

（2023-2024学年 第2学期）

**《知识工程》结课报告**

题 目 构建知识图谱模块实现

所在学院 自动化学院、人工智能学院

专 业 人工智能

年级班级 B210416

学 号 B21080526

姓 名 单家俊

授课教师 丁卉

# 引言或目标

**知识工程的背景知识**

知识工程作为一门致力于设计和构建智能系统的学科，其根源可追溯至上世纪80年代。随着计算机科学和人工智能领域的快速发展，知识的表示、管理和应用成为了研究的焦点。在这一背景下，知识图谱应运而生，成为知识工程的一个重要成果。知识图谱是一种图形化的知识表达形式，用于描述实体、概念、属性及其之间复杂的关系网络[1]。它的发展与互联网技术的进步密切相关。随着Web技术的飞速发展，如何有效地结构化和管理海量的在线信息成为了一个迫切需要解决的问题。

2002年，万维网联盟（W3C）提出了语义Web的概念，为Web文档和应用程序提供了一种结构化的表示方式[2]。这一概念的提出极大地推动了知识图谱的发展，为海量信息的智能化处理铺平了道路。

本体论在知识图谱的构建中扮演着关键角色。它提供了一个形式化的框架，用于定义概念、实体以及它们之间的关系这对于构建知识图谱至关重要。知识图谱是知识工程的一个重要成果，它以图形化的方式表达知识，使得知识的存储、检索和推理变得更加高效[3]。

随着大数据时代的到来和人工智能技术的进步，知识图谱在各个领域的应用日益广泛，从搜索引擎优化到智能问答系统，从推荐系统到智能决策支持，知识图谱正在改变我们与信息交互的方式，为智能化时代的知识管理和应用开辟了新的前景。因此，可以说，知识图谱不仅是知识工程的一个重要成果，更是推动信息技术进步、实现智能化应用的关键技术之一。随着人工智能技术的进一步发展和应用场景的不断拓展，知识图谱在未来将继续发挥重要作用，为我们理解和利用世界上的信息资源提供强大的支持和基础。

**挑战**

自谷歌提出知识图谱概念以来，这项技术一直备受广泛关注。随着深度学习、自然语言处理等相关领域的发展，知识图谱的研究热度持续攀升。然而，知识图谱的发展至今仍面临诸多挑战，特别是在知识融合、知识推理等关键技术的实现及应用方面。

知识融合技术作为知识图谱的核心技术之一，其主要任务是有效地将新获得的知识整合到已有的知识图谱中，确保知识的准确性和完整性。然而，面对当前的挑战，首先是如何提升知识评估的能力，尤其是针对动态知识的评估手段仍然相对缺乏。其次，由于自然语言的特殊性，如何准确处理同义异名实体对齐和同名异义实体消歧，以避免知识图谱中的冗余和缺失，也是亟待解决的问题。此外，针对多语言环境下的实体对齐和消歧，其准确率更是面临巨大挑战。

知识推理技术则致力于通过已知的知识推导新知识，以丰富知识图谱的内容和应用。然而，当前的挑战之一是如何有效处理多元关系的推理问题，因为传统方法往往通过拆分多元关系来简化推理过程，但这往往会导致结构信息的丢失。此外，现有的推理模型通常依赖于大量高质量的训练数据，如何在少样本或零样本情况下实现推理仍然是一个尚未克服的难题。另外，知识图谱中的知识具有时空动态性，如何有效利用动态约束信息完成动态推理也是当前研究的重要方向。

最后，知识的表达、存储与查询是贯穿整个知识图谱应用的基础问题。当前，知识图谱的构建往往依赖于人工的参与，成本较高且效率有限，大多数研究工作主要针对知识图谱的半自动构建[4]，自动化构建高质量知识图谱仍是一个重要挑战。此外，如何利用知识图谱中的知识指导机器学习过程，提高数据利用效率，以及如何利用符号化知识图谱解释深度学习等黑盒模型，增强其可解释性，也是当前亟需解决的问题之一。

**动机**

知识工程发展的动机主要源于人类对于处理和利用大量复杂知识的需求，特别是在信息技术迅速发展的背景下。

1. **信息爆炸和信息过载**：随着互联网和数字化技术的普及，我们生活在一个信息爆炸的时代。每天产生的数据和信息呈指数级增长，如何有效地组织、管理和利用这些信息成为了一个迫切的问题。知识工程致力于通过各种技术手段，如知识图谱和本体论，将海量信息结构化地表示和管理，使得信息能够更加智能地被理解和利用。

2. **知识的复杂性和不确定性**：现实世界中的知识通常是复杂且不完全的。人类的知识包括事实、规则、概念、经验等多种形式，而且这些知识之间可能存在不确定性和模糊性。知识工程通过建立形式化的知识表示和推理机制，试图解决知识的复杂性和不确定性，使得系统能够更好地处理这些知识。

