# 第 3 章

# 医药疾病知识图谱

本章以医药疾病知识图谱的构建作为实战案例。通过获取相关医药数据，经过实体的抽取、三元组的构建等操作，构建形成知识图谱并针对其应用进行相关描述。

本章主要学习 NumPy 的用法和最大前向匹配算法，掌握通过半结构化数据搭建简单知识图谱的基础流程。

## 3.1 项目设计

医药疾病与知识图谱的结合，在近些年非常受欢迎，可以帮助医药疾病领域的学者厘清脉络、挖掘关系。医学类知识图谱慢慢成为了大数据时代进行医学数据挖掘和知识发现的重要基础设施，逐步变成了医学领域知识发现的技术支撑。

### 3.1.1 需求分析

医药疾病知识图谱构建离不开大量的三元组，而三元组的获取除了 IS-A 上下位抽取，另一项就是关系抽取。关系抽取是信息抽取领域中的重要任务之一，目的在于抽取文本中的实体对，以及识别实体对之间的语义关系。例如“弥漫性肺泡出血易合并肺部感染”中，“弥漫性肺泡出血”与“肺部感染”都是疾病，它们之间的关系是＜疾病，合并症，疾病＞。存在于海量医药文本中的知识体系网络，可以为其他 NLP 技术（实体链接、query 解析、问答系统、信息检索等）提供可解释性的先验知识（知识表示）和推理。

与人们认识世界一样，实体关系相当于事物与事物之间的联系，而属性则丰富了对事物本身的认识。同理，医药疾病文本中也存在描述实体属性的信息，如“通过用手搔抓患癣的部位如足趾间，或与患者共用鞋袜、手套、浴巾、脚盆等是手癣的主要传播途径。”这句话中，“手癣”的“传播途径”是“用手搔抓……等”。通过例子可以发现，属性名通常是一个名词短语，但是属性值可以是词，也可以是句子，又如“发生丙肝的主要原因是丙型肝炎病毒”中，“丙肝”的“主要原因”是“丙型肝炎病毒”。属性的概念本身就具备较宽泛的灵活性，学术界目前也没有一个统一的标准，所以需要在具体落地场景中根据实际情况做相应的设计。

在医药疾病文本数据中进行信息抽取，必须对其有一定的认识和分析。关系复杂，密度大，但基本无歧义，指代情况明显，由于表达相对简短，上下文信息没有固定模式，重叠现象普遍存在。因此，需要一定的医药疾病领域先验知识和模型结构上的处理。

### 3.1.2 工作流程

总体技术框架如图 3-1 所示，其构建的具体流程如下。

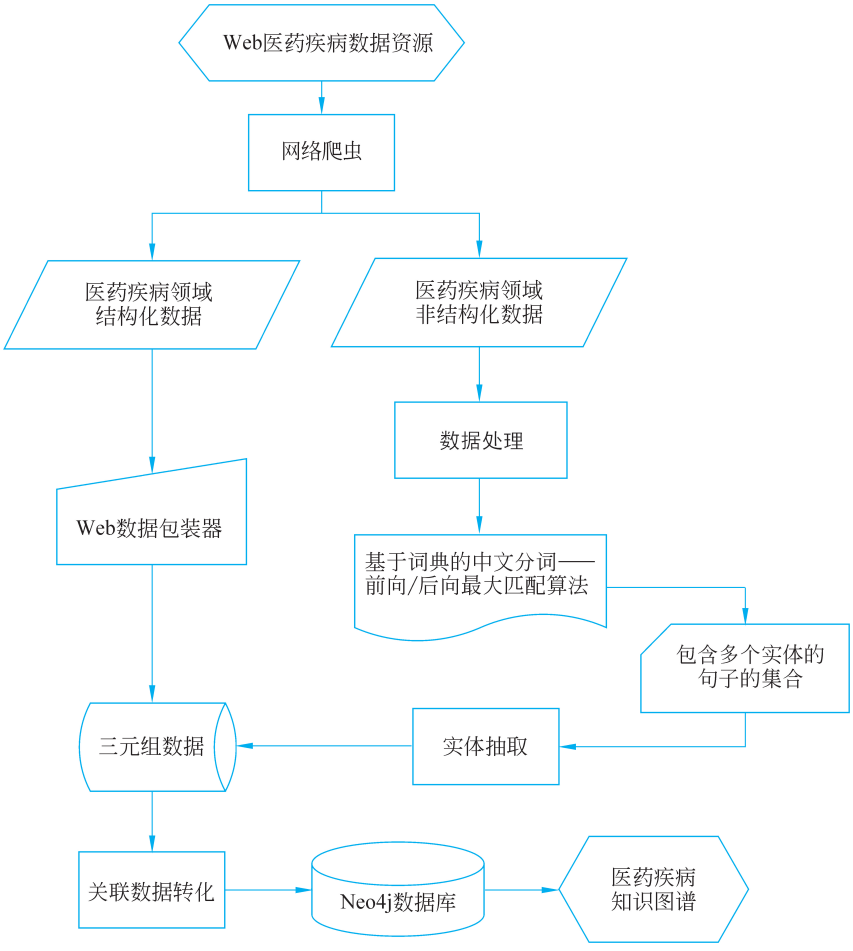


图 3-1　医药疾病知识图谱构建流程

（1）通过网络爬虫技术，从 Web 资源中获取医药疾病领域的半结构化数据和非结构化数据。

（2）对于半结构化数据，通过自定义的 Web 数据包装器直接抽取其中的三元组知识。对于非结构化数据，例如并发症中的描述数据，其中的并发症与中心实体疾病有关联，可以对其进行数据的预处理，通过基于字典的中文分词前后向最大匹配算法识别出句子中的命名实体。

