第 1 章 n8n概述

2025年，我们学会了与AI聊天：向它提问，让它写诗，用它辅助思考和集思广益。虽然这一过程十分神奇，但本质上，我们仍然是“思考者”，而AI只是我们的“对话伙伴”。

进入2026年，AI自主执行任务的时代正在加速到来。我们不再局限于单纯地与AI对话，而是具备了调度、指挥AI完成具体工作的能力；我们从“思考者”逐渐转变为“指挥官”，而AI也从“聊天机器人”进化为能够执行复杂任务的“智能体”（Agent）。

这并非科幻构想，而是一场正在各行业逐步展开的生产力变革。支撑这场变革的关键，正是本书将深入讲解的自动化体系——一个由人工智能赋予全新能力的智能化工作体系。

本章将从自动化、低代码、工作流和智能体几个基础概念讲起，帮助读者理解为什么n8n能够成为连接AI与业务系统的重要工具。读完本章后，读者将了解n8n适合解决哪些问题，它与其他自动化平台有哪些区别，以及后续章节为什么会围绕“工作流+智能体”的组合方式展开。

1.1 欢迎来到智能体时代：自动化是你的新超能力

随着大语言模型（Large Language Model，LLM，也简称为大模型）技术与自动化技术的深度融合，数字化工作正式迈入智能体时代。

区别于传统的工具化自动化，智能体的核心优势在于同时具备流程标准化执行能力，以及智能理解和自主决策能力，能够帮助人类完成更多复杂、重复性的业务工作。而掌握自动化能力，将成为普通人和职场从业者适应智能体时代的重要能力。

自动化的核心价值并非简单的工具展示，而是将人类从重复、固定、规律性的事务中解放出来，让系统承接那些步骤标准化、输入输出明确、高频重复的工作任务，使人力能够更多聚焦于决策、创新、审核等高价值工作。

我们可以通过几个常见业务场景，直观对比传统手工方式与自动化方式的差异，具体如表1-1所示。

表 1-1　常见场景中的手工方式与自动化方式对比

场　　景	手工方式	自动化方式
客服答疑	人工查文档、复制回复	用户提问后自动检索知识库并回复
内容生产	人工收集素材、写稿、排版	自动抓取热点、AI 生成草稿、推送审核
数据监控	每天打开多个网站查看	定时抓取数据并发送异常提醒
文件处理	手动下载、重命名、上传	触发后自动处理并归档

在业务工作中，只要任务满足“步骤清楚、输入输出明确、高频重复”三大基本条件，就可以考虑使用自动化技术提升效率。

而智能体时代的核心突破，在于它不仅延续了基础自动化的固定规则执行能力，还进一步赋予系统理解和处理复杂任务的能力。这一能力的实现，离不开低代码平台提供的流程支撑，以及AI大模型带来的智能能力。

1.1.1　低代码平台：智能自动化的流程基础

想要实现系统化、可应用的业务自动化，并不一定需要从零编写完整代码开发程序。低代码平台（Low-code Platform）正是搭建自动化流程的重要载体，也是智能体应用于业务场景的基础工具。

低代码平台通过可视化、模块化的方式，帮助用户快速搭建完整业务流程。其核心能力包含以下四点：

- 可视化画布：通过拖曳节点的方式快速搭建、编排业务流程，无须专业编码即可完成流程设计。
- 多端连接器：支持快速连接飞书、邮箱、数据库、HTTP API等外部系统。
- 数据处理能力：内置数据清洗、转换、合并、筛选等功能，满足业务数据预处理需求。
- 高扩展能力：支持自定义代码插入，解决复杂业务逻辑，适配个性化场景需求。

本书的核心工具n8n，正是低代码自动化编排平台的典型代表。它区别于简单的模拟单击类工具，具备系统连接、数据处理、AI调用和复杂逻辑编排等完整能力，是搭建轻量级业务智能体的重要工具，为智能化和自动化提供底层流程支撑。

1.1.2　AI 大模型：让自动化具备智能体能力

传统低代码自动化只能执行固定、标准化的规则流程，难以应对模糊、非结构化的复杂业务场景，这也是传统自动化的主要局限。

而AI大模型的普及，突破了这一限制，使自动化系统具备理解、学习和处理复杂信息的能力，让传统低代码自动化进一步升级为智能体自动化。

依托大模型的自然语言理解、内容生成和逻辑推理能力，自动化系统可以处理非结构化信息。例如，面对用户投诉、咨询、反馈等无固定格式的内容，系统可通过AI自动识别用户意图、提取核心关键信息，再结合n8n工作流中的规则编排，自主执行自动回复、转接人工、数据归档等差异化操作，实现“智能判断+流程执行”的协同能力。

基于智能体时代的自动化核心逻辑，本书不会局限于讲解n8n的节点连接、流程搭建等基础操作，而是重点培养读者的智能自动化思维，帮助读者掌握以下三类能力的边界划分：

（1）精准界定标准化业务场景，交由工作流稳定、高效执行。

（2）精准界定非结构化、需要理解判断的场景，交由AI完成智能处理。

（3）精准把控核心风险节点，保留人工确认与审核权限，保障业务合规性。

掌握以上核心思维，才能跳出工具演示的局限，真正发挥n8n+AI智能体自动化的价值，将其应用到真实业务场景中，实现业务效率的提升。

1.2　认识工作流与智能体：它们是你的左膀右臂

在n8n的世界里，有两个重要概念：工作流（Workflow）和智能体（Agent）。我们很容易将二者混为一谈，毕竟它们都能够帮助系统自动完成任务；但实际上，它们代表着两种不同的自动化方式。理解二者的区别，是充分发挥n8n全部潜力的关键。

1.2.1　工作流：稳定的流程执行器

工作流适合处理规则明确的事务。我们可以把它理解为一条流水线：每个节点负责一项具体任务，数据按照预设路径依次流转。

例如，一个“每日新闻简报”工作流可以这样设计：定时触发→读取RSS新闻源→过滤最近24小时内容→调用AI生成摘要→发送到飞书群。

这类任务步骤清楚、边界明确，非常适合通过n8n进行流程编排。

1.2.2　智能体：具备工具调用能力的 AI 助手

智能体并不是单纯的聊天机器人。更准确地说，它是一个能够根据目标理解任务，并自主调用工具完成工作的AI助手。

例如，当用户询问“产品的退款政策是什么”时，普通大模型只能根据已有训练数据或上下文进行回答，存在信息过时或回答不准确的问题；而接入知识库工具的智能体，则可以先检索企业文档，再基于检索结果生成回答。

表1-2列出了n8n中智能体的几个常见组成部分。

表 1-2　n8n 中智能体的常见组成

组　　成	作　　用
Chat Model	大模型本身，用于语义理解和文本生成
Memory	用于保存对话上下文信息
Tool	用于让 AI 调用外部能力，如知识库、计算器、HTTP API
Output Parser	用于约束 AI 输出结构化数据

1.2.3　二者如何　合

工作流擅长稳定执行，智能体擅长理解和判断。在真实项目中二者通常并非二选一，而是组合使用：

- 工作流负责触发、鉴权、数据流转、数据入库和消息通知。
- 智能体负责理解文本、分析语义、生成内容和调用工具。
- 人工审批负责处理高风险操作。

在后续案例中，我们会反复看到这种组合方式：第7章的智能客服使用RAG智能体，第12章的热

搜分析使用AI进行综合判断，第13章的公众号发布则结合自动化流程和人工确认机制。

1.2.4　小结

用一句话总结：工作流让事情“按步骤发生”，智能体让系统“理解目标并调用工具”，而n8n把这两种能力融合到了同一个可视化平台中。

1.3　选择 n8n：构建 AI Agent 的执行控制层

在上一节中，我们探讨了工作流与智能体的本质区别。随之而来的一个核心问题是：在当前自动化、低代码以及AI应用编排平台百花齐放的背景下，本书为何选择n8n作为核心实践工具？

要回答这个问题，我们需要跳出单一功能对比的维度，从AI应用架构的角度理解n8n所扮演的角色。

一个能够在生产环境中稳定运行的AI Agent，通常包含三个核心层：

（1）认知与计算层：即LLM（如GPT、Claude、DeepSeek），负责语言理解、逻辑推理和内容生成。

（2）能力集成层：包括Agent能够调用的外部工具、API、数据库以及企业内部系统等。

（3）执行控制层：负责解析认知与计算层的决策，调度能力集成层执行具体操作，并将外部系统的状态反馈回认知与计算层，从而形成完整的业务闭环。

当前，多数平台致力于提供更强大的模型认知或更丰富的SaaS（Software-as-a-Service，软件即服务）工具连接，但一个Agent能否稳定、安全地应用于真实业务场景，往往取决于执行控制层的可靠性与扩展能力。

n8n的价值正体现于此：它并非单纯的聊天助手构建工具，也不仅是用于SaaS连接的“集成胶水”，而是一个可视化、可编程并具备企业级扩展能力的AI工作流编排平台。

1.3.1　掌握自动化平台完整技术自主权

企业和技术团队在评估AI平台时，除了关注构建效率，还需要关注另一个关键指标：数据主权与控制边界。

如果将自动化逻辑、敏感数据流转以及权限体系完全托管于第三方黑盒SaaS平台，系统在安全合规与定制化能力方面可能面临双重限制。

n8n的重要优势之一，在于它赋予用户更高程度的技术自主权。结合n8n官方提供的部署方式，平台支持三类部署形态：

（1）官方托管云服务（Managed Cloud）。

（2）自托管企业版（Self-hosted Enterprise）。

（3）开源社区版（Community Edition）。

这种灵活的部署架构带来了以下现实价值：

- 数据安全与合规：在处理客户信息、内部财务数据或核心知识库时，自托管模式允许企业划定清晰的数据边界，满足不同业务场景下的监管要求。
- 私有化集成：对于企业内部的核心资产（如本地ERP、自建数据库、遗留的定制化接口），n8n

凭借其强大的API、Webhook、Code节点以及自定义节点开发能力，能够穿透内外网隔离，实现深度集成。

- 架构的长期可控性：用户可全权自主管理部署环境、日志监控体系、CI/CD流程以及访问控制策略。

选择n8n，本质上是选择了一套能够自主掌控、可随着业务发展持续演进的自动化基础设施。

1.3.2　n8n 具备的特性

一个优秀的执行控制层，不仅需要“连接万物”，还需要在架构设计上具备以下四个特点。

1. 可视化的业务编排

n8n提供直观的图形化画布。开发者可通过拖曳节点和连线完成复杂数据流的编排。这不仅大幅降低了非研发人员的理解门槛，更让业务逻辑审查、团队协作与后期维护更加清晰。

2. 强大的可编程性

纯粹依靠可视化配置，往往难以应对复杂的业务场景。n8n将JavaScript/Python代码节点、自定义HTTP请求以及自定义节点开发作为重要能力。它践行的是“低代码优先，全代码托底”的理念，确保开发者在遭遇平台内置功能限制时，随时可以通过代码实现逻辑突破。

3. 广泛的连接生态

在生态建设方面，n8n遵循开放原则。它不仅支持数百种主流SaaS服务，还通过HTTP Request、Webhook、GraphQL等通用协议节点，确保平台具备连接各种标准API的能力。这使得Agent不会局限于某一特定生态圈，而能够连接企业现有技术栈。

4. 企业级的生产治理能力

进入正式生产环境后，平台需要具备完善的治理机制。n8n企业版提供身份认证集成（如SSO、SAML、LDAP）、企业级密钥管理（如HashiCorp Vault、AWS Secrets Manager等）、原生Git版本控制与发布流程，以及分布式执行架构（如Queue mode、Multi-main）等能力。

这表明n8n的产品定位已经超越个人效率工具，逐渐成为企业级自动化体系中台的有力竞争者。

1.3.3　主流自动化与 Agent 平台的定位差异

除了n8n之外，当前市场上还存在许多优秀的自动化与Agent平台，如Coze、Zapier、Make等。它们都能够实现不同类型的自动化任务，但在产品定位和设计理念上存在明显差异，如表1-3所示。