3. **智能化应用的需求**：随着人工智能和机器学习等技术的发展，智能化应用成为了信息技术发展的主流方向。这些应用需要大量的结构化知识作为基础，例如语义理解、推理、智能搜索、自动问答等。知识工程为这些智能化应用提供了基础设施和技术支持，使得它们能够更加智能地处理和理解人类语言和行为。

4. **跨学科交叉与整合**：知识工程是一个跨学科的领域，涉及计算机科学、人工智能、认知科学、语言学等多个学科的交叉与整合。通过整合这些学科的理论和方法，知识工程旨在探索和发展更加强大和智能的信息处理技术，从而推动人类社会和科技的进步。

知识工程的动机在于解决信息过载、处理知识的复杂性和不确定性、支持智能化应用的发展，以及促进跨学科的交叉与整合。通过这些努力，我们希望能够更好地理解和利用人类的知识，为社会和经济发展提供更加智能和有效的解决方案。

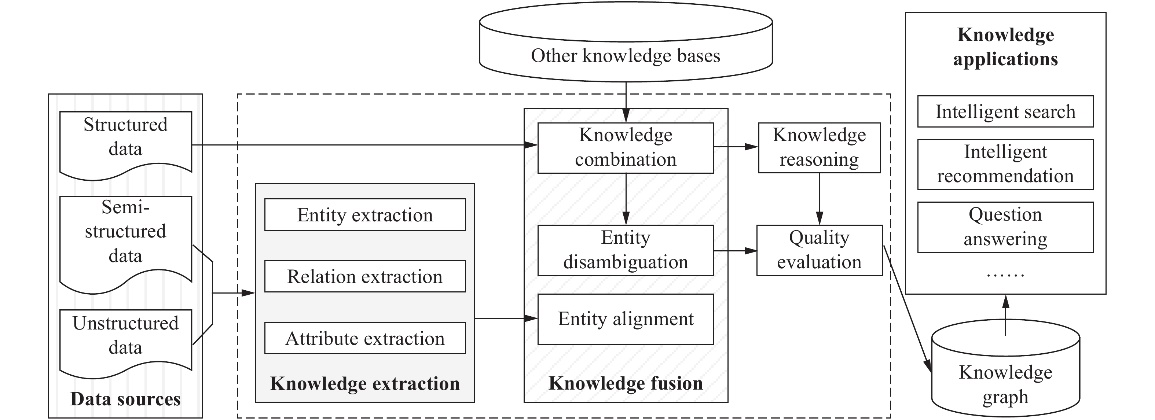
# 方法

**模型**

知识图谱的数据模型主要定义了实体、关系和属性之间的结构和语义含义。**数据模型**是知识图谱的基础，采用RDF（资源描述框架）和OWL（Web本体语言）等标准。RDF模型以三元组形式描述实体（如人、地点、概念）、关系（描述实体之间的连接）和属性（描述实体的特征）。OWL模型在此基础上扩展，引入类、属性和约束的概念，提供更强大的语义表达能力，支持复杂的推理和语义理解。

**基本原理**

对于知识图谱而言，首要的问题是如何从海量的数据中提取有用的信息，并有效地表示和储存这些信息，这就是知识抽取与表示技术的核心内容。知识抽取与表示，也可以称为信息抽取，其目标是从数据源中抽取特定类型的信息，如实体、关系和属性，并通过适当的形式表达和储存这些信息。在知识图谱的场景中，通常采用RDF来描述知识，形式上将有效信息表示为（主语，谓语，宾语）三元组的结构。根据抽取的信息类型不同，知识抽取可以分为实体抽取、关系抽取和属性抽取。知识工程的基本原理可用图1表示。



图一 知识工程的基本原理

实体抽取，又称为命名实体识别，是从文本样本中识别出特定类型的命名实体的过程。实体是知识图谱最基本的元素，实体抽取的完整性、准确率、召回率将直接影响知识图谱的质量[5]。在知识图谱中，实体是最基本的元素，实体抽取的质量直接影响知识图谱的建设和应用效果。文献[6]将实体抽取的方法归纳为三种主要类型：**基于规则与词典的方法、基于统计机器学习的方法（**Liu等[7]将K近邻算法和线性条件随机场模型结合来识别实体**）**和**面向开放域的抽取方法**。

通过实体抽取获取的实体通常是离散且缺乏关联性的。为了建立实体之间的语义链接，关系抽取技术成为必要的步骤。关系抽取主要分为三种主要方法：**基于模板的关系抽取**、**基于监督学习的关系抽取**、**基于半监督或无监督学习的关系抽取**。这些关系抽取方法各有其优缺点，选择合适的方法取决于具体的应用场景、数据特点和资源限制。有效的关系抽取技术能够帮助构建和维护知识图谱，从而支持更复杂的数据分析和智能应用。

