# 第7章 基于知识图谱的医疗诊断系统

通过上一章节的实际项目演示，我们已经学习了如何基于图数据库的电影知识数据来构建一个电影知识问答系统。为了便于大家的理解与学习，我们在很多地方做了简化处理，那么实际工业实践中是如何构建一个更加智能化的问答系统呢？本章节我们将从零开始，一步步学习如何从无到有构建一个复杂且高效的问答系统。希望通过本章的学习，你可以根据自己的实际需求，按照本章的步骤进行操作，在本章结束的时候即可从零开始搭建一个符合实际需求的问答系统。

下面将开始搭建一个基于知识图谱的电影问答系统，它能够理解用户的查询意图，并将用户查询的相关电影知识返回给用户。

本章主要涉及到的知识点有：

* 系统介绍：介绍系统实现的框架。
* 基于知识图谱的电影知识问答系统实现：介绍项目中用到的数据集的采集整理。

注意：本章代码地址：xxx

## 7.1 系统介绍

本节介绍一下基于知识图谱的电影知识问答系统的主要功能及系统实现的框架。

### 6.1.1 基于知识图谱的电影问答系统

首先要确定一下基于知识图谱的电影问答系统能做什么，我们的问答系统是基于上一章节构建的电影知识来实现的，因此我们可以查询电影相关的信息，比如电影的剧情简介信息、电影评分、电影上映时间以及电影类型或风格信息。同样我们也可以查询演员的相关信息，比如演员出生日期、演员的国籍信息、演员简介信息等，此外通过电影与演员之间的关系，我们也可以查询某演员出演过那些电影以及某电影有那些演员出演、某演员出演过那些类型风格的电影等信息。

为了提高我们问答系统的用户体验，让其更具智能化，我们可以让我们的问答系统与用户进行简单的问候互动比如理解用户的问候信息及结束问答等。

该问答系统的实现架构图如图6.1所示：



图6.1 基于知识图谱的电影知识问答系统架构图

### 6.1.2 基于知识图谱的电影知识问答系统模块介绍

基于知识图谱的电影知识问答系统主要分为三个功能模块，分别为文本预处理模块、用户意图识别模块与答案查询模块，下面分别介绍下各个功能模块的实现思路。

文本预处理模块：该模块主要完成对用户输入问题的文本预处理，主要包括对输入问题进行文本清理、分词、去除停用词及文本结构化，同时在该模块中对用户输入的问候语及结束问答进行相应识别回复。

用户意图识别模块：该模块主要完成对用户输入问题提取关键信息，如提取电影名或演员名，这里采用jieba分词来实现，根据词性标注信息来抽取关键信息。同时该模块还将使用预训练的分类模型对用户输入的问题进行分类，获取相应的问题模板，最后结合关键信息与问题模板获取用户的意图信息。

答案查询模块：该模块根据用户意图生成相应的知识图谱查询语句，通过对电影知识数据的查询获取最终的答案并返回给用户。

## 6.2 基于知识图谱的电影问答系统

### 6.2.1 文本预处理模块实现

我们的电影知识问答系统首先需要接收用户的问题文本信息，由于用户输入的文本信息往往并不规范，如包含无效字符等情况。为了便于后续对问题文本进行分析，我们需要对用户输入问题文本进行基本的预处理。这里仅进行了初步的预处理，包括去除文本中的中文标点符号及英文标点符号，并通过正则表达式过滤掉一些不合法字符。具体实现代码如下：

注意，在实际项目中，针对不同的需求，可以根据具体需求对问题文本做相应的处理，这里的文本预处理仅用于简单演示。

**代码6.1 问题文本预处理代码**

#!/usr/bin/env python  
# \_\*\_ coding:utf-8 \_\*\_  
import re  
  
  
def text\_processing(text):  
 *'''  
 对用户输入的问题文本进行清理* ***:param*** *text:* ***:return****:  
 '''* text = text.replace(' ', '')  
 clean\_text = clean\_punctuation(text)  
 # 过滤非中文字符  
 pattern = re.compile("[^\u4e00-\u9fa5]")  
 clean\_text = re.sub(pattern, '', clean\_text)  
 print(clean\_text)  
 return clean\_text  
  
  
def clean\_punctuation(text):  
 *'''  
 清理标点符号* ***:param*** *text:* ***:return****:  
 '''* # 清理中文标点符号  
 cn\_punctuation = "！？｡＂＃＄％＆＇（）＊＋，－／：；＜＝＞＠［＼］＾＿｀｛｜｝～｟｠｢｣､、〃》「」『』【】〔〕〖〗〘〙〚〛〜〝〞〟〰〾〿–—‘'‛“”„‟…‧﹏."  
 cn\_re\_punctuation = "[{}]+".format(cn\_punctuation)  
 text = re.sub(cn\_re\_punctuation, "", text)  
 # 清理英文标点符号  
 en\_punctuation = '!"#$%&\'()\*+,-./:;<=>?@[\\]^\_`{|}~'  
 en\_re\_punctuation2 = "[{}]+".format(en\_punctuation)  
 text = re.sub(en\_re\_punctuation2, "", text)  
 return text  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # test  
 user\_input = '你好，请问电影卧虎藏龙的评分是多少？'  
 text\_processing(user\_input)

