Tutorial : 使用elasticsearch搭建简单知识问答系统

本文档简述基于elasticsearch搭建一个简易的知识问答系统。该问答系统可以解析输入的自然语言问句生成ES查询，然后执行得到结果。目前实现了按照名称检索实体，实体属性，多跳检索，以及检索符合多对属性要求的实体。在功能逻辑完成后，可搭建网站将其可视化，一个演示的demo在此. 实现的数据和代码在此

1. demo功能介绍

1.1 实体检索

实体检索即输入实体名称，返回该实体的所有属性和属性值。

示例输入

- 姚明
- 佟大为

属性名称	属性值
birthPlace	上海
gender	男
children	姚沁蕾
职业	运动员 篮球运动员 其他 上海大鲨鱼队老板
birthDate	1980年9月12日
height	226
nationality	中国
subj	姚明
alumniOf	上海体育技术教育学院
民族	汉族

2.实体的属性检索

输入实体名称和一个属性名称，如果该实体存在该属性值，则返回该属性值。

示例输入

- 姚明是在哪儿出生的
- 佟大为有多高

3. 多跳查询

多跳查询即形如"姚明的女儿的身高"的查询,即"姚明:女儿"查询得到的是实体"姚明"的一个属性，但同时这个属性值也作为一个实体存在于数据集中，那么就可以接着对该实体继续查询其属性。

示例输入

- 姚明的女儿的母亲是谁
- 姚明的女儿的母亲有多高

4. 根据属性值查询实体

输入隐含多对 [属性名 opearotr 属性值]的自然语言问句， 它们之间的关系可以是AND， OR，NOT，同时属性值可以是等于，大于，小于一个输入值，返回满足这些属性限制的实体(由于显示原因返回10个)。

示例输入

- 中国足球运动员
- 中国女运动员
- 身高大于170的中国或美国的作家
- 身高>180的中国演员
- 身高>200,体重小于200的中国篮球运动员

实体名称	查询链接
林晴萱	林晴萱
谢莉·杜瓦尔	谢莉·杜瓦尔
卡里姆·阿卜杜拉·贾巴尔	卡里姆·阿卜杜拉·贾巴尔
余秋雨	余秋雨
何炅	何炅
曹德权	曹德权
迈克尔·克莱顿	迈克尔·克莱顿
陆绒	陆绒
卢丽莉	卢丽莉
陈久全	陈久全

2. 环境准备

2.1 elasticsearch安装

- 去官网找到elasticsearch的安装包，下载.
- 运行安装包目录下的/bin/elasticsearch.sh(在windows上运行/bin/elasticsearch.bat，本实验在ubuntu上完成，后续步骤涉及到一些linux指令)

> 注意：该命令已经运行了elasticsearch. 可能提示不能在root账户下运行，此时请切换到非root账户下运行。如果想让elasticsearch在后台一直运行，在上述命令最后加-d参数即可。

- 至此，安装完成。可以通过访问本地9200端口来访问elasticsearch:

```
curl 'http://localhost:9200'
```

> 注：可通过修改配置文件使elasticsearch可以远程访问

3. 数据准备

实验所使用的数据集为一个基于cnschema标准的人物属性数据集。该数据集由三元组组成，每个三元组描述一个人物实体的某个属性。在将此数据集导入elasticsearch之前，需要考虑其在elasticsearch中存储的方式。最简单的方式就是将每个三元组视作一个文档，其中包含3个字段, 分别为三元组的(subject, predicate, object). 但本实验采取的是另一种方式，即：一个实体的所有属性和属性值为一个文档。 具体细节及原因见下述。

3.1 知识库格式转换 (preprocess.py)

