《海贼王》知识图谱构建-项目报告

1. 项目背景

《海贼王》(英文名ONE PIECE) 是由日本漫画家尾田荣一郎创作的热血少年漫画，因为其宏大的世界观、丰富的人物设定、精彩的故事情节、草蛇灰线的伏笔，受到世界各地的读者欢迎，截止2019年11月7日，全球销量突破4亿6000万本[1]，并被吉尼斯世界纪录官方认证为“世界上发行量最高的单一作者创作的系列漫画”[2]。

《海贼王》从1997年开始连载至今，以及将近22年，在900多话的漫画中大量性格鲜明的角色相继登场，故事发生的地点也在不断变化，这既给我们带来阅读的乐趣，同时也为我们梳理故事脉络带来了挑战。

本次任务试图为《海贼王》中出现的各个实体，包括人物、地点、组织等，构建一个知识图谱，帮助我们更好的理解这部作品。

2. 项目内容

本项目内容包括数据采集、知识存储、知识抽取、知识计算、知识应用五大部分

1. 数据采集

   本次项目主要采集构建了两个知识图谱和一个关系抽取数据集

   - 人物知识图谱：主要包含各个人物的信息
   - 关系抽取数据集：标注出自然语言中存在的实体以及他们之间的关系
   - 实体关系知识图谱：构建《海贼王》中各个实体之间关系的知识图谱
2. 知识存储

   尝试使用了三元组数据库**Apace Jena**和原生图数据库**Neo4j**，并分别使用RDF结构化查询语言**SPARQL**和属性图查询语言**Cypher**，在知识图谱上进行查询。
3. 知识抽取

   基于之间构建的关系抽取数据集，利用**deepke**中提供的工具进行关系抽取实践，测试了包括PCNN、GCN、BERT等模型在我们构建数据集上的效果
4. 知识计算
   - 图计算：在Neo4j上对实体关系知识图谱进行了图挖掘，包括最短路径查询、权威结点发现、社区发现等
   - 知识推理：在Apache Jena上对关系知识图谱进行了知识推理，补全了一部分的数据
5. 知识应用
   - 智能问答：基于**REfO**实现一个对于《海贼王》中人物的知识库问答系统(KBQA)。
   - 可视化图片：通过D3对实体关系图片进行可视化，并整合了人物知识图谱中的信息，进行展示。

3. 数据采集

3.1. 数据来源

本次项目中所使用的数据主要来源为两个：一个是从别的知识图谱中获取已经存在的知识信息，另一个是从相关网页中爬取解析半结构化的自然语言文本信息

3.2. 人物知识图谱构建

3.2.1. 抽取通用知识图谱中已有的目标域知识

知识图谱技术近些年来快速发展，一些公司机构已经构建了许多通用知识图谱，我们可以从中抽取出我们目标领域内相关的实体知识信息，作为我们知识图谱的冷启动数据。

CN-DBpedia[3]是由复旦大学知识工场[4]实验室研发并维护的大规模通用领域结构化百科，我们选择其作为通用知识图谱来源。

整个处理流程如下：

1. 构建《海贼王》实体词汇库
2. 获取实体列表

3. 筛选实体列表
4. 获取图谱中对应实体的三元组知识

构建《海贼王》实体词汇库

主要通过领域WiKi获取《海贼王》中的实体词汇库。在这里，我们在萌娘百科的相关页面[5]中获取由粉丝爱好者整理的词条名信息，作为词汇库。

我们将原始的半结构化词条数据保存在`cndbpedia/data/raw_moegirl_onepiece_entries.txt`中，并利用正则表达式对其进行解析

```
python cndbpedia/parse_raw_moegirl_onepiece_entries.py
```

输出的结果保存在`cndbpedia/data/processed_moegirl_onepiece_entries.txt`中，一共提取了**509个词条名**

获取实体列表

我们利用知识工厂提供的API[6]，将词条名作为输入实体指称项名称(mention name)，获取返回对应实体(entity)的列表。

```
python cndbpedia/get_onepiece_cndbpedia_entities.py
```

总共获取了**1014个**不同的实体名，并输出了两个文件，输出的结果保存在`cndbpedia/data`文件夹中。

- `cndbpedia_onepiece_entities_list.txt`：保存了所有识别出的CN-DBpedia中的实体名，例如

  ```
  爱德华·纽盖特（《航海王燃烧意志》游戏角色）
  爱德华·纽盖特（日本漫画《海贼王》中的角色）
  爱莎（《弒神者！》中的弒神者之一）
  爱莎（《海贼王》中的角色）
  爱莎（艾尔之光游戏人物）
  ```

- `moelgirl_cndbpedia_entities_mapping.json`：保存着从moegirl的的条目作为实体指称项名称，在api上查找到的对应的实体列表，例如

  ```
  "夏奇": [
      "夏奇（日本动漫《海贼王》角色）",
      "夏奇（福建人民艺术剧院主持人）",
      "夏奇（深圳市夏奇实业有限公司）",
      "夏奇（《永嘉诗联》主编）"
  ],
  "布拉曼克": [
      "布拉曼克"
  ],
  "艾佛兰德拉": [
      "艾佛兰德拉"
  ],
  "顶上战争": [
      "大事件（漫画《海贼王》中顶上战争）"
  ```

```
],
"堪十郎": [
    "堪十郎"
],
```

筛选实体列表

由于自然语言和现实世界的多义性，往往一个mention name可能对应着知识图谱中的多个不同实体。就拿 `布鲁克` 这个名字来说，在api返回的实体列表中，就有好多不同的实体

```
布鲁克
布鲁克（奥地利城市穆尔河畔布鲁克缩写）
布鲁克（广告策划师）
布鲁克（日本动漫《海贼王》中的人物）
布鲁克（温力铭演唱歌曲）
布鲁克（游戏《赛尔号》中的精灵）
布鲁克（西班牙2010年拍摄电影）
```

而其中第四个才是我们需要的。

因此我们可以设置一些筛选条件，只有当实体名中**包含：** `海贼王，航海王，海贼，航海，onepiece，one piece，动漫，漫画` 这些关键词之一时，才认为是我们需要的实体

```
python cndbpedia/filter_moelgirl_cndbpedia_entities_mapping_file.py
```

输出的结果保存在 `cndbpedia/data` 文件夹中

筛选结果：在509个词条中

- 有162个词条在CN-DBpedia没有对应的实体名，这些词条被保存在 `moelgirl_cndbpedia_api_no_results_mention_name_list.txt`；
- 有11个词条虽然有实体名，但所有对应实体名中都没有包含上面提到的关键词，这些词条被保存在 `filter_out_entities_mapping.json`
- 剩余336个词条中，都有对应符合条件的实体名，一共有357个。这些词条被保存在 `query_avpair_entities_list.txt`，此外 `query_avpair_entities_mapping.json` 中保存着这些合法词条名和实体名对应的字典。

获取图谱中对应实体的三元组知识

我们利用知识工厂提供的API[6]，根据前面筛选的实例列表，获取图谱中对应实体的三元组知识

```
python cndbpedia/get_onepiece_cndbpedia_avpair.py
```

输出结果保存在 `cndbpedia/data` 文件夹中

- `query_avpair_cndbpedia_onepiece_results.json`：保存着每个实体对应的三元组知识的字典，采用两级索引结构，第一级索引是mention name，第二级索引是实体名字，示例如下

```
"砂糖": {
        "砂糖（《海贼王》人物）": {
            "性别": "女",
            "配音": "詹雅菁（台湾）",
```

```
            "中文名": "砂糖",
            "登场作品": "海贼王",
            "初次登场": "漫画第682话、动画608话",
            "恶魔果实": "超人系童趣果实",
            "职位": "唐吉诃德家族梅花军特别干部",
            "外文名称": "Sugar",
            "年龄": "外貌年龄10岁，真实年龄22岁",
            "CATEGORY_ZH": "人物",
            "DESC": "砂糖是日本动漫《海贼王》中的人物，童趣果实能力者。唐吉诃德家族干部，隶属梅花军托雷波尔。被家族视为重要的干部，多弗朗明哥为此特别安排家族最高干部托雷波尔担任她的贴身保镖。"
        }
    },
```


