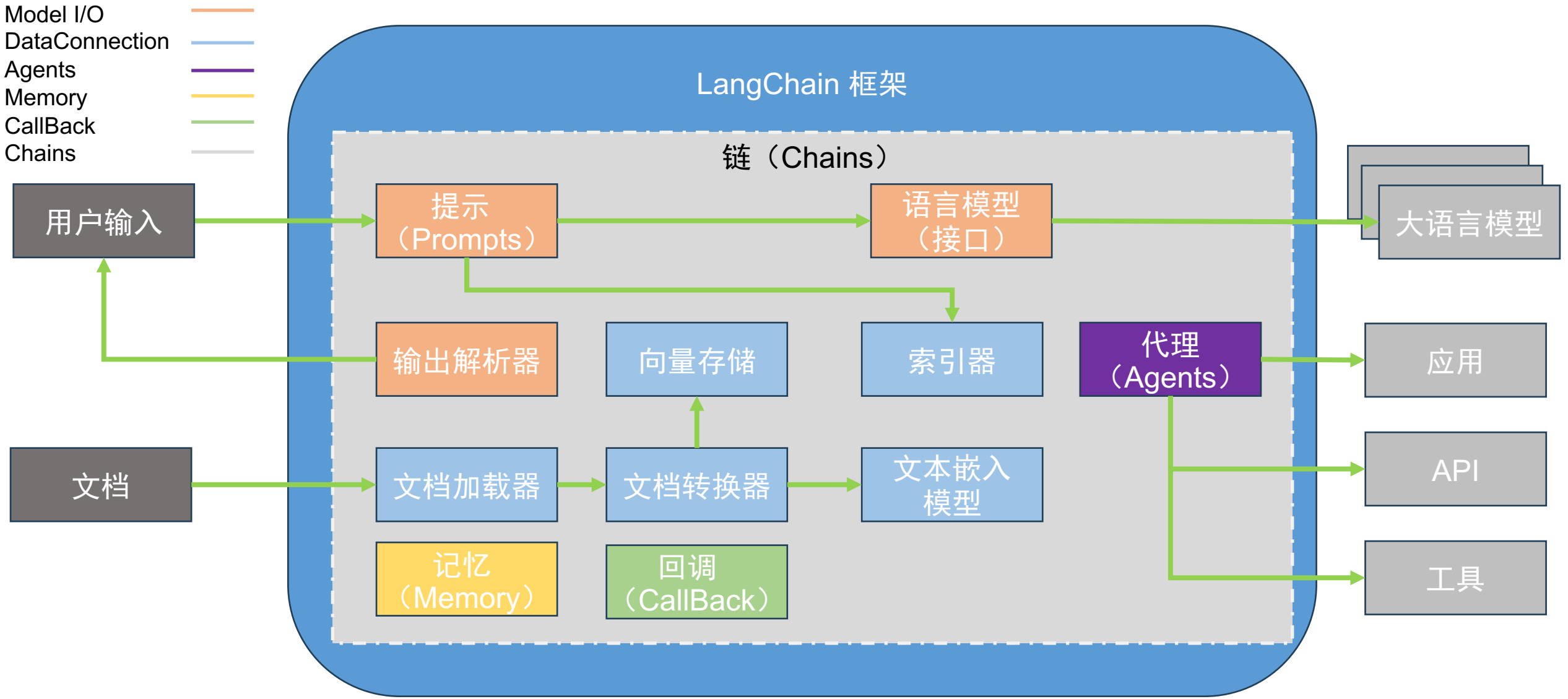
5-1 Chains 组件概述

- 回顾组件地图
- Chains 组件基本概念
- 本章课程介绍

LangChain 6 大组件回顾



Chain 组件概述

是什么：Chain是一种接口，它将多个组件（如语言模型，提示模板等）串联在一起，以形成更复杂的应用或功能。它可以理解为将一个或多个任务或操作链接在一起的编程模式。

为什么：使用Chain的主要原因如下：

1.模块化: 将复杂的应用拆解成多个更简单、更易于管理和维护的模块。每个模块都可以独立进行开发和测试，这使得开发过程更加灵活和高效。

2.可重用性: 创建可重用的组件和功能，这意味着开发人员可以避免重复编写代码，从而提高效率。

3.可扩展性: 使用Chain，可以轻松地添加、修改或删除链中的组件，以满足应用的新需求或改变现有功能。这使得应用更具有适应性和灵活性。

本章课程介绍

- Chain基本用法
  - 异步API调用
  - 如何Debug
  - 调用外部资源
  - 与Memory合作
  - 持久化
- Chain的分类
  - LLMChain
  - RouterChain
  - SequentialChain
  - TransformationChain
- 常用套路
  - API/SQL Chains
  - Retrieval QA
  - 摘要

