第二章 模型，提示和输出解释器

本章我们将简要介绍关于 LLM 开发的一些重要概念：模型、提示与解释器。如果您已完整学习过前面两个部分的内容，对这三个概念不会陌生。但是，在 LangChain 的定义中，对这三个概念的定义与使用又与之前有着细微的差别。我们仍然推荐您认真阅读本章，以进一步深入了解 LLM 开发。同时，如果您直接学习本部分的话，本章内容更是重要的基础。

我们首先向您演示直接调用 OpenAI 的场景，以充分说明为什么我们需要使用 LangChain。

一、直接调用OpenAI

1.1 计算1+1

我们来看一个简单的例子，直接使用通过 OpenAI 接口封装的函数`get_completion`来让模型告诉我们：`1+1是什么？`

```
from tool import get_completion

get_completion("1+1是什么？")
```

```
'1+1等于2。'
```

1.2 用普通话表达海盗邮件

在上述简单示例中，模型`gpt-3.5-turbo`为我们提供了关于1+1是什么的答案。而现在，我们进入一个更为丰富和复杂的场景。

设想一下，你是一家电商公司的员工。你们的客户中有一位名为海盗A的特殊顾客。他在你们的平台上购买了一个榨汁机，目的是为了制作美味的奶昔。但在制作过程中，由于某种原因，奶昔的盖子突然弹开，导致厨房的墙上洒满了奶昔。想象一下这名海盗的愤怒和挫败之情。他用充满海盗特色的英语方言，给你们的客服中心写了一封邮件：`customer_email`。

```
customer_email = """
嗯呐，我现在可是火冒三丈，我那个搅拌机盖子竟然飞了出去，把我厨房的墙壁都溅上了果汁！
更糟糕的是，保修条款可不包括清理我厨房的费用。
伙计，赶紧给我过来！
"""
```

在处理来自多元文化背景的顾客时，我们的客服团队可能会遇到某些特殊的语言障碍。如上，我们收到了一名海盗客户的邮件，而他的表达方式对于我们的客服团队来说略显生涩。

为了解决这一挑战，我们设定了以下两个目标：

- 首先，我们希望模型能够将这封充满海盗方言的邮件翻译成普通话，这样客服团队就能更容易地理解其内容。
- 其次，在进行翻译时，我们期望模型能采用平和和尊重的语气，这不仅能确保信息准确传达，还能保持与顾客之间的和谐关系。

为了指导模型的输出，我们定义了一个文本表达风格标签，简称为`style`。

```
# 普通话 + 平静、尊敬的语调
style = """正式普通话 \
用一个平静、尊敬、有礼貌的语调
"""
```

下一步我们需要做的是将`customer_email`和`style`结合起来构造我们的提示:`prompt`

```
# 要求模型根据给出的语调进行转化
prompt = f"""把由三个反引号分隔的文本\
翻译成一种{style}风格。
文本: ```{customer_email}```
"""

print("提示:", prompt)
```

```
提示:
 把由三个反引号分隔的文本翻译成一种正式普通话 用一个平静、尊敬、有礼貌的语调
风格。
文本: ```
嗯呐，我现在可是火冒三丈，我那个搅拌机盖子竟然飞了出去，把我厨房的墙壁都溅上了果汁！
更糟糕的是，保修条款可不包括清理我厨房的费用。
伙计，赶紧给我过来！
```
```

经过精心设计的`prompt`已经准备就绪。接下来，只需调用`get_completion`方法，我们就可以获得期望的输出——那封原汁原味的海盗方言邮件，将被翻译成既平和又尊重的正式普通话表达。