- `query_avpair_keys_list_file.txt`：保存在所有属性名称的列表

3.2.2. 抽取网页中半结构化的知识

生命卡(vivre card)[7]是海贼王官方整理发布的角色资料图鉴，包含着丰富的角色信息。国内的粉丝爱好者也将其翻译成了中文版本，并发布在了网页上[8]。这部分就是希望抽取Talkop论坛中相关网页中存在的半结构化信息，构建对应人物的知识图谱。

抽取流程

由于格式较为固定，因此采用**模板匹配**的方式来抽取知识，整个流程如下：

1. 从网页中获取原始文本信息
2. 人工删除不相关的文本
3. 利用代码以模板匹配的方式，自动抽取人物属性信息

   ```
   cd talkop
   python parse_processed_manual_talkop_vivre_card.py
   ```

   输出的文件保存在`talkop/data/processed_manual_talkop_vivre_card`文件夹中，每个网页对应着三个输出文件
   - `xxx-predicate_key_list.txt`：所有解析得到的predicate
   - `xxx-entities_id_name_list.txt`：所有解析得到的id和实体名
   - `xxx-entities_avpair.json`：抽取到所有实体的属性知识，以json的格式保存
4. 人工校验：例如：查看是否抽取到了所有的实体、通过查看抽取的predicate结果来调整模板。整个过程中是代码自动抽取和人工校验构成闭环的过程，在闭环过程中不断补充模板信息，改善抽取结果

在整个过程中，2、3、4是不断循环往复的过程，直至抽取的知识满足我们的需要。

汇总结果

在上面部分中，我们分别抽取了各个网页中人物实体的属性信息，现在将这些信息进行进一步的汇总

```
cd talkop
python parse_processed_manual_talkop_vivre_card.py
```

从汇总的结果可以看到，一共包含**660个不同的实体，164个不同的predicate**

输出的文件保存在 `talkop/data/processed_manual_talkop_vivre_card` 文件夹中，一共有两个文件：

- `summary_predicate_set.txt`：所有predicate的汇总
- `summary_entities_id_name_list.txt`：所有抽取得到的实体名以及对应ID的汇总

3.3. 关系抽取数据集构建

- **标注数据来源**：在前面构建的人物知识图谱中，有一项重要的属性是**历史信息**，记录着每个人物在故事中的时间线以及对应的故事。每个人的历史信息记录着其**与其他实体之间交互的信息**，我们可以利用它来构建我们垂直领域内的关系抽取数据集
- **标注工具**：精灵标注助手[8]
- **构建方法**：自底向上构建，在构建过程中逐步构建整个图谱的schema
- **数据标注格式**：精灵标注助手提供导出json格式，其具体形式如下所示，其中 `T` 和 `E` 分别表示标注出的实体信息和关系信息


```
{
    "content": "xxxx"
    "labeled": true,
    "outputs": {
        "annotation": {
            "A": [""],
            "E": [""],
            "R": ["",{
                    "arg1": "Arg1",
                    "arg2": "Arg2",
                    "from": 1,
                    "name": "到过",
                    "to": 2
                },
            ],
            "T": ["",{
                    "attributes": [],
                    "end": 7,
                    "id": 1,
                    "name": "人",
                    "start": 0,
                    "type": "T",
                    "value": "蒙其·D·路飞"
                },
            ]
        }
    },
    "path": "D:\\annot\\fuseki_vivrecard_sentence_item.txt",
    "time_labeled": 1578072175246
}
```


- **数据存储位置**： 被标注的原始数据被保存在 `deepke-master/data/vivrecard/rawfuseki_vivrecard_sentence_item.txt` 原始标注结果被保存在 `deepke-master/data/vivrecard/annot/outputs/fuseki_vivrecard_sentence_item.json`。

  为了方便后续关系抽取模型处理，我们将标注数据转为符合deepke项目格式的数据

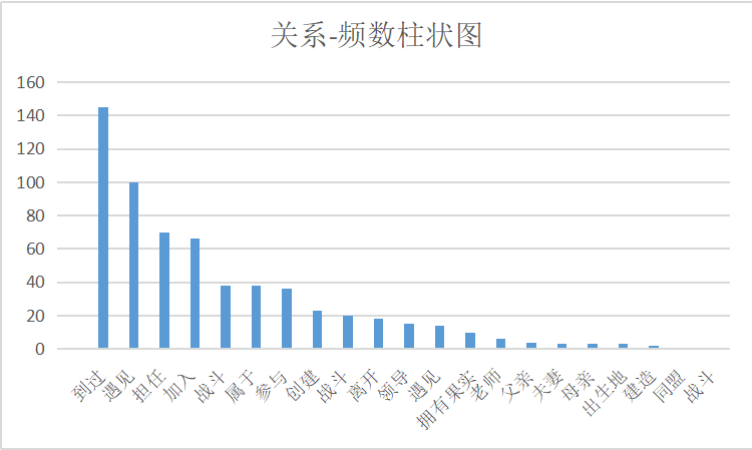
并保存在 `deepke-master/data/vivrecard/origin`，具体详情参见知识抽取部分

3.1. 数据集统计信息

- **实体类型**：一共**7种**实体：'事件', '组织', '船只', '地点', '职务', '恶魔果实', '人'
- **关系类型**：一共**22种**关系

head_type	tail_type	relation	index	freq
None	None	None	0	0
人	事件	参与	1	36
人	人	同盟	2	1
人	人	夫妻	3	3
人	人	战斗	4	38
人	人	母亲	5	3
人	人	父亲	6	4
人	人	老师	7	6
人	人	遇见	8	100
人	地点	出生地	9	3
人	地点	到过	10	145
人	恶魔果实	拥有果实	11	10
人	组织	创建	12	23
人	组织	加入	13	66
人	组织	属于	14	38
人	组织	战斗	15	20
人	组织	离开	16	18
人	组织	遇见	17	14
人	组织	领导	18	15
人	职务	担任	19	70
人	船只	建造	20	2
组织	组织	战斗	21	1

这些关系的频数柱状图如下图所示，可以看到这些关系展现出明显的**长尾分布**



- 训练正样本个数：616个

3.4. 实体关系知识图谱构建

在进行关系抽取数据集的标注过程中，我们将标注的实体和关系单独导出，构建《海贼王》实体关系数据集

在上述过程中，一共标注了**307个**不同的实体，**569个**不同结点间的关系

```
cd deepke-master
python utils/convert_vivrecard2deepke.py
```

输出的实体关系数据保存在 `deepke-master/data/vivrecard/summary/vizdata_vivrecard_relation.json`，可用于后续进行知识图谱可视化，具体参见知识图谱可视化部分

4. 知识存储

4.1. 基于RDF 三元组数据库：Apache Jena

4.1.1 Jena 简介[9]

Jena[10]是 Apache 顶级项目,其前身为惠普实验室开发的 Jena 工具包.Jena 是语义 Web 领域主要的开源框 架和 RDF 三元组库,较好地遵循了 W3C 标准,其功能包括:RDF 数据管理、RDFS 和 OWL 本体管理、SPARQL 查询处理等.Jena 具备一套原生存储引擎,可对 RDF 三元组进行基于磁盘或内存的存储管理.同时,具有一套基 于规则的推理引擎,用以执行 RDFS 和 OWL 本体推理任务.