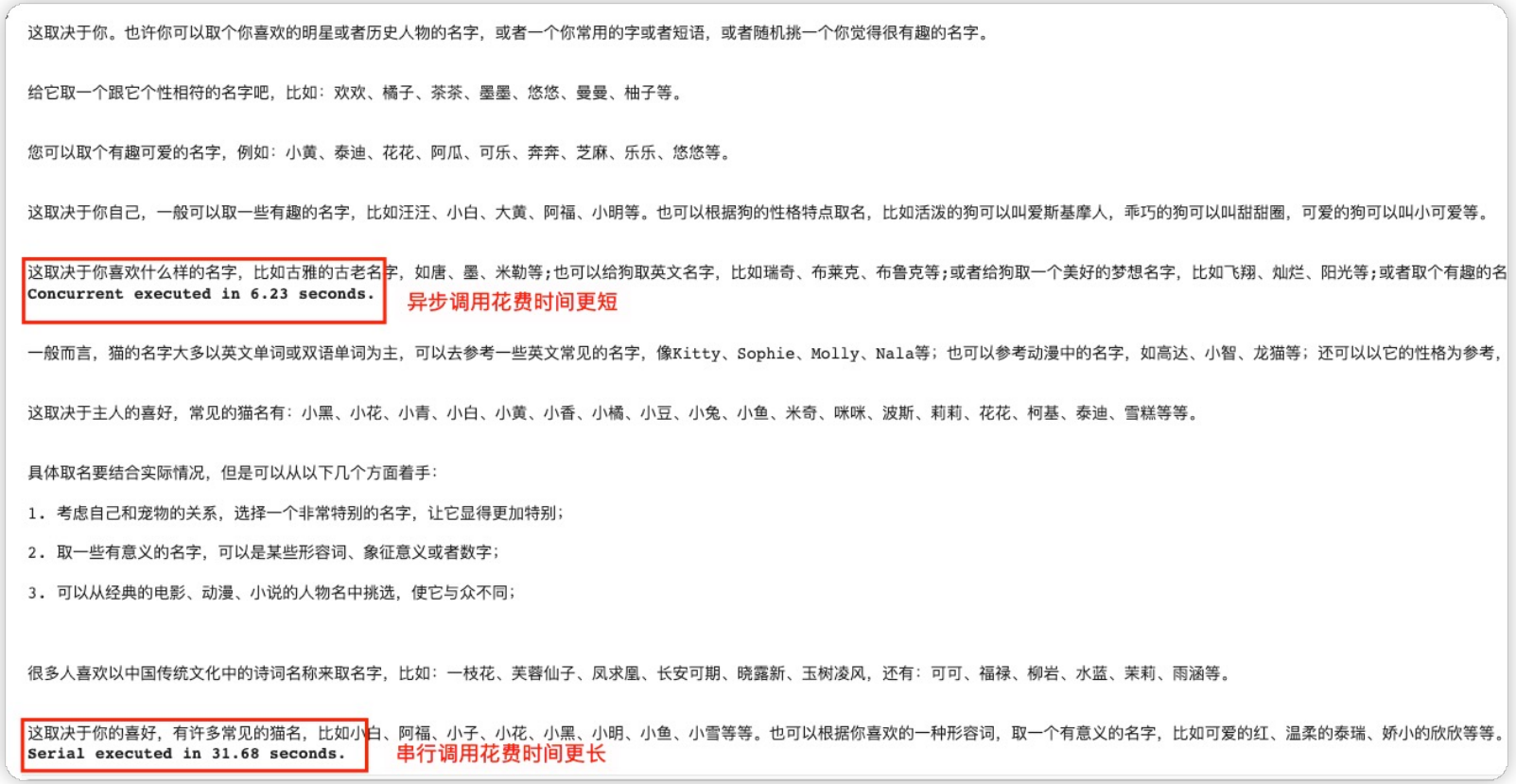
5-2 Chain 基本用法

- API 异步调用
- 如何Debug
- 调用外部资源
- 与Memory合作
- 持久化

API的异步调用

异步调用：

1. 生成LLMChain，其中链接LLM和Prompt
2. Async_generate方法将对应的Chain转化成task
3. Async中的gather方法执行异步任务



打开Chain的Debug 开关

通过设置LLMChain 对象中的verbose属性为True

```python
from langchain.llms import OpenAI
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain

llm = OpenAI(temperature=0.5)
prompt = PromptTemplate(
    input_variables=["pet"],
    template="我养了一只{pet}起个什么名字好呢?",
)
#verbose 为tur 开启 debug开关，可以跟踪 chain的完成情况
chain = LLMChain(llm=llm, prompt=prompt,verbose=True)
# Run the chain only specifying the input variable.
print(chain.run("小狗"))
```

打印Chain的执行情况

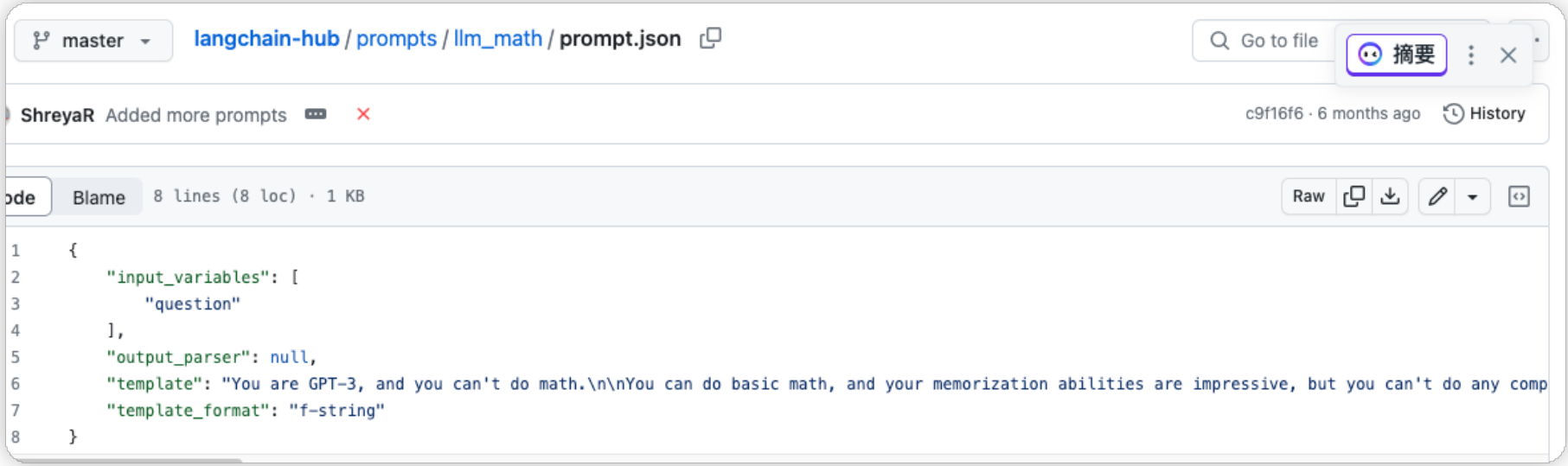
```
> Entering new LLMChain chain...
Prompt after formatting:
我养了一只小狗起个什么名字好呢?

> Finished chain.
```

链接外部资源-LangChainHub

”Prompt template": "你是GPT-3，你不能做数学运算。
你可以做基础数学运算，你的记忆能力令人印象深刻，但你无法做任何一个人不能在脑子里做的复杂计算。你也有一种恼人的倾向，那就是仅仅是编造出高度具体但错误的答案。所以我们将你连接到Python 3内核，现在你可以执行代码。如果有人给你一个困难的数学问题，只需使用这种格式，我们将处理剩下的事情:

问题: ${{涉及到复杂计算的问题}}
python${{打印你需要知道的代码}}
output\n${{你的代码的输出}}
答案: ${{答案}}
否则，使用这种更简单的格式:
问题: ${{不涉及复杂计算的问题}}
答案: ${{答案}}开始。
问题: 37593 * 67 是多少?
Python
print(37593 * 67)
Output 2518731
答案: 2518731

LangChainHub是一个收集与LangChain基础元素（如提示，链和代理）相关工具组件的地方。



与Memory合作、保存Chain 到磁盘

- ConversationChain = ChatOpenAI + Conversation BufferMemory
- 记住上次对话的信息
- 保存Chain文件到磁盘

```python
from langchain.llms import OpenAI
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain

llm = OpenAI(temperature=0.5)
prompt = PromptTemplate(
    input_variables=["pet"],
    template="我养了一只{pet}起个什么名字好呢?",
)
#将llm 和 prompt链接起来
chain = LLMChain(llm=llm, prompt=prompt)
# Run the chain only specifying the input variable.
print(chain.run("小狗"))

chain.save("llm_chain.json")
```

```python
from langchain.chains import ConversationChain
from langchain.memory import ConversationBufferMemory
from langchain.chat_models import ChatOpenAI

chat = ChatOpenAI(temperature=0.5)
conversation = ConversationChain(
    llm=chat,
    memory=ConversationBufferMemory()
)
#chat中的第一个问题
print(conversation.run("世界由哪几个大洲组成的，说前三个?"))
#chat中的第二个问题，通过memory记忆上下文
print(conversation.run("把剩余的洲告诉我?"))
```

```
世界由七个大洲组成，按照面积从大到小排列，前三个大洲分别是亚洲、非洲和北美洲。
剩余的大洲是南美洲、南极洲、欧洲和大洋洲。
```

