第 5 章

人物关系智能问答

本章构建了一个以人物为中心，辅以学校、作品等对象的人物关系图谱，并在图谱的基础上基于模板研发简易的智能问答程序，实现根据用户提出的问题给出合适的回答。

本章主要学习 Flask Web 框架、BootStrap 前端开发框架、Jieba 分词的用法，掌握通过结构化数据搭建模式匹配型智能问答系统的基础流程。

5.1 项目设计

项目采用基于模板的方法实现基于知识图谱的智能问答服务。基于模板的方法又称为基于模式匹配的方法，通常包括模板定义、模板生成和模板匹配几个步骤。此方法简化了问句分析的过程，通过预定义的模板替代了本体映射，而且模板查询速度快，准确率高，且人为可控。由于这个特点，它在各种场景中得到广泛应用。

5.1.1 需求分析

在日常生活中，人与人之间保持着各种关系，可以是父母儿女、亲戚朋友，可以是老师/同学/校友，也可以是公司同事或者上级等。人物关系图还可以是自己喜欢的电影或书籍中人物关系的呈现。

智能问答以自然语言对话的形式，统一入口快速便捷完成用户查询诉求。无论是信息检索，还是简单的业务办理，用户只需提出问题，智能系统会自动识别自然语言，理解用户的真实意图，获取相关的知识，通过推理计算形成自然语言表达的答案并及时回复，缩短了用户获取价值信息的时间，改善了用户体验。

例如，用户提问“苏轼是哪里人”时，人物关系知识图谱自动根据关键词“苏轼”定位找到相应的实体，进而找到“出生地”属性得出答案返回给用户。当前，知识问答技术广泛应用于智能对话系统、智能客服或智能助理等服务领域。它不但是知识图谱的重点应用之一，而且是自然语言处理的重要研究方向。

人物关系问答系统主要用于实现两个功能：一是构建人物关系知识图谱；二是对人物的查找与智能问答。用户输入想要查询的人物名称，系统自动返回所查询人物的相关信息。系统拥有问句分析功能，对用户提出的问题进行分析并提取问句中的关键词。对于问题模板外的问题不进行回答，并且提示用户输入与人物相关的问题。

项目从网络上搜集人物信息，将人物相关的知识进行提取、汇总，并构建起人物与人物、人物与其他对象间的内部联系，形成体系化的知识图谱，并利用中文分词技术对用户的自然语言问题进行分析，基于构建的知识图谱进行相关知识搜索，并通过推理将正确答案返回给用户。

本章利用知识图谱把复杂的人物关系结合在一起，通过问答服务可以为用户找出想要获取的某人的准确信息，以及和其他人之间的关系，为用户提供更有价值的深层次信息。问答系统构建的整体思路如下。

（1）将用户输入的问题与预设问题模板进行匹配，判断用户询问的问题属于哪一问题类别。

（2）对用户输入的内容进行理解，提取问句中的实体内容。本例中提取的是人物。

（3）结合问题类别和人物名称构建 Cypher 查询语句，调用知识图谱返回查询的结果。

（4）将返回的查询结果匹配至相应的回复语句，输出问题的答案，结束此次问答的整个过程。

5.1.2　工作流程

基于知识图谱的人物关系智能问答系统以 Web 系统的外在形式呈现给用户，通过网页端提供中国近代历史人物知识智能问答服务，其系统架构如图 5-1 所示。

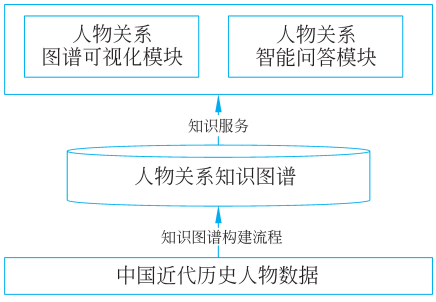

图 5-1　系统架构

该智能问答系统的工作流程主要包括 4 部分。

（1）知识图谱构建部分。通过采集网页知识的方式获取人物关系数据，经数据清洗、预处理等操作后，导入到图数据库 Neo4j 中。本章选择使用程序的方式将人物关系数据导入 Neo4j 数据库中，形成人物关系的知识库。