实验所使用的数据集的格式如下：

```
A.J.万德 affiliation 篮球
A.J.万德 description A.J.万德（A.J. Wynder），1964年出生，前美国篮球运动员。
A.J.万德 nationality 美国
A.J.万德 weight 82公斤
A.J.库克 birthDate 1978年7月22日
A.J.库克 birthPlace #加拿大安大略省奥沙瓦
A.J.库克 description A.J.库克（A.J. Cook），1978年7月22日在加拿大安大略省奥沙瓦出生，演员。|||1997年出道在电视电影《父亲大人》饰演了配
角Lisa 。1999年在电影处女作《处女之死》中饰演五女儿之一的Mary Lisbon 。2003年在电影《死神来了2》中出演主角 。2005年开始在CBS美剧《犯罪
心理》饰演常驻角色Jennifer Jareau；2010年短暂离开《犯罪心理》剧组，2011年重新回归再次成为常驻角色 。2011年还在电视电影《带艾瑟莉回家》
中饰演主角Libba并凭借此片提名棱镜奖迷你剧或电视电影最佳表演奖 。
A.J.库克 height 1.69m
A.J.库克 nationality 加拿大
A.J.库克 代表作品 《死神来了2》
A.J.库克 职业 演员
A.J.英格利什 birthDate 1967年7月11日
A.J.英格利什 description A·J·英格利什（英语全名：Albert Jay English，1967年7月11日－），为美国NBA联盟的前职业篮球运动员。他在1990年的N
BA选秀中第2轮第10顺位被华盛顿子弹选中。
A.J.英格利什 nationality 美国
```

elasticsearch要求文档的输入格式为json。将实验数据集转化为json格式后，每个实体对应一个json的object，也即elasticsearch中的一个文档：

```
{
    "subj":"A.J.万德",
    "weight":"82公斤",
    "height":None,
    "po":[{"pred":"affiliation","obj":"篮球"},
          {"pred":"description","obj":"A.J.万德（A.J. Wynder），1964年出生，前美国篮球运动员。"},
          {"pred":"nationality","obj":"美国"},
         ]
}

{
  ...
}
```

如上所示，数据集中"A.J.万德"的所有属性及属性值汇总在一起，存储在一个json对象中作为一篇文档导入elasticsearch, 其它的每个实体类似。

- 其中，所有属性除了"height"及"weight"两个属性之外，都存在一个名为"po"的list对象中，每个属性及其属性值作为一个小的object，分别用键"pred"和"obj"来标识属性名和属性值。
- 之所以要将"height"和"weight"单独考虑，而不是和其它属性一样也存储在list中，是因为这两个属性要支持范围搜索，即"height>200"这样的搜索，因此要求它们在存储时的数据类型为integer，而list中的所有属性的属性值的存储类型都为keyword(不分词的string，只支持全文匹配)。

- 之所以每一对(属性名，属性值)存储为一个object，并放入一个list中，是因为这是elasticsearch定义的一种nested object的数据类型，这种数据类型能存储大量拥有相同的key的对象，并且可以对之进行有效的检索。这样，不论数据集中有多少种种类不同的属性，都可以以相同的格式存储。
- 之所以不是每一个三元组存储为一篇文档，而是一个实体相关的所有属性及属性值存储为一篇文档，是因为要支持通过多对(属性，属性值)联合检索满足要求的实体，以这种格式存储，能提高检索效率，具体原因见后面对应部分。

另外，实验数据集中某些属性的属性值不是很规范，例如height，weight的属性值存在单位不同，包含无关字符等问题，其它属性的属性值也存在多个值以空格等字符连接作为一个值(例如，"职业：运动员 足球运动员", 这个为了检索时匹配方便，应该将其拆成两个)的问题，因此在格式转换的同时也要对属性值做一些清理。

3.2 属性同义词扩展(可选) (attr_mapping.txt)

因为实验的数据集较小，包含的属性种类不多，因此可以人工增加一些同义的属性词。下面的文件中每一行的第一个词为数据中存在的属性，后面的为后来添加的同义的属性词。在解析查询语句的时候，如遇到同义的属性词，可将其映射到数据集中存在的属性上。

```
weight 重量 多重 体重
relatedTo 相关 有关
telephone 电话 号码 电话号 电话号码 手机 手机号 手机号码
birthDate 出生日期 出生时间 生日 时候出生 年出生
height 高度 海拔 多高 身高
sibling 兄弟 哥哥 姐姐 弟弟 妹妹 姐妹
workLocation 工作地点 在哪工作 在哪上班 上班地点
children 子女 孩子 女儿 儿子
年龄 几岁 多大
代表作品 代表作 著作 成就 作品
homeLocation 家庭住址 住哪 住在哪 住在什么
职业 工作 做什么 干什么
colleague 大学 高校 毕业于
birthPlace 出生地 在哪出生 出生在
description 简介 是什么 描述 什么是 概述
parent 父母 双亲
jobTitle 工作 职业
award 奖项 奖励
address 地址 在哪 位置 在什么地方
nationality 国籍 哪国人
spouse  配偶 丈夫 妻子 老婆 老公
deathData 去世日期 逝世日期 时候死
affiliation 从属
gender 性别 是男是女
民族
deathPlace 去世地点 在哪死 在哪去世
memberOf 成员
alumniOf 校友 毕业于 毕业
```

