



黑马程序员线上品牌

大模型Function Call的原理及应用

一样的教育，不一样的品质





目录

Contents

1. 什么是 Function Call
2. Function Call 的工作原理
3. Function Call 的单一函数应用
4. Function Call 的多个函数应用
5. Function Call 实现数据库查询应用

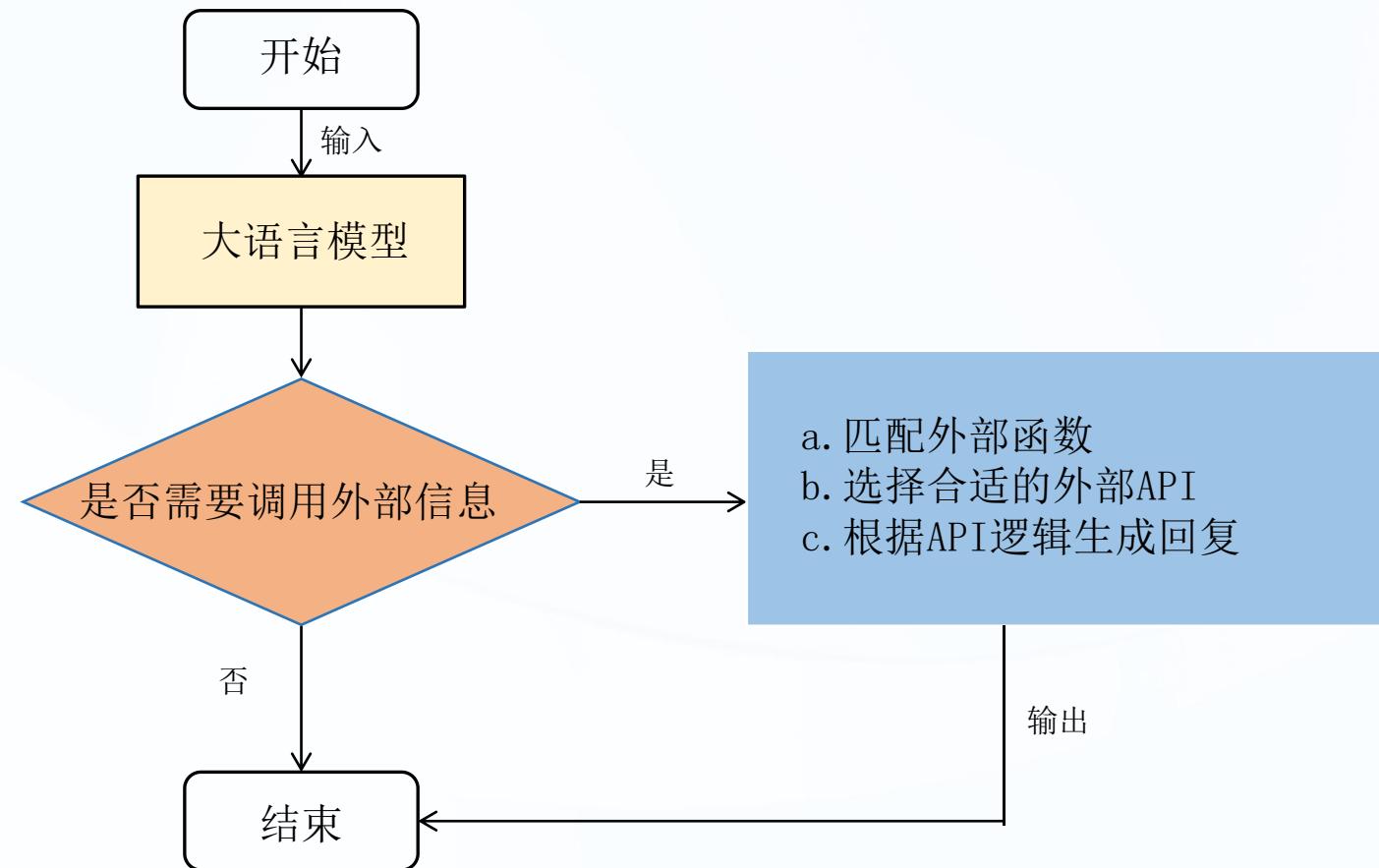
01

什么是Function Call

什么是Function Call

- 2023年6月13日 OpenAI 公布了 Function Call（函数调用）功能，该功能指的是在语言模型中集成外部功能或API的调用能力，这意味着模型可以在生成文本的过程中调用外部函数或服务，获取额外的数据或执行特定的任务。