（2）问题分析部分。针对用户输入的问句进行模式匹配，识别出问题类型词、实体词，以及实体和实体之间的依赖关系。

（3）查询结果部分。预先建立好查询模板，其中包含一些空槽位，通过前序模块的处理结果进行填槽，形成完整的查询语句。

（4）结果返回部分。如果在 Neo4j 图数据库中能找到答案，就返回相应的结果；如果查找失败，就返回自定义信息。

从图 5-2 可知，用户输入问题，系统将对问题进行分词处理，识别问句中的实体、问题类型，将处理的结果与问句模板进行匹配，得到意图分类值，再从已设定的分类数据库中得到对应的分类 Cypher 语句，替换实体值，执行 Cypher 语句后得到查询结果，最后将结果返回给用户。

5.1.3　技术选型

本章主要使用了 Flask 框架、BootStrap 前端开发框架、Jieba 分词等技术。下面对几种技术做一个简要说明。

1. Flask 框架

Flask 是当今较为流行的轻量级 Web 框架，由 Python 语言编写而成。Flask 最显著的特点是“微”框架，轻便灵活的同时又易于扩展。默认情况下，Flask 不会指定数据库和模板引擎等对象，开发者可以根据需要自己选择各种数据库。Flask 自身不提供表单验证功能，在项目实施过程中可以自由配置，从而为应用程序开发提供数据库抽象层基础组件，支持进

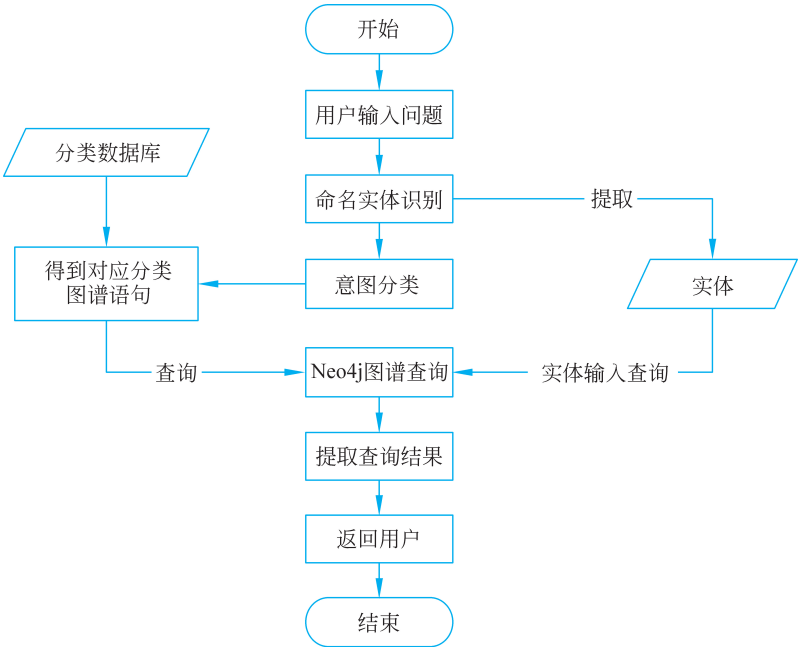

图 5-2 人物关系知识问答流程图

行表单数据合法性验证、文件上传处理、用户身份认证和数据库集成等功能。Flask 主要包括 Werkzeug 和 Jinja2 两个核心函数库，分别负责业务处理和安全方面的功能。

Flask 的工作流程如图 5-3 所示，在 Flask 中每个 URL 代表一个视图函数。当用户访问这些 URL 时，系统会调用相应视图函数，并将结果返回给用户。

图 5-3 Flask 的工作流程

2. BootStrap 前端开发框架

BootStrap 是基于 HTML、CSS、JavaScript 开发的简洁、直观、强悍的前端开发框架，使得 Web 开发更加快捷。目前，BootStrap 是最受欢迎的 HTML、CSS 和 JS 框架，用于开发响应式布局、移动设备优先的 Web 项目。BootStrap 让前端开发更快速、简单。市面上常见的所有设备都可以适配、所有项目都适用、所有开发者都能快速上手，因此受到广泛欢迎。

