

## 三、LangGraph接入MCP

在2024年底，claude大模型的开发公司Anthropic提出了一个MCP(Model Context Protocol)协议。通过MCP协议，可以让应用程序以一种统一的方式向LLM大语言模型提供工具调用。

而随着几千个MCP服务突然冒出，百度、阿里等厂商也开始大规模接入MCP，这个协议也迅速在互联网上引起轩然大波。按照那些流量派见风就是雨的性子，MCP也着实被大吹特吹了一波。随着MCP服务越来越火爆，LangGraph的Agent中也提供了MCP的集成。这次我们就分几个部分，逐步深度拆解MCP服务。

- MCP快速上手
- 理解MCP的stdio和sse两种实现模式
- LangGraph的Agent接入MCP服务
- 补充：自己开发一个MCP服务

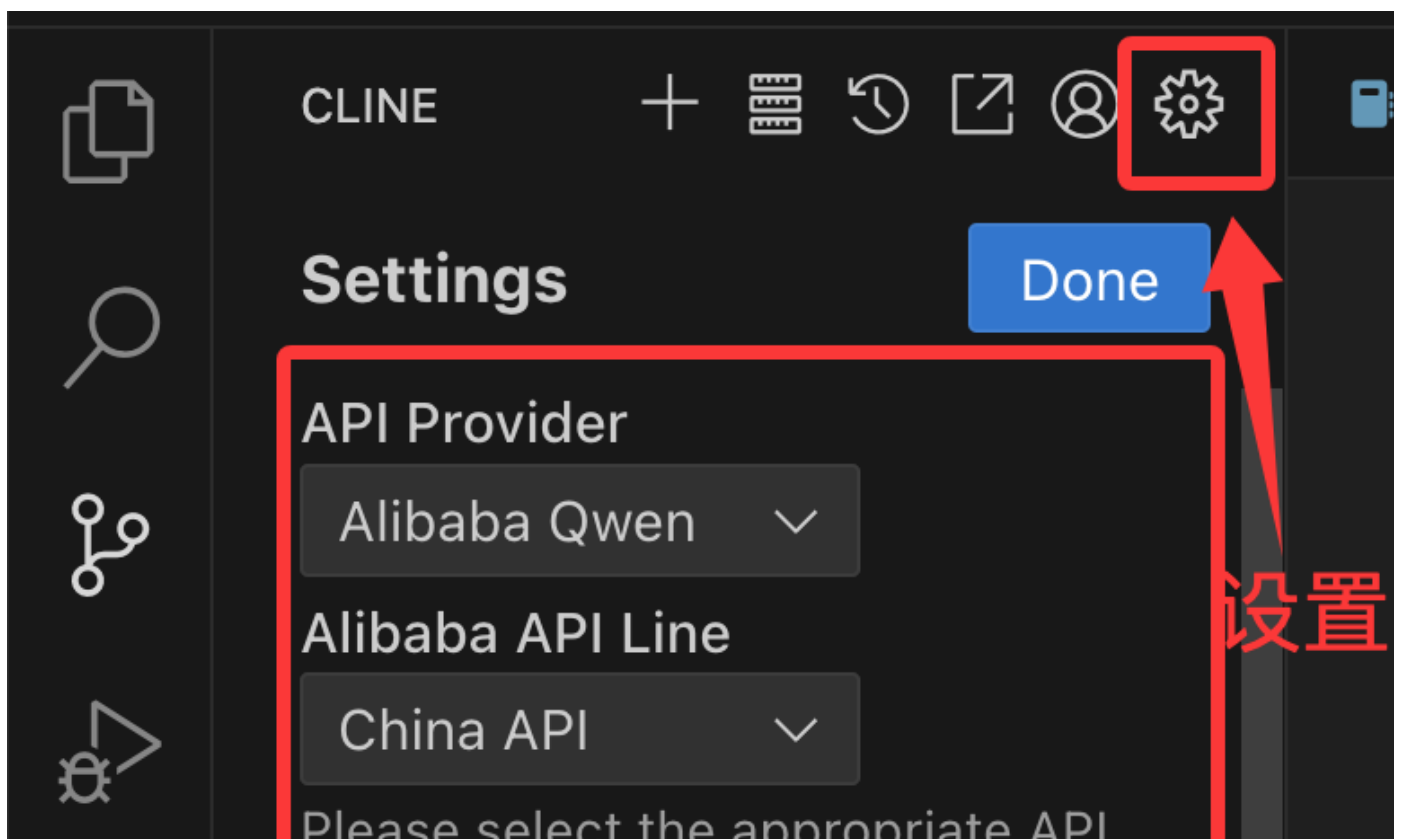
### 一、MCP快速上手

MCP最神奇的地方就是只要做下简单的配置，就能完成很多复杂的工作。那这次，我们就先不来介绍MCP那些花里胡哨的概念，直接上手来玩玩MCP。

首先，我们需要使用一个支持MCP的客户端工具。这类工具现在有很多，以后肯定也会越来越多。这次我们使用一个免费的。在VSCode中有一个开发插件，Cline。通过这个插件可以快速和AI大模型交互。

安装VSCode和Cline过程略。需要稍微注意下的是，cline插件默认是访问Claude大模型。使用Cline需要先登录。登录时，可以使用google邮箱或者github账号登录。

然后，对Cline插件进行配置，让他访问国内的模型，这样就不需要科学上网了。





interface based on your location. If you are in China, choose the China API interface. Otherwise, choose the International API interface.

### Qwen API Key

.....

This key is stored locally and only used to make API requests from this extension.

