

Function Calling 与 MCP 协议 | 深究 MCP 协议的设计



该文档在线链接和 PDF 版本均在视频简介。

一、Function Calling

2.1 要解决的问题

传统聊天大模型只会说话，**没有工具调用能力**，这使得大模型：

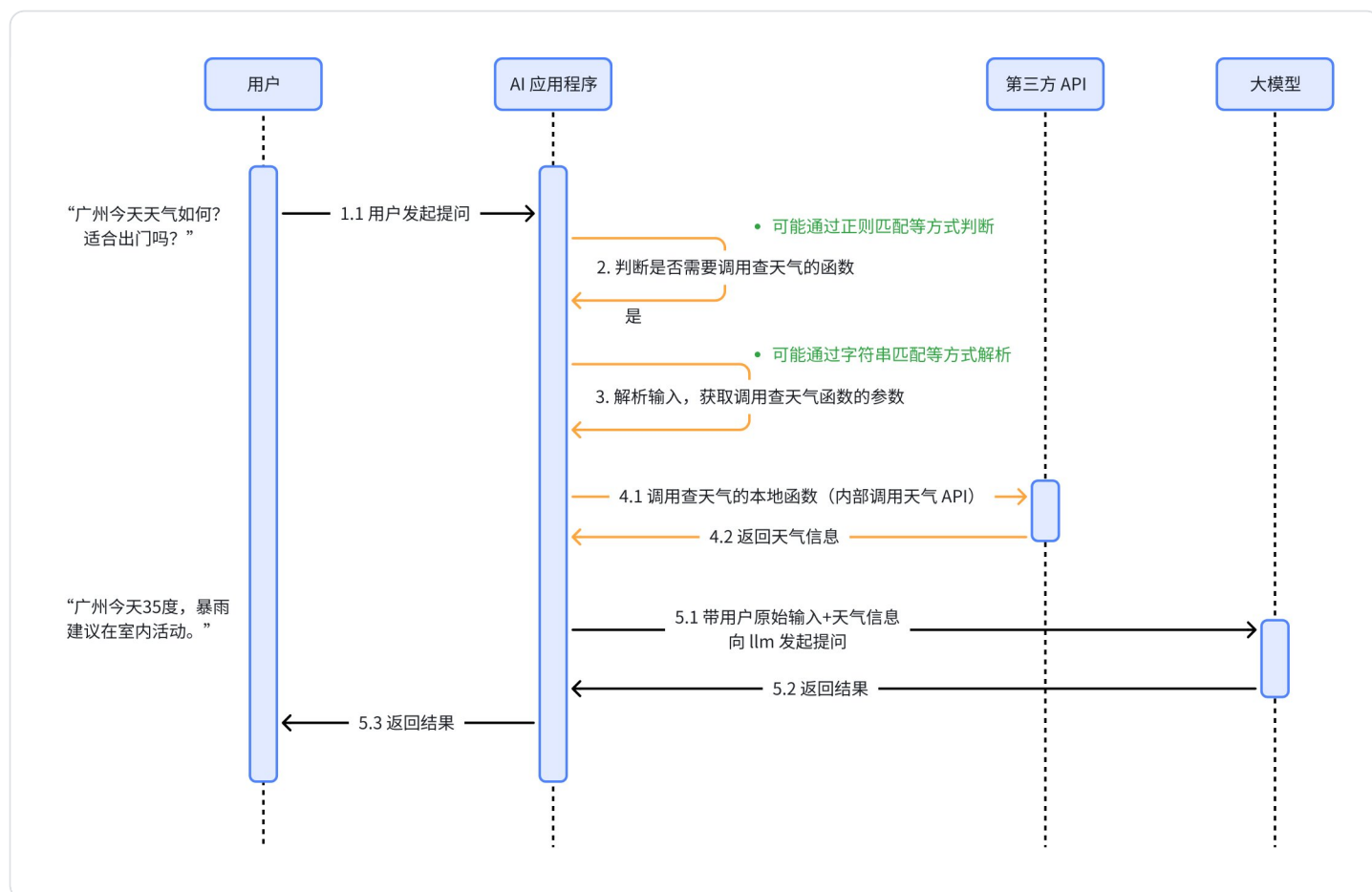
1. **无法感知环境**：无法与外部数据源交互，如通过 API 查询网页、查看用户本地文件、访问远程数据库等等
2. **无法改变环境**：无法帮用户实际执行任务，如跑代码、发邮件、上传作业等

2.2 如何解决问题

后端 + LLM

传统方案

工作流程



Hard Code 方案 | by @bilibili 堂吉诃德拉曼查的英豪 (Carbon Based)

