AI 使用完全指南：从零基础到高效工作流

> **副标题**：一个普通人的 AI 进化之旅
>
> **目标读者**：想要用 AI 提效工作，但不知道从何入门的普通人
>
> **学习目标**：读完本书，你能独立设计并运行自己的 AI 工作流

目录

1. 第一章：第一次对话 - 基础对话与提示词工程
2. 第二章：知识的诅咒 - RAG 与私有知识库
3. 第三章：工具人的觉醒 - 工具调用与自动化
4. 第四章：工具箱的混乱 - MCP 标准化协议
5. 第五章：重复劳动的噩梦 - Skill 技能封装
6. 第六章：失忆症的困扰 - 持久化记忆系统
7. 第七章：创作与沉淀 - 工具链与工作流
8. 第八章：复杂任务的编排 - Agent 与工作流
9. 第九章：进化之路 - 技术选型与最佳实践
10. 第十章：实战案例集 - 完整项目实战

如何使用本书

🎯 阅读路径

路径 A：速通模式（2小时） - 只读每章的【故事场景】和【Aha Moment】 - 快速了解概念，建立整体认知

路径 B：实践模式（1周） - 按顺序阅读，每章跟着【实践练习】动手做 - 适合想要立即应用的学习者

路径 C：深度模式（1个月） - 完整阅读 + 完成所有练习 + 实现第十章案例 - 适合想要成为 AI 工作流专家的学习者

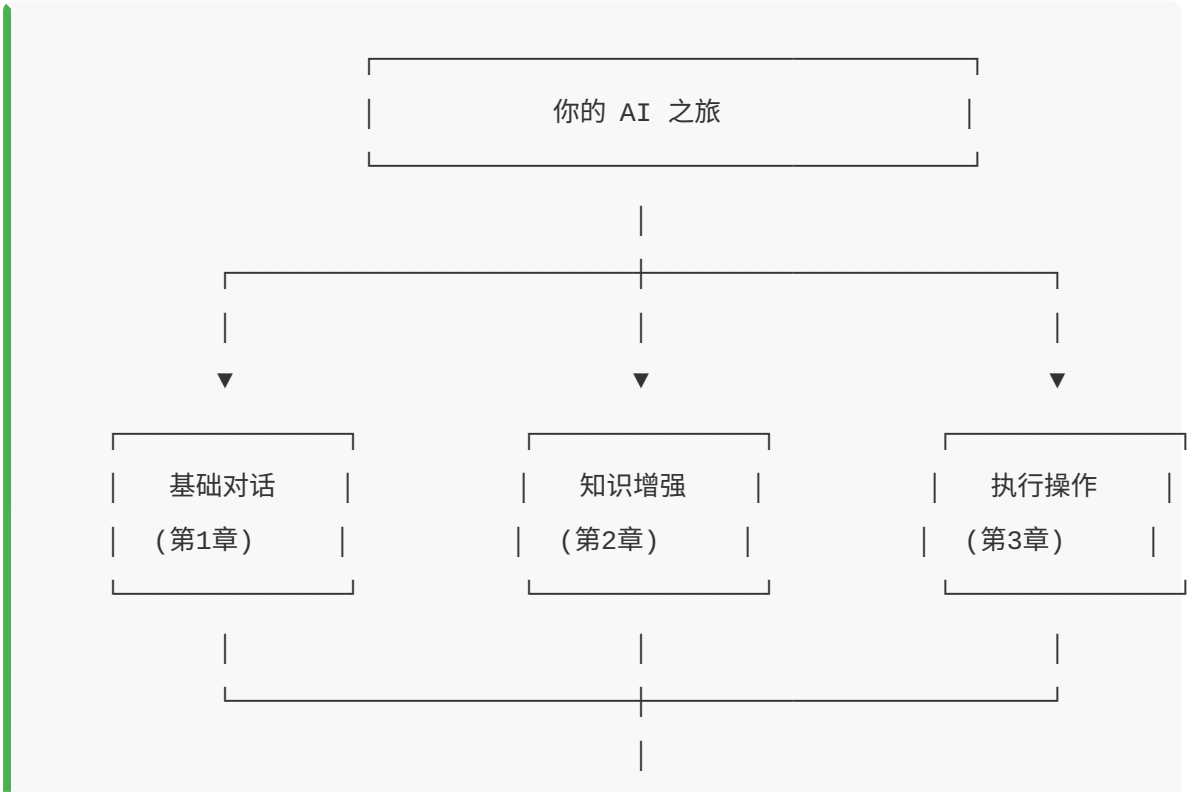
🛠️ 环境准备

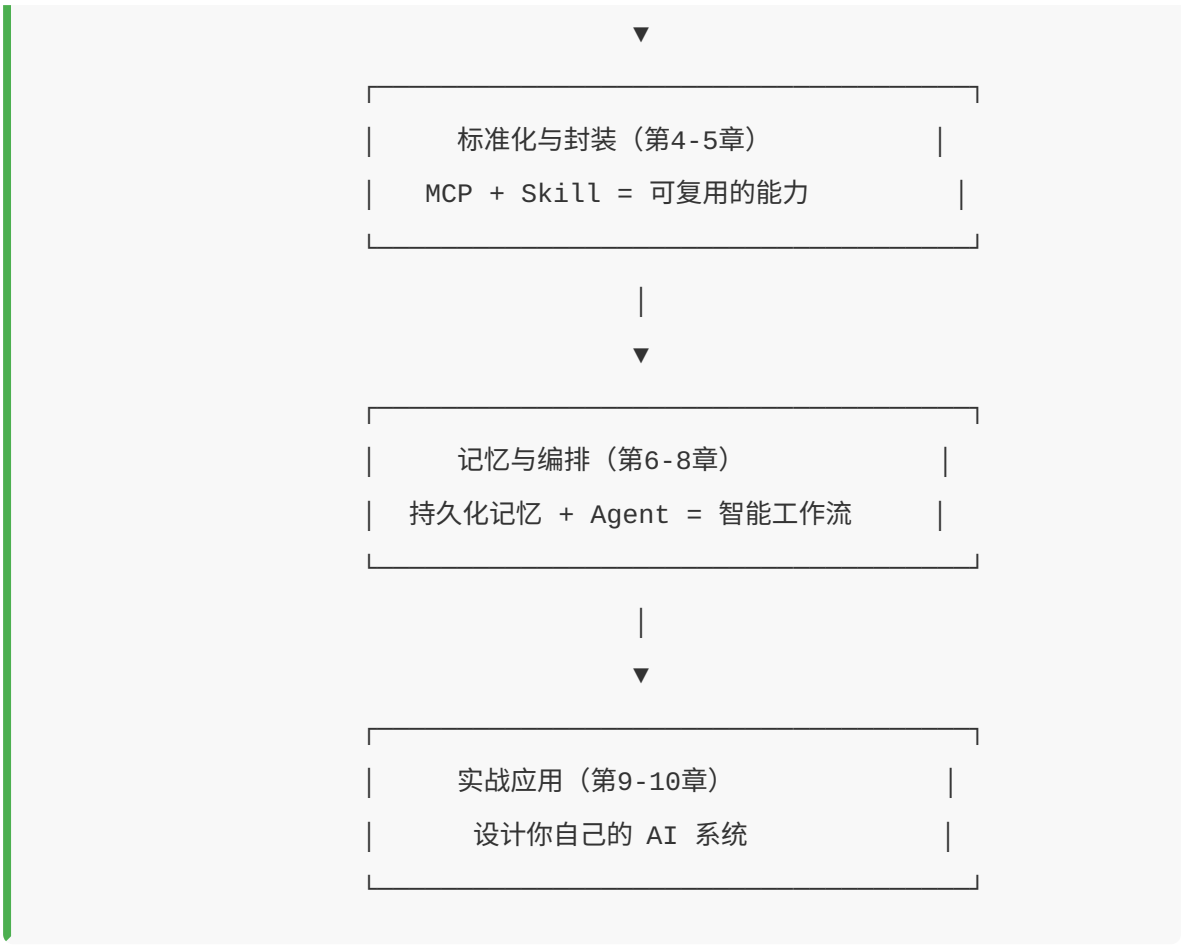
本书示例基于 **OpenClaw** 平台，你可以： 1. 使用自己的 OpenClaw 环境 2. 或使用其他支持工具调用的 AI 平台（Cursor、Claude Desktop 等） 3. 部分概念也适用于纯网页版 ChatGPT/Claude

💡 核心建议

> **不要只看不练！**
>
> 每章的【实践练习】都设计了具体任务，只有动手做过，概念才能真正内化。

学习地图





核心概念速查

概念	一句话解释	类比
Prompt	告诉 AI 你想要什么	像给助理下达指令
RAG	AI + 你的私有知识库	像给 AI 配了一套内部资料
Tool	AI 能调用的外部功能	像给 AI 装上了手脚
MCP	工具的标准接口	像 USB 接口，统一规范
Skill	封装好的 AI 工作流	像手机里的 App
Memory	AI 的持久化记忆	像 AI 的笔记本

概念	一句话解释	类比
Agent	能自主决策的 AI	像能独立完成任务的助理

让我们开始这段 AI 进化之旅吧！

第一章：第一次对话 —— 从 "聊天" 到 "对话"

🎬 故事场景：小白的第一次

时间：某天晚上，你刚听说 ChatGPT 很火 **地点**：电脑前，打开了一个 AI 聊天窗口 **心情**：好奇 + 期待

你输入了第一个问题：

```
你：什么是人工智能？

AI：人工智能（Artificial Intelligence，简称 AI）是指由人制造出来的
系统所表现出来的智能。它包括学习、推理、感知、理解语言等能力...

[省略 500 字详细的定义和解释]
```

你心想："哇，太厉害了！比我百度查到的还详细！"

于是你继续问：

```
你：帮我写一份周报，我是做销售的。

AI：好的，以下是一份销售周报模板：

本周工作总结：
1. 完成客户拜访 5 次
2. 签约新客户 2 家
3. 跟进潜在客户 10 个
```

> 下周计划：
> 1. 继续跟进重点客户
> 2. 参加行业展会
> 3. 完成季度目标冲刺
>
> [一个看起来挺像回事的周报]

你心想："还行，虽然有点模板化，但改改就能用了。"

⚡ Aha Moment 1：幻觉的代价

几天后...

你在准备一个产品发布会的演讲稿，需要介绍公司的新产品功能。你问 AI：

> 你：我们公司的"智能客服系统"有哪些核心功能？帮我列一下。
>
> AI：根据智能客服系统的常见功能，包括：
>
> 1. **智能语音识别** - 支持多语言语音输入
> 2. **自然语言理解** - 准确理解用户意图
> 3. **知识图谱问答** - 基于知识库的智能回答
> 4. **情感分析** - 识别用户情绪并调整回复策略
> 5. **多轮对话管理** - 支持复杂场景的多轮交互
> 6. **人工接管** - 复杂问题无缝转接人工
> 7. **数据分析看板** - 实时监控服务质量
>
> [看起来非常专业的功能列表]

你觉得这个回答很好，直接贴到了 PPT 里。

发布会当天...

客户举手提问："您刚才提到的'知识图谱问答'功能，我想看看实际演示。"

你懵了："啊...这个功能...呃...其实还在开发中..."

客户："那你 PPT 上怎么写着'已上线功能'？"

场面一度尴尬。

🤯 顿悟时刻

事后你反思：

> "原来 AI 不是'知道'我公司的产品，它只是'编'了一个看起来合理的产品功能列表！
>
> 它说得那么自信，我还以为是基于什么内部信息...
>
> **AI 会'自信地胡说'！**"

这就是 AI 领域的专业术语 —— **"幻觉"（Hallucination）**。

幻觉问题示意图

图 1-1：AI 幻觉现象 —— 当 AI 遇到知识盲区时，会生成看似合理但实际错误的内容

📚 概念引入：提示词工程（Prompt Engineering）

为什么会产生幻觉？

AI（大语言模型）的工作原理是：**基于概率预测下一个词**。

它并不"理解"内容的真实性，只是根据训练数据中的模式，生成"看起来最合理"的回答。

当你问： - ❌"你们公司的产品有什么功能？" → AI 不知道你的公司 - ❌"最新的行业数据是什么？" → AI 的知识有截止日期 - ❌"这个技术方案可行吗？" → AI 无法验证技术细节

解决方案：更好的提问方式

技巧 1：给上下文（Context）

❌**差：**

> 写一份周报。

✅**好：**

> 我是某互联网公司的产品经理，负责用户增长。
> 这周主要工作：
> - 完成了 A/B 测试方案设计
> - 分析了上月的留存数据
> - 和开发团队讨论了 3 个需求
>
> 请帮我写一份周报，要求：
> 1. 包含具体数据和成果
> 2. 语言简洁专业
> 3. 下周计划要可执行

技巧 2：角色设定（Role）

❌**差：**

> 解释一下什么是区块链。

✅**好：**

> 你是一位资深的区块链技术专家，有 10 年从业经验。
>
> 请向一位完全不懂技术的 60 岁老人解释什么是区块链，要求：
> 1. 使用生活中的类比

> 2. 不要出现专业术语（如果必须出现，请解释）
> 3. 控制在 300 字以内

技巧 3：思维链（Chain of Thought）

❌**差：**

> 这道数学题怎么做：一个水池有 2 个进水管和 1 个出水管...

