代码说明文档

# 目录

1. 介绍

2. 环境设置

3. Neo4jGraph 实例创建与架构刷新

4. 对话上下文的内存设置

5. 系统提示和回答提示的定义

6. LLM 实例的创建

7. GraphCypherQAChain 实例的创建

8. 用户交互循环

9. 示例和运行逻辑

10. 总结

# 1. 介绍

本说明文档详细介绍了一段用于与 Neo4j 图数据库交互，并通过语言模型生成 Cypher 查询语句和回答用户问题的 Python 代码。该代码利用了 `langchain` 库及其社区扩展、`dotenv` 库来加载环境变量，以及 OpenAI 的聊天模型。

# 2. 环境设置

```python

from dotenv import load\_dotenv

load\_dotenv()

```

此部分代码导入了 `dotenv` 库并加载环境变量文件（.env）。这允许程序从 .env 文件中读取配置，如数据库连接字符串和 API 密钥等敏感信息。

# 3. Neo4jGraph 实例创建与架构刷新

```python

from langchain\_community.graphs import Neo4jGraph

# 创建Neo4jGraph实例并刷新schema

graph = Neo4jGraph()

graph.refresh\_schema()

```

这段代码导入 `Neo4jGraph` 类并创建其实例。`refresh\_schema()` 方法刷新了图数据库的架构信息，确保实例使用的是最新的数据库结构。

# 4. 对话上下文的内存设置

```python

from langchain.memory import ConversationBufferMemory

# 设置对话上下文的内存

memory = ConversationBufferMemory(memory\_key="chat\_history", return\_messages=True)

```

此部分代码设置了对话的内存缓存。`ConversationBufferMemory` 类用于存储对话历史，以便生成查询时可以参考之前的对话内容。

# 5. 系统提示和回答提示的定义

python

from langchain.prompts import PromptTemplate

system\_prompt = """

任务：生成查询图数据库的 Cypher 语句。

指令：只使用提供的关系类型和属性进行查询。不要使用任何未提供的关系类型或属性。

模式：{schema}

生成指令步骤：

1. 根据对话上下文和当前问题，生成最佳的当前指令。

2. 基于生成的当前指令，构建相应的 Cypher 查询语句。

{chat\_history}

这个内容是你必须遵守的：

1. 不要在你的响应中包含任何解释或道歉。

2. 不要回答任何要求你构建 Cypher 语句之外的问题。

3. 除了生成的 Cypher 语句，不要包含任何文本。

重要提示：

1. 带有 `参展公司` 标签的节点不会包含 `name` 属性，请勿在查询中使用此属性。

2. 带有 `参展公司` 标签的节点中，有些包含`上下游合作伙伴` 属性，有些不包含，请在查询中考虑这一点。

3. 使用 WHERE n.`属性` CONTAINS '值' 形式的查询语句，而不是 WHERE n.`属性` = '值'。

4. `会馆` 标签的节点下方的关系只有 `第一层` 和 `第二层`，不要使用其他关系类型。

5. Cypher 语句中不能直接使用 `UNION` 将结果集连接起来。

6. `UNION` 必须连接具有相同列名和类型的结果集。在返回结果中，`RETURN` 子句中的属性名必须完全一致。

7. 忽略 `馆内静态公共设施`这一个标签与其相关的所有内容。

以下是一些针对特定问题生成的 Cypher 语句示例：

1. 来自德国的参展公司有哪些？

MATCH (n:`参展公司`)

WHERE n.`国家\_地区` CONTAINS '德国'

RETURN n.公司名称, n.外语名称, n.基本信息, n.主要展品类型, n.展区位置

2. 会馆3内部有哪些公司？

MATCH (h:`会馆`)-[:第一层|第二层]->(n:`参展公司`)

RETURN n.上下游合作伙伴 , n.外语名称 , n.基本信息 , n.主要展品类型 , n.公司名称

3. 印度参展的公司展区分别在哪里？

MATCH (n:`参展公司`)

WHERE n.`国家\_地区` = '印度'

RETURN n.展区位置, n.外语名称, n.公司名称, n.国家\_地区

问题是：{question}

"""

CYPHER\_GENERATION\_PROMPT = PromptTemplate(

input\_variables=["schema", "question", "chat\_history"],

template=system\_prompt

)

参数解释

1. schema:

意义: 表示图数据库的当前架构，包括节点标签、属性和关系类型等信息。

作用: 模板利用此参数为 Cypher 查询生成提供上下文，使生成的查询语句与数据库架构相匹配。

编写要求: 需要确保架构信息准确、完整，包含所有相关的节点标签、属性和关系类型。

1. question:

意义: 用户输入的问题或查询。

作用: 模板利用此参数生成与用户问题对应的 Cypher 查询语句。

编写要求: 用户问题需要清晰明确，以便语言模型准确生成相应的 Cypher 查询。

1. chat\_history:

意义: 包含之前的对话历史记录。

作用: 模板利用此参数插入先前的对话内容，提供更多上下文信息以生成更准确的 Cypher 查询。

编写要求: 确保对话历史记录详细且与当前查询相关，以便生成的查询语句准确。

模板作用

此模板用于指导语言模型生成 Cypher 查询语句。模板包含了生成查询的规则和示例，通过利用 schema、question、memory 和 chat\_history 等参数，确保生成的查询语句准确且符合用户需求。

#### 回答提示定义

```python

answer\_prompt = """

你是一个助手，帮助形成友好且易于理解的答案。

...

有用的回答：

"""

QA\_GENERATION\_PROMPT = PromptTemplate(

input\_variables=["context", "question"],

template=answer\_prompt

)

```

此部分定义了一个回答提示模板，用于指导语言模型生成用户问题的答案。模板确保生成的答案简洁明了。

# 6. LLM 实例的创建

```python

from langchain\_openai import ChatOpenAI

# 创建LLM实例

llm = ChatOpenAI(

temperature=1, # 设置较低的温度以提高生成结果的一致性

model="glm-4",

openai\_api\_key="7d02576d2b9c13a647ccc4e34586465a.hlnxicK2PkAmllYl",

openai\_api\_base="https://open.bigmodel.cn/api/paas/v4/"

)

```

此部分代码创建了一个 OpenAI 的聊天模型实例。`temperature` 参数控制生成文本的多样性，值越低结果越一致。`openai\_api\_key` 和 `openai\_api\_base` 分别是 API 密钥和 API 基础 URL。

# 7. GraphCypherQAChain 实例的创建

```python

from langchain.chains import GraphCypherQAChain

# 创建 GraphCypherQAChain 实例

chain = GraphCypherQAChain.from\_llm(

llm,

graph=graph,

memory=memory,

verbose=True,

cypher\_prompt=CYPHER\_GENERATION\_PROMPT,

qa\_prompt=QA\_GENERATION\_PROMPT

)

```

此部分代码创建了一个 `GraphCypherQAChain` 实例，该实例整合了语言模型、图数据库和对话记忆，用于生成 Cypher 查询并回答用户问题。

# 8. 用户交互循环

```python

# 开始用户交互循环

def main():

while True:

content = input('user:')

if content == 'quit':

exit()

response=chain.invoke({"query": content})

if isinstance(response, dict) and 'result' in response:

print('answer:', response['result'])

else:

print('answer:', response)

main()

```

此部分代码定义了一个主函数，启动一个用户交互循环。用户输入查询后，调用 `chain.invoke` 方法生成 Cypher 查询并返回结果。用户输入 `quit` 时，程序退出。

# 9. 示例和运行逻辑

#### 示例：

```python

# 用户问题示例

user: 哪些公司来自德国？

# 生成的 Cypher 查询示例

MATCH (n:`参展公司`)

WHERE n.`国家\_地区` CONTAINS '德国'

RETURN n.公司名称, n.外语名称, n.基本信息, n.主要展品类型, n.展区位置

```

#### 运行逻辑：

1. 用户输入查询问题。

2. 程序通过 `GraphCypherQAChain` 生成相应的 Cypher 查询语句。

3. 连接在neo4j中进行查询

4. 查询结果通过 `ChatOpenAI` 模型生成回答。

5. 返回回答给用户。

# 10. 总结

该代码利用 `langchain` 和 `Neo4j` 实现了一个智能问答系统，能够根据用户输入的问题生成并执行 Cypher 查询，并通过语言模型生成友好的答案。整个过程包括了环境变量加载、图数据库实例创建、对话记忆管理、系统提示定义、语言模型实例创建以及用户交互循环等多个步骤。