**红楼梦人物关系知识图谱**

付冰洋 刘一辰 何谦佩 施含容

1. **项目流程**

本项目计划构建《红楼梦》中的各人物以及他们之间的人物关系的知识图谱。主要包括四部分工作：（1）数据获取与预处理；（2）命名实体识别；（3）实体关系抽取；（4）可视化与知识问答。

1. **数据获取与预处理**

构建知识图谱首先需要收集项目所需的文本数据，从而进行实体（人物）和关系（人物关系）的识别抽取。本项目主要从《红楼梦人物列表》维基百科网站[[1]](#footnote-1)中提取红楼梦各个人物实体以及人物之间的关系,并从百度百科[[2]](#footnote-2)中爬取得到红楼梦各角色的介绍。

**2.1实体及关系数据获取**

我们选择《红楼梦人物列表》维基百科网站中的表格数据文本作为数据爬取的目标。如图所示，我们提取的到表格中的姓名数据作为实体，提取得到表格中的人物简介，并对半结构化数据进行分析，从中抽取出人物关系三元组。最终我们可以得到包含两个实体一个关系的三元组格式，作为后续深度关系抽取技术结果的补充和比对。



**2.1.1 网站表格数据获取**

红楼梦人物列表维基百科网站中提供了红楼梦中各人物实体的名称以及人物简介，我们首先从该网站中爬取表格数据。表格数据为结构化的数据，对应于维基百科中class="infobox"的table标签，其直接保存的是与当前实体有关的属性。提取表格数据主要用到pandas.DataFrame.to\_excel()函数。此外，为了将网站页面中所有表格数据汇总到一个表格上，我们实现了append\_excel()函数。

def **append\_excel**(data,excelname,sheetname,insert\_type):

    original\_file = **pd**.**DataFrame**(**pd**.**read\_excel**(excelname, sheet\_name=sheetname))  *# 读取原数据文件和表*

    original\_row = original\_file.shape[0]  *# 获取原数据的行数*

    if insert\_type=='w':       *#选择写入excel数据方式，w为覆盖模式，a+为追加模式*

        startrow=1

    elif insert\_type=='a+':

        startrow=original\_row + 1

    book = **openpyxl**.**load\_workbook**(excelname)

    writer = **pd**.**ExcelWriter**(excelname, engine='openpyxl')

    writer.book = book

    writer.sheets = **dict**((ws.title, ws) for ws in book.worksheets)

*# 将data数据写入Excel中*

    data.to\_excel(writer, sheet\_name=sheetname, startrow=startrow, index=False, header=False)

    writer.save()

网站表格数据获取操作指令

Input: ./raw\_relation.html

Output: ./raw\_data.xlsx

cd raw\_data

python extract\_raw\_data.py

**2.1.2 实体获取**

我们将提取得到的表格数据的“姓名”列当作实体数据，保存到entities.txt文档。

df = **pd**.**read\_excel**('./raw\_data.xlsx', sheet\_name='Sheet1')

*# extract entities*

all\_entities = []

for i in **range**(**len**(df)):

    all\_entities.**append**(df.iloc[i, 0])

all\_entities = **sorted**(**list**(all\_entities))

with **open**('entities.txt', 'w', encoding='utf-8') as f:

    for t in all\_entities:

        f.**write**(t.strip() + '\n')

实体数据获取操作指令

Input: ./raw\_data.xlsx

Output: ./entities.txt

cd raw\_data

python extract\_entities.py

**2.1.3 人物关系三元组获取**

我们主要对提取得到的表格数据中的“人物简介”列的半结构化数据进行分析，从中抽取得到包含两实体一关系的人物关系三元组数据，保存到raw\_relations.txt文档。

df = **pd**.**read\_excel**('./raw\_data.xlsx', sheet\_name='Sheet1')

all\_relations = []

for i in **range**(**len**(df)):

    line = df.iloc[i, 1]

*# line是float类型*

    if **isinstance**(line, **float**):

        continue

    if line == "贾府远房宗族":

        relation = df.iloc[i, 0]

        relation += ",贾府,远房宗族"

        all\_relations.**append**(relation)

    line = line.**split**('，')

    for t in line:

        t = t.**split**('。')

        for s in t:

            relation = df.iloc[i, 0]

            if '的' in s:

                s = s.**split**('的')

                for r in s:

                    relation+=","+r

            elif '之' in s:

                s = s.**split**('之')

                for r in s:

                    relation+=","+r

            elif '长子' in s:

                relation += ","+s[:-2]+",长子"

            elif '次子' in s:

                relation += ","+s[:-2]+",次子"

            else:

                continue

            all\_relations.**append**(relation)

with **open**('raw\_relations.txt', 'w', encoding='utf-8') as f:

    for t in all\_relations:

        f.**write**("".**join**(t).**strip**() + '\n')

