



中国科学院大学

University of Chinese Academy of Sciences

# 自然语言处理

## 练习3 搭建一个多智能体系统

王石 资康莉 刘瑜

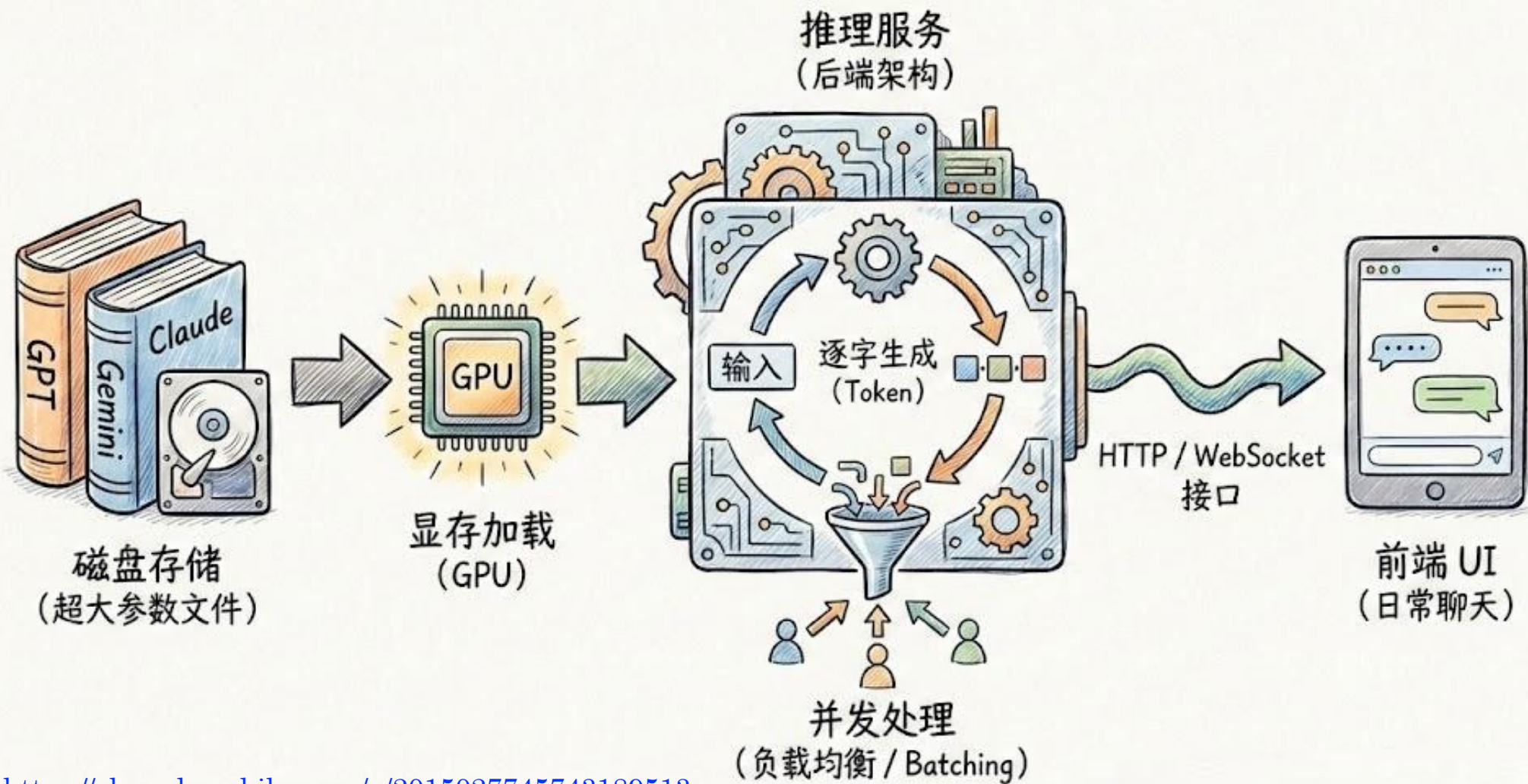
2026年春季课程

<https://ictkc.github.io/teaching/>



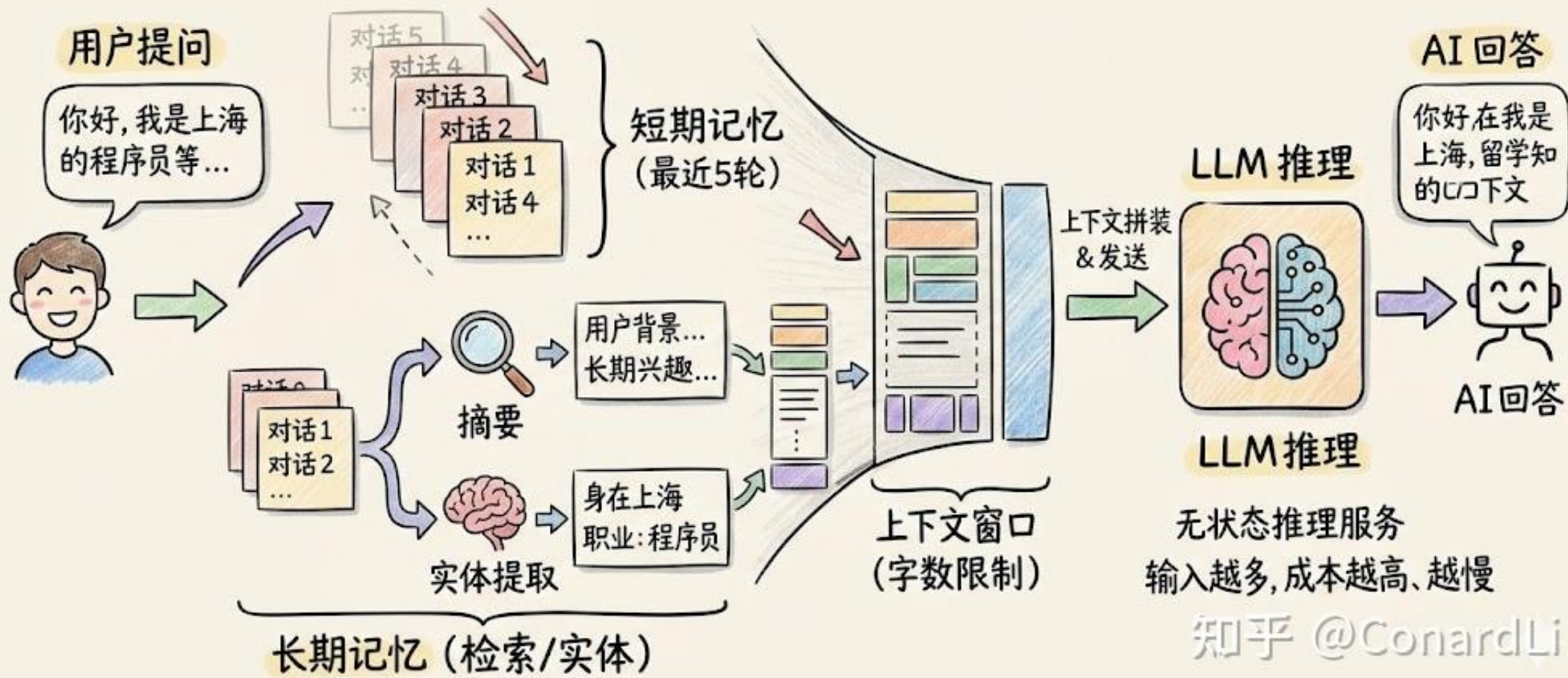
# 练习3

## 搭建一个MAS



<https://zhuatlan.zhihu.com/p/2015027745743189513>

# Memory 机制: 大模型记忆管理

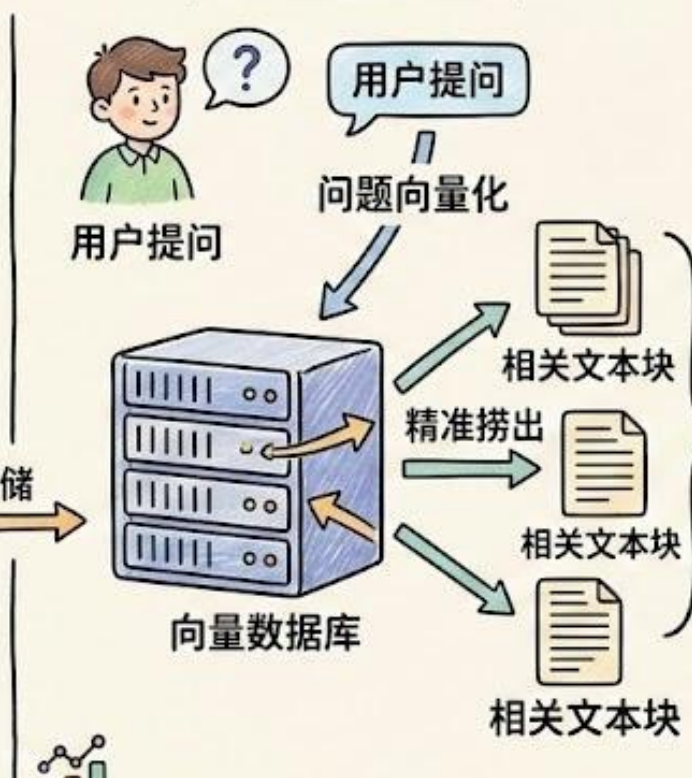


# RAG 检索增强生成方案的核心流程

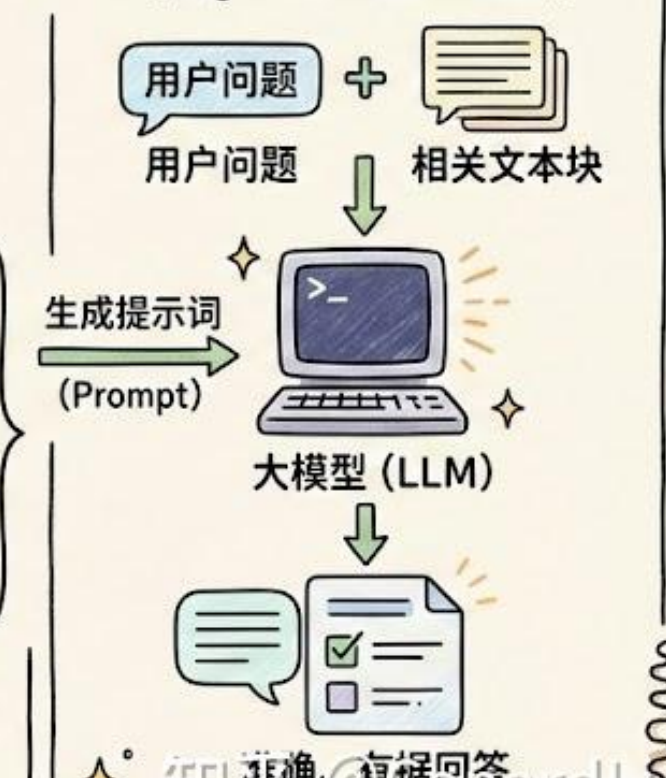
## 1. 数据准备与向量化 (Data Preparation & Vectorization)



## 2. 语义检索 (Semantic Retrieval)

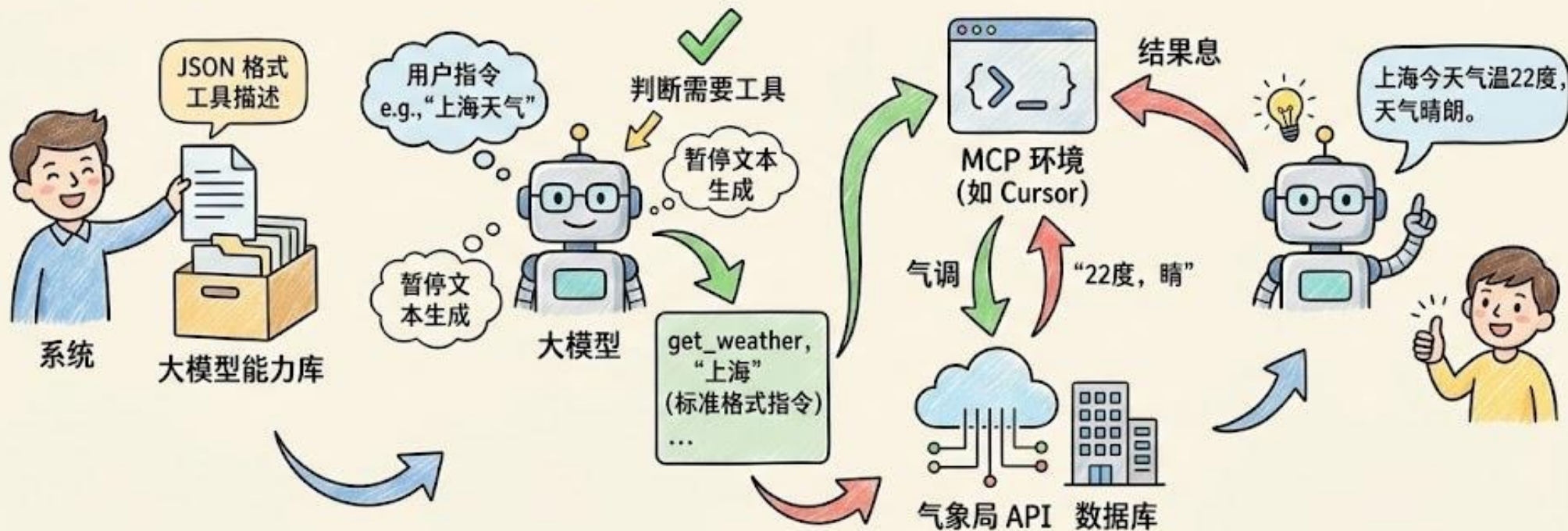


## 3. 增强生成 (Augmented Generation)



# MCP (Model Context Protocol) 与工具调用 (Function Calling) 流程图

1. 能力注册 (Capability Registration)
2. 意图解析与输出 (Intent Parsing & Output)
3. 外部执行与回传 (External Execution & Callback)
4. 整理输出 (Final Output)



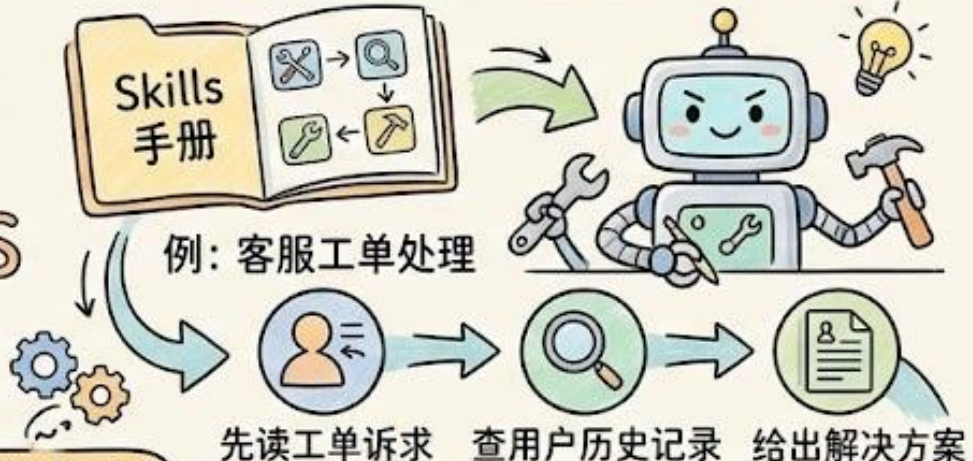
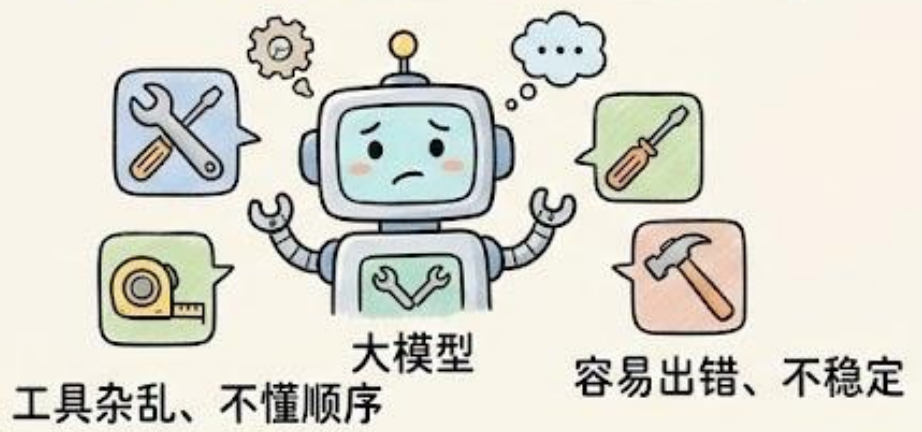
知乎 @Conard