Function Call 应用基本流程（简化）：



Function Call 的功能

Function Call 可以解决大模型什么问题：

01

信息实时性

大模型训练的数据集无法包含最新的信息，如最新的新闻、实时股价等。通过Function Call，模型可以实时获取最新数据，提供更加时效的服务。

02

数据局限性

模型训练数据虽多但有限，无法覆盖所有领域，如医学、法律等领域的专业咨询，Function Call允许模型调用外部数据库或API，获取特定领域的详细信息。

03

功能扩展性

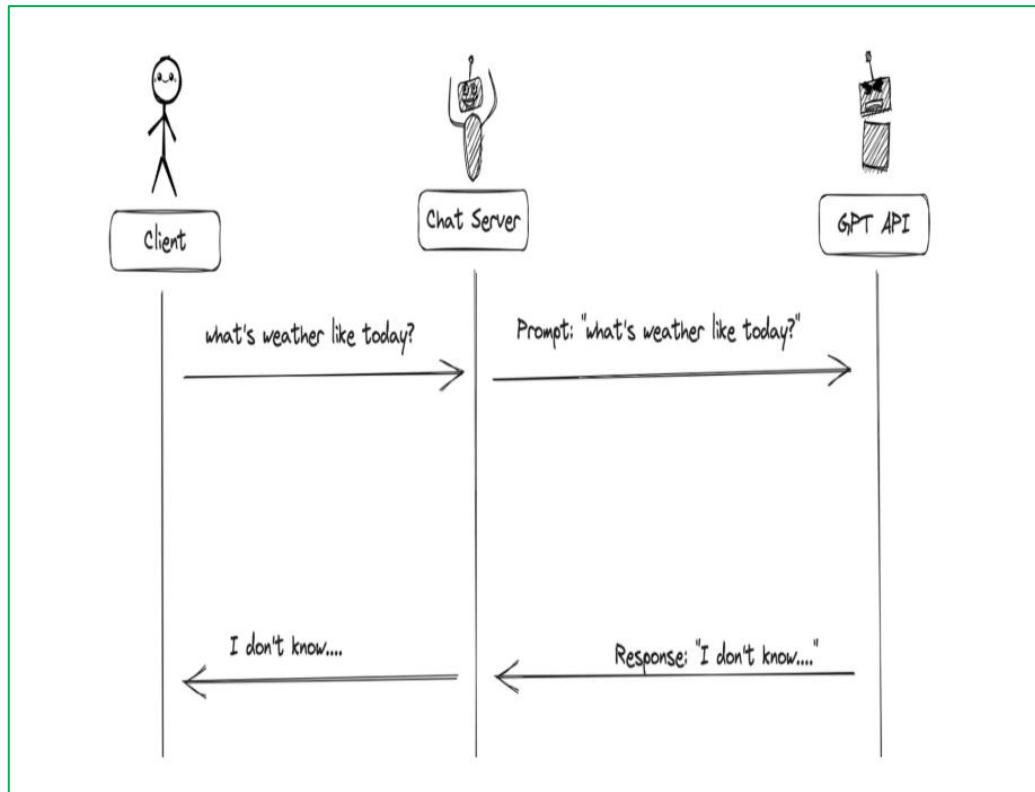
大模型虽然功能强大，但不可能内置所有可能需要的功能。通过Function Call，可以轻松扩展模型能力，如调用外部工具进行复杂计算、数据分析等。

