使用 MCP C# SDK 实现 MCP Tool

追逐时光者 2026年1月14日 07:00 广东

以下文章来源于amazingdotnet，作者WeihanLi

amazingdotnet

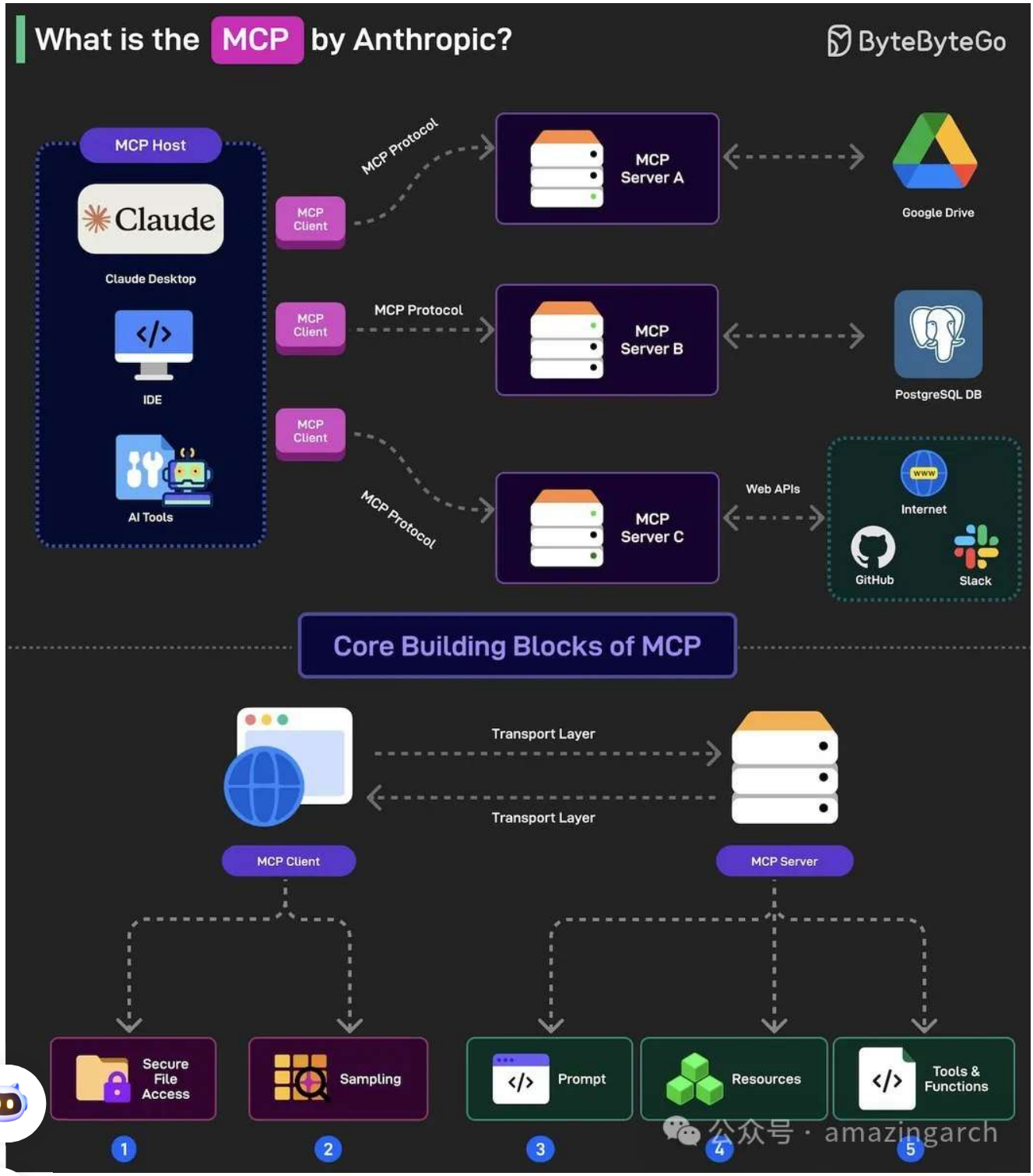
dotnet 开发知识库，了不起的 dotnet，dotnet 奇淫怪巧

使用 MCP C# SDK 实现 MCP

Intro

MCP是由Anthropic创建的一个开放协议 现在有官方 C# SDK 了，官方 C# SDK 由原来的 mcpdotnet 发展而来，基于 Microsoft.Extensions.AI 实现，截止写文章的时候（2025-4-1）目前最新版本时 0.1.0-preview 4 了，这一版本中增加了 `ModelContextProtocol.AspNetCore` NuGet 包，使得实现基于 ASP.NET Core SSE 的 Mcp Server 更加简单了，之前的版本中需要将示例中的扩展拷贝到自己项目中去，有了这个 NuGet 包之后就不需要了，McpServer 目前主要的两种实现方式是 Stdio(标准输入输出) 和 SSE（Server Sent Event)，新的规范里提出了基于 Http 的支持，目前暂时还未支持

What



SSE Sample

首先我们需要在项目里添加 NuGet 包引用，这里我们直接引用 `ModelContextProtocol.AspNetCore`

```
dotnet add package ModelContextProtocol.AspNetCore --prerelease
```

然后注册我们的 McpServer 示例如下：

```
builder.Services
```

```
builder.Services
    .AddMcpServer() // 注册 McpServer 相关服务
    .WithToolsFromAssembly() // 从程序集中扫描添加 tools
    ;
```

之后需要注册 Mcp Sse 的 Endpoint

```
var app = builder.Build();

app.MapMcp(); // 注册 mcp sse endpoint

await app.RunAsync();
```

最后我们要实现我们 Mcp tool 了，来看一个最简单的示例

```
[McpServerToolType]
public static class EchoTool
{
    [McpServerTool, Description("Echoes the input back to the client.")]
    public static string Echo(string message)
    {
        return "hello " + message;
    }
}
```

McpTool 类型需要添加 `McpServerToolType` 的 attribute，同时 tool 对应的方法也需要添加 `McpServerTool` 的 attribute，默认 tool 的名字是方法名，也可以通过 `Name` 来执行，同时可以通过 `DescriptionAttribute` 来描述这个 tool 的功能，客户端可以根据 tool 的介绍来选择合适的 tool 来实现功能