# MCP 与 Skills: 大模型如何学会使用工具

没有 Skills 的 MCP vs 引入 Skills 后的大模型

仅有 MCP: 只有工具, 不懂流程

引入 Skills: 结构化的操作手册



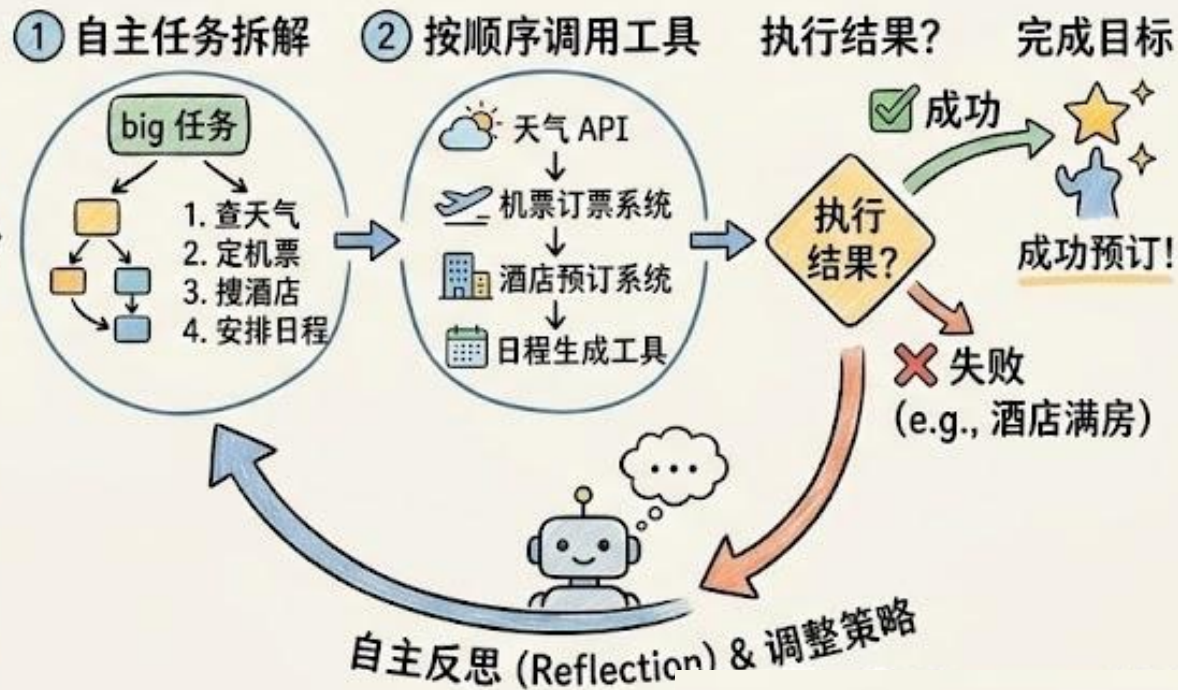
知乎 @ConardLi

# AI Agent (智能体) 的组成与自主行动流程



## Agent 自主规划 (Planning) & 反思 (Reflection)

📍 目标: 策划下周去东京旅行

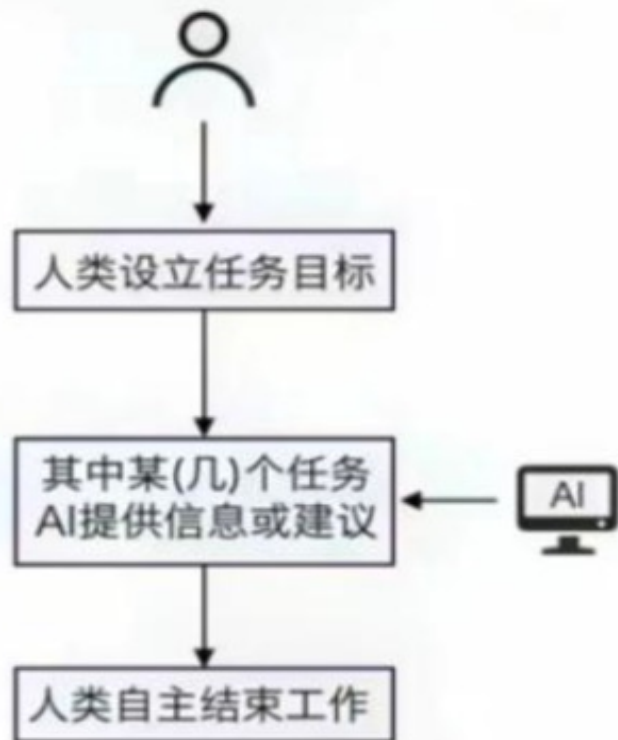


## 人类与AI协同的三种模式

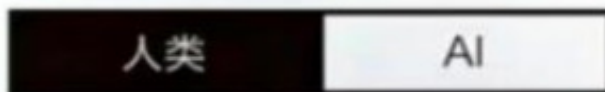
### Embedding 模式



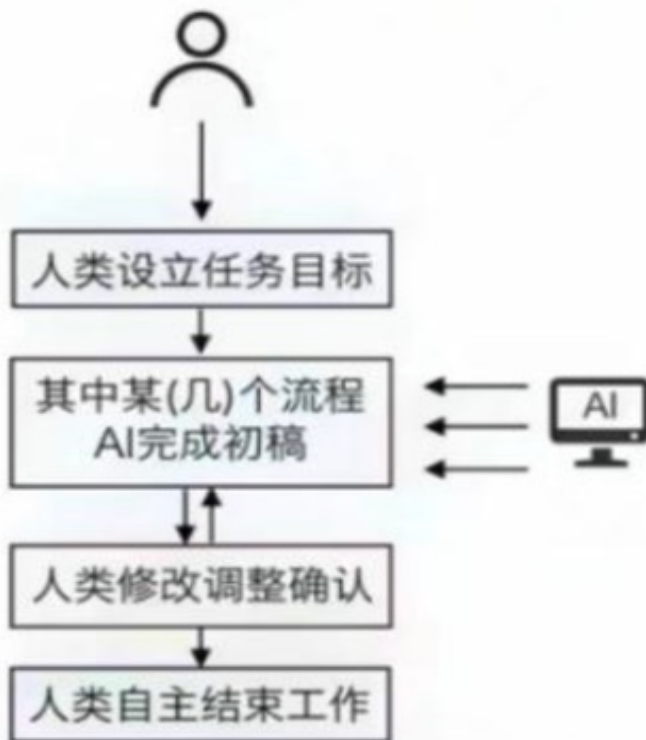
人类完成绝大部分工作



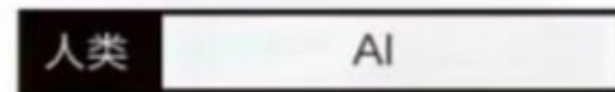
### Copilot 模式



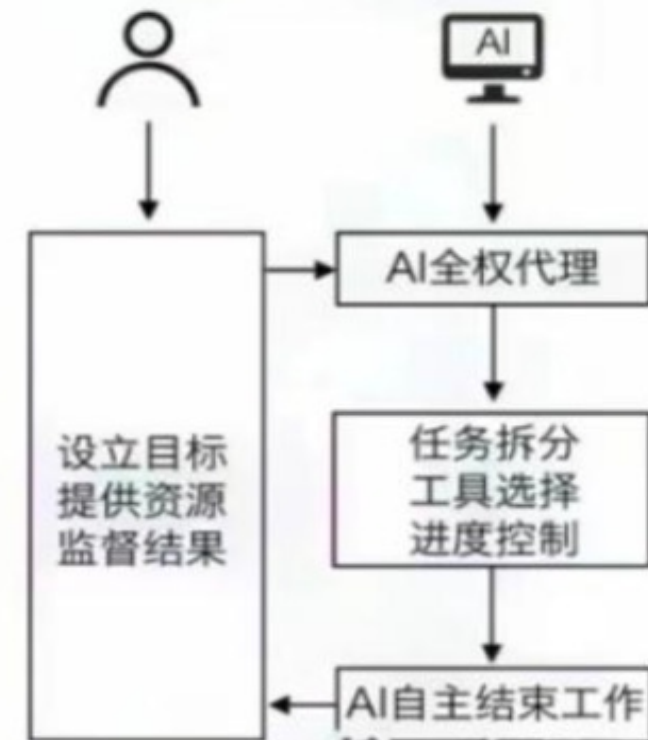
人类和AI协作工作



### Agents 模式



AI完成绝大部分工作





# 目 录

1

LangChain / LangGraph

---

2

---

3

---

---

# 1.1 什么是langchain

---



## LangChain

LLM驱动应用程序的开发框架

# 1.2 LangChain 模块

01

## Agents

智能体将语言模型与工具结合，创建能够对任务进行推理、决定使用哪些工具并迭代寻求解决方案的系统。

02

## Models

模型是 agent 的推理引擎。它们驱动代理的决策过程，决定调用哪些工具、如何解释结果以及何时提供最终答案。

03

## Messages

消息是 LangChain 中模型上下文的基本单位。它们代表模型的输入和输出，携带内容和元数据，用于在与 LLM 交互时表示对话状态。

04

## Tools

工具是 agents 执行操作的组件。它们通过允许模型通过定义明确的输入和输出与世界交互来扩展模型的功能。

# 1.2 LangChain 模块

05

## Memory

记忆系统能帮助 AI 代理保留过往交互、学习反馈并适应用户偏好，从而在复杂、多轮任务中提升效率和用户满意度。

06

## 流式传输

LangChain 实现了一个流式传输系统来提供实时更新。渐进式地显示输出，改善用户体验

07

## 中间件

更精细地控制智能体 (Agent) 内部的执行流程。

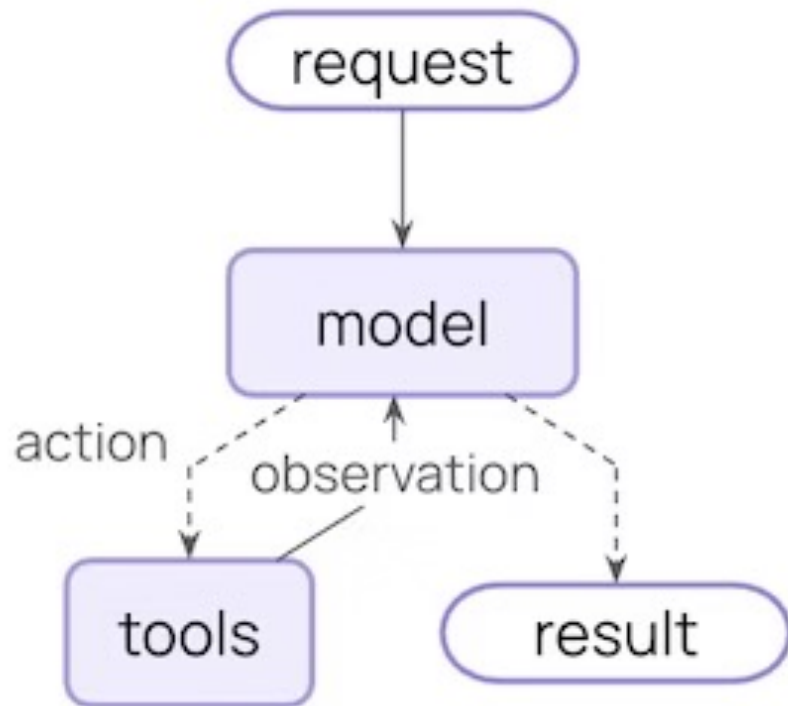
08

## 结构化输出

结构化输出允许智能体 (agents) 以特定的、可预测的格式返回数据。

# 1.2.1 Agents

---



```
from langchain.agents import create_agent

agent = create_agent(
    "openai:gpt-5",
    tools=tools
)
```

# 1.2.2 Models

---

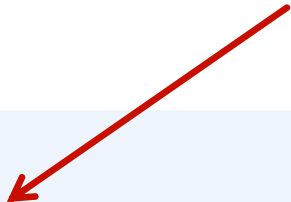
## 模型初始化

```
init_chat_model(  
    model: str | None = None,  
    *,  
    model_provider: str | None = None,  
    configurable_fields: Literal['any'] | list[str] | tuple[str, ...] | None = None,  
    config_prefix: str | None = None,  
    **kwargs: Any = {}  
) -> BaseChatModel | _ConfigurableModel
```

## 1.2.2 Models

### 模型初始化

```
1 import os
2 from langchain.chat_models import init_chat_model
3
4 os.environ["OPENAI_API_KEY"] = "sk-..."
5
6 model = init_chat_model("openai:gpt-4.1")
```



```
1 import os
2 from langchain.chat_models import init_chat_model
3
4 os.environ["ANTHROPIC_API_KEY"] = "sk-..."
5
6 model = init_chat_model("anthropic:claude-sonnet-4-5")
```

## 1.2.2 Models

### 模型调用

```
1 response = model.invoke("为什么鹦鹉有五颜六色的羽毛? ")  
2 print(response)
```

单样本推理

```
1 responses = model.batch([  
2     "为什么鹦鹉有五颜六色的羽毛? ",  
3     "飞机是如何飞行的? ",  
4     "什么是量子计算? "  
5 ])  
6 for response in responses:  
7     print(response)
```

批量推理

# 1.2.3 Messages

## 模型上下文的基本单位

01

### 角色

标识消息类型（例如 system、user）

02

### 内容

表示消息的实际内容（例如文本、图像、音频、文档等）

03

### 元数据

可选字段，例如响应信息、消息 ID 和令牌使用情况

LangChain 提供了一种标准消息类型，可在所有模型提供商之间工作，确保无论调用哪个模型都能保持一致的行为

# 1.2.3 Messages

---

## 消息类型

SystemMessage: 告诉模型如何行为并为交互提供上下文

HumanMessage: 表示用户输入和与模型的交互

AIMessage: 模型生成的响应, 包括文本内容、工具调用和元数据

ToolMessage: 表示工具调用的输出

# 1.2.3 Messages

---

## SystemMessage

```
1  from langchain.messages import SystemMessage, HumanMessage
2
3  system_msg = SystemMessage("""
4  You are a senior Python developer with expertise in web frameworks.
5  Always provide code examples and explain your reasoning.
6  Be concise but thorough in your explanations.
7  """)
8
9  messages = [
10     system_msg,
11     HumanMessage("How do I create a REST API?")
12 ]
13 response = model.invoke(messages)
```

# 1.2.3 Messages

## HumanMessage

文本方式

```
1 response = model.invoke([  
2     HumanMessage("What is machine learning?")  
3 ])
```

```
1 # 使用字符串是单个 HumanMessage 的快捷方式  
2 response = model.invoke("What is machine learning?")
```

文本+元消息

```
human_msg = HumanMessage(  
    content="Hello!",  
    name="alice", # 可选: 标识不同用户  
    id="msg_123", # 可选: 用于追踪的唯一标识符  
)
```

# 1.2.3 Messages

## ToolMessage

```
1 # 模型进行工具调用后
2 ai_message = AIMessage(
3     content=[],
4     tool_calls=[{
5         "name": "get_weather",
6         "args": {"location": "San Francisco"},
7         "id": "call_123"
8     }]
9 )
10
11 # 执行工具并创建结果消息
12 weather_result = "Sunny, 72°F"
13 tool_message = ToolMessage(
14     content=weather_result,
15     tool_call_id="call_123" # 必须匹配调用 ID
16 )
17
18 # 继续对话
19 messages = [
20     HumanMessage("What's the weather in San Francisco?"),
21     ai_message, # 模型的工具调用
22     tool_message, # 工具执行结果
23 ]
24 response = model.invoke(messages) # 模型处理结果
```

