

MCP (Model Context Protocol)

模型上下文协议

2026年5月25日 | AI Agent 知识分享 | 第5期

一、一句话概括

MCP (Model Context Protocol) 是 Anthropic 于 2024 年底开源的标准协议，为 AI 模型提供了一种「通用 USB-C 接口」——让 Agent 通过统一的协议标准连接任意外部工具和数据源（数据库、文件系统、API 服务），无需为每个工具单独写适配代码，从根本上解决了「N 个模型 × M 个工具」的集成爆炸问题。

二、核心原理

>> 客户端-服务端架构 (Client-Server Architecture)

MCP 采用三层架构：MCP Host（承载 AI 模型的应用，如 Claude Desktop、IDE 插件）→ MCP Client（负责与 Server 建立 1:1 连接）→ MCP Server（每个工具或数据源对应一个独立 Server 进程）。每个 Server 通过 stdio 或 HTTP+SSE 协议暴露其能力（Resources、Tools、Prompts），Host 通过 Client 统一发现和调用。

>> 三大核心原语 (Resources / Tools / Prompts)

Resources（资源）：暴露只读数据，类似 RESTful GET，如数据库表、文件内容、API 响应。Tools（工具）：暴露可执行操作，类似 POST，如发送邮件、创建工单。Prompts（提示模板）：暴露可复用的 prompt 片段。三者通过统一的 JSON-RPC 协议传输，模型通过「工具发现」机制动态获取每个 Server 的能力列表，无需在 prompt 中硬编码。

>> 与传统 Function Calling 的本质区别

传统 Function Calling 是模型提供商的 API 功能（如 OpenAI 的 tools 参数），需要开发者手动注册每个函数，且绑定特定模型。MCP 是协议层标准，不依赖特定模型或框架——同一个 MCP Server 可被任何支持 MCP 的 Host 调用（Claude、VS Code、Cursor、自定义 Agent），实现「一次开发，多处复用」。A2A (Agent-to-Agent) 则是 MCP 的补充，解决 Agent 之间的通信问题。

核心要点：MCP 是 AI Agent 时代的「标准化外设接口」——就像 USB-C 统一了外设连接，MCP 统一了 Agent 的工具接入。截至2026年5月，MCP 生态已覆盖 2000+ 个官方/社区 Server，覆盖数据库（PostgreSQL、SQLite）、开发工具（Git、Docker、Kubernetes）、办公套件（Google Workspace、Notion、Slack）、浏览器自动化（Playwright）等。

三、测试实战

测试一：MCP Server 发现与健康检查

操作：安装 mcp-cli (pip install mcp-cli)，运行 mcp start path/to/server.py 启动任意 MCP Server，然后用 mcp list 查看暴露的能力列表 (resources/tools/prompts)，验证 Server 是否正确注册了所有能力

工具：mcp-cli (<https://github.com/modelcontextprotocol/mcp-cli>)，或直接使用 Claude Desktop 的 MCP 配置面板

指标：能力发现完整率 (已注册能力 ÷ 预期能力 × 100%)、Server 启动成功率、心跳检测通过率

测试二：跨 Host 兼容性测试

操作：用同一个 MCP Server (如文件系统 Server)，分别连接 Claude Desktop、VS Code (Cline 插件)、以及自定义 Python Agent (使用 mcp Python SDK)，验证三种 Host 下工具调用结果是否一致

工具：MCP Inspector (<https://github.com/modelcontextprotocol/inspector>) — 官方可视化调试工具

指标：跨 Host 结果一致率、调用成功率、响应时间偏差 (Host 间差异应 < 200ms)

测试三：工具安全边界测试

操作：针对文件系统 Server (@modelcontextprotocol/server-filesystem)，构造路径穿越攻击 (如传入 ../etc/passwd、%2e%2e/ 等编码路径)，验证 Server 是否在沙箱内有效拦截越权访问

工具：MCP filesystem server (<https://github.com/modelcontextprotocol/servers/tree/main/src/filesystem>)，配合 Burp Suite 或 Python requests 做模糊测试

指标：越权访问拦截率 (理想值 100%)、路径穿越攻击检出率、异常请求日志完整率

测试四：并发请求与资源隔离测试

操作：同时启动 3 个 MCP Client 连接同一个 Server，每个 Client 并发发送 10 个工具请求，验证 Server 能否正确复用连接、不互相干扰、不泄漏跨会话数据

工具：k6 (<https://k6.io/>) 或 locust 发送并发 JSON-RPC 请求，搭配 MCP Server 日志分析

指标：并发错误率 (< 1% 为合格)、请求吞吐量 (RPS)、内存泄漏检测 (10轮后内存增长 < 5%)

四、常见坑点

坑点1：自定义 Server 的 JSON-RPC 实现容易出错

MCP 协议基于 JSON-RPC 2.0，很多人手写 Server 时容易踩的坑：忘了在 initialize 握手阶段返回正确的 protocolVersion、initialize 的 capabilities 字段遗漏导致 Host 无法发现工具、请求 id 不唯一导致请求-响应匹配错乱。建议直接用官方 SDK (Python/TypeScript) 而非手写 JSON-RPC，SDK 已处理好所有协议细节。

坑点2：Tool 描述质量依然是关键瓶颈

MCP 解决了「工具发现」问题，但没解决「工具选错」问题。MCP Server 返回的工具列表中每个 tool 的 description 字段仍然由开发者编写，质量参差不齐。和 Function Calling 一样，模糊的描述会导致模型选错工具。而且 MCP 环境下工具列表可能来自多个 Server，选错概率更大。建议每个 tool 的 description 包含明确的「使用条件」和「参数边界」。

坑点3：stdio 模式下的进程生命周期管理

最常用的 MCP 传输方式是 stdio（子进程通信），这意味着每个 MCP Server 都是一个独立子进程。生产环境中如果 Host 崩溃或异常退出，这些子进程会成为孤儿进程持续运行。而且 stdio 模式不支持跨网络调用（只能本机）。建议生产部署使用 HTTP+SSE 传输模式，并配置进程管理工具（supervisor/systemd）兜底子进程生命周期。

五、延伸阅读

- Anthropic MCP 官方文档（含架构说明和快速入门）：<https://modelcontextprotocol.io/>
- MCP 官方 Python SDK: <https://github.com/modelcontextprotocol/python-sdk>
- MCP Server 市场（2000+ 个社区 Server）：<https://github.com/modelcontextprotocol/servers>
- A2A（Agent-to-Agent Protocol）Google 官方: <https://developers.google.com/a2a>
- 深入理解 MCP 协议规范（JSON-RPC 细节）：<https://spec.modelcontextprotocol.io/>