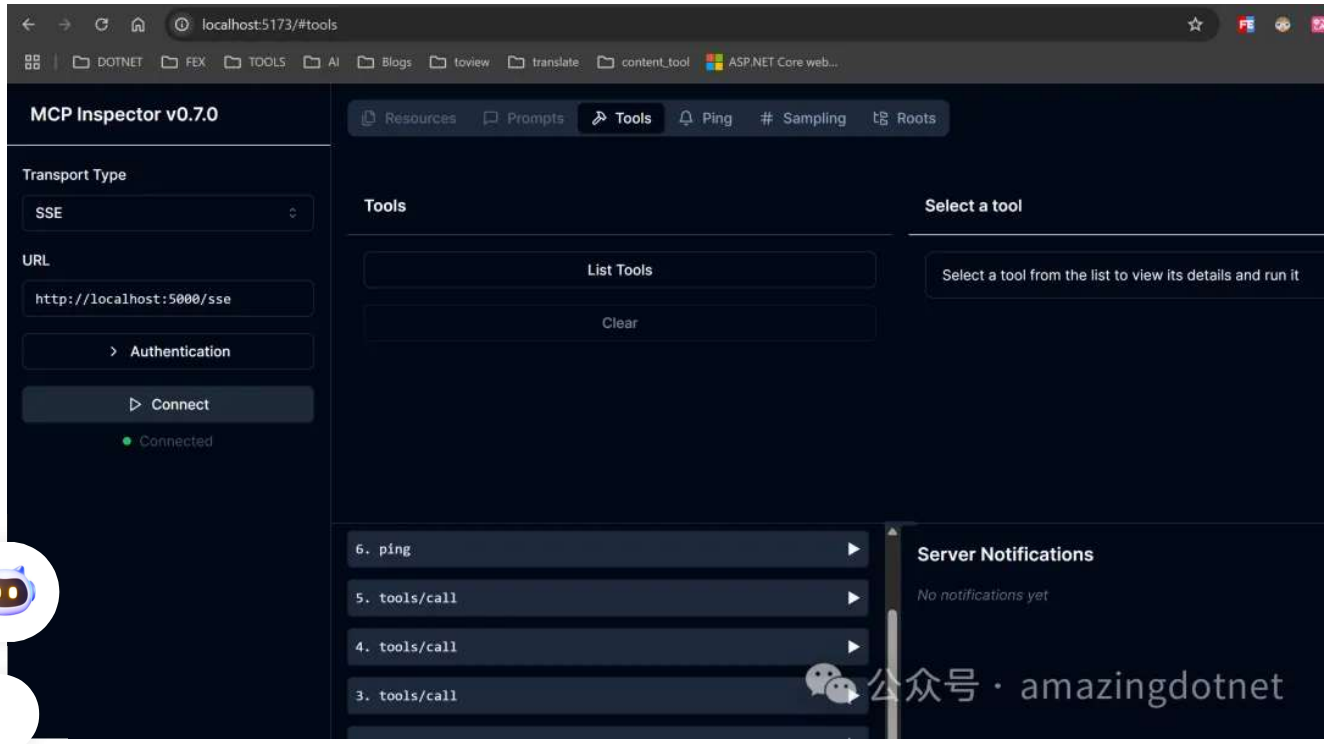
最简单的功能差不多就是这样了，最早的版本里也需要是静态方法才能注册，使得想要使用依赖注入的话可能得自己实现一个 service locator 得模式，在第二个版本中支持了简单得依赖注入，我们可以直接在方法参数中添加对应得服务，示例如下：

```
[McpServerToolType]
public class TranslationTool
{
    [McpServerTool(Name = "translation"), Description("Translate English to Simplified Chinese")]
    public static async Task<string> TranslationEnZh(
        [Description("The source english text needs to be translated to Simplified Chinese")]
```

```
            string sourceText,
            IChatClient chatClient,
            CancellationToken cancellationToken
            )
        {
            var response = await chatClient.GetResponseAsync(
                // ...
                );
            // ...
        }
    }
```

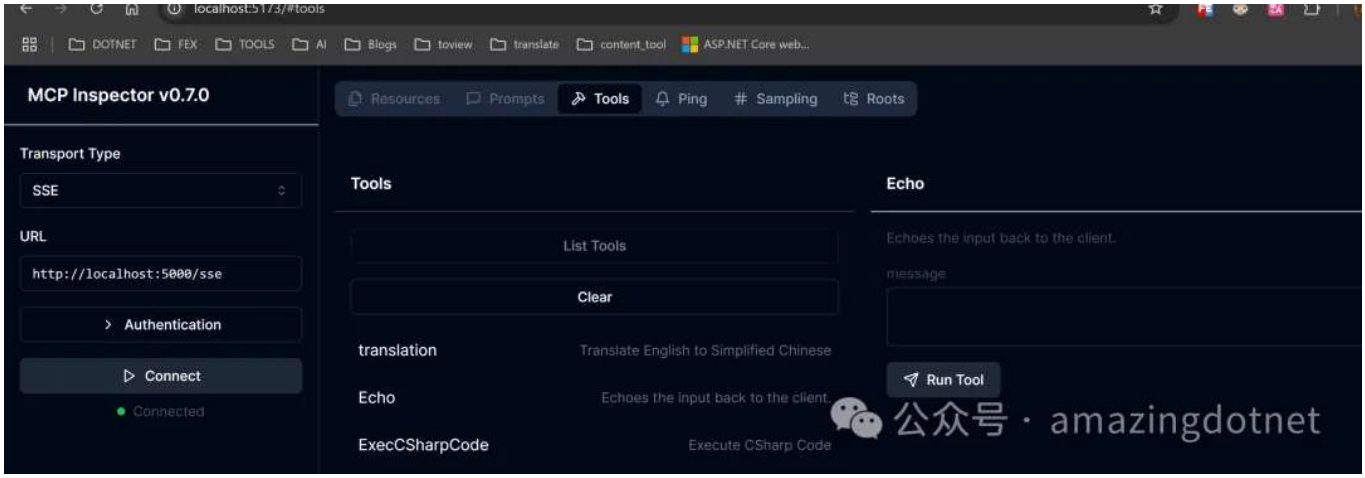
这里我们就可以跑起来我们的 McpServer 了，跑起来之后可以通过官方的 mcp inspector 来调试

可以使用 `npx @modelcontextprotocol/inspector` 来运行 mcp inspector，运行起来之后效果如下：

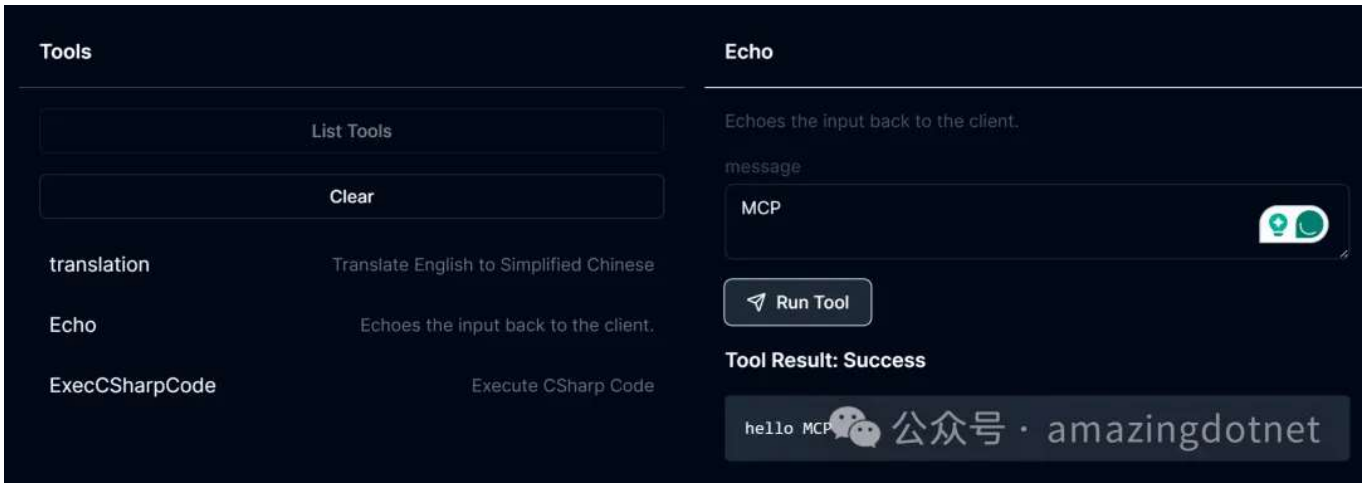


我们可以选择 Transport Type 为 SSE，URL 为我们 web server 的地址 `/sse`，默认 path 是 `sse` 然后点击 `Connect` 按钮即可

