

橙皮书

Hermes Agent

从入门到精通

第一个出厂就带缰绳的 AI Agent
Nous Research 开源框架实战指南

The Agent That Grows With You

关键词：自改进Agent · 跨会话记忆 · Skill系统 · MCP · 多平台

适合读者：想搭建个人AI Agent的开发者和AI爱好者

版本：v260407

花叔

公众号「花叔」· B站「AI进化论-花生」

本手册基于 Hermes Agent v0.7.0 编写。AI 工具迭代迅速，部分内容可能随版本更新而变化，请以官方文档为准。

目录

CONTENTS

Part 1: 概念

§01 不是又一个Agent：从Harness到Hermes

§02 Hermes Agent全景：60秒看懂

Part 2: 核心机制

§03 学习循环：Agent自己给自己造绳绳

§04 三层记忆：从金鱼到老友

§05 Skill系统：会自我进化的能力

§06 40+工具与MCP：连接一切

Part 3: 动手搭建

§07 安装与配置：三种方式

§08 第一次对话：让Hermes认识你

§09 多平台接入：在哪都能找到它

§10 自定义Skill：教Hermes新技能

§11 MCP集成：连接你的工具栈

Part 4: 实战场景

§12 个人知识助手：跨会话记忆的威力

§13 开发自动化：代码审查到部署

§14 内容创作：从调研到成稿

§15 多Agent编排：让三匹马同时跑

Part 5: 深度思考

§16 Hermes vs OpenClaw vs Claude Code：不是选择题

§17 自改进Agent的边界：它能走多远

§01 不是又一个Agent：从Harness到Hermes

Not Another Agent: From Harness to Hermes

OpenClaw热潮还没散，Hermes Agent就来了。它不是「又一个Agent工具」，而是Harness Engineering概念的第一次产品化。

又来一个？

我理解你的疲惫。

OpenClaw在2025年底掀起了龙虾热。2600万用户，国内大厂纷纷跟进做自己的龙虾类产品。那段时间你的朋友圈大概率被「我养了一只虾」刷过屏。龙虾热还没完全散去，又有一个新东西冒出来了。

2026年2月，Nous Research发布了Hermes Agent。两个月不到，GitHub stars飙到27000+。

你的第一反应可能是：龙虾我还没搞明白，又来？

我花了一周时间把Hermes从头到尾拆了一遍，发现它和OpenClaw走的是完全不同的路。**Hermes不是又一个龙虾**，它在做一件我们一直在讨论但没人做成产品的事。

Harness Engineering是什么

如果你读过上一本橙皮书《Harness Engineering》，这部分可以跳过。没读过的话，30秒讲清楚。

2026年初，AI编程圈出现了一个共识：**瓶颈不是模型，是环境**。LangChain团队做了一个实验，用同一个模型（GPT-5.2-Codex），只调整周围的「缰绳」配置，成绩从52.8%涨到66.5%，排名从Top 30跳到Top 5。模型一行没改。

Mitchell Hashimoto（Terraform的创造者）第一个给这件事命名：**Harness Engineering**。他的做法很朴素，每次AI犯错，就加一条规则，让它永远不再犯同一个错。文件是活的，一直在长。

我在那本书里把Harness拆成了五个组件。这五个组件，是理解Hermes的钥匙。

五组件映射

Harness Engineering讲的是方法论，是「你应该给AI造什么样的缰绳」。但方法论有一个问题：**执行全靠人**。你得自己写CLAUDE.md，自己配hooks，自己搭记忆系统，自己设计 workflow。

Hermes做的事情是：把这五个组件全部内建了。

Harness五组件	手动实现方式	Hermes内建系统
指令层	手写CLAUDE.md / AGENTS.md	Skill系统（markdown skill文件，自动创建+自改进）
约束层	配置hooks / linter / CI	Tool permissions + sandbox + toolset按需启用
反馈层	人工审查 / 评估者Agent	自改进学习循环（完成任务后自动复盘优化）
记忆层	手动维护knowledge base	三层记忆（会话/持久/Skill）+ Honcho用户建模
编排层	自己搭多Agent pipeline	子Agent委派 + cron调度

看左列和右列的对比。左边全是手动操作，你得是一个有经验的工程师才能搭出来。右边是开箱即用，**装完就有**。

这就是Hermes和OpenClaw的本质区别。OpenClaw给你一套配置即行为的系统，你写SOUL.md，它变成你想要的样子。它的记忆系统功能完善（Daily Logs + MEMORY.md + 语义搜索），Skill生态庞大，但主要靠人工编写和维护。Hermes把这五个维度全部内建了，而且让它们自动运转。

概念连接：如果你用过Claude Code的CLAUDE.md + hooks + memory，你已经在手动实现Harness了。Hermes做的事情是把这套手动流程变成了一个自动运行的系统。从「你给AI造缰绳」变成「AI自己给自己造缰绳」。

Nous Research为什么做这件事

做Hermes Agent的不是大公司，是一个开源AI研究实验室。

Nous Research在AI圈被形容为「开源社区中的神秘力量」。核心人物Teknium主导微调和数据集策划，早期靠Redmond AI提供算力支持，团队规模一直不大。但他们做出来的Hermes模型家族（从8B到Hermes 4的14B/70B/405B），**纯靠后训练就达到了前沿水平**。不需要从头预训练，不需要巨额算力预算。

这个理念延续到了Hermes Agent上：**用开源工具+任意LLM API，个人也能部署媲美商业方案的AI Agent**。MIT许可证，完全开源。

他们的核心原则挺鲜明的。用户控制优先，模型是可控的（steerable），你可以按需调整行为，不受企业内容策略限制。审查这事儿他们明确不做，声明模型是「unencumbered by censorship, neutrally aligned」。同时在创造力和数学、编码、推理性能上都不妥协。

这些原则决定了Hermes Agent的设计取向：**不是替你做决定，而是你设规则、它学规则、然后越做越好**。控制权始终在用户手里，执行的复杂度由系统承担。

不是替代，是递进

有一个常见的误解：Hermes要取代Claude Code或OpenClaw。其实不是，这三个工具解决的是不同层面的问题。

Claude Code做交互式编码。你坐在终端前，和它来回对话，实时协作。它是你的结对工程师。

OpenClaw做配置即行为。你写一个SOUL.md，它就变成你想要的样子。配置透明，生态成熟，ClawHub有44000+社区Skill。

Hermes做自主后台+自改进。你不需要坐在它旁边。它自己跑，自己学，自己进化。24/7在线，通过Telegram或Discord随时可达。

一个有意思的点是，这三个工具都采用agentskills.io标准，Skill可以互通。你在Claude Code里写的Skill，Hermes也能用，反过来也一样。它们不是三条平行线，更像是一个生态里分工不同的三个角色。

核心建议

如果你只是写代码，Claude Code够用了。如果你想要一个7x24小时帮你看守任务、会自己变聪明的后台Agent，那就该看Hermes了。

出厂就带缰绳

回到开头的问题：为什么这一次不一样？

龙虾热教会了普通人一件事：AI可以是「你养出来的」，不是「你打开就用的」。OpenClaw的SOUL.md、记忆系统、个性化配置，让人第一次体验到什么叫「我的AI」。

但养虾的人也发现了一个问题：**缰绳全得自己造**。写SOUL.md，手动调技能，定期清理记忆。龙虾越用越好用，前提是你愿意花时间喂它。

Hermes的思路不同：**把缰绳直接焊进出厂设置，而且让缰绳自己长大**。

你第一次和Hermes对话，它就开始自动写入记忆、自动提炼Skill、自动优化工作流。用的时间越长，它对你的理解越深，做事的质量越高。不是你在训练它，是它在训练自己。

Hermes是第一个出厂就带缰绳的Agent。而且缰绳会自己长大。

§02 Hermes Agent全景：60秒看懂

Hermes Agent at a Glance: 60 Seconds

一张流程图、一组数据、一张对比表。看完这三样，你就知道Hermes是怎么回事了。

架构一张图

Hermes Agent的架构可以用一条线串起来：



从左到右，每个模块的职责一句话讲清楚：

学习循环是Hermes的心脏。每次完成一个任务，它会自动复盘：该记住什么、该提炼什么Skill、现有Skill需不需要优化。这个循环是持续运转的，不需要你手动触发。

三层记忆是Hermes的大脑。会话记忆记住「刚才发生了什么」，持久记忆记住「你是谁、你喜欢什么」，Skill记忆记住「怎么做事」。三层各司其职，用SQLite + FTS5索引存储，按需检索而不是全量加载。

Skill系统是Hermes的技能库。每个Skill是一个独立的markdown文件，存在 `~/.hermes/skills/` 目录下。来源有三种：仓库自带的、Agent自己创建的、从社区Hub安装的。关键特性：**Skill不是静态的，会在使用中自我改进**。

40+内置工具是Hermes的手脚。分五大类：执行（跑代码、操作文件）、信息（搜索、抓取）、媒体（图片、视频）、记忆（读写存储）、协调（子Agent委派）。再加上MCP集成，可以连接6000+外部应用。

多平台Gateway是Hermes的入口。Telegram、Discord、Slack、WhatsApp、Signal、CLI，12+平台支持。你可以在Telegram上给它发消息，它在VPS后台处理，跨平台对话连续。

关键数据

v0.7.0版本，2026年4月3日发布。几个数字值得记一下：

指标	数据
GitHub stars	27,000+（发布两个月）
首月增长	6,000+ stars
内置工具	40+
支持平台	12+
MCP可接入	6,000+ 应用
子Agent并发	最多3个
最低部署成本	\$5/月 VPS
内存占用	<500MB（不跑本地LLM）
许可证	MIT（完全开源）

两个月27000+ stars，这个增速很快。要知道OpenClaw靠龙虾热加持才有今天的体量。Hermes在没有类似社交传播效应的情况下达到这个数字，说明开发者社区确实觉得它解决了一个真实的问题。

和OpenClaw的关键区别

龙虾热之后，大家最关心的问题是：Hermes和OpenClaw到底有什么不一样？

维度	Hermes Agent	OpenClaw
核心理念	自改进学习循环	配置即行为（SOUL.md）
记忆	三层自改进（会话/持久/Skill）	多层记忆（Daily Logs/MEMORY.md/语义搜索），人工维护为主
Skill维护	Agent自动创建+自改进	人工编写和维护
用户建模	Honcho辩证建模（多维身份推断）	基于SOUL.md配置
多平台接入	12+平台Gateway	50+消息平台（Telegram/Discord/WhatsApp/Slack/Signal等）
生态规模	40+内置工具 + MCP 6000+	ClawHub 44000+ 社区Skill
部署方式	自托管（\$5 VPS起）	官方托管/自托管
Skill互通	均采用agentskills.io标准	

差距最大的两个维度是学习能力和用户建模。OpenClaw的Skill主要靠人工编写和调整，它的进化依赖社区和用户的主动维护。Hermes用的越久，Skill越精准，记忆越深，做事越顺手。

但OpenClaw也有Hermes比不了的地方：**生态成熟度**。ClawHub上44000+社区Skill，各种场景都有现成方案。Hermes的社区还在早期阶段。2600万用户的龙虾热给OpenClaw带来了巨大的网络效应，这不是技术能弥补的。

一句话区分：OpenClaw是你养出来的龙虾，Hermes是自己会长大的龙虾。一个靠你用心喂养，一个靠自己从经验中学习。

\$5就能跑起来

成本是很多人关心的。答案可能会让你意外。

Hermes本身免费，MIT开源。你只需要付LLM API的调用费。部署成本取决于你选什么方案：

最省钱：随便一台\$5/月的VPS（Hetzner CX22大概\$4/月，DigitalOcean、Vultr都行），Ubuntu 22.04，不跑本地LLM的话内存占用不到500MB。配合OpenRouter走Claude Haiku或DeepSeek，API费用也很低。

更省钱：Serverless方案。用Daytona或Modal做后端，空闲时环境休眠，收到消息时自动唤醒。会话间成本几乎为零。

注重隐私：在VPS上跑Ollama，本地运行8B或70B的开源模型。API费用为零，但需要更大的VPS（建议16GB以上内存）。

不管哪种方案，**\$5 VPS + Telegram Bot**就能搭出一个24/7在线的个人AI Agent。这个性价比，比订阅制的商业Agent方案划算太多了。

核心建议

对比一下：Claude Code的Pro订阅\$20/月，Max订阅\$200/月。Hermes的\$5 VPS + API费方案，大部分使用场景下月成本在\$10以内。当然两者定位不同，不能直接比价格，但成本门槛确实低很多。