属性抽取的目标是补全实体信息，通过从样本源中获取实体属性信息或属性值。实体属性可以看作是属性值与实体间的一种关系，因而可以通过关系抽取的解决思路来获得。Wu与Weld[8]利用百科类网站的半结构化数据，训练抽取模型，之后将抽取模型应用在非结构化数据中抽取属性。

**知识图谱设计**

知识图谱的架构设计包括如何组织、存储和管理数据，以及支持知识获取和应用开发的技术选择和实施方案。这包括数据抽取和整合技术，从不同来源获取并整合数据；存储和查询策略，如图数据库和查询语言的应用；以及推理和语义理解能力的实现，用于从数据中推导新的知识和支持智能应用的开发。

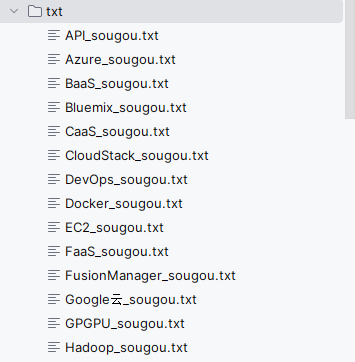
# 实验和结果

**1．数据获取**

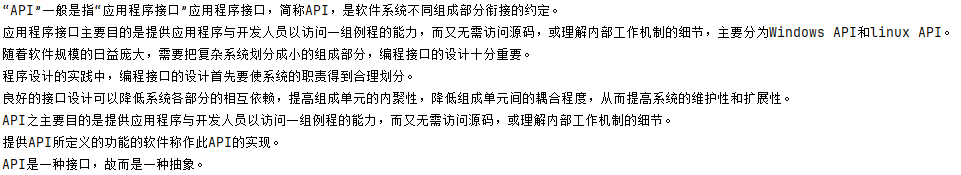
构建知识图谱的前提是数据准备，我打算通过网络爬虫技术爬取搜狗百科网页文本信息来构建一个简单的数据集，以供后续使用Neo4j软件对知识图谱进行可视化展示。

网络爬虫模块代码：**import** json  
**import** requests, threading, re  
**from** lxml **import** etree  
  
header = {  
 **"user-agent"**: **"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 "  
 "Safari/537.36 Edg/120.0.0.0"**,  
 **'Accept'**: **'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8,'  
 'application/signed-exchange;v=b3;q=0.7'**,  
 **'Accept-Encoding'**: **'gzip, deflate, br'**,  
 **'Accept-Language'**: **'zh-CN,zh-TW;q=0.9,zh;q=0.8,en;q=0.7,en-GB;q=0.6,en-US;q=0.5'**,  
 **'Connection'**: **'keep-alive'**}  
 **def** get\_page\_url\_tail(key: str):  
 url = **"https://baike.sogou.com/bapi/searchBarEnter?searchText="** res = requests.get(**"%s%s"** % (url, key), headers=header)  
 url\_tail = res.content.decode(**'utf-8'**)  
 *# url\_tail为网址后面的小尾巴，用于构成完整的网址* **return** url\_tail  
  
**def** get\_page(key: str):  
 url = **"https://baike.sogou.com"** url\_tail = get\_page\_url\_tail(key)  
 **if "sp=0" in** url\_tail:  
 print(key + **"未找到"**)  
 **return** []  
 page\_url = **"%s%s"** % (url, url\_tail)  
 res = requests.get(page\_url, headers=header)  
 html = etree.HTML(res.content)  
 more\_info = html.xpath(**'//script/text()'**)pattern = re.compile(**r"window.lemmaData=(.\*?);$"**, re.MULTILINE | re.DOTALL)data\_str = pattern.search(more\_info[0]).group(1)  
  
 *# print(data\_str)* json1 = list(json.loads(data\_str)[**'paragraph'**].values())[0]content = **""  
 for** i **in** json1:  
 **try**:  
 content = **"%s%s"** % (content, i[**'content'**])  
 **except**:  
 **pass** html2 = etree.HTML(content)  
 detail1 = html2.xpath(**'//text()'**)  
 detail2 = html.xpath(**'//p/text()'**)  
 **del** detail2[-1]  
 full\_content = **""**.join(detail2 + detail1)  
 full\_content = re.sub(**"\（.\*?\）"**, **''**, full\_content)  
 full\_content = re.sub(**"\(.\*?\)"**, **''**, full\_content)result = full\_content.split(**"。"**)**return** result  
  