表 1-3　主流自动化与 Agent 平台的定位差异

平　台	核心产品定位	差异化主要特点
n8n	面向技术团队的 AI 工作流与自动化底座	部署自由度、代码扩展性、系统控制边界、复杂的生产治理能力
Coze	消费级/企业级 AI Agent 应用开发工厂	自然语言开发体验、云端托管、丰富的多模态模型接入与多端一键分发
Zapier	标准化的 SaaS 自动化总线	极简的非技术用户上手体验、庞大且成熟的 SaaS 互联生态
Make	强可视化的业务集成与编排平台	卓越的画布表达力（路由/循环处理）、超过 3000+的云端应用集成

1.3.4　深入解析三类典型竞品

为了更清晰地呈现n8n的适用场景，下面进一步分析上述三类平台的特性。

1. Coze：注　“前端体　与应用分发”

Coze致力于提供从Prompt工程到Agent发布的“一站式全栈开发体验”。它具备较低的使用门槛，让用户能够通过自然语言快速生成具有独立界面的AI应用或插件。

- 适用场景：快速验证AI产品概念、构建面向C端（用户）或内部独立运行的对话式AI助手。
- 局限性：尽管其开源版本在探索私有化部署，但整体生态仍高度依赖云端黑盒环境。对于需要深度连接企业内部遗留系统、进行复杂后端流程编排的场景，其能力和灵活性仍需结合具体需求评估。

2. Zapier：主攻“轻　级SaaS数据流转”

Zapier是标准化SaaS互联领域的代表平台之一。它的设计围绕“触发器-动作”的线性或简单分支模型展开。

- 适用场景：非技术人员（如市场人员、销售人员）快速搭建跨SaaS的通知、数据同步等轻量级工作流。
- 局限性：Zapier的计费模式严格绑定Tasks（任务步数）。对于包含长链路计算、频繁条件判断或需要高频轮询的复杂自动化，其使用成本会呈指数级上升。此外，它对底层代码扩展的支持相对有限。

3. Make：聚焦“复杂的可视化　成编排”

Make以其强大的Visual-first画布引擎著称（可视化流程设计能力），极大地丰富了图形化界面表达复杂逻辑（如多分支路由、迭代器）的能力。

- 适用场景：依赖主流云端SaaS集成，且业务流程分支繁多的系统集成项目。
- 局限性：与Zapier类似，Make采用Credits/Operations的细粒度计费模式，每个节点的动作均计入成本。同时，在本地自托管和极端的代码层级定制方面，其开放度不及n8n。

如果目标只是快速体验生成式AI应用，其他平台或许能更快带来使用反馈；如果目标是深入理解自动化与AI的工程结合方式，并构建可长期演进、满足商业级稳定性的底层架构，那么n8n提供了当下体系更完善、技术深度更充足的选型。

在明确选型逻辑后，接下来将进一步解构n8n，探讨其核心理念与关键特性如何支撑这些企业级能力。

1.4　n8n 的核心设计理念与关　特性

在上一节中，我们明确了n8n在AI Agent架构中扮演的角色：一个可视化、可编程，并具备企业级扩展能力的执行控制层。

换一个更形象的说法，n8n像AI Agent的“神经系统”——连接“大脑”（LLM 认知层）与“身体”（外部工具和业务系统）的关键基础设施。拥有这样一套能够自主控制、灵活扩展的执行系统，是企业和个人在智能时代构建自动化能力的重要基础。那么，这套“神经系统”究竟是如何设计的？

它有哪些独特特性，能够支撑AI Agent在真实业务中的应用？

本节将深入n8n的核心能力，探索其背后的设计理念，并重点介绍自2.0版本以来引入和增强的一系列关键特性。这些变化不仅提升了原有自动化能力，也进一步强化了n8n面向AI Agent时代的应用能力。

在展开具体内容之前，有必要说明n8n的两个基本属性：开放模式和部署方式。n8n采用Fair-Code（公平代码）模式，兼顾源码透明性与商业可持续发展（将在1.5.2节深入探讨）；同时支持自托管部署，使用户能够根据自身需求管理运行环境，保障数据控制能力（在1.3.1节已有详细论述）。

下面，我们正式开启n8n核心能力的探索，了解它如何将复杂的自动化流程与AI能力结合，转化为更加轻量化、常态化的数字生产力。

1.4.1　可视化编排界

1.3.2节介绍了n8n的四个核心特性，其中首个便是“可视化业务编排”。这是n8n最直观，也是最具特色的设计之一——将抽象、复杂的代码逻辑转化为一张直观的流程图。

我们不再需要记忆大量函数和语法，只需要将代表不同功能的节点（Node）拖曳到画布（Canvas）中，再通过连线将它们组合起来。

第一次进入编排界面时，可以先观察画布中的节点与连线布局，如图1-1所示。

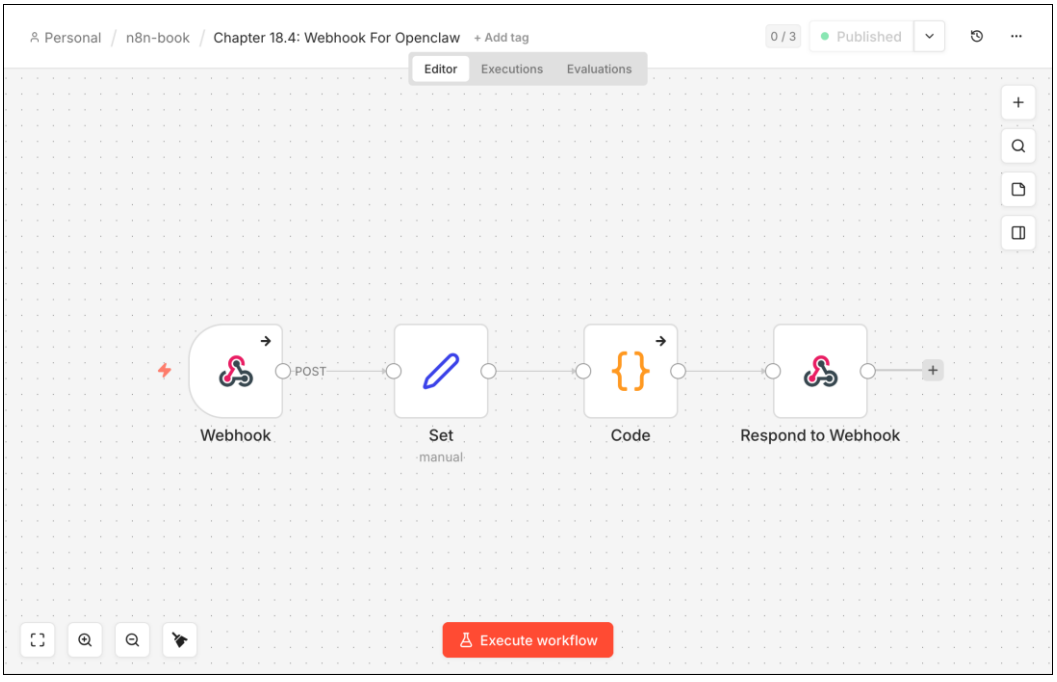

图 1-1　n8n 画布中的节点与连线布局

数据流向、逻辑分支、任务并行与串行关系，都可以在流程图中直观看到。这种类似“搭积木”的编程体验，大幅降低了构建和维护复杂自动化流程的心理门槛和技术门槛。

无论是产品经理、市场运营人员，还是专业开发工程师，都能够快速理解流程逻辑，并将业务需求转化为实际运行的工作流。

在n8n 2.0版本中，这种可视化体验被进一步优化。例如，Focused Nodes（聚焦视图）功能允许用户只关注画布中的某几个特定节点，暂时隐藏其他部分。在处理包含上百个节点的超大型工作流时，

该功能能够帮助用户减少视觉干扰，更专注于当前逻辑。同时，画布内的交互体验和数据传输展示也得到了优化，使整个开发过程更加流畅。

1.4.2　丰富的第三方应用　成

如果说可视化编排是“神经系统”的结构，那么广泛的集成能力就是其“丰富的感受器与效应器”。自动化的本质之一就是“连接”，即在原本相互独立的数字系统之间建立数据流转和业务协同。一个神经系统如果无法感知外界（输入）或无法驱动肢体（输出），那么它将毫无用处。

n8n在连接能力方面表现突出。它提供了超过一千个由官方和社区维护的集成节点，包括Google Workspace、Slack、Microsoft Teams等协同工具，以及各类数据库（如PostgreSQL、MongoDB）、CRM（如Salesforce、HubSpot）、开发者工具（如GitHub、Docker）和主流的AI服务。它就像一个万能适配器，无论“身体”部件来自哪个厂商，它都能找到合适的接口进行连接。

在n8n 2.0版本中，这个“万能适配器”的扩张速度还在不断加快。例如，新增的Firecrawl节点可以帮助用户轻松抓取和转化网页内容，Pinecone Quick Connect和Data Table节点则进一步简化了向量数据库和结构化数据的连接操作。这些能力使AI Agent能够访问和处理更加丰富的数据来源，进一步拓展应用范围。

1.4.3　原生 AI Agent 能力的支持

n8n的版本迭代实现了产品定位的根本性升级：1.x 版本的核心能力聚焦于固定规则的工作流自动化，主要解决标准化、流程化的任务调度问题；而2.0版本则实现了能力跃迁，核心定位升级为轻量化 AI Agent 构建平台，支持用户通过可视化方式搭建具备自主理解、自主决策和自主执行能力的智能体。

业界通用的AI Agent架构包含四大核心要素：规划（Planning）、记忆（Memory）、工具（Tool）、行动（Action）。n8n 2.0围绕这四大核心要素进行了全方位能力升级，将传统需要大量代码开发和逻辑调试的Agent搭建工作，简化为可视化拖曳与参数配置，大幅降低了智能体的开发门槛。

在配置AI Agent节点时，需要重点关注节点下方的几个连接端口（见图1-2）：Chat Model负责智能推理与内容生成，Memory负责保存对话上下文，Tool负责调用外部能力。三者相互配合，共同支撑Agent完成完整的规划、决策与执行流程。

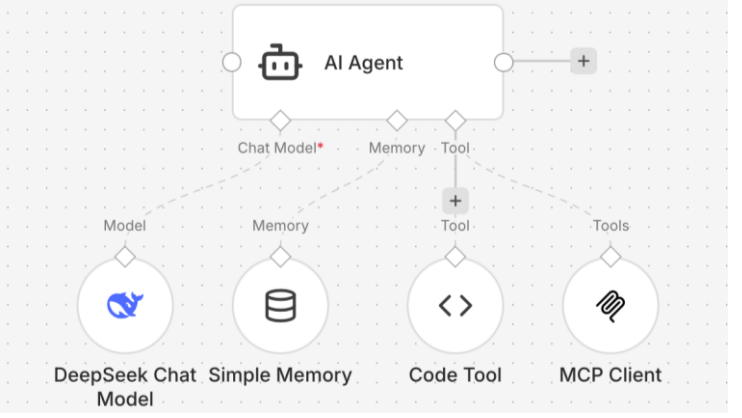

图 1-2　AI Agent 节点及其模型、记忆和工具连接区

1. 智能对话交互：全新Chat Hub交互体系

n8n 2.0提供统一的Chat Hub对话交互界面，补齐了传统工作流“只能后台执行、无法实时交互”的短板。用户可通过该界面直接与自建AI Agent进行实时对话、下发任务指令、观察Agent的推理与执行过程，并可动态调整模型参数、工作流规则与工具权限。这一能力使AI Agent的开发、调试和优化过程更直观，实现了可视化迭代开发。

2. 智能化工具生成与调用能力

为解决传统自动化流程固定、适配性差、开发效率低的问题，n8n 2.0强化了工具自主生成与调用能力，核心包含两项关键功能：

1）AI Builder

AI Builder是大幅降低Agent开发门槛的革命性功能。无须用户手动搭建节点、配置流程，仅需通过自然语言描述业务目标与执行要求，平台即可自动解析需求并生成完整、可运行的自动化工作流。该能力实现了“自然语言需求→可落地业务流程”的一键转化，极大缩短了AI Agent从构思到落地的开发周期。

