🦜Langchain 0.2 RAG 快速入门 #8 self-query

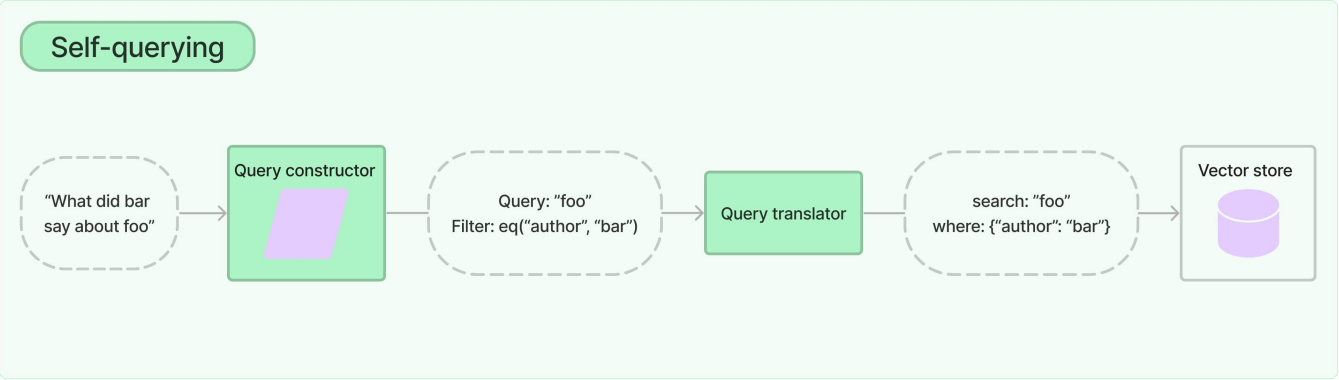
self-query： 用户问题 → **查询结构（新问题，metadata参数）**→执行查询结构

文档要求：特定类型的文档，可以通过不同的metadata 信息分类

比如： 电影概要， 包括的信息（metadata）：上映日期、导演、题材、评分...

自查询检索器正如其名称所示，具有查询自身的能力。

具体来说，给定任何自然语言查询，该检索器使用查询构建的大型语言模型链来编写一个结构化查询，然后将该结构化查询应用于其底层的向量存储。这使得检索器不仅能够使用用户输入的查询与存储文档内容的语义相似性进行比较，而且能够从用户查询中提取存储文档元数据的过滤器并执行这些过滤器。




```
Your goal is to structure the user's query to match the request schema provided below.

<< Structured Request Schema >>
When responding use a markdown code snippet with a JSON object formatted in the following schema:

```json
{
    "query": string \ text string to compare to document contents
```
```

```
    "filter": string \ logical condition statement for filtering documents
}

#############
{
    “query”: string # 用于与文档内容进行比较的文本字符串
    “filter”: string # 用于过滤文档的逻辑条件语句
}

```

查询字符串应只包含预期与文档内容匹配的文本。**过滤器中的任何条件也不应在查询中提及**。

逻辑条件语句由一个或多个比较语句和逻辑运算语句组成。

关键提示词

```
Your goal is to structure the user's query to match the request schema provided below.

<< Structured Request Schema >>
When responding use a markdown code snippet with a JSON object formatted in the following schema:

```json
{
    "query": string \ text string to compare to document contents
    "filter": string \ logical condition statement for filtering documents
}
```

The query string should contain only text that is expected to match the contents of documents. Any conditions in the filter should not be mentioned in the query as well.

A logical condition statement is composed of one or more comparison and logical operation statements.

A comparison statement takes the form: `comp(attr, val)`:
- `comp` (eq | ne | gt | gte | lt | lte): comparator
- `attr` (string): name of attribute to apply the comparison to
- `val` (string): is the comparison value

