DeepSeek Harness
应用开发实践

郭涛　著

清华大学出版社

DeepSeek Harness 应用开发实践

出版发行：清华大学出版社

本书内容简介

DeepSeek Harness（命令行简称 `dsh`）是 DeepSeek 官方开源的智能体基座，其核心设计哲学只有一句话——**万物皆插件**：模型适配、工具调用、会话管理，乃至驱动智能体运转的 Agent 循环本身，都是可以替换的插件。

本书是一本面向开发者的实战手册，系统讲解如何基于 DeepSeek Harness 构建智能体应用，聚焦四大主线：**快速上手与架构解析、安装部署与多模型接入、插件开发与智能体构建、实战案例**。全书以大学生备考助手 ShuTongBuddy（书童）作为贯穿案例——第 5 章为它开发插件包，第 8 章把它集成为完整的 Web 应用 ShuTongBuddy Studio，让读者在"动手做"中掌握从插件架构到智能体工程的完整路径。

本书适合有编程基础、希望快速上手智能体应用开发的工程师、研究者与高校学生阅读。阅读本书只需具备基础的 TypeScript 与 Python 阅读能力，以及调用大模型 API 的经验。

作者简介

郭涛，资深技术实践者、畅销书作者。深耕计算数学、人工智能、现代软件工程、数智农业等前沿交叉领域，专注技术落地与知识沉淀。累计出版著（译）作 40 余部，代表作包括《大模型设计模式》《Agentic AI 智能体应用开发（第 2 版）》《大模型前沿与实践》等。

自序

设想一位刚入学的大学生，面对一门全新的专业课：教材、课堂笔记、历年试卷与错题散落在不同的文件夹里，复习时既不知从何下手，也难以判断自己真正薄弱的知识点在哪里。这恰恰是无数学习者备考时的真实困境。智能体技术的出现，为这一困境提供了一种新的解法——一个懂你、陪你备考的"书童"。

本书的缘起，正是想回答一个更普遍的问题：**如何用 DeepSeek Harness 这样的智能体基座，把"想法"变成"能用的智能体应用"？** 在写作过程中，我始终把"动手做"放在第一位：先把 Harness 跑起来，再把模型接进去，接着把插件写出来，最后把智能体做出来。全书的贯穿案例——大学生备考助手 ShuTongBuddy——正是这条路径的自然产物：它从一个插件包出发，最终长成一个完整的 Web 应用。

本书的内容整理自 DeepSeek Harness 官方仓库的 README、用户指南、架构文档与 Cookbook。DeepSeek AI 团队以 MIT 协议开源了这一项目，为本书提供了坚实的技术底座，在此致以诚挚的谢意。同时感谢清华大学出版社的编辑团队在书稿打磨中给予的专业支持。

由于 DeepSeek Harness 仍处于开发者预览阶段，迭代迅速，个别细节以官方文档为准，书中已用"【待核】"标出。若书中存在疏漏，恳请读者指正。

郭涛

目 录

第一部分　入门

第 1 章 快速上手：十分钟跑通 DeepSeek Harness

本章目标只有一个：让你在最短时间内把 DeepSeek Harness 跑起来，并完成第一个智能体任务。跑通之后，我们再在第 2 章回过头解释背后发生了什么。

1.1 DeepSeek Harness 是什么

DeepSeek Harness（命令行简称 `dsh`）是 DeepSeek 官方开源的智能体基座（Agent Harness），以 MIT 协议发布。一句话概括它的设计哲学：Everything is a Plugin——万物皆插件。

注意它与 DeepSeek 大模型的区别：

- DeepSeek V4 Pro/Flash、DeepSeek-R1 是模型，是"大脑"；
- DeepSeek Harness 是承载大脑的"身体"——模型适配、工具调用、会话管理、人机交互界面的完整骨架。

更确切地说，Harness 里连"骨架"本身也是插件：模型适配器、工具注册表、会话日志，乃至驱动智能体运转的 Agent 循环（Agent Loop）本身，都是可以从配置层替换的插件。整个系统建立在 Cordis 插件框架之上（其设计论文《A Programming Paradigm for Spatiotemporal Composability》已公开发表），没有任何一个"特权核心"需要你打补丁——扩展 Harness 的方式，永远是把你的插件挂载到官方插件旁边。

这个设计带来两个直接好处：

- 换模型不改代码。DeepSeek、Kimi、GLM、OpenAI、Anthropic……接入新模型只是注册或配置一个适配器；
- 加能力不动主干。加工具、加钩子、加界面，都是写插件，而不是 Fork 改源码。

版本提示：DeepSeek Harness 处于开发者预览阶段，官方明确声明"会存在破坏兼容性的变更"。学习它，既是学一个工具，也是学一套正在成形的智能体工程范式。

1.2 环境准备

开始之前，请确认两样东西：

（1）**Node.js 22.19+ 或 24 及以上**。在终端执行：

```
node -v
```

输出 v22.19.0 或更高即可。如果没有安装，请到 nodejs.org 下载 LTS 版本

°（2）一个 DeepSeek API Key。 登录 DeepSeek 开放平台（platform.deepseek.com）创建。没有它也能启动界面，但无法发起真实的模型请求。

1.3 三分钟启动 Web UI

确保网络畅通，执行

```
npx @deepseek-ai/dsh web
```

npx 会自动下载并运行 dsh 命令。首次运行稍慢，稍候你会看到终端打印出访问地址，默认是

```
http://127.0.0.1:3080
```

本地启动时浏览器会自动打开该地址；如果你不想要这个行为，加

```
npx @deepseek-ai/dsh web --no-open
```

在 SSH 远程会话中，dsh 不会尝试打开浏览器（转发地址由你的 SSH 客户端或编辑器管理），只打印 URL，你在本地浏览器手动访问即可。

端口冲突怎么办： `dsh web` 是 `--profile web` 的别名，Web 应用自己的参数（如 `--port 8080`）跟在 profile 之后传入。第 3 章会详细解释这套命令语法。

1.4 配置第一个模型

相信大部分人已经到终端按 Ctrl+C。

界面打开后，第一件事是告诉 Harness 用哪个"大脑"。

1. 打开 Settings → Models（设置 → 模型）页面；
2. 找到 DeepSeek 卡片，它只暴露一个 API Key 输入框；
3. 粘贴你的 Key，保存。

保存后模型路由**立即生效，无需重启服务**。这里有一个值得注意的设计：Key 是"只写"的——保存之后，页面只会收到一个打码后的描述符，永远不会回显你的明文密钥。密钥本体存放在 Harness 主目录的 `$DSH_HOME/.credentials.yaml` 中，设置文件里只保留对它的引用。

此时模型选择器中已经出现 DeepSeek 的模型目录（如 `deepseek-v4-pro`、`deepseek-flash`），任选一个作为默认模型。第 4 章会展开讲 DeepSeek V4 Pro/Flash 的差异与 Kimi 等其他模型的接入。

1.5 选择工作区，下达第一个任务

刚启动的 Web UI 还没有"工作区"的概念——dsh 进程虽然以启动目录为默认文件位置，但你需要显式地把一个目录交给它：

1. 点击 Choose workspace（选择工作区）；
2. 添加你启动 dsh
   时所在的项目目录（或任意一个你希望智能体操作的目录）；
3. 选中它。

选好工作区之前，会话输入框是不可用的——这是一道刻意的安全闸：没有工作区，智能体就没有可操作的边界。

```
总结这个仓库的结构，找出主要的包，并说明它们各自的职责。
```

（官方指南给出的经典示例是："Summarize this repository and identify its main packages."）

接下来你会看到 Harness 的典型工作方式：

- 智能体开始规划（Plan），把任务拆成若干步骤；
- 它调用工具读取工作区文件、执行命令——每一次工具调用在界面上都有卡片式的呈现；
- 涉及需要审批的操作时（按当前权限策略），界面会停下来询问你，确认后才继续；
- 最终产出一段结构化的仓库摘要。

这个过程中发生了的一切——每一条消息、每一次工具调用、每一个中间状态——都被写进了会话日志。这是 Harness 的核心不变量：**"模型可见的，必须已落日志"**（Model-visible means logged）。第 2 章会解释这条不变量为什么是整个架构的基石。

1.6 核心概念速览

第一个任务跑完了。趁印象深刻，把后面会反复打交道的八个概念一次看清楚。

概念	一句话解释
Plugin（插件）	Harness 的一切都是插件：一个导出 apply
ctx（上下文）	Cordis 提供的共享上下文，插件通过它注册服
Session（会话）	一次持续的人机协作；底层是一份只追加的事件日志
Turn（轮次）	从零到多个 Step 构成；在首个输入被认领前开启，在"不

再欠任何工作"时关闭

Step（步骤）	一次模型请求加上它所触发的工具调用
Tool（工具）	面向模型的能力单元（读文件、跑命令、搜索……），注册到 ctx.tools
Profile（运行配置）	一组命名好的插件树组合：web、headless、sdk、sdk-minimal、acp
Seam（能力接缝）	一个可替换能力的三角色契约：接口定义、提供者、消费者

现在你只需要对这张表有印象。第 2 章会逐一展开。

本章小结

本章你完成了：

- 理解 DeepSeek Harness 的定位——模型之上的开源智能体基座，万物皆插件；
- 用 npx @deepseek-ai/dsh web 启动了 Web UI；
- 在 Settings → Models 中配置了 DeepSeek API Key（只写存储，无需重启）；
- 选择工作区并跑通了第一个智能体任务；

建立了八个核心概念的初步印象。

练习题

1. 启动 `dsh web`，在 Settings → Models 中配置一个 DeepSeek API Key，观察保存后模型路由是否立即生效。
2. 选择一个工作区，下达任务"总结这个仓库的结构"，观察规划、工具调用、审批、产出四类卡片。
3. 不看正文，用一句话分别解释 Plugin、ctx、Session、Turn、Step、Tool、Profile、Seam 八个概念。

第 2 章 架构解析：万物皆插件意味着什么

> **本章目标**：理解"万物皆插件"的微内核主张，掌握 Cordis、Profile/Bundle/Patch、能力接缝、事件系统、Turn/Step 与会话日志。

第 1 章你已经把 Harness 跑了起来。这一章回答一个问题：你启动的那个进程，内部到底长什么样？理解这一章，后面的安装、接模型、写插件都会变成"顺理成章"的事。

2.1 没有特权核心的微内核

大多数智能体框架的扩展方式是"留钩子"：框架作者预判你可能想改哪里，在那里埋一个回调点。预判错了，你就只能 Fork 源码。

Harness 走了一条更彻底的路：**微内核**（Microkernel）。整个系统中没有一个"特权核心"可以被打补丁——模型适配器、工具注册表、会话日志、权限策略，甚至驱动智能体运转的 Agent 循环本身，统统是插件。官方架构文档对每个产品功能做了一张"功能 → 机制"对照表，每一行都对应某个公开扩展点上的监听器，**没有一行需要修改循环本体**。这是可检验的微内核主张。

对使用者的现实意义：

- 官方能做到的，你也能做到——因为官方用的扩展点和你用的是同一批；
- 升级框架不破坏你的扩展——只要扩展点契约稳定，你的插件就能跟着升级；
- 出问题时边界清晰——能力来自哪个插件，就从哪个插件查起。