人物关系三元组获取操作指令

Input: ./raw\_data.xlsx

Output: ./raw\_relations.txt

cd raw\_data

python extract\_relations.py

**2.1.4 人物关系三元组清理**

我们对提取得到的原始人物关系三元组信息做进一步的数据清理。这是由于我们分析得到的结果中可能存在不规则的信息，抑或是错误的关系分类。

不规范文本实例（包含顿号，需要拆分为两条人物关系三元组）

赵嬷嬷,赵天梁、赵天栋,母

昭儿,贾琏、王熙凤,小厮

庆儿,贾琏、王熙凤,小厮

王信,贾琏、王熙凤,小厮

来旺,贾琏、王熙凤,仆人

贾巧姐,贾琏、王熙凤,女

for d in relations:

    if **len**(d) != 3: continue

    if d[2] == "一": continue

    if "、" in d[1]:

        e1 = d[1].split("、")[0]

        e2 = d[1].split("、")[1]

        clean\_relations.**append**([d[0], e1, d[2]])

        clean\_relations.**append**([d[0], e2, d[2]])

        continue

    clean\_relations.**append**(d)

人物关系三元组清理操作指令

Input: ./raw\_relations.txt

Output: ./relations.txt

cd raw\_data

python clean\_relations.py

**2.2 人物介绍数据获取**

我们从红楼梦人物介绍的百度百科中爬取人物的图像信息以及人物介绍信息，其中图像保存到figures/文件夹下，爬取得到的人物介绍信息以json格式保存。

headers = {}

headers["User-Agent"] = "Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_10\_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/44.0.2403.157 Safari/537.36"

def **get\_json**(character\_arr):

    data={}

    for i in **set**(character\_arr):

**print**(i)

        url=r'https://baike.baidu.com/item/'+i

        url = **quote**(url, safe = **string**.printable)

        req = **request**.**Request**(url, headers=headers)

        response = **request**.**urlopen**(req, timeout=20)

        try:

            html = response.read().decode('utf-8')

            soup = **BeautifulSoup**(html, 'html.parser', )

            res = soup.**find**(class\_="summary-pic")

            pic\_name = **str**(i) + '.jpg'

            img\_src = res.**find**('img').**get**('src')

**request**.**urlretrieve**(img\_src,pic\_name)

        except :

**print**("找不到图片")

        res\_key=soup.**find\_all**(class\_ ="basicInfo-item name")

        res\_val=soup.**find\_all**(class\_ ="basicInfo-item value")

        key=[ik.get\_text().strip().replace("\n","、") for ik in res\_key]

        value = [iv.get\_text().strip().replace("\n", "、") for iv in res\_val]

        item=**dict**(**zip**(key,value))

        data[**str**(i)]=item

    if not **os**.path.**exists**("../json"):

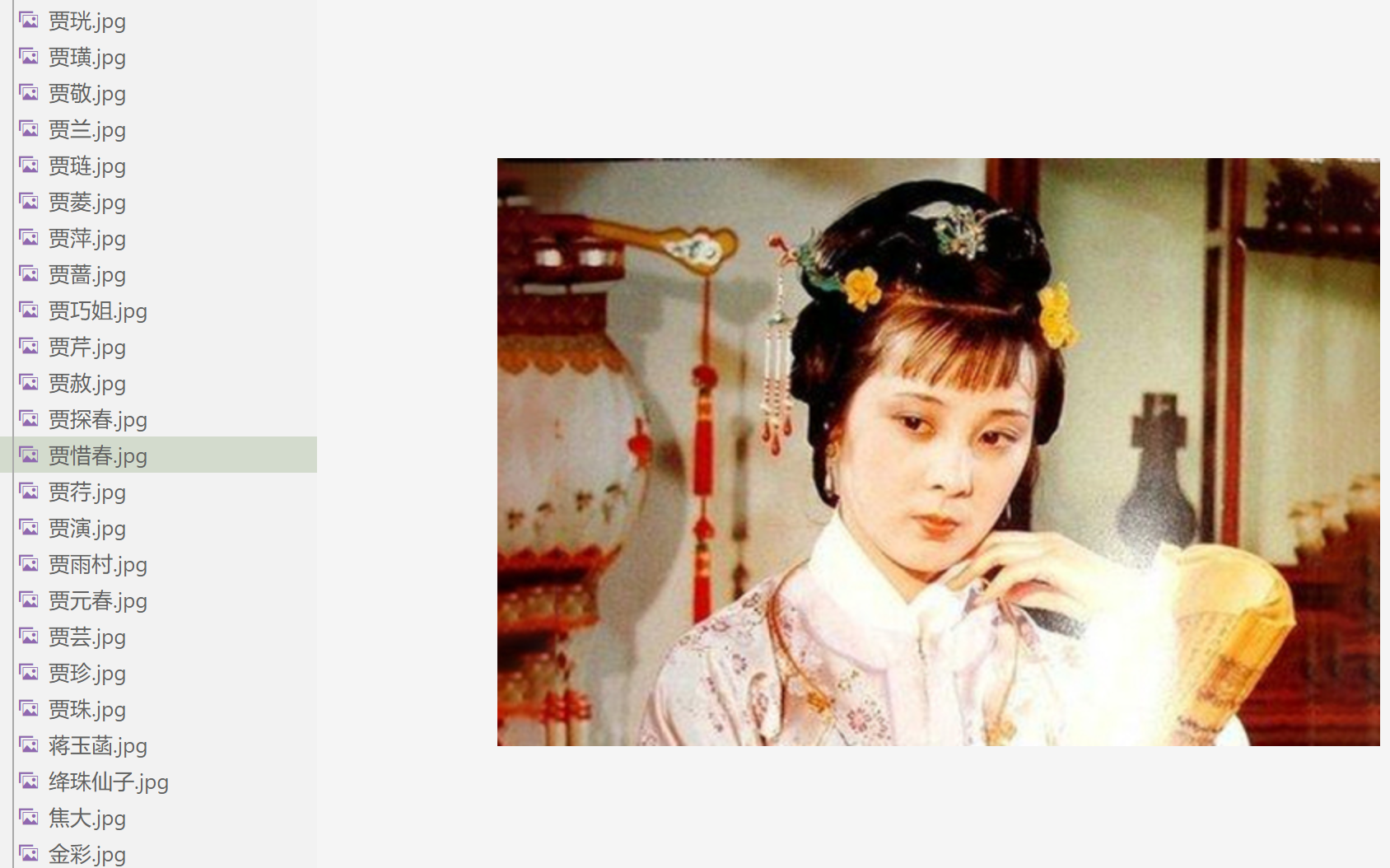
**os**.**mkdir**("../json")