```

```
A logical operation statement takes the form `op(statement1, statement2, ...)`:
- `op` (and | or): logical operator
- `statement1`, `statement2`, ... (comparison statements or logical operation statements): one or more statements to apply the operation to

Make sure that you only use the comparators and logical operators listed above and no others.
Make sure that filters only refer to attributes that exist in the data source.
Make sure that filters only use the attributed names with its function names if there are functions applied on them.
Make sure that filters only use format `YYYY-MM-DD` when handling date data typed values.
Make sure that filters take into account the descriptions of attributes and only make comparisons that are feasible given the type of data being stored.
Make sure that filters are only used as needed. If there are no filters that should be applied return "NO_FILTER" for the filter value.

<< Example 1. >>
Data Source:
```json
{
    "content": "Lyrics of a song",
    "attributes": {
        "artist": {
            "type": "string",
            "description": "Name of the song artist"
        },
        "length": {
            "type": "integer",
            "description": "Length of the song in seconds"
        },
        "genre": {
            "type": "string",
            "description": "The song genre, one of "pop", "rock" or "rap""
        }
    }
}
```

User Query:
What are songs by Taylor Swift or Katy Perry about teenage romance under 3 minutes long in the dance pop genre

Structured Request:
```json
{
    "query": "teenager love",
```
```

```
    "filter": "and(or(eq(\"artist\", \"Taylor Swift\"), eq(\"artist\", \"Katy Perry\")), lt(\"length\", 180), eq(\"genre\", \"pop\"))"
}
```


<< Example 2. >>
Data Source:
```json
{
    "content": "Lyrics of a song",
    "attributes": {
        "artist": {
            "type": "string",
            "description": "Name of the song artist"
        },
        "length": {
            "type": "integer",
            "description": "Length of the song in seconds"
        },
        "genre": {
            "type": "string",
            "description": "The song genre, one of "pop", "rock" or "rap""
        }
    }
}
```

User Query:
What are songs that were not published on Spotify

Structured Request:
```json
{
    "query": "",
    "filter": "NO_FILTER"
}
```


<< Example 3. >>
Data Source:
```json
{
    "content": "Brief summary of a movie",
    "attributes": {
    "genre": {
```

```
        "description": "The genre of the movie. One of ['science fiction', 'comedy', 'drama', 'thriller', 'romance', 'action', 'animated']",
        "type": "string"
    },
    "year": {
        "description": "The year the movie was released",
        "type": "integer"
    },
    "director": {
        "description": "The name of the movie director",
        "type": "string"
    },
    "rating": {
        "description": "A 1-10 rating for the movie",
        "type": "float"
    }
}
}
```

User Query:
Has Greta Gerwig directed any movies about women

Structured Request:

问题：Has **Greta Gerwig** directed any movies about women

模型输出

```json
{
    "query": "women",
    "filter": "eq(\"director\", \"Greta Gerwig\")"
}
```

解析器输出

```
query: women
filter:
  comparator: eq
  attribute: director
```

```
  value: Greta Gerwig
```

问题：What's a highly rated (above 8.5) science fiction film?

模型输出

```
```json
{
    "query": "science fiction film",
    "filter": "gt(\"rating\", 8.5)"
}
```
```

解析器输出

```
"query": "science fiction film",
"filter": {
  "comparator": "gt",
  "attribute": "rating",
  "value": 8.5
}
```

