华 北 水 利 水 电 大 学

知 识 表 示 与 推 理

实 验 报 告

20 22 ——20 23 学年

第 二 学期

课 题 名 称 ：	关键词提取与可视化的实现
班 级 ：	2020185
组 别 ：	第 4 组
团 队 成 员 ：	202018526 高树林、202018525 郭彦溥
	202018527 黄士杰、202018523 徐柏婷
专 业 ：	人工智能

一、 实验目的

通过构建基于党史的知识问答系统，使学生能够掌握知识图谱的构建和问答系统的相关技术。

二、 实验内容

基于给定的党史数据，构建完善的党史知识图谱，并实现知识问答功能。

三、 实验要求

本实验需要完成以下 4 个部分：

1.完成知识体系的构建；

2.完成实体识别和关系抽取；

3.完成知识图谱的存储；

4.完成基于知识图谱的问答；

四、 问题分析与所需技术的分析

问题分析：要建立知识图谱，首先需要清洗和整理收集到的数据，去除冗余信息、格式转换、数据标准化；其次，从文本中提取出实体（人物、事件、组织）、关系（人物与事件的关系、事件与组织的关系）等重要信息；接着将抽取到的实体和关系进行建模，构建知识图谱的基本框架。再通过链接相似的实体和关系，将知识图谱中的信息进行关联和丰富。知识推理：基于知识图谱的结构和规则，进行推理和推断，填充缺失的信息或发现隐藏的关联。可视化展示：将知识图谱以可视化的方式展示出来，便于用户浏览和交互。

所需技术分析：自然语言处理（NLP）：用于文本的清洗、实体识别、关系抽取等任务。信息抽取和知识图谱构建：使用信息抽取算法和图数据库技术构建知识图谱的基本结构。实体链接和关系抽取：通过实体链接技术将文本中的实体链接到知识图谱中的实体，同时抽取实体之间的关系。可视化工具：使用可视化工具将知识图谱以图形化的方式展示出来，如使用图形库或可视化平台。需要注

意的是，以上只是一般的问题分析和技术分析，具体的实现还需要根据具体的需求和实际情况进行调整和选择。

五、 实验步骤及其实现及测试

1. 实体关系确立工作

1.1 Protege 介绍

Protege 是一个本体建模工具软件，由斯坦福大学基于 java 语言开发的，属于开放源代码软件。软件主要用于语义网中本体的构建和基于本体的知识应用，是本体构建的核心开发工具，Protege 具有以下功能：

1. 建模。Protege 提供了一个图形化用户界面来建模类（包括概念）和它们的属性以及关系。
2. 实例编辑。根据创建的类型，Protégé会自动产生交互的形式，可以根据类之间的关系获得相应实例的约束，并对实例进行编辑。
3. 模型处理。Protégé有一些插件库，可以定义语义、解答询问以及定义逻辑行为。
4. 模型交换。最终的模型（类、实例、关系、属性等）能以各种各样的格式被保存和加载，包括 XML、UML、RDF、OWL 等。

1.2 下载与安装

1. 软件安装

软件到官网 https://protege.stanford.edu/products.php 直接下载直接运行 Protege.exe 即可。

Protege 是由 java 开发的，运行时需要 java 运行时环境，运行时可能需要配置一下 java 的运行时环境，按照提示进行配置就可以了。

图 1 Protege 软件

2. 可视化插件的安装

下载插件：http://www.graphviz.org/download/；选择 windows 的稳定版本。（graphviz-2.38）解压后放到 protégé安装包的目录下。（也可以放在其他路径）在 protégé的 file 引用插件，单击“Preferences”，新页面找到“OWLViz” tab，修改“Path to DOT”，为对应的地址。