为了提高我们电影知识问答系统的易用性，便于与用户之间进行交互，我们这里对输入的问题文本做一个简单的处理，即判定用户的输入是否合法，对于用户常见的输入可以给与相应的回复及简单的引导，便于用户方便快捷的使用本问答系统。

注意，这里仅简单设置了几种规则，当用户的输入满足相应的规则，则问答系统可以快速给予相应的回复，同时给与一定的指引信息，帮助用户快速掌握该问答系统的使用方式。针对不同的使用场景，可以设置相应的回复方式。具体代码如下：

**代码6.2 对问题进行规则性回复**

#!/usr/bin/env python  
# \_\*\_ coding:utf-8 \_\*\_  
  
import random  
  
  
def rule\_based\_reply(user\_input):  
 *'''  
 对用户的输入信息判定是否需要基于规则进行回复* ***:param*** *user\_input:* ***:return****:  
 '''* # 是否激活规则回复  
 is\_active\_rule\_base = False  
 greeting\_user\_dia = ['你好', '您好', '早上好', '早安', '哈喽', '嗨']  
 greeting\_sys\_dia = ['你好', '您好', '哈喽', '嗨']  
 bye\_user\_dia = ['再见', '拜拜']  
 bye\_sys\_dia = ['再见']  
 chat\_user\_dia = ['你会做什么', '你会什么', '你能干什么']  
 chat\_sys\_dia = ['您可以查询电影相关信息。', '您可以查询演员相关信息。']  
 if user\_input in greeting\_user\_dia:  
 greeting\_info = '，请输入您要查询的电影相关信息，如李连杰演过什么电影？'  
 rule\_based\_res = random.choice(greeting\_sys\_dia) + greeting\_info  
 is\_active\_rule\_base = True  
 if user\_input in bye\_user\_dia:  
 rule\_based\_res = random.choice(bye\_sys\_dia)  
 is\_active\_rule\_base = True  
 if user\_input in chat\_user\_dia:  
 rule\_based\_res = random.choice(chat\_sys\_dia)  
 is\_active\_rule\_base = True  
 if is\_active\_rule\_base:  
 return is\_active\_rule\_base, rule\_based\_res  
 else:  
 return is\_active\_rule\_base, user\_input

### 6.2.2 用户意图识别模块实现

在文本预处理模块处理后，我们需要对用户输入的问题文本进行意图识别，确定用户想查询的信息内容。对用户意图的识别主要分三步来完成，首先通过命名实体识别抽取问题中的关键信息，然后对问题进行文本分类，确定该问题想要获取的信息，最后将获取到的信息填充为预先定义的问题模板。

首先对于关键信息的抽取，这部分工作采用命名实体识别来完成，主要用来识别问题文本中出现的人名信息、电影名信息等关键信息，这里我们使用jieba分词中的词性标注功能来完成。具体代码如下：

**代码6.3 抽取问题文本中关键信息**

#!/usr/bin/env python

# \_\*\_ coding:utf-8 \_\*\_

import jieba.posseg

import re

def jieba\_pos\_tagging(text):

'''

对文本进行词性标记

:param text:

:return:

'''

# 用户自定义词库路径

user\_dict\_path = './../data/userdict3.txt'

# 导入用户自定义词库

jieba.load\_userdict(user\_dict\_path)

# 对文本进行词性分析

text\_pos\_seg = jieba.posseg.cut(text)

# 存储分词及词性分析结果

result = []

# 存储分词结果

text\_word = []

# 存储词性分析结果

text\_pos = []

# 遍历词性分析结果并返回

for word\_pos in text\_pos\_seg:

curr\_word\_pos = f"{word\_pos.word}/{word\_pos.flag}"

result.append(curr\_word\_pos)

word = word\_pos.word

flag = word\_pos.flag

text\_word.append(str(word).strip())

text\_pos.append(str(flag).strip())

# 判断分词结果与词性分析结果是否匹配

assert len(text\_pos) == len(text\_word)

print('词性标注结果：', result)

return result, text\_word, text\_pos

if \_\_name\_\_ == '\_\_main\_\_':

# test

jieba\_pos\_tagging('章子怡演过多少部电影')

如对于问题文本“章子怡演过多少部电影”，经过jieba的词性标注，其标注结果如下：

['章子怡/nr', '演/v', '过/ug', '多少/m', '部/n', '电影/n']

