# 链的使用

#### 异步支持

LangChain通过利用asyncio库为链（Chain）提供了异步支持。

目前在LLMChain（通过arun, apredict, acall）、LLMMathChain（通过arun和acall）、ChatVectorDBChain以及QA链中支持异步方法。其他链的异步支持正在规划中。

#### 使用方法

所有的链都可以像函数一样被调用。当链对象只有一个输出键（也就是说，它的output\_keys中只有一个元素）的时候，我们预期的结果只需要一个字符串，可以使用run方法。

在LangChain中，所有继承自Chain类的对象，提供了一些用于执行链逻辑的方式。其中一种比较直接的方式就是使用\_\_call\_\_方法。\_\_call\_\_ 方法是 Chain 类的一个方法，它让 Chain 类的实例可以像函数一样被调用，比如 result = chain(inputs, return\_only\_outputs=True)就完成了调用链。

先看看\_\_call\_\_方法的定义：

def \_\_call\_\_(  
 self,  
 inputs: Union[Dict[str, Any], Any],  
 return\_only\_outputs: bool = False,  
 callbacks: Callbacks = None,  
 \*,  
 tags: Optional[List[str]] = None,  
 include\_run\_info: bool = False,  
 ) -> Dict[str, Any]:

这个 \_\_call\_\_ 方法接收的参数，最有用的是以下三个：

* inputs：这个参数是要传递给链的输入。它的类型是 Any，这意味着可以接收任何类型的输入。
* return\_only\_outputs：这个参数是一个布尔值，如果设为 True，则只返回输出结果。如果设为 False，则可能返回其他额外的信息。
* callbacks：这个参数是回调函数的列表，它们将在链执行过程中的某些时刻被调用。

\_\_call\_\_ 方法返回一个字典，这个字典包含了链执行的结果和可能的其他信息。

在 Python 中，如果一个类定义了 \_\_call\_\_ 方法，那么这个类的实例就可以像函数一样被调用。例如，如果 chain 是 Chain 类的一个实例，那么你可以像调用函数一样调用 chain：

result = chain(inputs, return\_only\_outputs=True)

在这个调用中，inputs 是要传递给链的输入，return\_only\_outputs=True 表示只返回输出结果。返回的 result 是一个字典，包含了链执行的结果。

使用的时候，最重要的参数是 inputs:

chat = ChatOpenAI(temperature=0)  
prompt\_template = "Tell me a {adjective} joke"  
llm\_chain = LLMChain(llm=chat, prompt=PromptTemplate.from\_template(prompt\_template))  
  
llm\_chain(inputs={"adjective": "corny"})

返回的结果是：

{'adjective': 'corny',  
 'text': 'Why did the tomato turn red? Because it saw the salad dressing!'}

你可以通过设置return\_only\_outputs为True来配置它只返回输出键值。

llm\_chain("corny", return\_only\_outputs=True)

返回的结果就不包含 "adjective": "corny"：

{'text': 'Why did the tomato turn red? Because it saw the salad dressing!'}

然而，当链对象只有一个输出键（也就是说，它的output\_keys中只有一个元素）的时候，我们可以使用run方法。

# llm\_chain only has one output key, so we can use run  
llm\_chain.output\_keys

['text']

output\_keys中只有一个元素 ['text']，我们可以run方法：

llm\_chain.run({"adjective": "corny"})

如果输入的键值只有一个，预期的输出也是一个字符串，那么输入可以是字符串也可以是对象，可以使用run方法也可以使用\_\_call\_\_方法。

run方法将整个链的输入键值（input key values）进行处理，并返回处理后的结果。需要注意的是，与\_\_call\_\_方法可能返回字典形式的结果不同，run方法总是返回一个字符串。这也是为什么当链对象只有一个输出键的时候，我们倾向于使用run方法，因为这时候处理结果自然只有一个，返回字符串形式更直观也更便于处理。

例如，假设我们有一个链对象，它的任务是根据输入的文本生成摘要，那么在调用run方法的时候，我们可以直接将待摘要的文本作为参数输入，然后得到摘要后的文本。在这种情况下，你可以直接输入字符串，而无需指定输入映射。

另外，你可以很容易地将一个Chain对象作为一个工具，通过它的run方法集成到你的Agent中，这样可以将链的处理能力直接用于你的Agent逻辑中。

#### 支持自定义链

你可以子类化Chain并实现你自己的自定义链。从其输出中仅仅调试链对象可能会比较困难，因为大多数链对象涉及到相当多的输入提示预处理和LLM输出后处理。

#### 链的调试

将verbose设置为True将会在运行链对象时打印出一些链对象的内部状态。

conversation = ConversationChain(  
 llm=chat,  
 memory=ConversationBufferMemory(),  
 verbose=True  
)  
conversation.run("What is ChatGPT?")

#### 加记忆的链

链可以使用Memory对象进行初始化，这将使得在调用链时数据持久化，使得链具有状态。

from langchain.chains import ConversationChain  
from langchain.memory import ConversationBufferMemory  
  
conversation = ConversationChain(  
 llm=chat,  
 memory=ConversationBufferMemory()  
)  
  
conversation.run("Answer briefly. What are the first 3 colors of a rainbow?")  
# -> The first three colors of a rainbow are red, orange, and yellow.  
conversation.run("And the next 4?")  
# -> The next four colors of a rainbow are green, blue, indigo, and violet.

#### 链序列化

我们使用的序列化格式是json或yaml。目前，只有一些链支持这种类型的序列化。我们将随着时间的推移增加支持的链的数量。首先，让我们看看如何将链保存到磁盘。这可以通过.save方法完成，并指定一个带有json或yaml扩展名的文件路径。我们可以使用load\_chain方法从磁盘加载链。