



图 10.1 智能体的基本组成

10.1 智能体为什么成为现实:从工作流到闭环

这一组先回答“为什么此时能从聊天走向行动”——它涉及模型能力、上下文工程和驾驶舱工程三层基础设施同时成熟,也涉及智能体与传统工作流在闭环结构上的根本差别。

10.1.1 为什么智能体在今天成为现实

如果把“让机器人完成任务”看作一个老目标,那么智能体并不是一个全新的梦想. 真正的新变化在于,直到近几年,这件事才第一次同时具备了几个关键条件. 第一,基础大模型已经拥有足够强的通用语言理解和局部推理能力,能够在开放环境里读懂指令、归纳状态、生成中间计划;第二,外部世界越来越多地以 API、网页、文档和软件界面的形式暴露为可访问接口;第三,推断成本和工具生态已经下降到足以支撑频繁试错. 缺少其中任何一项,智能体都很难从概念变成可运行系统.

这也是为什么过去很多自动化系统看起来和今天的智能体有相似目标,却没有形成同样的技术形态. 过去的规则系统和流程引擎更擅长处理边界清晰、步骤固定的任务;一旦环境变复杂、输入变开放、异常情况变多,人们就需要写出大量分支规则. 大模型的价值恰恰在于,它把“写死规则”替换成了“在更开放的状态空间里做近似判断”. 于是,原来只有人类能接住的大量灰色地带,开始变成模型和系统可以共同处理的区域.

但也必须看清,今天的智能体之所以可行,并不意味着它已经成熟. 更准确地说,我们正处在一个能力阈值刚刚被跨过的阶段:模型已经足够强,可以让闭环跑起来;可与此同时,系统的脆弱性、评测困难和治理难题也都还非常明显. 正因为如此,本书后面的智能体部分才会同时强调接口、记忆、工程和治理. 智能体不是突然从模型里“长出来”魔法,而是若干条件刚好在这个时间点汇合之后的结果.

10.1.2 工作流与智能体的区别

今天很多系统都带有一点“智能体味道”,但并不是每个自动化流程都应该叫智能体. 最容易混淆的是**工作流**(workflow). 工作流通常是预先定义好步骤的:先检索,再总结,再生成邮件,再

发给审批者。每一步做什么、先后顺序如何,大多由人提前写死。模型在其中扮演的是某个局部环节的执行器。

智能体则更强调**根据环境变化做条件性决策**。它不只是沿着固定流程往前走,而是会根据中间观察到的信息调整后续动作。例如,在浏览网页时,如果发现页面结构和预期不同,它要决定重新搜索、换关键词、点击别的入口,还是回退一步重新规划。这种“边看边改”的能力,是智能体和普通工作流的分水岭。

不过,两者并不是绝对对立的。现实系统里,最稳定的做法往往是“工作流包住智能体”:把高风险、强约束的部分做成明确流程,把需要局部适应和探索的部分交给智能体。这样做的意义,在于不要把所有问题都扔给开放式决策。不是每个系统都需要高度自主性,很多时候,**结构化约束本身就是可靠性的来源**。

10.1.3 观察、思考、行动、反馈

理解智能体最方便的方法,是把它看成一个闭环。第一步是观察,也就是接收环境状态:可能是一段用户指令,一页网页,一个终端输出,一张截图,或者某个工具返回的结构化结果。第二步是思考,即基于当前目标和已有状态,决定下一步该做什么。第三步是行动,可能是调用工具、点击按钮、发起搜索、修改文件、提交表单。第四步则是反馈,系统要重新读取行动后的结果,判断动作是否有效,然后继续下一轮。

这个闭环看起来很自然,但它和纯对话模型有本质差别。纯对话模型很多时候是在一次性地生成一个回复,而智能体是在**与外部世界反复交互**。它的正确性不只取决于“这句话说得对不对”,还取决于“上一步操作有没有把状态推进到目标附近”“当前观察是不是足够支撑下一步决策”。一旦进入闭环,错误也会呈现新的形态:一次误读页面、一次错误点击、一次不必要的工具调用,都可能在后续回合中被放大。

