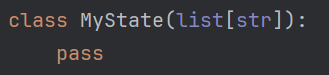
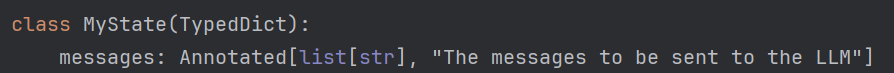
# LangGraph

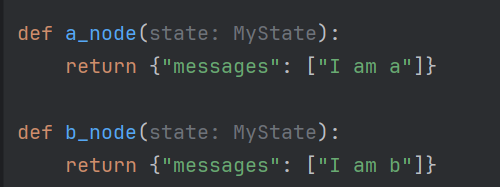
LangGraph中的状态

状态的定义

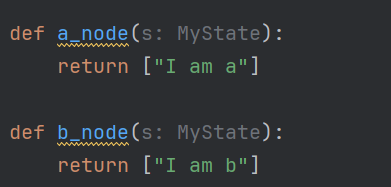


状态的定义决定了每个节点接收的参数类型以及返回值类型。

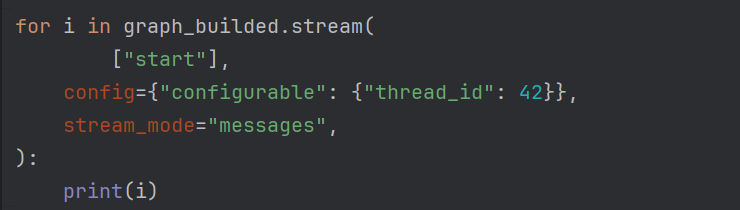
对于第一种



对于第二种



## Graph的流式输出



通过指定stream\_mode可改变流式输出的结果

## Graph的ToolNode

仿写LangGraph 预构建的 ToolNode逻辑

文本

AI 生成的内容可能不正确。

# RAG分块技术

# 高级RAG

原文：[15种高级RAG技术：从预检索到生成全面提升RAG效果\_making retrieval-augmented language models robust -CSDN博客](https://blog.csdn.net/xx_nm98/article/details/141611559)

优化策略可分以下几类：

* 预检索优化
* 检索/召回策略
* 检索后优化
* 生成优化

## 预检索优化（检索前优化）

**使用 LLM 提高信息密度**

原始文本可能包含了大量无关信息，信息密度，使用LLM提高文本的信息密度，之后再进行嵌入

**创建假设性问答对**

用LLM对每个chunk生成问题和答案，将问题和答案进行嵌入，在检索时，系统将检索问题及其对应的答案并提供给 LLM。

## 检索/召回优化（检索中优化）

**使用 LLM 优化搜索请求**

将用户提出的问题交给LLM进行优化，返回一个更好的问题；

原始问题：

LLM优化：



对于对话系统需要进一步改进：

客户：“你们的定期存款利率是多少？”

``助手：“我们的利率是 XYZ。”

``客户：“哪种信用卡适合旅行？”

``助手：“XYZ 信用卡适合旅行，原因是 ABC。”

``客户：“告诉我更多关于那个利率的信息。”

如果只是把最后一次对话交由LLM优化显然是不行的，因为信息不足。如果将整个对话记录交由LLM优化，也不一定好，因为客户只想知道信用卡利率，但LLM优化后通过知识库搜索的文档也可能包含之前聊天内容的关于定期存款的内容。

此时就需要设置更好的提示词，让LLM 根据对话生成最优的搜索查询：

“请构造一个搜索查询，用于检索可用于回复用户的相关文档。”

**HyDE：假设性文档 embedding**

先用LLM预先为用户的问题生成多个假设性的回答，再用假设性回答去进行检索而不是使用用户查询。这里假定文档或片段将比用户查询本身具有更大的语义相似性

## 检索后优化

**使用重排模型**

在获得检索后的文档后，由于这些文档与问题的相关性等参差不齐，所以组合这些文档时的顺序对最终的结果也是有很大影响。比如，研究表明，将最相关的文档放置在提示中最接近查询的位置可以提高 RAG 的性能。

针对这个现象，重排序模型优化了给定查询的块搜索结果的优先级。

# \_\_await\_\_魔法方法

\_\_await\_\_魔法方法用于定义可等待对象（awaitable object）的行为；当在异步函数中使用await关键字等待一个对象时，实际上就是在调用该对象的\_\_await\_\_方法