2.2 Cordis：插件框架底座

Harness 建立在 Cordis 之上。Cordis 的设计来自论文《A Programming Paradigm for Spatiotemporal Composability》，它给插件提供了三样原语，全部挂在共享上下文 `ctx` 上：

原语	作用	例子
`Service`（服务）	插件向全局发布的能力接口	`ctx.tools`（工具注册表）、`ctx.llm`（模型适配
`Typed Events`（类型化事件）	插件间通信与拦截的通道	`tools/pre-execute`、`agent/assistant-stream`

Reversible Effects（可逆效果）	注册即效果，卸载即回退	插件卸载时，它注册的工具、监听器、定时器自动清理

第三点尤其值得强调。在 Harness 里，"注册"不是往一个全局表里塞东西、再手动移除，而是声明一个**效果（Effect）**：插件存活时效果存在，插件卸载（或热重载）时效果自动回退。因此"插件热重载"在 Harness 里不是特殊功能，而是架构的自然结果——官方文档的原话是：每一次注册都是一个 `ctx.effect`，所以热替换"自然而然就能工作"。

2.3 Profile、Bundle 与 Patch：插件树如何组合

一个运行中的 `dsh`，是一棵**启动时按序组合出来的插件树**。理解这棵树的三层结构，就理解了 Harness 的部署与定制方式。

Bundle（功能包）：能力的分发单位，官方提供的核心 Bundle 有：

- `dsh-base`：所有运行形态共享的底座——模型适配器、工具、持久化、沙箱与审批策略、设置、凭据、遥测；
- `dsh-web-app`：在底座上叠加浏览器应用；
- `dsh-headless`：叠加一次性运行器（无服务器，跑完即退出）；
- `dsh-sdk-app`：叠加 SDK 的 JSON-RPC 服务；
- `dsh-sdk-minimal`：一个刻意的例外——自带完整显式的插件树，不叠加 `dsh-base`；
- `dsh-acp-app`：叠加仅供自动化使用的 ACP 服务。

Profile（运行配置）：存放在 Harness 主目录中的命名组合，声明它叠加哪些 Bundle、持有哪些外部插件、以及用户自己的补丁文件。官方模板有五个：`web`、`headless`、`sdk`、`sdk-minimal`、`acp`。你在第 1 章执行的 `dsh web`，就是 `--profile web` 的别名。

Patch（补丁层）：对插件树的声明式覆写。补丁按 id 定位插件树中的某一行，整体替换其配置，或插入新行。层的应用顺序是：

```
Profile 中按序列出的每个 Bundle 的补丁
  → Profile 自己的 cordis.patch.yml
  → 主目录级别的 $DSH_HOME/cordis.patch.yml
  → 启动时通过 --patch 传入的覆盖层
```

两个实用推论：

1. **可检视**：执行 `dsh --profile web --dump-config`，你能看到本机实际启动的整棵插件树；输出里的任何一行，都可以被你自己的 Patch 替换。定制之前先

dump，是 Harness 玩家的标准动作。

2. 可热更：自定义 Profile 默认开启实时补丁重载（live patch reload），web Profile 也是实时的；而 headless、sdk、sdk-minimal、acp 只在启动时应用一次——因为一次性任务或 stdio 服务在占用工作期间换依赖，会破坏其生命周期。

2.4 能力接缝：一次替换，全局生效

Harness 扩展设计里最重要的概念是**能力接缝**（Capability Seam）。一个接缝是可替换能力的完整契约，包含三个角色：

1. Service Definition——声明接口；
2. Service Provider——实现接口；
3. Consumer——消费接口，通常是面向模型的工具。

一个包可以身兼多个角色，但单独一个角色不构成接缝。**添加一项能力 = 设计全部三个角色**。

接缝的威力用一个例子就能说透：文件系统提供者（ctx.fs）和子进程提供者（ctx.subprocess）共享同一个"执行世界"。把这两个提供者指向一个远程沙箱，Bash 工具、终端工具、LSP 工具就**一起**迁到了远程——不需要逐个 Fork 修改。这就是"一次提供者替换，改变整个产品"。

子智能体提供者（ctx.subagents）是另一个典型接缝：同一个接口背后，可以是全新的进程内子智能体、Fork 出的分支智能体，甚至是委托给另一个产品（Codex、Claude Code）的一轮对话。此外还有一个实验性的 Agent Teams 接缝（ctx.agentTeams），在可持续的子智能体之上提供持久名册、任务看板和信箱的多智能体协调。

2.5 事件系统：扩展点就是事件

在 Harness 里，**事件就是扩展点**。大多数改动的第一个决策，是"该挂到哪个事件域"。一共三个域：

- 会话事件（Session events）：追加到日志并通过 session/event 广播的持久事实。凡是重载后必须存活的信息，走这个域；
- 智能体事件（agent/*）：携带实时 Agent 对象——收件箱、步骤、状态、请求、校验、续接。要观察或拦截"正在进行的工作"，走这个域；
- 能力事件（fs/*、tools/*、telemetry/*）：把策略和适配器挂

到接缝上，且不会产生 import 循环。

事件里有一类特殊的 Waterfall（瀑布）事件：监听器必须调用 next() 才会把控制权交下去，因此可以在中途改写或拒绝——agent/pre-step、agent/request、llm/stream 和三个 tools/* 事件都属于此类。第 5 章的权限门插件，用的正是 tools/pre-execute 这个瀑布。

2.6 Turn 与 Step：一次任务的完整解剖

第 1 章你发出的"总结仓库"，在 Harness 内部经历了一段精确的旅程。先记住两个定义：

Step（步骤）= 一次模型请求 + 它调用的工具。Turn（轮次）= 零到多个 Step，在第一个输入被认领前开启，在"不再欠任何工作"时关闭。

```
turn/start
  认领下一步输入 + 一条排队消息
  装配提示词各节 + 工具模式；投影运行时上下文
  -> agent/pre-step                          （可重写或拒绝输入）
     step/start
     agent/request -> prepareCall            （解析模型路由；取消不落任何内容）
     追加 user/message；冻结模型历史
     流式调用 -> llm/stream -> agent/assistant-stream start
       agent/assistant-stream chunk*         （实时 token 流）
       assistant/message -> agent/assistant-stream end
     tool/call* -> tools/pre-execute -> tools/execute
                -> tools/post-execute -> tool/result*
     step/end
     工具还欠一次请求，或有新输入 -> 认领 -> 下一个 step
  -> agent/turn-stopping
turn/end
```

留意三个设计细节：

- 不可变请求 + 实时取消：循环发出的请求是冻结的，但取消信号始终有效；agent/request 和 prepareCall()
  在系统提示词和用户消息提交之前解析路由，任一阶段取消都不会落下半个请求；
- 输入走单一收件箱（Inbox）：有些消息立即唤醒循环，注入的上下文则在收件箱里等待，直到下一条消息到达；
- 重试不重复装配：失败重试不会重新执行装配和 agent/pre-step。

2.7 会话日志：唯一的事实来源

Harness 有一条贯穿全局的核心不变量：

Model-visible means logged.（凡模型可见的，必须已落日志。）

任何进入模型请求的内容，都必须能从事件日志（Session Log）重建出来——而且有运行时断言守着这条线。会话日志是一份只追加（append-only）的 `SessionEvent` 流，模型上下文由 `deriveMessages()` 从日志投影而来

这条不变量换来了一连串能力：

- 分叉与续跑：Fork、Resume 都是"从日志重放到某点，再继续"；
- 回放与审计：assistant/message
  内嵌了产生它的精确流式内容，assistant/attempt
  保留了失败、重试、取消的完整痕迹；
- 遥测与持久化：全部派生自这些持久落账，不需要第二套数据源。

推论也很直接：**想给模型喂新的输入类型，就得先定义新的会话事件**——扩展 `SessionEventMap`，再从日志渲染。日志的物理格式由 JSONL 提供者管理（`session.vN.jsonl[.zstd]`），版本迁移按 `vN → vN+1` 单步进行，已提交的历史文件永不改名、替换或删除。

本章小结

本章拆解了 Harness 的五块基石：

- 微内核：没有特权核心，每个产品功能都挂在公开扩展点上；
- Cordis
  三原语：服务、类型化事件、可逆效果——热重载是架构的自然结果；
- Profile / Bundle / Patch：插件树按层组合，可
  dump、可覆写、可热更；
- 能力接缝：定义/提供者/消费者三角色，一次替换全局生效；

事件与日志：三个事件域 + "模型可见即已落日志"的不变量，支撑分叉、续跑、回放与审计。

练习题

1. 用自己的话说清"微内核"与"留钩子"式扩展的根本区别，并举一个 Harness 里的例子。
2. 写出 Profile 补丁的层序（从低到高），并说明 `--patch`

为什么"总是最后说话"。

3. 解释"模型可见的，必须已落日志"（Model-visible means logged）为什么是整个架构的基石。

第二部分　部署与模型

第 3 章 安装部署与 Web UI

> **本章目标**：掌握 npx 与源码两种安装方式、五种运行 Profile、命令行参数规则、$DSH_HOME 目录结构与安全基线。

第 1 章用的是最省事的路径。本章把安装部署这件事讲完整：两种安装方式、五种运行 Profile、命令行语法，以及必须认真对待的安全基线。

3.1 环境要求

项目	要求
Node.js	`^22.19.0 或 >=24.0.0`
包管理器（仅源码方式）	pnpm 11.7.0（仓库 packageManager 字段锁定）
操作系统	Windows / macOS / Linux 均可
网络	npx 方式需访问 npm registry；运行期需访问模型 API

Windows 用户提示：命令示例中的 `npx`、`pnpm` 在 PowerShell 与 Git Bash 中均可直接使用。

3.2 方式一：npx 直接运行（推荐日常使用）

-

```
npx @deepseek-ai/dsh web
```

这是官方给出的标准入口：无需克隆仓库，npx 拉取发布包后直接启动 Web UI，默认地址为 http://127.0.0.1:3000，常用变体：

```
npx @deepseek-ai/dsh web --no-open    # 不自动打开浏览器
```

适合：日常使用、快速体验、跟随本册绝大多数章节操作。

3.3 方式二：源码构建（推荐给要改代码的读者）

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

两个要点：

- `pnpm run build`

负责准备仓库构建产物（宿主库、客户端库、前端），先构建再运行，pnpm dsh web 直接使用已构建的产物，不会重复构建；

- pnpm dsh <args...> 运行的是 TypeScript 入口并原样转发所有参数——也就是说，本册所有 dsh 命令在源码树下写成 pnpm dsh 即可。

适合：第 5 章插件开发（教程要求从完成源码构建的仓库检出开始）、阅读源码、向官方贡献代码。

3.4 五种运行 Profile

dsh 命令是唯一的 Node 应用启动器，所有运行形态都是"profile + 参数"：

命令	用途
dsh web	--profile web 的别名：启动 Web UI 服务
dsh --profile headless "任务"	一次性运行：开一个全新持久会话，打印最终答案后退出
dsh --profile sdk	以 JSON-RPC stdio 服务 SDK 客户端，直到关闭或断开
dsh --profile sdk-minimal	用独立的最小插件树服务 SDK 客户端
dsh --profile acp	以 ACP stdio 服务自动化客户端，直到断开

五个官方 Profile 首次使用时会从内置模板自动初始化。desktop 这个名称保留给 Electron 桌面应用，CLI 会拒绝以它启动或管理插件。

```
dsh --profile mylab --from-default-profile web
```

