# 第8章 基于任务导向的对话系统

通过前面的学习，我们已经掌握了如何从零开始搭建一个完整的问答系统的过程，然而实际生产生活中，为了准确获取用户的真实意图，往往需要人机之间进行多轮次的交互对话，进而获取到必要的信息，得到用户明确的指令。相较于问答系统这种单轮对话所表现出的一问一答的形式，多轮对话需要系统有效利用对话中的上下文信息，并对对话状态做出管理。。

多轮对话一般可以分为开放领域的多轮对话以及特定领域的多轮对话，比如陪伴型的闲聊机器人即属于开放领域的多轮对话，而任务导向的多轮对话即属于限定领域的多轮对话。限定领域的问答系统和限定领域的对话系统的区别主要在于是否维护需要维护一个对话状态，以及是否需要一个决策过程来完成任务。

任务导向型对话系统是指用户带着明确的任务来，希望得到明确满足特定限定条件的信息或者服务，比如订餐、订票或者购买某件商品等，由于用户的需求比较复杂，需要多轮陈述，且用户与系统在交互的过程中也会不断修正完善自己的需求，此外，当用户陈述的信息不够具体或者明确的情况瞎，系统也可以通过询问、澄清或者确认来帮助用户找到满意的回复结果。因此任务导向的多轮对话不仅是一个自然语言理解的问题，还需要在对话过程中，不断根据当前的对话状态下决策下一步应该采取的动作，进而有效快速地帮助用户完成信息获取服务获取的任务。

下面我们将重点介绍基于任务导向的多轮对话系统的架构设计及各个模块主要的功能，同时，我们将介绍几种当前常用的基于任务导向的多轮对话系统的框架。。

本章主要涉及到的知识点有：

* 系统介绍：任务导向多轮对话系统的架构。
* 自然语言理解模块
* 对话状态管理模块
* 自然语言生产模块
* 常用框架Rasa介绍

注意：本章代码地址：xxx

## 8.1 任务导向多轮对话系统介绍

生活中很多人都有过订餐的经历，在订餐的过程中，对方会询问你们一些基本的问题，比如就餐人数、就餐时间、预定的菜品信息、是否有忌口、是否有喜欢的位置等等，通过对这些问题的沟通，对方会明确用户的需求，进而给出满足以上条件的最优解决策略。

任务导向的多轮对话系统要实现的功能和点餐的过程是很相似的，常见的任务导向多轮对话系统的架构如下图所示：

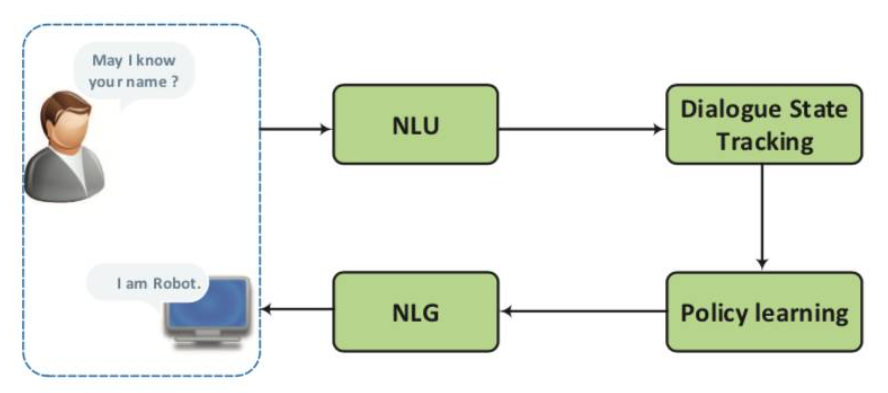


图8.1 任务导向型对话系统框架

在该框架中较为重要的有三部分分别是自然语言理解模块（NLU）、对话状态管理模块（DM）及对话生成模块（NLG）。其中对话状态管理模块一般分为两部分，分别是对话状态追踪（DST）及对话策略管理（PL）。下面我们分别介绍下每个模块的主要功能：

## 8.2 自然语言理解模块

### 8.2.1 领域信息识别

为了构建一个医疗诊断问答系统，我们需要掌握一些医疗知识，这里我们从寻医问药网站采集一些医疗信息。这是一家医疗信息提供平台，上面的医疗信息已经做了很好的分类标注，通过采集其中的公开数据，我们可以获取一些医疗知识，比如药品的信息，疾病的信息，常见的症状表现等。如下图所示，展示了扁桃体发炎的病因及如何检查，一般表现出那些症状以及可能是由于那些疾病所导致的。



图7.4 扁桃体发炎相关医疗知识

数据采集的具体实现代码如下：

注意，在实际项目中，针对不同的需求，可以根据具体需求获取相应的数据，这里的文本预处理仅用于简单演示。

**代码7.1 医疗知识数据采集代码**

#!/usr/bin/env python  
# \_\*\_ coding:utf-8 \_\*\_  
  
import urllib.request  
import urllib.parse  
from lxml import etree  
import pymongo  
import re  
  
  
class MedicalSpider:  
 *'''  
 基于寻医问药网站的医疗知识数据采集  
 '''* def \_\_init\_\_(self):  
 self.conn = pymongo.MongoClient()  
 self.db = self.conn['medical']  
 self.col = self.db['data']  
  
 def get\_html(self, url):  
 *'''  
 根据url地址，请求页面信息* ***:param*** *url:* ***:return****:  
 '''* headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) '  
 'Chrome/51.0.2704.63 Safari/537.36'}  
 req = urllib.request.Request(url=url, headers=headers)  
 res = urllib.request.urlopen(req)  
 html = res.read().decode('gbk')  
 return html  
  
 def url\_parser(self, content):  
 *'''  
 解析页面内容* ***:param*** *content:* ***:return****:  
 '''* selector = etree.HTML(content)  
 urls = ['http://www.anliguan.com' + i for i in selector.xpath('//h2[@class="item-title"]/a/@href')]  
 return urls  
  
 def spider\_main(self):  
 *'''  
 遍历查询网站不同模块的医疗知识* ***:return****:  
 '''* for page in range(1, 11000):  
 try:  
 basic\_url = 'http://jib.xywy.com/il\_sii/gaishu/%s.htm' % page  
 cause\_url = 'http://jib.xywy.com/il\_sii/cause/%s.htm' % page  
 prevent\_url = 'http://jib.xywy.com/il\_sii/prevent/%s.htm' % page  
 symptom\_url = 'http://jib.xywy.com/il\_sii/symptom/%s.htm' % page  
 inspect\_url = 'http://jib.xywy.com/il\_sii/inspect/%s.htm' % page  
 treat\_url = 'http://jib.xywy.com/il\_sii/treat/%s.htm' % page  
 food\_url = 'http://jib.xywy.com/il\_sii/food/%s.htm' % page  
 drug\_url = 'http://jib.xywy.com/il\_sii/drug/%s.htm' % page  
 data = {}  
 data['url'] = basic\_url  
 data['basic\_info'] = self.basicinfo\_spider(basic\_url)  
 data['cause\_info'] = self.common\_spider(cause\_url)  
 data['prevent\_info'] = self.common\_spider(prevent\_url)  
 data['symptom\_info'] = self.symptom\_spider(symptom\_url)  
 data['inspect\_info'] = self.inspect\_spider(inspect\_url)  
 data['treat\_info'] = self.treat\_spider(treat\_url)  
 data['food\_info'] = self.food\_spider(food\_url)  
 data['drug\_info'] = self.drug\_spider(drug\_url)  
 print(page, basic\_url)  
 self.col.insert(data)  
 except Exception as e:  
 print(e, page)  
 return  
  
 def basicinfo\_spider(self, url):  
 *'''  
 爬取基本信息* ***:param*** *url:* ***:return****:  
 '''* html = self.get\_html(url)  
 selector = etree.HTML(html)  
 title = selector.xpath('//title/text()')[0]  
 category = selector.xpath('//div[@class="wrap mt10 nav-bar"]/a/text()')  
 desc = selector.xpath('//div[@class="jib-articl-con jib-lh-articl"]/p/text()')  
 ps = selector.xpath('//div[@class="mt20 articl-know"]/p')  
 infobox = []  
 for p in ps:  
 info = p.xpath('string(.)').replace('\r', '').replace('\n', '').replace('\xa0', '').replace(' ',  
 '').replace(  
 '\t', '')  
 infobox.append(info)  
 basic\_data = {}  
 basic\_data['category'] = category  
 basic\_data['name'] = title.split('的简介')[0]  
 basic\_data['desc'] = desc  
 basic\_data['attributes'] = infobox  
 return basic\_data  
  
 def treat\_spider(self, url):  
 *'''  
 爬取治疗信息* ***:param*** *url:* ***:return****:  
 '''* html = self.get\_html(url)  
 selector = etree.HTML(html)  
 ps = selector.xpath('//div[starts-with(@class,"mt20 articl-know")]/p')  
 infobox = []  
 for p in ps:  
 info = p.xpath('string(.)').replace('\r', '').replace('\n', '').replace('\xa0', '').replace(' ',  
 '').replace(  
 '\t', '')  
 infobox.append(info)  
 return infobox  
  
 def drug\_spider(self, url):  
 *'''  
 爬取药品信息* ***:param*** *url:* ***:return****:  
 '''* html = self.get\_html(url)  
 selector = etree.HTML(html)  
 drugs = [i.replace('\n', '').replace('\t', '').replace(' ', '') for i in  
 selector.xpath('//div[@class="fl drug-pic-rec mr30"]/p/a/text()')]  
 return drugs  
  
 def food\_spider(self, url):  
 *'''  
 爬取忌口信息* ***:param*** *url:* ***:return****:  
 '''* html = self.get\_html(url)  
 selector = etree.HTML(html)  
 divs = selector.xpath('//div[@class="diet-img clearfix mt20"]')  
 try:  
 food\_data = {}  
 food\_data['good'] = divs[0].xpath('./div/p/text()')  
 food\_data['bad'] = divs[1].xpath('./div/p/text()')  
 food\_data['recommand'] = divs[2].xpath('./div/p/text()')  
 except:  
 return {}  
  
 return food\_data  
  
 def symptom\_spider(self, url):  
 *'''  
 爬取症状信息* ***:param*** *url:* ***:return****:  
 '''* html = self.get\_html(url)  
 selector = etree.HTML(html)  
 symptoms = selector.xpath('//a[@class="gre" ]/text()')  
 ps = selector.xpath('//p')  
 detail = []  
 for p in ps:  
 info = p.xpath('string(.)').replace('\r', '').replace('\n', '').replace('\xa0', '').replace(' ',  
 '').replace(  
 '\t', '')  
 detail.append(info)  
 symptoms\_data = {}  
 symptoms\_data['symptoms'] = symptoms  
 symptoms\_data['symptoms\_detail'] = detail  
 return symptoms, detail  
  
 def inspect\_spider(self, url):  
 *'''  
 爬取检查信息* ***:param*** *url:* ***:return****:  
 '''* html = self.get\_html(url)  
 selector = etree.HTML(html)  
 inspects = selector.xpath('//li[@class="check-item"]/a/@href')  
 return inspects  
  
 def common\_spider(self, url):  
 *'''  
 爬取常用信息* ***:param*** *url:* ***:return****:  
 '''* html = self.get\_html(url)  
 selector = etree.HTML(html)  
 ps = selector.xpath('//p')  
 infobox = []  
 for p in ps:  
 info = p.xpath('string(.)').replace('\r', '').replace('\n', '').replace('\xa0', '').replace(' ',  
 '').replace(  
 '\t', '')  
 if info:  
 infobox.append(info)  
 return '\n'.join(infobox)  
  
 def inspect\_crawl(self):  
 *'''  
 启动数据采集* ***:return****:  
 '''* for page in range(1, 3685):  
 try:  
 url = 'http://jck.xywy.com/jc\_%s.html' % page  
 html = self.get\_html(url)  
 data = {}  
 data['url'] = url  
 data['html'] = html  
 self.db['jc'].insert(data)  
 print(url)  
 except Exception as e:  
 print(e)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 handler = MedicalSpider()  
 handler.inspect\_crawl()

