Agent Skills 原理 + 应用 + 落地

1. Skills 定义

核心定义

先用一句话定义：

> ***Skill = 可发现、可按需加载的任务 SOP 模块（能力包）。***

它不是一段提示词，它是一个**产品化的能力单元**——包含了元数据、执行指令、示例模板、校验规则，甚至可以带脚本和参考文档。

用 Anthropic 官方的话说：Skills 是用于扩展 AI Agent 功能的**模块化能力包**。每个技能以文件夹形式存在，包含指令、元数据和可选的资源文件。当 Agent 判断某个技能与当前任务相关时，会**自动加载并使用**。

目录结构规范

```shell
skill-name/
├── SKILL.md              # 核心：指令和元数据
├── scripts/              # 可执行代码（Python/Bash/JS）
│   ├── init_skill.py
│   └── validator.sh
├── references/           # 按需加载的文档
│   ├── REFERENCE.md
│   └── api-schema.json
├── assets/               # 静态资源（模板/图片）
│   ├── template.html
│   └── diagram.png
└── LICENSE.txt
```

和 Prompt 的核心差别

维度	Prompt	Skill

生命周期	一次性	持久化、可复用
可发现性	没有，每次要手动贴	Agent 自动匹配
可移植性	跟着聊天窗口走	跨项目、跨平台
版本管理	基本没有	Git 管理，可迭代
团队协作	各写各的	统一分发、统一标准

一句话总结：**Prompt 是便签纸，Skill 是员工手册。**

它解决的 3 个痛点

1. **反复解释规则**——每次新对话都要重新告诉 AI "请遵循以下格式"。有了 Skill，触发即加载，不用重复。
2. **输出风格不稳定**——同样的任务，今天输出 A 格式，明天变 B 格式。Skill 内置模板和校验清单，输出更稳。
3. **团队流程无法复用**——小王写的提示词小李用不了。Skill 标准化后，团队共享一套能力包。

一个直观的例子

假设你经常做 **"某个热点方向的研究报告（调研→对比→结论→路线图）"**：

没有 Skill 的时候：

> *"请帮我做一份研究报告。要求：先给出 1 页 TL;DR；再按'背景/问题定义/技术路线/关键论文与方法对比/数据与评测/落地应用/风险与局限/未来趋势'分章节；对每条路线给优缺点表格；最后给 3 条可执行建议和一个 90 天路线图。写作要偏工程落地，不要空泛，还要保留引用格式....."*

每次都要把**报告结构、对比维度、写作口径、交付物格式**重新讲一遍，稍不注意就会变成"堆概念、没结论"。

有 Skill 的时候：

```text
/research-report [主题] [目标读者] [时长/深度]
```

一句话搞定。因为所有规则已经封装在 Skill 里：

- 自动生成目录与对比框架（路线、指标、数据集、成本、风险）
- 强制输出"结论先行 + 可执行建议 + 路线图"
- 统一引用与表格模板（避免风格漂移）

> ***金句：Skill 的本质，是把个人经验变成组织资产。***

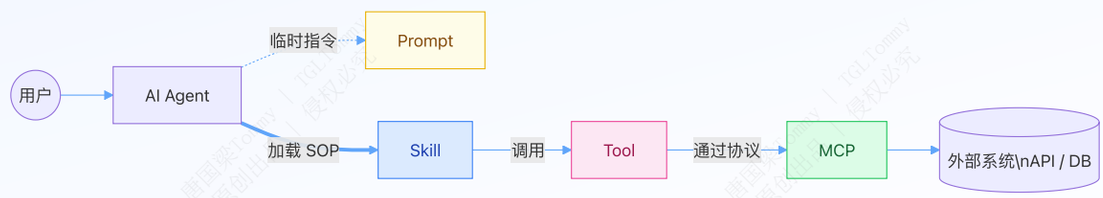
---

2. Prompt / Tool / Skill / MCP 区别

很多人搞不清 Prompt、Tool、Skill、MCP 这四个东西的边界。

四个角色

概念	本质	一句话	类比
Prompt	临时指令	告诉 AI "现在做这个"	口头交代
Tool	一个动作/接口	AI 可以调用的单一操作	一把锤子
MCP	外部系统接入协议	告诉 AI "去哪里拿数据、调什么 API"	电话簿 + 翻译器
Skill	可复用 SOP 包	教 AI "遇到这种事该怎么做"	员工手册