02

Function Call工作原理

Function Call 工作原理

当没有函数调用(function-call)时候，我们调用GPT构建AI应用的模式非常简单。

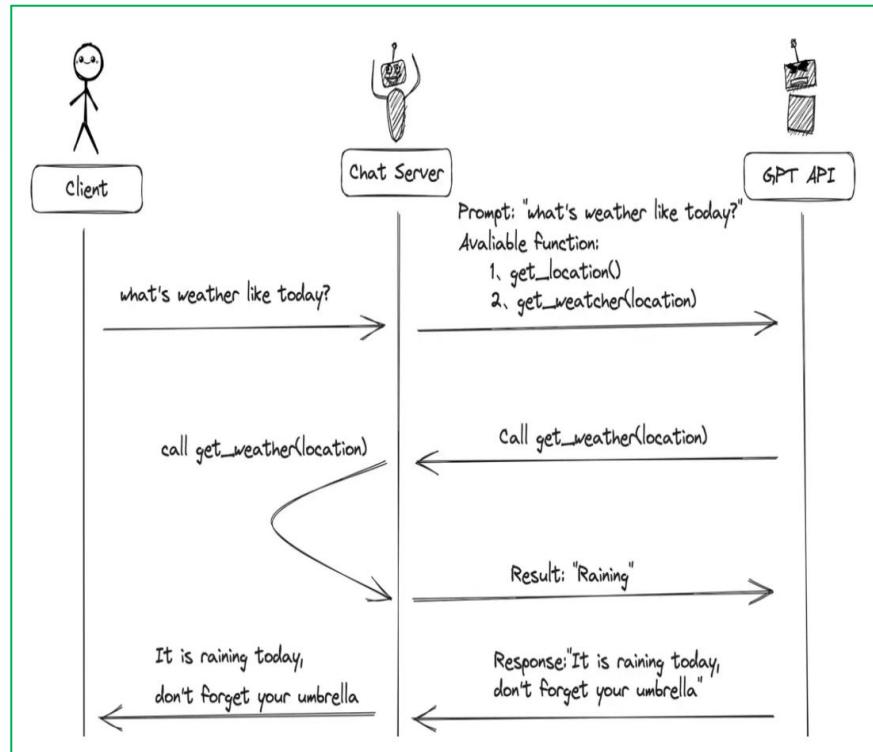


主要步骤：

1. 用户(Client)发请求给我们的服务(Chat Server)
2. 我们的服务(Chat Server)给GPT提示词
3. 重复执行

Function Call 工作原理

当有函数调用(function-call)时候，我们调用GPT构建AI应用的模式比之前要复杂一些。



主要步骤：

1. 用户 (Client) 发请求prompt以及functions给我们的服务 (Chat Server)
2. GPT模型根据用户的prompt，判断是用普通文本还是函数调用的格式响应我们的服务 (Chat Server)
3. 如果是函数调用格式，那么Chat Server就会执行这个函数，并且将结果返回给GPT
4. 然后模型使用提供的数据，用连贯的文本响应。返回

注意：大模型的 Function call 不会调用函数，仅返回函数的参数。开发者利用模型输出的参数在应用中调用函数。

思考总结

Thinking summary

1. 什么是Function Call?

答案：在语言模型中集成外部功能或API的调用能力.

2. LLM模型是否能够直接运行function?

答案：不会调用函数，仅返回函数的参数。开发者利用模型输出的参数在应用中调用函数.

03

Function Call的单一函数应用

Function Call 的单一函数应用



假设我们要创建一个具备查询实时天气的聊天机器人。

基本流程如下：

1. 准备工作

定义外部函数，
描述函数功能

3. 模型应用Function Call

申请API-KEY，配
置环境变量。

2. 定义Function Tool
s

输入prompt，模型输出函数参
数，然后调用本地函数得到结
果，融合消息再次送入模型，
得到结果。

Function Call 的单一函数应用



第一步：准备工作：

模型选择：

国内外支持Function Call的模型，如：ChatGPT、百度文心一言，智谱ChatGLM3、讯飞星火3.0等。

此次应用基于智谱AI的ChatGLM来实现，注册申请API-KEY：<https://open.bigmodel.cn/dev/howuse/functioncall>