它适合谁

最后一个问题：这本书适合你吗？

满足以下任意一条，就接着看：

你用过Claude Code或OpenClaw，想要一个能**自主跑后台任务**的Agent。不是你坐在旁边盯着的那种，是你睡觉它也在干活的那种。

你对Harness Engineering有了解，好奇**这套方法论被产品化之后是什么样子**。

你想在自己的VPS上部署一个私有AI Agent，**数据不离开自己的服务器**。

你单纯好奇一个两个月拿到27000+ stars的开源项目，到底做对了什么。

接下来的章节，我们从学习循环开始，一层一层拆开Hermes的核心机制。

§03 学习循环：Agent自己给自己造缰绳

The Learning Loop: Self-Harnessing Agent

Hermes Agent最让人意外的不是它能做什么，而是它会变。用得越多，它越好用。这不是营销话术，是一个可观察、可验证的闭环机制。

从一个真实场景说起

假设你第一次让Hermes帮你写一个Python脚本。你说：帮我写一个爬虫，抓取某个网站的标题和摘要。

它会写出一个能用的脚本。但风格可能不是你喜欢的，变量命名可能跟你的习惯不一样，错误处理的方式也未必符合你的预期。挺正常的，毕竟它不认识你。

到了第十次，情况完全不同。它知道你偏好用httpx而不是requests，知道你习惯把错误日志写到文件而不是打印到终端，知道你的项目结构通常是src/目录下按模块分，知道你讨厌过长的函数。

没有人教它这些。它是自己学会的。

这就是学习循环在做的事。

五个环节，一个闭环

Hermes的学习循环有五个环节，单看都不复杂，但串起来形成了一个持续改进的飞轮。



看起来像五个独立的功能？其实它们之间有因果关系。记忆提供了Skill创建的素材，Skill在使用中积累反馈触发自改进，FTS5让历史经验被精准召回，用户建模把这些碎片拼成一幅完整的画。

一个一个拆开看。

环节一：策划记忆

每轮对话结束后，Hermes会主动决定哪些信息值得记住。**是主动决定，不是被动存储。**

传统的对话记忆很粗暴：把整个聊天历史塞进上下文。聊得越多，上下文越长，直到撑不下为止。这就像一个人把所有经历过的事同时放在脑子里想，正常人干不了这事。

Hermes的做法更像人类写日记。每次对话结束，它会回头看一遍：这次聊了什么？有新发现吗？用户表达了什么偏好？然后把值得记住的内容写入SQLite数据库，建好FTS5全文索引。

系统还有一个周期性的nudge机制，定时提醒Agent回顾最近的交互。有点像手机上的日记App弹通知：今天有什么值得记录的？

环节二：自主创建Skill

当Hermes完成了一个相对复杂的任务，它会问自己一个问题：这个解决方案以后还会用到吗？

如果答案是yes，它就把解决方案提炼成一个Skill文件，存到 `~/.hermes/skills/` 目录下。这个Skill是一个markdown文件，包含了任务描述、执行步骤、注意事项。

举个例子：你让Hermes帮你把一个CSV文件清洗并导入数据库。它做完之后，可能会创建一个叫 `csv-to-database.md` 的Skill，记录了数据清洗的常见步骤、你偏好的数据库连接方式、你通常需要的字段验证规则。

下次你再说「帮我导入这个CSV」，它不是从零开始，而是加载这个Skill，按你之前验证过的方式来做。

环节三：Skill自改进

Skill创建出来不是终点。每次使用的过程中，如果你给了反馈，Hermes会拿这些反馈修改Skill本身。

比如你说「这个导入脚本应该先检查表是否存在」，Hermes不只是这次加上这个检查，它会回去修改那个Skill文件，把这条规则写进去。下次再用这个Skill，检查步骤就是默认包含的。

这个过程很像软件开发里的持续改进。每次上线出了问题，修复之后不只打个补丁，还要更新文档和规范，防止同类问题再发生。

关键区别：传统AI工具的记忆是对话记录的堆积，Hermes的记忆是经验的蒸馏。一个是录像带，一个是笔记本。录像带越来越长最终会溢出，笔记本可以一直用下去。

环节四：FTS5跨会话召回

记住了这么多东西，关键是能在对的时候找出来。

Hermes用SQLite的FTS5扩展做全文索引。每次新对话开始前，它会根据当前话题搜索历史记忆，把相关内容加载到上下文中。不是加载全部历史，是按需检索。

这比你想象的重要。大多数AI工具要么不记得上次说了什么，要么把所有历史都塞进来拖慢速度。Hermes的做法是：你问数据库相关的问题，它去翻数据库相关的记忆；你问前端的问题，它去翻前端的记忆。像一个整理得很好的笔记系统，有目录有索引，需要什么查什么。

FTS5还有一个好处：它是纯本地的。你的记忆数据不需要上传到任何服务器，就本地的SQLite文件里。搬家的时候，拷贝 `~/.hermes/` 目录就行。

环节五：用户建模

最后一个环节是Honcho用户建模，这是可选的外部集成。它做的事情比记住你说过什么更进一步：它在推理你是什么样的人。

每次对话结束后，Honcho会分析这次交流，推导出你的偏好、习惯、目标。这些推导不只是记录你说了什么，而是从你的行为模式中归纳出更深层的特征。

比如你从来没有明确说过「我喜欢简洁的代码风格」，但Honcho通过分析你多次修改代码的模式，推断出这个结论。下次生成代码时，它会默认走简洁路线。

关于Honcho的多维身份建模，我们在下一章详细展开。

Mitchell Hashimoto的理念，自动化了

如果你读过Harness Engineering那本橙皮书，可能记得Mitchell Hashimoto的做法。他用Claude Code时有个习惯：每次Agent犯了一个错，他就在CLAUDE.md里加一条规则。

「不要在这个项目里用any类型。」

「测试文件放在__tests__目录下，不要放在src里。」

「commit message用英文，动词开头。」

一条一条加，几周下来，CLAUDE.md变成了一份非常详细的项目规范。Agent从一个什么都不知道的新人，变成了解项目所有暗规则的老手。Mitchell说这感觉就像在训练一个新队员。

Hermes做的事情本质上一样，但它把这个过程自动化了。

你不需要手动写CLAUDE.md，不需要每次犯错后自己总结规则。Hermes自己观察、自己总结、自己写入Skill、自己在下次调用时应用这些规则。

维度	Mitchell的方式（手动）	Hermes的方式（自动）
规则来源	人观察到问题后手写	Agent自己从反馈中提炼
存储位置	CLAUDE.md（单个文件）	多个Skill文件 + 记忆数据库
触发改进	人记得要加规则才会加	每次使用后自动评估
跨项目迁移	需要手动复制CLAUDE.md	Skill全局生效，所有项目共享
改进速度	取决于人的勤快程度	持续自动，不会懒

当然，自动不等于完美。Mitchell手动写的规则往往更精准，因为人类对自己的需求有更清晰的认知。Hermes自动总结的规则可能需要调整，可能会误判。但关键是：它把门槛降到了零。不是每个人都有Mitchell那样的耐心去维护一份精细的规则文件。Hermes让不想折腾配置的人也能享受到越用越好用的体验。

飞轮的加速效应

五个环节单独看都不算新鲜。记忆、Skill、检索、用户画像，AI领域都见过。

Hermes的创新在于把它们组成了一个闭环。记忆喂养Skill，Skill使用中产生新记忆，新记忆触发Skill改进，改进后的Skill产生更好的结果，更好的结果让用户建模更准确，更准确的画像又让下一次记忆策划更有针对性。

这是一个正反馈循环。用得越多，每个环节都在变强，而且是同时变强。就像亚马逊的飞轮效应：更多用户带来更多数据，更多数据带来更好推荐，更好推荐带来更多用户。

区别在于，Hermes的飞轮是为单个用户转的。它不需要百万用户的数据来变好，只需要你自己的使用历史。用上三五天，你就能感受到明显的变化。

核心建议

学习循环的效果和你的使用频率直接相关。如果你一周只用一两次，改进会很慢。但如果你把它当作日常工作伙伴，每天都用，飞轮转得会非常快。这也是为什么Hermes特别适合部署在\$5 VPS上24/7运行，让它持续积累。

这意味着什么

回到标题：Agent自己给自己造缰绳。

在Harness Engineering里，缰绳是人造的。你写CLAUDE.md，配置hooks，设计反馈机制。这些都需要人的持续投入。

Hermes的思路是：**让Agent在奔跑的过程中，自己编织自己的缰绳**。它跑偏了，自己修正路线并记住教训。它找到了好方法，自己沉淀成Skill供下次复用。它遇到了新用户，自己建立理解模型。

这不是说人类不需要参与了。你随时可以手动修改Skill文件，删除不合适的记忆，调整用户画像。但默认状态下，这个系统是自驱动的。

下一章我们来看这个循环中最关键的基础设施：三层记忆系统。如果学习循环是引擎，记忆就是燃料。



§04 三层记忆：从金鱼到老友

Three-Layer Memory: From Goldfish to Old Friend

大多数AI聊天工具的记忆像金鱼，上一轮说的话下一轮就忘了。Hermes的目标是做一个老友：记得你说过什么，知道你是什么样的人，还学会了你做事的方式。

为什么记忆是最难的问题

你可能觉得AI的记忆不就是存聊天记录吗？存下来，下次加载进去就行了。

没那么简单。一个活跃用户每天跟AI聊几千字，一个月就是几万字。全塞进上下文窗口，要么装不下，要么模型因为信息太多反而变迟钝。而且聊天记录里大部分是废话和重复，真正有价值的信息可能只占10%。

好的记忆系统不是存得多，是找得准。

Hermes用三层架构解决这个问题，每一层处理不同类型的记忆。

第一层：会话记忆

会话记忆回答的问题是：**发生了什么？**

每轮对话的内容、工具调用和返回结果，全部写入SQLite数据库，同时建FTS5全文搜索索引。这是情景记忆，类似人类大脑中的海马体。

关键设计决策是按需检索而不是全量加载。新对话开始时，Hermes不会把过去所有对话历史都塞进来。它根据当前话题，用FTS5搜索相关的历史片段，只加载需要的部分。

这个方案的好处：

方式	全量加载	按需检索（Hermes）
上下文占用	随对话增长线性膨胀	基本恒定
检索精度	什么都在，什么都找不到	按关键词精准匹配
长期可用性	几天后就撑不下了	可以用几个月甚至几年
响应速度	越来越慢	基本不变

FTS5是SQLite的全文搜索扩展，不需要额外装数据库。所有数据存在本地SQLite文件里，没有网络依赖，没有隐私顾虑。你的对话记忆不会离开你的机器。

第二层：持久记忆

持久记忆回答的问题是：**你是谁？**

这一层存的不是对话内容，而是从对话中提炼出的持久状态。比如你的编码偏好、项目结构习惯、常用工具链、工作时间规律。这些跨会话保持，不会因为开了新对话就丢失。

技术实现上，持久记忆也存在SQLite里，由memory工具管理。整个方案纯文件级：没有外部服务器，没有云端同步，数据就在 `~/.hermes/` 目录下。

这意味着你可以：

- 把 `~/.hermes/` 备份到U盘，换台电脑继续用
- 用Docker部署时，挂载 `/opt/data` 目录到宿主机保持状态
- 多设备之间用网盘同步（虽然不推荐实时同步SQLite，但定期拷贝没问题）

便携性是一个被低估的特性。很多AI工具的记忆锁在云端，你换个工具就得从零开始。Hermes的记忆是你自己的文件，想怎么搬怎么搬。

第三层：Skill记忆

Skill记忆回答的问题是：**怎么做事？**

前两层记住了发生过什么、你是什么样的人。第三层记住的是方法论和操作规范。每个Skill是 `~/.hermes/skills/` 下的一个markdown文件，可读可编辑。

这三层对应了认知科学里的三种记忆类型：



人类学骑自行车的过程就是这三层记忆的协作：你记得上次摔了（情景），你知道重心要放低（语义），你的身体会自动保持平衡（程序性）。Hermes处理任务的过程类似：它记得上次你怎么改的代码，知道你的偏好是什么，手头有一套验证过的执行方案。

三层协作的例子：你说「帮我部署这个项目」。Hermes先用FTS5搜索会话记忆，找到你上次部署时遇到的端口冲突问题（情景）。再查持久记忆，知道你用的是阿里云ECS、Nginx反向代理（语义）。最后加载deployment-checklist这个Skill，按你验证过的步骤执行（程序性）。三层各司其职。