✅**好：**

> 请解决这道数学题：一个水池有 2 个进水管和 1 个出水管...
>
> 要求：
> 1. 一步一步展示你的思考过程
> 2. 每步说明你在算什么
> 3. 最后给出答案并验证

技巧 4：明确边界（Constraints）

❌**差：**

> 给我一些营销方案。

✅**好：**

> 背景：我们是做高端茶叶的电商品牌，客单价 500-2000 元。
> 目标：提升复购率
> 限制条件：
> - 预算 5 万元以内
> - 不要打折促销（不符合品牌调性）
> - 需要可量化的效果指标

```
请给出 3 个具体的营销方案，每个方案包含：
1. 核心策略
2. 执行步骤
3. 预期效果
4. 成本估算
```

💻 代码示例：OpenClaw 中的提示词实践

在 OpenClaw 中，你可以这样与 AI 对话：

示例 1：结构化输出

```
# 让 AI 以特定格式返回数据
prompt = """
分析以下用户反馈，提取关键信息：

用户反馈："物流太慢了，等了5天才到。但是包装很好，茶叶质量不错，会回购。"

请以 JSON 格式返回：
{
  "sentiment": "正面/中性/负面",
  "issues": ["问题1", "问题2"],
  "positives": ["优点1", "优点2"],
  "action_items": ["建议1", "建议2"]
}
"""

# AI 会返回结构化的 JSON 数据
```

示例 2：Few-Shot 示例

```
# 给 AI 几个示例，让它按示例的格式回答
prompt = """
请将以下描述转化为产品标签：

示例1：
描述："这款跑鞋采用轻量透气材质，适合长距离跑步"
标签：["运动鞋", "跑步", "透气", "轻量"]

示例2：
描述："高端商务笔记本，碳纤维外壳，续航12小时"
标签：["笔记本", "商务", "高端", "长续航"]

现在请处理：
描述：""
标签：
"""
```

示例 3：多轮对话优化

```
# 第一轮：建立上下文
user: """
我是做跨境电商的，主要在亚马逊卖家居用品。
我会在后续对话中咨询一些运营问题。
请先确认你理解了我的背景。
"""

ai: """
已理解您的背景：
- 平台：亚马逊
- 类目：家居用品
- 业务模式：跨境电商
```

```
请随时提出您的问题，我会基于这个背景给出建议。
"""

# 第二轮：具体问题
user: """
我的广告 ACOS 太高了，有什么优化建议?
"""

# AI 会基于"亚马逊家居用品"的上下文给出针对性建议
```

🏃 实践练习

练习 1：识别幻觉

阅读以下 AI 回答，找出可能存在的问题：

> **用户问题**："2024年诺贝尔文学奖得主是谁？"
>
> **AI 回答**："2024年诺贝尔文学奖得主是中国作家残雪。她在2024年10月凭借作品《新世纪爱情故事》获得该奖项，成为首位获得诺贝尔文学奖的中国女作家。"

问题分析：

```
1. AI 的知识截止日期可能早于 2024 年 10 月
2. AI 可能"编造"了一个看似合理的答案
3. 需要核实：残雪是否真的获得了 2024 年诺奖?
```

练习 2：改写提示词

将以下模糊的问题改写成清晰、具体的提示词：

原始问题：

> 帮我写个文案。

改写示例：

> 背景：我们是新锐咖啡品牌"晨醒"，主打精品手冲咖啡。
> 目标受众：25-35岁一线城市白领。
> 使用场景：小红书种草文案。
> 产品卖点：
> - 100%阿拉比卡豆
> - 烘焙后48小时内发货
> - 附赠手冲教程
>
> 要求：
> 1. 标题吸引人点击
> 2. 正文300字左右
> 3. 语气亲切自然，像朋友推荐
> 4. 包含3-5个相关话题标签
>
> 请输出3个不同风格的版本。

练习 3：实际应用

选择一个你工作中的真实场景，使用今天学到的技巧写一个提示词。

场景：__

提示词：

> [在这里写你的提示词]

📝 本章小结

核心要点

1. **AI 会"幻觉"** —— 它会自信地生成错误信息，不要盲信
2. **给上下文** —— 告诉 AI 你的背景、限制、目标
3. **定角色** —— 让 AI 以特定身份回答问题
4. **要思考过程** —— 复杂问题要求 AI 展示推理过程
5. **设边界** —— 明确格式、长度、风格等限制

常用模板

```
你是一位[角色]，擅长[专业领域]。

背景：[你的具体情况]

任务：[具体要做什么]

要求：
1. [要求1]
2. [要求2]
3. [要求3]

格式：[期望的输出格式]
```

下一步预告

下一章，我们将解决一个问题：

> "如果 AI 只能基于训练数据回答，我怎么让它了解我的私有信息（公司内部文档、个人笔记等）？"

这就是 **RAG（检索增强生成）** 技术要解决的问题。

📖 扩展阅读

- Prompt Engineering Guide - 英文，但非常全面
- OpenAI 官方提示词工程指南
- Claude 提示词最佳实践

✅ 自我检查清单

完成本章后，你应该能够：

- [ ] 识别 AI 回答中可能的"幻觉"
- [ ] 写出一个包含角色、上下文、任务的完整提示词
- [ ] 使用约束条件控制 AI 输出格式
- [ ] 解释为什么"给更多上下文"能得到更好的回答

如果还有不清楚的地方，建议回顾【Aha Moment】和【代码示例】部分，然后完成【实践练习】。

第二章：知识的诅咒 —— 当 AI 遇到你的私有知识

🎬 故事场景：内部文档的困境

时间：周一早上，你需要汇总公司最新的报销政策 **地点**：办公桌前，面对着散落在各处的文档 **心情**：焦虑 + 低效

场景 1：通用答案的尴尬

你问 AI：

```
你：公司的差旅报销标准是什么？

AI：根据一般企业的差旅报销标准，通常包括：
1. 飞机票：经济舱全额报销
2. 火车票：二等座全额报销
3. 住宿：一线城市 500元/晚，二线城市 300元/晚
4. 餐补：100元/天
...

[一个看起来很合理的通用标准]
```

你："这和我公司实际标准不一样啊...我们公司一线城市住宿限额是 800 元..."

问题：AI 只能基于训练数据回答，它不知道你公司的**实际规定**。

场景 2：复制粘贴的噩梦

于是你换了个策略：

步骤 1：打开公司内部 Wiki，找到《财务报销制度 v3.2.pdf》 **步骤 2**：选中全文，复制 **步骤 3**：粘贴到 AI 对话框 **步骤 4**：等 AI 总结

结果：

> AI：根据您提供的文档，公司差旅报销标准为...
> [详细且准确的总结]

第一次："太好了！这就是我要的！"

场景 3：重复劳动的地狱

一周后，另一个同事问你：

> 同事：上次那个报销标准，能发我一份吗？
>
> 你：好的，我问问 AI。

于是你再次： 1. 打开 Wiki 2. 复制粘贴 50 页文档 3. 等 AI 总结 4. 转发给同事

一个月后，报销政策更新了：

> 你：糟了，住宿标准变了，我得重新复制粘贴一遍...

第 N 次后：

> 你："为什么每次都要我手动搬运？AI 就不能记住我公司的东西吗？！"

⚡ Aha Moment 2：AI 需要长驻记忆

深夜加班的你突然顿悟：

> "每次都要复制粘贴内部资料，这不就跟每次见医生都要重复一遍病史一样愚蠢吗？
>
> **AI 也需要一个'病历本'，记录我的私有知识！**
>
> 这样我问问题的时候，它先查'病历本'，再基于里面的信息回答，不就不需要我每次都搬运了吗？"

这就是 **RAG（Retrieval-Augmented Generation，检索增强生成）** 的核心思想。

RAG 工作原理

图 2-1：RAG 工作原理 —— AI 在回答前先检索你的私有知识库

📚 概念引入：RAG（检索增强生成）

什么是 RAG？

简单理解：**RAG = AI + 你的私有知识库**

传统 AI 对话：

> 用户提问 → AI 基于训练数据回答

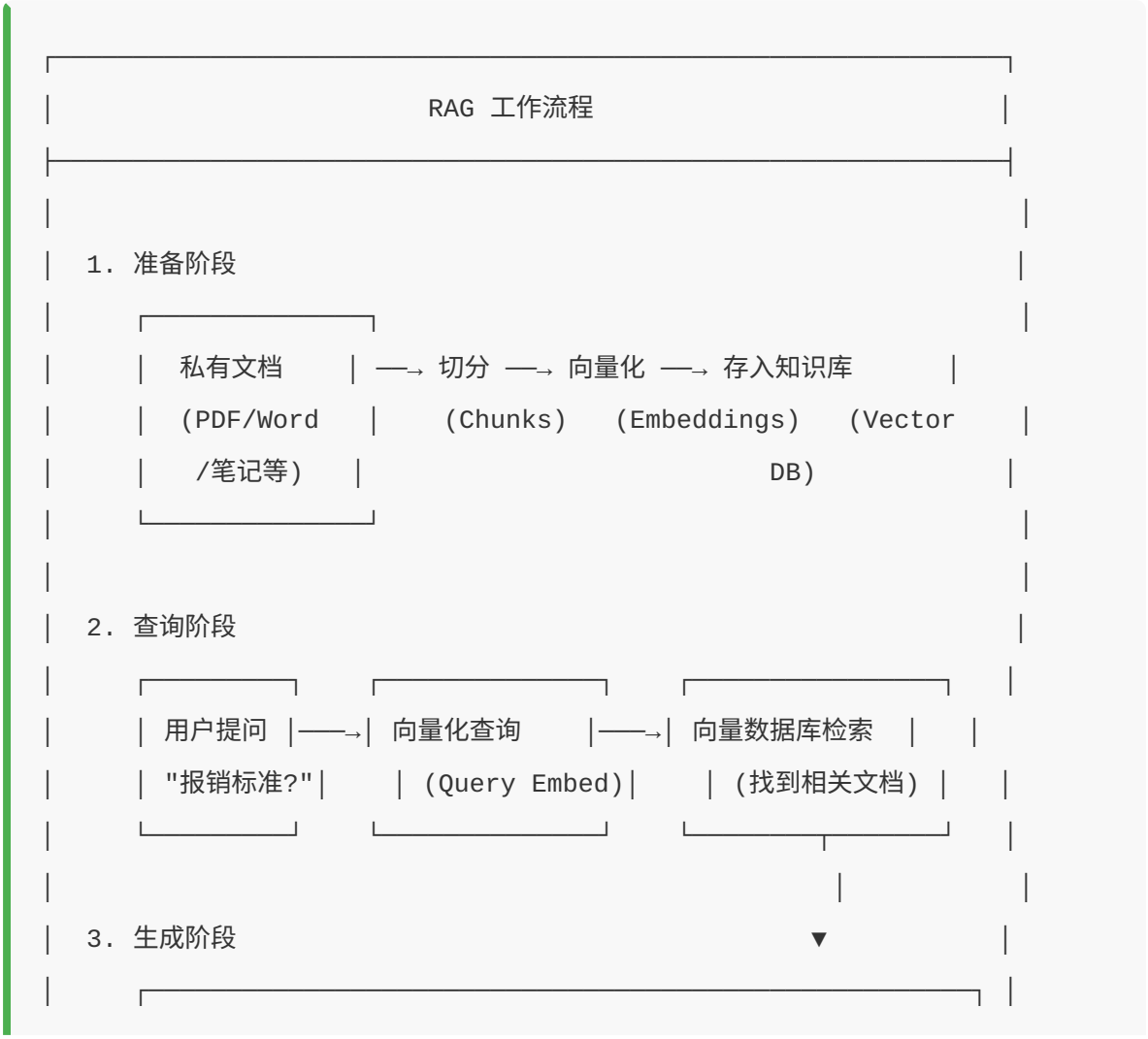
RAG 增强的对话：

> 用户提问 → 在知识库中检索相关内容 → AI 基于检索结果回答

RAG 能解决什么问题?

问题	传统 AI	RAG 增强
公司内部政策	❌不知道	✅查文档后回答
个人笔记整理	❌无法访问	✅基于笔记回答
最新行业数据	❌知识过期	✅查最新资料
专业领域知识	❌泛泛而谈	✅精准回答
私有产品信息	❌完全错误	✅准确描述

RAG 的工作流程



```
|      |  AI 基于检索到的相关内容生成回答                          | |
|      |  "根据《财务报销制度v3.2》..."                            | |
|      └──────────────────────────────────────────────────────┘ |
|                                                                |
└────────────────────────────────────────────────────────────────┘
```

核心组件

1. **文档加载器**：读取 PDF、Word、网页等各种格式
2. **文本切分器**：将长文档切成小块（Chunks）
3. **向量化模型**：将文本转化为向量（Embeddings）
4. **向量数据库**：存储和检索向量
5. **检索器**：根据查询找到最相关的文档块
6. **生成器**：基于检索结果生成回答

💻 代码示例：OpenClaw 中的 RAG 实践

方案 1：基于文件的简单 RAG（MEMORY.md）

OpenClaw 使用的是**文件化记忆系统**，虽然不像传统 RAG 那样使用向量数据库，但原理类似：

MEMORY.md 就是你的"知识库"：

```
# MEMORY.md - 我的长期记忆

## 公司信息

### 财务报销标准（2024年版）
- **住宿标准**:
  - 一线城市（北上广深）：800元/晚
  - 二线城市：500元/晚
```

```
  - 其他城市：300元/晚
- **交通标准**：
  - 飞机：经济舱，需提前3天审批
  - 高铁：二等座
- **餐补**：150元/天（含早中晚）

### 产品信息
- **产品A**：面向企业客户，年费 10 万起
- **产品B**：面向个人用户，月费 99 元

## 项目信息
...
```

使用方式：

```
# 在对话中，AI 会自动读取 MEMORY.md
# 当你问相关问题时，AI 会基于里面的内容回答

user: "我出差住酒店，每晚能报多少？"

# AI 会先读取 MEMORY.md 找到报销标准
# 然后回答：
ai: """
根据您的公司报销标准（MEMORY.md 中的记录）：

住宿费用限额取决于出差城市：
- 一线城市（北上广深）：800元/晚
- 二线城市：500元/晚
- 其他城市：300元/晚

请问您出差到哪个城市？我可以帮您计算具体限额。
"""
```

方案 2：每日笔记（Daily Notes）

对于临时性的、不需要长期记忆的信息：

```
# 2024-03-15.md

## 今天的会议记录
- 下午 3 点和客户 X 开会，讨论了 Y 需求
- 决定下周二提交方案

## 临时的想法
- 可以尝试用 AI 优化周报生成流程
```

工作流程：

```
新项目开始
    ↓
创建项目目录 + 记忆文件
    ↓
每次对话前自动读取相关记忆
    ↓
对话中自动更新记忆
    ↓
项目结束后归档到长期记忆
```

方案 3：结合搜索工具

对于需要实时检索的信息：

```
# 伪代码示例
workflow:
  1. 用户提问关于最新政策
  2. 调用 web_search 工具搜索最新信息
```

```
3. 读取本地 MEMORY.md 的私有知识
4. AI 结合两者生成回答
```

🏃 实践练习

练习 1：建立你的 MEMORY.md

创建以下文件：`~/.openclaw/workspace/MEMORY.md`

```
# MEMORY.md - 我的个人知识库

## 工作信息

### 我的职责
- 职位：产品经理
- 负责产品：用户增长平台
- 团队规模：5人

### 常用工具
- 项目管理：飞书多维表格
- 数据分析：SQL + Python
- 文档协作：飞书文档

### 关键项目
- **项目A**：留存优化（进行中，目标提升 5%）
- **项目B**：新用户引导改版（规划中）

## 个人偏好

### 沟通习惯
- 喜欢简洁直接的沟通
- 周报格式：数据 + 结论 + 下一步
```

```
- 会议时间：上午 10-12 点效率最高

### AI 使用习惯
- 生成内容后需要我确认
- 重要决策需要邮件确认
- 默认使用 Kimi K2.5 模型
```