开发环境：

python版本：3.10以上

pip install zhipu

Function Call 的单一函数应用



第二步：定义Function Tools：



目的

- 1. 定义查询天气的外部函数； 2. 描述函数功能； 3. 解析模型参数调用函数

代码路径

`./ChatGLM3_FunctionCall/weather/tools.py`

具体代码

一共包含2个自定义函数，一个函数功能描述，具体请看下文

Function Tools代码实现

◆ 导入必备的工具包

```
import json
import requests
```

◆ 自定义get_current_weather函数

```
def get_current_weather(location):
    """得到给定地址的当前天气信息"""
    # 读取城市及对应的邮政编码
    with open('./cityCode_use.json', 'r') as file:
        # 使用 json.load() 函数加载 JSON 数据
        data = json.load(file)
        city_code = ""
        weather_info = {}
        for loc in data:
            if location == loc["市名"]:
                city_code = loc["编码"]
```

```
# 调用天气查询API接口
if city_code:
    weather_url =
        "http://t.weather.itboy.net/api/weather/city/" + city_code
    response = requests.get(weather_url)
    result1 = eval(response.text)
    forecast = result1["data"]["forecast"][0]
    weather_info = {
        "location": location,
        "high_temperature": forecast["high"],
        "low_temperature": forecast["low"],
        "week": forecast["week"],
        "type": forecast["type"],
    }
    return json.dumps(weather_info, ensure_ascii=False)
```

Function Tools代码实现

◆ 自定义函数描述功能

```
tools = [
{
    "type": "function",
    "function": {
        "name": "get_current_weather",
        "description": "获取给定位置的当前天气",
        "parameters": {
            "type": "object",
            "properties": {
                "location": {
                    "type": "string",
                    "description": "城市或区，例如北京、海淀",
                },
            },
            "required": ["location"],
        },
    }
}
```

参数名称	类型	是否必填	参数说明
type	String	是	设置为function
function	Object	是	
name	String	是	函数名称
description	String	是	用于描述函数功能，模型会根据这段描述决定函数调用方式。
parameters	Object	是	parameters字段需要传入一个Json Schema对象，以准确地定义函数所接受的参数。若调用函数时不需要传入参数，省略该参数即可。
required		否	指定哪些属性在数据中必须被包含。

Function Tools代码实现

◆ 解析模型参数调用函数

```
def parse_response(response):
    #根据模型回复来确定是否调用工具函数，如果调用返回函数的结果
    response_message = response.choices[0].message
    # 根据模型的response内容，检测是否需要调用函数
    if response_message.tool_calls:
        # 调用函数
        available_functions = { "get_current_weather": get_current_weather} # 这里我们定义了一个函数，也可以定义多个选择
        function_name = response_message.tool_calls[0].function.name # 从模型回复中获得函数名
        fuction_to_call = available_functions[function_name]
        function_args = json.loads(response_message.tool_calls[0].function.arguments)
        function_response = fuction_to_call(
            location=function_args.get("location"),)
        return function_response
```

Function Call 的单一函数应用



第三步：模型应用Function:



目的

1. 实现大模型Function Call的功能应用

代码路径

`./ChatGLM3_FunctionCall/weather/weather_zhipu.py`

具体代码

一共包含2个函数：调用模型函数和主逻辑函数。如下文：

模型应用 Function Call

◆ 导入必备的工具包

```
import os
from dotenv import load_dotenv, find_dotenv
from tools import *
from zhipuapi import ZhipuAI
_ = load_dotenv(find_dotenv())
zhupu_ak = os.environ['zhupu_api']
client = ZhipuAI(api_key=zhupu_ak) # 填写您自己的APIKey
ChatGLM = "glm-4"
```

模型应用 Function Call

◆ 定义模型调用函数

```
def chat_completion_request(messages, tools=None, tool_choice=None, model=ChatGLM):  
    try:  
        response = client.chat.completions.create(  
            model=model,  
            messages=messages,  
            tools=tools,  
            tool_choice=tool_choice)  
        return response  
    except Exception as e:  
        print("Unable to generate ChatCompletion response")  
        print(f"Exception: {e}")  
    return e
```

模型应用 Function Call

◆ 主逻辑函数main:

```
def main():
    #定义prompt
    messages = []
    messages.append({"role": "system", "content": "你是一个天气播报小助手，你需要根据用户提供的地址来回答当地的天气情况，如果用户提供的问题具有不确定性，不要自己编造内容，提示用户明确输入"})
    messages.append({"role": "user", "content": "今天北京的天气如何"})
    # 将prompt输入给模型（第一次）
    response = chat_completion_request(messages, tools=tools, tool_choice="auto")
    # 将模型得到结果进行解析（是否调用函数，得到函数结果）
    function_response = parse_response(response)
```