Honcho：比你更了解你自己

三层本地记忆之外，Hermes还有一个可选的外挂：Honcho用户建模系统，由Plastic Labs开发。

Honcho做的事情比记住你说了什么更深一步。它在推理你这个人的特征。官方叫辩证用户建模，覆盖多个身份维度。

具体什么意思？用一个场景来说明。

假设你连续三周每天都在让Hermes帮你写Python脚本。Honcho在这个过程中可能推断出：

- **技术水平：**你不是纯新手，但也不是专家。你能看懂代码但不太写得出来
- **工作节奏：**你通常在晚上9-11点活跃，可能是下班后的个人项目

- **沟通风格**：你喜欢先看结果再问原理，不喜欢长篇大论的解释
- **目标推断**：你可能在准备一个数据分析项目，因为最近的任务都围绕数据处理
- **情绪模式**：你在代码报错时会有点急躁，这时候简洁直接的回答效果更好
- **偏好矛盾**：你口头上说要写完整注释，但实际review代码时从不看注释

注意最后一条。**Honcho能捕捉到你言行不一致的地方**。你嘴上说的偏好和实际行为展现的偏好可能不同，辩证建模就是同时关注这两者。

这些推断会注入到后续对话的prompt里，作为隐形上下文。你看不到这些信息，但会感受到Hermes变得更懂你了。

和Claude Code auto-memory的对比

Claude Code也有记忆系统：CLAUDE.md文件和auto-memory。它们和Hermes的记忆有什么区别？

维度	Claude Code	Hermes Agent
记忆格式	CLAUDE.md + auto-memory文本文件	SQLite数据库 + FTS5索引 + Skill文件
写入方式	CLAUDE.md手动写，auto-memory半自动	全自动写入，人可以随时覆盖
检索方式	启动时全量加载CLAUDE.md	按需FTS5全文检索
记忆粒度	项目级（每个项目一份CLAUDE.md）	全局级 + 项目级都有
用户建模	无（需要用户自己写偏好）	Honcho自动推理用户画像
程序性记忆	CLAUDE.md中的指令	独立Skill文件，可自改进
跨项目共享	~/ .claude/CLAUDE.md（全局指令文件）	所有记忆天然全局
存储上限	CLAUDE.md建议控制在几KB	SQLite理论上限很高

两者的设计哲学不同。Claude Code的CLAUDE.md是人编写、AI执行的模式，好处是人有完全的控制权，坏处是需要持续投入维护。Hermes是AI自写、人审核的模式，好处是门槛低、自动化程度高，坏处是自动生成的内容不一定都准确。

哪种更好取决于使用场景。如果你是重度Claude Code用户，已经花了几周精心打磨CLAUDE.md，那你手工编织的缰绳可能比Hermes自动生成的更精确。但如果你不想花时间维护配置文件，Hermes的全自动方案确实省心不少。

记忆不是万能的

说了这么多好处，也得聊聊记忆的局限。

什么该记：

- 用户偏好和习惯（代码风格、工具选择、沟通方式）
- 项目上下文（架构决策、技术栈、文件结构）
- 验证过的解决方案（Skill）
- 重复出现的模式（比如某类错误的处理方式）

什么不该记：

- 一次性任务的细节（帮我写个生日贺词——这不需要记住）
- 过时的信息（三个月前的API版本号，现在可能已经变了）
- 错误的推断（Hermes可能误判你的偏好，这些应该被清理）
- 敏感信息（密码、密钥、个人身份信息不应该进记忆库）

注意

Hermes的记忆系统目前没有自动过期机制。如果你长期使用，记忆数据库会持续增长。建议定期检查 `~/hermes/` 目录的大小，清理过时的Skill文件。这是一个已知的待改进点。

还有一个问题是记忆污染。如果Hermes在早期对话中记住了错误信息，这个错误可能持续影响后续行为。比如它错误地记住了你偏好Python 2，之后生成的代码可能都带着Python 2的语法。

所以定期审查记忆很有必要。看看 `~/hermes/skills/` 下有哪些Skill，删掉不合适的。查看持久记忆，修正错误的推断。就像整理笔记本，偶尔翻翻会发现不少需要更新的内容。

从存储到理解

回到标题：从金鱼到老友。

金鱼的问题不是没有眼睛，是没有记忆。它每次看到你都像第一次见面。大多数AI工具就是这样，每次打开新对话，一切从零开始。

老友不一样。老友知道你的脾气、你的习惯，知道你嘴上说不在意其实很在意。老友不需要你每次都解释背景，因为背景已经在那了。

Hermes的三层记忆加上Honcho用户建模，走的就是这条路。会话记忆提供素材，持久记忆提供画像，Skill记忆提供方法论，Honcho把这些碎片拼成对你的完整理解。

用的时间越长，理解越深。这不是口号，是三层记忆加闭环学习的数学必然。

下一章我们看看Hermes的另一个核心机制：Skill系统。如果记忆是「知道」，Skill就是「会做」。

§05 Skill系统：会自我进化的能力

Skills: Self-Evolving Capabilities

OpenClaw的Skill需要你手动写、手动维护。Hermes的Skill会自己长出来，还会自己变好。这个差异，决定了两者完全不同的使用体验。

Skill到底是什么

在Hermes里，每个Skill是一个独立的markdown文件，存在 `~/hermes/skills/` 目录下。它记录的是Agent怎么做某件事的程序性记忆。

打个比方：你教一个新同事做周报。第一次你得一步步带，第二次他可能还问几个问题，第三次他就自己搞定了。**Skill**就是那个「第三次之后」的状态，Agent把方法固化成一份可复用的文档。

Skill有三种来源：

来源	说明	数量级
Bundled Skills	安装时自带的预置能力，覆盖MLOps、GitHub工作流、研究等常见场景	40+
Agent自主创建	完成复杂任务后，Agent自动将解决方案提炼为Skill	按使用积累
Skills Hub	社区贡献的技能包，可一键安装	持续增长

三种来源不是并列关系。Bundled Skills是起点，Skills Hub是加速器，**Agent自主创建才是Hermes真正的杀手锏。**

agentskills.io：Skill的通用语言

Hermes的Skill不是封闭生态。它采用了agentskills.io标准，目前已经有30+个工具支持，包括Claude Code、Cursor、Copilot、Gemini CLI。

你为Claude Code写的Skill，可以直接在Hermes里用。反过来也一样。

这和App Store的逻辑不同。App Store是每个平台一套生态，开发者要适配多端。agentskills.io更像USB接口，一个Skill插到哪都能跑。

对于已经在用Claude Code的人来说，你积累的Skill资产不会被锁死在某一个工具里。切换到Hermes，或者同时用两个，Skill都可以无缝迁移。

自改进：Skill会越用越好

这是Hermes和所有其他Agent Skill系统最大的区别。

传统的Skill需要人工维护。你写了一个代码审查Skill，它会按你写的步骤执行。实际使用中发现某个步骤效果不好？你得手动去改。OpenClaw的44000+社区Skill主要以这种方式运作。

Hermes的Skill是活的。它跑在学习循环里，根据实际反馈自动优化。

具体机制：

- 1

执行Skill

Agent按照Skill中记录的步骤完成任务
- 2

收集反馈

用户的反应（满意/不满意/修正）被记录到会话记忆
- 3

更新Skill

Agent分析反馈，自动修改Skill文件中的相关步骤
- 4

下次执行时使用新版本

改进后的Skill在后续任务中自动生效

听起来很理想化？确实，效果取决于LLM的能力和反馈的质量。但方向是对的：**让Agent从经验中学习，不是等人来维护。**

和Mitchell Hashimoto的对比： Mitchell说他用Claude Code时，「每次犯错就加一条规则到CLAUDE.md」。Hermes把这个过程自动化了。你不用亲自加规则，Agent自己观察、总结、写进Skill。代价是你对规则的控制力会降低一些。

和OpenClaw Skill的关键区别

OpenClaw的ClawHub有44000+社区Skill，数量上远超Hermes。但两者的设计哲学完全不同：

维度	OpenClaw Skill	Hermes Skill
创建方式	人工编写SOUL.md	Agent自主创建+人工编写
维护方式	人工更新	自动进化+人工干预
个性化	通用模板，用户fork后自定义	从你的使用习惯中自然生长
互通性	agentskills.io标准	agentskills.io标准（互通）
生态规模	44000+（大）	40+预置+社区（成长中）

OpenClaw的优势在规模和透明度。看一眼SOUL.md就知道这个Skill干了什么，每一步都是你写的，心里有数。

Hermes的优势在适应性。同一个写代码的Skill，Python开发者和Rust开发者用三周后，会演化出两个截然不同的版本。不是通用模板，是量身定制。

两者不是替代关系。agentskills.io标准让Skill可以互通。你完全可以从ClawHub装一个Skill到Hermes里，然后让Hermes在使用中持续改进它。

实战：让Hermes自己创建一个Skill

说了这么多机制，来看一个具体过程。

假设你每天早上需要Hermes帮你整理昨天的GitHub通知，总结重要的PR和Issue。前几次你每次都得说一遍需求：

核心建议

「帮我看看昨天的GitHub通知，按重要程度排序，PR和Issue分开，忽略bot的自动通知。」

第三次或第四次之后，Hermes会在后台做一件事：把这个反复出现的任务模式提炼成一个Skill文件。你可以在 `~/.hermes/skills/` 目录下看到一个新的markdown文件，内容大概是这样的：

```
# GitHub Daily Digest

## 触发条件
用户提到"GitHub通知"、"每日总结"等关键词

## 执行步骤
1. 调用GitHub MCP获取过去24小时的通知
2. 过滤掉bot账号的自动通知
3. 按类型分组 (PR / Issue / Discussion)
4. 按重要程度排序 (mention > review request > 其他)
5. 以简洁列表形式呈现

## 用户偏好
- 不需要详细内容，只要标题和状态
- PR和Issue分开显示
```

从这一刻起，你只需要说一句「看看GitHub」，Hermes就知道该做什么。

更有意思的是后续。某天你说「这次把Discussion也加上」，Hermes不只是这次加上，它会更新Skill文件里的规则。**下次你不说，它也会带上Discussion。**

这就是「自我进化」的实际含义。没有什么神秘的AI突破，就是把「用户纠正 -> 规则更新 -> 下次执行」自动化了的闭环。

注意

Skill自改进的前提是你的反馈足够清晰。如果你只是觉得「不太对」但不说具体哪里不对，Agent很难准确改进。好的反馈 = 好的进化方向。

§06 40+工具与MCP：连接一切

40+ Tools & MCP: Connect Everything

Agent再聪明，没有工具也干不了实事。Hermes内置40+工具，从跑代码到发消息全覆盖。MCP再把触角伸到6000+外部应用。

五大类工具速览

Hermes的工具按功能分成五大类。不用全记住，知道有这些能力就行，用到时再查。

类别	核心工具	干什么用
执行类	terminal、code_execution、file	跑命令、执行代码（沙箱隔离）、读写文件
信息类	web、browser、session_search	搜索网页、浏览器自动化、检索历史对话
媒体类	vision、image_gen、tts	理解图片、生成图片、文字转语音
记忆类	memory、skills、todo、cronjob	操作记忆层、管理Skill、任务规划、定时调度
协调类	delegation、moa、clarify	委派子Agent、多模型推理、向用户请求澄清

几个值得单独拎出来说的：

session_search 是Hermes比较独特的能力。它用FTS5全文索引搜索历史对话，配合LLM摘要，让Agent能快速回忆「上周我们讨论过的那个方案」。大多数Agent没这个能力，每次对话都从零开始。

moa (Multi-model Orchestrated Answering) 让Hermes同时调用多个LLM，综合多个模型的回答给出最终结果。需要高可靠性的场景比较有用，比如事实核查或技术判断。

cronjob 用自然语言定义定时任务。你说「每天早上9点帮我看GitHub通知」，它就创建一个定时触发。不用写cron表达式，不用配调度器。

Toolset：不是全开，是按需启用

40+个工具全开不合理。帮你写代码的Agent不需要Home Assistant权限，管日程的Agent也用不到code_execution。

Hermes用Toolset机制解决这个问题。工具按功能分组，在config.yaml里按需启用或禁用：

```
# config.yaml 示例
toolsets:
  - web          # 网页搜索
  - terminal     # 终端命令
  - file         # 文件操作
  - skills       # Skill管理
  - delegation   # 子Agent委派
# - homeassistant # 不需要就注释掉
# - rl           # 强化学习相关，大多数人用不到
```