1.3 添加环境变量

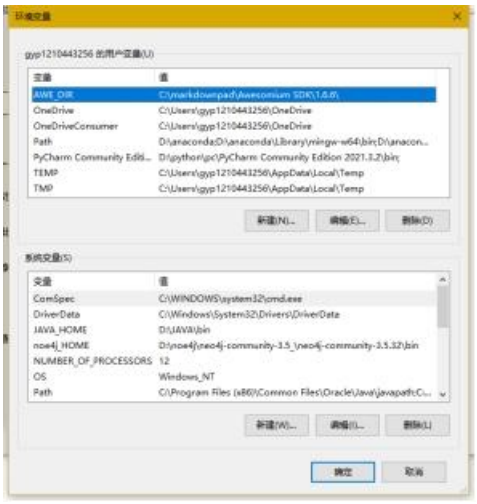


图 2 环境变量量配置

1.4 创建类，创建实体

打开 Classes 界面，owl:Thing 上的三个按钮从左至右，依次为“添加子目标”“添加平级目标”和“删除目标”。其它界面同理。

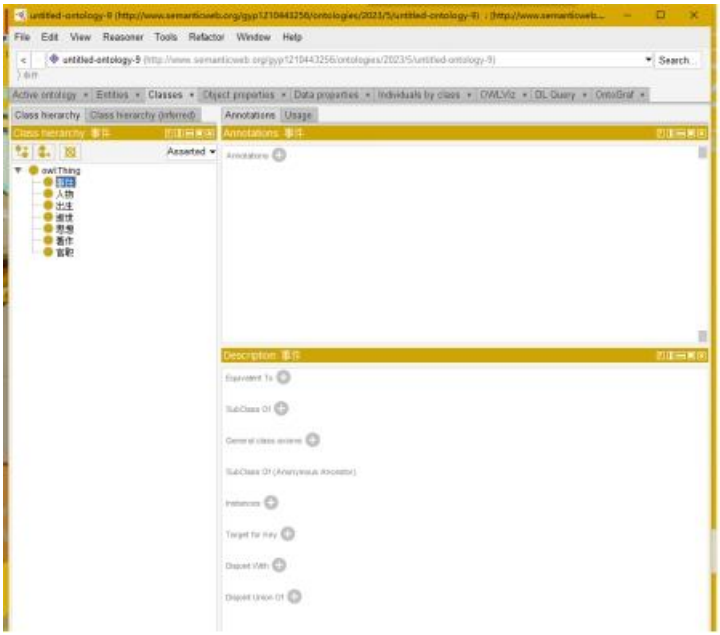


图 3 Class 界面

1.5 创建实体属性

进入 Object Properties 功能界面，构建属性。

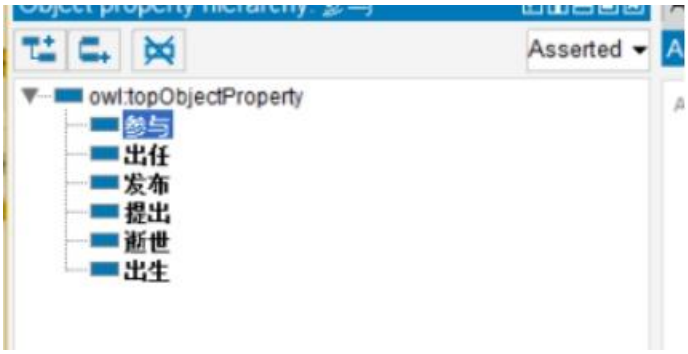


图 4　Object Properties 功能界面

右下角创建实体之间的关系

Domains 指主语，Ranges 为宾语

主题关系为“人物”“参与”“事件”

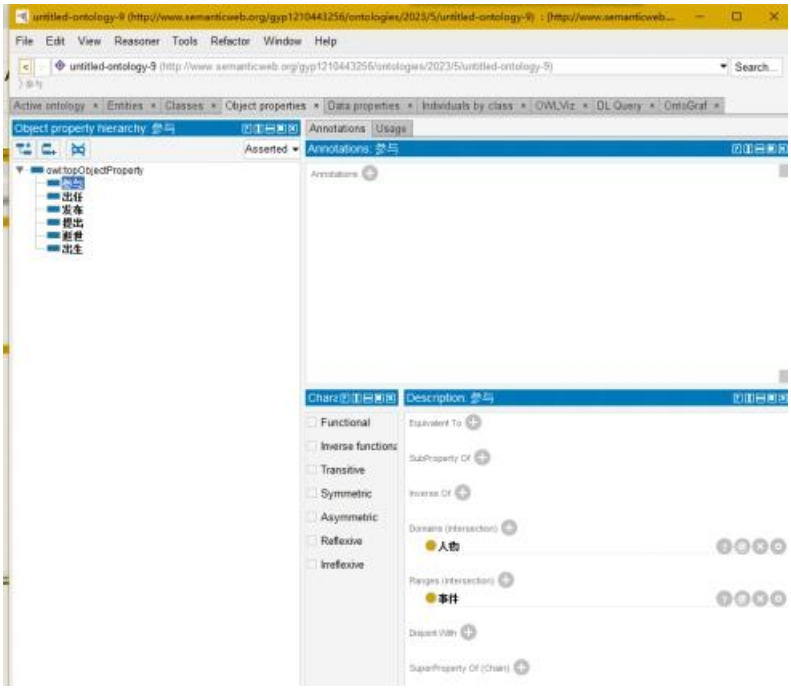


图 5 创建实体

1.6 创建实体属性

进入 Data properties 功能界面，创建实体的属性。

例如人物的属性有“原名”“字”“号”“笔名”“民族”

