MCP计算器 - 模型上下文协议应用示例

项目简介

本项目是一个**MCP（模型上下文协议）**计算器应用的入门示例，专为初学者设计。它展示了如何创建一个可以被大语言模型（如Ollama）调用的工具服务。

为什么需要MCP? 大语言模型本身可能不擅长精确计算或访问实时数据，但通过MCP协议，模型可以调用外部工具来弥补这些不足，大大扩展其能力范围。

通过本项目，您将学习：

- 如何搭建一个简单的MCP服务器
- 如何定义和注册实用工具函数
- 如何让大模型找到并调用这些工具
- 完整的交互流程和工作原理

技术栈

- **Python**: 主要开发语言
- **FastAPI**: 高性能的Python Web框架
- **Pydantic**: 数据验证和设置管理
- **Uvicorn**: ASGI服务器
- **Ollama**: 本地大语言模型（用于示例调用）

项目结构

```
mcp-calculator/
├── main.py                 # 应用程序入口
├── mcp/                    # MCP核心模块
│   ├── __init__.py
│   ├── models.py           # 数据模型定义
│   └── server.py           # MCP服务器实现
├── tools/                  # 工具模块
```

```
│   ├── __init__.py
│   └── calculator.py        # 计算器工具实现
├── ollama_mcp_example.py    # Ollama调用MCP示例
└── requirements.txt         # 项目依赖
```

MCP（模型上下文协议）工作原理

什么是MCP?

MCP就像是一座连接大语言模型和外部工具的「桥梁」。想象一下：

- 大语言模型是一个聪明但有些「手无缚鸡之力」的大脑
- 各种工具（如计算器、数据库查询等）是它的「双手」
- MCP就是传递指令和结果的「神经系统」

通过这个桥梁，模型可以：

- 精确执行数学计算
- 查询数据库获取信息
- 调用外部API
- 执行文件操作
- 几乎可以做任何编程能做的事情！

MCP的核心组件

1. **工具注册系统**：就像「工具商店」，所有可用工具都在这里登记
2. **API接口层**：标准化的「对话格式」，确保模型和工具能正确交流
3. **参数验证**：「质量检查站」，确保传递给工具的信息正确有效
4. **结果返回**：「快递服务」，将工具执行结果格式化后送回给模型

工作流程图解

具体步骤：

1. **工具注册**：开发者编写工具函数（如加法、乘法等），并告诉MCP服务器「我有这些工具可以使用」
2. **服务启动**：MCP服务器启动并等待模型的调用请求
3. **模型调用**：当用户问「123456的立方根是多少」时，模型会生成一个工具调用请求
4. **工具执行**：MCP服务器收到请求后，找到对应的工具（这里是cube_root）并执行
5. **结果处理**：工具执行完成后，结果通过MCP返回给模型，模型再用自然语言回答用户

快速开始

环境准备

新手注意：如果你是第一次使用Python项目，请确保已完成以下步骤：

1. **安装Python**：确保已安装Python 3.8或更高版本。你可以在命令行中输入 `python --version` 来检查版本
2. **克隆或下载项目**：将本项目的所有文件下载到你的电脑上
3. **安装依赖**：
   - 打开命令行工具（Windows下是CMD或PowerShell）
   - 进入项目文件夹：`cd mcp-calculator`（请替换为你的实际路径）
   - 运行：`pip install -r requirements.txt`
   - 这一步会自动安装所有需要的Python库