    f = **codecs**.**open**('../json/data.json','w','utf-8')

    f.**write**(**json**.**dumps**(data,  ensure\_ascii=False))

    f.**close**()

以贾惜春为例，爬取得到的图像信息如下：



爬取得到的数据信息如下：

"贾惜春\r": {

        "中文名": "贾惜春",

        "外文名": "Jia Xichun",

        "别    名": "四姑娘、四丫头、藕榭、藕丫头",

        "饰    演": "胡泽红（87版电视剧）、王培祎、徐飒（10版电视剧）、崔傲菲儿（17版电视剧）",

        "性    别": "女",

        "登场作品": "红楼梦、[1]",

        "排    行": "金陵十二钗正册第八位",

        "居    所": "大观园暖香坞",

        "所属府邸": "宁国府",

        "父    亲": "贾敬",

        "哥    哥": "贾珍",

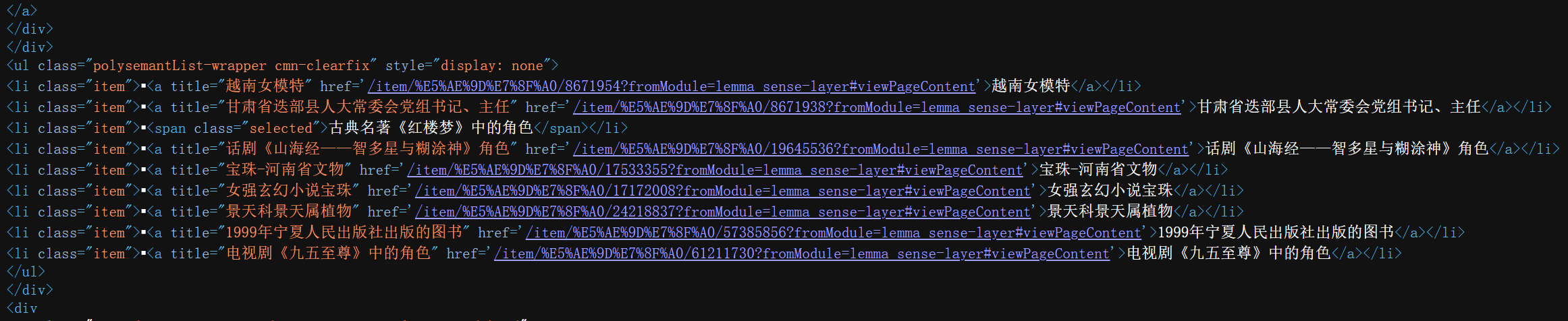
        "嫂    子": "尤氏",

        "侄    子": "贾蓉"

    }

此外，百度百科中通常包含许多词条包含多义项，一般的爬虫程序默认选择第一条义项，但很多时候第一条义项并非我们想要的红楼梦中的人物，因此，我们还做了多义项词条的处理。





我们首先判断该词条是否为多义项词条，若是，则我们进一步找到词条所包含的标题文字内容。找到标题文字内容中包含“红楼梦”的，为我们所需的词条信息。

if soup.find('div', class\_="polysemant-list"):

    res = soup.find\_all('li', class\_="item")

    title = res[0].find('span').get\_text()

    if "红楼梦" not in title:

        for r in res[1:]:

            title = r.find('a').get('title')

            if "红楼梦" in title:

                url = 'https://baike.baidu.com'+r.find('a').get('href')

                url = **quote**(url, safe = **string**.printable)

**print**(url)

                req = **request**.**Request**(url, headers=headers)

                response = **request**.**urlopen**(req, timeout=20)

                html = response.read().decode('utf-8')

                soup = **BeautifulSoup**(html, 'html.parser', )