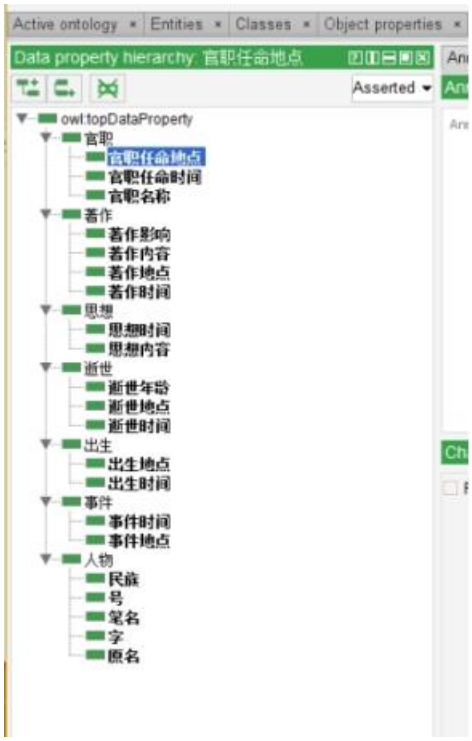


图 6　Data properties 功能界面

右下角

Domains 写所属的哪一个主体

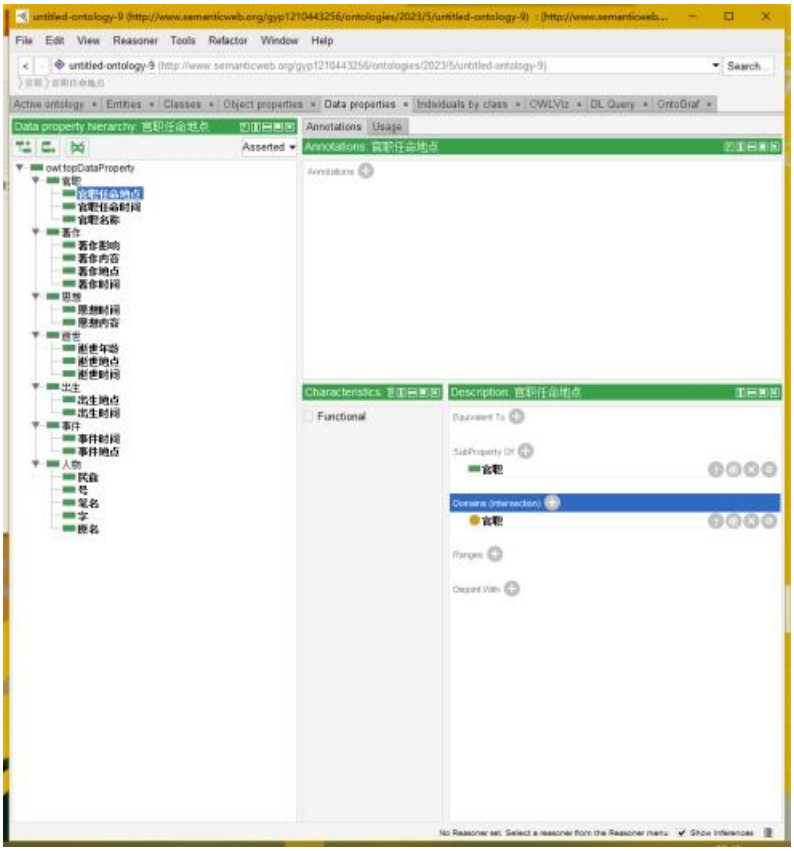


图 7 图 7　Domains 写所属主体

1.7 展示界面

进入 OntoGraf 功能界面，点击即可出现结果

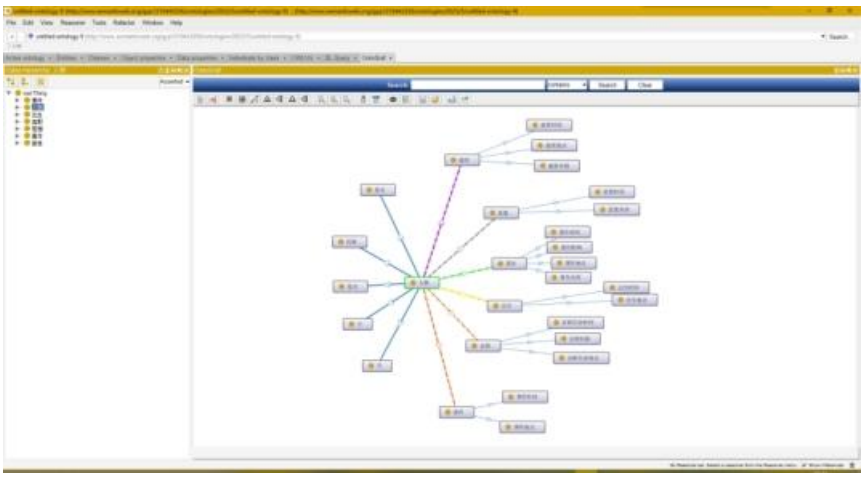


图 8　OntoGraf 功能界面

1.8 任务过程

完成此任务最重要的过程是将老师发的文档进行阅读和总结，完成这一步，梳理好所有的实体关系和属性，按照步骤来会事半功倍。剩下的难点，无非是添加环境变量之类，查阅资料即可。

