3-1 Model I/O 组件概述

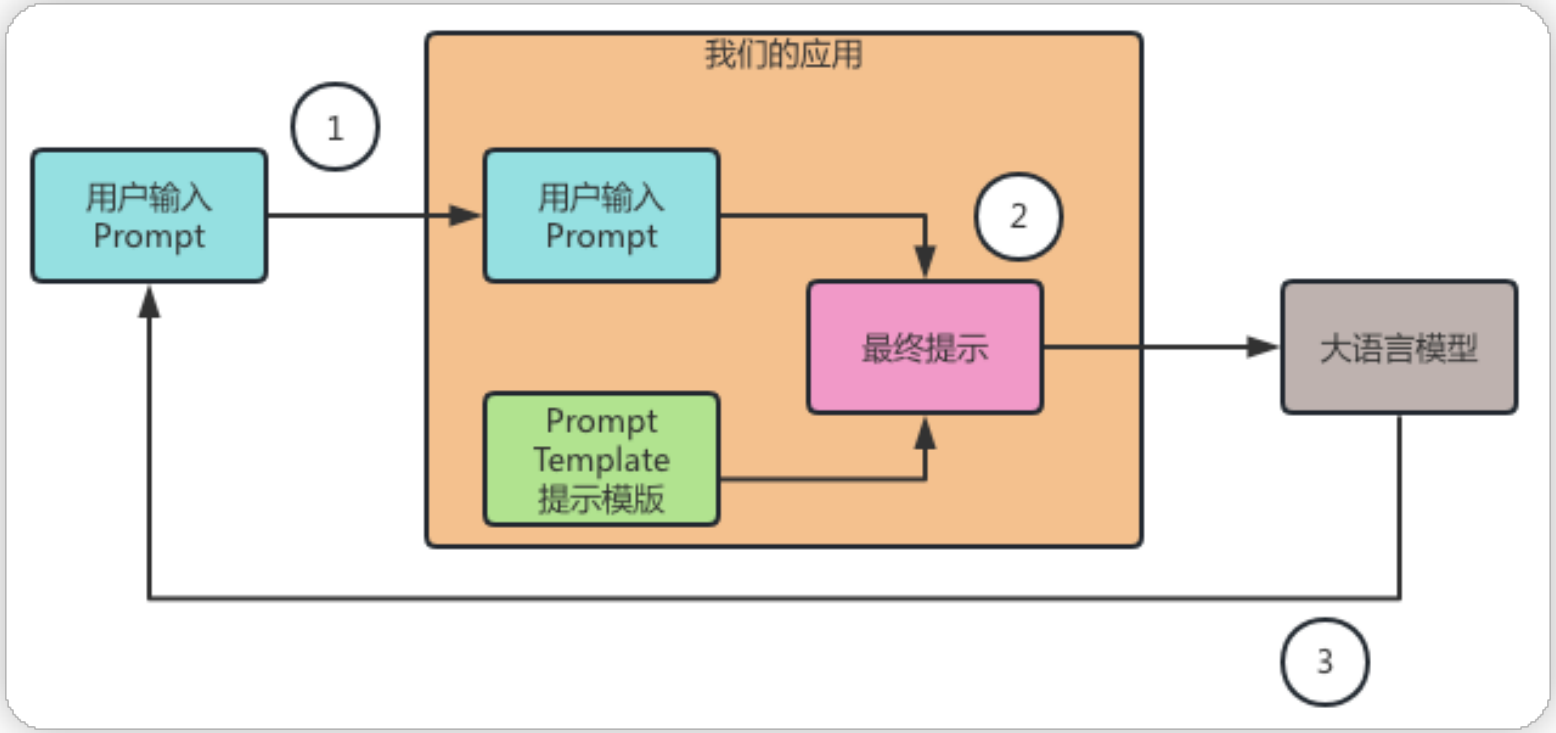
- 本章课程介绍
- Prompts 与Prompt Template
- ChatMessagePromptTemplate 通过Role实现精准回应
- MessagesPlaceholder 实现多类型Message合作

本章课程介绍

- Prompt Template（提示模版）
  - Prompts 、Prompt Template 基本介绍
  - ChatMessagePromptTemplate 精准回应
  - MessagesPlaceholder 实现多类型Message合作
  - Partial Prompt Template（模版的部分加载）
  - Composition（Prompt template 组合）
  - Serialization（序列化保存Prompt template）
- Example selectors（示例选择器）
  - Select by length（根据Prompt 控制示例长度）
  - Select by similarity（选择与Prompt 类似的示例）
- Language Models（语言模型的应用）
  - Async API（异步调用语言模型）
  - FakeListLLM（模拟LLM）
  - Caching（缓存LLM请求）
  - Serialization（保存LLM 相关配置）
  - Streaming（利用LLM实现流式输出）
- Output parsers（输出解析器）
  - List parser（列表解析）
  - Datetime parser（日期解析）
  - Auto-fixing parser （自动修复解析器）

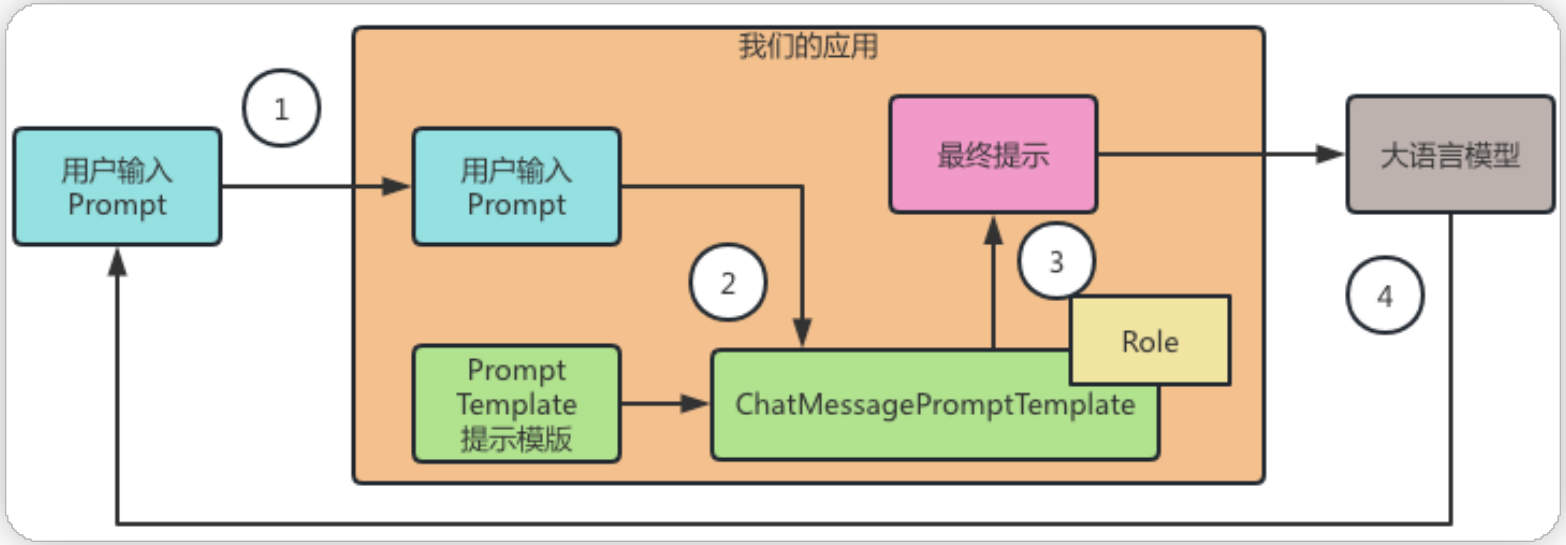
Prompts 与Prompt Template

- 1 用户输入的Prompt
- 2 讲Prompt Template定义的内容和用户输入的Prompt进行结合，生成最终提示
- 3 大语言模型根据最终提示，进行回应