MCP 🆚 Skill

Skill 负责 How（流程与规范），MCP 负责 What/Where（数据与动作）。

两个秒懂场景

场景 1："帮我生成本周周报"

这更像 **Skill**。因为你需要的是一套固定流程：

- 先从任务记录里提取本周完成的事项
- 按模板分"本周完成 / 下周计划 / 风险事项"三段
- 输出格式是 Markdown，口吻要职业化

这些"怎么做"的规范，就是 Skill 的活儿。

场景 2："从 Jira 里拉取本周的任务列表"

这更像 **MCP**。因为你需要的是：

- 连接到 Jira 的 API
- 用我的账号鉴权
- 拉取指定时间范围的 ticket

这些"去哪里连、拿什么数据"，就是 MCP 的活儿。

详细对比表

维度	Agent Skills	MCP
定位	领域知识和工作流的打包	外部工具和数据源的接口
实现方式	提示词注入 + 上下文修改	RPC 协议（服务器-客户端）
部署	本地文件夹	独立服务器进程
交互	单向（指令注入）	双向（请求-响应）
适用场景	复杂多步工作流、领域专业知识	外部 API、数据库、实时数据

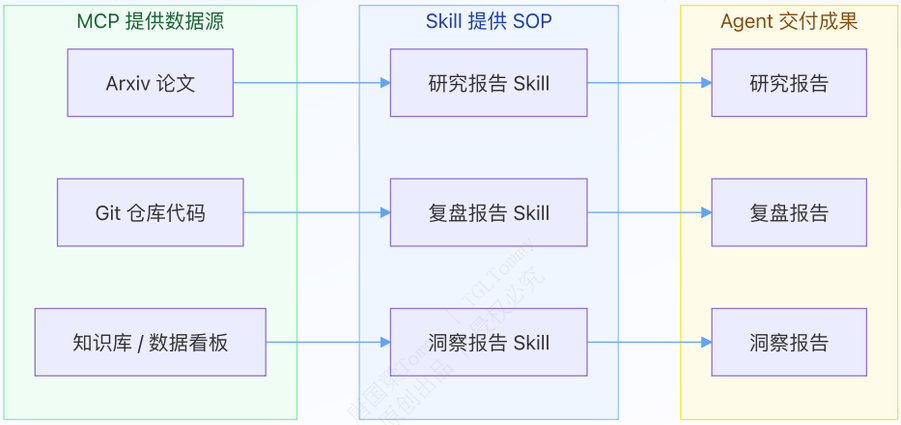
它们是互补关系

最强大的组合是：**MCP 提供数据和工具 → Skill 提供 SOP → Agent 执行闭环。**

比如：

- MCP 连接 Arxiv 拉取论文与引用 → Skill 自动生成"研究报告"

- MCP 读取 Git 仓库的实验代码与配置→ Skill 输出"复现实验复盘报告"
- MCP 拉取内部知识库/会议纪要/数据看板 → Skill 产出"季度技术洞察报告"



> ***金句：Skill 教 AI 怎么干活，MCP 帮 AI 连接世界。***

---

3. 核心原理——按需加载

这个模块讲 Skills 最核心的技术原理——**渐进式披露（Progressive Disclosure）**。

听起来很学术，其实道理很简单：**不要一次性把所有东西都塞给 AI，而是它需要什么再给什么。**

三层加载模型

为什么这比"全塞进去"好？

传统方式（全量加载）：

- 假设你有 50 个工具/技能
- 预加载所有定义 → 约 72,000 tokens
- 加上对话历史和系统提示 → 约 77,000 tokens
- 还没开始干活，上下文就用了大半

Skills 按需加载方式：

- 启动时只加载元数据 → 约 5,000 tokens（50 个技能 × 100 tokens）
- 激活 1 个技能 → 加载约 3,000-5,000 tokens
- 总计约 8,000-10,000 tokens