## 1.2.3 Tools

### 创建

```
1 from langchain.tools import tool
2
3 @tool
4 def search_database(query: str, limit: int = 10) -> str:
5     """Search the customer database for records matching the query.
6
7     Args:
8         query: Search terms to look for
9         limit: Maximum number of results to return
10    """
11    return f"Found {limit} results for '{query}'"
```

名称

工具描述

## 1.2.3 Tools

### 设计原则

- 职责单一
- 参数设计清晰
- 提供足够的描述
- 合理使用枚举

✗ 不好: `process_order(action, ...)` # 可创建、更新、取消  
✓ 好: `create_order(...)`、`update_order(...)`、`cancel_order(...)`

✗ 不好: `date="20260301"` # 字符串, 格式不清  
✓ 好: `date="2026-03-01"` # ISO 格式, 有说明

```
{  
  "name": "calculate_shipping",  
  "description": "计算订单的运费。当用户询问运费或下单时需要计算  
运费时调用。需要提供收货地址和商品重量。"  
} ✓  
  
{  
  "priority": {  
    "type": "string",  
    "enum": ["low", "medium", "high"],  
    "description": "任务优先级"  
  }  
} ✓
```

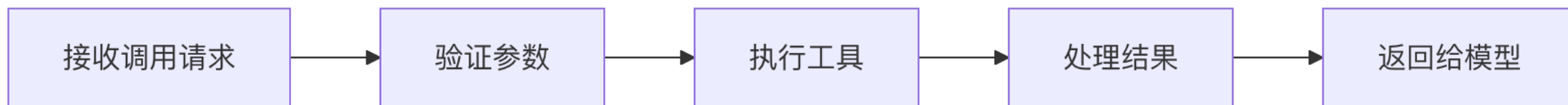
## 1.2.3 Tools

### 常见误区

•**误区一**：提供越多工具越好 正解：工具过多会增加选择复杂度和Token 开销，应按需动态加载

•**误区二**：模型会准确调用工具 正解：需要充分测试和优化工具定义，模型可能误用工具

•**误区三**：直接返回原始工具结果 正解：应该格式化和适当压缩，便于模型理解和使用



## 1.2.4 Memory

### 形式：历史对话形式

```
from langchain.agents import create_agent
from langgraph.checkpoint.memory import InMemorySaver # [!code highlight]

agent = create_agent(
    "openai:gpt-5",
    [get_user_info],
    checkpointer=InMemorySaver(), # [!code highlight]
)

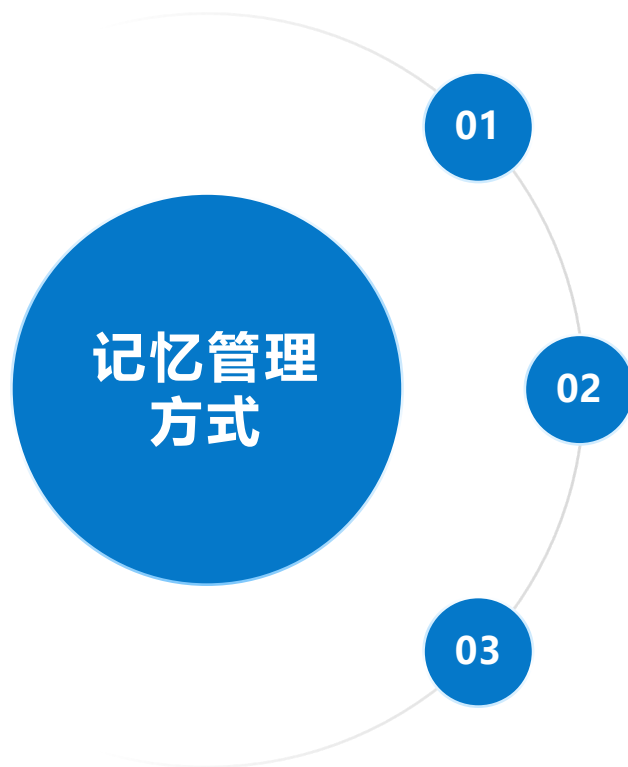
agent.invoke(
    {"messages": [{"role": "user", "content": "Hi! My name is Bob."}]},
    {"configurable": {"thread_id": "1"}}, # [!code highlight]
)
```



通过InMemorySaver进行  
管理

# 1.2.4 Memory

## 管理方式：修剪、删除、压缩



### 修剪

移除最初或最后的 N 条消息（在调用 LLM 之前）。

### 从会话状态中永久删除消息。

从 LangGraph 状态中永久删除消息。

### 压缩

总结历史记录中较早的消息，并用摘要替换它们。

# 1.2.4 Memory

## 删除信息

```
from langchain.messages import RemoveMessage # [!code highlight]


def delete_messages(state):
    messages = state["messages"]
    if len(messages) > 2:
        # remove the earliest two messages
        return {"messages": [RemoveMessage(id=m.id) for m in messages[:2]]}
```



清除特定信息

```
from langgraph.graph.message import REMOVE_ALL_MESSAGES # [!code highlight]

def delete_messages(state):
    return {"messages": [RemoveMessage(id=REMOVE_ALL_MESSAGES)]}
```



清除全部信息

# 1.2.4 Memory

## 摘要

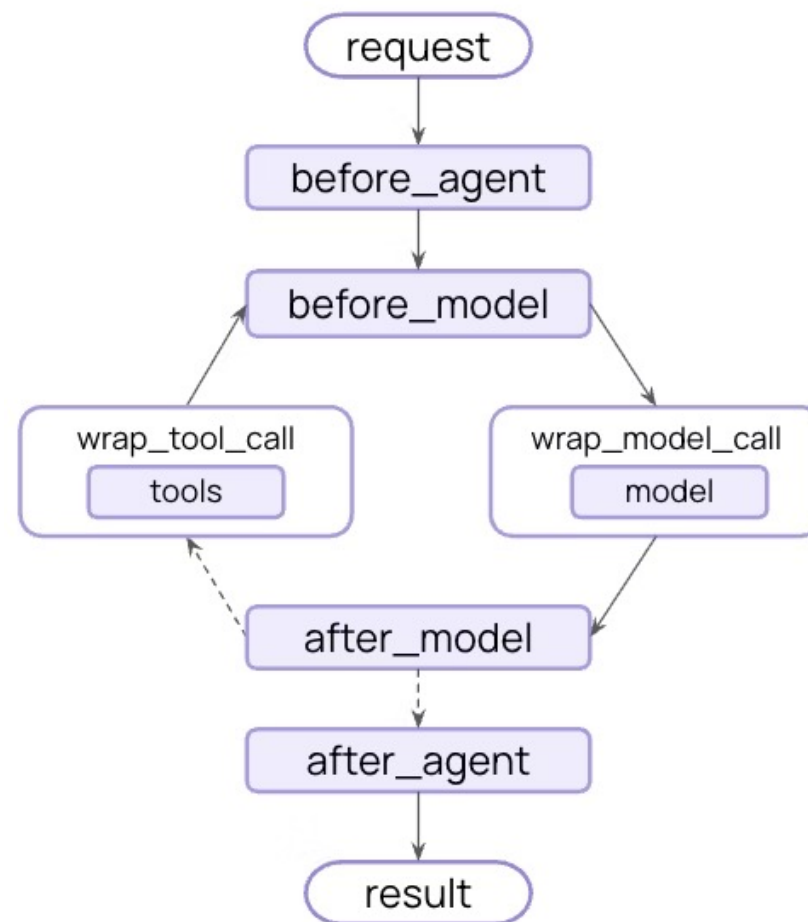
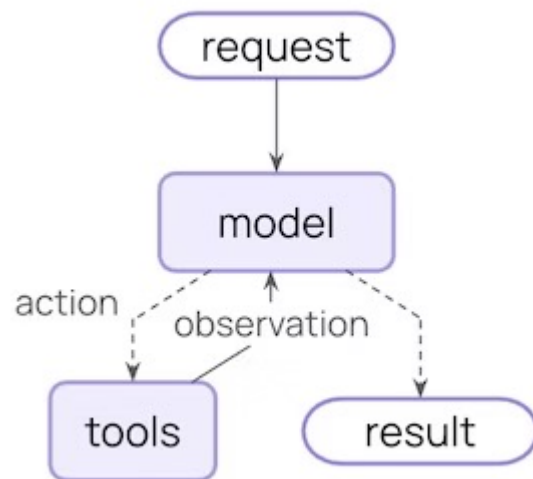
```
checkpointer = InMemorySaver()

agent = create_agent(
    model="openai:gpt-4o",
    tools=[],
    middleware=[
        SummarizationMiddleware(
            model="openai:gpt-4o-mini",
            max_tokens_before_summary=4000, # Trigger summarization at 4000 tokens
            messages_to_keep=20, # Keep last 20 messages after summary
        )
    ],
    checkpointer=checkpointer,
)
```



# 1.2.5 中间件

作用：细粒度控制Agent



# 1.2.5 中间件

## Human in the loop

在工具执行之前，暂停代理执行以供人工批准、编辑或拒绝工具调用。

```
agent = create_agent(  
    model="openai:gpt-4o",  
    tools=[read_email_tool, send_email_tool],  
    checkpointer=InMemorySaver(),  
    middleware=[  
        HumanInTheLoopMiddleware(  
            interrupt_on={  
                # 要求对发送邮件进行批准、编辑或拒绝  
                "send_email_tool": {  
                    "allowed_decisions": ["approve", "edit", "reject"],  
                },  
                # 自动批准读取邮件  
                "read_email_tool": False,  
            }  
        ),  
    ],  
)
```



# 1.2.5 中间件

## 限制模型调用

限制模型调用的次数，以防止无限循环或成本过高。

```
agent = create_agent(  
    model="openai:gpt-4o",  
    tools=[...],  
    middleware=[  
        ModelCallLimitMiddleware(  
            thread_limit=10, # 每个线程（跨多次运行）最多 10 次调用  
            run_limit=5, # 每次运行（单次调用）最多 5 次调用  
            exit_behavior="end", # 或者 "error" 以引发异常  
        ),  
    ],  
    1,  
)
```



# 1.2.5 中间件

## 限制工具调用调用

限制特定工具或所有工具的调用次数。

```
# 限制所有工具调用
global_limiter = ToolCallLimitMiddleware(thread_limit=20, run_limit=10)

# 限制特定工具
search_limiter = ToolCallLimitMiddleware(
    tool_name="search",
    thread_limit=5,
    run_limit=3,
)

agent = create_agent(
    model="openai:gpt-4o",
    tools=[...],
    middleware=[global_limiter, search_limiter],
)
```



# 1.2.5 中间件

## 模型回退

当主要模型失败时，自动回退到替代模型。

```
agent = create_agent(  
    model="openai:gpt-4o", # 主要模型  
    tools=[...],  
    middleware=[  
        ModelFallbackMiddleware(  
            "openai:gpt-4o-mini", # 错误时首先尝试  
            "anthropic:claude-3-5-sonnet-20241022", # 然后尝试这个  
        ),  
    ],  
)
```



# 1.2.5 中间件

## 个人信息检测

### 检测和处理对话中的个人身份信息

```
agent = create_agent(  
    model="openai:gpt-4o",  
    tools=[...],  
    middleware=[  
        # 涂改用户输入中的电子邮件  
        PIIMiddleware("email", strategy="redact", apply_to_input=True),  
        # 掩盖信用卡 (显示后 4 位)  
        PIIMiddleware("credit_card", strategy="mask", apply_to_input=True),  
        # 带有正则表达式的自定义 PII 类型  
        PIIMiddleware(  
            "api_key",  
            detector=r"sk-[a-zA-Z0-9]{32}",  
            strategy="block", # 如果检测到, 则引发错误  
        ),  
    ],  
)
```



# 1.2.5 中间件

## 自定义中间件

### 可以通过修饰器构建中间件

节点式 (Node-style) (在特定的执行点运行):

- `@before_agent` - 代理启动前 (每次调用一次)
- `@before_model` ↗ - 每次模型调用前
- `@after_model` ↗ - 每次模型响应后
- `@after_agent` - 代理完成时 (每次调用最多一次)

包装式 (Wrap-style) (拦截并控制执行):

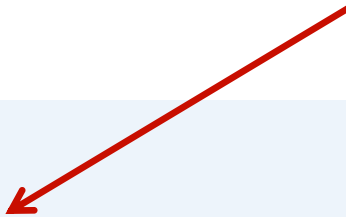
- `@wrap_model_call` ↗ - 每次模型调用周围
- `@wrap_tool_call` ↗ - 每次工具调用周围

## 1.2.6 结构化输出

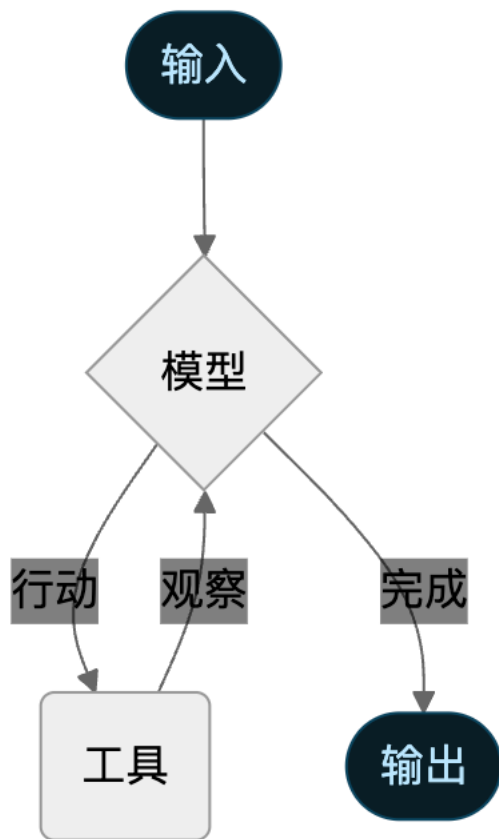
### 作用

以特定的、可预测的格式返回数据

```
def create_agent(  
    ...  
    response_format: Union[  
        ToolStrategy[StructuredResponseT],  
        ProviderStrategy[StructuredResponseT],  
        type[StructuredResponseT],  
    ]
```



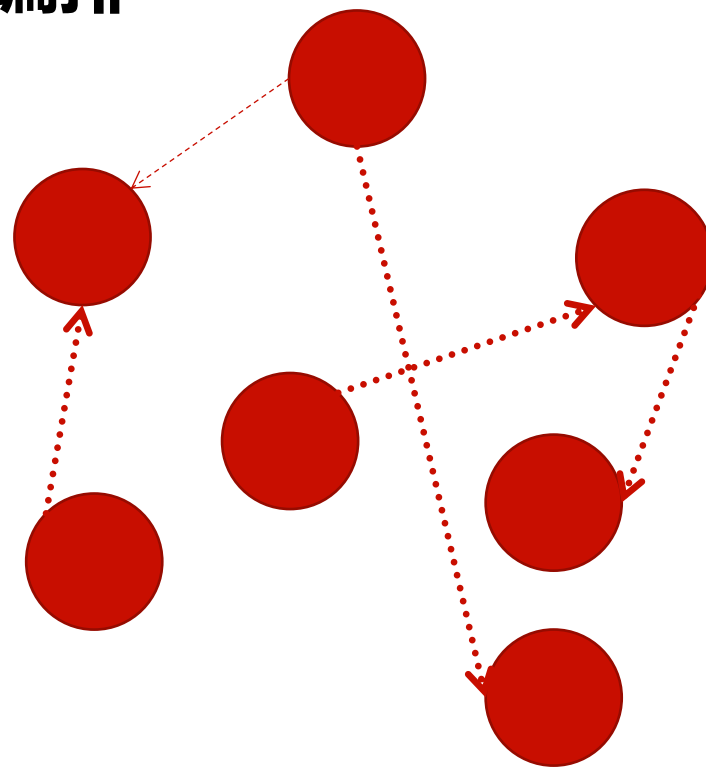
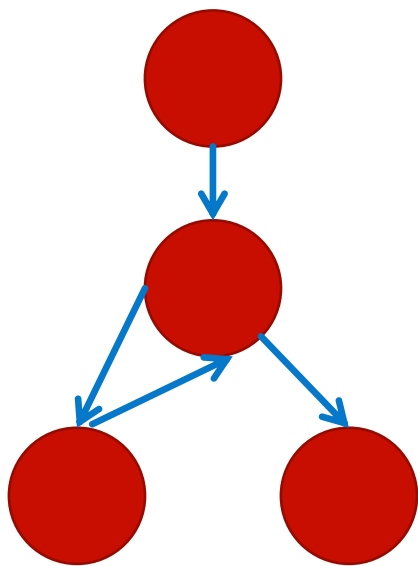
## 1.2.6 编排



```
1 from langchain.agents import create_agent
2
3 weather_agent = create_agent(
4     model="openai:gpt-5",
5     tools=[get_weather],
6     system_prompt="Help the user by fetching the weather in their city.",
7 )
8
9 result = agent.invoke({"role": "user", "what's the weather in SF?"})
```