模型应用 Function Call

◆ 主逻辑函数main:

```
assistant_message = response.choices[0].message
print(f'assistant_message-->{assistant_message}')
messages.append(assistant_message.model_dump()) #将第一次模型的回复结果添加到message(prompt)
function_name = response.choices[0].message.tool_calls[0].function.name # 基于第一次模型回复，得到调用的函数
#名称
function_id = response.choices[0].message.tool_calls[0].id # 基于第一次模型回复，得到函数的id
# 整理函数返回的结果，依然添加到messages中
messages.append({
    "role": "tool",
    "tool_call_id": function_id,
    "name": function_name,
    "content": function_response })
# 将messages送入模型（第二次包含函数返回内容，得到最终模型输出结果）
last_response = chat_completion_request(messages, tools=tools, tool_choice="auto")
```

模型应用 Function Call

◆ 结果展示：

```
last_response--》 CompletionMessage(content='根据您的查询，我获取到了北京市当前的天气情况。今天是星期一，北京的天气情况是晴天，最高气温为33℃，最低气温为17℃。请注意天气变化，做好防晒和保暖措施。', role='assistant', tool_calls=None)
```



思考总结

Thinking summary

1. Function Call 单一函数应用流程?

答案:

1. 定义函数: 函数可以是真实的外部API或工具, 也可以是模拟函数。
2. 提供函数定义给模型: 将定义好的函数作为参数(tools)提供给模型。
3. 模型生成函数调用JSON: 包含要调用的函数名称和参数值。
4. 后端系统执行函数: 后端系统根据JSON中的内容, 调用相应的函数结果。
5. 将函数结果返回给模型: 函数返回结果给模型, 作为模型的额外上下文信息。
6. 模型生成最终响应: 模型综合原始查询、函数调用结果等信息, 生成最终输出。

04

Function Call的多个函数应用

Function Call 的单一函数应用



假设我们要创建一个具备查询航班功能的聊天机器人

流程和环境同上一章节一致，但是由于涉及多个函数，因此在代码设计结构上有所改动

完整代码包含三个部分：`muti_function_zhipu.py`、`airplane_function_tools.py`、`muti_utils.py`。

`muti_utils.py`： 主要用来定义多个Functions，以及函数的应用

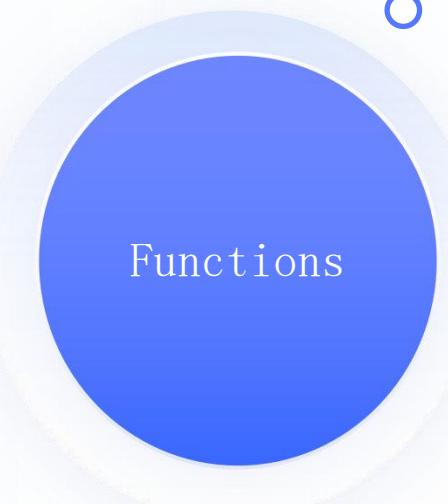
`airplane_function_tools.py`： 主要用来实现函数的功能描述Tools

`muti_function_zhipu.py`： 主逻辑函数，实现模型Function Call的应用

Function Call 的多个函数应用



第一步：构建脚本multi_utils.py



目的

1. 定义查询飞机航班的函数；
2. 定义查询飞机票价的函数；
3. 解析模型参数调用函数

代码路径

`./ChatGLM3_FunctionCall/airplane/muti_utils.py`

具体代码

一共包含3个自定义函数，具体请看下文

Functions代码实现

◆ 导入必备的工具包

```
import json
```

◆ 自定义get_plane_number函数

```
def get_plane_number(date, start , end):  
    plane_number = {  
        "北京": {  
            "深圳": "126",  
            "广州": "356",  
        },  
        "郑州": {  
            "北京": "1123",  
            "天津": "3661",  
        }  
    }  
    return {"date": date, "number": plane_number[start][end]}
```

◆ 自定义 get_ticket_price函数

```
def get_ticket_price(date:str ,  
                     number:str):  
    print(date)  
    print(number)  
    return {"ticket_price": "668"}
```