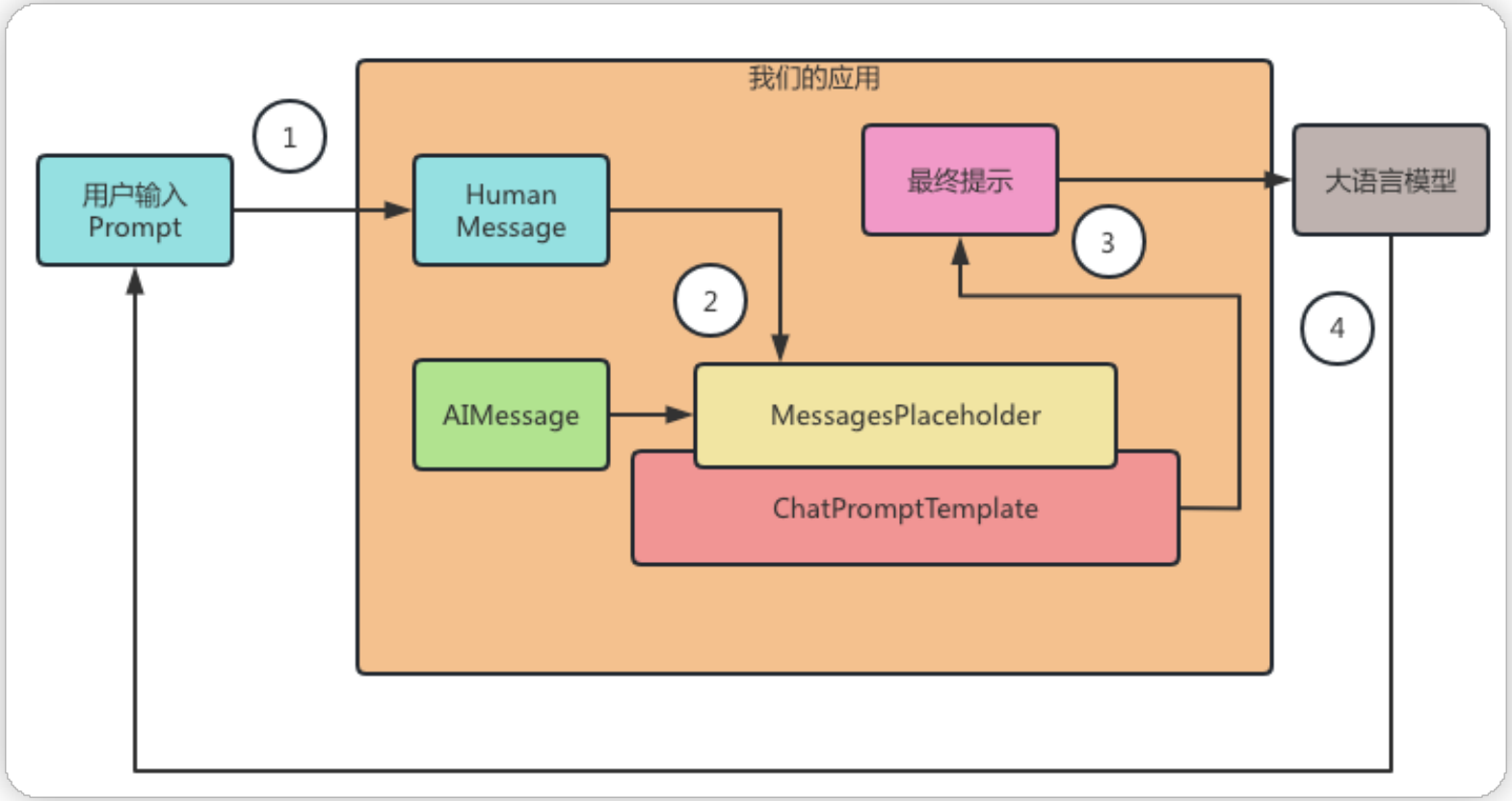
ChatMessagePromptTemplate
通过Role实现精准回应

- 1 用户输入的Prompt
- 2 ChatMessagePromptTemplate将Prompt 和Prompt Template整合，并且设置Role的方式
- 3 生成最终提示
- 4 大语言模型进行回应



MessagesPlaceholder 实现多类型Message合作

- HumanMessage
- AIMessage
- SystemMessage

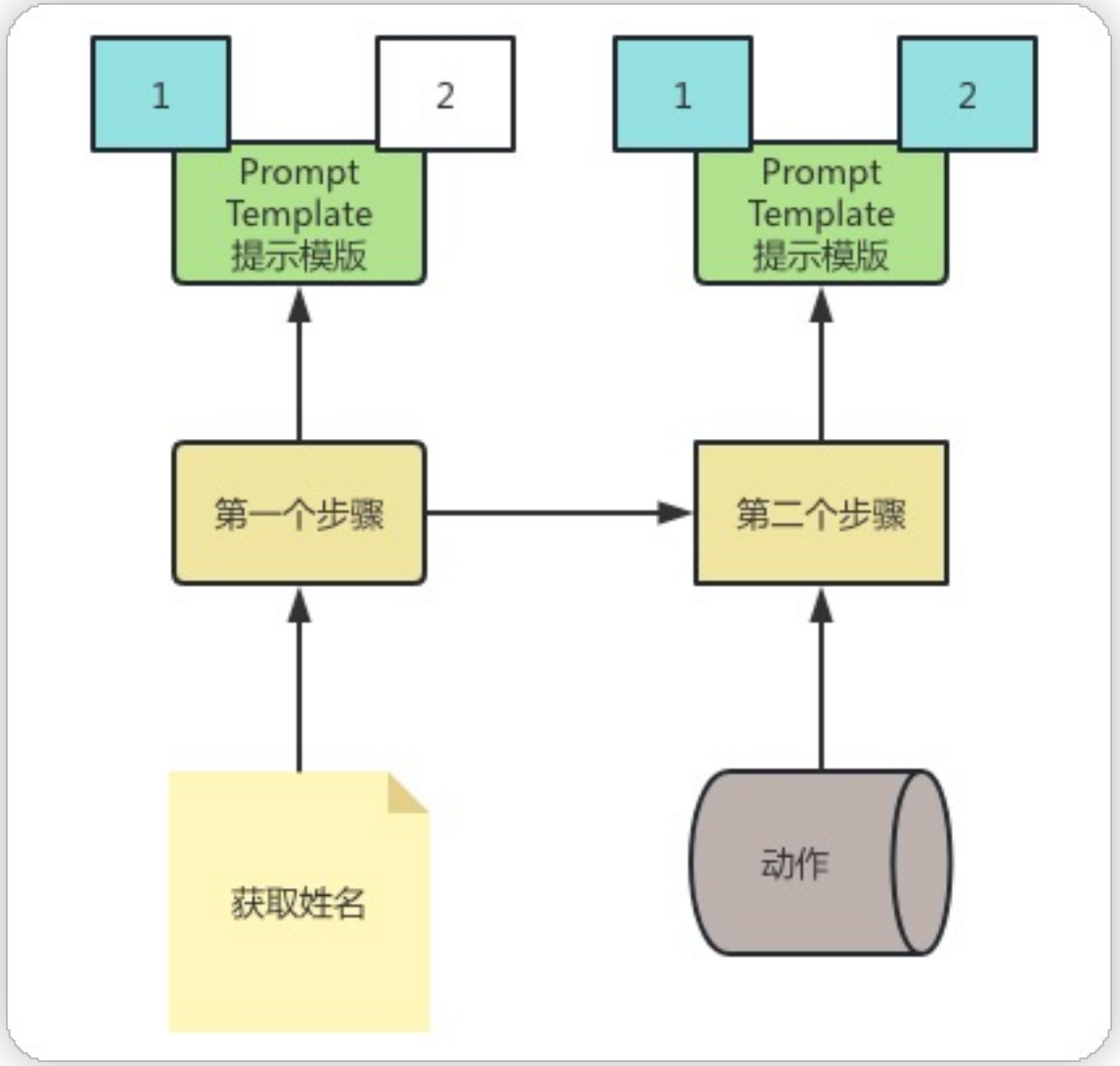


3-2 Prompt template 探索

- Partial prompt templates 模版的部分加载
- Composition 模版组合
- Serialization（序列化保存Prompt template）