```
response = get_completion(prompt)
print(response)
```

```
非常抱歉，我现在感到非常愤怒和不满。我的搅拌机盖子竟然飞了出去，导致我厨房的墙壁上都溅满了果汁！更糟糕的是，保修条款并不包括清理我厨房的费用。先生/女士，请您尽快过来处理这个问题！
```

在进行语言风格转换之后，我们可以观察到明显的变化：原本的用词变得更为正式，那些带有极端情绪的表达得到了替代，并且文本中还加入了表示感激的词汇。

> 小建议：你可以调整并尝试不同的提示，来探索模型能为你带来怎样的创新性输出。每次尝试都可能为你带来意想不到的惊喜！

二、通过LangChain使用OpenAI

在前面的小节中，我们使用了封装好的函数`get_completion`，利用 OpenAI 接口成功地对那封充满方言特色的邮件进行了翻译。得到一封采用平和且尊重的语气、并用标准普通话所写的邮件。接下来，我们将尝试使用 LangChain 解决该问题。

2.1 模型

现在让我们尝试使用LangChain来实现相同的功能。从`langchain.chat_models`导入`OpenAI`的对话模型`ChatOpenAI`。 除去OpenAI以外，`langchain.chat_models`还集成了其他对话模型，更多细节可以查看 Langchain 官方文档(https://python.langchain.com/en/latest/modules/models/chat/integrations.html)。

```
from langchain.chat_models import ChatOpenAI

# 这里我们将参数temperature设置为0.0，从而减少生成答案的随机性。
# 如果你想要每次得到不一样的有新意的答案，可以尝试调整该参数。
chat = ChatOpenAI(temperature=0.0)
chat
```

```
ChatOpenAI(cache=None, verbose=False, callbacks=None, callback_manager=None, tags=None,
metadata=None, client=<class 'openai.api_resources.chat_completion.ChatCompletion'>,
model_name='gpt-3.5-turbo', temperature=0.0, model_kwargs={}, openai_api_key='sk-
IBJfPyi4LiaSSiYxEB2wT3BlbkFJjfw8KCwmJez49eVF1O1b', openai_api_base='', openai_organization='',
openai_proxy='', request_timeout=None, max_retries=6, streaming=False, n=1, max_tokens=None,
tiktoken_model_name=None)
```
```

上面的输出显示ChatOpenAI的默认模型为`gpt-3.5-turbo`

2.2 使用提示模版

在前面的例子中，我们通过f字符串把Python表达式的值`style`和`customer_email`添加到`prompt`字符串内。

`langchain`提供了接口方便快速的构造和使用提示。

2.2.1 用普通话表达海盗邮件

现在我们来看看如何使用`langchain`来构造提示吧！

```
from langchain.prompts import ChatPromptTemplate

# 首先，构造一个提示模版字符串：`template_string`
template_string = """把由三个反引号分隔的文本\
翻译成一种{style}风格。\
文本: ```{text}```
"""

# 然后，我们调用`ChatPromptTemplatee.from_template()`函数将
# 上面的提示模版字符`template_string`转换为提示模版`prompt_template`

prompt_template = ChatPromptTemplate.from_template(template_string)


print("\n", prompt_template.messages[0].prompt)
```

```
 input_variables=['style', 'text'] output_parser=None partial_variables={} template='把由三个反引号分隔的文本翻译成一种{style}风格。文本: ```{text}```\n' template_format='f-string' validate_template=True
```

对于给定的`customer_style`和`customer_email`，我们可以使用提示模版`prompt_template`的`format_messages`方法生成想要的客户消息`customer_messages`。

提示模版`prompt_template`需要两个输入变量： `style` 和 `text`。 这里分别对应

- `customer_style`：我们想要的顾客邮件风格
- `customer_email`：顾客的原始邮件文本。

```
customer_style = """正式普通话 \
用一个平静、尊敬的语气
"""

customer_email = """
嗯呐，我现在可是火冒三丈，我那个搅拌机盖子竟然飞了出去，把我厨房的墙壁都溅上了果汁！
更糟糕的是，保修条款可不包括清理我厨房的费用。
伙计，赶紧给我过来！
"""

# 使用提示模版
customer_messages = prompt_template.format_messages(
                    style=customer_style,
                    text=customer_email)
# 打印客户消息类型
print("客户消息类型:",type(customer_messages),"\n")

# 打印第一个客户消息类型
print("第一个客户客户消息类型类型:", type(customer_messages[0]),"\n")

# 打印第一个元素
```

```
print("第一个客户客户消息类型类型: ", customer_messages[0],"\n")
```

```
客户消息类型:
 <class 'list'>

第一个客户客户消息类型类型:
 <class 'langchain.schema.messages.HumanMessage'>

第一个客户客户消息类型类型:
 content='把由三个反引号分隔的文本翻译成一种正式普通话 用一个平静、尊敬的语气\n风格。文本: ```\n嗯呐，我现在可是火冒三丈，我那个搅拌机盖子竟然飞了出去，把我厨房的墙壁都溅上了果汁！\n更糟糕的是，保修条款可不包括清理我厨房的费用。\n伙计，赶紧给我过来！\n```\n' additional_kwargs={} example=False
```