2）原生Python Code Tool能力

在原有JavaScript代码能力的基础上，n8n 2.0支持Python代码的直接调用与执行。Python是数据分析、机器学习、算法运算和智能处理等场景中的核心生态语言，原生Python能力的接入，补齐了Agent在复杂数据运算、模型调用和专业算法处理等方面的能力短板，使Agent能够承接更高阶的智能化业务场景，进一步拓展了其能力边界。

依托上述原生AI Agent能力的全面升级，n8n彻底释放了大语言模型的智能潜力。单纯的大模型主要具备对话与推理能力，属于“被动响应式智能”；而结合n8n的记忆存储、工具调用、流程编排和代码执行能力后，AI Agent便拥有了感知业务、自主调度和落地执行的完整能力。由此，AI Agent可以从以对话和推理为主的“静态智能体”，升级为可规划、可记忆、可调用工具、可自主执行的“动态业务智能体”，真正实现从理论智能向业务实践的落地转型。

1.4.4　人机协作审批

人机协作审批（Human-in-the-Loop，简称HITL）是n8n 2.0引入的一项关键功能。该功能提供了一种更可靠的行动监督机制，允许用户在Agent执行某些关键操作（如发送重要邮件、删除数据库记录）之前，设置一个人工审批环节。

当Agent执行到需要人工确认的操作时，会暂停当前流程并向用户请求授权；在获得批准后，流程才会继续执行。这相当于为Agent增加了一道安全控制机制，确保它在具有较高自主权的同时，避免因错误判断而执行不可逆的危险操作。

1.4.5　按　编写 JavaScript/Python 脚本

可视化能力虽然强大，但任何图形界面都不可避免地存在能力边界。当遇到复杂的数据转换、自定义算法，或需要与没有现成节点支持的内部系统进行交互时，如何突破这些限制？

n8n的设计理念是“按需编码”（Code-when-needed），它提供了一种灵活的解决方案：将代码作为最后但也是最强大的扩展能力。我们将其称为“可编程的神经元”。在大多数常规场景下，用户可以通过图形化节点完成自动化流程搭建；而在特殊需求场景中，则可以随时打开Code（代码）节点，

通过JavaScript或Python编写自定义代码，实现更加复杂的功能扩展。

Code节点的脚本编辑区域如图1-3所示。

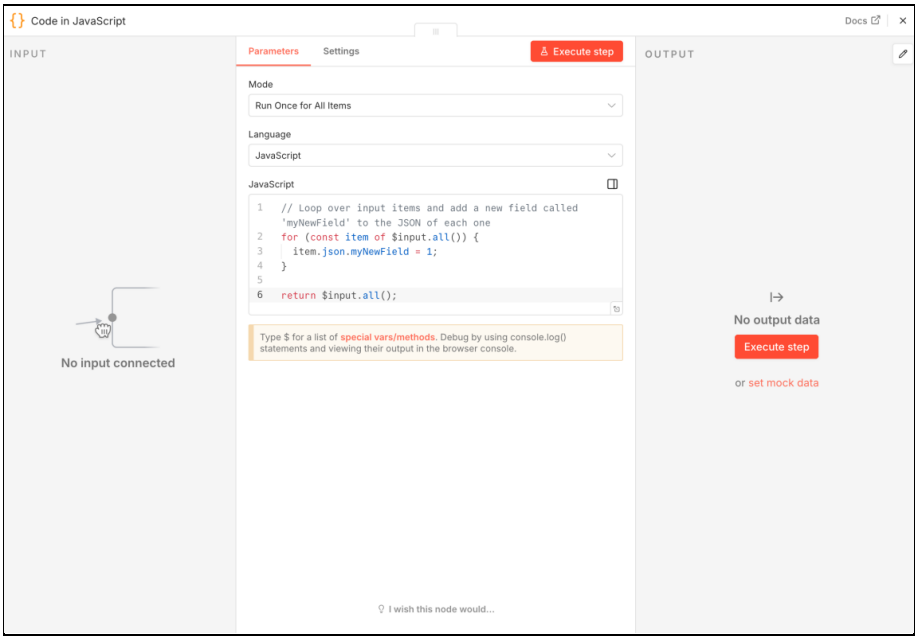

图 1-3　Code 节点的脚本编辑区域

这种设计既保证了普通用户进行快速搭建和原型验证时的高效率，也为专业开发者提供了更高的灵活性和能力扩展空间。

在n8n 2.0版本中，该扩展能力的安全性和稳定性也得到了进一步增强：

- Task Runners：所有Code节点默认运行在独立的沙箱（Sandbox）环境中。这意味着，即使代码由于异常情况运行失败，也只会影响当前执行任务，而不会导致整个“神经系统”（n8n主进程）受到影响。这对于生产环境的稳定运行具有重要意义。
- 与时俱进的底层环境：n8n的运行环境升级至当前版本的Node.js v24，使开发者能够利用最新的语言特性和性能优化能力。

1.4.6　私有化　署与沙箱安全　离

“我的数据我做主”，这不仅是n8n支持自托管的核心优势，也是它在2.0版本安全架构中的重要体现。n8n的底层设计始终遵循一项核心原则：一个强大的“神经系统”必须建立在安全、可靠的基础之上。

为此，n8n 2.0引入了一系列安全增强措施：

- 隔离执行：Code节点在隔离沙箱环境中执行，从根本上降低了恶意代码影响主服务的风险。
- 环境变量访问限制：默认情况下，Code节点无法直接读取服务器的环境变量，从而避免敏感信息（如API Key、数据库密码）被意外暴露。
- 危险操作禁用：ExecuteCommand（执行任意服务器命令）、LocalFileTrigger（监控本地文件系统）等高风险节点默认处于禁用状态，需要管理员显式开启。这一机制有效降低了因配置不当而导致的安全风险。

- 强制持久化: n8n取消了在内存中处理二进制数据的模式，所有文件数据必须经过持久化存储。这不仅提升了处理大文件的稳定性，也避免了敏感数据残留于内存中的风险。

这些看似严苛的限制，实际上构成了对用户数据和系统安全的重要保障。

1.4.7　执行数据记录与可视化调试

任何复杂系统在运行过程中都难免遇到问题。一个优秀的自动化工具，不仅需要帮助用户完成流程构建，更需要提供高效、透明的调试能力。n8n在这一方面提供了高度透明的执行记录机制，使故障排查过程更加直观。

每一次工作流的执行，n8n都会记录详细的执行快照。用户可以像看电影一样，清晰了解：

- 每个节点在执行前接收到的输入数据。
- 每个节点在执行后产生的输出结果。
- 数据在节点之间的转换过程和流动路径。

这种高度透明的执行记录机制，相当于为自动化流程提供了一个全方位的“黑匣子”。当流程出现异常时，用户无须依赖经验去猜测，只需回溯执行历史，即可快速、精准地定位问题所在。

n8n 2.0进一步增强了调试体验，例如引入执行数据预览和交互式代码diff（差异）显示等功能，使用户能够更直观地分析数据变化过程，并快速找到问题的症结。

1.4.8　Publish 与自动保存

这是n8n 2.0在工作流开发体验上的一项重大升级。它彻底告别了1.x时代“保存即上线”的危险模式，构建了从“自动备份”到“保存草稿”再到“发布生产”的完整安全链路，为团队协作和生产环境的稳定性提供了更加可靠的保障。

“保存即上线”的特性对于企业级应用而言存在较大风险——任何细微的配置失误，都可能直接影响线上业务。为了解决这个问题，n8n 2.0引入了全新的状态管理机制：

- 自动保存（Autosave）：n8n会在后台自动保存当前工作进度。即使遇到浏览器意外关闭、网络中断等突发情况，也能够尽可能避免编辑内容丢失。同时，它还具备连接状态检测能力：当检测到与服务器断开连接时，会暂停保存，以避免产生状态冲突，从而为用户提供更加稳定的编辑环境。
- 保存草稿（Draft Save）：在当前的n8n工作流生命周期中，用户的修改会先被保存为“草稿”（Draft）。草稿与线上正在运行的正式版本相互隔离，因此无论在草稿中进行修改还是测试，都不会直接影响当前生产环境中的运行版本。
- 发布（Publish）：只有当草稿经过充分测试且确认功能完善后，单击Publish按钮，相关修改才会被正式发布（推送到生产环境），成为新的线上运行版本。该入口如图1-4所示。

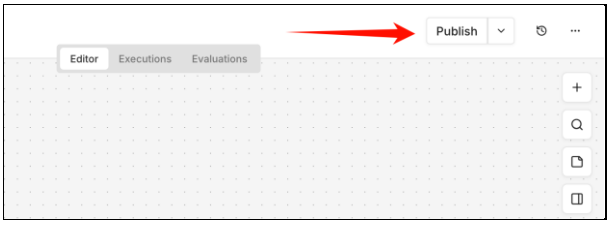

图 1-4　n8n 工作流的 Publish 按钮

这套机制和文稿撰写的逻辑相似：自动保存用于防止内容丢失，草稿用于反复调整和校验，确认无误后再正式发布。

进一步来看，n8n还提供了独立的workflow:publish和workflow:unpublish权限。这意味着在团队内部，可以根据角色划分不同权限。例如，初级工程师主要负责工作流编辑和草稿保存，而资深工程师或技术负责人拥有最终发布权限，以此形成更加规范、完整的“开发-测试-发布”流程（CI/CD）。

1.4.9　团　协作与工作流版本管理

除了Publish/Save范式，n8n 2.0还带来了一系列旨在提升团队协作效率和开发体验的新功能，使多人共同构建和维护大型自动化系统更加便捷。

- Tab作用域协作：当多个用户同时编辑同一个工作流时，每个人的修改会在各自的浏览器标签页中保持相对独立，互不干扰。用户可以清晰地看到其他协作者正在修改，但他们的修改不会实时同步到用户的当前界面，从而降低多人同时操作带来的冲突和混乱。
- AI Usage设置页：团队管理员可以统一查看和管理AI相关功能（如AI Agent节点）的使用情况和成本，方便进行预算控制和资源分配。
- 版本历史命名：在发布新版本时，可以为版本添加具有实际含义的名称，如“feat:添加了飞书审批功能”，而不仅仅依赖自动递增的版本号。这样可以使版本回溯和变更追踪更加直观。
- 更精细的配置选项：例如，在导入CSV文件时，可以更加方便地处理列名映射和重命名；在配置OAuth凭证时，也可以根据实际需求在托管模式和自定义模式之间轻松切换。

这些看似细微的改进，共同构成了n8n 2.0更现代化、更适合团队协作的开发体验。

从可视化编排的直观，到按需编码的灵活；从原生AI的智能，到企业级的安全与协作，n8n 2.0的这些核心设计理念与关键特性，共同铸就了这套强大而可靠的“神经系统”。

理解了这些设计背后的“为什么”，我们才能更好地掌握“如何用”。接下来将介绍n8n理念和生态系统。你会发现，今天我们在n8n中所见的每一个强大功能，都源于其诞生之初的远见和对自动化未来的深刻洞察。

1.5　n8n 的理念与生态

n8n最早来自一个朴素的需求：让技术人员和业务人员都能更灵活地连接系统。它选择了“可视化+可扩展+可自托管”的路线，因此被很多技术团队用于内部自动化、数据同步、AI工作流和运维工具建设。

1.5.1　开放与商业的平衡

n8n采用Fair-Code模式：源码公开可用，并在Sustainable Use License（可持续使用许可证）允许的范围内支持免费自托管使用，同时通过云服务和企业能力形成商业化。这种模式对读者有两个直接好处：