*# 获取的信息被写入文本文件，每个句子后面跟着一个句号***def** write\_txt(search\_list: list):  
 **for** key **in** search\_list:  
 result = get\_page(key)  
 **with** open(**"txt/%s\_sougou.txt"** % key, **'w+'**, encoding=**'utf-8'**) **as** f:  
 **for** i **in** result:  
 f.write(i + **"。\n"**)  
  
**def** run():  
 print(**"开始爬取搜狗百科"**)  
 **with** open(**"filter/newwords.txt"**, **"r"**, encoding=**"UTF-8"**) **as** f, open(**'filter/equal.csv'**, **'r'**, encoding=**"UTF-8"**) **as** e:  
 search\_dict = []  
 f\_content = f.readlines()  
 **for** line **in** f\_content:  
 line = line.strip()  
 search\_dict.append(line)e\_content = e.readlines()  
 words = []  
 **for** line **in** e\_content:  
 line = line.strip()  
 word = line.split(**','**)  
 **if '' in** word:  
 word.remove(**''**)  
 words.extend(word)search\_dict.extend(words)search\_dict1 = search\_dict[0:int(len(search\_dict) / 4)]  
 search\_dict2 = search\_dict[int(len(search\_dict) / 4):int(len(search\_dict) / 2)]  
 search\_dict3 = search\_dict[int(len(search\_dict) / 2):int(len(search\_dict) \* 3 / 4)]  
 search\_dict4 = search\_dict[int(len(search\_dict) \* 3 / 4):]  
  
 thread1 = threading.Thread(name=**'t1'**, target=write\_txt, args=(search\_dict1,))  
 thread2 = threading.Thread(name=**'t2'**, target=write\_txt, args=(search\_dict2,))  
 thread3 = threading.Thread(name=**'t3'**, target=write\_txt, args=(search\_dict3,))  
 thread4 = threading.Thread(name=**'t4'**, target=write\_txt, args=(search\_dict4,))  
  
 thread1.start()  
 thread2.start()  
 thread3.start()  
 thread4.start()  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 run()

爬取结果：



图二 爬取结果文件目录



图三 API\_sougou.txt文件中部分内容

**2.数据清洗：**

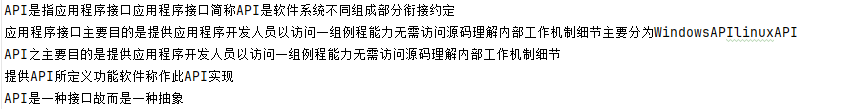
对于爬取过后的数据，句子中可能含有一些停用词等，我们需要将这些停用词去除。此外，该句子可能不含有我们想要的关键词，我们还需要将这样的句子删去。

去除停用词模块代码：

*# -\*- coding: utf-8 -\*-***import** csv  
**from** snownlp **import** SnowNLP  
**import** jieba.posseg **as** pseg  
**import** re  
**import** os  
**import** jieba  
**def** readFolder(url):path = url *# 文件夹目录* files = os.listdir(path) *# 得到文件夹下的所有文件名称* txts = []  
 **for** file **in** files: *# 遍历文件夹* **if** os.path.splitext(file)[1] == **'.txt'**:  
 position = path + **'\\'** + file txts.append(position)  
 **return** txts  
 **def** preTrain(input, filepath): *# 寡去除特殊符号* input = re.sub(**r"[0-9\s+\.\!\/\_,\-.$%^\*()?;；：:【】《》+\"\'+\[+\]]+|[+——！，;:。？、“”~@#￥%……&\*（）]+"**, **""**, str(input))  
 **if** input == **''**:  
 **return ''** *# 繁体转简体* input = SnowNLP(input).han  
  
 *# 单一停用词词典* filepath = **"E:/数据获取/代码/filter/data/stopwords2.txt"** stopword = [line.strip() **for** line **in** open(filepath, **'r'**, encoding=**'utf8'**).readlines()] *# 以行的形式读取停用词表，同时转换为列表* cur = input[0]  
 output = []  
 word\_list = list(jieba.cut(input, cut\_all=**False**))  
 **for** word **in** word\_list:  
 **if** word **not in** stopword:  
 output.append(word)  
 **return** output  
 *# 删除停用词***def** delStopWords(input, path):  
 print(path)  
 file = os.path.normpath(path + **'stopwords2.txt'**)  
 **return** preTrain(input, file)  
  
*# 删除模糊词***def** delFuzzyWord(input, path):  
 file = path + **'fuzzyword.txt'  
 return** preTrain(input, file)  
  
**def** run():  
 print(**"开始数据清洗"**)url = os.path.normpath(\_\_file\_\_ + **'\\..\\data\\'**)  
  
 path = **r"E:\数据获取\代码\txt"** *# 文件夹目录* files = os.listdir(path) texts = []  
 **for** file **in** files: position = path + **'\\'** + file  *# print(position)* **with** open(position, **"r"**, encoding=**'UTF-8'**) **as** f: *# 打开文件* **for** line **in** f:  
 texts.append(line.strip())  
  