BootStrap 主要包括如下内容。

(1) 基本结构：提供了一个带有网格系统、链接样式、背景的基本结构。

(2) CSS：自带多个特性，如全局的 CSS 设置、定义基本的 HTML 元素样式、可扩展的 class，以及一个先进的网格系统。

(3) 组件：包含了十几个可重用的组件，用于创建图像、下拉菜单、导航、警告框、弹出框等。

(4) JavaScript 插件：包含了十几个自定义的 jQuery 插件。可以直接包含所有的插

件，也可以逐个包含这些插件。

（5）定制：可以定制组件、LESS 变量和 jQuery 插件来得到自己的版本。

3. Jieba 分词

Jieba 是 Python 中文分词组件。Jieba 分词的算法为：基于前缀词典实现高效的词图扫描，生成句子中汉字所有可能成词情况所构成的有向无环图（DAG）；采用了动态规划查找最大概率路径，找出基于词频的最大切分组合；对于未登录词，采用了基于汉字成词能力的隐马尔可夫模型（Hidden Markov Model，HMM）进行组词，并使用了 Viterbi 算法。

Jieba 库的主要功能包括分词、添加自定义词典、关键词提取和词性标注。它支持 3 种分词模型：精确模式、全模式和搜索引擎模式。开发者可以指定自己定义的词典，以便包含 Jieba 词库里没有的词。虽然 Jieba 有新词识别能力，但是自行添加新词可以保证更高的正确率。

5.1.4　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、miniconda。

（3）数据库：Neo4j＋py2neo 驱动。

（4）开发工具：PyCharm/Sublime Text 3 等。

（5）Python Web 框架：Flask。

（6）浏览器：Chrome 浏览器。

2. 文件夹组织结构

本章采用 Flask Web 框架进行开发。由于 Flask 框架的灵活性，可以任意组织项目的目录结构。在本项目中，使用包和模块的方式组织程序。文件夹组织结构如下所示。

```
├─person_school_works
|   ├─common                    #常用工具
|   |   ├─conn_neo4j.py         #连接 Neo4j 数据库
|   |   ├─get_config.py         #读取配置信息
|   |   ├─constant.py           #常量配置
|   |   ├─nlp_util.py           #自然语言处理工具
|   |   ├─file_util.py          #文件处理工具
|   ├─data                      #存放问题模板、数据库配置信息、Jieba 自定义词典等文件
|   |   ├─question              #用于存储问题模板文件
        |   ├─handler           #数据处理文件
        |   ├─model             #问题分类、问题回复模板
        |   ├─service           #业务处理
        |   ├─static            #静态资源文件
        |   ├─templates         #存放 HTML 模板
        |   |   ├─index.html    #主页面
        |   ├─app.py            #启动文件
        |   ├─README.md
        |   ├─requirements.txt      #依赖包文件
```

5.2 数据准备和预处理

本章的数据选用 OpenKG.CN 网站提供的近代历史人物数据，并对人物数据格式进行分析和整理。根据提供的人物属性信息创建问题分类及其问题模板，使用 Jieba 分词对问题模板内容进行分词处理。

5.2.1 数据准备

在数据准备阶段，依据前期的调研结果，首先确定关系图谱涉及的人员群体范围。中国近代历史涌现了大量的杰出人物和事迹，史料文献丰富，构成了一个庞大的知识体系，本章选取 OpenKG.CN 网站提供的中国近代历史人物知识图谱数据，以中国近代历史人物作为研究对象。

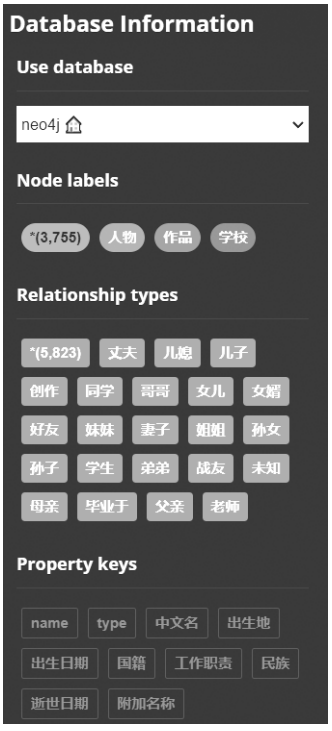

图 5-4 人物数据组成元素

从网页 http://www.openkg.cn/dataset/zgjdlsrw 下载数据到本地计算机，数据文件内容格式如图 5-4 所示。从图中可以看出，人物的数据主要包含人物的名字、国籍、出生地、生卒年月、作品、人物关系等信息。

5.2.2 数据预处理

用户在提出人物相关问题时，往往会出现关于人物、学校和作品的名词，而现有分词工具并不能将这些名词完整地分开。若 Jieba 不加载用户自定义的词典，很难将“春牡”这种词从所提问题中分离出来。针对这种情况，本章制作了一个作品的自定义词典，其对应的词性为 works。