该命令以官方 web 为模板创建一个名为 mylab 的新 Profile 并启动它。此后你可以修改 $DSH_HOME/profiles/mylab/ 下的 package.json（声明外部插件依赖与 Bundle 顺序）和 cordis.patch.yml（你的补丁层），而不污染官方模板。

```
dsh plugin --profile mylab add some-dsh-plugin
```

dsh plugin 会把参数原样转发给 pnpm，在该 Profile 的目录里执行——安装、移除、更新外部插件都走这条路。

```
dsh --profile mylab --dump-config           # 查看实际组合的插件树
dsh --profile mylab --dump-default-config   # 查看默认组合
```

3.5 命令行语法的两条规则

dsh 的参数解析有一个容易踩的坑，记住两条规则就不会错：

1. 启动器只解析自己的旗标（--profile、--patch、--dump-config 等），第一个不认识的参数开始，全部交给被启动的应用；
2. 同一个词，位置不同，含义不同：

```
dsh --profile web --port 8080    # --port 属于 Web 应用
dsh --profile web --help         # 显示 Web 应用的帮助
dsh --help                       # 显示启动器自身的帮助
dsh --profile headless "run the tests"   # 引号内是交给一次性会话的任务
```

无效命令、张冠李戴的参数、配置错误、启动失败，都会以非零码退出——写脚本时可以依赖这一点。

3.6 Harness 主目录（$DSH_HOME）

运行期的一切用户态数据都在 $DSH_HOME 下，值得熟悉的四个位置：

路径	内容
settings.yaml	全部设置，包括模型路由的进阶字段（第 4 章的主角）
.credentials.yaml	API 密钥本体（设置文件里只存引用）
cordis.patch.yml	主目录级别的补丁层，对所有 Profile 生效
profiles/<name>/	各 Profile 的目录：自己的 package.json 与 cordis.patch.yml

浏览器与服务同机时，可以在 Settings 页头部点 Open configuration file 直接打开 settings.yaml；适配器会在下一次请求时重读它，**改配置不需要重启服务**。

3.7 安全基线：必须照做的六条

Harness 能执行模型生成的代码与命令、加载第三方插件、访问网络与文件。官方 SAFETY.md 的措辞非常直白：它是开发者预览软件，**未经安全审计，不得视为安全或生产就绪**；沙箱、审批提示、权限控制能降低风险，但不保证隔离。请把下面六条当作使用前置条件：

1. 最小权限：以完成工作所需的最小权限与最小访问面运行；
2. 一次性环境：优先使用可丢弃的虚拟机、容器或专用环境，不要在存有重要数据的主力机上裸跑高风险任务；

3. 备份：Harness 可触及的文件，事前有备份；
4. 凭据自律：不接受风险，就不要把敏感凭据与数据暴露给它；
5. 先审后放：允许插件、配置、待执行命令生效前，先审查——界面的审批提示不是摆设；
6. 不依赖单一防线：不要把 Harness 自身的限制当作不可信负载的唯一安全控制。

第 5 章会教你用 `tools/pre-execute` 写一个权限门插件，把"先审后放"从人工动作变成可编程策略。

本章小结

- 日常使用走 npx @deepseek-ai/dsh web；改代码走源码构建 pnpm install && pnpm run build && pnpm dsh web；
- 五种 Profile 覆盖 Web、一次性、SDK、最小 SDK、自动化五类场景；自定义 Profile 用 --from-default-profile 创建，dsh plugin 管理其外部插件；
- 命令行两条规则：启动器只认自己的旗标；同一参数位置不同含义不同；
- $DSH_HOME 集中存放设置、凭据、补丁层与 Profile，配置改动热生效；

安全六条基线是使用 Harness 的前置条件，不是可选项。

练习题

1. 用 `--dump-config` 查看 web Profile 实际组合的插件树，找到模型适配器所在的那一行。
2. 列出五种运行 Profile（web/headless/sdk/sdk-minimal/acp）各自适用的一次性任务场景。
3. 复述六条安全基线，并说明"先审后放"在陌生仓库上的落地方式。

第 4 章 多模型 API 接入：DeepSeek V4 Pro/Flash、Kimi 与任意兼容端点

> **本章目标**：掌握 DeepSeek V4 Pro/Flash、Kimi 与任意 OpenAI 兼容端点的接入，学会 settings.yaml 进阶配置与故障排查。

Harness 的模型接入层是"为物盲插件"最好的广告牌：添加一个模型提供商，只是注册一个适配器；切换一个模型，不需要重启服务。本章把 DeepSeek V4 Pro/Flash、Kimi（Moonshot AI）以及任意 OpenAI 兼容网关的接入方法讲透，并给出一份故障排查速查表。

4.1 模型接入层全景

模型接入由 `llm/llm` 包承担，它定义了**消息与流的统一词汇表**和适配器接缝。`ctx.llm` 是它的服务键。官方自带两个参考实现：

适配器	风格	服务的路由
`dsh-llm-deepseek`	直连 HTTP，SSE	DeepSeek 官方 API
`dsh-llm-pi-ai`	包装通用 LLM 库	内置目录里的其他厂商 + 全部自定义端点

你在 Settings → Models 里做的一切配置，最终都落在 `$DSH_HOME/settings.yaml` 里；界面只暴露"让一条路由存在"所需的最小字段，其余进阶字段（推理挡位、图像输入、请求兼容开关、超时重试……）都在这个 YAML 文件里编辑，**保存后下一次请求即生效**。

4.2 接入 DeepSeek V4 Pro / Flash

DeepSeek 是 Harness 的一等公民路由，接入只需一步：打开 Settings → Models，在 DeepSeek 卡片填入 API Key 并保存。模型选择器随即列出官方目录中的模型，其中两个是本册的主力：

- `deepseek-v4-pro`：深度推理主力，适合规划、复杂分析、代码架构类任务；
- `deepseek-flash`：快速响应与多模态入口，适合摘要、改写、检索后处理，且默认带图像输入能力。

DeepSeek 路由的一个特色是**推理挡位**（Reasoning Effort）：其模型原生提供 `off`、`low`、`high`、`max` 四挡，模型选择器中的 Effort 菜单直接可选；路由级默认挡位在 `settings.yaml` 里设置：

```
llm-deepseek:
  reasoningEffort: max
```

经验法则：让 Flash 保持默认挡位跑流量，把 high/max 留给 Pro 处理硬任务。第 8 章的实例会把这条法则用成一套多模型分工策略。

4.3 接入 Kimi（Moonshot AI）

Kimi 走"内置提供商目录"通道，无需手写任何端点信息：

1. 打开 Settings → Models → Add provider；
2. 在列表中选择 moonshotai（即 Kimi；同目录里还有 anthropic、openai、zai（GLM）等）；
3. 填入 Kimi 开放平台的 API Key，保存。

安装目录内置了该厂商的端点、协议与模型清单，保存即可用。至此，模型选择器里同时存在 DeepSeek 与 Kimi 两个路由——**选中某个模型即把它设为后续新会话的默认模型**；已经发过请求的会话，则忠实保留自己日志里记录的模型，不会跟着默认模型跑。

注意：以 OAuth 登录的厂商（如 Codex）暂不支持在此页面接入。

4.4 接入任意 OpenAI 兼容端点

公司网关、自建服务、目录外的厂商，走 Add a custom provider。表单要求五项：

- Provider ID：小写标识，保存后不可更改——因为请求、已存会话、模型默认值与凭据引用都使用它。想改名，只能新建后删除旧的；
- Base URL：端点地址；
- API protocol：三选一——openai-completions（OpenAI Chat Completions）、openai-responses（OpenAI Responses API）、anthropic-messages（Anthropic Messages API）。一个提供商只说一种协议；一个网关若同时提供两种协议，就建两个提供商；
- 凭据：API Key；
- 至少一个模型。

模型清单可以手写，也可以点 Fetch available models 让 Harness 询问端点（对 OpenAI 兼容端点即调用 `GET /models`），在可搜索的清单里勾选添加。发现失败或返回为空时，**手写模型 id 一样能用**——发现只是便利，不是前提。

一个典型用法：把 DeepSeek V4 挂到公司统一网关后面，作为自定义提供商管理。

4.5 settings.yaml 进阶：三类高频定制

界面表单是刻意做小的。真正让多模型接入"好用"的，是 `settings.yaml` 里的三类进阶字段。

（1）图像输入：input 与 defaultInput

手工录入的模型默认被视为纯文本——因为没有途径询问端点支持哪些模态。给自定义端点上的视觉模型开图像输入，只需一行：

```
llm-pi-ai:
  providers:
    my-gateway:
      apiKeyEnv: GATEWAY_API_KEY
      api: openai-completions
      baseURL: https://gateway.example/v1
      models:
        - id: legacy-chat
        - id: vision-preview
          input: [text, image]
```

若该路由下所有手工模型都支持图像，在路由上设一次兜底即可：defaultInput: [text, image]（默认值是 [text]，它是兜底而非覆盖，不会摘掉目录模型已有的图像能力）。内置提供商的模型要用 modelOverrides 按模型 id 收窄。

（2）推理挡位：reasoningEfforts

手工录入的模型默认不声明推理挡位（Effort 菜单不出现，端点自行决定是否思考）。用 `reasoningEfforts` 声明，键是菜单里显示的挡位，值是发送到线上的 `reasoning_effort` 拼写：

```
models:
  - id: my-reasoner
    reasoningEfforts:
      off:
      high: high
      max: max
```

只有 off 允许留空——对多数端点，"不思考"就是参数缺省。但有一类模型默认思考、需要显式关闭，OpenAI 兼容网关后的 DeepSeek V4 就是典型。这时需要给模型加一个兼容开关 compat.thinkingFormat: deepseek：off 会发送 thinking: {type: disabled}，其余挡位发送 thinking: {type: enabled} 并附带挡位值：

（3）请求兼容：compat

```
models:
  - id: deepseek-v4-pro
    compat:
      thinkingFormat: deepseek
    reasoningEfforts:
      off:
      high: high
      max: max
```

网关的地址可达、密钥正确，仍可能拒绝每一个请求——因为它接受的请求形状与 OpenAI 本尊不同。最高发的两处：**推理模型的系统提示词**以 `role: "developer"` 发送（许多网关直接拒绝），以及**输出上限**以 `max_completion_tokens` 发送（只认 `max_tokens` 的服务会拒绝）。在路由上纠正

```
llm-pi-ai:
  providers:
    my-gateway:
      apiKeyEnv: GATEWAY_API_KEY
      api: openai-completions
      baseURL: https://gateway.example/v1
      compat:
        supportsDeveloperRole: false
        maxTokensField: max_tokens
      models:
        - id: my-model
```

路由的 compat 是其模型的默认值，模型自己的 compat 逐字段胜出。两条纪律：写了名字的开关必须给值（空值会被拒绝而非忽略）；开关按协议生效，张冠李戴会报错并列出该协议支持的开关。全部开关见配置目录中 PiAiCompatProfile 一节——它由源码生成，永远不会落后于适配器实际接受的范围。

4.6 模型切换的正确姿势

Harness 里"切模型"有三个层次，按场景选用：

层次	做法	适用
会话级	模型选择器直接换；新会话跟随新默认	日常手动切换
路由级	同一 Provider ID 下并列多个模型，按需选	一key多模型（V4 Pro/Flash 同路由）
编排级	不同子智能体/任务绑定不同模型	多模型分工协作（第 8 章）

