技术蓝皮书 · Technical Blue Paper

DeepSeek Harness

架构 · 机制 · 生态

摘要

本蓝皮书介绍 DeepSeek Harness（dsh）——DeepSeek AI 开源、用来给大模型装上「手脚」、让它真正把事办成的智能体运行框架。它以「一切皆插件」为核心：不止能力是积木，连指挥模型干活的循环本身都可拆换；再配上一台记录每一步的日志黑匣子，做到可回放、可审计、省 Token，并支持多智能体协作与「运行时改装自己」。本文面向开发者、技术决策者与研究社区，用尽量通俗的语言讲清架构、机制、能力生态、安全与路线图。

版本	v1.0（对应源码 0.1.0-rc.5）
日期	2026 年 8 月
状态	开发者预览（Developer Preview）
许可证	MIT
目标读者	开发者、技术决策者、AI 研究者

目录

1 摘要

一分钟读懂

先用三个生活化类比建立直觉，后面每一章都在为这三句话补细节：

1. **它是什么**：裸的大模型只会「你说一句、我答一句」；Harness 是给它装上「手」和「脚」的通用底座——让模型能读文件、跑命令、上网、委派子任务，把一件事**真的做完**；
2. **它怎么拼**：整个 Harness 像一套**乐高**。不止能力是积木，连「指挥模型干活的循环」本身也是一块可拆换的积木，没有一块是焊死的；
3. **它怎么保证靠谱**：模型看到的每一句话都被写进**黑匣子日志**，可回放、可审计、省 Token，还能随时「分叉/恢复」会话。

DeepSeek Harness（下文简称 Harness，命令行工具记作 `dsh`）是 DeepSeek AI 于 2026 年 8 月开源的智能体（Agent）运行框架。与仅能「根据文字生成文字」的大语言模型（LLM）不同，Harness 为模型装配了可执行能力（文件读写、终端、检索、子智能体等），并提供一套「思考 调用工具 观察结果 再思考」的循环驱动器，使模型能够真正完成跨越多个步骤的实际任务。

Harness 的核心主张是「**一切皆插件（Everything is a plugin）**」：模型适配器、工具注册表、会话日志、沙箱与审批策略，乃至智能体循环本身，全部实现为可替换的插件。这一主张建立在底层 Cordis 插件框架之上（Cordis 的设计范式被学术界概括为「时空可组合」，通俗说就是**任何部件都能在运行时被替换、组合与撤销**）。基于此，Harness 实现了三项区别于多数同类框架的关键性质：

1. **没有焊死的核心**：不存在需要「打补丁」的核心黑盒，扩展即「把插件挂到别的插件旁边」，装上的东西卸载时自动还原；
2. **一份日志管到底**：会话的每一步都记成事件，存储、回放、生成上下文、排查问题全靠它，铁律是「模型看到了什么，日志里就一定有什么」；
3. **可拼装的多智能体**：既能派一个「子智能体」去干小活，也能用「工作流」把任务扇出给一大批子智能体并行干。

本蓝皮书的主体结构如下：第 2 章阐述问题背景；第 3–4 章给出愿景、定位与设计原则；第 5–7 章剖析系统架构、运行时机制与核心子系统；第 8–9 章梳理能力生态与多智能体特性；第 10–11 章讨论安全与开发者生态；第 12–14 章进行对比分析、场景归纳与路线图展望；第 15 章说明风险与局限；最后以附录提供术语与命令速查。

关于版本：Harness 当前处于开发者预览阶段，版本号为 `0.1.0-rc.5`，官方明确提示「将会有破坏兼容性的变更」。本蓝皮书对源码与文档的描述以该版本为准，具体接口、命令与默认值请以官方仓库最新文档为准。

2 背景与问题

2.1 从「对话模型」到「能办事的智能体」

大语言模型很会「说话」，但要完成一件真实任务（修 bug、重构代码、写调研、迁移数据），光会说话不够，还得能 **动手**：读文件、跑命令、看结果、再修正，循环直到把事情做成。裸模型自己做不到，需要一个「脚手架」给它配上手脚和循环——这就是智能体框架，也叫运行底座（Agent Harness）。

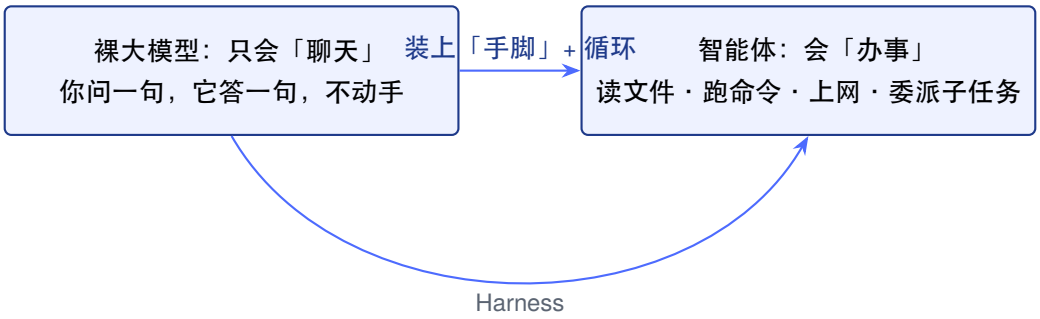

图 1: 智能体 = 大模型 + 运行底座（Harness 提供能力与循环）

2.2 传统方案的痛点

在自建智能体系统时，开发者通常反复遇到以下五类问题：

1. **循环逻辑琐碎且易错**：请求构造、工具调度、并发控制、错误重试、中止条件等需要自行实现，且容易产生细节分歧；
2. **能力难以插拔**：文件、终端、检索等能力一旦写死，替换（如本地文件切换为远程沙箱）需要逐处改造；
3. **上下文膨胀**：长任务的历史消息持续增长，导致 Token 消耗与延迟随任务长度线性恶化；
4. **可观测性不足**：智能体「为何如此决策」难以事后复盘、回放与审计；
5. **生态碎片化**：各框架的工具协议、插件接口互不兼容，迁移成本高。