最终获取的医疗知识数据保存在json格式的数据文件中，数据格式如下所示：

{ "\_id" : { "$oid" : "5bb578b6831b973a137e3ee7" },

"name" : "百日咳",

"desc" : "百日咳(pertussis，whoopingcough)是由百日咳杆菌所致的急性呼吸道传染病。其特征为阵发性痉挛性咳嗽，咳嗽末伴有特殊的鸡鸣样吸气吼声。病程较长，可达数周甚至3个月左右，故有百日咳之称。多见于5岁以下的小儿，幼婴患本病时易有窒息、肺炎，脑病等并发症，病死率高。百日咳患者，阴性感染者及带菌者为传染源。潜伏期末到病后2-3周传染性最强。百日咳经呼吸道飞沫传播。典型患者病程6-8周，临床病程可分3期：1.卡他期，从发病到开始出现咳嗽，一般1-2周。2,痉咳期，一般2-4周或更长，阵发性痉挛性咳嗽为本期特点。3，恢复期，一般1-2周，咳嗽发作的次数减少，程度减轻，不再出现阵发性痉咳。一般外周血白细胞计数明显增高，分类以淋巴细胞为主。在诊断本病时要注意与支气管异物及肺门淋巴结结核鉴别。近年来幼婴及成人发病有增多趋势。",

"category" : [ "疾病百科", "儿科", "小儿内科" ],

"prevent" : "1、控制传染源：在流行季节，若有前驱症状应及早抗生素治疗。\n2、切断传播途径：由于百日咳杆菌对外界抵抗力较弱，无需消毒处理，但应保持室内通风，衣物在阳光下曝晒，对痰液及口鼻分泌物则应进行消毒处理。",

"cause" : "(一)发病原因\n病原菌是鲍特菌属(Bordetella)中的百日咳鲍特菌(B.pertussis)，常称百日咳杆菌，已知鲍特菌属有四种杆菌，除百日咳鲍特菌外还有副百日咳鲍特菌(B.parapertussis)，支气管败血鲍特菌(B.bronchiseptica)和鸟型鲍特菌(B.avium)，鸟型鲍特菌一般不引起人类致病，仅引起鸟类感染，百日咳杆菌长约1.0～1.5μm，宽约0.3～0.5μm，有荚膜，不能运动，革兰染色阴性，需氧，无芽孢，无鞭毛，用甲苯胺蓝染色两端着色较深，细菌培养需要大量(15%～25%)鲜血才能繁殖良好，故常以鲍-金(Border-Gengous)培养基(即血液，甘油，马铃薯)分离菌落，百日咳杆菌生长缓慢，在35～37℃潮湿的环境中3～7天后，一种细小的，不透明的菌落生长，初次菌落隆起而光滑，为光滑(S)型，又称I相细菌，形态高低一致，有荚膜和较强的毒力及抗原性，致病力强，如将分离菌落在普通培养基中继续培养，菌落由光滑型变为粗糙(R)型，称Ⅳ相细菌，无荚膜，毒力及抗原性丢失，并失去致病力，Ⅱ相，Ⅲ相为中间过渡型，百日咳杆菌能产生许多毒性因子，已知有五种毒素：\n1、百日咳外毒素(PT)：是存在百日咳杆菌细胞壁中一种蛋白质，过去称作为白细胞或淋巴细胞增多促进因子(leukocytosis or lymphocyte promoting factor，LPE)，组胺致敏因子(histamin sensitizing factor，HSF)，胰岛素分泌活性蛋白(insulin activating protein，IAP)，百日咳外毒素由五种非共价链亚单位(S1～S5)所组成，亚单位(S2～S5)为无毒性单位，能与宿主细胞膜结合，通过具有酶活力的亚单位S1介导毒性作用，S1能通过腺苷二磷酸(ADP)-核糖转移酶的活力，催化部分ADP-核糖从烟酰胺腺嘌呤二核苷酸(NAD)中分离出来，转移至细胞膜抑制鸟苷三磷酸(CTP)结合即G蛋白合成，导致细胞变生，同时还能促使淋巴细胞增高，活化胰岛细胞及增强免疫应答。\n2、耐热的内毒素(endotoxin，ET)，100℃60min只能部分破坏，180℃才能灭活，此毒素能引起机体发热及痉咳。\n3、不耐热毒素(HLT)这种毒素加热55℃30min后能破坏其毒性作用，此毒素抗体对百日咳杆菌感染无保护作用。\n4、气管细胞毒素(TCT)：能损害宿主呼吸道纤毛上皮细胞，使之变性，坏死。\n5、腺苷环化酶毒素(ACT)：存在百日咳杆菌细胞表面的一种酶，此酶进入吞噬细胞后被调钙蛋白所激活，催化cAMP的生成，干扰吞噬作用，并抑制中性粒细胞的趋化和吞噬细胞杀菌能力，使其能持续感染，ACT也是一种溶血素，能起溶血作用，百日咳的重要抗原是百日咳菌的两种血凝活性抗原，一种为丝状血凝素(filamentous hemagglutinin，FHA)，因来自菌体表面菌毛故又称菌毛抗原，FHA在百日咳杆菌黏附于呼吸道上皮细胞的过程中起决定作用，为致病的主要原因。实验发现，FHA免疫小鼠能对抗百日咳杆菌致死性攻击，因此FHA为保护性抗原，另一种凝集原(aggluginogens，AGG)为百日咳杆菌外膜及菌毛中的一种蛋白质成分，主要含1，2，3三种血清型凝血因子，AGG-1具有种特异性；AGG-2，3具有型特异性，通过检测凝集原的型别来了解当地流行情况，目前认为这两种血凝素抗原相应抗体是保护性抗体，百日咳杆菌根据不耐热凝集原抗原性不同分为七型凝集原，1型凝集原为所有百日咳杆菌均具备，7型凝集原为鲍特菌属(包括副百日咳杆菌，支气管败血性杆菌)所共有，2～6型以不同的配合将百日咳杆菌分为不同血清型，测定血清型主要是研究流行时菌株的血清型和选择特殊血清型菌株生产菌苗，此外，副百日咳杆菌与百日咳杆菌无交叉免疫，亦可引起流行，百日咳杆菌对外界理化因素抵抗力弱，55℃经30min即被破坏，干燥数小时即可杀灭，对一般消毒剂敏感，对紫外线抵抗力弱，但在0～10℃存活较长。\n(二)发病机制\n1、发病机制：百日咳发病机制不甚清楚，很可能是百日咳毒素对机体综合作用的结果，当细菌随空气飞沫浸入易感者的呼吸道后，细菌的丝状血凝素黏附于咽喉至细支气管黏膜的纤毛上皮细胞表面；继之，细菌在局部繁殖并产生多种毒素如百日咳外毒素，腺苷环化酶等引起上皮细胞纤毛麻痹和细胞变性，使其蛋白合成降低，上皮细胞坏死脱落，以及全身反应，由于上皮细胞的病变发生和纤毛麻痹使小支气管中黏液及坏死上皮堆聚潴留，分泌物排出受阻，不断刺激呼吸道的周围神经，传入大脑皮质及延髓咳嗽中枢，反射性引起痉挛性咳嗽，由于长期刺激使咳嗽中枢形成兴奋灶，以致非特异性刺激，如进食，咽部检查，冷风，烟雾以及注射疼痛等，均可引起反射性的痉咳，恢复期间亦可因哭泣及其他感染，诱发百日咳样痉咳，近来研究表明百日咳发生机制与百日咳杆菌毒素类物质损害宿主细胞免疫功能有关，CD4+T细胞和Th1细胞分泌的细胞因子所介导的免疫反应，在百日咳杆菌感染中起重要作用。\n2、病理解剖：百日咳杆菌侵犯鼻咽，喉，气管，支气管黏膜，可见黏膜充血，上皮细胞的基底部有多核白细胞，单核细胞浸润及部分细胞坏死。支气管及肺泡周围间质除炎症浸润外，可见上皮细胞胞质空泡形成，甚至核膜破裂溶解，坏死，脱落，但极少波及肺泡。若分泌物阻塞可引起肺不张，支气管扩张，有继发感染者，易发生支气管肺炎，有时可有间质性肺炎；若发生百日咳脑病，镜检或肉眼可见脑组织充血水肿，点状出血，皮质萎缩，神经细胞变性，脑水肿等改变，此时常可见到肝脏脂肪浸润等变化。",

"symptom" : [ "吸气时有蝉鸣音", "痉挛性咳嗽", "胸闷", "肺阴虚", "抽搐", "低热", "闫鹏辉", "惊厥" ],

"yibao\_status" : "否",

"get\_prob" : "0.5%",

"easy\_get" : "多见于小儿",

"get\_way" : "呼吸道传播",

"acompany" : [ "肺不张" ],

"cure\_department" : [ "儿科", "小儿内科" ],

"cure\_way" : [ "药物治疗", "支持性治疗" ],

"cure\_lasttime" : "1-2个月",

"cured\_prob" : "98%",

"common\_drug" : [ "穿心莲内酯片", "百咳静糖浆" ],

"cost\_money" : "根据不同医院，收费标准不一致，市三甲医院约（1000-4000元）",

"check" : [ "耳、鼻、咽拭子细菌培养", "周围血白细胞计数及分类检验", "血常规", "酶联免疫吸附试验", "白细胞分类计数" ],

"do\_eat" : [ "南瓜子仁", "圆白菜", "樱桃番茄", "小白菜" ],

"not\_eat" : [ "螃蟹", "海蟹", "海虾", "海螺" ],

"recommand\_eat" : [ "清蒸鸡蛋羹", "百合双耳鸡蛋羹", "排骨汤", "罗汉果雪耳鸡汤", "小黄瓜凉拌面", "黄瓜三丝汤", "黄瓜拌兔丝", "黄瓜拌皮丝" ],