可以看出

- `customer_messages`变量类型为列表(`list`)
- 列表里的元素变量类型为langchain自定义消息(`langchain.schema.HumanMessage`)。

现在我们可以调用模型部分定义的`chat`模型来实现转换客户消息风格。

```
customer_response = chat(customer_messages)
print(customer_response.content)
```

```
非常抱歉，我现在感到非常愤怒。我的搅拌机盖子竟然飞了出去，导致我厨房的墙壁上都溅满了果汁！更糟糕的是，保修条款并不包括清理我厨房的费用。伙计，请你尽快过来帮我解决这个问题！
```

2.2.2 用海盗方言回复邮件

到目前为止，我们已经实现了在前一部分的任务。接下来，我们更进一步，将客服人员回复的消息，转换为海盗风格英语，并确保消息比较有礼貌。 这里，我们可以继续使用起前面构造的的langchain提示模版，来获得我们回复消息提示。

```
service_reply = """嘿，顾客， \
保修不包括厨房的清洁费用， \
因为您在启动搅拌机之前 \
忘记盖上盖子而误用搅拌机，\
这是您的错。 \
倒霉！ 再见！
"""

service_style_pirate = """\
一个有礼貌的语气 \
使用海盗风格\
"""
service_messages = prompt_template.format_messages(
    style=service_style_pirate,
    text=service_reply)

print("\n", service_messages[0].content)
```

```
 把由三个反引号分隔的文本翻译成一种一个有礼貌的语气 使用海盗风格风格。文本: ```嘿，顾客， 保修不包括厨房的清洁费用， 因为您在启动搅拌机之前 忘记盖上盖子而误用搅拌机，这是您的错。 倒霉！ 再见！
```
```

```
# 调用模型部分定义的chat模型来转换回复消息风格
service_response = chat(service_messages)
print(service_response.content)
```
```

嘿，尊贵的客户啊，保修可不包括厨房的清洁费用，因为您在启动搅拌机之前竟然忘记盖上盖子而误用了搅拌机，这可是您的疏忽之过啊。真是倒霉透顶啊！祝您一路顺风！

2.2.3 为什么需要提示模版

在应用于比较复杂的场景时，提示可能会非常长并且包含涉及许多细节。**使用提示模版，可以让我们更为方便地重复使用设计好的提示。**英文版提示2.2.3 给出了作业的提示模版案例：学生们线上学习并提交作业，通过提示来实现对学生的提交的作业的评分。

此外，LangChain还提供了提示模版用于一些常用场景。比如自动摘要、问答、连接到SQL数据库、连接到不同的API。通过使用LangChain内置的提示模版，你可以快速建立自己的大模型应用，而不需要花时间去设计和构造提示。

最后，我们在建立大模型应用时，通常希望模型的输出为给定的格式，比如在输出使用特定的关键词来让输出结构化。英文版提示2.2.3 给出了使用大模型进行链式思考推理结果示例 -- 对于问题：*What is the elevation range for the area that the eastern sector of the Colorado orogeny extends into?* 通过使用LangChain库函数，输出采用"Thought"（思考）、"Action"（行动）、"Observation"（观察）作为链式思考推理的关键词，让输出结构化。