Harness 的设计目标，即是用一套统一的**插件协议 + 事件系统 + 会话日志**，系统性地化解上述痛点。

3 愿景与定位

3.1 愿景

提供一个**可组合、可审计、可自演进**的智能体运行时：任何能力都可插拔，任何环节都可替换，任何决策都可回放，并允许智能体在运行时理解并修改自身的运行形态。

3.2 定位

Harness 是「智能体运行底座」而非「某一垂直应用」。它面向三类人群：

- **最终用户**：通过 Web UI 或命令行直接驱动智能体完成编程与办公任务；
- **应用开发者**：通过 Python SDK、ACP（Agent Client Protocol）或 JSON-RPC SDK 将 Harness 嵌入自有系统；
- **插件开发者**：以 npm 包（统一前缀 `@deepseek-ai/dsh-*`）发布可复用能力，参与插件生态。

在赛道对标上，Harness 常被与 Anthropic 的 Claude Cowork 类比，同属「面向编程/办公任务的智能体运行时 + Web UI + 插件生态」。其差异化将在第 12 章展开。

4 设计理念与核心原则

Harness 的全部设计可归结为一条主线与四条纪律：

主线：一切皆插件 模型适配器、工具注册表、会话日志、沙箱、审批，乃至智能体循环本身，皆为可替换插件。产品没有「特权内核」，扩展方式是把插件挂载到其他插件旁边。

纪律一：注册即副作用 提示词片段、工具 schema、适配器、监听器均通过 `ctx.effect()` / `ctx.on()` 安装，并在插件卸载时按预期撤销；每个注册都应有对应的资源释放函数（disposer）。

纪律二：依赖声明式 插件通过 `inject` 声明所需服务，等待其就绪后才启动；加载顺序由依赖表达，而非手动编排。

纪律三：模型可见即已记录 凡进入模型请求的内容，必须能从会话日志重建；新增一种模型可见输入，就必须新增一种会话事件。

纪律四：面向扩展点编程 新行为附加到已文档化的扩展点上；改变循环本身时才更新架构映射。

5 系统架构

5.1 分层总览

Harness 自底向上分为五层。底层是内置（vendored）的 Cordis 框架；向上依次是核心主干、能力 seam、应用组装与接入层。各层之间以「服务（`ctx.*`）+ 事件」解耦。

5.2 插件树与组装：Profile、组合包与 patch

可以把一个正在运行的 `dsh` 想象成一摞**透明胶片**：从下往上一层一层叠上去，**越靠上越有话语权**（后叠的可以覆盖先叠的）。这摞「胶片」就是**插件树**，每一层由三种东西之一提供：

- **组合包（Bundle）**：官方预先配好的一盒积木，比如 `dsh-base`（含模型、工具、存储、沙箱等基础件）；

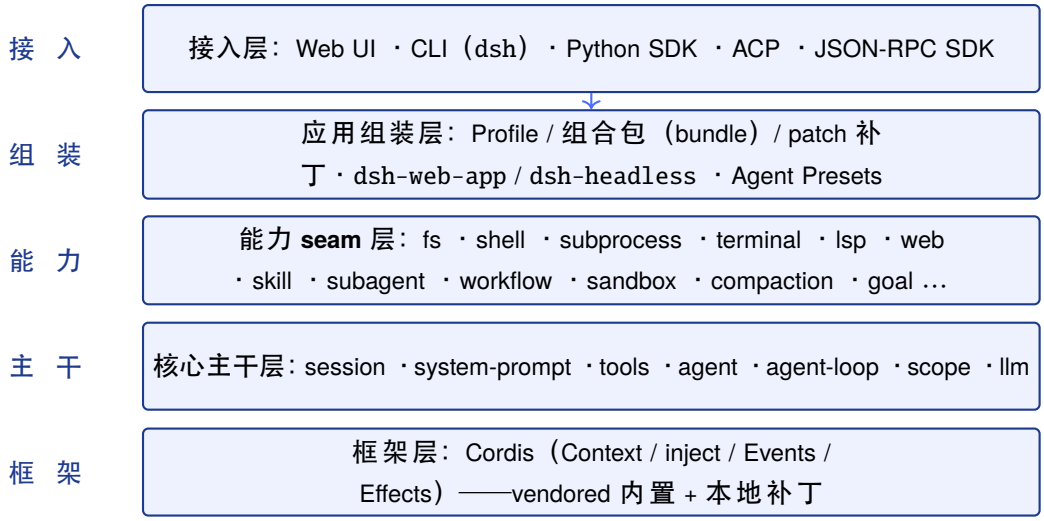

图 2: DeepSeek Harness 五层架构总览

- **Profile（配置集）**：一份「我要用哪几盒积木、按什么顺序叠」的清单，外加你自己的微调文件 `cordis.patch.yml`；
- **patch（补丁）**：一张「覆盖条」，能按 id 找到某块积木、替换它的设置，或塞进一块新积木。