4.1.2. 项目实践

avpair to triple

以vivrecard人物属性知识图谱为例，首先我们将之前获得的数据，转换为Jena支持解析的 `N-Triple` 三元组格式，命名空间前缀为 `<http://kg.course/talkop-vivre-card/>`

```
cd talkop
python avpair2ntriples_talkop_vivre_card.py
```

导出的 `N-Triple` 格式的数据保存在 `talkop/data/processed_manual_talkop_vivre_card/ntriples_talkop_vivre_card.nt`，一共有**14055个**，其中非空triples有12863个

NOTE:

1. 在项目构建过程中，我们也将从CN-DBpedia获取的知识转换为 `N-Triple` 格式，命名空间前缀为 `<http://kg.course/onepiece/>`

   ```
   python cndbpedia/avpair2ntriples_onepiece_cndbpedia.py
   ```

   结果保存在 `cndbpedia/data/ntriples_cndbpedia_onepiece.nt`，一共有**4691个triple**

启动 Fuseki

按照陈华均老师提供文件：https://github.com/zjunlp/kg-course/blob/master/tutorials/Tutorial-Jena.pdf

进一步配置fuseki，上传数据集就可以查询了

4.1.3 SPARQL查询示例

SPARQL[11] 是 W3C 制定的 RDF 知识图谱标准查询语言.SPARQL 从语法上借鉴了 SQL.SPARQL 查询的基本单元是三元组模式(triple pattern),多个三元组模式可构成基本图模式(basic graph pattern).SPARQL 支持多 种运算符,将基本图模式扩展为复杂图模式(complex graph pattern).SPARQL 1.1 版本引入了属性路径(property path)机制以支持 RDF 图上的导航式查询.下面使用图 2 所示的电影知识图谱 RDF 图,通过示例介绍 SPARQL 语言的基本功能.[9]

下面给出了使用SPARQL在我们构建的数据库上进行查询的示例

1. 查询前五个角色的身高

   ```
   PREFIX : <http://kg.course/talkop-vivre-card/>
   select ?s ?name ?zhname ?height ?o where {
       ?s ?height ?o .
       FILTER(?height in (:身高, :身长)) .
       OPTIONAL { ?s :名称 ?name. ?s :外文名 ?zhname.}
   }
   limit 5
   ```

   结果

   ```
    "s" , "name" , "zhname" , "height" , "o" ,
    ":0001" , "【蒙其·D·路飞/Monkey D Luffy】" , "Monkey D Luffy" , ":身高" ,
   "174cm" ,
    ":0004" , "【乌索普/Usopp】" , "Usopp" , ":身高" , "174cm" ,
    ":0511" , "【乔艾莉·波妮/Jewelry Bonney】" , "Jewelry Bonney" , ":身高" ,
   "174cm" ,
    ":0002" , "【罗罗诺亚·索隆/Roronoa Zoro】" , "Roronoa Zoro" , ":身高" ,
   "181cm" ,
    ":0224" , "【缇娜/Hina】" , "Hina" , ":身高" , "181cm" ,
   ```

2. 筛选生日范围

```
PREFIX : <http://kg.course/talkop-vivre-card/>
select ?s ?name ?o where {
    ?s :生日 ?o .
    ?s :名称 ?name .
    filter(?o > '4月1日' && ?o < '5月1日')
}
limit 5
```

结果

```
 "s" , "name" , "o" ,
 ":0009" , "【布鲁克/Brook】" , "4月3日" ,
 ":0660" , "【伯尔杰米/Porchemy】" , "4月3日" ,
 ":0010" , "【甚平/Jinbe】" , "4月2日" ,
 ":0076" , "【哲夫/Zeff】" , "4月2日" ,
 ":0028" , "【克比/Koby】" , "5月13日" ,
```

4.2. 基于原生图数据库：Neo4j

4.2.1. Neo4j简介

Neo4j [12] 是由 Neo 技术公司开发的图数据库.可以说,Neo4j 是目前流行程度最高的图数据库产品.Neo4j 基 于属性图模型,其存储管理层为属性图的节点、节点属性、边、边属性等元素设计了专门的存储方案.这使得 Neo4j 在存储层对于图数据的存取效率优于关系数据库.

4.2.2. 项目实践

relation to triple

以实体关系知识图谱为例，首先我们将之前获得的各个实体之间关系的数据，转换为Jena支持解析的 `N-Triple` 三元组格式，命名空间前缀为 `<http://kg.course/talkop-vivre-card/deepke/>`

```
cd deepke-master
python utils/convert_vivrecard2deepke.py
```

导出的 `N-Triple` 格式的数据保存在 `deepke-master/data/vivrecard/summary/vivrecard_ntriples.nt`，一共有**1848个**

启用 Neo4j

Neo4j的下载安装可以参考： https://neo4j.com/download-thanks-desktop/?edition=desktop&flavour=winstall64&release=1.2.4&offline=true

```
cd D:\neo4j\bin
neo4j.bat console
```

之后访问：`http://localhost:7474/` 就可以了

默认的用户名和密码都是 `neo4j`

4.2.3. Cypher查询示例

Cypher 最初是图数据库 Neo4j 中实现的属性图数据查询语言，是一种声明式的语言，用户只需要声明查什么，而不需要关系怎么查。

下面给出了使用Cypher在我们构建的数据库上进行查询的示例

1. 导入

```
CREATE INDEX ON :Resource(uri)

CALL semantics.importRDF("${PROJECT_PATH}/deepke-master/data/vivrecard/summary/vivrecard_ntriples.nt","N-Triples")
```