"recommand\_drug" : [ "琥乙红霉素片", "琥乙红霉素颗粒", "百咳静糖浆", "穿心莲内酯片", "红霉素肠溶片", "环酯红霉素片" ],

"drug\_detail" : [ "惠普森穿心莲内酯片(穿心莲内酯片)", "北京同仁堂百咳静糖浆(百咳静糖浆)", "邦琪药业百咳静糖浆(百咳静糖浆)", "东新药业百咳静糖浆(百咳静糖浆)", "达发新(环酯红霉素片)", "康美药业红霉素肠溶片(红霉素肠溶片)", "旺龙药业琥乙红霉素颗粒(琥乙红霉素颗粒)", "白云山医药琥乙红霉素片(琥乙红霉素片)", "国瑞琥乙红霉素片(琥乙红霉素片)", "利君制药红霉素肠溶片(红霉素肠溶片)", "东信药业琥乙红霉素颗粒(琥乙红霉素颗粒)", "石药欧意红霉素肠溶片(红霉素肠溶片)", "平光制药红霉素肠溶片(红霉素肠溶片)", "北京曙光药业红霉素肠溶片(红霉素肠溶片)", "迪瑞制药琥乙红霉素颗粒(琥乙红霉素颗粒)", "永定制药百咳静糖浆(百咳静糖浆)", "东信药业琥乙红霉素片(琥乙红霉素片)", "利君制药琥乙红霉素片(琥乙红霉素片)", "北京中新制药琥乙红霉素片(琥乙红霉素片)", "华南药业红霉素肠溶片(红霉素肠溶片)", "佐今明百咳静糖浆(百咳静糖浆)", "恒益药业琥乙红霉素颗粒(琥乙红霉素颗粒)", "利君沙(琥乙红霉素颗粒)" ] }

注意，由于篇幅的限制，这里仅列举了其中一条医疗知识数据，详细数据可以访问随书附带项目原文件获取。

### 8.2.2 意图信息识别

通过上面获取的医疗知识数据，我们将其整理存储在图数据库中，通过对医疗知识数据的分析整理，我们整理了医疗数据知识图谱的七种实体类型，分别是诊断检查项目、医疗科目、疾病信息、药品信息、食物信息、在售药品信息和疾病症状信息。如下表所示：

表7.1 医疗数据知识图谱实体类型表

|  |  |  |  |
| --- | --- | --- | --- |
| **实体类型** | **中文含义** | **实体数量** | **举例** |
| Check | 诊断检查项目 | 3,353 | 支气管造影;关节镜检查 |
| Department | 医疗科目 | 54 | 整形美容科;烧伤科 |
| Disease | 疾病 | 8,807 | 血栓闭塞性脉管炎;胸降主动脉动脉瘤 |
| Drug | 药品 | 3,828 | 京万红痔疮膏;布林佐胺滴眼液 |
| Food | 食物 | 4,870 | 番茄冲菜牛肉丸汤;竹笋炖羊肉 |
| Producer | 在售药品 | 17,201 | 通药制药青霉素V钾片;青阳醋酸地塞米松片 |
| Symptom | 疾病症状 | 5,998 | 乳腺组织肥厚;脑实质深部出血 |
| Total | 总计 | 44,111 | 约4.4万实体量级 |

同时，在上面的实体类型之间，我们提取了十种实体关系，具体如下表所示：

表7.2 医疗数据知识图谱实体关系类型表

|  |  |  |  |
| --- | --- | --- | --- |
| **实体关系类型** | **中文含义** | **关系数量** | **举例** |
| belongs\_to | 属于 | 8,844 | <妇科,属于,妇产科> |
| common\_drug | 疾病常用药品 | 14,649 | <阳强,常用,甲磺酸酚妥拉明分散片> |
| do\_eat | 疾病宜吃食物 | 22,238 | <胸椎骨折,宜吃,黑鱼> |
| drugs\_of | 药品在售药品 | 17,315 | <青霉素V钾片,在售,通药制药青霉素V钾片> |
| need\_check | 疾病所需检查 | 39,422 | <单侧肺气肿,所需检查,支气管造影> |
| no\_eat | 疾病忌吃食物 | 22,247 | <唇病,忌吃,杏仁> |
| recommand\_drug | 疾病推荐药品 | 59,467 | <混合痔,推荐用药,京万红痔疮膏> |
| recommand\_eat | 疾病推荐食谱 | 40,221 | <鞘膜积液,推荐食谱,番茄冲菜牛肉丸汤> |
| has\_symptom | 疾病症状 | 5,998 | <早期乳腺癌,疾病症状,乳腺组织肥厚> |
| acompany\_with | 疾病并发疾病 | 12,029 | <下肢交通静脉瓣膜关闭不全,并发疾病,血栓闭塞性脉管炎> |
| Total | 总计 | 294,149 | 约30万关系量级 |

同时，从实体类型中获取了八种属性信息，分别是疾病名称、疾病简介、疾病病因、预防措施、治疗周期、治疗方式、治愈概率及疾病易感人群。如下表所示：

表7.3 医疗数据知识图谱实体属性表

|  |  |  |
| --- | --- | --- |
| **属性类型** | **中文含义** | **举例** |
| name | 疾病名称 | 喘息样支气管炎 |
| desc | 疾病简介 | 又称哮喘性支气管炎... |
| cause | 疾病病因 | 常见的有合胞病毒等... |
| prevent | 预防措施 | 注意家族与患儿自身过敏史... |
| cure\_lasttime | 治疗周期 | 6-12个月 |
| cure\_way | 治疗方式 | "药物治疗","支持性治疗" |
| cured\_prob | 治愈概率 | 95% |
| easy\_get | 疾病易感人群 | 无特定的人群 |

确定医疗知识图谱的结构后，我们将上面获取的数据导入Neo4j图数据库中，导入代码如下所示：

注意，导入的过程需要消耗较长的时间。

**代码7.2 构建医疗知识图谱**

#!/usr/bin/env python  
# \_\*\_ coding:utf-8 \_\*\_  
  
  
import os  
import json  
from py2neo import Graph, Node  
  
  
class MedicalGraph:  
 def \_\_init\_\_(self):  
 # 确定当前工作路径  
 cur\_dir = '/'.join(os.path.abspath(\_\_file\_\_).split('/')[:-1])  
 # 数据存储路径  
 self.data\_path = os.path.join(cur\_dir, 'data/medical.json')  
 # 连接图数据库  
 self.g = Graph("bolt://localhost:11008")  
  
 def read\_nodes(self):  
 *'''  
 读取数据文件，获取数据中的节点及节点间实体关系* ***:return****:  
 '''* # 数据库中包含7类节点  
 drugs = [] # 药品  
 foods = [] # 食物  
 checks = [] # 检查  
 departments = [] # 科室  
 producers = [] # 药品大类  
 diseases = [] # 疾病  
 symptoms = [] # 症状  
 disease\_infos = [] # 疾病信息  
  
 # 构建节点实体关系  
 rels\_department = [] # 科室－科室关系  
 rels\_noteat = [] # 疾病－忌吃食物关系  
 rels\_doeat = [] # 疾病－宜吃食物关系  
 rels\_recommandeat = [] # 疾病－推荐吃食物关系  
 rels\_commonddrug = [] # 疾病－通用药品关系  
 rels\_recommanddrug = [] # 疾病－热门药品关系  
 rels\_check = [] # 疾病－检查关系  
 rels\_drug\_producer = [] # 厂商－药物关系  
 rels\_symptom = [] # 疾病症状关系  
 rels\_acompany = [] # 疾病并发关系  
 rels\_category = [] # 疾病与科室之间的关系  
  
 count = 0  
 # 遍历医疗数据将其保存在字典中  
 for data in open(self.data\_path, encoding='utf-8'):  
 disease\_dict = {}  
 count += 1  
 print(count)  
 data\_json = json.loads(data)  
 disease = data\_json['name']  
 disease\_dict['name'] = disease  
 diseases.append(disease)  
 disease\_dict['desc'] = ''  
 disease\_dict['prevent'] = ''  
 disease\_dict['cause'] = ''  
 disease\_dict['easy\_get'] = ''  
 disease\_dict['cure\_department'] = ''  
 disease\_dict['cure\_way'] = ''  
 disease\_dict['cure\_lasttime'] = ''  
 disease\_dict['symptom'] = ''  
 disease\_dict['cured\_prob'] = ''  
  
 if 'symptom' in data\_json:  
 symptoms += data\_json['symptom']  
 for symptom in data\_json['symptom']:  
 rels\_symptom.append([disease, symptom])  
  
 if 'acompany' in data\_json:  
 for acompany in data\_json['acompany']:  
 rels\_acompany.append([disease, acompany])  
  
 if 'desc' in data\_json:  
 disease\_dict['desc'] = data\_json['desc']  
  
 if 'prevent' in data\_json:  
 disease\_dict['prevent'] = data\_json['prevent']  
  
 if 'cause' in data\_json:  
 disease\_dict['cause'] = data\_json['cause']  
  
 if 'get\_prob' in data\_json:  
 disease\_dict['get\_prob'] = data\_json['get\_prob']  
  
 if 'easy\_get' in data\_json:  
 disease\_dict['easy\_get'] = data\_json['easy\_get']  
  
 if 'cure\_department' in data\_json:  
 cure\_department = data\_json['cure\_department']  
 if len(cure\_department) == 1:  
 rels\_category.append([disease, cure\_department[0]])  
 if len(cure\_department) == 2:  
 big = cure\_department[0]  
 small = cure\_department[1]  
 rels\_department.append([small, big])  
 rels\_category.append([disease, small])  
  
 disease\_dict['cure\_department'] = cure\_department  
 departments += cure\_department  
  
 if 'cure\_way' in data\_json:  
 disease\_dict['cure\_way'] = data\_json['cure\_way']  
  
 if 'cure\_lasttime' in data\_json:  
 disease\_dict['cure\_lasttime'] = data\_json['cure\_lasttime']  
  
 if 'cured\_prob' in data\_json:  
 disease\_dict['cured\_prob'] = data\_json['cured\_prob']  
  
 if 'common\_drug' in data\_json:  
 common\_drug = data\_json['common\_drug']  
 for drug in common\_drug:  
 rels\_commonddrug.append([disease, drug])  
 drugs += common\_drug  
  
 if 'recommand\_drug' in data\_json:  
 recommand\_drug = data\_json['recommand\_drug']  
 drugs += recommand\_drug  
 for drug in recommand\_drug:  
 rels\_recommanddrug.append([disease, drug])  
  
 if 'not\_eat' in data\_json:  
 not\_eat = data\_json['not\_eat']  
 for \_not in not\_eat:  
 rels\_noteat.append([disease, \_not])  
  