运行MCP服务器

完成环境准备后，让我们启动服务器：

```
python main.py
```

执行后，你会看到类似这样的输出，表示服务器已成功启动：

```
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [1234]
INFO:     Started server process [5678]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

提示：服务器会在 `http://localhost:8000` 运行，不要关闭这个命令行窗口

查看API文档（可选但推荐）

FastAPI自动生成了交互式文档，这对于理解和测试API非常有用：

- **Swagger UI**: http://localhost:8000/docs
  - 这是一个交互式界面，你可以直接在这里测试各种API调用
  - 点击「Try it out」可以填写参数并发送请求
- **ReDoc**: http://localhost:8000/redoc
  - 提供了更详细的API文档说明

API接口说明

MCP服务器提供了两个主要的API接口：

1. 获取工具列表

GET /api/v1/tools

这个接口用于获取所有可用的工具信息，包含每个工具的名称、描述、参数和返回值说明。

实际示例： 你可以直接在浏览器中访问 http://localhost:8000/api/v1/tools，就能看到所有可用的计算器工具列表。

2. 调用工具

POST /api/v1/tool/call

这是最核心的接口，用于调用具体的工具函数。

请求格式示例（计算1+2）：

```
{
  "tool_name": "add",
  "tool_params": {
    "a": 1,
    "b": 2
  }
}
```

响应格式示例：

```
{
  "call_id": null,
  "success": true,
  "result": 3.0,
  "error": null
}
```

测试提示： 使用Swagger UI（http://localhost:8000/docs）可以直接在网页上测试这些API调用，无需编写代码！

可用工具

本项目实现了以下计算器工具：

工具名称	描述	参数	返回值
add	加法运算	a: float, b: float	两数之和
subtract	减法运算	a: float, b: float	两数之差
multiply	乘法运算	a: float, b: float	两数之积
divide	除法运算	a: float, b: float	两数之商
power	幂运算	a: float, b: float	a的b次方
square_root	平方根运算	a: float	a的平方根

工具名称	描述	参数	返回值
cube_root	立方根运算	a: float	a的立方根
sum_list	列表求和	numbers: list	列表所有数之和
average	计算平均值	numbers: list	列表所有数的平均值
maximum	求最大值	numbers: list	列表中的最大值
minimum	求最小值	numbers: list	列表中的最小值
calculate_expression	计算表达式	expression: str	表达式计算结果

使用Ollama调用MCP示例

这个部分将展示如何让大语言模型（Ollama）调用我们刚刚搭建的MCP计算器服务。

准备工作

在运行示例之前，请确保：

1. **安装Ollama**：
    - 从官网 (https://ollama.com/) 下载并安装Ollama
    - 安装完成后，确保Ollama服务正在运行
    - 拉取一个模型，例如：`ollama pull qwen3:0.6b`（或使用你喜欢的其他模型）
2. **启动MCP服务器**：
    - 确保前面启动的MCP服务器仍在运行
    - 服务器地址：http://localhost:8000
3. **模型配置**：
    - 打开 `ollama_mcp_example.py` 文件
    - 如果你使用了不同的模型，修改第154行的 `model="qwen3:0.6b"` 为你的模型名称

调用流程图解

运行示例

打开一个新的命令行窗口（不要关闭MCP服务器窗口），然后：

```
python ollama_mcp_example.py
```

示例解析

这个示例程序执行以下步骤：

1. **发送查询**：向Ollama发送用户问题 "计算123456的立方根"
2. **获取工具调用请求**：Ollama收到查询后，根据系统提示词分析需要调用工具，并返回一个工具调用请求的JSON
3. **解析请求**：示例程序解析这个JSON，提取出要调用的工具名称和参数
4. **调用MCP工具**：向MCP服务器发送请求，调用cube_root工具
5. **处理结果**：获取MCP服务器的执行结果，然后将结果返回给Ollama
6. **生成回答**：Ollama根据结果生成自然语言回答

输出示例

运行后，你将看到类似这样的输出：

```
用户查询: 计算123456的立方根
ollama响应: {"tool_call":{"name":"cube_root","params":{"a":123456}}}
提取到工具调用: 工具名称=cube_root, 参数={'a': 123456}
```

```
MCP工具调用结果: {'call_id': None, 'success': True, 'result': 49.8, 'error': None}
最终回答: 123456的立方根约等于49.8。
```

提示：你可以修改示例中的用户查询，尝试不同的计算任务！

构建自己的MCP应用

步骤1：创建MCP服务器