- 学习阶段可以用社区版免费或低成本实践。
- 真正进入企业场景时，也有团队协作、权限、安全和运维相关能力可选。

1.5.2　生态资源

学习n8n时，建议重点关注三类资源，如表1-4所示。

表 1-4　n8n 生态中的常用资源

资　源	用　途
官方文档	查询节点参数、部署配置和版本变化
模板库	学习成熟工作流的设计方式
社区节点	补充官方节点暂未覆盖的应用能力

本书会优先讲通用方法，不要求读者记住所有节点。在遇到新场景时，优先查看是否有现成节点；若没有节点，再考虑使用HTTP API或扩展代码。

1.5.3　小结

n8n的生态价值不在于“节点越多越好”，而在于它提供了一套可扩展的自动化方法。只要理解触发、数据流、节点、凭据和调试，就能不断接入新的工具。

1.6　n8n 的技术架构：深入了解底层设计

前面已经介绍了n8n的特性、设计理念和生态。本节换一个角度，从技术架构来看它为什么适合作为自动化与AI工作流平台。这里不对源码细节进行介绍，只保留后续部署、扩展和排错会用到的核心概念。

1.6.1　核心技术栈与组成

n8n的后端主要基于Node.js与TypeScript，前端编辑器基于Vue 3与TypeScript。这套技术栈适合频繁处理外部API、Webhook、定时任务和可视化交互，也方便社区持续扩展节点能力。

从运行结构看，n8n由四部分组成：

- 编辑器：负责节点拖曳、参数配置、执行结果查看和调试。
- 后端服务：提供API、Webhook入口、触发器调度和权限控制。
- 执行引擎：按照工作流的节点顺序处理数据、调用服务并记录状态。
- 数据库：保存工作流定义、凭据、执行记录和系统配置。

后续排错时，也可以按这四层定位问题：界面打不开时先看浏览器、域名和反向代理；Webhook不触发时先看入口、触发器和后端日志；节点执行失败时先看参数、凭据和外部API；数据丢失或迁移异常时重点检查数据库、挂载目录和加密密钥。

1.6.2　执行模式：　认模式与　列模式

学习阶段通常使用默认模式。在这种模式下，编辑器、API、触发器和工作流执行功能主要由同一个n8n实例承担。此模式部署简单、资源占用低，适合个人学习、概念验证和中小规模自动化。

当工作流数量、执行频率或并发量明显增加时，可以考虑队列模式（Queue Mode）。队列模式通常由Main、Redis、Worker和PostgreSQL等组件构成：Main负责接收请求、管理触发器并将执行任务加

入队列，Redis负责承载任务队列与Main、Worker之间的任务协调，Worker负责执行工作流，PostgreSQL负责保存核心数据。当执行压力变大时，可以通过增加Worker提升吞吐量。

需要注意的是，队列模式不等同于自动高可用，它主要解决执行能力扩展问题。如果要做生产级高可用，还要考虑多Main（该能力需要相应的Enterprise许可）、负载均衡、统一PostgreSQL、统一Redis、相同的N8N_ENCRYPTION_KEY，以及适合分布式环境的二进制数据存储方式；否则可能出现某个实例无法解密凭据，或者不同Worker无法访问所需的二进制数据，从而导致执行失败。在队列模式下，不应使用仅依赖单机本地磁盘的filesystem模式存储二进制数据，可根据部署方案使用数据库存储；符合许可条件时也可使用S3外部存储。

1.6.3　署建议：以 Docker Compose 为主线

n8n支持npm安装、云服务和Docker等多种方式。为了降低环境差异，本书后续统一采用Docker Compose作为学习和实践主线。它可以把n8n、PostgreSQL、挂载目录和环境变量放在同一套配置中管理，便于启动、升级、迁移和备份。

使用Docker Compose还有一个好处：后续如果要从默认模式扩展到Redis和多个Worker，可以继续在同一套Compose配置上演进，而不必重新设计部署方式。

最后要特别注意N8N_ENCRYPTION_KEY。n8n会用它加密凭据中的敏感信息。在迁移服务器、更换容器或恢复备份时，数据库和这个密钥必须一起保存；如果密钥不一致，原有凭据可能无法解密。

1.6.4　小结

本节只需要记住三点：n8n由编辑器、后端服务、执行引擎和数据库组成；默认模式适合学习和中小规模场景，队列模式适合更高并发但不自动等同于高可用；本书后续以Docker Compose为部署主线，并会反复用到数据库、挂载目录和N8N_ENCRYPTION_KEY这些概念。

1.7　社区、模板与学习路径

学习n8n不需要闭门造车。很多常见场景已经有人做过模板，社区也沉淀了大量节点和排错经验。

1.7.1　模板的价值

模板不是用来让我们原样照抄的，而是用来学习结构的。在看到一个模板时，可以重点观察：

- 它用什么触发器启动。
- 数据经过哪些清洗和判断。
- 哪些地方调用了AI或外部API。
- 最终结果发送到哪里。

后续我们在设计自己的工作流时，也可以按这个顺序拆解。

1.7.2　社区节点的使用原则

社区节点能补充官方节点的不足，但使用前要注意：

- 是否仍在维护。

- 是否支持当前n8n版本。
- 是否需要额外凭据或外部服务。
- 是否适合生产环境。

本书会在确有必要时使用社区节点，并说明为什么不用HTTP API或官方节点。

1.7.3　小结

社区资源可以加速我们的学习，但真正重要的是理解工作流设计方法。模板解决的是“从哪里开始”，方法解决的是“如何独立搭建”。

1.8　企业应用场景概览

在企业中，n8n常用于连接系统、减少重复劳动和引入AI能力。本节不使用长篇案例，先用几个典型的企业应用场景建立一个整体印象：它通常出现在那些需要跨系统流转、需要记录状态、需要AI参与判断或生成的流程里，如表1-5所示。

表 1-5　n8n 在企业中的典型应用场景

场　景	典型流程	n8n 价值
IT 运维	监控告警→判断级别→通知或创建工单	缩短响应时间，减少人工分派
客服支持	用户问题→检索知识库→AI 生成回复→必要时转人工	提升一线回复效率，降低重复咨询成本
市场运营	抓取热点→AI 分析→生成内容草稿→推送审核	加快选题和内容生产速度
数据同步	多系统数据→清洗→入库或同步到报表	减少手工导表和复制粘贴
审批流程	外部请求→权限校验→人工确认→执行动作	在自动化和安全之间取得平衡

这些场景有一个共同点：它们通常不是依靠单个AI对话就能完整解决的，而是需要连接多个系统、处理数据并执行动作。n8n的优势正是在这里体现出来。

第1章帮助读者建立对全书的基础认知：n8n是一个面向自动化和AI工作流的编排平台。第2章将带领读者搭建可实际使用的n8n环境，为后面的案例做好准备。

第 2 章

n8n环境搭建

2

在正式开启n8n的自动化探索之前，首要任务是搭建一个稳定、可靠的运行环境。n8n作为一个灵活的工具，提供了多种部署方式以适应不同用户的需求。从个人开发者在本地计算机上进行开发级部署，到企业在服务器上进行商业级部署，n8n都能胜任。

本章将详细介绍n8n的版本区别、部署方式，并指导读者完成从零到一的安装与配置过程，为后续章节的实践打下坚实的基础。

2.1 n8n 的版本区别

n8n采用了Fair-Code与商业服务并行的模式，其核心功能对社区版用户开放，同时为企业级用户提供增值服务和更强大的协作功能。

我们可以将n8n的主要版本归纳为以下3类：

（1）n8n Community（社区版）：这是n8n的免费自托管版本，源码公开可用。用户可以获取其全部源码，并将其部署在自己选择的硬件上，无论是个人计算机、本地服务器还是云主机。它提供了n8n所有的核心自动化功能，工作流数量本身不受付费套餐额度限制，实际执行能力主要取决于部署环境和资源。选择社区版意味着拥有最高的数据控制权和自定义能力，但同时也需要自行负责环境的搭建、维护、升级和安全保障。

（2）n8n Cloud（云服务版）：这是官方提供的托管服务。用户无须关心服务器、数据库等底层部署和大部分日常运维工作，只需在官方网站上注册账号即可使用。n8n Cloud根据不同订阅计划提供工作流共享、用户与项目管理、管理角色等协作能力，部分高级功能需要相应的高级套餐。该版本主要根据工作流执行次数及所选套餐进行收费，非常适合希望快速启动项目、不想投入运维精力的团队或个人。

（3）n8n Enterprise（企业版）：该版本专为具有较高安全、治理、合规和扩展需求的组织设计。它可以由n8n Cloud官方托管，也可以部署在企业自己的基础设施上（Self-hosted），从而适应不同的数据管理和安全策略要求。此外，企业版还提供专属的技术支持服务和更高级的治理功能，如日志流、外部密钥存储以及专属SLA支持等。当前n8n还提供Self-hosted Business等付费方案，其功能和定位介于普通云端套餐与Enterprise之间，具体功能以官方当前套餐说明为准。

三者的差异如表2-1所示，读者可以先据此建立整体印象，再根据自己的部署和运维能力进行选择。

表 2-1　n8n Community、Cloud 与 Enterprise 的版本差异

特　性	n8n Community	n8n Cloud	n8n Enterprise
部署方式	用户自托管	官方云托管	用户自托管或官方托管
价格	免费	付费订阅	定制报价
核心功能	✔	✔	✔
团队协作	有限	✔	✔
用户管理	基础	根据套餐提供不同级别的管理能力	高级（RBAC）
单点登录	不支持	部分高级套餐支持	✔
运维责任	用户	n8n 官方	根据托管方式由用户或 n8n 官方承担
数据控制权	完全控制	用户自行管理	取决于托管方式
技术支持	社区支持	官方支持	专属支持（SLA）

对于初学者来说，可以从n8n Community的Docker版或npm安装方式入手，在本地环境中无成本地学习和测试。对于希望将n8n用于生产环境但缺乏运维经验的团队，n8n Cloud是一个省时省力的选择。

2.2　Cloud 版本　署

n8n Cloud是最快的入门方式。我们只需要注册账号，就可以直接进入编辑器创建工作流，不需要准备服务器、数据库或域名。对于刚接触n8n的读者来说，先用Cloud体验画布、节点和执行结果，再决定是否自托管，是一个比较轻量的学习路径。

2.2.1　适用场景

建议在以下情况下优先使用n8n Cloud：

- 只是想先体验n8n的画布和节点。
- 不熟悉服务器部署。
- 数据和业务场景允许使用云端托管服务。
- 希望把时间放在工作流设计上，而不是运维上。

2.2.2　基本使用流程

n8n Cloud的使用流程如下：

（1）打开浏览器，访问n8n官网https://n8n.io/，单击页面中的Get Started按钮或类似入口，如图2-1所示。

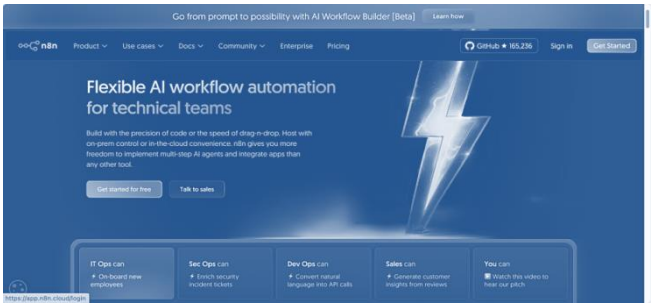

图 2-1　n8n 官网的 Get Started 入口

（2）单击页面中的Get Started后进入注册流程，按提示填写邮箱、密码等信息，如图2-2所示。这里的Account name通常会成为Cloud工作区访问地址的一部分，例如your-instance.app.n8n.cloud，建议使用容易识别且唯一的名称。工作区注册信息填写如图2-3所示。