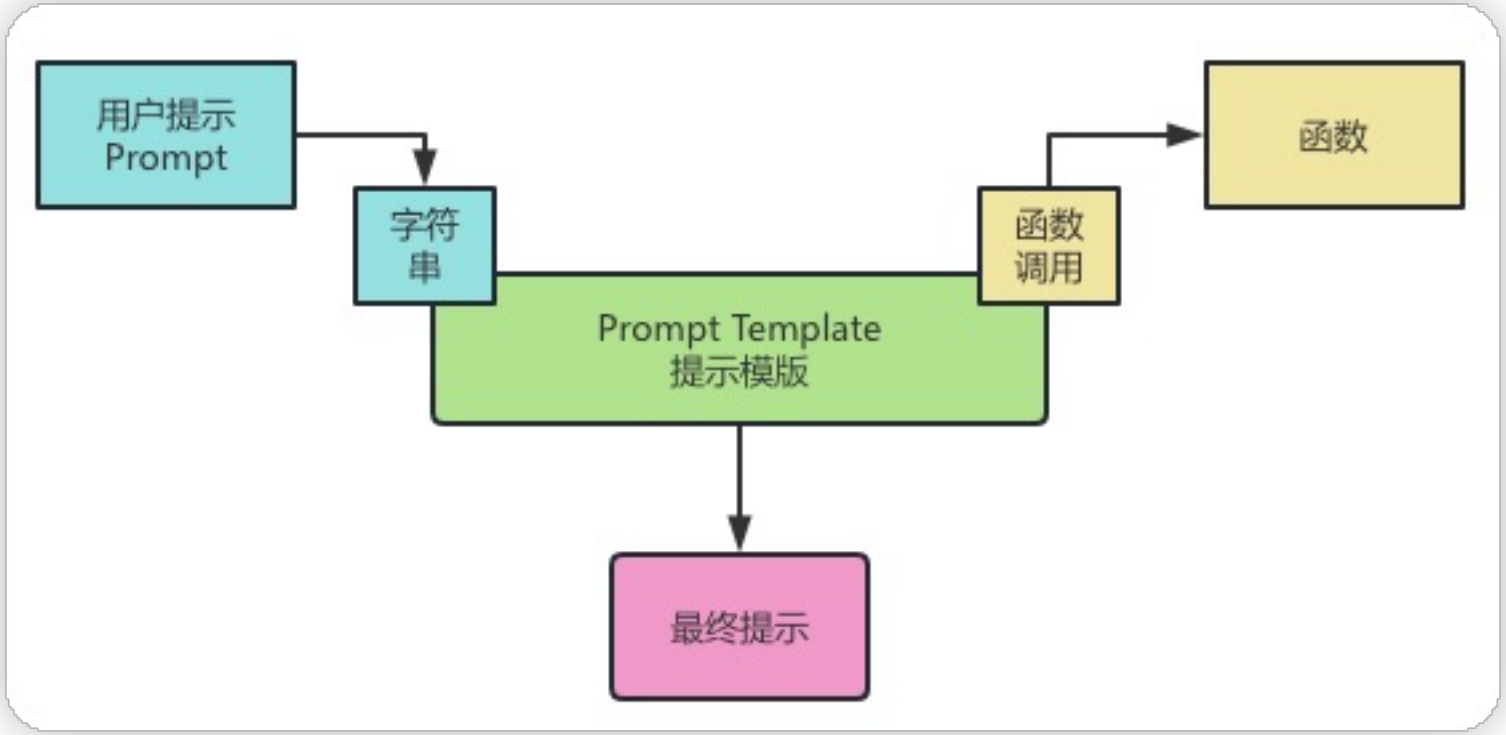
Partial prompt templates 模版的部分加载

- 字符串：Prompt Template中存在多个参数信息，无法一次获得，需要多个步骤获得的情况。



Partial prompt templates 模版的部分加载

- 函数方式：模版的部分内容依赖与动态信息，这部分信息需要通过函数生成的。



Composition 模版组合

PipelinePromptTemplate 主要在于需要创建复杂、模块化的提示时。这种情况下，你可能希望将提示分解为多个部分，每个部分都可以独立地定义和格式化，然后再将它们组合在一起。

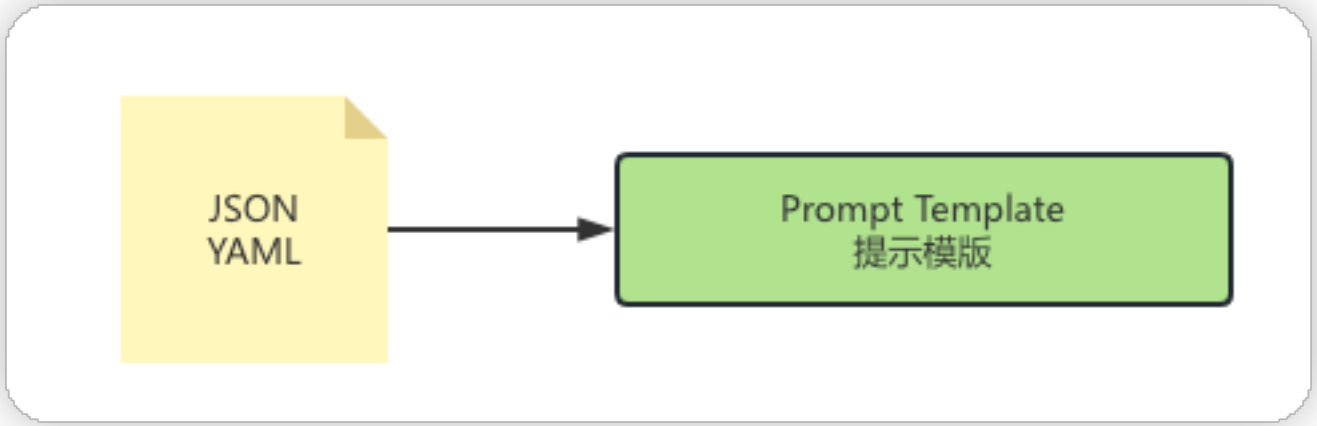
Serialization 序列化保存Prompt template

Prompt template 序列化

支持JSON和YAML两种格式。

支持在一个文件中指定所有内容，或者将不同的组件（模板、示例等）存储在不同的文件中并进行引用。

提供了一个加载提示的单一入口点



3-3 Example Selector

- Example selectors（示例选择器）
- Select by length（根据Prompt 控制示例长度）
- Select by similarity（选择与Prompt 类似的示例）