练习 2：测试 RAG 效果

完成练习 1 后，向 AI 提问：

```
"我现在在做的项目是什么？目标是什么？"
```

预期结果：AI 应该能基于 MEMORY.md 准确回答。

练习 3：更新知识

修改 MEMORY.md 中的某个信息（比如项目状态从"进行中"改为"已完成"），然后再次提问，观察 AI 的回答变化。

📝 本章小结

核心要点

1. **AI 不知道你的私有知识** —— 训练数据截止于某个时间点，不包含你的内部信息
2. **RAG = AI + 知识库** —— 让 AI 在回答前先查你的资料
3. **不要重复搬运** —— 一次性整理到知识库，永久受益
4. **保持更新** —— 知识库需要定期维护，过期信息要及时更新

RAG 的应用场景

场景	示例	实现方式
企业知识库	公司制度、产品手册	文档向量化存储
个人知识管理	学习笔记、灵感记录	MEMORY.md + Daily Notes
客服系统	常见问题、解决方案	RAG + 自动回复
专业领域	法律条文、医学知识	专业知识库

OpenClaw 的记忆系统

下一步预告

下一章，我们将解决一个问题：

> "AI 能基于我的知识回答问题了，但它只能'说'，不能'做'。我怎么让它帮我执行实际操作？"

这就是 **Tool Use（工具调用）** 技术要解决的问题。

📖 扩展阅读

- RAG 系统架构详解
- 向量数据库入门
- LangChain RAG 教程

✅ 自我检查清单

完成本章后，你应该能够：

- [ ] 解释什么是 RAG 以及为什么需要它
- [ ] 建立自己的 MEMORY.md 文件
- [ ] 区分长期记忆和短期记忆的使用场景
- [ ] 设计一个简单的知识管理流程

💡 进阶思考

思考 1：如果公司有大量文档（几千份），文件化记忆系统还适用吗？

思考 2：RAG 和 Fine-tuning（微调）有什么区别？什么时候用哪个？

思考 3：如何保证知识库中的敏感信息不被泄露？

（提示：这些问题的答案会在后续章节逐步揭晓。）

第三章：工具人的觉醒 —— 从"能说话"到"能动手"

🎬 故事场景：周报生成之痛

时间：周五下午5点，你还没开始写周报 **地点**：办公桌前，面对着多个系统 **心情**：焦虑 + 手忙脚乱

场景 1：数据的海洋

作为产品经理，你的周报需要包含： - 本周数据（DAU、留存率、转化率）——在数据后台 - 项目进展——在飞书文档 - 问题汇总——在JIRA - 下周计划——在脑中

传统流程：

> 1. 打开数据后台，截图 → 保存到桌面
> 2. 打开飞书，找到文档 → 复制相关段落
> 3. 打开 JIRA，筛选本周问题 → 截图
> 4. 打开 Word，开始写周报
> 5. 把刚才搜集的材料拼凑起来
> 6. 排版、调格式
> 7. 保存、发送邮件
>
> 耗时：2小时 😫

你问 AI：

> 你：帮我写一份周报，我是产品经理，负责用户增长。

> AI：好的，以下是一份产品经理周报模板：
>
> 本周工作总结：
> 1. 数据分析：本周 DAU 环比上升 5%，主要得益于...
> 2. 项目进展：完成了 A/B 测试方案设计...
> 3. 问题处理：解决了 3 个用户反馈的问题...
>
> [看起来很专业的模板]

你："又是模板...里面的数据都是编的，根本不能用！"

场景 2：AI 的局限

你意识到问题：

> AI 能：生成文字、提供建议、回答问题
> AI 不能：
> - ❌ 登录数据后台查数字
> - ❌ 读取你的飞书文档
> - ❌ 发送邮件给老板
> - ❌ 创建日程提醒

你心想："如果 AI 能直接操作这些系统就好了..."

⚡ Aha Moment 3：AI 需要一双手

深夜 11 点，你终于写完周报

盯着屏幕上的 AI 对话框，你突然顿悟：

> "AI 就像一个知识渊博但手脚被绑住的人。
>
> 它能告诉我'应该怎么做'，但不能'真的去做'。

> **如果给 AI 装上'手脚'——让它能调用各种工具，那它不就真的能帮我干活了吗？**
>
> 查数据？调用数据 API。 发邮件？调用邮件服务。 写文档？调用飞书 API。
>
> AI 负责思考，工具负责执行！"

这就是 **Tool Use（工具调用）** 的核心思想。

Tool Use 工作原理

图 3-1：Tool Use —— AI 不仅能说话，还能调用工具执行操作

📚 概念引入：Tool Use（工具调用）

什么是 Tool Use?

Tool Use = AI + 可执行的外部功能

传统 AI：

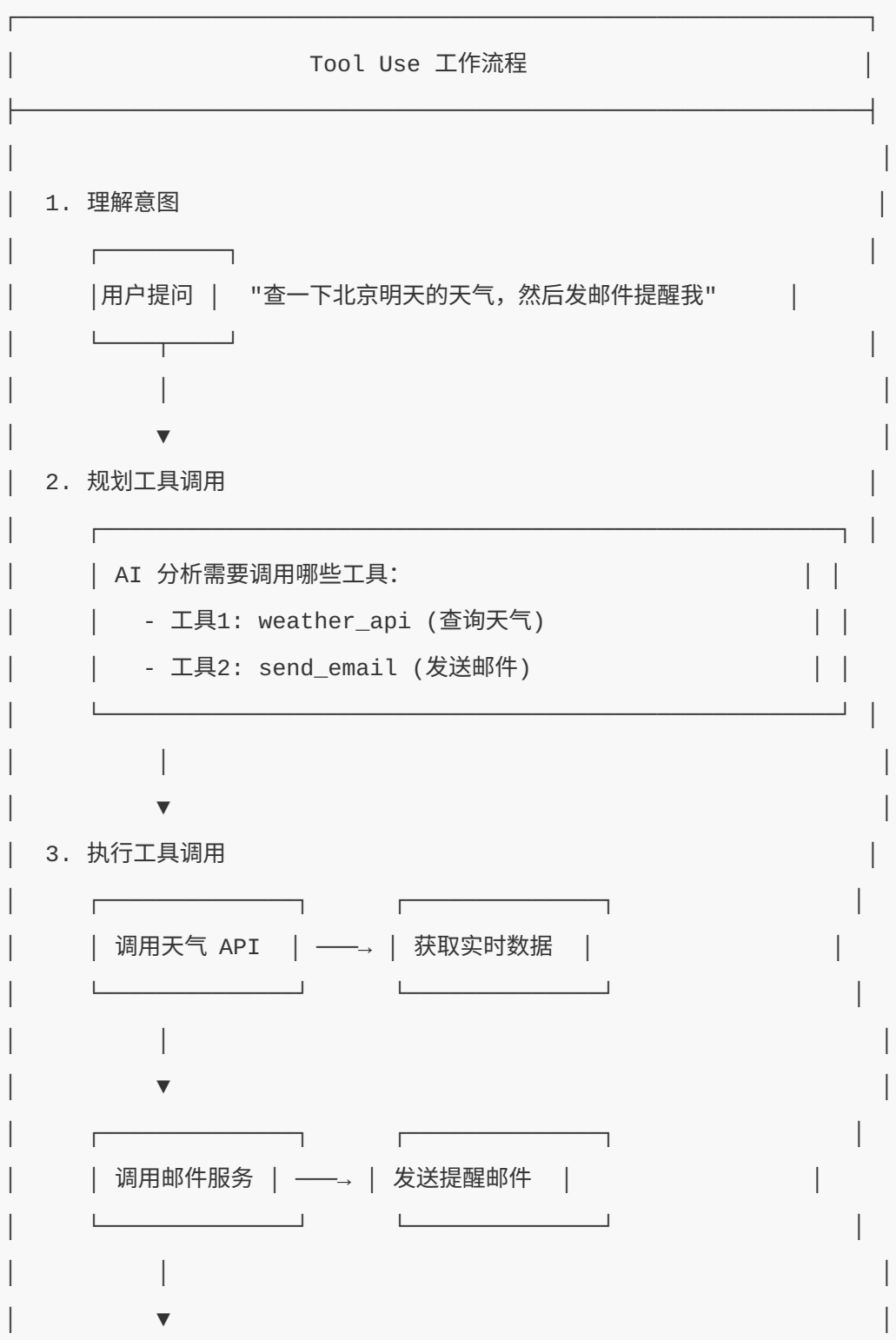
```
用户：查一下明天的天气
AI：北京明天晴，气温 15-25 度

[但 AI 其实"不知道"天气，只是基于训练数据"猜测"]
```

Tool Use 增强的 AI：

```
用户：查一下明天的天气
AI：我需要调用天气查询工具
     ↓
    调用 weather_api(city="北京", date="明天")
     ↓
    返回：{"temp": "15-25°C", "weather": "晴"}
     ↓
AI：北京明天晴，气温 15-25 度
```

```
[AI 真实获取了实时数据]
```

Tool Use 的工作流程



```
|  4. 整合结果                                                    |
|     ┌──────────────────────────────────────────────────────┐ |
|     | AI 基于工具返回的结果生成最终回答:                      | |
|     | "已查询到北京明天晴，15-25°C。提醒邮件已发送至..."     | |
|     └──────────────────────────────────────────────────────┘ |
|                                                               |
└──────────────────────────────────────────────────────────────┘
```

常见工具类型

工具类型	示例	用途
数据查询	SQL 查询、API 调用	获取实时数据
文件操作	读写文件、生成 PDF	处理文档
通讯工具	发送邮件、飞书消息	信息传递
网络工具	网页抓取、API 请求	获取外部信息
系统命令	执行脚本、运行程序	自动化操作
日历工具	创建日程、查询时间	时间管理

💻 代码示例：OpenClaw 中的 Tool Use

示例 1：基础工具调用

OpenClaw 提供了丰富的工具集：

```
# 查询天气（使用 web_search 工具）
weather = web_search("北京明天天气预报")

# 读取文件（使用 read 工具）
```

```
content = read("/path/to/file.md")

# 执行命令（使用 exec 工具）
result = exec("git status")

# 发送消息（使用 message 工具）
message(action="send", content="周报已生成", target="坤哥")

# 操作飞书文档（使用 feishu_doc 工具）
feishu_doc(action="read", doc_token="xxx")
```

示例 2：自动化周报生成

```
# 完整的周报生成工作流
workflow "weekly_report":

    # 步骤1: 从数据库获取数据
    data = exec("""
        cd /project && python3 get_weekly_data.py
    """)

    # 步骤2: 读取项目文档
    project_docs = feishu_doc(
        action="read",
        doc_token="project_doc_id"
    )

    # 步骤3: AI 分析并生成周报
    report = ai_generate("""
       基于以下数据生成周报:

       数据: {data}
       项目背景: {project_docs}
```

```
        格式要求：
        1. 本周数据亮点
        2. 项目进展
        3. 遇到的问题
        4. 下周计划
    """)

    # 步骤4: 保存到文件
    write(
        file_path="/tmp/weekly_report.md",
        content=report
    )

    # 步骤5: 发送邮件
    send_qq_email(
        to="boss@company.com",
        subject="周报 - 第X周",
        content=report
    )

    return "周报已生成并发送"
```

示例 3：浏览器自动化

```
# 使用 browser 工具操作网页
workflow "check_yami_product":

    # 打开浏览器
    browser(action="start")

    # 访问页面
    browser(action="open", targetUrl="https://www.yami.com/product/123")

    # 获取页面信息
```

```
    snapshot = browser(action="snapshot")

    # AI 分析页面内容
    product_info = ai_extract("从快照中提取产品名称和价格", context=snapshot)

    # 关闭浏览器
    browser(action="close")

    return product_info
```

🏃 实践练习

练习 1：设计一个简单工具链

设计一个"早安助手"工具链： 1. 查询今日天气 2. 查询今日日程 3. 生成早安提醒消息 4. 发送到手机/飞书

伪代码：

```
# 在这里写你的设计
```

练习 2：实际运行一个工具

在 OpenClaw 中尝试：

```
# 读取你的 MEMORY.md 文件
content = read("~/.openclaw/workspace/MEMORY.md")
print(content[:500])  # 打印前500字符
```

练习 3：自动化脚本

写一个自动化脚本，每天自动： 1. 检查某个网站是否有更新 2. 如果有更新，发送通知

📝 本章小结

核心要点

1. **AI 只有"脑"没有"手"** —— 它能思考但不能直接操作外部系统
2. **Tool Use 给 AI 装上手脚** —— 让它能调用 API、操作文件、发送消息
3. **组合使用威力大** —— 多个工具串联形成工作流
4. **安全第一** —— 敏感操作（发送邮件、删除数据）需要确认

Tool Use vs RAG

特性	RAG	Tool Use
解决什么问题	AI 不知道私有知识	AI 不能执行操作
数据来源	静态知识库	实时 API/系统
典型应用	问答、文档理解	自动化、数据处理
组合效果	RAG + Tool = 智能助手	

下一步预告

下一章，我们将解决一个问题：

> "工具越来越多，每个工具的配置方式都不一样，管理起来很混乱，怎么办？"

这就是 **MCP（Model Context Protocol）** 要解决的问题 —— 工具的标准化接口。

✅ 自我检查清单

完成本章后，你应该能够：

- [ ] 解释 Tool Use 的概念和价值
- [ ] 列举 3 种常用工具类型及其用途
- [ ] 设计一个简单的工具链工作流
- [ ] 使用 OpenClaw 的至少 3 个工具

💡 进阶思考

思考 1：如果 AI 可以执行任何命令，如何保证安全性？

思考 2：Tool Use 和传统的"脚本自动化"有什么区别？

思考 3：什么时候应该使用 Tool Use，什么时候直接用脚本？

第四章：工具箱的混乱 —— 当工具越来越多

> ## 🎬 故事场景：工具大爆炸

时间：使用 AI 三个月后 **地点**：你的 AI 工作台 **心情**：兴奋但混乱

场景 1：工具越来越多

刚开始用 AI 时，你只有几个简单的工具： - ✅ 查天气 - ✅ 发邮件 - ✅ 读取文件

三个月后，你的工具箱已经膨胀： - 📧 邮件工具（QQ邮箱、企业邮箱、Gmail） - 📄 文档工具（飞书文档、Notion、Google Docs） - 🌐 浏览器工具（Playwright、Selenium、Browser Use） - 💬 通讯工具（飞书、钉钉、Slack、微信） - 📊 数据工具（SQL查询、Excel处理、图表生成） - 🔍 搜索工具（百度、Google、内部搜索引擎） - 🤖 其他 API（天气、股票、翻译、OCR...）

问题出现了...