这是 85% 的 token 节省！

而且不只是省 token——准确率也提升了。根据 SkillPort 的测试数据：

- Claude Opus 4：准确率从 49% 提升到 74%
- Claude Opus 4.5：准确率从 79.5% 提升到 88.1%

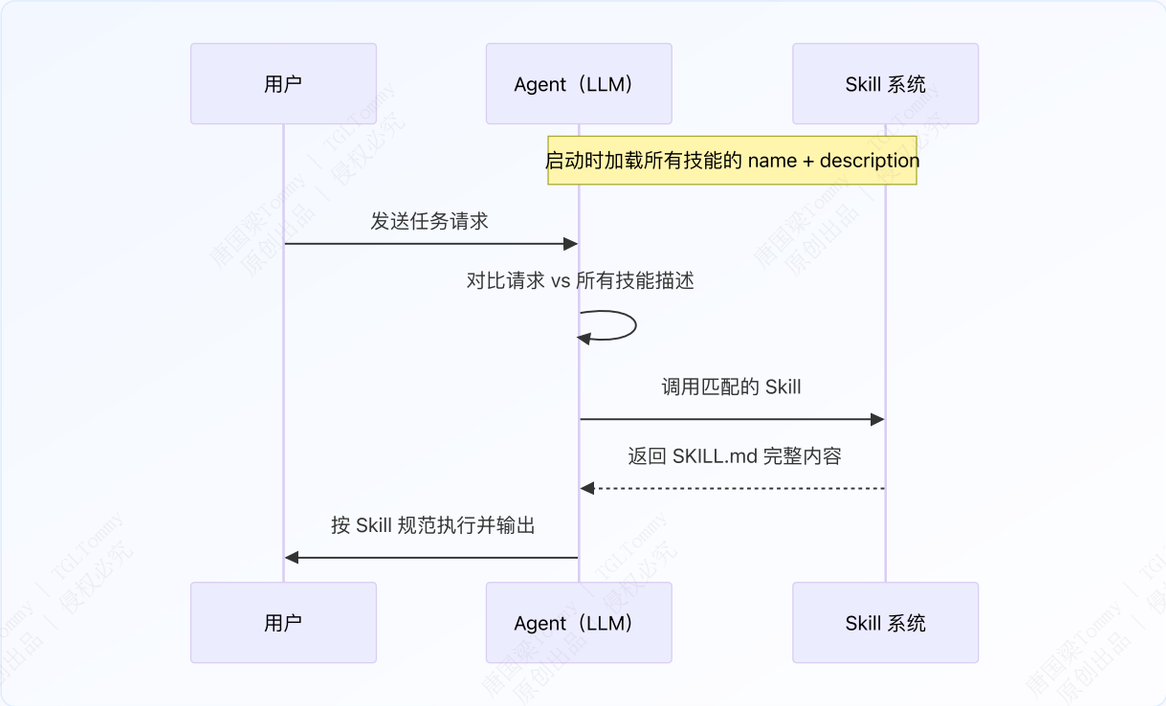
技能选择机制

一个很关键的设计细节：**Skills 的选择完全靠 LLM 自身的推理能力**，没有算法路由，没有意图分类器，没有语义搜索。

流程是这样的：

1. 系统把所有技能的 name + description 格式化成文本，放入 Skill 工具的描述中
2. 用户发来请求
3. LLM 阅读请求，对比所有技能描述，自主判断该用哪个
4. 决定后，通过内部的 "Skill" 工具调用，加载该技能的完整内容

这意味着：**description 写得好不好，直接决定了你的 Skill 能不能被正确触发。**



> ***金句：三层加载 = 最小信息做判断 + 按需加载做执行 = 省上下文 + 更稳定。***

---

4. Skill 的最小结构——SKILL.md 怎么写

这个模块手把手教你写第一个 Skill。

最小结构

一个 Skill 的最小形态就是：**一个文件夹 + 一个 SKILL.md 文件**。

```text
my-skill/
└── SKILL.md          ← 这一个文件就够了
```

复杂一点的 Skill 可以有更多文件：

```text
my-skill/
├── SKILL.md            # 必需：技能定义
├── scripts/            # 可选：可执行脚本
│   └── validator.py
├── references/         # 可选：详细参考文档
│   └── REFERENCE.md
└── assets/             # 可选：模板和静态资源
    └── template.md
```