这不只是功能开关。启用的工具越少，Agent注意力越集中，响应越快，token消耗也越少。如果你只需要一个帮你整理文件的助手，开file和memory两个Toolset就够了。

Toolset也是安全边界。§ 03提到的约束层，在工具层面就是通过Toolset实现的。你可以精确控制Agent能碰什么、不能碰什么。

MCP：6000+应用的统一接口

40+内置工具覆盖了通用场景。但每个人的工具栈不一样，你可能用Jira管项目、用Notion写文档、用Slack沟通。怎么让Hermes接入这些？

MCP（Model Context Protocol）。

MCP是一个开放协议，定义了AI Agent和外部工具之间的通信标准。Hermes支持通过stdio或HTTP两种方式连接任意MCP Server。目前MCP生态覆盖了6000+应用，GitHub、Slack、Jira、Google Drive、数据库都有。

接入很简单，在config.yaml里加一段配置：

```
# 接入GitHub MCP Server
mcp_servers:
  - name: github
    transport: stdio
    command: npx @modelcontextprotocol/server-github
    env:
      GITHUB_TOKEN: ${GITHUB_TOKEN}
```

配好之后，Hermes就能用GitHub的能力了：创建Issue、审查PR、查看仓库状态。不用写代码，不用开发自定义工具，MCP Server就是即插即用的能力扩展。

Hermes还支持per-server工具过滤。一个MCP Server可能暴露了20个工具，但你可能只想让Agent用其中3个。工具过滤让你精确控制暴露给Agent的能力范围。

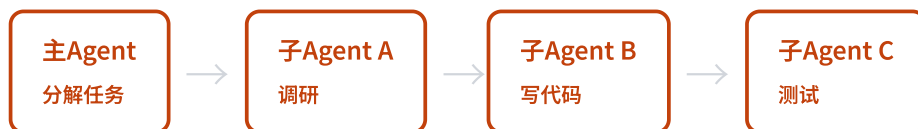
子Agent委派：让三匹马同时跑

delegation是Hermes最强大的协调工具。它能生成子Agent实例，把任务分发出去并行执行。

独立上下文。每个子Agent有自己的对话上下文，互不干扰。主Agent分配任务时传入必要的背景信息，子Agent在独立空间里工作。

受限工具集。你可以指定每个子Agent能用哪些工具。负责调研的只需要web和browser，写代码的只需要terminal和file。既是效率优化，也是安全措施。

最多3个并发。这个限制是刻意的。3个并发子Agent已经覆盖大多数并行场景（同时调研、写代码、跑测试），再多就难以有效协调了。



用过Claude Code多实例并行的人会觉得这很熟悉。区别在于Claude Code的多实例是你手动开的，实例之间没有协调关系。Hermes的子Agent委派是Agent自主决定何时分发、怎么汇总结果。

实际体感：子Agent最适合的场景是「做几件不相关的事然后汇总」。比如你让Hermes写一篇技术博客，它可能同时派一个子Agent调研最新资讯、一个子Agent分析竞品文章、一个子Agent整理代码示例，最后主Agent把三方结果整合成初稿。

工具权限与沙箱：约束层的落地

前面几章讲了学习循环、记忆、Skill，都是让Agent变强的机制。但Agent越强，约束越重要。

Hermes在工具层面做了三重约束：

Toolset控制。只有config.yaml里启用的工具才能被Agent调用，最粗粒度的开关。

code_execution沙箱。代码在隔离环境中运行，和你的系统环境分开。即使Agent执行了有问题的代码，也不会影响你的文件系统。

子Agent受限工具集。委派子Agent时可以指定它能用的工具子集。负责搜索的子Agent不需要也不应该有terminal权限。

如果你读过Harness Engineering橙皮书，会发现这就是**约束层（hooks/linter）在Hermes中的具体实现**。理念一样：给Agent足够的能力完成任务，但不给多余的权限。

核心建议

对于安全敏感的场景（比如在生产服务器上运行），建议只启用必要的Toolset，并通过MCP的per-server过滤进一步限制暴露给Agent的工具。宁可Agent来问你「我需要XX权限」，也不要默认全开。

三重约束加在一起，构成了一个务实的安全模型。不追求理论上的完美隔离，而是在实用性和安全性之间找平衡。你给Agent越多信任，它能做的事越多。但即使完全不改默认配置，Hermes也不会在你不知情的情况下做危险操作。

§07 安装与配置：三种方式

Installation & Configuration

从零到能跑，最快5分钟。这一节覆盖三种安装方式，从本地开发到24/7服务器，选适合你的那个。

方式一：本地安装（5分钟上手）

本地安装最直接，适合先体验再决定要不要长期跑的人。唯一前提是你的机器上有git。

1 运行一键安装脚本

打开终端，粘贴这一行：

```
curl -fsSL https://raw.githubusercontent.com/NousResearch/hermes-agent/main/scripts/install.sh | bash
```

安装器会自动处理Python、Node.js和所有依赖。macOS、Linux、WSL2都能跑。

2 配置LLM后端

安装完成后，编辑配置文件：

```
# 配置文件位置
~/hermes/config.yaml
```

填入你的模型API Key（后面详细说怎么选模型）。

3 启动Hermes

```
hermes
```

没错，就一个词。看到欢迎消息就说明安装成功了。

核心建议

如果你用uv管理Python，也可以通过 `uv pip install hermes-agent` 安装。效果一样，选你习惯的方式。

方式二：Docker（隔离干净）

不想在本机装一堆依赖？Docker是最干净的选择。

```
docker pull nousresearch/hermes-agent:latest
docker run -v ~/.hermes:/opt/data nousresearch/hermes-agent:latest
```

关键参数： `-v ~/.hermes:/opt/data` 把容器内的数据卷映射到宿主机。Hermes的所有状态（记忆、Skill、配置）都存在 `/opt/data` 这一个目录里。容器删了重建，数据不丢。

这个设计挺好。不像有些工具状态散落在各种路径里，Hermes的一切都在 `~/.hermes/` 下面，迁移的时候打包这个目录就行。

方式三：\$5 VPS 24/7运行

如果你想让Hermes随时在线，不依赖你的电脑开着，一台\$5/月的VPS就够了。

推荐配置：

VPS提供商	月费	说明
Hetzner CX22	~\$4/月	性价比最高，欧洲节点
DigitalOcean Droplet	\$5/月	新加坡/美西节点
Vultr	\$5/月	东京节点延迟低

选Ubuntu 22.04 LTS，SSH登录后跑安装脚本，和本地安装一模一样。不跑本地模型的话，内存占用不到500MB，\$5的机器绰绰有余。

配合Telegram Gateway（\$ 09会讲），手机上随时给Hermes发消息，它就在VPS上响应。一杯咖啡的钱，换一个24小时在线的AI助手。

Serverless方案： Hermes还支持Daytona和Modal两种serverless后端。空闲时环境休眠，收到消息时自动唤醒，会话间成本趋近于零。适合用量不大但又想保持可达的场景。在config.yaml里设置 `terminal: daytona` 或 `terminal: modal` 即可。

config.yaml 详解

不管哪种安装方式，核心配置都在一个文件里：`~/.hermes/config.yaml`。

一个最小可用的配置长这样：

```
# ~/.hermes/config.yaml
model:
  provider: openrouter          # 模型提供商
  api_key: sk-or-xxxxx         # 你的API Key
  model: anthropic/claude-sonnet-4 # 使用的模型

terminal: local                # 终端后端 (local/docker/ssh/daytona/modal)

gateway:                       # 消息网关 (可选, §09详细讲)
  telegram:
    token: YOUR_BOT_TOKEN
  discord:
    token: YOUR_BOT_TOKEN
```

配置项不多，逐个过一遍。

provider 和 model

Hermes支持的模型来源相当广：

Provider	推荐模型	适用场景
OpenRouter	Claude Sonnet 4 / GPT-4o	200+模型可选，灵活切换
Nous Portal	Hermes 3系列	官方推荐，和Agent深度适配
OpenAI	GPT-4o / o3	直连OpenAI API
z.ai / 智谱	GLM-5	国内用户友好
Ollama	Hermes 3 8B/70B	完全离线，隐私优先

注意

注意：2026年4月起，Anthropic封禁了第三方工具通过Claude订阅（Pro/Max）访问Claude。Hermes、OpenClaw等Agent框架都受影响。你仍然可以通过API key按量付费使用Claude，但成本会高得多。建议优先考虑OpenRouter或Nous Portal的Hermes 3系列。

我的建议：起步阶段用OpenRouter，可以随时切模型试手感。等确定了常用模型，再直连对应API省中间费用。

terminal

六种后端，决定Hermes在哪里执行代码：

- **local**: 直接在本机运行, 最简单
- **docker**: 容器内运行, 隔离安全
- **ssh**: 连接远程服务器
- **daytona / modal**: serverless, 按需唤醒
- **singularity**: HPC集群环境

大多数人选local就行。如果你担心Agent在本机执行代码的安全性, Docker是个好折中。

常见问题排查

注意

安装脚本卡住？ 检查网络连接。脚本需要从GitHub和PyPI下载依赖。如果在国内，可能需要设置代理或使用镜像源。

注意

hermes命令找不到？ 安装脚本会把命令添加到PATH，但如果你的shell配置比较特殊（比如用了fish），可能需要手动 `source ~/.bashrc` 或重新打开终端。

核心建议

想确认安装是否成功？ 运行 `hermes --version`，看到版本号就对了。当前最新版是v0.7.0。

注意

Docker容器启动后没反应？ 确认 `~/.hermes/config.yaml` 已经存在且配置了model信息。容器会读取宿主机映射进来的配置。如果config.yaml不存在，Hermes会在启动时引导你创建。

核心建议

VPS上的安全提醒： 如果在VPS上跑，建议设置 `terminal: docker` 让代码在容器内执行，避免Agent直接操作宿主机文件系统。\$5的VPS性能足够跑容器。

配置就这些。Hermes的设计思路是能省则省，一个`config.yaml`搞定所有事，不搞分散的环境变量和多层配置文件。功能这么丰富的Agent，配置做到这个程度，挺克制的。

下一节直接开聊。Hermes不需要你配完所有东西才能用，装好、填上API Key、启动，就可以开始第一次对话了。

§08 第一次对话：让Hermes认识你

First Conversation

装好了，打开了，然后呢？这一节带你走完第一次对话。重点不是你说了什么，而是Hermes在背后做了什么。

从空白开始

终端输入 `hermes`，你会看到一个简洁的对话界面。没有引导流程，没有设置向导，就是一个等你说话的光标。

随便说点什么：

你好，我是花叔，做AI自媒体的。我用Claude Code比较多，最近想试试Hermes。

Hermes会正常回复你。但有意思的事情发生在你看不到的地方。

记忆在悄悄写入

第一轮对话结束后，去看看 `~/hermes/` 目录：

```
~/hermes/  
├── config.yaml # 你的配置  
├── state.db # SQLite数据库（对话历史+FTS5索引）  
├── skills/ # Skill目录（还是空的）  
│   └── bundled/ # 内置Skill  
└── memories/ # 持久记忆
```

state.db已经有内容了。你刚才那句自我介绍，连同Hermes的回复，都写进了SQLite数据库，并且建了FTS5全文索引。下次启动Hermes，它不会从零开始，而是能搜索找到这段对话。

这和ChatGPT那种「看起来有记忆但其实每次都重新加载全部历史」不同。**Hermes是按需检索**，只在相关时才调出历史。数据库积累几个月的对话也不会变慢。

继续聊，触发更深的记忆

再多聊几轮。比如告诉它你的工作习惯：

我平时用macOS，主力编辑器是Cursor。写文章用Markdown，习惯用「」引号不用""。

Hermes会把这些信息写入持久记忆层。对应§04讲的三层记忆：你刚才的对话内容是会话记忆（发生了什么），偏好习惯是持久记忆（你是谁）。

你不需要用特殊命令告诉它「记住这个」。Hermes会自己判断哪些信息值得持久化。用过Claude Code的auto-memory的话，这个体验很熟悉。但Hermes更激进一些，它会主动策划该记什么。

触发第一次Skill创建

试着给Hermes一个稍微复杂点的任务：

帮我把这段Markdown转成微信公众号兼容的HTML，保留加粗和代码块的样式。

第一次做，Hermes会摸索着完成。可能调用终端执行脚本，也可能直接在对话里生成HTML。

完成之后，有意思的事情来了。再看 `~/.hermes/skills/` 目录：

```
~/.hermes/skills/
├── bundled/ # 内置Skill
└── markdown-to-wechat.md # 新出现的!
```