```
from mcp import MCPServer

# 创建服务器实例
mcp_server = MCPServer()
```

步骤2：定义工具函数

```
def your_tool(param1, param2):
    """您的工具功能实现"""
    # 工具逻辑
    return result
```

步骤3：注册工具

```
mcp_server.register_tool(
    name="your_tool_name",
    description="工具功能描述",
    parameters=[
        {
            "name": "param1",
            "type": "类型描述",
            "description": "参数描述",
            "required": True
        },
        # 其他参数...
    ],
    return_description="返回值描述",
    func=your_tool
)
```

步骤4：启动服务器

```
app = mcp_server.app

if __name__ == "__main__":
    import uvicorn
    uvicorn.run("main:app", host="0.0.0.0", port=8000, reload=True)
```

配置大模型与MCP交互

系统提示词设计

为了让大模型能够正确使用MCP工具，需要设计详细的系统提示词，包括：

1. 工具列表及描述
2. 工具参数说明
3. 返回值说明
4. 工具调用格式示例

参考 `ollama_mcp_example.py` 中的 `SYSTEM_PROMPT` 部分。

交互流程

1. 向模型发送用户查询和系统提示词
2. 解析模型返回的工具调用请求
3. 调用MCP服务器执行工具
4. 将执行结果返回给模型
5. 获取模型生成的最终回答

安全性考虑

在实际应用中，请注意以下安全事项：

1. **参数验证**：确保所有输入参数都经过严格验证
2. **权限控制**：考虑添加访问控制机制

3. **请求限制**：实现速率限制防止滥用
4. **代码执行安全**：避免使用不安全的代码执行方式（如eval）

扩展与优化

添加新工具

要添加新工具，只需：

1. 在适当的工具模块中定义函数
2. 添加工具描述信息
3. 将工具注册到MCP服务器

性能优化

1. 考虑使用异步处理提高并发性能
2. 对于计算密集型任务，可考虑使用线程池或进程池
3. 添加缓存机制，避免重复计算

故障排除

常见问题解答 (FAQ)

1. 服务器启动失败，提示端口被占用怎么办？

- 查看是否有其他程序占用了8000端口
- 修改`main.py`中的端口号，例如改为8001：`port=8001`

2. 安装依赖时出现错误怎么办？

- 确保pip是最新版本：`pip install --upgrade pip`
- 尝试使用虚拟环境：

```
python -m venv venv
venv\Scripts\activate  # Windows系统
pip install -r requirements.txt
```

3. Ollama调用失败，显示连接错误怎么办?

- 确认Ollama服务是否正在运行
- 检查模型名称是否正确
- 确保MCP服务器仍在运行

4. 我可以添加自己的工具函数吗?

- 当然可以！在`tools/calculator.py`中添加新的函数，然后在`CALCULATOR_TOOLS`列表中注册它
- 重启服务器后，新工具将可用

5. 如何修改用户查询示例?

- 打开`ollama_mcp_example.py`，修改第149行的`user_query`变量

下一步学习

恭喜你成功运行了MCP计算器应用！如果你想深入学习，可以尝试：

1. **添加更多工具**：实现更复杂的计算功能或其他类型的工具
2. **改进错误处理**：增强系统的健壮性，处理各种异常情况
3. **添加用户界面**：创建一个简单的Web界面，让用户可以直接与MCP服务交互
4. **集成认证**：为API添加简单的认证机制，提高安全性
5. **探索其他模型**：尝试与其他大语言模型集成，如OpenAI、Anthropic等

许可证

联系方式

如有问题或建议，请联系项目维护者。

祝你学习愉快！ 通过本项目，你已经迈出了构建AI工具集成应用的第一步。随着你的深入学习，你将能够创建更复杂、功能更强大的MCP应用！