### Model

deepseek-r1



- × Does not support images
- × Does not support computer use
- ✓ Supports prompt caching

Max output: 8,000 tokens

Cache writes price: \$0.55/million tokens

Cache reads price: \$0.14/million tokens

Output price: \$2.19/million tokens

### Custom Instructions

e.g. "Run unit tests at the end",

选择模型，传入API-Key

cline插件

接下来，就可以正常的访问大语言模型了。



到这里，还一切正常，没什么不同的。接下来，在cline的上方有个小小的MCP Server按钮。通过这里，就可以打开MCP的体验之路。

以高德地图提供的MCP服务为例。高德地图开放平台提供了非常详细的MCP接入介绍：<https://lbs.amap.com/api/mcp-server/summary>。只需要在高德开放平台注册账号，登录后，在左侧菜单中选择“应用管理”-》“我的应用”，然后创建新应用，就可以申请一个API-KEY



有了这个API-KEY之后，在MCP Server按钮的右侧，选择MCP Server。在MCP的配置文件中写入以下内容：

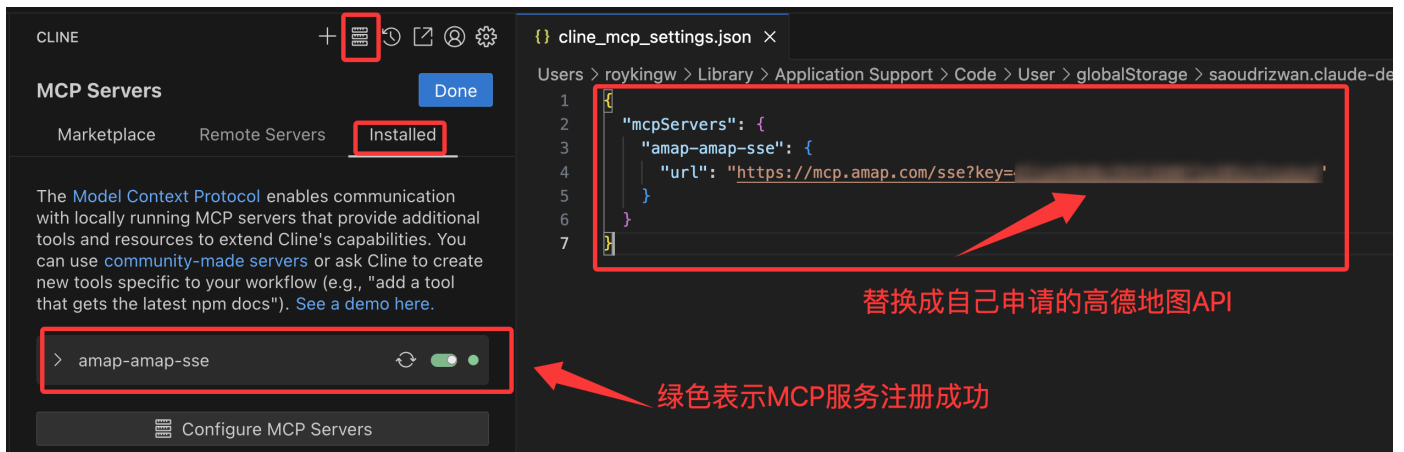
```
{
  "mcpServers": {
    "amap-amap-sse": {
      "url": "https://mcp.amap.com/sse?key=你在高德官网上申请的key"
    }
  }
}
```

> 另外一种配置方式，效果是一样的

```
"amap-maps": {
  "command": "npx",
```

```
"args": [
  "-y",
  "@amap/amap-maps-mcp-server"
],
"env": {
  "AMAP_MAPS_API_KEY": "高德地图key"
}
}
```

配置完成，右侧就能看到这个MCP服务被安装上来了。



接下来，在Cline中，如果你询问和地图相关的一些问题，那么，Cline在调用大模型的过程中，就会自动调用高德地图的MCP服务，获得地图相关的信息。

> Task: 帮我设计一条从长沙去桂林的自驾路线

\$0.0215



Checkpoint

## 用户问题

Compare

Restore



API Request

\$0.0095



为了设计一条从长沙去桂林的自驾路线，我需要获取两地之间的驾车路径信息。我可以使用amap-amap-sse MCP服务器提供的maps\_direction\_driving工具来规划这条路线。



Cline wants to use a tool on the amap-amap-sse MCP server:



maps\_direction\_driving



Auto-approve

驾车路径规划 API 可以根据用户起终点经纬度坐标规划以小客车、轿车通勤出行的方案，并且返回通勤方案的数据。

ARGUMENTS

```
{
  "origin": "112.97564,28.23305",
  "destination": "110.29332,25.2707"
}
```

## 调用高德地图的MCP服务

Response

Rich Display



```
{"origin":"112.97564,28.23305","destination":"110.29332,25.2707","paths":[{"path":"","distance":"508731","duration":"21158","steps":[{"instruction":"沿湘江北路途径湘江中路向南行驶1.0千米左转调头","road":"湘江北路","distance":"1013","orientation":"南","duration":"195"},{"instruction":"沿湘江中路向北行驶194米右转","road":"湘江中路","distance":"194","orientation":"北","duration":"31"},{"instruction":"沿三一大道向东北行驶212米靠右","road":"三一大道","distance":"212","orientation":"东"}]}]}
```



Auto-approve: Safe Commands, Browser, MCP >



Type a message...