5-3 Chain 分类 – 上

- LLMChain
  - prompt template 的参数数组化
  - 多参数
  - 定义输出格式
- RouterChain

LLMChain

参数化数组与多参数输入：

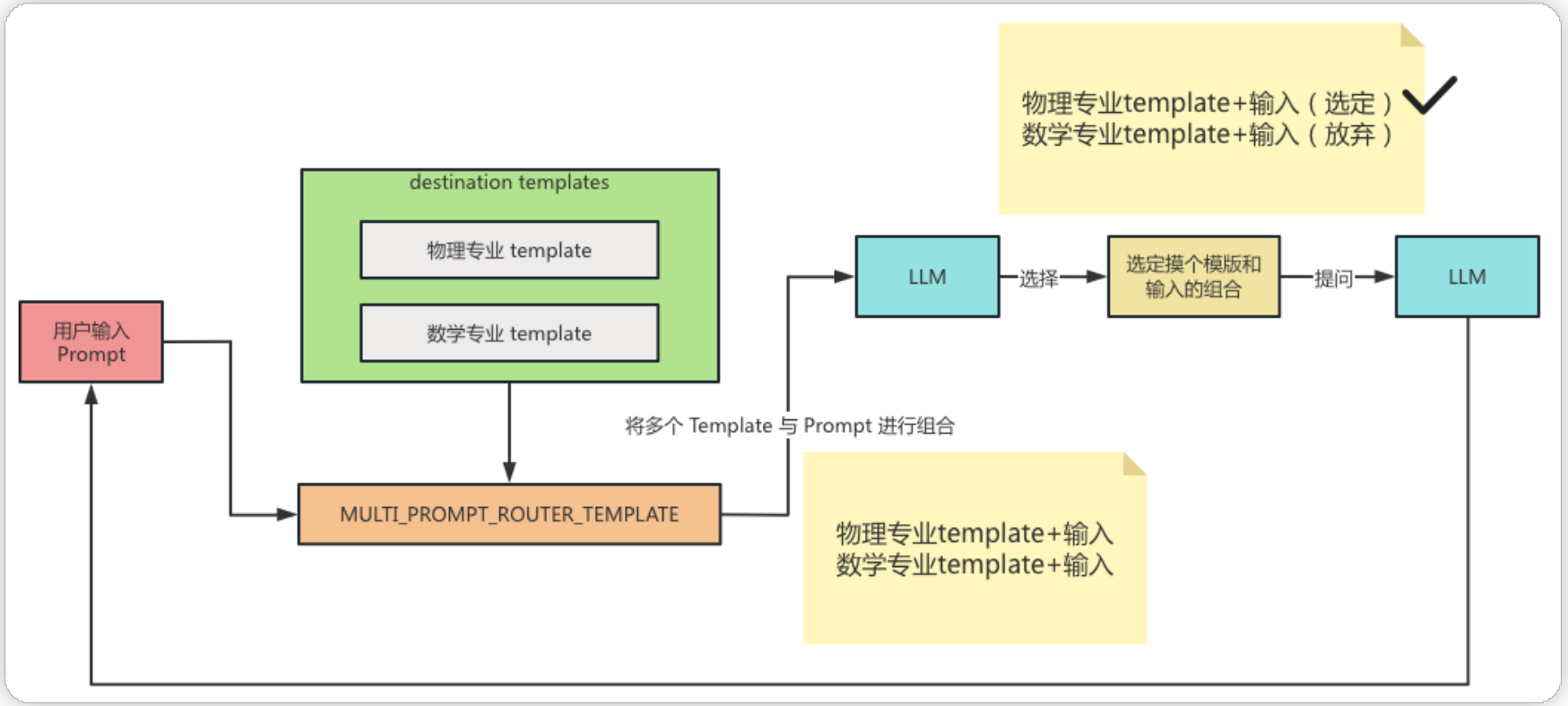
- `apply`方法: 这个方法以数组的形式输入运行链。例如，输入列表可能包含了多个不同的产品名称，`apply`方法可以一次性处理这些输入，并返回每个输入对应的结果。
- `predict`方法: 支持关键字参数的输入。可以直接将变量作为参数传递给`predict`方法，而不需要创建一个字典。

解析输出：
默认情况下，`LLMChain`不会解析输出，即使底层的提示对象有一个输出解析器。如果想在LLM的输出上应用输出解析器，可以使用`predict_and_parse`替代`predict`，使用`apply_and_parse`替代`apply`。

RouterChain

RouterChain动态选择链条的方法。它包括：RouterChain负责调用的链条和目标链条，RouterChain可选择的链条。

工作流程如下：
1. RouterChain收到一个输入。
2. 使用LLMRouterChain将输入和所有目标链模板传递给LLM。
3. LLM输出最合适的目标链条的模板。
4. RouterChain然后将原始输入+选择的目标链条模版，传递LLM，LLM产生最终输出。



5-4 Chain 分类 – 下

- SequentialChain
  - 简单顺序执行
  - 定义输出，输入参数执行
  - 与Memory结合执行
- TransformationChain