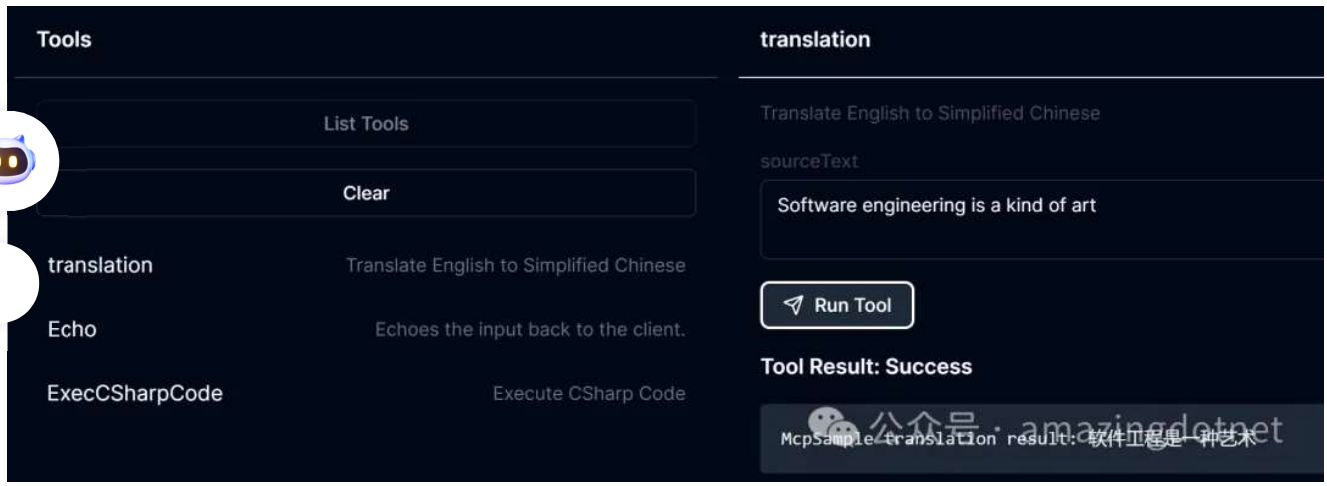
Connect 成功之后，下面会显示 `Connected`，然后我们可以点击中间的 `List Tools` 去向 server 请求支持的 tools，之后会列出来 server 支持的 tool，我们可以点击任意一个 tool 在右侧会出现对应的参数信息，我们填写之后点击 Run Tool 就可以调用 server 端的这个 tool