qwen:qwen-plus

Plan

Act

并且，最终参考高德地图给出的路线信息，大模型会给出一个完整的路线规划

我设计了一条从长沙去桂林的自驾路线。路线如下：

根据规划，从长沙到桂林的自驾路线如下：

1. 沿湘江北路途径湘江中路向南行驶1.0千米左转调头
2. 沿湘江中路向北行驶194米右转
3. 沿三一大道向东北行驶212米靠右
4. 沿三一大道向东行驶839米靠左沿主路行驶
5. 沿三一大道向东行驶321米靠右
6. 沿三一大道途径浏阳河大桥、G6021杭长高速出口向东行驶7.6千米减速行驶到达收费站
7. 沿G6021杭长高速向东行驶175米向右前方行驶进入右岔路
8. 沿长沙北枢纽途径G4京港澳高速、浏阳河大桥、安子坪大桥、马家河湘江大桥、朱亭湘江大桥、洙水大桥向南行驶147.3千米靠左沿主路行驶
9. 沿G4京港澳高速向南行驶15.7千米向右前方行驶进入匝道
10. 沿洪市枢纽途径G72泉南高速向西行驶15.6千米靠左沿主路行驶
11. 沿G72泉南高速途径东阳渡桥向西行驶17.4千米靠左沿主路行驶
12. 沿G72泉南高速途径白水大桥、九牛岭桥向西南行驶117.1千米靠左沿主路行驶
13. 沿G72泉南高速途径湘江大桥、G59呼北高速向西南行驶128.9千米靠左沿主路行驶
14. 沿G72泉南高速途径土寨大桥、梁木塘高架桥、峡背漓江大桥向西南行驶43.7千米靠左
15. 沿阳沟河中桥向南行驶413米靠右
16. 沿阳沟河中桥向南行驶653米减速行驶到达收费站
17. 沿福利路途径圣隆路向南行驶5.3千米左转进入主路
18. 沿圣隆路途径环城北一路向东行驶416米向右前方行驶进入右转专用道
19. 沿中山北路向东南行驶79米右转
20. 沿中山北路向南行驶1.9千米右转
21. 沿中山北路向西行驶66米左转
22. 沿中山北路途径中山中路向西南行驶1.5千米左转
23. 沿文明路向东南行驶168米右转
24. 沿文明路一巷向西南行驶33米到达目的地

总距离约为506公里，预计耗时约5小时52分钟。

☒ Auto-approve: Safe Commands, Browser, MCP >

到这里，我们就完成了MCP服务的初体验。只需要简单配置一个JSON文件，就可以给大模型提供更多扩展的功能。

当然，除了这里演示到的高德地图的MCP服务外，还有很多五花八门的MCP服务。甚至还有很多MCP的信息聚合网站。比如 <https://mcp.so> 上，目前就聚合了好几千个各种各样的MCP服务。通过这些服务，我们可以让AI大模型完成聊天以外的各种各样的功能。包括发送邮件、浏览网页等。甚至直接操作本地文件也可以。还有阿里云百炼平台 <https://bailian.console.aliyun.com/?tab=mcp#/mcp-market> 更是将各种MCP服务聚合成了很多独立的服

务。只要开通对应的服务后，就可以在阿里云百炼的智能应用中，用图形化的方式直接集成这些MCP服务。

通过MCP服务，可以让大模型不再只是扮演一个大脑的角色，而是拥有了四肢，直接替代人的工作能力。这样看起来，MCP确实给大模型带来了一次质的飞跃。但是，事实情况真的是这样吗？

作为纯小白，或许你只能跟在别人身后，听风就是雨。但是，作为程序员，我们是有能力更全面的去理解MCP的。

## 二、详细梳理MCP工作机制

MCP协议，全程Model Context Protocol，中文翻译是模型上下文协议。MCP协议是由Anthropic公司提出的，是一个专门用于与AI大语言模型进行交互的协议。但是本质上，MCP协议只是允许应用程序以一种统一的方式向大语言模型提供Function call函数调用。而Function call是很多大语言模型自身就提供的一种功能机制。

例如，通过下面的案例，我们就可以很简单的给大模型提供路线规划的能力。

```
from config.load_key import load_key
from langchain_openai import ChatOpenAI
# 构建阿里云百炼大模型客户端
llm = ChatOpenAI(
    model="qwen-plus",
    base_url="https://dashscope.aliyuncs.com/compatible-mode/v1",
    openai_api_key=load_key("BAILIAN_API_KEY"),
)

import datetime
from langchain.tools import tool

# 定义工具 注意要添加注释
@tool(description="规划行车路线")
def get_route_plan(origin_city:str,target_city:str):
    """规划行车路线
    Args:
        origin_city: 出发城市
        target_city: 目标城市
    """
    result = f"从城市 {origin_city} 出发，到目标城市 {target_city} ,使用意念传送，只需要三分钟即可到达。"
    print(">>>> get_route_plan >>>>"+result)
    return result

# 大模型绑定工具
llm_with_tools = llm.bind_tools([get_route_plan])

# 工具容器
all_tools = {"get_route_plan": get_route_plan}

# 把所有消息存到一起
query = "帮我规划一条从长沙到桂林的自驾路线"
messages = [query]

# 询问大模型。大模型会判断需要调用工具，并返回一个工具调用请求
ai_msg = llm_with_tools.invoke(messages)
print(ai_msg)
messages.append(ai_msg)

# 打印需要调用的工具
```



```
print(ai_msg.tool_calls)
if ai_msg.tool_calls:
    for tool_call in ai_msg.tool_calls:
        selected_tool = all_tools[tool_call["name"].lower()]
        tool_msg = selected_tool.invoke(tool_call)
        messages.append(tool_msg)
llm_with_tools.invoke(messages).content
```