场景 2：配置地狱

邮件工具的配置：

```
# email_tools.yaml
smtp_server: smtp.qq.com
port: 587
username: your_qq@qq.com
```

```
password: your_auth_code
ssl: true
```

飞书工具的配置：

```
# feishu_tools.yaml
app_id: cli_xxx
app_secret: xxx
verification_token: xxx
webhook_url: https://xxx
```

浏览器工具的配置：

```
# browser_tools.yaml
headless: false
timeout: 30000
user_agent: Mozilla/5.0...
proxy: http://proxy:8080
```

每个工具： - ❌配置格式不一样 - ❌认证方式不一样
- ❌调用方式不一样 - ❌错误处理不一样 - ❌文档风格不一样

你心想："学一个新工具的成本太高了！"

场景 3：切换成本

周一：用飞书写文档

```
feishu_doc(action="write", content="...")
```

周二：客户要求用 Notion

```
# 完了，我要重新学 Notion 的 API
notion_page_create(title="...", content="...")
# 参数格式完全不同！
```

周三：又要用 Google Docs

```
# 又要重新学...
gdocs_create_document(title="...", body="...")
```

你崩溃："为什么没有一个统一的方式？"

⚡ Aha Moment 4：工具需要通用语言

深夜整理工具文档时，你突然顿悟：

> "电脑有 USB 接口，无论什么设备，插上就能用。
>
> 003e 为什么 AI 工具不能有类似的'标准接口'？
>
> 003e **如果所有工具都遵循同一个协议：** 003e - 发现工具的方式统一 003e - 调用工具的格式统一 003e - 传递参数的规范统一 003e - 返回结果的格式统一 003e 003e 那我学一次，就能用所有工具！"

这就是 **MCP（Model Context Protocol，模型上下文协议）** 的核心思想。

MCP 工作原理

图 4-1：MCP —— 工具的标准接口，类似 USB 协议

📚 概念引入：MCP（模型上下文协议）

什么是 MCP?

MCP = AI 工具的 USB 接口标准

类比： - 🔌 USB：让键盘、鼠标、U盘、打印机都能插到电脑上 - 🔄 MCP：让各种 AI 工具都能被 AI 统一调用

没有 MCP 的混乱

> 工具A的调用方式：
>
> ```
> call_tool("weather", {"city": "北京"})
> ```
>
> 工具B的调用方式：
>
> ```
> invoke_api("email.send", to="xxx", body="yyy")
> ```
>
> 工具C的调用方式：
>
> ```
> run_action("browser.navigate", url="...")
> ```
>
> 每个工具都不一样！学习成本高，切换困难。

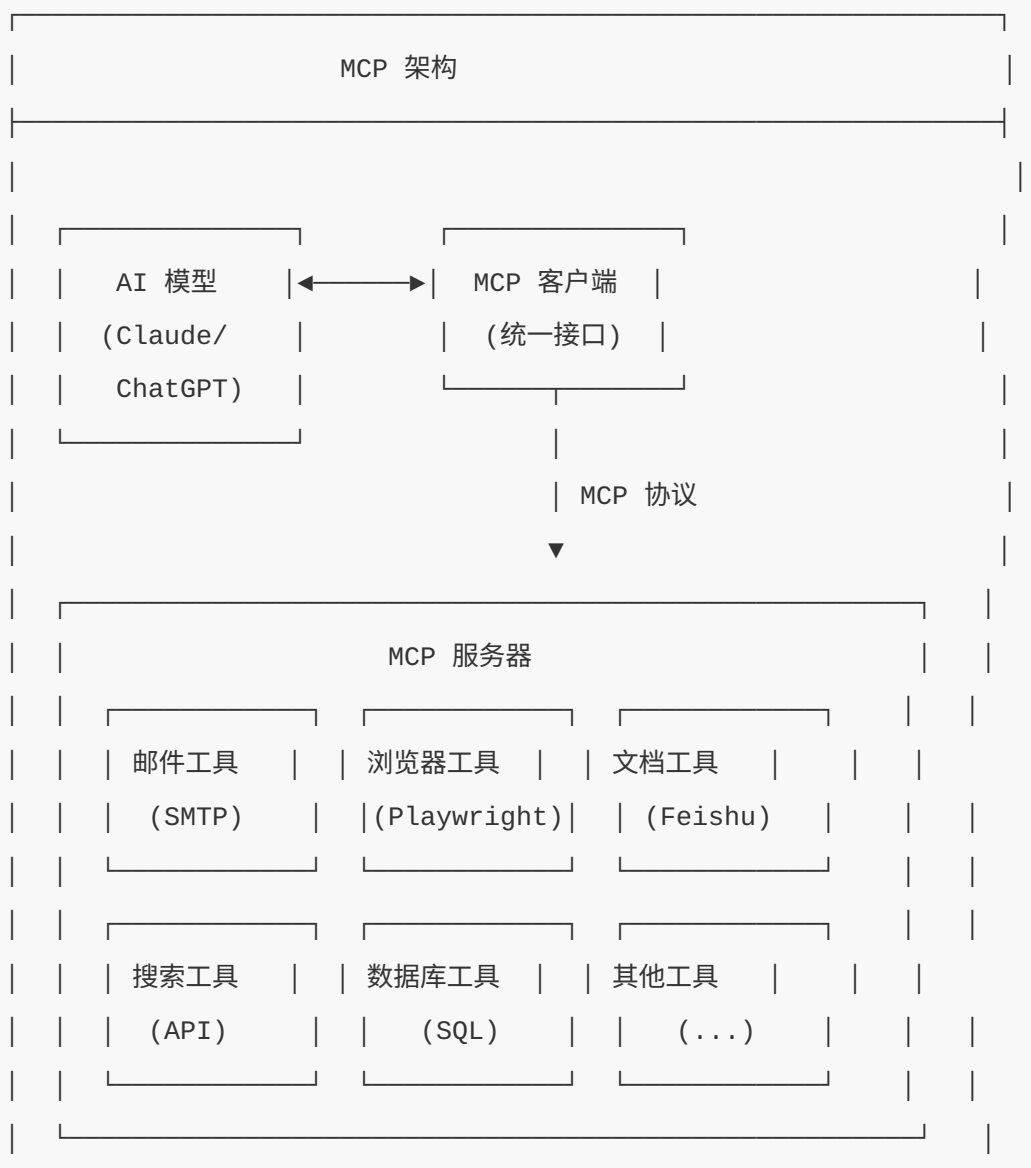
有 MCP 的统一

> 所有工具的统一调用方式：
>
> ```
> {
>   "tool": "weather",
>   "params": {"city": "北京"}
> }
>
> {
>   "tool": "email.send",
>   "params": {"to": "xxx", "body": "yyy"}
> ```

```
  }

  {
    "tool": "browser.navigate",
    "params": {"url": "..."}
  }
```

学一次 MCP，所有工具都一样用！

MCP 的核心组件



```
|                                                                    |
└────────────────────────────────────────────────────────────────────┘
```

1. **MCP 客户端**：嵌入在 AI 模型中，负责与 MCP 服务器通信
2. **MCP 服务器**：运行各种工具的独立进程
3. **MCP 协议**：标准化的通信格式

MCP 的优势

优势	说明
统一接口	所有工具调用方式一致
即插即用	新工具接入无需修改 AI 代码
生态丰富	一个标准，无限扩展
易于调试	标准化日志和错误处理
跨平台	不同 AI 平台可以复用相同工具

💻 代码示例：MCP 实践

MCP 工具的定义格式

```
{
  "name": "send_email",
  "description": "发送邮件到指定地址",
  "parameters": {
    "type": "object",
    "properties": {
      "to": {
        "type": "string",
```

```
        "description": "收件人邮箱地址"
      },
      "subject": {
        "type": "string",
        "description": "邮件主题"
      },
      "body": {
        "type": "string",
        "description": "邮件正文"
      }
    },
    "required": ["to", "subject", "body"]
  }
}
```

MCP 工具的调用示例

```
// AI 决定调用工具
{
  "tool": "send_email",
  "params": {
    "to": "boss@company.com",
    "subject": "周报",
    "body": "本周工作总结..."
  }
}

// 工具执行后返回
{
  "status": "success",
  "result": {
    "message_id": "msg_12345",
    "sent_at": "2024-03-15T10:30:00Z"
```

```
  }
}
```

OpenClaw 中的 MCP 支持

```
# OpenClaw 工具调用（类似 MCP 风格）

# 发送邮件
result = message(
    action="send",
    target="boss@company.com",
    content="周报已完成"
)

# 操作飞书文档
result = feishu_doc(
    action="write",
    doc_token="xxx",
    content="新内容"
)

# 浏览器操作
result = browser(
    action="open",
    targetUrl="https://example.com"
)
```

🏃 实践练习

练习 1：比较工具差异

列出你常用的 3 个工具，比较它们的： - 配置方式 - 调用语法 - 错误处理

思考：如果它们都遵循 MCP 标准，会有什么好处？

练习 2：设计 MCP 工具定义

为一个"天气查询"工具设计 MCP 格式的定义：

```
{
  "name": "get_weather",
  "description": "...",
  "parameters": {
    // 在这里填写
  }
}
```

练习 3：思考适用场景

哪些场景特别适合使用 MCP？ - 场景1：__ - **场景2：_ - 场景3：___**

📝 本章小结

核心要点

1. **工具多了会混乱** —— 每个工具配置、调用方式不同
2. **MCP 是标准接口** —— 类似 USB，统一规范
3. **一次学习，到处使用** —— 掌握 MCP 就能用所有工具
4. **生态效应** —— 标准促进工具丰富度

MCP vs 传统工具调用

特性	传统方式	MCP 方式
学习成本	每个工具都要单独学	学一次 MCP，通用
工具发现	需要文档说明	自动发现可用工具
参数格式	各工具不一致	统一 JSON Schema
错误处理	各工具不同	统一错误码
生态扩展	难	易

下一步预告

下一章，我们将解决一个问题：

> "虽然有标准接口了，但每次都要写提示词、组合工具、处理流程，能不能把常用的操作封装起来？"

这就是 **Skill（技能封装）** 要解决的问题。

✅ 自我检查清单

完成本章后，你应该能够：

- [ ] 解释 MCP 的概念和价值
- [ ] 类比 USB 说明 MCP 的作用
- [ ] 列举 MCP 的核心优势
- [ ] 理解工具标准化的重要性

💡 进阶思考

思考 1：MCP 和 API 标准（如 RESTful、GraphQL）有什么区别？

思考 2：如果所有 AI 平台都支持 MCP，对开发者意味着什么？

思考 3：MCP 会取代现有的工具集成方式吗？

第五章：重复劳动的噩梦 —— 封装你的工作流

🎬 故事场景：每周五的诅咒

时间：周五下午4点 **地点**：办公桌前 **心情**：疲惫 + 厌倦

场景 1：重复的周报

每到周五，你都要做同样的事：

```
第1步：打开数据后台
第2步：导出本周数据
第3步：打开Excel
第4步：整理数据
第5步：分析趋势
第6步：打开飞书
第7步：写周报文档
第8步：发送给老板

耗时：2小时
频率：每周一次
心情：🔥 烦躁
```

你问 AI：

```
你：帮我写周报
AI：好的，请提供本周数据...
```

> 你：（复制粘贴数据）
>
> AI：（基于数据生成周报）
>
> 你：（修改格式）
>
> 你：（发送邮件）
>
> AI：周报已完成！
>
> 你：下周又要重复这一过程...

场景 2：每次都要解释

更痛苦的是，每次都要重新解释：

> 第1周：
> 你：我是产品经理，需要周报包含数据、项目进展...
> AI：好的，我了解了
>
> 第2周：
> 你：帮我写周报
> AI：你是谁？需要什么格式的周报？
> 你：（再次解释一遍）
>
> 第3周：
> 你：周报
> AI：什么周报？什么格式？
> 你：（再次再次解释）
>
> ...
>
> 第52周：
> 你：（已经不想说话了）

你心想："为什么每次都要我重复同样的话？AI 就不能记住我的偏好吗？"

⚡ Aha Moment 5：重复的工作应该被封装

周五晚上11点，你终于写完今年的第52份周报

看着屏幕上 AI 的对话框，你突然顿悟：

> "每周都要做同样的事，说同样的话，这不是在浪费生命吗？
>
> **这跟我手动做周报有什么区别？**
>
> 应该把这套流程封装起来： - 我的身份（产品经理） - 我的偏好（简洁风格、数据驱动） - 数据来源（从哪查数据） - 输出格式（Markdown、邮件格式） - 发送对象（给谁发邮件）
>
> 封装好之后，我只需要说两个字：'周报'
>
> AI 就会自动： 1. 查询数据源 2. 分析数据 3. 生成内容 4. 发送邮件
>
> **一次配置，永久复用！"

这就是 **Skill（技能封装）** 的核心思想。

Skill封装示意图

图 5-1：Skill —— 把你常用的工作流封装成可复用的"技能"