4. 导入elasticsearch

4.1 elastcisearch的index和type简介

elasticsearch用index和type管理导入的文档。其中index可以类比为一个单独的数据库，其中存放的是结构相似的文档。type是index的一个子结构，可以存放不同部分的数据，可以类比为一张表，而每一篇文档都存储在一个type中，类似于一条记录存储在一张表中。

4.2 在elasticsearch中新建index和type

为实验数据集新建index('demo')和type('person')。elasticsearch使用Restful API可以方便的交互，通过elasticsearch的mapping文件可以创建index和type，并指定每个字段在elasticsearch中存储的类型。

下述示例用curl命令在命令行中与elasticsearch交互。其中, height, weight存储为integer数据类型，而实体名subj和其他属性存储为keyword类型。所有其他属性存储在一个nested object对象中。 打开命令行，运行:

```
curl -XPUT 'localhost:9200/demo?pretty' -H 'Content-Type: application/json' -d'
```

```
{
    "mappings": {
        "person": {
            "properties": {
                "subj": {"type": "keyword"},
                "height": {"type": "integer"},
                "weight": {"type": "integer"},
                "po":{
                    "type": "nested",
                    "properties":{
                        "pred":{"type":"keyword"},
                        "obj":{"type":"keyword"}
                    }
                }
            }
        }
    }
}
'
```

> 注：如果没有curl命令，可以安装一下, sudo apt-get install curl

4.3 导入数据 (insert.py)

往新建的type中导入实验数据集，导入同样使用Restful API，可以使用elasticsearch提供的insert方法。一个示例的python导入脚本如下：

```
#coding:utf-8
'''
将一个知识图谱中的数据导入elastic search
'''
try:
    import simplejson as json
except:
    import json

import sys
import requests

def bulk_insert(base_url, data):
    response = requests.post(base_url, headers={"Content-Type":"application/x-ndjson"}, data=data)

def begin_insert_job(index_name, type_name, json_filepath, bulk_size=1000):
    '''
        index_name: 要导入的index的名称
        type_name, 要导入的type的名称
        json_filepath: 要导入的json文件的路径
        bulk_size: 批导入时一次导入的文档数目
    '''
    base_url = "http://localhost:9200/" + index_name + "/" + type_name + "/_bulk"
    f = open(json_filepath)
    cnt, es_id = 0, 1
    data = ""
    for line in f:
        action_meta = '{"index": {"_id":"' + str(es_id) + '"}}'
        data = data + action_meta + "\n" + line

        es_id += 1
        cnt += 1
        if cnt >= bulk_size:
            bulk_insert(base_url, data)
            cnt, data = 0, ""
        if not (es_id % bulk_size):
            print es_id
```

```
    if cnt:
        bulk_insert(base_url, data)

if __name__ == '__main__':
    begin_insert_job("demo", "person", "./data/person.json")
```

运行此脚本就可以将已经转换好格式的数据文件导入刚才新建的index和type中。

> 注意更改其中文件的路径。

此时已经可以检索该知识库了，例如，按照实体名称检索：

```
curl -XGET 'localhost:9200/demo/person/_search?&pretty' -H 'Content-Type:application/json' -d'
{
    "query":{
        "bool":{
            "must":{
                "term":{"subj":"姚明"}
            }
        }
    }
}
'
```

> 注：elasticsearch的查询除了常见的get方式，即将参数和参数值作为链接的一部分提交，也支持如上所示将查询参数写入一个json结构体，用该请求体查询的方式。这种方式由于表达方式更加灵活，因此可以表达较为复杂的查询。具体细节可以参考elasticsearch文档。

5 自然语言转化为Logical form (views.py)

