是让模型通过Bash自由组合使用。工具越少，模型的自由度越大，这正是减法思维的体现。

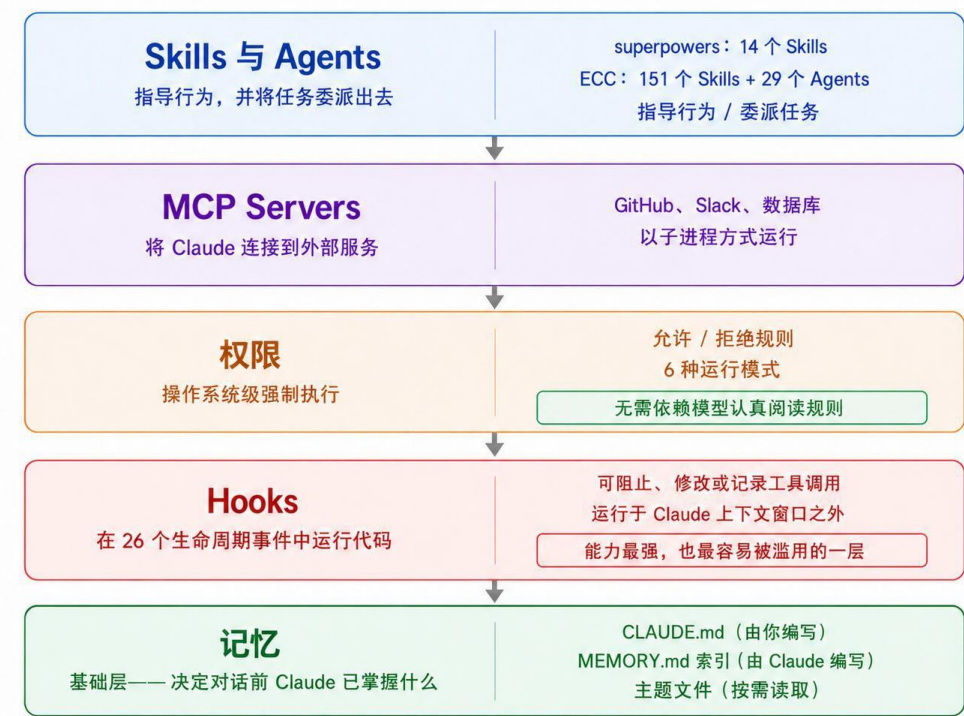
核心思想：给模型最少的束缚，最大的自由度，框架只做确定性的事，所有智能下沉给模型。