存在的问题

1. 是否调用工具、调用什么工具由后端负责判断，逻辑复杂且容易误判。



AI 这么智能，为什么不让它来帮我判断？

2. 调用工具的参数由后端负责构建，难度很大。



AI 这么智能，为什么不让它来帮我生成参数？

Function Calling 方案

Function Calling 是什么

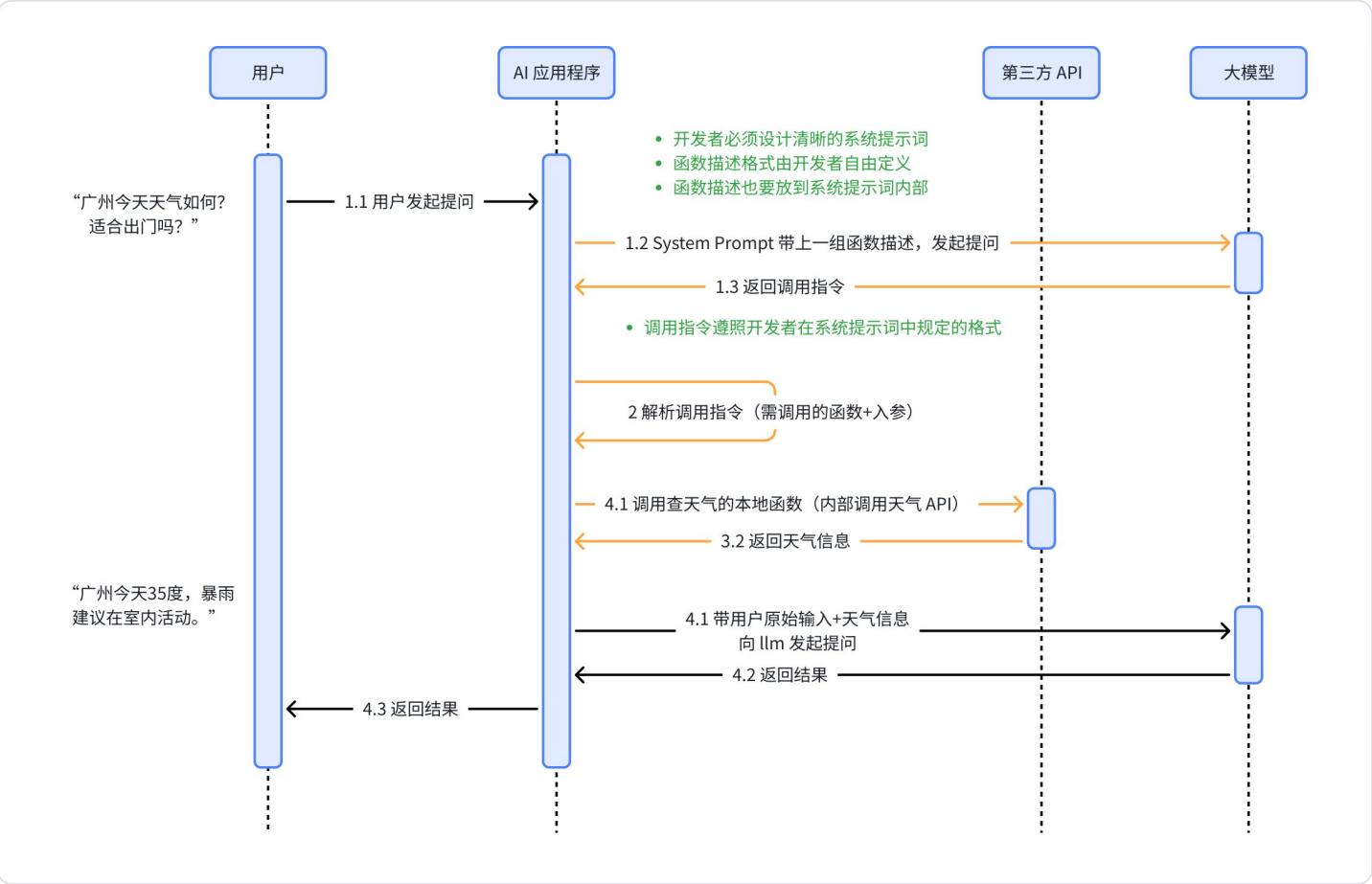
广义的 **Function Calling** 是指让大模型能够调用外部工具的一种技术实现：先向大模型提供可用函数的列表及说明，由大模型在对话过程中智能判断是否需要调用函数，并自动生成调用所需的参数，最终用文字返回符合约定格式的函数调用请求。

狭义的 **Function Calling** 特指大模型提供商在模型内部与 API 层面做了支持的一种能力，它最早由 OpenAI 引入：

- **在模型层面：**模型提供商需对大模型进行特别优化，使其具备根据上下文正确选择合适函数、生成有效参数的能力（比如有监督微调、强化学习）。
- **在 API 层面：**模型提供商需额外开放对 Function Calling 的支持（比如 GPT API 中提供了一个 `functions` 参数）。

基于提示词的 Function Calling

工作流程



基于提示词的方案 | by @bilibili 堂吉诃德拉曼查的英豪 (Not Ai, Carbon Based Version)

System Prompt 示例

```
1  # 你的角色
2  你是一个函数调用助手，我将提供多个函数的定义信息，包括函数名称、作用、参数及参数类型。
3
4  # 你的任务
5  - 根据用户的输入，判断是否需要调用某个函数
6  - 如果需要，请**严格按照以下格式**输出函数调用指令：
7  ```json
8  { "name": "函数名", "arguments": { "参数名": "参数值" } }
9  ```
10
11 # 函数定义信息
12 1. **get_weather**
13    - 作用：查询指定城市的天气情况
```

```
14     - 参数：
15     - `city` (string) : 城市名称
16 2. **get_time**
17     - 作用：查询指定城市的当前时间
18     - 参数：
19     - `city` (string) : 城市名称
20
```