询问大模型嘛。每次询问的结果都是不一样的。这里一次典型的执行结果是这样的：

```
'看起来我们找到了一种非常规的交通方式——意念传送，按照这种方法，从长沙到桂林仅需三分钟。但请注意，这并不适用于常规自驾旅行。

对于实际的自驾路线，您将需要遵循以下大致路径：

1. 从长沙出发，进入长韶娄高速。
2. 转入沪昆高速，随后进入包茂高速。
3. 沿着包茂高速行驶，直至抵达桂林。

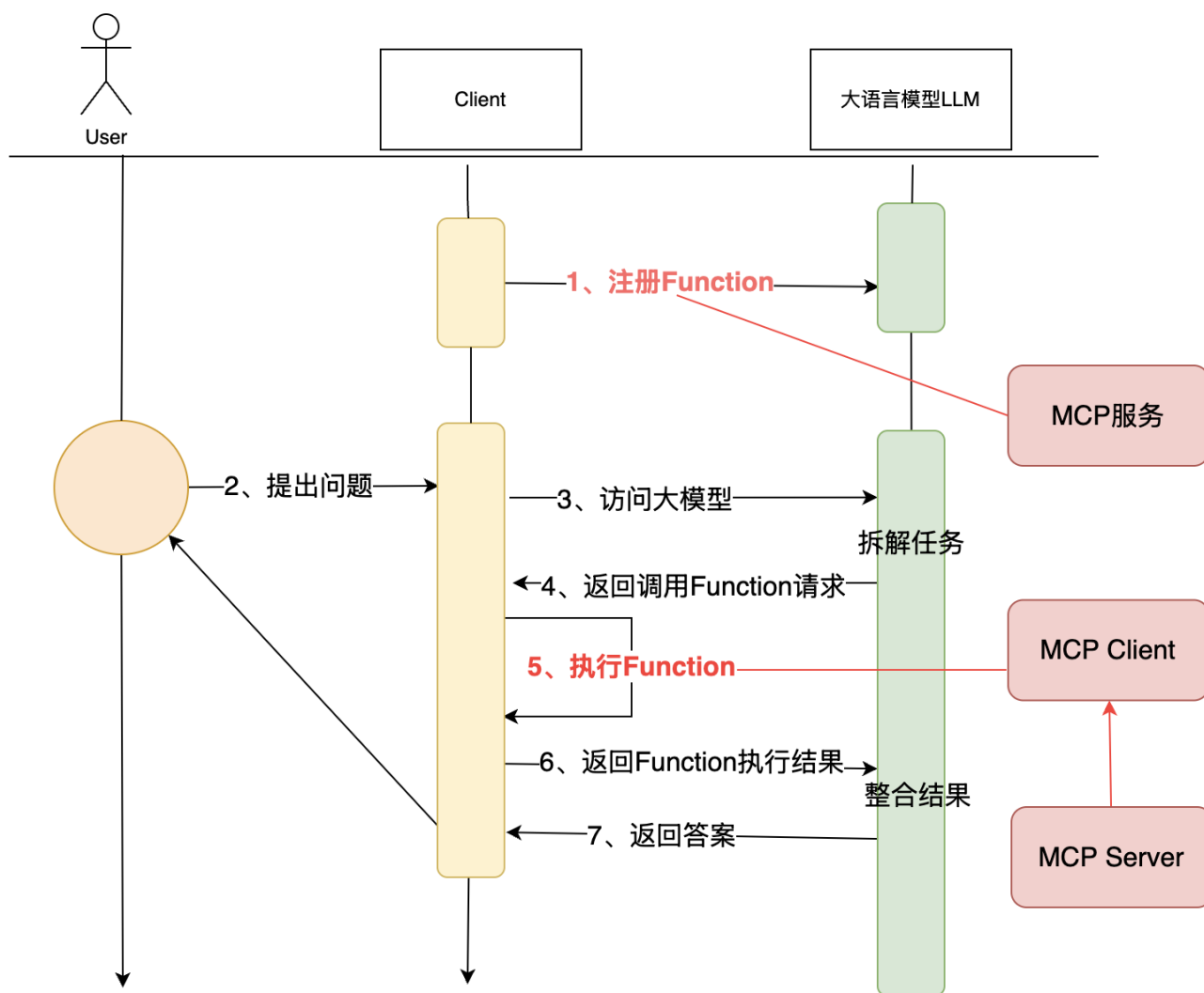
全程大约800公里，预计耗时8-9小时，具体时间取决于实际路况和您的休息次数。请确保在出发前检查车辆状况，并规划好途中的休息点。安全驾驶！如果您需要更详细的道路指引，请告知我，我会提供更精确的信息。'
```

这个案例的工作效果和MCP是如出一辙的。只不过，高德의MCP服务给的数据比较靠谱，所以大模型直接采纳了他的数据，而我们给出的意念传送的数据显然不是很靠谱，所以大模型虽然参考了我们给的方案，但是最终并没有全部采纳我们的结果。

从这里也能看出这种工具机制在处理一些具体问题时暴露出的一些核心问题。就是要不要调用工具，本身是大模型说了算。而最后不管工具给出什么样的答案，最终大模型回复什么内容，也还是大模型说了算。至于工具执行过程当中做了什么事情，大模型完全不管。

了解了这个案例后，就可以看出，MCP其实只是大模型工作机制的一种应用层的协议。





MCP协议虽然极具大模型的应用特色，但是本质上，MCP协议本身不包含任何具体的工具实现。他只是用协议的形式规定了应用程序如何向大模型提供函数调用的能力。

至于为什么MCP服务使用起来这么简单，其实是Cline这样的工具封装了客户端的实现能力。这是工具的一种简化实现，和MCP协议本身是没有太大关系的。

这个关系就好像我们以往使用HTTP协议访问各种各样的网站一样的。我们这些普通人，可以在完全不用了解HTTP协议是什么东东，只要简单的使用浏览器就可以访问网站。但是，这并不意味着HTTP就是一个简单的协议。在HTTP协议层面，要考虑的问题也肯定不能只是简单的保证数据传输，还需要对网络传输的规范性、安全性等等各个方面做出很多的设计。

从这个功能层面上来说，MCP协议和HTTP协议本质上是相同的。他基于大模型的Function Call工具实现，只不过是通过协议的方式定义了这些工具要如何工作，这样可以极大的提升各种工具的复用能力。但是，**作为一个协议，MCP要考虑的事情，同样不应该只是考虑这些工具功能如何实现，还需要在各个方面保证这些工具，不会出乱子。**

但是事实是怎样的呢？接下来我们再来梳理一下MCP协议的两种实现方式 SSE和STDIO。

### 三、拆解MCP的两种实现方式：SSE和STDIO

MCP还没听明白，怎么又冒出SSE和STDIO了？不用担心，还是老规矩，直接上实例。

还是以我们之前使用的高德地图的MCP服务为例。在高德地图开放平台的介绍中，提供了两种接入高德地图的MCP配置方式。

一种是我们之前使用过的。配置一个网站地址。这就是典型的SSE实现机制。

```
{
  "mcpServers": {
    "amap-amap-sse": {
      "url": "https://mcp.amap.com/sse?key=你在高德官网上申请的key"
    }
  }
}
```