2.3 输出解析器

2.3.1 不使用输出解释器提取客户评价中的信息

对于给定的评价`customer_review`，我们希望提取信息，并按以下格式输出：

```
{
  "gift": False,
  "delivery_days": 5,
  "price_value": "pretty affordable!"
}
```

```
from langchain.prompts import ChatPromptTemplate

customer_review = """\
这款吹叶机非常神奇。 它有四个设置：\
吹蜡烛、微风、风城、龙卷风。 \
两天后就到了，正好赶上我妻子的\
周年纪念礼物。 \
我想我的妻子会喜欢它到说不出话来。 \
到目前为止，我是唯一一个使用它的人，而且我一直\
每隔一天早上用它来清理草坪上的叶子。 \
它比其他吹叶机稍微贵一点，\
但我认为它的额外功能是值得的。
"""

review_template = """\
对于以下文本，请从中提取以下信息：

礼物：该商品是作为礼物送给别人的吗？ \
如果是，则回答 是的；如果否或未知，则回答 不是。

交货天数：产品需要多少天\
到达？ 如果没有找到该信息，则输出-1。

价钱：提取有关价值或价格的任何句子，\
并将它们输出为逗号分隔的 Python 列表。

使用以下键将输出格式化为 JSON：
礼物
交货天数
价钱

文本: {text}
```

```
"""

prompt_template = ChatPromptTemplate.from_template(review_template)
print("提示模版：", prompt_template)


messages = prompt_template.format_messages(text=customer_review)


chat = ChatOpenAI(temperature=0.0)
response = chat(messages)

print("结果类型:", type(response.content))
print("结果:", response.content)
```

```
提示模版：
 input_variables=['text'] output_parser=None partial_variables={} messages=
[HumanMessagePromptTemplate(prompt=PromptTemplate(input_variables=['text'],
output_parser=None, partial_variables={}, template='对于以下文本，请从中提取以下信息：\n\n礼物：该商品
是作为礼物送给别人的吗？ 如果是，则回答 是的；如果否或未知，则回答 不是。\n\n交货天数：产品需要多少天到达？ 如果
没有找到该信息，则输出-1。\n\n价钱：提取有关价值或价格的任何句子，并将它们输出为逗号分隔的 Python 列表。\n\n使
用以下键将输出格式化为 JSON：\n礼物\n交货天数\n价钱\n\n文本: {text}\n', template_format='f-string',
validate_template=True), additional_kwargs={})]

结果类型:
 <class 'str'>

结果:
 {
  "礼物": "是的",
  "交货天数": 2,
  "价钱": ["它比其他吹叶机稍微贵一点"]
}
```

可以看出`response.content`类型为字符串（`str`），而并非字典(`dict`), 如果想要从中更方便的提取信息，我们需要使用`Langchain`中的输出解释器。

2.3.2 使用输出解析器提取客户评价中的信息

接下来，我们将展示如何使用输出解释器。

```
review_template_2 = """\
对于以下文本，请从中提取以下信息：：

礼物：该商品是作为礼物送给别人的吗？
如果是，则回答 是的；如果否或未知，则回答 不是。

交货天数：产品到达需要多少天？ 如果没有找到该信息，则输出-1。

价钱：提取有关价值或价格的任何句子，并将它们输出为逗号分隔的 Python 列表。

文本: {text}

{format_instructions}
"""

prompt = ChatPromptTemplate.from_template(template=review_template_2)

from langchain.output_parsers import ResponseSchema
from langchain.output_parsers import StructuredOutputParser

gift_schema = ResponseSchema(name="礼物",
```

```
                                description="这件物品是作为礼物送给别人的吗？\
                                如果是，则回答 是的，\
                                如果否或未知，则回答 不是。")

delivery_days_schema = ResponseSchema(name="交货天数",
                                      description="产品需要多少天才能到达？\
                                      如果没有找到该信息，则输出-1。")

price_value_schema = ResponseSchema(name="价钱",
                                    description="提取有关价值或价格的任何句子，\
                                    并将它们输出为逗号分隔的 Python 列表")


response_schemas = [gift_schema,
                    delivery_days_schema,
                    price_value_schema]
output_parser = StructuredOutputParser.from_response_schemas(response_schemas)
format_instructions = output_parser.get_format_instructions()
print("输出格式规定：",format_instructions)
```

```
输出格式规定：
 The output should be a markdown code snippet formatted in the following schema, including the leading and trailing "```json" and "```":