用户提问示例

```
1  “广州的天气怎么样？”
```

模型回复示例

```
1  { "name": "get_weather", "arguments": { "city": "广州" } }
```

存在的问题

1. **输出格式不稳定。**如调用指令中存在多余自然语言。
2. **容易出现幻觉。**模型可能编造并不存在的函数名或参数。



大模型提供商能否对模型进行微调、强化学习，提升大模型在这一方面的能力？

3. **对开发者依赖度高。**函数描述、调用指令格式、提示词逻辑完全由开发者设计。

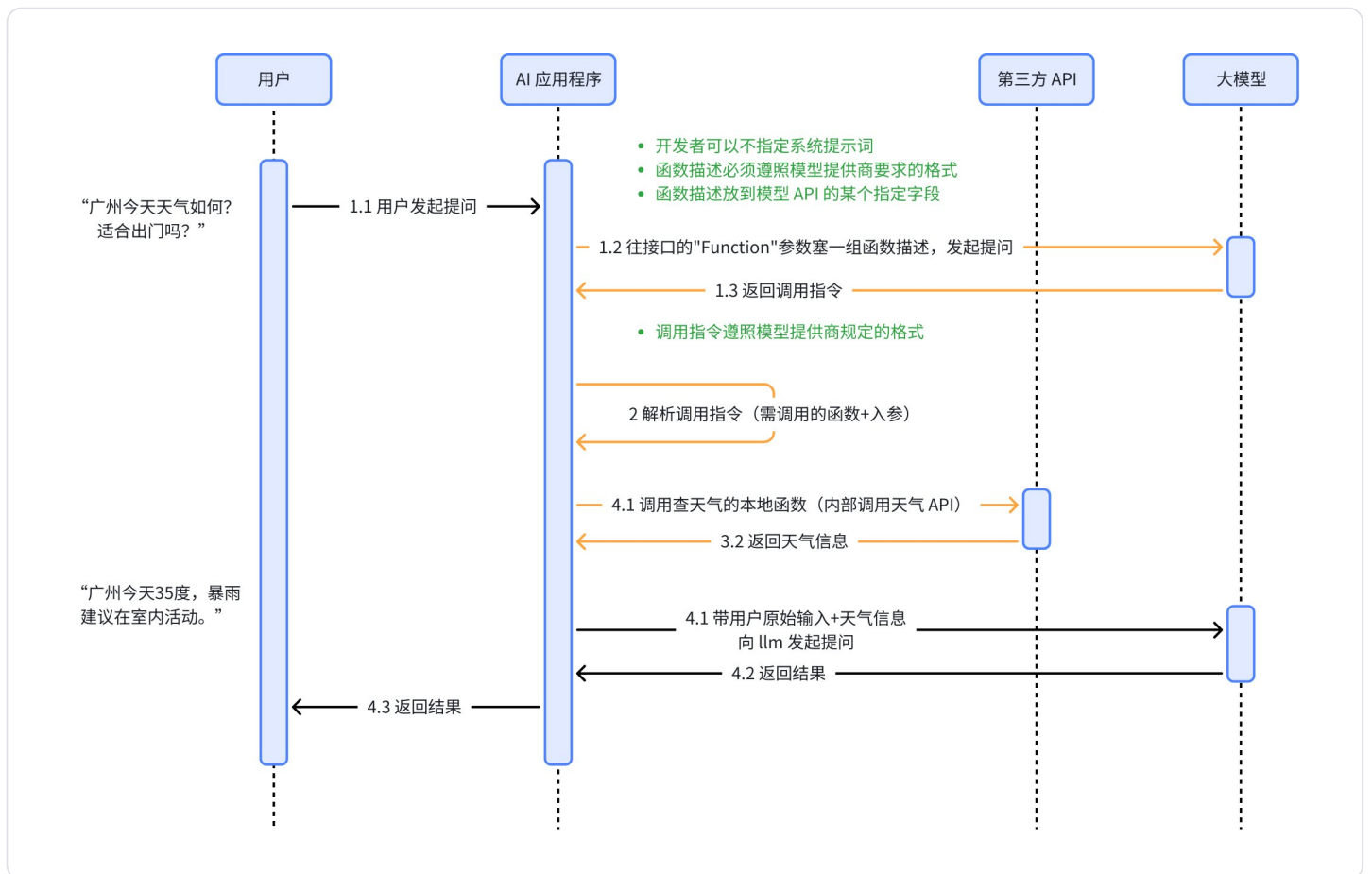


函数描述、调用指令格式能否由大模型提供商来指定？系统提示词中的“说明与规则”逻辑能否由大模型提供商来兜底？

4. **上下文冗长，Token 消耗大。**为确保调用逻辑正确，往往需要在 system prompt 中加入大量说明与规则。

基于 API 的 Function Calling

工作流程



基于 API 的 Function Calling | by @bilibili 堂吉诃德拉曼查的英豪 (Not Ai, Carbon Based Version)

1. 用户发起提问

用户通过自然语言提出问题，例如：“广州今天天气如何？适合出门吗？”

2. 后端第一次向大模型 API 发起请求，获取函数调用指令

后端向大模型 API 传入用户原始输入、**函数描述**和其他上下文信息，获取调用指令。函数描述包括函数名称、用途说明、参数结构等。

接口入参

```
1  {
2    "messages": [
3      {
4        "role": "system",
5        "content": "你是一个助手，可以根据用户的请求调用工具来获取信息。"
6      },
7      {
8        "role": "user",
9        "content": "广州今天天气如何？适合出门吗？"
10     }
11   ],
12   "functions": [
13     {
14       "name": "getWeather",
15       "description": "获取指定城市的天气",
```

```
16     "parameters": {
17         "type": "object",
18         "properties": {
19             "location": {
20                 "type": "string",
21                 "description": "城市名称, 比如北京"
22             },
23             "date": {
24                 "type": "string",
25                 "description": "日期, 比如 2025-08-07"
26             }
27         },
28         "required": ["location", "date"]
29     }
30 }
31 ]
32 }
33
```

3. 模型生成调用指令

模型会智能判断是否需要调用函数，选择合适的函数，并基于上下文自动生成结构化的调用指令（函数名 + 参数），例如：

调用指令示例（OpenAI Function Calling）

```
1  {
2      "function_call": {
3          "name": "getWeather",
4          "arguments": {
5              "location": "Guangzhou",
6              "date": "2025-07-17"
7          }
8      }
9  }
```

4. 后端解析调用指令，并执行实际的函数调用

后端接收到模型返回的调用指令后，解析调用指令，得到函数名称和参数，执行对应的方法（如调用天气查询函数），并获取结果。调用指令例如：

5. 后端第二次向大模型 API 发起请求，将刚才的调用结果和其他上下文信息一起传给模型，生成最终的回复

后端将函数执行结果 + 其他上下文信息（包括用户原始输入）传给模型，模型判断此时已有足够的信息回答问题，不再需要调用函数了，于是直接生成最终结果，例如：“广州今天35度，暴雨，建议在室内活动”。

存在的问题

1. 后端应用适配不同大模型时存在大量冗余开发
2. 可选模型有限

二、MCP 协议

3.1 要解决的问题

1. 工具接入的冗余开发问题

AI 应用接入他人开发的新工具需完整 **copy** 代码和函数描述，接入几个就要 copy 几次。

2. 工具复用困难

环境问题导致 **copy** 的代码不一定能跑；很多企业不提供可供 **copy** 的源码；跨语言的代码 **copy** 了没用。

3.2 如何解决问题



如果是你会怎么解决以上问题？（假设现在完全没有 MCP 协议）

从问题出发

-> 冗余开发、复用困难问题基本都是由 copy 代码带来的，怎么才能用上别人的方法，但又不用 copy 别人的代码？

- 思路一：导包式接入，AI 应用开发者将工具代码拉到本地调用。
 - 可行吗？⚠️ 跨语言调用问题无法解决
 - 跨语言问题是客观无解的吗？-> 能否本地另起一个进程运行该语言的执行环境，工具代码原封不动运行在对应语言环境中，AI 应用再通过进程间通信（如管道、套接字）获取标准化的返回结果？
 - 所以可行吗？✅ -> 并且，这种接入方式更适合被称为“本地服务式接入”
- 思路二：远程服务式接入，工具开发者将工具独立部署，封装成标准化 API，约好统一的请求/响应格式，AI 应用开发者只需按规定传入参数并解析返回值即可，不需要关心工具的实现语言、运行环境、内部逻辑。
 - 可行吗？✅

从目标出发

-> 接入工具、复用工具对开发者来说最理想的方式是什么？

- 开发者只需添加一条配置（如工具的唯一标识或访问地址）就可以接入自己 / 别人的工具

-> 当前的非理想状态是什么？

- 当前每新增一个工具，开发者需要在链路中做**两处**人工适配：
 - a. 补充工具描述
 - b. 补充工具代码

-> 从非理想状态到理想状态必须满足什么条件？

- 要让“配置替代人工适配”，必须满足以下两点：
 - 新增配置后，AI 应用后端要能根据配置**自动获取工具的描述信息**
 - 新增配置后，AI 应用后端要能根据配置**自动定位工具调用入口并执行调用**

从问题到技术需求

-> 本地服务式接入场景

- 如何在**任意** AI 应用中通过一条标准化配置：
 - 拉取**任意**工具的包到本地，并在本地起一个进程，将工具作为一个服务运行起来（只要工具开发者有提供对应的包）
 - 自动获取工具的描述信息、自动完成调用过程（通过**本地进程间通信**）

-> 远程服务式接入场景

- 如何在**任意** AI 应用中通过一条标准化配置：
 - 访问**任意**工具的远程服务（只要工具开发者有对外提供服务）
 - 自动获取工具的描述信息、自动完成调用过程（通过**远程服务调用**）