Example selectors

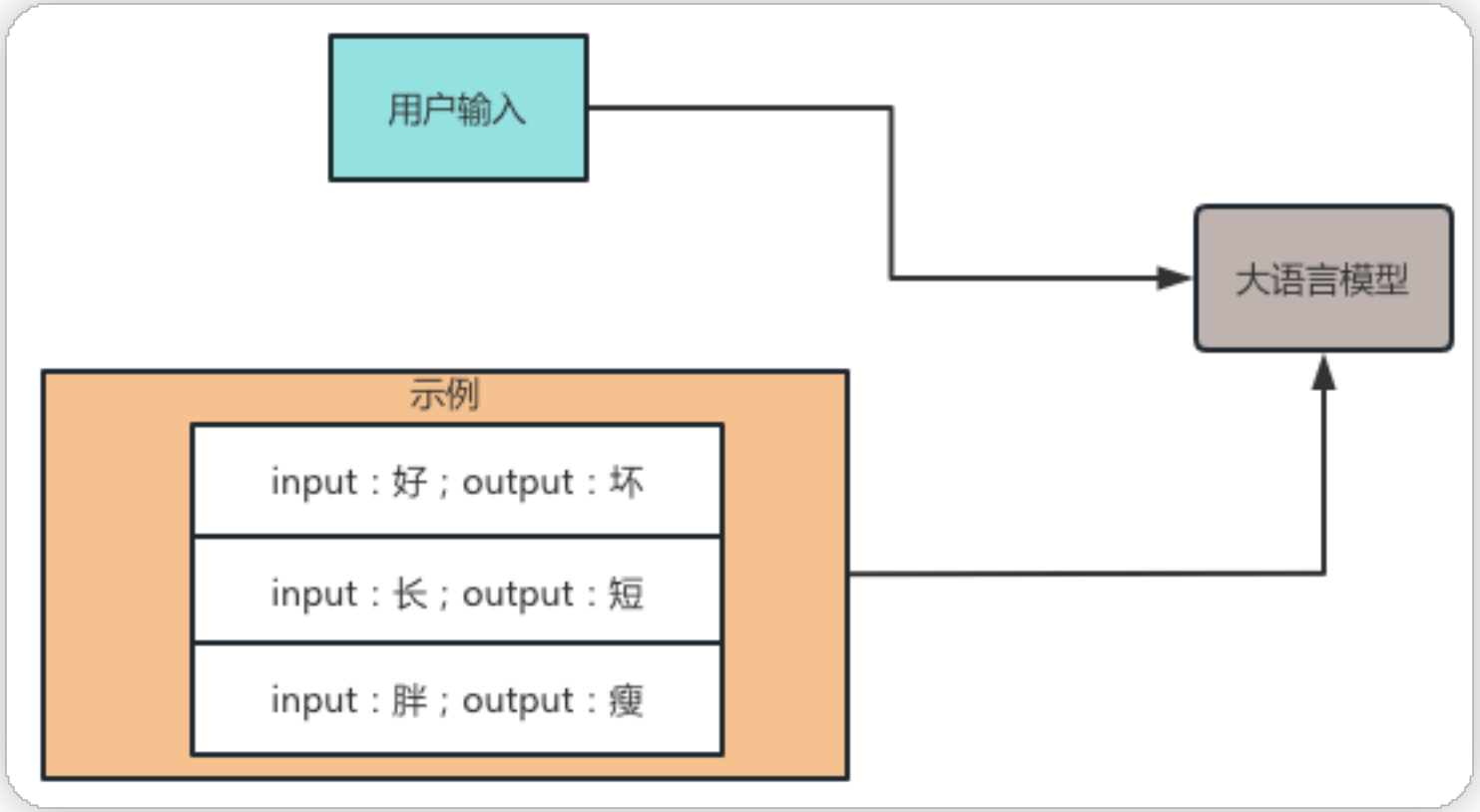
Example存在的意义：

- Prompt tuning：不用修改LLM的结构和参数，就可以提升响应销量。
- 通过加入Examples（示例）的方式唤醒LLM对某方面能力的记忆。
- 帮助模型理解任务的具体要求。

问题：

示例也会作为LLM的输入，LLM会对输入长度有限制，如果示例太长会影响 提示的输入（Prompt 用户输入）

如果存在多个示例，需要找出和输入相似度最高的进行使用，从而提升回应的效率。

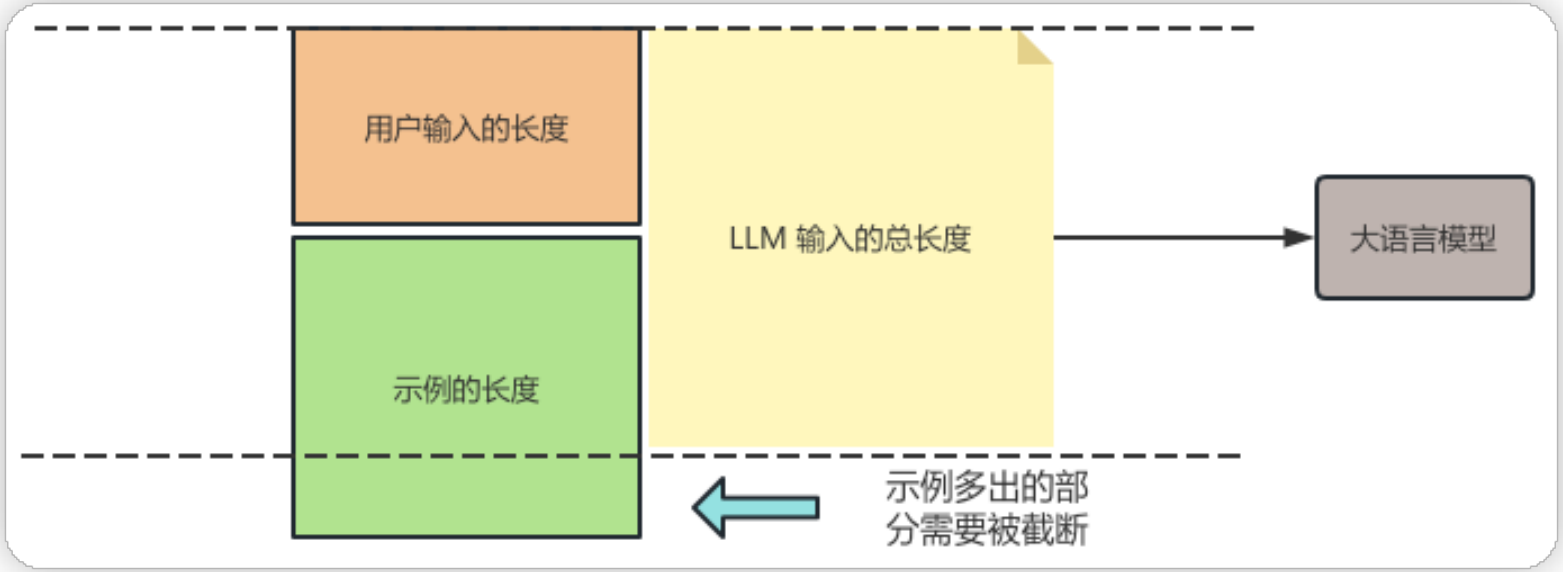


Select by length（根据Prompt 控制示例长度）

LLM 输入的总长度是一定的

LLM输入的总长度 = 用户输入的字符长度（Prompt）+示例字符的长度（Example）

如果输入的总字符长度大于LLM输入的总长度，就需要牺牲示例的长度将其截断。



Select by similarity（选择与Prompt 类似的示例）

如果存在多个示例，需要找出和输入相似度最高的进行使用，从而提升回应的效率。

如果，用户输入："学生"

前三条记录是预期相似的，如果输入前三条记录会帮助LLM更好地产生回应

```
# 创建一个示例列表
examples = [
    {"input": "老师", "output": "教室"},
    {"input": "医生", "output": "医院"},
    {"input": "司机", "output": "汽车"},
    {"input": "树", "output": "土地"},
    {"input": "鸟", "output": "鸟巢"},
]
```