Functions代码实现

◆ 自定义 parse_function_call函数

```
def parse_function_call(model_response):
    function_result = ''
    if model_response.choices[0].message.tool_calls:
        tool_call = model_response.choices[0].message.tool_calls[0]
        args = tool_call.function.arguments
        function_result = {}
        if tool_call.function.name == "get_plane_number":
            function_result = get_plane_number(**json.loads(args))
        if tool_call.function.name == "get_ticket_price":
            function_result = get_ticket_price(**json.loads(args))
    return function_result
```

Function Call 的多个函数应用



第二步：构建脚本airplane_function_tools.py



目的

1. 定义函数描述功能，相当于给大模型应用的工具描述： tools

代码路径

`./ChatGLM3_FunctionCall/airplane/airplane_function_tools.py`

具体代码
请看下文

Function Tools代码实现

◆ 自定义函数描述功能：

```
tools = [
    {
        "type": "function",
        "function": {
            "name": "get_plane_number",
            "description": "根据始发地、目的地和日期，查询对应日期的航班号",
            "parameters": {
                "type": "object",
                "properties": {
                    "start": {
                        "description": "出发地",
                        "type": "string"
                    },
                    "end": {
                        "description": "目的地",
                        "type": "string"
                    },
                    "date": {
                        "description": "日期",
                        "type": "string",
                    }
                },
                "required": ["start", "end", "date"]
            }
        }
    }
}
```

```
{
    "type": "function",
    "function": {
        "name": "get_ticket_price",
        "description": "查询某航班在某日的价格",
        "parameters": {
            "type": "object",
            "properties": {
                "number": {
                    "description": "航班号",
                    "type": "string"
                },
                "date": {
                    "description": "日期",
                    "type": "string",
                }
            },
            "required": [ "number", "date"]
        }
    }
}
```

Function Call 的多个函数应用



第三步：构建multi_function_zhipu.py：

Function
应用

目的

1. 实现大模型Function Call的功能应用

代码路径

./ChatGLM3_FunctionCall/airplane/multi_function_zhipu.py

具体代码
如下文：

模型应用 Function Call

◆ 导入必备的工具包

```
from zhipuai import ZhipuAI
from dotenv import load_dotenv, find_dotenv
from muti_utils import *
from airplane_function_tools import *
import os
_ = load_dotenv(find_dotenv())
# 获取环境变量 ZhiPu_API_KEY
zhupu_ak = os.environ['zhupu_api']
client = ZhipuAI(api_key=zhupu_ak) # 填写您自己的APIKey
ChatGLM = "glm-4"
```

模型应用 Function Call

◆ 定义模型调用函数

```
def chat_completion_request(messages, tools=None, tool_choice=None, model=ChatGLM):  
    try:  
        response = client.chat.completions.create(  
            model=model,  
            messages=messages,  
            tools=tools,  
            tool_choice=tool_choice,)  
        return response  
    except Exception as e:  
        print("Unable to generate ChatCompletion response")  
        print(f"Exception: {e}")  
    return e
```

模型应用 Function Call

◆ 主逻辑函数main:

```
def main():
    messages = []
    messages.append({"role": "system", "content": "现在你是一个航班查询助手，将根据用户问题提供答案，但是不要假设或猜测传入函数的参数值。如果用户的描述不明确，请要求用户提供必要信息"})
    messages.append({"role": "user", "content": "帮我查询2024年4月2日，郑州到北京的航班的票价"})
    # 1. 第一次调用模型得到回复
    first_response = chat_completion_request(messages, tools=tools, tool_choice="auto")
    assistant_message1 = first_response.choices[0].message # 打印查看
    # 2. 将第一次得到的模型回复结果加入messages
    messages.append(first_response.choices[0].message.model_dump())
    # 3. 基于first_response需要调用函数
    first_function = parse_function_call(model_response=first_response)
    # 4. 将函数的结果添加到messages中，继续送入模型问答
```