（3）将半结构化数据和非结构化数据中的三元组知识进行整合，转化为 RDF 形式的关联数据。

（4）将整合的所有数据，使用 Python 编程语言和 Neo4j 的数据库语句，写入图数据库中，得到医药疾病领域的知识图谱，进而进行展示和运用。

### 3.1.3　技术选型

#### 1. NumPy

NumPy(Numerical Python 的缩写)是 Python 的一个开源的扩展程序库，用来支持

Python 语言进行高性能数值计算。NumPy 提供了大量的高级数学和数值处理功能，是 Python 科学计算环境中的核心扩展库。

NumPy 的主要功能是其给 Python 提供了一种数组类型，即 ndarray，它是一种高效并且便于使用的多维数组对象，可以用来存储同类型的数据。NumPy 的 ndarray 提供了许多有用的方法，用于对数据进行处理、操作和计算，支持如下操作。

（1）用简单的语法定义、操作以及调用数组。

（2）轻松地执行对整个数组的数学运算，如加、减、乘、除等。

（3）针对整个数组或部分数组执行函数和统计分析功能，如最大、最小、平均值、标准差等。

（4）使用广播功能对不同形状的数组进行运算。

（5）通过对数组的分片和索引，以及基于布尔值和条件的索引等方式进行数据筛选和过滤。

NumPy 支持许多其他的计算功能。

（1）线性代数：包括矩阵操作、矩阵分解、求解线性方程组等。

（2）快速傅里叶变换（FFT）：用于频域操作，如信号处理和图像处理。

（3）随机数生成和分布：例如，高斯分布、泊松分布、均匀分布等。

NumPy 的性能是其他 Python 数据结构的几个数量级，而且还有多线程支持，可以跨平台运行。因此，NumPy 在计算机视觉、机器学习、扩展科学计算等领域广泛应用。

NumPy 的代码库是使用 C 语言开发的，因此执行速度非常快。NumPy 可以与其他科学计算工具配合使用，如 SciPy（科学计算专用 Python 工具包）、Matplotlib（基于 NumPy 的绘图库）、Pandas（数据分析包）等，组成 Python 科学计算生态系统，使得 Python 成为一个完整的科学计算工具。

### 2. 最大前向匹配算法

考虑到在医药疾病领域中，对于专业术语的词不能被随意切分，如“小儿金黄色葡萄球菌肺炎”不能被切分成“小儿”“金黄色”“葡萄”“球菌”“肺炎”多个单独的词汇。在获取数据时需要对数据进行清洗，为了避免专业术语词汇切分不准确的问题，在实战中使用基于词典的词汇进行切分，其主要原理流程如图 3-2 所示。

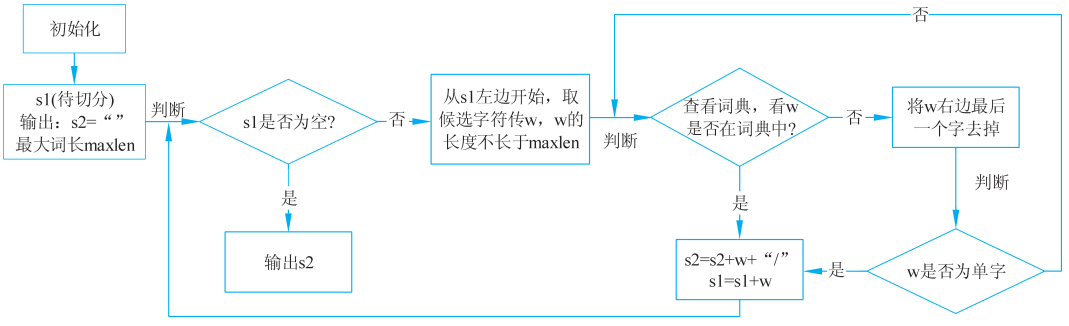


图 3-2 最大前向匹配算法原理

其中，s1 为待切分的字符串，s2 为输出词串，maxlen 为最大字符串长度，w 为候选字符

串。首先对 s1、s2、maxlen 进行初始化，判断 s1 是否为空，若为空则说明字符串已经切分完，输出 s2；若为非空，则从 s1 左边开始取不大于 maxlen 长度的字符串 w，查询词典。若 w 在词典中则进行计算“s2＝s2＋w＋‘/’”“s1＝s1＋w”，接着开始新一轮判断 s1；若 w 不在词典中，则将 w 最右边的一个字符去掉，对 w 进行更新，然后接着又判断 w 是否为单字，若是再按上述方法对 s1 和 s2 进行更新，若否则判断其是否为词典中的词，以此迭代循环，最终得到结果 s2。

### 3.1.4　开发准备

#### 1. 系统开发环境

本章的软件开发及运行环境如下。

(1) 操作系统：Windows 7、Windows 10、Linux。

(2) 虚拟环境：virtualenv 或者 miniconda。

(3) Python 第三方库：requests、lxml、NumPy。

(4) 数据库和驱动：Neo4j＋py2neo。

(5) 开发工具：PyCharm。

(6) 浏览器：Chrome 浏览器。

#### 2. 文件夹组织结构

文件夹组织结构如下所示。

```
├─medical_graph
|   ├─common
|   |   ├─ConnNeo4j.py              #连接 Neo4j 数据库
|   ├─crawler
|   |   ├─BaseSpider.py             #基础爬虫
|   |   ├─ MedicalSpider.py         #医药数据采集
|   ├─data                          #数据
|   ├─utils
|   |   ├─common_util.py
|   |   ├─get_config.py             #读取配置信息
|   |   ├─max_cut.py                #最大向前匹配算法
|   |   ├─web_common_util.py
|   ├─config.ini                    #配置信息
|   ├─config.py                     #读取配置信息
|   ├─readme.md                     #项目的说明信息
|   ├─requirements.txt              #项目所依赖的 pip 安装列表
```