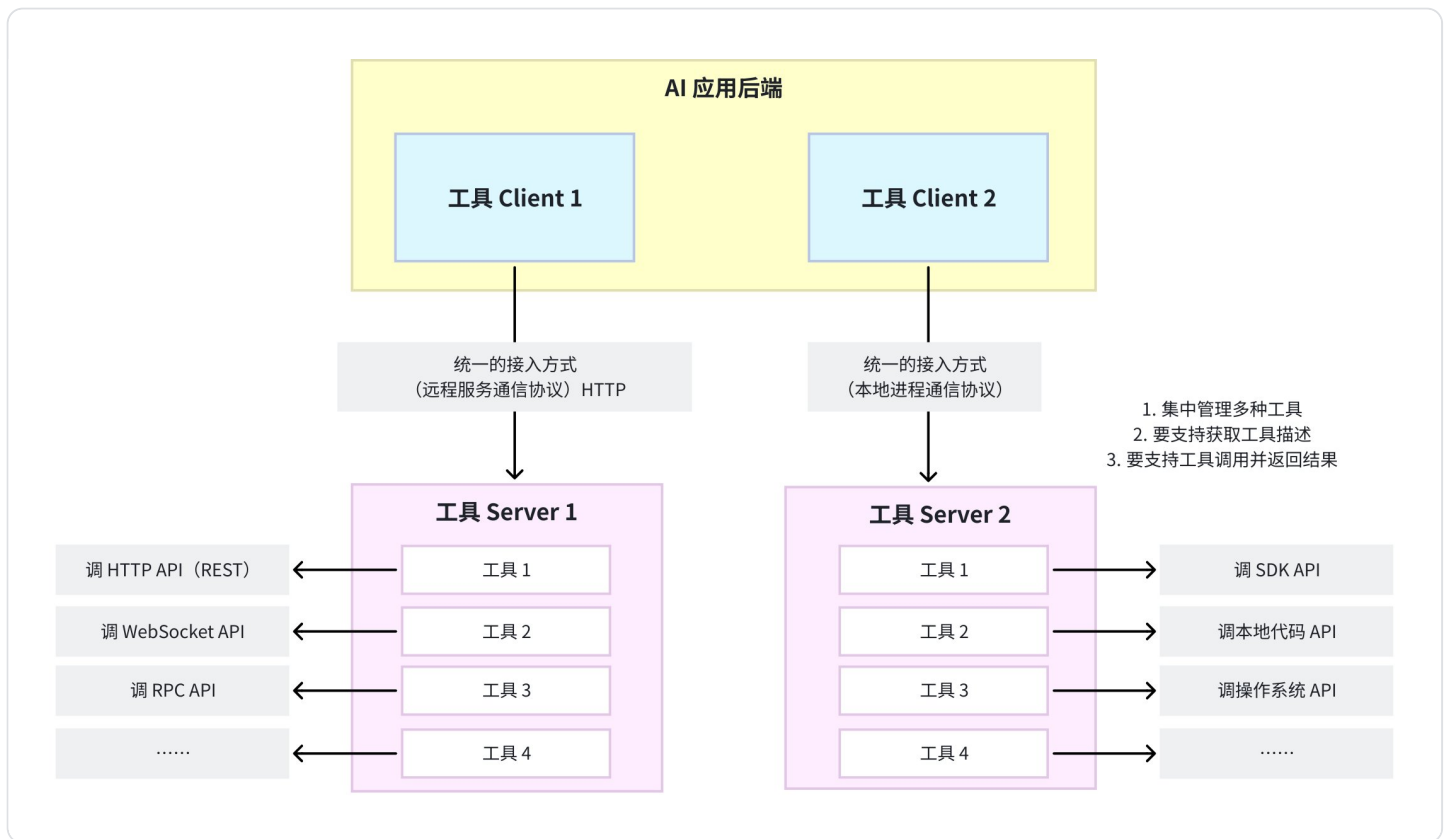
技术方案思路

1. 工具与AI 应用必须解耦合。

2. 工具与AI 应用之间的交互必须标准化。

- a. AI 应用和工具服务的**通信协议**需要**统一**（本地进程间的通信协议 / 远程服务调用的协议）
- b. AI 应用和工具服务的**接口定义**需要**统一**（需要提供哪些接口、接口需包含哪些参数）
- c. AI 应用和工具服务的**数据交换格式**需要**统一**（接口的请求 / 响应格式等）
- d. AI 应用接入工具的**配置内容**需要进行**标准化**定义
- e. 所有**工具服务**必须提供**标准化**的**接入方式**，以支持通过标准化配置即可加载工具
- f. 所有 **AI 应用内部**需实现**标准化**的**工具加载调用逻辑**，以支持通过标准化配置即可加载工具

系统架构设计



MCP 协议的雏型架构 | by @bilibili 堂吉诃德拉曼查的英豪 (Carbon Based)

3.3 MCP 协议是什么

- 诞生：

2024 年 11 月由 Anthropic（一家美国人工智能初创公司）提出，[官方文档](#)。

- 定义：

MCP is an open protocol that standardizes how applications provide context to large language models (LLMs). Think of MCP like a USB-C port for AI applications. Just as USB-C provides a standardized way to connect your devices to various peripherals and accessories, MCP provides a standardized way to connect AI models to different data sources and tools. MCP enables you build agents and complex workflows on top of LLMs and connects your models with the world. [\[1\]](#)

MCP 是一个开放协议，用于标准化应用程序向大语言模型（LLM）提供上下文的方式。你可以把 MCP 想象成 AI 应用的 USB-C 接口——正如 USB-C 提供了一种将设备连接到各种外设和配件的标准化方式一样，MCP 提供了一种将 AI 模型连接到不同数据源和工具的标准方式。借助 MCP，你可以在 LLM 之上构建智能体和复杂工作流，并将你的模型与外部世界相连接。

- 如何理解：

- **应用程序**：集成了 LLM 的具体应用。包括各家大模型的在线对话网站、集成了大模型的 IDE（如 Claude desktop）、各种 **Agent**（比如 Cursor 就是一个 Agent）、以及其他接入了大模型的普通应用。

- **上下文**：指的是模型在决策时可访问的所有信息，如当前用户输入、历史对话信息、**外部工具（tool）信息**、**外部数据源（resource）信息**、**提示词（prompt）信息**等等（这里重点只讲工具）。



和传统 API 的区别？

3.4 MCP 核心架构

MCP follows a client-server architecture where an MCP host — an AI application like [Claude Code](#) or [Claude Desktop](#) — establishes connections to one or more MCP servers. The MCP host accomplishes this by creating one MCP client for each MCP server. Each MCP client maintains a dedicated one-to-one connection with its corresponding MCP server. The key participants in the MCP architecture are:

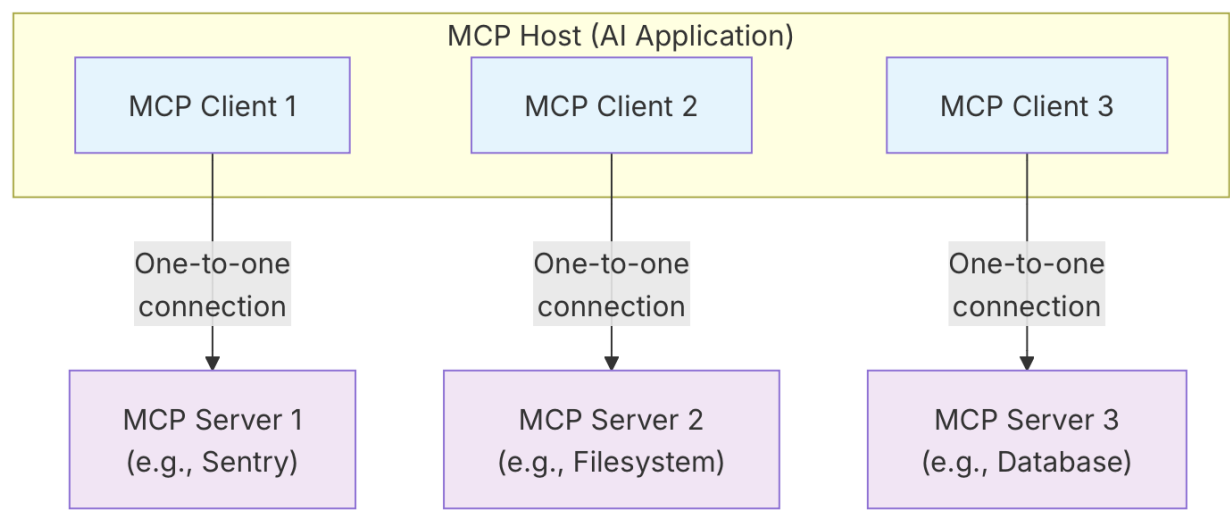
- **MCP Host**: The AI application that coordinates and manages one or multiple MCP clients
- **MCP Client**: A component that maintains a connection to an MCP server and obtains context from an MCP server for the MCP host to use
- **MCP Server**: A program that provides context to MCP clients

For example: Visual Studio Code acts as an MCP host. When Visual Studio Code establishes a connection to an MCP server, such as the [Sentry MCP server](#), the Visual Studio Code runtime instantiates an MCP client object that maintains the connection to the Sentry MCP server. When Visual Studio Code subsequently connects to another MCP server, such as the [local filesystem server](#), the Visual Studio Code runtime instantiates an additional MCP client object to maintain this connection, hence maintaining a one-to-one relationship of MCP clients to MCP servers. Note that MCP server refers to the program that serves context data, regardless of where it runs. MCP servers can execute locally or remotely. For example, when Claude Desktop launches the [filesystem server](#), the server runs locally on the same machine because it uses the STDIO transport. This is commonly referred to as a “local” MCP server. The official [Sentry MCP server](#) runs on the Sentry platform, and uses the Streamable HTTP transport. This is commonly referred to as a “remote” MCP server. [2]

MCP遵循客户端-服务器架构，其中 MCP Host——[Claude Code](#) 或 [Claude Desktop](#) 等AI应用程序——与一个或多个MCP Server 建立连接。MCP 主机通过为每个 MCP Server 创建一个 MCP Client 来实现这一目标。每个 MCP Client 都与相应的 MCP Server 保持专用的一对一连接。MCP架构的主要组成者是：

- **MCP Host**：协调和管理一个或多个 MCP Server 的人工智能应用程序
- **MCP Client**：一个组件，用于维护与 MCP 服务器的连接，并从 MCP 服务器获取上下文，供 MCP 主机使用
- **MCP Server**：一个为 MCP Client 提供上下文的程序

例如：Visual Studio Code 充当 MCP 主机。当 Visual Studio Code 建立与MCP服务器（如[Sentry MCP服务器](#)）的连接时，Visual Studio Code 运行时实例化了维护与Sentry MCP服务器连接的MCP客户端对象。当Visual Studio Code 随后连接到另一个MCP服务器时，例如[本地文件系统服务器](#)，Visual Studio Code 运行时实例化一个额外的MCP客户端对象来维护此连接，从而保持MCP客户端与MCP服务器的一对一关系。



3.5 MCP 的传输协议

MCP supports two transport mechanisms:

- Stdio transport: Uses standard input/output streams for direct process communication between local processes on the same machine, providing optimal performance with no network overhead.
- Streamable HTTP transport: Uses HTTP POST for client-to-server messages with optional Server-Sent Events for streaming capabilities. This transport enables remote server communication and supports standard HTTP authentication methods including bearer tokens, API keys, and custom headers. MCP recommends using OAuth to obtain authentication tokens.

The transport layer abstracts communication details from the protocol layer, enabling the same JSON-RPC 2.0 message format across all transport mechanisms. [\[3\]](#)

Stdio 传输

Stdio 传输本质上是本地进程间通信（IPC）的一种形式，它最常用的底层机制就是管道（pipe）。

1. 什么是 **stdio**?
- **stdio**（standard I/O）是进程的标准输入/输出接口。每个进程启动时，操作系统会给它分配三个文件描述符：

代码块

- 1 **0** → **stdin** (标准输入, 默认是键盘)
- 2 **1** → **stdout** (标准输出, 默认是屏幕)
- 3 **2** → **stderr** (标准错误输出, 默认是屏幕)

- 程序里的 `printf`、`scanf`、`cin`、`cout`、`read`、`write` 都是通过这些接口和外界交换数据的。

2. 什么是管道 (pipe) ?

- **管道**是操作系统内核提供的一种**进程间通信 (IPC) 机制**, 它允许一个进程的输出直接作为另一个进程的输入, 实现数据在两个进程之间的流动。

3. 总结: 什么是 Stdio 传输?

- 所谓 Stdio 传输, 就是通过**标准输入**和**标准输出**这两个数据流来传输数据、通过管道来连接两个进程的**标准输入/输出接口**, 使得一个进程的输出直接传给另一个进程输入, 实现进程间数据传输 (本质上是一个基于字节流的全双工通信通道)
- `stdio` 是接口, 管道是连接这接口的通道。

• 举例:

命令

```
1 ps aux
```

没有用管道

```
1 [键盘] → shell → ps(stdin) → ps(stdout) → [屏幕]
```

命令

```
1 ps aux | grep python
```

用了管道 (Stdio 传输)

- 1 # **ps** 和 **grep** 在执行时都会各自成为一个独立的进程
- 2 # **ps aux** 列出进程 → **grep python** 过滤后留下包含 "python" 的进程
- 3 # 管道 | 把 **ps aux** 的 **stdout** 作为 **grep python** 的 **stdin**
- 4 **[键盘]** → shell → **ps(stdin)** → **ps(stdout)** → **grep(stdin)** → **grep(stdout)** → 屏幕



为什么在这么多本地进程间通信的方式中选了 Stdio 传输?