人物介绍数据获取操作指令

Output: ./json/data.json

Output: ./figures/

cd crawler

python crawler\_character.py

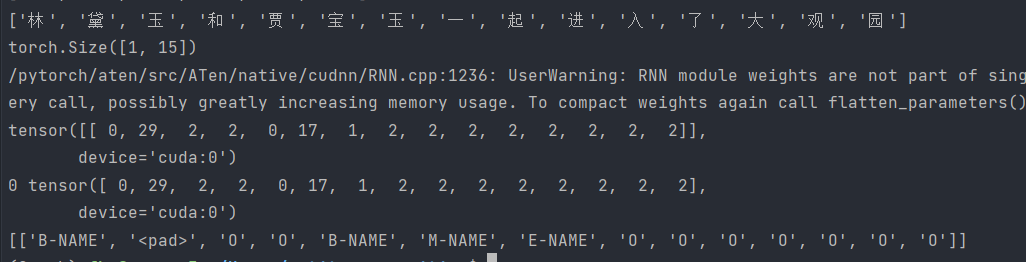
1. **命名实体识别与实体关系抽取**

**3.1 实体识别**

**3.1.1 基于模型的实体识别**

我们尝试使用BiLSM构建了一个中文命名实体识别模型来抽取句子中的实体，对其进行BIOES标注。

运行train\_model.py训练实体识别模型，并使用test.py对我们要处理的数据集中的句子进行测试：



但是我们发现利用识别模型进行识别会出现一些问题，例如，《红楼梦》原文并非白话文，很多句子模型都无法理解，基本上会将所有字都标注为‘O’，即不是名词短语，而且多数时候文中出现的人名只有这个人的名，而没有姓，也会影响抽取的结果。所以我们改为使用白话文版本的红楼梦进行实体抽取，但是可能由于训练集语料的局限性，导致模型并不认识许多人名中的不常用字，例如林黛玉的‘黛’字。所以我们最终选择使用基于匹配的实体识别方法。

**3.1.2 基于匹配的实体识别** 我们通过在2.1.2中获取的实体名称，与原文的句子进行比对匹配。先使用标点符号对文章进行分句，并筛掉一些太短的句子。然后对每个句子进行实体匹配，并保留至少有两个及以上实体的句子。

*# 分句*

cutLineFlag = ["？", "！", "。", "!", "“", "”", "：", "；"]

sentenceList = []

for words in total\_lines:

    oneSentence = ""

    for word in words:

        if word not in cutLineFlag:

            oneSentence = oneSentence + word

        else:

            oneSentence = oneSentence + word

            if oneSentence.**\_\_len\_\_**() > 4:

                sentenceList.append(oneSentence.strip())

            oneSentence = ""

*# 获取所有的实体*

with **open**("data/entities.txt", "r", encoding="utf-8") as f:

    total\_id = list(set([line.strip() for line in f.readlines()]))

**print**("共有实体：", **len**(total\_id))

new\_data = []

for sentence in sentenceList:

    id\_loc = []

    id\_list = []

    for **id** in total\_id:

        if **id** in sentence:

            loc = [(item.start(), item.end()-1) for item in re.finditer(**id**, sentence)]

            id\_list.append(**id**)

            id\_loc.append(loc[0])

*# print(id\_loc)*

    if **len**(id\_loc) >= 2:

        permute = list(itertools.combinations(**range**(**len**(id\_list)), 2))

*# print(len(permute))*

        for idx in permute:

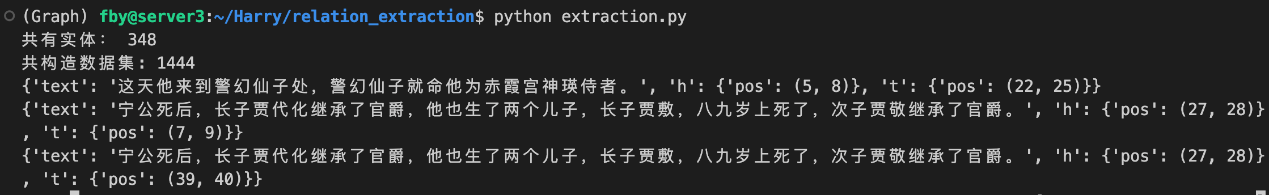
            if id\_list[idx[0]] not in id\_list[idx[1]] and id\_list[idx[1]] not in id\_list[idx[0]]:

                new\_data.append({'text':sentence, 'h': {'pos': id\_loc[idx[0]]}, 't': {'pos': id\_loc[idx[1]]}})

**print**("共构造数据集:", **len**(new\_data))

**print**(new\_data[0:3])

部分抽取结果如下，会将实体在句子中的具体位置标注出来：



**3.2 关系抽取**

我们使用了哈工大的BERT-wwm关系抽取模型，使用了中文维基数据（下载地址：<https://github.com/ymcui/Chinese-BERT-wwm>）进行训练。

运行relation\_extractio/train.py对预训练模型和数据进行进一步训练。然后载入模型对我们构造出的已经进行了实体识别的数据集进行关系抽取：

from tqdm import tqdm

relation\_list = []

for data in tqdm(new\_data):

    text = data['text']

    t\_pos = data['t']['pos']

    h\_pos = data['h']['pos']

    rela = model.infer(data)

    relation\_list.append([text[t\_pos[0]:t\_pos[1]+1], text[h\_pos[0]:h\_pos[1]+1], rela])

relation\_df = pd.DataFrame(relation\_list)

relation\_df.to\_csv("data/relation\_infer.csv", header=False, index=False)

obj\_list = {}

for item in relation\_list:

    if item[2][0] != 'unknown' and item[2][1] > 0.95:

        if (item[0], item[1]) not in obj\_list.keys():

            obj\_list[(item[0], item[1])] = {'relation': [item[2][0]], 'prob': [item[2][1]]}

        else:

            if item[2][0] not in obj\_list[(item[0], item[1])]['relation']: *# 合并重复的实体对，以及其所有的候选关系*

                obj\_list[(item[0], item[1])]['relation'].append(item[2][0])

                obj\_list[(item[0], item[1])]['prob'].append(item[2][1])

new\_relation\_list = []