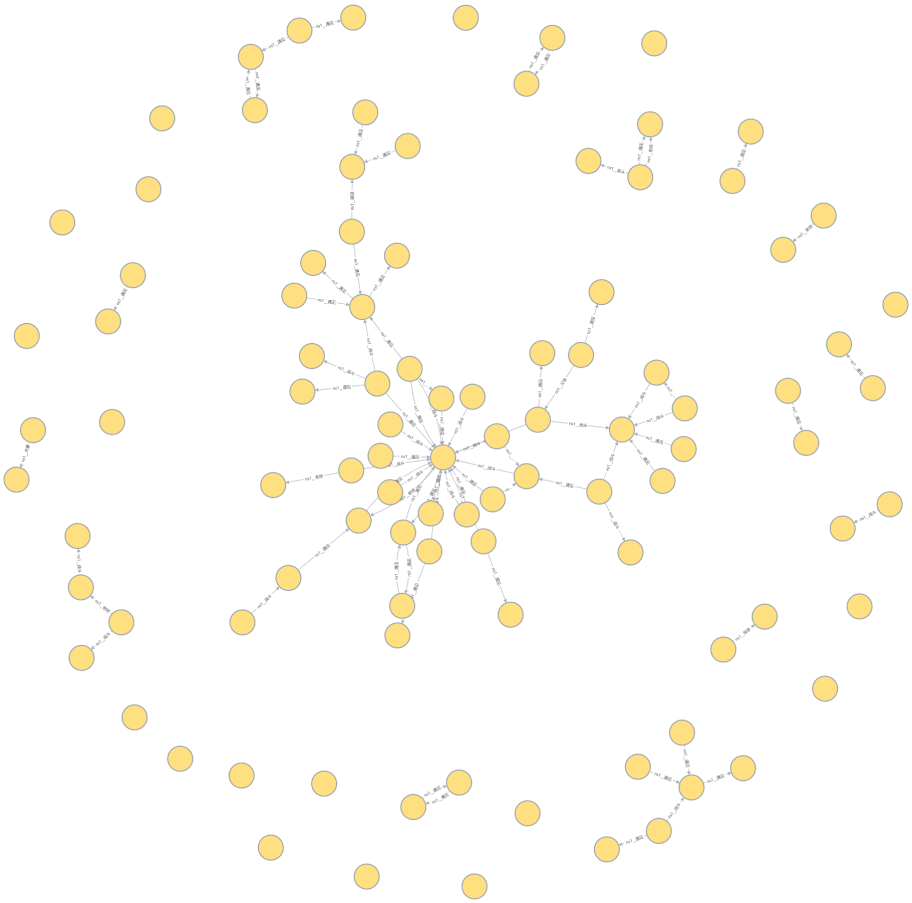
2. 查看schema

```
call db.schema()
```

把resource屏蔽掉，就能清楚的看到schema了

3. 查询前100个人

```
MATCH (n:ns0__人) RETURN n LIMIT 100
```



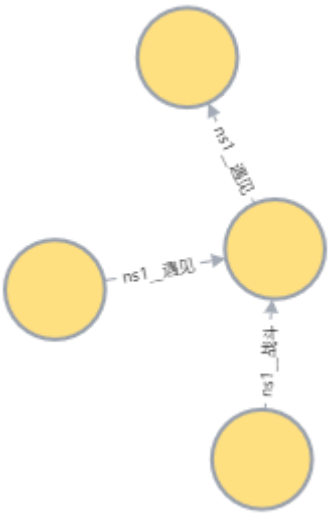
4. 查询属于人的结点中，URI里面包含 `薇薇` 的结点

```
MATCH (n:ns0__人)
WHERE n.uri CONTAINS '薇薇'
RETURN n.uri
```

n.uri
"http://kg.course/talkop-vivre-card/deepke/人/寇布拉•薇薇"
"http://kg.course/talkop-vivre-card/deepke/人/奈菲鲁塔丽•薇薇"
"http://kg.course/talkop-vivre-card/deepke/人/薇薇"

5. 根据**uri**筛选名字间最短路径

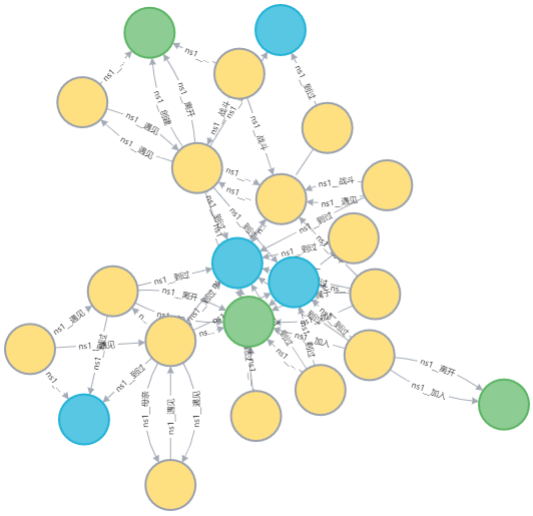
```
MATCH p=shortestPath(
(n1)-[*]-(n2)
)
WHERE n1.uri CONTAINS '斯摩格' and n2.uri CONTAINS '罗宾'
RETURN p
```



6. 根据名字筛选德雷斯罗萨到司法岛里面**四跳的路径**

```
# 五跳内能到的所有路径
# 9312 多萝菲（Miss.圣诞快乐）
# 9306 本·贝克曼
MATCH p = ((n1)-[*4]-(n2))
WHERE n1.uri CONTAINS '司法岛' and n2.uri CONTAINS '德雷斯罗萨'
RETURN p
```

可以发现这里面存在一些环路的情况，即同一个结点在路径中出现两次



5. 知识抽取

DeepKE[13] 基于 Pytorch 的深度学习中文关系抽取处理套件。在这部分中我们利用之前构建的关系抽取数据集和deepke，进行中文关系抽取实践

5.1. 数据转换&标注统计

在这部分，我们需要完成以下三部分内容：

1. 将我们的标注结果转换为deepke所接收的格式
2. 为了保证关系分布均匀，将数据随机打乱
3. 完成训练集、测试集、验证集的划分，目前按 **7:2:1进行划分**

使用 `deepke-master/utils/convert_vivrecard2deepke.py` 完成数据格式转换

```
cd deepke-master
python utils/convert_vivrecard2deepke.py
```

输出

一共有616个训练正样本，其中train、test、valid分别有：431/123/62个

输出的文件保存在 `deepke-master/data/vivrecard/` 中的 `origin` 和 `summary` 文件夹中

```
├── annot
|   └── outputs
|       └── formatted_fuseki_vivrecard_sentence_item.json # 对json文件进行缩进等格式化
├── origin                                                 # 输出转换得到deepke训练数据到该文件夹
|   ├── relation.csv
|   ├── test.csv
|   ├── train.csv
|   └── valid.csv
└── summary
    ├── all_sent.txt                        # 所有的句子
    ├── annot_entity_sent.txt               # 被标记上实体的句子
    ├── annot_relation_sent.txt             # 被标记上关系的句子
    ├── entities_type_name_dict.json        # 标注数据中所有的实体类型，以及属于该类型的所有实体名字
    ├── relation.csv                        # 标注数据中的存在的所有数据
    ├── unannot_entity_sent.txt             # [未被]标记上实体的句子
    └── unannot_relation_sent.txt           # [未被]标记上关系的句子
```

5.2. 训练

在训练过程中我们尝试使用了deepke所提供的PCNN, rnn, gcn, capsule, transformer, bert 这些模型，epoch 设置为 50，`num_relations` 根据我们数据集的实际情况修改为19，需要注意的是基于BERT的语言模型进行训练时，需要先在相关网页[14]下载好预训练模型

新的数据集有**22种**关系(包括None)，需要通过 `num_relations` 来更改

```
cd deepke-master

python main.py show_plot=False data_path=data/vivrecard/origin
out_path=data/vivrecard/out num_relations=22 epoch=50 model=cnn

python main.py show_plot=False data_path=data/vivrecard/origin
out_path=data/vivrecard/out num_relations=22 epoch=50 model=rnn

python main.py show_plot=False data_path=data/vivrecard/origin
out_path=data/vivrecard/out num_relations=22 epoch=50 model=gcn

python main.py show_plot=False data_path=data/vivrecard/origin
out_path=data/vivrecard/out num_relations=22 epoch=50 model=capsule
```

```
python main.py show_plot=False data_path=data/vivrecard/origin
out_path=data/vivrecard/out num_relations=22 epoch=50 model=transformer

# lm bert layer=1
python main.py show_plot=False data_path=data/vivrecard/origin
out_path=data/vivrecard/out num_relations=22 epoch=50 model=lm
lm_file=~/ZJU_study/Knowledge_Graph/deepke/pretrained/ num_hidden_layers=1

# lm bert layer=2
python main.py show_plot=False data_path=data/vivrecard/origin
out_path=data/vivrecard/out num_relations=22 epoch=50 model=lm
lm_file=~/ZJU_study/Knowledge_Graph/deepke/pretrained/ gpu_id=0
num_hidden_layers=2


# lm bert layer=3
python main.py show_plot=False data_path=data/vivrecard/origin
out_path=data/vivrecard/out num_relations=22 epoch=50 model=lm
lm_file=/home/zenghao/ZJU_study/Knowledge_Graph/deepke/pretrained/ gpu_id=1
num_hidden_layers=3
```