HTTP + SSE 传输（旧方案，2024.10）

客户端通过 HTTP POST 向服务端发请求，服务端通过 SSE 通道返回响应结果。

- **SSE**（Server-Sent Events 服务器发送事件），是一种**服务器单向推送数据给客户端**的技术，基于 **HTTP 协议**。
- 基本原理
 - 客户端先向服务端发起一个普通的 **HTTP 请求**。
 - 服务端保持这个连接**不断开**，以 `text/event-stream` 作为响应类型，源源不断地往里写数据。
 - 客户端收到数据后会触发相应的事件回调（比如浏览器前端实时更新界面）。
- 和普通 HTTP 的核心差异
 - 支持服务端**主动、流式**地推送消息



为什么在这么多远程服务调用的协议中选了 HTTP + SSE？

- 服务端推送的必要性：MCP Server 中的工具发生了更新，需要主动向 MCP Client 推送通知

Why Notifications Matter

This notification system is crucial for several reasons:

1. Dynamic Environments: Tools may come and go based on server state, external dependencies, or user permissions
2. Efficiency: Clients don't need to poll for changes; they're notified when updates occur
3. Consistency: Ensures clients always have accurate information about available server capabilities
4. Real-time Collaboration: Enables responsive AI applications that can adapt to changing contexts

This notification pattern extends beyond tools to other MCP primitives, enabling comprehensive real-time synchronization between clients and servers. [4]

Streamable HTTP 传输（新方案，2025.03）

HTTP + SSE 传输方案的升级版，目前正在逐步取代原有的 HTTP + SSE 传输方案

- **Streamable HTTP** 并不是一个标准协议名，而是一个通用描述，指的是**基于 HTTP 协议的“可流式传输”技术**。它的核心思想是：在一个 HTTP 连接里，服务端可以**持续不断地发送数据**给客户端，客户端边接收边处理，类似“流”一样。与传统 HTTP 请求响应“一次性完成”不同，Streamable HTTP 保持连接不关闭，数据分片持续传输。常见实现方式包括：
 - HTTP/1.1 长连接 + 分块传输编码（Chunked Transfer Encoding）

- HTTP/2 流式数据
- HTTP/3 QUIC 流式传输

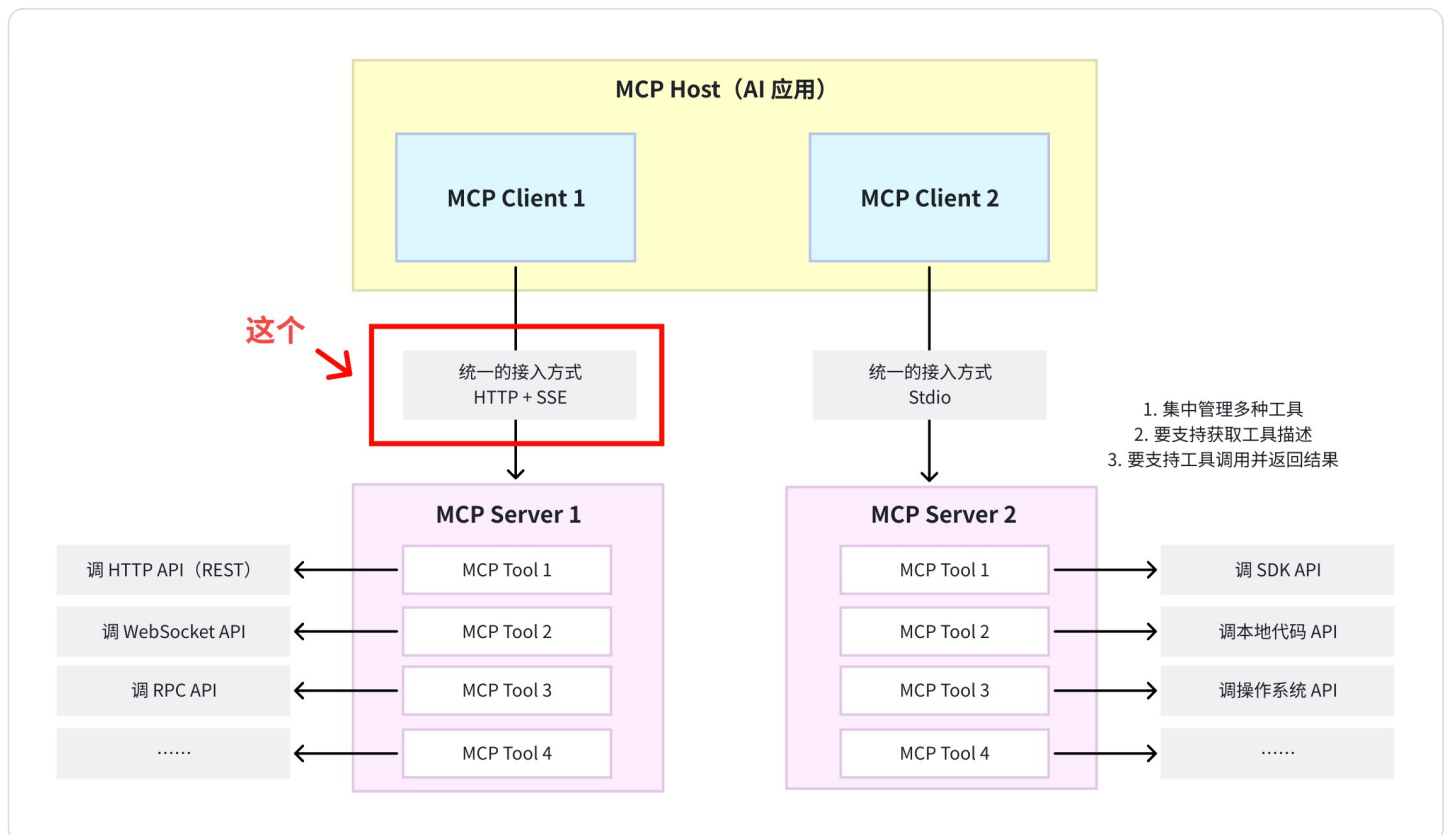


为什么 HTTP + SSE 要升级成 Streamable HTTP ?