模型应用 Function Call

◆ 主逻辑函数main:

```
tool_call = first_response.choices[0].message.tool_calls[0]
messages.append({"role": "tool", "tool_call_id": tool_call.id, "content":
str(json.dumps(first_function))})

# 5. 第二次调用模型
second_response = chat_completion_request(messages, tools=tools, tool_choice="auto")

# 6. 将第二次得到模型结果加入信息中
messages.append(second_response.choices[0].message.model_dump())

# 7. 基于second_response需要调用函数
second_function = parse_function_call(model_response=second_response)
tool2_call = second_response.choices[0].message.tool_calls[0]

# 8. 将函数的结果添加到messages中，继续送入模型问答
messages.append({"role": "tool", "tool_call_id": tool2_call.id, "content":
str(json.dumps(second_function))})

# 9. 将融合的信息再次送入模型，得到最终结果
last_response = chat_completion_request(
```

模型应用 Function Call

◆ 结果展示：

```
last_response--》 CompletionMessage(content='2024年4月2日，郑州到北京的航班号为1123，票价为668元。', role='assistant', tool_calls=None)
```

04

Function Call 实现数据库查询

Function Call 的单一函数应用



假设我们要创建一个可以连接数据库查询的聊天机器人

流程和环境同上一章节一致，但是在代码设计结构上有所改动

完整代码包含两个部分：sql_zhipu.py、sql_function_tools.py

sql_function_tools： 主要用来定义查询数据库函数、函数功能描述，以及函数的应用，

sql_zhipu.py： 主逻辑函数，实现模型Function Call的应用

Function Call 的多个函数应用



第一步：构建脚本sql_function_tools.py



目的

1. 定义查询数据库的函数； 2. 描述函数功能tools； 3. 解析模型参数调用函数

代码路径

`./ChatGLM3_FunctionCall/sql/sql_function_tools.py`

具体代码

一共包含2个自定义函数，具体请看下文

Sql_Function_Tools代码实现

◆ 导入必备的工具包

```
import json
import pymysql
```

◆ 描述数据库表结构（单一个表格）

```
database_schema_string = """
CREATE TABLE `emp` (
`empno` int DEFAULT NULL, --员工编号, 默认为空
`ename` varchar(50) DEFAULT NULL, --员工姓名, 默认为空
`job` varchar(50) DEFAULT NULL, --员工工作, 默认为空
`mgr` int DEFAULT NULL, --员工领导, 默认为空
`hiredate` date DEFAULT NULL, --员工入职日期, 默认为空
`sal` int DEFAULT NULL, --员工的月薪, 默认为空
`comm` int DEFAULT NULL, --员工年终奖, 默认为空
`deptno` int DEFAULT NULL, --员工部分编号, 默认为空
)""""
```

Sql_Function_Tools代码实现

◆ 描述数据库表结构（多个表格）

```
database_schema_string1 = """
CREATE TABLE `emp` (
    `empno` int DEFAULT NULL, --员工编号, 默认为空
    `ename` varchar(50) DEFAULT NULL, --员工姓名, 默认为空
    `job` varchar(50) DEFAULT NULL, --员工工作, 默认为空
    `mgr` int DEFAULT NULL, --员工领导, 默认为空
    `hiredate` date DEFAULT NULL, --员工入职日期, 默认为空
    `sal` int DEFAULT NULL, --员工的月薪, 默认为空
    `comm` int DEFAULT NULL, --员工年终奖, 默认为空
    `deptno` int DEFAULT NULL, --员工部分编号, 默认为空
);
CREATE TABLE `DEPT` (
    `DEPTNO` int NOT NULL, -- 部门编码, 默认为空
    `DNAME` varchar(14) DEFAULT NULL, --部门名称, 默认为空
    `LOC` varchar(13) DEFAULT NULL, --地点, 默认为空
    PRIMARY KEY (`DEPTNO`)
);
```

Sql_Function_Tools代码实现

◆ 自定义ask_database()函数

```
def ask_database(query):
    """连接数据库，进行查询"""
    # 1. 连接到 MySQL 数据库
    print("进入函数内部")
    conn = pymysql.connect(
        host='localhost',
        port=3306,
        user='root',
        password='123456',
        database='it_heim',
        charset='utf8mb4',  # 指定游标类，返回结果为字典
    )
```

```
# 2. 创建游标
cursor = conn.cursor()
print(f'开始测试')

# 3. 执行sql语句测试
# 示例：执行 SQL 查询
# sql = "SELECT * FROM emp"
cursor.execute(query)

# 4. 获取查询结果
result = cursor.fetchall()

# 5. 关闭游标
cursor.close()

# 6. 关闭连接
conn.close()

return result
```