3-4 Language Models（语言模型的应用）

- Language Models（语言模型的应用）
- Caching（缓存LLM输入+结果）
- FakeListLLM（模拟LLM）

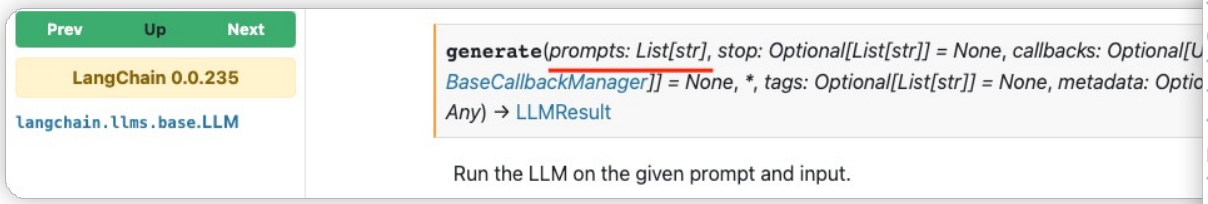
Language Models（语言模型的应用）

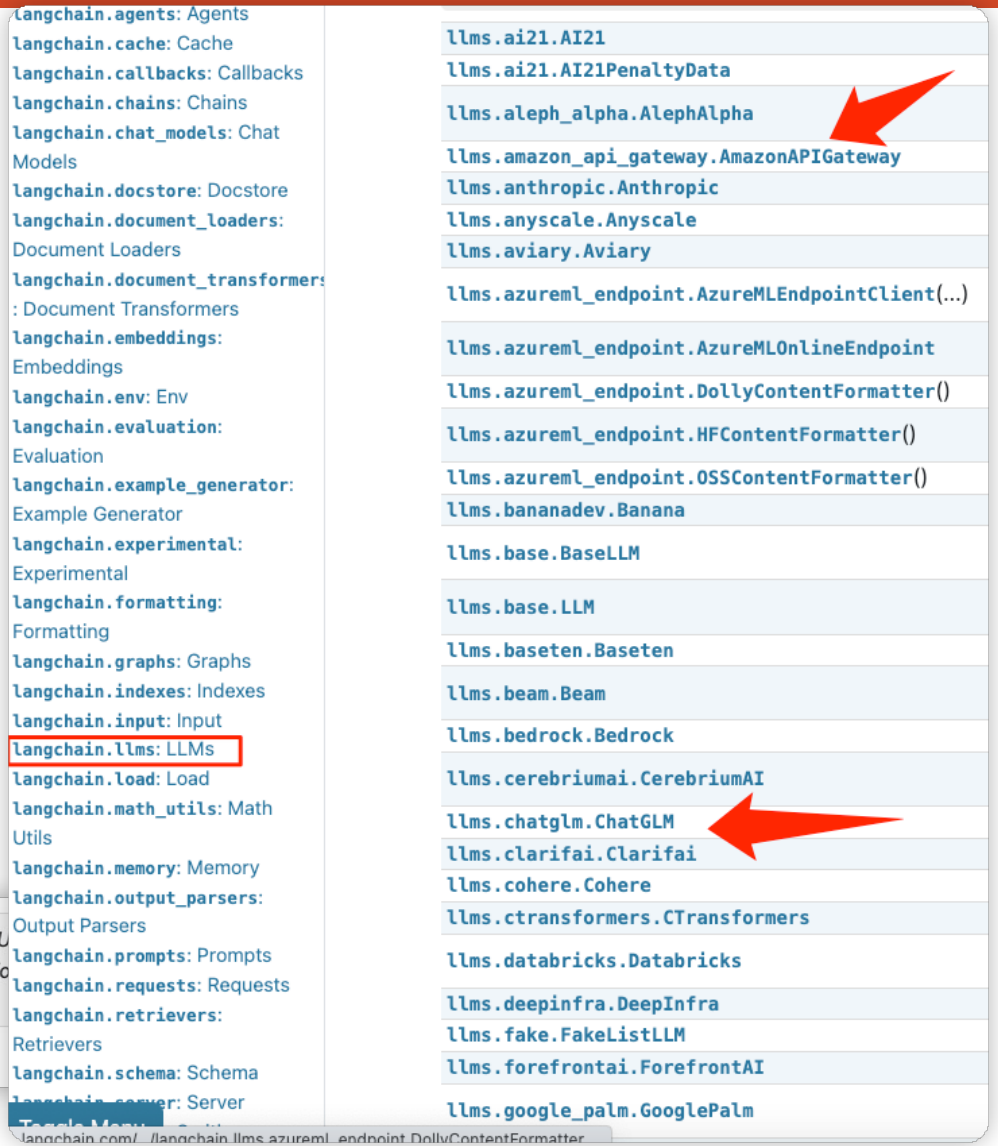
- LLMs是LangChain的核心组件，LangChain并不提供自己的LLMs，而是提供了一个与多种LLMs交互的标准接口。

- 引入包+API 的方式进行使用

```
# 导入OpenAI的LLM类
from langchain.llms import OpenAI

# 初始化LLM类，传入API密钥
llm = OpenAI(openai_api_key="你的API密钥")
```