Hermes自动把刚才的解决方案提炼成了一个Skill。打开这个markdown文件，你会看到它记录了输入格式、转换规则和输出要求。下次你再让它做类似的事，它会直接调用这个Skill，不用重新摸索。

这就是 § 03讲的学习循环在起作用。不是你教它怎么做，是它自己把做过的事沉淀成能力。

Skill自改进：如果你对结果不满意，告诉它哪里需要调整，Hermes不仅会修正当次输出，还会更新那个Skill文件。下次执行时自动用改进后的版本。

你不需要配置，只需要用

回顾一下刚才发生了什么：



整个过程你没写一行配置、没编辑一个文件、没设置任何规则。这和Claude Code需要你手写CLAUDE.md、OpenClaw需要你配置yaml的体验完全不同。

当然，你完全可以手动编辑Skill文件（§ 10会讲），但起步阶段真不需要。用就对了，Hermes会自己长出适合你的形状。

这也是Hermes最独特的地方。其他Agent工具需要你先想清楚要什么、怎么配置、怎么约束。Hermes反过来：**你先用，它在使用过程中自己形成结构。**

下一节，让Hermes走出终端，出现在你的手机上。

§09 多平台接入：在哪都能找到它

Multi-Platform Access

Hermes不只活在终端里。配个Telegram Bot，手机上随时找它。加上Discord和Slack，团队也能用。关键是：所有平台共享同一个大脑。

Gateway：一个进程管所有平台

Hermes的多平台接入靠Messaging Gateway模块。不是给每个平台写一套代码，而是一个统一的网关进程，同时监听所有已配置的平台。

架构是这样的：



Gateway下面连着同一个Hermes Agent实例、同一套记忆、同一组Skill。从Telegram发的消息和从CLI输入的命令，对Hermes来说没有区别。

Telegram Bot配置（最推荐的入口）

为什么推荐Telegram？配置最简单、体验最好的移动端入口。创建Bot不需要审核，Token即开即用。

创建Telegram Bot

在Telegram里找 @BotFather ，发送 /newbot ，按提示取个名字。BotFather会给你一个Token，长得像这样：

```
123456789:ABCdefGhIJKLmNoPQRsTUVwxyz
```

写入config.yaml

```
# ~/.hermes/config.yaml
gateway:
  telegram:
    token: 123456789:ABCdefGhIJKLmNoPQRsTUVwxyz
```

启动Hermes

```
hermes
```

Hermes启动时会自动连接Telegram Gateway。在Telegram里给你的Bot发消息，它就会回复。

三步，总共不到两分钟。如果Hermes跑在VPS上（\$ 07讲的\$5方案），这就是一个**24小时在线、随时可达、有持久记忆**的私人AI助手。月费\$5加模型调用费。

核心建议

Telegram还支持语音消息。直接发语音备忘录，Hermes会自动转写成文字再处理。通勤路上想到什么就说，不用打字。

Discord和Slack接入

流程和Telegram类似，区别在于Token的获取方式。

Discord

去Discord Developer Portal创建Application，在Bot页面获取Token：

```
gateway:
  discord:
    token: YOUR_DISCORD_BOT_TOKEN
```

把Bot邀请到你的Server，它就能在频道里响应了。**适合团队协作场景**，比如在开发频道里让Hermes帮忙review代码，或者在运营频道里让它汇总数据。

Slack

```
gateway:
  slack:
    token: xoxb-YOUR-SLACK-BOT-TOKEN
```

需要在Slack App管理页面创建App并安装到Workspace。权限设置比Telegram复杂一些，但**企业内部使用更正规**，IT部门看着也放心。

还有更多平台：Hermes一级支持WhatsApp和Signal，社区还扩展了钉钉、飞书、企业微信、Home Assistant等。完整列表在官方文档的Messaging页面。大多数只需要在config.yaml里加对应Token。

跨平台对话连续性

这是Hermes多平台设计里最实用的特性。

假设你早上在通勤路上用Telegram给Hermes说：

帮我调研一下Hermes Agent的部署方案，整理成文档。

Hermes开始工作，把调研结果存进记忆。到了办公室，你打开终端继续：

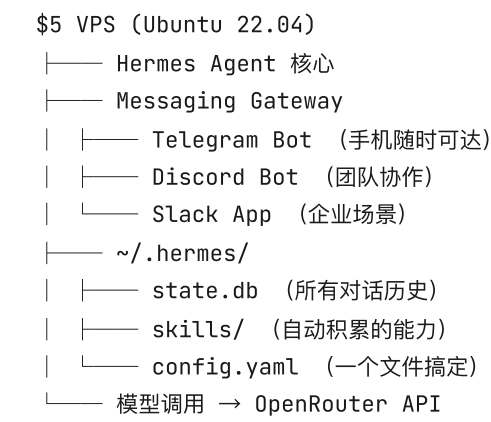
刚才那个调研做得怎么样了？把结果给我看看。

Hermes知道你在说什么。它**不区分消息来自哪个平台**，所有平台共享同一个Agent实例和同一套记忆。Telegram上说的话，CLI里能接着聊。Discord频道里讨论的内容，Slack里也能引用。

这和你在不同设备上开不同的ChatGPT窗口、每次都得重新解释背景的体验完全不同。**Hermes只有一个大脑**，不管你从哪个入口进来。

实际部署方案

把前面几节串起来，一个典型的部署方案长这样：



整套成本：VPS \$5/月 + 模型调用费（轻度使用每月\$2-5）。总共不到一杯精品咖啡的价格，换一个有记忆、有能力、24小时在线的AI助手。

核心建议

关于模型调用费：如果你想进一步省钱，可以用Ollama在VPS上跑开源模型（比如Hermes 3 8B）。\$5的VPS内存可能不够跑大模型，但\$10-15/月的VPS就能跑8B了，之后模型调用完全免费。

自动化调度

除了被动响应消息，Gateway还支持cron调度。你可以让Hermes每天早上8点自动发一份新闻简报，或者每周五下午自动汇总本周的GitHub提交。

定时任务的结果会通过Gateway推送到你指定的平台。它不只是等你说话才动，它可以主动干活。

到这里，你已经有了一个能用的Hermes：本地或VPS安装、配好模型、手机上随时可达。接下来讲怎么手动创建和优化Skill（\$ 10），以及怎么通过MCP接入更多工具（\$ 11），让这个助手更强。

§10 自定义Skill：教Hermes新技能

Custom Skills: Teaching Hermes New Tricks

Hermes的学习循环会自动创建Skill，但你也可以手动教它。这一节讲怎么写Skill、从社区装Skill、以及把Claude Code的Skill搬过来。

Skill到底是什么

在Hermes里，一个Skill就是一个markdown文件。没有什么框架要学，没有API要调，就是一段文字，告诉Hermes在特定场景下应该怎么做。

思路和CLAUDE.md一样：用自然语言定义行为。区别在于CLAUDE.md全局生效，Skill按需激活。你让Hermes帮你写周报，它会自动加载「写周报」Skill；你让它做代码审查，它加载另一个。

Skill文件存在 `~/.hermes/skills/` 目录下，每个Skill一个 `.md` 文件。

手动创建一个Skill

我们来写一个实际有用的Skill：让Hermes帮你做Git提交信息的规范化。

在 `~/.hermes/skills/` 下创建一个文件 `git-commit-style.md`：


```
# Git Commit Style
```

```
## 触发条件
```

当用户让我提交代码、写commit message、或review提交历史时激活。

```
## 行为规则
```

```
### Commit Message格式
```

- 第一行：类型(范围)：简述（不超过50字符）
- 空一行
- 正文：解释为什么改，而非改了什麼

```
### 类型枚举
```

- feat：新功能
- fix：修复bug
- refactor：重构（不改功能）
- docs：文档
- test：测试
- chore：构建/工具链

```
### 约束
```

- 用中文写正文，类型用英文
- 不要写「修改了XX文件」这种废话
- 一个commit只做一件事

```
## 示例
```

```
、、、
```

```
feat(auth)：支持微信扫码登录
```

之前只能用手机号登录，微信用户需要先绑定手机号再登录，流程太长。

现在直接扫码就能登录，未绑定手机号的用户会自动创建账号。

```
、、、
```

保存之后，下次你对Hermes说「帮我提交这些改动」，它就会按这个格式来。不需要额外设置。

核心建议

Skill的触发是自动的。你不需要手动说「用XX Skill」，Hermes会根据你的请求自动匹配最相关的Skill。背后靠的是FTS5全文检索加语义理解。

一个好Skill的结构

写了几十个Skill之后，我总结出好Skill的共性结构：

部分	作用	必须?
标题	让Hermes快速识别这个Skill的用途	是
触发条件	什么时候激活这个Skill	强烈建议
行为规则	具体怎么做，步骤、约束、格式	是
示例	一个完整的输入→输出示例	强烈建议
不要做什么	明确的边界，防止漂移	可选

触发条件写得越具体，Skill的命中率越高。「当用户提到代码」太模糊了；「当用户让我提交代码、写commit message、或review提交历史时」就很好。

从Skills Hub安装社区Skill

Hermes自带Skills Hub，社区开发者贡献了各种现成的Skill。安装流程很简单：

浏览可用的Skill

在对话中直接问Hermes：「有什么可用的社区Skill？」它会列出Hub里的热门Skill，按类别分组。你也可以指定方向，比如「有没有Python开发相关的Skill？」

安装

找到想要的Skill后，告诉Hermes「安装XX Skill」。它会把对应的markdown文件下载到 `~/.hermes/skills/`，然后立刻生效。不需要重启。

定制

安装完的Skill就是普通markdown文件，你可以直接打开编辑。社区Skill是起点，改成适合自己的才是终点。

Hermes的bundled skills有40多个，覆盖MLOps、GitHub工作流、研究助理等场景。出厂自带，不需要额外安装。

Skill调试技巧

写好Skill之后，怎么知道它有没有被正确触发？

直接问。对Hermes说「你现在加载了哪些Skill？」它会告诉你当前激活的Skill列表。如果你期望的Skill没出现，多半是触发条件写得太窄。

看日志。日志文件会记录每次请求匹配了哪些Skill、为什么选中、为什么跳过。日志在 `~/.hermes/logs/` 目录下。

渐进式测试。别急着用复杂场景测。先用最简单的请求确认触发和基本行为，再试边界情况。

注意

Skill之间可能冲突。如果两个Skill的触发条件有重叠，Hermes会优先选择匹配度更高的那个，但结果不一定符合你的预期。遇到行为异常，先检查是不是Skill冲突。

实战：把Claude Code Skill移植到Hermes

很实际的需求：你在Claude Code里积累了一堆好用的Skill，切到Hermes之后不想从零开始。**agentskills.io**标准让Skill在不同Agent之间互通，所以迁移成本很低。

拿一个真实的例子来演示。我在Claude Code里有个「公众号审校」Skill，核心逻辑是三遍审校（事实、风格、细节）。SKILL.md大概长这样：

```
# 公众号文章审校

## 触发条件
当用户提到「审校」「降低AI味」「太AI了」「润色」时激活。

## 审校流程
### 第一遍：事实审校
- 检查所有数据、时间、产品名是否准确
- 标注不确定的信息

### 第二遍：风格审校
- 删除AI高频词（首先/其次/综上所述）
- 拆解AI句式
- 替换书面词汇为口语

### 第三遍：细节打磨
- 句子控制在15-25字
- 段落控制在手机屏幕3-5行
- 约10处加粗标记重点
```

把这个文件复制到 `~/.hermes/skills/proofreading.md`，Hermes就能直接用。不需要改格式，不需要适配API，因为大家都是markdown，语义结构一致。

如果Skill里引用了Claude Code特有的工具（比如特定的MCP Server），那部分需要调整成Hermes对应的工具名。但核心逻辑、触发条件、行为规则都是通用的。

agentskills.io的意义：Skill不再是某个Agent的专属资产。你在Claude Code、Cursor、Gemini CLI里积累的Skill，Hermes里直接用。反过来也一样。选Agent时不用担心迁移成本。

下一节看MCP集成。Skill教Hermes怎么做事，MCP让它连接外部工具。两者配合起来，能覆盖的场景就非常广了。

§11 MCP集成：连接你的工具栈

MCP Integration: Connecting Your Tool Stack

Hermes内置40多个工具已经很能打了，但真实工作场景远不止这些。MCP让Hermes接入GitHub、数据库、Slack、Jira等6000多个外部服务，不用写一行适配代码。

