1. Langchain-Prompt提示词

1.1 Prompt（提示）是什么?

Prompt（提示）是一种指导语言模型生成特定类型输出的文本模板。

它通常包含一些预设的文本框架，以及一些变量占位符，当实际运行时，这些占位符会被具体的值所替换。

Prompt的作用在于引导语言模型按照预期的方式生成回复，确保输出符合特定的格式、语境或者目的。

1.2 Langchain使用Prompt

1.2.1 单个变量

PromptTemplate 是一个基础的模板类，用于构建格式化的提示字符串。它允许你定义一个模板字符串，并在运行时根据传入的变量替换其中的占位符。

对于单个变量（句子里有一个占位符可以自定义）可以通过以下两种方式格式化Prompt。

```
from langchain_core.prompts import PromptTemplate

# 定义一个包含单个变量的模板字符串
template = """
今天{location}天气如何？
"""

# 方式一
# 使用 PromptTemplate 类从模板字符串创建一个提示对象
# from_template 方法可以直接从字符串模板创建 PromptTemplate 对象
prompt = PromptTemplate.from_template(template)

# 使用 format 方法将变量值填入模板中，并打印结果
print(prompt.format(location="北京"))

# 方式二
# 另一种方式创建 PromptTemplate 对象
```

```
# 显式指定输入变量和模板字符串
prompt = PromptTemplate(input_variables=["location"],
template=template)

# 使用 format 方法将变量值填入模板中，并打印结果
print(prompt.format(location="北京"))
```

1.2.2 多个变量

PromptTemplate 是一个基础的模板类，用于构建格式化的提示字符串。

它允许你定义一个模板字符串，并在运行时根据传入的变量替换其中的占位符。

对于多个变量（句子里有多个占位符可以自定义）可以通过以下两种方式格式化 Prompt。

```
from langchain_core.prompts import PromptTemplate
# 多个变量
template = """
{date}{location}天气如何？
"""
prompt = PromptTemplate.from_template(template)
print(prompt.format(date="今天",location="北京"))

prompt = PromptTemplate(input_variables=["date","location"],
template=template)
print(prompt.format(date="今天", location="北京"))
```

1.2.3 聊天提示模板

- ChatPromptTemplate 是一个用于创建复杂聊天对话提示的模板，它允许将多种不同类型的消息（例如系统消息、AI 消息和用户消息）结合起来形成一个完整的对话流。使用这个类，你可以动态创建一个聊天会话的提示，帮助机器人基于不同的消息内容生成合适的回复。
- SystemMessagePromptTemplate 是一种特殊的模板，用于创建系统消息。系统消息通常用于设置聊天机器人的角色或定义对话的初始条件。例如，你可以用它告诉模型它的角色是“一个专业的助理”。
- AIMessagePromptTemplate 用于创建 AI 消息的模板，这类消息通常代表机器人的回复内容。通过使用这个类，你可以为机器人的输出指定一个格式或结构，确保生成的回复符合预期。

- HumanMessagePromptTemplate 用于创建用户输入（也称为人类消息）的模板。它帮助将用户的输入嵌入到聊天提示中，通常作为对话的起点或反馈信息。
- AIMessage, HumanMessage, SystemMessage 这三种类是 langchain.schema 模块中的消息类，分别用于表示 AI 消息、人类消息和系统消息。这些类通常用于标识对话中的消息来源，帮助构建更复杂的对话流。

PromptTemplate形式和Message的区别在于是否可以在中间挖空。使用Message就是直接写死，使用PromptTemplate就是使用模板从而动态调整提示词。

```
# -*- coding utf-8 -*-

from langchain.prompts import (
    ChatPromptTemplate,
    SystemMessagePromptTemplate,
    AIMessagePromptTemplate,
    HumanMessagePromptTemplate,
)

# Step 1: 创建一个系统消息，用于定义机器人的角色
system_message = SystemMessagePromptTemplate.from_template(
    "你是一个专业的助理，帮助用户学习编程。"
)
# additional_kwargs 的用途：可以用来存储消息的额外信息，例如消息的时间戳、消息来源、消息类型等。
system_message.additional_kwargs = {
    'timestamp': '2024-11-02T10:50:00Z',
    'source': 'system'
}

# Step 2: 创建一个人类消息，用于接收用户的输入
human_message = HumanMessagePromptTemplate.from_template(
    "用户问: {user_question}"
)

# Step 3: 创建一个AI消息，用于输出机器人的回复（一般可能用于工具调用）
ai_message = AIMessagePromptTemplate.from_template("")

# Step 4: 将这些模板结合成一个完整的聊天提示
chat_prompt = ChatPromptTemplate.from_messages([
    system_message,
    human_message,
    ai_message,
```

```
])

# Step 5: 格式化最终的对话
# 假设用户输入问题为 "如何开始学习Python？"
user_input = "如何开始学习Python？"
print(chat_prompt.format_messages(user_question=user_input))
```

```
# 聊天提示模板方式二
from langchain.schema import (
    AIMessage,
    HumanMessage,
    SystemMessage
)

from langchain.prompts import ChatPromptTemplate

# additional_kwargs 的用途：可以用来存储消息的额外信息，例如消息的时间戳、消息来源、消息类型等。
# 将模板结合成一个完整的聊天提示
chat_prompt = ChatPromptTemplate.from_messages([
    SystemMessage(content='你是一个专业的助理，帮助用户学习编程。',
additional_kwargs={
        'timestamp': '2024-11-02T10:50:00Z',
        'source': 'system'
    }),
    HumanMessage(content='用户问: 如何开始学习Python？',
additional_kwargs={}),
    AIMessage(content='', additional_kwargs={}),
])

# 格式化最终的对话
print(chat_prompt.format_messages())
```

实际上，我们发现LangChain的Prompt其实没什么作用，它的格式化我们自己也可以直接format实现，用它的反而变得困难起来了。