5.3. 训练结果

	PCNN	RNN	GCN	CAPSULE	TRANSFORMER	LM(BERT) LAYER=1	LM(BERT) LAYER=2	LM(BERT) LAYER=3
VALID	80.11	83.87	55.91	75.27	82.26	89.79	**90.86**	89.78
TEST	86.18	85.64	63.15	82.66	86.18	91.87	91.33	**92.14**

可以到基于bert的语言模型效果最好，明显由于其他模型。GCN的效果最差。这也说明在小规模数据上利用预训练的语言模型还是能够抽取到比较好的特征的。

但是在我们后面对于实际数据的预测结果发现，语言模型的泛化效果似乎不如PCNN模型的好

我们猜测是由于我们的数据存在长尾分布问题，模型可能趋向于预测某些特定关系来作弊，已达到准确率提高的效果

6. 知识计算

6.1. 图计算

知识图谱的一个很重要的特征就是其的图结构，不同实体之间的结构本身就内含着许多的隐式的信息，可以被进一步的挖掘利用。

在这部分中，我们参考他人在类似领域的实践[15][16]，利用Neo4j提供的图算法，对我们构建的实体关系知识图谱，用图算法进行一定的计算分析，包括计算最短路径、关键结点、结点中心度、社区发现等。

6.1.1. 人物网络分析

人物数量

万事以简单开始。先看看上图上由有多少人物：

```
MATCH (c:`ns0__人`) RETURN count(c)
```

count(c)
134

概要统计

统计每个角色接触的其它角色的数目：

```
MATCH (c:`ns0__人`)-[]->(:`ns0__人`)
WITH c, count(*) AS num
RETURN min(num) AS min, max(num) AS max, avg(num) AS avg_characters, stdev(num)
AS stdev
```

min	max	avg_characters	stdev
1	6	1.8374999999999997	1.1522542572790615

图（网络）的直径

网络的直径或者测底线或者最长最短路径：

```
// Find maximum diameter of network
// maximum shortest path between two nodes
MATCH (a:`ns0__人`), (b:`ns0__人`) WHERE id(a) > id(b)
MATCH p=shortestPath((a)-[*]-(b))
RETURN length(p) AS len, extract(x IN nodes(p) | split(x.uri,
'http://kg.course/talkop-vivre-card/deepke')[-1]) AS path
ORDER BY len DESC LIMIT 4
```

len	path
10	["/人/克拉巴特尔", "/职务/管家", "/人/克洛", "/职务/船长", "/人/甚平", "/事件/顶上战争", "/人/缇娜", "/事件/世界会议", "/人/Dr.古蕾娃", "/人/乔巴", "/人/Dr.西尔尔克"]
9	["/人/Dr.西尔尔克", "/人/乔巴", "/人/Dr.古蕾娃", "/事件/世界会议", "/人/伊卡莱姆", "/组织/草帽一伙", "/人/库洛卡斯", "/地点/伟大航路", "/人/哥尔·D·罗杰", "/人/西奇"]
9	["/人/Dr.西尔尔克", "/人/乔巴", "/人/Dr.古蕾娃", "/事件/世界会议", "/人/缇娜", "/组织/草帽一伙", "/人/娜美", "/组织/恶龙一伙", "/人/哞哞", "/人/卡里布"]
9	["/人/克拉巴特尔", "/职务/管家", "/人/克洛", "/职务/船长", "/人/东利", "/人/路飞", "/人/克利克", "/地点/伟大航路", "/人/哥尔·D·罗杰", "/人/西奇"]

我们能看到网络中有许多长度为9的路径。

最短路径

使用Cypher 的shortestPath函数找到图中任意两个角色之间的最短路径。让我们找出**克洛克达尔**和**加尔帝诺（Mr.3）**之间的最短路径：

```
MATCH p=shortestPath(
(n1)-[*]-(n2)
)
WHERE n1.uri CONTAINS '克洛克达尔' and n2.uri CONTAINS '加尔帝诺'
RETURN p
```

还可以对路径中的结点进行一些限制，例如路径中**不能包含某种类型的结点**

```
MATCH p=shortestPath((n1)-[*]-(n2))
WHERE n1.uri CONTAINS '克洛克达尔' and n2.uri CONTAINS '加尔帝诺' and id(n2) >
id(n1) and NONE(n IN nodes(p) WHERE n:`ns0__组织`)
RETURN p
```

路径中**只能包含某种类型的结点**

例子：所有从索隆到强尼的1到3跳的路径中，只经过人物结点的路径

```
MATCH p=(n1)-[*1..3]-(n2)
WHERE n1.uri CONTAINS '索隆' and n2.uri CONTAINS '强尼' and all(x in nodes(p)
where 'ns0__人' IN LABELS(x))
RETURN p
```

所有最短路径

联结**斯摩格**和**一本松**之间的最短路径可能还有其它路径，我们可以使用Cypher的allShortestPaths函数来查找：

```
MATCH (n1:`ns0__人`), (n2:`ns0__人`) WHERE n1.uri CONTAINS '克洛克达尔' and n2.uri
CONTAINS '加尔帝诺' and id(n2) > id(n1)
MATCH p=allShortestPaths((n1)-[*]-(n2))
RETURN p
```

6.1.2. 关键节点

在网络中，如果一个节点位于其它两个节点所有的最短路径上，即称为关键节点。下面我们找出网络中所有的关键节点：

```
// Find all pivotal nodes in network
MATCH (a:`ns0__人`), (b:`ns0__人`) WHERE id(a) > id(b)
MATCH p=allShortestPaths((a)-[*]-(b)) WITH collect(p) AS paths, a, b
MATCH (c:`ns0__人`) WHERE all(x IN paths WHERE c IN nodes(x)) AND NOT c IN [a,b]
RETURN a.uri, b.uri, c.uri AS PivotalNode SKIP 490 LIMIT 10
```

a.uri	b.uri	PivotalNode
"http://kg.course/talkop-vivre-card/deepke/人/萨奇斯"	"http://kg.course/talkop-vivre-card/deepke/人/妮可·罗宾"	"http://kg.course/talkop-vivre-card/deepke/人/路飞"
"http://kg.course/talkop-vivre-card/deepke/人/萨奇斯"	"http://kg.course/talkop-vivre-card/deepke/人/瓦波尔"	"http://kg.course/talkop-vivre-card/deepke/人/路飞"
"http://kg.course/talkop-vivre-card/deepke/人/萨奇斯"	"http://kg.course/talkop-vivre-card/deepke/人/诺琪高"	"http://kg.course/talkop-vivre-card/deepke/人/路飞"
"http://kg.course/talkop-vivre-card/deepke/人/萨奇斯"	"http://kg.course/talkop-vivre-card/deepke/人/诺琪高"	"http://kg.course/talkop-vivre-card/deepke/人/娜美"

从结果表格中我们可以看出有趣的结果：娜美和路飞是萨奇斯和诺琪高的关键节点。这意味着，所有联结萨奇斯和诺琪高的最短路径都要经过娜美和路飞。我们可以通过可视化萨奇斯和诺琪高之间的所有最短路径来验证：

```
MATCH (n1:`ns0__人`), (n2:`ns0__人`) WHERE n1.uri CONTAINS '萨奇斯' and n2.uri
CONTAINS '诺琪高' and id(n1) <> id(n2)
MATCH p=shortestPath((n1)-[*]-(n2))
RETURN p
```

6.1.3. 节点中心度

节点中心度给出网络中节点的重要性的相对度量。有许多不同的方式来度量中心度，每种方式都代表不同类型的“重要性”。

度中心性(Degree Centrality)