MCP是什么，为什么重要

MCP全称Model Context Protocol，Anthropic在2024年底提出的开放标准。可以理解为**AI工具世界的USB接口**：只要MCP Server实现了这个协议，任何支持MCP的Agent都能直接调用它提供的工具。

对Hermes来说，MCP意味着**不用为每个外部服务开发专用工具**。想接GitHub？装个GitHub MCP Server。想查数据库？装个数据库MCP Server。社区已经有几千个现成的Server可用。

两种连接方式：stdio和HTTP

Hermes支持两种方式连接MCP Server，区别在于Server运行在哪里。

方式	Server位置	适用场景	性能
stdio	本机子进程	本地工具、文件系统、数据库	快，无网络开销
HTTP (SSE)	远程服务器	云服务、团队共享的Server	取决于网络

大多数情况下用**stdio**就够了。MCP Server作为Hermes的子进程运行，通信走标准输入输出，速度快、配置简单。HTTP方式适合Server需要独立部署或多个Agent共享的场景。

配置在 config.yaml 的 mcp_servers 部分：

```
# stdio方式
mcp_servers:
  github:
    command: "npx"
    args: ["-y", "@modelcontextprotocol/server-github"]
    env:
      GITHUB_PERSONAL_ACCESS_TOKEN: "ghp_xxxxx"

# HTTP方式
mcp_servers:
  remote-tools:
    url: "https://your-server.com/mcp"
    transport: "sse"
```

实战：接入GitHub MCP

GitHub是最常用的MCP集成之一。接入后Hermes可以直接创建Issue、提PR、审查代码、管理项目面板。

准备GitHub Token

去GitHub Settings → Developer settings → Personal access tokens，生成一个token。权限至少勾选 repo 和 read:org。如果你要操作Issues和PR，把 issues 和 pull_requests 的写权限也打开。

配置config.yaml

在 config.yaml 里添加GitHub MCP Server：

```
mcp_servers:
  github:
    command: "npx"
    args: ["-y", "@modelcontextprotocol/server-github"]
    env:
      GITHUB_PERSONAL_ACCESS_TOKEN: "ghp_your_token_here"
```

token建议放在环境变量里而不是直接写在配置文件中。可以用 \${GITHUB_TOKEN} 引用。

重启Hermes并验证

重启后，对Hermes说「列出我GitHub上的仓库」或「查看XX仓库最近的Issue」。如果它能正确返回信息，说明连接成功。

日常使用

连接之后，你可以用自然语言操作GitHub。比如：

「帮我在alchaincyf/my-app仓库创建一个Issue，标题是修复登录页面跳转问题」
「看看这个PR的改动，帮我做一下代码审查」
「最近一周有哪些新的Issue被打开了？按标签分类列出来」

GitHub MCP提供的工具涵盖仓库管理、Issue操作、PR审查、代码搜索、分支管理等。你不需要记住这些工具的名字，用自然语言说需求就行，Hermes自动选对应的工具调用。

实战：接入数据库

数据库是另一个高频场景。接入后Hermes可以直接查数据、生成报表、分析趋势，不用你手写SQL。

以PostgreSQL为例：

```
mcp_servers:
  postgres:
    command: "npx"
    args: ["-y", "@modelcontextprotocol/server-postgres"]
    env:
      POSTGRES_CONNECTION_STRING: "postgresql://user:pass@localhost:5432/mydb"
```

配置完成后，你可以对Hermes说「查一下这个月注册用户有多少」「最近30天每天的订单金额趋势」，它会自动生成SQL查询、执行、返回结果。

注意

数据库MCP默认有读写权限。如果你只想让Hermes查询而不修改数据，建议使用只读的数据库账号连接。生产环境的数据库尤其要注意这一点。

SQLite、MySQL同样有对应的MCP Server。配置方式大同小异，换个Server包名和连接字符串就行。

per-server工具过滤

连接多个MCP Server之后，可用工具会变得很多。GitHub一个Server就提供十几个工具，加上数据库、文件系统、Slack，工具列表可能有几十上百个。

工具太多会降低Agent的决策质量。面对100个工具，匹配准确率不如面对20个。而且有些工具你根本不想让它碰。

Hermes支持per-server的工具过滤，在配置中指定每个Server只暴露哪些工具：

```
mcp_servers:
  github:
    command: "npx"
    args: ["-y", "@modelcontextprotocol/server-github"]
    env:
      GITHUB_PERSONAL_ACCESS_TOKEN: "ghp_xxxxx"
    allowed_tools:
      - "list_issues"
      - "create_issue"
      - "get_pull_request"
      - "create_pull_request_review"
```

这样即使GitHub MCP Server提供了删除仓库、修改设置等高权限工具，Hermes也用不到。**最小权限原则在Agent时代比以往任何时候都重要。**

什么时候用MCP，什么时候用原生工具

Hermes自带40多个工具，MCP又能接入几千个。怎么选？

推荐

用原生工具：

终端命令、文件操作、Web搜索、图像生成、记忆管理、子Agent委派。这些内建工具经过深度优化，和学习循环、记忆系统紧密集成，**响应快，行为可预测。**

不推荐

用MCP：

GitHub、数据库、Slack、Jira、Google Drive等外部服务。这些需要特定API协议才能访问，用MCP是最合理的选择。终端也能调CLI完成，但MCP提供结构化输入输出，**准确率更高。**

一个简单的判断标准：**如果Hermes已经内置了这个能力，用内置的；如果需要和外部服务交互，用MCP。**

有些场景两者都能做，比如操作Git。terminal工具可以直接跑git命令，GitHub MCP也能操作仓库。**区别在于：**terminal走本地操作，适合当前仓库的日常commit/push；GitHub MCP通过API操作，适合跨仓库管理、批量操作Issue、PR审查

这类需要平台能力的场景。

实践建议：刚开始别一次性接入太多MCP Server。先接一两个最常用的（GitHub、数据库），用熟了再加。每多一个Server，工具选择空间大一圈，决策路径也变长。

MCP + Skill的组合

MCP解决「能连什么」，Skill解决「怎么用」。两者配合效果更好。

比如：你接入了GitHub MCP，又创建了一个「代码审查」Skill。Skill定义审查标准（命名规范、错误处理、测试覆盖率），MCP提供读取PR改动的能力。两者结合，Hermes就能按你的标准自动审查代码。

再比如：数据库MCP让Hermes能执行SQL，「周报生成」Skill定义了周报格式和重点指标。每周五下午配合cron定时任务，Hermes自动查数据、生成周报、发到Slack。**MCP、Skill和原生工具三者配合，才是真正的威力所在。**

到这里，Hermes的核心能力你都了解了。下一个Part进入实战场景，看看这些能力组合起来能做出什么。

§12 个人知识助手：跨会话记忆的威力

Personal Knowledge Assistant: The Power of Cross-Session Memory

一个真正有记忆的AI助手，和你每天都得重新自我介绍的AI，体验差距有多大？用一个持续三周的项目来感受。

三周调研，从头讲起

假设你在调研一个新领域。比如你是个独立开发者，想搞清楚2026年AI Agent的部署方案：本地、云端、Serverless，各有什么坑。

第一周，你问了三四个问题，聊了Docker部署的内存占用、VPS的价格对比、Daytona的免费额度限制。这些信息散落在不同对话里。

第二周你想继续深挖Serverless方案。打开ChatGPT或者Claude，第一件事是什么？**重新解释你在干什么。**

「我在调研AI Agent部署方案，上周看了Docker和VPS，现在想了解Serverless，之前发现Daytona有免费额度但有限制……」

每次开新对话都得花3到5分钟铺垫背景。这不是AI的能力问题，**是架构问题**：传统AI没有跨会话记忆，每次都是一张白纸。

Hermes记住了什么

同样的场景，用Hermes来做。

第一周的对话结束后，Hermes的三层记忆分别记下了不同的东西：

记忆层	记录内容	用途
会话记忆（SQLite + FTS5）	你问了什么、它查了什么、对话原文	需要细节时精确检索
持久记忆	「用户在调研AI Agent部署，已排除方案X，偏好低成本」	下次对话自动加载上下文
Skill记忆	「调研类任务：先列维度→逐个深挖→每轮汇总」	方法论复用

第二周你打开Hermes，直接说「继续看Serverless方案」。不需要重新解释背景，持久记忆已经知道你在做什么。**它甚至会主动提醒你：上周你提到Daytona有免费额度限制，要不要先确认最新政策？**

这不是魔法，是FTS5全文搜索在工作。Hermes不会把上周的对话全部塞进上下文，那样太浪费token。它根据你当前的问题，检索最相关的历史片段。

检索 vs 全量加载

这个设计选择值得展开说。

很多人以为「记忆」就是把所有历史对话塞进prompt。Claude Code的auto-memory确实这么做：把关键信息写进MEMORY.md，每次启动全部读取。编码场景够用了。

但知识助手场景不一样。三周的调研对话可能有几万字，全部加载会出两个问题：**token成本爆炸**，以及**信息过载反而降低回答质量**。大模型在超长上下文中的注意力分布不均匀，关键信息容易被淹没。

Hermes的做法是：**持久记忆存摘要（几百字）**，需要细节时用FTS5搜索原始对话，只取最相关的片段注入上下文。相当于随身带一页纸笔记，需要时再去翻档案柜。

推荐

Hermes方式：持久记忆（摘要）+ 按需检索（FTS5）。
token消耗可控，信息精准。

不推荐

全量加载方式：历史全塞进prompt。短期有效，三周后prompt爆满，成本翻倍。

Honcho：它比你更了解你

如果启用了Honcho用户建模，记忆的深度还会再上一个台阶。

Honcho不只是记录你说了什么，**它会推导你没说的东西**。比如你连续三次调研都选了成本最低的方案，Honcho会推导出「这个用户对成本敏感」。下次推荐方案时，价格信息自动前置。

这种辩证建模覆盖多个身份维度，从技术水平、偏好风格到沟通习惯。**用得越久，它对你的理解越精准。**

花叔的体验：我用了两周后，Hermes开始自动用短句回复我，因为它发现我倾向于直接要结论而不是长篇分析。这种适应是渐进的，不需要你显式配置任何东西。

和传统AI对话的体验差距

用一个类比来总结：**传统AI像旅馆前台，每天换人，你每次都得重新自我介绍**。Hermes像你的私人助理，认识你三个月了，知道你喜欢喝美式、讨厌写周报、最近在调研什么项目。

具体到知识助手场景，差距体现在三个方面：

启动成本为零

不用每次铺垫背景。说「继续」就是继续。

调研有连续性

上周排除的方案不会被重新推荐。已确认的信息不会被重新查证。整个调研像一条线而不是一堆散点。

方法论会成长

Hermes帮你做第一个调研项目时总结出的方法（先列维度、逐个深挖、每轮汇总），会被存为Skill。**第二个调研项目自动复用，不用你再教一遍。**

核心建议

跨会话记忆不是万能药。如果任务是一次性的，比如「帮我翻译这段话」，Hermes的记忆系统没有任何优势。**记忆的价值和任务的持续时间成正比**。任务越长、上下文越复杂，记忆带来的效率增益越大。

§13 开发自动化：代码审查到部署

Dev Automation: From Code Review to Deployment

Claude Code写代码，Hermes看守流水线。两个工具不是二选一，而是各管一段。

一个开发者的早晨

早上9点打开电脑，Telegram弹出三条消息。不是同事发的，是Hermes发的：

「昨晚23:17，main分支有一个PR合并，新增387行代码。审查了一下有两个问题：auth模块的token过期逻辑没处理边界情况，测试覆盖率从82%掉到76%。详细报告已存到项目Skill里。」

「凌晨2:40，CI流水线跑了一轮回归测试，3个用例失败。2个是昨天那个PR引入的，1个是已知的flaky test。」

「今天的日报初稿已生成，基于昨天的4个commit和2个PR。需要你确认后发送。」

这不是假想场景。**Hermes的cron调度 + GitHub MCP + 记忆系统**，可以让它在你睡觉时持续工作。

代码审查自动化

传统代码审查有两个痛点：**延迟**，提了PR得等reviewer有空；**一致性**，不同reviewer关注点不同，有人看架构有人看格式。

用Hermes做自动代码审查的配置很简单：

接入GitHub MCP

在config.yaml中配置GitHub MCP Server，Hermes就能读取仓库的PR、diff、issue。

设置cron调度

用自然语言告诉Hermes：「每6小时检查一次main分支的新PR，做代码审查」。它会自动创建cron任务。

定义审查标准

把你的代码规范写成一个Skill。比如「函数不超过50行」「错误处理必须用自定义类型」「所有API端点必须有测试」。Hermes按这个标准审查每个PR。

关键在第三步。审查标准是Skill，**它会随着你的反馈自动演进**。你标记了某个Hermes没发现的问题，下次它就会注意同类模式。传统的lint规则是静态的，Hermes的审查标准是活的。