- LLM（）
- Generate





Caching

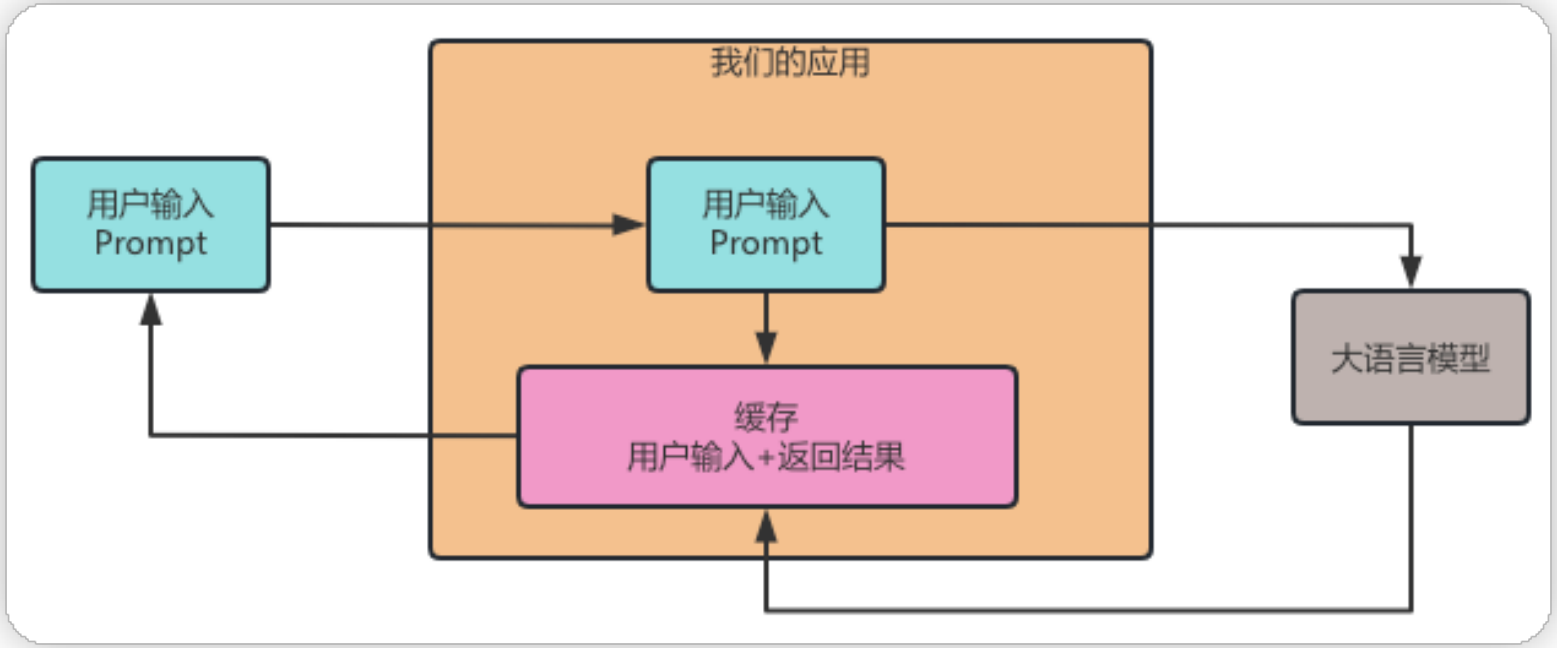
缓存在两个方面非常有用：

1.减少对LLM提供商的API调用次数来节省费用，特别是当你经常请求相同的完成多次时。

2.减少对LLM提供商的API调用次数来加速应用。

3.缓存用户输入和返回结果

4.下次用户输入的时候直接查询缓存，如果发现同样的输入，直接返回结果，不用请求 LLM



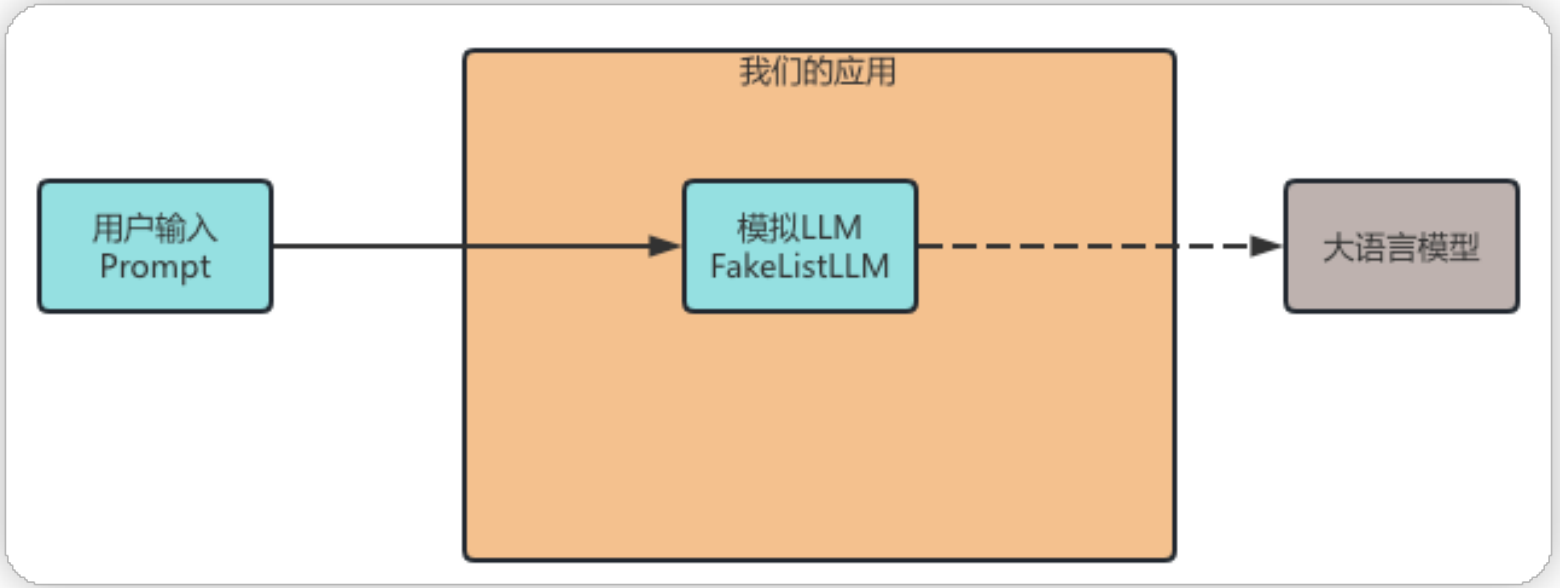
FakeListLLM（模拟LLM）

FakeListLLM可以模拟LLM的行为

测试和调试：在调试新的功能时，不用每次都实际调用LLM就可以用模拟的LLM

开发原型：开发新应用是，还不确定最终的设计方案，使用哪个LLM。就先用模拟的LLM。

离线工作：在没有互联网的环境中工作，用模拟的LLM替代真实的LLM



3-5 Language Models（语言模型的应用）

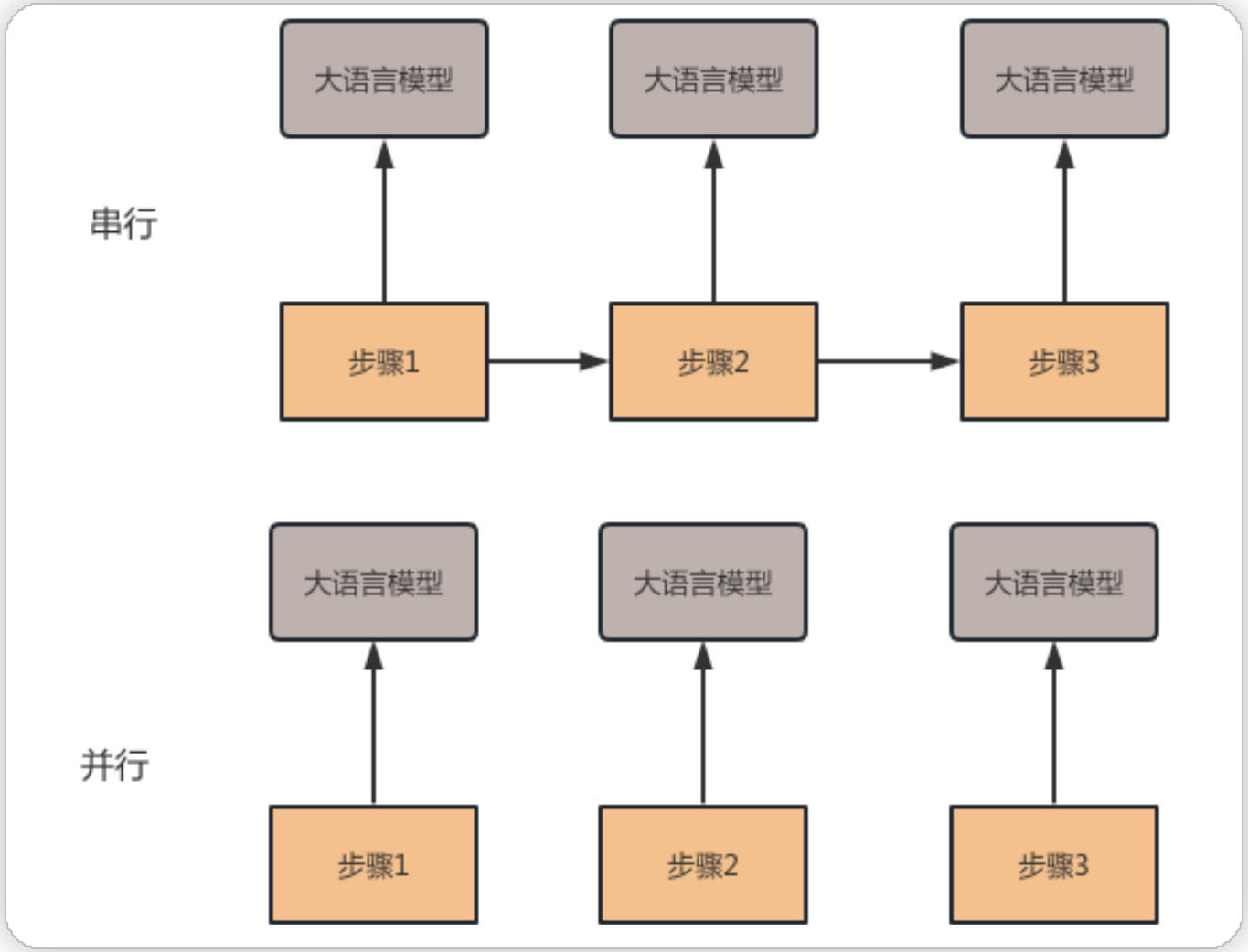
- Async API（异步调用语言模型）
- Serialization（保存LLM 相关配置）
- Streaming（利用LLM实现流式输出）

Async API（异步调用语言模型）

LangChain如何通过利用asyncio库来为LLM提供异步支持。

异步支持对于并发调用多个LLM特别有用。

目前，OpenAI、PromptLayerOpenAI、ChatOpenAI和Anthropic都支持异步。



Serialization（保存LLM 相关配置）

```
#加载

llm = load_llm("llm.json")

#保存

llm.save("llm.json")
```

```
# llm.json 文件内容
# {
#     "model_name": "text-davinci-003",
#     "temperature": 0.7,
#     "max_tokens": 256,
#     "top_p": 1.0,
#     "frequency_penalty": 0.0,
#     "presence_penalty": 0.0,
#     "n": 1,
#     "best_of": 1,
#     "request_timeout": null,
#     "_type": "openai"
# }
```