SequentialChain

•**SimpleSequentialChain**：这是最简单的顺序链条形式，其中每个步骤都有单一的输入/输出，且一个步骤的输出是下一个步骤的输入。

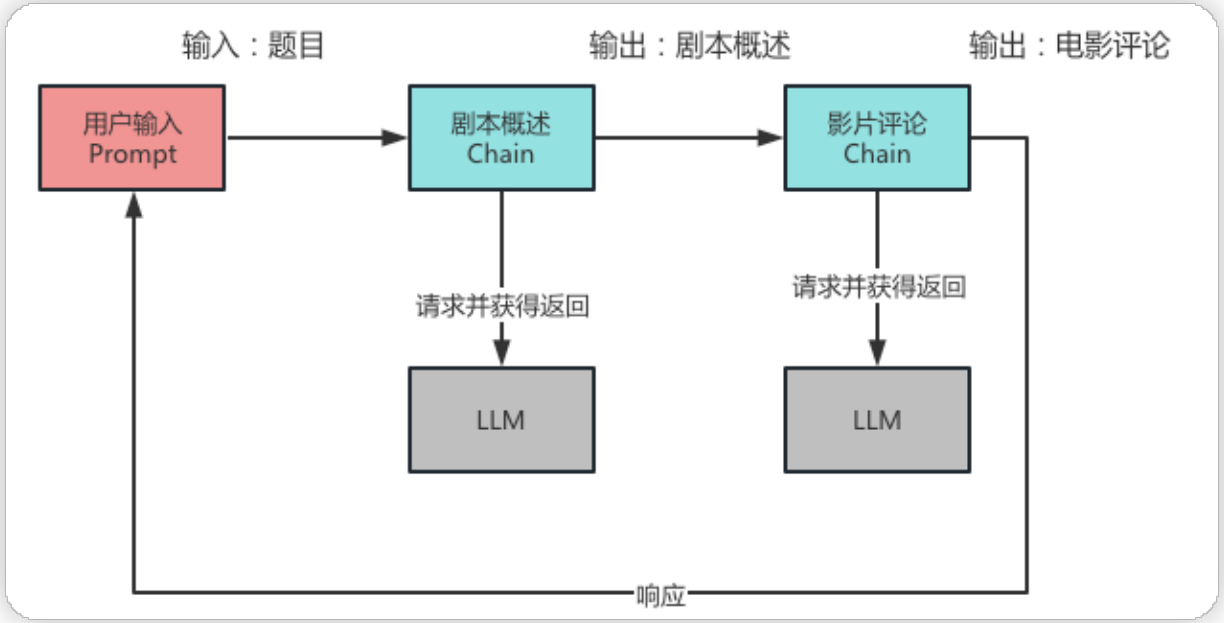
•Chain 与Chain 之间的 输出 与 输入 是默认的。

•支持单个参数

•**SequentialChain**：这是更通用的顺序链条形式，允许多个输入/输出。

•Chain 与chain 之间的输出与输入是指定的

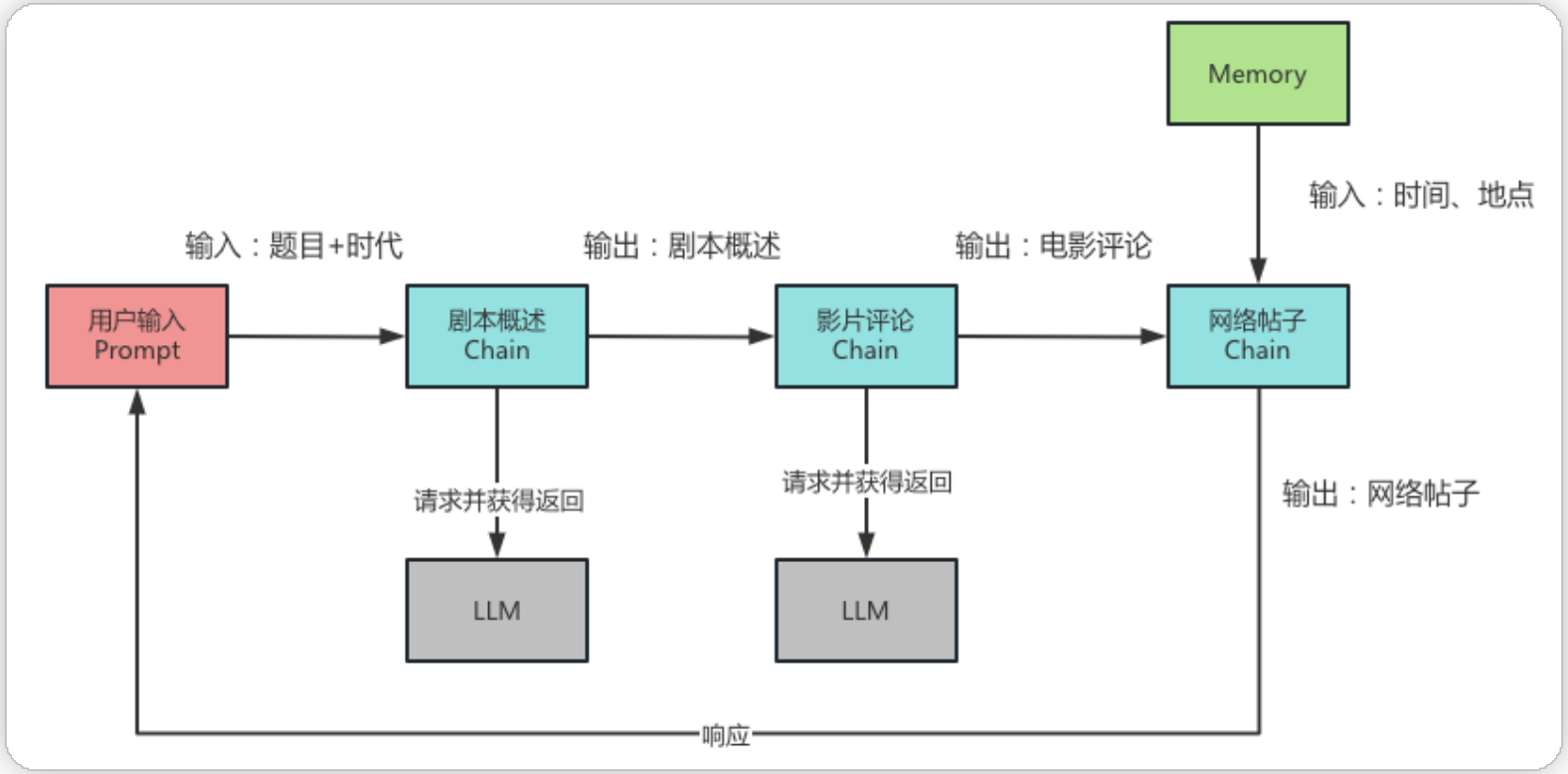
•支持多参数



SequentialChain

•与Memory协作

•在链条的每个步骤中或链条的后续部分中使用一些上下文信息，但是管理和链接输入/输出变量可能会很混乱。在这种情况下，我们可以使用SimpleMemory来方便地管理这些上下文信息。



TransformationChain

•转换链执行过程：

1. 输入文本（"text"）被读入和处理。
2. 使用一个转换函数（"transform_func"），把输入文本中的前三段转换为摘要（"output_text"）。
3. 该摘要被填充到一个模板中，生成一个新的提示（"prompt"），用于提供给语言生成模型。
4. 这个提示被提供给OpenAI模型，用于生成新的文本。
5. 所有这些步骤被组织在一个"SimpleSequentialChain"中，这个链按照顺序执行以上所有步骤。

5-5 SQL Chain –上

- 创建演示数据库
- 连接Sqlite 数据库查询数据
- 根据条件进行连接查询
- 通过Prompt 进行table的指定

创建演示数据库

Langchain支持如下sql

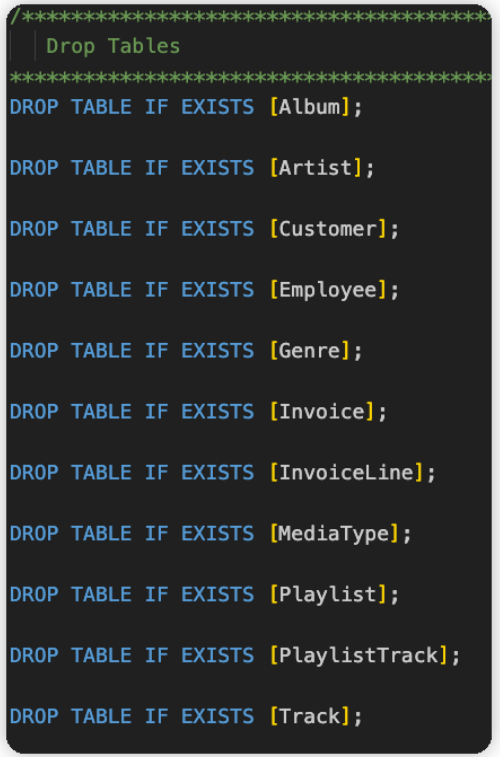
MS SQL, MySQL, MariaDB, PostgreSQL, Oracle SQL, Databricks and SQLite

MySQL connection URL: mysql+pymysql://user:pass@some_mysql_db_address/db_name.