测试生成和执行

Hermes做测试和Claude Code做测试有个根本区别：Claude Code是你告诉它「给这个函数写测试」，它写完你验收。

Hermes是自己发现哪些函数没测试，自己写，自己跑，跑不过自己修，最后给你一个报告。

这个能力来自几个工具的组合：

工具	作用
file	扫描代码目录，找到缺少测试的模块
code_execution	在沙箱中运行测试
terminal	执行覆盖率报告生成
memory	记住哪些模块已审查、哪些测试是flaky的

可以配合cron做周度覆盖率检查。每周一早上自动扫描，覆盖率低于阈值就发通知提醒。

日报和周报自动汇总

这个功能听起来简单，但用起来会上瘾。

Hermes通过GitHub MCP拿到当天的commit记录、PR状态、issue变化，结合会话记忆中你讨论的内容，**生成一份你自己都不需要回忆就能确认的日报。**

周报更有意思。因为有跨会话记忆，Hermes能看到整周的脉络：周一修了什么bug，周三为什么改了架构方案，周五的PR为什么被打回。**不是简单的commit列表，而是有因果关系的叙述。**

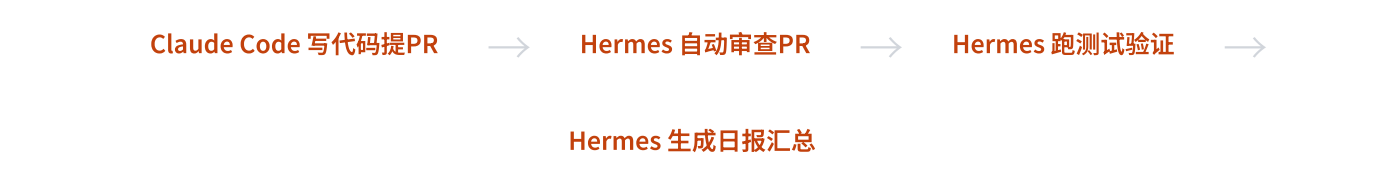
实用建议：让Hermes生成日报时，同时更新一个项目Skill记录本周的技术决策。下周做周报时它能直接引用决策记录，不用从commit message里猜你当时在想什么。

Claude Code和Hermes的分工

这两个工具不是竞争关系。它们擅长的东西完全不同：

维度	Claude Code	Hermes Agent
交互模式	你在旁边，实时对话	后台运行，定时汇报
擅长	写代码、重构、调试	巡检、监控、汇总、调度
时间维度	一个session内完成	跨天、跨周持续运行
触发方式	你主动发起	cron自动触发或事件驱动

一句话总结：Claude Code是工匠，Hermes是管家。工匠负责做东西，管家负责确保一切都在正轨上。你不会让管家去砌墙，也不会让工匠去巡夜。



这条流水线跑起来之后，工作重心从「写代码+看代码+跑测试+写日报」变成「写代码+确认结果」。**中间环节全部自动化。**

注意

Hermes的代码审查是辅助而非替代。它擅长发现模式化的问题（命名不规范、缺少测试、边界条件遗漏），但架构层面的判断仍需人工。别因为自动化了就跳过人工review，至少核心模块要保留人工环节。

§14 内容创作：从调研到成稿

Content Creation: From Research to Final Draft

用AI写文章不稀奇，让AI记住你的写作风格、调研习惯、甚至读者反馈，持续进化成你的专属编辑，这才是Hermes做内容的方式。

一篇文章的前世今生

我用Claude Code写了超过100篇公众号文章。流程已经很熟了：给它一个选题，它帮我搜索、写草稿、审校、配图。整个过程大概2到3小时。

但有一个问题始终存在：**每次开新会话，它都不记得上一篇文章的教训。**

上周写那篇文章时，我让它别用「综上所述」，它改了。这周写新文章，「综上所述」又出来了。我在CLAUDE.md里加了禁用词清单，好了一些，但CLAUDE.md是静态文件，它不会自己更新。

Hermes做内容创作的不同，就在这个点上。

持续性内容项目

假设你在做一个系列选题，比如连续写五篇关于AI Agent的文章。用传统方式，每篇都是独立的：你得重新说明读者画像、重新定义风格、重新说明哪些话题前面已经写过了。

用Hermes做这个系列，情况完全不同：

第一篇文章写完，Hermes的记忆系统记下了几件事：这个系列的定位、目标读者、你对草稿的修改偏好（比如你把它写的长句全部拆成了短句），以及哪些技术点已经在第一篇里解释过了。

第二篇开始时，你只需要说「写这个系列的第二篇，主题是XXX」。它知道该用什么风格、知道哪些概念不用再解释、知道你上次不满意的地方。

到第五篇的时候，它对你写作偏好的理解已经很精准了。不是因为你配置了什么，**是因为它从你的反馈中自己学会了。**

子Agent并行调研

写一篇深度技术文章，调研往往比写作花时间更多。传统做法是线性的：搜索A话题，整理完了搜索B话题，再整理C话题。

Hermes的delegate_task工具可以让这个过程并行。比如你要写一篇AI Agent对比文章，可以同时派出三个子Agent：



三个子Agent同时工作，各自返回调研结果。主Agent汇总后，你得到一份结构化的调研素材。**原来要花一个多小时的调研，20分钟就能搞定。**

每个子Agent可以分配不同的工具集。调研架构的那个需要web搜索和browser，搜集社区反馈的可能只需要web搜索。
受限工具集不只是安全设计，也是效率设计——子Agent不会因为工具太多而分心。

Skill积累写作风格

这是Hermes做内容创作最有价值的部分。

传统AI写作工具的风格控制靠prompt。你在prompt里写「口语化、句子不超过20字、不要用AI高频词」。每次都得写一遍，或者维护一个很长的系统提示词。

Hermes的方式是把风格规则存成Skill。一开始这个Skill可能只有几条简单规则：别用「综上所述」、段落保持3到5行、多用「我觉得」「其实」这类口语词。

关键来了：这个Skill会自改进。

你审校草稿时做的修改，Hermes会观察并学习。你连续三次把「进行优化」改成「优化一下」，它就会在Skill里加一条：「避免用'进行+动词'的句式」。你删掉了一段它加的总结升华，它学到「结尾不要强行升华」。

一个月后，这个写作风格Skill已经积累了几十条规则，全部来自你的真实反馈。**它变成了你的专属编辑手册，而且是自动维护的。**

核心建议

Hermes的Skill自改进不是黑箱。每次Skill更新都可以在 `~/.hermes/skills/` 目录下看到diff。如果某条规则学偏了，你可以手动修正，Hermes会把你的修正也纳入学习。

和用Claude Code写文章有什么不同

我同时在用两个工具做内容创作，感受很明确：

维度	Claude Code	Hermes Agent
适合的内容	单篇文章、一次性任务	系列内容、持续性项目
风格控制	CLAUDE.md + 手动维护	Skill自动积累和进化
调研效率	线性搜索	子Agent并行调研
上下文连续性	依赖auto-memory，容量有限	三层记忆，按需检索
学习能力	不学习，规则靠人写	从你的反馈中自动学习

不是说Claude Code不好。写单篇文章，Claude Code的交互体验更流畅，你能实时看到修改、实时给反馈。**Hermes的优势在长线**。每周写两篇、持续三个月，Hermes在第十篇的时候已经比第一篇好太多了，Claude Code第十篇的表现和第一篇差不多。

花叔的做法：单篇商单文章用Claude Code，因为交互快、品牌方反馈能即时调整。系列选题和个人专栏用Hermes，让它的写作Skill慢慢长出来。两者配合，不矛盾。

§15 多Agent编排：让三匹马同时跑

Multi-Agent Orchestration: Running Three Horses at Once

一个Agent不够用的时候，让三个Agent并行工作。delegate_task是Hermes最强大也最容易用错的工具。

为什么需要多Agent

一个Agent的能力上限取决于它的上下文窗口和工具集。当任务足够复杂时，单Agent会撞上两堵墙。

上下文爆炸。一个Agent同时处理调研、编码和测试，三件事的信息全挤在同一个上下文里，互相干扰。调研的网页内容占了大量token，留给代码推理的空间就不够了。

时间瓶颈。线性执行三个任务，每个30分钟，总共90分钟。如果能并行，总时间就是最慢那个的时间。

Hermes的delegate_task工具就是为了解决这两个问题。它能同时启动最多3个子Agent，每个有独立的上下文和工具集。

delegate_task详解

delegate_task不是简单的「开个子线程」，它做了几件关键的事。

特性	说明
独立上下文	子Agent有自己的对话历史，不会污染主Agent的上下文
受限工具集	可以指定子Agent只能使用哪些工具。调研Agent只给web和browser，编码Agent只给terminal和file
独立终端会话	每个子Agent有自己的终端，互不干扰
并发上限3个	硬编码限制，防止资源耗尽
结果回传	子Agent完成后，结果返回给主Agent做汇总

3个并发的限制是经过考虑的设计。Nous Research在测试中发现，超过3个子Agent后，主Agent的汇总质量会急剧下降。**不是算力不够，是大模型在整合过多独立信息源时的注意力分散问题。**

一个实际案例：竞品分析报告

假设你要写一份AI编程工具的竞品分析，覆盖三个产品：Claude Code、Cursor、Hermes Agent。传统做法是一个一个调研，然后手动整合。

用delegate_task，流程变成这样：



中间的并行执行展开看：

主Agent定义任务模板

「请按以下维度调研[产品名]：定位、核心功能、技术架构、定价、社区规模、优缺点。结果用markdown表格输出。」

派出三个子Agent

子Agent A调研Claude Code，子Agent B调研Cursor，子Agent C调研Hermes。**每个子Agent只分配web和browser两个工具**，不需要file或terminal。

主Agent汇总

三个子Agent返回各自的调研结果后，主Agent整合成一份对比报告，补充自己的判断和推荐。

三个调研任务并行，总时间从「A+B+C」变成「 $\max(A, B, C)$ 」。实测下来，**一份通常要40分钟的竞品分析，15分钟就能出。**

受限工具集的安全设计

delegate_task的受限工具集不只是效率工具，更是安全机制。

想象一个场景：你让子Agent去网上搜索一段代码，它搜到了一个有恶意注入的代码片段。如果这个子Agent同时有terminal权限，它可能会执行这段代码。但如果它只有web权限，搜索结果只会作为文本返回给主Agent审查。

这就是受限工具集的价值：**最小权限原则在Agent层面的落地。**

推荐	不推荐
好的实践： 调研子Agent只给web+browser，编码子Agent只给terminal+file+code_execution，汇总子Agent什么外部工具都不给，只处理文本。	坏的实践： 每个子Agent都给全部工具。方便是方便，但失去了隔离的安全保障。

和Anthropic三Agent架构的关系

Anthropic在Agent设计指南中提出了一个经典的三Agent架构：Planner负责规划、Generator负责执行、Evaluator负责评估。这个模式在很多场景下被证明有效。

Hermes的delegate_task和这个架构有相似之处，但也有关键区别：

维度	Anthropic三Agent	Hermes delegate_task
角色划分	固定角色（规划/执行/评估）	任务驱动，角色灵活
通信方式	链式传递（规划→执行→评估）	星形结构（主Agent↔子Agent）
并行性	通常串行	最多3个并发
记忆	无内置记忆	主Agent保持完整记忆

Hermes的模式更灵活。你可以用delegate_task实现Anthropic的三Agent架构——一个子Agent做规划、一个做执行、一个做评估。但你也可以让三个子Agent做同类型的并行任务（比如同时调研三个产品）。架构不是固定的，取决于你的任务怎么拆。

Anthropic的三Agent架构是思维框架，告诉你「复杂任务可以拆成规划、执行、评估」。Hermes的delegate_task是执行工具，让你把框架落地。**一个管设计，一个管实现。**

核心建议
多Agent编排容易过度设计。不是所有任务都需要拆成子Agent。如果一个任务在单Agent的上下文窗口里就能处理好，拆分反而会增加通信开销和汇总错误。**只在上下文不够用或者需要并行加速时，才考虑delegate_task。**