另一种，没有使用过的STDIO的配置方式是这样的：

```
"amap-maps": {
  "command": "npx",
  "args": [
    "-y",
    "@amap/amap-maps-mcp-server"
  ],
  "env": {
    "AMAP_MAPS_API_KEY": "高德地图key"
  }
}
```

这种方式配置方式需要在本地安装Nodejs。很显然是通过在本机执行npx指令，执行了一个应用程序，从而获得高德地图的数据。

至于这个数据是如何获得的？是调用远端高德地图的服务获得的？还是读取本地某个神秘文件获取的？这就只有在高德地图提供的Nodejs源码@amap/amap-maps-mcp-server中才能知道了。

从这个案例中我们就能理解出SSE和STDIO到底是怎样的工作机制。

- SSE：其实是一种基于HTTP协议实现的长连接协议，只不过SSE协议是一种从服务端向客户端单向推送数据的长连接协议。也就是高德地图需要提供一个HTTP服务，然后客户端可以和这个HTTP服务建立一个长连接，这样客户端就可以不断的访问高德地图的HTTP服务，获得高德地图的服务数据。这时候的工作机制其实和以往我们熟悉的基于HTTP的工作机制本质上是很像的。只是服务端的性能压力会大一点而已。
- STDIO：这种工作机制的本质是在客户端本地执行一个应用程序，然后通过应用程序获得对应的结果。这时候MCP的核心问题就出来了。**MCP的服务是由MCP的服务提供者设计的，但是执行却是在客户端的机器执行。**也就是说，这给服务提供者提供了一种操作客户端机器的机会。这里面会带来多少安全问题？修改一下你本地的文件，或者给你植入一个病毒程序，或者。。。。大家可以发挥一下自己的想象。

## 四、Agent接入MCP服务

LangChain中提供了一个新的功能模块 langchain-mcp-adapters 来支持MCP服务

```
# 安装对应依赖
!pip install langchain-mcp-adapters
```

Looking in indexes: <https://mirrors.aliyun.com/pypi/simple/>  
Requirement already satisfied: langchain-mcp-adapters in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (0.1.1)  
Requirement already satisfied: langchain-core<0.4,>=0.3.36 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from langchain-mcp-  
adapters) (0.3.51)  
Requirement already satisfied: mcp>=1.7 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from langchain-mcp-  
adapters) (1.9.1)  
Requirement already satisfied: langsmith<0.4,>=0.1.125 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from langchain-  
core<0.4,>=0.3.36->langchain-mcp-adapters) (0.3.18)  
Requirement already satisfied: tenacity!=8.4.0,<10.0.0,>=8.1.0 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from langchain-  
core<0.4,>=0.3.36->langchain-mcp-adapters) (9.0.0)  
Requirement already satisfied: jsonpatch<2.0,>=1.33 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from langchain-  
core<0.4,>=0.3.36->langchain-mcp-adapters) (1.33)  
Requirement already satisfied: PyYAML>=5.3 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from langchain-  
core<0.4,>=0.3.36->langchain-mcp-adapters) (6.0.2)  
Requirement already satisfied: packaging<25,>=23.2 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from langchain-  
core<0.4,>=0.3.36->langchain-mcp-adapters) (24.2)  
Requirement already satisfied: typing-extensions>=4.7 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from langchain-  
core<0.4,>=0.3.36->langchain-mcp-adapters) (4.12.2)  
Requirement already satisfied: pydantic<3.0.0,>=2.5.2 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from langchain-  
core<0.4,>=0.3.36->langchain-mcp-adapters) (2.10.6)  
Requirement already satisfied: anyio>=4.5 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from mcp>=1.7->langchain-  
mcp-adapters) (4.9.0)  
Requirement already satisfied: httpx-sse>=0.4 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from mcp>=1.7->langchain-  
mcp-adapters) (0.4.0)  
Requirement already satisfied: httpx>=0.27 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from mcp>=1.7->langchain-  
mcp-adapters) (0.28.1)  
Requirement already satisfied: pydantic-settings>=2.5.2 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from mcp>=1.7->langchain-  
mcp-adapters) (2.8.1)  
Requirement already satisfied: python-multipart>=0.0.9 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from mcp>=1.7->langchain-  
mcp-adapters) (0.0.20)  
Requirement already satisfied: sse-starlette>=1.6.1 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from mcp>=1.7->langchain-  
mcp-adapters) (1.8.2)  
Requirement already satisfied: starlette>=0.27 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from mcp>=1.7->langchain-  
mcp-adapters) (0.46.1)

Requirement already satisfied: uvicorn>=0.23.1 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from mcp>=1.7->langchain-mcp-adapters) (0.34.0)

Requirement already satisfied: exceptiongroup>=1.0.2 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from anyio>=4.5->mcp>=1.7->langchain-mcp-adapters) (1.2.2)

Requirement already satisfied: idna>=2.8 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from anyio>=4.5->mcp>=1.7->langchain-mcp-adapters) (3.10)

Requirement already satisfied: sniffio>=1.1 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from anyio>=4.5->mcp>=1.7->langchain-mcp-adapters) (1.3.1)

Requirement already satisfied: certifi in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from httpx>=0.27->mcp>=1.7->langchain-mcp-adapters) (2025.1.31)

Requirement already satisfied: httpcore==1.\* in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from httpx>=0.27->mcp>=1.7->langchain-mcp-adapters) (1.0.7)

Requirement already satisfied: h11<0.15,>=0.13 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from httpcore==1.\*->httpx>=0.27->mcp>=1.7->langchain-mcp-adapters) (0.14.0)

Requirement already satisfied: jsonpointer>=1.9 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from jsonpatch<2.0,>=1.33->langchain-core<0.4,>=0.3.36->langchain-mcp-adapters) (3.0.0)

Requirement already satisfied: orjson<4.0.0,>=3.9.14 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from langsmith<0.4,>=0.1.125->langchain-core<0.4,>=0.3.36->langchain-mcp-adapters) (3.10.15)