其中“nr”表示该词语为人名，“v”表示该词的词性为动词，“m”表示该词的词性为数量词，jieba分词中词性标注表如下所示：

表6.1 jieba分词词性标注表

|  |  |  |
| --- | --- | --- |
| 词性标注 | 词性含义 | 命名规则 |
| Ag | 形语素 | 形容词性语素。形容词代码为 a，语素代码ｇ前面置以A。 |
| a | 形容词 | 取英语形容词 adjective的第1个字母。 |
| ad | 副形词 | 直接作状语的形容词。形容词代码 a和副词代码d并在一起。 |
| an | 名形词 | 具有名词功能的形容词。形容词代码 a和名词代码n并在一起。 |
| b | 区别词 | 取汉字“别”的声母。 |
| c | 连词 | 取英语连词 conjunction的第1个字母。 |
| dg | 副语素 | 副词性语素。副词代码为 d，语素代码ｇ前面置以D。 |
| d | 副词 | 取 adverb的第2个字母，因其第1个字母已用于形容词。 |
| e | 叹词 | 取英语叹词 exclamation的第1个字母。 |
| f | 方位词 | 取汉字“方” |
| g | 语素 | 绝大多数语素都能作为合成词的“词根”，取汉字“根”的声母。 |
| h | 前接成分 | 取英语 head的第1个字母。 |
| i | 成语 | 取英语成语 idiom的第1个字母。 |
| j | 简称略语 | 取汉字“简”的声母。 |
| k | 后接成分 |  |
| l | 习用语 | 习用语尚未成为成语，有点“临时性”，取“临”的声母。 |
| m | 数词 | 取英语 numeral的第3个字母，n，u已有他用。 |
| Ng | 名语素 | 名词性语素。名词代码为 n，语素代码ｇ前面置以N。 |
| n | 名词 | 取英语名词 noun的第1个字母。 |
| nr | 人名 | 名词代码 n和“人(ren)”的声母并在一起。 |
| ns | 地名 | 名词代码 n和处所词代码s并在一起。 |
| nt | 机构团体 | “团”的声母为 t，名词代码n和t并在一起。 |
| nz | 其他专名 | “专”的声母的第 1个字母为z，名词代码n和z并在一起。 |
| o | 拟声词 | 取英语拟声词 onomatopoeia的第1个字母。 |
| p | 介词 | 取英语介词 prepositional的第1个字母。 |
| q | 量词 | 取英语 quantity的第1个字母。 |
| r | 代词 | 取英语代词 pronoun的第2个字母,因p已用于介词。 |
| s | 处所词 | 取英语 space的第1个字母。 |
| tg | 时语素 | 时间词性语素。时间词代码为 t,在语素的代码g前面置以T。 |
| t | 时间词 | 取英语 time的第1个字母。 |
| u | 助词 | 取英语助词 auxiliary |
| vg | 动语素 | 动词性语素。动词代码为 v。在语素的代码g前面置以V。 |
| v | 动词 | 取英语动词 verb的第一个字母。 |
| vd | 副动词 | 直接作状语的动词。动词和副词的代码并在一起。 |
| vn | 名动词 | 指具有名词功能的动词。动词和名词的代码并在一起。 |
| w | 标点符号 |  |
| x | 非语素字 | 非语素字只是一个符号，字母 x通常用于代表未知数、符号。 |
| y | 语气词 | 取汉字“语”的声母。 |
| z | 状态词 | 取汉字“状”的声母的前一个字母。 |
| un | 未知词 | 不可识别词及用户自定义词组。取英文Unkonwn首两个字母。(非北大标准，CSW分词中定义) |

在关键信息抽取中，为了提高信息抽取的准确度，我们在这里人工构建了电影名及人名词典，作为jieba分词的自定义词库，该词库包含三列，第一列为当前问答系统中出现的电影名及人名信息，第二列为该词的词频信息，为了便于处理，这里默认设置为15，第三列为词性信息。如下所示：

白玉老虎 15 nm

武馆 15 nm

合家欢 15 nm

孔雀王朝 15 nm

Bian cheng san xia 15 nm

黄飞鸿少林拳 15 nm

Fei yan jin dao 15 nm

香港也疯狂 15 nm

巩俐 15 nr

乔宏 15 nr

李连杰 15 nr

梁朝伟 15 nr

张曼玉 15 nr

章子怡 15 nr

甄子丹 15 nr

抽取问题中的关键信息后，为了理解问题的意图信息，我们这里设计了一些问题模板，通过对用户输入的问题文本进行分类，进而实现对用户意图的识别。对用户问题进行分类时，我们采用朴素贝叶斯分类器来实现，通过预先根据用户习惯构造的问题数据来训练该分类器，具体代码如下：

**代码6.4 问题文本分类**

#!/usr/bin/env python

# \_\*\_ coding:utf-8 \_\*\_

from question\_classification import Question\_classify

def get\_question\_template(text\_word, text\_pos):

'''

获取问题模板

:param text\_word:

:param text\_pos:

:return:

'''

# 抽取问题中的名词信息

for item in ['nr', 'nm', 'ng']:

while (item in text\_pos):

idx = text\_pos.index(item)

text\_word[idx] = item

text\_pos[idx] = item + "ed"

# 将用户输入问题转换为抽象问题

str\_question = "".join(text\_word)

print("抽象问题为：", str\_question)