## 3.2　数据准备和预处理

### 3.2.1　数据描述

医药数据是医药疾病知识图谱的基础，如果希望织出一张庞大的关于医药疾病领域的“网”，那么首先就需要大量的医药知识的“线”穿梭其中。相较于国外的英文医药疾病数据，

国内的中文医药没有较全面的开源数据集。因医药数据量比较庞大，牵扯的元素繁多，在标注整理方面难度异常艰难，也没有人整理出比较完整的且带有标注信息的医药疾病数据集。因此，对于医药方面知识图谱应用，需要从其他途径进行获取和处理。

在本章实战案例中所运用的医药疾病数据集，是在“寻医问药”医药网站上爬取的医药疾病数据，将此作为知识图谱的源数据集使用。其中包括疾病的分类，疾病常识（病因、预防、并发症），诊断方法（症状、检查），治疗方案（治疗、护理、饮食保健），检查方案等数据，医药疾病数据信息还比较全面。为方便教学使用，部分网页已下载供分析。

### 3.2.2 数据获取

构建医药疾病知识图谱最基础的工作，就是挑选和获取合适且全面的数据。全面高质量的医药疾病数据，能为后续的知识图谱构建工作打好基底，让知识图谱的内容更加全面和详细。爬取医药疾病数据的具体流程如图 3-3 所示。

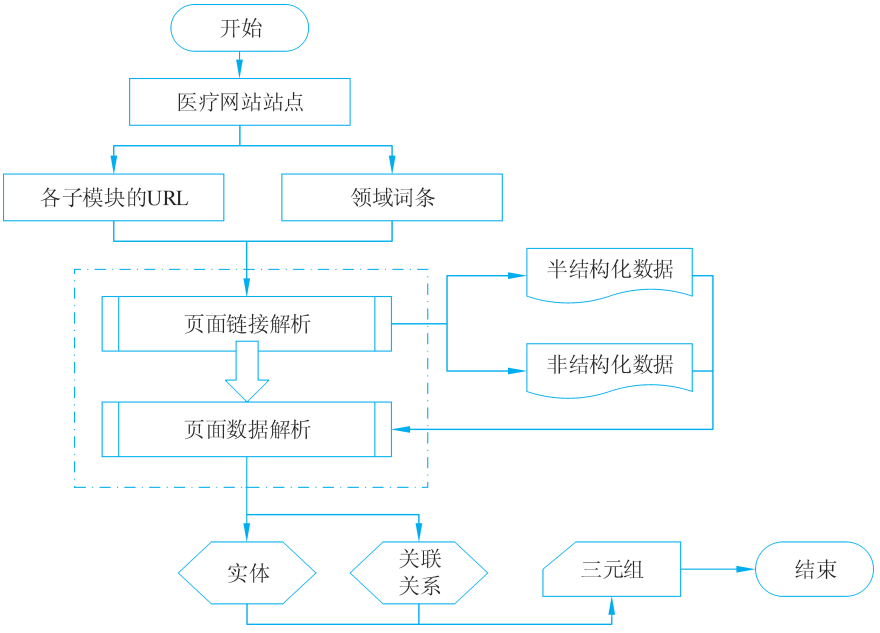


图 3-3 医药数据采集流程

首先分析指定医药网站的具体结构。该网站包括有子模块重定向链接和领域词条，如图 3-4 所示。再对子模块 URL 下的页面和领域词条模块进行页面链接解析和页面数据解析。在该页面中包括半结构化数据（如词条信息）与非结构化数据（如预防方法的介绍、疾病的介绍等）。通过对半结构化数据的爬取与非结构化数据的有效提取和利用，可以从中抽取医药知识中间的实体和关系形成三元组，以更好地帮助知识图谱的构建工作。

在网站中包括疾病种类、病因、预防方法、治疗药物、饮食保健等医药关键数据，可以将这些数据爬取下来作为知识图谱的元数据。在本项目中，采用 Python 语言中的 request 网页请求对网页进行抓取工作。首先以网站站点的各子模块的初始 URL 作为输入，然后通过数据解析模块对页面进行解析。数据解析模块包括页面链接解析和页面数据解析。其中，页面链接解析是将每个子模块的 URL 以循环嵌套的形式，将每种疾病的各个链接提取


图 3-4　医药网站结构分析

出来，作为下一次页面请求的输入。页面数据解析则负责具体提取每种疾病的简介、病因、预防、并发症、症状、治疗药物、饮食保健等信息。

在这里展示部分关键代码，如下所示。

```
1. #分离出的各网页 URL 标签
2. page_list =['gaishu', 'cause', 'prevent', 'neopathy', 'symptom', 'inspect',
   'treat', 'food', 'drug']
3. #爬取疾病数据，范围(a,b)中数字表示疾病的 id
4. for i in range(1, 3):
5.     disease_dict ={}
6.     data_all ={}
7.     url_list =[]
8.     #拼接网站 URL
9.     for p in page_list:
10.        url ='http://jib.xywy.com/il_sii/%s/%s.htm' %(p, i)
11.        url_list.append(url)
12.    basicinfo =self.basicinfo_spider(url_list[0])
13.    diseas_cause =self.disease_cause(url_list[1])
14.    prevent =self.disease_cause(url_list[2])
15.    disease_together =self.disease_together(url_list[3])
16.    symptom =self.symptom(url_list[4])
17.    inspect =self.inspect(url_list[5])
18.    teart =self.treat_spider(url_list[6])
19.    food =self.food_spider(url_list[7])
```