Requirement already satisfied: requests<3,>=2 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from langsmith<0.4,>=0.1.125->langchain-core<0.4,>=0.3.36->langchain-mcp-adapters) (2.32.3)

Requirement already satisfied: requests-toolbelt<2.0.0,>=1.0.0 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from langsmith<0.4,>=0.1.125->langchain-core<0.4,>=0.3.36->langchain-mcp-adapters) (1.0.0)

Requirement already satisfied: zstandard<0.24.0,>=0.23.0 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from langsmith<0.4,>=0.1.125->langchain-core<0.4,>=0.3.36->langchain-mcp-adapters) (0.23.0)

Requirement already satisfied: annotated-types>=0.6.0 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from pydantic<3.0.0,>=2.5.2->langchain-core<0.4,>=0.3.36->langchain-mcp-adapters) (0.7.0)

Requirement already satisfied: pydantic-core==2.27.2 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from pydantic<3.0.0,>=2.5.2->langchain-core<0.4,>=0.3.36->langchain-mcp-adapters) (2.27.2)

Requirement already satisfied: python-dotenv>=0.21.0 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from pydantic-settings>=2.5.2->mcp>=1.7->langchain-mcp-adapters) (1.0.1)

Requirement already satisfied: fastapi in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from sse-starlette>=1.6.1->mcp>=1.7->langchain-mcp-adapters) (0.115.12)

Requirement already satisfied: click>=7.0 in  
/opt/anaconda3/envs/langchain\_env/lib/python3.10/site-packages (from uvicorn>=0.23.1->mcp>=1.7->langchain-mcp-adapters) (8.1.8)

```
Requirement already satisfied: charset-normalizer<4,>=2 in
/opt/anaconda3/envs/langchain_env/lib/python3.10/site-packages (from requests<3,>=2-
>langsmith<0.4,>=0.1.125->langchain-core<0.4,>=0.3.36->langchain-mcp-adapters) (3.4.1)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/opt/anaconda3/envs/langchain_env/lib/python3.10/site-packages (from requests<3,>=2-
>langsmith<0.4,>=0.1.125->langchain-core<0.4,>=0.3.36->langchain-mcp-adapters) (2.3.0)
```

接下来接入MCP服务也相当简单，只要增加MCP的配置文件就行。

```
from langchain_mcp_adapters.client import MultiServerMCPClient
from langgraph.prebuilt import create_react_agent
from config.load_key import load_key

from langchain_community.chat_models import ChatTongyi
# 构建阿里云百炼大模型客户端
llm = ChatTongyi(
    model="qwen-plus",
    api_key=load_key("BAILIAN_API_KEY"),
)
# 相比Cline客户端配置，只要增加transport属性即可。不过测试streamable_http有问题。不知道是不是版本的原因。
client = MultiServerMCPClient(
    {
        "amap-amap-sse": {
            "url": "https://mcp.amap.com/sse?key=451ad40d0e39453600f2a305e31eabe4",
            "transport": "streamable_http"
        },
        # "amap-maps": {
        #     "command": "npx",
        #     "args": [
        #         "-y",
        #         "@amap/amap-maps-mcp-server"
        #     ],
        #     "env": {
        #         "AMAP_MAPS_API_KEY": "451ad40d0e39453600f2a305e31eabe4"
        #     },
        #     "transport": "stdio"
        # }
    }
)
tools = await client.get_tools()
agent = create_react_agent(
    model=llm,
    tools=tools
)
response = await agent.ainvoke(
    {"messages": [{"role": "user", "content": "帮我规划一条从长沙梅溪湖到溁湾镇的自驾路线"]}}
)
response
```

+ Exception Group Traceback (most recent call last):

```

| File "/opt/anaconda3/envs/langchain_env/lib/python3.10/site-
packages/IPython/core/interactiveshell.py", line 3577, in run_code
|     await eval(code_obj, self.user_global_ns, self.user_ns)
| File
"/var/folders/6d/_yp_wgds4_7bh0yx1w8q62g80000gn/T/ipykernel_3273/1147480625.py", line 31,
in <module>
|     tools = await client.get_tools()
| File "/opt/anaconda3/envs/langchain_env/lib/python3.10/site-
packages/langchain_mcp_adapters/client.py", line 132, in get_tools
|     tools = await load_mcp_tools(None, connection=connection)
| File "/opt/anaconda3/envs/langchain_env/lib/python3.10/site-
packages/langchain_mcp_adapters/tools.py", line 105, in load_mcp_tools
|     async with create_session(connection) as tool_session:
| File "/opt/anaconda3/envs/langchain_env/lib/python3.10/contextlib.py", line 217, in
__aexit__
|     await self.gen.athrow(typ, value, traceback)
| File "/opt/anaconda3/envs/langchain_env/lib/python3.10/site-
packages/langchain_mcp_adapters/sessions.py", line 265, in create_session
|     async with _create_streamable_http_session(
| File "/opt/anaconda3/envs/langchain_env/lib/python3.10/contextlib.py", line 217, in
__aexit__
|     await self.gen.athrow(typ, value, traceback)
| File "/opt/anaconda3/envs/langchain_env/lib/python3.10/site-
packages/langchain_mcp_adapters/sessions.py", line 198, in _create_streamable_http_session
|     async with streamablehttp_client(
| File "/opt/anaconda3/envs/langchain_env/lib/python3.10/contextlib.py", line 217, in
__aexit__
|     await self.gen.athrow(typ, value, traceback)
| File "/opt/anaconda3/envs/langchain_env/lib/python3.10/site-
packages/mcp/client/streamable_http.py", line 463, in streamablehttp_client
|     async with anyio.create_task_group() as tg:
| File "/opt/anaconda3/envs/langchain_env/lib/python3.10/site-
packages/anyio/_backends/_asyncio.py", line 772, in __aexit__
|     raise BaseExceptionGroup(
| exceptiongroup.ExceptionGroup: unhandled errors in a TaskGroup (1 sub-exception)
+-+----- 1 -----
| Exception Group Traceback (most recent call last):
| File "/opt/anaconda3/envs/langchain_env/lib/python3.10/site-
packages/mcp/client/streamable_http.py", line 491, in streamablehttp_client
|     yield (
| File "/opt/anaconda3/envs/langchain_env/lib/python3.10/site-
packages/langchain_mcp_adapters/sessions.py", line 201, in _create_streamable_http_session
|     async with ClientSession(read, write, **(session_kwargs or {})) as session:
| File "/opt/anaconda3/envs/langchain_env/lib/python3.10/site-
packages/anyio/_backends/_asyncio.py", line 772, in __aexit__
|     raise BaseExceptionGroup(
| exceptiongroup.ExceptionGroup: unhandled errors in a TaskGroup (1 sub-exception)
+-+----- 1 -----
| Traceback (most recent call last):
| File "/opt/anaconda3/envs/langchain_env/lib/python3.10/site-
packages/langchain_mcp_adapters/sessions.py", line 202, in _create_streamable_http_session
|     yield session