## 2.1 Langgraph

定位：langchain的基座，专注于代理编排



# 2.1 LangChain VS LangGraph

## LangChain

- 构建遵循 模型→工具→响应 的智能体
- 快速交付

## LangGraph

- 需要更多监督或人工介入的敏感 workflows
- 高度定制或复杂的工作流
- 需要仔细控制延迟和/或成本的应用程序
- 包含确定性与智能体组件的混合 workflows

LangChain 和 LangGraph的选用



# 目 录

**1** LangChain / LangGraph

---

**2** AgentScope

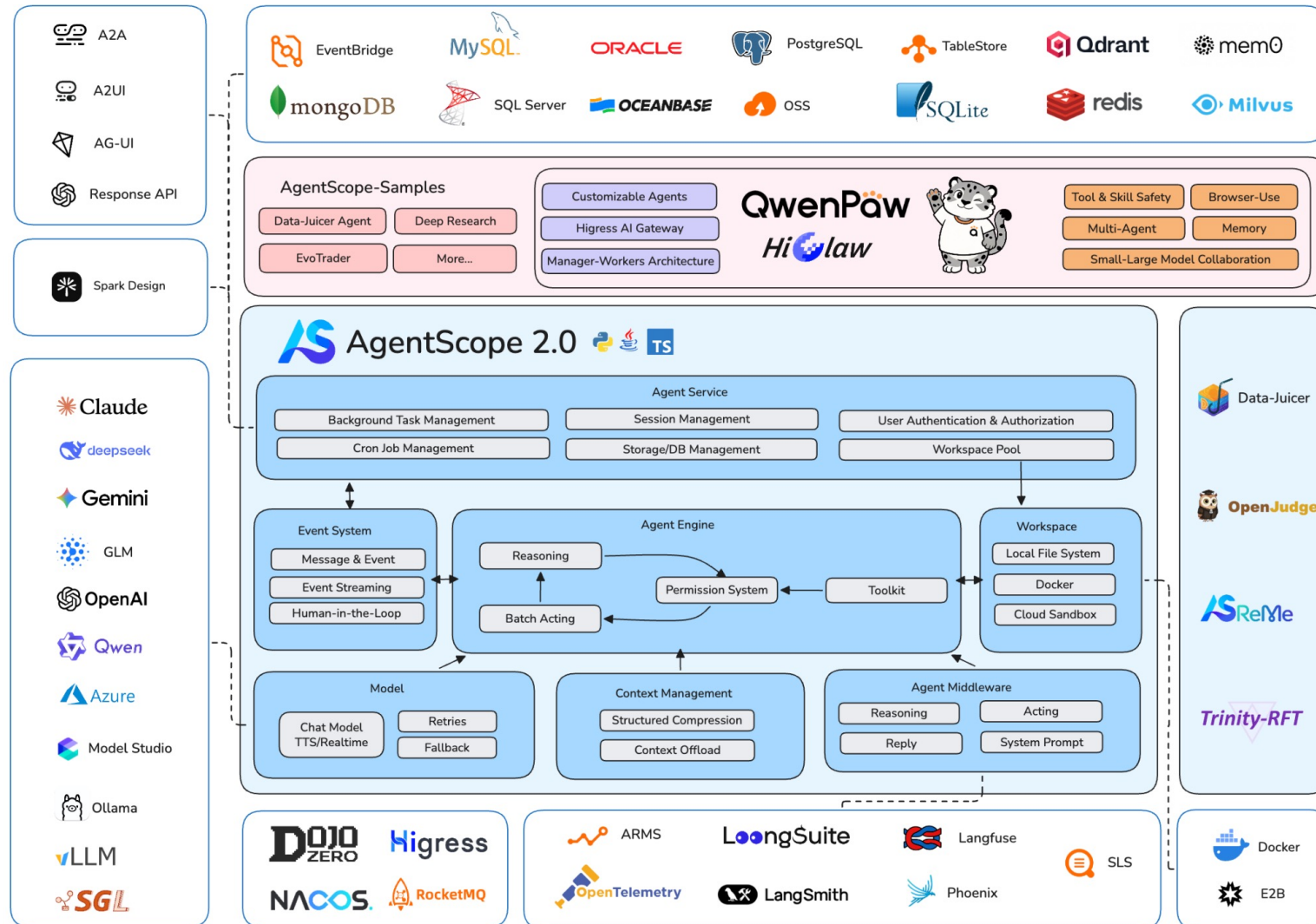
---

**3**

---

---

# 3.1 AgentScope 介绍



## 3.2 AgentScope模块

### 核心模块

#### 消息与事件

类似LangChain Messages

#### 智能体

类似LangChain, 流程更复杂

#### Model

类似LangChain

#### Context

管理 agent 的上下文窗口, 让长任务稳定运行。类似LangChain Memory

# 3.1 AgentScope

## 核心模块

### Tool

除了Tools, 对MCP、Skills做了更多的支持

### 权限管理

精细控制 agent 可以执行哪些 tool、何时执行

### 中间件

类似LangChain

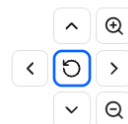
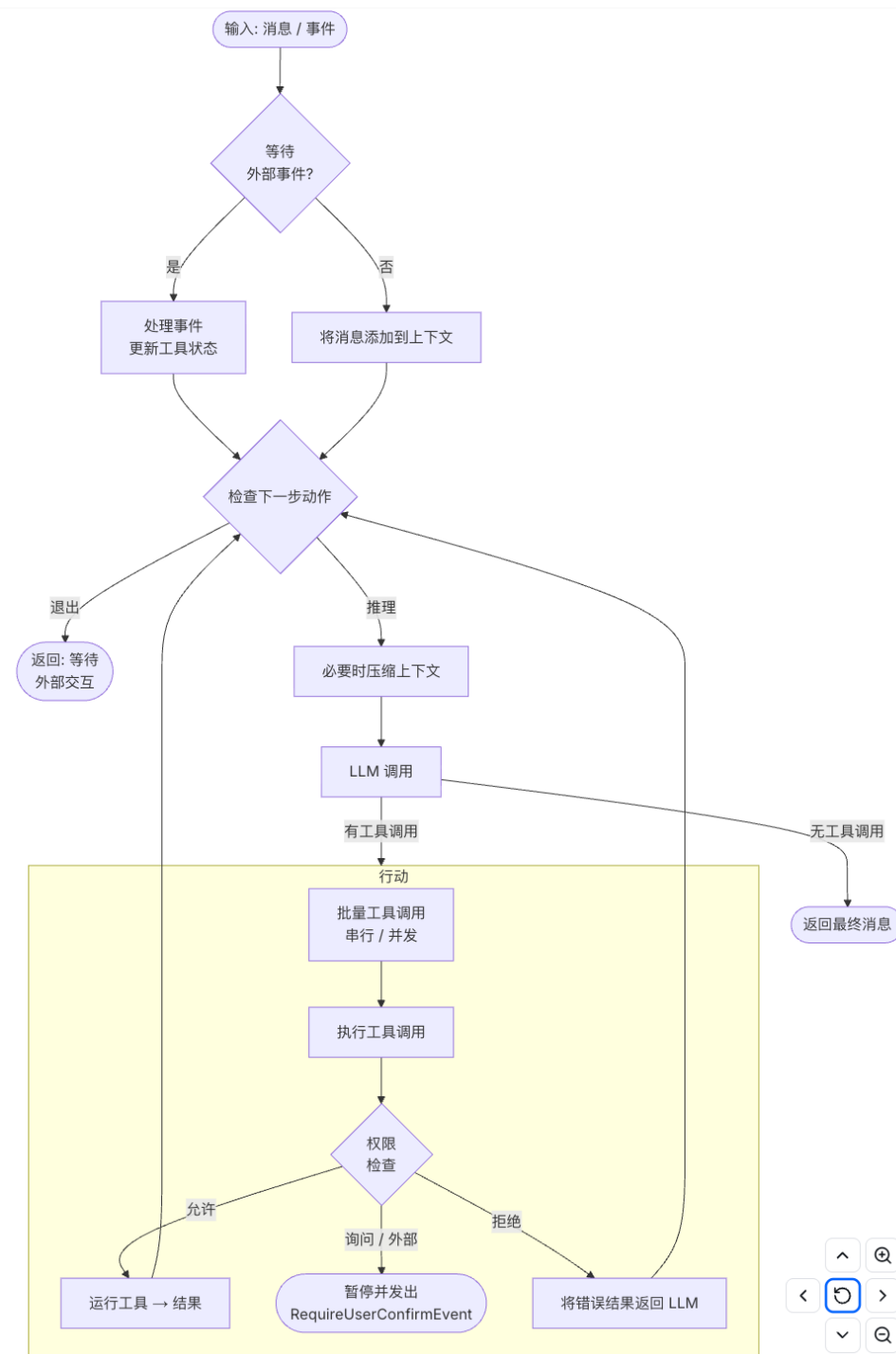
### Workspace

智能体执行环境

# 3.2.1 Agents

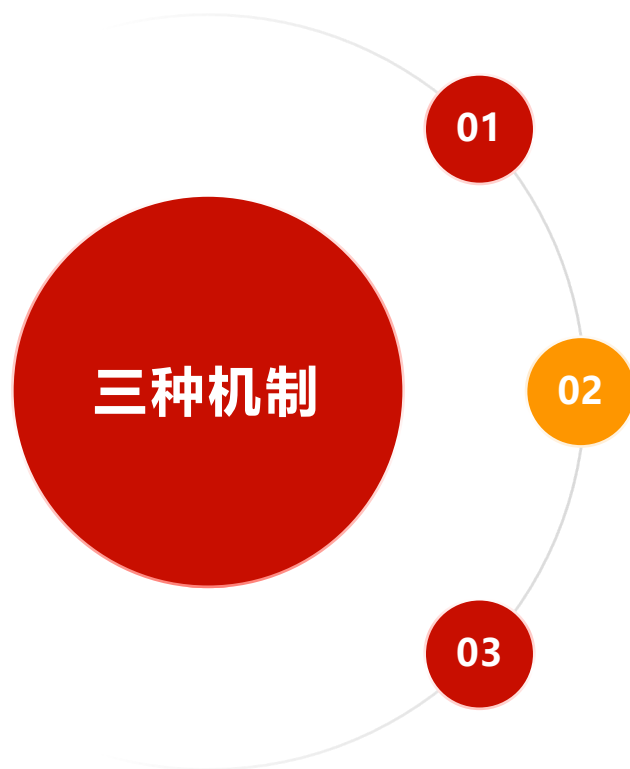
## 核心接口

- replay () 运行推理-行动循环，返回最终 Msg。如右图



## 3.2.2 Context (Memory)

**定位：类似langchain memory**



### 上下文压缩

当 token 用量逼近模型上限时，把较早的消息汇总成一段摘要。

### 工具结果截断

在过大的工具输出进入上下文之前先截断。

### 上下文offload

把已经从上下文中移除的内容持久化到外部存储，供 agent 后续检索。

## 3.2.2 Context (Memory)

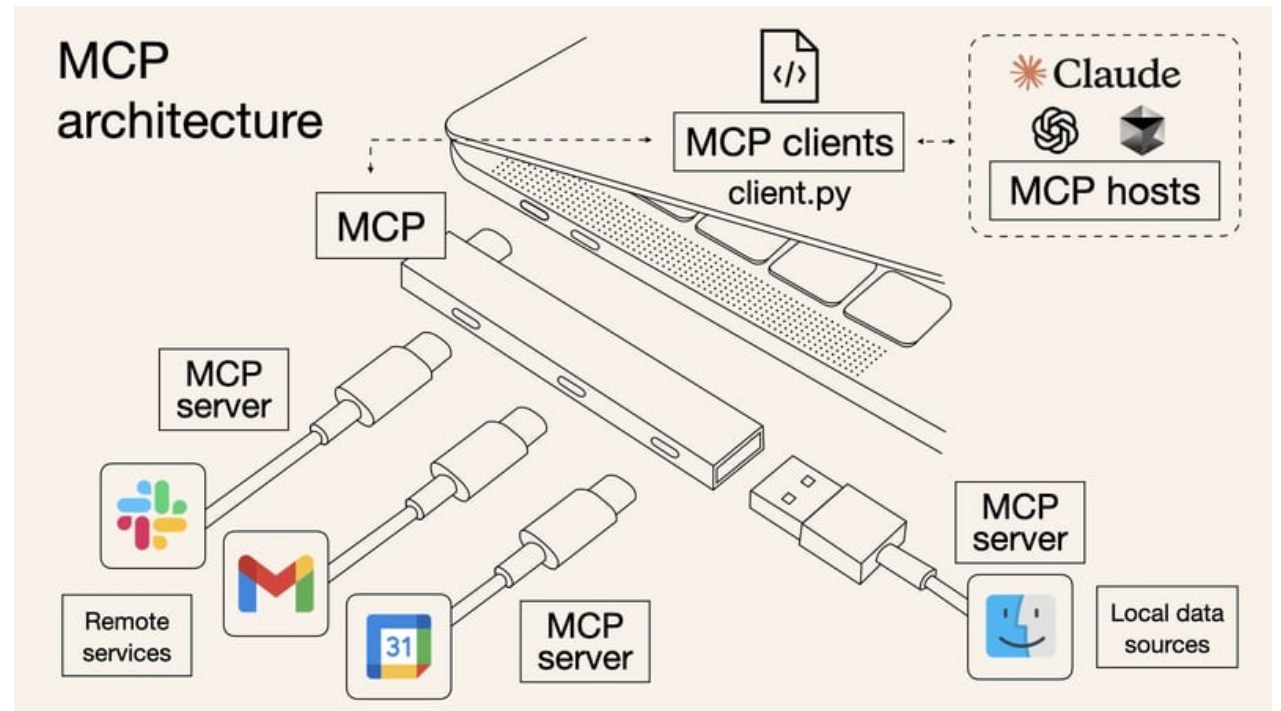
压缩：每次推理步骤前自动执行



# 3.2.3 Tools

## MCP

```
client = MCPClient(  
    name="filesystem",  
    is_stateful=True,  
    mcp_config=StdioMCPConfig(  
        command="mcp-server-filesystem",  
        args=["--root", "/my/project"],  
    ),  
)  
  
await client.connect()  
  
toolkit = Toolkit(mcps=[client])
```