 foods += not\_eat  
 do\_eat = data\_json['do\_eat']  
 for \_do in do\_eat:  
 rels\_doeat.append([disease, \_do])  
  
 foods += do\_eat  
 recommand\_eat = data\_json['recommand\_eat']  
  
 for \_recommand in recommand\_eat:  
 rels\_recommandeat.append([disease, \_recommand])  
 foods += recommand\_eat  
  
 if 'check' in data\_json:  
 check = data\_json['check']  
 for \_check in check:  
 rels\_check.append([disease, \_check])  
 checks += check  
 if 'drug\_detail' in data\_json:  
 drug\_detail = data\_json['drug\_detail']  
 producer = [i.split('(')[0] for i in drug\_detail]  
 rels\_drug\_producer += [[i.split('(')[0], i.split('(')[-1].replace(')', '')] for i in drug\_detail]  
 producers += producer  
 disease\_infos.append(disease\_dict)  
 return set(drugs), set(foods), set(checks), set(departments), set(producers), set(symptoms), set(  
 diseases), disease\_infos, \  
 rels\_check, rels\_recommandeat, rels\_noteat, rels\_doeat, rels\_department, rels\_commonddrug, rels\_drug\_producer, rels\_recommanddrug, \  
 rels\_symptom, rels\_acompany, rels\_category  
  
 def create\_node(self, label, nodes):  
 *'''  
 创建知识图谱中节点* ***:param*** *label:* ***:param*** *nodes:* ***:return****:  
 '''* count = 0  
 # 遍历节点信息  
 for node\_name in nodes:  
 # 使用节点对象Node来创建节点  
 node = Node(label, name=node\_name)  
 self.g.create(node)  
 count += 1  
 print(count, len(nodes))  
 return  
  
 def create\_diseases\_nodes(self, disease\_infos):  
 *'''  
 创建知识图谱中疾病的节点* ***:param*** *disease\_infos:* ***:return****:  
 '''* count = 0  
 for disease\_dict in disease\_infos:  
 node = Node("Disease", name=disease\_dict['name'], desc=disease\_dict['desc'],  
 prevent=disease\_dict['prevent'], cause=disease\_dict['cause'],  
 easy\_get=disease\_dict['easy\_get'], cure\_lasttime=disease\_dict['cure\_lasttime'],  
 cure\_department=disease\_dict['cure\_department']  
 , cure\_way=disease\_dict['cure\_way'], cured\_prob=disease\_dict['cured\_prob'])  
 self.g.create(node)  
 count += 1  
 print(count)  
 return  
  
 def create\_graphnodes(self):  
 *'''  
 创建知识图谱实体节点类型schema* ***:return****:  
 '''* Drugs, Foods, Checks, Departments, Producers, Symptoms, Diseases, disease\_infos, rels\_check, rels\_recommandeat, rels\_noteat, rels\_doeat, rels\_department, rels\_commonddrug, rels\_drug\_producer, rels\_recommanddrug, rels\_symptom, rels\_acompany, rels\_category = self.read\_nodes()  
 self.create\_diseases\_nodes(disease\_infos)  
 self.create\_node('Drug', Drugs)  
 print(len(Drugs))  
 self.create\_node('Food', Foods)  
 print(len(Foods))  
 self.create\_node('Check', Checks)  
 print(len(Checks))  
 self.create\_node('Department', Departments)  
 print(len(Departments))  
 self.create\_node('Producer', Producers)  
 print(len(Producers))  
 self.create\_node('Symptom', Symptoms)  
 return  
  
  
 def create\_graphrels(self):  
 *'''  
 创建知识图谱实体关系边* ***:return****:  
 '''* Drugs, Foods, Checks, Departments, Producers, Symptoms, Diseases, disease\_infos, rels\_check, rels\_recommandeat, rels\_noteat, rels\_doeat, rels\_department, rels\_commonddrug, rels\_drug\_producer, rels\_recommanddrug, rels\_symptom, rels\_acompany, rels\_category = self.read\_nodes()  
 self.create\_relationship('Disease', 'Food', rels\_recommandeat, 'recommand\_eat', '推荐食谱')  
 self.create\_relationship('Disease', 'Food', rels\_noteat, 'no\_eat', '忌吃')  
 self.create\_relationship('Disease', 'Food', rels\_doeat, 'do\_eat', '宜吃')  
 self.create\_relationship('Department', 'Department', rels\_department, 'belongs\_to', '属于')  
 self.create\_relationship('Disease', 'Drug', rels\_commonddrug, 'common\_drug', '常用药品')  
 self.create\_relationship('Producer', 'Drug', rels\_drug\_producer, 'drugs\_of', '生产药品')  
 self.create\_relationship('Disease', 'Drug', rels\_recommanddrug, 'recommand\_drug', '好评药品')  
 self.create\_relationship('Disease', 'Check', rels\_check, 'need\_check', '诊断检查')  
 self.create\_relationship('Disease', 'Symptom', rels\_symptom, 'has\_symptom', '症状')  
 self.create\_relationship('Disease', 'Disease', rels\_acompany, 'acompany\_with', '并发症')  
 self.create\_relationship('Disease', 'Department', rels\_category, 'belongs\_to', '所属科室')  
  
 def create\_relationship(self, start\_node, end\_node, edges, rel\_type, rel\_name):  
 *'''  
 创建实体关联边* ***:param*** *start\_node:* ***:param*** *end\_node:* ***:param*** *edges:* ***:param*** *rel\_type:* ***:param*** *rel\_name:* ***:return****:  
 '''* count = 0  
 # 去重处理  
 set\_edges = []  
 for edge in edges:  
 set\_edges.append('###'.join(edge))  
 all = len(set(set\_edges))  
 for edge in set(set\_edges):  
 edge = edge.split('###')  
 p = edge[0]  
 q = edge[1]  
 query = "match(p:%s),(q:%s) where p.name='%s'and q.name='%s' create (p)-[rel:%s{name:'%s'}]->(q)" % (  
 start\_node, end\_node, p, q, rel\_type, rel\_name)  
 try:  
 self.g.run(query)  
 count += 1  
 print(rel\_type, count, all)  
 except Exception as e:  
 print(e)  
 return  
  
 def export\_data(self):  
 *'''  
 导出并保存数据* ***:return****:  
 '''* Drugs, Foods, Checks, Departments, Producers, Symptoms, Diseases, disease\_infos, rels\_check, rels\_recommandeat, rels\_noteat, rels\_doeat, rels\_department, rels\_commonddrug, rels\_drug\_producer, rels\_recommanddrug, rels\_symptom, rels\_acompany, rels\_category = self.read\_nodes()  
 f\_drug = open('drug.txt', 'w+')  
 f\_food = open('food.txt', 'w+')  
 f\_check = open('check.txt', 'w+')  
 f\_department = open('department.txt', 'w+')  
 f\_producer = open('producer.txt', 'w+')  
 f\_symptom = open('symptoms.txt', 'w+')  
 f\_disease = open('disease.txt', 'w+')  
  
 f\_drug.write('\n'.join(list(Drugs)))  
 f\_food.write('\n'.join(list(Foods)))  
 f\_check.write('\n'.join(list(Checks)))  
 f\_department.write('\n'.join(list(Departments)))  
 f\_producer.write('\n'.join(list(Producers)))  
 f\_symptom.write('\n'.join(list(Symptoms)))  
 f\_disease.write('\n'.join(list(Diseases)))  
  
 f\_drug.close()  
 f\_food.close()  
 f\_check.close()  
 f\_department.close()  
 f\_producer.close()  
 f\_symptom.close()  
 f\_disease.close()  
 return  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # 初始化医疗数据知识图谱  
 handler = MedicalGraph()  
 # 创建知识图谱节点信息  
 handler.create\_graphnodes()  
 # 创建知识图谱关系信息  
 handler.create\_graphrels()  
 # 导出数据并保存  
 handler.export\_data()

### 8.2.3 槽位信息抽取

导入后的医疗知识数据图谱效果如下图所示：

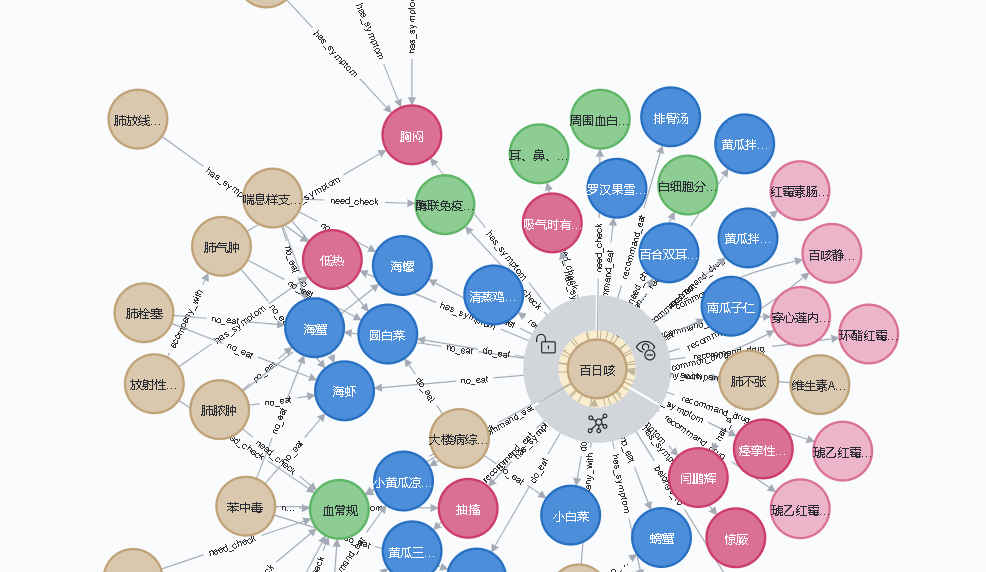


图7.5 医疗知识数据知识图谱效果展示

通过上面的医疗知识图谱可以发现，与疾病“百日咳”相关的症状有“胸闷”、“抽搐”、“惊厥”等，其治疗所用的常用药物为"穿心莲内酯片"、 "百咳静糖浆"等信息。

## 8.3 对话状态管理模块

构建好医疗知识图谱后，接下来就是基于该医疗知识图谱搭建我们的医疗诊断问答系统，该问答系统主要分为两大模块，首先对于用户输入的问题文本进行理解，抽取其中的关键信息，理解用户想要查询的信息，将其转换为知识图谱数据库的查询语句，然后通过访问医疗知识图谱数据库获取其查询信息，并将查询结果按照问题类型选用不同的模板，构造相应的答案文本，最终返回给用户，下面详细介绍一下这两个模块的实现方式。