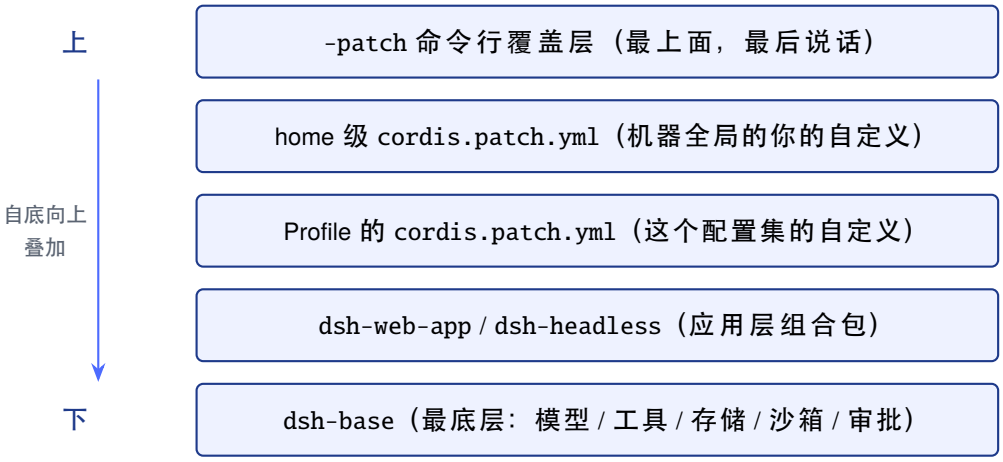

图 3: 插件树的「叠胶片」模型：越靠上，优先级越高

叠加顺序（自底向上）：先按 Profile 声明的顺序叠每个组合包，再叠 Profile 自己的 `cordis.patch.yml`，再叠 home 级的，最后叠命令行的 `-patch`。最底层 `dsh-base` 提供模型适配器、工具、持久化、沙箱与审批等基础件；`dsh-web-app` 与 `dsh-headless` 分别在其上追加浏览器应用与一次性运行器。用 `dsh -dump-config` 可以打印出你机器上真正叠成的这摞胶片，其中任何一条都能被你的 patch 替换。

5.3 能力 seam：可替换能力的三件套

「一切皆插件」回答了**谁在运行**；而 **seam（可替换能力）**回答了**一项能力怎么被设计成「能换」**。可以把它想成一个**标准插座**：插座有统一的接口规格（Definition），插头有不同厂家（Provider），电器只管按规格用电（Consumer）——换一个厂家的插头，电器照常工作。

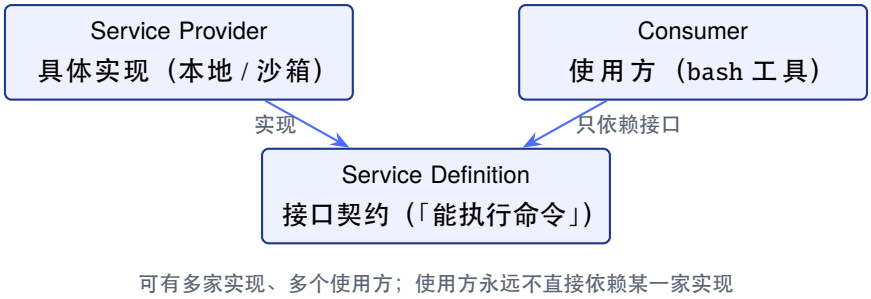

图 4: seam 的「插座—插头—电器」关系

角色	职责	示例（Shell 执行能力）
Service Definition	声明接口（ctx.<key> + 类型）	dsh-shell（抽象执行器定义）
Service Provider	实现该接口	dsh-bash-local、dsh-bash-sandbox
Consumer	使用该接口（通常为面向模型的工具）	dsh-tool-bash（bash 工具）

表 1: 能力 seam 的三件套

seam 的关键价值在于「**换一个 Provider，整个产品就跟着变**」：因为文件系统（fs）和进程（subprocess）用的是同一个插座规格，你把它们指向远程沙箱，就等于顺带把 Bash、PTY、LSP 一起搬进了沙箱，**不用为每种能力单独开发一份远程版**。源码里那条铁律——「扩展插件依赖接口，绝不依赖某家具体实现」——正是由此而来。

6 运行时机制

6.1 事件系统与分发模式

事件是 Harness 的扩展点，也是插件间「传话」的机制。可以把事件想成广播：出了事就喊一嗓子，谁关心谁听。Cordis 提供四种「喊法」（分发模式）：

模式	是否等待	顺序	返回值	典型用途
emit	否	按注册顺序	否	广播观察
waterfall	否	按注册顺序	是	可拦截/包装/替换的链式决策
parallel	是	并行扇出	否	并行处理
serial	是	按注册顺序	是	按序执行

表 2: 四种事件分发模式

其中 `waterfall`（瀑布式）最常用，也最像「传话接力」：每个监听器收到一个「往下传」的把手（`next`），喊一声 **`next()` 才继续传给下家**；不喊、直接返回，就是**半路截胡**（短路）。这天然适合做「审批链」——比如在 `agent/pre-step` 事件里，任何一个插件都能改写甚至拒绝模型即将看到的消息。