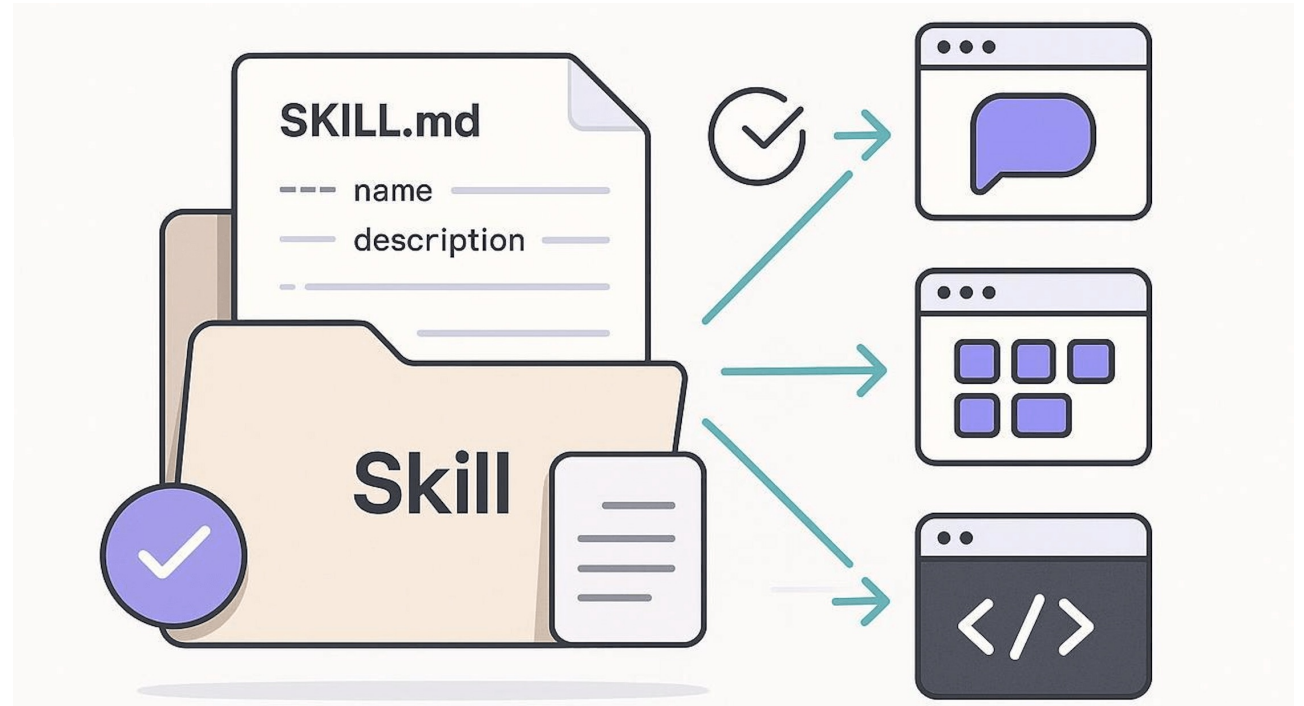
## 3.2.3 Tools

### Skills

```
from agentscope.tool import Toolkit

toolkit = Toolkit(
    skills_or_loaders=["/path/to/skills"],
)
```

<https://skillsmp.com/zh>



## 3.2.3 Tools

### Skills

skillmap上的  
“代码检  
查” skill

~/skillsmp |

搜索

职业技能

```
$ pwd: ~ / occupations / ... / 软件质量保证分析师与测试员 / verify-quality
```

### verify-quality

// 代码质量校验关卡。检测复杂度、重复代码、命名规范、函数长度等质量指标。当用户提到代码质量、复杂度检查、代码异味、重构建议、lint检查、代码规范时使用。在复杂模块、重构完成时自动触发。

在 Manus 中运行

```
$ git log --oneline --stat
```

☆ stars: 5,319    🔗 forks: 410    📅 updated: 2026年3月9日 07:06

文件资源管理器 2 个文件

> 文件夹 scripts

SKILL.md

3.5 KB

## 3.2.4 权限管理

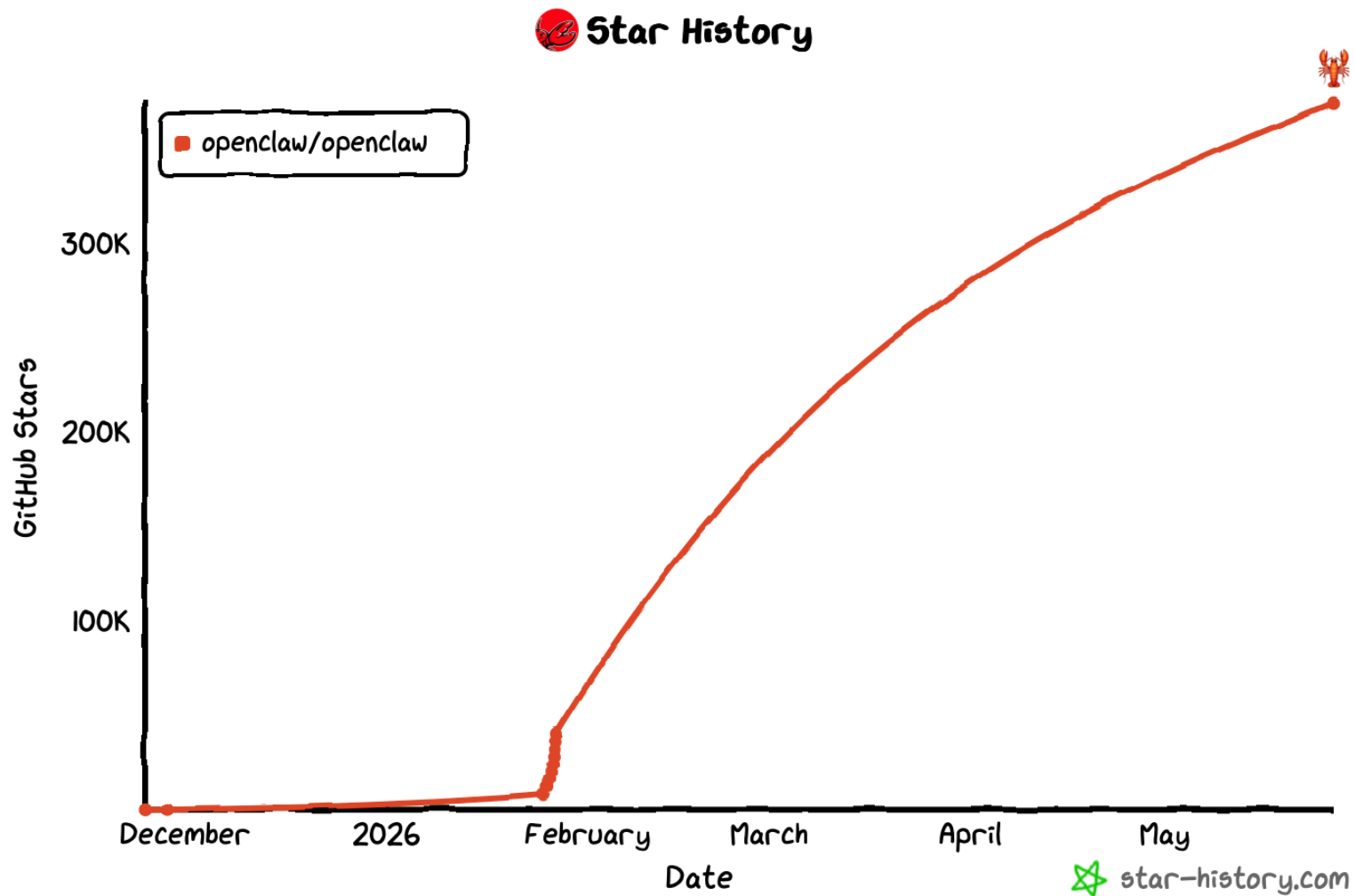
Mode	行为	适用场景
DEFAULT	所有操作都需要显式规则或用户确认	最安全，推荐默认值
ACCEPT_EDITS	自动放行工作目录内的文件操作	用户在场的活跃开发
EXPLORE	只读：放行读、拒绝所有写与命令	代码探索、规划
BYPASS	放行一切（deny / ask 规则仍生效）	完全可信的沙箱
DONT_ASK	把所有 ASK 转为 DENY	无人值守 / 计划任务



# 目 录

- 1 LangChain / LangGraph
- 2 AgentScope
- 3 OpenClaw

# 龙虾来了





# OpenClaw

THE AI THAT ACTUALLY DOES THINGS.

Clears your inbox, sends emails, manages your calendar, checks you in for flights.  
All from WhatsApp, Telegram, or any chat app you already use.

**NEW** Where OpenClaw Security Is Heading →

## > What People Say

[View all →](#)

...been saying for like six months that even if  
...suddenly stopped improving, we could  
...d \*years\* discovering new transformative...  
**@kjaquith**



"Why @openclaw is nuts: your context and skills  
live on YOUR computer, not a walled garden. It's  
open source. Growing community building skill...  
**@danpeguine**



"Yeah this  
subscription  
Codex se  
**@natelias**

...ture is now old man: @openclaw!"  
**@**



"AGI is a lobster, it will never sink!"  
**@FrancisBrero**



"Feels like we're living in the fut  
**@Philo01**

# ‘OpenClaw: Agent 的极致开源实践’

**(LLM 推理层)**

## 可插拔模型推理

OpenAI Claude Gemini Ollama 本地模型

随时切换、本地运行

**(Memory)**

## 持久化记忆

文本 + 向量存储

Markdown 对话记忆 向量数据库 (like LanceDB)