本实验支持4种类型的查询，首先为这4种查询预定义logical form的模板和ES查询的模板，填充模板得到最终ES查询语句

查询类型	自然语言查询语句	Logical form
实体检索	姚明是谁	姚明
实体的属性检索	姚明有多高	姚明:height
实体属性的多跳检索	姚明的女儿的母亲是谁	姚明:女儿:母亲
多种属性条件检索实体	身高大于180的中国或美国的作家	身高>180 And 国籍:中国 Or 国籍:美国 And 职业:作家

5.1 logical form模板

logical form里有如下几个元素:

- 三元组的成分: S(subject), P(predicate), O(object)
- 单个属性条件的OP(operator): :, <, >, <=, >=. 例如 “职业:演员”,”身高>200”
- 属性条件之间的与、或关系:, And, Or. 例如 “职业:演员 And 身高>200”

查询类型	Logical form模板	示例
实体检索	S	姚明
实体的属性检索	S:P	姚明:height
实体属性的多跳检索	S:P1:P2 ...	姚明:女儿:母亲
多种属性条件检索实体	P1 OP O1 And/Or P2 OP O2 ...	身高>180 And 国籍:中国 Or 国籍:美国 And 职业:作家

5.2 解析自然语言

首先，对于输入的自然语言问句，识别出其中出现在知识库中的实体名，属性名以及属性值 识别的流程如下：

- 分词
- 对于属性，由于种类较少，直接用字典记录知识库中的所有属性，用匹配的方法找出所有属性名
- 对于实体名，可将分词的结果(为保证正确性，可以将分词算法的词典替换成所有实体名，分词工具例如jieba分词支持自定义词典)查询elasticsearch，判断是否存在以该词语为实体名的文档，若存在表明该词语是一个实体名
- 对于属性值，由于变化较大，可以采用模糊匹配的方法，也可以采用分词后n-gram检索es的办法。在判断一个短语是属性值后，还要统计该属性值对应的属性名，当查询语句中缺省了属性名，例如"(国籍是)中国(的)运动员"，缺省了"国籍"，就用该属性值对应的最频繁的属性名作为补全的属性名。

例子：

```
#属性名识别
def _map_predicate(nl_query,map_attr=True):    #找出一个字符串中是否包含知识库中的属性

    #将同义属性映射到知识库种的属性
    def _map_attr(word_list):
        ans = []
        for word in word_list:
            ans.append(attr_map[word.encode('utf-8')][0].decode('utf-8'))
        return ans

    match = []
    # 预先读取字典，通过匹配的方法找出问句的属性名
    for w in attr_ac.iter(nl_query.encode('utf-8')):
        match.append(w[1][1].decode('utf-8'))
    if not len(match):
        return []

    ans = _remove_dup(match)
    if map_attr:
        ans = _map_attr(ans)
    return ans
```

```
#实体名识别
def _entity_linking(nl_query):     #找出一个字符串中是否包含知识库中的实体，这里是字典匹配，可以用检索代替
    parts = re.split(r'的|是|有', nl_query)
    ans = []
    for p in parts:
        pp = jieba.cut(p) #分词
        if pp is not None:
            for phrase in _generate_ngram_word(pp): #分词结果的n-gram查找
                if phrase.encode('utf-8') in ent_dict: #匹配字典，数据量大时可以用搜索ES代替
```

```
                    ans.append(phrase)
    return ans
```

5.3 生成logical form

在识别出查询中所有的实体名，属性名和属性值后，依据它们的数目及位置，确定查询的类型，以便映射到的对应的logical form

Logical form生成流程:

- 如果有实体名
  - 如果有多个属性名，那么是属性值的多跳查询
  - 如果有一个属性名，那么判断实体名和属性名的位置及中间的连接词(“是”“在”“的”等)。若实体名在前，则是实体的属性查询，例如“姚明的身高”；若属性名在前，则是依据属性查询实体，例如“女儿是姚沁蕾”
- 如果没有实体名，则认为是依据属性查询实体
  - 根据所有属性名和属性值位置的相对关系，确定它们之间的对应关系
  - 对于缺省属性名的属性值，补全属性名
  - 对于缺乏属性值的属性名，例如“身高>200”，如果是特殊类型的属性名,例如“身高”，“体重”，则通过正则表达式识别出范围查询的数值；否则，则忽视。

```
# 伪代码
def translate_NL2LF(nl_query):
    '''
    args:   nl_query: 自然语言查询语句
    return:   lf_query: logical form查询语句
    '''
    entity_list = _entity_linking(nl_query)   #识别实体名
    attr_list = _map_predicate(nl_query)   #识别属性名
    if entity_list:          #识别到实体
        if not attr_list:    #### 问题类别：   查询单个实体 ####
            lf_query = entity_list[0]   #生成对应的Logical form
        else:
            if len(attr_list) == 1:    #### 问题类别：   单个实体属性查询 ####
                if first_entity_pos < first_attr_pos:  #判断实体名和属性名的出现位置
                    lf_query = "{}:{}".format(entity_list[0], attr_list[0]) #SP 生成对应的Logical form
                else:
                    lf_query = "{}:{}".format(attr_list[0], entity_list[0]) #PO 生成对应的Logical form
            else:                      #### 问题类别：   多跳查询 ####
                lf_query = entity_list[0]
                for pred in attr_list:
                    lf_query += ":" + pred  #生成对应的Logical form
    else:   #### 问题类别：   多个属性条件检索实体 ####
        val_d = _val_linking(nl_query)   #识别属性值
        retain_attr = _map_attr_val(attr_list, val_d) #根据相对位置找到属性名和属性值的对应关系，剩下没有对应的属性名
        for a in retain_attr:
            if a == 'height' or a == 'weight':
                value = get_value(a, nl_query) #正则表达式找出数值
                val_d.append((value, a))
        for v in val_d:
            if v in prev or find_or: #同类型属性出现过，或者解析到“或者”等词语
                lf_query += ' OR ' + '{}:{}'.format(pred, v)
            else:
                lf_query += ' AND ' + '{}:{}'.format(pred, v)
    return lf_query
```

6 Logical form 翻译成ES查询语句

在生成logical form之后，需要查询的实体和属性，以及查询的类型都已经明确了,因此可以直接用对应的ES模板将logical form翻译成es查询

6.1 ES模板

对于实体属性查询，包括多跳查询，都是先检索实体，然后获取对应的属性；对于多个属性条件检索实体，先为每种单个的属性条件创建ES查询的模板，最后组合成完整的查询。 因此，先为每个独立的部分生成部分的ES查询语句，最后再合并成完整的语句

细分的查询类型	Logical form	ES模板_部分查询
检索实体	S1	"query":{"bool":{ "must":{ "term":{"subj“ : S1} }}}
属性条件 op为等于	P1 : O1	"query":{"bool":{"must":[{"term":{"po.obj":O1}}, {"term":{"po.pred":P1}}]}}

属性条件 op为范围检索	P1 >= O1	"range":{
		P1:{"gte":O1}
		}

生成每个属性条件对应的部分ES查询语句后，用如下模板合并成完整查询. 下表中part_query表示单个属性条件对应的部分查询

连接条件	Logical form	ES模板_整体
And	P1 OP O1 And P2 OP O2	"query": {"bool":{"must":[part_query1, part_query2]}}
Or	P1 OP O1 Or P2 OP O2	"query": {"bool":{“should":[part_query1, part_query2]}}

6.2 按名称检索实体

按名称检索实体，并返回该实体的所有属性和属性值。将Logical form种的实体名填入ES实体查询的模板即可生成对应的ES查询。 对于结果，只需要将查询的结果解析一下，写入一个python dict对象返回即可

```
def _search_single_subj(entity_name):
    query = json.dumps({"query": { "bool":{"filter":{"term" :{"subj" : entity_name}}}}}) #组装query
    response = requests.get("http://localhost:9200/demo/person/_search", data = query) #查询
    res = json.loads(response.content)

    if res['hits']['total'] == 0:
        return None, 'none'
    else:
        card = dict()  #解析查询结果，将结果写入dict对象，该实体的知识卡片返回
        card['subj'] = entity_name
        s = res['hits']['hits'][0]['_source']
        if 'height' in s:
            card['height'] = s['height']
        if 'weight' in s:
            card['weight'] = s['weight']
        for po in s['po']:
            if po['pred'] in card:
                card[po['pred']] += ' ' + po['obj']
            else:
                card[po['pred']] = po['obj']
        return card, 'done'
```