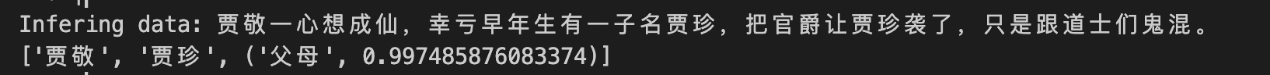
for key, value in obj\_list.items():

    new\_relation\_list.append([key[0], key[1], value['relation'][np.argmax(value['prob'])]])

relation\_df = pd.DataFrame(new\_relation\_list)

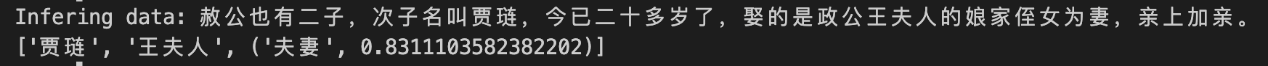
relation\_df.to\_csv("data/relation\_raw.csv", header=False, index=False)

从每个句子中提取出两个实体之间的关系并计算出其置信度，部分结果如下：



D:\Tencent_record\578031026\Image\C2C\Image4\e2e68a5e8af410128c7da9a666f4fb7e.JPG

当然也有部分实体关系推断错误的结果，当句子中人物较多关系较为复杂的时候，就会出现错误结果：



我们根据置信度对结果进行了筛选，并根据从网页上提取的结果和一些常识对错误的结果做了手动剔除，最终的结果保存在relation\_extraction/data/relation\_raw.csv中。

1. **可视化与知识问答**

**4.1 可视化实现**

**4.1.1 实现过程介绍**

使用Neo4j图形数据库实现红楼梦人物关系数据的可视化。

将前面步骤提取出的关系三元组添加进入Neo4j数据库：

with **open**("../raw\_data/relations.txt") as f:

graph.delete\_all() *# 删除已有的所有内容*

for line in f.**readlines**():

rela\_array = line.**strip**("\n").**split**(",")

**print**(rela\_array)

rela\_array = [rela\_array[0], rela\_array[-1], rela\_array[1]]

graph.run("MERGE(p: Entity{Name: '%s'})" % (rela\_array[0]))

graph.run("MERGE(p: Entity{Name: '%s'})" % (rela\_array[2]))

graph.run(

"MATCH(e: Entity), (cc: Entity) \

WHERE e.Name='%s' AND cc.Name='%s'\

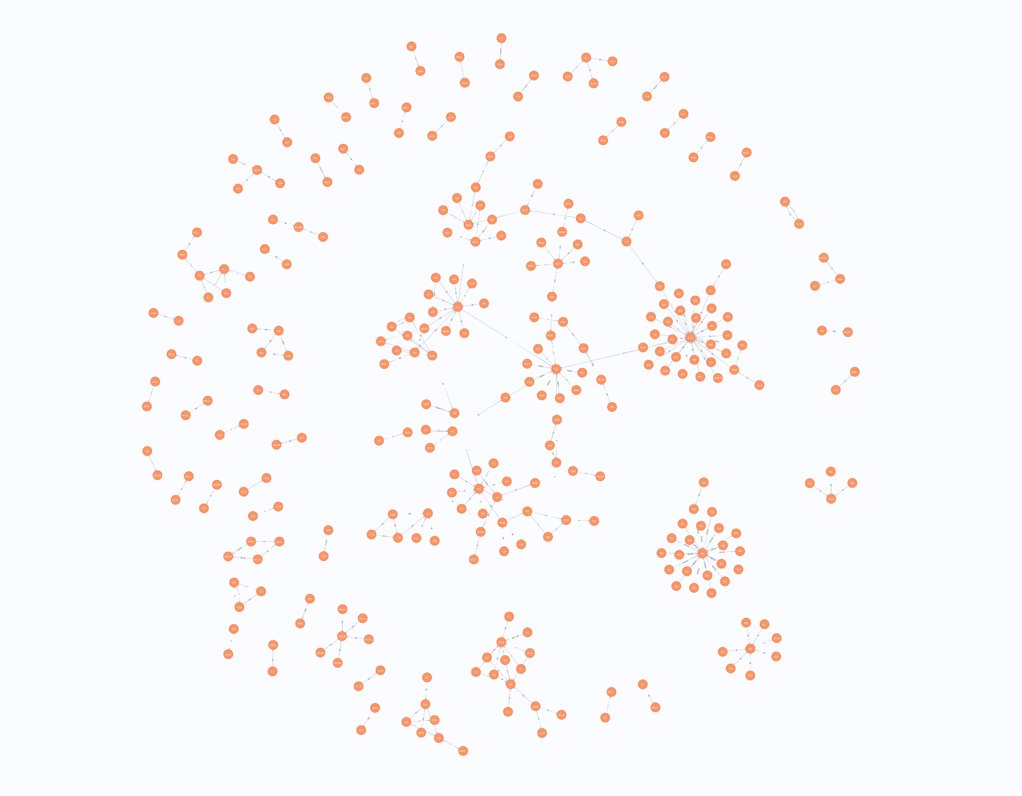
CREATE(e)-[r:%s{relation: '%s'}]->(cc)\

RETURN r" % (rela\_array[0], rela\_array[2], rela\_array[1], rela\_array[1])