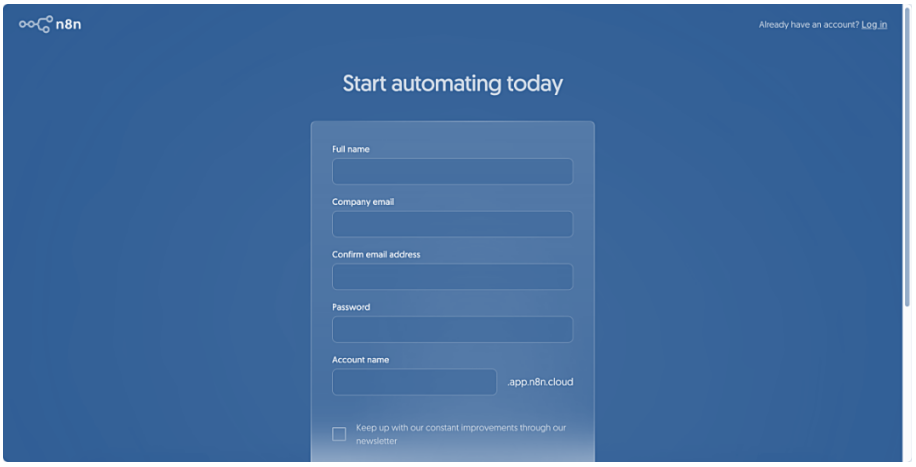

图 2-2　进入 n8n Cloud 注册流程

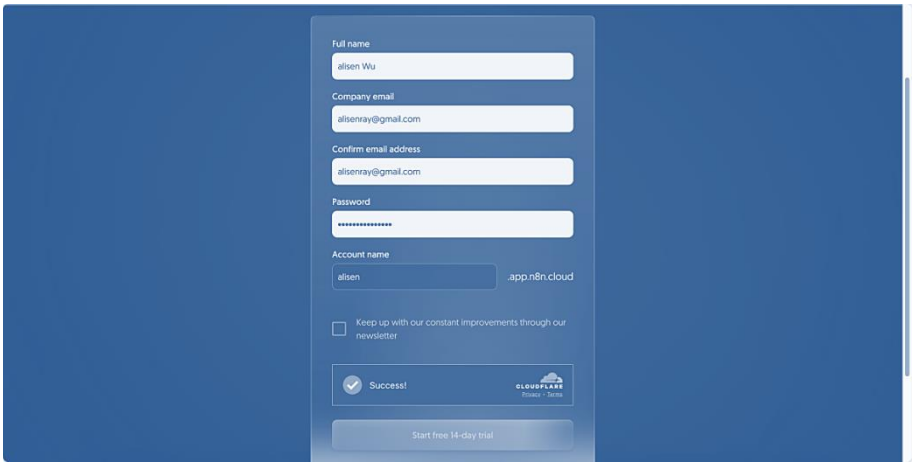

图 2-3　填写 n8n Cloud 工作区注册信息

（3）注册完成后，可能会弹出一个简单的问卷调查页面，用来了解我们的使用场景。问卷不影响后续学习，按实际情况填写即可。

（4）如果系统提示邀请团队成员，可以先单击Skip按钮跳过。个人学习阶段不需要配置团队协作。

（5）单击Start automating按钮进入工作区首页（即主界面），如图2-4所示。进入后，先确认左侧的导航栏、右上角的Create workflow（创建工作流）入口和中间的工作流区域，后面创建与管理流程都会从这个首页开始。

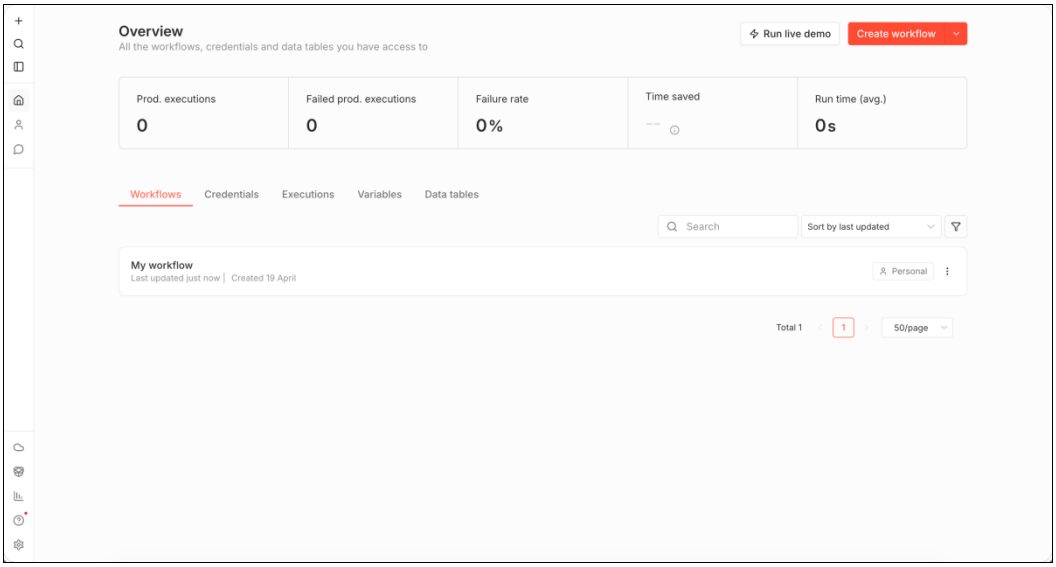

图 2-4　n8n Cloud 工作区首页

（6）在工作区首页，单击Create workflow按钮，添加第一个触发器和动作节点，手动执行并查看节点输出。第3章会详细介绍画布、节点、数据流和表达式等核心概念。

2.2.3　套　与使用建议

n8n Cloud通常会提供试用期，试用结束后可以根据每月执行次数、并发执行需求、共享项目数据和安全需求选择对应套餐。它的计费重点是工作流执行次数，而不是节点数量：一个工作流从触发到结束完整运行一次，通常计为一次执行，与其中包含多少个节点无关。

选择套餐时，不必一开始就看企业功能，重点是比较执行次数、并发执行数量、共享项目数量和是否需要高级安全能力。n8n Cloud套餐价格与功能分层如图2-5所示。

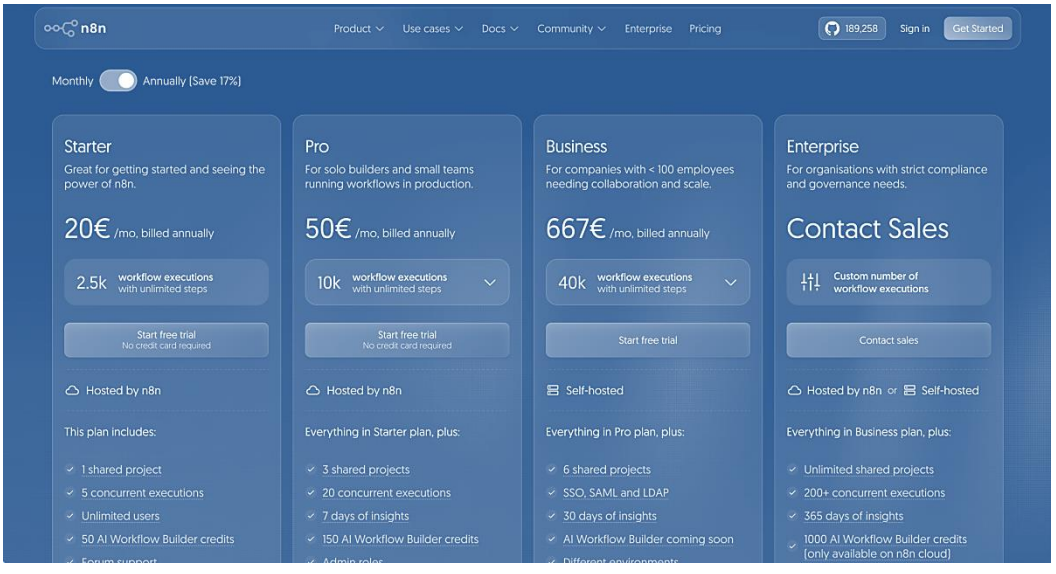

图 2-5　n8n Cloud 套餐价格与功能分层

初学阶段不必纠结套餐细节，先用试用额度完成本章和第3章的练习即可。如果后续要长期运行自动化任务，再根据执行频率和团队需求选择Starter、Pro或Enterprise方案。

2.3　Community 社区版本　署

对于希望自主掌控数据和部署环境，或者后续要跟着本书完成多个实战案例的读者来说，自行部署n8n Community版本是一种比较合适的选择。本节以Docker Compose作为主线部署方式。后续章节中的案例，也都会默认基于Docker Compose环境展开。

本书基于n8n v2.10.3撰写。n8n的迭代速度很快，不同版本的界面和参数可能会有差异。为了避免在跟随本书操作时遇到“对不上”的问题，建议读者使用相同版本。

2.3.1　署前的准备

在开始安装之前，需要确认运行n8n所需的基本条件是否已具备。n8n本身是比较轻量的，但其实际资源消耗会随着工作流复杂度、执行频率、AI调用和文件处理量的增加而上升。表2-2给出了学习环境和长期运行环境的两组参考值。

表 2-2　n8n 社区版　署前的资源准备建议

目	学习环境建议	期运行建议
CPU	1 核即可	2 核及以上
内存	2GB	4GB 及以上
磁盘	20GB	40GB 及以上，并定期备份
Docker	Docker Desktop / Docker Engine	Docker Engine+反向代理
数据库	PostgreSQL+pgvector	PostgreSQL+pgvector

本书推荐使用PostgreSQL，而不是SQLite。原因有两个：

（1）后续AI/RAG案例会用到向量能力，pgvector可以提前准备好环境。

（2）PostgreSQL更适合长期运行，在并发访问、备份和迁移等方面通常比SQLite更适合生产环境。

2.3.2　创建　目目录

先在本地创建一个专门存放n8n配置的目录。例如：

```
mkdir my-n8n
cd my-n8n
```

在这个目录中，我们会放置3个关键内容，如表2-3所示。

表 2-3　本地　署目录中的关　文件和目录

文件或目录	作　　用
docker-compose.yml	定义 n8n 和 PostgreSQL 服务
n8n_data/	保存 n8n 本地配置和加密相关数据
local_files/	给后续工作流读写本地文件使用

我们不需要手动创建n8n_data和local_files，Docker Compose在启动时会按挂载配置自动创建。为使结构更加清晰，也可以提前创建：

```
mkdir n8n_data local_files
```

2.3.3　编写 docker-compose.yml

在my-n8n目录下创建docker-compose.yml文件，并编写以下内容：

```
version: "3.8"

services:
   postgres:
      # 使用内置pgvector扩展的PostgreSQL镜像，便于后续的RAG案例使用
      image: pgvector/pgvector:pg16
      container name: postgres
      restart: unless-stopped
      environment:
         POSTGRES USER: n8n
         POSTGRES PASSWORD: my super secret db password
         POSTGRES DB: n8n
      volumes:
         - postgres_data:/var/lib/postgresql/data
      ports:
         - "5432:5432"

   n8n:
      # 锁定本书使用的n8n版本，避免界面和节点参数变化影响学习
      image: n8nio/n8n:2.10.3
      container_name: n8n
      restart: unless-stopped
      ports:
         - "5678:5678"
      environment:
         # --- 数据库连接配置 ---
         - DB_TYPE=postgresdb
         - DB_POSTGRESDB_HOST=postgres
         - DB_POSTGRESDB_PORT=5432
         - DB_POSTGRESDB_DATABASE=n8n
         - DB_POSTGRESDB_USER=n8n
         - DB_POSTGRESDB_PASSWORD=my_super_secret_db_password

         # --- 网络配置 ---
         - N8N_HOST=localhost
         - N8N_PORT=5678
         - N8N_PROTOCOL=http
         - N8N_SECURE_COOKIE=false
         - NODE_ENV=production
         - GENERIC_TIMEZONE=Asia/Shanghai
      depends_on:
         - postgres
      volumes:
```

```
            - ./n8n_data:/home/node/.n8n
            - ./local_files:/files

volumes:
   postgres_data:
```

配置完成后，先不要急着启动。表2-4列出了docker-compose.yml中最需要理解的几项配置，后续排错时也会经常回到这些位置。

表 2-4　docker-compose.yml 中的关　置说明

置	说　明
n8nio/n8n:2.10.3	指定 n8n 版本。学习阶段不建议使用 latest
pgvector/pgvector:pg16	PostgreSQL 16 +pgvector 扩展，后续知识库和向量检索会用到
DB_POSTGRESDB_HOST=postgres	在 Docker Compose 内部，服务之间通过服务名访问，所以这里写 postgres
N8N_HOST=localhost	本地学习时使用 localhost；部署到服务器时应改为实际访问所使用的域名
N8N_SECURE_COOKIE=false	本示例使用本地 HTTP 访问，因此设为 false；正式部署并启用 HTTPS 时应使用安全 Cookie 设置
./local_files:/files	把本地目录映射到容器内的/files 上，后续读写文件时会用到