```

```
| File "/opt/anaconda3/envs/langchain_env/lib/python3.10/site-  
packages/langchain_mcp_adapters/sessions.py", line 274, in create_session  
|     yield session  
| File "/opt/anaconda3/envs/langchain_env/lib/python3.10/site-  
packages/langchain_mcp_adapters/tools.py", line 106, in load_mcp_tools  
|     await tool_session.initialize()  
| File "/opt/anaconda3/envs/langchain_env/lib/python3.10/site-  
packages/mcp/client/session.py", line 127, in initialize  
|     result = await self.send_request(  
| File "/opt/anaconda3/envs/langchain_env/lib/python3.10/site-  
packages/mcp/shared/session.py", line 294, in send_request  
|     raise McpError(response_or_error.error)  
| mcp.shared.exceptions.McpError: Session terminated  
+-----
```

## 五、手写实现一个MCP服务

如何实现一个MCP服务？MCP的官网上提供了一系列的SDK来辅助实现MCP的客户端和服务端。官网地址：<https://modelcontextprotocol.io/introduction>





# Model Context Protocol



Python SDK

TypeScript SDK

Java SDK

Kotlin SDK

C# SDK

那么接下来的事情就简单了。我们用python客户端来实现一个MCP服务看看。

## 1、手写SSE实现

首先需要安装MCP依赖

```
pip install mcp
```

然后，就可以参照官网案例，快速实现一个MCP服务

```
from mcp.server.fastmcp import FastMCP

mcp = FastMCP("roymcpdemo")

@mcp.tool()
def add(a: int, b: int) -> int :
```

```

    """Add two numbers together."""
    print(f"roy mcp demo called : add({a}, {b})")
    return a + b

@mcp.tool()
def weather(city: str):
    """获取某个城市的天气
        Args:
            city: 具体城市
        """
    return "城市" + city + ",今天天气不错"

@mcp.resource("greeting://{name}")
def greeting(name: str) -> str:
    """Greet a person by name."""
    print(f"roy mcp demo called : greeting({name})")
    return f"Hello, {name}!"

if __name__ == "__main__":
    # 以sse协议暴露服务。
    mcp.run(transport='sse')
    # 以stdio协议暴露服务。
    # mcp.run(transport='stdio')