数据预处理阶段主要是生成 Jieba 分词的自定义词典，以保证作品名称分词正确。词典格式是一个词占用一行。每一行分三部分：词语、词频（可省略）、词性（可省略），中间用空格隔开，顺序不可以颠倒。文件名若为路径或二进制方式打开的文件，则文件必须为 UTF-8 编码。

生成自定义词的代码如下。

```
1.  #json 文件路径
2.  data_file_path =os.path.join(constant.DATA_DIR, "data-json.json")
3.  #自定义词典存储路径
4.  self_defining_dict_path =os.path.join(constant.DATA_DIR, 'self_define_
    dict.txt')
5.  #读取 json 文件内容
6.  content =json.load(open(data_file_path, 'r', encoding='UTF-8'))
7.
8.  with open(self_defining_dict_path, 'w', encoding='utf-8') as f:
9.      for json in content:
10.         #判断 json 中是否包含“作品”键
```

```
            if '作品' in json.keys():
                works =json['作品']
                for work in works:
                    work =re.sub("[【】《》!,。?、~@#￥%……&*()]+", "", work)
                    #将“词语 词频 词性”写入自定义词典文件
                    f.write(work +" 100 works")
                    f.write("\n")
```

生成后的自定义词典内容如下。

```
1. 实用英语文体学 100 works
2. 春牡 100 works
3. 夏荷 100 works
4. 秋菊 100 works
5. 冬梅 100 works
6. 五十述怀 100 works
7. 抗战八年回忆 100 works
```

Jieba 加载用户自定义词典后，即可将作品从问句中正确地分离出来，示例如下。

```
1. jieba.load_userdict(self_defining_dict_path)
2. text =u'《春牡》是谁创作的?'
3. clean_txt =re.sub("[\s+\.\!\/_,$%^*(+\"\')]+|[+——()?【】《》“”!,。?、~@#￥%
   ……&*()]+", "", text)
4. words =jieba.posseg.cut(clean_txt)
5. for w in words:
6.     print(w)
```

执行结果如下。

```
1. 春牡/works
2. 是/v
3. 谁/r
4. 创作/vn
5. 的/uj
```

从上述结果可以看出，Jieba 加载自定义词典后，将“春牡”看作是一个词语。

5.3　知识建模和存储

知识建模是建立知识图谱概念模式的过程，相当于关系数据库的表结构定义。为了对人物及关系知识进行合理的组织，更好地描述知识本身与知识之间的关联，需要结合人物及关系数据特点与应用特点来完成模式的定义。

5.3.1　知识建模及描述

人物关系知识图谱的构建以人物为中心，涉及组织（学校）、成就（作品）等维度。图谱构建的重点是梳理人物与人物、人物与学校以及人物与作品之间的关系。

从数据内容分析可以得知人物与人物之间的关系可细分为 7 大类、21 个具体关系，具体内容如下所示。

```
1.'亲子':['儿子', '女儿', '父亲', '母亲'],
2.'祖孙':['孙子', '孙女', '爷爷', '奶奶'],
3.'兄弟':['哥哥', '妹妹', '弟弟', '姐姐'],
4.'配偶':['丈夫', '妻子'],
5.'婿媳':['女婿', '儿媳'],
6.'师生':['学生', '老师'],
7.'其他':['战友', '同学', '好友']
```

人物与组织(学校)的关系为“毕业于”,人物与成就(作品)之间的关系为“创作”。根据上述关系可以设计问题的匹配模板,比如:

(1) 人物与人物之间的关系可以设计的问题模板有:["这个人的家庭成员有哪些?","这个人的配偶是谁?","这个人的学生有哪些人?","这个人的好友是哪些人?"……]。

(2) 人物与学校间的关系可以设计的问题模板有:["这个人毕业于哪所学校?","这个人就读的学校有哪些?","这个人在哪几所学校就读过?","这个人在哪里读书","这个人在哪里上过学"……]。

(3) 人物与成就(作品)间的关系可以设计的问题模板有:["这个人的代表作有哪些","这个人的作品有哪些?","这个人创作了哪些作品","这个人的著作有哪些?"……]。