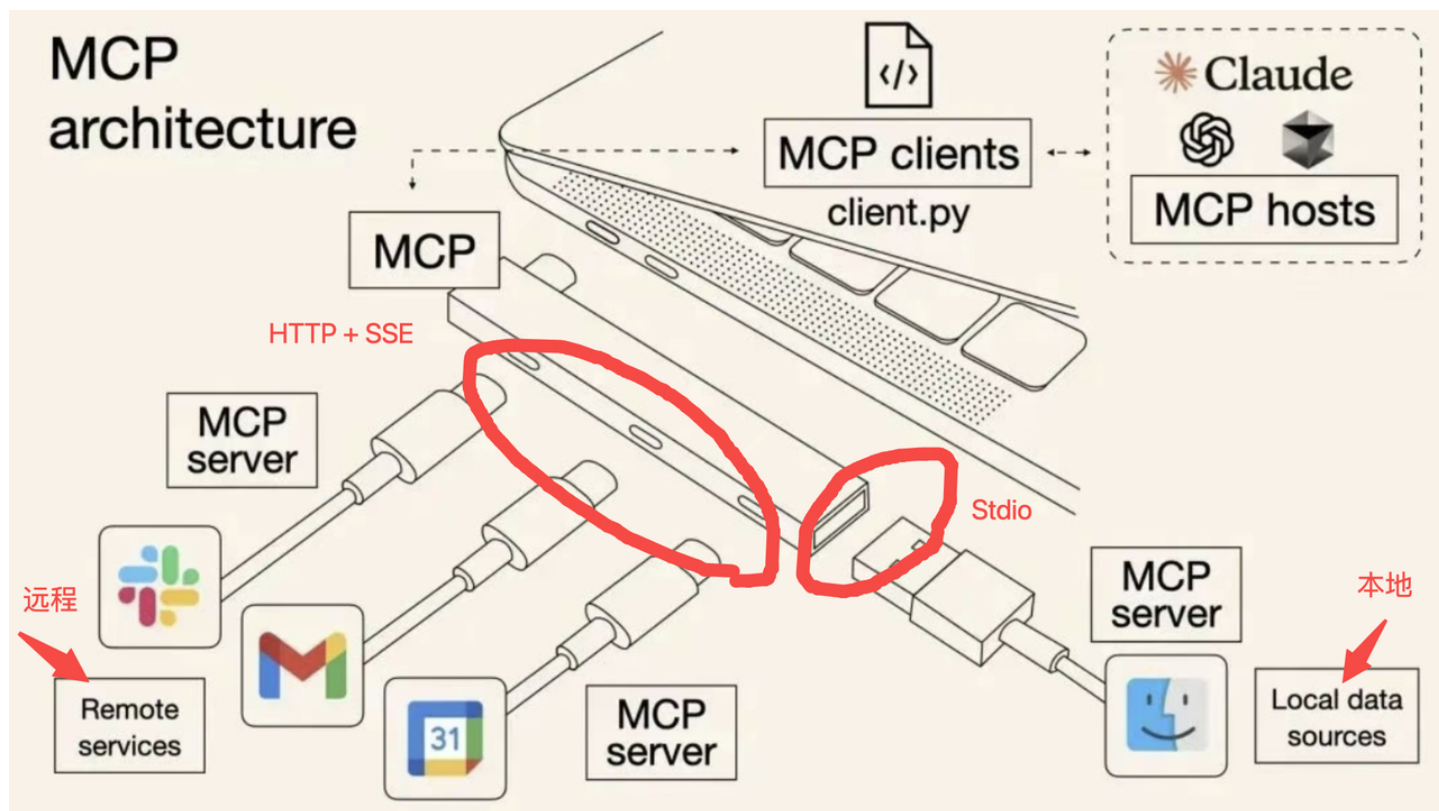
- **数据格式限制问题**：SSE 的 `Content-Type: text/event-stream` 只支持文本格式；Streamable HTTP 的 `Content-Type` 支持任意格式，如 JSON、HTML、二进制等，**更适合 AI 场景**（可能要传 JSON + 音频 + 图片）
- **跨平台兼容问题**：SSE 支持的客户端主要是浏览器端和少量语言库；而 Streamable HTTP 支持多种客户端。
- **性能问题**：SSE 是基于 **HTTP/1.1** 长连接，Streamable HTTP 可以基于 **HTTP/2/3**，支持**多路复用和双向流**。且 HTTP/2/3 的流控制和优先级机制使得高吞吐和低延迟成为可能；SSE 消息只能**文本格式**，Streamable HTTP 支持其他采用更紧凑的编码方式（比如二进制分包、压缩等）。

必须选用以上传输协议吗？

——No，因为无论哪种传输方式，都只是把各种工具的不同接入方式统一起来，对外暴露一种协议的接口而已。



MCP 协议的架构 | by @bilibili 堂吉诃德拉曼查的英豪 (Carbon Based)



3.6 回顾：技术方案最终是怎么实现的

1. 工具与AI 应用必须解耦合。

- ✓ 客户端 - 服务端架构 (MCP Host、MCP Client、MCP Server)

2. 工具与AI 应用之间的交互必须标准化。

- a. AI 应用和工具服务的通信协议需要统一 (本地进程间的通信协议 / 远程服务调用的协议)

- ✓ 本地：Stdio 传输；远程：HTTP + SSE 或 Streamable HTTP

- b. AI 应用和工具服务的接口定义需要统一 (需要提供哪些接口、接口需包含哪些参数)

- ✓ 工具服务需提供的接口：1. **tools/list** (用于返回方法列表) 2. **tools/call** (用于执行方法并返回结果。3. **notifications/tools/list_changed** (服务端主动推送，用于告知客户端方法更新) [5]

- ✓ 接口参数定义，见文档。以 tools/list 为例：

3.1 Listing Tools

To discover available tools, clients send a `tools/list` request. This operation supports pagination.