示例中的my_super_secret_db_password只是演示值。实际使用时，请换成读者自己的强密码，并妥善保存。

2.3.4　启动 n8n

确认docker-compose.yml已保存后，推荐使用以下命令：

```
docker-compose up -d
```

较早版本的Docker Compose也可能使用以下旧命令：

```
docker compose up -d
```

两者的区别主要是命令形式不同：docker-compose是旧版Docker的独立命令，docker compose是当前Docker Compose使用的子命令形式。

启动后，可以使用以下命令查看容器状态：

```
docker ps
```

正常情况下，应该能看到n8n和postgres都处于运行状态，其中PostgreSQL健康检查通过后，n8n才会启动。

2.3.5　访　n8n 并创建管理员账号

n8n启动完成后，打开浏览器访问以下网址：

```
http://localhost:5678
```

第一次访问时会进入管理员账号创建页面，如图2-6所示。按照页面提示完成邮箱、姓名和密码的初始化，后续登录和管理实例都会依赖这个账号。

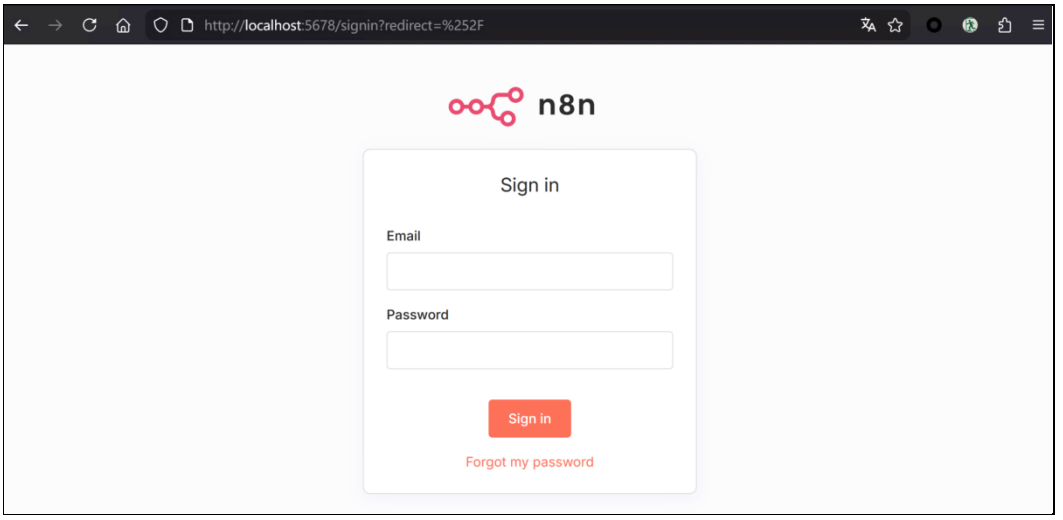

图 2-6　首次访问 n8n 时的管理员账号创建页面

创建完成后，就会进入n8n的主界面。至此，一个可用于后续案例的n8n环境就搭建好了。

2.3.6　证 PostgreSQL 和 pgvector

因为后续会用到数据库和向量检索，所以建议在启动n8n后进行一次简单验证。

首先进入PostgreSQL容器：

```
docker exec -it postgres psql -U n8n -d n8n
```

然后在PostgreSQL命令行中执行以下命令：

```
CREATE EXTENSION IF NOT EXISTS vector;
SELECT '[1,2,3]'::vector + '[4,5,6]'::vector;
```

如果看到类似下面的结果，就说明pgvector可以正常使用：

```
 ?column?
----------
 [5,7,9]
```

最后退出PostgreSQL命令行：

```
\q
```

不是每次启动后都要进行验证，只需在第一次部署后确认PostgreSQL和pgvector环境正常。

2.3.7　常用管理命令

在后续的使用过程中，会频繁启动、停止、查看日志和重启服务，常用命令如表2-5所示。

表 2-5　n8n Docker Compose 环境常用管理命令

操　作	命　令
启动服务	docker-compose up -d
停止服务	docker-compose down
查看容器	docker ps

（续表）

操　　作	命　　令
查看日志	docker logs -f n8n
重启 n8n	docker restart n8n
重启全部服务	docker-compose restart

如果修改了docker-compose.yml，通常重新执行以下命令即可：

```
docker-compose up -d
```

Docker会根据配置变化重建需要更新的容器，并保留已挂载的卷数据。

2.3.8　常见　　排查

问题1：浏览器打不开http://localhost:5678。

可能原因：容器没有启动成功，或者端口被占用。

解决办法：执行以下命令查看容器和日志。

```
docker ps
docker logs -f n8n
```

如果日志中提示端口冲突，可以把Compose文件中的端口映射改成其他端口，例如：

```
ports:
   - "5679:5678"
```

之后访问http://localhost:5679。

问题2：n8n连接不上PostgreSQL。

可能原因：数据库密码不一致，或DB_POSTGRESDB_HOST写错。

解决办法：确认n8n服务里的DB_POSTGRESDB_PASSWORD与postgres服务里的POSTGRES_PASSWORD完全一致；在Docker Compose中，数据库主机应写服务名postgres，不要写127.0.0.1。

问题3：本地文件读写失败。

可能原因：文件没有存放到映射目录，或路径写错。

解决办法：把要处理的文件存放到my-n8n/local_files/，在n8n节点中使用容器内路径/files/文件名，并确认容器对相应目录具有读写权限。

2.3.9　更新与备份

n8n更新频繁。跟随本书学习时，不建议频繁升级；学完全书案例后，再根据需要统一升级。

如果要升级n8n版本，基本步骤如下：

（1）备份数据库和n8n_data，并确认local_files等业务文件已按需要备份。

（2）修改docker-compose.yml中的n8n镜像版本。

（3）拉取镜像并重建容器：

```
docker-compose pull
docker-compose up -d
```

n8n更新与备份要点，如表2-6所示。

表 2-6　n8n 更新与备份要点

数　据	说　明
postgres_data	保存工作流、凭据、执行记录、Data Table 等数据
./n8n_data	保存 n8n 本地配置和实例相关配置数据
./local_files	保存你在工作流中读写的本地文件

不要认为只要导出工作流JSON就备份完成了。很多运行数据和凭据并不包含在单个工作流文件中。对于本书当前的部署方式，至少应同时备份PostgreSQL数据、n8n_data目录以及需要长期保留的本地业务文件。

2.3.10　其他　署方式

除了Docker Compose，n8n也可以通过npm、进程管理工具或源码方式部署。例如，在通过npm直接运行n8n时，可以结合PM2等Node.js进程管理工具管理进程。这类方式适合熟悉Node.js运维、希望直接在宿主机上运行n8n的用户；源码部署则主要面向参与n8n核心开发或深度二次开发的用户。

本书后续案例统一以Docker Compose环境为准。这样可以保证数据库、文件路径、容器网络和依赖版本始终保持一致，减少不必要的环境差异。

2.4　置中文界

n8n的官方界面当前主要使用英文，尚未提供官方维护的完整中文界面。对于初学者来说，这会带来一定门槛，尤其在第一次接触节点名称、参数选项和错误提示时，很容易被英文术语卡住。

当前比较常见的中文界面配置有两类：一类是使用浏览器翻译插件，另一类是社区中文Docker镜像。本节会介绍这两种方案，但建议优先使用浏览器翻译插件。原因很简单：它不改动n8n本体，稳定、可回退，也方便我们逐步熟悉英文术语。

2.4.1　使用浏览器翻译插件（推荐）

配置n8n中文界面最推荐的方式是使用浏览器翻译插件（如“沉浸式翻译”），或者浏览器自带的翻译能力。它的原理是在浏览器侧翻译页面文本，不会修改n8n的Docker镜像、容器文件或数据库。

开启沉浸式翻译后，可以先对照菜单和节点配置项，确认中文辅助说明不会影响英文术语识别，效果如图2-7所示。

图 2-7　使用沉浸式翻译查看 n8n 界面

这种方式适合大多数学习场景，主要优点有三个：

（1）无侵入：不修改n8n文件，不影响升级。
（2）中英对照：保留Webhook、Credential、Expression等英文术语，方便后续查官方文档和社区资料。
（3）随时关闭：遇到翻译不准确时，可以临时关闭翻译查看英文原文。

配置步骤如下：

01 在 Chrome、Edge 或 Firefox 应用商店中搜索并安装“沉浸式翻译”。
02 在浏览器中打开 n8n 地址，例如 http://localhost:5678。
03 单击浏览器右上角的翻译插件图标。
04 选择翻译当前网页；如果后续经常使用，可以设置为“总是翻译此网站”。

使用浏览器翻译插件时，个别节点名称可能翻译得不够准确，这是正常现象。本书后续会尽量采用“英文节点名+中文解释”的写法，例如HTTP Request（HTTP请求）、Credential（凭据），帮助读者建立术语对应关系。

2.4.2　使用社区中文 Docker　像

如果希望整个界面直接显示中文，可以考虑使用社区维护的中文镜像。例如，社区项目n8n-i18n-chinese提供了相关汉化资源和镜像。进入项目页后，重点查看镜像名称、支持版本和更新说明，这些信息通常集中在项目说明页中，如图2-8所示。

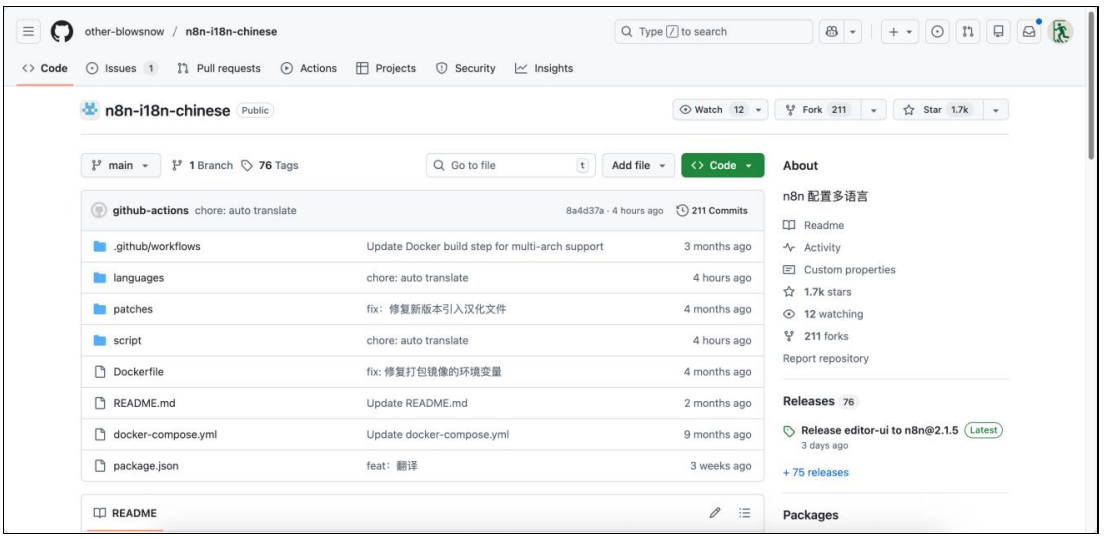

图 2-8　n8n-i18n-chinese 项目的镜像与版本说明入口

这种方式的优点是界面更接近中文软件，适合内部培训或团队成员普遍不适应英文界面的场景。它也有明显的缺点：汉化内容会依赖n8n版本，一旦n8n更新，社区镜像可能需要等待同步。

如果读者已经使用Docker Compose部署，只需要修改docker-compose.yml中n8n服务的镜像即可。原配置类似这样：

```
n8n:
    image: n8nio/n8n:2.10.3
```

如果确认要使用社区中文镜像，可以改成：

```
n8n:
    image: blowsnow/n8n-chinese:latest
```

其余数据库、端口、环境变量和挂载目录保持不变。修改保存后，在docker-compose.yml所在目录执行以下命令：

```
docker-compose up -d
```

