河马的CC商业实战 · 会员专属资料 #7

2026年8月全面修订 · MCP 新规范 · 31 个 Hooks 事件

Claude Code 兵器谱

Tools · MCP · Skills 全景实战手册

工具生态全景 · 内置 Tools 完全解析 · MCP 服务与 2026 新规范 · Skills 与 CLAUDE.md 模板库
斜杠命令 · Hooks 自动化 · 国内可用性盘点 · 配置模板合集

河马的CC商业实战 出品

2026年8月 · 会员专属版

会员专属资料 · 第 7 期

Claude Code 兵器谱

MCP · Skills · Tools 全景实战手册（2026年8月版）

出品 河马的CC商业实战（知识星球）

版本 2026年8月 · 会员专属版（基于2026年5月版全面核实更新）

读者 已入门 Claude Code、想把工具生态用透的开发者

> 本资料为「河马的CC商业实战」星球会员专属，每周/不定期更新。欢迎转发给需要的朋友——如果他们想持续收到这个系列，封底有入口。工具生态迭代极快，命令与配置以官方文档和你实际版本为准，组合思路不变。

河马卷首语

星友们，「会员专属资料」第 7 期来了。

这一期和第 6 期是「姊妹篇」：第 6 期解决「国内怎么用上 Claude Code」，这一期解决「用上之后怎么玩到精通」。这本《兵器谱》把 Claude Code 的五大子系统——内置 Tools、MCP、Skills、斜杠命令、Hooks——挨个拆开讲，每件兵器都标了国内能不能直连、给了能直接抄的配置。

这三个月生态变化很大，本版已全部同步：MCP 捐赠给 Linux 基金会 AAIF、7 月发布了迄今最大规范改版；Hooks 从 12 个事件扩到 31 个、执行方式从 1 种扩到 4 种；还更正了 5 月版的一处错误（子代理工具是 Task 更名 Agent，不是反过来）。国内生态新增了高德、支付宝等官方 MCP Server。

用法建议：第一章建立全景认知；第三、六章（MCP 和 Hooks）是含金量最高的两章，值得细读；第七章国内可用性盘点和附录配置模板，建议收藏当工具书查。

—— 河马，2026年8月

目 录

FOREWORD

前言 为什么需要这份手册

CHAPTER ONE

第一章 工具生态全景图

1.1 架构全景图

1.2 核心组件速查表

CHAPTER TWO

第二章 内置 Tools 完全解析

2.1 文件操作类

2.2 搜索与目录类

2.3 命令执行类

2.4 网络获取类

2.5 任务与规划类

2.6 其他内置工具

2.7 内置工具权限管理与安全配置

CHAPTER THREE

第三章 MCP 服务——让 Claude Code 连接万物

3.1 MCP 协议简述

3.2 MCP 安装与管理命令

3.3 官方 MCP Server 详解

3.4 社区 MCP Server 分类推荐

3.5 国内开发者 MCP 推荐清单

CHAPTER FOUR

第四章 Skills 技能系统与 CLAUDE.md 实战

4.1 CLAUDE.md 三层结构

4.2 Skills 机制详解

4.3 高质量 CLAUDE.md 编写指南

4.4 CLAUDE.md 实战模板库

CHAPTER FIVE

第五章 斜杠命令与自定义工作流

5.1 内置斜杠命令完整清单

5.2 自定义斜杠命令

CHAPTER SIX

第六章 Hooks 钩子——自动化你的开发管道

6.1 四种 Hook 类型

6.2 Hooks 配置方法

6.3 实用 Hook 配置案例

CHAPTER SEVEN

第七章 国内可用性综合评估

7.1 无需任何配置即可使用的工具清单

7.2 需要 API Key 但国内可直连的工具

7.3 需要代理才能使用的工具

7.4 国内开发者推荐工具栈

APPENDIX

附录

A. MCP Server 分类速查表（浓缩版）

B. 推荐配置模板合集

常见工具使用问题排查

C. 参考资源链接

D. 信息来源与免责声明

FOREWORD

前言　为什么需要这份手册

Claude Code 在 2026 年已经不再是那个简单的「命令行聊天工具」。经过两年多的迭代，它演变成了一个完整的智能代理开发环境——一个以终端为界面、以代码仓库为中心、以工具调用为行动方式的自主 Agent 平台。对于已经具备基础使用经验的中国开发者而言，真正制约效率提升的瓶颈，往往不在于「如何用」，而在于「能用什么」以及「如何组合使用」。

在日常开发中，你是否遇到过这些场景：

- 想让 Claude Code 直接查询数据库 Schema，却不知道该接哪个 MCP Server；
- 看到同事用一条 `/deploy` 命令完成测试、构建、发布全流程，想了解如何配置；
- 担心 Claude 自动执行 `rm -rf` 或向外部发送请求，想建立安全边界却不知从何入手；
- 国内网络环境下，不确定哪些工具能直连、哪些需要 API Key、哪些必须挂代理；
- 面对大量内置 Tools、MCP Server、Skills 和 Hooks，不知道它们的边界和组合方式。

本手册将系统性整理 Claude Code 截至 2026 年 8 月的全部可用工具、扩展和配置能力，覆盖内置 Tools、MCP 服务、Skills 技能、斜杠命令和 Hooks 钩子五大子系统。每个工具都标注国内可用性（✅ 直连 / ⚠️ 需 API Key 或代理 / 🔴 不可用），并提供可直接复制运行的配置示例。

全书结构遵循「先全貌、后细节、再实战」的原则：第一章给出架构全景，第二至六章深入每个子系统，第七章针对国内开发者做可用性盘点，附录提供速查表和配置模板。你可以按顺序阅读建立完整认知，也可以按需跳转查阅特定工具。

版本说明：本手册基于 Claude Code CLI 2.1.x 版本（2026 年 8 月最新稳定版，v2.1.23x 系列）编写。工具名称以官方文档 code.claude.com 为准，已确认以下关键变更：

- `Task` 工具已正式更名为 `Agent`（v2.1.63；5月版手册所述方向有误，本版已按官方 changelog 更正），配置中两个名字暂均兼容；
- `MultiEdit` 已合并入 `Edit`，不再作为独立工具存在；
- `EnterPlanMode` / `ExitPlanMode` 仍然保留，但新增了 Plan Panel 侧边栏可视化支持；
- `.mcp.json` 已成为项目级 MCP 配置的首选方式，取代早期的 settings.json 内嵌方式；
- Hooks 系统 2026 年大幅扩展：事件达 31 个（新增 PermissionRequest、SessionStart/End、PreCompact/PostCompact、SubagentStart/Stop、PostToolUseFailure、WorktreeCreate/Remove 等），handler 类型扩为 command / prompt / agent / http 四种；
- Skills 格式于 2025 年 12 月成为跨平台开放标准（agentskills.io），Cursor 已官方支持、Codex CLI 兼容（读取 .agents/skills）；
- 2026 年 6-7 月模型换代：Opus 5 成为 Claude Code 默认模型（7月24日），Fast 模式（/fast）同步推出；
- 2026 年 8 月 14 日 Auto Mode 对付费账户默认开启；
- MCP 于 2025 年 12 月捐赠给 Linux 基金会新成立的 Agentic AI Foundation（AAIF），2026 年 7 月 28 日发布迄今最大规范改版（无状态核心 + server/discover）。

CHAPTER ONE

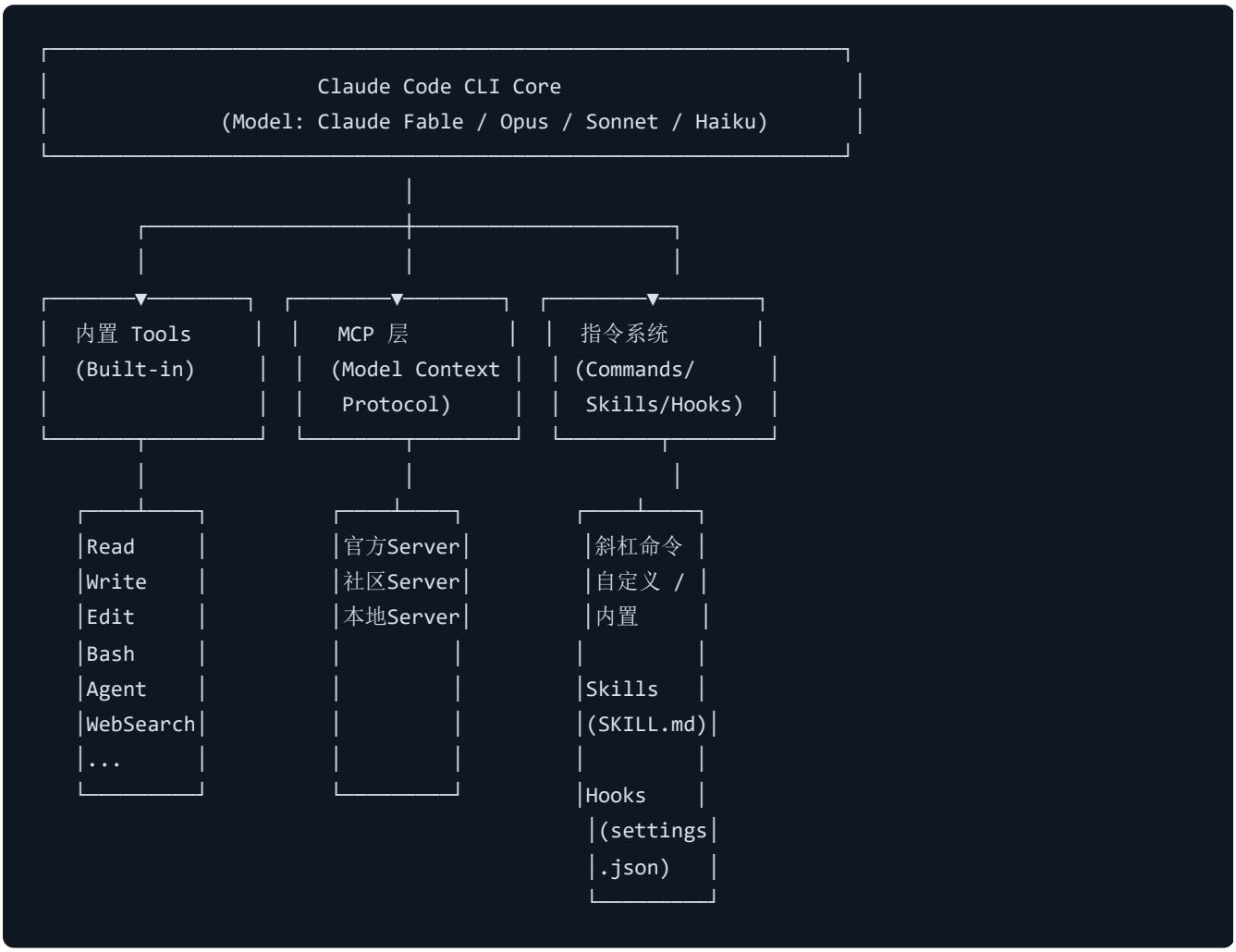
第一章　工具生态全景图

1.1 架构全景图

1.2 核心组件速查表

1.1 架构全景图

Claude Code 的工具生态由五个相互协作的子系统构成，可以用以下 ASCII 架构图表示：



子系统关系说明：

1. **内置 Tools**：Claude Code 原生集成的工具，无需任何配置即可使用，覆盖文件操作、命令执行、网络请求、任务规划等核心能力。这是每个会话的基础工具集，构成了 Claude 与本地系统交互的「原子能力」。
2. **MCP 层**：通过 Model Context Protocol 连接外部服务，让 Claude Code 获得访问数据库、浏览器、第三方 API 的能力。MCP Server 以独立进程运行，通过 JSON-RPC 与 Claude Code 通信。MCP 的设计哲学是「USB-C for AI」——一个标准化接口连接无限外设。
3. **指令系统**：包含斜杠命令（快速触发）、Skills（按需加载的领域知识包）和 Hooks（生命周期钩子），三者共同构成 Claude Code 的「可编程接口」。如果说内置 Tools 和 MCP 是 Claude 的「手脚」，那么指令系统就是它的「肌肉记忆」和「条件反射」。

五个子系统并非孤立运作。一个典型的工作流可能是：用户输入 `/deploy`（斜杠命令）→ 触发 `deploy` Skill → Skill 指令调用 Bash 运行测试 → 调用 MCP 的 Docker Server 构建镜像 → 最后通过 GitHub MCP Server 创建 PR。Hooks 则贯穿全程，在 Write 后自动格式化、在 Bash 前检查危险命令、在任务完成后发送通知。

1.2 核心组件速查表

组件	功能定位	配置位置	使用频率	是否需配置
内置 Tools	文件、命令、网络、规划的原子操作	无需配置，开箱即用	极高	否
MCP Server	连接数据库、浏览器、外部 API	`.mcp.json` / `~/.claude.json`	高	是
Skills	按需加载的可复用领域指令	`.claude/skills/` / `~/.claude/skills/`	高	是
斜杠命令	快速触发内置或自定义功能	`.claude/commands/`	中	可选
Hooks	生命周期自动化钩子	`settings.json`（项目/用户级）	中	是
CLAUDE.md	项目级系统提示与上下文	仓库根目录	高	可选

配置层级优先级（从高到低）：

1. `.claude/settings.local.json` —— 个人项目级（Git 忽略）
2. `.claude/settings.json` —— 项目级（团队共享）
3. `~/.claude/settings.json` —— 用户全局级
4. `/etc/claude-code/managed-settings.json` —— 企业策略级（Linux）
5. `CLAUDE.md` —— 项目上下文（始终加载，但可被上层配置覆盖部分行为）

理解配置优先级至关重要。当同一设置在不同层级冲突时，编号小的层级胜出。例如，若 `~/.claude/settings.json` 将默认模型设为 `sonnet`，而 `.claude/settings.json` 设为 `opus`，则当前项目使用 `opus`。

CHAPTER TWO

第二章 内置 Tools 完全解析

2.1 文件操作类

2.2 搜索与目录类

2.3 命令执行类

2.4 网络获取类

2.5 任务与规划类

2.6 其他内置工具

2.7 内置工具权限管理与安全配置

2.1 文件操作类

截至 2026 年 8 月，Claude Code 共有约 20 个主要内置工具（不含企业版特有工具），按功能分为六类。以下逐一详解，每个工具包含：一句话总结、详细描述、使用示例、最佳实践场景、国内可用性、推荐优先级。