6.3 检索实体的属性，以及多跳查询

检索一个实体的某个属性的值，也是先检索该实体，然后判断返回的结果中是否包含所检索的属性，如果包含，则返回对应的值，因此，这种检索的查询语句同上。 如果是多跳查询，则在检索出一个属性对应的属性值后，需要再判断知识库中是否存在以该属性值为名称的实体，如果存在，则以该属性值为实体名称检索对应的实体，再判断结果是否包含检索的第2个属性，如此循环，直到得到最终结果。

```
def _search_multihop_SP(parts):
    has_done = parts[0]
    v = parts[0]
    for i in range(1, len(parts)):
        en = _entity_linking(v) #判断知识库中是否存在名称为v的实体
        if not len(en):
            return '执行到: ' + has_done, '==> 对应的结果为:' + v + ', 知识库中没有该实体: ' + v
        card, msg = _search_single_subj(en[-1])  # 同上，检索实体v
        p = _map_predicate(parts[i])  #判断知识库中是否存在以part[i]为名称的属性
        if not len(p):
            return '执行到: ' + has_done, '==> 知识库中没有该属性: ' + parts[i]
        p = p[0]
        if p not in card:   #判断该实体是否存在以part[i]为名称的属性
            return '执行到: ' + has_done, '==> 实体 ' + card['subj'] + ' 没有属性 ' + p
        v = str(card[p])
        has_done += ":" + parts[i]
    return v, 'done'
```

6.4 根据多对(属性名，属性值)检索实体

这里要支持根据多对(属性名，属性值)检索实体，而且不同的属性值对之间可以有and或or关系，并且可以对单个属性值对取not操作。而且，部分属性，例如height,weight支持范围搜索。

6.4.1 查询构建

这里涉及到的elasticsearch查询要稍微复杂一些。假设已经解析好了查询语句的组成部分，即：每对属性值对，它们之间的and或or关系，not操作，以及每个属性值对的操作是等于还是范围检索，那么可以构造出一个查询直接返回满足这些要求的实体。

例子

查询重量>=50 AND 国籍:中国 AND 职业:篮球运动员,其对应的查询语句如下:

```
curl -XGET 'localhost:9200/demo/person/_search?&pretty' -H 'Content-Type:application/json' -d'
{
    "query": {
        "bool":{
            "must":[     # must关键字:  其内的查询条件是AND关系
                {        # 查询条件1:  重量>= 50
                    "range":{  #weight
                        "weight":{
                            "gte":50
                        }
                    }
                },
                {        # 查询条件2:   国籍:中国
                    "nested":{       # 查询nested object
                        "path":"po",  # 制定nested obect位置
                        "query":{
                            "bool":{
                                "must":[
                                    {"bool":{"must_not":{"term":{"po.obj":"中国"}}}},
                                    {"term":{"po.pred":"nationality"}}
                                ]
                            }
                        }
```

```
                    }
                },
                {        # 查询条件3:   职业：篮球运动员
                    "nested":{
                        "path":"po",
                        "query":{
                            "bool":{
                                "must": [
                                    {"term":{"po.obj":"篮球运动员"}},
                                    {"term":{"po.pred":"职业"}}
                                ]
                            }
                        }
                    }
                }
            ]
        }
    }
}
'
```

因此，实现该功能的第一步是解析查询语句，构造出对应的查询语句。这样就能通过执行该查询语句一次查询得出结果，如果存储方式为一个三元组一篇文档，那么为了实现多个属性值对检索实体，必须对每个属性值对检索一次，最后再将结果合并起来，需要查询多次。

> 注： 如果关系是OR，那么对应elasticsearch的should关键字；如果在属性值对前加了否定NOT，那么，对应的elasticsearch关键字是must_not
> 上述语句中的注释会影响执行，如需执行，将注释删掉