页面数据解析主要负责分析请求到的页面数据，根据网页中定义的规则与标签数据，对数据进行解析和拆解，获取各词条、各部分所包含的医药信息。接着将数据通过提取和转

换，分离出实体，创建、提取、转换关联关系，形成三元组。最后将采集到的数据存储在本地，写入 Neo4j 数据库。

### 3.2.3 数据预处理

数据的预处理主要是将从医药网站抓取的数据进行清洗，然后按照一定的规则进行格式化，最后将处理好的数据存储在数据库中。因为在该网页中抓取的词条数据中包含一些无关的 HTML 文本信息，如网页标题信息、标签信息、空白字符、换行字符等。这些信息对知识图谱的构建毫无帮助，因此需要进行预处理，将无用的信息剔除，得到有用的医药数据。抓取的数据分为概述、病因、预防方法、并发症、饮食建议等多个部分，在此只对一部分进行举例教学。

以医药疾病中的基本信息为例，其代码如下所示。

```
def basicinfo_spider(self, url):
    #获取 HTML,返回 response 响应代码
    responce =self.get_html(url)
    #获取 HTML 文本
    html =responce.text
    selector =etree.HTML(html)
    #xpath 对网页进行解析
    title =selector.xpath('//title/text()')[0]
    title =title.split('的简介')[0]
    #类别
    type =selector.xpath('//div[@class="wrap mt10 nav-bar"]/a/text()')
    #简介
    desc =selector.xpath('//div[@class="jib-articl-con jib-lh-articl"]/p/text()')
    desc_info =desc[0].replace(' ', '').replace('\n', '').replace('\t', '').replace('\r', '')
    #常识
    common =selector.xpath('//div[@class="mt20 articl-know"]/p')
    common_list =[]
    for _ in common:
        #去空格、去符号'/n',
        info1 =_.xpath('string(.)').replace('\n', '').replace('\t', '').replace('\xa0', '')
        common_list.append(info1)
    summery ={}
    summery["disease_name"] =title
    summery["visit_department"] =type
    summery["disease_intro"] =desc_info
    #提取常识中数组中的元素并分离,放入字典
    for attr in common_list:
        attr_pair =attr.split(': ')
        if len(attr_pair) ==2:
            key =attr_pair[0]
            value =attr_pair[1].strip()
            summery[key] =value
    summery_modify ={}
    #根据字典替换 key 值
```

```
    for k, v in summery.items():
        attr_en =self.instead_key.get(k)
        if attr_en in ["insurance_disease", "disease_prob", "easy_get_
        people", "infect_mode",
                       "treat_cycle", "cure_rate", "treat_cost"]:
            summery_modify[attr_en] =v
        elif attr_en in ["visit_department", "treat_mode", "always_drug",
        "complication"]:
            summery_modify[attr_en] =[i for i in re.split(r"[ ]+", v) if i]
        else:
            summery_modify[k] =summery.get(k)
```

首先请求 URL，通过 request 方法拿到 HTML 的响应值，提取出 HTML 的文本值后通过 etree 模块中的 xpath 对文本中的词条进行提取和预处理，去掉文本中字里行间的换行、回车、空格、制表符等无用信息。再清理中文名，抽取其中的数组，然后分离出 key 值与 value 值，通过字典将 key 中的中文替换成英文，与整体的数据保持一致，最终得到最基础的医药疾病概要数据。

另外，其他子模块链接下数据模块的预处理，也与该方法类似，在此不做赘述。

## 3.3　知识建模和存储

### 3.3.1　实体抽取

实体抽取的主要任务是要识别与抽取文本中出现的医学专有名称和有意义的数量短语并加以归类，如疾病名称、药品名称、病因、科室名等。面对医药疾病数据的实体抽取，需要采用一定的抽取手段，这也是信息抽取的重要子任务。

#### 1. 半结构化数据

在本章中，经过预处理后的医药疾病数据结构形式主要以半结构化与非结构化数据为主。半结构化数据是一种特殊的结构形式数据，虽然不符合关系数据库或其他形式的数据表形式，但又包含标签或其他标记，可用来分离语义元素并保持记录和数据字段的层次结构。自互联网出现以来，半结构化数据越来越丰富，成为知识获取的重要来源。通过抓取的方式可以获取重要的半结构化数据，进而处理成固定的形式。

抽取的内容主要包括如下。

(1) 标签：抽取词条的标题，并将其定义为实体的标签。

(2) 简介：抽取词条页面的介绍性模块，定义为实体的简介，也可定义词条信息的长度。

(3) 重定向：抽取词条重定向链接，并抽取其页面中的标题作为实体的标签，其子标题及其介绍作为实体的概述。

(4) 分类：抽取词条所属的分类。

(5) 信息框：从词条页面信息框中提取到的结构化信息。

在本章中，爬取后的数据主要处理成 JSON 格式，如格式化后的“颅骨纤维异常增生症”

主要信息如下所示。

```
{
    'disease_name': '颅骨纤维异常增生症',
    'visit_department': [
        '内科',
        '神经内科'
    ],
    'disease_intro': '颅骨纤维异常增生症是一种有纤维组织替代骨质而引起颅骨增厚,
    变形的疾病。病变可只累及颅骨,也可同时累及身体其他部位的骨骼。',
    'insurance_disease': '否',
    'disease_prob': '0.025%',
    'easy_get_people': '无特定人群',
    'infect_mode': '无传染性',
    'complication': [
        '耳聋'
    ],
    'treat_mode': [
        '药物治疗',
        '康复治疗',
        '支持性治疗'
    ],
    'treat_cycle': '3个月',
    'cure_rate': '60%',
    'always_drug': [
        '盐酸米诺环素胶囊',
        '益髓颗粒'
    ],
    'treat_cost': '根据不同医院,收费标准不一致,市三甲医院约(5000~10000元)'
}
```