- model_name: 模型的名称。
- temperature: 控制生成文本的随机性。值越高，输出的文本就越随机；值越低，输出的文本就越确定。（感情）
- max_tokens: 生成文本的最大长度。
- frequency_penalty：控制生成的文本中常见词的频率。值越高，生成的文本中常见词的频率就越低。
- presence_penalty: 控制生成的文本中新词的频率。值越高，生成的文本中新词的频率就越高。
- n: 控制生成的文本的数量。
- best_of: 从多少个生成的文本中选择最好的一个。
- request_timeout: 控制请求的超时时间。如果设置为null，那么将使用默认的超时时间。
- _type: 指定了LLM的类型。

Streaming（利用LLM实现流式输出）

流式（Streaming）响应的LLM。流式响应意味着可以在响应可用时立即开始处理，而不需要等待整个响应返回。

目前，支持 OpenAI、ChatOpenAI 和 ChatAnthropic 实现的流式响应。

要使用流式响应，需要使用一个实现了 on_llm_new_token 的 CallbackHandler。

12.流式响应

```
# 导入所需的库
from langchain.llms import OpenAI
from langchain.callbacks.streaming_stdout import StreamingStdOutCallbackHandler

# 创建一个使用流式响应和回调处理器的 OpenAI LLM 实例
llm = OpenAI(streaming=True, callbacks=[StreamingStdOutCallbackHandler()], temperature=(

# 使用 LLM 生成一首关于气泡水的歌曲
resp = llm("给我写一首关于冰淇淋的歌。")
```

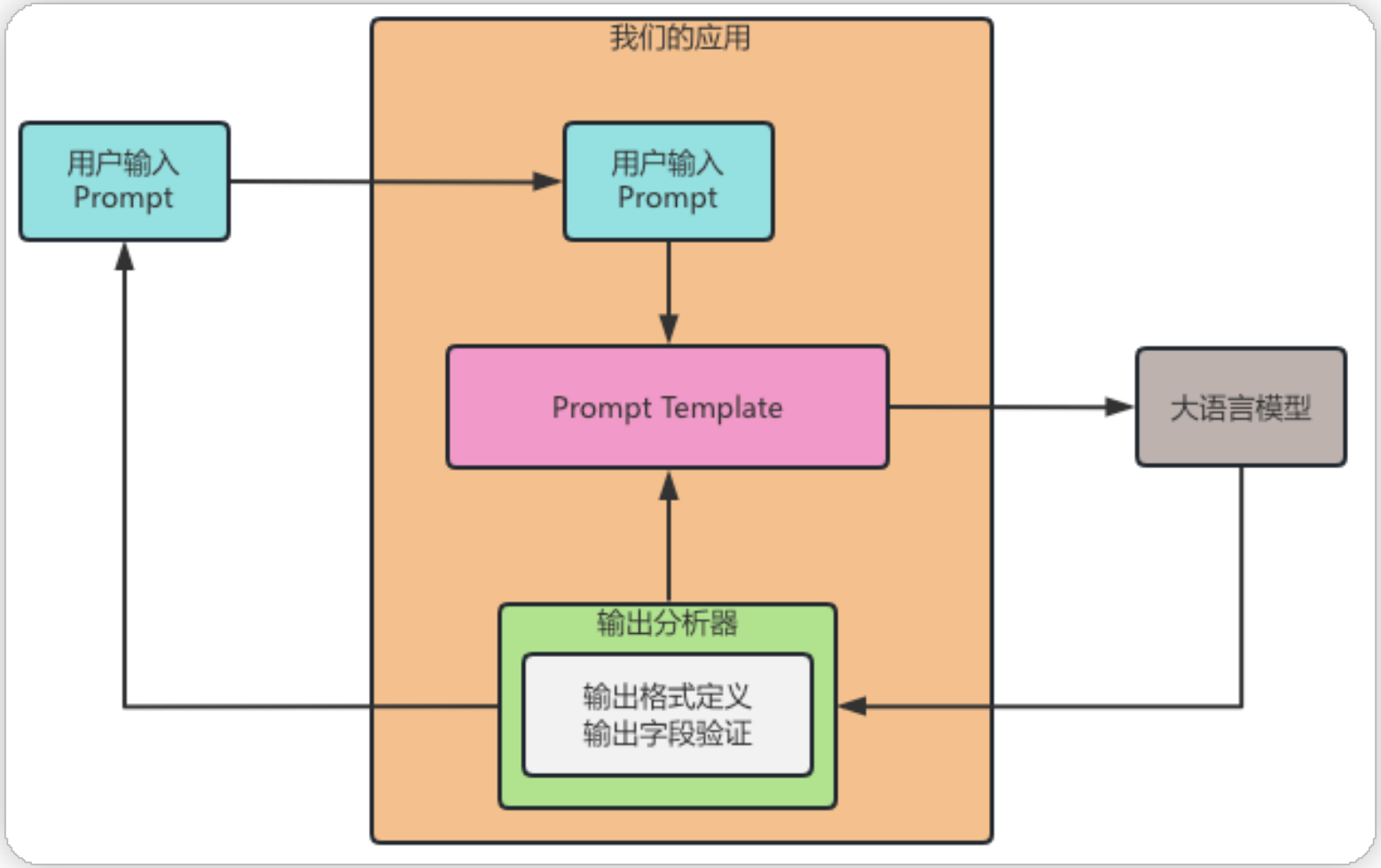
3-6 Output parsers（输出解析器）

- Output parsers（输出解析器）
- List parser（列表解析）
- Datetime parser（日期解析）
- Auto-fixing parser （自动修复解析器）

Output parsers（输出解析器）

在输入的时候通过Prompt Template

- 定义好输出分析器
  - 输出格式定义：字段，类型
  - 输出字段验证



List parser（列表解析）

通过get format instructions 指令在prompt template 阶段定义输出的格式为list，并且定义输出分析器为CommaSeparatedListOutputParser