# 通过预训练的分类器获取抽象问题相应的模板编号

classify\_model = Question\_classify()

question\_template\_num = classify\_model.predict(str\_question)

print("抽象问题相应模板编号：", question\_template\_num)

tmp = get\_question\_mode()

question\_template = tmp[question\_template\_num]

print("问题模板：", question\_template)

question\_template\_id\_str = str(question\_template\_num) + "\t" + question\_template

return question\_template\_id\_str

def get\_question\_mode():

# 读取问题模板

with(open("./../data/question/question\_classification.txt", "r", encoding="utf-8")) as f:

question\_mode\_list = f.readlines()

question\_mode\_dict = {}

for one\_mode in question\_mode\_list:

# 读取一行

mode\_id, mode\_str = str(one\_mode).strip().split(":")

# 处理一行，并存入

question\_mode\_dict[int(mode\_id)] = str(mode\_str).strip()

# print(self.question\_mode\_dict)

return question\_mode\_dict

在上面的代码中将问题转换为抽象问题，如问题文本“章子怡演过多少部电影”将其中的人名信息转换为抽象问题后为：“nr演过多少部电影”，将该抽象问题传入问题文本分类器中进行分类。

问题文本分类器采用sklearn的朴素贝叶斯来实现，具体代码如下：

**代码6.5 问题文本分类器**

#!/usr/bin/env python  
# \_\*\_ coding:utf-8 \_\*\_  
import pandas as pd  
from pandas import Series, DataFrame  
from sklearn.naive\_bayes import MultinomialNB  
from sklearn.feature\_extraction.text import TfidfVectorizer  
import os  
import re  
import jieba  
  
  
class Question\_classify():  
 *'''  
 定义用于问题模板分类的类函数  
 '''* # 类的初始化  
 def \_\_init\_\_(self):  
 # 读取训练数据及标签信息  
 self.train\_x, self.train\_y = self.read\_train\_data()  
 # 训练分类模型  
 self.model = self.train\_model\_NB()  
  
 def read\_train\_data(self):  
 *'''  
 获取训练数据* ***:return****:  
 '''* # 存储训练数据  
 train\_x = []  
 # 存储类别信息  
 train\_y = []  
 # 训练数据文件存储地址  
 train\_data\_path = './../data/question/'  
 # 读取训练数据文件  
 file\_list = getfilelist(train\_data\_path)  
 # 遍历所有文件  
 for fname in file\_list:  
 # 正则匹配文件名中的数字  
 num = re.sub(r'\D', '', fname)  
 # 若该文件名有数字，则读取该文件  
 if str(num).strip() != '':  
 # 将文件名中的数字作为分类标签信息  
 label\_num = int(num)  
 # 读取文件内容  
 with(open(fname, 'r', encoding='utf-8')) as fr:  
 data\_list = fr.readlines()  
 for one\_line in data\_list:  
 # 对文本进行分词  
 word\_list = list(jieba.cut(str(one\_line).strip()))  
 # 将文本存储为训练数据  
 train\_x.append(" ".join(word\_list))  
 train\_y.append(label\_num)  
 return train\_x, train\_y  
  
 def train\_model\_NB(self):  
 *'''  
 训练朴素贝叶斯分类器* ***:return****:  
 '''* # 获取训练数据  
 X\_train, y\_train = self.train\_x, self.train\_y  
 # 初始化TfidfVectorizer实例  
 self.tv = TfidfVectorizer()  
 # 通过TF-IDF对文本数据进行向量化处理  
 train\_data = self.tv.fit\_transform(X\_train).toarray()  
 # 初始化贝叶斯分类器并设置参数alpha为0.01  
 clf = MultinomialNB(alpha=0.01)  
 # 训练模型  
 clf.fit(train\_data, y\_train)  
 return clf  
  
 def predict(self, question):  
 *'''  
 使用训练好的模型对文本进行预测* ***:param*** *question:* ***:return****:  
 '''* # 对输入文本分词  
 question = [" ".join(list(jieba.cut(question)))]  
 # 将文本转换为向量  
 test\_data = self.tv.transform(question).toarray()  
 # 对文本向量进行预测  
 y\_predict = self.model.predict(test\_data)[0]  
 print("predict type:", y\_predict)  
 return y\_predict  
  
  
def getfilelist(root\_path):  
 *'''  
 获取该路径下全部文件的路径信息* ***:param*** *root\_path:* ***:return****:  
 '''* # 存储文件路径信息  
 file\_path\_list = []  
 # 存储文件名信息  
 file\_name = []  
 # 遍历文件目录  
 walk = os.walk(root\_path)  
 for root, dirs, files in walk:  
 for name in files:  
 filepath = os.path.join(root, name)  
 file\_name.append(name)  
 file\_path\_list.append(filepath)  
 # print('文件名信息：', file\_name)  
 # print('文件路径信息：', file\_path\_list)  
 return file\_path\_list  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 qc = Question\_classify()  
 qc.predict("张学友的个人信息")