为了教学方便，我们使用SQLite作为演示数据库。

需要安装sqlite3 的命令如下：

```
#安装sqlite3
!apt-get install -y sqlite3
#创建数据库
!sqlite3 Chinook.db
#通过命令加载数据库脚本
.read Chinook_Sqlite.sql
```



连接Sqlite 数据库查询数据

```python
from langchain import OpenAI, SQLDatabase, SQLDatabaseChain

db = SQLDatabase.from_uri("sqlite:///Chinook.db")
llm = OpenAI(temperature=0, verbose=True)
db_chain = SQLDatabaseChain.from_llm(llm, db, verbose=True)

db_chain.run("有多少个员工?")
```

将自然语言转化为 SQL

```
> Entering new SQLDatabaseChain chain...
有多少个员工?
SQLQuery:SELECT COUNT(*) FROM "Employee";
SQLResult: [(8,)]
Answer:8个员工
> Finished chain.
'8个员工'
```

根据条件进行表连接查询

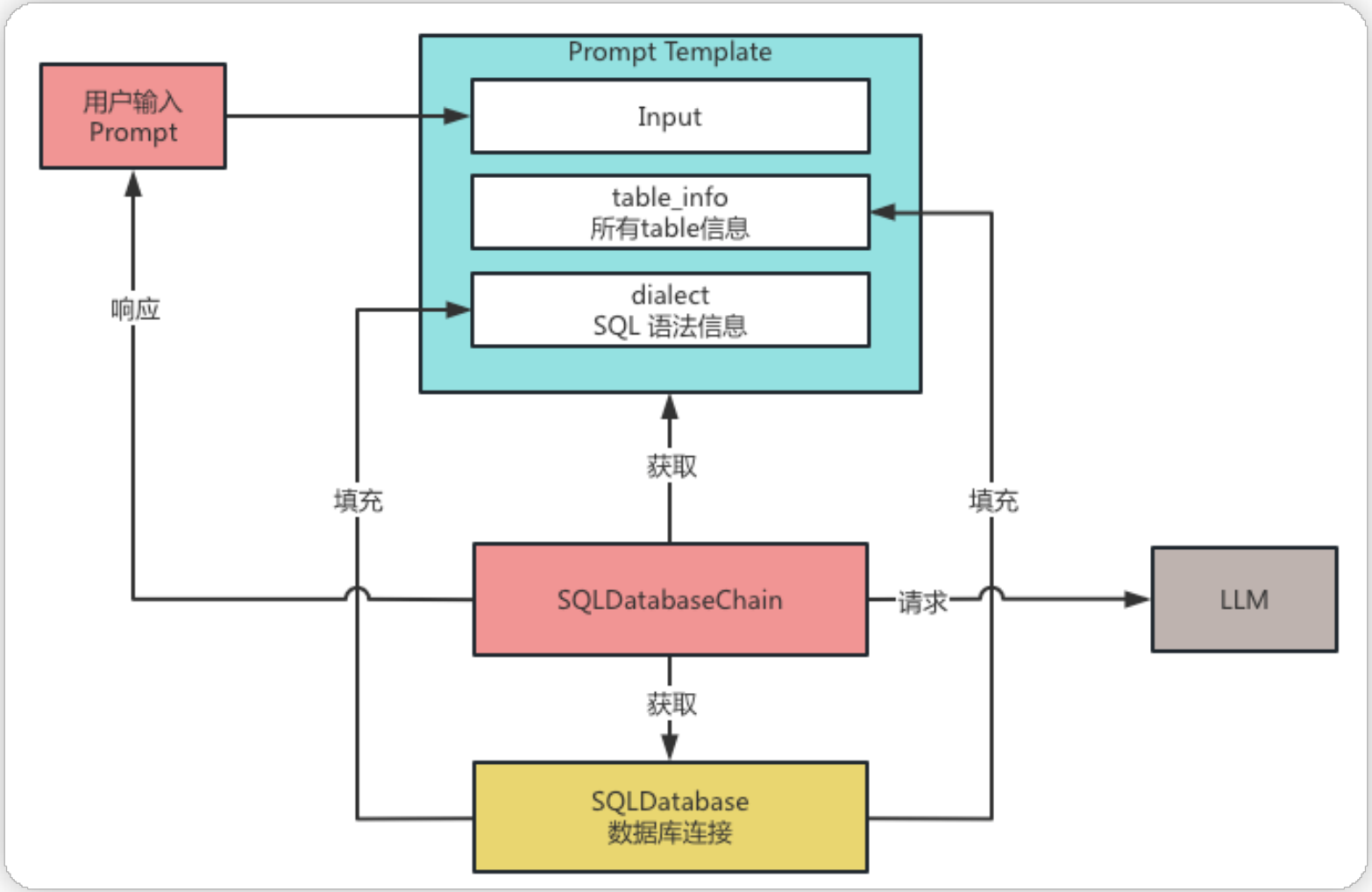
7.3根据条件进行表连接查询

```
db_chain = SQLDatabaseChain.from_llm(llm, db, verbose=True, use_query_checker=True)
db_chain.run("有多少个Aerosmith的专辑?")
```

```
> Entering new SQLDatabaseChain chain...
有多少个Aerosmith的专辑?
SQLQuery:SELECT COUNT(*) FROM Album WHERE ArtistId = (SELECT ArtistId FROM Artist WHERE Name = 'Aerosmith');
SQLResult: [(1,)]
Answer:Aerosmith有1张专辑。
> Finished chain.
'Aerosmith有1张专辑。'
```

通过prompt 进行table的指定

{table_info}和{dialect}是预定的占位符，它们在SQLDatabaseChain类中有特定的用途。 由SQLDatabaseChain.from_llm(llm, db, prompt=PROMPT, verbose=True)这段代码在执行过程中自动填充的。 {table_info}代表了数据库中所有表格的信息。这些信息一般包括表名、表的字段、字段的类型等等。 {dialect}代表的是你使用的数据库的SQL方言。SQL语言有很多种方言，比如MySQL、PostgreSQL、SQLite等等