TAOR循环：极简的Agent Loop



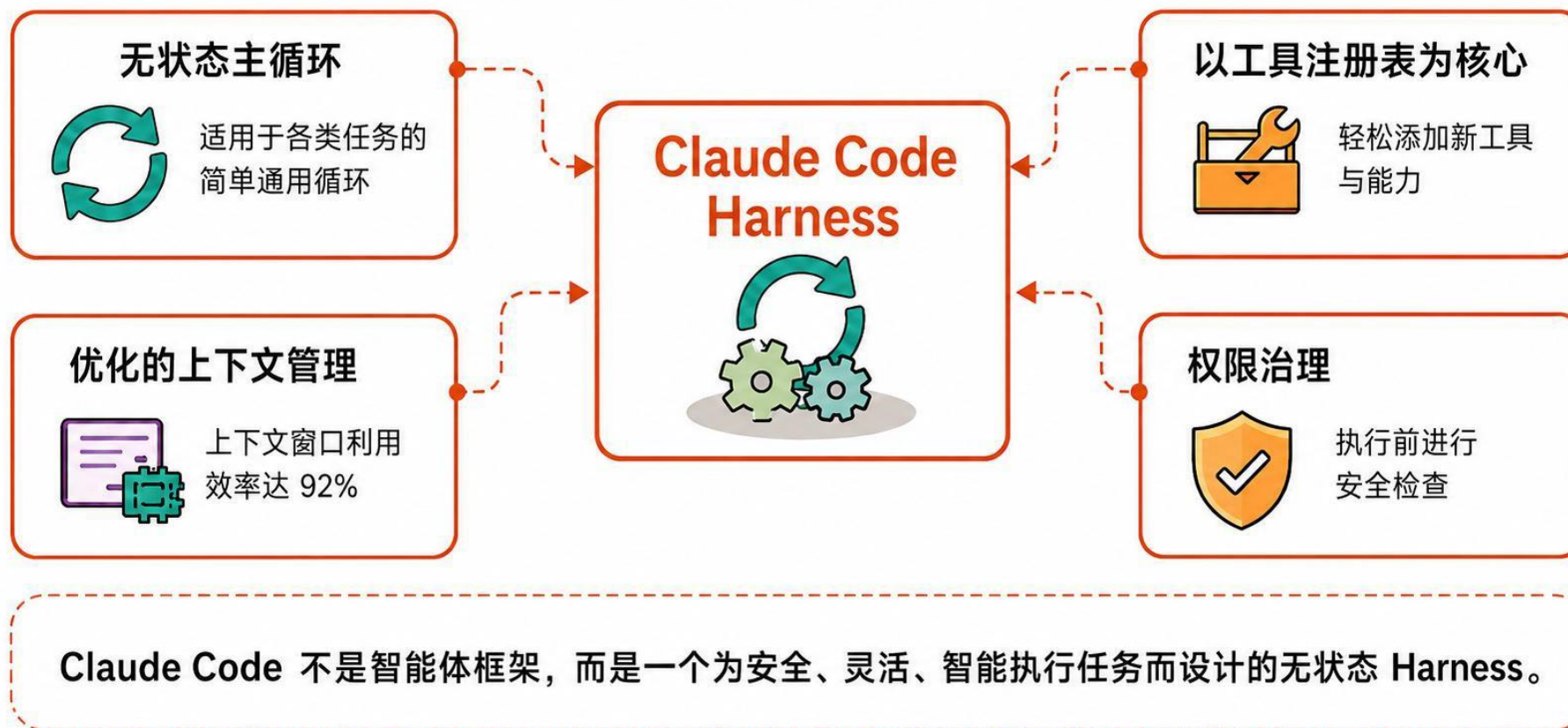
Orchestrator极其轻量化，所有推理、决策、何时停止，全部交给模型。

Claude Code：五层 Harness 架构



Claude Code 核心机制

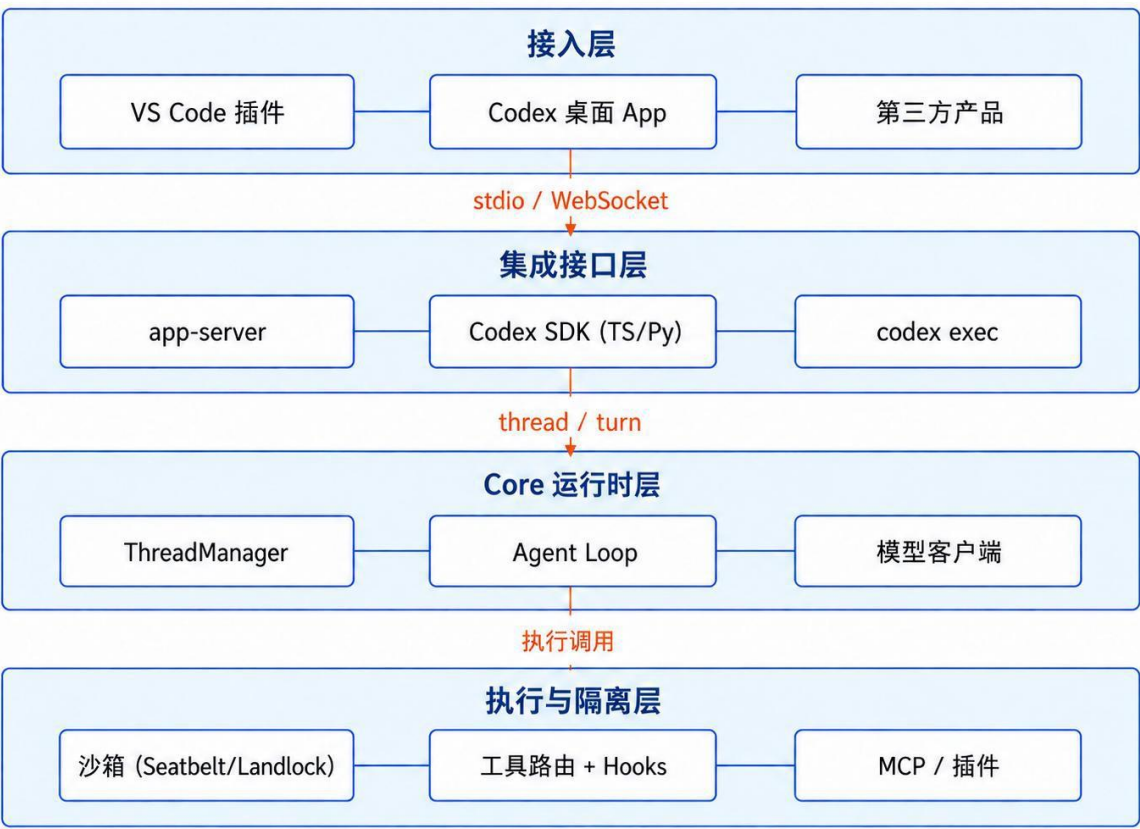
Claude Code 的 Harness 由四大核心组件构成：无状态主循环提供适用于各类任务的简单通用循环；优化的上下文管理课大幅提升上下文窗口利用效率；以工具注册表为核心，支持轻松添加新工具与能力；权限治理在每次执行前进行安全检查。四者共同构成一个薄而稳的执行底座——**智能在模型里，确定性在 Harness 里。**



(2) Codex 设计理念与架构

设计理念：让Agent在真实计算环境中安全、自主地工作

执行边界（沙箱/审批）十分重要，不是先做Agent再补安全，而是把安全内建到架构核心



Agent Loop与三原语模型

Agent Loop: RegularTask→run_turn→run_sampling_request



三原语模型是会话管理的核心抽象：Thread是会话容器，Turn是一次完整交互，Item是单条消息/工具调用。

安全双闸门

闸门一：操作系统级沙箱

三种权限级别，macOS Seatbelt /Linux Landlock

闸门二：独立审批策略层

与沙箱解耦，可自定义审批规则，企业级管控

Codex Harness 开源组件

2026年8月19日，OpenAI宣布Codex Harness作为开源平台（Apache2.0协议）全面开放，将Codex从一个会写代码的助手重新定位为一个可被嵌入任何软件产品的AI Agent平台。本次开源的核心是Agent Loop，位于应用程序与AI模型之间的核心执行系统。



具体开源了以下三大核心组件

Codex CLI

通过Codex exec命令用于CI/CD流水线等自动化场景中的非交互式任务执行。主要面向一次性任务执行，对话不保留、任务完成即终止，适用于排程脚本、CI检查与每日报表等轻量级自动化场景。

Codex SDK

提供TypeScript与Python版本编程接口对智能体任务进行精细程序化控制。面向程序化嵌入，支持对话续接与多轮交互，适用于后端服务集成、自动化 workflows 等需将Agent能力嵌入自有应用的场景。

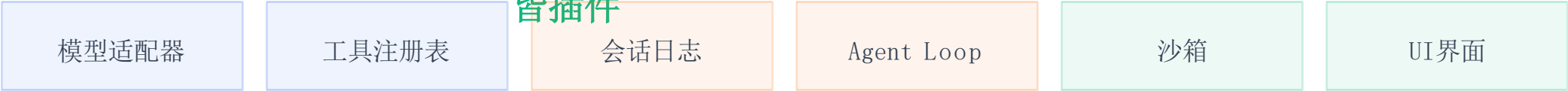
Codex App Server

通过JSON-RPC协议连接应用，支持持久化会话、实时事件流与人机协作审批。面向产品级集成，支持持久化会话与实时事件流，适用于需人工审批关卡的系统等用户直接交互的AI产品。

(3) Deep Seek Harness 设计理念与架构

DeepSeek于2026年8月正式开源了DeepSeek Harness (DSH) 的开发者预览版。这款智能体工具基于“一切皆插件”的理念构建，将几乎所有组件分解为可独立加载和卸载的插件。与Claude Code和Codex等主流Harness产品不同，DSH不提供固定的、开箱即用的体验，它将重塑底层架构的权力交还给开发者。

Everything is a Plugin • 一切
皆插件



底层基石：Cordis微内核

DeepSeek Harness构建在Cordis内核之上。Cordis负责挂载和卸载插件、解析插件之间的依赖关系，并提供它们协同所需的服务与事件。
模型连接器是插件，工具、技能、会话、沙箱、存储、agent循环、调度和界面也都是插件。

时间可组合性

组件卸载后，副作用可完整撤销。
插件可以随时加载、卸载、替换，不会留下脏状态或残留影响。

空间可组合性

组件依赖关系可动态解析。
插件之间的依赖不需要硬编码，运行时自动发现和注入，支持复杂的组合拓扑。

革命性在于：不是「又一个Agent框架」，而是「用来造Agent框架的框架」

Deep Seek Harness 核心机制

核心包结构（7个包）

Session（会话日志）、system prompt（系统提示组装）、Tools（工具系统）、Agent（Agent注册与接口）、Agent Loop（智能体循环）、Scope（作用域注册）和LLM（模型适配接口），每一个都可以被独立替换或重写。

四种Runtime模式

标准模式 • 极简模式 • 代码模式（PTC） • 创作者模式
不同模式对应不同的Agent Loop实现和工具集配置

沙箱：跨平台统一封装

本地后端通过Node插件封装：

Linux Landlock • macOS Seatbelt • Windows ACL

模型无关

提供商目录涵盖：

Anthropic • OpenAI • AWS Bedrock • Azure • Gemini

会话日志：只追加写入

所有请求内容可还原，不可篡改。

建立在此之上的六种能力：

- 恢复：中断后从断点继续；
- 分叉：从任意节点创建新分支；
- 回放：重新执行历史任务；
- 转录：完整记录执行过程；
- 遥测：提取运行性能指标；
- Web界面：可视化监控执行；

独特能力：管控其他Harness

可把Claude Code和Codex作为子智能体收编调度

DeepSeek做上层编排，其他Harness做下层执行

野心：不仅自己做Harness，还能「管控」别人的Harness

目录

01  Agent Harness的概念与本质

02  智能编程Harness主流方案

03  个人助理Harness典型框架

04  GitHub热门Harness开源项目

05  Harness的选型指南与发展趋势

三大个人A I助理

项目	核心定位	关键组件
OpenClaw	跑在自己设备上的个人AI助理，通过各种聊天/设备入口驱动任务	Local-first Gateway •Channels •Nodes •Skills •Plugins
Hermes Agent	会积累记忆、生成/改进skills、跨平台运行的自改进Agent	AI Agent Loop •Tool Registry •SQLite/FTS5 •Skills Learning •Gateway
OpenHuman	UI优先、连接个人账号数据并构建长期记忆树的桌面AI助理	Tauri/Rust Core •Memory Tree •OAuth Integrations •MCP/Socket.io

三个方向，三条路线

OpenClaw入口与控制平面

让Agent住在用户已有的设备和聊天渠道里，随时被唤起

Hermes Agent自我演化与执行运行时

让Agent从一次次执行中学到东西，把经验沉淀为可复用能力

OpenHuman个人上下文与产品体验

通过118+集成和自动同步，让Agent分钟级内真正理解用户

(4) Open Claw设计理念与架构模式

设计理念：本地优先的个人AI控制平面

OpenClaw上层对接多渠道输入，下层连接会话路由、插件体系、记忆模块及运行时，核心则是一条标准化的执行主线。
Harness是OpenClaw为LLM打造的一个结构化执行外壳，让其真正获得了可扩展、可管控、能落地的Agent执行力。