 **with** open(**"brevity.txt"**, **"w"**, newline=**''**, encoding=**'UTF-8'**) **as** f:  
 **for** text **in** texts:text = **''**.join(delStopWords(text, path=url)) *# 停用词* f.writelines(str(text + **'\n'**)) **if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 run()

删去不含关键词句子模块代码：**import** csv  
**def** run():  
 *# 定义关键词列表* **with** open(**"E:/数据获取/代码/filter/newwords.txt"**, **"r"**, encoding=**"UTF-8"**) **as** f:  
 keywords = []  
 **for** line **in** f:  
 keywords.append(line.strip())  
  
 *# 关键词中加入同义词* **with** open(**'E:/数据获取/代码/filter/equal.csv'**, **'r'**, encoding=**"UTF-8"**) **as** f:  
 words = [line.strip(**'\n'**).split(**','**) **for** line **in** f]  
 **for** w **in** words:  
 **for** i **in** w[1:]:  
 **if** i != **''**:  
 keywords.append(i)  
 print(keywords)  
  
 *# 定义要过滤的句子* **with** open(**"brevity.txt"**, **"r"**, encoding=**"UTF-8"**) **as** f:  
 sentences = []  
 **for** line **in** f:  
 sentences.append(line.strip())  
  
 **with** open(**"brevity.txt"**, **"w"**, newline=**''**, encoding=**'UTF-8'**) **as** f:**for** sentence **in** sentences:  
 **for** keyword **in** keywords:  
 **if** keyword **in** sentence:f.writelines(str(sentence + **'\n'**))**break  
 else**:**pass  
  
if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 run()

数据清洗后的结果：



图四 数据清洗后的部分结果

**主体抽取和关系抽取：**

为了构建知识图谱，需要从句子中抽取（主，谓，宾）三元组，此处我想用深度学习技术进行处理，但是能力有限，最终选择直接调用哈工大开源的**ltp**进行数据处理。

主体抽取和关系抽取模块代码：

*# -\*- coding: utf-8 -\*-* **import** os  
**import** pprint  
**import** re  
**import** csv  
**import** threading  
**from** multiprocessing.pool **import** ThreadPool  
**from** ltp **import** LTP  
**class** LtpParser:  
 **def** \_\_init\_\_(self):  
 self.ltp = LTP(**"small3"**)  
 self.ltp.init\_dict(path=**"filter/newwords.txt"**, max\_window=8)**'''语义角色标注'''  
 def** format\_labelrole(self, hidden):  
 roles = self.ltp.srl(hidden, keep\_empty=**False**)  
 roles\_dict = {}  
 **for** role **in** roles[0]:  
 roles\_dict[role[0]] = {arg[0]: [arg[0], arg[1], arg[2]] **for** arg **in** role[1]}  
 **return** roles\_dict  
  
 **'''句法分析---为句子中的每个词语维护一个保存句法依存儿子节点的字典'''  
 def** build\_parse\_child\_dict(self, words, postags, arcs):  
 child\_dict\_list = []  
 format\_parse\_list = []  
 **for** index **in** range(len(words)):  
 child\_dict = dict()  
 **for** arc\_index **in** range(len(arcs)):  
 **if** arcs[arc\_index][1] == index + 1: *# arcs的索引从1开始* **if** arcs[arc\_index][2] **in** child\_dict:  
 child\_dict[arcs[arc\_index][2]].append(arc\_index)  
 **else**:  
 child\_dict[arcs[arc\_index][2]] = []  
 child\_dict[arcs[arc\_index][2]].append(arc\_index)  
 child\_dict\_list.append(child\_dict)  
 rely\_id = [arc[1] **for** arc **in** arcs] *# 提取依存父节点id* relation = [arc[2] **for** arc **in** arcs] *# 提取依存关系* heads = [**'Root' if** id == 0 **else** words[id - 1] **for** id **in** rely\_id] *# 匹配依存父节点词语* **for** i **in** range(len(words)):a = [relation[i], words[i], i, postags[i], heads[i], rely\_id[i] - 1, postags[rely\_id[i] - 1]]  
 format\_parse\_list.append(a)  
 **return** child\_dict\_list, format\_parse\_list **def** parser\_main(self, sentence):  
 words, hidden = self.ltp.seg([sentence])  
 postags = self.ltp.pos(hidden)  
 arcs = self.ltp.dep(hidden)  
 words, postags, arcs = words[0], postags[0], arcs[0]child\_dict\_list, format\_parse\_list = self.build\_parse\_child\_dict(words, postags, arcs)  
 roles\_dict = self.format\_labelrole(hidden)  
 **return** words, postags, child\_dict\_list, roles\_dict, format\_parse\_list