配合会话日志的"模型可见即落账"不变量，每个会话用过哪个模型全程可审计——多模型实验的结果对比因此非常干净。

4.7 故障排查速查表

症状	原因与处置
MISSING_CREDENTIAL	通过 Models 页存 Key，或补上被引用的环境变量
UNKNOWN_MODEL	选择已配置的模型，或在自定义提供商下补上该模型
发现模型返回 401	检查 Key；不支持 GET /models 的端点请手写模型
网关拒绝一切请求但 Key/URL 无误	请求形状差异：先试路由级 compat.supportsDeveloperRole: false 与 compat.maxTokensField:
只有推理模型失败	网关拒绝 developer 角色：设 compat.supportsDeveloperRole: false
手工模型没有 Effort 菜单	未声明挡位：在 settings.yaml 为该模型加 reasoningEfforts
off 挡不住 DeepSeek 模型思考	端点默认思考：给模型或路由设 compat.thinkingFormat: deepseek

兼容开关报"没有值"	冒号后留空了：要么给值，要么删掉该
图片发送前被拒	模型未声明图像模态：加 input: [text, image]；DeepSeek 路由请选用支持图像的目录条目（默认
提供商拒绝了带图请求	声明的模态超出端点实际能力：收窄 input/defaultInput 后开新会话（旧会话日志里仍带着图）

本章小结

- DeepSeek 是一等路由：一个 Key 接入，deepseek-v4-pro/deepseek-flash 即刻可用，推理挡位原生支持；
- Kimi 走内置目录（moonshotai），一键添加；GLM（zai）、OpenAI、Anthropic 同理；
- 任意兼容端点走自定义提供商，三协议选一，Provider ID 一经保存不可改；
- 进阶三件事都在 settings.yaml：input 管图像、reasoningEfforts 管挡位、compat 管请求形状；

切换模型有三个层次，会话历史里的模型使用全程可审计。

练习题

1. 按 4.3 节的方法接入一个 Kimi 模型，并在会话中切换过去验证路由。
2. 写一段 settings.yaml，接入一个 OpenAI 兼容网关（含 compat.supportsDeveloperRole 与 maxTokensField）。
3. 分别说明遇到 401、UNKNOWN_MODEL、图片被拒时各自的排查路径。

第三部分　插件与智能体

第 5 章 插件开发实战：ShuTongBuddy 备考助手插件

> **本章目标**：掌握工具插件与钩子插件的完整写法，理解 defineTool 契约，能独立为备考助手扩展新插件。

本章以一个贯穿始终的真实项目学习插件开发：**为大学生备考助手 ShuTongBuddy 开发一组 DeepSeek Harness 插件**。这个插件包将在第 8 章被 ShuTongBuddy Studio（Web 备考助手）直接调用。学完本章，你不仅掌握了插件开发的全部基本功，还得到了一个可以立刻投入使用的备考智能体后端。

5.1 从业务到插件：备考助手需要什么

先把备考的业务流程摆出来——这是一名大学生准备一门考试时的经典闭环：

```
知识点梳理 → 题库刷题 → AI 答疑 → 错题分析 → 复习计划 → 学习报告
```

逐步映射到 Harness 的机制（第 2、6 章的知识在这里全部派上用场）：

业务阶段	智能体行为	Harness 机制
知识点梳理	读教材与笔记，产出知识图谱与章节清单	会话 + fs 工具
题库刷题	按知识点抽题、判分、记录成绩	自定义工具 question_bank /
AI 答疑	讲解错题、举一反三、扩展同类题	explain_mistake 工具 + 模型（并发时用子智能体
错题分析	归类错因、定位薄弱知识	钩子插件（错题门禁）
复习计划	按遗忘曲线排程，生成每日任务	review 工具（确定性排程）
学习报告	汇总进度、导出周报	报告汇总工具

可以看到：模型负责"讲解与出题"这类语言工作，插件负责"确定性保障"——题目不走样、判分不遗漏、错题必归档、计划可复现。这就是本章要写的五个插件：**工具插件 ×4（题库、刷题、复习、答疑）+ 钩子插件 ×1（错题门禁），外加一套子智能体协同编排**——它们共同撑起备考助手的四件套功能（题库刷题、AI 答疑、复习计划、错题分析）。

5.2 热身：插件的最小形态

在 Harness 里，插件是一个导出 `apply` 函数的 TypeScript 模块。框架加载

```
import type { Context } from '@deepseek-ai/cordis'

export const name = 'my-plugin'

export function apply(ctx: Context) {
  // 在这里注册能力
}
```

name 是插件标识；要消费别的服务（如工具表 tools、模型层 llm），用

```
export const name = 'my-tool-plugin'
export const inject = ['tools']

export function apply(ctx: Context) {
  // 此处 ctx.tools 已就绪
  ctx.tools.register(/* ... */)
}
```

```
mkdir -p scratch-plugin/src           # 1. 建插件目录
```

写 scratch-plugin/cordis.yml 覆盖层（路径用绝对路径，先在仓库根 pwd）

```
- insert:
    - id: hello
      name: '/absolute/path/to/deepseek-harness/scratch-plugin/src/my-plu
```

终端打印出你插件里的日志，说明它已挂进官方插件树。回顾第 2.3 节的层序：patch 覆盖层最后应用，因此它总是最后说话。

可逆效果：通过 `ctx` 注册的一切（监听器、工具、定时器）随插件卸载自

```
export function apply(ctx: Context) {
  ctx.effect(() => {
    const timer = setInterval(() => console.log('heartbeat'), 5000)
    return () => clearInterval(timer)   // 插件卸载时执行
  })
}
```

web Profile 默认实时重载：改完插件代码，旧效果自动回退、新效果自动

挂上，调试反馈环极短。除函数形态外，插件还有对象形态与类形态（插件要作为服务提供给他人时，用继承 Service 的类形态）。

5.3 平台骨架：shu-tong-buddy 插件包

进入正题，规划插件包结构（一个包内多个模块，按职责拆分）：

```
shu-tong-buddy/
  src/
    question-bank.ts    # 题库工具（5.4）
    practice.ts         # 刷题组卷工具（5.5）
    mistake-gate.ts     # 错题门禁钩子（5.6）
    review.ts           # 复习计划工具（5.7）
    tutor.ts            # 答疑工具（5.8）
    state.ts            # 类型与 JSON 持久化
    index.ts            # 插件包汇总入口
```

备考数据的存储选型：直接用一个 JSON 文件放在工作区内。理由：① 刷题中途换会话/换智能体时题库与错题不丢——它们是工作区事实，不属于任何单一对话；② 人类可以随时手工编辑；③ 第 8 章的 ShuTongBuddy Studio 前端可以直接读写同一个文件做题库与错题本面板。这正体现了 Harness 的设计哲学：持久事实落盘，模型上下文从事实投影。

三个状态文件的结构先约定出来，后面每个插件都围绕它们读写：

```
[
  {
    "id": "math-001",
    "subject": "高等数学",
    "topic": "极限",
    "question": "求极限 lim(x→0) sin(x)/x",
    "answer": "1",
    "difficulty": 1,
    "tags": ["基础"]
  }
]
```

mistakes.json（错题本）：

```
[
  {
    "questionId": "math-001",
    "userAnswer": "0",
    "correctAnswer": "1",
    "topic": "极限",
    "recordedAt": "2026-09-14T08:00:00.000Z",
    "reviewCount": 0,
    "mastered": false
  }
]
```

（知识图谱 graph.json 示意）

```
{
  "course": "高等数学",
  "chapters": [
    { "id": "ch01", "title": "函数与极限", "topics": ["函数", "极限", "连续"]
    { "id": "ch02", "title": "导数", "topics": ["导数定义", "求导法则", "高阶
  ]
}
```

三个文件都是"工作区事实"：换会话不丢、可手工编辑、第 8 章前端直接读写。

5.4 工具插件一：题库 question_bank

用 `defineTool` 实现一个多动作题库工具（`add` / `search` / `list` / `stats`）：

```
import type { Context } from '@deepseek-ai/cordis'
import { defineTool } from '@deepseek-ai/dsh-tools'
import { loadJSON, saveJSON, type Question } from './state'

export const name = 'stb-question-bank'
export const inject = ['tools']

export function apply(ctx: Context) {
  ctx.tools.register(defineTool({
    name: 'question_bank',
    description: '备考题库：登记、检索、列出、统计题目。刷题前必须先检索目标知识点。
    parameters: {
      action: { type: 'string', required: true, description: 'add | search
      workspace: { type: 'string', required: true, description: '工作区绝对
      id: { type: 'string', description: '题目 id（add 时必填）' },
      subject: { type: 'string', description: '学科（add 时必填）' },
      topic: { type: 'string', description: '知识点（add / search 时使用）'
      question: { type: 'string', description: '题干（add 时必填）' },
      answer: { type: 'string', description: '参考答案（add 时必填）' },
      difficulty: { type: 'number', description: '难度 1-3（add 时使用）' },
    },
    output: {
      schema: { type: 'object' },
      render: (_args, value) => [{ type: 'text', text: JSON.stringify(valu
    },
    async execute(args, _exec) {
      const bank = await loadJSON<Question[]>(args.workspace, 'question-ba
      switch (args.action) {
        case 'add': {
          const q: Question = {
            id: args.id!, subject: args.subject!, topic: args.topic ?? ''
            question: args.question!, answer: args.answer!,
            difficulty: (args.difficulty as 1 | 2 | 3) ?? 2,
          }
          const i = bank.findIndex(e => e.id === q.id)
          if (i >= 0) bank[i] = q; else bank.push(q)
          await saveJSON(args.workspace, 'question-bank.json', bank)
          return { ok: true, total: bank.length, upserted: q }
        }
        case 'search': {
          const kw = (args.topic ?? '').toLowerCase()
```

```
return {
```

```
hits: bank.filter(q =>
```

```
!kw || q.topic.toLowerCase().includes(kw) || q.question.toL
```

```
}
```

```
}
```

```
case 'list':
```

```
return { total: bank.length, questions: bank }
```

```
case 'stats': {
```

```
const bySubject: Record<string, number> = {}
```

```
for (const q of bank) bySubject[q.subject] = (bySubject[q.subje
```

```
return { total: bank.length, bySubject }
```

```
}
```

```
default:
```

```
    throw new Error(`unknown action: ${args.action}`)
```

```
}
```

```
},
```

```
}))
```

```
}
```

对照第 6 章工具契约检查一遍：parameters 里 action/workspace 必填；execute 返回规范 JSON（命中列表、总数），渲染文本交给 output.render；模型看到的 description 明确写了"刷题前必须先检索目标知识点"——工具描述本身就是提示词工程，注册即装配，它会自动进入系统提示词。