架构模式：Gateway-centric

Gateway控制平面

长期运行 · WebSocket

消息渠道接入、会话管理、工具执行、事件分发、节点连接、权限与路由
Gateway是长期运行的控制平面，所有入口通过WebSocket接入

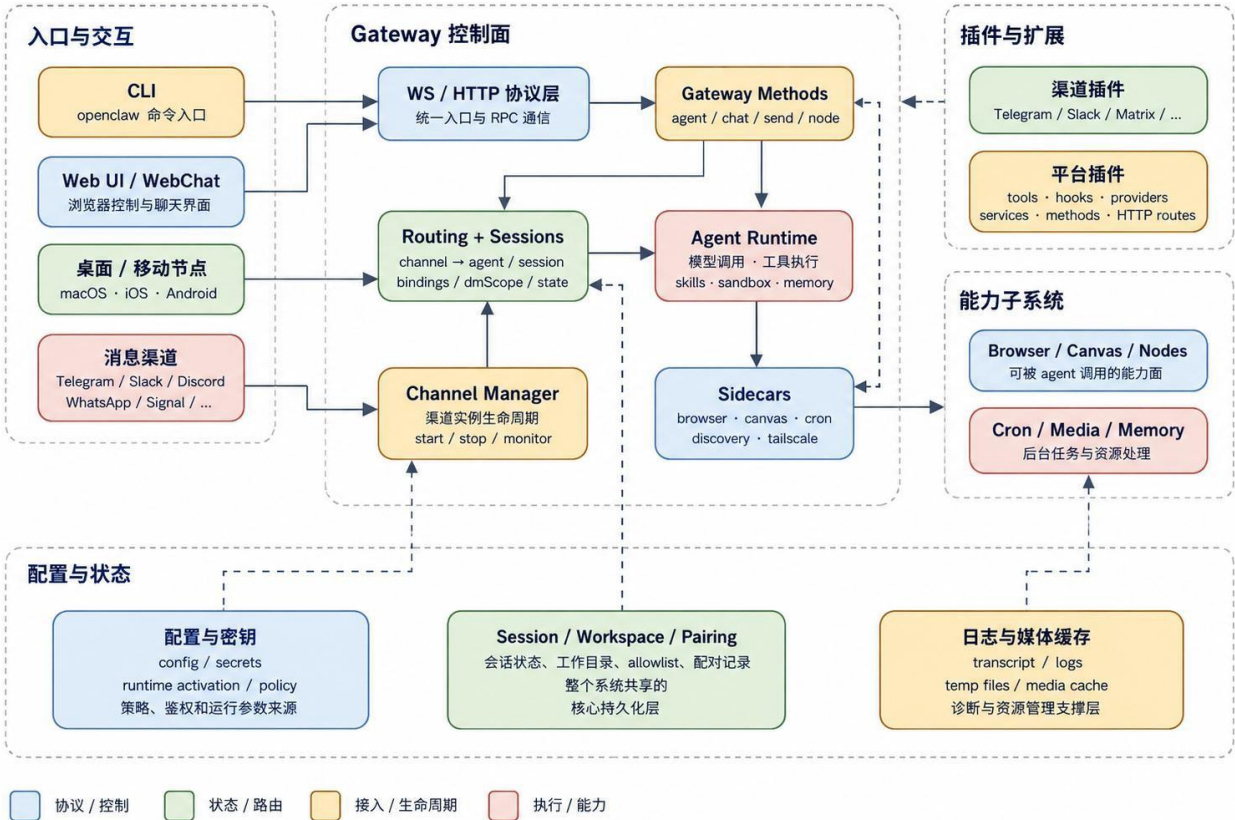
多端接入：设备即节点

客户端入口

CLI · WebUI · 自动化任务
全部通过WebSocket连接Gateway，会话可跨端流转

设备节点（可声明能力）

macOS · iOS · Android · headless nodes
接入后声明能力：canvas、camera、screen、location等
Gateway按需调度对应设备执行任务



Open Claw 核心机制

Agent Loop：六步串行流程

1. Intake接收用户输入

2. Context Assembly组装上下文

3. Model Inference模型推理

4. Tool Execution工具执行

5. Streaming流式返回

6. Persistence持久化记录

每个session串行执行，避免工具调用和transcript写入互相打架



技术底座

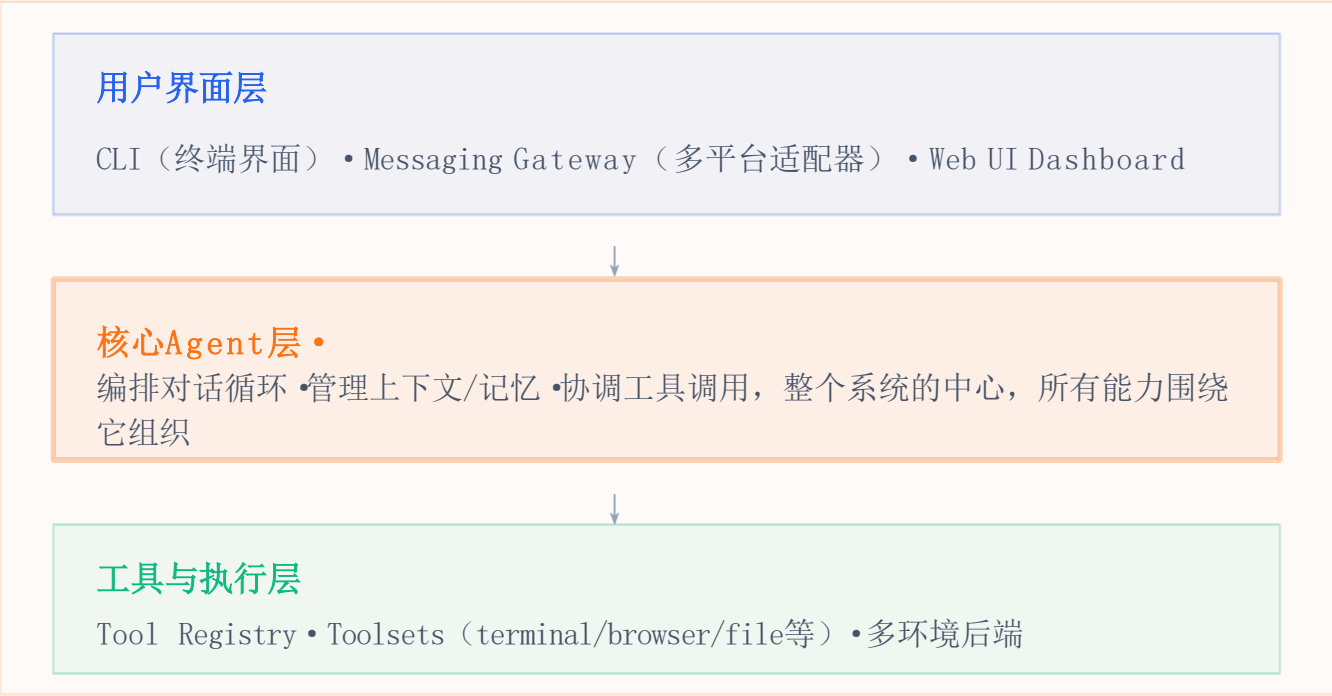
Node.js+WebSocket Gateway，像一个AI助理操作系统入口层，价值在于把用户已有入口整合起来