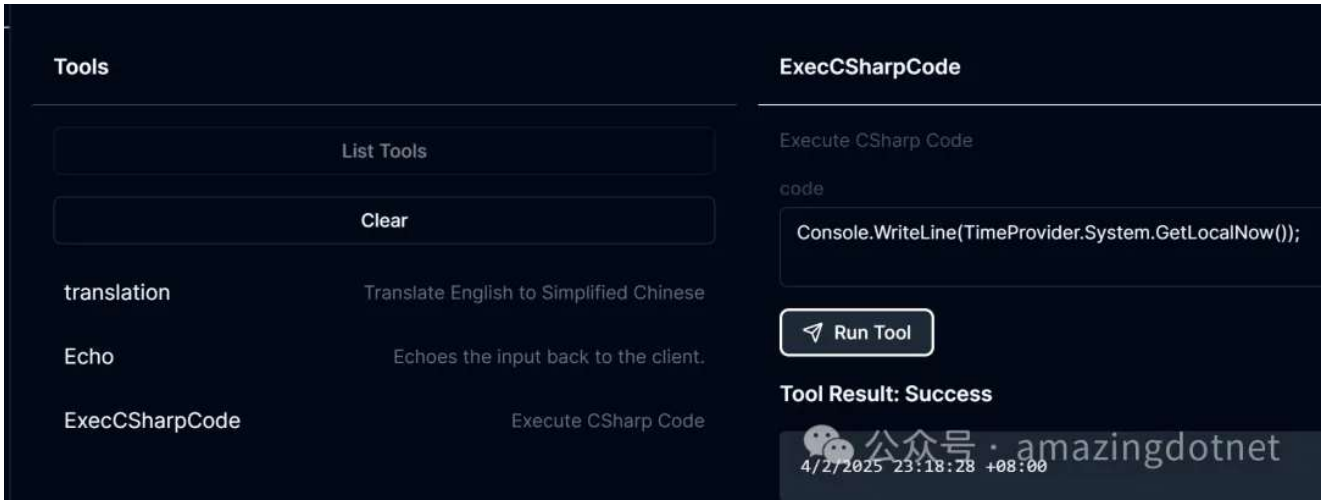
tools



echo-tool

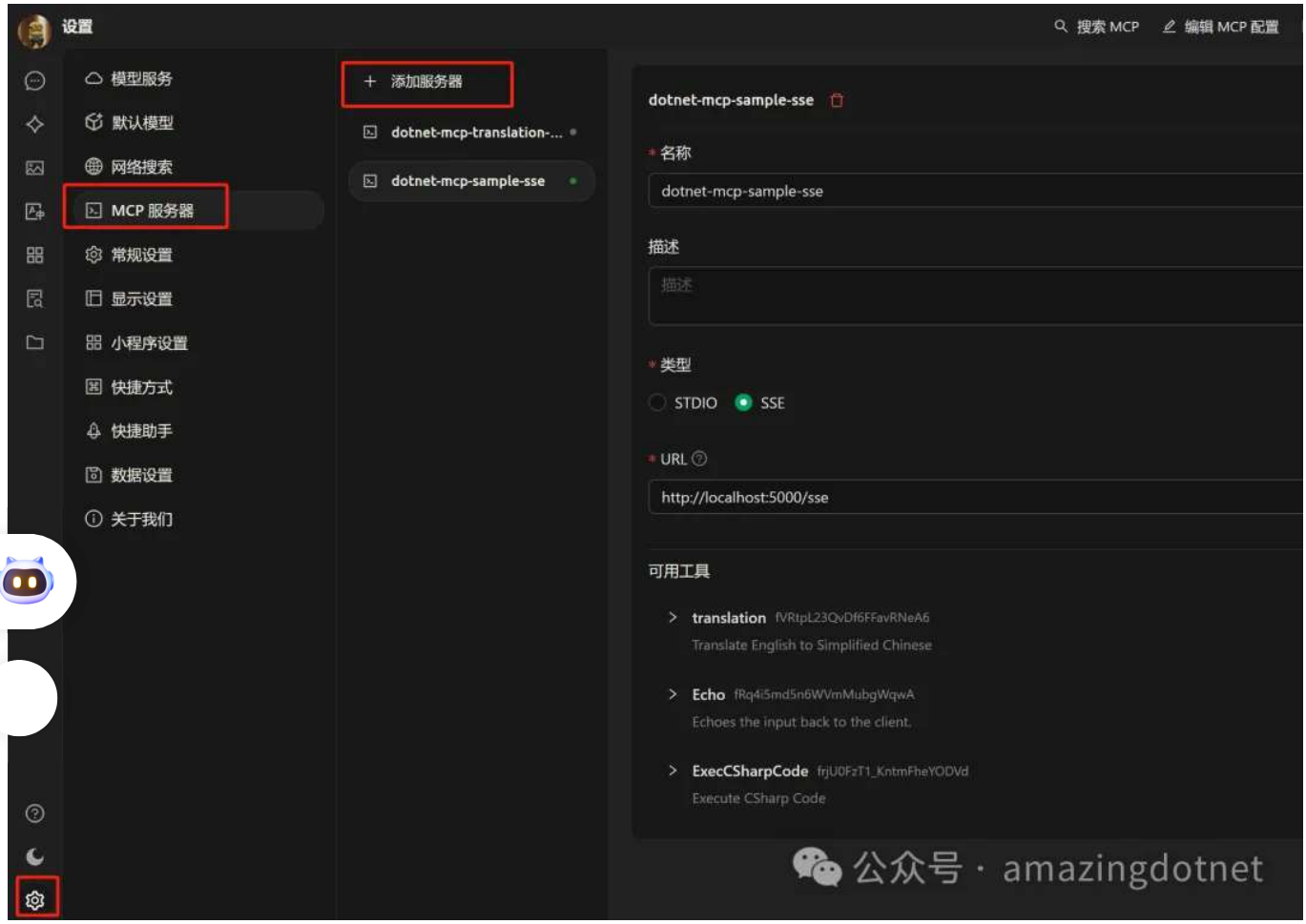


translation-tool



csharp-exec

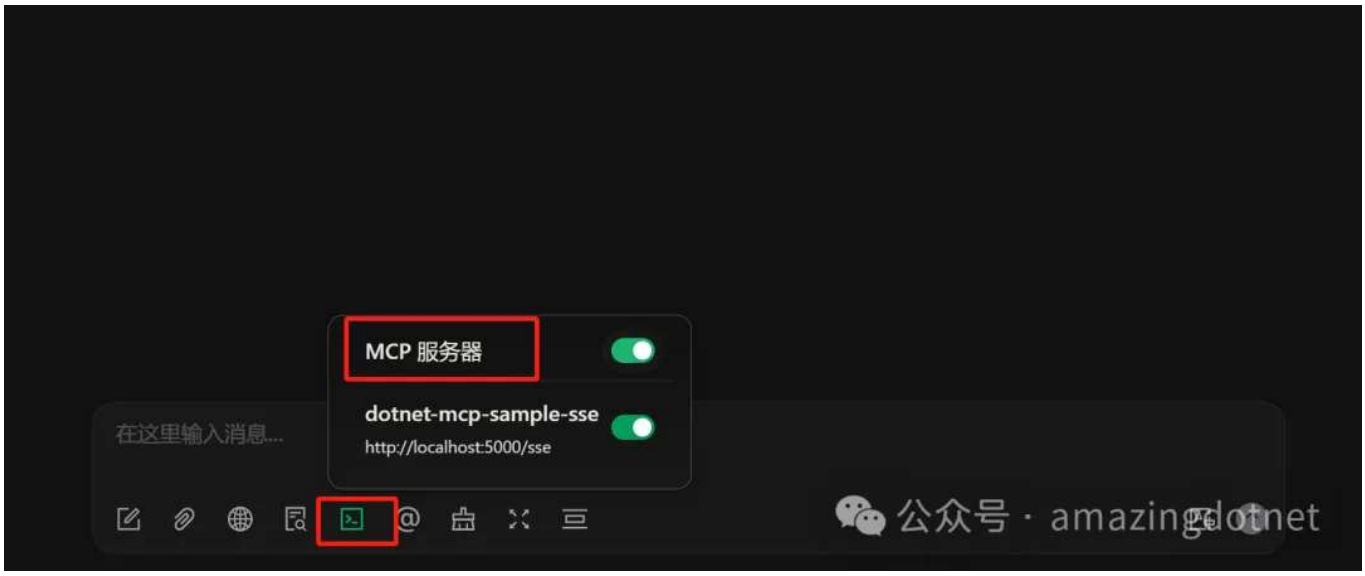
除了 mcp inspector 和 Claude 目前也有一些 chat 工具支持了 mcp tool，我目前有用 Cherry Studio，已经支持了 Mcp Tool，我们也可以在 Cheery Studio 中试一下（如果之前已经安装但是没有 Mcp Server 支持，推荐更新到最新版本）



CherryStudio McpServer Configure

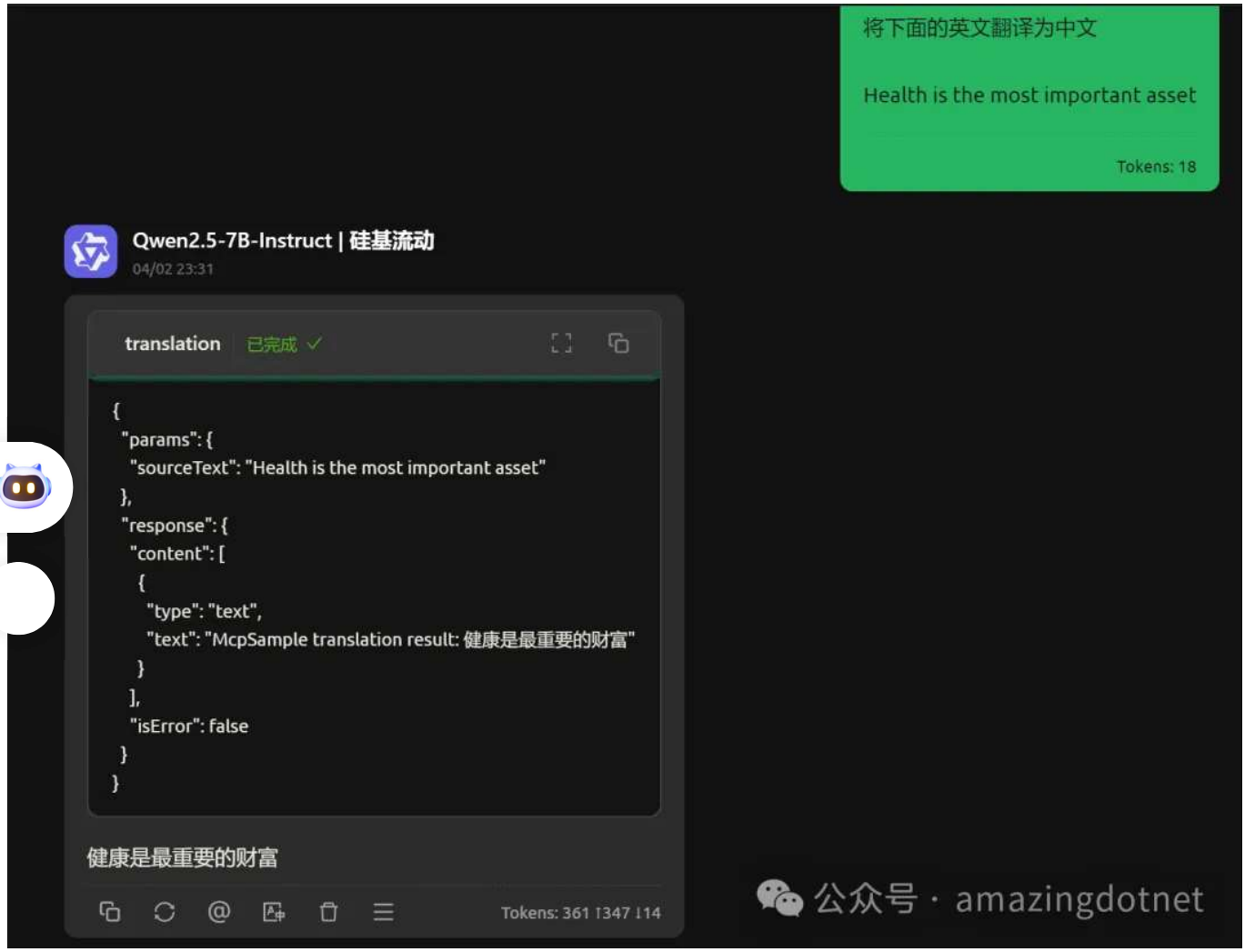
类似地，我们添加 Mcp Server 的时候可以选择 SSE 并填写刚才的 URL，最早支持 MCP 的版本没有列出来可以调用的工具，在新版本里列出来了可用的工具，更加直观和方便了