把四种「喊法」画成图，一眼就能看懂差别：

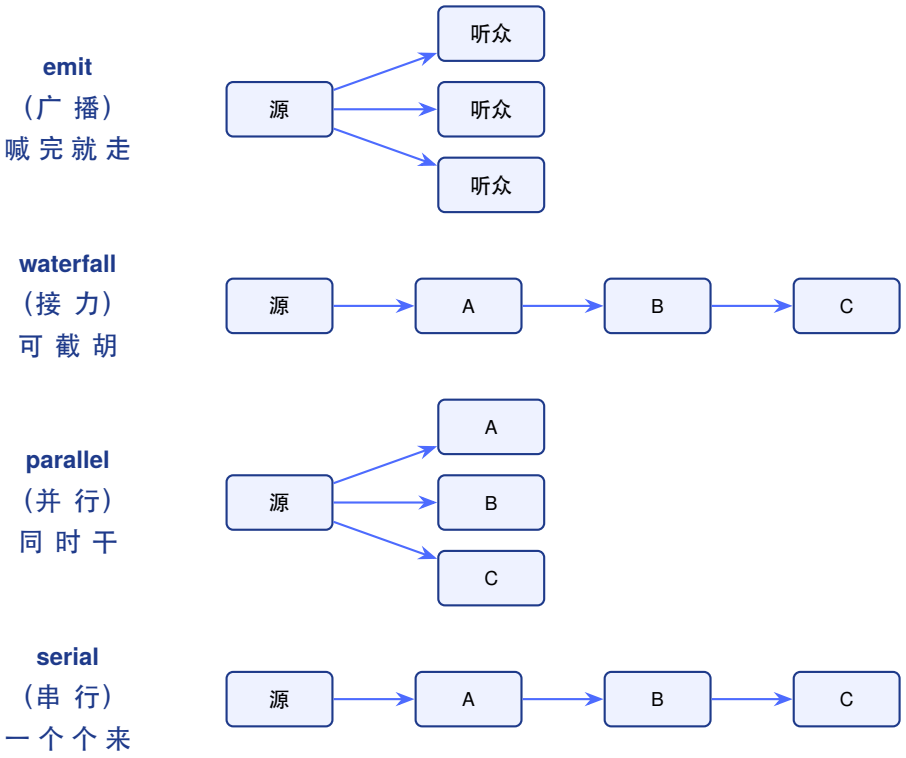

图 5: 四种事件分发模式：广播 / 接力 / 并行 / 串行

按领域，事件分三类：**会话事件**（写进黑匣子日志的持久事实）、**Agent 事件**（`agent/*`，携带活跃智能体的实时协调信号，如收件箱、步骤、状态）、**能力事件**（`fs/*`、`tools/*`、`telemetry/*`，用来给某类能力挂上策略与适配器）。

6.2 会话日志与「模型可见即已记录」

Harness 里有一台**黑匣子**：模型做的每一步（用户说了啥、它回了啥、调了哪个工具、结果是啥）都被记成一条条「事件」，**只增不改、按顺序追加**。这台黑匣子就是**会话日志**，它是全系统的唯一事实来源：

- **喂给模型看什么**，是从日志里「按需重算」出来的，而不是另存一份容易对不上的副本；
- **回放**（逐字重演模型当时的输出）、**分叉/恢复**（fork / resume 会话）、**转录**、**排查**，全部读同一份日志。

这套机制由一条铁律约束：

模型可见即已记录（Model-visible logged）。凡抵达模型请求的内容都必须能从日志重建，并由运行时不变式断言。因此，新增一种模型可见输入，就必须新增一种会话事件（扩展 `SessionEventMap` 并从日志渲染）。

一份日志、四处受用，就像一台黑匣子同时喂饱四个仪表盘：

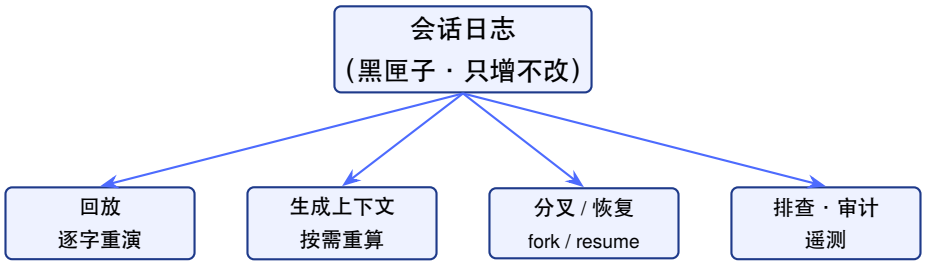

图 6: 一份会话日志，同时支撑回放、上下文、分叉与排查

正因为「喂给模型的内容是从日志按需重算的」，Harness 才能精细控制上下文、少发重复内容，从而省 **Token**。配合两个「瘦身」手段，长对话也不会无限膨胀：

- 压缩（**compaction**）：对话太长时，把历史「挤水分」成一段摘要，而不是原样全发；
- 溢出（**spill**）：把不常用的工具结果「挪到仓库」存起来，需要时再取回，不占着对话窗口。