待容器重建完成后，刷新浏览器访问n8n。如果镜像和当前版本兼容，界面会显示为中文。

上面的镜像名称和维护状态可能会随社区项目变化而变化。正式使用前，请以社区项目最新说明为准。如果遇到界面异常，最简单的回退方式是把镜像改回n8nio/n8n:2.10.3，再重新执行docker-compose up-d。

2.4.3　不推荐在本书主线中手动替换容器文件

网上也有一些教程会教我们下载汉化包，然后把文件挂载或覆盖到n8n容器内部目录中。这种方式更灵活，但也更容易因为版本路径变化而失效。

由于本书后续案例都基于Docker Compose环境，并且重点是学习n8n工作流设计，而不是维护汉化文件，因此不建议把“手动替换前端文件”作为主线操作。除非明确知道自己在做什么，否则优先使用浏览器翻译或社区中文Docker镜像即可。

2.4.4　方案对比与选择建议

如果读者还在两种方案之间犹豫，可以按表2-7对比它们的稳定性、升级便利性和学习成本等。

表 2-7　浏览器翻译插件与社区中文 Docker　像的选择建议

评估维度	浏览器翻译插件	社区中文 Docker　像
稳定性	高，不依赖 n8n 内部文件	中等，依赖社区维护和版本同步
升级便利性	高，n8n 可正常升级	中等，升级前要确认镜像是否跟进
学习英文术语	好，中英可对照	较弱，容易脱离官方术语
配置难度	低，只安装浏览器插件	中等，需要修改 Docker 镜像
适用场景	个人学习、长期使用、需要保持官方运行环境的场景	内部培训、短期降低英文门槛

个人学习优先使用浏览器翻译插件；如果是团队内部培训，可以临时使用社区中文Docker镜像；生产环境建议优先使用官方镜像，并通过浏览器侧翻译辅助阅读，减少因社区镜像版本同步和第三方修改带来的额外不确定性。

2.5　变　　置

环境变量（Environment Variables）是控制n8n行为的重要方式。可以把它理解为n8n的外部配置面板：数据库怎么连、服务地址是什么、时区用哪个，都可以通过环境变量设置。

2.5.1　为什么　要环境变

配置环境变量的好处有以下3点：

（1）配置和程序分离，升级镜像时不用改内部代码。

（2）敏感信息可以集中管理，避免写进工作流，但仍需要妥善保护环境变量和相关配置文件。

（3）同一套部署文件可以适配本地、测试和生产环境。

在Docker Compose中，环境变量通常写在environment字段里；也可以先放入.env文件，再在Compose文件中引用。

2.5.2　常用环境变　　查

学习阶段先掌握常用环境变量即可，如表2-8所示。

表 2-8　n8n 常用环境变

变	作　　用	示　　例
DB_TYPE	数据库类型	postgresdb
DB_POSTGRESDB_HOST	PostgreSQL 主机名	postgres
DB_POSTGRESDB_DATABASE	数据库名	n8n
DB_POSTGRESDB_USER	数据库用户	n8n
DB_POSTGRESDB_PASSWORD	数据库密码	自行设置强密码
N8N_HOST	n8n 对外主机名	localhost 或域名
N8N_PROTOCOL	访问协议	http/https
N8N_PORT	服务端口	5678
WEBHOOK_URL	生产 Webhook 对外地址	https://n8n.example.com/
GENERIC_TIMEZONE	默认时区	Asia/Shanghai
N8N_ENCRYPTION_KEY	凭据加密密钥	生产环境务必固定保存

其中需要注意的是WEBHOOK_URL和N8N_ENCRYPTION_KEY：

- 如果Webhook地址不正确，外部平台回调会失败。
- 如果加密密钥丢失，已有凭据可能无法解密。

2.5.3　使用.env 管理　置

推荐先把敏感配置放到.env文件中，再由docker-compose.yml引用。需要注意，.env本质上仍是明文配置文件，因此应严格控制文件权限和访问范围。

例如，在.env文件中存放如下敏感配置：

```
POSTGRES PASSWORD=change me
N8N ENCRYPTION KEY=replace with a long random string
GENERIC_TIMEZONE=Asia/Shanghai
```

在Compose文件中引用上述配置：

```
environment:
    - DB_POSTGRESDB_PASSWORD=${POSTGRES_PASSWORD}
    - N8N_ENCRYPTION_KEY=${N8N_ENCRYPTION_KEY}
    - GENERIC_TIMEZONE=${GENERIC_TIMEZONE}
```

.env文件不要提交到公开仓库。在团队协作时，可以提供.env.example，只保留变量名和示例值。

环境变量不需要一次性全部记住。学习阶段先掌握数据库、地址、时区和加密密钥这几类。后续遇到文件大小、执行记录清理、日志级别等问题时，再查官方文档即可。

2.6　权　路径与数据存储

完成Docker Compose的部署后，还需要弄清楚两类路径：一类是n8n自己保存实例配置和本地持久化数据的路径，另一类是后续工作流读写本地文件时使用的路径。若路径不清楚，后面进行文件上传、图片生成、视频落盘、数据库备份时就很容易出错。

回顾2.3.3节中的网络配置：

```
volumes:
    - ./n8n_data:/home/node/.n8n
    - ./local_files:/files
```

第二行表示宿主机的./n8n_data会映射到容器内的/home/node/.n8n；第三行表示宿主机的./local_files会映射到容器内的/files。后续只要说“把文件放到/files”，实际就是把文件放到宿主机的local_files目录中。

2.6.1　.n8n 目录　保存什么

在本书的部署方案中，n8n的核心数据主要保存在PostgreSQL中，例如工作流、用户、凭据密文、执行记录和Data Table数据。./n8n_data目录并不是全部数据的唯一来源，它更多保存本地配置、加密相关信息，以及根据配置存放的二进制数据、社区节点等本地数据。

常见内容包括：config、binaryData/或其他本地存储目录、nodes/等。其中config尤为重要：如果没有显式配置N8N_ENCRYPTION_KEY，n8n会在首次启动时自动生成加密密钥，并将其保存在~/.n8n目录的配置文件中。凭据密文本身在数据库里，但解密需要密钥，所以迁移或备份时不能忽略这个目录。

在n8n 2.x的常规运行模式下，二进制数据默认使用filesystem模式保存到本地文件系统；队列模式则默认使用database模式。因此，在本书当前的普通Docker Compose部署中，n8n_data目录中的本地存储内容也应纳入备份范围。

如果没有配置PostgreSQL，n8n可能会默认使用SQLite，并在.n8n目录下生成database.sqlite。本书后续案例统一基于PostgreSQL + Docker Compose，因此这里不介绍SQLite路径。

2.6.2　/files 路径怎么用

/files是后续案例中最常用的本地文件路径。比如把一个文件存放在宿主机中：

```
my-n8n/local_files/product-manual.pdf
```

那么在n8n节点中应该填写：

```
/files/product-manual.pdf
```

反过来，如果某个工作流把图片写入/files/output.png，我们就可以在宿主机的my-n8n/local_files/output.png中找到它。请读者记住：宿主机中存放local_files，n8n里写/files。这个规则会贯穿后面的图文生成、文件读取和视频落盘案例。

2.6.3　备份哪些内容

很多新手只备份n8n_data，恢复后发现工作流不见了。原因是本书使用PostgreSQL保存核心数据，所以完整备份至少包含PostgreSQL数据库、n8n_data目录、需要长期保留的local_files目录以及docker-compose.yml和.env等配置文件。

推荐的最小备份命令如下：

```
# 导出PostgreSQL数据库
docker exec -t postgres pg_dump -U n8n > n8n_full_backup.sql

# 打包本地配置与文件目录
tar -czf n8n_files_backup.tar.gz docker-compose.yml n8n_data local_files
```

恢复时，先还原目录，再启动PostgreSQL，最后在目标数据库为空或已做好覆盖准备的前提下把SQL导回数据库。导入命令示例如下：

```
cat n8n_full_backup.sql | docker exec -i postgres psql -U n8n
```

如果在.env或环境变量中设置了N8N_ENCRYPTION_KEY，那么一定要把这个值一起保存。密钥丢失后，即使数据库还在，已有凭据也无法解密。

2.6.4　常见权

Docker部署中最常见的权限报错是：EACCES:permission denied, open '/home/node/.n8n/config'。这是因为n8n官方容器默认使用UID为1000的node用户运行，而宿主机上的n8n_data或local_files目录可能是由root或其他用户创建，导致容器内用户没有相应的读写权限。官方n8n镜像当前也明确以USER node运行。

在Linux宿主机上，如果确认挂载目录的属主导致权限问题，可以在docker-compose.yml所在目录执行以下命令：

```
mkdir -p n8n_data local_files
sudo chown -R 1000:1000 ./n8n_data ./local_files
docker restart n8n
```

其中1000:1000通常对应官方镜像中的node用户，但实际环境中仍应以容器所使用的UID和GID为准。有些教程建议在Compose文件中加入user: root，这样确实能绕过权限问题，但不建议作为常规方案。更安全的做法是保持n8n使用默认用户运行，只修正宿主机挂载目录的权限。

本节只需要记住以下3点：

（1）PostgreSQL保存核心结构化数据，n8n_data保存本地配置和需要持久化的本地数据。

（2）后续工作流读写本地文件统一使用/files，对应宿主机的local_files。

（3）遇到权限问题时优先检查挂载目录的属主和权限，在Linux环境下可根据容器UID/GID使用chown进行修正，不要轻易让容器以root身份运行。

2.7　n8n 快　开发入

完成环境搭建后，我们先用两个练习建立手感：第一个是普通工作流，第二个是AI Agent。它们不追求复杂度，但会覆盖n8n开发最常见的动作：添加节点、配置参数、单步测试、查看输出。

2.7.1　第一个工作流：定时网站监控

第一个工作流的目标很简单：每隔一段时间访问一个网页，确认它是否能正常返回。这个练习会用到Schedule Trigger（定时触发器）和HTTP Request（HTTP请求）两个节点。具体操作如下：

01 在 n8n 主界面单击 Create workflow 按钮，进入空白画布。

02 单击画布中央的 Add first step（添加第一步）按钮，搜索并添加 Schedule Trigger，如图 2-9 所示。

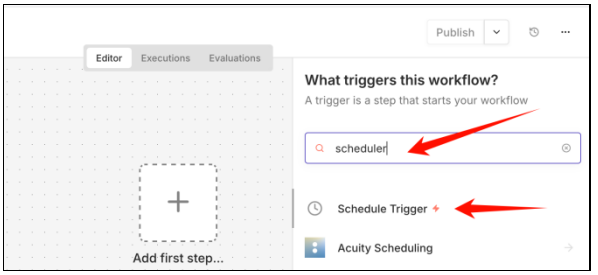

图 2-9　搜索并添加 Schedule Trigger 节点

03 在 Schedule Trigger 的配置面板中，将 Trigger Interval（触发间隔）设置为 Minutes，数值设置为 1。这表示工作流发布后每 1 分钟触发一次。

04 单击 Schedule Trigger 节点右侧的“+”按钮，搜索并添加 HTTP Request 节点。

05 在 HTTP Request 节点配置面板中，设置 Method 为 GET，URL 填写 https://www.baidu.com 或其他能正常访问的网站，如图 2-10 所示。

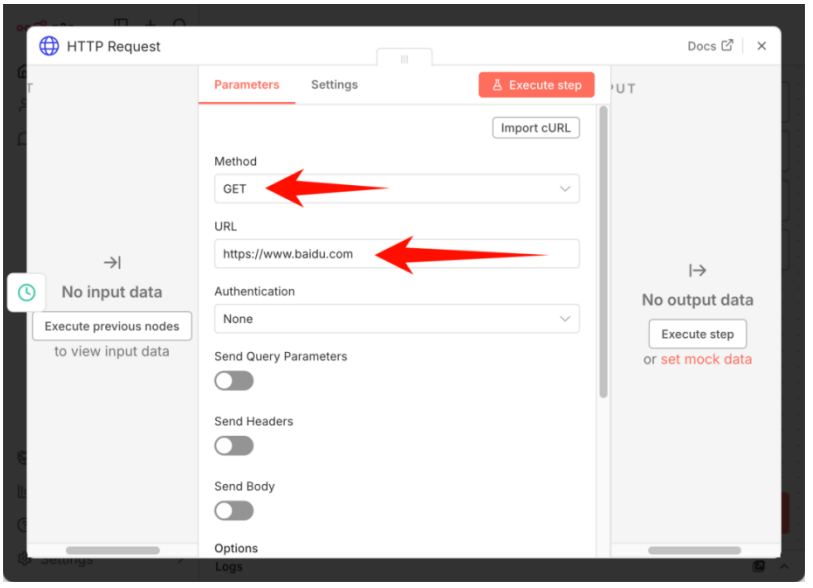

图 2-10　HTTP Request 节点的配置

06 单击 HTTP Request 节点右上角的 Execute step（执行步骤）按钮，先测试单个节点。执行成功后，右侧 Output 面板会显示网页响应内容，如图 2-11 所示。