5.5 工具插件二：practice 抽题组卷

`practice` 的设计要点：**出题与判分由模型完成，工具负责确定性部分**——按条件抽题、返回题单、约定落盘路径：

```
import type { Context } from '@deepseek-ai/cordis'
import { defineTool } from '@deepseek-ai/dsh-tools'
import { loadJSON, type Question } from './state'

export const name = 'stb-practice'
export const inject = ['tools']

export function apply(ctx: Context) {
  ctx.tools.register(defineTool({
    name: 'practice',
    description: '抽题组卷：按学科 / 知识点 / 难度抽取题目组成一次练习，返回题单供逐
    parameters: {
      workspace: { type: 'string', required: true, description: '工作区绝对
      subject: { type: 'string', description: '学科（可选）' },
      topic: { type: 'string', description: '知识点（可选）' },
      difficulty: { type: 'number', description: '难度 1-3（可选）' },
      count: { type: 'number', required: true, description: '抽题数量' },
      session: { type: 'string', required: true, description: '本次练习标识
    },
    output: {
      schema: {
        type: 'object',
        properties: {
          session: { type: 'string' },
          picked: { type: 'array' },
          out_path: { type: 'string' },
        },
      },
      render: (_args, value) => [{
        type: 'text',
        text: `已抽取 ${value.picked.length} 题，题单写入 ${value.out_path}`
      }],
    },
    async execute(args, _exec) {
      const bank = await loadJSON<Question[]>(args.workspace, 'question-ba
      let pool = bank
      if (args.subject) pool = pool.filter(q => q.subject === args.subject
      if (args.topic) pool = pool.filter(q => q.topic.includes(args.topic
      if (args.difficulty) pool = pool.filter(q => q.difficulty === args.d
      const picked = pool.slice(0, args.count)
      return {
```

```
session: args.session,
```

```
picked,
```

```
out_path: `practice/${args.session}.json`,
```

```
}
```

```
},
```

```
}))
```

```
}
```

配套地，模型在下一轮逐题作答，并把判分结果用内置 write 工具写入 `practice/<session>-result.json`——判分结果的约定结构（`results` 数组，含 `correct` 布尔）成为第 5.6 节错题门禁拦截的依据。工具的返回值（out_path、picked）进入会话日志，成为后续错题分析与复习计划的事实依据。

5.6 钩子插件：错题门禁 mistake_gate

错题管理的硬指标——**答错的题必须进错题本**——不该靠模型自觉，而该写成门禁。用 `tools/pre-execute` 瀑布拦截判分落盘的动作，发现答错的题就自动归档：

```
import type { Context } from '@deepseek-ai/cordis'
import type { PreToolDecision } from '@deepseek-ai/dsh-tools'
import { loadJSON, saveJSON, type Mistake } from './state'

export const name = 'stb-mistake-gate'
export const inject = ['tools']

export function apply(ctx: Context) {
  ctx.on('tools/pre-execute', async (exec, next): Promise<PreToolDecision
    // 只拦截判分结果落盘
    if (exec.name !== 'write') return next()
    const path = String(exec.arguments.path ?? '')
    if (!path.includes('-result.json')) return next()

    const content = String(exec.arguments.content ?? '')
    let parsed: any
    try {
      parsed = JSON.parse(content)
    } catch {
      return next()   // 非法 JSON 交给下游正常处理
    }

    const results: any[] = parsed?.results ?? []
    const wrong = results.filter(r => r && r.correct === false)
    if (wrong.length === 0) return next()

    // 确定性归档：把错题追加进错题本，作为工作区事实
    const ws = String(exec.arguments.workspace ?? process.cwd())
    const mistakes = await loadJSON<Mistake[]>(ws, 'mistakes.json', [])
    for (const r of wrong) {
      mistakes.push({
        questionId: String(r.questionId),
        userAnswer: String(r.userAnswer ?? ''),
        correctAnswer: String(r.correctAnswer ?? ''),
        topic: String(r.topic ?? ''),
        recordedAt: new Date().toISOString(),
        reviewCount: 0,
        mastered: false,
      })
    }
    await saveJSON(ws, 'mistakes.json', mistakes)
```

```
// 放行，归档结果已落盘，成为后续阶段的事实
```

```
    return next()
```

```
})
```

```
}
```

这是第 5.6 节权限门的同款机制用在错题场景：门禁不打断正常流程，而是把"错题必归档"这条确定性规则焊死在落盘路径上——无论模型记不记得整理错题，错题本都一定更新。

5.7 工具插件三：复习计划 review

四件套的最后一环——**复习计划**——同样遵循"确定性归工具"的原则。遗忘曲线的排程是纯计算：错题的复习次数越多，下次复习的间隔越长。这个计算交给模型既慢又不准，交给工具则一目了然：

```
import type { Context } from '@deepseek-ai/cordis'
import { defineTool } from '@deepseek-ai/dsh-tools'
import { loadJSON, type Mistake } from './state'

export const name = 'stb-review'
export const inject = ['tools']

// 艾宾浩斯遗忘曲线复习间隔（天）
const INTERVALS = [1, 2, 4, 7, 15]

export function apply(ctx: Context) {
  ctx.tools.register(defineTool({
    name: 'review_plan',
    description: '按遗忘曲线为错题生成复习计划：读取错题本，按下次复习日期排程，返回
    parameters: {
      workspace: { type: 'string', required: true, description: '工作区绝对
      days: { type: 'number', description: '排程天数（默认 7）' },
    },
    output: {
      schema: {
        type: 'object',
        properties: {
          total: { type: 'number' },
          schedule: { type: 'array' },
        },
      },
      render: (_args, value) => [{
        type: 'text',
        text: `待复习错题 ${value.total} 道\n` + (value.schedule || []).map
          `  ${s.date}: ${s.items.length} 道（${s.items.map((i: any) => i.
      }],
    },
    async execute(args, _exec) {
      const mistakes = await loadJSON<Mistake[]>(args.workspace, 'mistake
      const pending = mistakes.filter(m => !m.mastered)
      const today = new Date()
      const schedule: any[] = []
      for (const m of pending) {
        // reviewCount 越大，间隔越长（遗忘曲线）
        const interval = INTERVALS[Math.min(m.reviewCount, INTERVALS.leng
        const due = new Date(today.getTime() + interval * 86_400_000)
```

```
const date = due.toISOString().slice(0, 10)
```

```
let slot = schedule.find(s => s.date === date)
```

```
if (!slot) { slot = { date, items: [] }; schedule.push(slot) }
```

```
slot.items.push({ questionId: m.questionId, topic: m.topic })
```

```
}
```

```
schedule.sort((a, b) => a.date.localeCompare(b.date))
```

```
return { total: pending.length, schedule }
```

```
},
```

```
}))
```

```
}
```

对照契约检查：`execute` 只做纯计算——读错题本、过滤未掌握的、按 `reviewCount` 映射遗忘曲线间隔、按日期分桶；返回的 `schedule` 是确定性的排程结果，模型拿到后只需照单安排每日任务。复习的"内容"由模型定，"节奏"由工具定——这正是四件套里"复习计划"与前三者的分工。

5.8 工具插件四：答疑 explain_mistake

答疑是模型的主场——讲解错因、举一反三，本来就是语言模型最擅长的事。工具只需做一件事：**把错题本里的错题干净地取出来，交给模型**。`explain_mistake` 只读不改：

```
import type { Context } from '@deepseek-ai/cordis'
import { defineTool } from '@deepseek-ai/dsh-tools'
import { loadJSON, type Mistake } from './state'

export const name = 'stb-tutor'
export const inject = ['tools']

export function apply(ctx: Context) {
  ctx.tools.register(defineTool({
    name: 'explain_mistake',
    description: '讲解错题：读取错题本中的一道错题，返回题干、用户答案与正确答案，供
    parameters: {
      workspace: { type: 'string', required: true, description: '工作区绝对
      questionId: { type: 'string', required: true, description: '错题 id'
    },
    output: {
      schema: {
        type: 'object',
        properties: {
          questionId: { type: 'string' },
          topic: { type: 'string' },
          userAnswer: { type: 'string' },
          correctAnswer: { type: 'string' },
          reviewCount: { type: 'number' },
        },
      },
      render: (_args, value) => [{
        type: 'text',
        text: `错题 ${value.questionId}（${value.topic}）：你的答案「${value.
      }],
    },
    async execute(args, _exec) {
      const mistakes = await loadJSON<Mistake[]>(args.workspace, 'mistake
      const m = mistakes.find(x => x.questionId === args.questionId)
      if (!m) return { error: `错题不存在：${args.questionId}` }
      return {
        questionId: m.questionId,
        topic: m.topic,
        userAnswer: m.userAnswer,
        correctAnswer: m.correctAnswer,
        reviewCount: m.reviewCount,
```

```
}
```

```
},
```

```
}))
```

```
}
```

注意它的两个设计取舍：① **只读**——讲解不改错题本，`reviewCount` 的递增留给"复习打卡"动作，职责单一；② **返回值是规范 JSON 而非渲染文本**——这样 PTC 模式下（第 6.3 节）程序可以直接拿字段，渲染文本只是给人看的副产品。

并发讲解多道错题时，才需要动子智能体——那属于 5.9 节的"多智能体协同"，用 `spawn-in-process` 为每题开一个分身，会话隔离、错题共享。

5.9 多智能体协同：备考任务的分工

备考六阶段 → 智能体分工与模型配置（第 8 章的 ShuTongBuddy Studio 就按这张表调度）：

阶段	智能体	建议模型	协同机制
知识点梳理	librarian	DeepSeek V4 Pro（high）	主会话，产出 outline.json（知识图谱+章节清单
题库刷题	drill	DeepSeek Flash	主会话调用 question_bank /
AI 答疑	tutor	DeepSeek V4 Pro	ctx.subagents 进程内分身，错题逐一讲解
错题分析	analyst	DeepSeek V4 Pro（max）	主会话 + stb-mistake-gate
复习计划	planner	Flash	review 工具确定性排程
学习报告	reporter	Flash	汇总进度、导出周报

并发讲解多道错题时，每个子智能体持有自己的会话日志，错题约束却来自同一份 `mistakes.json`——**状态共享走文件，上下文隔离走会话**，这是多智能体协同不打架的关键设计。子智能体提供者的选择参考第 6.5 节：本地分身用 `spawn-in-process`，需要绑定不同模型（如答疑给 V4 Pro、刷题给 Flash）的环节用 `acp/dsh-sdk` 委托给另一个配置好的 dsh 实例。

5.10 加载、调试与持久安装

开发期：`pnpm dsh web --patch ./shu-tong-buddy/cordis.yml` 加实时重载；`--dump-config` 确认插件挂树；临时写一个挂 `session/event` 的插件观察全量事件流，是排查"工具没被调用"类问题的杀手锏。

持久安装（给第 8 章的 ShuTongBuddy Studio 用户）：

```
export DSH_HOME=/absolute/path/to/app-dsh-home
dsh plugin --profile sdk add file:/absolute/path/to/shu-tong-buddy
```

dsh plugin 转发 pnpm 完成安装，并把导出 dsh.bundle 层的包登记进该 Profile——之后 SDK 启动 sdk Profile 时，备考助手插件自动就位。

生态：仓库打上 `dsh-plugin` 主题标签即可被社区检索。官方的 `dsh-tool-*`、`dsh-plan-mode`、`dsh-compaction-basic` 都是可照抄的生产级范例。