```json
{
    "礼物": string  // 这件物品是作为礼物送给别人的吗？                                    如果是，则回答 是的，
                            如果否或未知，则回答 不是。
    "交货天数": string  // 产品需要多少天才能到达？                                      如果没有找到
该信息，则输出-1。
    "价钱": string  // 提取有关价值或价格的任何句子，                                    并将它们输出
为逗号分隔的 Python 列表
}
```
```

```
messages = prompt.format_messages(text=customer_review,
format_instructions=format_instructions)
print("第一条客户消息:",messages[0].content)
```

```
第一条客户消息:
 对于以下文本，请从中提取以下信息：：

礼物：该商品是作为礼物送给别人的吗？
如果是，则回答 是的；如果否或未知，则回答 不是。

交货天数：产品到达需要多少天？ 如果没有找到该信息，则输出-1。

价钱：提取有关价值或价格的任何句子，并将它们输出为逗号分隔的 Python 列表。

文本: 这款吹叶机非常神奇。 它有四个设置：吹蜡烛、微风、风城、龙卷风。 两天后就到了，正好赶上我妻子的周年纪念礼物。 我想我的妻子会喜欢它到说不出话来。 到目前为止，我是唯一一个使用它的人，而且我一直每隔一天早上用它来清理草坪上的叶子。 它比其他吹叶机稍微贵一点，但我认为它的额外功能是值得的。

The output should be a markdown code snippet formatted in the following schema, including the leading and trailing "```json" and "```":

```json
{
    "礼物": string  // 这件物品是作为礼物送给别人的吗？                                    如果是，则回答 是的，
                            如果否或未知，则回答 不是。
```
```

```
    "交货天数": string  // 产品需要多少天才能到达？                                    如果没有找到该信息，则输出-1。
    "价钱": string  // 提取有关价值或价格的任何句子，                                  并将它们输出为逗号分隔的 Python 列表
}
```

```
response = chat(messages)

print("结果类型:", type(response.content))
print("结果:", response.content)
```

```
结果类型:
 <class 'str'>

结果:
 ```json
{
    "礼物": "不是",
    "交货天数": "两天后就到了",
    "价钱": "它比其他吹叶机稍微贵一点"
}
```
```

```
output_dict = output_parser.parse(response.content)

print("解析后的结果类型:", type(output_dict))
print("解析后的结果:", output_dict)
```

```
解析后的结果类型:
 <class 'dict'>

解析后的结果:
 {'礼物': '不是', '交货天数': '两天后就到了', '价钱': '它比其他吹叶机稍微贵一点'}
```

`output_dict`类型为字典(`dict`), 可直接使用`get`方法。这样的输出更方便下游任务的处理。

三、英文版提示

1.2 用美式英语表达海盗邮件

```
customer_email = """
Arrr, I be fuming that me blender lid \
flew off and splattered me kitchen walls \
with smoothie! And to make matters worse,\
the warranty don't cover the cost of \
cleaning up me kitchen. I need yer help \
right now, matey!
"""

# 美式英语 + 平静、尊敬的语调
style = """American English \
in a calm and respectful tone
"""


# 要求模型根据给出的语调进行转化
prompt = f"""Translate the text \
that is delimited by triple backticks
```

```
into a style that is {style}.
text: ```{customer_email}```
"""

print("提示：", prompt)

response = get_completion(prompt)

print("美式英语表达的海盗邮件：", response)
```

```
提示：
 Translate the text that is delimited by triple backticks
into a style that is American English in a calm and respectful tone
.
text: ```
Arrr, I be fuming that me blender lid flew off and splattered me kitchen walls with smoothie! And to make matters worse,the warranty don't cover the cost of cleaning up me kitchen. I need yer help right now, matey!

美式英语表达的海盗邮件：
 I am quite frustrated that my blender lid flew off and made a mess of my kitchen walls with smoothie! To add to my frustration, the warranty does not cover the cost of cleaning up my kitchen. I kindly request your assistance at this moment, my friend.
```