> **2026年8月重要更名提醒**：`Task` 工具已在 v2.1.63（2026 年上半年）正式更名为 `Agent`（与 5月版手册所述方向相反，5月版有误，本版已按官方 changelog 更正）。权限配置和 Hooks 的 matcher 中 `Task` 与 `Agent` 两个名字目前都被接受以保持兼容，但新配置请使用 `Agent`。此外 2026 年新增的内置工具还有 **Cron 系列**（v2.1.71，配合 /loop 定时调度）与 **LSP**（v2.0 起内置，代码智能感知）。

Read（读取文件内容）

一句话总结：读取指定路径的文件内容，支持文本、图片和 PDF 解析。

详细描述：Read 是 Claude Code 最基础的工具之一。它会将文件内容读入上下文窗口，供模型分析和处理。支持 UTF-8 文本文件；遇到 `.png`、`.jpg`、`.pdf` 等格式时，会自动提取内容（PDF 转文本、图片 OCR）。对于大文件，Read 会自动截断并提示用户。

Read 支持两个可选参数：`offset`（起始行号）和 `limit`（读取行数），这让 Claude 可以分段读取大文件，避免一次性占用过多上下文。当 Claude 需要分析一个 5000 行的日志文件时，它会先读取前 100 行了解结构，再按需读取后续部分。

使用示例：

```
{
  "file_path": "src/components/Header.tsx",
  "offset": 1,
  "limit": 50
}
```

最佳实践场景： - 分析现有代码文件以理解逻辑 - 读取配置文件（`package.json`、`tsconfig.json`）获取项目元数据 - 查看日志文件排查错误 - 读取 CLAUDE.md 或 SKILL.md 加载项目上下文

国内可用性：✅ 直连可用，无需任何配置。 **推荐优先级**：★★★★★（每日必用）

Write（创建新文件）

一句话总结：在指定路径创建新文件，若文件已存在则报错。

详细描述：Write 工具用于生成全新文件。与 Edit 不同，它不负责修改已有文件——如果目标路径已存在文件，Write 会拒绝执行并提示用户先删除或使用 Edit。这个安全设计避免了意外覆盖。

Write 对于代码生成场景至关重要。当你让 Claude「创建一个 React 组件」时，Claude 会调用 Write 将生成的代码写入新文件。Write 同样适用于生成配置文件、文档、脚本等任何需要从头创建的内容。

使用示例：

```
{
  "file_path": "docs/api.md",
  "content": "# API 文档\n\n## 认证\n..."
}
```

最佳实践场景： - 创建新组件、新模块文件 - 生成 README、文档、配置文件 - 批量生成脚手架代码时配合 Bash 使用

国内可用性：✅ 直连可用。 **推荐优先级**：★★★★★（每日必用）

Edit（精确差异编辑）

一句话总结：对已有文件执行精确的字符串替换或基于行号的范围修改。

详细描述：Edit 是文件修改的主力工具。2025 年底的更新将原先独立的 `MultiEdit` 功能完全合并入 Edit，现在 Edit 支持： - 单行精确替换（old_string + new_string） - 多行范围替换 - 多文件批量编辑（原 MultiEdit 能力）

Edit 要求文件必须已被 Read 加载到上下文中才能执行，这个依赖关系模型确保了 Claude 在修改前已理解文件内容。Edit 的核心匹配算法基于精确字符串匹配，这意味着 old_string 必须与原文件内容完全一致（包括空格和换行），否则编辑会失败。

在实际使用中，Claude 通常会先 Read 文件内容，然后构造 Edit 调用。如果文件在行号调用之间发生了变化（例如用户在外部编辑器中修改了文件），Edit 可能会失败并提示内容不匹配。

使用示例：

```
{
  "file_path": "src/config.ts",
  "old_string": "const API_URL = 'http://localhost:3000';",
  "new_string": "const API_URL = process.env.API_URL || 'http://localhost:3000';"
}
```

最佳实践场景： - 修改函数实现、调整变量名 - 重构代码模式（配合 Grep 定位后批量替换） - 更新配置项、修复拼写错误 - 在多个文件中同步修改接口定义

国内可用性：✅ 直连可用。 **推荐优先级**：★★★★★（每日必用）

> **重要说明**：`MultiEdit` 已合并入 `Edit`，不再作为独立工具存在。若你在旧文档中看到 MultiEdit，请直接替换为 Edit 的多文件调用方式。Claude 现在可以在单次请求中对多个文件分别调用 Edit，实现原 MultiEdit 的批量编辑效果。

2.2 搜索与目录类

Glob（文件模式匹配）

一句话总结：基于通配符模式查找文件路径列表。

详细描述：Glob 使用标准通配符语法（`*` 匹配任意字符，`?` 匹配单个字符，`**` 递归匹配目录）来查找文件。底层在可用时会调用 `bfs`（breadth-first search 工具）提升性能。返回结果按路径排序，方便 Claude 按顺序处理。

Glob 是 Claude 理解项目结构的第一步。当进入一个新仓库时，Claude 通常会先执行 `Glob("**/*.ts")` 或 `Glob("package.json")` 来了解技术栈和文件组织方式。合理使用 Glob 可以显著减少 Claude 的「探索时间」。

使用示例：

```
{
  "pattern": "src/**/*.test.ts"
}
```

最佳实践场景： - 查找所有测试文件以分析覆盖率 - 定位特定类型的配置文件（`*.config.js`） - 批量获取文件列表后配合 Read 逐个分析 - 检查特定目录下的文件命名规范

国内可用性：✅ 直连可用。 **推荐优先级**：★★★★☆（高频使用）

Grep（内容正则搜索）

一句话总结：在文件内容中执行正则表达式搜索。

详细描述：Grep 支持标准正则语法，可在指定路径或全仓库范围内搜索。底层在可用时调用 `ugrep` 提升性能。返回匹配文件的路径、行号和上下文片段。Grep 是理解大型代码库的利器——通过搜索函数名、变量名或特定模式，Claude 能快速定位相关代码。

Grep 的 `output_limit` 参数控制返回结果数量，防止大代码库的搜索淹没上下文。当搜索非常通用的模式时，Claude 会自动限制结果数量，并建议缩小搜索范围。

使用示例：

```
{
  "pattern": "useAuth\(",
  "path": "src",
```

```
  "output_limit": 20
}
```

最佳实践场景： - 查找函数定义和调用点 - 追踪变量在代码库中的传播路径 - 发现重复代码模式或待重构的坏味道 - 检查特定 API 的使用方式

国内可用性： ✅ 直连可用。 **推荐优先级**：★★★★★（每日必用）

ToolSearch（搜索延迟加载的工具）

一句话总结：搜索当前可用的延迟加载工具（包括 Skills 和 MCP Tools）。

详细描述：ToolSearch 用于发现那些未在初始上下文中加载、但当前环境可用的工具。当你安装了 MCP Server 或 Skills 后，ToolSearch 可以帮助 Claude 了解「我现在能做什么」。这个工具在复杂项目中特别有用——它能列出超出内置工具集的扩展能力。

ToolSearch 的查询参数是自由文本，Claude 会将查询与工具的 name、description 进行匹配。对于 MCP Tools，ToolSearch 会显示其完整的名称（如 `mcp__postgres__query`），以便后续调用。

使用示例：

```
{
  "query": "database"
}
```

最佳实践场景： - 不确定当前环境有哪些 MCP 工具可用时 - 查找与特定任务相关的 Skill - 探索插件生态中的能力边界 - 确认某个 MCP Server 是否正确加载

国内可用性： ✅ 直连可用。 **推荐优先级**：★★★☆☆（按需使用）

2.3 命令执行类

Bash（执行系统命令）

一句话总结：在本地 shell 中执行命令，支持输出捕获和交互式输入。

详细描述：Bash 是 Claude Code 与操作系统交互的桥梁。它可以运行任何 shell 命令，包括编译、测试、Git 操作、包管理器等。Bash 支持 `timeout` 参数防止命令无限挂起，也支持 `wait_for_input` 处理交互式程序。企业版还支持沙箱模式（sandbox）隔离执行环境。

Bash 的输出会被捕获并返回给 Claude。对于长时间运行的命令（如 `npm install`），Claude 可以设置较长的 timeout。如果命令需要用户交互（如 `git rebase -i`），`wait_for_input` 模式会让 Claude 暂停并提示用户输入。

使用示例：

```
{
  "command": "npm test",
  "timeout": 120000
}
```

最佳实践场景： - 运行测试套件验证代码修改 - 执行 Git 操作（status、diff、commit） - 调用构建脚本（`npm run build`、`make`） - 查询系统信息（`df -h`、`ps aux`） - 安装依赖（`npm install`、`pip install`）

国内可用性：✅ 直连可用。注意：若命令本身涉及外部网络（如 `curl` 访问境外 API），则网络连通性取决于命令本身，而非 Bash 工具。
推荐优先级：★★★★★（每日必用）

Agent（执行子任务/子代理，原 Task）

一句话总结：启动一个独立的子代理会话执行特定任务，v2.1.63 起由 Task 更名为 Agent。

详细描述：Agent 工具（原名 Task，2026 年上半年 v2.1.63 更名）是 Claude Code 实现「分而治之」的核心机制。它创建一个具备独立上下文窗口的子代理，可以并行或串行执行复杂任务。主会话可以通过 Agent 将大任务拆解为多个子任务，每个子任务获得独立的工具权限和模型配置。子代理完成后返回结果，主代理整合输出。

Agent 工具对于复杂项目尤为关键。当任务涉及多个不相关模块时，并行子代理可以显著加速完成时间。每个子代理的上下文窗口是独立的，这意味着一个子代理的失误不会污染主会话的上下文。2026 年的子代理体系进一步成熟：默认嵌套深度 3、取消单会话子代理数量上限，配合 SendMessage 可在代理间传递消息、TaskStop 可终止运行中的代理。

使用示例：

```
{
  "description": "Refactor auth module",
  "prompt": "Refactor the authentication module in src/auth/ to use JWT instead of session cookies. Run tests after each change.",
  "tools": ["Read", "Edit", "Bash"]
}
```

最佳实践场景： - 大型重构任务拆解为模块级子任务 - 并行执行独立的代码审查或测试 - 隔离高风险操作（如数据库迁移）到受限权限子代理 - 同时处理前端和后端的相关但独立变更

国内可用性：✅ 直连可用。 **推荐优先级**：★★★★☆（复杂任务必用）

> **重要说明**：v2.1.63 起 `Task` 正式更名为 `Agent`。权限配置和 Hooks matcher 中两个名字都被接受以保持兼容，但新配置请使用 `Agent`。功能上完全一致，仅名称变更。（5月版手册此处写反了更名方向，本版已按官方 changelog 更正。）

2.4 网络获取类

WebFetch（获取网页内容并转 Markdown）

一句话总结：获取指定 URL 的网页内容，自动转换为 Markdown 格式供分析。

详细描述：WebFetch 负责将网页内容拉取到 Claude 的上下文中。它不仅是简单的 HTTP GET——还会执行智能内容提取，过滤导航栏、广告等噪音，保留正文和代码块。对于技术文档、博客文章、API 文档等场景，WebFetch 比纯文本抓取更易读。

WebFetch 支持常见的 HTTP 协议，但对于需要复杂 Cookie、Header 或身份验证的页面，可能不如 MCP 的 Fetch Server 灵活。WebFetch 的内容长度也受上下文窗口限制，超长页面会被截断。

使用示例：

```
{
  "url": "https://react.dev/reference/react/useEffect"
}
```

最佳实践场景： - 查阅官方技术文档 - 获取最新库的版本说明或 Changelog - 参考 Stack Overflow 或 GitHub Issues 的解决方案 - 获取 API 文档辅助代码生成

国内可用性：⚠️ 取决于目标网站。访问境外网站（如 GitHub、npmjs.org）通常需要代理；国内站点（如 juejin.cn、csdn.net）可直连。WebFetch 工具本身没有网络限制，限制来自目标站点的可达性。 **推荐优先级**：★★★★☆（高频使用）

WebSearch（网页搜索）

一句话总结：执行网络搜索并返回相关结果摘要。

详细描述：WebSearch 让 Claude Code 具备实时信息获取能力。它调用底层搜索引擎（通常是 Bing 或 Google）执行查询，返回标题、链接和片段摘要。WebSearch 是解决「知识截止日期」问题的关键工具——当 Claude 需要了解 2026 年最新发布的库版本或框架更新时，WebSearch 可以补充训练数据之外的信息。

WebSearch 返回的结果通常包含 5-10 条摘要，Claude 可以根据这些摘要判断是否需要进一步 WebFetch 某个页面。由于搜索结果依赖外部搜索引擎，其质量和延迟不可控。

使用示例：

```
{
  "query": "Claude Code MCP server filesystem 2026"
```

```
}
```

最佳实践场景： - 查询最新版本号或发布说明 - 调研技术方案对比（"PostgreSQL vs MySQL performance 2026"） - 搜索报错信息的解决方案 - 了解新框架或库的基本用法

国内可用性： ⚠️ 需要代理或配置国内搜索引擎。Claude Code 的默认搜索后端面向国际网络，国内直连可能不稳定。建议配合 MCP 层的 SearXNG 或 Brave Search Server 使用。 **推荐优先级**：★★★☆☆（按需使用）

2.5 任务与规划类

EnterPlanMode / ExitPlanMode（规划模式）

一句话总结：进入只读规划模式让 Claude 制定方案，退出后开始执行。

详细描述：Plan Mode 是 Claude Code 的「设计先行」机制。调用 EnterPlanMode 后，Claude 进入只读状态，分析现有代码库并输出一份 Markdown 格式的实施计划到 `.claude/plans/` 目录。用户在 Plan Panel（2026 年 4 月桌面端新增侧边栏）中审阅并批准后，Claude 调用 ExitPlanMode 读取计划文件并开始执行。这种模式特别适合复杂重构或架构调整——避免 Claude 在思考不周时直接修改代码。

Plan Mode 的本质是一个状态机。EnterPlanMode 注入额外的系统提示，提醒 Claude 当前处于规划状态；ExitPlanMode 则清除该状态并加载计划文件内容作为后续执行的指令。根据社区分析，Plan Mode 并不真正禁用编辑工具——而是通过提示约束 Claude 的行为。

使用示例：

```
// EnterPlanMode
{}

// ExitPlanMode（在计划完成后调用）
{
  "summary": "Plan complete, ready to execute"
}
```