(5) Hermes Agent 设计理念与架构模式

设计理念：Self-improving • 自我改进

官方定位「The agent that grows with you」，拥有内置学习闭环的Agent，从经验中创建skills，在使用中改进它们。

架构模式：Agent-loop-centric



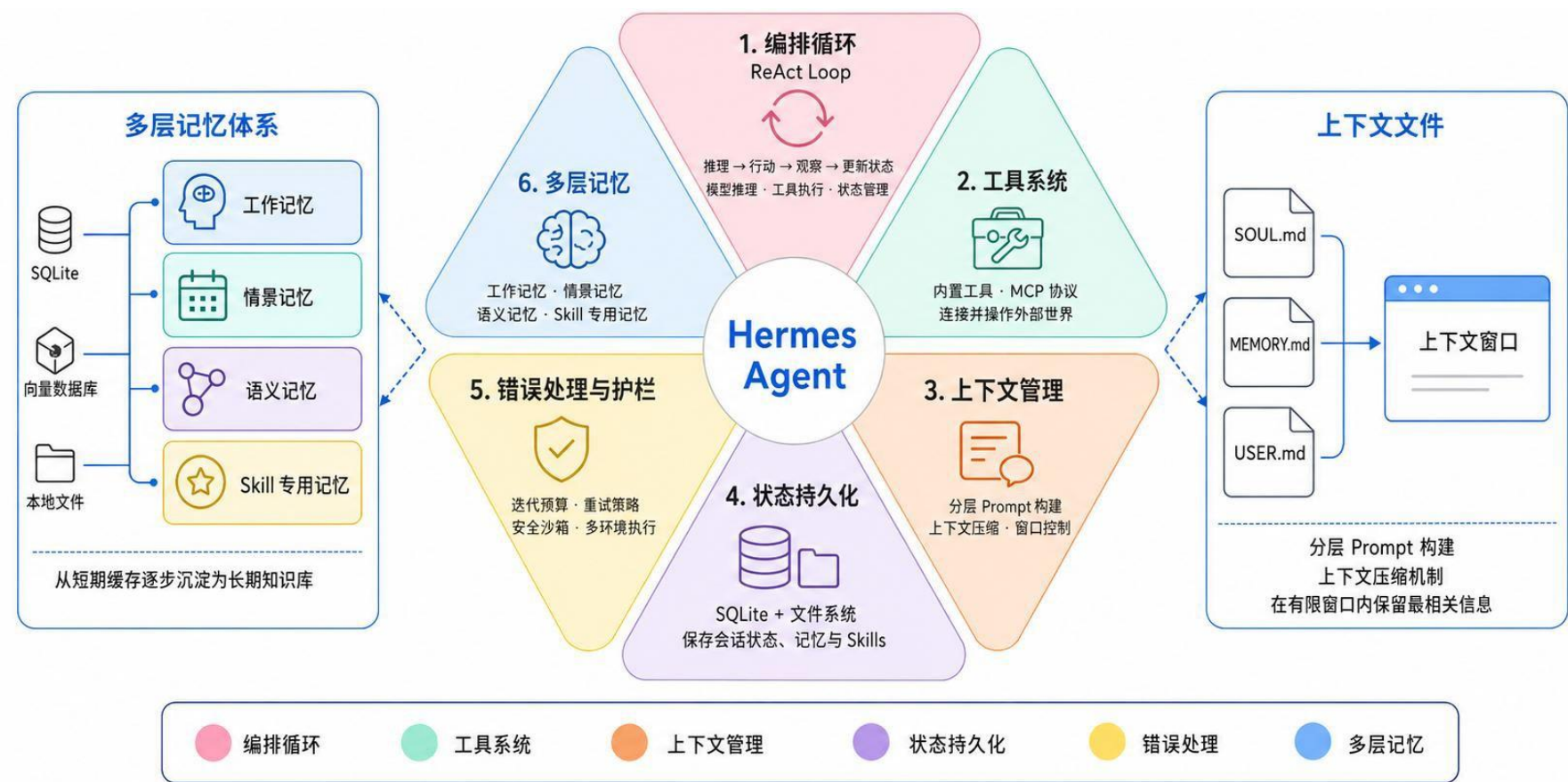
多种入口



Hermes 的定位是「会成长的Agent」——让Agent从经验里变强

Hermes Agent：Harness 架构的六维分解

Hermes Agent的Harness由六个核心维度构成：编排循环驱动ReAct主循环，工具系统连接外部世界，上下文管理在有限窗口内保留最相关信息，状态持久化保存会话与记忆，错误处理与护栏保障安全运行，多层记忆从短期缓存逐步沉淀为长期知识库。六者协同，将大语言模型从推理引擎转化为可自主执行的智能体。

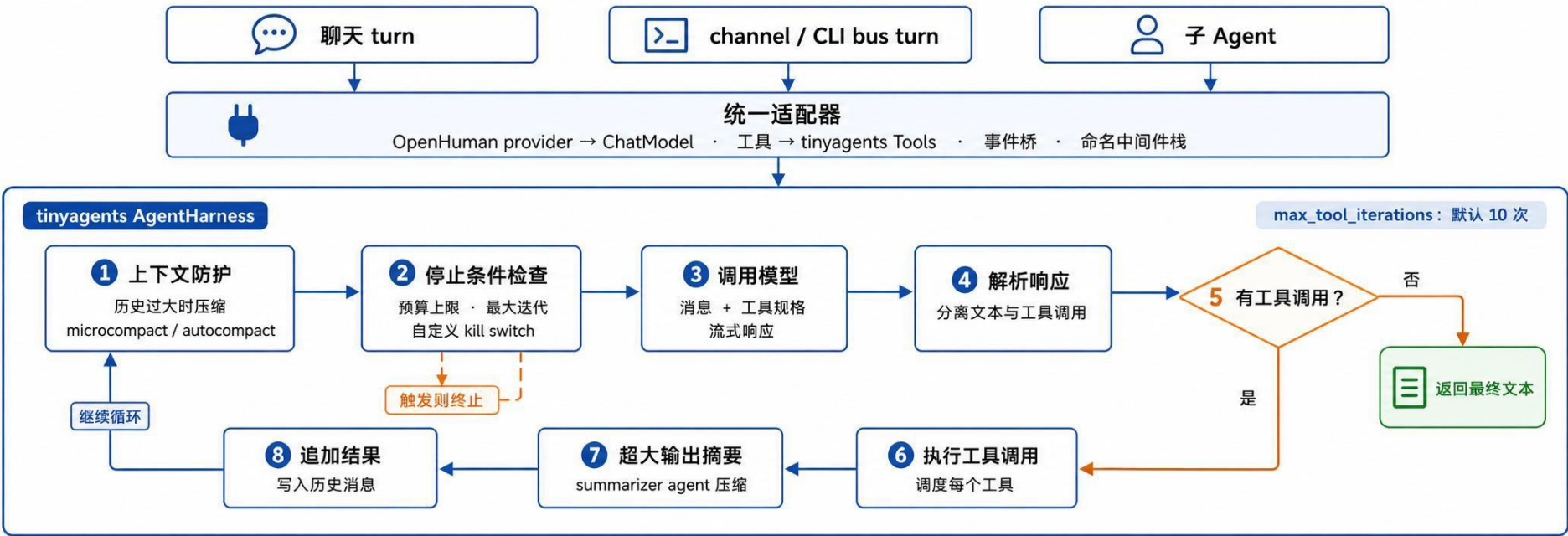


(6) Open Human设计理念与架构模式

设计理念：UI-first，个人数据驱动

一个AI助手只有具备了用户的上下文信息，才能真正发挥作用，在用户输入第一个提示之前就已经了解用户。

核心：工具调用循环



- 一次引擎，三个入口：循环只存在于tinyagents AgentHarness，聊天turn、channel/CLI busturn、子Agent都通过适配器接缝驱动，避免多套循环逻辑漂移。
- 中间件栈：审批安全门控 · 工具策略与CLI/RPC专用拒绝 · 格式错误参数恢复 · 成本预算预检查 · 重复工具失败断路器 · 上下文修剪压缩。
- KV-cache优先设计：system prompt仅首轮构建，后续直接复用渲染结果；动态上下文（记忆召回、新学片段）以用户可见消息追加，不破坏前缀缓存。