SKILL.md 的两部分

SKILL.md 由**两部分**组成：

第一部分：YAML Frontmatter（元数据区）

```yaml
---
name: agent-design-reviewer
description: >
  用于生成"Agent 系统设计评审包"。当用户提供一个 Agent 项目想法或现有方案
  （目标、工具、数据源、约束、指标）时使用；自动输出：系统架构图、模块拆解、
  风险清单、评测方案与里程碑计划。不适用于单纯写代码或逐段翻译说明文档。
---
```

第二部分：Markdown 正文（执行指令区）

```text
# Agent 系统设计评审 Skill

## 执行步骤
1. 需求澄清：目标用户/核心任务/输入输出/成功指标（若缺失先列出缺失项）
2. 任务分解：把目标拆成 3–7 个子任务（可工具化/可评估）
3. 架构设计：输出端到端架构（Agent Loop：Think→Plan→Act→Observe→Reflect）
4. 模块拆解：Planner / Tools / Memory / Retriever / Evaluator / Guardrails
5. 风险与边界：失败模式（幻觉、工具误用、权限、数据泄露、成本、延迟）
6. 评测方案：离线集 + 在线集 + 指标（成功率/成本/时延/引用正确率/鲁棒性）
7. 交付计划：30/60/90 天里程碑（PoC→MVP→Prod），每阶段验收标准

## 输出模板
### TL;DR（10 行内）
- 做什么：
- 给谁用：
- 成功标准：
- 关键差异点：

### 架构图（Mermaid）
```mermaid
flowchart LR
U[User] --> P[Planner]
P --> T[Tool Router]
T -->|Call Tools| X[(Tools/MCP)]
X --> O[Observe]
O --> M[Memory/Retrieval]
```
```

```
M --> P
P --> A[Answer]
```

```
## 模块清单（职责与接口）
模块 输入 输出 关键设计点  可替代方案

## 风险清单（Failure Modes）
- 幻觉风险：如何降低（证据引用/自检/多样性采样）
- 工具风险：参数错误/误调用（权限分级/工具白名单）
- 成本风险：token/工具调用频率（缓存/批处理/早停）
- 数据风险：敏感信息（脱敏/审计/最小权限）

## 评测与验收
- 离线：固定测试集（任务成功率、引用正确率、步骤长度、成本）
- 在线：A/B（用户满意度、完成时长、复访率）
- 失败回放：日志字段规范（trace_id、tool_calls、evidence）

## 30/60/90 天计划
- 30 天：PoC（核心链路跑通 + 基线指标）
- 60 天：MVP（覆盖主要任务 + 失败回放闭环）
- 90 天：Prod（权限治理 + 监控报警 + 成本优化）
```

YAML 字段详解

字段	必填	说明
`name`	是	技能名称，小写+连字符，≤64 字符。在 Claude Code 中会变成 `/name` 斜杠命令
`description`	是	≤1024 字符。**最关键的字段**——Agent 靠它判断何时触发
`allowed-tools`	否	技能可自动使用的工具，如 `"Read,Write,Bash"`
`model`	否	指定使用的模型，默认继承当前模型
`version`	否	版本号，便于管理
`disable-model-invocation`	否	设为 true 则 Agent 不会自动触发，只能手动 `/name` 调用

description 写作公式

description 是**决定触发率**的关键。推荐用这个公式：

> ***当用户要做【任务 A / 任务 B】并且需要【输出格式/目标】时使用；当【反例场景】不要使用。***

好的 description 示例：

```yaml
description: >
  从 PDF 文件中提取文本和表格，填写 PDF 表单，合并多个 PDF。
  当需要处理 PDF 文档或用户提到 PDF、表单、文档提取时使用。
```

差的 description 示例：

```yaml
description: 帮助处理 PDF。
```

差在哪？

- 没说清能做什么（提取文本？填表单？合并？）
- 没说清何时用（什么场景触发？）
- 太模糊，Agent 难以判断是否匹配

> ***金句：name 决定你怎么调用它，description 决定 AI 什么时候自动用它。***

唐国梁Tommy 原创出品 | TGLTommy 侵权必究

5. 最佳实践——把 Skill 当"产品"打磨

会写 Skill 只是第一步，写好 Skill 才是关键。这里分享经过多个生产级项目验证的最佳实践。

三条黄金规则

#1 多给 examples
好结果长什么样

每条规则都要有"错误示范"和"正确示范"——AI 通过对比学得最快。

Incorrect 缺少索引导致全表扫描

```
SELECT * FROM orders WHERE customer_id = 123;
-- Seq Scan, 2500ms
```