**class** TripleExtraction():  
 **def** \_\_init\_\_(self):  
 self.parser = LtpParser()  
  
 **'''文章分句处理, 切分长句，冒号，分号，感叹号等做切分标识'''  
 def** split\_sents(self, content):  
 **return** self.parser.ltp.sent\_split([content])  
  
 **'''利用语义角色标注,直接获取主谓宾三元组,基于A0,A1,A2'''  
 def** ruler1(self, words, postags, roles\_dict, role\_index):  
 v = words[role\_index]  
 role\_info = roles\_dict[role\_index]  
 **if 'A0' in** role\_info.keys() **and 'A1' in** role\_info.keys():  
 s = **''**.join([words[word\_index] **for** word\_index **in** range(role\_info[**'A0'**][1], role\_info[**'A0'**][2] + 1) **if** postags[word\_index][0] **not in** [**'w'**, **'u'**, **'x'**] **and** words[word\_index]])  
 o = **''**.join([words[word\_index] **for** word\_index **in** range(role\_info[**'A1'**][1], role\_info[**'A1'**][2] + 1) **if** postags[word\_index][0] **not in** [**'w'**, **'u'**, **'x'**] **and** words[word\_index]])  
 **if** s **and** o:  
 **return '1'**, [s, v, o]  
 **return '4'**, [] **def** ruler2(self, words, postags, child\_dict\_list, arcs, roles\_dict):  
 svos = []  
 **for** index **in** range(len(postags)):tmp = 1  
 *# 先借助语义角色标注的结果，进行三元组抽取* **if** index **in** roles\_dict:  
 flag, triple = self.ruler1(words, postags, roles\_dict, index)  
 **if** flag == **'1'**:  
 svos.append(triple)  
 tmp = 0  
 **if** tmp == 1:  
 *# 如果语义角色标记为空，则使用依存句法进行抽取* **if** postags[index]:  
 *# 抽取以谓词为中心的事实三元组* child\_dict = child\_dict\_list[index]  
 *# 主谓宾* **if 'SBV' in** child\_dict **and 'VOB' in** child\_dict:  
 r = words[index]  
 e1 = self.complete\_e(words, postags, child\_dict\_list, child\_dict[**'SBV'**][0])  
 e2 = self.complete\_e(words, postags, child\_dict\_list, child\_dict[**'VOB'**][0])  
 svos.append([e1, r, e2])  
  
 *# 定语后置，动宾关系* relation = arcs[index][0]  
 head = arcs[index][2]  
 **if** relation == **'ATT'**:  
 **if 'VOB' in** child\_dict:  
 e1 = self.complete\_e(words, postags, child\_dict\_list, head - 1)  
 r = words[index]  
 e2 = self.complete\_e(words, postags, child\_dict\_list, child\_dict[**'VOB'**][0])  
 temp\_string = r + e2  
 **if** temp\_string == e1[:len(temp\_string)]:  
 e1 = e1[len(temp\_string):]  
 **if** temp\_string **not in** e1:  
 svos.append([e1, r, e2])  
 *# 含有介宾关系的主谓动补关系* **if 'SBV' in** child\_dict **and 'CMP' in** child\_dict:  
 e1 = self.complete\_e(words, postags, child\_dict\_list, child\_dict[**'SBV'**][0])  
 cmp\_index = child\_dict[**'CMP'**][0]  
 r = words[index] + words[cmp\_index]  
 **if 'POB' in** child\_dict\_list[cmp\_index]:  
 e2 = self.complete\_e(words, postags, child\_dict\_list, child\_dict\_list[cmp\_index][**'POB'**][0])  
 svos.append([e1, r, e2])  
 **return** svos  
  
 **'''对找出的主语或者宾语进行扩展'''  
 def** complete\_e(self, words, postags, child\_dict\_list, word\_index, deep=0):  
 deep += 1  
 **if** deep == 3:  
 **return ""** child\_dict = child\_dict\_list[word\_index]  
 prefix = **''  
 if 'ATT' in** child\_dict:  
 **for** i **in** range(len(child\_dict[**'ATT'**])):  
 prefix += self.complete\_e(words, postags, child\_dict\_list, child\_dict[**'ATT'**][i], deep)  
 postfix = **''  
 if** postags[word\_index] == **'v'**:  
 **if 'VOB' in** child\_dict:  
 postfix += self.complete\_e(words, postags, child\_dict\_list, child\_dict[**'VOB'**][0], deep)  
 **if 'SBV' in** child\_dict:  
 prefix = self.complete\_e(words, postags, child\_dict\_list, child\_dict[**'SBV'**][0], deep) + prefix  
 **return** prefix + words[word\_index] + postfix **def** triples\_main(self, content):  
 sentences = self.split\_sents(content)  
 sentences = [sent **for** sent **in** sentences **if** len(sent.strip()) > 5]  
 svos = []  
 **for** sentence **in** sentences:  
 words, postags, child\_dict\_list, roles\_dict, arcs = self.parser.parser\_main(sentence) svo = self.ruler2(words, postags, child\_dict\_list, arcs, roles\_dict)  
 svos += svo  
 **return** svos  
  