Open Human 核心机制

上下文治理：五层递进机制（长工具链防撑爆）

层级	机制	效果
①	TokenJuice 内容感知压缩	JSON→表格/代码折叠函数体/日志保留失败信号/Diff保留变更
②	Tool-result budget	超限硬截断+解释标记，让模型知道没看到完整输出
③	Microcompact/Autocompact	旧轮次压缩成摘要，保留system prompt和最近轮次（KV-cache稳定）
④	Summarizer detour	超大payload路由到专门子Agent按提取契约压缩，父Agent只看摘要
⑤	Filesystem offload	大结果移出上下文写磁盘，父Agent只拿路径指针，可file_read随时取回

关键设计：每次有损压缩都把原始内容offload到CCR存储，behind[[tj:<hash>]]标记→压缩实际无损，Agent可调用tokenjuice_retrieve按需取回完整内容。

多Agent编排

为什么要拆：单Agent system prompt过大；拆成专家可获窄prompt、过滤工具、历史不泄漏。

三级防递归爆炸



- chat→可生成reasoning/worker，不能生成chat
- reasoning→可生成worker，不能生成reasoning/chat
- worker→什么都不能生成
- 双重保障：加载时静态检查+运行时深度门控（MAX_SPAWN_DEPTH=3）

可靠性三机制+成本控制

机制	说明
No-progress 断路器	3次连续相同工具失败→Nudge→Halt
Sub-agent handback	Completed/AwaitingUser（检查点落盘，回答后恢复）/Incomplete
Classified Failure	MissingPermission/Timeout/Denied→映射UI 状态（自动重试/需确认/拒绝）
成本控制： 每次响应带charged_amount_usd，TurnCost累加→预算stop hook自动切断失控turn	

目录

01  Agent Harness的概念与本质

02  智能编程Harness主流方案

03  个人助理Harness典型框架

04  GitHub热门Harness开源项目

05  Harness的选型指南与发展趋势

其他热门开源Harness项目

除了Claude Code、Codex这类主流编程智能体，GitHub上还活跃着一批高热度的开源Harness项目。

OpenCode

多Harness协作的结构化编码框架，强调多Harness协作与结构化编码 workflow。

CLI 框架

Stars: 205k+

开源AI编程Agent赛道最高星标，模型无关（75+ 提供商）

Pi

极简、可自我扩展的编码Agent，小内核+大扩展，Agent可自行编写扩展。

CLI/SDK

Stars: 102k+

核心仅4个原子工具（read/write/edit/bash），押注Agent自己的双手

OpenHands

AI软件开发Agent平台，事件驱动架构，内置评估Harness，可观测可回溯。

终端/浏览器

Stars: 86.8k+

提供事件驱动架构、工具调用、沙箱环境与软件开发 Agent 能力

Open Interpreter

面向低成本开放模型的轻量编码Agent，多Harness仿真，Rust重写，兼容Codex协议。

终端CLI

Stars: 68.3k+

新版进一步强化开放模型与轻量化运行

Clin

IDE原生的AI编码助手Harness，可自主规划、读写代码。

VS Code 插件/SDK

Stars: 67.7k+

可自主规划、读写代码、调用终端与浏览器，并通过审批机制控制高风险操作

Goose

Linux基金会托管的本地优先通用Agent框架，微内核+MCP标准协议，完全离线运行。

桌面/CLI/API

Stars: 54k+

支持 MCP 等扩展协议，可连接工具、服务与开发环境，在本地完成多步骤任务

(7) Open Code 概述

OpenCode是一款开源编程智能体，可通过终端界面、测试版桌面应用或IDE扩展使用。其目录支持75+提供商，包括DeepSeek以及本地模型。Build是其具备完整访问权限的开发智能体，Plan则限制为分析用途。General、Explore和Scout子智能体分别覆盖复杂任务、只读代码库搜索和依赖项研究。权限、LSP、MCP 服务器、插件和自定义工具可扩展工作流，无需重新组合其核心架构。

基本信息

开源协议：MIT

GitHub Stars: **205k+** (开源AI编程Agent最高星标)

技术栈：TypeScript

运行形态：CLI (TUI) + 桌面App

核心理念

- 将模型选择权、成本控制权与数据主权交还给开发者。
- 模型无关：支持75+种模型提供商 (Claude、OpenAI、Google、本地模型等)
- 不做供应商锁定，可随时切换模型，甚至在同一任务中混用

为什么它如此受欢迎？

开源+MIT协议，可自由商用和修改
模型选择自由度高，75+提供商，不绑定任何云服务
社区插件生态丰富，允许深度定制工作流

Open Code 核心架构

C/S架构（客户端-服务器）

与大多数单一进程的CLI工具不同，OpenCode采用C/S架构，基于TypeScript和Bun运行时构建，整体分为客户端层、核心服务层、扩展层、模型适配层：

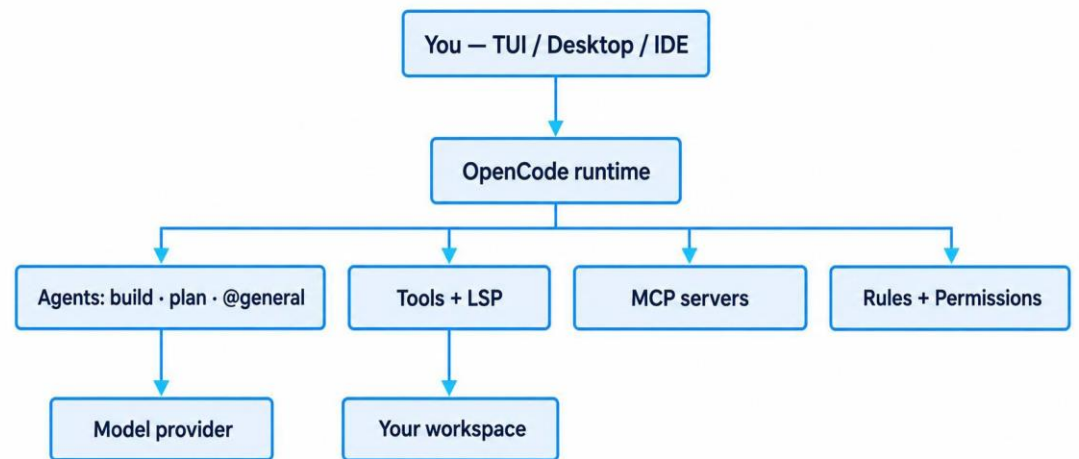
- **客户端层**：终端TUI（核心）、桌面应用、IDE扩展（VS Code/Cursor等），负责用户交互与指令输入；
- **核心服务层**：代理调度、任务管理、工具引擎，是OpenCode的大脑，负责任务拆解与执行；
- **扩展层**：插件系统、配置管理，负责功能扩展与个性化适配；
- **模型适配层**：多模型兼容接口、本地/远程模型调度，负责与LLM提供商的对接与适配。

OpenCode虽以终端TUI为主要交互方式，但架构设计支持多端扩展（桌面端、IDE插件），本质是“**核心服务复用+多端交互适配**”，这也是其能无缝融入不同开发者工作流的关键。

多智能体系统（Multi-Agent）

内置多个拥有特定权限集和系统提示词的专用Agent：

- **主Agent**（Tab切换）：Build Agent（全权限代码实现）、Plan Agent（只读架构规划）
- **子Agent**：General Subagent（复杂多步任务）、Explore Subagent（并行探索代码库）



(8) Pi Agent 概述

Self-Extensible Coding Agent

Pi是一个开箱即用的终端编程智能体，但每个零件都可拆、可换、可由Agent自己生成。

维度	说明
开发方	Earendil Inc.
开源协议	MIT
技术栈	TypeScript (Monorepo)
运行形态	CLI终端TUI (可直接使用) + SDK (可嵌入)
GitHub Stars	102k+ (高热度)
核心目标	提供轻量灵活的Agent执行环境，开发者可像搭乐高一样构建智能体

核心理念

[There are many agent harnesses, but this one is yours.]