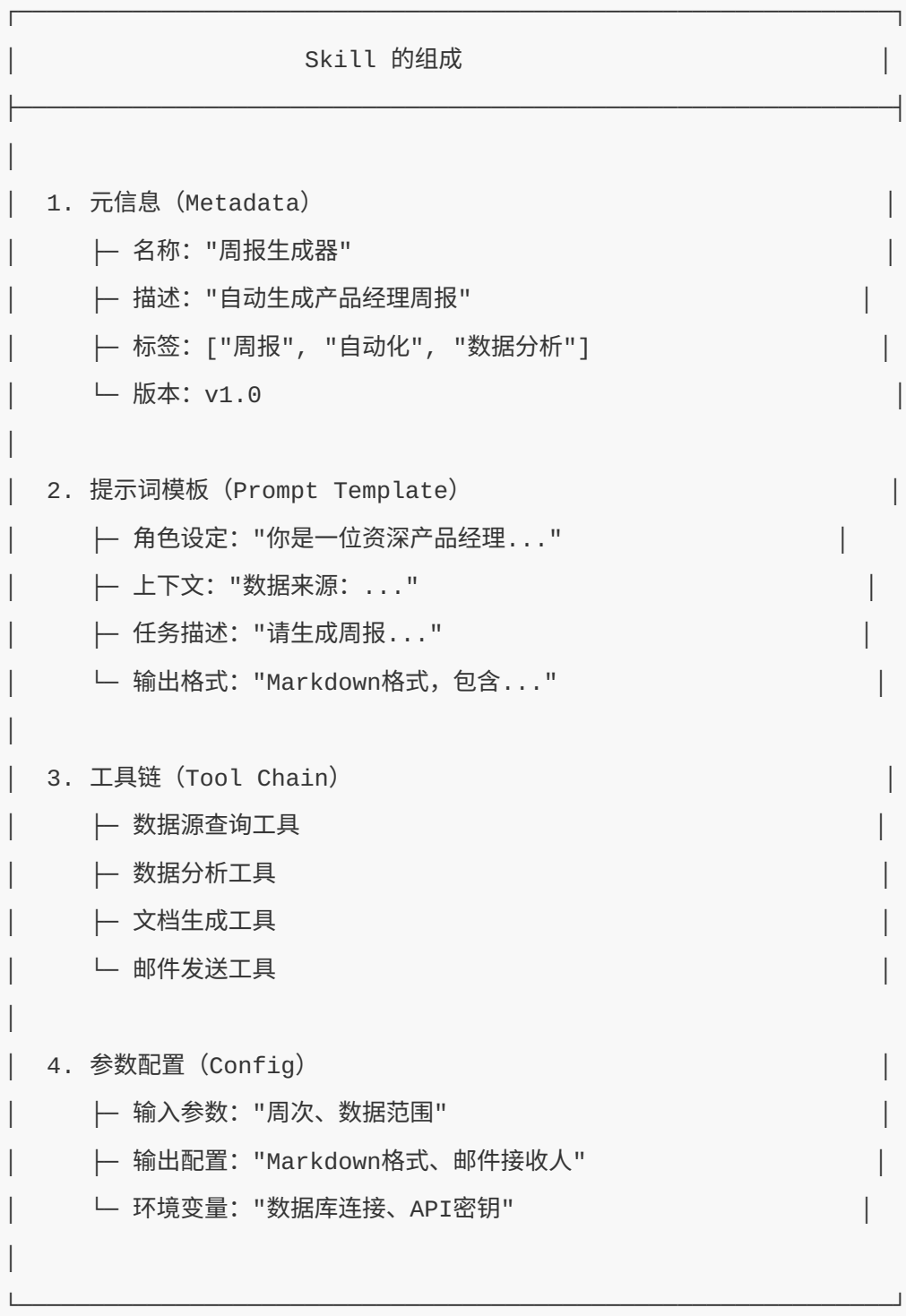
📚 概念引入：Skill（技能封装）

什么是 Skill？

Skill = 预定义的工作流 + 提示词模板 + 工具链配置

类比： - 📱 手机里的 App：一次安装，随时使用 - 🛠️AI 的 Skill：一次封装，随时调用

Skill 的组成


Skill 的组成
1. 元信息（Metadata）
├─ 名称："周报生成器"
├─ 描述："自动生成产品经理周报"
├─ 标签：["周报", "自动化", "数据分析"]
└─ 版本：v1.0
2. 提示词模板（Prompt Template）
├─ 角色设定："你是一位资深产品经理..."
├─ 上下文："数据来源：..."
├─ 任务描述："请生成周报..."
└─ 输出格式："Markdown格式，包含..."
3. 工具链（Tool Chain）
├─ 数据源查询工具
├─ 数据分析工具
├─ 文档生成工具
└─ 邮件发送工具
4. 参数配置（Config）
├─ 输入参数："周次、数据范围"
├─ 输出配置："Markdown格式、邮件接收人"
└─ 环境变量："数据库连接、API密钥"

Skill 的定义格式（OpenClaw风格）

```
# weekly_report.skill.md
name: 周报生成器
description: 自动查询数据、分析趋势、生成产品经理周报
version: 1.0.0
tags: [周报, 自动化, 数据分析]

# 提示词模板
prompt: |
  你是一位资深产品经理，擅长数据分析和报告撰写。

  基于以下数据生成周报：
  {{DATA}}

  格式要求：
  1. 本周数据亮点（用📈标记增长，📉标记下降）
  2. 项目进展（已完成/进行中/计划）
  3. 遇到的问题（如果有）
  4. 下周计划（可执行的3个任务）

  语言风格：简洁专业，突出数据

# 工具链
tools:
  - name: query_database
    description: 查询DAU、留存率等核心指标
  - name: generate_chart
    description: 生成趋势图
  - name: send_email
    description: 发送邮件给直属领导

# 输入参数
inputs:
```

```
  - name: week_number
    type: string
    description: 第几周
    default: "当前周"
  - name: data_range
    type: string
    description: 数据时间范围
    default: "本周一至周五"

# 输出配置
outputs:
  format: markdown
  actions:
    - save_to_file
    - send_email
```

💻 代码示例：创建和使用 Skill

示例 1：定义一个简单 Skill

```
# my_skill.skill.md
---
name: "翻译助手"
description: "将中文翻译成英文，保持专业术语准确"
version: "1.0"
---

## 角色设定

你是一位专业的技术文档翻译，擅长将中文技术文档翻译成英文。

## 翻译要求
```

```
1. 保留所有技术术语（如：API、数据库、函数名）
2. 使用简洁的英文表达
3. 保持段落结构
4. 输出双语对照格式

## 示例

输入：
"本函数用于查询用户数据。"

输出：
| 中文 | English |
|------|---------|
| 本函数用于查询用户数据。 | This function is used to query user data. |

## 使用方法

用户输入中文，自动输出双语对照翻译。
```

示例 2：在 OpenClaw 中使用 Skill

```
# 创建技能目录
mkdir -p ~/.openclaw/workspace/skills/weekly-report

# 创建 SKILL.md 文件
cat > ~/.openclaw/workspace/skills/weekly-report/SKILL.md << 'EOF'
---
name: 周报生成器
description: 自动生成产品经理周报
---

## 工作流程
```

````
1. 查询数据库获取本周数据
2. 分析数据趋势
3. 生成周报（Markdown格式）
4. 保存到文件
5. 发送邮件通知

## 代码示例

```python
# 查询数据
data = exec("python3 get_weekly_data.py")

# 生成周报
report = ai_generate(f"基于以下数据生成周报：{data}")

# 保存文件
write("/tmp/weekly_report.md", report)

# 发送邮件
send_qq_email("boss@company.com", "周报", report)
````

EOF
````

使用技能

openclaw skill weekly-report

````
### 示例 3：复杂的 Skill 工作流

```python
# seo_landing_page.skill.py
"""
SEO着陆页生成技能
自动生成针对特定关键词的SEO优化页面
"""

skill = {
    "name": "SEO着陆页生成器",
    "description": "基于产品信息生成SEO优化的着陆页",
    "steps": [
        {
            "name": "产品研究",
            "tool": "browser",
            "action": "scrape_yami_product",
            "output": "product_info"
        },
        {
            "name": "关键词分析",
            "tool": "serpapi",
            "action": "analyze_keywords",
            "input": "{{product_info.name}}",
            "output": "keywords"
        },
        {
````
````

```
            "name": "内容生成",
            "tool": "ai_generate",
            "prompt": "基于{{product_info}}和{{keywords}}生成SEO内容",
            "output": "seo_content"
        },
        {
            "name": "页面构建",
            "tool": "code",
            "action": "generate_html",
            "template": "landing_page.html",
            "data": "{{seo_content}}",
            "output": "html_page"
        },
        {
            "name": "预览部署",
            "tool": "webhook",
            "action": "deploy_preview",
            "file": "{{html_page}}",
            "output": "preview_url"
        },
        {
            "name": "发送确认",
            "tool": "feishu",
            "action": "send_card",
            "content": "SEO页面已生成：{{preview_url}}"
        }
    ]
}
```

🏃 实践练习

练习 1：设计你的第一个 Skill

思考你工作中重复做的事情，设计一个 Skill：

场景：___

Skill名称：___

输入参数： - 参数1：__ - 参数2：__

输出结果：___

工作流程： 1. __ 2. __ 3. ___

练习 2：Skill vs 脚本

比较 Skill 和传统脚本的差异：

特性	Shell脚本	Skill
学习成本	_____	_____
可复用性	_____	_____
灵活性	_____	_____
人机协作	_____	_____

练习 3：Skill 市场

如果你有 10 个 Skill，如何组织它们?

```
skills/
├── productivity/          # 效率类
│   ├── weekly-report/     # 周报生成
│   ├── meeting-notes/     # 会议记录
```

```
│   └── email-drafter/      # 邮件起草
├── content/                  # 内容类
│   ├── blog-generator/     # 博客生成
│   ├── social-posts/       # 社媒文案
│   └── video-script/       # 视频脚本
└── data/                     # 数据类
    ├── data-analysis/      # 数据分析
    ├── chart-generator/    # 图表生成
    └── report-builder/     # 报告构建
```

📝 本章小结

核心要点

1. **重复是效率杀手** —— 每周做同样的事，浪费大量时间
2. **Skill 是封装好的工作流** —— 一次配置，永久复用
3. **比脚本更智能** —— 包含 AI 推理，不只是机械执行
4. **可组合可扩展** —— 小 Skill 组合成大工作流

Skill 的应用场景

场景	传统方式	Skill方式	节省时间
周报生成	2小时/周	说两个字"周报"	95%
邮件回复	30分钟/封	输入要点自动生成	70%
数据处理	半天	自动调用工具链	80%
内容创作	4小时/篇	模板+AI生成	60%

Skill 与前几章的关系

```
Prompt（第一章） → 更准确的指令
    ↓
RAG（第二章） → 使用私有知识
    ↓
Tool Use（第三章） → 调用工具
    ↓
MCP（第四章） → 标准化接口
    ↓
Skill（本章） → 封装成可复用的工作流
```

下一步预告

下一章，我们将解决一个问题：

> "虽然 Skill 封装好了，但每次对话 AI 都会"忘记"之前的上下文，怎么办？"

这就是 **持久化记忆系统** 要解决的问题。

✅ 自我检查清单

完成本章后，你应该能够：

- [ ] 解释 Skill 的概念和价值
- [ ] 设计一个简单的 Skill 工作流
- [ ] 区分 Skill 和脚本的不同
- [ ] 列举 3 个可以用 Skill 自动化的场景

💡 进阶思考

思考 1：Skill 和普通函数/脚本有什么区别？

思考 2：Skill 可以互相调用吗？如何设计依赖关系？

思考 3：如何共享和交易 Skill？（技能市场）

第六章：失忆症的困扰 —— 让 AI 拥有持久记忆

🎬 故事场景：似曾相识的感觉

时间：周三下午 **地点**：与 AI 的对话界面 **心情**：困惑 + 无奈

场景 1：昨天的方案

周二，你和 AI 讨论了一个项目方案：

```
你：我们在做一个用户留存优化项目

AI：好的，了解了。请告诉我具体情况

你：目标是把次月留存率从 30% 提升到 35%
    目前尝试了 A/B 测试、推送提醒、新手引导
    下一步计划是...

AI：基于您的情况，我建议：
    1. 优化新用户首周体验
    2. 建立流失预警机制
    3. ...

你：这个方案很好，我先去和团队讨论
```

周三，你带着新想法回来：

```
你：关于昨天讨论的方案，我想补充一点

AI：您好！我是您的 AI 助手。
    请问您想讨论什么方案？

你：（？？？）就是那个留存优化方案啊！

AI：抱歉，我不记得我们有讨论过留存优化方案。
    能给我更多背景信息吗？

你：（崩溃）昨天明明讨论了两小时！
```

场景 2：长对话的诅咒

另一种情况：

```
第1轮对话（上午）：
你：分析这份数据
AI：好的...
[讨论 20 轮]

第21轮（中午）：
你：基于刚才的分析，生成报告
AI："刚才的分析"是指什么？
    请提供更多上下文

你：就是前面说的留存率下降的原因分析啊！

AI：抱歉，我只能记住最近的对话内容
    太早期的内容已经"遗忘"了

你：（看着上面20轮的记录）...
```

场景 3：跨项目失忆

项目 A：

> 你：（详细描述项目A的背景、目标、约束）
> AI：明白了，这是项目A的情况...
> [深入讨论]

切换到项目 B：

> 你：（讨论项目B）
> AI：了解了，这是项目B...

回到项目 A：

> 你：关于项目A那个问题...
> AI：项目A？能给我介绍一下项目A的背景吗？
>
> 你：（再次崩溃）昨天不是说了一上午吗！

⚡ Aha Moment 6：AI 需要一本笔记本

周三深夜，你终于意识到问题所在：

> "每次对话对 AI 来说都是全新的开始， 就像每次见医生都要重新描述病史一样荒谬。
>
> **AI 需要一本'笔记本'！**
>
> 这本笔记本应该： - 记录我是谁（身份、偏好） - 记录我们在做什么（项目背景） - 记录我们讨论过什么（历史对话要点） - 记录我们达成过什么共识（决策、方案）
>
> 每次对话前，AI 先读一遍这本笔记本 每次对话后，AI 更新这本笔记本
>
> **这样 AI 就再也不会'失忆'了！ "

这就是 **持久化记忆系统** 的核心思想。

📚 概念引入：持久化记忆系统

记忆的三个层次

1. **长期记忆**（几年~永久）：身份、价值观、技能
2. **中期记忆**（几周~几个月）：项目状态、决策记录
3. **短期记忆**（当前对话）：本次对话的上下文

OpenClaw 的记忆文件系统

- `MEMORY.md`：长期记忆
- `memory/YYYY-MM-DD.md`：每日笔记
- `projects/`：项目档案
- `.temp/`：临时文件

MEMORY.md 示例

```
# MEMORY.md

## 身份信息
- 职位：产品经理
- 负责领域：用户增长

## 当前项目
1. 留存优化（目标：留存率30%→35%）

## 个人偏好
- 周报格式：数据+结论+下一步
- 沟通风格：简洁直接
```

💻 代码示例

读取长期记忆

```
memory = read("~/.openclaw/workspace/MEMORY.md")
```

更新中期记忆

```
# 追加到今日笔记
append("~/.openclaw/workspace/memory/2024-03-15.md",
       "今天完成了A/B测试分析")
```

会话间记忆传递

```
# 保存会话摘要
write("session-summary-latest.md", summary)

# 下次读取
context = read("MEMORY.md") + read("session-summary-latest.md")
```

📝 本章小结

1. **AI 每次对话都是"失忆"的**
2. **持久化记忆 = AI 的笔记本**
3. **三层记忆架构**：长期、中期、短期
4. **主动维护**：记忆需要刻意记录

✅ 自我检查清单

- [ ] 解释为什么 AI 需要持久化记忆
- [ ] 创建 MEMORY.md
- [ ] 设计记忆更新流程

第七章：创作与沉淀 —— 从想法到成果

🎬 故事场景：想法无法落地

时间：深夜 **地点**：书桌前 **心情**：兴奋但无力

场景 1：好的开始，烂的结局

你有了一个绝妙的想法：

```
你：（兴奋地和AI讨论3小时）
"这个博客主题很棒！"
"这个结构设计得很巧妙！"
"这个观点很有洞见！"

AI：（提供了详细的建议和大纲）
"建议从这几个角度展开..."
"可以用这样的结构..."

你："太棒了！明天开始写！"

--- 第二天 ---

你："昨天我们讨论的那个博客..."
AI："什么博客？"

你：（抓狂）又要从头开始？！
```

场景 2：创作的孤岛

好不容易写完了文章：

> 你：（把AI生成的内容复制到Word）
> （调整格式）
> （保存到桌面）
>
> --- 一周后 ---
>
> 你："那篇文章放哪了？"
> （翻遍桌面找不到）
> "算了，重写吧..."