在文本分类器的训练中，我们需要读取训练数据，该数据依据类别存储在不同的文本文件中，如关于电影评分的文本存储在文件“【0】评分.txt”中，该文件中内容如下：

nm的评分是多少

nm得了多少分

nm的评分有多少

nm的评分

nm的分数是

nm电影分数是多少

nm评分

nm的分数是多少

nm这部电影的评分是多少

该文件中内容为当用户询问电影评分信息时，根据用户习惯总结出用户输入的文本，通过对用户输入信息的分类，即可判定用户意图查询的信息内容。在分类器的训练中，将这些文本内容通过TF-IDF表示为向量，然后通过构建朴素贝叶斯分类器进行训练，最后利用训练好的分类模型对用户输入的新问题进行分类，返回用户输入问题所属的问题类别编号，通过该类别编号信息，获取相应的问题模板信息。

问题所属类别编号及相应问题模板信息如下：

0:nm 评分  
1:nm 上映时间  
2:nm 类型  
3:nm 简介  
4:nm 演员列表  
5:nnt 介绍  
6:nnt ng 电影作品  
7:nnt 电影作品  
8:nnt 参演评分 大于 x  
9:nnt 参演评分 小于 x  
10:nnt 电影类型  
11:nnt nnr 合作 电影列表  
12:nnt 电影数量  
13:nnt 出生日期

如对于上面的抽象问题“nr演过多少部电影”，经过分类判定该问题类型编号为12，根据上面的信息，确定该问题模板信息为“nnt 电影数量”，表示该问题意图查询的信息为nnt出演的电影数量。

### 6.2.3 答案查询模块实现

根据用户意图识别模块的分析，我们已经可以确定用户想要查询的信息，根据已经获取的问题模板，在答案查询模块中，我们将问题模板转换为相应的图数据库查询条件，查询相应的信息，并将其按照答案模板生成相应的答案信息。

**代码6.6 答案查询模块**

#!/usr/bin/env python  
# \_\*\_ coding:utf-8 \_\*\_  
  
  
*'''  
0:nm 评分  
1:nm 上映时间  
2:nm 类型  
3:nm 简介  
4:nm 演员列表  
5:nnt 介绍  
6:nnt ng 电影作品  
7:nnt 电影作品  
8:nnt 参演评分 大于 x  
9:nnt 参演评分 小于 x  
10:nnt 电影类型  
11:nnt nnr 合作 电影列表  
12:nnt 电影数量  
13:nnt 出生日期  
'''*from query import Query  
import re  
  
  
class QuestionTemplate():  
 def \_\_init\_\_(self):  
 self.q\_template\_dict = {  
 0: self.get\_movie\_rating,  
 1: self.get\_movie\_releasedate,  
 2: self.get\_movie\_type,  
 3: self.get\_movie\_introduction,  
 4: self.get\_movie\_actor\_list,  
 5: self.get\_actor\_info,  
 6: self.get\_actor\_act\_type\_movie,  
 7: self.get\_actor\_act\_movie\_list,  
 8: self.get\_movie\_rating\_bigger,  
 9: self.get\_movie\_rating\_smaller,  
 10: self.get\_actor\_movie\_type,  
 11: self.get\_cooperation\_movie\_list,  
 12: self.get\_actor\_movie\_num,  
 13: self.get\_actor\_birthday  
 }  
  
 # 连接数据库  
 self.graph = Query()  
 # 测试数据库是否连接成功  
 result = self.graph.run("match (m:Movie)-[]->() where m.title='卧虎藏龙' return m.rating")  
 # if result:  
 # print('图数据库连接成功，查询结果如下：',result)  
 # exit()  
  
 def get\_question\_answer(self, question, template):  
 # 判定问题模板的格式是否正确  
 assert len(str(template).strip().split("\t")) == 2  
 template\_id, template\_str = int(str(template).strip().split("\t")[0]), str(template).strip().split("\t")[1]  
 self.template\_id = template\_id  
 self.template\_str2list = str(template\_str).split()  
  
 # 预处理问题文本  
 question\_word, question\_flag = [], []  
 for one in question:  
 word, flag = one.split("/")  
 question\_word.append(str(word).strip())  
 question\_flag.append(str(flag).strip())  
 assert len(question\_flag) == len(question\_word)  
 self.question\_word = question\_word  
 self.question\_flag = question\_flag  
 self.raw\_question = question  
 # 根据问题模板id来做对应的查询处理，获取答案  
 answer = self.q\_template\_dict[template\_id]()  
 return answer  
  
 # 获取电影名字  
 def get\_movie\_name(self):  
 # 获取nm在原问题中的下标  
 tag\_index = self.question\_flag.index("nm")  
 # 获取电影名称  
 movie\_name = self.question\_word[tag\_index]  
 return movie\_name  
  
 def get\_name(self, type\_str):  
 name\_count = self.question\_flag.count(type\_str)  
 if name\_count == 1:  
 # 获取nm在原问题中的下标  
 tag\_index = self.question\_flag.index(type\_str)  
 # 获取电影名称  
 name = self.question\_word[tag\_index]  
 return name  
 else:  
 result\_list = []  
 for i, flag in enumerate(self.question\_flag):  
 if flag == str(type\_str):  
 result\_list.append(self.question\_word[i])  
 return result\_list  
  
 def get\_num\_x(self):  
 x = re.sub(r'\D', "", "".join(self.question\_word))  
 return x  
  