Pi的目标是适应你的工作流，而不是让你去适应它

核心原子工具 (仅4个)

read

write

edit

bash

通过Unix理念式的组合完成复杂任务
而非为每个场景堆砌专用工具

Pi与其他项目不同

它不是功能大而全的成品，而是为开发者准备的积木

Pi Agent 技术架构与核心能力

四种运行模式

01 交互模式

默认，完整TUI界面

02 非交互模式 (Print/JSON)

用于脚本、管道或单次任务

03 RPC模式

通过stdio JSON协议通信，方便非Node.js应用集成

04 SDK模式

作为库嵌入你自己的应用

Extension扩展机制

使用TypeScript编写扩展，添加自定义工具、命令甚至修改行为；子Agent、权限控制等Pi不内置的功能均可通过Extension实现——「可选功能」被刻意排除，由社区通过扩展自行提供

Monorepo核心包结构

包	职责
<code>@earendil-works/pi-ai</code>	统一LLMAPI层，支持15+家模型提供商（OpenAI/Anthropic/Google等）
<code>@earendil-works/pi-agent-core</code>	Agent运行时核心（状态管理、工具调用、Agent Loop）
<code>@earendil-works/pi-coding-agent</code>	交互式编码Agent CLI（用户直接交互的工具）
<code>@earendil-works/pi-tui</code>	终端UI库

Pi vs. Deep Seek Harness

Pi和DeepSeek Harness这两个开源Agent框架，都有一个核心特点：可扩展，区别是Agent自己长出手脚还是把每项能力外包给社区。

- Pi: 给房子添家具
- Agent Loop是承重墙，不可替代
 - 扩展=往房子里加家具
 - 编写者: Agent自己写

- DeepSeek Harness: 可以换户型
- Agent Loop是插件之一，整面墙都能换
 - 扩展=换掉整层能力
 - 编写者: 社区开发者分工

对比维度	Pi	DeepSeek Harness
口号	Self-extensible（自我扩展）	Everything is a Plugin（一切皆插件）
内核大小	小: Agent Loop+基础工具	更小: 只剩「装插件」这一个动作
Agent Loop	承重墙，不可替代	插件之一，整面墙都能换
扩展本质	给房子添家具	可以换户型
扩展编写者	Agent自己写	社区开发者分工
扩展粒度	工具、命令、UI组件、模型提供商	整层能力（LLM层、权限层、沙箱层）

(9) Open Hands 概述

OpenHands 由 All Hands AI (前身为 OpenDevin) 开发, 通过 Agent-Compute 接口, 在 shell、代码编辑器和浏览器之间运行一个完整的循环, 并提供一个 SDK 用于在其基础上进行构建。

开源根基

由社区主导, 采用 MIT 宽松协议, 完全开源且商业友好, 支持自由分发与二次开发。

架构理念

无状态事件驱动设计, 搭配受控沙箱环境, 保障执行安全, 支持高并发与灵活扩展。

平台定位: 不仅是 Agent 框架, 更是为 AI 软件开发者量身定制的全栈工程化平台。Server 与 Client 解耦设计, 打通从需求分析到代码交付的闭环。

三大核心能力引擎



代码智能生成与重构

支持多语言代码自动编写、文件实时读写与代码优化, 精准理解开发意图, 生成符合工程规范的可运行代码, 大幅提升研发效率。



系统级命令与环境操作

无缝执行 Shell 命令、安装依赖库、运行自动化测试与项目构建, 具备完整的系统环境操作权限, 模拟真实开发流程。



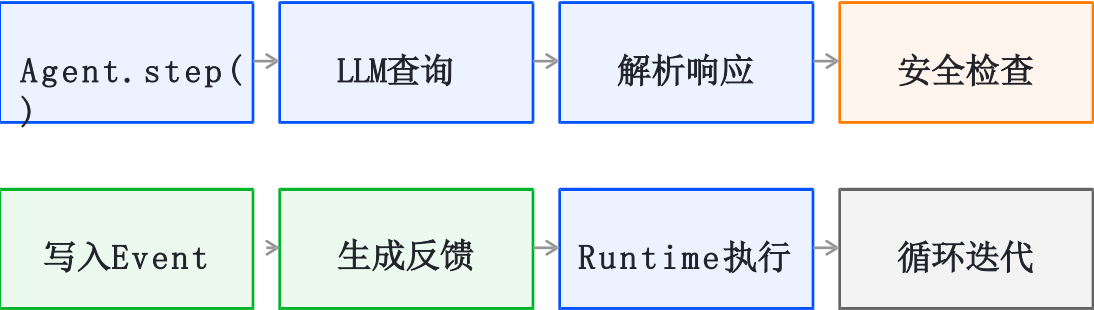
自主网页浏览与文档抓取

内置浏览器能力, 可自主访问网页、抓取技术文档与 API 说明, 解决开发过程中的信息获取与上下文理解难题。

Open Hands 核心架构

核心组件	核心职责	关键特性与价值
LLM交互层	模型统一接入与交互	基于LiteLLM封装，无缝兼容多种主流模型，提供统一调用接口，屏蔽底层差异。
Agent决策层	系统的“智能大脑”	读取State状态，生成Action指令，支持ReAct、Plan-and-Execute等多种决策模式。
AgentController	循环驱动引擎	初始化Agent实例，管理状态流转，驱动“思考-行动”的主循环，确保流程有序执行。
State状态层	系统的“工作记忆”	存储当前步骤、历史事件记录、长期计划与上下文信息，支持状态持久化与回溯。
EventStream	事件中枢神经	基于发布/订阅模式，实现组件间解耦通信，确保全链路可观测、可追溯。

01推理-行动闭环(Reasoning-Action Loop)



02核心事件类型

Action（主动请求事件）

Agent发出的操作指令，是系统的“行动指令”。例如：`write_file`（写文件）、`bash`（执行命令）等。它触发Runtime执行具体任务，是系统产生变化的源头。

Observation（反馈结果事件）

执行后的状态反馈，是系统的“感官输入”。包含命令输出、文件内容、环境变化等。Agent基于此更新State并进行下一步决策，形成闭环的关键反馈环节。

(10) Open Interpreter 概述

为低成本开放模型打造的轻量级编码Agent，可用自然语言控制计算机。基于OpenAI Codex CLI深度改造，用Rust从零重写。

开发主体
Open Interpreter开源社区

开源协议
Apache-2.0(商业友好)

核心技术栈
Rust 96.6%(高性能)