经验法则：如果你发现自己要给主Agent写很长的汇总指令来整合子Agent的结果，说明任务拆分可能有问题。好的拆分应该让汇总变得简单：每个子Agent的输出是自包含的、格式统一的、可以直接拼接的。

§16 Hermes vs OpenClaw vs Claude Code：不是选择题

Not a Choice — A Combination

三个工具不是三条路，是三匹马。你要做的不是挑一匹，是搞清楚谁拉货、谁赶路、谁看家。

三个物种

过去一年，AI Agent生态里冒出了太多工具。但真正值得认真看的，我觉得是三个：Claude Code、OpenClaw、Hermes Agent。

不是因为它们最火，而是因为它们代表了三种完全不同的设计理念。

Claude Code是交互编码工具。你坐在终端前，给它需求，它写代码、跑测试、提交git。你全程在场，像带着一个很强的工程师结对编程。核心价值是实时的代码生产力。

OpenClaw是「配置即行为」的框架。你把Agent的人格、知识、技能全部写进SOUL.md和Skill文件，配置文件决定了Agent是什么样的。核心价值是可预测、可审计、可复制。

Hermes Agent是自主后台引擎。你把它部署在服务器上，它24/7运行，自己记忆、自己创建Skill、自己改进。核心价值是自主性和自改进能力。

维度	Claude Code	OpenClaw	Hermes Agent
核心理念	交互式编码	配置即行为	自主后台+自改进
你的角色	坐在终端前指挥	写配置文件定义行为	部署后偶尔检查
记忆机制	CLAUDE.md + auto-memory	多层记忆（SOUL.md + Daily Logs + 语义搜索），透明可控	三层自改进记忆
Skill来源	手动安装	ClawHub 44000+	Agent自创+社区Hub
运行模式	按需启动	按需启动	24/7后台运行
部署方式	本地CLI（订阅制）	本地CLI（免费+API费）	\$5 VPS / Docker / Serverless

看出区别了吗？这三个工具解决的根本不是同一个问题。

什么场景用哪个

我用了半年Claude Code、几个月OpenClaw、最近又在折腾Hermes。我的体感是，选工具不看工具本身有多强，看你的场景适合哪种交互模式。

场景	推荐工具	理由
写新功能、重构代码	Claude Code	需要实时反馈和人的判断
给团队搭标准化Agent	OpenClaw	SOUL.md一目了然，可审计可复制
7x24小时代码审查	Hermes	cron调度+GitHub MCP，无人值守
个人知识助手	Hermes	三层记忆跨会话积累，越用越懂你
搭建客服/社区Bot	Hermes	原生12+平台Gateway，多端互通
快速验证产品想法	Claude Code	启动快、迭代快、实时调整方向
需要高度可控的企业场景	OpenClaw	配置透明、行为可预测
长期内容创作项目	Hermes + Claude Code	Hermes持续调研积累，Claude Code写稿

最后一行挺重要。很多场景不是一个工具就能搞定的。长期内容项目里，Hermes负责每天自动搜集信息、积累记忆，Claude Code负责坐下来把东西写出来。各管一段。

趋同还是分化

一个有意思的现象：这三个工具在互相学习对方的长处。

Claude Code加了auto-memory，在向Hermes的持久记忆靠拢。OpenClaw的ClawHub有44000+个社区Skill，Hermes也开始建自己的Skill Hub。Hermes支持agentskills.io标准，意味着它可以直接使用Claude Code生态的Skill。

看起来是趋同。但我觉得底层分化其实在加大。

Claude Code的本质是人和AI的实时对话。不管加多少记忆和自动化能力，你坐在那里看着它干活这件事不会变。Anthropic的商业模式决定了这一点：订阅制，按你使用的时间收费。

Hermes的本质是AI在后台自主运行。不管加多少交互界面，它的核心价值是你不在的时候它也在工作。MIT开源+自托管的模式决定了这一点。

OpenClaw在中间。它既不像Claude Code那么强调实时交互，也不像Hermes那么强调自主运行。它的独特价值是「透明可控」。SOUL.md让你一眼就能看清这个Agent会做什么、不会做什么。在企业合规场景里，这个属性无可替代。

agentskills.io：Skill互通的意义

2026年初，agentskills.io标准开始被多个工具采用。目前已有30+个工具支持这个标准，包括Claude Code和Hermes。

这意味着什么？

意味着你给Claude Code写的Skill，可以直接被Hermes使用。你在Hermes里自动创建的Skill，也可以被拿到Claude Code里用。Skill不再绑定某个特定工具，它变成了一种可移植的能力单元。

这件事的长远影响可能比任何单个工具本身都大。因为它在说：**不管你选哪匹马，马鞍是通用的。**

你花在写Skill上的时间不会因为换工具而浪费。你积累的Skill库是你自己的资产，不是某个平台的附属品。

OpenClaw的ClawHub有44000+个Skill。如果这些Skill能通过agentskills.io标准被Hermes直接调用，Hermes的能力边界瞬间就展开了。反过来，Hermes自动创建和改进的Skill也可以反哺回整个生态。

不是选择题，是组合题

写了三本橙皮书（Claude Code、Harness Engineering、OpenClaw），我现在对这个领域最大的判断是：**最终胜出的不是某个工具，而是会组合工具的人。**

我现在的工作方式就是组合。Claude Code处理所有需要我在场的事：写文章、写代码、做产品决策。它是我的「白天团队」。

Hermes（或者类似的自主Agent）处理不需要我在场的事：监控代码仓库、定时跑调研、维护知识库。它是我的「夜班团队」。

OpenClaw的SOUL.md和Skill系统给了我一套标准化的配置语言。不管底下跑的是Claude Code还是Hermes，行为约束的写法是一致的。

花叔的判断：不要在这三个工具之间做「选择」。问自己三个问题：什么任务需要我盯着？什么任务可以放后台跑？什么场景需要透明可审计？答案自然把它们分到不同的位置上。

Agent工具的竞争不会收敛到一个赢家。就像你不会用锤子拧螺丝一样，交互编码、配置管理、自主运行是三种不同的工作模式，它们会长期共存。

真正有意思的问题不是「哪个更好」，而是「怎么让它们协作」。agentskills.io已经在铺这条路了。

§17 自改进Agent的边界：它能走多远

The Boundaries of Self-Improving Agents

Hermes最让人兴奋的能力，也是最让人不安的能力。

重提那张图

在Harness Engineering橙皮书的最后一章，我写了Kief Morris画的那张图。三层：in the loop、on the loop、out of the loop。

In the loop：逐行审查Agent的输出。On the loop：不看输出，只管缰绳。Out of the loop：你说要什么，Agent全搞定。

当时我的结论是，on the loop可能是最好的平衡点。你不重复劳动，但你还在。

Hermes Agent把这个讨论推到了新的位置。

它的学习循环是自动的。它自己创建Skill、自己改进Skill、自己决定什么该记住。你部署好之后，它会越来越强。这不是on the loop了。你连缰绳都不用亲手调，缰绳自己在长。

这到底是进步还是风险？

Skill自改进会失控吗

先说技术层面。Hermes的Skill自改进有几个约束。

Skill文件是可读的markdown。不是黑箱神经网络权重，是你打开就能看到的文本。它改了什么，你能看到diff。

记忆数据在本地。基于SQLite + FTS5，数据在本地磁盘上，你可以直接查看和删除。不存在「Agent偷偷学了什么你不知道」的情况。

工具权限有沙箱。Agent不能随意获取新的系统权限，工具集需要显式配置。

从技术上看，Hermes的自改进是受控的、可审计的。你能看到它改了什么，能回滚，能删除。

但技术上受控不等于实际上受控。

问题出在人这边。

你会真的每天去看Agent改了哪些Skill吗？你会去审查它的记忆数据库吗？大概率不会。部署Hermes的吸引力就在于「不用管它」。如果你每天都要检查它的自改进结果，那和手动维护Skill有什么区别？

这个矛盾是根本性的：自主Agent的价值在于你不用盯着，但安全需要你盯着。

Kief Morris的那个判断在这里再次成立：**in the loop**和**on the loop**的区别，在你对结果不满意的时候最明显。但如果Agent自改进了一个Skill而你从来没发现结果不对，你什么时候才会注意到？

Nous Research的选择

Nous Research的创始团队在这个问题上做了一个清晰的选择：用户控制优先。

他们明确说过，模型应该是「可控的」（steerable），用户可以按需调整行为，不受企业内容策略的限制。

这不只是口号。Hermes Agent的MIT许可证意味着你拥有全部源代码，可以审计学习循环的每一步逻辑，可以修改自改进的阈值、频率、范围，也可以把Skill自动创建功能直接关掉。

和闭源Agent相比，这种透明性给了你一个底线：最坏的情况下，你能看到所有代码在做什么。

但我要说一句不太舒服的话。

「你能看到代码」和「你看了代码」是两回事。绝大多数用户不会读源代码。绝大多数部署Hermes的人会用默认配置。MIT许可证给了你审计的权利，但不保证你会行使这个权利。

开源vs闭源：信任的不同形状

这就引出了一个更大的问题：在自改进Agent这件事上，开源和闭源哪个更值得信任？

直觉上的答案是开源。代码透明，社区审计，MIT许可。

但现实比直觉复杂。

Claude Code是闭源的。你不知道Anthropic的Agent内部逻辑长什么样。但Anthropic有一个明确的商业激励让Agent行为可预测：如果Agent失控导致用户的代码仓库出问题，他们会丢掉订阅用户。商业压力是一种约束。

Hermes是开源的。你可以看到所有代码。但如果Agent的自改进产生了问题，Nous Research没有商业义务帮你修。MIT许可的另一面是：后果自负。

两种信任的形状：一种是「我信任你的商业动机」，另一种是「我信任自己的审计能力」。

有技术能力的人，开源显然更好，你能控制一切。不想碰代码只想用的人，闭源商业服务可能反而更安全，因为有人帮你兜底。

自改进的天花板在哪

回到那个核心问题：自改进Agent能走多远？

我的判断是：**天花板不在技术，在反馈信号。**

Hermes的自改进循环依赖一个关键假设：它能判断自己的改进是好是坏。它改了一个Skill，下次任务做得更好了，这就是正反馈。但「更好」是谁定义的？

如果你在场给反馈，循环是有效的。你说「不对」，它调整。这是带监督的改进。

如果你不在场，Agent只能用自己的评估标准。它觉得这次回复更快了、更准了。但「快」和「准」不等于「对」。有些错误需要领域知识才能发现。Agent不知道自己不知道什么。

在Harness Engineering那本书里我写过：Mitchell Hashimoto能给Ghostty写好harness，因为他理解终端模拟器的每个细节。自改进Agent没有Mitchell的领域知识。它能优化执行效率，但不能判断方向是否正确。

自改进让Agent在已知方向上越跑越快。但方向本身，还是得人来定。

留给你的问题

写到这里，我不想给一个漂亮的结论。因为这些问题没有确定答案，而且答案会随着技术发展不断变化。

但我想把几个值得你持续想的问题列出来。

你愿意让Agent自主改进到什么程度？

改写Skill文件？可以。自动创建新Skill？大概可以。修改自己的核心配置？也许不行。修改自己的学习循环逻辑？绝对不行。你的边界画在哪里？

谁来审计自改进的结果？

你自己审计，多久审一次？社区审计，你信任社区的判断吗？没人审计，你接受这个风险吗？

自改进Agent需要「遗忘」机制吗？

人会遗忘，这不是bug而是feature。过时的经验被忘掉，才不会干扰当下的判断。Agent的记忆只增不减，三个月前学到的模式可能已经过时了。谁来告诉Agent「这条该忘了」？

Kief Morris的担忧，在Hermes这里变成了什么？

Morris担心的是：如果junior开发者不接触代码细节，将来谁来设计harness？在Hermes的语境下，问题变成了：如果Agent自己设计自己的缰绳，将来谁来判断缰绳设计得对不对？

也许答案是：永远需要一个人在某个层面保持in the loop。不是审查每一行代码，而是理解整个系统在做什么、为什么这么做。

也许答案是：我们还不知道。

花叔的判断：自改进Agent是这个领域最让人兴奋的方向，但它的天花板由人的参与程度决定。完全放手不管的自改进Agent，会在效率上赢、在方向上输。最好的状态可能是：让Agent在「怎么做」上自改进，你只管「做什么」和「别做什么」。这不是偷懒，是另一种on the loop。