6.3 轮次与步骤生命周期

先记两个最小时间单位：**步骤（step）= 模型想一次 + 动手一次；轮次（turn）= 从接下一条任务，到把这件事彻底干完（中间可能想很多次、动很多次手）**。整个运行过程就像下图这样循环：

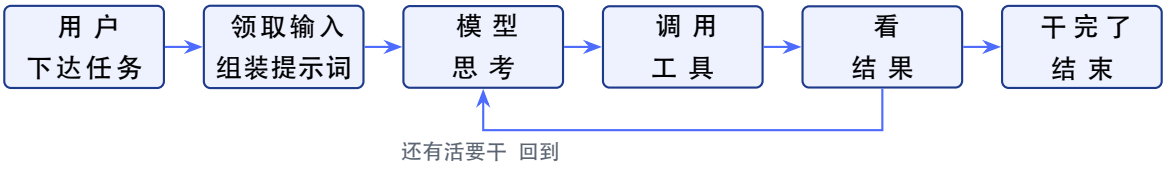

图 7: 智能体循环：思考—动手—看结果，直到把事情干完

把这个图读成大白话，就是五个环节：

1. **领取**：任务从统一的「收件箱（inbox）」进来，驱动器认领「下一步要干的事」；
2. **组装**：把各插件贡献的提示词片段和工具说明拼成发给模型的请求；
3. **思考**：模型给出回复（可能要求「调用某工具」）；
4. **动手**：执行工具。这一步有一道**拦截点**（`agent/pre-step`），任何插件都能在此改写甚至拒绝模型即将看到的内容；
5. **判断**：如果还有活要干，回到 ；干完了就关闭本轮。

三个容易踩坑的细节：

- **被拒绝也要记账**：即使第一步就被拦截或改成空，仍会关闭一个「啥也没干」的轮次，保证日志忠实记录这次尝试；
- **工具的并发有规矩**：多个工具先按顺序「准备」，再并发「执行」，最后按顺序「收尾」，并受并发上限约束；
- **出错能续命**：请求失败会走一个可拦截的错误事件，允许插件决定「重试」；上下文太长时，先删掉次要的工具结果，再决定是否压缩摘要。

完整的底层事件流（面向插件作者的进阶参考）见附录 C。

7 核心子系统

表 3 列出向 Cordis 树贡献内容的核心包及其 `ctx` 键。

包	职责	ctx 键
`core/session`	仅追加的 `SessionEvent` 日志与内存存储	`ctx.sessions`
`core/system-prompt`	提示词片段与工具 schema 的组装	`ctx.systemPrompt`
`core/tools`	作用域化工具注册表 + 带把关的执行流水线	`ctx.tools`
`core/agent`	`Agent` 接口、活跃注册表、`agent/*` 事件	`ctx.agents`
`core/agent-loop`	实现该接口的默认驱动器（可替换）	`ctx.agentLoop`
`core/scope`	按 agent 划分作用域的注册原语	库，无 ctx 键
`llm/llm`	消息与流式词汇表 + 适配器 seam	`ctx.llm`

表 3: 核心主干包

此外，`packages/` 下按约 **50** 个分组组织了 **200** 余个 **npm** 包（其中 22 个面向模型的工具包），统一以 `@deepseek-ai/dsh-*` 命名。代表性分组有 `core`、`api`、`typert`、`llm`、`shell`、`fs`、`lsp`、`skill`、`subagent`、`workflow`、`session`、`settings`、`credentials`、`interaction`、`host`、`client` 等，每个组负责一类关注点，组 README 维护「包 / `ctx` 键」映射。

8 能力与工具生态

8.1 面向模型的能力工具

表 4 汇总 Harness 提供的代表性面向模型工具（按能力分组）。

能力	工具
文件系统	read、write、edit、read_image、glob、grep
Shell / 终端	bash、pwsh、terminal_open、terminal_send、terminal_read、terminal_signal、terminal_close、terminal_list
代码运行	run_code
LSP	lsp
联网	web_search、web_fetch
会话检索	session_search、session_event_read、session_event_search、session_trace、session_event_trace
规划	todo_write、exit_plan_mode、create_goal、get_goal、update_goal
定时 / 后台	schedule_create、schedule_list、schedule_delete、job_list、job_output、job_kill
子智能体	subagent、list_agents、send_message、interrupt_agent
编排	workflow、ralph
技能 / 交互	skill、ask_user_question
自我改装	cordis_define、cordis_undefine、cordis_inspect_list、cordis_inspect_query、cordis_inspect_self、cordis_run、cordis_stop

表 4: 代表性面向模型工具

8.2 模型提供方

模型适配器同样是插件——可以把它想成**发动机**：车身（工具、日志、循环）不变，只换发动机。开箱自带 DeepSeek 官方这台「原装发动机」，也能通过目录或自定义表单接入 Anthropic、OpenAI、Bedrock、Vertex、Azure、Codex 等提供方，以及任意 OpenAI 兼容端点（自定义提供方需声明 `baseURL`、协议、凭据与模型；图片模态需按模型声明 `input: [text, image]`）。

9 多智能体与自我改装

9.1 作用域与子智能体

当多个智能体一起干活时，得管好「谁看得见什么」。Harness 用**作用域（scope）**来解决：每个智能体都有一份自己的工具箱——工具、提示词、变量要么是「全局公用」的，要么只属于某一个智能体。规则只有两条：