社区热度
68,300+GitHub Stars

演进路径
2023 Python原型→2025 Rust重构(极致性能)→2026 生态标准化

核心使命：填补开源模型的应用空白

Open Interpreter致力于打破壁垒，让开放权重模型成为AI应用的“一等公民”，打通从能力到落地的最后一公里。

产品理念：兼容并蓄，灵活演进

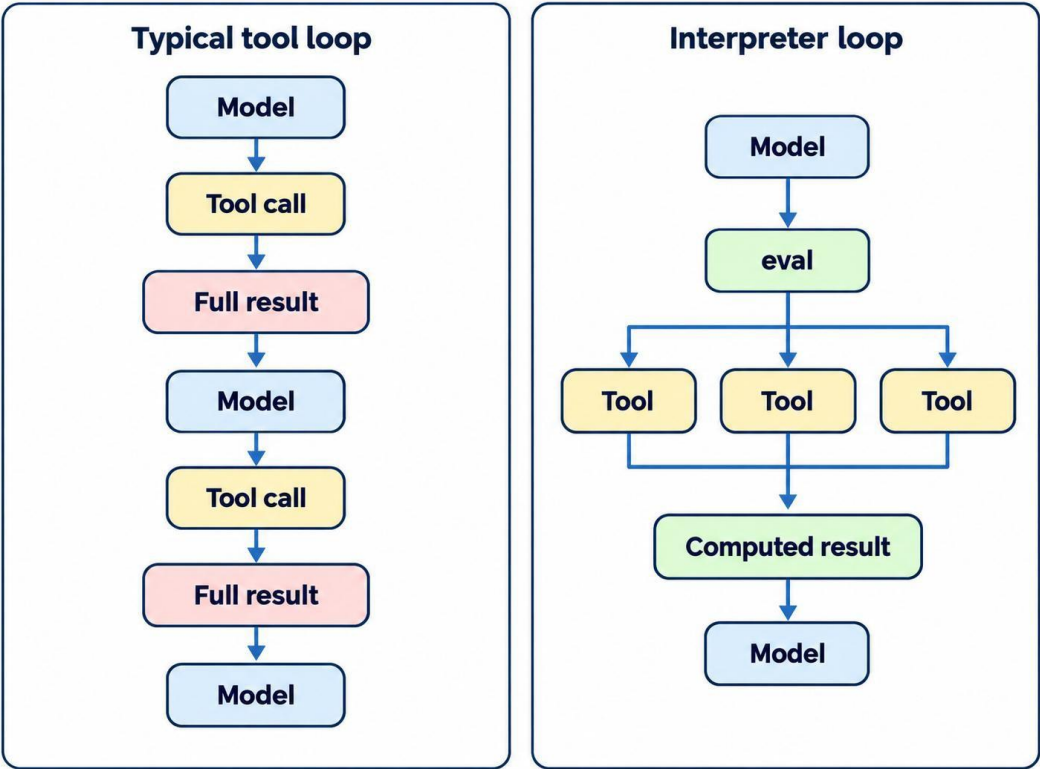
01 可移植优先(Portability First)
拒绝专属格式锁定，优先采用AGENTS.md、MCP、ACP等工具中立标准。确保Agent具备极强的跨平台迁移能力，不被单一生态绑定，让模型能力在任意环境中无缝复用。

02 Codex协议无缝兼容
兼容OpenAI Codex执行协议，可直接替换SDK底层二进制。企业无需重构代码架构，即可实现从闭源API到开源模型的低成本迁移，大幅降低落地门槛。

Open Interpreter 系统架构与循环

典型工具循环中，模型每轮输出一工具调用，Harness解析并执行后将完整结果回灌上下文，N步任务对应N次模型调用，成本与延迟随任务深度线性增长；Open Interpreter则是模型写代码，代码连续调用工具，精简结果返回，N步可能仅需一两次调用。

两种Agent执行循环模式



五层系统架构(Rust Monorepo)



(11) Cline 概述

IDE原生的AI编码助手Harness

Cline是从VS Code编程助手演化而来的开源、可插拔Agent运行时，其核心是将原先与IDE紧耦合的Agent逻辑重构为独立可复用的TypeScript 分层SDK ， 从而实现Agent Loop的无头化（headless）。



开源归属

Cline社区(原claude-dev)



开源协议

Apache-2.0商业友好



核心技术

TypeScript全栈开发



社区热度

67,700+GitHub Stars



多元运行形态

VS Code插件 • 独立CLI工具 • @cline/sdk开发包

演进路径与产品理念



01 起源：IDE插件起步

始于claude-dev，专注于VS Code编辑器内的智能补全与对话，解决编码场景下的即时需求，成为开发者的贴身助手。



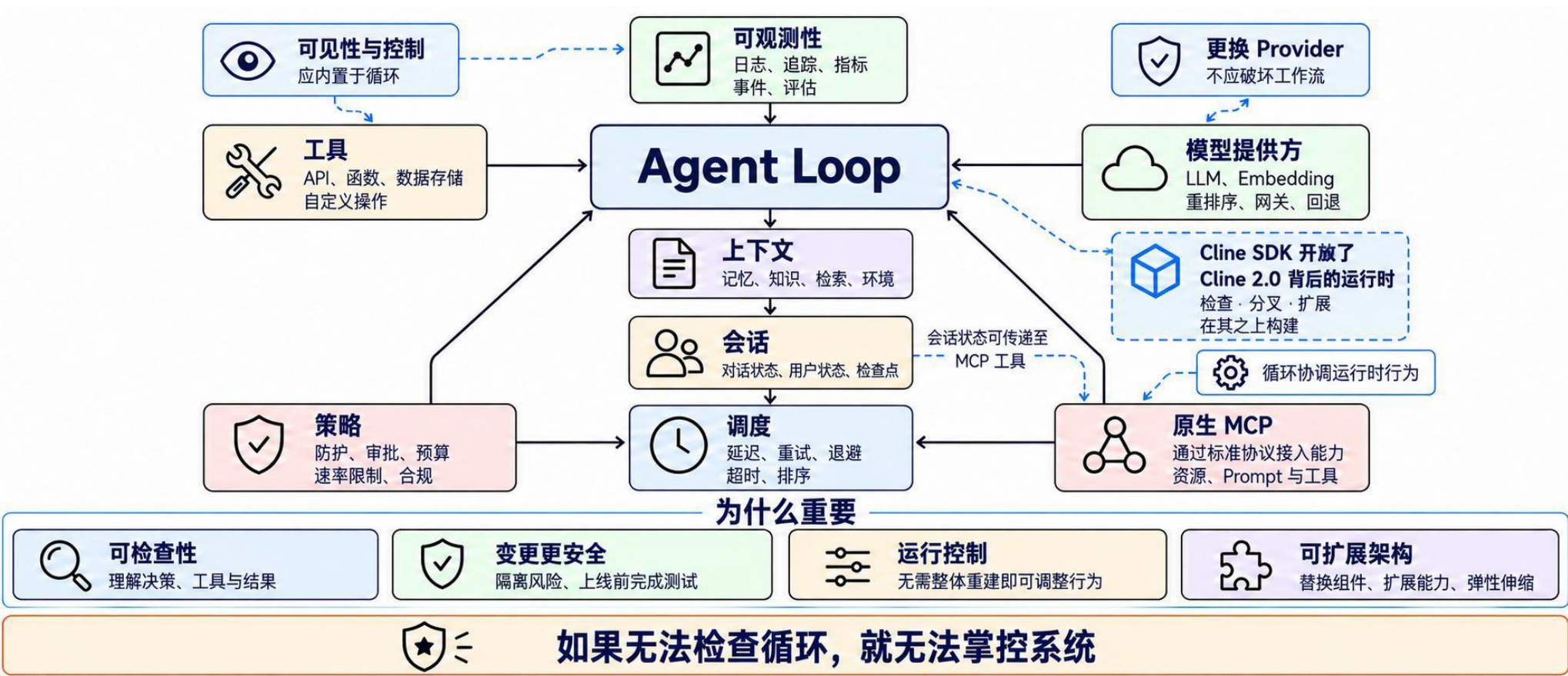
02 进化：SDK化引擎

推出@cline/sdk，将AI Agent能力封装为可复用的开发库，支持脚本调用、自定义工具集成，实现从“工具”到“能力”的跨越。

“The loop is served on a silver platter , the rest is up to you.” 提供稳定的Agent循环脚手架，开发者通过插件与钩子自由定义逻辑，实现高度定制化。

Cline 核心循环

Agent Loop是Harness的中枢，协调工具、模型、上下文、策略、调度、可观测性等组件协同运行。运行时不应是黑盒，可检视性是可控性的前提，若无法检视循环，便无法控制系统。Cline SDK 2.0将运行时开放为可编程接口，支持检视、复刻、扩展与自定义构建。