### 7.3.1 用户意图理解模块

在上一章节的电影知识问答系统中，我们对于用户输入的问题文本抽取其中的关键信息，只是简单使用分词器来完成，实际工业生产中，为了抽取文本中的语义信息，我们需要对文本进行更加深入精细的分析，首先要识别用户输入的问题文本中的医疗实体信息，这部分的工作主要通过命名实体任务来完成，也就是NER（Named Entity Recognition），命名实体识别任务主要用于识别出待处理文本中实体信息，如实体类、时间类和数字类以及人名、机构名、地名、时间、日期、货币和百分比等。  
  比如通过对句子“小明早上8点去学校上课。”的命名实体分析，我们可以提取到以下信息：

人名：小明，时间：早上8点，地点：学校。

这里我们采用深度学习的方法来完成医疗实体识别，通过BiLSTM+CRF网络结构来实现。同时为了生成用于训练深度网络的大规模数据，我们采用了自然语言处理的数据增强技术，通过将采集到的实体信息随机填充到预先设置的问题文本模板的槽中，进而对问题文本进行命名实体识别训练，其中对问题文本的命名实体的标注采用通用的BIOES标注法，其中各个实体及其标注表示如下表所示：

表7.4 医疗知识命名实体标注标签表

|  |
| --- |
|  |
| **实体** | **序号** | **含义** |
| **O** | **0** | **其它** |
| **B-dis** | **1** | **疾病实体开头** |
| **I-dis** | **2** | **疾病实体中间** |
| **E-dis** | **3** | **疾病实体末尾** |
| **B-sym** | **4** | **症状** |
| **I-sym** | **5** |  |
| **E-sym** | **6** |  |
| **B-dru** | **7** | **药品** |
| **I-dru** | **8** |  |
| **E-dru** | **9** |  |
| **S-dis** | **10** | **单个-疾病实体** |
| **S-sym** | **11** |  |
| **S-dru** | **12** |  |

同时对于问题语句的分析，我们会同时对其进行问题类型的分类，其中问题类别的信息如下表所示：

表7.5 问题语句类别表

|  |
| --- |
|  |
| **类别** | **序号** | **含义** |
| disease\_symptom | 0 | 疾病有啥症状 |
| symptom\_curway | 1 | 症状有啥治疗方法 |
| symptom\_disease | 2 | 症状对应啥疾病 |
| disease\_drug | 3 | 疾病要吃啥药品 |
| drug\_disease | 4 | 药品治疗啥疾病 |
| disease\_check | 5 | 疾病要做啥检查检查 |
| disease\_prevent | 6 | 疾病有啥预防方式 |

对问题文本的分析主要包括识别问题中的医疗实体信息及对问题文本进行分类，判定该问题文本意图询问的为那种问题类型，便于我们获取问题查询的条件，进而根据问题去医疗知识图谱中查找相应的信息，问题文本分析的具体代码如下：

**代码7.3 问题文本分析模块代码**

#!/usr/bin/env python  
# \_\*\_ coding:utf-8 \_\*\_  
  
import tensorflow as tf  
from classifyApp import classifyApplication  
from nerApp import nerAppication  
import os  
  
# 屏蔽通知信息、警告信息和报错信息  
os.environ["TF\_CPP\_MIN\_LOG\_LEVEL"] = "3"  
  
# 使用allow\_growth option，刚一开始分配少量的GPU容量，然后按需慢慢的增加  
allow\_growth = True  
# 是否打印设备分配日志  
log\_device\_placement = True  
# 如果你指定的设备不存在，允许TF自动分配设备  
allow\_soft\_placement = True  
# 每个进程占用30%显存  
gpu\_options = tf.GPUOptions(per\_process\_gpu\_memory\_fraction=0.3)  
# 配置tf.ConfigProto  
session\_conf = tf.ConfigProto(gpu\_options=gpu\_options, allow\_soft\_placement=allow\_soft\_placement,  
 log\_device\_placement=log\_device\_placement)  
  
  
class question\_ays:  
 *'''  
 问题文本分析类  
 '''* def \_\_init\_\_(self, device='/cpu:0'):  
 # 为每个类(实例)单独创建一个计算图  
 self.g1 = tf.Graph()  
 self.g2 = tf.Graph()  
 self.device = device  
 self.id2state = {0: 'O',  
 1: 'B-dis', 2: 'I-dis', 3: 'E-dis',  
 4: 'B-sym', 5: 'I-sym', 6: 'E-sym',  
 7: 'B-dru', 8: 'I-dru', 9: 'E-dru',  
 10: 'S-dis', 11: 'S-sym', 12: 'S-dru'}  
 # 命名实体识别采用g1计算图  
 self.sess\_ner = tf.Session(graph=self.g1, config=session\_conf)  
 # 分类采用g2计算图  
 self.sess\_classify = tf.Session(graph=self.g2, config=session\_conf)  
  
 self.classifyApp = classifyApplication(self.sess\_classify, device)  
  
 self.nerApp = nerAppication(self.sess\_ner, device)  
  
 self.state2entityType = {'dis': 'disease', 'sym': 'symptom', 'dru': 'drug'}  
 self.label2id = {"disease\_symptom": 0, "symptom\_curway": 1, "symptom\_disease": 2, "disease\_drug": 3,  
 "drug\_disease": 4, "disease\_check": 5, "disease\_prevent": 6,  
 "disease\_lasttime": 7, "disease\_cureway": 8}  
 self.id2label = {0: "disease\_symptom", 1: "symptom\_curway", 2: "symptom\_disease", 3: "disease\_drug",  
 4: "drug\_disease", 5: "disease\_check", 6: "disease\_prevent",  
 7: "disease\_lasttime", 8: "disease\_cureway"}  
  
 def analysis(self, text):  
 res = {}  
 args = {}  
 question\_types = []  
 data\_line, lable\_line, efficient\_sequence\_length = self.nerApp.questionNer(self.sess\_ner, text)  
 for idx in range(len(data\_line)):  
 middle\_question = []  
 \_entity = ''  
 for each in range(efficient\_sequence\_length[idx]):  
 middle\_question.append(data\_line[idx][each])  
 \_entityType = self.id2state[int(lable\_line[idx][each])]  
 if \_entityType[0] == 'B' or \_entityType[0] == 'I':  
 \_entity += data\_line[idx][each]  
 elif \_entityType[0] == 'E' or \_entityType[0] == 'S':  
 \_entity += data\_line[idx][each]  
 \_entityType\_short = \_entityType[-3:]  
 middle\_question.append(self.state2entityType[\_entityType\_short])  
 if \_entity not in args:  
 args.setdefault(\_entity, [self.state2entityType[\_entityType\_short]])  
 else:  
 args[\_entity].append(self.state2entityType[\_entityType\_short])  
 \_entity = ''  
 else:  
 \_entity = ''  
 question\_text = ''.join(middle\_question)  
 \_classify\_idx = self.classifyApp.questionClassify(self.sess\_classify, question\_text)  
 \_classify\_label = self.id2label[\_classify\_idx[0]]  
 question\_types.append(\_classify\_label)  
 res['args'] = args  
 res['question\_types'] = question\_types  
 return res  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 ques = question\_ays()  
 text = "我发烧流鼻涕应该怎么治疗"  
 while (text != "" and text != " "):  
 # text = input("请描述您的问题：")  
 if text == "quit" or text == "" or text == " ":  
 break  
 else:  
 res = ques.analysis(text)  
 print(res)  
 break

在问题文本分析中，对于医疗实体的识别部分，这里采用双向LSTM与CRF结合的方式来对问题文本完成命名实体识别，该深度网络模型结构如下如所示：

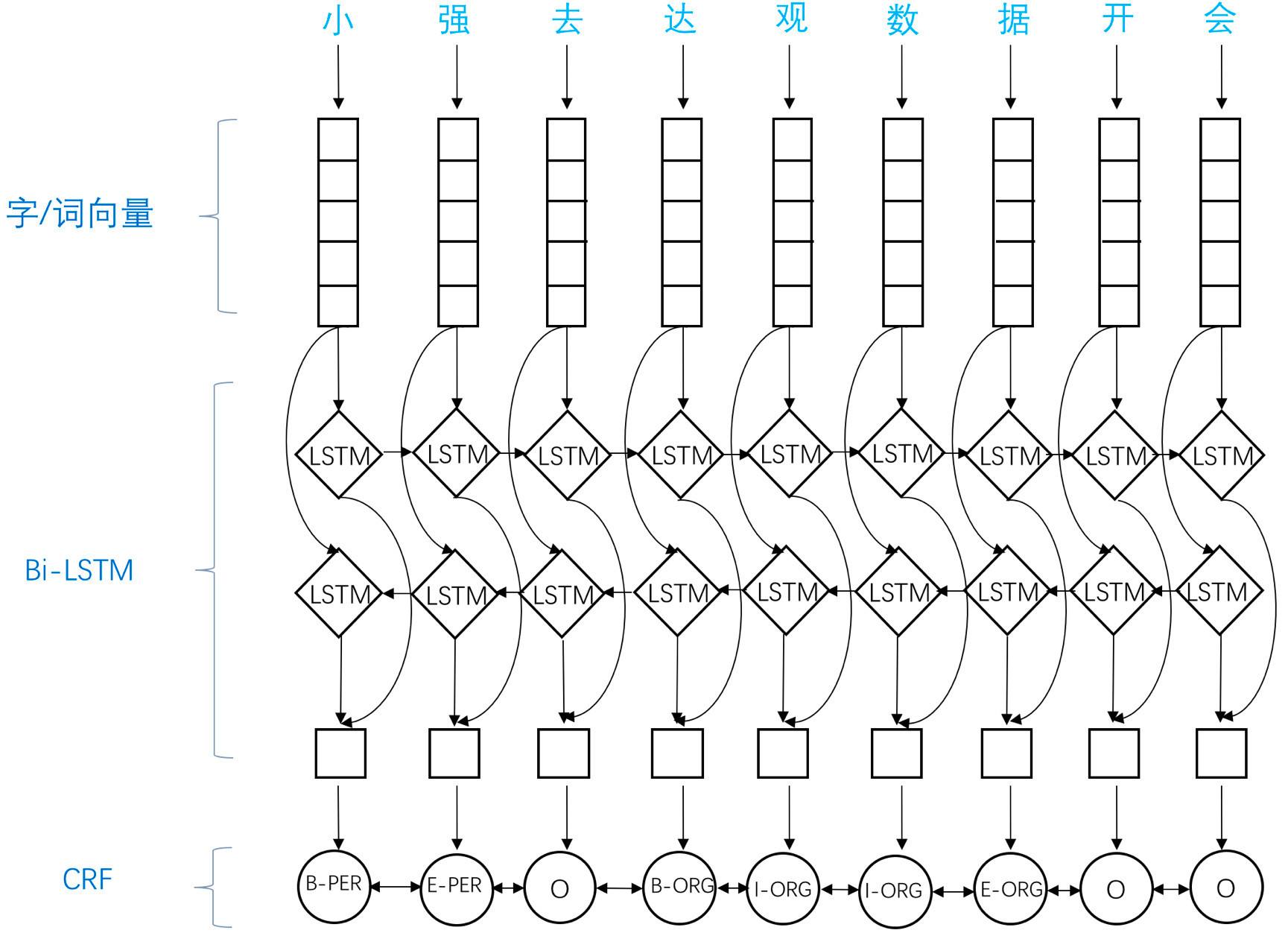


图7.6 Bi-LSTM+CRF模型

注意：限于篇幅限制，这里不再具体展开相关内容，感兴趣的朋友可以查询相关内容进行学习。

通过调用离线训练的NER模型，对输入的问题文本进行医疗实体识别，如对“我感冒了”的识别结果如下“我:0 感冒:10 了:0 ”-，可以发现经过实体识别，可以准确识别问题文本中的“感冒”属于实体“疾病名称”，具体代码如下：