5-6 SQL Chain –下

显示查询的中间过程（表结构、SQL）

限制返回的行数

在表中加入例子记录

定制化表（关注的表和字段）+示例

SQL Sequential Chain

中间过程与返回行数

显示查询的中间过程（表结构、SQL）

```
db_chain = SQLDatabaseChain.from_llm(llm, db, prompt=PROMPT, verbose=True, use_query_c

result = db_chain("在foobar表中有多少个员工?")
result["intermediate_steps"]
```

```
> Entering new SQLDatabaseChain chain...
在foobar表中有多少个员工?
SQLQuery:SELECT COUNT(*) FROM Employee;
SQLResult: [(8,)]
Answer:8
> Finished chain.
[{'input': '在foobar表中有多少个员工?\nSQLQuery:SELECT COUNT(*) FROM Employee;\nSQLResult: [(8,)]\nAnswer:',
  'top_k': '5',
  'dialect': 'sqlite',
  'table_info': '\nCREATE TABLE "Album" (\n\t"AlbumId" INTEGER NOT NULL, \n\t"Title" NVARCHAR(160) NOT NULL,
\n\t"ArtistId" INTEGER NOT NULL, \n\tPRIMARY KEY ("AlbumId"), \n\tFOREIGN KEY("ArtistId") REFERENCES "Artist"
("ArtistId")\n)\n\n/*\n3 rows from Album table:\nAlbumId\tTitle\tArtistId\n1\tFor Those About To Rock We
Salute You\t1\n2\tBalls to the Wall\t2\n3\tRestless and Wild\t2\n*/\n\n\nCREATE TABLE "Artist" (\n\t"ArtistId"
INTEGER NOT NULL, \n\t"Name" NVARCHAR(120), \n\tPRIMARY KEY ("ArtistId")\n)\n\n/*\n3 rows from Artist
table:\nArtistId\tName\n1\tAC/DC\n2\tAccept\n3\tAerosmith\n*/\n\n\nCREATE TABLE "Customer" (\n\t"CustomerId"
INTEGER NOT NULL, \n\t"FirstName" NVARCHAR(40) NOT NULL, \n\t"LastName" NVARCHAR(20) NOT NULL, \n\t"Company"
NVARCHAR(80), \n\t"Address" NVARCHAR(70), \n\t"City" NVARCHAR(40), \n\t"State" NVARCHAR(40), \n\t"Country"
NVARCHAR(40), \n\t"PostalCode" NVARCHAR(10), \n\t"Phone" NVARCHAR(24), \n\t"Fax" NVARCHAR(24), \n\t"Email"
NVARCHAR(60) NOT NULL, \n\t"SupportRepId" INTEGER, \n\tPRIMARY KEY ("CustomerId"), \n\tFOREIGN
KEY("SupportRepId") REFERENCES "Employee" ("EmployeeId")\n)\n\n/*\n3 rows from Customer
table:\nCustomerId\tFirstName\tLastName\tCompany\tAddress\tCity\tState\tCountry\tPostalCode\tPhone\tFax\tEmail
- Empresa Brasileira de Aeronáutica S.A.\tAv. Brigadeiro Faria Lima, 2170\tSão José dos
Campos\tSP\tBrazil\t12227-000\t+55 (12) 3923-5555\t+55 (12) 3923-
5566\tluisg@embraer.com.br\t3\n2\tLeonie\tKöhler\tNone\tTheodor-Heuss-Straße
34\tStuttgart\tNone\tGermany\t70174\t+49 0711
2842222\tNone\tleonekohler@surfeu.de\t5\n3\tFrançois\tTremblay\tNone\t1498 rue
Bélanger\tMontréal\tQC\tCanada\tH2G 1A7\t+1 (514) 721-4711\tNone\tftremblay@gmail.com\t3\n*/\n\n\nCREATE TABLE
"Employee" (\n\t"EmployeeId" INTEGER NOT NULL, \n\t"LastName" NVARCHAR(20) NOT NULL, \n\t"FirstName"
NVARCHAR(20) NOT NULL, \n\t"Title" NVARCHAR(30), \n\t"ReportsTo" INTEGER, \n\t"BirthDate" DATETIME,
```

限制返回的行数

```
db_chain = SQLDatabaseChain.from_llm(llm, db, verbose=True, use_query_checker=True, top_k=3)

db_chain.run("作曲家约翰·塞巴斯蒂安·巴赫的一些示例曲目是什么？ ")

db_chain.run("What are some example tracks by composer Johann Sebastian Bach? ")
```

在表中加入例子记录

"添加每个表的示例行"的含义是将数据表中的一些实际数据（即一些行）包含在提示中。这样做的目的是使语言模型（LLM）在生成最终查询之前可以理解数据的格式。
这段文字中的示例是将两行从“Track”表中的数据添加到提示中，以让LLM知道艺术家是用他们的全名保存的。这种方式可以帮助模型更准确地理解并处理数据。

```
CREATE TABLE "Track" (
        "TrackId" INTEGER NOT NULL,
        "Name" NVARCHAR(200) NOT NULL,
        "AlbumId" INTEGER,
        "MediaTypeId" INTEGER NOT NULL,
        "GenreId" INTEGER,
        "Composer" NVARCHAR(220),
        "Milliseconds" INTEGER NOT NULL,
        "Bytes" INTEGER,
        "UnitPrice" NUMERIC(10, 2) NOT NULL,
        PRIMARY KEY ("TrackId"),
        FOREIGN KEY("MediaTypeId") REFERENCES "MediaType" ("MediaTypeId"),
        FOREIGN KEY("GenreId") REFERENCES "Genre" ("GenreId"),
        FOREIGN KEY("AlbumId") REFERENCES "Album" ("AlbumId")
)

/*
2 rows from Track table:
TrackId Name    AlbumId MediaTypeId     GenreId Composer        Milliseconds    Bytes   UnitPrice
1       For Those About To Rock (We Salute You) 1       1       1       Angus Young, Malcolm Young, Brian Johnson       343719  11170334
2       Balls to the Wall       2       2       1       None    342562  5510424 0.99
*/
```

定制化表

这个CREATE TABLE语句只是告诉模型"Track"表的结构，包括它的列名、列的数据类型以及主键。这对于模型来说是非常重要的信息，它需要这些信息来生成正确的SQL查询并理解查询结果。

实际上，这个CREATE TABLE语句并不会影响真实的"Track"表，它不会对该表做出任何更改。数据库的实际结构和内容都不会被这个CREATE TABLE语句影响。

```
custom_table_info = {
    "Track": """CREATE TABLE Track (
    "TrackId" INTEGER NOT NULL,
    "Name" NVARCHAR(200) NOT NULL,
    "Composer" NVARCHAR(220),
    PRIMARY KEY ("TrackId")
)
/*
3 rows from Track table:
TrackId Name    Composer
1   For Those About To Rock (We Salute You) Angus Young, Malcolm Young, Brian Johnson
2   Balls to the Wall   None
3   My favorite song ever   The coolest composer of all time
*/"""
}
```

SQL Sequential Chain

SQLDatabaseSequentialChain 是一个查询 SQL 数据库的顺序链。
这个链的过程如下：
1.根据查询确定要使用哪些表。
2.根据这些表，调用正常的 SQL 数据库链。
如果数据库有非常多的表，全面地考虑所有可能的表和它们之间的关联将非常耗费计算资源和时间。