 # 0:nm 评分  
 def get\_movie\_rating(self):  
 # 获取电影名称，这个是在原问题中抽取的  
 movie\_name = self.get\_movie\_name()  
 cql = f"match (m:Movie)-[]->() where m.title='{movie\_name}' return m.rating"  
 print(cql)  
 answer = self.graph.run(cql)[0]  
 print(answer)  
 answer = round(answer, 2)  
 final\_answer = movie\_name + "电影评分为" + str(answer) + "分！"  
 return final\_answer  
  
 # 1:nm 上映时间  
 def get\_movie\_releasedate(self):  
 movie\_name = self.get\_movie\_name()  
 cql = f"match(m:Movie)-[]->() where m.title='{movie\_name}' return m.releasedate"  
 print(cql)  
 answer = self.graph.run(cql)[0]  
 final\_answer = movie\_name + "的上映时间是" + str(answer) + "！"  
 return final\_answer  
  
 # 2:nm 类型  
 def get\_movie\_type(self):  
 movie\_name = self.get\_movie\_name()  
 cql = f"match(m:Movie)-[r:is]->(b) where m.title='{movie\_name}' return b.name"  
 print(cql)  
 answer = self.graph.run(cql)  
 answer\_set = set(answer)  
 answer\_list = list(answer\_set)  
 answer = "、".join(answer\_list)  
 final\_answer = movie\_name + "是" + str(answer) + "等类型的电影！"  
 return final\_answer  
  
 # 3:nm 简介  
 def get\_movie\_introduction(self):  
 movie\_name = self.get\_movie\_name()  
 cql = f"match(m:Movie)-[]->() where m.title='{movie\_name}' return m.introduction"  
 print(cql)  
 answer = self.graph.run(cql)[0]  
 final\_answer = movie\_name + "主要讲述了" + str(answer) + "！"  
 return final\_answer  
  
 # 4:nm 演员列表  
 def get\_movie\_actor\_list(self):  
 movie\_name = self.get\_movie\_name()  
 cql = f"match(n:Person)-[r:actedin]->(m:Movie) where m.title='{movie\_name}' return n.name"  
 print(cql)  
 answer = self.graph.run(cql)  
 answer\_set = set(answer)  
 answer\_list = list(answer\_set)  
 answer = "、".join(answer\_list)  
 final\_answer = movie\_name + "由" + str(answer) + "等演员主演！"  
 return final\_answer  
  
 # 5:nnt 介绍  
 def get\_actor\_info(self):  
 actor\_name = self.get\_name('nr')  
 cql = f"match(n:Person)-[]->() where n.name='{actor\_name}' return n.biography"  
 print(cql)  
 answer = self.graph.run(cql)[0]  
 final\_answer = answer  
 return final\_answer  
  
 # 6:nnt ng 电影作品  
 def get\_actor\_act\_type\_movie(self):  
 actor\_name = self.get\_name("nr")  
 type = self.get\_name("ng")  
 # 查询电影名称  
 cql = f"match(n:Person)-[]->(m:Movie) where n.name='{actor\_name}' return m.title"  
 print(cql)  
 movie\_name\_list = list(set(self.graph.run(cql)))  
 # 查询类型  
 result = []  
 for movie\_name in movie\_name\_list:  
 movie\_name = str(movie\_name).strip()  
 try:  
 cql = f"match(m:Movie)-[r:is]->(t) where m.title='{movie\_name}' return t.name"  
 # print(cql)  
 temp\_type = self.graph.run(cql)  
 if len(temp\_type) == 0:  
 continue  
 if type in temp\_type:  
 result.append(movie\_name)  
 except:  
 continue  
 answer = "、".join(result)  
 print(answer)  
 final\_answer = actor\_name + "演过的" + type + "电影有:\n" + answer + "。"  
 return final\_answer  
  
 # 7:nnt 电影作品  
 def get\_actor\_act\_movie\_list(self):  
 actor\_name = self.get\_name("nr")  
 answer\_list = self.get\_actorname\_movie\_list(actor\_name)  
 answer = "、".join(answer\_list)  
 final\_answer = actor\_name + "演过" + str(answer) + "等电影！"  
 return final\_answer  
  
 def get\_actorname\_movie\_list(self, actorname):  
 # 查询电影名称  
 cql = f"match(n:Person)-[]->(m:Movie) where n.name='{actorname}' return m.title"  
 print(cql)  
 answer = self.graph.run(cql)  
 answer\_set = set(answer)  
 answer\_list = list(answer\_set)  
 return answer\_list  
  