# Python设置环境变量

os.environ['DASHSCOPE\_API\_KEY'] 是用于在 Python 程序中设置环境变量 DASHSCOPE\_API\_KEY，且是在当前 Python 进程中临时设置环境变量，只在当前python进程内有效，而不是在操作系统层面永久设置。

api\_key = os.environ.get('DASHSCOPE\_API\_KEY') ：从环境变量中获取 API Key

# LiteLLM

Litellm是一个轻量级的python库，用于简化与多个大型语言模型（LLM）API 的交互，支持 OpenAI、Anthropic、Hugging Face 等主流厂商的模型，提供统一的调用接口，帮助开发者快速切换和测试不同模型，同时具备请求路由、负载均衡、错误处理等高级功能。

1. 统一 API 接口

文本

AI 生成的内容可能不正确。

上面这种是直接与llm对话，下面试着通过litellm来创建llm对象

由于litellm不支持国内的大模型，但是如果使用的是兼容OpenAI的国内大模型，那么可以在model参数中加上前缀‘openai/’，以便litellm 知道您正在尝试调用 openai /chat/completions 终端节点。可以通过这种方法来调用国内大模型。

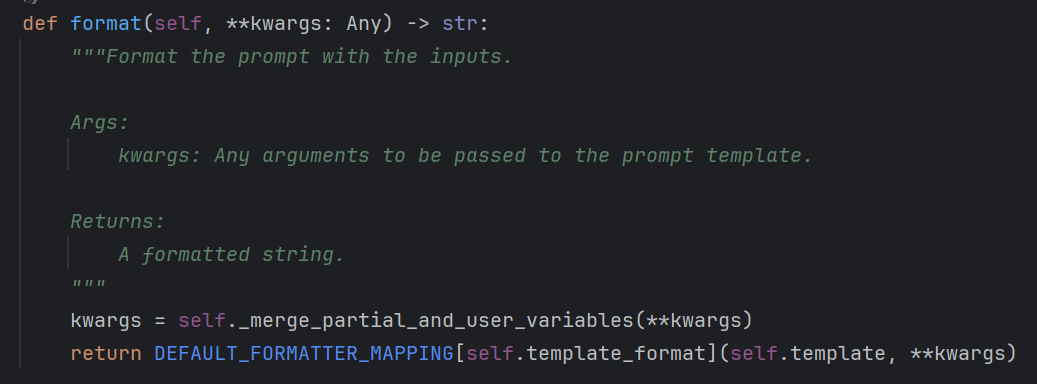
详见：https://docs.litellm.ai/docs/providers/openai\_compatible

文本

AI 生成的内容可能不正确。

# PromptTemplate

原生的PromptTemplate中的format方法，作用是将模板中的变量替换为实际值，生成最终的提示文本

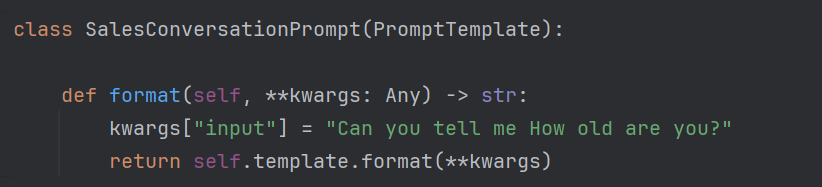


Format方法接受一个关键字参数，每个参数对应模板中的一个变量。

当你在chain中使用promptTemplate，并调用chain.invoke({"input":"Who are you?"})

那么底层会自动调用promptTemplate的format方法，将{"input":"Who are you?"}组合成kwargs传递给format方法。

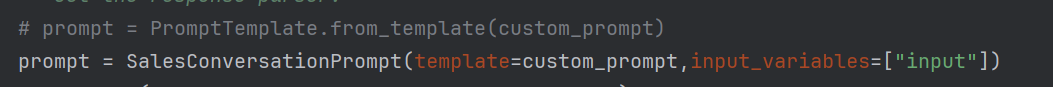
## 自定义合适的format规则



自己创建一个类让其继承PromptTemplate，重写format；

如上图代码功能：将input全部替换为“Can you tell me How old are you?