"有多少员工也是客户?" 时，链首先确定要使用的表（在这种情况下，为 "Employee" 和 "Customer" 表）。然后，它在这些表上运行正常的 SQL 数据库链，最终返回查询结果。

```
from langchain.chains import SQLDatabaseSequentialChain
db = SQLDatabase.from_uri("sqlite:///Chinook.db")

chain = SQLDatabaseSequentialChain.from_llm(llm, db, verbose=True)

chain.run("How many employees are also customers?")
```

```
> Entering new SQLDatabaseSequentialChain chain...
/usr/local/lib/python3.10/dist-packages/langchain/chains/llm.py:275: UserWarning: The predict_and_parse m
  warnings.warn(
Table names to use:
['Customer', 'Employee']

> Entering new SQLDatabaseChain chain...
How many employees are also customers?
SQLQuery:SELECT COUNT(*) FROM Employee e INNER JOIN Customer c ON e.EmployeeId = c.SupportRepId;
SQLResult: [(59,)]
Answer:There are 59 employees who are also customers.
> Finished chain.

> Finished chain.
'There are 59 employees who are also customers.'
```

确定要查的表

查询并返回结果

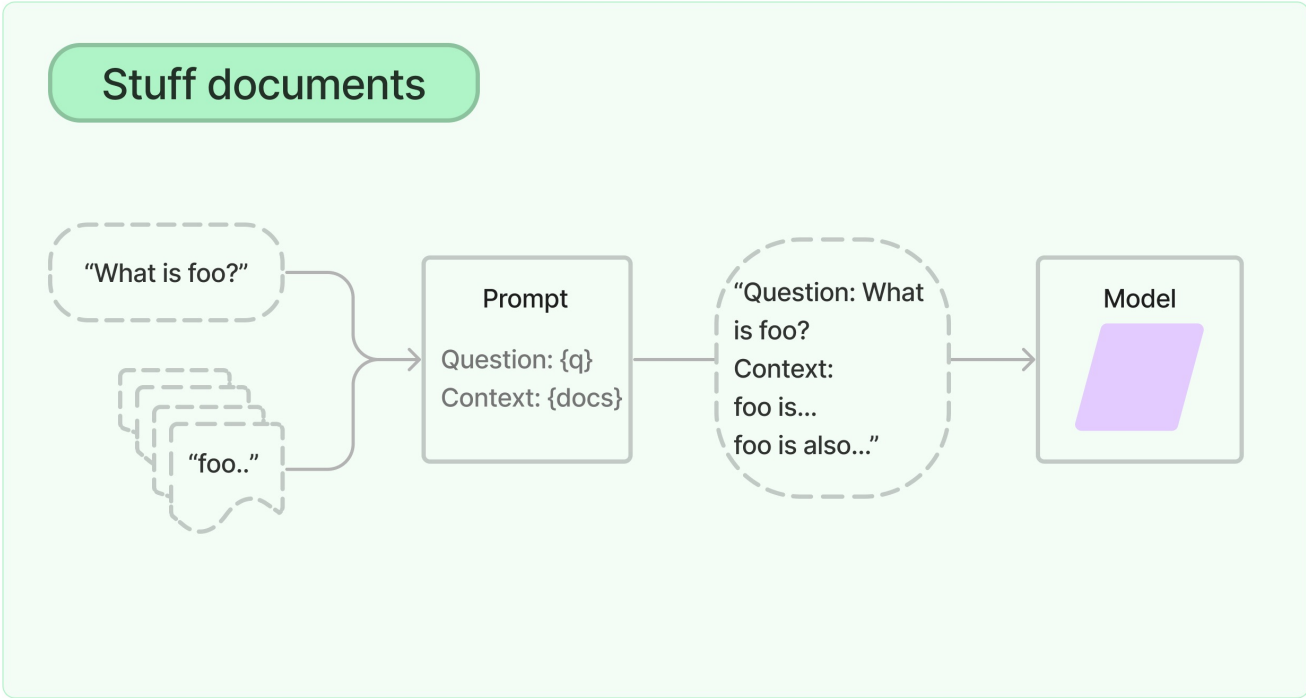
5-7 摘要

- Stuff document
- Map Reduce document

Stuff

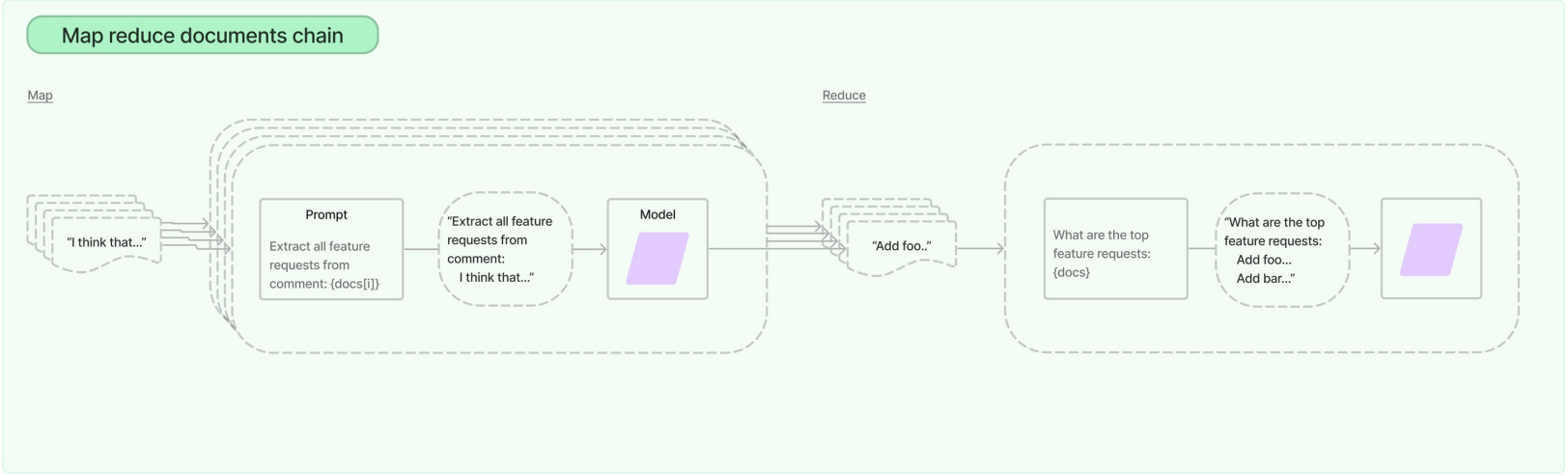
"Stuff Documents Chain"是文档链中最直接的一个。它接收一系列文档，将它们全部插入到一个提示语句中，然后将这个提示语句传递给语言模型（LLM）。