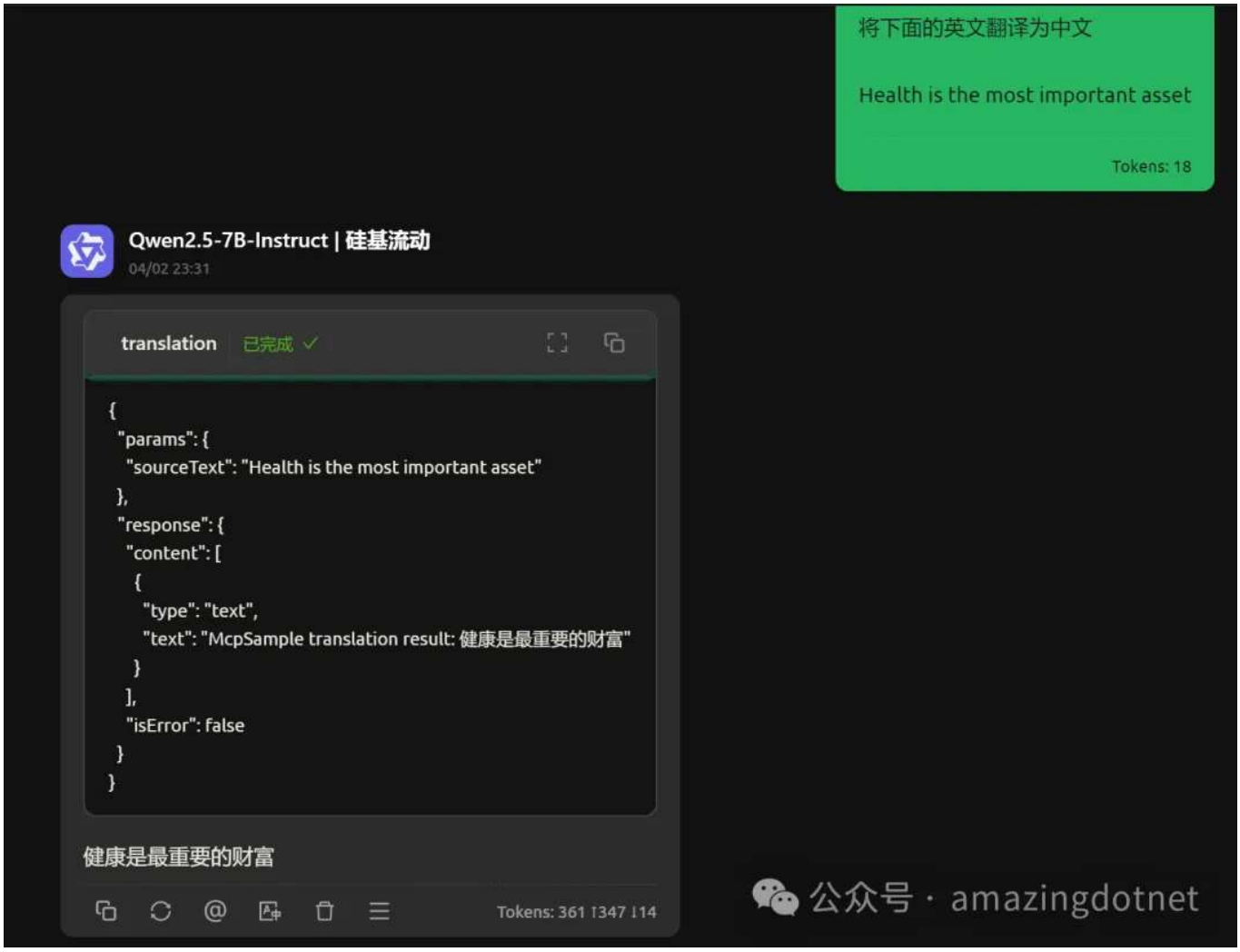
接着我们在聊天窗口中也需要启用一下 Mcp Server，默认没有启用



chat with Mcp Server

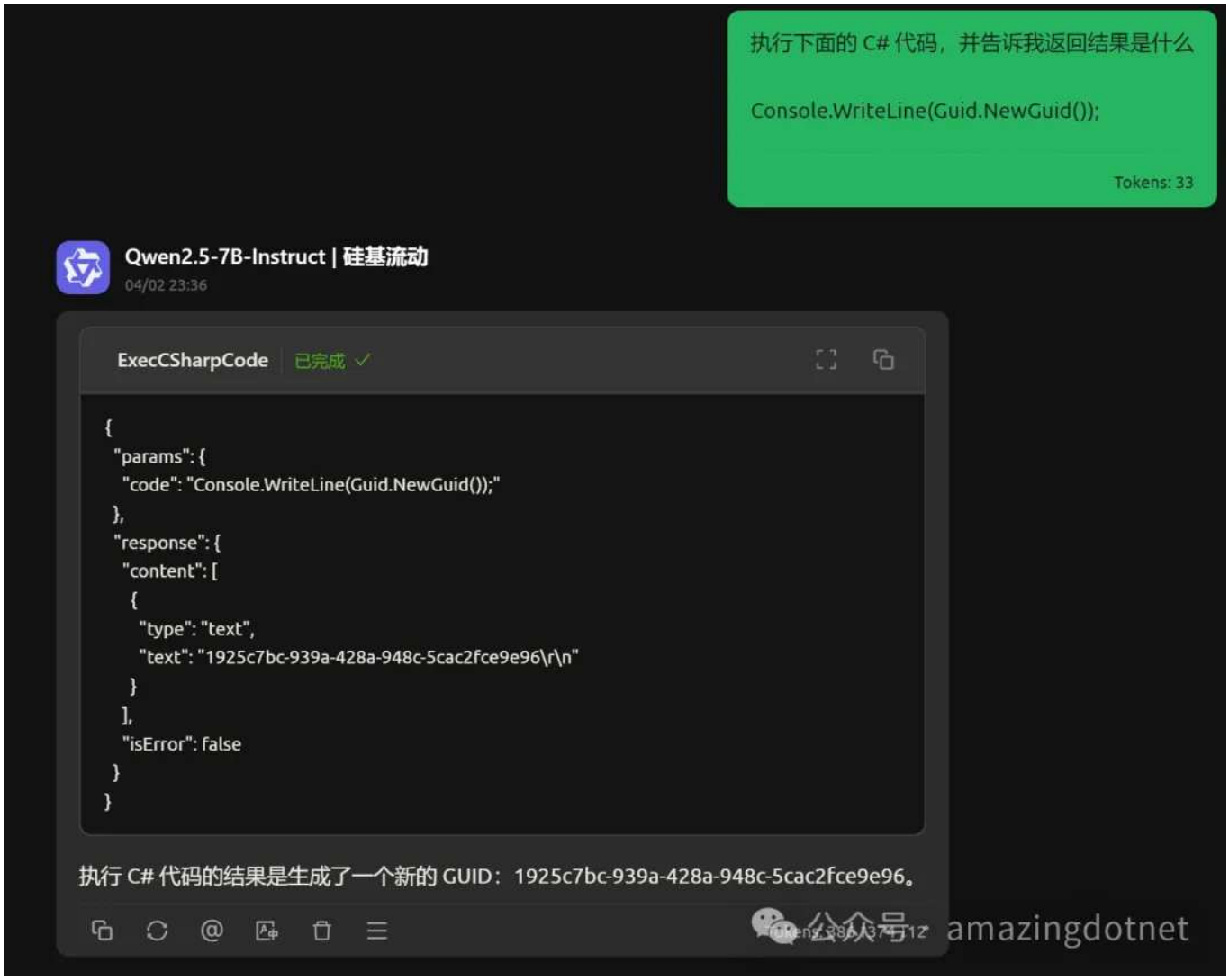
然后就可以测试我们的 Mcp Server 了

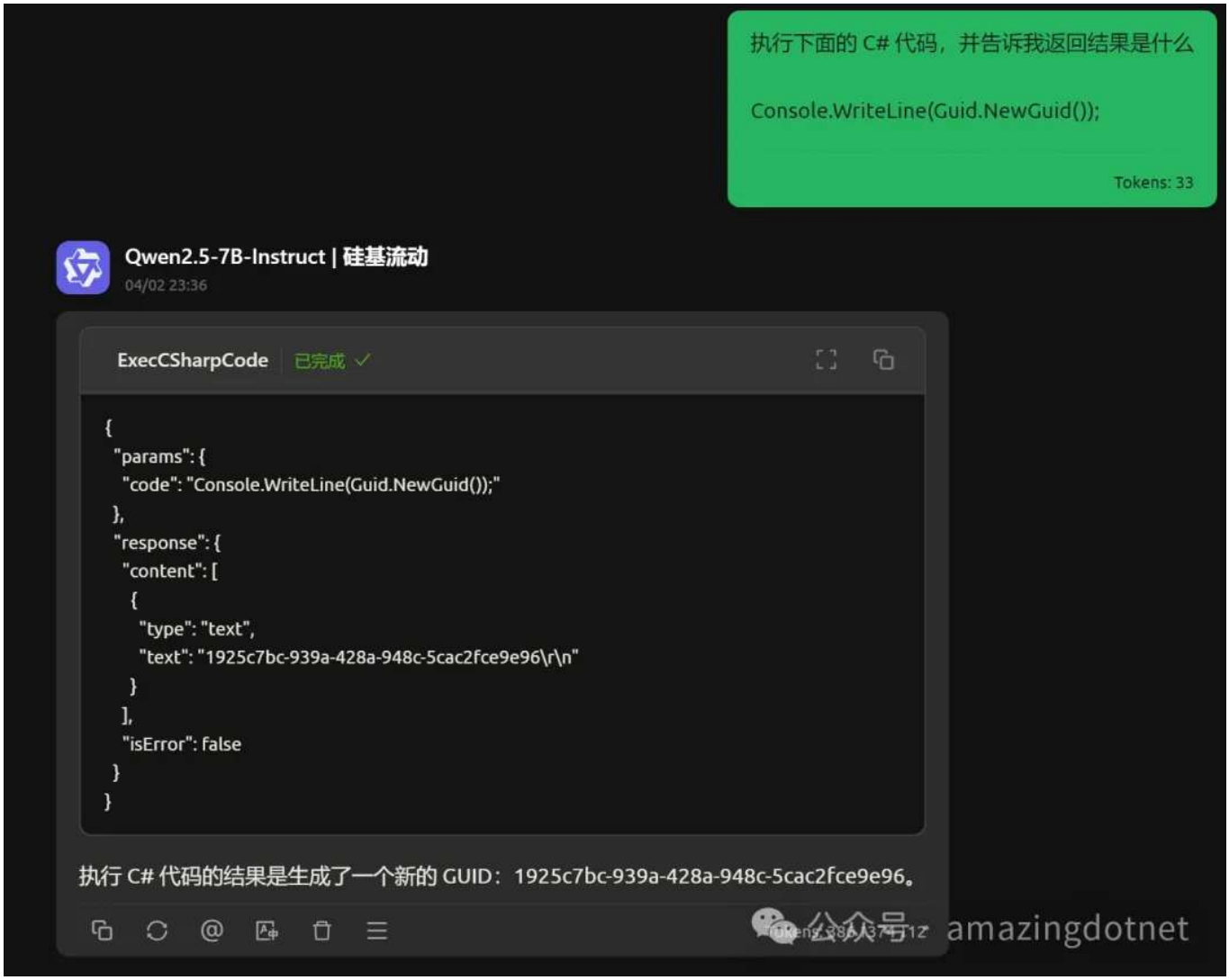




chery translation

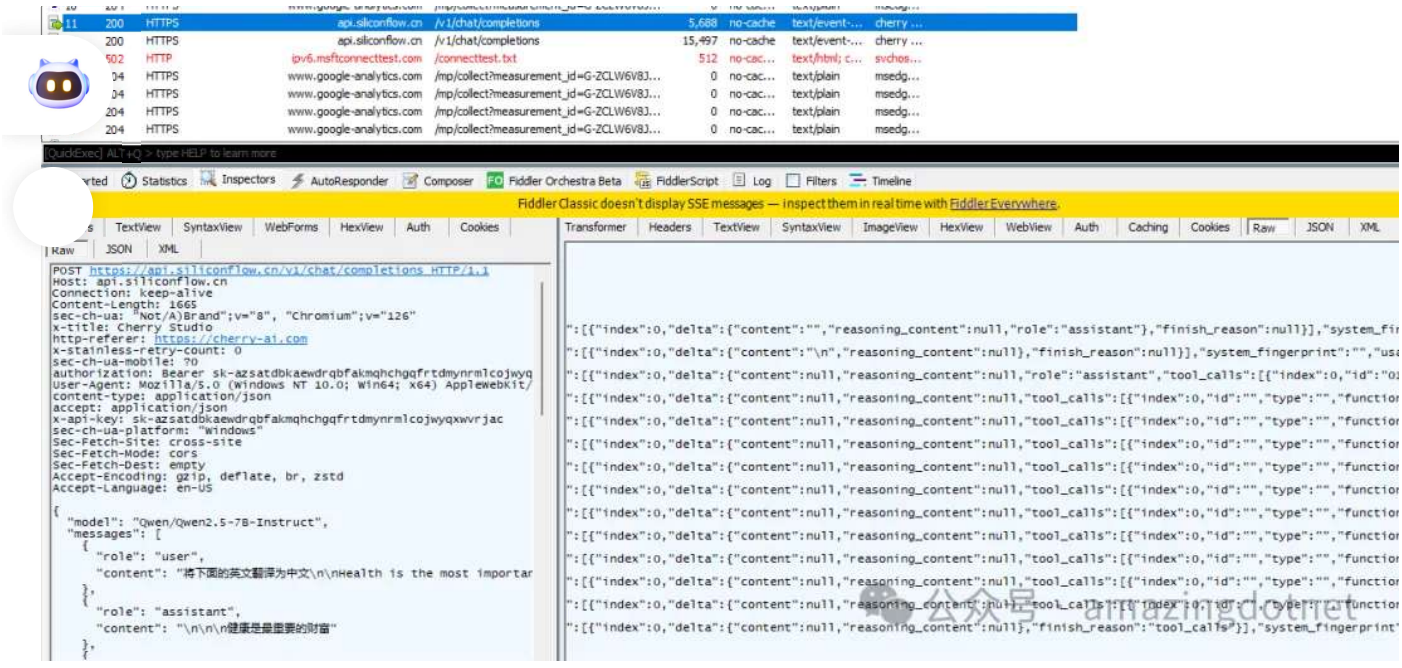
可以看到这里调用了我们 translation tool 并根据返回的结果生成了回答


执行下面的 C# 代码，并告诉我返回结果是什么
Console.WriteLine(Guid.NewGuid());
Tokens: 33
Qwen2.5-7B-Instruct | 硅基流动
04/02 23:36
ExecCSharpCode
已完成
{
"params": {
"code": "Console.WriteLine(Guid.NewGuid());"
},
"response": {
"content": [
{
"type": "text",
"text": "1925c7bc-939a-428a-948c-5cac2fce9e96\r\n"
}
],
"isError": false
}
}
执行 C# 代码的结果是生成了一个新的 GUID：1925c7bc-939a-428a-948c-5cac2fce9e96。
amazingdotnet