最佳实践场景： - 大型重构前的方案设计（迁移框架、调整架构） - 多步骤复杂任务的事前规划 - 需要人类审批的关键代码变更 - 探索性分析后的实施计划制定

国内可用性：✅ 直连可用。Plan Mode 完全本地运行，不依赖外部网络。 **推荐优先级**：★★★★☆（复杂任务推荐）

EnterWorktree / ExitWorktree（Git worktree 管理）

一句话总结：将当前会话切换到独立的 Git worktree，实现并行分支开发。

详细描述：EnterWorktree 为当前任务创建一个隔离的 Git worktree（通常位于 `.claude/worktrees/`），让 Claude 在一个干净的分支副本上工作，不影响主仓库的当前状态。这是并行开发、实验性重构和代码审查的理想工具。Worktree 的生命周期与 Claude 会话绑定——会话结束时可以选择保留或清理。

Worktree 的价值在于隔离性。当你在主分支上有一个进行中的功能时，可以立刻创建一个 worktree 去修复紧急 Bug，两个任务互不干扰。Claude Code 的桌面端在 2026 年初存在自动创建 worktree 的行为，若不需要可在 CLI 中通过权限配置禁用 EnterWorktree。

使用示例：

```
// EnterWorktree
{
  "path": "feature-auth",
  "branch": "feature/auth-refactor"
}

// ExitWorktree（返回主仓库）
{
  "keep": true
}
```

最佳实践场景： - 在功能分支上独立开发，不污染当前分支 - 并行审查多个 PR（每个 worktree 一个 PR） - 实验性重构的「安全沙盒」 - 紧急热修复而不打断当前功能开发

国内可用性：✅ 直连可用。纯本地 Git 操作，无网络依赖。 **推荐优先级**：★★★☆☆（分支管理推荐）

CronCreate / CronList / CronDelete（定时任务）

一句话总结：在 Claude Code 会话内创建、列出和删除定时提醒/循环任务。

详细描述：这组工具构成了 Claude Code 的内置调度器。CronCreate 创建任务（支持自然语言如 "every 10 minutes"、"at 3pm"），CronList 查看当前任务，CronDelete 取消任务。每个任务获得一个 8 字符 ID，会话内最多 50 个任务。注意：这些任务是会话级——退出 Claude Code 后全部消失，需要持久化调度请使用系统 cron 或 CI/CD。

除了通过工具调用，用户还可以直接在对话中使用自然语言创建定时任务，如 "remind me at 3pm to push the release branch" 或 "/loop 5m check if the deployment finished"。Claude 会自动调用 CronCreate 处理这些请求。

使用示例：

```
// CronCreate
{
  "prompt": "check if the deployment finished",
  "interval": "5m"
}

// CronList
{}

// CronDelete
{
  "task_id": "a1b2c3d4"
}
```

最佳实践场景： - 长时间构建/部署过程的轮询检查 - 定时提醒会议或代码审查 - 循环监控日志或指标 - 定期备份或状态检查

国内可用性： ✅ 直连可用。 **推荐优先级**：★★☆☆☆（特定场景使用）

2.6 其他内置工具

LSP（代码智能）

一句话总结：与语言服务器通信，获取定义、引用、诊断等 IDE 级代码智能。

详细描述：LSP 工具让 Claude Code 具备了类似 VS Code 的代码理解能力。它通过 Language Server Protocol 与项目已配置的语言服务器交互，支持：跳转到定义、查找引用、获取类型信息、读取诊断错误等。对于 TypeScript、Python、Rust 等强类型语言，LSP 显著提升了 Claude 重构代码的准确性。

LSP 的价值在于「超越文本」的理解。通过 LSP，Claude 可以知道一个函数的确切类型签名、一个变量在所有文件中的引用位置、当前代码的诊断错误等。这些信息仅靠文本搜索难以获取。

使用示例：

```
{
  "language": "typescript",
  "command": "definition",
  "file_path": "src/App.tsx",
  "line": 15,
  "column": 8
}
```

最佳实践场景： - 重构时确认符号的完整引用链 - 理解复杂类型的定义来源 - 在大型代码库中导航 - 确认接口实现是否完整

国内可用性：✅ 直连可用。LSP 与本地语言服务器通信，无外部网络依赖。 **推荐优先级**：★★★★☆（大型项目推荐）

NotebookEdit（Jupyter Notebook）

一句话总结：读取和编辑 `.ipynb` Jupyter Notebook 文件。

详细描述：NotebookEdit 让 Claude Code 可以像操作普通代码文件一样操作 Jupyter Notebook。它支持读取 cell 内容、修改代码/文本 cell、添加或删除 cell。对于数据科学和机器学习工作流，NotebookEdit 让 Claude 能够直接参与 notebook 的迭代开发。

NotebookEdit 将 `.ipynb` 的 JSON 结构抽象为 cell 级别的操作，Claude 无需理解底层 JSON 即可修改 notebook。这对于将探索性 notebook 转换为生产代码的工作流非常有价值。

使用示例：

```
{
  "file_path": "analysis.ipynb",
  "cell_index": 3,
  "new_source": "import pandas as pd\ndf = pd.read_csv('data.csv')"
}
```

最佳实践场景： - 重构数据分析 notebook 中的重复代码 - 将 notebook 转换为模块化 Python 脚本 - 批量更新 notebook 中的参数或路径 - 清理和优化 notebook 结构

国内可用性：✅ 直连可用。 **推荐优先级**：★★★☆☆（数据科学场景）

Skill（执行 Skill）

一句话总结：按需加载并执行 `.claude/skills/` 或 `~/.claude/skills/` 中定义的技能。

详细描述：Skill 工具本身不实现具体功能——它是一个「加载器」。当 Claude 判断当前任务匹配某个 Skill 的描述时，调用 Skill 工具加载该 Skill 的完整指令（SKILL.md 内容）到上下文，然后按指令执行。Skill 的渐进式披露机制（先加载描述，匹配后再加载全文）确保了上下文窗口的高效利用。

一个项目可以包含数十个 Skills，但启动时只有它们的 name 和 description（约 100 tokens/个）被加载。只有当 Claude 判断任务与某个 Skill 相关时，才会加载该 Skill 的完整内容（可能数千 tokens）。这种设计让大型技能库成为可能。

使用示例：

```
{
  "name": "deploy"
}
```

最佳实践场景： - 触发团队共享的标准化工作流 - 加载领域特定的代码审查清单 - 执行复杂的、多步骤的部署或发布流程 - 按需加载编码规范或架构指南

国内可用性：✅ 直连可用。 **推荐优先级**：★★★★☆（团队协作推荐）

ListMcpResourcesTool / ReadMcpResourceTool（MCP 资源访问）

一句话总结：列出和读取已配置 MCP Server 暴露的资源（Resources）。

详细描述：这两个工具是 MCP 层与内置工具层的桥梁。MCP Server 不仅可以暴露「工具」（Tools），还可以暴露「资源」（Resources）——只读的数据源，如数据库 Schema、API 文档、配置文件等。ListMcpResourcesTool 发现可用资源，ReadMcpResourceTool 读取具体资源内容。与 Tools 不同，Resources 是只读的，更适合作为上下文注入而非操作执行。

Resources 的 URI 格式由各个 MCP Server 定义，通常遵循 `protocol://path` 的形式。例如，PostgreSQL Server 可能暴露 `schema://tablename` 资源，GitHub Server 可能暴露 `repo://owner/name` 资源。

使用示例：

```
// ListMcpResourcesTool
{
  "server": "postgres"
}

// ReadMcpResourceTool
{
  "server": "postgres",
```

```
  "uri": "schema://users"
}
```

最佳实践场景： - 查询数据库 Schema 以生成类型定义 - 读取 API 规范文档辅助代码生成 - 获取知识库资源作为回答依据 - 在操作数据库前确认表结构

国内可用性：✅ 直连可用（依赖已配置的 MCP Server）。 **推荐优先级**：★★★☆☆（使用 MCP 时配套）

BashOutput / KillShell（Shell 辅助）

一句话总结：BashOutput 获取长时间运行命令的输出；KillShell 终止后台 Shell 进程。

详细描述：BashOutput 用于获取先前启动的异步 Bash 命令的输出流。KillShell 则用于强制终止挂起或失控的 Shell 进程。这两个工具是 Bash 的补充——当 Bash 命令以异步模式运行时，BashOutput 可以轮询结果，KillShell 提供紧急停止能力。

在 Claude Code 的交互式使用中，你很少需要直接调用这两个工具——Claude 会在需要时自动使用它们。例如，当 Bash 命令超时或用户按下 Ctrl+C 时，Claude 可能调用 KillShell 清理进程。

使用示例：

```
// KillShell
{
  "pid": 12345
}
```

最佳实践场景： - 终止卡死的测试进程 - 清理后台启动的开发服务器 - 中断失控的构建过程

国内可用性：✅ 直连可用。 **推荐优先级**：★★☆☆☆（应急使用）

AskUserQuestion（用户交互）

一句话总结：向用户展示问题并等待文本回复。

详细描述：当 Claude 需要人类判断或补充信息时，调用 AskUserQuestion 在终端显示问题并阻塞等待输入。这是 Claude Code 「人在回路」（Human-in-the-loop）设计的关键组成部分，确保关键决策由人类把关。

AskUserQuestion 支持单选和多选模式，可以呈现选项供用户选择。在自动模式（auto mode）下，Claude 会尽量避免使用此工具；在手动模式下，会更频繁地询问确认。

使用示例：

```
{
  "question": "The tests failed after the refactor. Should I revert the changes or attempt to fix the tests?"
}
```

最佳实践场景： - 需要人类决策的歧义情况 - 多个等效方案的选择 - 确认高风险操作（如删除数据） - 补充缺失的必要信息

国内可用性：✅ 直连可用。 **推荐优先级**：★★★☆☆（交互式场景）

TodoWrite（任务列表）

一句话总结：创建或更新结构化任务列表，帮助跟踪多步骤进度。

详细描述：TodoWrite 让 Claude 显式管理任务状态。它会在上下文中维护一个 todo 列表，标记每个任务的 pending / in_progress / completed 状态。虽然这个功能对模型推理本身没有强制约束，但它显著提升了 Claude 在复杂多步骤任务中的条理性和可观测性。

TodoWrite 的输出对用户可见，你可以在对话中看到 Claude 的任务进度。这在长时间运行的复杂任务中特别有用——你可以随时了解 Claude 进行到哪一步。

使用示例：

```
{
  "todos": [
    {"id": "1", "content": "Read authentication module", "status": "completed", "priority": "high"},
    {"id": "2", "content": "Refactor login flow", "status": "in_progress", "priority": "high"}
  ]
}
```

最佳实践场景： - 复杂多步骤任务的状态跟踪 - 长时间运行的重构或迁移工作 - 需要人类了解当前进度的后台任务

国内可用性：✅ 直连可用。 **推荐优先级**：★★★☆☆（复杂任务推荐）

2.7 内置工具权限管理与安全配置

权限白名单/黑名单配置

Claude Code 的权限系统通过 `settings.json` 中的 `permissions` 字段配置，支持三级控制：

```
{
  "permissions": {
    "allow": [
      "Read",
      "Glob",
      "Grep",
      "Bash(npm run:*)",
      "Bash(git:*)",
      "Bash(make:*)",
      "Edit(src/**)",
      "Write(src/**)"
    ],
    "deny": [
      "Read(.env*)",
      "Read(secrets/**)",
      "Bash(rm -rf:*)",
      "Bash(sudo:*)",
      "Edit(package-lock.json)"
    ],
    "ask": [
      "WebFetch",
      "Bash(curl:*)",
      "Bash(docker:*)"
    ],
    "defaultMode": "acceptEdits"
  }
}
```

权限规则语法：

语法	含义	示例
`ToolName`	允许/拒绝整个工具	`"Read"`
`ToolName(pattern)`	带参数模式匹配	`"Bash(git:*)"`
`ToolName(path)`	带路径限制	`"Edit(src/**)"`
`mcp__server__*`	MCP 工具通配	`"mcp__github__*"`

`defaultMode` 控制默认权限行为： - `default`：每次操作都询问（最保守） - `acceptEdits`：自动接受 Edit/Write，其他询问（平衡） - `auto`：尽可能自动执行（最大自动化，需配合 Hooks 安全检查）

不同工具的风险等级分类表

风险等级	工具	潜在风险	建议策略
🔴 高危	Bash	执行任意系统命令（`rm -rf`、`sudo`）	默认 ask，限制命令模式
🔴 高危	Write	覆盖关键文件、写入恶意代码	限制路径范围，禁止写入 `.git/`
🟡 中危	Edit	修改业务逻辑引入 Bug	允许但配合 PostToolUse hook 自动测试
🟡 中危	WebFetch	访问恶意网站、数据外泄	限制 URL 白名单或默认 ask
🟡 中危	Agent（子代理）	子代理可能执行受限外操作	为子代理单独配置权限
🟢 低危	Read / Glob / Grep	信息泄露（仅读取）	通常 allow，但禁止读取 secrets
🟢 低危	NotebookEdit	修改 notebook 内容	与 Edit 同等管理
🟢 低危	LSP	读取代码诊断信息	通常 allow

项目级 vs 用户级权限最佳实践

层级	配置文件	用途	是否入 Git
用户级	`~/.claude/settings.json`	个人全局偏好（模型、主题）	否
项目级	`.claude/settings.json`	团队共享的权限规则	是
本地级	`.claude/settings.local.json`	个人 API Key、本地路径	否（加入 .gitignore）
企业级	`/etc/.../managed-settings.json`	组织强制策略	管理员部署

最佳实践组合：

1. 在项目级 `settings.json` 中定义 `allow` 和 `deny` 规则，确保团队成员使用一致的权限边界；
2. 在本地级 `settings.local.json` 中存放敏感配置（API Key、个人 MCP Token）；
3. 将 `.claude/settings.local.json` 加入 `.gitignore`，防止密钥泄露；
4. 对 Bash 工具使用命令模式前缀限制（如 `Bash(git:*)` 只允许 Git 命令），而非完全开放或完全禁止；

5. 对于开源项目或涉及敏感数据的项目，使用 `deny` 明确禁止读取 `.env*` 和 `secrets/`；
6. 企业环境使用 `managed-settings.json` 统一部署策略，禁止用户覆盖；
7. 定期审查项目级的 `.mcp.json` 和 `settings.json`，移除不再使用的 Server 和过度宽松的权限规则；
8. 为每个仓库创建独立的 Git worktree 进行高风险实验，避免污染主分支。