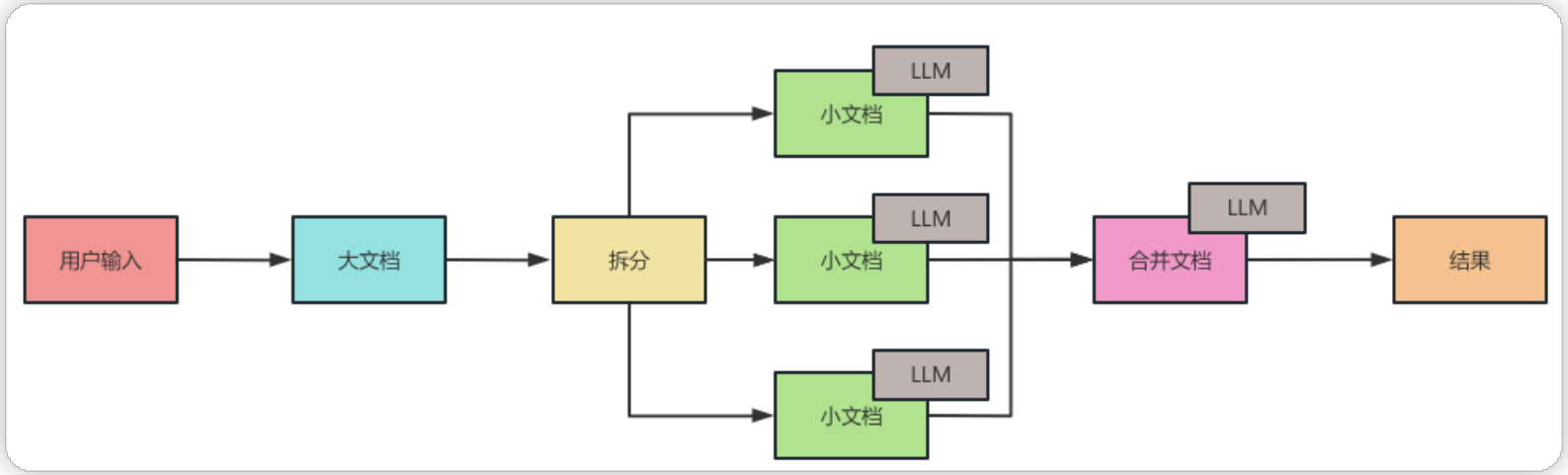
该链适合于处理文档较小，且大多数调用只传入少量文档的应用场景。



Map Reduce

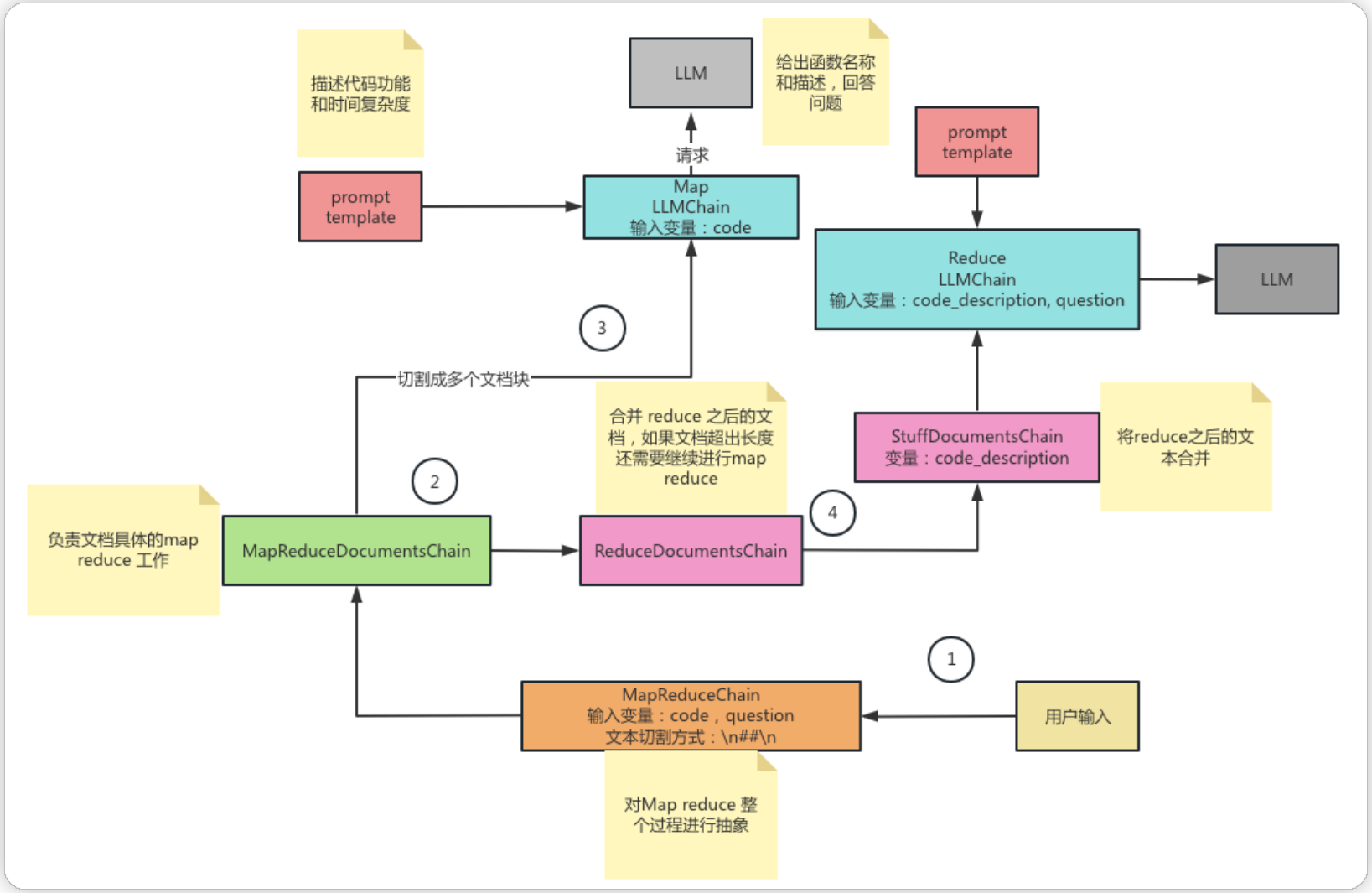
"Map Reduce Documents Chain" 首先将LLM链单独应用到每个文档（Map步骤），并将链输出视为新的文档。然后，它将所有新文档传递给另一个组合文档链，以得到单一的输出（Reduce步骤）。如果需要，它可以首先压缩或合并映射的文档，以确保它们适合组合文档链（这将经常将它们传递给LLM）。如果必要，这个压缩步骤会递归进行。





Map Reduce – 自定义

1. 接受输入，定义文件分割规则
2. 对文本进行 map reduce
3. Map chain请求LLM返回结果
4. Reduce chain 对返回结果进行合并，形成最终文档作为输出



章节回顾

- Chain基本用法
  - 异步API调用
  - 如何Debug
  - 调用外部资源
  - 与Memory合作
  - 持久化
- Chain的分类
  - LLMChain
  - RouterChain
  - SequentialChain
  - TransformationChain
- 常用套路
  - SQL Chains
  - 摘要