度中心性是最简单度量，即为某个节点在网络中的联结数。在《海贼王》的图中，某个角色的度中心性是指该角色接触的其他角色数。作者使用Cypher计算度中心性：

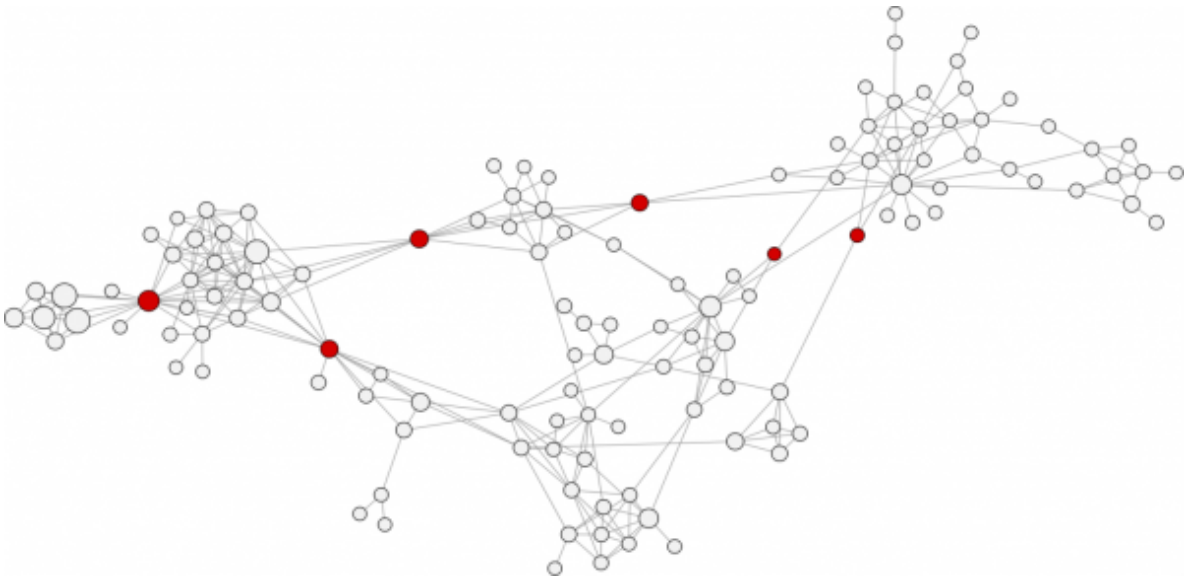
```
MATCH (c:`ns0__人`)-[]-()
RETURN split(c.uri, 'http://kg.course/talkop-vivre-card/deepke')[-1] AS
character, count(*) AS degree ORDER BY degree DESC
```

character	degree
"/人/路飞"	33
"/人/缇娜"	20
"/人/娜美"	19
"/人/山治"	15

从上面可以发现，在《海贼王》网络中路飞和最多的角色有接触。鉴于他是漫画的主角，我们觉得这是有道理的。

介数中心性（Betweenness Centrality)

介数中心性：在网络中，一个节点的介数中心性是指其它两个节点的所有最短路径都经过这个节点，则这些所有最短路径数即为此节点的介数中心性。介数中心性是一种重要的度量，因为它可以鉴别出网络中的“信息中间人”或者网络聚类后的联结点。



图中红色节点是具有高的介数中心性，网络聚类的联结点。

为了计算介数中心性，**需要安装 `algo` 库**

```
CALL algo.betweenness.stream('ns0__人', 'ns1__遇见',{direction:'both'})
YIELD nodeId, centrality

MATCH (user:`ns0__人`) WHERE id(user) = nodeId

RETURN user.uri AS user,centrality
ORDER BY centrality DESC;

或者

CALL algo.betweenness.stream('ns0__人', null,{direction:'both'})
YIELD nodeId, centrality

MATCH (user:`ns0__人`) WHERE id(user) = nodeId

RETURN user.uri AS user,centrality
ORDER BY centrality DESC;
```

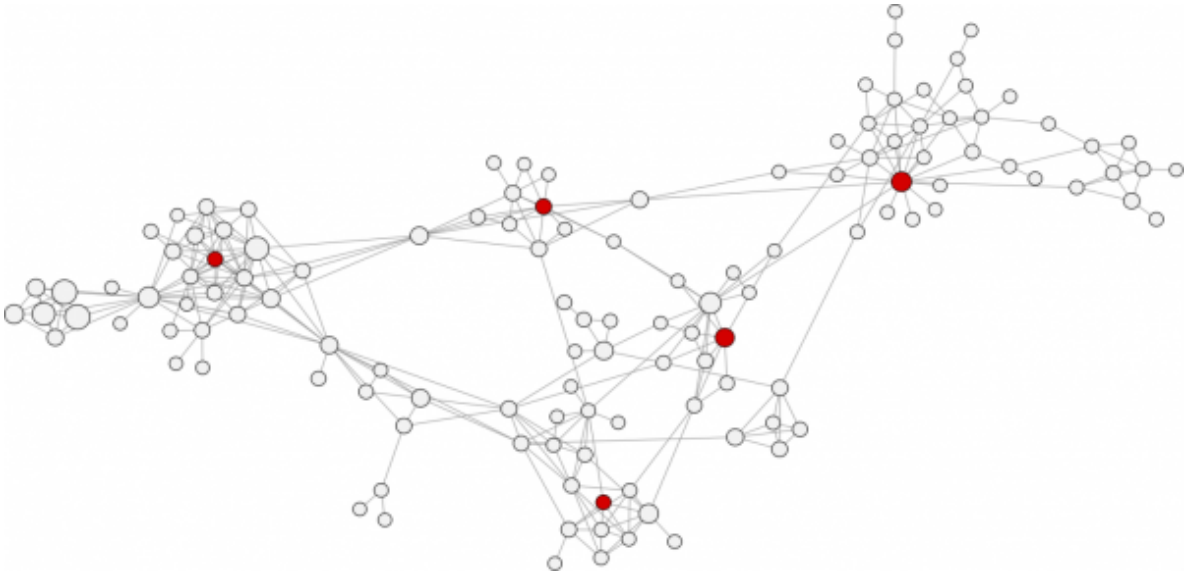
user	centrality
"http://kg.course/talkop-vivre-card/deepke/人/路飞"	759.0
"http://kg.course/talkop-vivre-card/deepke/人/缇娜"	335.0
"http://kg.course/talkop-vivre-card/deepke/人/加尔帝诺（Mr.3）"	330.0

NOTE：上面的是不考虑方向的，所以设置为 `{direction:'both'}`。如果考虑方向，可以

> - loading incoming relationships: 'INCOMING','IN','I' or '<'
> - loading outgoing relationships: 'OUTGOING','OUT','O' or '>'

紧度中心性（Closeness centrality）

紧度中心性是指到网络中所有其他角色的平均距离的倒数。在图中，具有高紧度中心性的节点在聚类社区之间被高度联结，但在社区之外不一定是高度联结的。



网络中具有高紧度中心性的节点被其它节点高度联结

```
MATCH (c:`ns0__人`)
WITH collect(c) AS characters
CALL algo.closeness.stream('ns0__人', null)
YIELD nodeId, centrality

RETURN algo.asNode(nodeId).uri AS node, centrality
ORDER BY centrality DESC
LIMIT 20;
```

node	centrality
"http://kg.course/talkop-vivre-card/deepke/人/Miss黄金周"	1.0
"http://kg.course/talkop-vivre-card/deepke/人/范德·戴肯"	1.0
"http://kg.course/talkop-vivre-card/deepke/人/杰斯"	1.0