本章小结

- 插件 = 导出 apply(ctx) 的模块；inject 声明依赖；ctx.effect() 让注册可逆；
- 备考助手的插件化拆解：讲解与出题归模型，确定性保障（题库、判分、错题归档、复习排程）归插件；
- question_bank / practice 展示了 defineTool 契约的实战用法：参数校验、规范 JSON、"描述即提示词"、按条件抽题以节约上下文；
- review_plan 展示了"纯计算归工具"：遗忘曲线排程是确定性计算，交给工具不交给模型；
- explain_mistake 展示了"只读工具"：答疑由模型完成，工具只负责把错题干净地取出来；
- stb-mistake-gate 展示了钩子插件的实战用法：tools/pre-execute 拦截判分落盘，把"错题必归档"焊死在确定性路径上；
- 多智能体协同的核心设计：状态共享走文件，上下文隔离走会话；

持久安装走 `dsh plugin`，为第 8 章的 ShuTongBuddy Studio 铺平了道路。

练习题

1. 给 question_bank 的 search 动作增加一个 `difficulty` 过滤参数。
2. 写一个钩子插件，在 tools/pre-execute 拦截 practice 的抽题，限制单次最多 20 题。

3. 给 review_plan 增加"按 topic 分组"的输出，让每日任务按知识点归类。

第 6 章 智能体构建原理：Turn、装配、上下文与委托

> **本章目标**：理解 Agent 的接口/驱动器分离、提示词装配、PTC 模式、上下文工程与子智能体委托，建立"能力地图"。

第 2 章看了架构全景，第 5 章学会了自己动手。本章聚焦一个问题：**一个"智能体"在 Harness 里究竟由哪些可替换的部件构成？** 读完你会得到一张"能力地图"——想实现任何智能体行为，先查地图找机制，而不是改循环。

6.1 Agent 的两半：接口与驱动器

智能体能力由两个包分治：

- `core/agent`：定义 Agent 接口、实时注册表和 `agent/*` 事件，服务键 `ctx.agents`；
- `core/agent-loop`：实现该接口的默认驱动器，服务键 `ctx.agentLoop`。

"接口"与"驱动"分离意味着：驱动器可以整体替换（比如换成你自己的调度策略），而所有消费 `ctx.agents` 的界面与协议桥——Web UI、ACP、SDK——完全无感。UI 集成的官方路径也由此而来：**驱动** `ctx.agents`，从 `session/event` **渲染**。

驱动器的输入只有一个收件箱（Inbox）：用户消息、注入的上下文、插件的跟进消息，都从这一个口子进入循环。输入被认领后，第 2.6 节那张 Turn/Step 事件序列开始运转。

6.2 系统提示词：装配而非拼接

系统提示词由 `core/system-prompt` 包负责组装（`ctx.systemPrompt`），单位是**节**（Section）：任何插件都可以注册一节提示词，带排序与作用域。两个现成例子：

- `AGENTS.md`（根目录）：一个读取该文件的节提供者——你在仓库根放一份 `AGENTS.md`，它的内容自动成为提示词的一节；
- 子目录 `AGENTS.md` 与文件变更提醒：由 watcher 或工具结果监听器通过 `agent.inject()` 注入，进入下一个被认领的请求。

装配本身也是一个可挂的扩展点（system-prompt/assemble），属于"专家级"的整体变换：它的返回值是权威装配结果，监听者要负责保留进行中的 PTC 模式与结构化输出协议的贡献。只想过滤工具时，官方建议用 ctx.tools.restrict()——它能让"呈现给模型的集合、查找、执行"三者始终对齐，这也是 ToolSearch（渐进式披露）的实现机制。

6.3 工具系统与 PTC 模式

第 5 章从生产者角度看了工具，本章从消费者角度补两块。

作用域与对齐：工具注册表是带作用域的。想让某个会话只看到一部分工具（ToolSearch 场景），替换该作用域的 ctx.tools.restrict() 注册即可；注册表保证呈现、查找、执行三者一致——不会出现"模型看见了却调不到"或"能调用却看不见"的错位。

PTC 模式（程序化工具调用）：在 PTC 模式下，每个可见的注册工具自动成为 await tools.<name>(args)，无需额外集成。参数与返回值的精确类型从同一套模式推导，调用重新进入正常执行管线——策略、审批、审计一个不少。成功时拿到的是**规范 JSON 值**（不是渲染文本），失败时拒绝的是真实 ToolCallError。这就是为什么第 5 章强调"把 output.schema 当程序 API 来设计"。

6.4 上下文工程：投影、注入与压缩

长任务最大的敌人是上下文窗口。Harness 给出三件武器：

（1）**投影（Projection）**：模型上下文从会话日志派生（deriveMessages()），运行时状态由 dsh-session-projection 接缝（ctx.sessionProjections）把已提交事件折叠成类型化状态——Agent 循环就为它的读者注册了共享的 turnBoundary 状态。宿主消费者要么在激活时拿到该服务，要么在缺失时显式失败，不存在"读到半个状态"。

（2）**注入（Injection）**：agent.inject() 追加持久上下文，进入下一个模型请求。它是"给进行中的智能体递纸条"的标准方式：定时器触发、watcher 发现文件变化、后台任务完成，都走这条路。注意它不是唤醒器——空闲的智能体保持空闲，直到下一条真实输入到来。

（3）**压缩（Compaction）**：上下文压缩是一个能力接缝（ctx.compaction），默认实现是 dsh-compaction-basic。三条触发路径分工明确：

- 自动压力压缩：在串行的 agent/pre-step 上运行——每步之前评估压力，防患于未然；
- 溢出恢复：模型请求因超长被拒时，在 agent/request-error 上运行规范恢复；
- 手动压缩：用户手动调用的是同一个 compact 服务。

接缝意味着压缩策略可替换：换成"按主题分段压缩""保留代码块优先压缩叙述"都只需要换一个 Provider。

6.5 子智能体与多智能体协作

复杂任务的解法是分解。**子智能体委托**是一个提供者注册表（`ctx.subagents`），官方自带六种提供者：

提供者	语义
dsh-subagent-spawn-in-process	进程内新建子智能体
dsh-subagent-fork-in-process	进程内 Fork 分支（带上下文）
dsh-subagent-acp	委托给 ACP 对端
dsh-subagent-codex	委托给 Codex 的一轮
dsh-subagent-claude-code	委托给 Claude Code 的一轮
dsh-subagent-dsh-sdk	委托给另一个 dsh SDK 实例

`dsh-tool-subagent` 把配置好的一个提供者暴露给模型——于是"把子任务委派出去"成为模型自己可调用的工具。同一个接口后面，可以是本地的分身，也可以是另一个产品里的同事：这正是能力接缝"一次替换、全局生效"的又一例证。回到 ShuTongBuddy：答疑阶段就是每道错题 spawn 一个子智能体并行讲解，同一个 `ctx.subagents` 接口既可以是进程内分身，也可以是另一个 dsh 实例里的同事。

再往上是实验性的 Agent Teams（`ctx.agentTeams`）：在可续接的子智能体之上，提供持久名册（roster）、任务看板（task board）与信箱（mailbox）的多智能体协调。想给单个会话配不同能力集，则通过组合 Agent 预设（preset）实现——其中的服务行需要 `isolate` 域。

6.6 计划、目标与循环：给智能体装上"工作习惯"

一批实用的"工作习惯"功能，同样全部落在扩展点上：

- 计划模式（Plan Mode）：dsh-plan-mode 包——落账的 plan/mode 状态、plan:policy 指导节、/plan [message] 进入、/plan off 直接退出，以及经用户评审的 exit_plan_mode。执行约束保持在独立的沙箱与审批轴上，计划模式本身只管"先规划、后动手"的行为约定；
- 目标（/goal）：ctx.goals 持有持久状态，dsh-goal-round-driver 通过公开 Agent 接口调度同会话的推进轮次，命令与工具两个入口分别面向人和

模型；

- 循环（/loop）：在 turn/end 会话事件上 followup() 下一轮，或强制续接——十几行逻辑就是一个定时迭代器；
- 定时任务：插件注册模型可调度的工具，时间到则 followup()（空闲时）或 inject() 通知（忙碌时）；
- 技能（Skills）与记忆（Memory）：均为"节提供者 + 工具"的组合——技能在被调用时把内容 inject() 进去，记忆则同时维护提示词节与读写工具。

把这张表和第 2 章的"功能 → 机制"对照表连起来看，你会发现一个惊人的事实：**这些都不是框架内置的特例，而是插件**。这正是微内核主张最有说服力的证据，也是你设计自己的智能体行为时的范本库。

本章小结

- Agent = 接口（ctx.agents）+ 可替换驱动器（ctx.agentLoop）；输入走单一收件箱；
- 提示词是"节"的装配：AGENTS.md、技能、记忆都是节提供者；
- 工具的作用域限制用 ctx.tools.restrict()；PTC 模式让工具成为可编程 API；
- 上下文三件套：日志投影、inject() 注入、可替换的压缩接缝；
- 子智能体是提供者注册表，六种官方实现覆盖本地分身到跨产品委托；Agent Teams 提供实验性多智能体协调；
- 计划、目标、循环、定时、技能、记忆——全是插件，全部可仿写。

练习题

1. 说明 PTC 模式下工具调用与普通工具调用的区别，以及它为什么要求把 output.schema 当程序 API 设计。
2. 列举上下文压缩的三条触发路径（自动压力、溢出恢复、手动），各举一个场景。
3. 六种子智能体提供者（spawn-in-process 等）分别适合什么场景？备考答疑该用哪个？

第四部分　实战案例

第 7 章 实例一：备考资料库体检智能体

> **本章目标**：用内置能力完成备考资料库体检，掌握 headless 批量化运行与自定义统计工具的编写。

第一个实例不急着写插件——先用 Harness 开箱即用的能力，完成一个真实任务：对大学生的备考资料库做一次"体检"，产出一份结构化报告。你会发现，仅靠内置工具与一个好模型，事情已经能做成八成；剩下两成，再用一个自定义工具补上。

7.1 任务定义

输入：一个备考资料库（本地目录，含教材 PDF、课堂笔记、历年试卷、错题集等，建议先用一门课的小型资料库练手）。

产出：`./study_report.md`，包含：

1. 资料库结构总览与主要科目/章节识别；
2. 资料盘点与潜在缺口（缺失章节、过期资料、薄弱知识点）；
3. 资料规模统计（文件数、类型分布、主要科目占比）。

模型选择：`deepseek-v4-pro`，推理挡位 `high`——体检是分析型任务，值得用 Pro 的深度。

7.2 准备工作

```
cd /path/to/course-materials    # 在资料库目录启动，便于圈定工作区
npx @deepseek-ai/dsh web
```

界面中完成三件事：确认 DeepSeek 路由可用（第 4 章）；Choose workspace 选中该资料库；模型选择器切到 deepseek-v4-pro，Effort 选 high

安全提示：体检涉及读取资料库里的文件，Harness 会按权限策略处理。对一个陌生资料库，建议先只允许"读取类"操作、拒绝"执行/改写类"操作——这正是 3.7 节"先审后放"的实战。

7.3 第一轮：结构总览

发送

```
总结这个资料库的结构，找出主要的科目和章节，说明它们各自的覆盖范围，并指出入口文件（如课
```

观察过程（对照第 2.6 节的事件序列）：

- 智能体规划出"先看目录树" → 读大纲/目录文件 →

抽样读笔记与试卷"的步骤；

- 工具调用卡片逐条出现：列目录、读文件、搜索关键词；
- 最终给出分层的结构说明。

如果它漏掉了某个重要科目目录，**直接在同一会话里追问纠偏**——会话日

```
「概率论」目录下还有一个往年真题子目录你没有分析，补上。
```

7.4 第二轮：缺口盘点与报告落盘

```
基于你已掌握的情况，盘点以下问题：1）哪些章节缺教材或缺笔记；2）哪些知识点没有对应练习题
```

两个看点：

- 写文件是 diff 卡片：write
  工具的调用在界面上呈现为逐文件的差异卡片，落盘前你能清楚看到它打算写什么；
- 报告质量取决于你已给的上下文：因为它"记得"前两轮看到的一切，报告会引用具体文件与章节，而不是空泛的套话。