**代码7.4 医疗实体识别代码**

#!/usr/bin/env python  
# \_\*\_ coding:utf-8 \_\*\_  
  
import tensorflow as tf  
from tensorflow.contrib import crf  
import random  
from nerUtils import \*  
import logging  
import datetime  
from BiLSTM\_CRF import BiLSTM\_CRF  
  
debug = False  
batch\_size = 100  
  
  
class nerAppication:  
 # 参数  
 def \_\_init\_\_(self, sess, device='/gpu:1'):  
 with sess.as\_default():  
 with sess.graph.as\_default():  
 self.dataGen = DATAPROCESS(train\_data\_path="./data\_ai/nerData/train\_cutword\_data.txt",  
 train\_label\_path="./data\_ai/nerData/label\_cutword\_data.txt",  
 test\_data\_path="./data\_ai/nerData/test\_data.txt",  
 test\_label\_path="./data\_ai/nerData/test\_label.txt",  
 word\_embedings\_path="./data\_ai/cbowData/document.txt.ebd.npy",  
 vocb\_path="./data\_ai/cbowData/document.txt.vab",  
 batch\_size=batch\_size  
 )  
 self.dataGen.load\_wordebedding()  
 self.tag\_nums = 13 # 标签数目  
 self.hidden\_nums = 650 # bi-lstm的隐藏层单元数目  
 self.sentence\_len = self.dataGen.sentence\_length # 句子长度,输入到网络的序列长度  
 self.model\_checkpoint\_path = "./data\_ai/nerModel/"  
 self.model = BiLSTM\_CRF(  
 batch\_size=batch\_size,  
 tag\_nums=self.tag\_nums,  
 hidden\_nums=self.hidden\_nums,  
 sentence\_len=self.sentence\_len,  
 word\_embeddings=self.dataGen.word\_embeddings,  
 device=device  
 )  
 self.saver = tf.train.Saver(max\_to\_keep=1)  
 ckpt = tf.train.get\_checkpoint\_state(self.model\_checkpoint\_path)  
 if ckpt and ckpt.model\_checkpoint\_path:  
 self.saver.restore(sess, ckpt.model\_checkpoint\_path)  
 logging.info("model loading successful")  
  
 def nerApp(self, sess):  
 with sess.as\_default():  
 with sess.graph.as\_default():  
 text = "application"  
 while (text != "" and text != " "):  
 text = input("请输入一句话：")  
 if text == "quit" or text == "" or text == " ": break  
 data\_line, data\_x, efficient\_sequence\_length = self.dataGen.handleInputData(text)  
 if debug:  
 print(np.array(data\_x).shape)  
 print(data\_x)  
 print(np.array(efficient\_sequence\_length).shape)  
 feed\_dict = {self.model.input\_x: data\_x,  
 self.model.sequence\_lengths: efficient\_sequence\_length,  
 self.model.dropout\_keep\_prob: 1  
 }  
 predict\_labels = sess.run([self.model.crf\_labels],  
 feed\_dict) # predict\_labels是三维的[1,1,25]，第1维包含了一个矩阵  
 lable\_line = []  
 if debug:  
 print(type(predict\_labels))  
 print(predict\_labels)  
 print(np.array(predict\_labels).shape)  
 for idx in range(len(predict\_labels[0])):  
 \_label = predict\_labels[0][idx].reshape(1, -1)  
 lable\_line.append(list(\_label[0]))  
 for idx in range(len(data\_line)):  
 for each in range(efficient\_sequence\_length[idx]):  
 print("%s:%s" % (data\_line[idx][each], lable\_line[idx][each]), end=" ")  
 print('\n')  
  
 def questionNer(self, sess, text):  
 with sess.as\_default():  
 with sess.graph.as\_default():  
 if text == " ":  
 print("文本为空，错误")  
 return  
 data\_line, data\_x, efficient\_sequence\_length = self.dataGen.handleInputData(text)  
  
 feed\_dict = {self.model.input\_x: data\_x,  
 self.model.sequence\_lengths: efficient\_sequence\_length,  
 self.model.dropout\_keep\_prob: 1}  
 predict\_labels = sess.run([self.model.crf\_labels], feed\_dict) # predict\_labels是三维的[1,1,25]，第1维包含了一个矩阵  
 lable\_line = []  
 for idx in range(len(predict\_labels[0])):  
 \_label = predict\_labels[0][idx].reshape(1, -1)  
 lable\_line.append(list(\_label[0]))  
 return data\_line, lable\_line, efficient\_sequence\_length  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
  
 graph = tf.Graph()  
 log\_device\_placement = True # 是否打印设备分配日志  
 allow\_soft\_placement = True # 如果你指定的设备不存在，允许TF自动分配设备  
 gpu\_options = tf.GPUOptions(per\_process\_gpu\_memory\_fraction=0.3)  
 session\_conf = tf.ConfigProto(gpu\_options=gpu\_options, allow\_soft\_placement=allow\_soft\_placement,  
 log\_device\_placement=log\_device\_placement)  
  
 sess = tf.Session(graph=graph, config=session\_conf)  
 app = nerAppication(sess)  
  
 text = "我发烧流鼻涕怎么办"  
 while (text != "" and text != " "):  
 text = input("请输入一句话：")  
 if text == "quit" or text == "" or text == " ": break  
 data\_line, lable\_line, efficient\_sequence\_length = app.questionNer(sess, text)  
 for idx in range(len(data\_line)):  
 for each in range(efficient\_sequence\_length[idx]):  
 print("%s:%s" % (data\_line[idx][each], lable\_line[idx][each]), end=" ")  
 print('\n')

问题分析的另一部分为对问题文本进行分类，判定用户意图查询的信息类型，这里我们采用常用的文本分类模型textCNN对问题文本进行分类，选用该模型的原因在于，我们的问题文本的长度往往较短，并且结合前面识别的医疗实体，可以很好的理解用户的意图。

对问题文本的分类首先要将问题文本表示为向量，这里采用CBOW的方式来训练词向量，通过词向量将问题文本表示为向量，然后通过textCNN分类模型进行分类，其中textCNN网络结构如下图所示：

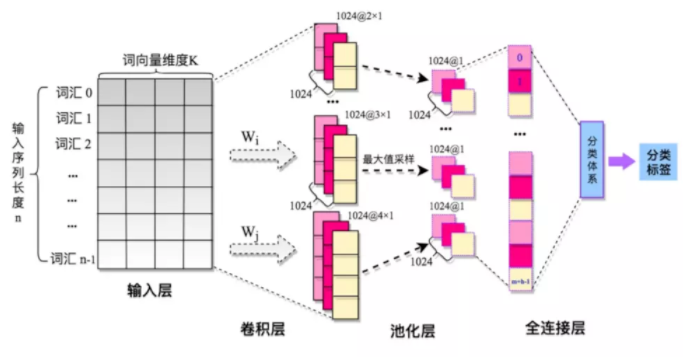


图7.7 textCNN模型

具体代码如下：

**代码7.5问题文本分类**

#!/usr/bin/env python  
# \_\*\_ coding:utf-8 \_\*\_  
  
import tensorflow as tf  
import numpy as np  
import os  
import time  
import datetime  
from classifyUtils import data\_process  
from text\_cnn import TextCNN  
import math  
from tensorflow.contrib import learn  
import jieba  
  
  
# tf.reset\_default\_graph()  
  
class classifyApplication:  
 def \_\_init\_\_(self, sess, device='/gpu:1'):  
 with sess.as\_default():  
 with sess.graph.as\_default():  
 self.word\_embedings\_path = "./data\_ai/cbowData/classifyDocument.txt.ebd.npy"  
 self.vocb\_path = "./data\_ai/cbowData/classifyDocument.txt.vab"  
 self.model\_path = "./data\_ai/classifyModel"  
 self.num\_classes = 9  
 self.max\_sentence\_len = 20  
 self.embedding\_dim = 200  
 self.filter\_sizes = "2,3,4"  
 self.dropout\_keep\_prob = 1.0  
 self.l2\_reg\_lambda = 0.0  
 self.num\_filters = 128  
 self.num\_checkpoints = 1  
  
 self.data\_helpers = data\_process(  
 train\_data\_path="",  
 word\_embedings\_path=self.word\_embedings\_path,  
 vocb\_path=self.vocb\_path,  
 num\_classes=self.num\_classes,  
 max\_document\_length=self.max\_sentence\_len)  
 self.data\_helpers.load\_wordebedding()  
 self.cnn = TextCNN(  
 w2v\_model=self.data\_helpers.word\_embeddings,  
 sequence\_length=self.max\_sentence\_len,  
 num\_classes=self.num\_classes,  
 embedding\_size=self.embedding\_dim,  
 filter\_sizes=list(map(int, self.filter\_sizes.split(","))),  
 num\_filters=self.num\_filters,  
 l2\_reg\_lambda=self.l2\_reg\_lambda,  
 device=device  
 )  
 self.saver = tf.train.Saver(max\_to\_keep=self.num\_checkpoints)  
 ckpt = tf.train.get\_checkpoint\_state(self.model\_path)  
 if ckpt and ckpt.model\_checkpoint\_path:  
 self.saver.restore(sess, ckpt.model\_checkpoint\_path)  
 print("restore from history model.")  
 else:  
 print("there is no classify model.")  
  