场景 3：最后一公里

AI 生成了很好的内容，但无法发布：

> AI：以下是为您生成的文章...
> [完美的内容]
>
> 你："怎么发布到我的博客上？"
> AI："抱歉，我无法直接操作您的博客..."
>
> 你：（手动复制粘贴）
> （调整格式）
> （上传图片）
> （发布）
>
> 耗时：1小时

⚡ Aha Moment 7：创作需要闭环

看着桌面上散落的文档，你突然顿悟：

> "AI 能帮我生成内容，但创作的'最后一公里'——保存、整理、发布、分享——都要我手动完成。
>
> 003e **这不就跟厨师做好菜，但得自己端上桌、洗碗一样吗？** 003e 003e 应该把整个流程串起来： 003e 1. AI 生成内容 003e 2. 自动保存到正确位置 003e 3. 自动格式化 003e 4. 自动发布/分享 003e 003e **从想法到成果，一气呵成！"

这就是 **工具链与工作流** 的核心思想。

创作闭环示意图

图 7-1：创作闭环 —— 从想法到成果的完整流程

📚 概念引入：工具链与工作流

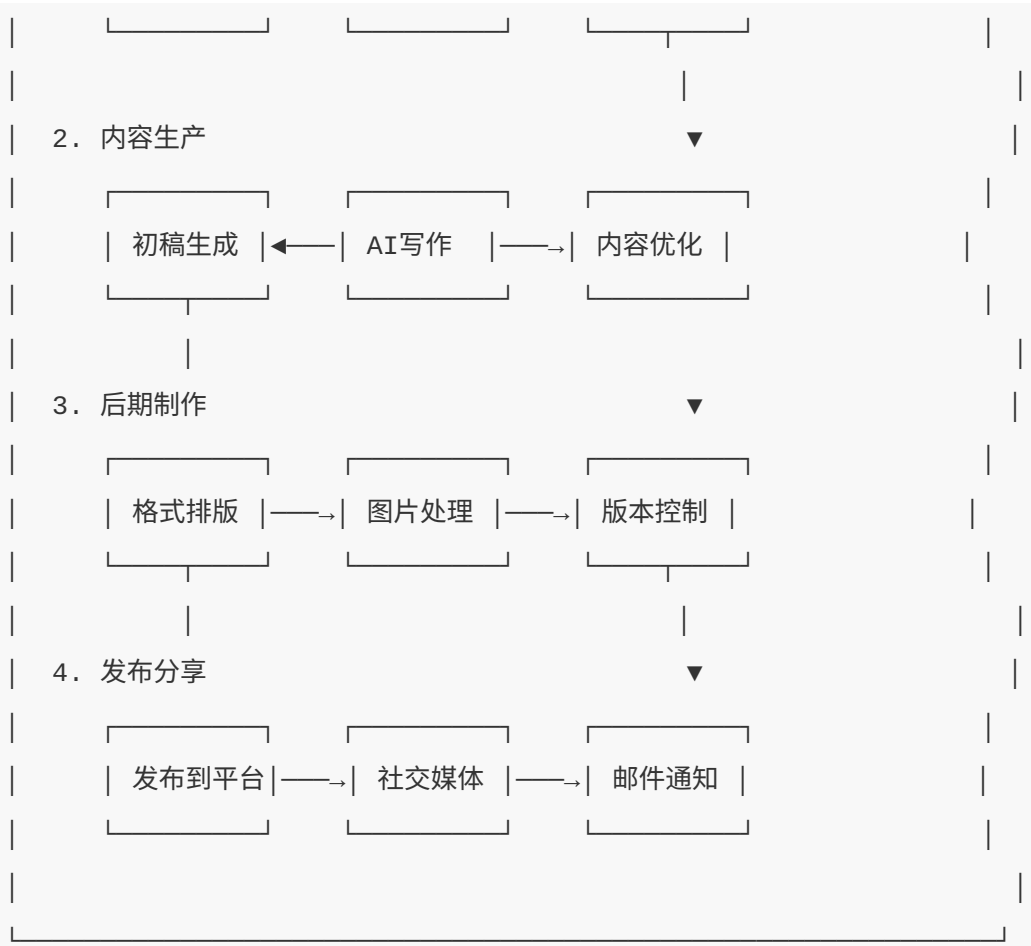
什么是工具链？

工具链 = 多个工具的组合，形成完整工作流

类比： - 🏭 工厂流水线：原料 → 加工 → 组装 → 质检 → 包装 → 出货 - 📝 创作工具链：想法 → 生成 → 编辑 → 格式化 → 发布 → 分享

创作工作流的典型环节

```
┌──────────────────────────────────────────────────────────┐
│                      创作工作流                          │
├──────────────────────────────────────────────────────────┤
│                                                          │
│  1. 创意激发                                             │
│     ┌──────────┐    ┌──────────┐    ┌──────────┐         │
│     │ 头脑风暴 │──→ │ 主题研究 │──→ │ 大纲设计 │         │
```



工具链的组合模式

模式	说明	示例
串行	步骤依次执行	写→审→发
并行	多个步骤同时进行	同时生成文案+图片
条件	根据结果决定下一步	审核通过才发布
循环	反复优化直到满意	修改→预览→修改

💻 代码示例：创作工具链实践

示例 1：博客创作工具链

```
# blog_creation_workflow.py

workflow "create_blog_post":

    # 步骤1：研究主题
    research = web_search("{topic} 最新趋势 2024")

    # 步骤2：生成大纲
    outline = ai_generate(f"""
       基于以下研究资料，生成博客大纲：
         {research}

       要求：
         1. 引人入胜的开头
         2. 3-5个核心观点
         3. 有力的结尾
    """)

    # 步骤3：生成正文
    content = ai_generate(f"""
       基于大纲生成完整文章：
         {outline}

       风格：专业但易懂
       字数：2000字左右
    """)

    # 步骤4：保存到文件
    file_path = f"blog/{date}_{slug(topic)}.md"
    write(file_path, content)
```

```
    # 步骤5：生成配图
    image_prompt = ai_extract("从文章中提取配图描述", content)
    image = generate_image(image_prompt)
    save(image, f"blog/images/{slug(topic)}.png")

    # 步骤6：提交到Git
    exec(f"""
        cd blog && \
        git add . && \
        git commit -m "Add: {topic}" && \
        git push
    """)

    # 步骤7：部署（Cloudflare Pages自动部署）
    url = f"https://yourblog.com/posts/{slug(topic)}"

    # 步骤8：发送通知
    message(action="send", content=f"新文章已发布：{url}")

    return f"✅ 文章已发布：{url}"
```

示例 2：SEO着陆页创作链

```
# seo_page_workflow.py

workflow "create_seo_landing_page":

    inputs:
        keyword: "目标关键词"
        product_url: "产品页面URL"

    # 步骤1：抓取产品信息
    product_info = browser_scrape(product_url)
```

```
# 步骤2：关键词研究
keyword_data = serpapi_research(keyword)

# 步骤3：生成SEO内容
seo_content = ai_generate(f"""
   关键词：{keyword}
   搜索意图：{keyword_data.intent}
   产品信息：{product_info}

   生成SEO优化的着陆页：
    1. Title（60字符以内）
    2. Meta Description（160字符以内）
    3. H1标题
    4. 正文内容（1500字）
    5. CTA按钮文案
""")

# 步骤4：生成HTML页面
html = render_template("landing_page.html", {
    "content": seo_content,
    "product": product_info,
    "styles": load_css("apple_style.css")
})

# 步骤5：保存并部署
page_path = f"landing-pages/{slug(keyword)}.html"
write(page_path, html)

# 步骤6：预览链接
preview_url = deploy_preview(page_path)

# 步骤7：发送确认卡片
feishu_send_card({
```

```
        "title": "SEO页面待确认",
        "content": f"关键词: {keyword}",
        "preview": preview_url,
        "buttons": ["确认发布", "需要修改"]
    })

    return preview_url
```

示例 3：文档协作链

```
# doc_collaboration_workflow.py

workflow "create_weekly_report":

    # 步骤1: 从多个数据源收集数据
    data = {
        "database": query_sql("SELECT * FROM weekly_metrics"),
        "projects": feishu_doc_read("project_status"),
        "issues": jira_query("status in (Done) AND updated >= -7d")
    }

    # 步骤2: AI分析并生成报告
    report = ai_generate(f"""
       基于以下数据生成年报:
        {json.dumps(data, indent=2)}

       格式: Markdown
       结构: 数据亮点、项目进展、问题汇总、下周计划
    """)

    # 步骤3: 保存到飞书
    feishu_doc_write(
        folder="周报/2024",
        title=f"周报_{current_week()}",
```

```
        content=report
    )

    # 步骤4：发送邮件
    send_email(
        to="team@company.com",
        subject=f"周报 - {current_week()}",
        body=report,
        attachments=["weekly_data.xlsx"]
    )

    # 步骤5：创建日程提醒
    calendar_create({
        "title": "下周计划复盘",
        "time": "next_friday_14:00",
        "attendees": ["team@company.com"]
    })

    return "✅ 周报已生成并分发"
```

🏃 实践练习

练习 1：设计你的创作工具链

选择一个你经常做的创作任务，设计完整工具链：

任务：___

工具链步骤：1. __ （工具：_） 2. __ （工具：_） 3. __ （工具：_） 4. __ （工具：_） 5. ___ （工具：_）

练习 2：工具链优化

分析当前工具链的瓶颈：

步骤	当前耗时	优化方案	预期节省

练习 3：版本控制

为你的创作内容建立版本控制流程： - 初稿 → 修改 → 审核 → 发布 - 如何标记版本？ - 如何回滚？

📝 本章小结

核心要点

1. **创作不只有"写"** —— 还有保存、编辑、发布、分享
2. **工具链形成闭环** —— 从想法到成果的完整流程
3. **自动化最后一公里** —— 让AI帮你完成发布等操作
4. **版本控制很重要** —— 创作内容需要管理和追溯

工具链的价值

没有工具链	有工具链
想法容易丢失	想法沉淀为成果
重复劳动多	自动化程度高
发布耗时长	一键发布
难以协作	流程标准化

下一步预告

下一章，我们将整合前面所有内容：

> "如何把这些技术组合起来，处理真正复杂的任务？"

这就是 **Agent 与工作流编排**。

✅ 自我检查清单

- [ ] 理解创作闭环的概念
- [ ] 能设计一个简单的创作工具链
- [ ] 知道如何选择合适的工具
- [ ] 理解版本控制的重要性

第八章：复杂任务的编排 —— Agent 与工作流

🎬 故事场景：端到端的挑战

时间：Q4规划会议前 **地点**：会议室 **任务**：制定完整的Q4增长策略

场景：一个复杂任务的组成

老板要求你制定 Q4 增长策略，需要：

```
任务分解：
├─ 1. 数据分析
│    ├─ 拉取Q1-Q3的历史数据
│    ├─ 分析增长趋势
│    └─ 识别增长瓶颈
│
├─ 2. 竞品调研
│    ├─ 收集竞品Q4动向
│    ├─ 分析竞品策略
│    └─ 找出差异化机会
│
├─ 3. 策略制定
│    ├─ 设定增长目标
│    ├─ 设计增长策略
│    ├─ 制定执行计划
│    └─ 评估资源需求
│
├─ 4. 方案产出
```

```
|    ├─ 撰写策略文档
|    ├─ 制作PPT
|    ├─ 准备数据看板
|    └─ 安排汇报会议
|
└─ 5. 汇报沟通
     ├─ 发送预览给领导
     ├─ 收集反馈意见
     └─ 调整最终方案

预计耗时：2周
涉及工具：10+
涉及人员：5+
```

传统做法：- 手动收集数据（3天）- 人工分析竞品（2天）- 反复开会讨论（N天）- 熬夜写PPT（2天）

你心想："这么复杂的任务，AI能帮到什么程度？"

⚡ Aha Moment 8：组合的力量

看着满屏的数据和未完成的PPT，你突然顿悟：

> "这不仅是'查数据'或'写文档'的单一任务， 003e 003e 而是一个**多步骤、多工具、多人协作**的复杂流程。 003e 003e 但如果我把前几章学的都组合起来： 003e - RAG（查历史数据） 003e - Tool Use（调用数据API、浏览器） 003e - MCP（标准化工具调用） 003e - Skill（封装常用分析套路） 003e - Memory（记住之前的分析结论） 003e 003e **让 AI 像一个'项目经理'一样，** 003e **自动协调各个步骤、调用各种工具、整合所有信息** 003e 003e 这就是 **Agent（智能体）** —— 能自主规划和执行复杂任务的 AI！"

📚 概念引入：Agent 与工作流编排

什么是 Agent?