如果任务中断（网络抖动、主动取消），会话仍在日志里——重开界面回到该会话即可继续，这就是"模型可见即落账"的日常红利。

7.5 批量化：headless 一键体检

交互式跑顺之后，把它变成可重复的命令——`headless` Profile 就是干这

```
dsh --profile headless "阅读当前目录的备考资料库，生成中文体检报告写入 STUDY_REPO
```

脚本化后，体检可以挂进学期初的资料整理流程或定时任务。注意 headless 只在启动时应用一次补丁层（第 2.3 节），且没有人在审批提示前把关——只在可信资料库上这样跑，或先配合第 5 章的权限门插件收紧策略。

7.6 补上短板：自定义资料统计工具

到这里你会发现一个短板：让模型一行行"数文件、算大小"既慢又不准。正确的做法是把确定性工作交给确定性工具——写一个 `material_stats` 工具插件，把第 5 章的知识用一遍。

在源码检出下创建 `scratch-plugin/src/material-stats.ts`：

```
import { readdir, stat } from 'node:fs/promises'
import { join, extname } from 'node:path'
import type { Context } from '@deepseek-ai/cordis'
import { defineTool } from '@deepseek-ai/dsh-tools'

const SKIP = new Set(['node_modules', '.git', '.obsidian', '.trash'])
const TYPE: Record<string, string> = {
  '.pdf': 'PDF 教材/试卷', '.md': 'Markdown 笔记', '.docx': 'Word 文档',
  '.txt': '纯文本', '.pptx': '课件', '.png': '图片', '.jpg': '图片',
}

async function walk(dir: string, acc: Record<string, { files: number; byte
  for (const entry of await readdir(dir, { withFileTypes: true })) {
    if (SKIP.has(entry.name)) continue
    const full = join(dir, entry.name)
    if (entry.isDirectory()) { await walk(full, acc); continue }
    const type = TYPE[extname(entry.name).toLowerCase()]
    if (!type) continue
    const s = await stat(full)
    const slot = (acc[type] ??= { files: 0, bytes: 0 })
    slot.files += 1
    slot.bytes += s.size
  }
  return acc
}

export const name = 'material-stats'
export const inject = ['tools']

export function apply(ctx: Context) {
  ctx.tools.register(defineTool({
    name: 'material_stats',
    description: '统计备考资料库中各类型文件的数量与总大小。',
    parameters: {
      path: { type: 'string', required: true, description: '资料库目录绝对路
    },
    output: {
      schema: {
        type: 'object',
        additionalProperties: {
          type: 'object',
```

```
properties: { files: { type: 'number' }, bytes: { type: 'number
```

```
},
```

```
},
```

```
render: (_args, value) => [{
```

```
type: 'text',
```

```
text: Object.entries(value)
```

```
.map(([t, s]) => `${t}: ${s.files} 个文件, ${(s.bytes / 1024 / 1
```

```
.join('\n'),
```

```
}],
```

```
},
```

```
async execute(args, exec) {
```

```
return walk(args.path, {})
```

```
},
```

```
}))
```

```
}
```

【待核：output.schema 的 additionalProperties 写法以你所用版本的 ValueSchemaSpec 为准；若不被接受，可改为返回数组 [{ type, files, bytes }]。】无需改代码

```
# scratch-plugin/cordis.yml
- insert:
    - id: material-stats
      name: '/absolute/path/to/deepseek-harness/scratch-plugin/src/materi
```

然后在会话里说

```
调用 material_stats 工具统计当前资料库，把结果并入 STUDY_REPORT.md 的「资料规模」
```

模型会在下一轮看到新工具的模式并主动调用它——注册即装配，不需要你改任何提示词。

本章小结

用到的能力	出处
工作区与权限审批	第 1、3 章
DeepSeek V4 Pro + 推理挡位	第 4 章
多轮会话与上下文记忆	第 2.7、6.4 节
headless 一次性运行	第 3.4 节
defineTool 自定义工具 + Patch 挂载	第 5 章
注册即装配（模型自动发现新工具）	第 6.3 节

一个"备考资料库体检智能体"就这样从纯交互式使用，渐进增强为"交互 + 自定义工具 + 可脚本化"的完整方案——全程没有修改 Harness 一行源码。

练习题

1. 对自己的备考资料库跑一次 headless 体检，生成 STUDY_REPORT.md。
2. 给 material_stats 增加"按一级子目录（科目）分组"的输出。
3. 把体检结果与错题本（mistakes.json）结合，产出一份"薄弱知识点"复习建议。

第 8 章 实例二：ShuTongBuddy Studio（Web 备考助手）

> **本章目标**：掌握 Web + FastAPI + Harness Python SDK 的完整应用构建，理解六阶段流水线与阶段级多模型分工。

终章实例把前七章攒成一件完整作品：一个 Web 端的**大学生备考助手** ShuTongBuddy Studio。前端跑在浏览器里，界面布局参考 Codex 风格；后端用 FastAPI 编排，智能体引擎就是 DeepSeek Harness——第 5 章写好的备考助手插件包在这里正式上岗。业务流程严格走六阶段：**知识点梳理 → 题库刷题 → AI 答疑 → 错题分析 → 复习计划 → 学习报告**，多模型分工与多智能体协同贯穿始终。

8.1 技术选型：为什么是 Web

决策	选择	理由
界面	Web（浏览器访问）	跨平台免安装：大学生 Windows / Mac / 手机浏览器都能用，无需
前端	原生 HTML + JS（可升级 Vite + React）	三栏 Codex 布局轻量可教，与后端
后端	FastAPI	Python 生态，异步长任务友好，
智能体引擎	DeepSeek Harness（Python SDK）	不重造轮子：工具、会话日志、权限、子智能体、多模型路由全部现成
连接方式	deepseek-harness-sdk pip 包	安装即带匹配的原生运行时与 dsh 命令，无需系统 Node.js
备考能力	第 5 章插件包	题库、刷题、错题门禁，一次开发两端复用

一句话架构：**浏览器负责"人"的体验，FastAPI 负责编排，Harness 负责"智能体"的一切**。

8.2 总体架构

进程模型要点：harness.run() 是阻塞调用，必须放进后台线程 worker，通过 SSE 把阶段进度与输出推回浏览器——前端界面永不冻结。

8.3 环境搭建与多模型配置

> **完整可运行代码**：本章核心代码已整理为可运行示例，位于仓库 `examples/shu-tong-buddy-studio/`（controller.py、pipeline.py、app.py、static/index.html、requirements.txt）。本章展示关键代码并逐段讲解，读者可对照示例从零跑通。

```
py -3.10 -m venv .venv
.venv\Scripts\Activate.ps1
python -m pip install deepseek-harness-sdk fastapi uvicorn
$env:DEEPSEEK_API_KEY = "sk-your-key-here"
```

```
$env:DSH_HOME = "C:\work\shutongbuddy-dsh-home"
dsh --profile sdk --dump-default-config | Out-Null
dsh plugin --profile sdk add file:C:/work/shu-tong-buddy
```

多模型配置：sdk Profile 与 Web 共用同一套 settings.yaml 体系——在 $DSH_HOME/settings.yaml 里按第 4 章的方法加入 moonshotai（Kimi）提供商，答疑阶段即可路由到 Kimi。DeepSeek 侧，deepseek-v4-pro 与

deepseek-v4-flash 开箱可用。【待核：sdk Profile 下多提供商的会话级指定方式，以你所用版本的 SDK 参考为准；备选方案是为 Kimi 单独起一个 dsh 实例，通过子智能体 ACP 委托，见 8.7 节。】

8.4 核心代码一：HarnessController

对 SDK 的封装，一次启动一阶段化执行，线程内复用

```
from pathlib import Path
from deepseek_harness import DeepSeekHarness

STAGE_MODEL = {
    "organize": ("deepseek-official", "deepseek-v4-pro"),    # 知识点梳理
    "drill":    ("deepseek-official", "deepseek-v4-flash"),  # 题库刷题
    "tutor":    ("deepseek-official", "deepseek-v4-pro"),    # AI 答疑
    "analyze":  ("deepseek-official", "deepseek-v4-pro"),    # 错题分析
    "plan":     ("deepseek-official", "deepseek-v4-flash"),  # 复习计划
    "report":   ("deepseek-official", "deepseek-v4-flash"),  # 学习报告
}

class HarnessController:
    """每个阶段使用独立的 DeepSeekHarness 实例（各自懒启动 dsh 进程），
    会话日志共享同一个 dsh_home，题库与错题本共享同一个 workspace。"""

    def __init__(self, workspace: str, dsh_home: str):
        self.workspace = str(Path(workspace).resolve())
        self.dsh_home = str(Path(dsh_home).resolve())

    def run_stage(self, stage: str, prompt: str, session_id: str) -> str:
        provider, model = STAGE_MODEL[stage]
        with DeepSeekHarness(
            provider=provider,
            model=model,
            max_tokens=49_152,
            cwd=self.workspace,
            dsh_home=self.dsh_home,
            profile="sdk",
        ) as harness:
            result = harness.run(prompt, session_id=session_id)
        return result.final_response
```

三个设计说明：

- `session_id` 策略：同一阶段内续用同一

id（多轮纠偏保留上下文），跨阶段换新
id（上下文隔离，靠工作区文件传递事实——正是第 5.7
节"状态共享走文件"的落地）；

- workspace 与 dsh_home 显式隔离：SDK 与示例都不会静默读取
~/.dsh，多门课程的备考项目天然互不污染；
- 安全前提：sdk-minimal Profile 锚定
danger-full-access，生产用 sdk Profile
并配合权限策略，工作区务必指向可隔离的目录（3.7
节安全基线在此同样适用）。

8.5 核心代码二：备考多阶段流水线

编排层把业务流程写成一段纯粹的数据与函数：

```
PIPELINE = [
    ("organize", "知识点梳理",  "通读 {course} 的教材与笔记，产出知识图谱 outlin
    ("drill",    "题库刷题",    "按 outline.json 的章节清单，逐章调用 practice
    ("tutor",    "AI 答疑",     "逐题讲解错题本中的题目，举一反三给出同类题。"),
    ("analyze",  "错题分析",    "对照错题本归类错因、定位薄弱知识点，输出 analysi
    ("plan",     "复习计划",    "调用 review 工具按遗忘曲线排程，生成每日复习任务
    ("report",   "学习报告",    "汇总学习进度与错题统计，生成周报 report.md。"),
]

def run_pipeline(course: str, controller: HarnessController, on_stage):
    for stage, title, tmpl in PIPELINE:
        on_stage(stage, title, "running")                # → SSE 事件
        prompt = tmpl.format(course=course)
        output = controller.run_stage(stage, prompt, session_id=f"{course
        on_stage(stage, title, "done", output)           # → SSE 事件
```