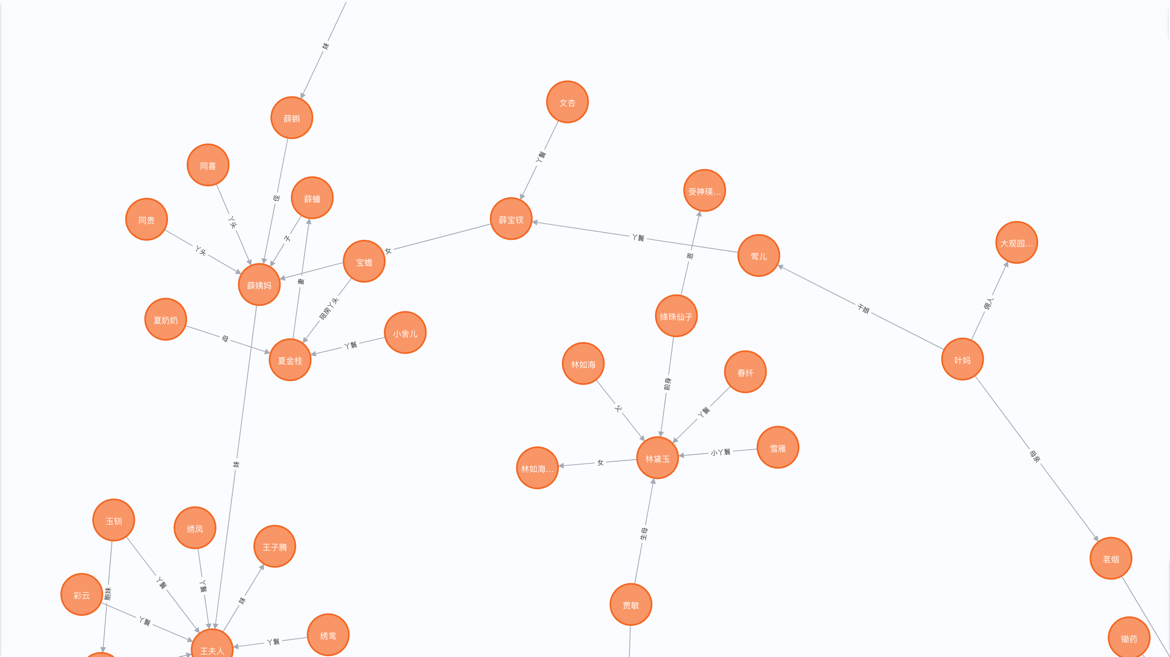
)

登录Neo4j数据库后，即可使用match语句查看想要查看的可视化数据。

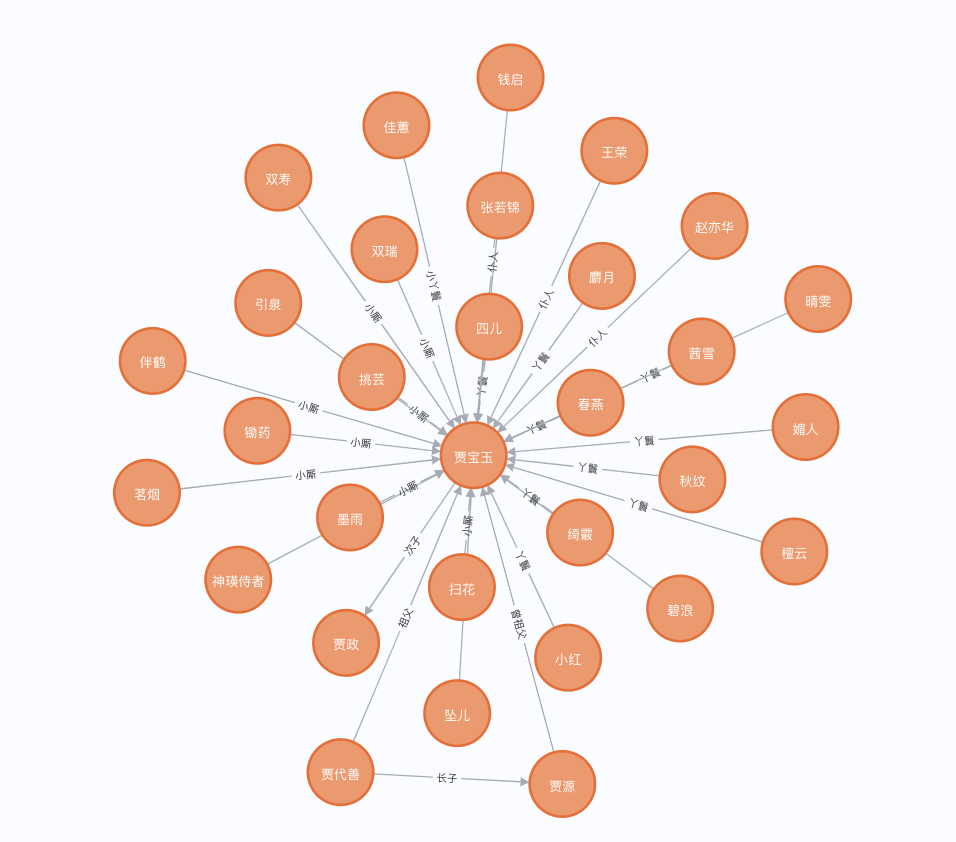
例如，查看所有的人物关系，可以使用MATCH (n) RETURN n，即显示所有任务节点和他们之间的关系。