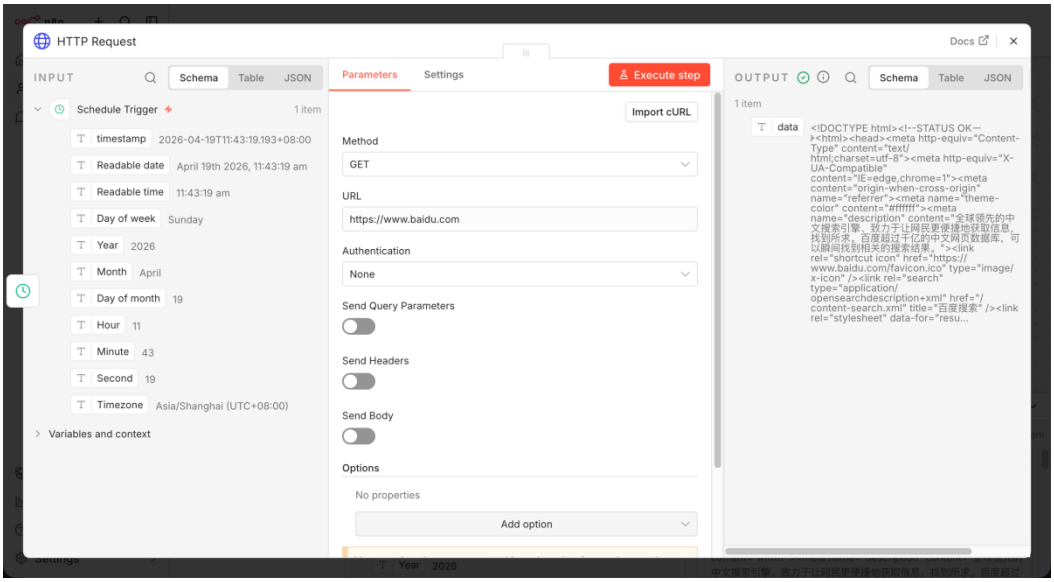

图 2-11　HTTP Request 节点的输出结果

07 回到画布，单击 Execute workflow（执行工作流）按钮，测试整个工作流。节点变为绿色，表示流程执行成功。

工作流执行成功后，右侧Output面板会显示响应数据；后续排错也会反复回到这个位置检查字段结构。

如果工作流执行失败，优先检查URL是否写错、网络是否能访问目标网站。我们也可以故意输入一个错误地址，观察n8n如何在Output面板中显示错误信息。后续在调试复杂工作流时，会经常使用单节点测试。

2.7.2　第一个 AI Agent：AI 聊天助手

接下来搭建一个最小可用的AI Agent。这个练习的目标不是介绍所有AI节点，而是让读者理解n8n中Agent的基本组成：聊天入口、Agent节点、Chat Model，以及可选的Memory。具体操作步骤如下：

01 新建一个工作流。

02 单击 Add first step 按钮，搜索并添加 When chat message received 节点。它会提供一个可测试的聊天入口。每收到一条聊天消息，都会触发一次工作流执行。

03 单击该节点右侧的“+”按钮，添加 AI Agent 节点，如图 2-12 所示。

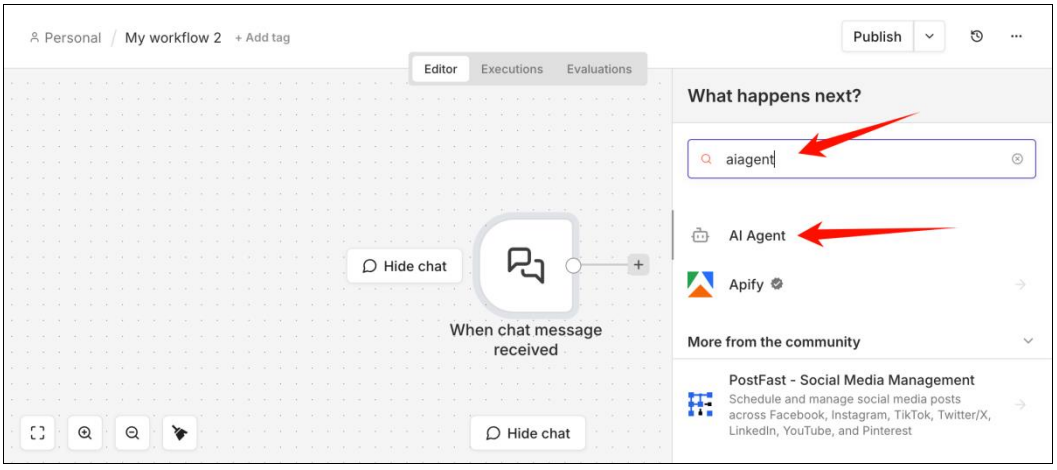

图 2-12　添加 AI Agent 节点

04 在 AI Agent 节点下方的 Chat Model 位置单击“+”按钮，选择一个可用的大模型节点，例如 DeepSeek、OpenAI、Gemini 或 OpenRouter。

05 按界面提示创建 Credential（凭据），填入对应平台的 API Key，保存后选择可用模型。

保存凭据前，如果当前凭据类型提供连接测试功能，可以使用Test Connection、Retry或界面中类似的测试入口确认API Key是否可用，测试通过后再继续连接模型，如图2-13所示。

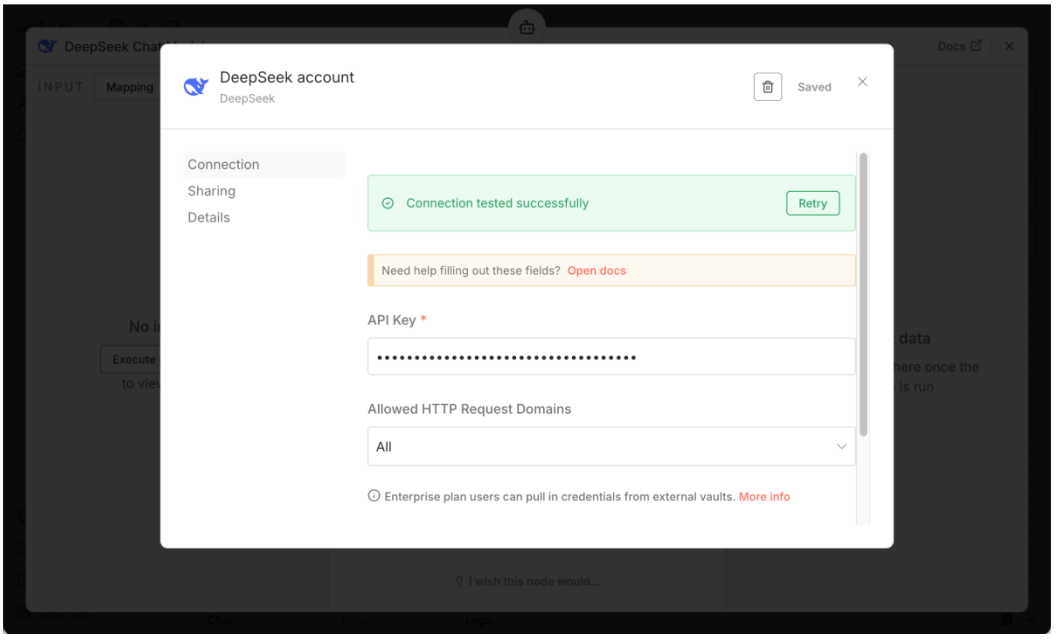

图 2-13　创建并测试大模型凭据

06 可选：在 AI Agent 下方的 Memory 位置单击“+”按钮，添加 Simple Memory。它适合学习和测试，可以让 Agent 在同一会话中保留一定范围的历史消息作为上下文。

07 保存工作流后，打开聊天测试入口，输入一个简单问题，例如“你好，你叫什么名字”，观察 Agent 的回复。

如果配置正确，聊天窗口会显示我们的提问和Agent的回复，这一步可以用来确认聊天入口、Agent和模型三者已经连通，如图2-14所示。

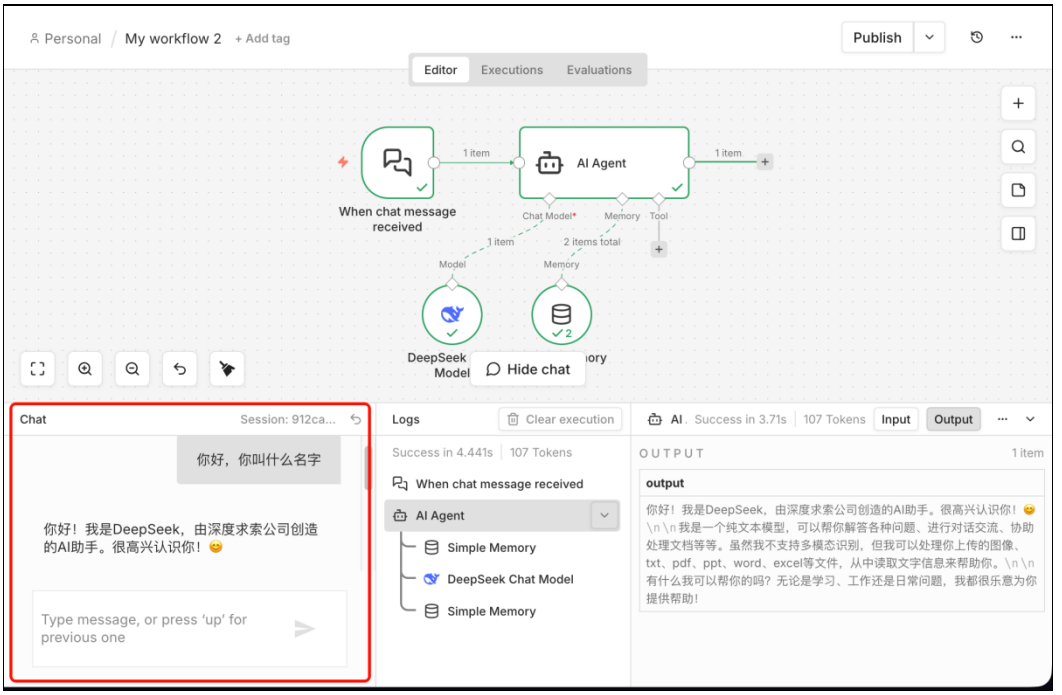

图 2-14　在聊天窗口测试 AI Agent

这里不展开介绍DeepSeek、OpenRouter等平台的完整注册和充值流程。读者只需要记住：Chat Model提供“大脑”，Credential保存API Key，Memory负责为Agent提供会话历史等上下文信息。后续真正介绍AI案例时，会结合具体场景讲模型选择、提示词和结构化输出。

2.7.3　调试习惯

建议从第一个工作流开始，就养成3个习惯。

（1）先测单节点，再测全流程。每新增一个关键节点，就用Execute step检测输出是否符合预期。

（2）重点看Input和Output。Input告诉我们上游传来了什么，Output告诉我们当前节点产生了什么结果。

（3）尽量拖曳复制字段。从Input面板或其他可引用节点的数据面板拖曳字段到表达式输入框，可以减少字段名拼写错误的问题。

复杂工作流不是一次性搭成的，而是一段一段验证出来的。后续所有案例都会沿用这个方法。