```

这里就用@mcp.tool()注解，快速声明了两个服务

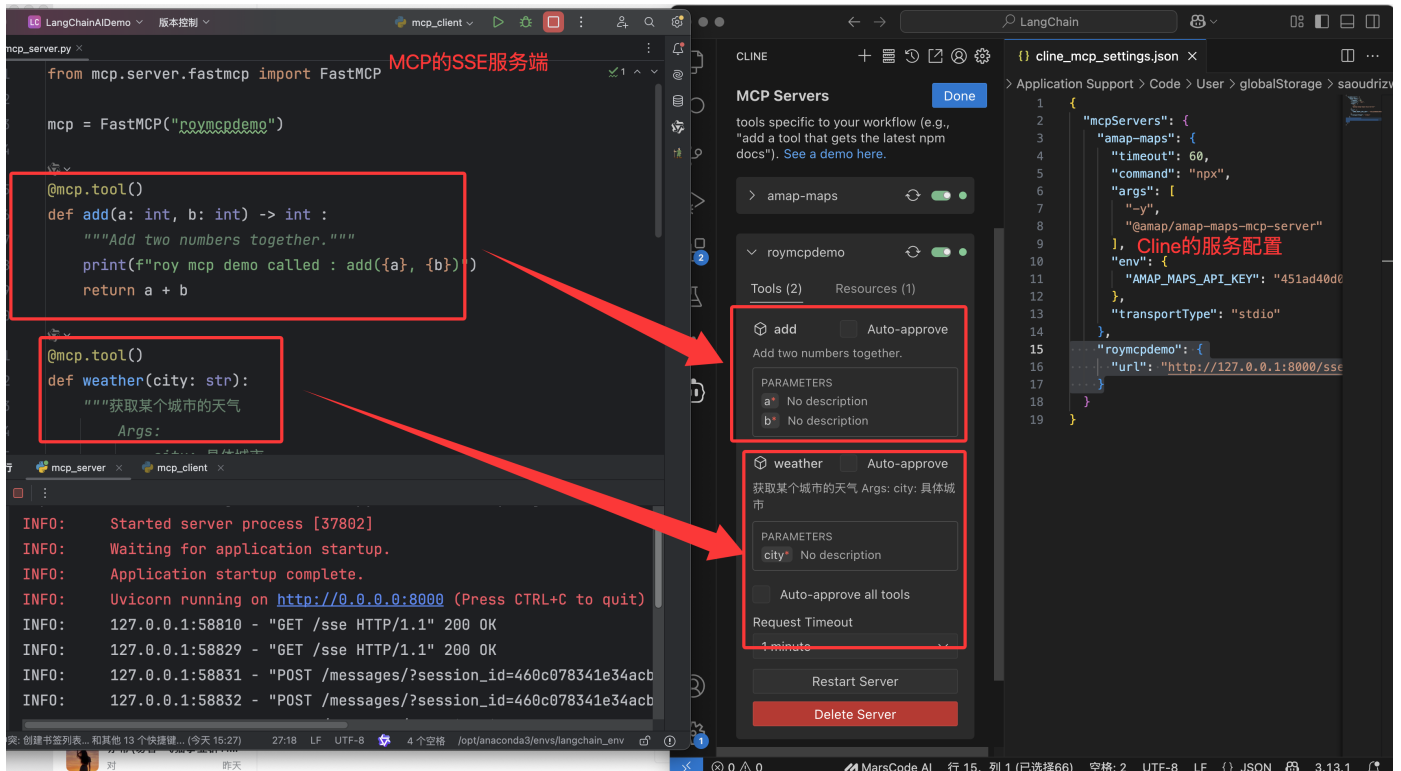
执行这个python代码后，就可以启动一个服务。接下来，就可以在cline客户端中配置对应的客户端服务了。

```

"roymcpdemo": {
    "url": "http://127.0.0.1:8000/sse"
}

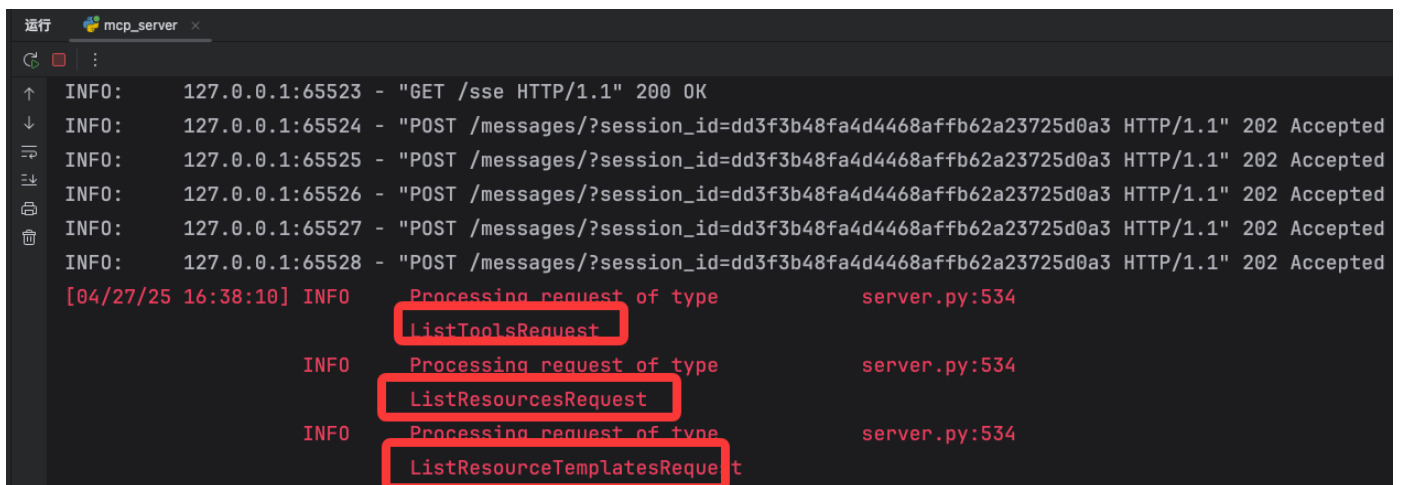
```

配置完成后，就可以在Cline中看到我们声明的工具和资源了。



这就是按照MCP的SSE提供的一种服务实现。

另外，跟踪服务端的日志，会发现，Cline客户端之所以能够发现这些服务，是因为发送了一些请求，获取到工具声明信息的。



## 2、切换到STDIO实现

要切换到STDIO的协议，也很简单。对于服务端，只需要修改最后一行代码。

```
if __name__ == "__main__":  
    # 以sse协议暴露服务。  
    #mcp.run(transport='sse')  
    # 以stdio协议暴露服务。  
    mcp.run(transport='stdio')
```

这时候，就需要一个客户端程序，来后去服务端的这些功能。整体上，也还是处理这几个请求。

```

from mcp import StdioServerParameters, stdio_client, ClientSession
import mcp.types as types

server_params = StdioServerParameters(
    command="python",
    args=["/Users/roykingw/Desktop/a-
work/LangChain/LangChainAIDemo/src/mcpdemo/mcp_server.py"],
    env=None
)

async def handle_sampling_message(message: types.CreateMessageRequestParams) ->
types.CreateMessageResult :
    print(f"sampling message: {message}")
    return types.CreateMessageResult(
        role="assistant",
        content=types.TextContent(
            type="text",
            text="Hello,world! from model"
        ),
        model="qwen-plus",
        stopReason="endTurn"
    )

async def run():
    async with stdio_client(server_params) as (read,write):
        async with ClientSession(read,write,sampling_callback=handle_sampling_message) as
session:
            await session.initialize()

            prompts = await session.list_prompts()
            print(f"prompts: {prompts}")

            tools = await session.list_tools()
            print(f"tools: {tools}")

            resources = await session.list_resources()
            print(f"resources: {resources}")

            result = await session.call_tool("weather",{"city":"北京"})
            print(f"result: {result}")

if __name__ == "__main__":
    import asyncio
    asyncio.run(run())

```

接下来，也可以用同样的方式尝试在LangGraph中接入服务。

当然，由于需要依赖服务端的代码实现，所以不能简单的配置一个python指令执行。需要打包成nodejs依赖启动。后续就不再做介绍了。

## 六、章节总结

通过深度演示MCP服务的客户端和服务端的交互过程，让大家对MCP服务有了初步的了解。在深度使用更多的MCP服务的同时，合理的看待MCP的安全问题。虽然MCP最大的意义在于简化客户端的调用过程，让我们以极小的代价快速接入更多的外部服务，但是并不代表MCP服务就是安全的。