也正因为如此,智能体能力的核心不只是推理强不强,而是闭环能否稳定。一个很会解释问题的模型,未必就是一个很会完成任务的智能体;相反,有些系统未必在开放问答里最惊艳,却因为观察、行动和反馈链条设计得更稳,在真实任务中表现更可靠。理解这一点,后面谈接口、记忆、评测与治理时就会清楚很多。

从这里也能看出一个常见误解:很多人把智能体理解成“更会思维链的大模型”。这当然抓住了一部分事实,因为复杂任务确实需要更多中间决策;但真正把系统推到另一个层级的,不是它不会写出更长的推理文字,而是它是否能在外部环境中持续纠偏。语言推理只是闭环中的一环,真正决定智能体成败的,往往是观察粒度够不够、动作接口稳不稳、反馈是否可验证。

闭环范式的三种代表: ReAct → ReWOO → CodeAct。这个抽象闭环在研究上有几条具体的实现路径,看一下学界给出的代表性范式,能让“观察—思考—行动—反馈”这件事更具体:

- **ReAct (2022-10)**: 让模型在生成里**交替写出推理痕迹 (reasoning trace) 和具体动作 (action)**——想一步、做一步、看反馈再调整。推理痕迹用来跟踪计划、处理异常,动作则让模型

通过外部知识库或环境拿到新观察。它是最贴近“观察—思考—行动—反馈”原始定义的一种范式。

- **ReWOO** (2023-05): 针对 ReAct 式“边想边取观察”在每一步都把上下文塞回模型导致的高 token 成本, 提出**把推理/计划和外部观察解耦**——先一次性写出整套计划(列出要调用的工具和变量), 再分别执行, 最后合成答案。它适合解释“**分开规划/执行**”的工程动机: 降低重复上下文、便于让较小的模型执行子步、在工具失败时更鲁棒。
- **CodeAct** (2024-02): 把 agent 的动作**统一为可执行的 Python 代码**。相比 JSON/tool-call 格式, 代码动作可以组合多个工具、保留中间变量、在观察反馈后动态修改前一步方案。它也解释了为什么 2025 年编程智能体特别重要——**代码不只是被写出来的对象, 也逐渐变成智能体行动的通用中介**。

这三者不是互斥替代, 而是**同一闭环在不同任务和成本约束下的变体**。ReAct 强调“推理与行动交替”, ReWOO 强调“规划与观察解耦”, CodeAct 强调“用可执行代码承载行动”; 今天产品里常常能同时看到三者的影子。

10.2 把任务定义清楚：环境、状态、目标与权限

闭环搭好之后, 真正决定智能体能不能“认真做事”的, 是任务怎么定义——它在什么世界里行动、看到的状态够不够、目标判据清不清、被允许动多少。

10.2.1 任务、环境与状态

智能体不是在真空中行动的。它总是围绕某个任务, 在某个环境中, 根据某种状态来决策。任务规定了目标, 例如“帮我订一张机票”“把这份周报整理成幻灯片”“修复这个报错并提交补丁”; 环境规定了可观察对象和可执行动作, 例如浏览器页面、操作系统、代码仓库、数据库或企业内部系统; 状态则记录了当前已经发生了什么、还剩什么没做、哪些信息已经拿到、哪些信息还不确定。

任务、环境和状态这三个概念看似简单, 却决定了智能体系统设计的边界。如果任务描述含糊, 智能体就很容易在中途偏航; 如果环境接口不清楚, 模型就不知道哪些动作可做、哪些反馈可信; 如果状态管理混乱, 系统就会忘记已经做过什么, 在长任务里反复试错。很多所谓“智能体不稳定”, 本质上并不是模型本身不够聪明, 而是**任务定义、环境接口和状态表示没有被设计清楚**。

环境还决定了智能体的动作空间。一个动作空间设计得好, 模型能够清楚知道哪些动作可用、每个动作的前置条件是什么、执行后会产生什么副作用、失败时能否重试或回滚; 动作空间设计得差, 模型就会在模糊按钮、含混命令和不可预测反馈之间试探。对人来说, 许多界面含义可以凭经验猜; 对智能体来说, 最可靠的做法是把这些经验变成更明确的接口约束。换句话说, 智能体不是只需要“会想”, 还需要一个**可理解、可操作、可验证的行动空间**。