Correct 添加索引后的高效查询

```
CREATE INDEX orders_customer_id_idx
  ON orders (customer_id);
SELECT * FROM orders WHERE customer_id = 123;
-- Index Scan, 2.3ms, 提升 1087x
```

#2 复杂内容下沉到 references
正文控制在 5,000 tokens 内

SKILL.md 正文保持精简，超出的内容放到 references/ 目录，Agent 只在需要时才加载。

```
## 详细规范

基础用法见上述步骤。更多高级配置请参阅：
- [表单处理指南](references/FORMS.md)
- [API Schema](references/api-schema.json)
```

#3 加输出校验清单
让 Agent 输出前自检

在 Skill 末尾加 checklist，让 Agent 在输出前逐项自检，显著提升输出质量。

- [x] 输出格式是否符合模板
- [x] 是否包含所有必需字段
- [x] 语气是否一致（口语化 vs 书面）
- [x] 代码示例是否可运行
- [x] 是否有遗漏的边界情况

版本管理

像管理代码一样管理 Skill：

- 用 Git 做版本控制
- 在 YAML 中加 `version` 字段
- 每次修改写 changelog
- 重大改动升版本号

> ***金句：Skill 的迭代方式更像写库/写组件，而不是写提示词。***

6. Skill × MCP——组合成生产级 Agent

前面我们分别讲了 Skill 和 MCP。这个模块讲最关键的——怎么把它们组合起来，变成真正能落地的生产系统。

组合架构

一句话：**MCP 给 Agent 连上"手和眼"，Skill 给 Agent 装上"脑"。**

三者的分工边界

维度	MCP Server	Skill	Agent
核心职责	连接外部系统	定义怎么做	综合执行
提供什么	数据 + 工具	流程 + 规范	决策 + 交付
类比	手和眼	操作手册	有经验的员工
示例	GitHub API、数据库、Slack	PR 审查流程、SQL 优化规范	Claude Code、Cursor Agent

> ***关键洞察**：MCP 和 Skill 是互补关系，不是替代关系。MCP 解决"能不能做"（能力），Skill 解决"怎么做好"（质量）。两者结合，Agent 才能既有能力又有章法。*

Agent Skills — 原理 + 应用 + 落地

课程官网：TGLTommy.com（国内访问需科学上网）

AI进阶学习

🖥️ **B站课堂**：https://space.bilibili.com/3546748815411393/pugv

🌐 **官方网站**： https://www.tgltommy.com/ (国内访问需科学上网)

如果你正在关注大模型 Agent、强化学习后训练、RLHF、DPO、GRPO、RLVR 等前沿方向，欢迎关注我最新上线的精品课程：《大模型 Agent 强化学习实战：从后训练、奖励设计到工业级对齐系统》

这门课程围绕当前大模型 Agent 强化学习的核心技术路线展开，不只是介绍零散算法和论文概念，而是希望帮助学习者建立一套完整的技术地图：从基础原理、论文脉络、关键算法，到开源项目源码解析与工业级系统实践，逐步理解 Agent RL 为什么重要、如何演进，以及在真实应用中如何落地。

课程配套专属飞书知识库 《Agent RL 学习宝典》，包含学习路线、论文资料、项目说明、源码阅读辅助资料和后续持续更新内容，方便长期查阅和跟进前沿技术。

Agent RL 变化很快，但越是变化快，越需要一套系统的学习框架。如果你希望系统进入 Agent RL 方向，而不是停留在碎片化了解阶段，这门课会是一个很好的起点。

课程官网：https://www.tgltommy.com/p/agent-rl（国内访问需科学上网）
B站课堂：https://www.bilibili.com/cheese/play/ss842375604

更多课程详情，扫描以下二维码咨询课程助理：

唐国梁Tommy 原创出品 | © 2026 熵智未来（深圳）科技有限公司。保留所有权利。

关注微信公众号
获取更多学习资源

官方网站：TGLTommy.com(需科学上网)

唐国梁Tommy 原创出品 | © 2026 熵智未来（深圳）科技有限公司。保留所有权利。