cherry exec-csharp-code

它背后是怎么实现的呢？我们可以抓个包看一看，在生成返回结果时有两个请求

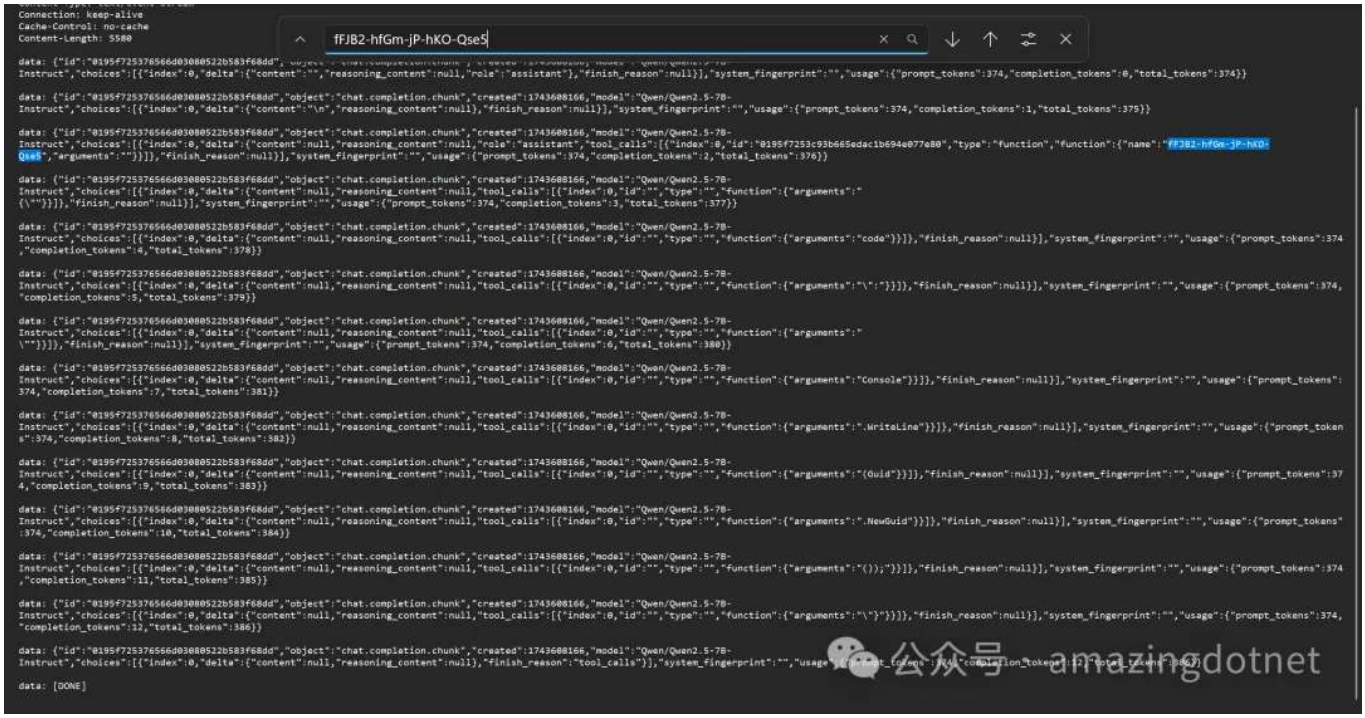


requests

第一个请求会将当前上下文的信息和启用的 mcp server tools 放在 request 中，请求服务器

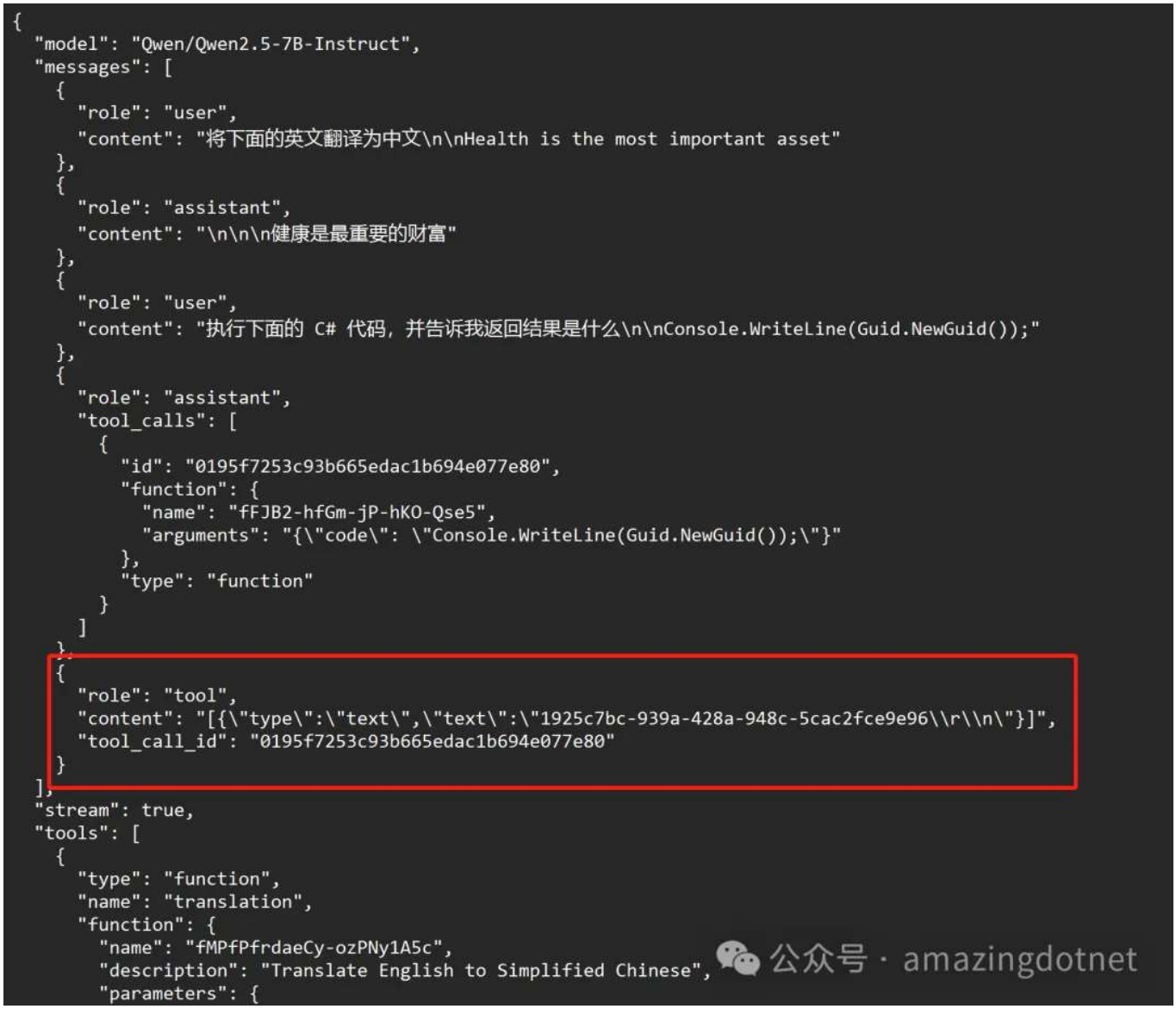
request1

服务器端返回是否需要调用某一个 tool，发现有一个 `ExecCSharpCode` 的 tool 可以调用



response1

接着客户端会去调用我们的 tool 之后再将 tool 返回的结果作为上下文的一部分再次请求服务器


{
"model": "Qwen/Qwen2.5-7B-Instruct",
"messages": [
{
"role": "user",
"content": "将下面的英文翻译为中文\n\nHealth is the most important asset"
},
{
"role": "assistant",
"content": "\n\n\n健康是最重要的财富"
},
{
"role": "user",
"content": "执行下面的 C# 代码，并告诉我返回结果是什么\n\nConsole.WriteLine(Guid.NewGuid());"
},
{
"role": "assistant",
"tool_calls": [
{
"id": "0195f7253c93b665edac1b694e077e80",
"function": {
"name": "fFJB2-hfGm-jP-hKO-Qse5",
"arguments": "{\"code\": \"Console.WriteLine(Guid.NewGuid());\"}"
},
"type": "function"
}
]
},
{
"role": "tool",
"content": "[{\"type\":\"text\",\"text\":\"1925c7bc-939a-428a-948c-5cac2fce9e96\\r\\n\"}]",
"tool_call_id": "0195f7253c93b665edac1b694e077e80"
}
],
"stream": true,
"tools": [
{
"type": "function",
"name": "translation",
"function": {
"name": "fMPfPfrdaeCy-ozPNy1A5c",
"description": "Translate English to Simplified Chinese",
"parameters": {
公众号 · amazingdotnet