Sql_Function_Tools代码实现

◆ 自定义tools

```
tools = [
    {
        "type": "function",
        "function": {
            "name": "ask_database",
            "description": "使用此函数回答业务问题，要求输出是一个SQL查询语句",
            "parameters": {
                "type": "object",
                "properties": {
                    "query": {
                        "type": "string",
                        "description": f"SQL查询提取信息以回答用户的问题。"
                        f"SQL应该使用以下数据库模式编写:{database_schema_string1}"
                        f"查询应该以纯文本返回，而不是JSON。"
                        f"查询应该只包含MySQL支持的语法。",
                    }
                },
                "required": ["query"],
            },
        },
    }
}
```

Sql_Function_Tools代码实现

◆ 自定义parse_response()函数

```
def parse_response(response):
    response_message = response.choices[0].message
    if response_message.tool_calls:
        # 调用函数
        available_functions = {
            "ask_database": ask_database
        } # only one function test in this example, but you can have multiple
        function_name = response_message.tool_calls[0].function.name
        function_to_call = available_functions[function_name]
        function_args = json.loads(response_message.tool_calls[0].function.arguments)
        function_response = function_to_call(
            query=function_args.get("query"),
        )
    return function_response
```

Function Call 的多个函数应用



第二步：构建脚本sql_zhipu.py

Function
应用

目的

1. 实现大模型Function Call的功能应用

代码路径

./ChatGLM3_FunctionCall/sql/sql_zhipu.py

具体代码

具体请看下文

模型应用 Function Call

◆ 导入必备的工具包

```
from zhipuai import ZhipuAI
from dotenv import load_dotenv, find_dotenv
from sql_function_tools import *
import os
_ = load_dotenv(find_dotenv())
# 获取环境变量 ZhiPu_API_KEY
zhupu_ak = os.environ['zhupu_api']
client = ZhipuAI(api_key=zhupu_ak) # 填写您自己的APIKey
ChatGLM = "glm-4"
```

模型应用 Function Call

◆ 定义模型调用函数

```
def chat_completion_request(messages, tools=None, tool_choice=None, model=ChatGLM):  
    try:  
        response = client.chat.completions.create(  
            model=model,  
            messages=messages,  
            tools=tools,  
            tool_choice=tool_choice,)  
        return response  
    except Exception as e:  
        print("Unable to generate ChatCompletion response")  
        print(f"Exception: {e}")  
    return e
```

模型应用 Function Call

◆ 主逻辑函数main:

```
def main():
    #定义prompt
    messages = []
    messages.append({"role": "system", "content": "通过针对业务数据库生成 SQL 查询来回答用户的问题"})
    messages.append({"role": "user", "content": "查询一下最高工资的员工姓名及对应的工资"})
    # 将prompt输入给模型（第一次）
    response = chat_completion_request(messages, tools=tools, tool_choice="auto")
    # 将模型得到结果进行解析（是否调用函数，得到函数结果）
    function_response = parse_response(response)
```

模型应用 Function Call

◆ 主逻辑函数main:

```
assistant_message = response.choices[0].message
print(f'assistant_message-->{assistant_message}')
messages.append(assistant_message.model_dump()) #将第一次模型的回复结果添加到message(prompt)
function_name = response.choices[0].message.tool_calls[0].function.name # 基于第一次模型回复，得到调用的函数
#名称
function_id = response.choices[0].message.tool_calls[0].id # 基于第一次模型回复，得到函数的id
# 整理函数返回的结果，依然添加到messages中
messages.append({
    "role": "tool",
    "tool_call_id": function_id,
    "name": function_name,
    "content": function_response })
# 将messages送入模型（第二次包含函数返回内容，得到最终模型输出结果）
last_response = chat_completion_request(messages, tools=tools, tool_choice="auto")
```

模型应用 Function Call

◆ 结果展示：

```
last_response--》 content='根据您的查询，我已经为您找到了工资最高的员工。这位员工的姓名是KING，  
他的工资是5000元。' role='assistant' tool_calls=None
```



黑马程序员线上品牌



扫码关注博学谷微信公众号