**def** run():  
 path = **r"E:\数据获取\代码\txt"** files = os.listdir(path) **with** open(**"filter/brevity.txt"**, **"r"**, encoding=**"UTF-8"**) **as** f:  
 texts = []  
 **for** line **in** f:  
 texts.append(line.strip())  
  
 extractor = TripleExtraction()  
 **with** open(**"filter/newwords.txt"**, **"r"**, encoding=**"UTF-8"**) **as** f:  
 keywords = []  
 **for** line **in** f:  
 keywords.append(line.strip())  
 print(keywords)  
  
 **with** open(**'filter/equal.csv'**, **'r'**, encoding=**"UTF-8"**) **as** f:  
 equal = [line.strip(**'\n'**).split(**','**) **for** line **in** f]  
 should\_equal = []  
 **for** w **in** equal:  
 **for** i **in** w[1:]:  
 **if** i != **''**:  
 should\_equal.append(i)  
  
 **with** open(**"result.csv"**, **"w"**, newline=**''**, encoding=**"UTF-8"**) **as** f, open(**"result\_to\_neo4j.csv"**, **"w"**, newline=**''**,  
 encoding=**"UTF-8"**) **as** f\_neo:  
 writer = csv.writer(f)  
 writer\_neo = csv.writer(f\_neo)  
 **for** text **in** texts:  
 res = extractor.triples\_main(text)  
 print(text)  
 **for** r **in** res:  
 *# 简单的知识融合，中英文替换* **for** word **in** should\_equal:  
 **if** word **in** r[0] **or** word **in** r[2]: **for** w **in** equal:  
 **for** i **in** w[1:]:  
 **if** r[0] == i:  
 r[0] = w[0]  
 **if** r[2] == i:  
 r[2] = w[0]  
 *# 端点包含字典里的词汇则归结于字典中的词汇* m1 = 0  
 m2 = 0  
 word\_now1 = **''** word\_now2 = **''  
 for** keyword **in** keywords:  
 **if** keyword **in** r[0]:  
 m1 += 1  
 word\_now1 = keyword  
 **if** keyword **in** r[2]:  
 m2 += 1  
 word\_now2 = keyword  
 **if** m1 == 1:  
 r[0] = word\_now1  
 **if** m2 == 1:  
 r[2] = word\_now2  
 writer.writerow(r)  
 **if** r[0] **in** keywords **and** r[2] **in** keywords **and** r[0] != r[2]:  
 writer\_neo.writerow(r)  
 pprint.pprint(res)  
 print(**"---------"** \* 5)  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 run()

进行主体抽取和关系抽取后的结果：



图五 部分（主，谓，宾）三元组

**Neo4j图数据库展示**

导入模块代码：

LOAD CSV FROM 'file:///result\_to\_neo4j.csv' AS line

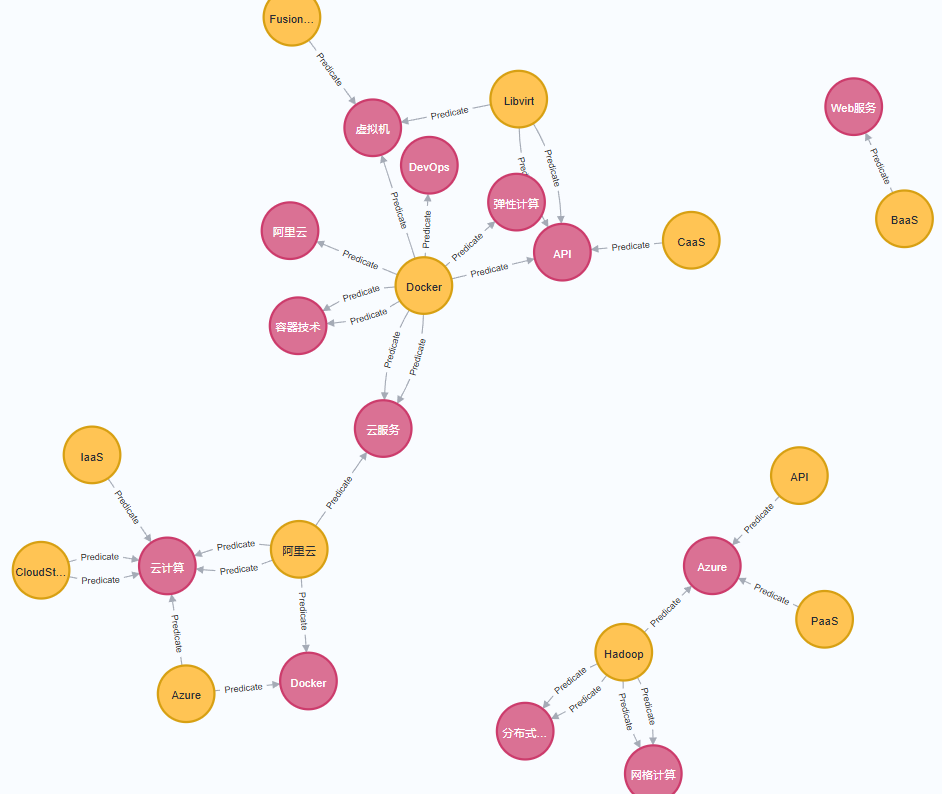
MERGE (n:Subject {name: line[0]})