综上所述,根据人物的信息、人物与其他实体间的关系,设计的问题如下:

```
【0】人物的简介:["nr","nr 的简介","nr 的生平","nr 的信息","nr 的基本信息","nr 的一生","nr 的详细信息","谁是 nr"]
【1】人物的出生日期:[nr 的出生日期,nr 的生日, nr 生日多少, nr 的出生是什么时候, nr 的出生是多少, nr 生日是什么时候, nr 生日什么时候, nr 出生日期是什么时候, nr 什么时候出生的, nr 出生于哪一天, nr 的出生日期是哪一天, nr 哪一天出生的]
【2】人物就读于哪所学校:["nr 就读于哪所学校", "nr 就读过哪所学校", "nr 在哪些学校就读过", "nr 在哪里上过学", "nr 的就读学校有哪些"]
【3】人物的作品:["nr 的作品有哪些","nr 的著作","nr 创作的作品是哪些","nr 创作了哪些作品","nr 的作品"]
```

对应的问题回答模板为:

```
【0】{}的信息:{}
【1】{}的出生日期是{}
【2】{}毕业的学校有:{}
【3】{}的作品有:{}
【4】{}的生日是{}
【5】{}和{}之间的是{}关系
【6】{}的作者是{}
```

5.3.2 数据存储

1. 将数据存储到 Neo4j 数据库

从数据文件 data-json.json 中分析数据的组成结构,找出所有的节点类型和节点间的关系。从数据节点结构来看,除属性“毕业于”“作品”和“相关人物”之外,其余均可看作是人物