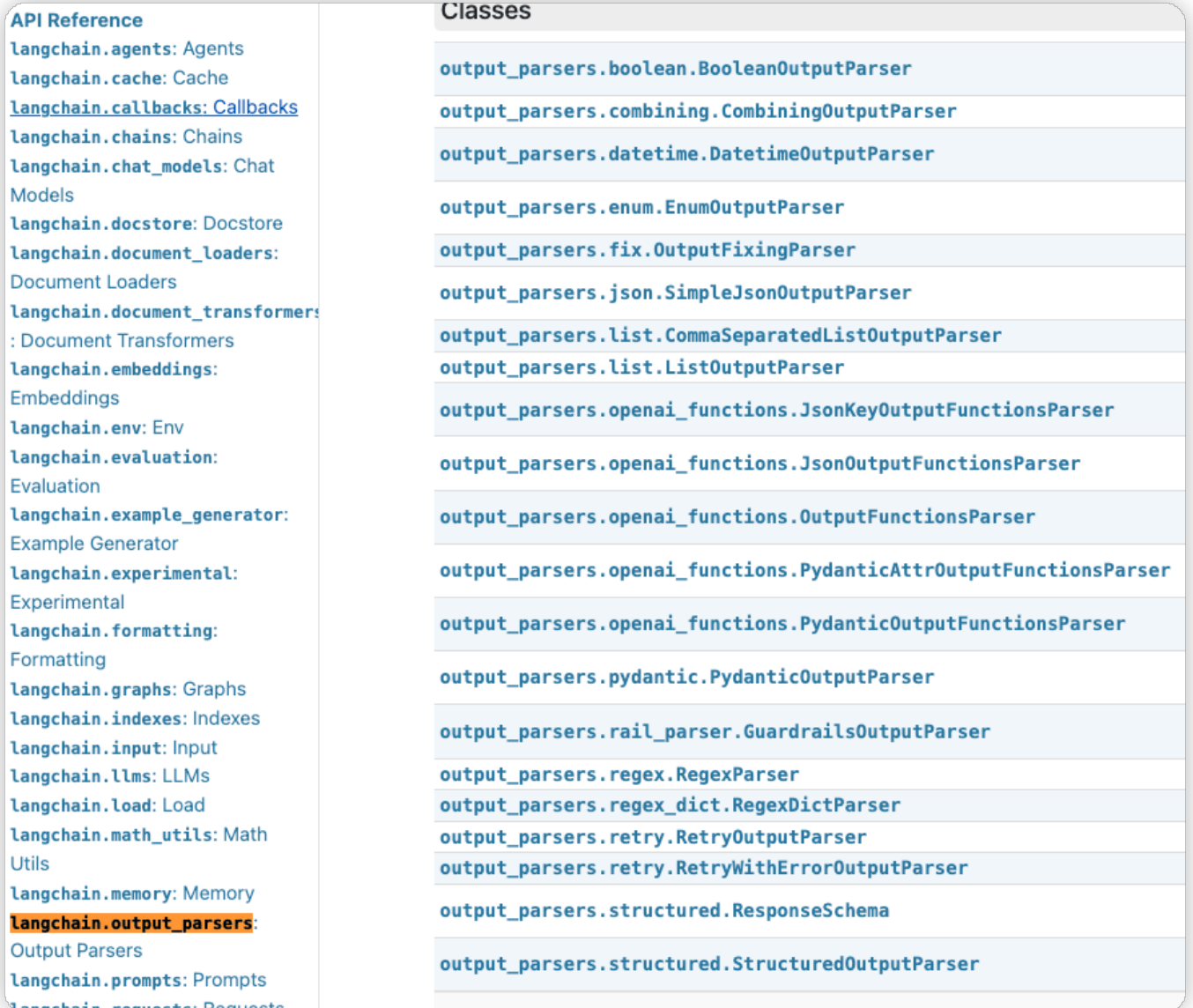
```python
# 创建一个逗号分隔的列表输出解析器
output_parser = CommaSeparatedListOutputParser()

# 获取格式化指示
format_instructions = output_parser.get_format_instructions()

# 创建一个提示模板
prompt = PromptTemplate(
    template="列出五种{subject}。\n{format_instructions}",
    input_variables=["subject"],
    partial_variables={"format_instructions": format_instructions}
)

# 创建一个OpenAI模型实例
model = OpenAI(temperature=0)

# 使用提示模板和主题输入变量来格式化输入
_input = prompt.format(subject="冰淇淋口味")

# 使用模型实例来生成输出
output = model(_input)

# 使用输出解析器来解析输出
output_parser.parse(output)
```

```
['草莓', '抹茶', '巧克力', '香草', '杏仁']
```

Datetime parser（日期解析）

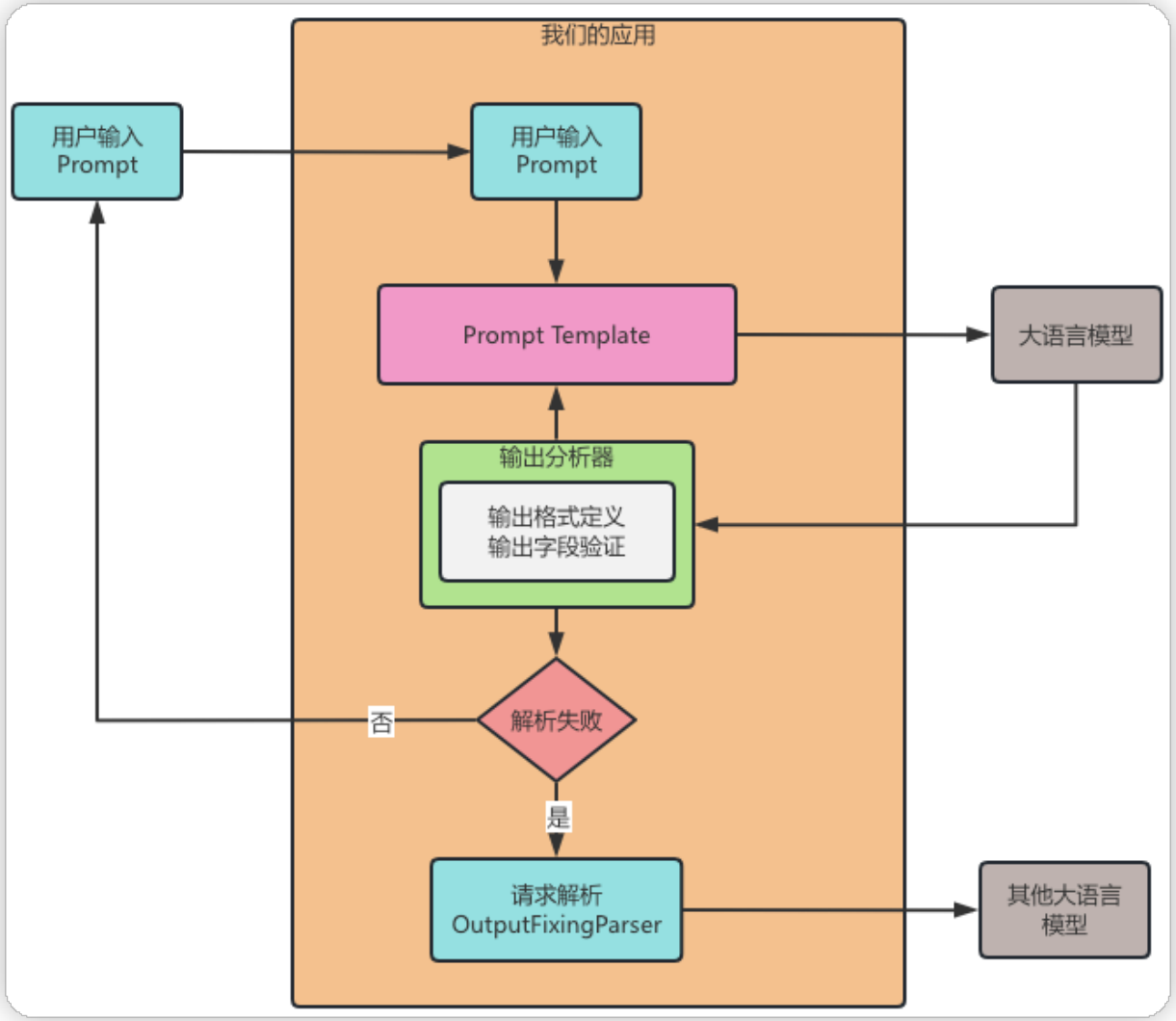
这次用的DatetimeOutputParser 日期的解析器。其实Langchain针对不同的场景建立了很多Parser

https://api.python.langchain.com/en/latest/api_reference.html#module-langchain.output_parsers



Auto-fixing parser （自动修复解析器）

OutputFixingParser是特殊的输出解析器，它的工作方式是：如果原始的解析器无法解析输入的字符串，那么会使用一个语言模型 “修复”输入的字符串，使其能够被原始的解析器解析。



本章总结

- Prompt Template（提示模版）
  - Prompts 、Prompt Template 基本介绍
  - ChatMessagePromptTemplate 精准回应 Role
  - MessagesPlaceholder 实现多类型Message合作
  - Partial Prompt Template（模版的部分加载）
  - Composition（Prompt template 组合）
  - Serialization（序列化保存Prompt template）
- Example selectors（示例选择器）
  - Select by length（根据Prompt 控制示例长度）
  - Select by similarity（选择与Prompt 类似的示例）
- Language Models（语言模型的应用）
  - Async API（异步调用语言模型）
  - FakeListLLM（模拟LLM）
  - Caching（缓存LLM请求）
  - Serialization（保存LLM 相关配置）
  - Streaming（利用LLM实现流式输出）
- Output parsers（输出解析器）
  - List parser（列表解析）
  - Datetime parser（日期解析）
  - Auto-fixing parser （自动修复解析器）