MERGE (m:Object {name: line[2]})

CREATE (n)-[r:Predicate]->(m)

SET r.name = line[1];

Neo4j结果展示：



图六 Neo4j展示

# 结论

知识图谱旨在模仿人类的认知方式，构建机器的知识库，是实现机器认知智能的关键技术，同时也是大数据时代中利用大数据的核心技术。本门课程从知识图谱构建过程中的关键技术出发，简要探讨了知识的抽取与表示，重点分析了知识融合和知识推理技术的研究成果。

目前, 已经涌现出一大批知识图谱, 其中具有代表性的有KnowItAll[9]、DBpedia[10]、Magi[11]等.这些知识图谱从大量数据资源中抽取、组织和管理知识, 希望为用户提供能够读懂用户需求的智能服务, 例如理解搜索的语义, 提供更精准的搜索答案。尽管已有诸多研究成果，但其实用性仍然不强，距离知识图谱成为机器大脑知识库，实现机器认知智能的终极目标仍有较大差距。

未来的研究中，基于网络数据自动构建的知识图谱将成为主流。因此，需要进一步提高知识抽取、知识融合和知识推理技术的准确性，以确保获取知识的质量；同时提高这些技术的效率，以应对大规模数据量级的实用性挑战。尽管知识图谱已在公安情报分析、反金融欺诈等实际应用中开始发挥作用，但其巨大潜力仍待进一步挖掘。如何将知识图谱技术应用于生活的各个方面，也是未来的主要研究方向之一。

此外，目前存在的大量知识图谱在结构或语言上存在差异，这增加了知识图谱应用的难度。因此，制定行业规范，整合各个知识图谱，构建通用知识图谱，也是未来知识图谱研究的重要方向。

# 参考文献

1. 杨玉基,许斌,胡家威,等.一种准确而高效的领域知识图谱构建方法.软件学报, 2018, 029(010):2931-2947.
2. Berners-Lee T, Handler J, Lassila O. The Semantic Web. Scientific American, 2013, 284(5):34-43.
3. Gruber T R. A translation approach to portable ontology specifications[J]. Knowledge acquisition, 1993, 5(2): 199-220.
4. Ji S X, Pan S R, Cambria E, et al. A survey on knowledge graphs: representation, acquisition and applications[J/OL]. *arXiv preprint* (2020-02-02)[2020-02-28].
5. 徐增林, 盛泳潘, 贺丽荣, 等. 知识图谱技术综述. 电子科技大学学报, 2016, 45(4):589
6. 赵军, 刘康, 周光有, 等. 开放式文本信息抽取. 中文信息学报, 2011, 25(6):98
7. Liu X H, Zhang S D, Wei F R, et al. Recognizing named entities in tweets // *Proceedings of the* 49*th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume* 1. *Association for Computational Linguistics*. Portland, 2011: 359
8. Wu F, Weld D S. Autonomously semantifying Wikipedia // *Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management*. Lisbon, 2007: 41
9. Etzioni O, Cafarella M, Downey D, Kok S, Popescu AM, Shaked T, Soderland S, Weld DS, Yates A. Web-Scale information extraction in knowitall (preliminary results). In: Proc. of the 13th Int'l Conf. on World Wide Web. New York: ACM Press, 2004. 100-110.
10. Auer S, Bizer C, Kobilarov G, Lehmann J, Cyganiak R, Ives Z. DBpedia: A nucleus for a Web of open data. In: Proc. of the 6th Int'l Semantic Web Conf. Berlin, Heidelberg: Springer-Verlag, 2007. 722-735.
11. Ji Y C. Welcome to Magi[EB/OL]. *Peak Labs* (2019-08-20)[2020-02-28].