跨会话延续

混合检索 语义匹配 精确提取实体

**(RAG & MCP)**

### 本地知识库 (RAG)

本地文档库 本地向量库

私人资料向量化, 精准检索, 数据不上云

### MCP 工具调用

Shell 命令 读写文件 浏览器操控 (Playwright) 日历邮件管理

原生 MCP 协议, 沙箱环境运行

沙箱 Docker

**(Skills)**

## 技能规划

技能 步骤1 步骤2 步骤2 步骤1

安装技能 ClawHub

结构化 workflow 封装, 按需调用

**GitHub**  
星标增长最快  
(Short Time, 24万+)

**OpenClaw**  
核心融合

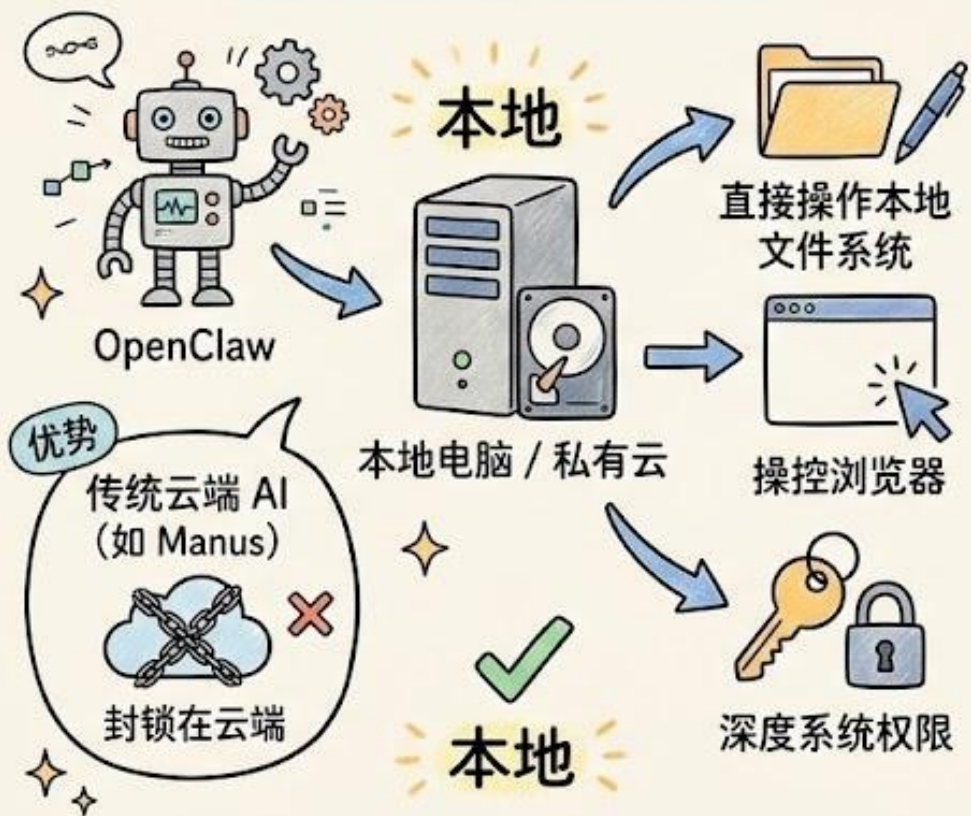
当下 Agent  
终极形态

知乎 @ConardLi

# OpenClaw 的两大核心特性

优势

## 完全的本地控制权



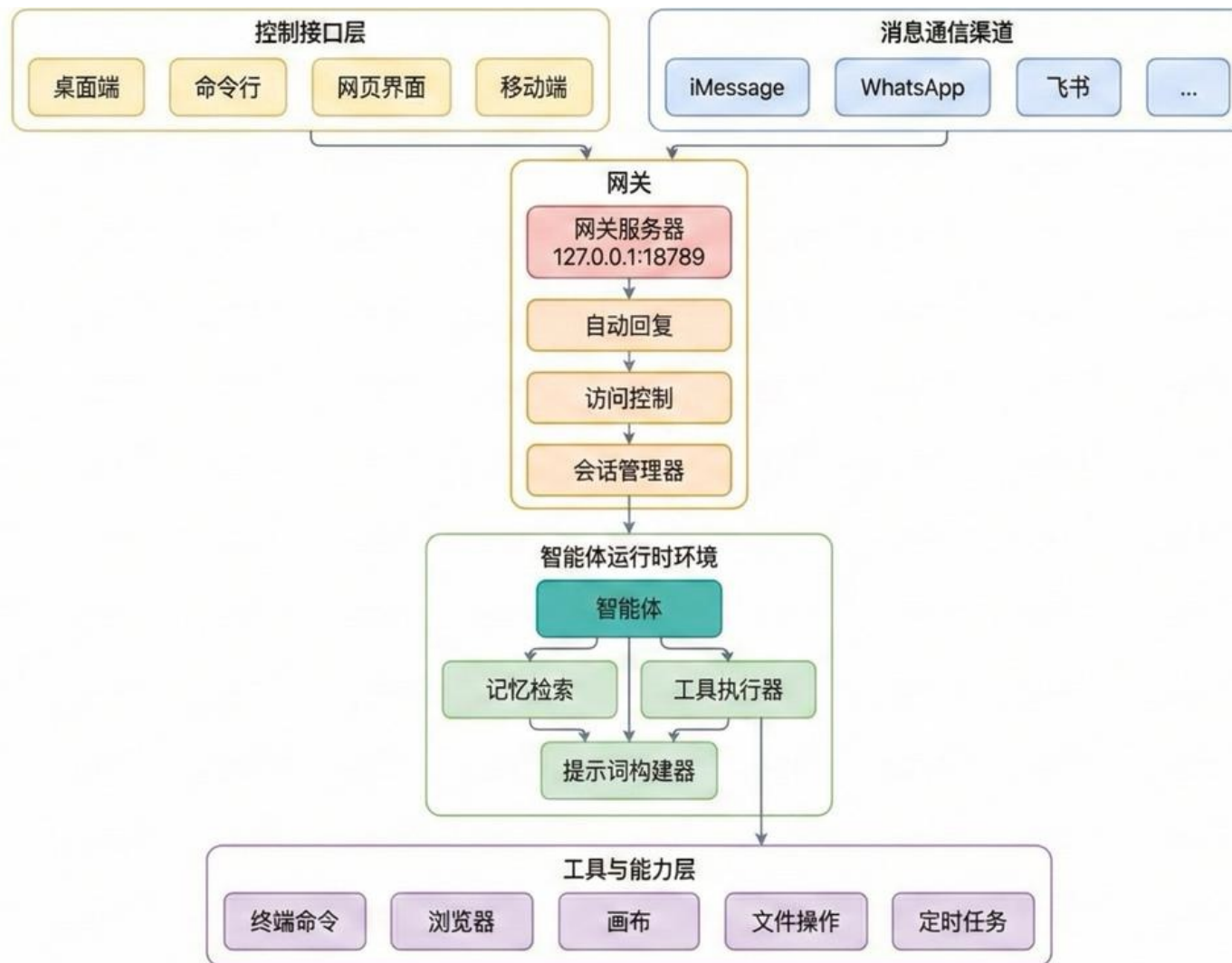
## 无缝的交互入口



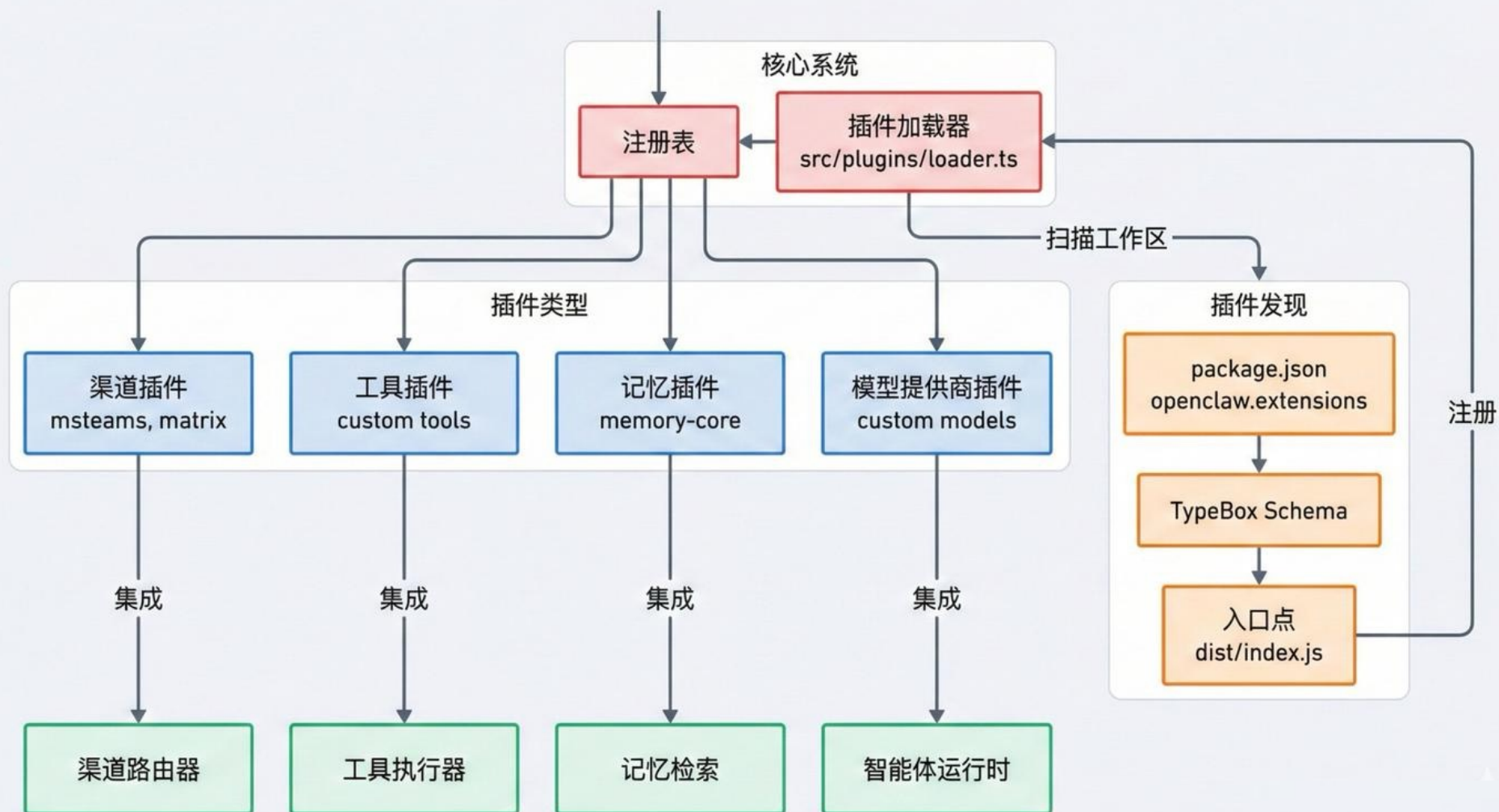
# 安全性不够



# 整体架构



# 通过插件实现可扩展性



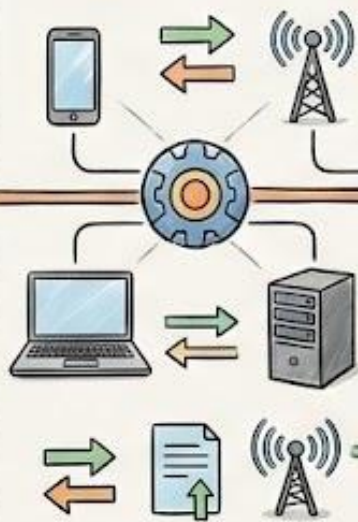
# OpenClaw 中的 Gateway: 核心中枢与守护进程

理论定位:  
后台守护进程  
(Daemon)



控制中心  
管理 AI 生命周期  
长期稳定在线

统一接入与  
协议处理



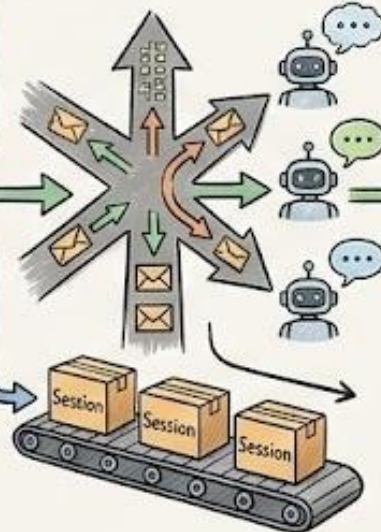
WebSocket 主控制协议(双向)  
HTTP 服务 (Hooks, 集成)  
Standardization 屏蔽差异

身份认证与  
安全边界管控



用户/设备 认证  
工具调用沙箱策略  
白名单 & 权限

消息路由与  
会话管理

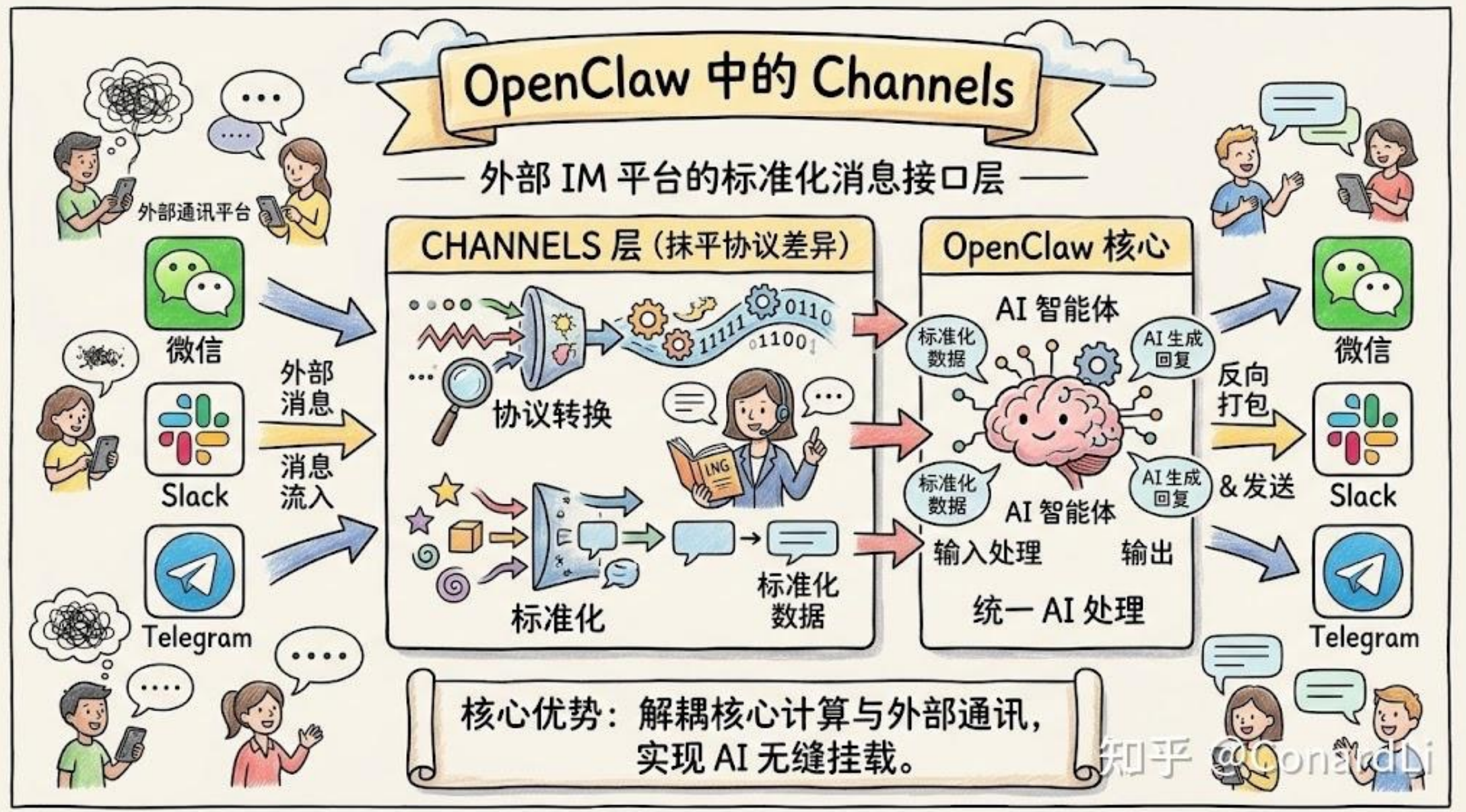


匹配 Agent 实例  
并发控制 (队列)  
上下文状态持久化

全局状态管理  
与广播



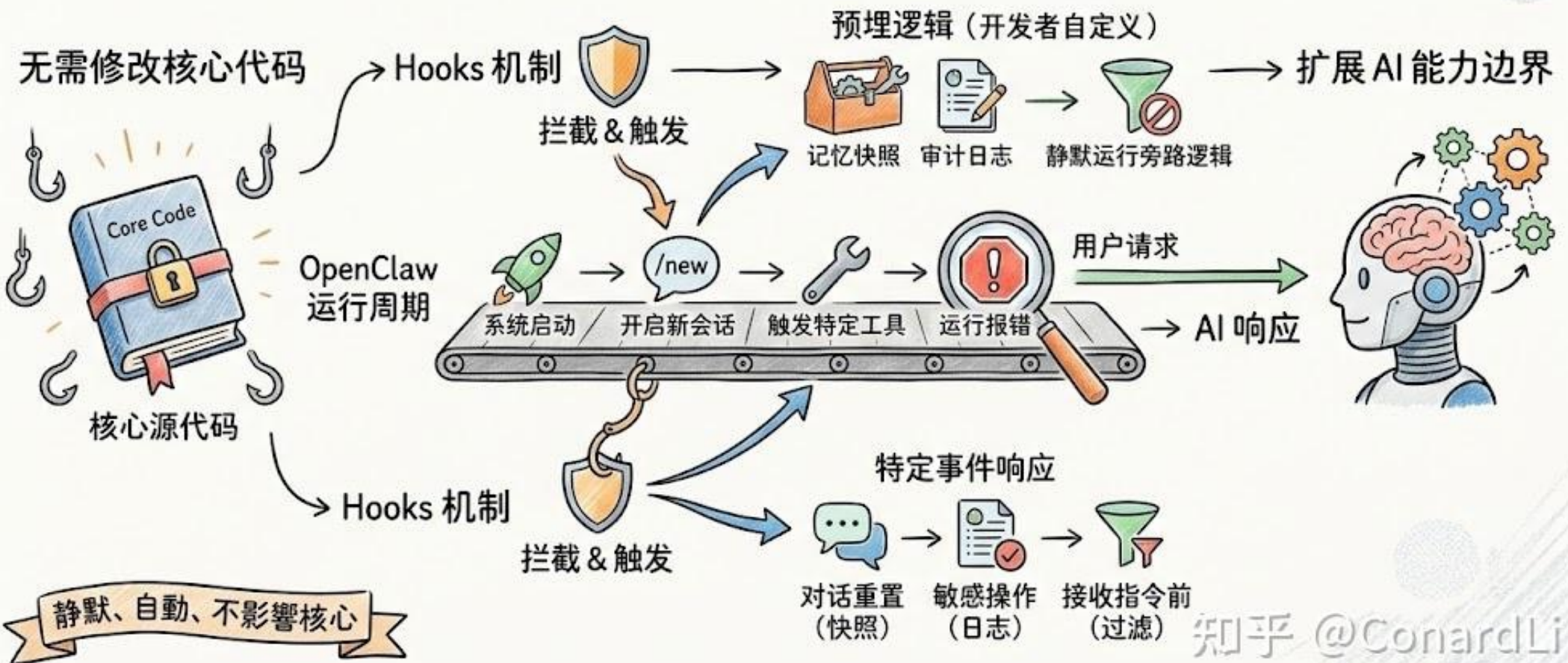
实时执行 & 任务广播  
跨终端状态同步  
用户体验一致



知乎 @Gonzali

# OpenClaw 中的 Hooks

事件驱动的底层自动化扩展机制



知乎 @ConardLi

# OpenClaw 中的 Cron Job: 基于时间的自动化任务调度

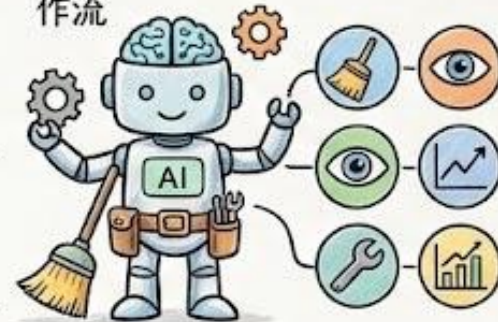
① 被动响应 (常规 AI)  
必须依赖 Prompt 唤醒才能工作



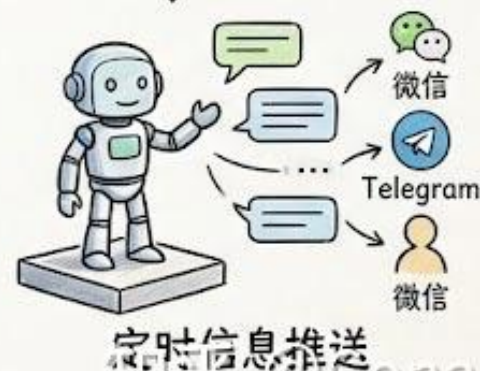
② Cron Job 打破限制  
每天早晨 8 点  
固定时间频率



③ 主动执行 (Cron Job)  
脱离人类即时指令, 主动执行工作流



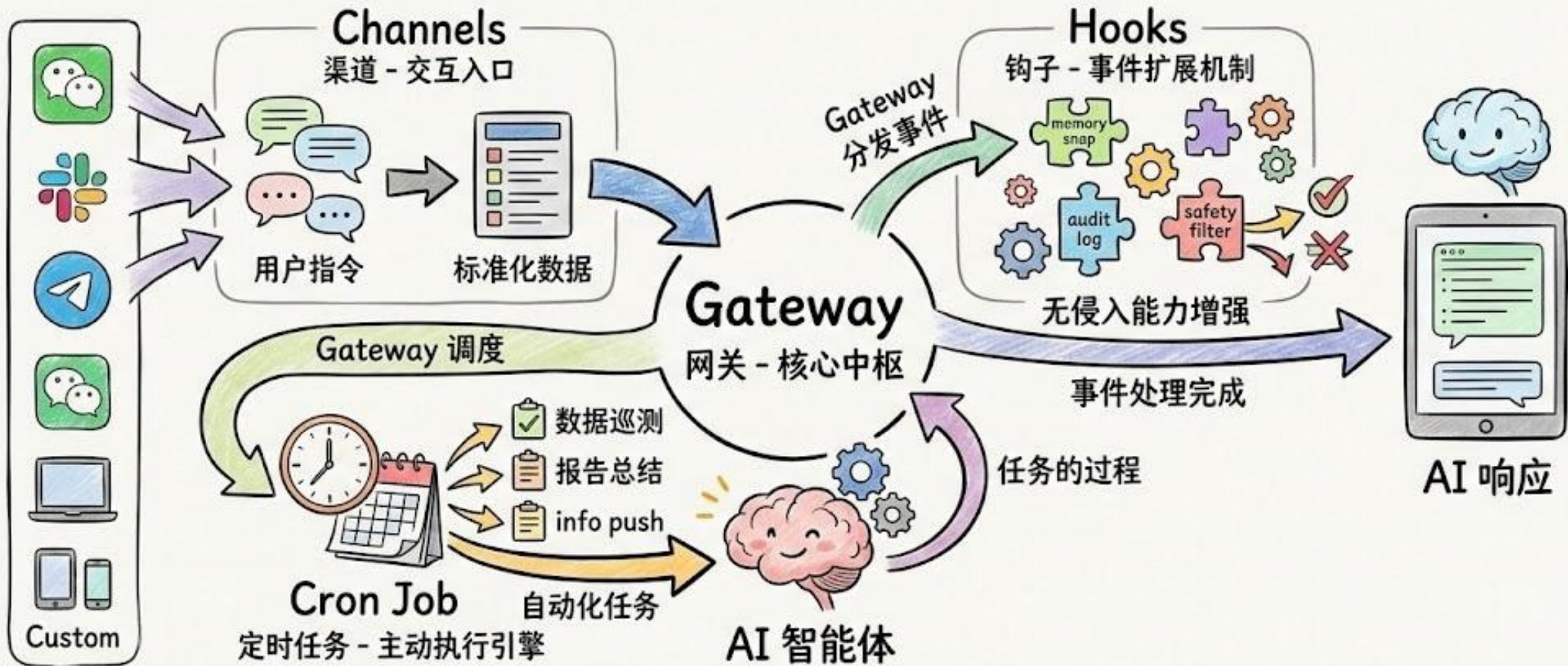
④ 实现功能 (示例)