在这里比较方便的是,对于半结构化的医药数据,抓取下来可以把部分关键字直接作为实体,如“疾病名称——disease_name”“疾病介绍——disease_intro”等抽象概念唯一的信息,可以直接将其抽取出来当作医药知识图谱的实体,而如“就诊科室——visit_department”“发病率——disease_prob”“传染方式——infect_mode”“饮食——food”这些信息中间包括许多重复的词条,比如在就诊科室中,包括妇产科、感染科、内分泌科、肝胆外科、妇科、耳鼻喉科、肿瘤科、传染科等,而在抓取的每种疾病的下面都包括就诊科室,且有些疾病就诊科室是相同的,那么就导致了放到字典中的就诊科室数据存在重复的情况。为了保持其实体的唯一性,需要将就诊科室的数据进行筛选和剔除。其他类似的情况,如传染方式可能相同、饮食方案中的食物名称相同、发病率的数值相等等这些数据,都需要对其进行去重,然后再抽取作为实体。

下面举一个例子进行说明,其部分代码如下所示。

```
complication_list =[]
visit_department_list =[]
……
……
#并发症实体去重
if "complication" in basicinfo:
```

```
    complication_list +=basicinfo['complication']

#就诊部门实体去重
if "visit_department" in basicinfo:
    visit_department_list +=basicinfo["visit_department"]
    #关系
    if len(basicinfo["visit_department"]) ==1:
        relation_diss_department.append([basicinfo.get("disease_name"),
        basicinfo["visit_department"]])
    if len(basicinfo["visit_department"]) ==2:
        relation_diss_department.append(
                [basicinfo.get("disease_name")[1], basicinfo["visit_
                department"][0]])
        relation_diss_department.append([basicinfo["visit_department"],
        basicinfo.get("disease_name")[1]])
……
……
complication_list =list(set(complication_list))
visit_department_list =list(set(visit_department_list))
```

将抓取的每种疾病相关的就诊科室、并发症、发病率、传染方式等数据信息分门别类地放到不同的数组当中，然后以循环的方式将每种疾病涉及的不同种类的相关数据，以不同的数组进行存储，然后通过数组去重的方式，将每类数组当中的数据进行去重，得到相应的实体数据。另外，在该医药网页中疾病的实体是唯一的，可以直接抽取后放入数组，为后续的知识图谱的构建工作搭好节点的基础。

#### 2. 非结构化数据

实体抽取的目的是从文本中抽取实体信息元素，通常包含人名、组织机构名、地理位置、时间、日期、字符值等标签，具体的标签定义可根据任务不同而调整。想要从文本中提取出实体，首先需要在文本中识别和定位实体，然后再将别的实体分类到预定义的类别当中去。

在本章中，以抽取并发症模块中所涉及的疾病为例，主要通过运用基于字典的双向最大匹配算法来实现。在医学专业领域中，专业词汇的切分也是需要考虑的问题。如在医药词汇中的“颅骨纤维异常增生症”，它作为一个专业医学词汇，不能被切分成“颅骨”“纤维”“异常”“增生”“症”多个独立词汇，所以需要基于词典来进行分词和切分。在此说明一下，如果需要疾病的词典，也可以从该医药网站进行爬取，因为在该网页中所有的疾病的 id 是唯一的，按 id 抓取数据写入 txt 同样可以使用。具体情况如图 3-5 所示。

在本章中，假设字典已经存在，以基于字典的双向最大匹配算法对抓取的并发症描述信息进行分析和匹配，得到的并发症数据放到一个数组中，最后将得到的数组进行去重，取得最终的实体数据，方便后面实体的数据入库与图谱构建。

### 3.3.2　三元组的抽取

三元组作为一种图数据结构，知识图谱的最小单元是两个节点及它们之间的关系。在医药疾病领域，无疑是抽取两个实体之间的关系，由两个实体和其之间的关系构成三元组，构成知识图谱中的小单元。


图 3-5　疾病信息采集分析

在本章中，抓取的数据为半结构化数据和非结构化数据，对于半结构化数据很容易处理格式化的数据，非结构化数据也通过上述算法进行处理，得到相应的实体，进而很容易找出三元组的关系。这里举个例子，如颅骨纤维异常增生症中有一词条数据为：

{并发症：耳聋，如进行手术治疗，可能出现以下并发症：1.术后视力下降 与术中操作碰触视神经有关，多可逐渐恢复。2.脑脊液鼻漏术中做骨瓣或切除增厚的骨质时打开额窦或筛窦，术中又未做严密修补所致，如经保守治疗数周不能治愈或治愈后又再复发时，需重新手术修补}

根据词典双向最匹配算法后，就可能得到若干个实体：耳聋、视力下降、脑脊液鼻漏术、骨瓣、额窦、筛窦，进而就可以找出三元组：<颅骨纤维异常增生症，并发症，耳聋><颅骨纤维异常增生症，并发症，脑脊液鼻漏术><颅骨纤维异常增生症，并发症，视力下降>等，得到很多个知识图谱的小单元。

在实战案例中的部分关键代码如下。

```
1. relation_diss_bad_food =[]
2. relation_diss_good_food =[]
3. relation_diss_recom_food =[]
4. relation_commonddrug =[]              #疾病－通用药品关系
5. relation_diss_neopath =[]             #疾病并发关系
6. relation_diss_department =[]
7.
8. #就诊科室与疾病关联
9. if len(basicinfo["visit_department"]) ==1:
10.     relation_diss_department.append([basicinfo.get("disease_name"),
       basicinfo["visit_department"]])
11.  if len(basicinfo["visit_department"]) ==2:
12.    relation_diss_department.append(
13.        [basicinfo.get("disease_name")[1], basicinfo["visit_department"]
           [0]])
```

```
        relation_diss_department.append([basicinfo["visit_department"],
        basicinfo.get("disease_name")[1]])

#饮食与疾病关系
if "bad_food" in food:
    #忌吃与疾病关联
    bad_food=food['bad_food']
    for _ in bad_food:
        relation_diss_bad_food.append([basicinfo.get("disease_name"), _])
    total_food_list+=bad_food
    #宜吃与疾病关联
    good_food=food['good_food']
    for _ in good_food:
        relation_diss_good_food.append([basicinfo.get("disease_name"), _])
    total_food_list+=good_food
    #推荐与疾病关联
    recommand_food=food['recommend_food']
    for _ in recommand_food:
        relation_diss_recom_food.append([basicinfo.get("disease_name"), _])
    total_food_list+=recommand_food

#伴随着症状与疾病关联
if "disease_neopath" in disease_together:
    disease_neopath=disease_together["disease_neopath"]
    for _ in disease_neopath:
        relation_diss_neopath.append([basicinfo.get("disease_name"), _])
```