6.1.4. 社区发现

```
CALL algo.beta.louvain.stream(null, null, {
 graph: 'huge',
 direction: 'BOTH'
}) YIELD nodeId, community, communities
RETURN algo.asNode(nodeId).uri as name, community, communities
ORDER BY community ASC
```

name	community	communities
"http://kg.course/talkop-vivre-card/deepke/人/瓦波尔"	151	null
"http://kg.course/talkop-vivre-card/deepke/组织/黑胡子海贼团"	151	null
"http://kg.course/talkop-vivre-card/deepke/地点/磁鼓岛"	151	null
"http://kg.course/talkop-vivre-card/deepke/人/克罗马利蒙"	151	null
"http://kg.course/talkop-vivre-card/deepke/组织/磁鼓王国"	151	null
"http://kg.course/talkop-vivre-card/deepke/组织/医生20"	151	null
"http://kg.course/talkop-vivre-card/deepke/人/杰斯"	151	null
"http://kg.course/talkop-vivre-card/deepke/组织/邪恶暗黑磁鼓王国"	151	null
"http://kg.course/talkop-vivre-card/deepke/人/宇宙小姐"	151	null

可以看到，基本把瓦波尔那一系列的community给检测出来了，包括在磁鼓岛和黑暗磁鼓王国

6.1.5. PageRank

```
CALL algo.pageRank.stream('ns0__人', null, {iterations:20, dampingFactor:0.85})
YIELD nodeId, score
RETURN algo.asNode(nodeId).uri AS page,score
ORDER BY score DESC
```

page	score
"http://kg.course/talkop-vivre-card/deepke/人/路飞"	2.9112886658942436
"http://kg.course/talkop-vivre-card/deepke/人/山治"	1.4952359730610623
"http://kg.course/talkop-vivre-card/deepke/人/拉布"	1.1878799288533628

6.2. 知识推理

TODO

7. 知识应用

7.1. 智能问答

在这部分中我们参考前人的工作[17][18]，基于REfO[19]实现了一个KBQA系统，主要流程为：解析输入的自然语言问句生成 SPARQL 查询，进一步请求后台基于 TDB 知识库的 Apache Jena Fuseki 服务, 得到结果。代码和数据存放在 `vivirecard-KB_query` 目录下

7.1.1. 支持的问题类型

1. 对于生日/英文名/血型/星座/霸气/身高的查询
2. 谁出生在哪里/出生在某个地方的有谁

7.1.2. 查询示例

运行 `python query_main.py` 就可以开始进行QA过程

```
cd vivirecard-KB_query
python query_main.py
```

直接输入问题，按回车后就会返回答案；当系统中没有对应知识时，会返回 `I don't know. :(`；当系统无法理解问题时会返回 `I can't understand. :(`

1. 雷利的身高是多少?

   > 188cm

2. 罗杰的血型是啥

   > S型

3. 谁出生在风车村?

   > 蒙其·D·路飞、玛琪诺、乔路叔&鸡婶、乌普·斯拉普

4. 出生在可可亚西村的有谁?

   > 娜美、诺琪高、阿健、贝尔梅尔、Dr.纳克、萨姆

5. 我想知道斯摩格的生日

   > 3月14日

6. 特朗普的生日是多少

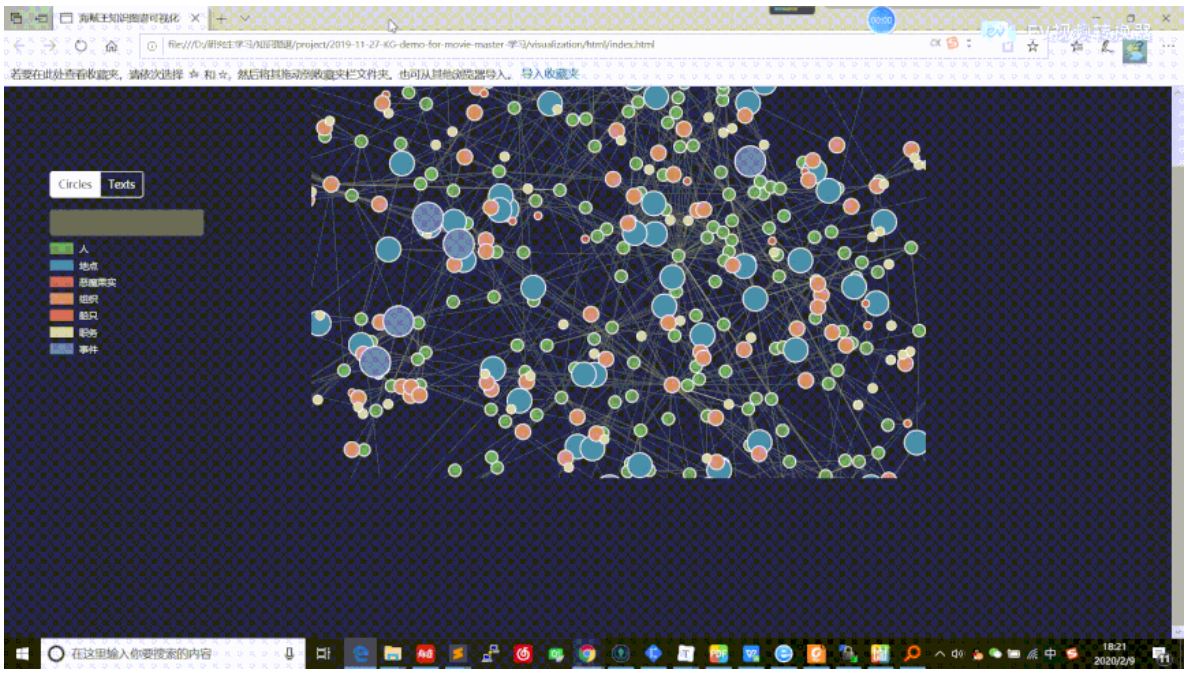
   > I don't know. :(

7. sasdasdasd

   > I can't understand. :(

7.2. 知识图谱可视化

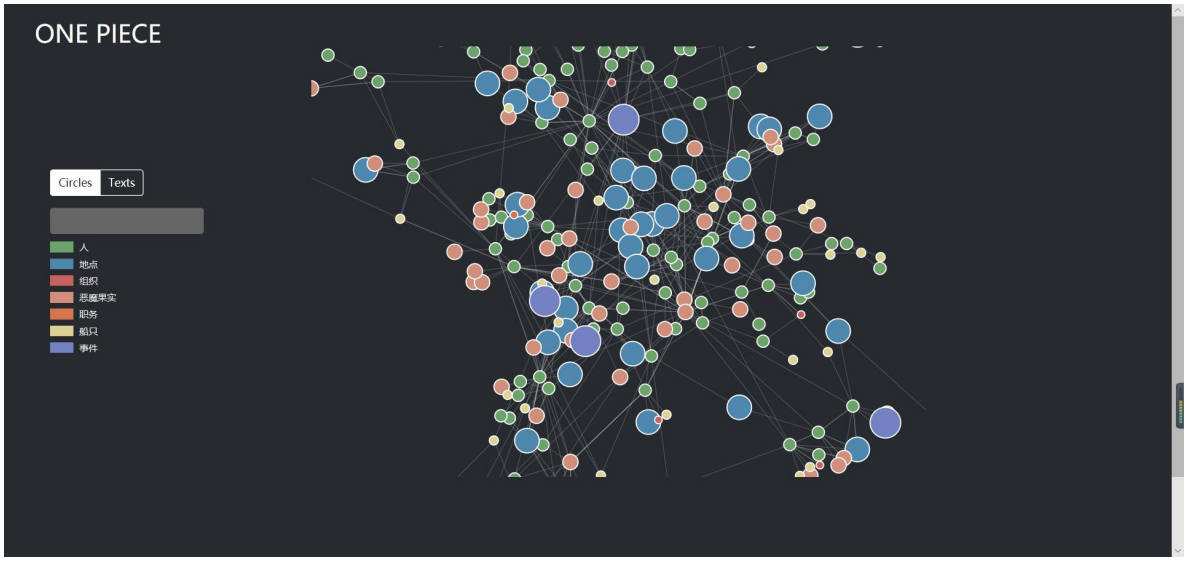
在这部分中，我们参考别人的工作[20]，利用D3[21]对之前构建的实体关系知识图谱提供可视化交互功能，包括结点连接关系可视化、查询相关结点信息。同时在这部分也整合了之间构建的人物属性知识图谱，提供了信息框的展示过程，相关的数据和代码存放在`visualization`目录下。整个可视化页面的交互过程如下面的gif图所示:



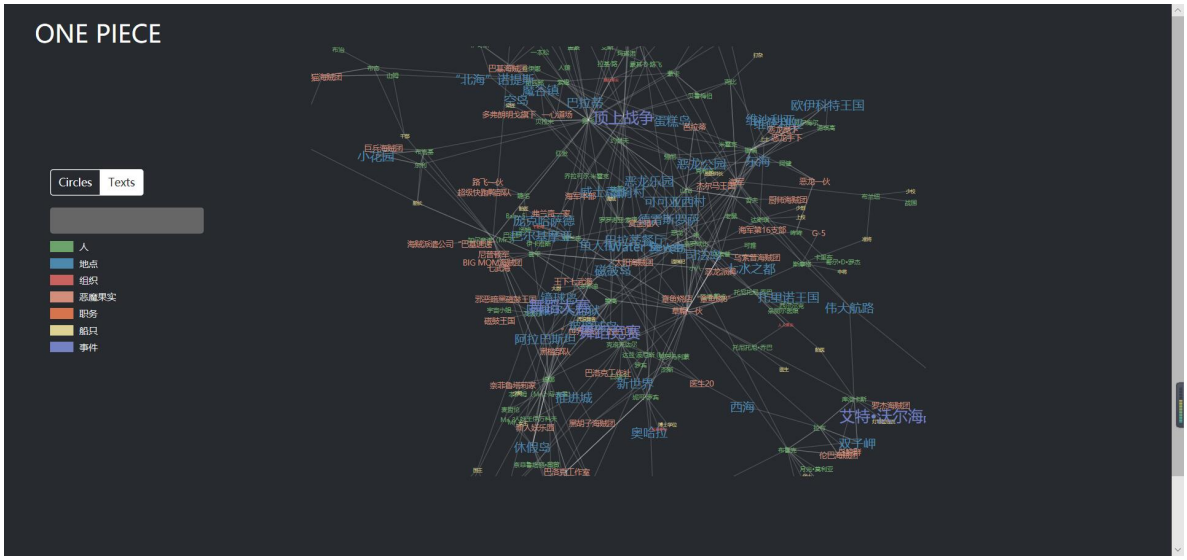
可视化网页存放于`visualization/html/index.html`，可以通过 **Microsoft Edge** 浏览器直接打开

如果需要在其他浏览器中打开，可能会加载不出来可视化结果。这是因为跨域请求在大多数浏览器中是禁止的，请求不到json数据。因此需要用 WAMP/LAMP/MAMP 配置一个Web网络环境。

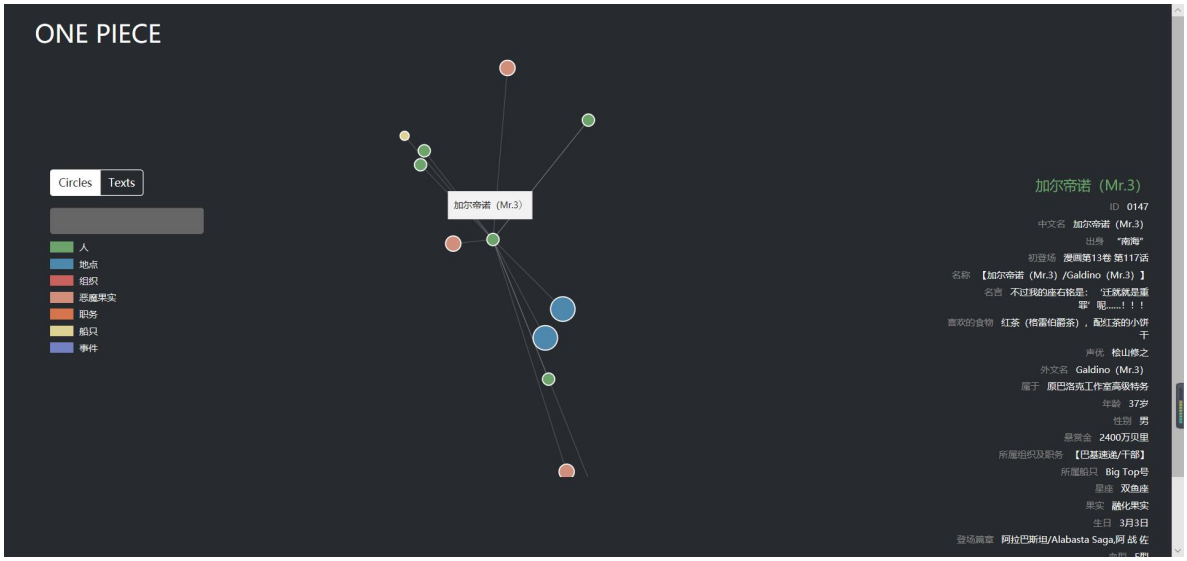
打开后可视化界面如下所示，不同的颜色代表不同类型的实体，具有关系的实体会用白色的细线连接，可以明显的看到有些实体与其他实体存在大量的连接



点击左上角的模式切换按钮，我们可以把结点展示从圆圈模式变换为文本模式，能够进行更加细致的观察



选中某个结点后，将只会显示该节点以及与其直接相连接的结点。特别的，如果该节点类型是人物，还会在页面右侧显示该人物的信息框



此外左侧还提供了搜索框的功能，可以方便我们查找结点信息

参考资料

1. http://bbs.talkop.com/forum.php?mod=viewthread&tid=119685↩
2. https://games.qq.com/a/20150615/057918.htm↩
3. http://kw.fudan.edu.cn/cndbpedia/intro/↩
4. http://kw.fudan.edu.cn/↩
5. https://zh.moegirl.org/index.php?title=Template:%E6%B5%B7%E8%B4%BC%E7%8E%8B&action=edit↩
6. http://kw.fudan.edu.cn/apis/cndbpedia/↩↩
7. http://bbs.talkop.com/forum.php?mod=forumdisplay&fid=49&filter=typeid&typeid=145↩
8. http://www.jinglingbiaozhu.com/↩↩
9. 王鑫,邹磊,王朝坤,彭鹏,冯志勇.知识图谱数据管理研究综述.软件学报,2019,30(7):2139□2174. http://www.jos.org.cn/1000-9825/5841.htm↩↩
10. https://jena.apache.org/↩
11. https://www.w3.org/TR/sparql11-query/↩
12. https://neo4j.com/↩
13. https://github.com/zjunlp/deepke↩
14. https://github.com/ymcui/Chinese-BERT-wwm↩
15. https://www.macalester.edu/~abeverid/thrones.html↩
16. https://www.maa.org/sites/default/files/pdf/Mathhorizons/NetworkofThrones.pdf↩
17. http://www.openkg.cn/tool/refo-kbqa↩
18. https://github.com/SimmerChan/KG-demo-for-movie↩
19. https://github.com/machinalis/refo↩
20. https://github.com/Honlan/starwar-visualization↩
21. https://d3js.org/↩