解析查询语句及构造elasticsearch查询的过程如下:

```
def _search_multi_PO(exps, bool_ops):  #处理 多对属性值检索实体 的查询
    '''
        exps: 按照 'AND','OR'对查询分割后的每对(属性名 op 属性值)
        bool_ops: 查询中的'AND','OR'连接符，保持在原句中的顺序
    '''
    ans_list = []
    po_list = []
    cmp_dir = {
        "<":"lt",
        "<=":"lte",
        ">":"gt",
        ">=":"gte"
    }

    for e in exps:          #解析每一对属性值
        if e == "":
            return "", 'AND 或 OR 后不能为空'

        begin_with_NOT = False  #是否有NOT操作符
        if e[0:3] == 'NOT':
            begin_with_NOT = True
            e = e[3:]
        elif 'NOT' in e:
            return e, 'NOT请放在PO对前面'

        op = re.findall(": |:|>|<|>=|<=",e)
        if len(op) != 1:          #没有操作符
            return e, '语法错误'
        op = op[0]
        if op == '<' or op == '>':
            index = e.find(op)
            if e[index+1] == '=':
                op = op + '='
```

```
        pred, obj = e.split(op)
        c_pred = _map_predicate(pred)
        if not len(c_pred):
            return e, '知识库中没有该属性: ' + pred
        if obj == '':
            return e, '属性值不能为空'
        pred = c_pred[0]

        part_query = ""               #该属性值对应的部分查询语句
        if not begin_with_NOT:
            if op == ':' or op == ': ':
                if pred == 'height' or pred == 'weight':
                    part_query = '{"term":{"' + pred + '":' + obj + '}}'
                else:
                    part_query = '{"nested":{"path":"po","query":{"bool":{"must":[{"term":
{"po.pred":"' + pred + \
                                 '"}},{"term":{"po.obj":"' + obj + '"}}]}}}}'
            else:
                if pred == 'height' or pred == 'weight':
                    part_query = '{"range":{"' + pred + '":{"' + cmp_dir[op] + '":' + obj + '}}}'
                else:
                    return e,'该属性不支持比较大小,目前只支持height,weight'
        else:
            if op == ':' or op == ': ':
                if pred == 'height' or pred == 'weight':
                    part_query = '{"bool":{"must_not":{"term":{"' + pred + '":' + obj + '}}}}'
                else:
                    part_query = '{"nested":{"path":"po","query":{"bool":{"must":[{"term":
{"po.pred":"' + pred + \
                                 '"}},{"bool":{"must_not":{"term":{"po.obj":"' + obj + '"}}}}]}}}}'
            else:
                if pred == 'height' or pred == 'weight':
                    part_query = '{"bool":{"must_not":{"range":{"' + pred + '":{"' + cmp_dir[op] +
'":' + obj + \
                                 '}}}}}'
                else:
                    return e,'该属性不支持比较大小,目前只支持height,weight'
        po_list.append(part_query)

    or_po = [False] * len(exps)     #根据AND，OR关系合并上述的查询语句部分，形成完整的查询语句
    should_list = []
    must_list = []
    i = 0
    while i < len(bool_ops):    #用OR操作符连接的子句优先级更高，先合并
        if bool_ops[i] == 'OR':
            adjacent_or = [po_list[i]]
            or_po[i] = True
            while i < len(bool_ops) and bool_ops[i] == 'OR':
                adjacent_or.append(po_list[i+1])
                or_po[i+1] = True
                i += 1
            should_list.append(",".join(adjacent_or))
        i += 1
    for i,po in enumerate(or_po):     #再合并AND操作符连接的子句
        if not po:
            must_list.append(po_list[i])
    must_list = ",".join(must_list)
    query = ""                #合并所有子句，形成最终query
    if must_list:
        query = '{"query":{"bool":{"must":[' + must_list + ']'
        if should_list:
            query += ","
            for s in should_list:
                query += '"should":[' + s + '],'
            query = query[:-1]
```

```
            query += '}}}'
        else:
            query = '{"query":{"bool":{'
            if should_list:
                for s in should_list:
                    query += '"should":[' + s + '],'
                query = query[:-1]
            query += '}}}'
            #... 下面部分为执行查询
```

6.4.2 执行查询

构建出查询语句后，执行改查询，解析查询结果即可

```
# 上面部分为查询语句构造...
response = requests.get("http://localhost:9200/demo/person/_search", data = query)
res = json.loads(response.content)

if res['hits']['total'] == 0:
    return None,'none'
else:
    ans = {}
    for e in res['hits']['hits']:
        name = e['_source']['subj']
        ans[name] = "/search?question="+name

    return ans, 'done'
```