CVE-2025-59536 安全提醒：2025 年 7 月披露的漏洞表明，恶意仓库可通过 `.claude/settings.json` 中的 Hooks 配置执行任意代码。已于 2025 年 8 月修复。当前版本的 Claude Code 会在加载项目级 Hooks 前提示用户确认，但建议： - 克隆不可信仓库时先检查 `.claude/settings.json` 内容； - 对未知来源的项目使用 `--bare` 模式启动（`claude -p --bare`）； - 企业环境启用 `allowManagedHooksOnly` 限制仅管理员审批的 Hooks。

工具组合最佳实践

单个工具的能力有限，但组合使用可以构建强大的工作流。以下是经过验证的工具组合模式：

模式 1：探索-分析-修改（Explore-Analyze-Modify）

```
Glob("src/**/*.ts") → Read(关键文件) → Grep(模式定位) → Edit(修改)
```

适用场景：代码重构、批量更新模式。先用 Glob 了解文件分布，Read 理解关键文件结构，Grep 定位所有需要修改的实例，最后 Edit 执行替换。

模式 2：验证驱动开发（Validation-Driven）

```
Bash(测试) → Read(失败文件) → Edit(修复) → Bash(验证)
```

适用场景：Bug 修复、测试驱动重构。循环执行测试→修复→验证，直到测试通过。

模式 3：规划-执行-验证（Plan-Execute-Verify）

```
EnterPlanMode → 分析 → ExitPlanMode → 执行 Edit/Write → Bash(验证)
```

适用场景：大型重构、架构调整。先制定详细计划，经审阅后执行，最后验证。

模式 4：多代理并行（Multi-Agent Parallel）

```
Agent(子代理A: 重构模块1) + Agent(子代理B: 重构模块2) → 整合结果
```

适用场景：独立模块的并行重构、多文件独立修改。

模式 5：MCP 增强工作流（MCP-Enhanced）

```
ListMcpResourcesTool → ReadMcpResourceTool(获取Schema) → Edit(生成代码) → MCP工具(验证)
```

适用场景：数据库操作、API 集成。先通过 MCP Resources 获取结构信息，再生成对应代码。

CHAPTER THREE

第三章 MCP 服务——让 Claude Code 连接万物

3.1 MCP 协议简述

3.2 MCP 安装与管理命令

3.3 官方 MCP Server 详解

3.4 社区 MCP Server 分类推荐

3.5 国内开发者 MCP 推荐清单

3.1 MCP 协议简述

Model Context Protocol（MCP）是 Anthropic 于 2024 年 11 月推出的开放标准，旨在为 AI 助手提供连接外部系统的通用接口。2025 年 3 月 OpenAI 正式采用；2025 年 12 月 9 日，Linux 基金会宣布成立 Agentic AI Foundation（AAIF），Anthropic 将 MCP 作为三个创始项目之一捐赠（另两个为 OpenAI 的 AGENTS.md 和 Block 的 goose），MCP 由此成为跨平台的行业标准。

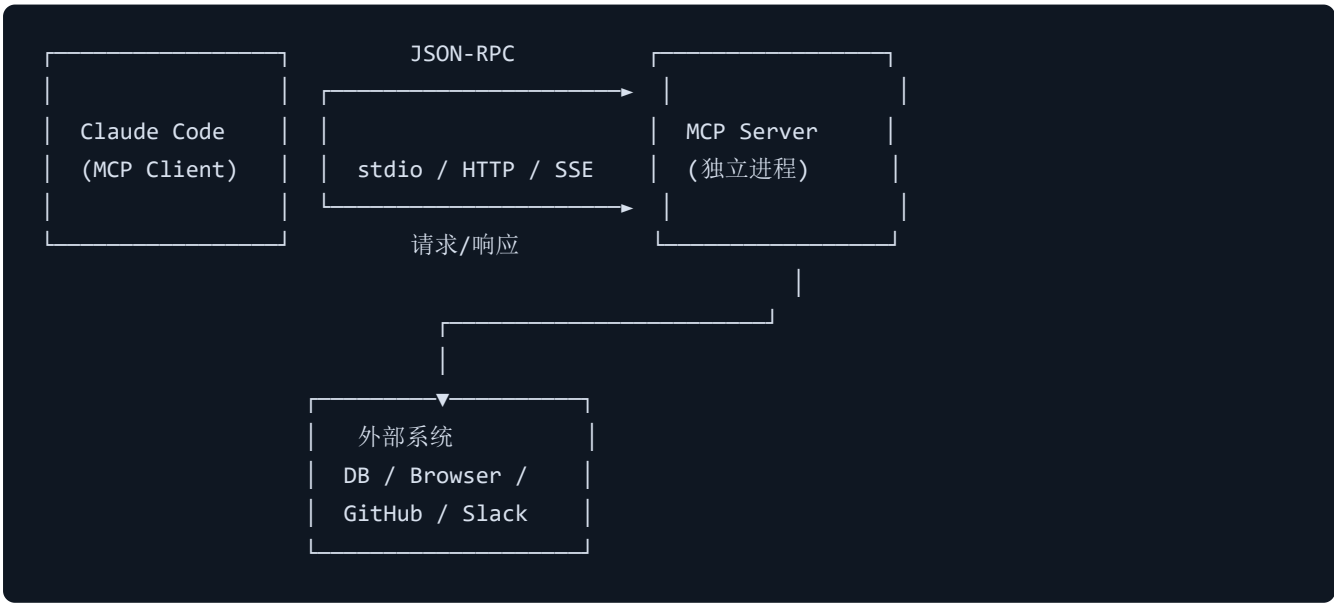
2026 年 7 月 28 日规范大改版：MCP 发布了迄今最大的一次规范更新（此前版本序列为 2024-11-05 → 2025-03-26 → 2025-06-18 → 2025-11-25），核心变化包括：

- **无状态核心**：移除 initialize 握手和 Mcp-Session-Id，协议核心改为无状态设计；
- **能力发现**：新增 `server/discover`，客户端可动态发现服务器能力；
- **传输优化**：HTTP 头路由；授权机制加固（DCR 弃用转向 CIMD）；
- **一等扩展**：Tasks、MCP Apps、EMA 成为正式扩展。

这是 MCP 首次有意的破坏性变更，新版 SDK 已同步支持，绝大多数现有 Server 的接入代码无需修改。生态规模方面，截至 2026 年上半年公开 MCP Server 已超 1.7 万个，官方发现入口为 **MCP Registry**（registry.modelcontextprotocol.io）；早期参考服务器仓库 `modelcontextprotocol/servers` 已归档为 `servers-archived`（标注 "no longer maintained"）。

MCP 的设计目标非常明确：让 AI Agent 能够像人类使用软件一样，通过标准化接口操作外部工具。在没有 MCP 之前，每个 AI 应用都需要为每个外部服务编写专用集成；有了 MCP 后，外部服务只需实现一次 MCP Server，所有兼容客户端都可以使用。

架构示意图



核心概念

概念	说明	类比
Client	MCP 客户端，如 Claude Code、Cursor、VS Code	浏览器
Server	暴露工具/资源/提示的独立进程	网站后端
Tools	可被客户端调用的操作（有副作用）	POST API
Resources	只读的数据源	GET API
Prompts	预定义的提示模板	快捷指令

MCP 的四大优势

1. **跨客户端兼容**：一个 MCP Server 开发完成后，可在 Claude Code、Cursor、Windsurf、VS Code Copilot、Cline 等所有支持 MCP 的客户端中使用。开发者无需为每个 AI 编辑器重复造轮子。
2. **进程隔离安全**：每个 Server 是独立进程，拥有独立的文件系统权限和凭证作用域。即使某个 Server 被攻破，攻击者也无法读取其他 Server 的内存或凭证。这种隔离模型类似操作系统的进程隔离。
3. **标准化接口**：统一基于 JSON-RPC 的通信协议，所有 Server 遵循相同的工具发现、调用和错误处理模式。客户端可以动态发现 Server 的能力，无需硬编码。
4. **动态发现**：Client 在连接时自动获取 Server 的工具列表、参数类型和描述。这意味着你可以随时升级 MCP Server 添加新工具，客户端无需任何配置变更即可使用新能力。

Transport 类型对比

MCP 支持三种传输方式：

Transport	适用场景	优势	劣势
stdio	本地运行的 Server	简单、无需网络、安全	仅限本机
HTTP	远程 API 服务	可访问云端服务、支持 OAuth	需要网络、延迟较高
SSE	服务器推送（已弃用但可用）	实时流式响应	复杂度高，HTTP 替代

Claude Code 2.1.x 推荐使用 HTTP transport 连接远程服务，stdio 连接本地服务。旧版 SSE transport 仍可用但不再推荐。

3.2 MCP 安装与管理命令

安装方法 3 种

方法 1：npx 直接运行（最快，无需持久化）

```
# 临时使用，不保存配置
npx -y @modelcontextprotocol/server-filesystem /path/to/allowed/dir
```

这种方式适合快速测试，不会在配置中留下痕迹。重启 Claude Code 后失效。

方法 2：claude mcp add 持久化（推荐）

```
# 添加项目级 MCP Server
claude mcp add --scope project --transport stdio filesystem \
  -- npx -y @modelcontextprotocol/server-filesystem /path/to/dir

# 添加用户级 MCP Server
claude mcp add --scope user --transport http github https://api.githubcopilot.com/mcp/
```

`claude mcp add` 提供交互式向导（不带参数直接运行），自动处理配置格式和权限审批。

方法 3：手动编辑配置文件

```
# 编辑项目级配置
vim .mcp.json

# 或编辑用户级配置
vim ~/.claude.json
```

适合批量配置或版本控制配置模板。

管理命令表

命令	功能	示例
`claude mcp list`	列出所有已配置 Server	`claude mcp list`
`claude mcp add`	添加 Server（交互式向导）	`claude mcp add`
`claude mcp remove <name>`	移除指定 Server	`claude mcp remove github`
`claude mcp serve`	将当前项目作为 MCP Server 运行	`claude mcp serve`
`claude mcp test <name>`	测试 Server 连接	`claude mcp test postgres`
`claude mcp get <name>`	查看指定 Server 详情	`claude mcp get github`
`claude mcp reset-project-choices`	重置项目级审批	`claude mcp reset-project-choices`
`claude mcp add-from-claude-desktop`	从 Desktop 导入配置	`claude mcp add-from-claude-desktop`
`claude mcp add-json <name> <json>`	从 JSON 添加	`claude mcp add-json weather '{"type":"http","url":"..."}'`

在 Claude Code REPL 中，也可以使用 `/mcp` 打开交互式 MCP 管理界面。

`.mcp.json` 配置文件完整示例

```
{
  "mcpServers": {
    "filesystem": {
      "type": "stdio",
      "command": "npx",
      "args": [
        "-y",
        "@modelcontextprotocol/server-filesystem",
        "/home/user/project"
      ]
    },
    "github": {
      "type": "http",
      "url": "https://api.github.com",
      "headers": {
        "Authorization": "Bearer ${GITHUB_TOKEN}",
        "Accept": "application/vnd.github.v3+json"
```

```
      }
    },
    "postgres": {
      "type": "stdio",
      "command": "npx",
      "args": ["-y", "@anthropic-ai/mcp-server-postgres"],
      "env": {
        "DATABASE_URL": "postgresql://user:pass@localhost:5432/db"
      }
    },
    "memory": {
      "type": "stdio",
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-memory"]
    },
    "sequential-thinking": {
      "type": "stdio",
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-sequential-thinking"]
    }
  }
}
```

环境变量扩展语法：`.mcp.json` 支持 `${VAR}` 和 `${VAR:-default}` 两种形式，便于在不同环境中复用配置。例如，`${DATABASE_URL:-postgresql://localhost:5432/postgres}` 会在环境变量未设置时使用默认值。

旧版 settings.json 内嵌方式 vs 当前 .mcp.json 方式

特性	旧版 settings.json	当前 .mcp.json
配置位置	`settings.json` 内 `mcpServers` 字段	独立 `.mcp.json` 文件
版本控制	与权限配置混杂	独立文件，更清晰
团队共享	需要分离敏感信息	天然分离，可配合 `${VAR}`
传输类型	仅 stdio	stdio / http / sse
OAuth 支持	不支持	支持交互式 OAuth 流
推荐度	已弃用	官方推荐

> **迁移建议**：如果你仍在使用旧版 `settings.json` 内嵌 `mcpServers`，请尽快迁移到 `.mcp.json`。Claude Code 2.1.x 仍向后兼容旧格式，但新功能（如 HTTP transport、OAuth）仅在 `.mcp.json` 中可用。

3.3 官方 MCP Server 详解

以下 MCP Server 来自 Anthropic 官方维护的 `@modelcontextprotocol` npm 组织（原 `modelcontextprotocol/servers` 仓库已于 2026 年归档为 `servers-archived`，官方发现入口现为 MCP Registry），可靠性最高。

Filesystem（安全文件访问）

一句话总结：让 Claude 在指定目录范围内安全地读写文件。

功能说明：Filesystem Server 为 Claude 提供受限的文件系统访问能力。启动时必须指定一个或多个允许访问的目录，Claude 只能在这些目录内操作。支持读取、写入、编辑、删除、列出目录、搜索文件等操作。这是防止 Claude 误删系统文件的关键安全层。

安装方法：

```
claude mcp add --scope project --transport stdio filesystem \
  -- npx -y @modelcontextprotocol/server-filesystem /allowed/path
```

最佳实践场景： - 在 monorepo 中跨子项目操作文件 - 访问构建产物或日志目录 - 限制 Claude 只能修改 `src/` 和 `docs/`，禁止触碰 `.env` 和 `.git/` - 为不同项目配置不同的允许目录

国内可用性：✅ 直连可用，纯本地操作。 **推荐优先级**：★★★★★（必装）

Fetch（网页内容获取）

一句话总结：通过 MCP 层获取网页内容，与内置 WebFetch 互补但支持更灵活的配置。

功能说明：Fetch Server 提供标准化的 HTTP 请求能力，支持 GET/POST 等方法。与内置 WebFetch 相比，Fetch Server 更适合需要自定义 Header、Cookie 或代理配置的场景，且可以在多个 MCP 客户端间复用。

安装方法：

```
claude mcp add --scope user --transport stdio fetch \
  -- npx -y @modelcontextprotocol/server-fetch
```

最佳实践场景： - 需要自定义 User-Agent 或 Authentication Header 的 API 调用 - 访问需要 Cookie 的受保护文档 - 在内网环境中代理特定域名请求 - 复用同一套 HTTP 配置到多个 MCP 客户端