让 AI 更像一个具备自主性的“数字员工”

知乎 @ConardLi

# OpenClaw 四大核心概念协同关系



四大核心架构：协同构建个人 AI 智能体全流程能力（交互、调度、扩展、自动化）

# 记忆系统：OpenClaw 的灵魂

层级	文件	作用
身份层	SOUL.md	定义 AI 的性格、语调、行为边界
用户层	USER.md	用户的个人资料和偏好
操作层	AGENTS.md	操作指南、工作流程、能力边界
索引层	MEMORY.md	核心信息索引，保持精简
项目层	memory/projects.md	各项目当前状态与待办
基础设施层	memory/infra.md	服务器、API、部署配置速查
经验层	memory/lessons.md	踩过的坑，按严重程度分级
日志层	memory/YYYY-MM-DD.md	每日原始记录

# 安装&配置

---

安装 OpenClaw 需要 Node.js 22 以上的版本。

```
brew install node
```

推荐使用 npm 来安装 openclaw。

```
npm install -g openclaw@latest
```

接下来，我们运行 openclaw 提供的初始化向导来完成一些必要的配置，执行下面的命令：

```
openclaw onboard --install-daemon
```

# 配置

---

接下来，我们运行 openclaw 提供的初始化向导来完成一些必要的配置，执行下面的命令：

```
openclaw onboard --install-daemon
```

OpenClaw Control x +

127.0.0.1:18789/overview

OPENCLAW GATEWAY DASHBOARD

版本 2026.2.26 健康状况 正常

### 概览

网关状态、入口点和快速健康读取。

#### 网关访问

仪表盘连接的位置及其身份验证方式。

WebSocket URL	网关令牌
ws://127.0.0.1:18789	4c9ace27b324109b6d5720b934c
密码 (不存储)	默认会话密钥
system or shared password	agent:main:main

语言

简体中文 (简体中文)

[连接](#) [刷新](#) 点击连接以应用连接更改。

#### 快照

最新的网关握手信息。

状态	运行时间	刻度间隔
正常	3m	不适用

最后频道刷新

just now

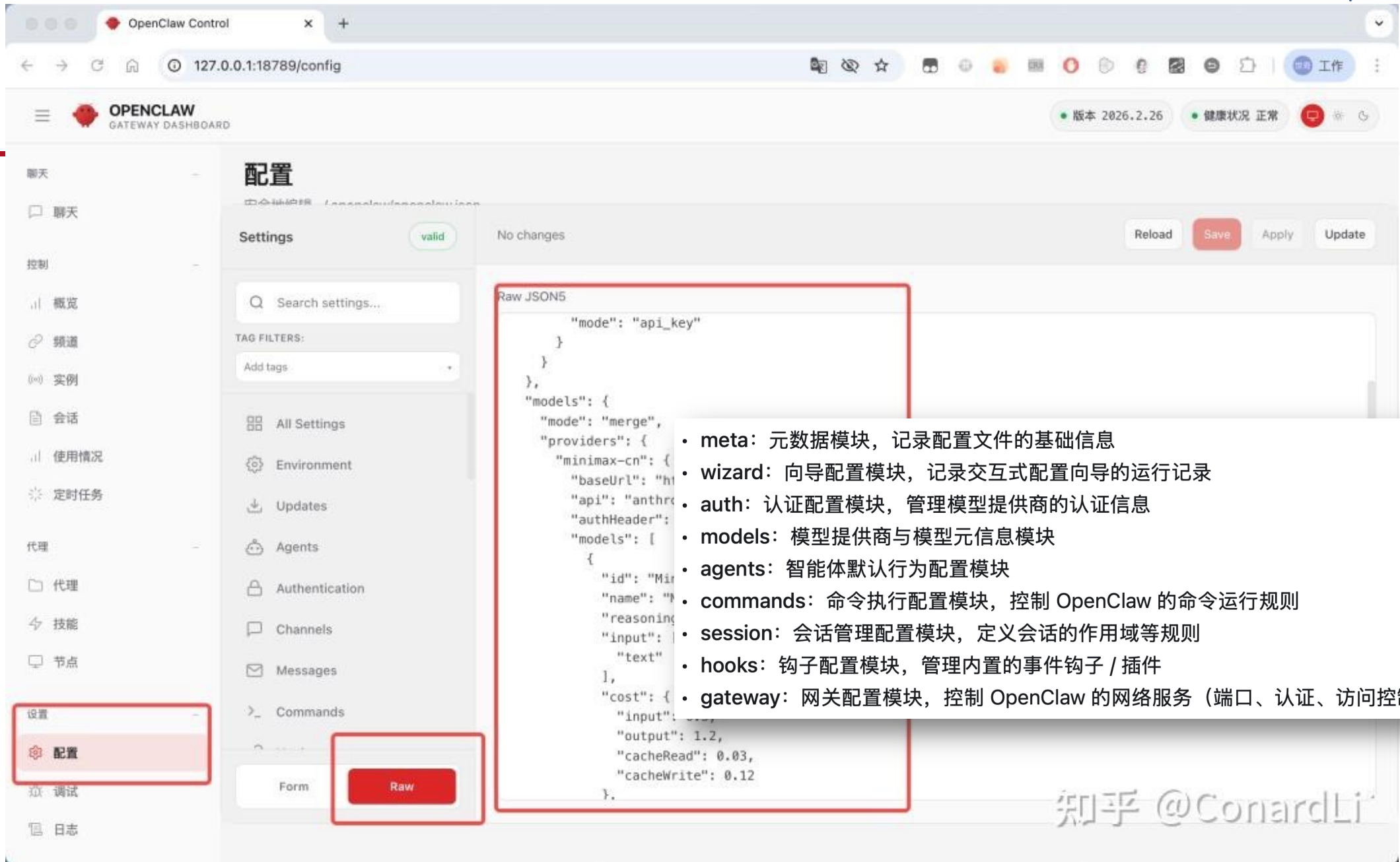
使用频道链接 WhatsApp、Telegram、Discord、Signal 或 iMessage。

实例	会话	定时任务
3	1	已启用
过去 5 分钟内的在线信号。	网关跟踪的最近会话密钥。	下次唤醒 n/a

#### 备注

远程控制设置的快速提醒。

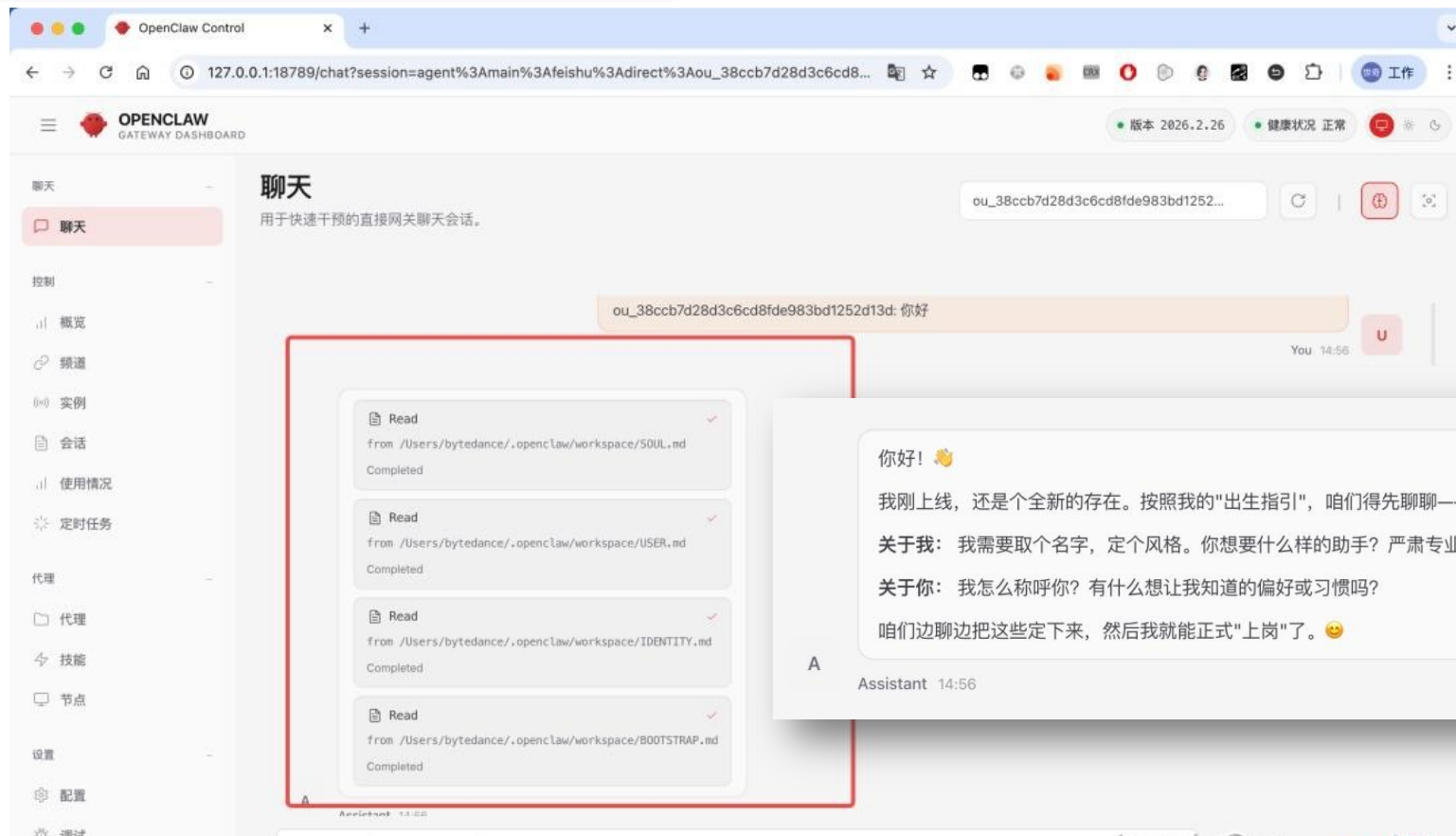
知乎 @ConardLi



- meta: 元数据模块, 记录配置文件的基础信息
- wizard: 向导配置模块, 记录交互式配置向导的运行记录
- auth: 认证配置模块, 管理模型提供商的认证信息
- models: 模型提供商与模型元信息模块
- agents: 智能体默认行为配置模块
- commands: 命令执行配置模块, 控制 OpenClaw 的命令运行规则
- session: 会话管理配置模块, 定义会话的作用域等规则
- hooks: 钩子配置模块, 管理内置的事件钩子 / 插件
- gateway: 网关配置模块, 控制 OpenClaw 的网络服务 (端口、认证、访问控制等)

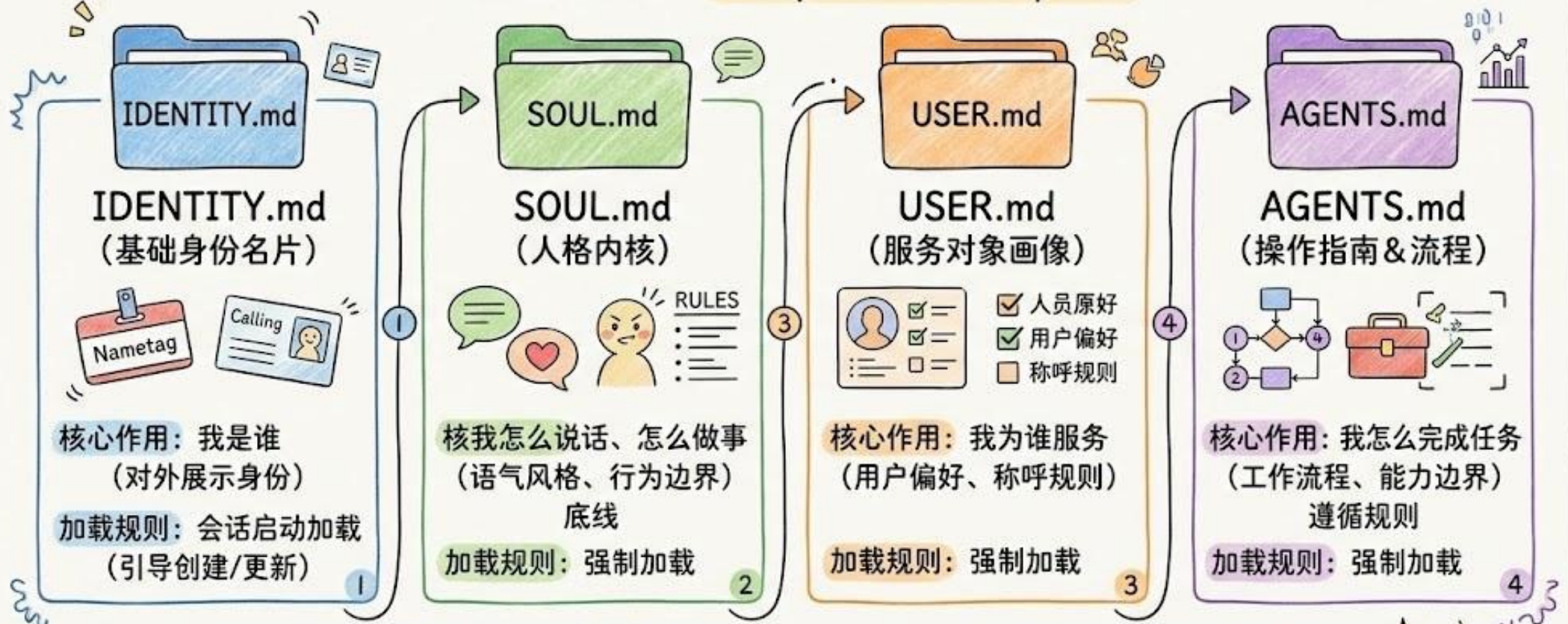
# 人设配置

我们看到，当我们发出第一句“你好”时，OpenClaw 首先去读取了这四个文件（SOUL.md、USER.md、IDENTITY.md、BOOTSTRAP.md）：