request2

response2

这次服务器返回了给用户的回复，客户端也返回了回答

Stdio Sample

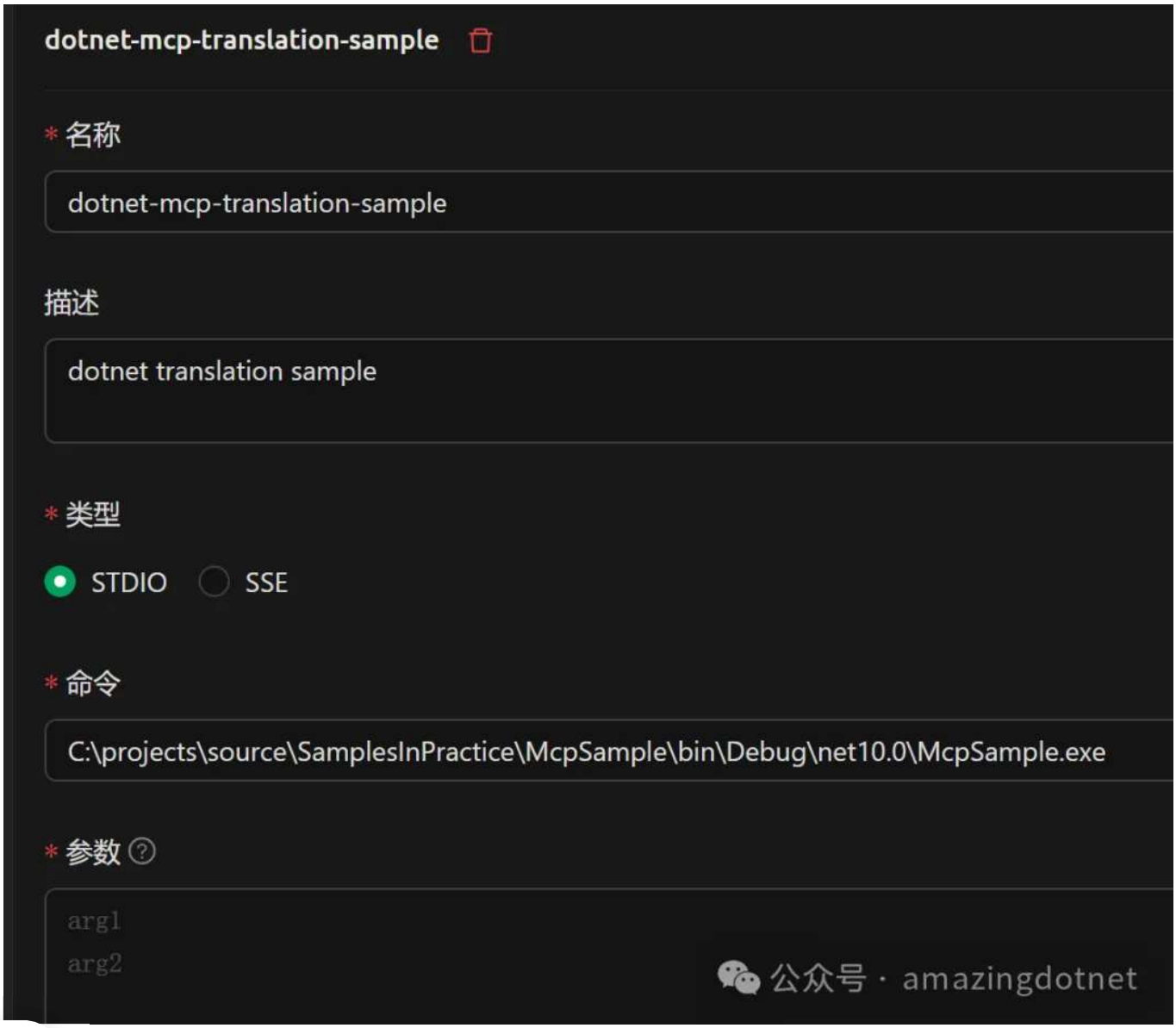
除了 SSE 之外 Stdio 标准输入输出也是一种常见的方式，不过个人不是太推荐，尤其是使用第三方的一些 mcp server 时，感觉有安全风险，你可能并不知道别人的程序会做一些什么样的操作，在本地执行时会不会出现问题，还有就是在调试的时候有遇到文件被占用需要手动去进管理器里杀死进程

首先添加对 `ModelContextProtocol` 的 NuGet package 引用

Stdio 模式是直接运行一个可执行的应用程序

```
var builder = Host.CreateApplicationBuilder(args);
builder.Services
    .AddMcpServer() // 注册 McpServer 相关服务
    .WithStdioServerTransport() // 添加标准输入输出传输支持
    .WithToolsFromAssembly() // 从程序集中扫描添加 tools
    ;
// ...
```

这样基于标准输入输出的 Mcp Server 骨架就搭好了，之后就和前面一样实现我们的 tools 就可以了，针对调试和使用也是类似的只是换成 stdio 方式即可，配置示例：



cherry-stdio-mcp-server-sample

Mcp Client

我们也可以基于 SDK 实现 MCP 客户端，示例代码如下：

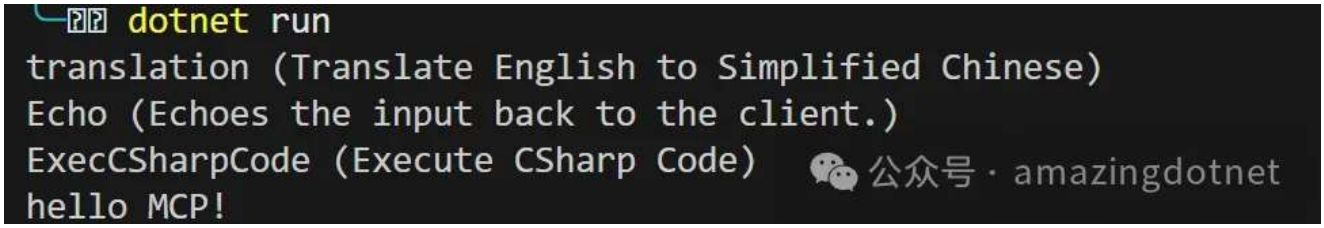
```
var client = await McpClientFactory.CreateAsync(new()
{
    Id = "aspnet-sample",
    Name = "ASP.NET McpServerSample",
    TransportType = TransportTypes.Sse,
    Location = "http://localhost:5000/sse",
```

```
});

// Print the list of tools available from the server.
foreach (var tool inawait client.ListToolsAsync())
{
    Console.WriteLine($"{tool.Name} ({tool.Description})");
}

// Execute a tool (this would normally be driven by LLM tool invocations).
var result = await client.CallToolAsync(
    "Echo",
    new Dictionary<string, object?> { ["message"] = "MCP!" },
    CancellationToken.None);

// echo always returns one and only one text content object
Console.WriteLine(result.Content.First(c => c.Type == "text").Text);
```



McpClient output

这里像 inspector 一样调用 Mcp Server 的工具，如果要做到 Cherry Studio 中在聊天上下文中调用还需要借助大语言模型的能力来实现

References

https://github.com/modelcontextprotocol/csharp-sdk

- https://github.com/WeihanLi/SamplesInPractice/tree/main/McpSample
- https://github.com/PederHP/mcpdotnet
- https://github.com/modelcontextprotocol/inspector
- https://github.com/CherryHQ/cherry-studio/releases