国内可用性：⚠️ 取决于目标 URL。Server 本身本地运行，但访问境外资源需要系统代理。 **推荐优先级**：★★★☆☆（按需）

Memory（知识图谱记忆）

一句话总结：为 Claude 提供跨会话的持久化记忆存储。

功能说明：Memory Server 维护一个本地知识图谱，Claude 可以在其中存储和检索信息。与 Claude Code 内置的临时上下文不同，Memory 的数据持久保存在本地文件中，跨会话可用。适合存储项目约定、个人偏好、常用代码片段等。

Memory 的设计类似一个本地向量数据库，但针对 Claude 的交互模式优化。Claude 可以「记住」你的偏好（如技术栈选择、命名规范），在后续会话中自动应用。

安装方法：

```
claude mcp add --scope user --transport stdio memory \
  -- npx -y @modelcontextprotocol/server-memory
```

最佳实践场景： - 存储团队编码规范，每次会话自动应用 - 记住个人开发环境偏好（如「我用 pnpm 不用 npm」） - 积累项目特定知识（架构决策、技术债记录） - 维护常用代码模板和 Snippet

国内可用性：✅ 直连可用，纯本地存储。 **推荐优先级**：★★★★☆（推荐安装）

Sequential Thinking（序列思考）

一句话总结：引导 Claude 将复杂问题分解为多步逻辑推理。

功能说明：Sequential Thinking Server 不提供外部数据访问，而是增强 Claude 的推理过程。它强制 Claude 在解决问题时显式地逐步思考，每一步都可以 revising 前面的假设。对于数学问题、架构设计、调试分析等需要严密推理的场景效果显著。

这个 Server 特别适合那些 Claude 容易「跳步」或过早下结论的场景。通过显式要求逐步推理，可以显著提高复杂技术决策的准确性。

安装方法：

```
claude mcp add --scope user --transport stdio sequential-thinking \
  -- npx -y @modelcontextprotocol/server-sequential-thinking
```

最佳实践场景： - 复杂 Bug 的根因分析 - 算法设计与正确性验证 - 多因素技术方案对比评估 - 安全漏洞的影响面分析

国内可用性：✅ 直连可用。 **推荐优先级**：★★★☆☆（复杂推理场景）

Git（Git 仓库操作）

一句话总结：让 Claude 通过 MCP 层执行高级 Git 操作。

功能说明：Git Server 提供更细粒度的 Git 仓库操作能力，包括读取提交历史、分支管理、差异对比、blame 分析等。与内置 Bash 执行 `git` 命令相比，Git Server 的输出结构更规整，更适合 Claude 解析和处理。

Git Server 将 Git 的文本输出转换为结构化数据，Claude 可以精确获取提交作者、时间、变更文件列表等信息，而无需解析 `git log` 的文本格式。

安装方法：

```
# 从源码运行（官方暂未发布独立 npm 包）
git clone https://github.com/modelcontextprotocol/servers-archived.git
cd servers/src/git
npm install && npm run build
claude mcp add --scope project --transport stdio git \
  -- node /full/path/to/servers/src/git/build/index.js
```

最佳实践场景： - 自动化的提交历史分析 - 批量分支管理 - 生成 Changelog 或发布说明 - 代码作者和变更溯源

国内可用性： ✅ 直连可用（纯本地 Git 操作）。 **推荐优先级**：★★★☆☆（Git 重度用户）

Time（时间时区转换）

一句话总结：提供精确的当前时间和时区转换能力。

功能说明：Time Server 让 Claude 获得准确的系统时间信息，支持多时区转换和格式化。虽然看似简单，但它是调度、日志分析和国际化功能的基础依赖。很多代码生成场景需要时间戳或日期格式化，Time Server 确保这些信息的准确性。

安装方法：

```
claude mcp add --scope user --transport stdio time \
  -- npx -y @modelcontextprotocol/server-time
```

最佳实践场景： - 生成带时间戳的日志或报告 - 跨时区会议时间计算 - CI/CD 流水线时间分析 - 定时任务的时区处理

国内可用性： ✅ 直连可用。 **推荐优先级**：★★☆☆☆（基础工具）

GitHub（GitHub 集成）

一句话总结：让 Claude 直接操作 GitHub 仓库、Issues 和 PR。

功能说明：GitHub Server 提供完整的 GitHub API 封装，支持读取仓库信息、创建/更新 Issues、审查 PR、管理分支等。与内置的 `/review` 斜杠命令相比，GitHub Server 的能力更底层、更灵活，可以构建完

全自动化的 GitHub 工作流。

GitHub Server 需要 `GITHUB_TOKEN` 环境变量，Token 需具备 `repo` 和 `read:org` 权限。对于私有仓库操作，需要额外的权限配置。

安装方法:

```
# 从源码运行
git clone https://github.com/modelcontextprotocol/servers-archived.git
cd servers/src/github
npm install && npm run build
claude mcp add --scope user --transport stdio github \
  -- node /full/path/to/servers/src/github/build/index.js
```

配置环境变量：`GITHUB_TOKEN`（需要 `repo` 和 `read:org` 权限）。

最佳实践场景： - 自动化 PR 审查和合并 - 批量 Issues 分类和标签管理 - 生成基于 GitHub 数据的报告 - 发布版本和创建 Release

国内可用性：⚠️ 需要代理访问 GitHub API。GitHub 在国内直连不稳定，建议配合代理使用。 **推荐优先级**：★★★★☆（GitHub 重度用户）

3.4 社区 MCP Server 分类推荐

社区 MCP Server 生态在 2026 年已超过 2000 个。以下按功能分组推荐，每个表格包含名称、功能、安装命令、国内可用性、优先级。

搜索类（至少 4 个）

名称	功能	安装命令	国内可用性	优先级
Brave Search	隐私优先的网络搜索	`npx -y @modelcontextprotocol/server-brave-search`	⚠️ 需 API Key	★★★★☆
Tavily Search	专为 AI 优化的搜索 API	`npx -y @modelcontextprotocol/server-tavily`	⚠️ 需 API Key	★★★★☆
Exa Search	语义搜索引擎	`npx -y @modelcontextprotocol/server-exa`	⚠️ 需 API Key	★★★☆☆
SearXNG	自托管元搜索聚合器	需自建 SearXNG 实例	✅ 自建后直连	★★★☆☆

说明：搜索类 Server 的国内可用性主要取决于搜索引擎本身是否可达。Brave 和 Tavily 的 API 在国内可直连，但需要注册获取 API Key。SearXNG 部署在境内服务器后可完全直连，是控制数据隐私的最佳选择。

数据库类（至少 7 个）

名称	功能	安装命令	国内可用性	优先级
PostgreSQL	查询 PostgreSQL 数据库	`npx -y @anthropic-ai/mcp-server-postgres`	✅ 本地直连	★★★★★
SQLite	查询 SQLite 数据库	`npx -y @modelcontextprotocol/server-sqlite`	✅ 本地直连	★★★★☆
MySQL	查询 MySQL 数据库	`npx -y @modelcontextprotocol/server-mysql`	✅ 本地直连	★★★★☆
MongoDB	查询 MongoDB 集合	`npx -y @modelcontextprotocol/server-mongodb`	✅ 本地直连	★★★☆☆
Redis	操作 Redis 键值	`npx -y @modelcontextprotocol/server-redis`	✅ 本地直连	★★★☆☆
Qdrant	向量数据库查询	`npx -y @modelcontextprotocol/server-qdrant`	✅ 本地直连	★★☆☆☆
Chroma	向量数据库查询	`npx -y @modelcontextprotocol/server-chroma`	✅ 本地直连	★★☆☆☆

说明：数据库类 Server 连接本地或内网数据库时完全直连可用。若连接云数据库（如 AWS RDS、阿里云 RDS），网络可达性取决于云服务商的网络策略。建议在 `.mcp.json` 中使用 `${DATABASE_URL}` 形式配置连接字符串，便于不同环境复用。

浏览器类（至少 3 个）

名称	功能	安装命令	国内可用性	优先级
Puppeteer	无头 Chrome 浏览器自动化	`npx -y @modelcontextprotocol/server-puppeteer`	✅ 本地运行	★★★★☆
Playwright	多浏览器自动化	`npx -y @modelcontextprotocol/server-playwright`	✅ 本地运行	★★★★☆
LSD	轻量级网页截图与解析	`npx -y @modelcontextprotocol/server-lsd`	✅ 本地运行	★★☆☆☆

说明：浏览器类 Server 在本地运行，本身无网络限制。但抓取境外网站时需要系统代理。Puppeteer 首次安装会下载 Chromium，在国内建议配置 `PUPPETEER_DOWNLOAD_HOST` 指向国内镜像。Playwright 同理，可使用 `PLAYWRIGHT_DOWNLOAD_HOST` 环境变量。

知识管理类（至少 4 个）

名称	功能	安装命令	国内可用性	优先级
Obsidian	访问 Obsidian 知识库	`npx -y @modelcontextprotocol/server-obsidian`	✅ 本地文件	★★★☆☆
Notion	读写 Notion 页面	`npx -y @modelcontextprotocol/server-notion`	⚠️ 需 API Key，Notion API 国内可直连	★★★☆☆
Context7	技术文档语义检索	`npx -y @modelcontextprotocol/server-context7`	⚠️ 需 API Key	★★☆☆☆
Markdownify	网页转 Markdown 并存储	`npx -y @modelcontextprotocol/server-markdownify`	✅ 本地运行	★★☆☆☆

DevOps 类（至少 6 个）

名称	功能	安装命令	国内可用性	优先级
GitHub	GitHub API 集成	见 3.3 节	⚠️ 需代理	★★★★☆
GitLab	GitLab API 集成	`npx -y @modelcontextprotocol/server-gitlab`	✅ 内网 GitLab 直连	★★★☆☆
Docker	容器管理	`npx -y @modelcontextprotocol/server-docker`	✅ 本地直连	★★★☆☆
Kubernetes	K8s 集群操作	`npx -y @modelcontextprotocol/server-kubernetes`	✅ 内网集群直连	★★☆☆☆
AWS	AWS 资源管理	`npx -y @modelcontextprotocol/server-aws`	⚠️ 需 API Key，AWS 国内区域可直连	★★☆☆☆
Vercel	Vercel 部署管理	`npx -y @modelcontextprotocol/server-vercel`	⚠️ 需 API Key，国内直连稳定	★★☆☆☆

通信协作类（至少 4 个）

名称	功能	安装命令	国内可用性	优先级
Slack	发送消息/查询频道	`npx -y @modelcontextprotocol/server-slack`	⚠️ 需 API Key，Slack API 国内可直连	★★★☆☆
Discord	发送消息/查询服务器	`npx -y @modelcontextprotocol/server-discord`	⚠️ 需 API Key	★★☆☆☆
Gmail	读取/发送邮件	`npx -y @modelcontextprotocol/server-gmail`	🔴 Google 服务国内不可用	★★☆☆☆
Lark / 飞书	飞书消息/文档	`npx -y @modelcontextprotocol/server-lark`	✅ 国内直连	★★★☆☆

说明：飞书/Lark Server 对国内开发者非常实用，可让 Claude 直接读取飞书文档、发送群消息，构建内网协作工作流。相比 Slack/Discord，飞书的国内连通性是巨大优势。

3.5 国内开发者 MCP 推荐清单

以下按「国内直连可用优先」排序，列出 Top 20 必装 MCP，包含安装命令。

排名	名称	功能	国内可用性	优先级	安装命令
1	Filesystem	安全文件访问	✅	★★★★★	`npx -y @modelcontextprotocol/server-filesystem /path`
2	PostgreSQL	数据库查询	✅	★★★★★	`npx -y @anthropic-ai/mcp-server-postgres`
3	SQLite	轻量数据库	✅	★★★★☆	`npx -y @modelcontextprotocol/server-sqlite`
4	Memory	持久记忆	✅	★★★★☆	`npx -y @modelcontextprotocol/server-memory`
5	Puppeteer	浏览器自动化	✅	★★★★☆	`npx -y @modelcontextprotocol/server-puppeteer`
6	Playwright	多浏览器支持	✅	★★★★☆	`npx -y @modelcontextprotocol/server-playwright`
7	MySQL	数据库查询	✅	★★★★☆	`npx -y @modelcontextprotocol/server-mysql`
8	Redis	缓存操作	✅	★★★☆☆	`npx -y @modelcontextprotocol/server-redis`
9	Lark/飞书	国内协作	✅	★★★☆☆	`npx -y @modelcontextprotocol/server-lark`
10	MongoDB	NoSQL 查询	✅	★★★☆☆	`npx -y @modelcontextprotocol/server-mongodb`
11	GitLab	内网代码托管	✅	★★★☆☆	`npx -y @modelcontextprotocol/server-gitlab`
12	Docker	容器管理	✅	★★★☆☆	`npx -y @modelcontextprotocol/server-docker`
13	Time	时间/时区	✅	★★☆☆☆	`npx -y @modelcontextprotocol/server-time`
14	Sequential Thinking	逻辑推理	✅	★★☆☆☆	`npx -y @modelcontextprotocol/server-sequential-thinking`

排名	名称	功能	国内可用性	优先级	安装命令
15	Brave Search	隐私搜索	⚠️	★★★★☆	`npx -y @modelcontextprotocol/server-brave-search`
16	Tavily Search	AI 搜索	⚠️	★★★☆☆	`npx -y @modelcontextprotocol/server-tavily`
17	Notion	知识库	⚠️	★★★☆☆	`npx -y @modelcontextprotocol/server-notion`
18	GitHub	代码托管	⚠️	★★★☆☆	见 3.3 节源码安装
19	Slack	团队消息	⚠️	★★☆☆☆	`npx -y @modelcontextprotocol/server-slack`
20	Fetch	自定义 HTTP	⚠️	★★☆☆☆	`npx -y @modelcontextprotocol/server-fetch`
21	高德地图（官方）	地理编码/路径规划/POI/天气	✅	★★★☆☆	`npx -y @amap/amap-maps-mcp-server` 或 SSE `mcp.amap.com`
22	支付宝（官方）	支付链路（国内首个支付 MCP）	✅	★★☆☆☆	阿里云百炼平台预置接入
23	腾讯位置服务（官方）	地图/位置能力	✅	★★☆☆☆	腾讯云开发平台接入