2.2.1 用标准美式英语表达海盗邮件

```
from langchain.prompts import ChatPromptTemplate

template_string = """Translate the text \
that is delimited by triple backticks \
into a style that is {style}. \
text: ```{text}```
"""


prompt_template = ChatPromptTemplate.from_template(template_string)

print("提示模版中的第一个提示：", prompt_template.messages[0].prompt)

customer_style = """American English \
in a calm and respectful tone
"""


customer_email = """
Arrr, I be fuming that me blender lid \
flew off and splattered me kitchen walls \
with smoothie! And to make matters worse, \
the warranty don't cover the cost of \
cleaning up me kitchen. I need yer help \
right now, matey!
"""

customer_messages = prompt_template.format_messages(
                    style=customer_style,
                    text=customer_email)


print("用提示模版中生成的第一条客户消息：", customer_messages[0])
```

```
提示模版中的第一个提示：
 input_variables=['style', 'text'] output_parser=None partial_variables={} template='Translate the text that is delimited by triple backticks into a style that is {style}. text: ```{text}```\n' template_format='f-string' validate_template=True

用提示模版中生成的第一条客户消息：
 content="Translate the text that is delimited by triple backticks into a style that is American English in a calm and respectful tone\n. text: ```\nArrr, I be fuming that me blender lid flew off and splattered me kitchen walls with smoothie! And to make matters worse, the warranty don't cover the cost of cleaning up me kitchen. I need yer help right now, matey!\n```\n" additional_kwargs={} example=False
```

2.2.2 用海盗方言回复邮件

```
service_reply = """Hey there customer, \
the warranty does not cover \
cleaning expenses for your kitchen \
because it's your fault that \
you misused your blender \
by forgetting to put the lid on before \
starting the blender. \
Tough luck! See ya!
"""
service_style_pirate = """\
a polite tone \
that speaks in English Pirate\
"""

service_messages = prompt_template.format_messages(
    style=service_style_pirate,
    text=service_reply)

print("提示模版中的第一条客户消息内容：", service_messages[0].content)

service_response = chat(service_messages)
print("模型得到的回复邮件：", service_response.content)
```

```
提示模版中的第一条客户消息内容：
 Translate the text that is delimited by triple backticks into a style that is a polite tone that speaks in English Pirate. text: ```Hey there customer, the warranty does not cover cleaning expenses for your kitchen because it's your fault that you misused your blender by forgetting to put the lid on before starting the blender. Tough luck! See ya!

模型得到的回复内容：
 Ahoy there, matey! I regret to inform ye that the warranty be not coverin' the costs o' cleanin' yer galley, as 'tis yer own fault fer misusin' yer blender by forgettin' to secure the lid afore startin' it. Aye, tough luck, me heartie! Fare thee well!
```

2.3.1 不使用输出解释器提取客户评价中的信息

```
customer_review = """\
This leaf blower is pretty amazing.  It has four settings:\
candle blower, gentle breeze, windy city, and tornado. \
It arrived in two days, just in time for my wife's \
anniversary present. \
I think my wife liked it so much she was speechless. \
So far I've been the only one using it, and I've been \
using it every other morning to clear the leaves on our lawn. \
It's slightly more expensive than the other leaf blowers \
out there, but I think it's worth it for the extra features.
```

```
"""

review_template = """\
For the following text, extract the following information:

gift: Was the item purchased as a gift for someone else? \
Answer True if yes, False if not or unknown.

delivery_days: How many days did it take for the product \
to arrive? If this information is not found, output -1.

price_value: Extract any sentences about the value or price,\
and output them as a comma separated Python list.

Format the output as JSON with the following keys:
gift
delivery_days
price_value

text: {text}
"""

from langchain.prompts import ChatPromptTemplate
prompt_template = ChatPromptTemplate.from_template(review_template)

print("提示模版：",prompt_template)

messages = prompt_template.format_messages(text=customer_review)

chat = ChatOpenAI(temperature=0.0)
response = chat(messages)
print("回复内容：",response.content)
```

```
提示模版：
 input_variables=['text'] output_parser=None partial_variables={} messages=
[HumanMessagePromptTemplate(prompt=PromptTemplate(input_variables=['text'],
output_parser=None, partial_variables={}, template='For the following text, extract the
following information:\n\ngift: Was the item purchased as a gift for someone else? Answer True
if yes, False if not or unknown.\n\ndelivery_days: How many days did it take for the product
to arrive? If this information is not found, output -1.\n\nprice_value: Extract any sentences
about the value or price,and output them as a comma separated Python list.\n\nFormat the
output as JSON with the following keys:\ngift\ndelivery_days\nprice_value\n\ntext: {text}\n',
template_format='f-string', validate_template=True), additional_kwargs={})]

回复内容：
 {
  "gift": false,
  "delivery_days": 2,
  "price_value": ["It's slightly more expensive than the other leaf blowers out there, but I
think it's worth it for the extra features."]
}
```