在该部分代码中，首先根据数据建立相应的关系数组，由于此处的数据比较清晰明了，没有再去运用深度学习的相关算法进行训练得出实体之间的关系，而是直接通过手动建立，然后把处理好的实体数据通过判断插入建立的关系数组中，然后将全部形成的三元组插入数组当中，形成数组元素，方便之后数据存储到数据库再进行知识图谱数据的构建。

### 3.3.3　数据存储

在以上工作都完成以后，接下来就是最关键的部分，将处理好的三元组存储到数据库当中，后续再以图谱的形式展示出来。

在本实战案例中，已形成了处理好的数据组，如下所示。

```
[['百日咳', '螃蟹'], ['百日咳', '海蟹'],……, ['苯中毒', '海虾']、['苯中毒', '海参(水浸)'], ['苯中毒', '辣椒(青、尖)']]、['百日咳', '肺炎'], ['百日咳', '支气管肺炎'], ['百日咳', '肺不张'], …… ['苯中毒', '再生障碍性贫血']]、[ ['苯中毒', ['急诊科']]]
```

下面通过建立的关系组合成三元组，写入 Neo4j 数据库，部分关键代码如下所示。

```
#创建节点
def create_node(self, label, nodes):
    node_matcer=NodeMatcher(self.graph)
    create_node_cnt=0
    for node in nodes:
        find_node=node_matcer.match(label, name=node).first()
        if find_node is None:
```

```
            node =Node(label, name=node)
            self.graph.create(node)
            create_node_cnt +=1
            print(f"create {create_node_cnt} nodes.")

#创建关联边方法
def create_rel(self, s_node, e_node, edges, rel_type, rel_name):
    create_rel_cnt =0
    array_edges_list =np.array(edges)
    unique_edges_list =np.unique(array_edges_list, axis=0)
    for edge in unique_edges_list:
        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)" %(
            s_node, e_node, p, q, rel_type, rel_name)
        try:
            self.graph.run(query)
            create_rel_cnt +=1
            print(rel_type, create_rel_cnt, all)
        except Exception as e:
            print(e)
    return
```

## 3.4 图谱可视化和知识应用

下面通过创建医药数据相关节点，创建医药数据的关系边，将三元组以知识图谱的形式展现出来，其展示结果如图 3-6 所示。

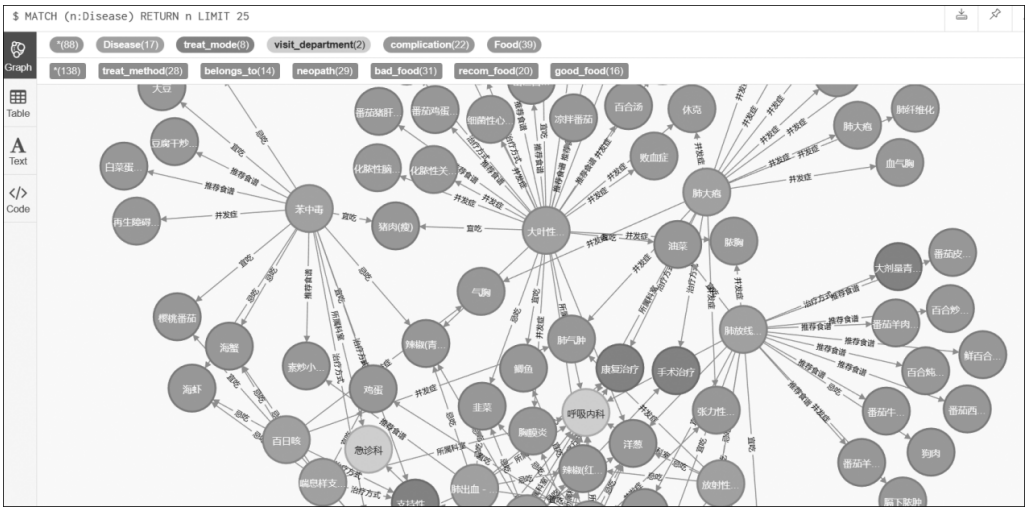


图 3-6 医药知识图谱情况

医药知识图谱可以将疾病主要信息、所属科室、常用药品、相关并发症、推荐膳食、忌吃食物、治疗方式等信息汇聚到一张图谱上，通过该知识图谱可以从大体的结构上了解各种医

药信息之间的联系。另外，医药信息本身复杂繁冗，通过图谱可以从海量的信息中提取出某一条清晰明了的医药信息线，从而从各个方面了解以疾病为中心的医药信息。

### 3.4.1　数据查询

如果需要查询某种疾病的医药信息，可以使用 Cypher 语句。以下是查询单个疾病的医药知识图谱之间的关系。

```
1. match (p:Disease {name:"百日咳"}) return p
```

执行后的结果如图 3-7 所示。


图 3-7　单个实体的数据查询

若是对实体某种关系的查询，也可以使用 Cypher 语句。比如查询的关系是忌口关系（bad_food），具体如下所示。

```
MATCH (p:Disease {name:"百日咳"})-[r:bad_food]->() RETURN p
```