说明：排名 1-14 为纯本地或国内直连可用，无需任何网络代理；排名 15-20 需要 API Key 或代理，但功能强大，建议有条件时配置。对于国内开发者，优先确保 1-14 的安装和配置，再根据实际需求扩展 15-20。排名 21-23 是 2025 年以来国内厂商推出的官方 MCP Server，国内直连、生态合规，是国内生态的重要补充（2026年8月核实）。

CHAPTER FOUR

第四章 Skills 技能系统与 CLAUDE.md 实战

4.1 CLAUDE.md 三层结构

4.2 Skills 机制详解

4.3 高质量 CLAUDE.md 编写指南

4.4 CLAUDE.md 实战模板库

4.1 CLAUDE.md 三层结构

CLAUDE.md 是放置于仓库根目录的特殊文件，作为项目级系统提示持续注入到 Claude 的上下文中。它的内容会影响 Claude 对项目的理解、代码风格偏好和工具使用方式。

ASCII 结构图展示三层优先级



每层的用途和典型内容

层级	位置	用途	典型内容
第一层	内置	Claude Code 的基础行为定义	工具描述、安全策略、输出格式
第二层	`~/.claude/settings.json`	用户个人偏好	默认模型、权限规则、主题
第三层	`CLAUDE.md`	项目专属上下文	技术栈、架构约定、编码规范、常见命令

关键原则：CLAUDE.md 中定义的内容会覆盖上层默认值。例如，若 Claude 内置偏好使用 4 空格缩进，但 CLAUDE.md 指定 "Use 2 spaces for indentation"，则项目内所有生成代码都会使用 2 空格。这种覆盖机制让 CLAUDE.md 成为塑造项目一致性的最强杠杆。

4.2 Skills 机制详解

Skills 定义：按需加载的可复用指令系统

Skill 是一个目录，内含 `SKILL.md` 文件（必须）以及可选的 `scripts/`、`references/`、`assets/` 子目录。SKILL.md 以 YAML frontmatter 开头，定义 metadata；后续 Markdown 内容则是给 Claude 的详细指令。Skills 的设计理念是「按需加载」——只在需要时才将完整内容注入上下文，避免启动时的上下文膨胀。

Skills 存储位置（项目级/用户级）

范围	路径	可见性	使用场景
项目级	`.claude/skills/`	当前仓库	团队共享的工作流、项目专属规范
用户级	`~/.claude/skills/`	所有项目	个人常用工具、跨项目复用能力
插件级	`~/.claude/plugins/...`	安装该插件后	第三方封装的能力包

2026 年初，Claude Code 的 Skills 加载机制进行了优化：项目级和用户级 Skills 在会话启动时自动扫描，新增或修改 Skills 无需重启 Claude Code 即可生效（热重载）。

Skills 的结构模板（YAML frontmatter + 详细指令）

```
---
name: deploy
description: Deploy the application to staging or production. Use when the user mentions deploying, releasing, or shipping.
tools: ["Bash", "Read"]
---

# Deployment Workflow

## Pre-deployment Checks
1. Run the full test suite: `npm test`
2. Check for uncommitted changes: `git status`
3. Verify you're on the correct branch

## Deploy Steps
1. Build the production bundle: `npm run build`
2. Run database migrations if needed
3. Deploy using: `./scripts/deploy.sh <environment>`
4. Verify the deployment health check

## Post-deployment
1. Monitor error rates for 15 minutes
2. Run smoke tests against the deployed environment
3. Update the deployment log
```

YAML frontmatter 中的 `description` 是最关键的字段——Claude 使用它来决定是否加载该 Skill。描述应该清晰说明 Skill 的功能和触发条件。`tools` 字段可选，用于限制该 Skill 可用的工具集。

Skills 与 CLAUDE.md 的核心区别（4 维度对比表）

维度	Skills	CLAUDE.md
加载时机	按需加载（任务匹配描述时才加载）	始终加载（会话开始时注入）
上下文成本	低（只加载匹配的 Skill）	中（始终占用上下文空间）
内容范围	特定工作流或领域知识	项目整体上下文和全局约定
复用方式	可跨项目复用（用户级 Skills）	项目专属（每个仓库独立）
触发方式	自动匹配或 `/skill-name` 手动调用	自动生效，无需触发

核心原则：CLAUDE.md 回答「这个项目是什么」；Skills 回答「这个任务怎么做」。两者互补，共同构成项目的 AI 上下文层。

Skills 生态与社区资源

Skills 的标准化催生了丰富的社区生态。以下是 2026 年主流的 Skills 资源：

资源	地址	说明
官方 Skills 仓库	github.com/anthropics/skills	Anthropic 维护的参考实现
Skills 包管理器	skills.sh	Vercel 推出的跨平台 Skills 管理工具
社区索引	skillmatic-ai/awesome-agent-skills	社区 curated 的优质 Skills 列表
跨平台兼容	agentskills.io	Agent Skills 开放标准规范

skills.sh 使用示例（Vercel 出品的 Skills 注册表，配套 `skills` CLI）：

```
# 搜索可用 Skills（无需全局安装，直接 npx 运行）
npx skills search deploy

# 安装到当前项目
npx skills add code-review
```

4.3 高质量 CLAUDE.md 编写指南

原则 1：保持简洁

❌ 啰嗦版（消耗上下文，降低信噪比）：

```
# CLAUDE.md

This is a very important project. It is built by our team over many years. We have many conventions that everyone must follow strictly. Please be ver
```

✅ 精简版（信息密度高，Claude 易于遵循）：

```
# Project Context

## Stack
- React 19 + TypeScript 5.5
- Tailwind CSS 4 + shadcn/ui
- Vite 6

## Conventions
- 2-space indentation
- Prefer `async/await` over raw Promises
- Use `src/components/ui` for shadcn components
- Run `npm run lint` after editing TypeScript files
```

原则 2：引用外部文件

❌ 将大量内容直接写入 CLAUDE.md：

```
# API 规范

GET /users 返回用户列表...
POST /users 创建用户...
（此处省略 200 行 API 文档）
```

✅ 引用外部文件，按需读取：

```
# API 规范

参见 `docs/api.md`。当需要调用后端接口时，先读取该文件确认请求格式和字段定义。
```

原则 3：包含验证指令

❌ 只描述应该做什么，不验证结果：

```
## 构建
运行 `npm run build` 生成生产包。
```

✅ 添加验证步骤，确保操作成功：

```
## 构建
1. 运行 `npm run build` 生成生产包
2. 验证 `dist/` 目录存在且包含 `index.html`
3. 若构建失败，先修复 TypeScript 错误再提交
```

原则 4：使用文件导入

❌ 重复定义已在其他地方存在的规范：

```
## 代码风格
我们使用 ESLint 规则：semi: true, singleQuote: true...
```

✅ 直接引用配置文件：

```
## 代码风格
遵循 `eslint.config.js` 和 `.prettierrc` 的定义。修改代码后必须运行 `npm run lint:fix`。
```

4.4 CLAUDE.md 实战模板库

模板 1：前端项目（React + TypeScript + Tailwind + shadcn/ui）

```
# CLAUDE.md — Frontend Project

## Tech Stack
- React 19 (Canary), TypeScript 5.5, Vite 6
- Tailwind CSS 4, shadcn/ui latest
- React Query 5, Zustand 4.5, React Router 7

## Project Structure
- `src/components/ui/` — shadcn/ui components (auto-generated, do not edit)
- `src/components/` — application components
- `src/hooks/` — custom React hooks
- `src/lib/` — utility functions
- `src/stores/` — Zustand state stores

## Conventions
- 2-space indentation, single quotes, no semicolons
- Components: PascalCase files, default export
- Hooks: `use` prefix, co-located with component or in `src/hooks/`
- Use `cn()` from `src/lib/utils` for conditional class names

## Commands
- `npm run dev` — start dev server
- `npm run build` — production build (must pass before commit)
- `npm run lint` — ESLint check
- `npm run test` — Vitest unit tests

## Rules
- Always use TypeScript strict mode
- Prefer server state in React Query, client state in Zustand
- Add shadcn components via `npx shadcn add <component>`
- Do not modify `components/ui/*` manually
```

模板 2：后端项目（Node.js + Express / Python FastAPI）

```
# CLAUDE.md — Backend API (Node.js + Express)

## Tech Stack
- Node.js 22 LTS, Express 5, TypeScript 5.5
- Prisma ORM + PostgreSQL 16
- Zod 3 for validation
```

```
- Jest 29 for testing

## Project Structure
- `src/routes/` — API route handlers
- `src/services/` — business logic
- `src/db/` — Prisma schema and client
- `src/middleware/` — Express middleware
- `tests/` — integration tests

## Conventions
- 2-space indentation
- Routes: `kebab-case.ts`, export Express Router
- Services: `camelCase.ts`, pure functions when possible
- Always validate request body with Zod before processing
- Use `src/db/client.ts` for all database access

## Commands
- `npm run dev` — tsx watch mode
- `npm run build` — tsc compile
- `npm run test` — Jest with coverage
- `npm run db:migrate` — Prisma migrate dev
- `npm run db:generate` — Prisma client generate

## Rules
- Never write raw SQL except in migrations
- Always return consistent error shape: `{ error: string, code: string }`
- Require authentication on all routes except `/health` and `/auth/*`
```

模板 3：Monorepo 项目

```
# CLAUDE.md — Monorepo

## Tech Stack
- pnpm 9 workspaces
- Turbo 2 for task orchestration
- apps/web — Next.js 15 frontend
- apps/api — Fastify backend
- packages/ui — shared UI library
- packages/shared — shared types and utilities

## Conventions
- Always use `pnpm` commands from repo root
- Add dependencies to correct workspace: `pnpm add --filter web <pkg>`
- Shared code goes in `packages/`, never duplicate across apps

## Commands
- `pnpm dev` — start all apps in dev mode (Turbo)
- `pnpm build` — build all packages and apps
- `pnpm lint` — lint entire repo
```

```
- `pnpm test` — run all test suites
- `pnpm changeset` — version bump workflow

## Rules
- Before editing `packages/ui`, check if web/api already use the component
- Keep package.json `exports` in sync with actual file structure
- Run `pnpm build` after modifying `packages/*` before testing apps
```

模板 4：自动部署流程（含 CI/CD 触发）

```
# CLAUDE.md — Deployment Pipeline

## Environments
- staging: auto-deploy on push to `develop`
- production: manual deploy via `npm run deploy:prod`

## Pre-deploy Checklist
1. `git branch` confirms correct branch
2. `npm run test` passes
3. `npm run build` passes
4. No uncommitted changes (`git status` clean)

## Deploy Steps
1. Merge PR via GitHub (squash and merge)
2. Wait for GitHub Actions `ci.yml` to pass
3. Run `npm run deploy:staging` to verify
4. Run `npm run deploy:prod` for production
5. Verify health check endpoint returns 200

## Rollback
- If deployment fails: `npm run deploy:rollback <previous-tag>`
- Notify #deployments Slack channel via `/deploy` skill

## Rules
- Never deploy to production without staging verification
- Always tag releases: `git tag -a v$(date +%Y%m%d-%H%M%S)`
```

模板 5：代码重构流程（含安全网机制）

```
# CLAUDE.md — Safe Refactoring

## Safety Rules
1. Never edit files without reading them first
2. Always run tests before and after changes
3. If tests fail, stop and ask user before proceeding
4. Create a git worktree for large refactorings: `claude -w refactor-name`
```

```
## Refactoring Steps
1. `git checkout -b refactor/<description>`
2. Read all files matching the pattern (use Glob + Read)
3. Write a plan to `.claude/plans/refactor.md` and enter plan mode
4. Execute changes incrementally (one module at a time)
5. Run `npm run test` after each module
6. If tests pass, commit; if not, fix or revert

## Verification
- Lint must pass: `npm run lint`
- Type check must pass: `npm run typecheck`
- Tests must pass: `npm run test`
- No console errors in dev mode

## Rollback
- Use `git diff` to review all changes before final commit
- Use `/rewind` in Claude Code if conversation turns wrong
```

模板 6：文档自动生成

```
# CLAUDE.md — Documentation Generator

## Documentation Rules
- All public APIs must have JSDoc / TSDoc comments
- README.md must include install, usage, and example sections
- Changelog follows Keep a Changelog format

## Auto-generation Workflow
1. When user says "update docs" or "generate docs":
2. Read `src/` to find exported functions and types
3. Extract existing JSDoc comments
4. Update `docs/api.md` with missing entries
5. Update `README.md` if new features added
6. Run `npm run docs:build` to verify

## Output Format
- API docs: Markdown tables with params, returns, examples
- Guides: Step-by-step with code blocks
- Always use ` ```ts ` for TypeScript examples
```

CHAPTER FIVE

第五章 斜杠命令与自定义工作流

5.1 内置斜杠命令完整清单

5.2 自定义斜杠命令

5.1 内置斜杠命令完整清单

斜杠命令以 `/` 开头，是 Claude Code 的快速操作入口。以下是截至 2026 年 8 月的完整内置命令表（按功能分组）：

会话管理

命令	功能	使用频率
`/clear`	清空对话历史重新开始	★★★★☆
`/compact [instructions]`	压缩上下文，可选保留指定内容	★★★★★
`/resume [session]`	恢复之前的会话	★★★☆☆
`/fork [name]`	从当前点分叉新会话	★★☆☆☆
`/rename [name]`	重命名当前会话	★★☆☆☆
`/rewind`	回退到上一状态	★★★☆☆
`/exit`	退出 Claude Code	★★★★☆

信息与诊断

命令	功能	使用频率
`/cost`	显示当前会话的 token 消耗和费用	★★★★☆
`/usage`	显示配额和速率限制状态	★★★☆☆
`/context`	可视化上下文窗口使用情况	★★★★☆
`/status`	显示版本、模型、账户信息	★★☆☆☆
`/doctor`	运行安装诊断	★☆☆☆☆
`/diff`	打开交互式 diff 查看器	★★★☆☆

模型与模式控制

命令	功能	使用频率
`/model [model]`	切换模型（sonnet/opus/haiku）	★★★★☆
`/fast [on|off]`	快速模式切换	★★☆☆☆
`/plan`	进入规划模式	★★★☆☆
`/output-style [style]`	切换输出风格	★☆☆☆☆
`/theme`	切换终端主题	★☆☆☆☆