”



在创建chain时将原先的PromptTemplate替换成自定义的SalesConversationPrompt类，这样在chain中调用chain.invoke({“input”：xxx})，无论xxx为什么，后面在替换时都只会替换为Can you tell me How old are you?

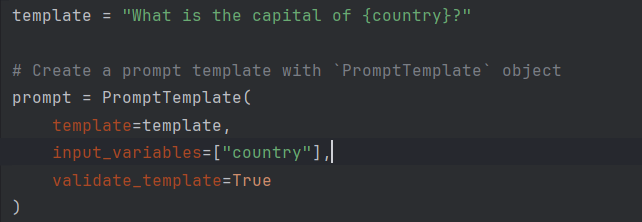
## input\_variables作用

input\_variabes是对模板中可被替换的变量加以明确，PromptTemplate可通过该变量对template进行验证。默认是不进行验证的。

文本

AI 生成的内容可能不正确。

这是PromptTemplate中的验证方法，当属性validate\_template为true时才会进行验证（默认为false），将input\_variables和partial\_variable中的元素加一块跟template中的变量进行比较来进行验证。



## partial\_variables 的作用

partial\_variables 参数用于为模板中的部分变量预先设置固定值。避免每次都重复传入。

文本

AI 生成的内容可能不正确。

好处：在定义prompt时，就在partial\_variables中指定了city的值，所以在后面每次进行format时，就只需要传入input\_variables中的变量值了，简化了参数量。

Format方法：会合并partial\_variables中变量和用户传入的变量，再对template格式化为最终字符串。

【注】如果是自定义的一个继承了PromptTemplate的子类，那么在重写format时要加上代码：kwargs = self.\_merge\_partial\_and\_user\_variables(\*\*kwargs)，这段代码就是合并partial\_variables中变量和用户传入的变量，否则在format时partial\_variables中定义的变量不会进行格式化。

# Python函数的多种传参形式

**位置参数**

文本

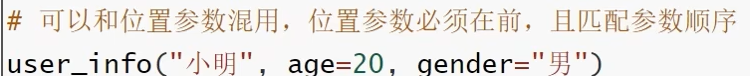
AI 生成的内容可能不正确。

**关键字参数**

图片包含 图表

AI 生成的内容可能不正确。

**位置参数与关键字参数混用**



【注】虽然可以混用，但位置参数必须在关键字参数的前面

**缺省参数**

文本

AI 生成的内容可能不正确。

在函数中给形参一个默认值，那么在调用时就可以省略有默认值的参数

**不定长参数（可变参数）**

不定长参数又分为：位置传递和关键字传递

位置传递：

图形用户界面, 文本, 应用程序

AI 生成的内容可能不正确。

传递给\*args的参数，会被转化为元组tuple，如（‘TOM’），（‘TOM’，18）

调用时参数的数量不限，但参数要以位置参数的形式传递

关键字传递：

图形用户界面, 文本, 应用程序, 聊天或短信

AI 生成的内容可能不正确。

传递给\*\*kwargs的参数，会被转化为字典对象，如{‘name’:’TOM’,’age’:18,’id’:110}

调用时参数的数量不限，但参数要以关键字参数的形式（也就是K-V）传递

**混用**

参数顺序为：位置参数🡪不定长位置参数🡪关键字参数🡪不定长关键字参数

文本, 聊天或短信

AI 生成的内容可能不正确。

# 自定义OutputParser

自定义outputParser需要继承BaseOutputParser类，并实现parse方法

文本

AI 生成的内容可能不正确。

在LCEL链式调用时，outputParser对象会自动解析llm输出，并将content内容传递给parse方法进行转换。

电脑萤幕画面

AI 生成的内容可能不正确。

可以看到最后的输出结果转换成了一个list

另外，BaseOutputParser对象还有一个get\_format\_instructions方法，用于生成格式化指令，用法如下：

文本

AI 生成的内容可能不正确。

在创建提示词模版时，指定模型的输出格式，方便后面的parse解析

# Agent的invoke

第一步，导进chains\base.py的invoke函数，在这里经过简单的输入验证\_validate\_inputs(inputs)，然后进入self.\_call

文本

AI 生成的内容可能不正确。

\_validate\_inputs(inputs)：判断inputs是否是字典，如果不是字典那么判断input中变量数是否为1，如果超过一个则报错。（也就是说chain能够接受一个变量的字符串输入，但超过一个变量那么就得用字典形式给出）