 def classifyApp(self, sess):  
 with sess.as\_default():  
 with sess.graph.as\_default():  
 text = "application"  
 while (text != "" and text != " "):  
 text = input("请输入一句话：")  
 if text == "quit" or text == "" or text == " ": break  
 text = text.strip()  
 seg\_list = list(jieba.cut(text))  
 x\_data = self.data\_helpers.handle\_input(' '.join(seg\_list))  
 feed\_dict = {self.cnn.input\_x: x\_data, self.cnn.dropout\_keep\_prob: self.dropout\_keep\_prob}  
 \_predic = sess.run([self.cnn.predictions], feed\_dict)  
 print("%s is %d" % (text, \_predic[0]))  
  
 def questionClassify(self, sess, text):  
 with sess.as\_default():  
 with sess.graph.as\_default():  
 text = text.strip()  
 seg\_list = list(jieba.cut(text))  
 x\_data = self.data\_helpers.handle\_input(' '.join(seg\_list))  
 feed\_dict = {self.cnn.input\_x: x\_data, self.cnn.dropout\_keep\_prob: self.dropout\_keep\_prob}  
 \_predic = sess.run([self.cnn.predictions], feed\_dict)  
 return \_predic[0]  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 graph = tf.Graph()  
 # 使用allow\_growth option，刚一开始分配少量的GPU容量，然后按需慢慢的增加  
 log\_device\_placement = True # 是否打印设备分配日志  
 allow\_soft\_placement = True # 如果你指定的设备不存在，允许TF自动分配设备  
 gpu\_options = tf.GPUOptions(per\_process\_gpu\_memory\_fraction=0.3)  
 session\_conf = tf.ConfigProto(gpu\_options=gpu\_options, allow\_soft\_placement=allow\_soft\_placement,  
 log\_device\_placement=log\_device\_placement)  
  
 sess = tf.Session(graph=graph, config=session\_conf)  
 classifyApp = classifyApplication(sess)  
 classifyApp.classifyApp(sess)

通过对问题文本的分类，我们可以判定该问题所属的类型，如对于问题“感冒了吃什么药”，经过分类判定该问题类型编号为3，也就是属于disease\_drug类型，表示该疾病类型应该服用那种药物。

### 7.3.2 生成数据查询语句

根据用户意图理解模块的分析，我们以及获取到用户输入的问题文本中的医疗实体信息以及用户的查询意图，基于以上信息，可以生成用户查询知识图谱的查询语句信息，具体代码如下：

**代码7.6 生成查询语句**

#!/usr/bin/env python  
# \_\*\_ coding:utf-8 \_\*\_  
  
  
class QuestionPaser:  
 *'''  
 问题解析类  
 '''* def build\_entitydict(self, args):  
 *'''  
 构建实体节点  
 '''* entity\_dict = {}  
 for arg, types in args.items():  
 for type in types:  
 if type not in entity\_dict:  
 entity\_dict[type] = [arg]  
 else:  
 entity\_dict[type].append(arg)  
  
 return entity\_dict  
  
  
 def parser\_main(self, res\_classify):  
 *'''  
 问题解析主方法* ***:param*** *res\_classify:* ***:return****:  
 '''* args = res\_classify['args']  
 entity\_dict = self.build\_entitydict(args)  
 question\_types = res\_classify['question\_types']  
 sqls = []  
 for question\_type in question\_types:  
 sql\_ = {}  
 sql\_['question\_type'] = question\_type  
 sql = []  
 if question\_type == 'disease\_symptom':  
 sql = self.sql\_transfer(question\_type, entity\_dict.get('disease'))  
  
 elif question\_type == 'symptom\_disease' or question\_type == ' symptom\_curway':  
 sql = self.sql\_transfer(question\_type, entity\_dict.get('symptom'))  
  
 elif question\_type == 'disease\_drug':  
 sql = self.sql\_transfer(question\_type, entity\_dict.get('disease'))  
  
 elif question\_type == 'drug\_disease':  
 sql = self.sql\_transfer(question\_type, entity\_dict.get('drug'))  
  
 elif question\_type == 'disease\_check':  
 sql = self.sql\_transfer(question\_type, entity\_dict.get('disease'))  
  
 elif question\_type == 'disease\_prevent':  
 sql = self.sql\_transfer(question\_type, entity\_dict.get('disease'))  
  
 elif question\_type == 'disease\_lasttime':  
 sql = self.sql\_transfer(question\_type, entity\_dict.get('disease'))  
  
 elif question\_type == 'disease\_cureway':  
 sql = self.sql\_transfer(question\_type, entity\_dict.get('disease'))  
  
 elif question\_type == 'disease\_desc':  
 sql = self.sql\_transfer(question\_type, entity\_dict.get('disease'))  
  
 if sql:  
 sql\_['sql'] = sql  
  
 sqls.append(sql\_)  
  
 return sqls  
  
  
 def sql\_transfer(self, question\_type, entities):  
 *'''  
 针对不同的问题类型，构造不同的查询语句，查询知识图谱中的信息* ***:param*** *question\_type:* ***:param*** *entities:* ***:return****:  
 '''* if not entities:  
 return []  
  
 # 构造查询语句  
 sql = []  
 # 查询疾病的原因  
 if question\_type == 'disease\_cause':  
 sql = ["MATCH (m:Disease) where m.name = '{0}' return m.name, m.cause".format(i) for i in entities]  
  
 # 查询疾病的防御措施  
 elif question\_type == 'disease\_prevent':  
 sql = ["MATCH (m:Disease) where m.name = '{0}' return m.name, m.prevent".format(i) for i in entities]  
  
 # 查询疾病的持续时间  
 elif question\_type == 'disease\_lasttime':  
 sql = ["MATCH (m:Disease) where m.name = '{0}' return m.name, m.cure\_lasttime".format(i) for i in entities]  
  
 # 查询疾病的治愈概率  
 elif question\_type == 'disease\_cureprob':  
 sql = ["MATCH (m:Disease) where m.name = '{0}' return m.name, m.cured\_prob".format(i) for i in entities]  
  
 # 查询疾病的治疗方式  
 elif question\_type == 'disease\_cureway':  
 sql = ["MATCH (m:Disease) where m.name = '{0}' return m.name, m.cure\_way".format(i) for i in entities]  
  
 # 查询疾病的易发人群  
 elif question\_type == 'disease\_easyget':  
 sql = ["MATCH (m:Disease) where m.name = '{0}' return m.name, m.easy\_get".format(i) for i in entities]  
  
 # 查询疾病的相关介绍  
 elif question\_type == 'disease\_desc':  
 sql = ["MATCH (m:Disease) where m.name = '{0}' return m.name, m.desc".format(i) for i in entities]  
  
 # 查询疾病有哪些症状  
 elif question\_type == 'disease\_symptom':  
 sql = ["MATCH (m:Disease)-[r:has\_symptom]->(n:Symptom) where m.name = '{0}' return m.name, r.name, n.name".format(i) for i in entities]  
  
 # 查询症状会导致哪些疾病  
 elif question\_type == 'symptom\_disease':  
 sql = ["MATCH (m:Disease)-[r:has\_symptom]->(n:Symptom) where n.name = '{0}' return m.name, r.name, n.name".format(i) for i in entities]  
  
 # 查询疾病的并发症  
 elif question\_type == 'disease\_acompany':  
 sql1 = ["MATCH (m:Disease)-[r:acompany\_with]->(n:Disease) where m.name = '{0}' return m.name, r.name, n.name".format(i) for i in entities]  
 sql2 = ["MATCH (m:Disease)-[r:acompany\_with]->(n:Disease) where n.name = '{0}' return m.name, r.name, n.name".format(i) for i in entities]  
 sql = sql1 + sql2  
  
 # 查询疾病的忌口  
 elif question\_type == 'disease\_not\_food':  
 sql = ["MATCH (m:Disease)-[r:no\_eat]->(n:Food) where m.name = '{0}' return m.name, r.name, n.name".format(i) for i in entities]  
  
 # 查询疾病建议吃的东西  
 elif question\_type == 'disease\_do\_food':  
 sql1 = ["MATCH (m:Disease)-[r:do\_eat]->(n:Food) where m.name = '{0}' return m.name, r.name, n.name".format(i) for i in entities]  
 sql2 = ["MATCH (m:Disease)-[r:recommand\_eat]->(n:Food) where m.name = '{0}' return m.name, r.name, n.name".format(i) for i in entities]  
 sql = sql1 + sql2  
  
 # 已知忌口查疾病  
 elif question\_type == 'food\_not\_disease':  
 sql = ["MATCH (m:Disease)-[r:no\_eat]->(n:Food) where n.name = '{0}' return m.name, r.name, n.name".format(i) for i in entities]  
  
 # 已知推荐查疾病  
 elif question\_type == 'food\_do\_disease':  
 sql1 = ["MATCH (m:Disease)-[r:do\_eat]->(n:Food) where n.name = '{0}' return m.name, r.name, n.name".format(i) for i in entities]  
 sql2 = ["MATCH (m:Disease)-[r:recommand\_eat]->(n:Food) where n.name = '{0}' return m.name, r.name, n.name".format(i) for i in entities]  
 sql = sql1 + sql2  
  
 # 查询疾病常用药品－药品别名记得扩充  
 elif question\_type == 'disease\_drug':  
 sql1 = ["MATCH (m:Disease)-[r:common\_drug]->(n:Drug) where m.name = '{0}' return m.name, r.name, n.name".format(i) for i in entities]  
 sql2 = ["MATCH (m:Disease)-[r:recommand\_drug]->(n:Drug) where m.name = '{0}' return m.name, r.name, n.name".format(i) for i in entities]  
 sql = sql1 + sql2  
  
 # 已知药品查询能够治疗的疾病  
 elif question\_type == 'drug\_disease':  
 sql1 = ["MATCH (m:Disease)-[r:common\_drug]->(n:Drug) where n.name = '{0}' return m.name, r.name, n.name".format(i) for i in entities]  
 sql2 = ["MATCH (m:Disease)-[r:recommand\_drug]->(n:Drug) where n.name = '{0}' return m.name, r.name, n.name".format(i) for i in entities]  
 sql = sql1 + sql2  
  
 # 查询疾病应该进行的检查  
 elif question\_type == 'disease\_check':  
 sql = ["MATCH (m:Disease)-[r:need\_check]->(n:Check) where m.name = '{0}' return m.name, r.name, n.name".format(i) for i in entities]  
  
 # 已知检查查询疾病  
 elif question\_type == 'check\_disease':  
 sql = ["MATCH (m:Disease)-[r:need\_check]->(n:Check) where n.name = '{0}' return m.name, r.name, n.name".format(i) for i in entities]  
  
 return sql  
  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 from question\_classifier import \*  
  
 handler = QuestionPaser()  
 QChandler = QuestionClassifier()  
 while 1:  
 question = input(' input an question:')  
 data = QChandler.classify(question)  
 print(data)  
 sqls = handler.parser\_main(data)  
 print(sqls)

针对不同的问题类型，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“感冒应了吃什么药”，根据获取到的问题类型为查询疾病与药物的问题类型，该查询语句如下所示：

“MATCH (m:Disease)-[r:common\_drug]->(n:Drug) where m.name = '感冒' return m.name, r.name, n.name”