配置与权限

命令	功能	使用频率
`/config`	打开配置界面	★★★☆☆
`/permissions`	查看/更新工具权限	★★★☆☆
`/init`	初始化 CLAUDE.md	★★☆☆☆
`/memory`	管理 CLAUDE.md 记忆文件	★★☆☆☆
`/hooks`	配置生命周期钩子	★★★☆☆
`/agents`	管理子代理配置	★★☆☆☆
`/skills`	列出可用技能	★★★☆☆
`/mcp`	管理 MCP 服务器	★★★★☆
`/plugin`	管理插件	★★☆☆☆
`/sandbox`	切换沙箱模式	★★☆☆☆

代码审查与 PR

命令	功能	使用频率
`/review [PR]`	审查 PR	★★★★☆
`/pr-comments [PR]`	获取 PR 评论	★★☆☆☆
`/security-review`	安全审查未提交变更	★★★☆☆

工作目录与集成

命令	功能	使用频率
`/add-dir <path>`	添加工作目录	★★☆☆☆
`/ide`	管理 IDE 集成	★☆☆☆☆
`/chrome`	配置 Chrome 集成	★☆☆☆☆
`/tasks`	管理后台任务	★★☆☆☆
`/feedback`	提交反馈	★☆☆☆☆

2025-2026 新增命令

命令	功能	版本引入
`/batch`	批量操作	v2.1.63
`/simplify`	代码简化	v2.1.63
`/effort`	调整努力程度滑块	v2.1.91 (Opus 4.7)
`/loop`	循环定时任务	v2.1.105
`/recap`	返回会话时恢复上下文	v2.1.108
`/ultrareview`	云端并行多代理审查（现为 `/code-review ultra` 别名）	v2.1.111
`/less-permission-prompts`	提议白名单	v2.1.111
`/tui fullscreen`	全屏渲染	v2.1.110
`/focus`	专注模式	v2.1.110
`/undo`	回退（`/rewind` 的别名）	2026
`/fast`	Fast 模式切换（仅限 Opus 5/4.8，约 2.5 倍速度）	2026-07（随 Opus 5）
`/goal`	目标管理	2026

5.2 自定义斜杠命令

创建方法：`.claude/commands/` 目录下写 Markdown

自定义斜杠命令在文件系统上以 Markdown 文件形式存在，放置于：

- 项目级：`.claude/commands/<command-name>.md`
- 用户级：`~/.claude/commands/<command-name>.md`

文件名（不含 `.md`）即为命令名。文件内容是 prompt 模板，支持使用 `$ARGUMENTS` 占位符接收用户输入参数。

自定义命令示例 1：自动提交+测试+推送+创建 PR

`.claude/commands/ship.md`：

```
---
name: ship
description: Run tests, build, commit, push, and create a PR
---

Run the full deployment pipeline:

1. Run `npm run test` — if fail, stop and report
2. Run `npm run build` — if fail, stop and report
3. Check `git status` for uncommitted changes
4. If changes exist, commit with message: "$ARGUMENTS" (or "ship: automated commit" if empty)
5. Push to current branch: `git push origin $(git branch --show-current)`
6. Create a draft PR using `gh pr create --draft --title "$ARGUMENTS" --fill`
7. Report the PR URL
```

使用方法：`/ship "feat: add user authentication"`

自定义命令示例 2：Bug 报告生成器

`.claude/commands/bugreport.md`：

```
---
name: bugreport
description: Generate a structured bug report from current state
---

Analyze the current codebase state and generate a bug report:
```

```
1. Run `git log --oneline -5` to get recent commits
2. Read `package.json` to get project metadata
3. Check if there are any error logs in recent terminal output
4. Gather environment info: Node version (`node -v`), OS (`uname -a`)
5. Output a structured bug report in this format:

## Bug Report

**Project**: [name from package.json]
**Version**: [version]
**Environment**: [node version, os]
**Last Commits**: [list]

### Current Behavior
[Describe what is wrong based on logs/code]

### Expected Behavior
[Infer from code context]

### Steps to Reproduce
1. [step]
2. [step]

### Possible Cause
[Your analysis]
```

使用方法：`/bugreport`

自定义命令示例 3：代码审查清单

`.claude/commands/cr.md`：

```
---
name: cr
description: Run a comprehensive code review on current changes
---

Perform a code review on the current uncommitted changes or the specified PR/file:

1. Get the diff: `git diff` (or read the specified file)
2. Check for:
   - Security issues (SQL injection, XSS, hardcoded secrets)
   - Performance concerns (N+1 queries, unnecessary re-renders)
   - Type safety (any types, missing null checks)
   - Error handling (missing try/catch, unhandled rejections)
   - Test coverage (new code without tests)
3. Rate each category: PASS / WARN / FAIL
4. Provide specific line-by-line suggestions where issues found
5. Output a summary with priority fixes
```

```
Categories:
- [ ] Security
- [ ] Performance
- [ ] Type Safety
- [ ] Error Handling
- [ ] Tests
- [ ] Style/Consistency
```

使用方法：`/cr` 或 `/cr src/auth.ts`

自定义命令与 Skills 的关系

Claude Code 支持两种自定义能力：Skills（推荐）和 Legacy Commands（`.claude/commands/`）。两者的选择建议：

场景	推荐方式	原因
复杂多步骤工作流	Skills	支持渐进式加载、附带脚本和资源
简单 prompt 模板	Legacy Commands	轻量、快速创建
需要自动触发	Skills	支持 description 匹配自动激活
团队共享	两者皆可	Skills 更易管理，Commands 更轻量

对于新项目，建议优先使用 Skills。Legacy Commands 虽然简单，但不支持渐进式加载——整个命令内容始终占用上下文空间。

CHAPTER SIX

第六章 Hooks 钩子——自动化你的开发管道

6.1 四种 Hook 类型

6.2 Hooks 配置方法

6.3 实用 Hook 配置案例

6.1 四种 Hook 类型

Hooks 是 Claude Code 最强大的可编程扩展机制。它们拦截 Claude 的生命周期事件，执行用户定义的命令或 LLM 提示。截至 2026 年 8 月，官方参考已包含 **31 个 Hook 事件**（5月版的 12 种之后，v2.1.x 陆续新增了 PostToolUseFailure、PostToolBatch、PermissionDenied、SubagentStart、TeammateIdle、TaskCreated、TaskCompleted、InstructionsLoaded、ConfigChange、CwdChanged、FileChanged、WorktreeCreate、WorktreeRemove、PostCompact、Elicitation、StopFailure 等），核心仍可归纳为四种类型：

类型	包含事件	触发时机	能否阻断
前置拦截型	PreToolUse, UserPromptSubmit, PermissionRequest	工具执行前 / 用户提交前 / 权限请求时	✅ 是
后置响应型	PostToolUse, PostToolUseFailure, SubagentStop, Stop	工具成功后 / 失败后 / 子代理结束 / 会话停止时	❌ 否（仅反馈）
会话生命周期型	SessionStart, SessionEnd, Setup, PreCompact, PostCompact	会话开始 / 结束 / 初始化 / 压缩前后	❌ 否
通知型	Notification	Claude 需要通知用户时	❌ 否

核心阻断机制：PreToolUse 和 PermissionRequest 返回 exit code 2 时可以阻断操作，这是构建安全边界的基石。

2026 年新增——四种 handler 类型：hook 不再只有 `command` 一种执行方式，现已支持 `command`（执行 shell 命令）、`prompt`（发给轻量模型做判断）、`agent`（交给完整子代理处理）、`http`（POST 到指定 URL，适合对接企业 webhook/通知网关）四种。

常用 Hook 事件速查（完整 31 个见官方文档）

Hook 事件	触发时机	最佳用途	阻断能力
PreToolUse	工具执行前	安全检查、自动批准、参数校验	✅ exit 2
PermissionRequest	权限对话框出现时	自动批准/拒绝特定操作	✅ exit 2
PostToolUse	工具成功执行后	自动格式化、运行测试、记录日志	❌
PostToolUseFailure	工具执行失败后	错误处理、自动回滚尝试	❌
UserPromptSubmit	用户按回车提交时	输入验证、上下文注入	✅ exit 2
Stop	Claude 完成响应时	任务完整性检查、强制继续工作	✅ exit 2
SubagentStart	子代理启动时	子代理初始化、环境设置	❌
SubagentStop	子代理结束时	子代理结果验证	✅ exit 2
SessionStart	会话开始/恢复时	加载上下文、设置环境变量	❌
SessionEnd	会话终止时	清理资源、归档日志	❌
Setup	使用 `--init` 时	项目初始化脚本	❌
PreCompact / PostCompact	上下文压缩前/后	备份关键信息、压缩后恢复	❌
WorktreeCreate / WorktreeRemove	worktree 创建/移除时	隔离环境的初始化与清理	❌
Notification	需要通知用户时	桌面通知、TTS、消息推送	❌

6.2 Hooks 配置方法

Hooks 定义在 `settings.json` 中，支持项目级（`.claude/settings.json`）和用户级（`~/.claude/settings.json`）。

配置结构

```
{
  "hooks": {
    "EventName": [
      {
        "matcher": "ToolPattern",
        "hooks": [
          {
            "type": "command",
            "command": "your-command-here",
            "timeout": 60
          }
        ]
      }
    ]
  }
}
```

matcher 语法：- 精确匹配：`"Edit"`、`"Bash"` - 正则匹配：`"Edit|Write"`、`"Notebook.*"` - 通配全部：`"*"` 或 `""` - MCP 工具：`"mcp__github__*"`

环境变量

Hook 命令中可以使用以下 Claude 提供的上下文变量：

变量	说明
`$CLAUDE_FILE_PATHS`	正在修改的文件路径
`$CLAUDE_TOOL_INPUT`	完整工具参数的 JSON
`$CLAUDE_PROJECT_DIR`	项目根目录
`$CLAUDE_NOTIFICATION`	通知消息内容

Prompt-based Hooks

除了 `command` 类型，Hooks 还支持 `prompt` 类型——将 hook 输入和自定义 prompt 发送给轻量级 Claude 模型（默认 Haiku），由模型返回 JSON 决策。这种类型适合需要「判断」而非「执行」的场景。

Prompt-based hook 的响应必须包含：

```
{"ok": true, "reason": "Explanation"}
```

6.3 实用 Hook 配置案例

案例 1：Write/Edit 后自动用 Prettier 格式化

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "npx prettier --write $CLAUDE_FILE_PATHS",
            "timeout": 30
          }
        ]
      }
    ]
  }
}
```

效果：每次 Claude 修改或创建文件后，自动运行 Prettier 格式化，无需手动执行。

案例 2：Bash 执行前检查危险命令

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": "echo \"$CLAUDE_TOOL_INPUT\" | jq -r '.command' | grep -qE '(rm -rf /|sudo |chmod 777 |curl .*\\|.*sh|wget .*\\|.*sh)' && echo
            "timeout": 10
          }
        ]
      }
    ]
  }
}
```

效果：拦截包含 `rm -rf /`、`sudo`、管道到 shell 等危险模式的 Bash 命令，exit 2 阻断执行并提示 Claude。

案例 3：任务完成发送桌面通知

```
{
  "hooks": {
    "Notification": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "osascript -e 'display notification \"$CLAUDE_NOTIFICATION\" with title \"Claude Code\"' || notify-send 'Claude Code' '$CLAUDE
            "timeout": 10
          }
        ]
      }
    ]
  }
}
```

效果：当 Claude 需要通知用户时（如长时间任务完成），自动触发 macOS Notification Center 或 Linux notify-send。

案例 4：Stop 时记录会话日志

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "echo '{\"timestamp\":\"'$(date -u +%Y-%m-%dT%H:%M:%SZ)'\",\"event\":\"session_stop\"}' >> $CLAUDE_PROJECT_DIR/.claude/logs/se
            "timeout": 10
          }
        ]
      }
    ]
  }
}
```

效果：每次 Claude 停止响应（任务完成或中断）时，向项目级日志文件追加一条 JSON 记录，便于后续分析使用模式。

案例 5：PreToolUse 自动批准低风险操作

```
{
  "hooks": {
    "PermissionRequest": [
      {
        "matcher": "Read|Glob|Grep|ListMcpResourcesTool",
        "hooks": [
          {
            "type": "command",
            "command": "echo '{\"permissionDecision\":\"allow\"}'",
            "timeout": 5
          }
        ]
      }
    ]
  }
}
```

效果：对所有只读工具（Read/Glob/Grep 等）自动批准权限请求，减少交互打断。注意：不要对 Write/Edit/Bash 使用此模式。

案例 6：Prompt-based Hook（LLM 决策）

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "prompt",
            "prompt": "Evaluate if all tasks are complete based on this session data: $ARGUMENTS. Check if there are uncommitted changes, failing tes
          }
        ]
      }
    ]
  }
}
```

效果：使用轻量级 Claude 模型（默认 Haiku）评估是否应该停止。若检测到未完成任务，自动要求 Claude 继续工作。Prompt-based Hooks 的决策质量取决于 prompt 的清晰度——建议包含明确的判断标准和输出格式要求。

CHAPTER SEVEN

第七章 国内可用性综合评估

7.1 无需任何配置即可使用的工具清单

7.2 需要 API Key 但国内可直连的工具

7.3 需要代理才能使用的工具

7.4 国内开发者推荐工具栈

7.1 无需任何配置即可使用的工具清单

> **2026年8月前提说明**：本章评估「工具在国内网络下的可用性」。自 2026 年起，国内用户使用 Claude Code 的主流方式是通过 `ANTHROPIC_BASE_URL` 接入国产模型官方端点（智谱 GLM、DeepSeek、Kimi 等，可用 cc-switch 一键切换）——该方式下 Claude Code 本体及下列本地工具全部零障碍可用。底层模型接入方案的详细对比见姊妹篇《Claude Code 国内用户完整使用手册》第二章。

以下工具在国内标准网络环境下开箱即用，无需代理、无需 API Key：

内置 Tools（约 20 个）： - Read、Write、Edit、Glob、Grep、Bash、Agent（原 Task）、NotebookEdit - EnterPlanMode、ExitPlanMode、EnterWorktree、ExitWorktree - CronCreate、CronList、CronDelete、LSP、TodoWrite - AskUserQuestion、Skill、TaskStop、SendMessage

官方 MCP Server（本地操作类）： - Filesystem（本地文件） - Memory（本地存储） - Sequential Thinking（纯推理） - Time（系统时间） - Git（本地仓库） - SQLite（本地数据库） - PostgreSQL（连接本地/内网） - MySQL（连接本地/内网） - Redis（连接本地/内网） - MongoDB（连接本地/内网） - Qdrant / Chroma（本地向量库） - Puppeteer / Playwright（本地浏览器，抓取国内站点） - Lark/飞书（国内服务） - Docker（本地容器） - Kubernetes（内网集群） - GitLab（内网实例）