刷题与答疑阶段的并发：错题之间无依赖时，可为每题起一个子智能体（第 6.5 节的 spawn-in-process 提供者），并发度受 API 配额约束；错题约束天然一致——所有子智能体读同一份 mistakes.json。判分落盘环节则有第 5.6 节的 stb-mistake-gate 门禁兜底：答错的题被自动归档进错题本，不依赖模型自觉。

8.6 核心代码三：Web 界面（Codex 布局）

后端用 FastAPI 起服务，后台线程跑流水线，SSE 把阶段事件推给前端：

```
import asyncio, json
from fastapi import FastAPI
from fastapi.responses import StreamingResponse, HTMLResponse

app = FastAPI()
controller = HarnessController(workspace="C:/work/course-math", dsh_home=

@app.post("/pipeline/start")
async def start_pipeline(course: str):
    async def event_stream():
        q = asyncio.Queue()
        def emit(stage, title, status, output=""):
            q.put_nowait(json.dumps({"stage": stage, "title": title,
                                     "status": status, "output": output}))
        # 后台线程跑流水线，事件经线程安全队列转 SSE
        import threading
        threading.Thread(
            target=run_pipeline, args=(course, controller, emit), daemon="
        ).start()
        while True:
            yield f"data: {await q.get()}\n\n"

    return StreamingResponse(event_stream(), media_type="text/event-strea

@app.get("/")
async def index():
    return HTMLResponse(open("index.html", encoding="utf-8").read())
```

前端骨架——三栏 + 底部输入框，EventSource 订阅阶段事件：

```
<!-- index.html -->
<div id="app">
  <aside id="stages"></aside>          <!-- 左栏：阶段/会话 -->
  <main id="chat"></main>              <!-- 中栏：对话流 + 工具卡片 -->
  <aside id="panel"></aside>           <!-- 右栏：题库/错题本 -->
</div>
<script>
const es = new EventSource('/pipeline/start?course=高数');
es.onmessage = (e) => {
  const ev = JSON.parse(e.data);
  const card = document.createElement('div');
  card.textContent = `[${ev.title}] ${ev.status} ${ev.output}`;
  document.getElementById('chat').appendChild(card);   // 渲染为卡片
  if (ev.status === 'done') refreshPanel();            // 重读 question-ba
};
</script>
```

与 Codex 布局的对应关系：左栏 ≈ 会话/任务列表；中栏对话流里每个阶段事件渲染成卡片（工具调用卡片显示 question_bank、practice、write 等调用摘要）≈ Codex 的消息流与工具卡；右栏 ≈ 工作区侧边栏，直接读写工作区的 question-bank.json 与 mistakes.json，实现人机共编同一题库——界面登记的题目，智能体下一轮刷题就用到。底部输入框 ≈ composer。这就是 Web 相对桌面的红利：同样的三栏布局，浏览器一行命令跑起来，任何设备都能开。

8.7 多模型与多智能体的落法

把第 4、6 章的机制落到这个平台上，收敛为三条可执行规则：

1. 阶段级模型分工：梳理/答疑/错题分析给 V4 Pro（错题分析开 max 挡位），刷题/复习/报告给 Flash，答疑切 Kimi——切换点都在 STAGE_MODEL 一张表里，改配置不改代码；
2. 题目级智能体分身：答疑阶段每道错题一个子智能体，会话隔离、文件共享，门禁兜底；
3. 跨实例委托：若 Kimi 路由在 sdk Profile 下不便指定，为 Kimi 单独起一个 dsh 实例，用 dsh-subagent-acp/dsh-sdk 把答疑阶段委托过去——主流程代码不变。

平台边界之外的功夫：备考助手的"平台感"恰恰来自这些 Harness 原生能力——会话日志让每门课程的备考全程可回放审计；`--dump-config` 让部

署环境可检视；补丁层让不同学生可以拿到不同门禁严格度的定制版。

本章小结

- Web（Codex 布局）+ FastAPI + Harness Python SDK 构成"体验层 + 编排层 + 智能体层"的干净分层；
- DeepSeekHarness 按阶段实例化，session_id 控制上下文边界，工作区文件承担跨阶段事实；
- 六阶段流水线 = 一份数据表 + 一个循环；多模型分工 = 一张 STAGE_MODEL 表；
- 第 5 章插件包在此复用：题库、刷题、错题门禁全部在线。

进阶：打包成桌面版——若学生想要"双击即用"，用 Tauri 把这套 Web 前端包成桌面壳：Tauri 内置一个本地 WebView 指向本地 FastAPI 服务（启动时自动拉起 dsh 与后端），一套 Web 代码同时获得桌面形态。这是"Web 优先、桌面可打包"的完整闭环，核心代码零改动。

入机共编题库、全程可审计的会话日志，是这个备考助手区别于"脚本调用 API"的本质。

练习题

1. 把 `examples/shu-tong-buddy-studio/` 跑起来，观察六阶段流水线在浏览器中的推进。
2. 给 PIPELINE 增加一个"模拟考试"阶段（调用 practice 抽题 + 限时作答 + 判分）。
3. （进阶）用 Tauri 把 static/index.html 打包成桌面应用，实现双击即用。

结束语：智能体工程及其未来发展方向

如今，智能体技术正以前所未有的速度演进，诸多颇具前景的工程模式既来自开源社区，也源于前沿模型开发者，共同塑造着未来智能体系统的设计范式。本书所依托的 DeepSeek Harness，正是这一浪潮中的一个典型样本。

万物皆插件的微内核架构正在成为智能体基座的共识。Harness 的 Cordis 插件框架、Profile/Bundle/Patch 组合机制、能力接缝与可逆效果，把"扩展"从框架作者的预判变成了使用者的权利——不再有需要打补丁的特权核心，加能力、换模型、改循环，都是挂一个插件。可以预见，随着智能体应用从"脚本调用 API"走向"可组合的插件生态"，这类架构会成为行业的基础设施。

多智能体协同是另一个清晰的方向。本书第 6、8 章展示了"状态共享走文件、上下文隔离走会话"这一朴素而有效的协同原则，以及本地分身、跨实例委托、Agent Teams 等多种协作形态。当单智能体不再能满足长任务、多角色的需求时，可续接的子智能体、持久的任务看板与信箱，将把"智能体"升级为"智能体团队"。

从助手到平台，是本书贯穿案例 ShuTongBuddy 给出的另一层启示。一个备考助手，本质上是一组插件加一个界面；但会话日志让全程可回放审计，补丁层让不同用户拿到不同门禁严格度的定制版，能力接缝让模型、工具、界面都能整体替换——正是这些"平台感"，让它区别于一段单纯的脚本。

展望未来，智能体工程还将与推理模型、检索增强、代码执行沙箱、权限治理等能力进一步融合。无论技术如何演进，本书希望传递的方法论不会过时：**把确定性交给插件，把语言工作交给模型，把事实落盘，把上下文投影**。掌握了这套思维方式，读者便能在快速演进的智能体领域游刃有余。

愿这本小册子，成为你踏上智能体工程之路的第一位"书童"。

附录

附录 A 命令速查

命令	作用
npx @deepseek-ai/dsh web	启动 Web UI（默认 http://127.0.0.1:3080）
npx @deepseek-ai/dsh web --no-open	启动但不自动打开浏览器
dsh --profile <name>	以指定 Profile 启动
dsh --profile headless "任务"	一次性运行：新会话、打印答案、退出
dsh --profile <name> --from-default-profile web	从官方模板创建自定义 Profile
dsh --profile <name> --dump-config	输出实际组合的插件树（不启动）
dsh --profile <name> --dump-default-config	输出默认组合的插件树
dsh plugin --profile <name> add	为 Profile 安装外部插件（转发
pnpm dsh web --patch ./x/cordis.yml	源码方式携带补丁覆盖层启动
pnpm install && pnpm run build	源码检出后的安装与构建

附录 B 关键文件位置（$DSH_HOME）

路径	内容
settings.yaml	全部设置（模型路由进阶字段在此编辑，热生效）
.credentials.yaml	API 密钥本体（只写存储）
cordis.patch.yml	主目录级补丁层
profiles/<name>/	Profile 目录（package.json + cordis.patch.yml）

附录 C settings.yaml 高频字段速查

字段	作用

llm-deepseek.reasoningEffort	DeepSeek 路由默认推理挡位（off/low/high/max
llm-pi-ai.providers.<id>.api	协议：openai-completions / openai-responses /
...models[].input	模型模态，如 [text, image]
...providers.<id>.defaultInput	路由级模态兜底（默认 [text]）
...models[].reasoningEfforts	手工模型的推理挡位声明
...compat.supportsDeveloperRole	网关兼容：拒绝 developer 角色时设 false
...compat.maxTokensField	网关兼容：改为 max_tokens
...compat.thinkingFormat: deepseek	默认思考型 DeepSeek 模型的开关格式

附录 D 故障排查一行表

MISSING_CREDENTIAL → 存 Key 或补环境变量；UNKNOWN_MODEL → 选已配置模型或补录；发现模型 401 → 查 Key 或手写模型；网关全拒 → compat.supportsDeveloperRole: false + maxTokensField: max_tokens；off 挡不住思考 → compat.thinkingFormat: deepseek；图片被拒 → 声明 input: [text, image] 并开新会话。

附录 E 资源链接

资源	地址
项目仓库	https://github.com/deepseek-ai/deepseek-harness
官方文档站	https://deepseek-harness.github.io/deepseek-harness/
问题反馈（Discussions）	https://github.com/deepseek-ai/deepseek-harness/discussions
插件发现标签	GitHub Topic：dsh-plugin
Cordis 框架	https://github.com/cordiverse/cordi
Cordis 设计论文	https://arxiv.org/abs/2608.25512
DeepSeek 开放平台	https://platform.deepseek.com/

附录 F 术语表（Glossary）

术语	英文	含义
插件	Plugin	导出 `apply` 函数的模块，向共享上下
上下文	Context（`ctx`）	Cordis 的共享上下文，插件通过它注册服务、监听事件、
会话	Session	一次持续的人机协作；底层是一份只追加的事件日
轮次	Turn	从零到多个 Step 构成；在"不再欠任何工
步骤	Step	一次模型请求加上它所触发的工具调用
工具	Tool	面向模型的能力单元，注册到 `ctx.tools`
运行配置	Profile	一组命名好的插件树组合：web、headless、sdk、sdk-minimal、acp
能力接缝	Seam	一个可替换能力的三角色契约：接口定义、提供者、消费者
微内核	Microkernel	没有特权核心的架构，一切皆插件
补丁	Patch	覆盖层，按层序替换插件树中的配置
可逆效果	Reversible Effects	注册即效果，卸载即回退
程序化工具调用	PTC（Programmatic Tool Calls）	工具成为可编程 API，返回规范 JSON
投影	Projection	从会话日志派生模型上下
注入	Injection	`agent.inject()` 追加持久上下文，进入下一个模型请求
压缩	Compaction	上下文压缩，一个可替换的能力接缝

子智能体	Subagent	通过 `ctx.subagents` 委托的智能体分身

初稿说明：本册基于 DeepSeek Harness 开发者预览版（0.1.5-rc.2）官方文档撰写，项目迭代迅速，若行为与描述不符，以仓库文档为准；文中【待核】处请在出版前对照最新文档确认。

第一部分　入门

第二部分　部署与模型

第三部分　插件与智能体

第四部分　实战案例

结语与附录

- [附录](12-appendix.md)