运用该查询语句得到的结果如图 3-8 所示。

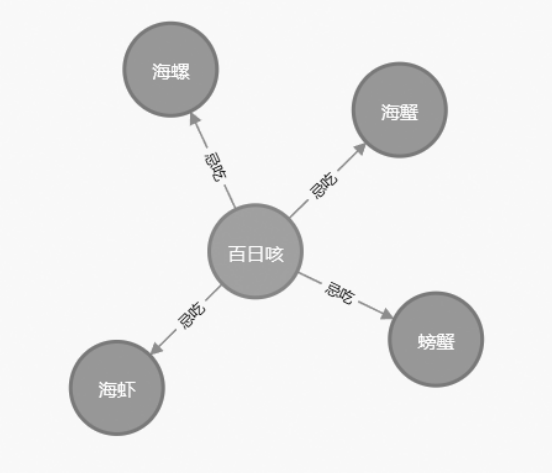


图 3-8　“百日咳”忌吃关系查询

查询到实体数据以后，可以对数据的信息进行展开，如图 3-9 所示。疾病实体的信息维度

包括基本详情、易感人群、治疗周期、治愈率等，可以通过数据库信息的钻取得到这些信息，其中还包括疾病的 URL 链接，通过该链接进入到该疾病的信息页，得到所有的详情信息。

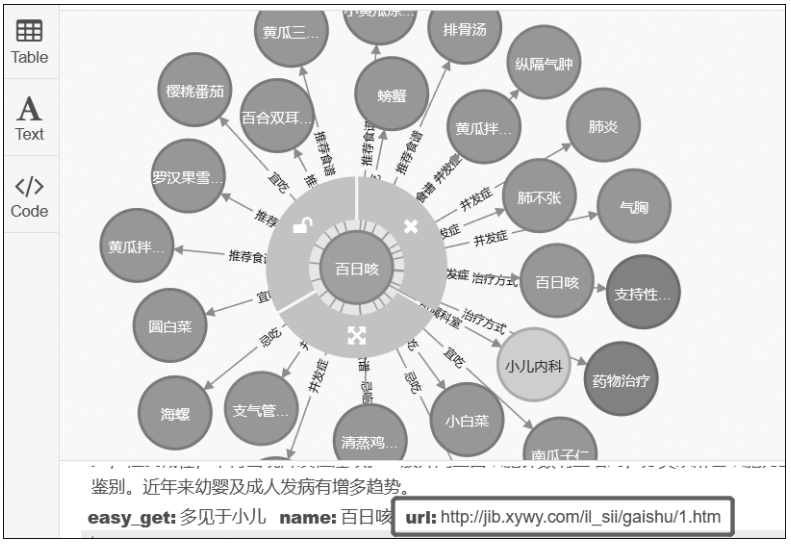


图 3-9 数据的钻取

同时也能通过数据库查看到其医疗信息，具体如下所示。

```
{
    "name": "百日咳",
    "cured_prob": "0.5%",
    "url": "http://jib.xywy.com/il_sii/gaishu/1.htm",
    "easy_get": "多见于小儿",
    "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。"
}
```

## 3.4.2 膳食维度分析

从膳食的角度，在已知某疾病的前提条件下，允许进食哪些食物，禁忌哪些食物，哪些病与这些食物存在联系，都可以通过一张图直观地了解到。

从图 3-10 可以看出，人们在患感冒时，应忌吃咸鱼、油条、白扁豆和猪油等食物。通过查询某疾病的忌吃关系，可向患者建议忌用的食物清单。

从图 3-11 来看，“芝麻”这一食物适合人们患感冒、气胸、鼾症、汞中毒等疾病时食用，从而可推出同一食物对于多种病的膳食医疗是有裨益的。因此，通过医药知识图谱，可以查询疾病与膳食之间的关系，为病人提供膳食推荐和制定膳食治疗方案。