Claude Code 核心功能： - 所有斜杠命令（`/compact`、`/cost`、`/mcp` 等） - Skills 系统（本地文件加载） - Hooks 系统（本地命令执行） - Plan Mode、Worktree、Git 集成

7.2 需要 API Key 但国内可直连的工具

以下工具的 API 服务端在国内可直连，但需要注册获取 API Key：

工具/服务	API Key 获取难度	国内直连	说明
Brave Search	简单（brave.com/search/api）	✅	每月有免费额度
Tavily Search	简单（tavily.com）	✅	专为 AI 设计
Exa Search	中等（exa.ai）	✅	语义搜索
Notion	简单（notion.so/my-integrations）	✅	个人/团队知识库
Slack	中等（api.slack.com/apps）	✅	工作区管理员权限
Vercel	简单（vercel.com/account/tokens）	✅	部署管理
AWS	复杂（需 AWS 账户 + IAM）	✅ 国内区域	阿里/腾讯/华为云需对应 MCP Server
OpenAI API	困难（需海外账户）	⚠️ 不稳定	建议用国内替代
Context7	中等（context7.com）	✅	技术文档检索

获取建议： - Brave Search 和 Tavily 是最易获取的搜索 API，注册后即刻可用； - Notion 的 Integration Token 在个人设置页面一键生成； - Slack App 需要工作区管理员审批，适合团队环境； - AWS 国内区域（北京/宁夏）需企业实名认证，个人开发者门槛较高。

7.3 需要代理才能使用的工具

以下服务因网络原因在国内无法稳定直连，需要代理或替代方案：

工具/服务	原因	替代方案
GitHub API	国内访问不稳定	内网 GitLab + GitLab MCP Server
WebSearch（默认）	依赖 Bing/Google	自部署 SearXNG + SearXNG MCP Server
WebFetch（境外站点）	网络限制	优先使用国内镜像/文档站点
Gmail	Google 服务不可用	飞书邮件 / 企业微信
Discord	无国内节点	飞书群 / Slack
OpenAI/Anthropic API	官方限制	国内中转 API（注意合规）
npm registry（部分）	偶发不稳定	配置淘宝/腾讯云 npm 镜像

Puppeteer/Playwright 首次安装注意事项： 浏览器 MCP Server 首次运行会自动下载 Chromium/WebKit。在国内建议预先设置环境变量：

```
export PUPPETEER_DOWNLOAD_HOST=https://registry.npmmirror.com/binary.html?path=chromium-browser-snapshots/
export PLAYWRIGHT_DOWNLOAD_HOST=https://registry.npmmirror.com/binary.html?path=playwright/
```

或者在项目 `.npmrc` 中配置：

```
puppeteer_download_host=https://registry.npmmirror.com/binary.html?path=chromium-browser-snapshots/
playwright_download_host=https://registry.npmmirror.com/binary.html?path=playwright/
```

7.4 国内开发者推荐工具栈

以下是一套完整的国内友好配置方案，可直接复制使用。

`.mcp.json` **（项目级）**

```
{
  "mcpServers": {
    "filesystem": {
      "type": "stdio",
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-filesystem", "."]
    },
    "postgres": {
      "type": "stdio",
      "command": "npx",
      "args": ["-y", "@anthropic-ai/mcp-server-postgres"],
      "env": {
        "DATABASE_URL": "${DATABASE_URL:-postgresql://localhost:5432/dev}"
      }
    },
    "memory": {
      "type": "stdio",
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-memory"]
    },
    "puppeteer": {
      "type": "stdio",
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-puppeteer"]
    },
    "sequential-thinking": {
      "type": "stdio",
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-sequential-thinking"]
    }
  }
}
```

`.claude/settings.json` **（项目级，入 Git）**

```
{
  "$schema": "https://json.schemastore.org/claude-code-settings.json",
  "permissions": {
    "allow": [
      "Read",
      "Glob",
      "Grep",
      "Bash(npm run:*)",
      "Bash(git:*)",
      "Bash(make:*)",
      "Edit(src/**)",
      "Write(src/**)",
      "mcp__filesystem__*",
      "mcp__memory__*",
      "mcp__postgres__*"
    ],
    "deny": [
      "Read(.env*)",
      "Read(secrets/**)",
      "Bash(rm -rf:*)",
      "Bash(sudo:*)",
      "Edit(package-lock.json)",
```

```
      "Edit(.git/**)"
    ],
    "ask": [
      "WebFetch",
      "Bash(curl:*)",
      "Bash(docker:*)"
    ],
    "defaultMode": "acceptEdits"
  },
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "npx prettier --write $CLAUDE_FILE_PATHS || true",
            "timeout": 30
          }
        ]
      }
    ],
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": "echo \"$CLAUDE_TOOL_INPUT\" | jq -r '.command' | grep -qE '(rm -rf /|sudo |chmod 777 )' && exit 2 || exit 0",
            "timeout": 10
          }
        ]
      }
    ]
  },
  "respectGitignore": true,
  "includeGitInstructions": false
}
```

`.claude/settings.local.json` **（个人级，不入 Git）**

```
{
  "env": {
    "DATABASE_URL": "postgresql://user:password@localhost:5432/mydb",
    "GITHUB_TOKEN": "ghp_xxxxxxxxxxxx",
    "ANTHROPIC_API_KEY": "sk-ant-xxxxxxxxxxx"
  }
}
```

使用说明： 1. 将 `.claude/settings.local.json` 加入 `.gitignore`； 2. 根据实际技术栈调整 `permissions.allow` 中的 Bash 前缀模式； 3. 若不使用 PostgreSQL，从 `.mcp.json` 和权限列表中移除相关项； 4. Puppeteer 首次使用需确保 Chromium 下载完成（配置国内镜像）； 5. 建议先从最小化配置开始，逐步添加所需 MCP Server，避免一次性配置过多导致权限管理复杂化。

APPENDIX

附录

A. MCP Server 分类速查表（浓缩版）

B. 推荐配置模板合集

常见工具使用问题排查

C. 参考资源链接

D. 信息来源与免责声明

A. MCP Server 分类速查表（浓缩版）

分类	推荐 Server	安装命令	国内可用
文件	Filesystem	`npx -y @modelcontextprotocol/server-filesystem /path`	✅
搜索	Brave Search	`npx -y @modelcontextprotocol/server-brave-search`	⚠️
数据库	PostgreSQL	`npx -y @anthropic-ai/mcp-server-postgres`	✅
浏览器	Puppeteer	`npx -y @modelcontextprotocol/server-puppeteer`	✅
记忆	Memory	`npx -y @modelcontextprotocol/server-memory`	✅
推理	Sequential Thinking	`npx -y @modelcontextprotocol/server-sequential-thinking`	✅
Git	Git / GitHub	见 3.3 节	✅ / ⚠️
协作	Lark	`npx -y @modelcontextprotocol/server-lark`	✅
DevOps	Docker	`npx -y @modelcontextprotocol/server-docker`	✅

B. 推荐配置模板合集

模板 1：最小化配置（个人学习）

`.mcp.json`：

```
{"mcpServers": {"memory": {"type": "stdio", "command": "npx", "args": ["-y", "@modelcontextprotocol/server-memory"]}}}
```

`.claude/settings.json`：

```
{"permissions": {"allow": ["Read", "Edit", "Write", "Bash", "Glob", "Grep"], "defaultMode": "acceptEdits"}}
```

模板 2：前端开发

`.mcp.json`：

```
{
  "mcpServers": {
    "filesystem": {"type": "stdio", "command": "npx", "args": ["-y", "@modelcontextprotocol/server-filesystem", "."]},
    "memory": {"type": "stdio", "command": "npx", "args": ["-y", "@modelcontextprotocol/server-memory"]},
    "fetch": {"type": "stdio", "command": "npx", "args": ["-y", "@modelcontextprotocol/server-fetch"]}
  }
}
```

`.claude/settings.json`：

```
{
  "permissions": {
    "allow": ["Read", "Edit", "Write", "Bash(npm:*)", "Bash(git:*)", "Glob", "Grep"],
    "deny": ["Bash(rm -rf:*)", "Read(.env*)"],
    "defaultMode": "acceptEdits"
  },
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [{"type": "command", "command": "npx prettier --write $CLAUDE_FILE_PATHS || true", "timeout": 30}]
      }
    ]
  }
}
```

模板 3：安全 CI（团队共享）

`.claude/settings.json`：

```
{
  "permissions": {
    "allow": ["Read", "Glob", "Grep", "Bash(npm run:*)", "Bash(git:*)", "Edit(src/**)", "Write(src/**)"],
    "deny": ["Read(.env*)", "Read(secrets/**)", "Bash(rm -rf:*)", "Bash(sudo:*)", "Bash(curl:*)", "Bash(wget:*)", "WebFetch", "WebSearch"],
    "ask": ["Bash(docker:*)", "mcp__*"],
    "defaultMode": "default"
  },
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [{"type": "command", "command": "echo \"$CLAUDE_TOOL_INPUT\" | jq -r '.command' | grep -qE '(sudo|rm -rf /|curl .*\\|.*sh|wget .*\\
      }
    ]
  }
}
```

模板 4：完整开发环境（全栈）

`.mcp.json`:

```
{
  "mcpServers": {
    "filesystem": {"type": "stdio", "command": "npx", "args": ["-y", "@modelcontextprotocol/server-filesystem", "."]},
    "postgres": {"type": "stdio", "command": "npx", "args": ["-y", "@anthropic-ai/mcp-server-postgres"], "env": {"DATABASE_URL": "${DATABASE_URL}"}},
    "redis": {"type": "stdio", "command": "npx", "args": ["-y", "@modelcontextprotocol/server-redis"]},
    "memory": {"type": "stdio", "command": "npx", "args": ["-y", "@modelcontextprotocol/server-memory"]},
    "puppeteer": {"type": "stdio", "command": "npx", "args": ["-y", "@modelcontextprotocol/server-puppeteer"]},
    "github": {"type": "stdio", "command": "npx", "args": ["-y", "@modelcontextprotocol/server-github"], "env": {"GITHUB_TOKEN": "${GITHUB_TOKEN}"}}
  }
}
```

`.claude/settings.json`:

```
{
  "permissions": {
    "allow": ["Read", "Edit", "Write", "Bash", "Glob", "Grep", "Agent", "mcp__filesystem__*", "mcp__memory__*", "mcp__postgres__*", "mcp__redis__*"],
    "deny": ["Read(.env*)", "Bash(rm -rf /:*)", "Bash(sudo:*)"],
    "ask": ["WebFetch", "WebSearch", "mcp__github__*", "mcp__puppeteer__*"],
    "defaultMode": "acceptEdits"
  },
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [{"type": "command", "command": "npx prettier --write $CLAUDE_FILE_PATHS || true", "timeout": 30}]
      },
      {
        "matcher": "mcp__postgres__*",
        "hooks": [{"type": "command", "command": "echo 'Database operation completed'", "timeout": 5}]
      }
    ]
  }
}
```

常见工具使用问题排查

问题	可能原因	解决方案
Edit 提示 "old_string not found"	文件内容已变更或空格不匹配	重新 Read 文件获取最新内容
Bash 命令超时	timeout 设置过短或命令确实卡住	增加 timeout 值，或 KillShell 终止
MCP Server 无法连接	进程未启动或端口冲突	`claude mcp test <name>` 诊断，检查日志
Skills 未被触发	描述不够精确或未匹配任务	优化 description，或手动 `/skill-name` 调用
Hooks 未生效	matcher 正则错误或事件名拼错	检查大小写，用 `/hooks` 查看注册状态
Plan Mode 无输出	当前目录非 Git 仓库	确认在 Git 仓库内运行，检查 `.git/` 存在
Worktree 创建失败	分支名冲突或路径已存在	使用唯一分支名，清理旧 worktree
WebFetch 返回空	目标站反爬虫或需要 JS 渲染	换 Puppeteer MCP Server 抓取

C. 参考资源链接

资源	URL
Claude Code 官方文档	https://code.claude.com/docs
MCP 规范文档	https://modelcontextprotocol.io
MCP Server 官方仓库（已归档）	https://github.com/modelcontextprotocol/servers-archived
MCP Registry（官方发现入口）	https://registry.modelcontextprotocol.io
MCP 市场/社区索引	https://mcpservers.com
Agent Skills 开放标准	https://agentskills.io
Claude Code 插件官方目录	https://claude.com/plugins（官方市场仓库 anthropics/claude-plugins-official）
JSON Schema 配置校验	https://json.schemastore.org/claude-code-settings.json
Claude Code 更新日志	`/release-notes`（Claude Code 内命令）
社区论坛	https://github.com/anthropics/claude-code/discussions
skills.sh 技能包管理器	https://skills.sh

D. 信息来源与免责声明

信息来源： - Anthropic 官方文档（code.claude.com/docs，截至 2026 年 8 月） - MCP 官方博客与规范仓库（2026-07-28 规范公告）、MCP Registry - Claude Code CLI 内置帮助（`claude --help`、`/help`）与 GitHub Releases - 社区博客与实测验证（博客日期 2025-12 至 2026-08） - Claude Code 源码分析（通过公开的系统提示和工具描述）

免责声明： 1. Claude Code 和 MCP 生态迭代迅速，部分工具名称或配置格式可能在未来版本中调整，建议以官方文档为准； 2. 国内可用性评估基于 2026 年 8 月网络环境，实际可达性可能因地区、运营商、时间而变化； 3. 涉及 API Key 的服务请遵守各平台服务条款，注意密钥安全，切勿将密钥提交到版本控制； 4. 本手册中的配置示例仅供参考，生产环境使用前请根据团队安全策略审查和调整； 5. Hooks 的自动执行特性可能带来安全风险，建议在企业环境中使用 `managed-settings.json` 统一管理； 6. 2026 年 8 月 14 日起 Auto Mode 对付费账户默认开启——请主动检查你的权限配置（`permissions` 白名单/黑名单）是否符合预期，避免过度授权。

本手册版本：2026 年 8 月版（基于 2026 年 5 月版全面核实更新） 最后更新：2026-08-22

会员专属资料 · 持续更新

一起玩转 AI 的地方

「河马的CC商业实战」星球里，每周都有看得懂、用得上的 AI 实战：

新栏目「河马带你逛GitHub」每周一期，实战长文、会员资料持续更新。

微信扫码 · 加入「河马的CC商业实战」

河马的CC商业实战 出品