放大后，可以查看细节：



可以使用match语句进行部分知识查询，例如，查看所有贾宝玉相关的人物关系，使用MATCH p=(n:Entity{Name:"贾宝玉"})<-[]-() RETURN p Union all MATCH p=(n:Entity{Name:"贾宝玉"})-[]->() RETURN p，即可进行知识查询，得到如下结果：



**4.2.2 使用方法**

1. 下载并安装Neo4j数据库，创建账户并修改密码。

2. 将neo\_db/config.py中的账户与密码改为自己的账户与密码，运行Neo4j数据库。

3. 运行neo\_db/query\_graph.py，将提取出的人物关系三元组导入Neo4j数据库中。

4. 在浏览器中访问<http://localhost:7474>，即可在数据库页面通过命令查看关系数据的可视化结果。

**4.2 知识问答**

**4.2.1 实现过程介绍**

1. 首先使用了pyltp工具库对中文自然语言问题进行解析。首先使用ltp模型中的cws模型对问题进行分词：

def **cut\_words**(words):

seg\_model\_path = **os**.path.**join**(LTP\_DATA\_DIR, 'cws.model')

segmentor = **pyltp**.Segmentor(seg\_model\_path)

words = segmentor.segment(words)

array\_str="|".**join**(words)

array=array\_str.**split**("|")

segmentor.release()

return array

然后对分词结果使用pos模型进行词性分析，选择第一个名词作为实体名称，第二个名词作为关系名称：

def **words\_mark**(array):

pos\_model\_path = **os**.path.**join**(LTP\_DATA\_DIR, 'pos.model')

postagger = **pyltp**.Postagger(pos\_model\_path)

postags = postagger.postag(array)

pos\_str=' '.**join**(postags)

pos\_array=pos\_str.**split**(" ")

postagger.release()

return pos\_array

2. 定义问题中关系名称的同义词并组成列表，并定义一组同义词对应的搜索词语列表，即需要在数据库中查询的关系名称。

例如问题中的关系名称为 ‘儿子’，需要在数据库中查询关系名称为 ['儿子','长子','次子','幺子','子'] 的关系条目。

3. 判定问题中的关系名称是否在同义词列表中。如果在，遍历对应的搜索词语列表，逐一提取词语作为关系名称，与实体名称一起，使用match语句在neo4j数据库中进行搜索，将所有搜索结果加入列表后输出结果。如果不在，直接使用问题中的关系名称进行查询，并输出查询结果。

def **get\_KGQA\_answer**(array):

data\_array=[]

for i in **range**(1):

if i==0:

name=array[0]

else:

name=data\_array[-1]['p.Name']

flag = False

for k in **range**(0,**len**(q\_word)):

for word in q\_word[k]:

if word==array[i+1]:

for search in search\_word[k]:

data = graph.run(

"match(n:Entity)-[:`%s`]->(m:Entity) where m.Name='%s' return n.Name" % (

search, name)

)

data = **list**(data)

data\_array.**extend**(data)

flag = True

break

if flag==True:

break

if flag==False:

data = graph.run(

"match(n:Entity)-[:`%s`]->(m:Entity) where m.Name='%s' return n.Name" % (

array[i+1], name)

)

data = **list**(data)

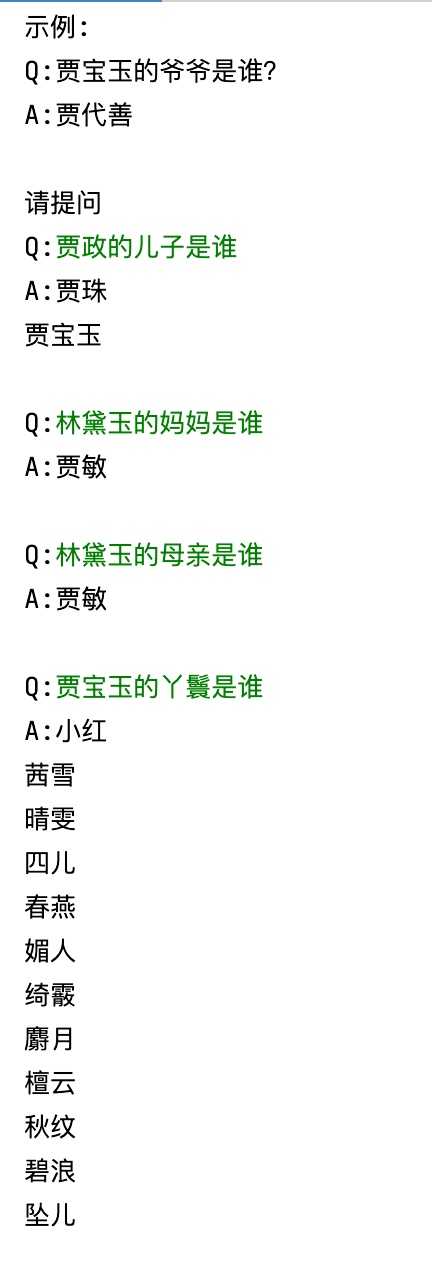
data\_array.**extend**(data)