 # 8:nnt 参演评分 大于 x  
 def get\_movie\_rating\_bigger(self):  
 actor\_name = self.get\_name('nr')  
 x = self.get\_num\_x()  
 cql = f"match(n:Person)-[r:actedin]->(m:Movie) where n.name='{actor\_name}' and m.rating>={x} return m.title"  
 print(cql)  
 answer = self.graph.run(cql)  
 answer = "、".join(answer)  
 answer = str(answer).strip()  
 final\_answer = actor\_name + "演的电影评分大于" + x + "分的有" + answer + "等！"  
 return final\_answer  
  
 def get\_movie\_rating\_smaller(self):  
 actor\_name = self.get\_name('nr')  
 x = self.get\_num\_x()  
 cql = f"match(n:Person)-[r:actedin]->(m:Movie) where n.name='{actor\_name}' and m.rating<{x} return m.title"  
 print(cql)  
 answer = self.graph.run(cql)  
 answer = "、".join(answer)  
 answer = str(answer).strip()  
 final\_answer = actor\_name + "演的电影评分小于" + x + "分的有" + answer + "等！"  
 return final\_answer  
  
 def get\_actor\_movie\_type(self):  
 actor\_name = self.get\_name("nr")  
 # 查询电影名称  
 cql = f"match(n:Person)-[]->(m:Movie) where n.name='{actor\_name}' return m.title"  
 print(cql)  
 movie\_name\_list = list(set(self.graph.run(cql)))  
 # 查询类型  
 result = []  
 for movie\_name in movie\_name\_list:  
 movie\_name = str(movie\_name).strip()  
 try:  
 cql = f"match(m:Movie)-[r:is]->(t) where m.title='{movie\_name}' return t.name"  
 # print(cql)  
 temp\_type = self.graph.run(cql)  
 if len(temp\_type) == 0:  
 continue  
 result += temp\_type  
 except:  
 continue  
 answer = "、".join(list(set(result)))  
 print(answer)  
 final\_answer = actor\_name + "演过的电影有" + answer + "等类型。"  
 return final\_answer  
  
 def get\_cooperation\_movie\_list(self):  
 # 获取演员名字  
 actor\_name\_list = self.get\_name('nr')  
 movie\_list = {}  
 for i, actor\_name in enumerate(actor\_name\_list):  
 answer\_list = self.get\_actorname\_movie\_list(actor\_name)  
 movie\_list[i] = answer\_list  
 result\_list = list(set(movie\_list[0]).intersection(set(movie\_list[1])))  
 print(result\_list)  
 answer = "、".join(result\_list)  
 final\_answer = actor\_name\_list[0] + "和" + actor\_name\_list[1] + "一起演过的电影主要是" + answer + "!"  
 return final\_answer  
  
 def get\_actor\_movie\_num(self):  
 actor\_name = self.get\_name("nr")  
 answer\_list = self.get\_actorname\_movie\_list(actor\_name)  
 movie\_num = len(set(answer\_list))  
 answer = movie\_num  
 final\_answer = actor\_name + "演过" + str(answer) + "部电影!"  
 return final\_answer  
  
 def get\_actor\_birthday(self):  
 actor\_name = self.get\_name('nr')  
 cql = f"match(n:Person)-[]->() where n.name='{actor\_name}' return n.birth"  
 print(cql)  
 answer = self.graph.run(cql)[0]  
 final\_answer = actor\_name + "的生日是" + answer + "。"  
 return final\_answer

针对不同的问题模板，需要构造不同的图数据库查询语句，如对于上面的例子，查询“章子怡演过多少部电影”时候，根据获取到的问题模板“nnt 电影数量”，通过方法“get\_actor\_movie\_num”来查询电影演员名为“章子怡”出演的电影数量，该查询语句如下所示：

“cql = f"match(n:Person)-[]->(m:Movie) where n.name='{actorname}' return m.title"”

查询结束后，将结果根据模板整理为相应的答案格式并返回。

“final\_answer = actor\_name + "演过" + str(answer) + "部电影!"”

调用以上答案查询模块的代码如下，此外对于无法获取答案的问题也需要做相应的处理。

**代码6.7 调用答案查询模块**

#!/usr/bin/env python  
# \_\*\_ coding:utf-8 \_\*\_  
from question\_template import QuestionTemplate  
  
# 根据问题模板的具体类容，构造cql语句，并查询  
def query\_template(text\_pos,question\_template\_id\_str):  
 # 调用问题模板类中的获取答案的方法  
 try:  
 questiontemplate = QuestionTemplate()  
 answer = questiontemplate.get\_question\_answer(text\_pos,question\_template\_id\_str)  
 except:  
 answer = "对不起，暂时无法回答该问题！"  
 return answer