- **个人版优先（shadowing）**：如果某个智能体自己配了一个和全局同名的工具，它的个人版会「盖过」全局版——这就是「给不同智能体定制不同人设/工具」的办法；
- **可以没收（restriction）**：也能把某个智能体不配拥有的全局工具「屏蔽」掉——被屏蔽的工具既不会出现在它的提示词里，也执行不了，和不存在一模一样。

父子关系用一份**家谱数据（lineage）**记录（谁是谁的子、委派了几层），但家谱只管「记关系」，**不影响能不能看见**——子智能体不会自动继承父级的工具箱。

这就好比一个**包工头**把活儿派给几个**师傅**：师傅们各自背着不同的工具箱，谁该用什么就带什么，互相不串：

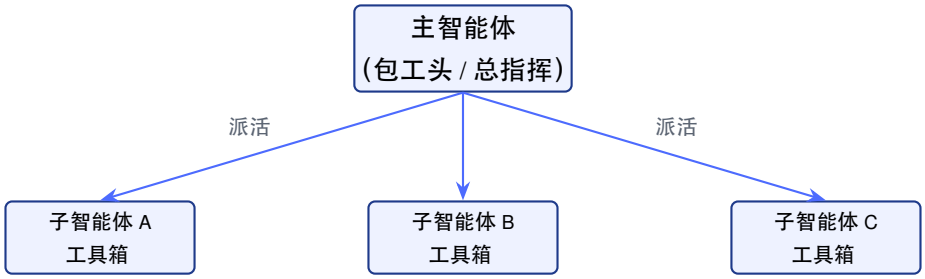

图 8: 多智能体协作：主智能体派活，每个子智能体只带自己的工具箱

9.2 目标（goal）

如果一件大活要跨很多轮才能干完，可以给会话贴一个**目标（goal）**：它带着进度状态（`active` 进行中 / `paused` 暂停 / `blocked` 卡住 / `complete` 完成）和「最多自动续跑几轮」的上限。

它的设计很克制：goal 只是一个**状态标签**，不是定时器，也不会偷偷另起一套对话——**日志仍是唯一真相**。也因此，会话恢复或分叉之后，自动续跑不会自己启动，必须经过一次人类点头（用 `/goal` 命令或模型工具）才会重新「上膛」。

9.3 工作流（workflow）与 Ralph

- **workflow（流水线总包）**：写一段编排脚本，把任务拆给一大批子智能体，分阶段并发地干、最后汇总结构化结果。就像总包商把一栋楼分包给各施工队，再收口验收——适合大规模审计、迁移与多角度研究；
- **Ralph（换个新脑子重看）**：前台「全新 agent」多轮迭代——每轮都开一个不带任何上一轮记忆的全新子会话，只靠共享工作区和一份有界的交接报告接力。就像每次换一个没被带偏的新人重新审一遍，避免旧思路越陷越深。

9.4 运行时自我改装

`packages/extensions` 提供一组 `cordis_*` 工具，使智能体能在运行时检查并修改自身的插件：`cordis_define/cordis_undefine` 定义/移除插件，`cordis_inspect_*` 实时检查插件与服务，`cordis_run/cordis_stop` 运行/停止插件。这是「自进化软件雏形」说法的技术来源。

10 安全与治理

安全同样被建模为可替换 seam。其中最核心的是沙箱——可以把它想成一个安全屋：智能体要干的「危险动作」（跑命令、写文件）都被关进屋里，只能在划定的范围（工作区）内动手，越界就被拦下。

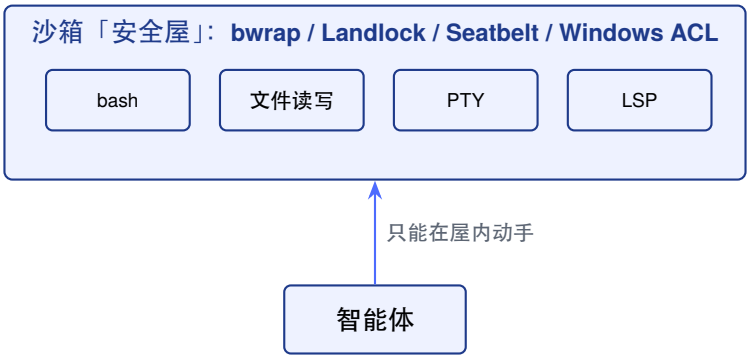

图 9: 沙箱安全屋：bash、文件、PTY、LSP 都被关在同一间屋里

- **沙箱（sandbox）**：进程限制 seam，是「与宿主共享内核/文件系统的限制」，而非容器或虚拟机。Linux 提供 bwrap 与 Landlock 后端（`native/landlock-run` 为 Landlock 的原生 Node 插件），macOS 提供 Seatbelt（`sandbox-exec`），Windows 提供基于 ACL 受限令牌的后端（部分强制）；消费方在启动进程前先让 `ctx.sandbox.confine()` 包装命令行参数；
- **审批 / 交互（interaction）**：人机协作平面，含批准 seam、权限预设、命令与 `ask_user_question` 工具，操作需审批时由 UI 先行询问；
- **凭据（credentials）**：凭据引用 seam，环境变量优先于 `.env`；Web UI 中 API key 只写，保存后仅返回脱敏描述符，明文仅落于 `\$DSH_HOME/.credentials.yaml`；