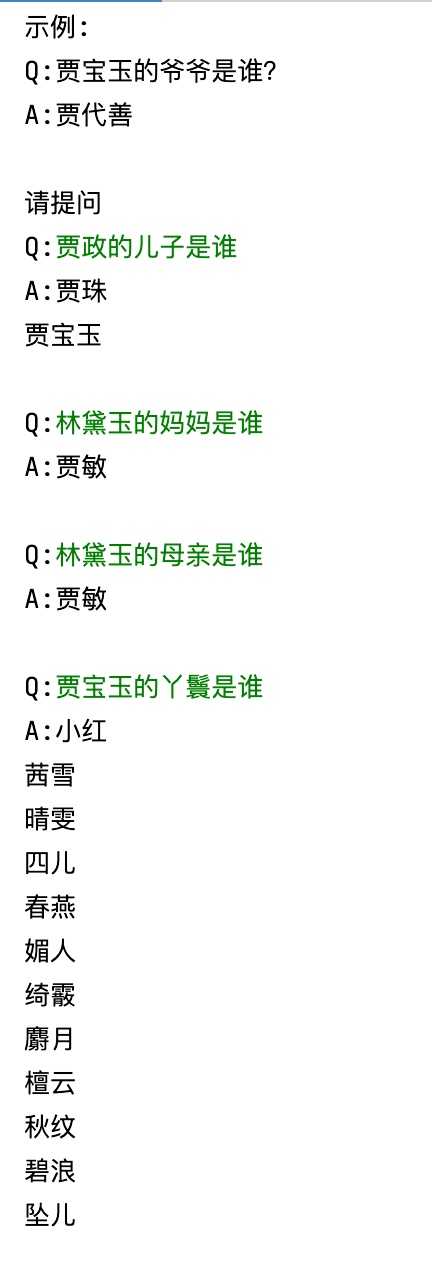
return data\_array

**4.2.2 使用方法与结果展示**

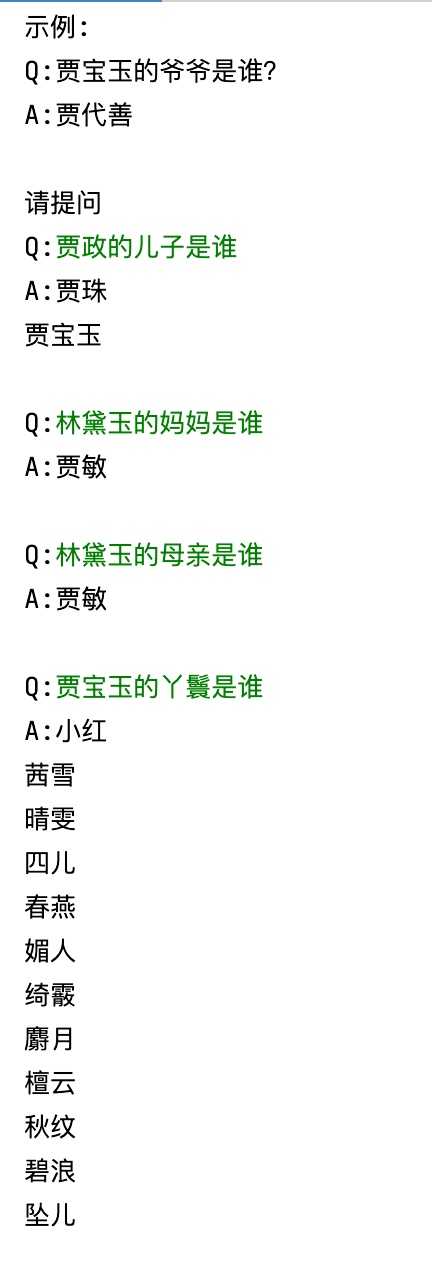
下载与pyltp库版本对应的ltp模型。作者使用的pyltp版本为0.4.0，ltp模型版本为3.4.0。将QA/ltp.py中的LTP\_DATA\_DIR改为正确的模型路径。

运行qa\_app.py，根据提示，在Q:后输入“xxx的xxx是谁”句式的问题，按下回车键。A:后会给出该问题的一条答案或多条答案：

****

****

关系名词为不同的同义词时均可以得出正确答案，例如：

****

1. 《红楼梦人物列表》维基百科网站<https://zh.wikipedia.org/wiki/%E7%BA%A2%E6%A5%BC%E6%A2%A6%E4%BA%BA%E7%89%A9%E5%88%97%E8%A1%A8> [↑](#footnote-ref-1)
2. 红楼梦角色介绍 <https://baike.baidu.com/item/%E8%B4%BE%E5%AE%9D%E7%8E%89/59563> [↑](#footnote-ref-2)