### 7.3.3 答案生成模块

通过查询语句查询医疗知识图谱，获取相应的答案信息，同时根据问题类型，选用相应的模板，生成答案文本并返回给用户，具体代码如下所示：

**代码7.7 答案生成模块**

#!/usr/bin/env python  
# \_\*\_ coding:utf-8 \_\*\_  
  
from py2neo import Graph  
  
  
class AnswerSearcher:  
 *'''  
 查询答案类  
 '''* def \_\_init\_\_(self):  
 # 连接Neo4j数据库  
 self.g = Graph("bolt://localhost:11008")  
 # 查询数据条数限制  
 self.num\_limit = 20  
  
  
 def search\_main(self, sqls):  
 *'''  
 执行cypher查询，并返回相应结果* ***:param*** *sqls:* ***:return****:  
 '''* final\_answers = []  
 for sql\_ in sqls:  
 question\_type = sql\_['question\_type']  
 queries = sql\_['sql']  
 answers = []  
 for query in queries:  
 ress = self.g.run(query).data()  
 answers += ress  
 final\_answer = self.answer\_prettify(question\_type, answers)  
 if final\_answer:  
 final\_answers.append(final\_answer)  
 return final\_answers  
  
  
 def answer\_prettify(self, question\_type, answers):  
 *'''  
 根据相应的问题类型，采用相应的答案生成模板* ***:param*** *question\_type:* ***:param*** *answers:* ***:return****:  
 '''* final\_answer = []  
 if not answers:  
 return ''  
 if question\_type == 'disease\_symptom':  
 desc = [i['n.name'] for i in answers]  
 subject = answers[0]['m.name']  
 final\_answer = '{0}的症状包括：{1}'.format(subject, '；'.join(list(set(desc))[:self.num\_limit]))  
  
 elif question\_type == 'symptom\_disease':  
 desc = [i['m.name'] for i in answers]  
 subject = answers[0]['n.name']  
 final\_answer = '症状{0}可能染上的疾病有：{1}'.format(subject, '；'.join(list(set(desc))[:self.num\_limit]))  
  
 elif question\_type == 'disease\_cause':  
 desc = [i['m.cause'] for i in answers]  
 subject = answers[0]['m.name']  
 final\_answer = '{0}可能的成因有：{1}'.format(subject, '；'.join(list(set(desc))[:self.num\_limit]))  
  
 elif question\_type == 'disease\_prevent':  
 desc = [i['m.prevent'] for i in answers]  
 subject = answers[0]['m.name']  
 final\_answer = '{0}的预防措施包括：{1}'.format(subject, '；'.join(list(set(desc))[:self.num\_limit]))  
  
 elif question\_type == 'disease\_lasttime':  
 desc = [i['m.cure\_lasttime'] for i in answers]  
 subject = answers[0]['m.name']  
 final\_answer = '{0}治疗可能持续的周期为：{1}'.format(subject, '；'.join(list(set(desc))[:self.num\_limit]))  
  
 elif question\_type == 'disease\_cureway':  
 desc = [';'.join(i['m.cure\_way']) for i in answers]  
 subject = answers[0]['m.name']  
 final\_answer = '{0}可以尝试如下治疗：{1}'.format(subject, '；'.join(list(set(desc))[:self.num\_limit]))  
  
 elif question\_type == 'disease\_cureprob':  
 desc = [i['m.cured\_prob'] for i in answers]  
 subject = answers[0]['m.name']  
 final\_answer = '{0}治愈的概率为（仅供参考）：{1}'.format(subject, '；'.join(list(set(desc))[:self.num\_limit]))  
  
 elif question\_type == 'disease\_easyget':  
 desc = [i['m.easy\_get'] for i in answers]  
 subject = answers[0]['m.name']  
  
 final\_answer = '{0}的易感人群包括：{1}'.format(subject, '；'.join(list(set(desc))[:self.num\_limit]))  
  
 elif question\_type == 'disease\_desc':  
 desc = [i['m.desc'] for i in answers]  
 subject = answers[0]['m.name']  
 final\_answer = '{0},熟悉一下：{1}'.format(subject, '；'.join(list(set(desc))[:self.num\_limit]))  
  
 elif question\_type == 'disease\_acompany':  
 desc1 = [i['n.name'] for i in answers]  
 desc2 = [i['m.name'] for i in answers]  
 subject = answers[0]['m.name']  
 desc = [i for i in desc1 + desc2 if i != subject]  
 final\_answer = '{0}的症状包括：{1}'.format(subject, '；'.join(list(set(desc))[:self.num\_limit]))  
  
 elif question\_type == 'disease\_not\_food':  
 desc = [i['n.name'] for i in answers]  
 subject = answers[0]['m.name']  
 final\_answer = '{0}忌食的食物包括有：{1}'.format(subject, '；'.join(list(set(desc))[:self.num\_limit]))  
  
 elif question\_type == 'disease\_do\_food':  
 do\_desc = [i['n.name'] for i in answers if i['r.name'] == '宜吃']  
 recommand\_desc = [i['n.name'] for i in answers if i['r.name'] == '推荐食谱']  
 subject = answers[0]['m.name']  
 final\_answer = '{0}宜食的食物包括有：{1}\n推荐食谱包括有：{2}'.format(subject, ';'.join(list(set(do\_desc))[:self.num\_limit]), ';'.join(list(set(recommand\_desc))[:self.num\_limit]))  
  
 elif question\_type == 'food\_not\_disease':  
 desc = [i['m.name'] for i in answers]  
 subject = answers[0]['n.name']  
 final\_answer = '患有{0}的人最好不要吃{1}'.format('；'.join(list(set(desc))[:self.num\_limit]), subject)  
  
 elif question\_type == 'food\_do\_disease':  
 desc = [i['m.name'] for i in answers]  
 subject = answers[0]['n.name']  
 final\_answer = '患有{0}的人建议多试试{1}'.format('；'.join(list(set(desc))[:self.num\_limit]), subject)  
  
 elif question\_type == 'disease\_drug':  
 desc = [i['n.name'] for i in answers]  
 subject = answers[0]['m.name']  
 final\_answer = '{0}通常的使用的药品包括：{1}'.format(subject, '；'.join(list(set(desc))[:self.num\_limit]))  
  
 elif question\_type == 'drug\_disease':  
 desc = [i['m.name'] for i in answers]  
 subject = answers[0]['n.name']  
 final\_answer = '{0}主治的疾病有{1},可以试试'.format(subject, '；'.join(list(set(desc))[:self.num\_limit]))  
  
 elif question\_type == 'disease\_check':  
 desc = [i['n.name'] for i in answers]  
 subject = answers[0]['m.name']  
 final\_answer = '{0}通常可以通过以下方式检查出来：{1}'.format(subject, '；'.join(list(set(desc))[:self.num\_limit]))  
  
 elif question\_type == 'check\_disease':  
 desc = [i['m.name'] for i in answers]  
 subject = answers[0]['n.name']  
 final\_answer = '通常可以通过{0}检查出来的疾病有{1}'.format(subject, '；'.join(list(set(desc))[:self.num\_limit]))  
  
 return final\_answer  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 searcher = AnswerSearcher()

### 7.3.4 医疗诊断问答系统系统实现

通过上面的步骤，我们已经完成了一个完整的医疗诊断问答系统的功能模块，通过对用户输入问题文本的分析，识别其中的医疗实体，了解用户的查询意图，构造查询语句获取用户意图获取的信息，并最终根据问题类型选用相应的答案生成模板，生成最终的答案返回给用户，具体实现代码如下：

#!/usr/bin/env python  
# \_\*\_ coding:utf-8 \_\*\_  
  
  
#from question\_classifier import \*  
from question\_parser import \*  
from answer\_search import \*  
from question\_analysis import \*  
  
  
class ChatBotGraph:  
 *'''  
 问答系统类  
 '''* def \_\_init\_\_(self):  
 #self.classifier = QuestionClassifier()  
 self.classifier = question\_ays()  
 self.parser = QuestionPaser()  
 self.searcher = AnswerSearcher()  
  
 def chat\_main(self, sent):  
 answer = '抱歉，您的问题暂时没有找到答案，我会将您的问题记录下来。'  
 res\_classify = self.classifier.analysis(sent)  
 # print(res\_classify)  
 if not res\_classify:  
 return answer  
 res\_sql = self.parser.parser\_main(res\_classify)  
 print(res\_sql)  
 final\_answers = self.searcher.search\_main(res\_sql)  
 if not final\_answers:  
 return answer  
 else:  
 return '\n'.join(final\_answers)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 handler = ChatBotGraph()  
 question= '##'  
 print('您好，我是小艾医生，请问您哪里不舒服？希望我的回答可以帮到您！')  
 while(question!="" and question!=" "):  
 question = input('用户:')  
 if question == "quit" or question=="" or question == " ":break  
 answer = handler.chat\_main(question)  
 print('小艾医生:', answer)  
 print("再见！")

### 7.3.5 医疗诊断问答系统效果展示

经过上面的系统实现，接下来验证下我们的医疗诊断问答系统的实际效果，具体展示情况如下所示：

您好，我是小艾医生，请问您哪里不舒服？希望我的回答可以帮到您！

用户:我感冒了应该吃什么药呢

小艾医生: 感冒通常的使用的药品包括：蒲公英颗粒；利巴韦林颗粒；头孢拉定胶囊；伤风停胶囊；愈美胶囊；抗病毒口服液；洛索洛芬钠胶囊；阿莫西林颗粒；麻黄止嗽丸；消炎片；酚咖片；银芩胶囊；匹多莫德分散片；肺宁片；喉痛灵片；头孢丙烯分散片；依托红霉素片；感冒灵颗粒；洛索洛芬钠片；风油精

用户:高血压有什么症状

小艾医生: 高血压的症状包括：重压感、紧箍感...；颈部搏动；头晕目眩；乏力；头晕；眼花；血压高；神经性呕吐；心脏主动脉瓣返流

用户:感冒怎么预防

小艾医生: 感冒的预防措施包括：本病全年皆可发病，冬春季节多发，可通过含有病毒的飞沫或被污染的用具传播，多数为散发性，但常在气候突变时流行，由于病毒的类型较多，人体对各种病毒感染后产生的免疫力较弱且短暂，并无交叉免疫，同时在健康人群中有病毒携带者，故一个人一年内可有多次发病。

日常预防

四种简易预防感冒的方法

1、冷水洗脸、热水泡足法：每日晨、晚养成用冷水浴面、热水泡足的习惯，这有助于提高身体抗病能力。

2、盐水漱口：每日早晚、餐后用淡盐水漱口，以清除口腔病菌。在流感流行的。

3、食醋熏蒸法：把陈醋加热，关上门窗，隔一段时间在房间里熏蒸一次，可有效杀除感冒等病毒。

4、饮用姜茶法：晚上睡觉前，用萝卜加醋熬汤，或以生姜、红糖适量煮水代茶饮，对防止感冒有很好的效果。

专业指导

1、补充维生素E、维生素C。维生素E、维生素C都能有效提高人体免疫力。

2、保证足够的睡眠。数据显示，只睡半宿的人，免疫力会下降大约三成。而在睡足8小时后，免疫力会立刻恢复。

3、进行鼻部按摩。大部分感冒中，鼻咽部是最初感染的部位，因此鼻部按摩能有效预防感冒。