Agent = AI + 自主规划 + 工具调用 + 记忆 + 反馈循环

传统 AI：你问一句，它答一句

```
你：分析Q3数据
AI：（给你分析方法，你自己去做）
```

Agent：理解目标，自主规划，执行到底

```
你：制定Q4增长策略
Agent：
  1. 分析任务：需要数据+竞品+策略+产出
  2. 制定计划：4个阶段，12个步骤
  3. 执行步骤1：调用数据API查询Q3数据
  4. 执行步骤2：使用浏览器抓取竞品信息
  5. 执行步骤3：分析并生成策略建议
  6. ...
  7. 生成最终报告
  8. 发送给你确认
```

Agent 的核心能力

```
┌─────────────────────────────────────────────────────────┐
│                    Agent 架构                           │
├─────────────────────────────────────────────────────────┤
│                                                         │
│  ┌─────────────────────────────────────────────┐        │
│  │              规划（Planning）               │        │
│  │  • 理解目标   • 分解任务   • 制定计划       │        │
```

工作流编排模式

模式	说明	示例
顺序执行	按顺序完成各步骤	数据分析 → 报告撰写 → 邮件发送
分支判断	根据条件走不同分支	数据质量差 → 重新采集；质量好 → 继续分析
并行处理	同时执行多个任务	同时抓取多个竞品信息
循环优化	反复执行直到满意	生成 → 检查 → 修改 → 检查...
人机协作	关键节点人工确认	AI生成 → 人工审核 → AI发布

复杂任务的分解示例

```
任务："生成SEO着陆页并发布"

分解：
├─ 阶段1：研究
│    ├─ 步骤1.1：分析关键词
│    ├─ 步骤1.2：研究竞品页面
│    └─ 步骤1.3：确定页面角度
│
├─ 阶段2：创作
│    ├─ 步骤2.1：生成页面结构
│    ├─ 步骤2.2：撰写SEO内容
│    ├─ 步骤2.3：生成配图
│    └─ 步骤2.4：构建HTML页面
│
├─ 阶段3：审核
│    ├─ 步骤3.1：部署预览版本
│    ├─ 步骤3.2：发送确认请求
│    └─ 步骤3.3：等待人工确认/修改
│
└─ 阶段4：发布
     ├─ 步骤4.1：正式部署
     ├─ 步骤4.2：更新索引
     └─ 步骤4.3：发送完成通知
```

💻 代码示例：Agent 工作流实践

示例 1：SEO页面生成 Agent

```
# seo_agent.py

class SEOAgent:
    """SEO着陆页生成 Agent"""

    def __init__(self):
        self.memory = {}
        self.tools = {
            'browser': BrowserTool(),
            'search': SearchTool(),
            'writer': WriterTool(),
            'deploy': DeployTool(),
            'notify': NotifyTool()
        }

    def run(self, keyword, product_url):
        """执行完整工作流"""

        # 阶段1：研究
        self.log("阶段1：市场研究")
        research = self.research_phase(keyword, product_url)

        # 阶段2：创作
        self.log("阶段2：内容创作")
        content = self.creation_phase(research)

        # 阶段3：审核（人机协作）
        self.log("阶段3：人工审核")
        approved = self.review_phase(content)
```

```
        if not approved:
            self.log("需要修改，重新创作")
            content = self.modify_phase(content)
            approved = self.review_phase(content)

        # 阶段4：发布
        if approved:
            self.log("阶段4：发布部署")
            result = self.publish_phase(content)
            return result
        else:
            return "任务取消"

    def research_phase(self, keyword, product_url):
        """研究阶段"""
        # 并行执行多个研究任务
        tasks = [
            self.tools['search'].analyze_keyword(keyword),
            self.tools['browser'].scrape_product(product_url),
            self.tools['search'].find_competitors(keyword)
        ]
        results = asyncio.gather(*tasks)

        # 保存到记忆
        self.memory['research'] = results
        return results

    def creation_phase(self, research):
        """创作阶段"""
        # 基于研究生成内容
        content = self.tools['writer'].generate_landing_page(
            keyword=research['keyword'],
            product=research['product'],
            competitors=research['competitors']
```

```
        )

        # 生成配图
        content['image'] = self.tools['writer'].generate_image(
            content['image_prompt']
        )

        # 构建HTML
        content['html'] = self.build_html(content)

        self.memory['content'] = content
        return content

    def review_phase(self, content):
        """审核阶段 - 人机协作"""
        # 部署预览
        preview_url = self.tools['deploy'].preview(content['html'])

        # 发送确认请求
        self.tools['notify'].send_review_card({
            'preview': preview_url,
            'keyword': content['keyword'],
            'options': ['确认发布', '需要修改', '取消']
        })

        # 等待人工确认（异步）
        response = self.wait_for_human_response(timeout=3600)

        return response == '确认发布'

    def publish_phase(self, content):
        """发布阶段"""
        # 正式部署
        live_url = self.tools['deploy'].production(content['html'])
```

```
        # 提交搜索引擎
        self.tools['search'].submit_to_google(live_url)

        # 发送完成通知
        self.tools['notify'].send_message(
            f"✅ SEO页面已发布：{live_url}"
        )

        return live_url
```

示例 2：工作流编排配置

```
# workflow_config.yaml

name: Q4增长策略制定
version: 1.0

stages:
  - name: 数据分析
    parallel: true
    steps:
      - tool: sql_query
        name: 拉取历史数据
        query: "SELECT * FROM growth_metrics WHERE date >= '2024-01-01'"

      - tool: data_analysis
        name: 趋势分析
        analysis_type: time_series

      - tool: ai_analysis
        name: 瓶颈识别
        prompt: "分析数据，找出增长瓶颈"
```

```
    output: analysis_report

  - name: 竞品调研
    parallel: true
    steps:
      - tool: browser_scrape
        name: 竞品信息收集
        targets: ["竞品A", "竞品B", "竞品C"]

      - tool: ai_analysis
        name: 竞品策略分析

    output: competitor_report

  - name: 策略制定
    depends_on: [数据分析, 竞品调研]
    steps:
      - tool: ai_generate
        name: 生成策略方案
        inputs:
          - {{analysis_report}}
          - {{competitor_report}}
        prompt: "基于分析和竞品研究，制定Q4增长策略"

    output: strategy_doc

  - name: 产出制作
    depends_on: 策略制定
    parallel: true
    steps:
      - tool: doc_generator
        name: 生成策略文档
        template: strategy_template.docx
```

```
      - tool: ppt_generator
        name: 制作PPT
        template: strategy_ppt.pptx

      - tool: dashboard_builder
        name: 制作数据看板

    output: deliverables

  - name: 审核发布
    depends_on: 产出制作
    steps:
      - tool: feishu_send
        name: 发送预览
        type: review_card

      - tool: human_approval
        name: 等待确认
        timeout: 86400

      - tool: conditional
        name: 条件分支
        if: approved
        then:
          - tool: email_send
            name: 正式发布
        else:
          - tool: revision_loop
            name: 修改循环

notifications:
  on_start: true
  on_complete: true
  on_error: true
```

🏃 实践练习

练习 1：分解复杂任务

选择一个你工作中的复杂任务，用 Agent 思维分解：

任务：___

分解：

```
阶段1：_________
  ├─ 步骤1.1：_________
  └─ 步骤1.2：_________

阶段2：_________
  ├─ 步骤2.1：_________
  └─ 步骤2.2：_________

阶段3：_________
  └─ ...
```

练习 2：设计人机协作点

在上述任务中，哪些节点需要人工确认？

节点	为什么需要人工	确认内容

练习 3：异常处理

如果某个步骤失败了，Agent 应该如何处理？

```
步骤X失败 →
  ├─ 重试3次 → 成功 → 继续
  └─ 重试失败 →
      ├─ 跳过该步骤 → 继续
      └─ 终止任务 → 通知人工
```

📝 本章小结

核心要点

1. **复杂任务需要分解** —— 把大任务拆成小步骤
2. **Agent 能自主执行** —— AI 不只是回答，还能规划和执行
3. **工作流编排是关键** —— 设计合理的执行顺序和依赖关系
4. **人机协作很重要** —— 关键节点需要人工确认

Agent vs Skill

特性	Skill	Agent
复杂度	简单、固定流程	复杂、动态规划
决策能力	按预设执行	能自主决策和调整
适用范围	重复性任务	复杂、多变的任务
关系	Skill 是 Agent 的组件	Agent 编排多个 Skill

前几章的整合

```
Prompt（提示词）
    ↓
RAG（知识增强）
```

```
    ↓
Tool Use（工具调用）
    ↓
MCP（标准化）
    ↓
Skill（封装）
    ↓
Memory（持久化）
    ↓
工作链（创作闭环）
    ↓
Agent（整合编排） ← 本章
```

下一步预告

下一章，我们将学习：

> "这么多技术，什么时候用哪个？如何组合？"

这就是 **技术选型与最佳实践**。

✅ 自我检查清单

- [ ] 理解 Agent 的概念和价值
- [ ] 能分解一个复杂任务
- [ ] 理解工作流编排的几种模式
- [ ] 能设计人机协作的节点

第九章：进化之路 —— 技术选型与最佳实践

🎯 本章目标

学完前面八章，你已经掌握了： - Prompt 工程 - RAG 知识增强 - Tool Use 工具调用 - MCP 标准化 - Skill 封装 - Memory 持久化 - 工具链创作 - Agent 编排

但关键问题：什么时候用哪个？

本章给你一张"决策地图"，让你能够： 1. 快速判断该用什么技术 2. 合理组合各种技术 3. 避免常见陷阱

一、技术选型决策树

```
遇到问题？
    │
    ├─ 需要 AI 回答问题？
    │    ├─ 是通用知识？ → 基础 Prompt 工程
    │    └─ 是私有知识？ → RAG / Memory
    │
    ├─ 需要 AI 执行操作？
    │    ├─ 单次简单操作？ → Tool Use
    │    ├─ 多次重复操作？ → Skill
    │    └─ 工具管理混乱？ → MCP
    │
    ├─ 需要创作内容？
    │    ├─ 单次创作？ → 提示词 + 工具链
    │    └─ 重复创作？ → Skill + 自动化
```

```
    │
    ├─ 任务复杂多步骤?
    │    ├─ 流程固定?  → Skill
    │    └─ 需要动态规划?  → Agent
    │
    └─ 需要跨会话记忆?
         ├─ 短期信息?  → Daily Notes
         └─ 长期信息?  → MEMORY.md
```

二、技术选型速查表

2.1 按场景选择

场景	推荐技术	不推荐	原因
简单问答	Prompt	RAG	杀鸡用牛刀
查公司内部政策	RAG + Memory	纯Prompt	需要私有知识
发一封邮件	Tool Use	Agent	太简单
每周发周报	Skill	手动	重复任务
管理10个工具	MCP	各自配置	标准化管理
写一篇博客	工具链	纯AI生成	需要发布闭环
制定季度规划	Agent	Skill	太复杂
记住用户偏好	Memory	每轮重复	持久化需求

2.2 按复杂度选择

> 复杂度递增：
>
> ```
> Level 1: 基础对话
>     └── Prompt 工程
>
> Level 2: 知识增强
>     └── Prompt + RAG/Memory
>
> Level 3: 单次操作
>     └── Prompt + Tool Use
>
> Level 4: 重复任务
>     └── Skill (Prompt + Tool + Config)
>
> Level 5: 多工具协作
>     └── Skill + MCP
>
> Level 6: 复杂创作
>     └── 工具链 (多 Skill 串联)
>
> Level 7: 智能决策
>     └── Agent (自主规划 + 执行)
> ```

2.3 组合推荐

常见需求	推荐组合	效果
智能客服	RAG + Tool + Memory	准确回答，能执行操作
自动化周报	Skill + Tool + Memory	一键生成，记住偏好
SEO内容工厂	Agent + Skill + MCP + 工具链	端到端自动化

常见需求	推荐组合	效果
个人知识助手	RAG + Memory + Tool	问答+操作+记忆
复杂数据分析	Agent + Tool + Skill	动态规划分析流程

三、最佳实践指南

3.1 Prompt 工程最佳实践

✅ **做：** - 给足够的上下文 - 明确输出格式 - 使用角色设定 - 提供示例（Few-shot）

❌ **不做：** - 期望 AI 读心术 - 不给任何背景 - 要求模糊 - 一次问太多

示例对比：

> ❌ 差：
> "写个周报"
>
> ✅ 好：
> "你是一位资深产品经理，擅长数据驱动的工作汇报。
>
> 背景：
> - 我是用户增长团队的产品经理
> - 本周完成了A/B测试和新功能上线
> - 数据：DAU增长5%，留存率提升2%
>
> 请生成周报，要求：
> 1. 结构：数据亮点+项目进展+下周计划
> 2. 风格：简洁专业，突出数据
> 3. 格式：Markdown，带emoji标注趋势"

3.2 RAG 最佳实践

✅ **做：** - 定期更新知识库 - 分块存储（Chunks） - 版本控制重要文档 - 结合搜索工具获取最新信息

❌ **不做：** - 把知识库当垃圾桶（什么都存） - 从不更新过期信息 - 敏感信息明文存储

3.3 Tool Use 最佳实践

✅ **做：** - 幂等设计（重复执行无副作用） - 错误处理和重试机制 - 操作前确认（危险操作） - 记录操作日志

❌ **不做：** - 直接删除数据不确认 - 无限制调用外部 API - 敏感操作不留痕迹

安全提示词模板：

> 在执行以下操作前，必须获得用户确认：
> - 发送邮件给外部人员
> - 删除任何数据
> - 修改配置文件
> - 花费超过 X 元

3.4 Skill 最佳实践

✅ **做：** - 单一职责（一个Skill做一件事） - 参数化配置 - 完善的错误处理 - 文档和示例

❌ **不做：** - 一个Skill做所有事 - 硬编码个人偏好 - 无版本管理

Skill 设计检查清单：

- [ ] 名称清晰（动词+名词）
- [ ] 输入参数明确
- [ ] 输出结果可预测
- [ ] 错误处理完善
- [ ] 有使用示例
- [ ] 文档完整

3.5 Memory 最佳实践

✅ **做：** - 区分长期/短期记忆 - 定期回顾和整理 - 敏感信息加密 - 结构化存储

❌ **不做：** - 把临时查询当记忆 - 存储密码等敏感信息 - 从不清理过期信息

记忆整理时间表：

频率	动作
每天	写 Daily Notes
每周	回顾本周，提取重要信息
每月	更新 MEMORY.md
每季度	大扫除，清理过期信息

四、常见误区与解决方案

误区 1：过度工程化

症状： - 为了用技术而用技术 - 简单任务复杂化

案例：

> ❌ 错误：
> "我要查个天气，先搭建一个RAG系统..."
>
> ✅ 正确：
> "直接调用天气API"

解决方案： - 从简单开始 - 只有遇到痛点才引入新技术 - KISS 原则（Keep It Simple, Stupid）

误区 2：期望过高

症状： - 期望 AI 100% 准确 - 不允许任何错误

现实： - AI 会犯错（特别是复杂推理） - 需要人工审核关键输出

解决方案： - 设计人机协作流程 - 关键节点人工确认 - 建立反馈改进机制

误区 3：忽视维护

症状： - 建好系统就不管了 - 知识库从不更新

后果： - 信息过时 - 工具失效 - 系统腐烂

解决方案： - 定期维护计划 - 监控和告警 - 版本管理

误区 4：数据安全隐患

风险： - 敏感数据泄露 - API 密钥暴露 - 越权操作

解决方案： - 最小权限原则 - 敏感信息加密 - 操作审计日志

五、进阶路径

5.1 初学者路径（1-4周）

目标：掌握基础，能完成简单任务

```
Week 1: Prompt 工程
    - 学习提示词技巧
    - 练习写清晰指令

Week 2: Tool Use
    - 了解可用工具
    - 练习调用工具
```

```
Week 3: Memory
    - 建立 MEMORY.md
    - 养成记录习惯

Week 4: Skill 入门
    - 封装第一个 Skill
    - 应用到实际工作
```