# OpenClaw 人设体系四大核心文件

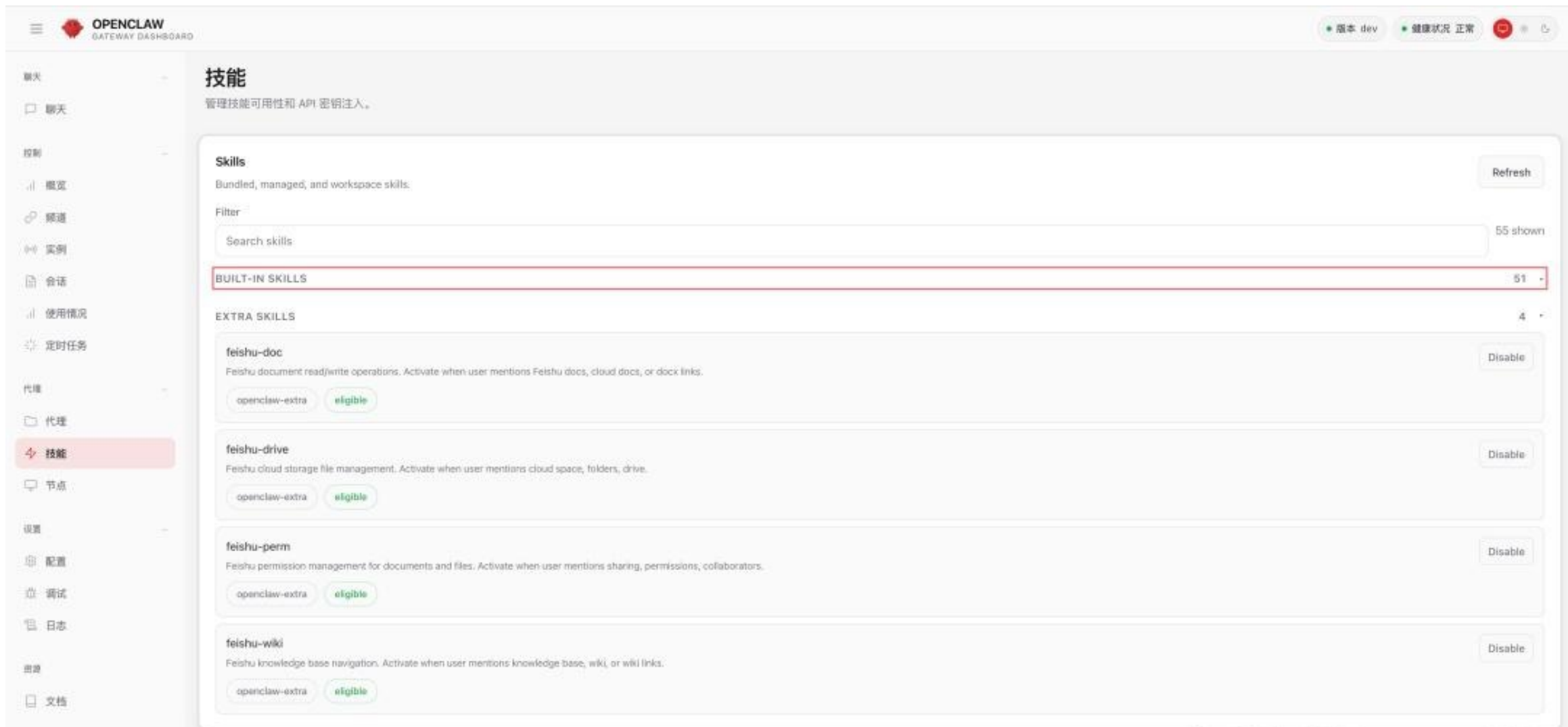
工作区默认路径: `~/ .openclaw/workspace/`



“最小必要”原则, 避免 Token 浪费

可落地、可执行, 避免空泛

# 内置技能



The screenshot shows the 'Skills' management interface in the Openclaw Gateway Dashboard. The page title is '技能' (Skills) with a subtitle '管理技能可用性和 API 密钥注入。' (Manage skill availability and API key injection). The interface includes a search bar, a 'Refresh' button, and a list of skills. The 'BUILT-IN SKILLS' section is highlighted with a red box and shows 51 items. The 'EXTRA SKILLS' section shows 4 items, each with a description, a status indicator (e.g., 'eligible'), and a 'Disable' button.

**Skills** Refresh

Bundled, managed, and workspace skills.

Filter

Search skills 55 shown

**BUILT-IN SKILLS** 51

**EXTRA SKILLS** 4

**feishu-doc** Disable

Feishu document read/write operations. Activate when user mentions Feishu docs, cloud docs, or docx links.

openclaw-extra eligible

**feishu-drive** Disable

Feishu cloud storage file management. Activate when user mentions cloud space, folders, drive.

openclaw-extra eligible

**feishu-perm** Disable

Feishu permission management for documents and files. Activate when user mentions sharing, permissions, collaborators.

openclaw-extra eligible

**feishu-wiki** Disable

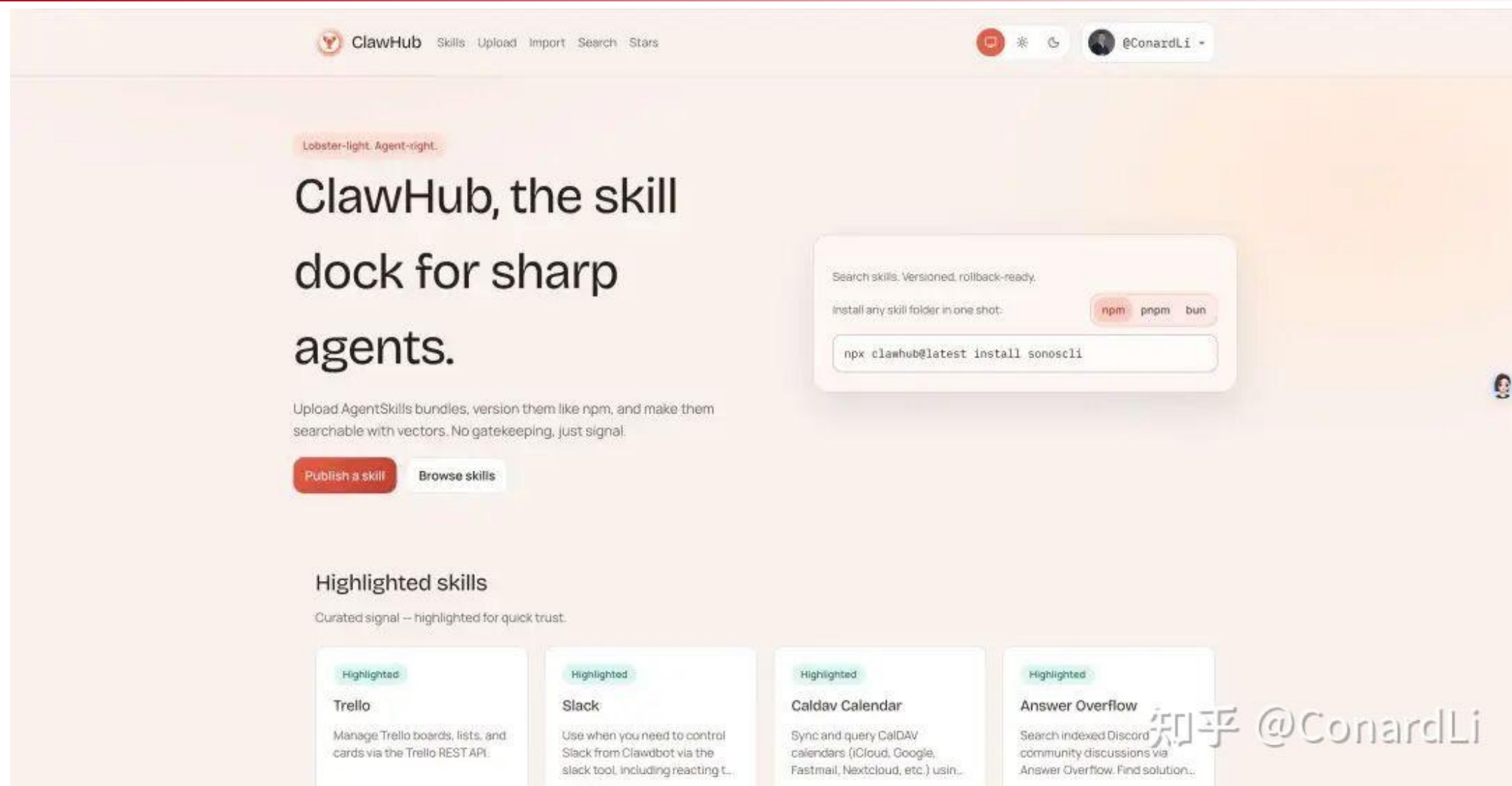
Feishu knowledge base navigation. Activate when user mentions knowledge base, wiki, or wiki links.

openclaw-extra eligible

# 安装技能

加载路径	说明	优先级	适用场景
/skills	当前项目工作区的技能目录，仅对当前 Agent 生效	★★★★★ 最高	项目专属的定制化技能
~/.openclaw/skills	本地托管目录，对本机所有 Agent 可见	★★★★ 高	个人通用的高频技能
内置技能 (Bundled)	随 OpenClaw 安装包自带的基础技能	★ 最低	通用基础能力（文件操作、系统命令等）

# 安装技能



The screenshot shows the ClawHub website interface. At the top, there is a navigation bar with the ClawHub logo and links for Skills, Upload, Import, Search, and Stars. A user profile for @ConardLi is visible in the top right corner. The main content area features a large heading: "ClawHub, the skill dock for sharp agents." Below this, there is a sub-heading "Lobster-light. Agent-right." and a paragraph explaining that users can upload AgentSkills bundles, version them like npm, and make them searchable with vectors. Two buttons are present: "Publish a skill" (in red) and "Browse skills" (in white). A search box is shown with the text "Search skills. Versioned, rollback-ready." and "install any skill folder in one shot:" followed by buttons for "npm", "pnpm", and "bun". Below the search box, a code input field contains the command: `npx clawhub@latest install sonoscli`. The "Highlighted skills" section is visible at the bottom, with a sub-heading "Curated signal -- highlighted for quick trust." and four skill cards: "Trello", "Slack", "Caldav Calendar", and "Answer Overflow". Each card has a "Highlighted" tag and a brief description of the skill's functionality.

ClawHub Skills Upload Import Search Stars @ConardLi

Lobster-light. Agent-right.

## ClawHub, the skill dock for sharp agents.

Upload AgentSkills bundles, version them like npm, and make them searchable with vectors. No gatekeeping, just signal.

[Publish a skill](#) [Browse skills](#)

### Highlighted skills

Curated signal -- highlighted for quick trust.

- Highlighted**  
**Trello**  
Manage Trello boards, lists, and cards via the Trello REST API.
- Highlighted**  
**Slack**  
Use when you need to control Slack from Clawbot via the slack tool, including reacting t...
- Highlighted**  
**Caldav Calendar**  
Sync and query CalDAV calendars (iCloud, Google, Fastmail, Nextcloud, etc.) usin...
- Highlighted**  
**Answer Overflow**  
Search indexed Discord community discussions via Answer Overflow. Find solution..

知乎 @ConardLi

# 安装第一个 Skill: Tavily



帮我搜索一下关于马斯克的最新新闻

新消息



我这边暂时没有配置好在线搜索接口，没法直接查实时新闻。  
你可以先用这些关键词在浏览器里搜一下：

- “马斯克 最新 动态”
- “埃隆·马斯克 今日 讲话”
- “马斯克 特斯拉 SpaceX 最新消息”

如果你愿意，把你看到的几条新闻链接或摘要发给我，我可以帮你：

- 快速总结重点
- 分析对特斯拉/SpaceX/马斯克本人影响
- 提炼适合发朋友圈/微博的评论文案

# 安装第一个 Skill: Tavily

tavily-search 是 ClawHub 下载量 Top 的核心技能，专为 AI Agent 优化的实时联网搜索能力，解决大模型知识 cutoff 问题，支持深度搜索、学术搜索、新闻搜索等场景。

我们可以先执行下面的命令来搜索这个技能：

```
clawhub search tavily-search
```

然后我们执行下面的命令安装技能：

```
clawhub install tavily-search
```

注意，clawhub 会默认把技能安装到当前 Agent 的 Workspace 目录下：

# 安装第一个 Skill: Tavily



用 tavily-search 搜索一下，马斯克的最新新闻



最近 1-2 天，跟马斯克相关的几条关键新闻：

1. **X 和 xAI 计划偿还 175 亿美元债务**
  - 包含 xAI 提前赎回约 30 亿美元的高收益债券，将按溢价赎回。
  - 来源：路透社等。
2. **特拉华州最高法院维持“马斯克推动的公司法改革”合宪**
  - 这套改革影响大部分美国大型公司治理结构，被法院裁定合宪继续实施。
  - 来源：Insurance Journal 等。
3. **伊朗黑客利用 Starlink 保持在线**
  - 报道称，伊朗的一些黑客组织在网络封锁环境下使用马斯克的 Starlink 卫星网络进行通信，并声称对美以发起网络报复。
  - 来源：Forbes。
4. **X 向广告主推销 Grok 的“品牌安全”能力**
  - 内部演示材料显示，X 用 Grok 的品牌安全评分和“屏蔽名单”机制来安抚广告主，此前平台因色情化内容等问题承压。
  - 来源：Business Insider。
5. **《纽约时报》调查马斯克在德州的资产与政治影响力**
  - 报道关注他在德州的企业布局、个人和公司利益交织，以及对当地政治与政策的影响。
  - 来源：CleanTechnica 对 NYT 报道的整理。

如果你更关心某一块（比如 xAI 融资/债务、Starlink 安全与合规、还是 X 的商业前景），我可以按一个主题帮你深挖一下。



# 目 录

- 1 [LangChain / LangGraph](#)
- 2 [AgentScope](#)
- 3 [OpenClaw](#)
- 4 [Hermes](#)

搜索



账户 购物袋

女士 男士 皮具 珠宝 腕表 香水与美妆 家居与生活艺术 马术系列 礼品 关于爱马仕

皮具

女士箱包/晚宴包 (227)

筛选器



即刻探索

Le Petit Sac手提包  
¥70,950



即刻探索

Sanglons手包  
¥78,500



Sabot手提包  
¥55,250



▷ 开源 · MIT 许可证 · Nous Research 出品

# Hermes Agent

随你成长的 AI 智能体。

部署在你的服务器上，连接你的消息账号，它就成为你的持久个人智能体——学习你的项目、自动构建技能、随时随地触达你。不是聊天机器人，不是代码补全工具，而是一个住在你机器上、每天都在变聪明的智能体。

免费开始使用 →

了解更多

<https://hermes-agent.org/zh/#features>

**40+**

内置技能

**5**

聊天平台

**MIT**

开源许可

**100%**

自托管

# Hermes v.s. OpenClaw

维度	Hermes Agent	OpenClaw	胜出方
开发者	Nous Research	OpenClaw Inc.	—
开源协议	MIT	Apache 2.0	平手
GitHub Star	60,000+ (增速极快)	300,000+ (成熟项目)	OpenClaw
核心定位	自进化个人 <b>智能体</b>	AI 编程助手	各有所长
学习系统	✅ 内置闭环学习循环	❌ 无自主学习	Hermes
记忆系统	✅ 三层持久记忆 + FTS5	⚠️ 基础上下文	Hermes
技能自创	✅ 自动创建 + 迭代	❌ 依赖手动配置	Hermes
运行模式	后台持续运行 (daemon)	按需启动	Hermes
消息平台	14+ IM 平台	IDE 为主	Hermes
IDE 集成	⚠️ 基础支持	✅ 深度集成 VS Code 等	OpenClaw
模型支持	200+ (含国产模型)	主流商业模型	Hermes
安全模型	五级权限管控 + 沙箱	基础沙箱	Hermes
数据存储	完全本地 SQLite	云端 + 本地混合	Hermes
安装难度	一行命令	一行命令	平手
社区成熟度	快速成长中	非常成熟	OpenClaw
插件生态	成长中 (MCP + 技能市场)	丰富 (大量第三方)	OpenClaw

# 本节复习

---

- ❑ Langchain / LangGraph
- ❑ AgentScope
- ❑ OpenClaw
- ❑ Hermes

# 致谢

- 胡玥、曹亚男、方芳：国科大《自然语言处理基础》
- 曹亚男、任昱冰：国科大《深度学习与自然语言处理概述》





# THANKS

<https://ictkc.github.io/teaching/2026spring-nlp>