- **守卫（guard）**：循环卫生守卫（重复调用提醒）与 `tools/execute` 截止时间强制执行器，防止循环空转与工具超时。

「替换一个 Provider 即全局生效」在此同样成立：把 `fs/subprocess` 指向沙箱，Bash、PTY、LSP 一并纳入安全屋。真正的容器 / 微虚拟机（如 E2B）则不走这条缝，而是整体替换 `fs/subprocess` 的实现。

11 开发者生态与扩展

11.1 三种接入形态

Harness 的核心是同一台「引擎」，外面套了好几种遥控器，你可以按场景挑顺手的用：

- **Web UI**：`npx @deepseek-ai/dsh web`，默认 `http://127.0.0.1:3080`；
- **CLI**：`dsh web`、`dsh -profile headless "task"`、`dsh plugin ...`、`dsh -profile web -dump-config` 等进入模式；
- **Python SDK（客户端）**：`deepseek-harness-sdk`，高层轮次 API 与 JSON-RPC 客户端；
- **Python SDK（运行时）**：`deepseek-harness-runtime-bin`，内置运行时，经 stdio 按行分隔 JSON-RPC 通信；
- **自动化入口**：ACP 服务器（`packages/acp`）、进程外 JSON-RPC SDK（`packages/sdk`）、Claude Code / Codex 钩子桥接（`packages/hooks`）。

11.2 扩展速查

表 5 汇总「新行为该放哪」。

11.3 工程规范

全仓库 ESM + TypeScript strict 模式，Node 版本 `^22.19 || >=24`，pnpm workspace 管理；CI 对 `packages/*/*/src` 要求每文件 100% 覆盖率；npm 包统一命名 `@deepseek-ai/dsh-*`；类型化事件用声明合并注册，事件 JSDoc 需 `@mode` 标注分发模式，并由生成目录与分发调用点交叉校验。

12 对比分析

表 6 从关键维度对比 Harness 与「通用自研循环」及「Claude Cowork（据公开报道）」。

核心差异可归纳为：Harness 把「可组合性」做到了框架层——不仅能力可插拔，连循环与运行时形态都可被替换与自省；通用自研方案则多在应用层临时拼装。需要说明的是，竞品细节以官方公开信息为准，本表仅作定位参照。

目标	机制
添加模型提供方	在 `ctx.llm` 注册适配器
添加面向模型的能力	在 `ctx.tools` 注册，schema 自动进提示词组装
让会话拥有不同能力集合	组装 agent preset（服务行用 `isolate` realm）
添加 shell 执行	注册 `ctx.shell` 后端(本地后端走 `ctx.subprocess`)
添加文件系统/策略	注册 `ctx.fs` 提供方或监听 `fs/*` 事件
限制启动进程	用 `ctx.sandbox` 后端
拦截请求/工具/轮次	用 `agent/*` 或 `tools/*` 事件
添加模型可见上下文	调用 `agent.inject()`
生成会话标题	注册唯一 `ctx.sessionTitle` 提供方
将注册限定到单个 agent	使用该 agent 的 `agent.ctx`

表 5: 新行为的归属位置

维度	DeepSeek Harness	通用自研循环	Claude Cowork*
核心定位	通用智能体运行时	项目内胶水代码	桌面智能体工作台
架构范式	一切皆插件（Cordis）	定制化单体	工具/技能 + 托管代理
扩展方式	挂载插件 / patch	改源码	生态插件 / MCP
循环可替换性	循环本身是插件	通常写死	产品内置
会话真源	单一事件日志	通常无统一日志	会话/检查点机制
自我改装	运行时 `cordis_*`	无	未公开同等能力
多智能体	subagent / workflow / ralph / goal	视实现而定	托管代理
开源	是（MIT）	是	否

表 6: 与同类方案的关键维度对比（* 据公开报道，仅供参考）

13 应用场景

- **软件工程**：代码库理解与重构、缺陷修复、测试生成与执行、依赖迁移；
- **研发运维**：批量脚本执行、CI 故障诊断、日志检索与根因分析；
- **数据与知识工作**：多源检索、报告撰写、文档迁移与格式转换；
- **多智能体编排**：大规模代码审计、并行调研、跨模块协同改造；
- **嵌入式集成**：经 Python SDK / ACP 将智能体能力嵌入自有产品与自动化流水线。

14 路线图

基于当前开发者预览阶段的源码与公开信息，可勾勒如下演进方向：

- **近期**：稳定 API 契约、首个带版本标记的正式发布、破坏性变更收敛；
- **能力侧**：扩展模型提供方与目录、完善沙箱跨平台覆盖（尤其 Windows）、沉淀 E2B 等 POC 为稳定能力；
- **生态侧**：插件市场与发现机制（`dsh-plugin` 话题）、第三方插件收录、文档与教程体系；
- **协作侧**：多智能体与目标/编排原语的进一步收敛，自我改装能力的治理与安全边界。

路线图基于当前可观测信息推演，具体里程碑以官方发布为准。

15 风险与局限

- **稳定性风险**：仍为 RC 阶段，接口与磁盘格式可能变更；SQLite 使用单调 `SCHEMA_VERSION`，`dsh-session` 的 `SESSION_FORMAT_VERSION` 停留在 0 且不承诺兼容；
- **能力边界**：默认 DeepSeek 官方 chat 路由为纯文本，图片需按模型显式声明模态；部分提供方（如 E2B）标注为 POC；
- **平台依赖**：沙箱能力依赖宿主平台（Linux 的 bwrap/Landlock、macOS 的 Seatbelt、Windows 的 ACL 受限令牌），其中 Windows 后端报告**部分强制**（partial），且沙箱本身只覆盖文件系统与内核级限制，不覆盖网络/凭据等；
- **安全责任**：允许智能体执行文件与进程操作，权限策略与沙箱配置需使用者审慎设定；
- **竞品信息**：对同类产品的对比基于公开报道，可能滞后于其实际演进。