### 6.2.4 电影知识问答系统实现

完成以上全部模块后，我们分别调用相应模块，搭建最终的电影知识问答系统，具体代码如下所示：

**代码6.8 电影知识问答系统**

#!/usr/bin/env python  
# \_\*\_ coding:utf-8 \_\*\_  
from question\_preprocess import text\_processing  
from rule\_based\_reply import rule\_based\_reply  
from text\_pos\_tagging import jieba\_pos\_tagging  
from compose\_question\_template import get\_question\_template  
from query\_template import query\_template  
  
  
def main():  
 # 获取用户输入  
 user\_input = input('您好，请输入您要查询的电影相关信息，如李连杰演过什么电影？：')  
 print(user\_input)  
 # 文本预处理  
 clean\_question = text\_processing(user\_input)  
 # 基于规则的问答回复  
 is\_active\_rule\_base, sys\_reply = rule\_based\_reply(clean\_question)  
 if is\_active\_rule\_base:  
 return sys\_reply  
 else:  
 # 词性标注获取关键信息及句子模板分类  
 result, text\_word, text\_pos = jieba\_pos\_tagging(clean\_question)  
 # 获取问题模板信息  
 question\_template\_id\_str = get\_question\_template(text\_word, text\_pos)  
 # 查询图数据库，根据答案生成模板获取答案  
 answer = query\_template(result, question\_template\_id\_str)  
 print(answer)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

### 6.2.5 电影知识问答系统效果展示

您好，请输入您要查询的电影相关信息，如李连杰演过什么电影？：章子怡演过什么电影

Building prefix dict from the default dictionary ...

Loading model from cache C:\Users\12261\AppData\Local\Temp\jieba.cache

Loading model cost 0.715 seconds.

Prefix dict has been built successfully.

章子怡演过龙在哪里？、太平轮(下)、奔爱、Fei chang wan mei、Godzilla vs. Kong、茉莉花开、Ye yan、非常幸运、조폭 마누라 2: 돌아온 전설、角色于我、紫蝴蝶、Mei Lanfang、最爱、2046、太平轮(上)、十面埋伏、危险关系、Untitled Cloverfield Movie、在一起、星星点灯、卧虎藏龙、Rush Hour 2、从天儿降、オペレッタ狸御殿、一代宗师、TMNT、무사、无问西东、Memoirs of a Geisha、Cause: The Birth of Hero、Shu shan zheng zhuan、Horsemen、我的父亲母亲、Godzilla: King of Monsters、越来越好之村晚、英雄、建国大业、罗曼蒂克消亡史等电影！

您好，请输入您要查询的电影相关信息，如李连杰演过什么电影？：梁朝伟演过多少部电影

Building prefix dict from the default dictionary ...

Loading model from cache C:\Users\12261\AppData\Local\Temp\jieba.cache

Loading model cost 0.682 seconds.

Prefix dict has been built successfully.

梁朝伟演过80部电影!

您好，请输入您要查询的电影相关信息，如李连杰演过什么电影？：卧虎藏龙主要讲什么

Building prefix dict from the default dictionary ...

Loading model from cache C:\Users\12261\AppData\Local\Temp\jieba.cache

Loading model cost 0.682 seconds.

Prefix dict has been built successfully.

match(m:Movie)-[]->() where m.title='卧虎藏龙' return m.introduction

卧虎藏龙主要讲述了一代大侠李慕白有退出江湖之意，托付红颜知己俞秀莲将自己的青冥剑带到京城，作为礼物送给贝勒爷收藏。这把有四百年历史的古剑伤人无数，李慕白希望如此重大决断能够表明他离开江湖恩怨的决心。谁知当天夜里宝剑就被人盗走，俞秀莲上前阻拦与盗剑人交手，但最后盗剑人在同伙的救助下逃走。有人看见一个蒙面人消失在九门提督玉大人府内，俞秀莲也认为玉大人难逃干系。九门提督主管京城治安，玉大人刚从新疆调来赴任，贝勒爷即不相信玉大人与此有关，也不能轻举妄动以免影响大局。 　　俞秀莲为了不将事情复杂化一直在暗中查访宝剑下落，也大约猜出是玉府小姐玉蛟龙一时意气所为。俞秀莲对前来京城的李慕白隐瞒消息，只想用旁敲侧击的方法迫使玉蛟龙归还宝剑，免伤和气。不过俞秀莲的良苦用心落空，蒙面人真的归还宝剑时，不可避免地跟李慕白有了一次正面的交锋。而李慕白又发现了害死师傅的碧眼狐狸的踪迹，此时李慕白更是欲罢不能。 　　玉蛟龙自幼被隐匿于玉府的碧眼狐狸暗中收为弟子，并从秘籍中习得武当派上乘武功，早已青出于蓝。在新疆之时，玉蛟龙就瞒着父亲与当地大盗“半天云”罗小虎情定终身，如今身在北京，父亲又要她嫁人，玉蛟龙一时兴起冲出家门浪迹江湖。 　　任性傲气的玉蛟龙心中凄苦无处发泄，在江湖上使性任气，俨然是个小魔星。俞秀莲和李慕白爱惜玉蛟龙人才难得，苦心引导，总是无效。在最后和碧眼狐狸的交手之中，李慕白为救玉蛟龙身中毒针而死。玉蛟龙在俞秀莲的指点下来到武当山，却无法面对罗小虎，在和罗小虎一夕缠绵之后，投身万丈绝壑。！