的基本属性；而属性“毕业于”对应节点类型“学校”，属性“作品”对应节点类型“作品”，“相关人物”对应节点类型“人物”。

数据存储流程如图 5-5 所示。通过 json.load 读取数据文件内容，遍历 json 数据中的每一项，对每一项 item 执行如下操作。

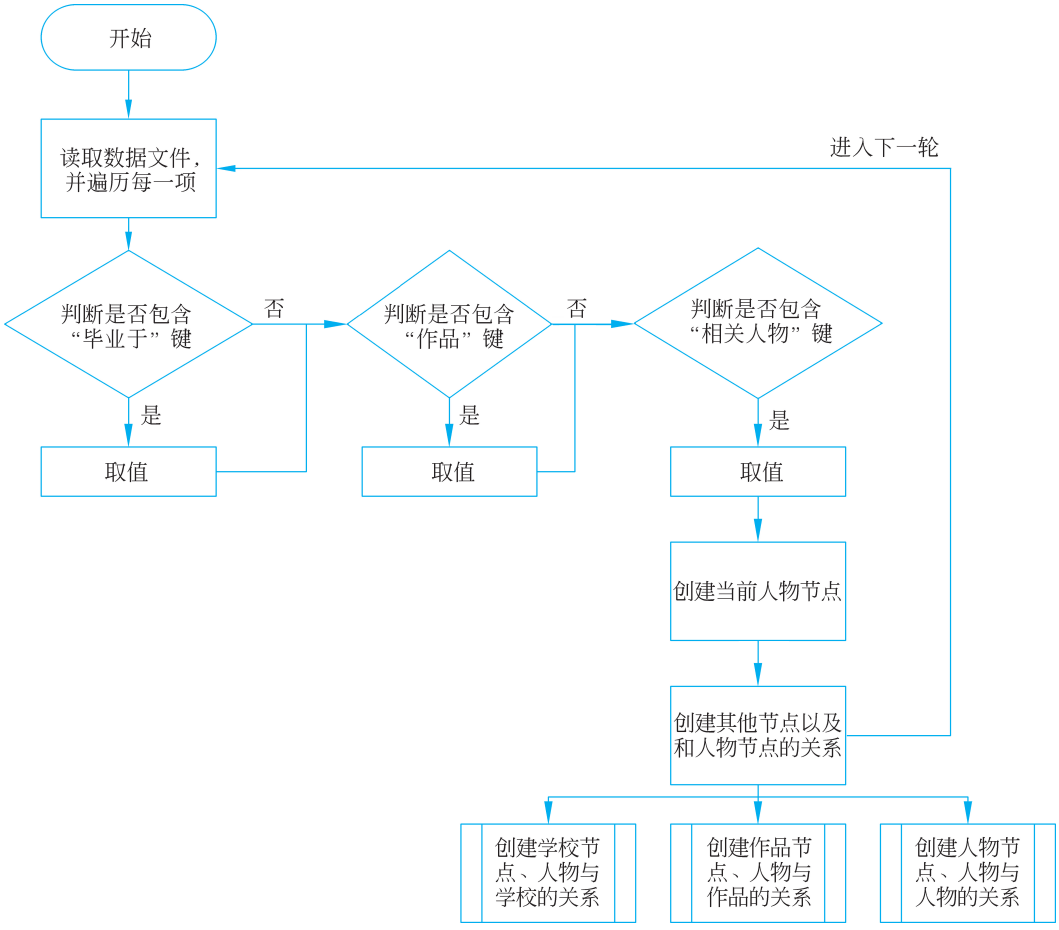

图 5-5　数据存储工作流程

（1）判断 item 字典是否包含“毕业于”键，若是则利用 item['毕业于']取值，并从 item 字典中移除“毕业于”键。

（2）判断 item 字典中是否包含“作品”键，若是则通过 item['作品']取值，并从 item 字典中移除“作品”键。

（3）判断 item 字典中是否包含“相关人物”键，若是则通过 item['相关人物']取值，并从字典中移除“相关人物”键。

（4）创建人物节点。先判断该人物节点是否存在，若不存在则通过 graph.create 方法创建，若存在则更新节点的属性值。

（5）创建学校、作品、相关人物等节点，创建人物与学校、人物与作品、人物与人物之间的关系。

关键代码如下所示。

```
#获得数据文件的路径
data_path =os.path.join(constant.DATA_DIR, "data-json.json")
#读取数据文件的内容
data =json.load(open(data_path, 'r', encoding='utf-8'))
print("人物数目: ", len(data))

#连接 Neo4j 服务器
neo4j =ConnNeo4j()
#遍历数据
    for item in data:
        item['name'] =item['中文名']
        #毕业于
        school =[]
        if '毕业于' in item.keys():
            school =item['毕业于']
            item.pop('毕业于')

        #作品
        works =[]
        if '作品' in item.keys():
            works =item['作品']
            item.pop('作品')

        #相关人物
        relate_persons ={}
        if '相关人物' in item.keys():
            relate_persons =item['相关人物']
            item.pop('相关人物')

        print(item)
        #创建人物节点
        neo4j.create_node("人物", item)
        #创建学校节点,人物与学校间的关系
        neo4j.create_node_relations("人物", item, "学校", school, "毕业于",
        {'type': '毕业于'}, False)
        #创建作品节点,人物与作品间的关系
        neo4j.create_node_relations("人物", item, "作品", works, "创作",
        {'type': '创作'}, False)
        #创建相关人物,人物社会关系
        for key in relate_persons.keys():
            tmp_value =relate_persons[key]
            tmp_rel_type =key
            if key in ['儿子', '女儿', '父亲', '母亲']:
                neo4j.create_node_relations("人物", item, "人物", tmp_value,
                tmp_rel_type, {'type': '父子'}, False)
            elif key in ['孙子', '孙女', '爷爷', '奶奶']:
```

```
44.                 neo4j.create_node_relations('人物', item, '人物', tmp_value,
                    tmp_rel_type, {'type': '祖孙'}, False)
45.             elif key in ['哥哥', '妹妹', '弟弟', '姐姐']:
46.                 neo4j.create_node_relations('人物', item, '人物', tmp_value,
                    tmp_rel_type, {'type': '兄弟姐妹'}, False)
47.             elif key in ['丈夫', '妻子']:
48.                 neo4j.create_node_relations('人物', item, '人物', tmp_value,
                    tmp_rel_type, {'type': '夫妻'}, False)
49.             elif key in ['女婿', '儿媳']:
50.                 neo4j.create_node_relations('人物', item, '人物', tmp_value,
                    tmp_rel_type, {'type': '婿媳'}, False)
51.             elif key in ['学生', '老师']:
52.                 neo4j.create_node_relations('人物', item, '人物', tmp_value,
                    tmp_rel_type, {'type': '师生'}, False)
53.             else:
54.                 neo4j.create_node_relations('人物', item, '人物', tmp_value,
                    tmp_rel_type, {'type': '其他'}, False)
```

2. 构造问题问句模板

打开浏览器，查看 Neo4j 的数据节点类型、关系类型和属性等信息。可以根据图 5-6 所表达的信息构造问题分类和问题问句模板。

图 5-6　人物图数据库属性信息

在浏览器中访问 Neo4j，查看人物、学校、作品的信息，以及节点间的关系数据。输入 Cypher 语句：

```
MATCH (n:'人物') WHERE n.name = '钱钟书' return n
```

单击“运行”按钮，可视化结果如图 5-7 所示。