5.2 进阶者路径（1-3个月）

目标：能设计复杂工作流

```
Month 1: RAG + MCP
    - 搭建知识库
    - 标准化工具管理

Month 2: 工具链 + Skill
    - 设计创作工具链
    - 开发多个 Skill

Month 3: Agent 编排
    - 学习工作流设计
    - 实现复杂自动化
```

5.3 专家路径（3-6个月）

目标：构建完整的 AI 工作系统

```
阶段1：系统架构
    - 设计整体架构
    - 选择合适的技术栈

阶段2：生态建设
    - 开发 Skill 库
```

- 建立最佳实践

阶段3：团队协作
- 推广到团队
- 建立协作流程

阶段4：持续优化
- 收集反馈
- 迭代改进

六、资源推荐

学习资源

资源	类型	适合人群
Prompt Engineering Guide	文档	所有人
LangChain 文档	文档	开发者
OpenClaw 官方文档	文档	OpenClaw 用户
AI 使用案例集	案例	实践者

社区

- OpenClaw Discord
- AI 工作流交流群
- GitHub 开源项目

✅ 本章检查清单

完成本章后，你应该能够：

- [ ] 根据场景选择合适的技术
- [ ] 避免常见的技术选型错误
- [ ] 设计合理的技术组合
- [ ] 制定自己的学习路径

🚀 下一步

恭喜你完成了理论学习！

下一章是 **实战案例集**，我们将通过 4 个完整的项目，把前面学的所有知识融会贯通。

准备好动手实践了吗？

第十章：实战案例集 —— 从理论到实践

概述

本章通过 4 个完整案例，展示如何综合运用前面学到的所有技术。

每个案例包含： - 背景与痛点 - 解决方案设计 - 技术选型 - 实现步骤 - 效果评估 - 可复制模板

案例 1：个人知识助手

背景

小明是一位产品经理，每天阅读大量文章、参加会议、产生灵感。但： - 收藏的文章从不回看 - 会议笔记分散各处 - 灵感想法容易遗忘 - 需要时找不到之前的内容

解决方案

构建个人知识助手系统：

```
┌─────────────────────────────────────────────────────────────┐
│                      个人知识助手系统                        │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│  输入层                                                     │
│  ├─ 微信/飞书收藏文章 → 自动保存到知识库                    │
│  ├─ 会议语音 → 转文字 → 提取要点                            │
│  ├─ 灵感想法 → 快速记录到 Daily Notes                       │
│  └─ 网页阅读 → 一键保存重点                                 │
│                                                             │
│  处理层                                                     │
│  ├─ RAG 系统：文章向量化存储                                │
```

```
│   ├─ AI 摘要：自动生成文章摘要                                    │
│   ├─ 标签分类：自动打标签                                        │
│   └─ 关联推荐：相关内容自动关联                                  │
│                                                                  │
│   输出层                                                         │
│   ├─ 问答："我之前看过的关于用户增长的文章"                      │
│   ├─ 周报生成：自动汇总本周阅读和思考                            │
│   ├─ 灵感激发：基于已有内容推荐新思路                            │
│   └─ 知识图谱：可视化知识关联                                    │
│                                                                  │
└──────────────────────────────────────────────────────────────────┘
```

技术选型

组件	技术	说明
知识存储	MEMORY.md + Daily Notes	文件化知识库
文章处理	Tool Use (web_fetch)	抓取网页内容
语音转写	Whisper	本地语音识别
检索增强	RAG (文件检索)	基于文件名/标签检索
摘要生成	AI (Kimi)	自动生成摘要
交互界面	Feishu/微信	移动端随时访问

实现步骤

Step 1: 建立知识库结构

```
knowledge_base/
├── articles/           # 收藏的文章
│   ├── 2024/
```

```
│   │   ├── 03/
│   │   │   ├── article_001_用户增长策略.md
│   │   │   └── article_002_AI趋势分析.md
│   │   └── 04/
│   └── by_tag/         # 按标签分类
│       ├── user_growth/
│       └── ai_tech/
├── meetings/           # 会议记录
│   ├── 2024-03-15_产品周会.md
│   └── 2024-03-18_需求评审.md
├── ideas/              # 灵感想法
│   ├── 2024-03-15_新功能设想.md
│   └── 2024-03-20_优化思路.md
└── MEMORY.md           # 核心知识索引
```

Step 2: 创建收集工具链

```
# collect_article.skill.md
name: 文章收集器
description: 收藏文章并自动生成摘要

workflow:
  1. 获取文章 URL
  2. 抓取文章内容 (web_fetch)
  3. AI 生成摘要
  4. AI 提取标签
  5. 保存到 articles/{year}/{month}/
  6. 更新 MEMORY.md 索引
  7. 发送确认消息
```

Step 3: 创建查询 Skill

```
# query_knowledge.skill.md
name: 知识查询
```

```
trigger: "搜索 {关键词}"

workflow:
  1. 在 articles/ 中搜索标题和内容
  2. 在 meetings/ 中搜索相关会议
  3. 在 ideas/ 中搜索相关想法
  4. AI 整合结果
  5. 按相关度排序返回
```

效果

指标	使用前	使用后
文章回看率	5%	60%
找资料时间	30分钟	2分钟
灵感记录率	20%	90%
周报准备时间	2小时	10分钟

案例 2：自动化周报系统

背景

产品经理每周要写周报，需要： - 从多个系统收集数据 - 分析数据趋势 - 撰写报告内容 - 发送给老板

耗时：2小时/周，且容易遗漏

解决方案

一键生成周报系统：

```
用户输入："周报"
    ↓
Agent 启动
    ↓
├─ 步骤1：数据收集（并行）
│    ├─ 查询数据库获取核心指标
│    ├─ 读取项目文档获取进展
│    └─ 查询 JIRA 获取任务状态
│
├─ 步骤2：AI 分析
│    ├─ 对比上周数据，识别变化
│    ├─ 分析趋势，发现问题
│    └─ 生成洞察和建议
│
├─ 步骤3：内容生成
│    ├─ 生成周报 Markdown
│    ├─ 生成数据图表
│    └─ 生成邮件正文
│
├─ 步骤4：发送
│    ├─ 保存到飞书文档
│    ├─ 发送邮件给老板
│    └─ 发送确认消息给用户
│
└─ 完成，耗时 30 秒
```

技术实现

核心 Skill：

```
# weekly_report.skill.md
name: 周报生成器
description: 自动生成产品经理周报
```

```
inputs:
  - name: week_number
    default: "current"

workflow:
  - name: 数据收集
    parallel: true
    steps:
      - tool: exec
        command: python3 get_metrics.py --week={{week_number}}
        output: metrics_data

      - tool: feishu_doc
        action: read
        doc_id: project_status_doc
        output: project_status

      - tool: exec
        command: jira query --status=Done --week={{week_number}}
        output: completed_tasks

  - name: AI分析
    tool: ai_generate
    prompt: |
     基于以下数据生成周报：

     数据：{{metrics_data}}
     项目进展：{{project_status}}
     完成任务：{{completed_tasks}}

     格式：
      1. 本周数据亮点（用emoji标注趋势）
      2. 项目进展（已完成/进行中/计划）
      3. 问题与风险
```

```
      4. 下周计划
    output: report_content

  - name: 生成图表
    tool: exec
    command: python3 generate_chart.py --data={{metrics_data}}
    output: chart_image

  - name: 发布
    steps:
      - tool: feishu_doc
        action: create
        title: "周报_{{week_number}}"
        content: {{report_content}}
        output: doc_url

      - tool: send_email
        to: boss@company.com
        subject: "周报 - {{week_number}}"
        body: {{report_content}}
        attachments: [{{chart_image}}]

      - tool: message
        action: send
        content: "✅ 周报已生成：{{doc_url}}"
```

使用效果

- **时间节省**：2小时 → 30秒
- **数据完整性**：从不遗漏到 100%
- **一致性**：格式统一，风格一致
- **及时性**：周五下班前自动完成

案例 3：SEO内容工厂

背景

需要批量生成 SEO 着陆页，抢占搜索流量： - 每周需要 3-5 个页面 - 每个页面需要：研究关键词、写内容、生成 HTML、部署 - 传统方式：每个页面 4-6 小时

解决方案

端到端自动化内容工厂：

```
┌─────────────────────────────────────────────────────────────┐
│                    SEO内容工厂                              │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│  阶段1: 机会发现（AI自动）                                  │
│  ├─ 抓取 yami.com 热销产品                                  │
│  ├─ 分析搜索趋势                                            │
│  ├─ 识别长尾关键词                                          │
│  └─ 建立内容候选队列                                        │
│                                                             │
│  阶段2: 内容生产（人机协作）                                │
│  ├─ AI 研究产品信息                                         │
│  ├─ AI 生成 SEO 内容                                        │
│  ├─ 生成 Apple 风格 HTML                                    │
│  ├─ 部署预览版                                              │
│  └─ 发送确认卡片 → 人工审核                                 │
│                                                             │
│  阶段3: 发布运营（AI自动）                                  │
│  ├─ Git 提交                                                │
│  ├─ Cloudflare 自动部署                                     │
│  ├─ 提交搜索引擎索引                                        │
│  └─ 发送发布通知                                            │
```

```
│                                                                    │
└────────────────────────────────────────────────────────────────────┘
```

核心代码

Agent 编排：

```
# seo_factory_agent.py

class SEOContentFactory:
    """SEO内容工厂 Agent"""

    def discover_opportunities(self):
        """发现内容机会"""
        # 抓取 yami.com 热销产品
        products = self.tools['browser'].scrape(
            'https://www.yami.com/best-sellers'
        )

        # 分析关键词机会
        for product in products:
            keyword_data = self.tools['serpapi'].analyze(
                product['name']
            )
            if keyword_data['opportunity_score'] > 0.7:
                self.queue.add({
                    'product': product,
                    'keyword': keyword_data
                })

    def create_content(self, task):
        """创建内容"""
        # 研究产品
        research = self.research_product(task['product'])
```

```
        # 生成 SEO 内容
        seo_content = self.ai.generate({
            'template': 'seo_landing_page',
            'product': research,
            'keyword': task['keyword']
        })

        # 生成 HTML
        html = self.build_html(seo_content)

        # 部署预览
        preview_url = self.deploy_preview(html)

        # 发送确认
        self.notify.send_review_card({
            'preview': preview_url,
            'keyword': task['keyword']['term'],
            'actions': ['确认发布', '需要修改', '跳过']
        })

        return {'preview_url': preview_url, 'status': 'pending_review'}

    def publish(self, content_id):
        """发布内容"""
        content = self.memory.get(content_id)

        # Git 提交
        self.tools['git'].commit({
            'files': [content['html_file']],
            'message': f'Add: {content["keyword"]}'
        })

        # 提交搜索引擎
```

```
        self.tools['search'].submit_index(content['live_url'])

        # 发送通知
        self.notify.send_message(
            f"✅ SEO页面已发布：{content['live_url']}"
        )
```

效果

- **生产效率**：4-6小时/页 → 15分钟/页
- **内容质量**：保持 Apple 风格标准
- **发布频率**：1页/周 → 5页/周
- **搜索排名**：3个月内 10+ 关键词进入前3页

案例 4：智能客服系统

背景

公司客服团队每天处理大量重复问题： - 80% 问题是重复的 - 回复速度慢 - 人工成本高 - 非工作时间无法响应

解决方案

AI 智能客服系统：

```
用户问题
    ↓
├─ 是否常见FAQ?
│    ├─ 是 → 直接返回预设答案
│    └─ 否 → 继续
│
├─ RAG 检索知识库
```

```
│    └─ 找到相关文档
│
├─ AI 生成回答
│    └─ 基于检索内容
│
├─ 是否需要人工?
│    ├─ 复杂问题 → 转人工
│    └─ 简单问题 → 直接回复
│
└─ 记录对话到知识库
```

技术架构

```
┌──────────────────────────────────────────────────────────┐
│                      智能客服系统                        │
├──────────────────────────────────────────────────────────┤
│                                                          │
│  渠道接入层                                              │
│  ├─ 网站在线客服                                         │
│  ├─ 微信公众号                                           │
│  ├─ 飞书/钉钉                                            │
│  └─ 邮件                                                 │
│                                                          │
│  处理层                                                  │
│  ├─ 意图识别（FAQ/咨询/投诉）                            │
│  ├─ RAG 检索知识库                                       │
│  ├─ AI 生成回答                                          │
│  └─ 人工接管判断                                         │
│                                                          │
│  知识库层                                                │
│  ├─ FAQ 数据库                                           │
│  ├─ 产品文档                                             │
│  ├─ 历史工单                                             │
│  └─ 对话记录                                             │
```

```
|                                                                    |
└────────────────────────────────────────────────────────────────────┘
```

效果

- **响应时间**：分钟级 → 秒级
- **人工成本**：降低 60%
- **满意度**：提升 20%
- **覆盖时间**：7×24 小时

总结与下一步

4 个案例的共同点

维度	案例1	案例2	案例3	案例4
核心问题	知识管理	重复劳动	内容生产	服务效率
关键技术	RAG+Memory	Skill	Agent+Skill	RAG+Tool
人机协作	低	低	高（审核）	中（转人工）
自动化程度	70%	95%	80%	85%

如何开始你的第一个项目

1. **选择场景**：找一个你最痛的、重复的工作
2. **从简单开始**：先用 Skill 封装，再逐步增强
3. **快速验证**：1周内做出 MVP，验证可行性
4. **持续迭代**：根据使用反馈优化

学习路径建议

```
第1个月：基础应用
  - 掌握 Prompt 工程
  - 建立 MEMORY.md
  - 封装 2-3 个简单 Skill

第2个月：工作流
  - 设计工具链
  - 实现 1 个复杂自动化
  - 学习 MCP

第3个月：系统化
  - 开发 Agent
  - 建立个人/团队 AI 系统
  - 持续优化迭代
```

结语

恭喜你完成了这本《AI 使用完全指南》！

从第一次对话，到构建复杂的 AI 工作流，你已经掌握了： - ✅ 如何与 AI 有效沟通 - ✅ 如何让 AI 使用你的私有知识 - ✅ 如何让 AI 执行实际操作 - ✅ 如何封装和复用 AI 工作流 - ✅ 如何构建端到端的 AI 系统

记住： - 技术只是手段，解决问题才是目的 - 从简单开始，逐步复杂 - 持续实践，不断优化

现在，开始构建你的第一个 AI 工作流吧！

祝你在 AI 提效的道路上一路顺风！

—— 艾登 (Aiden)