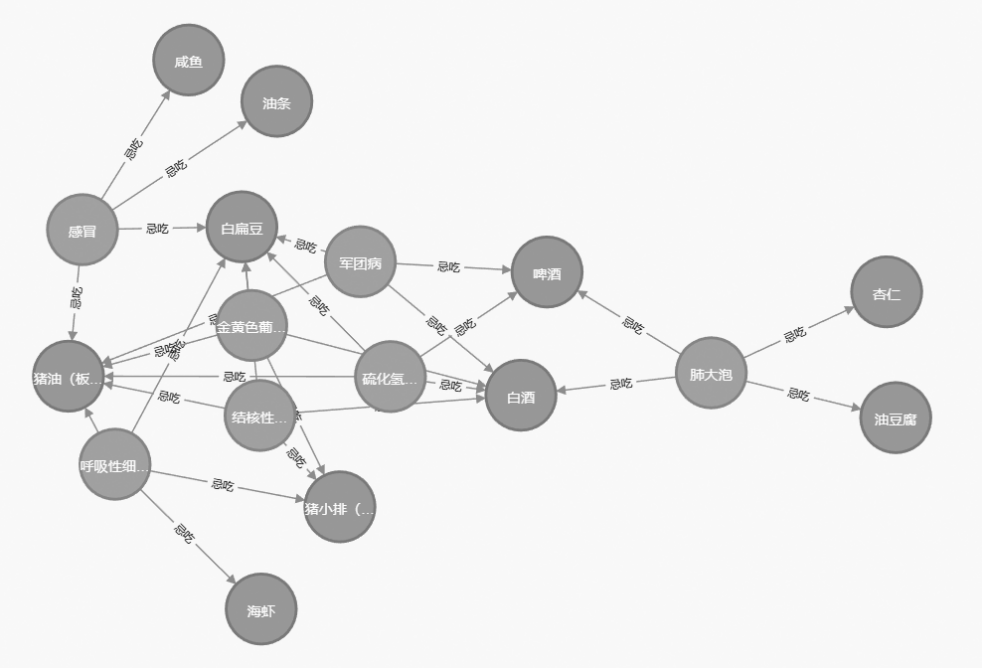


图 3-10　膳食维度分析—忌吃关系

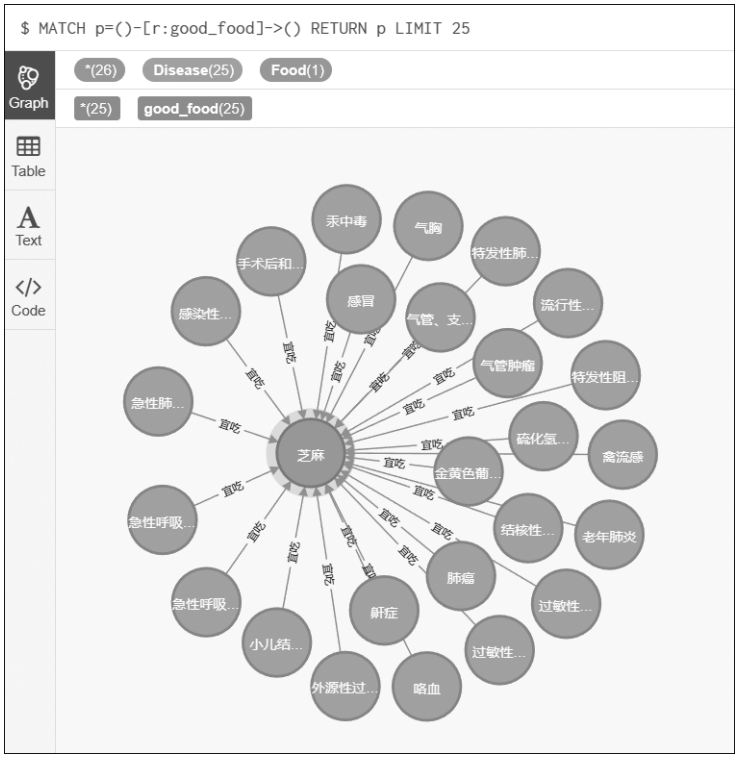


图 3-11　膳食维度分析—宜吃关系

### 3.4.3 用药维度分析

针对用药的维度分析，能够获得疾病与药品之间的关联关系，如图 3-12 所示。对于医生来说，对医药图谱可以从深层次的角度做总览性分析，将疾病与药品间的关系做一个参考；对患者来说，可以更清晰地了解自己的疾病情况，在某种程度上避免用错药。当然，由于这些数据来源于网络，真实性有待参考，不建议患者根据图谱内容盲目用药，而应去找专业医生对症下药。

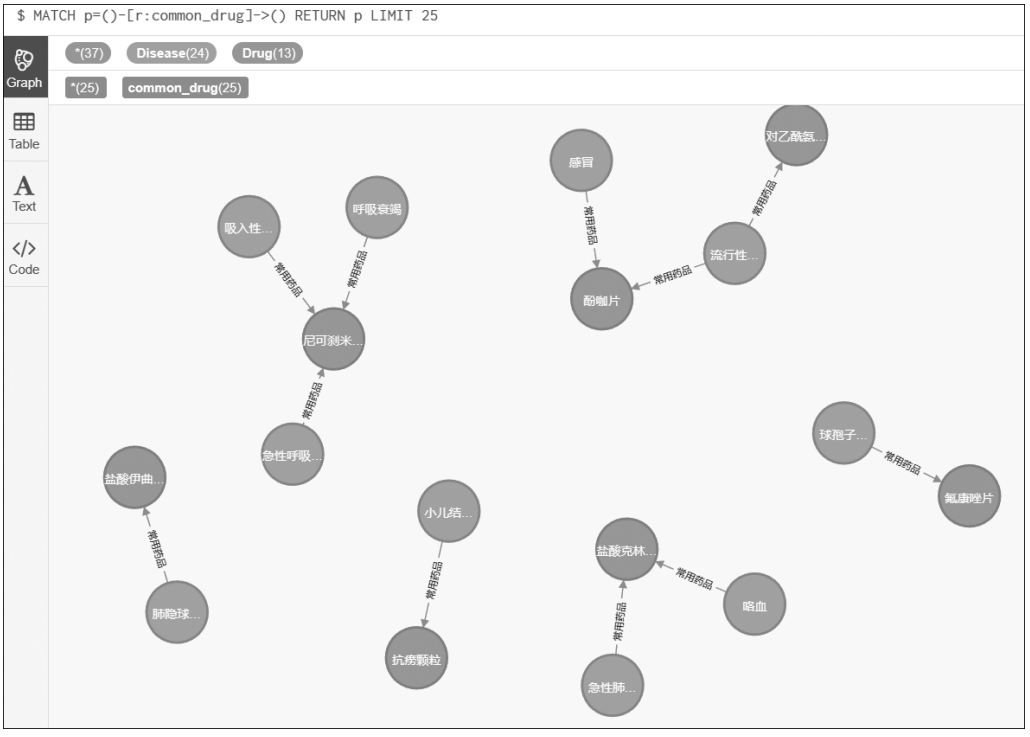


图 3-12 用药维度分析

## 3.5 小结和扩展

本章主要是针对医药疾病领域相关数据如何抓取、知识图谱如何构建等内容进行叙述。首先阐述了医药疾病知识图谱的相关理论，然后描述了如何构建知识图谱的框架，从 Web 资源的抓取到写入 Neo4j 数据库的整个流程做了介绍，接着对实体的抽取、三元组的抽取与构建，进行了具体的分析与阐述，并对相关代码进行了解释，然后说明了如何将数据写入数组，得到医疗领域的知识图谱实例，并进行结果展示和查询分析。

**思考题：**

（1）从医药疾病数据的特点来看，非常适合于智能问答场景。感兴趣的读者可自行设计问题模板，在学习过第 5 章技术后，尝试做一个简单的智能问答程序。

（2）扩宽查询范围，比如某疾病的就诊科室、相关检查、病因等。