测试代码

```
import os
from uuid import uuid4

from langchain_community.chat_models import QianfanChatEndpoint
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain_community.vectorstores import Chroma
from langchain_core.documents import Document


from langchain.chains.query_constructor.base import AttributeInfo
from langchain.retrievers.self_query.base import SelfQueryRetriever
from langchain_openai import ChatOpenAI

from llms.llm_cfg import BCE_EMBEDDING_MODEL_PATH, BCE_EMBEDDING_MODEL_KWARGS, BCE_EMBEDDING_MODEL_ENCODE_KWARGS, \
    MY_QIANFAN_AK, MY_QIANFAN_SK
```

```
16
17 # 从我本地的一个文件，导入我设置的ZHIPU_AK
18 from ai_docs_cfg import ZHIPU_AK,MY_LANGCHAIN_API_KEY
19
20 # Langsmith 配置，不用可注掉
21 unique_id = uuid4().hex[0:8]
22 os.environ["LANGCHAIN_PROJECT"] = f" RAG self-query - {unique_id}"
23 #os.environ["LANGCHAIN_TRACING_V2"] = 'true'
24 #os.environ["LANGCHAIN_API_KEY"] = MY_LANGCHAIN_API_KEY
25
26 if __name__ == '__main__':
27
28     # 初始化 模型
29     glm4_air_model = ChatOpenAI(
30         model_name="glm-4-air",
31         openai_api_base="https://open.bigmodel.cn/api/paas/v4",
32         openai_api_key=ZHIPU_AK,
33     )
34
35
36     llm = glm4_air_model
37
38     # 网易有道 BCE  嵌入
39     embed_model = HuggingFaceEmbeddings(
40         model_name=BCE_EMBEDDING_MODEL_PATH,
41         model_kwargs=BCE_EMBEDDING_MODEL_KWARGS,
42         encode_kwargs=BCE_EMBEDDING_MODEL_ENCODE_KWARGS
43     )
44
45     docs = [
46         Document(
47             page_content="A bunch of scientists bring back dinosaurs and
   mayhem breaks loose",
48             metadata={"year": 1993, "rating": 7.7, "genre": "science
   fiction"},
49         ),
50         Document(
51             page_content="Leo DiCaprio gets lost in a dream within a dream
   within a dream within a ...",
52             metadata={"year": 2010, "director": "Christopher Nolan", "rating":
   8.2},
53         ),
54         Document(
55             page_content="A psychologist / detective gets lost in a series of
   dreams within dreams within dreams and Inception reused the idea",
56             metadata={"year": 2006, "director": "Satoshi Kon", "rating": 8.6},
57         ),
58         Document(
```

```
            page_content="A bunch of normal-sized women are supremely wholesome and some men pine after them",
            metadata={"year": 2019, "director": "Greta Gerwig", "rating": 8.3},
        ),
        Document(
            page_content="Toys come alive and have a blast doing so",
            metadata={"year": 1995, "genre": "animated"},
        ),
        Document(
            page_content="Three men walk into the Zone, three men walk out of the Zone",
            metadata={
                "year": 1979,
                "director": "Andrei Tarkovsky",
                "genre": "thriller",
                "rating": 9.9,
            },
        ),
    ]
    vdb_path = 'D:\\LLM\\my_projects\\new_langchain_test\\AI_docs\\data\\vectorstore'

    vectorstore = Chroma(persist_directory=vdb_path, embedding_function=embed_model)
    vectorstore.add_documents(docs)

    metadata_field_info = [
        AttributeInfo(
            name="genre",
            description="The genre of the movie. One of ['science fiction', 'comedy', 'drama', 'thriller', 'romance', 'action', 'animated']",
            type="string",
        ),
        AttributeInfo(
            name="year",
            description="The year the movie was released",
            type="integer",
        ),
        AttributeInfo(
            name="director",
            description="The name of the movie director",
            type="string",
        ),
        AttributeInfo(
            name="rating", description="A 1-10 rating for the movie", type="float"
        ),
```

```
    ]
    document_content_description = "Brief summary of a movie"

    retriever = SelfQueryRetriever.from_llm(
        llm,
        vectorstore,
        document_content_description,
        metadata_field_info,
    )

    # This example only specifies a filter
    res = retriever.invoke("I want to watch a movie rated higher than 8.5")
    pass

    # This example specifies a query and a filter
    res = retriever.invoke("Has Greta Gerwig directed any movies about women")
    pass

    # This example specifies a composite filter
    res =  retriever.invoke("What's a highly rated (above 8.5) science fiction film?")
    pass

    # This example specifies a query and composite filter
    res = retriever.invoke(
        "What's a movie after 1990 but before 2005 that's all about toys, and preferably is animated"
    )
    pass

    # This example only specifies a relevant query
    res = retriever.invoke("What are two movies about dinosaurs")
    pass
```