16 结论

DeepSeek Harness 以「一切皆插件」为核心，将智能体运行时的每一环节都化为可替换、可撤销、可自省的插件，并以单一真源会话日志保证可回放、可审计与上下文经济。它并非又一个「套

壳聊天」，而是一套面向「让模型真正办事」的通用底座，其组合拳——无特权内核、能力 seam、白盒循环、多智能体原语与运行时自我改装——在同赛道中具备鲜明辨识度。

对于开发者，它提供了从 Web UI、CLI 到 Python SDK / ACP 的完整接入，以及清晰可文档化的扩展面；对于研究与决策者，它示范了「可组合、可审计、可自演进」智能体运行时的可行路径。在开发者预览阶段的快速迭代中，其接口与生态仍将演进，值得持续跟踪。

A 术语表

术语	含义
harness	智能体框架 / 运行底座本身
Cordis	底层插件框架，设计范式为「时空可组合」
plugin / Service	插件，实现某个服务的对象
context（`ctx`）	服务容器，通过 `ctx.<key>` 查找服务
inject	依赖注入声明
effect / disposer	可逆副作用 / 资源释放函数
seam	可替换能力（Definition + Provider + Consumer）
Profile	具名组装（组合包 + 补丁 + 树外插件）
bundle	组合包，一层预配插件
patch	按 id 替换/插入配置项的补丁
emit / waterfall / parallel / serial	四种事件分发模式
turn / step	轮次（一次排空）/ 步骤（一次请求 + 工具）
scope / shadowing	作用域 / 最具体者胜出的名称解析
lineage	父子血缘关系数据（不影响可见性）
goal / Goal Round	持久完成目标 / 一次目标续跑周期
workflow / ralph	大规模编排脚本 / 全新 agent 前台迭代
compaction / spill	上下文压缩 / 工具结果溢出存储
ACP	Agent Client Protocol，面向自动化的智能体客户端协议

B 关键命令

```
# 安装运行（Web UI，默认 :3080）
npx @deepseek-ai/dsh web
```

```
# 源码运行
git clone https://github.com/deepseek-ai/deepseek-harness.git
cd deepseek-harness && pnpm install && pnpm run build && pnpm dsh web

# 一次性任务
pnpm dsh --profile headless "run the tests"

# 查看配置树
dsh --profile web --dump-config
dsh --profile web --dump-default-config

# 管理 profile 插件
dsh plugin --profile <name> <pnpm args>
```

C 完整轮次事件流（进阶）

以下为 6.3 节轮次循环在源码层面的完整事件序列，供插件开发者定位「该在哪个事件上挂代码」时参考（agent/pre-step、agent/request、llm/stream 与三个 tools/* 事件为瀑布式，可拦截；agent/turn-stopping 为串行终局检查点）：

```
turn/start
  claim next-step input + one queued message
  assemble prompt sections + tool schemas
  -> agent/pre-step                    (waterfall: reject | enter(messages))
     step/start
     append entered messages as user/message
     derive model history from the log
     agent/request -> llm/stream -> assistant/chunk* -> assistant/message
     tool/call* -> tools/pre-execute -> tools/execute -> tools/post-execute ->
    tool/result*
     step/end
     (tools owe another request, or next-step input arrived) -> claim -> next step
  -> agent/turn-stopping               (serial: terminal checkpoint)
turn/end
```

D 参考资料

序号	来源
[1]	源码仓库：https://github.com/deepseek-ai/deepseek-harness

序号	来源
[2]	架构文档（中文）：https://github.com/deepseek-ai/deepseek-harness/blob/master/docs/architecture.zh.md
[3]	Cordis 框架：https://github.com/cordiverse/cordis
[4]	Cordis 设计论文：https://github.com/cordiverse/paper
[5]	插件发现话题：https://github.com/topics/dsh-plugin
[6]	一切皆插件、省 Token、自我改装（博客园）：https://www.cnblogs.com/itech/p/22460843
[7]	开源首日 28k Star 报道（搜狐）：https://www.sohu.com/a/1062527577_115128
[8]	黑色鲸鱼解读（36 氪）：https://www.36kr.com/p/3938566998834308
[9]	对标 Claude Cowork（IT 之家）：https://www.ithome.com/0/989/446.htm
[10]	对标 Claude Cowork（动点科技）：https://cn.technode.com/post/2026-08-14/deepseek-harness-open-source-agent-framework/
[11]	Cordis「时空可组合」解读（53AI）：https://www.53ai.com/news/OpenSourceLLM/2026081415798.html
[12]	深度体验（量子位）：https://www.qbitai.com/2026/08/472208.html

免责声明：本蓝皮书对源码与文档的描述以仓库版本 `0.1.0-rc.5` 为准；开发者预览阶段演进极快，具体接口、命令与默认值请以官方仓库最新文档为准。对同类产品的对比基于公开报道，仅供参考。