(12) G o o s e 概览

Goose的核心定位是将AI能力从云端带到用户本地机器，通过桌面应用、CLI和API三种方式交互，支持自主规划与执行复杂任务。

开源治理体系

由Linux基金会托管，确保项目中立性与长期可持续发展，避免单一厂商锁定。

Rust 高性能内核

底层采用Rust语言构建，兼顾内存安全与极致性能，支持跨平台高效运行。

54k+社区星标

GitHub上获得超5万开发者关注，社区贡献活跃，生态扩展迅速。

本地优先运行

支持本地优先与离线部署；使用Ollama 等本地模型时，可在无需云端模型服务的情况下运行。

多模型生态兼容

无缝适配 15+主流AI提供商，涵盖Anthropic、OpenAI、Ollama等。

多端部署形态

覆盖桌面App、CLI工具、REST API及ACP协议，适配多样化场景。

01. 核心理念：On-MachineAI Agent

“Goose is your on-machine AI Agent”

打破传统云端AI的限制，实现真正的本地化运行。无需依赖云端服务器，通过Ollama等本地模型即可构建闭环，数据全程留存本地，杜绝隐私泄露风险，实现完全离线的智能体验。

02. 产品理念：微内核+插件总线

“内核只管最少的事，真正的能力全靠插件挂载”

采用微内核设计，核心仅负责基础的调度与编排，所有功能模块（如模型调用、工具集成、工作流编排）均通过插件总线动态扩展。这种架构赋予了系统极高的灵活性，可根据业务需求自由组合功能，适配从个人助理到企业级自动化的各类场景。

Goose 架构体系

01 接口交互层

组件： CLI (goose-cli) 、 Electron 桌面端、自定义接口；提供多端统一的交互入口，支持命令行与可视化操作并行。

02 通信协议层

组件： REST API、ACP 双向通信协议；基于 HTTP 与 JSON-RPC 的高效通信，保障客户端与服务端的实时数据交互。

03 核心逻辑层

组件： ExtensionManager 、 SessionManager；系统的大脑，负责业务编排、权限管控及插件生命周期管理。

04 生态集成层

组件： MCP Servers (goose-mcp)，实现“一切皆 MCP”，标准化插件接口，无缝对接第三方能力。

核心Harness组件矩阵

Agent 中央编排器

统筹对话流与工具执行，负责子 Agent 协作、权限安全与任务分发。

ExtensionManager 扩展管家

支持热插拔扩展，自动发现工具与缓存，处理 OAuth 认证流程。

ProviderRegistry 模型抽象

统一多家 AI 提供商接口，兼容自定义插件，实现模型层解耦与灵活切换。

SessionManager 会话管家

持久化对话历史、工作目录与配方 (Recipe)，支持会话断点续传与恢复。

Exchange 执行引擎：智能调度核心



高性能 Rust 驱动

将 Rust 函数挂钩至模型工具循环，实现低延迟的任务调度与执行。



全链路错误感知

精细的错误处理机制，确保工具调用失败信息完全暴露给模型，辅助决策。



极简调用范式

运行流程本质为 `exchange.reply()` 调用，封装复杂通信细节。

目录

01  Agent Harness的概念与本质

02  智能编程Harness主流方案

03  个人助理Harness典型框架

04  GitHub热门Harness开源项目

05  Harness的选型指南与发展趋势

Harness 的选型指南

成熟度高的
编程智能体

Claude Code • Codex

Claude Code：开箱即用、终端原生、强自主编码体验；Codex：沙箱安全、SDK 嵌入、平台集成优先。

可深度定制的
Harness 底座

DeepSeek Harness • Pi • OpenCode

DeepSeek：替换模型/工具/Loop/沙箱等底层组件；Pi：极简内核 + Extension 自由扩展；OpenCode：多模型 + 成熟插件生态。

完整的软件开发
Agent 平台

OpenHands • Cline • Open Interpreter

OpenHands：完整软件开发执行平台；Cline：VS Code / IDE 场景 + 可嵌入 Runtime；Open Interpreter：开放模型 + 本地执行降成本。

常驻型个人
AI 助理

OpenClaw • Hermes Agent • OpenHuman

OpenClaw：多聊天渠道 + 设备节点 + Gateway 控制平面；Hermes：长期记忆 + Skills 自我改进；OpenHuman：个人数据整合 + 桌面 UI。

本地优先的
通用 Agent

Goose

强调本地运行、多模型兼容、MCP 扩展以及桌面 / CLI 多入口的通用任务场景，Linux 基金会托管生态最开放。

没有最好的 Harness，只有最适合自身场景的 Harness

先明确需求场景，再匹配产品特性——选型是维度匹配，而非追逐热门

发展趋势

01 Harness轻量化

模型内化通用能力，聚焦差异化治理与安全合规

行业长期趋势是Harness持续轻量化，模型与调度边界逐渐消融，通用调度能力将内化进模型原生行为。而企业专属权限管控、全链路审计、沙箱安全等差异化合规能力，仍将保留在Harness层。未来架构将形成「**模型内生推理+轻量化治理**」的协同模式，既释放模型智能潜力，又保障系统的安全可控与业务合规。

02 Harness-as-a-Service

托管服务降低门槛，构建智能体快速开发通道

针对自建Harness架构门槛高、运维负担重的痛点，HaaS将完整的Agent运行基座封装为托管服务。**开箱即提供任务循环、上下文管理、工具管控与安全校验等核心能力**，开发者无需自研底层调度框架，依托API即可快速搭建智能体，大幅降低开发成本，让团队更专注于业务逻辑设计与场景创新落地。

03 模型无关与自适应适配

打破单一绑定，实现多模型灵活切换与技术中立

构建多模型自适应层，打破对特定厂商的依赖，实现「模型即插即用」。通过统一适配层自动兼容不同模型的提示范式、工具调用格式及接口规范。开发者可根据任务特性（推理/创意/分析）自由选择最优模型，无需重构Agent框架，兼顾效果与成本，实现**真正的技术中立与弹性扩展**。

四大核心权衡

自主性↔管控Autonomy vs Control

高自主权限：干活快，但风险高；处处审批：安全但效率低。

- 不同任务类型的最优平衡点在哪里？
- 如何设计高效校验器，不成为效率瓶颈？

宽泛上下文↔精选上下文Broad vs Curated

全量塞入：简单粗暴，但模型被信息淹没；智能筛选：耗算力，但效果更好。

- 什么样的上下文筛选策略性价比最高？
- 如何单独评测上下文管理能力，排除模型干扰？

通用型↔专用型Generalist vs Specialized

通用Harness：一套系统搞定所有任务；专用Harness：领域定制，效果更强。

- Harness哪些部分可以跨领域复用？
- 是否存在通用控制内核，可外挂领域扩展？

开放权限↔强隔离容器Open vs Containment

本机权限：操作自由，但风险大；沙箱隔离：受限运行，但难搞破坏。

- 如何兼顾沙箱安全和任务实用性？
- 有哪些可跨项目复用的隔离模式？

未来的技术差距，可能不再主要来自大模型本身，而是来自Harness这层外围工程。好的Harness可以适配多种底层模型，弱化对单一昂贵大模型的依赖。

版权声明

一、内容版权说明

文字内容：本PPT所有文字内容均基于公开渠道信息整理汇编，包括但不限于行业报告、学术论文、厂商白皮书、开源社区文档及公开新闻资讯，版权归原作者或发布机构所有。

图片与素材：所有图片、示意图、表格等素材均来源于公开网络资源（如各类无版权图库，或企业官网、行业分析报告的公开配图），仅用于科普交流，无商业用途。

二、知识产权声明

本PPT内容为知识整理与二次创作成果，不涉及对任何企业、机构或个人的商业信息、技术专利的侵权。若部分内容无意中侵犯了您的知识产权，请通过以下联系方式告知，我们将立即删除或补充版权说明。

三、免责条款

本PPT仅用于技术交流、学习分享，禁止用于商业用途或盈利性传播。内容力求准确，但不保证完全无误，对因使用本PPT内容引发的任何直接或间接损失，不承担法律责任。图片与数据仅供参考，具体以官方最新发布信息为准。

四、联系方式

如有任何问题，请联系我们。