2.3.2 使用输出解析器提取客户评价中的信息

```
review_template_2 = """\
For the following text, extract the following information:

gift: Was the item purchased as a gift for someone else? \
Answer True if yes, False if not or unknown.

delivery_days: How many days did it take for the product\
to arrive? If this information is not found, output -1.
```

```
price_value: Extract any sentences about the value or price,\
and output them as a comma separated Python list.

text: {text}

{format_instructions}
"""

prompt = ChatPromptTemplate.from_template(template=review_template_2)

from langchain.output_parsers import ResponseSchema
from langchain.output_parsers import StructuredOutputParser

gift_schema = ResponseSchema(name="gift",
                             description="Was the item purchased\
                             as a gift for someone else? \
                             Answer True if yes,\
                             False if not or unknown.")

delivery_days_schema = ResponseSchema(name="delivery_days",
                                      description="How many days\
                                      did it take for the product\
                                      to arrive? If this \
                                      information is not found,\
                                      output -1.")

price_value_schema = ResponseSchema(name="price_value",
                                    description="Extract any\
                                    sentences about the value or \
                                    price, and output them as a \
                                    comma separated Python list.")


response_schemas = [gift_schema,
                    delivery_days_schema,
                    price_value_schema]
output_parser = StructuredOutputParser.from_response_schemas(response_schemas)
format_instructions = output_parser.get_format_instructions()
print(format_instructions)

messages = prompt.format_messages(text=customer_review,
format_instructions=format_instructions)

print("提示消息：", messages[0].content)

response = chat(messages)
print("回复内容：",response.content)

output_dict = output_parser.parse(response.content)
print("解析后的结果类型:", type(output_dict))
print("解析后的结果:", output_dict)
```

```
The output should be a markdown code snippet formatted in the following schema, including the
leading and trailing "```json" and "```":

```json
{
    "gift": string  // Was the item purchased                              as a gift for
someone else?                                Answer True if yes,
False if not or unknown.
```
```

```
    "delivery_days": string  // How many days                                    did it take
for the product                                    to arrive? If this
                information is not found,                                        output -1.
    "price_value": string  // Extract any                                     sentences about
the value or                                    price, and output them as a
                  comma separated Python list.
}
```

提示消息:
```
 For the following text, extract the following information:

gift: Was the item purchased as a gift for someone else? Answer True if yes, False if not or
unknown.

delivery_days: How many days did it take for the productto arrive? If this information is not
found, output -1.

price_value: Extract any sentences about the value or price,and output them as a comma
separated Python list.

text: This leaf blower is pretty amazing.  It has four settings:candle blower, gentle breeze,
windy city, and tornado. It arrived in two days, just in time for my wife's anniversary
present. I think my wife liked it so much she was speechless. So far I've been the only one
using it, and I've been using it every other morning to clear the leaves on our lawn. It's
slightly more expensive than the other leaf blowers out there, but I think it's worth it for
the extra features.
The output should be a markdown code snippet formatted in the following schema, including the
leading and trailing "```json" and "```":

```json
{
    "gift": string  // Was the item purchased                              as a gift for
someone else?                              Answer True if yes,
False if not or unknown.
    "delivery_days": string  // How many days                                    did it take
for the product                                    to arrive? If this
                information is not found,                                        output -1.
    "price_value": string  // Extract any                                     sentences about
the value or                                    price, and output them as a
                  comma separated Python list.
}
```
```

回复内容:
```
 ```json
{
    "gift": false,
    "delivery_days": "2",
    "price_value": "It's slightly more expensive than the other leaf blowers out there, but I
think it's worth it for the extra features."
}
```
```

解析后的结果类型:
```
 <class 'dict'>
```

解析后的结果:
```
 {'gift': False, 'delivery_days': '2', 'price_value': "It's slightly more expensive than the
other leaf blowers out there, but I think it's worth it for the extra features."}
```