Request:

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "tools/list",
  "params": {
    "cursor": "optional-cursor-value"
  }
}
```

Response:

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "tools": [
      {
        "name": "get_weather",
        "title": "Weather Information Provider",
        "description": "Get current weather information for a location",
        "inputSchema": {
          "type": "object",
          "properties": {
            "location": {
              "type": "string",
              "description": "City name or zip code"
            }
          },
          "required": ["location"]
        }
      },
      {
        "name": "get_current_weather",
        "title": "Current Weather Information Provider",
        "description": "Get current weather information for a location",
        "inputSchema": {
          "type": "object",
          "properties": {
            "location": {
              "type": "string",
              "description": "City name or zip code"
            }
          },
          "required": ["location"]
        }
      }
    ],
    "nextCursor": "next-page-cursor"
  }
}
```

c. AI 应用和工具服务的**数据交换格式**需要**统一**（接口的请求 / 响应格式等）

✓ JSON-RPC 2.0 [6] [7]

JSON-RPC 2.0 是一种轻量级的远程过程调用（RPC）协议，基于 JSON 格式进行通信，主要特点是所有消息都是 JSON 格式，便于解析和跨语言使用。

d. AI 应用接入工具的**配置内容**需要进行**标准化**定义

以接入高德地图 MCP Server 为例，参考文档：<https://lbs.amap.com/api/mcp-server/gettingstarted>（来自ModelScope 的 MCP Server 市场：<https://modelscope.cn/mcp>）

✓ 本地服务式接入（基于 Stdio 协议）

Stdio 方式接入配置

```
1 // 在 mcpServers 配置中新增一个叫 "amap-maps"（你自己起名）的 MCP Server
2 // 通过 npx -y @amap/amap-maps-mcp-server 命令，运行高德官方提供的 MCP
  server
3 // 运行时的环境变量是 "AMAP_MAPS_API_KEY": "你申请的 API Key"
4 {
5   "mcpServers": {
6     "amap-maps": {
7       "command": "npx",
8       "args": ["-y", "@amap/amap-maps-mcp-server"],
9       "env": {
10        "AMAP_MAPS_API_KEY": "您在高德官网上申请的key"
11      }
12    }
13  }
14 }
```

✓ 远程服务式接入（基于 SSE 协议）

SSE 方式接入配置

```
1 {
2   "mcpServers": {
3     "amap-maps-streamableHTTP": {
4       "url": "https://mcp.amap.com/mcp?key=您在高德官网上申请的key"
5     }
6   }
7 }
```

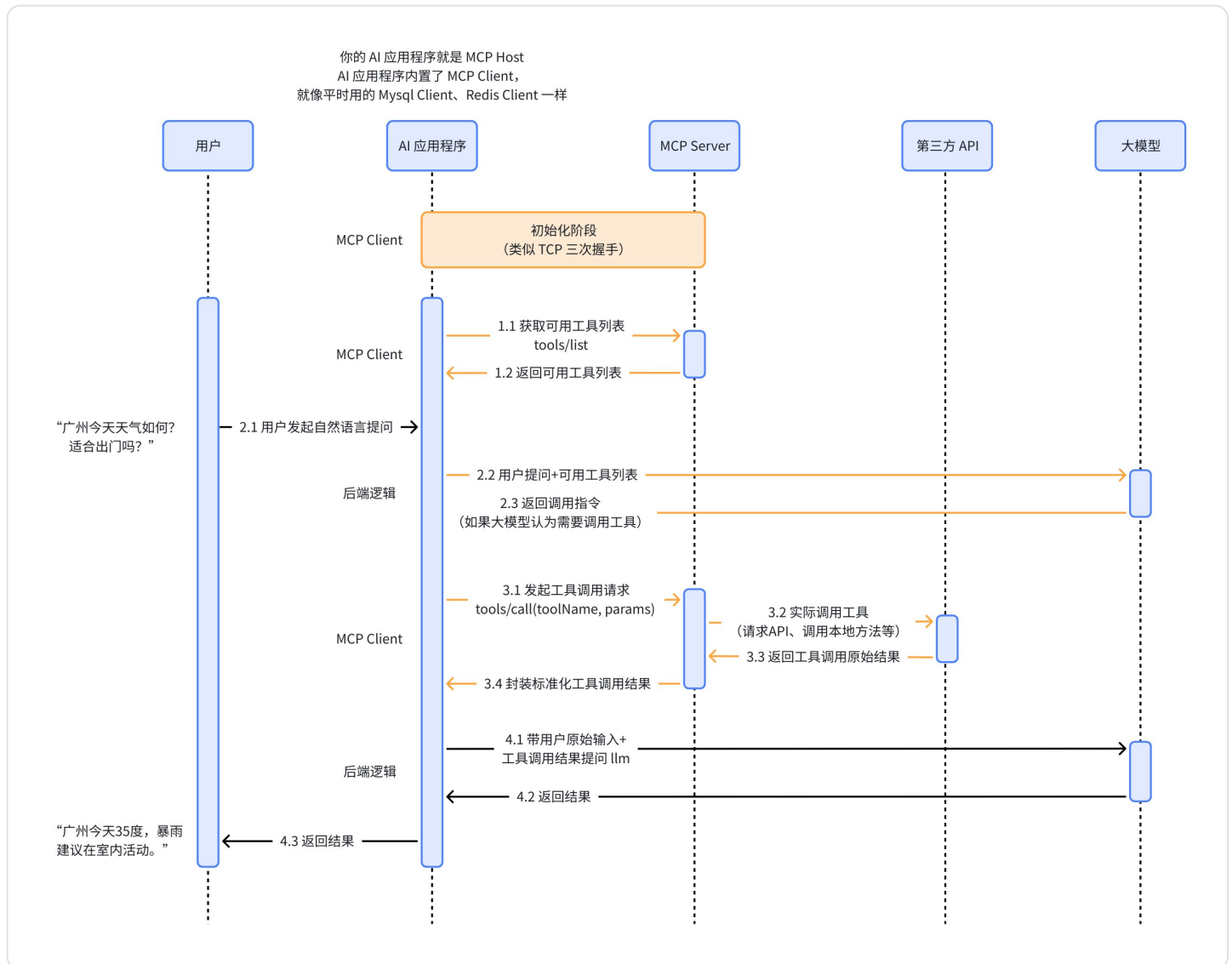
e. 所有**工具服务**必须提供**标准化**的**接入方式**，以支持通过标准化配置即可加载工具

✓ 遵循 MCP 协议开发 MCP Server，对外提供标准的接入方式。[多语言 SDK 地址](#)。

f. 所有 AI 应用内部需实现**标准化**的**工具加载调用逻辑**，以支持通过标准化配置即可加载工具

✅ 使用官方 SDK，在 AI 应用后端项目中实例化 MCP Client，调用 SDK 方法和 Server 交互。

3.7 MCP 工作流程



基于 MCP 协议的 Function Calling 工作流程 | by @bilibili 堂吉诃德拉曼查的英豪 (Carbon Based)



值得一提的是，MCP 协议和 Function Calling 之间绝不是“技术递进”的关系。所谓“MCP 协议会取代 Function Calling”的说法，其实是一种不严谨的表达。

3.8 What's More?

- 看 MCP 协议官方文档：<https://modelcontextprotocol.io/docs/getting-started/intro>，思考为何这么设计；
- 通过官方 SDK，尝试编写一个 MCP Server 并启动服务；初始化一个后端应用，创建 MCP Client，完成 Client - Server 完整交互流程的开发。MCP SDK 地址：
<https://modelcontextprotocol.io/docs/sdk>

- 找一个开源的支持了 MCP 协议的 Agent 框架，追溯其中涉及到 MCP Client、MCP Server 逻辑的所有代码。