2. 实体标注工作

2.1 数据实体标注

实体数据标注使用的是百度 easydl 平台的数据标注。EasyDL 是百度 2017 年底推出的零门槛 AI 开发平台，提供定制 AI 模型能力所需的一站式 EasyData 智能数据服务、模型训练、服务部署等全流程功能，内置丰富的预训练模型，支持图像分类、物体检测、图像分割、文本分类、情感倾向分析、序列化标注、音视频分类等多类模型，支持公有云/私有化/设备端等灵活部署方式。

Easydl 目前有以下几种功能：

EasyDL 图像：定制基于图像进行多样化分析的 AI 模型，实现图像内容理解分类、图中物体检测定位等，适用于图片内容检索、安防监控、工业质检等场景

EasyDL 文本：定制基于文心大模型的语义理解 AI 模型，提供一整套文本定制与应用能力，适用于文本内容审核、文本自动生成、留言分类、电商评价打分等场景

EasyDL 语音：定制语音识别模型，精准识别业务专有名词，适用于数据采集录入、语音指令、呼叫中心等场景，以及定制声音分类模型，适用于区分不同声音类别等场景

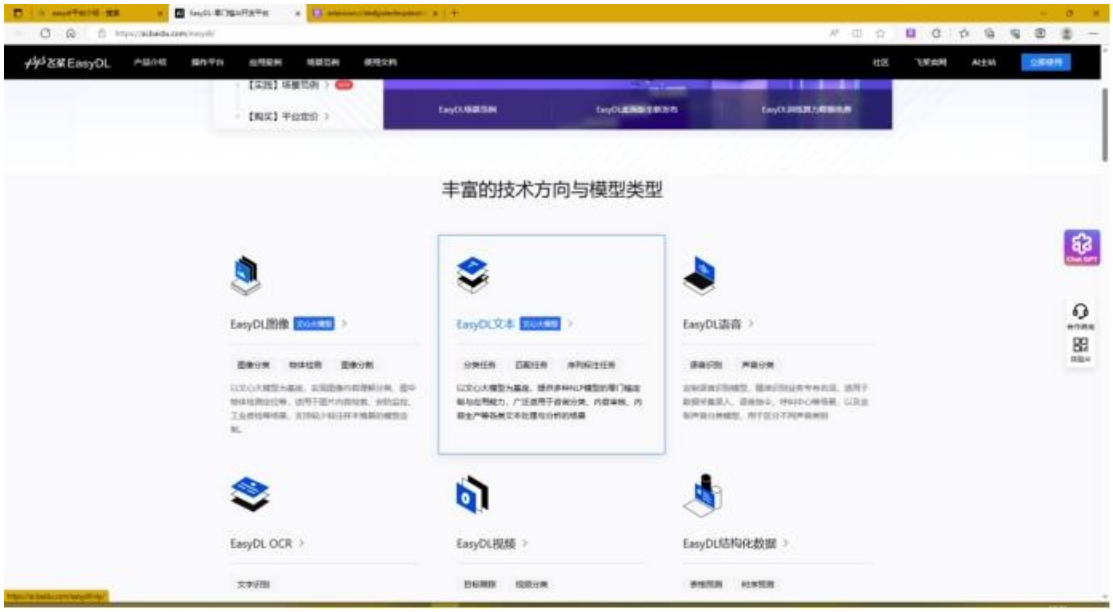
EasyDL OCR：定制文字识别模型，结构化输出关键字段内容，满足个性化卡证票据识别需求，适用于证照电子化审批、财税报销电子化等场景

EasyDL 视频：定制基于视频片段内容进行分类的 AI 模型，适用于区分不同短视频类别等场景，以及定制目标追踪 AI 模型，实现跟踪视频中特定目标对象及轨迹，适用于视频内容审核、人流/车流统计、养殖场牲畜移动轨迹分析等场景

EasyDL 结构化数据：挖掘数据中隐藏的模式，解决二分类、多分类、回归等问题，适用于客户流失预测、欺诈检测、价格预测等场景

EasyDL 零售行业版：面向零售场景的 ISV、零售行业服务商等企业用户，提供基于商品识别场景的 AI 服务解决方案，适用于货架巡检、自助结算台、无人零售柜等场景

本次实验所用到的是 easydl 中文本功能中文本实体关系抽取功能。



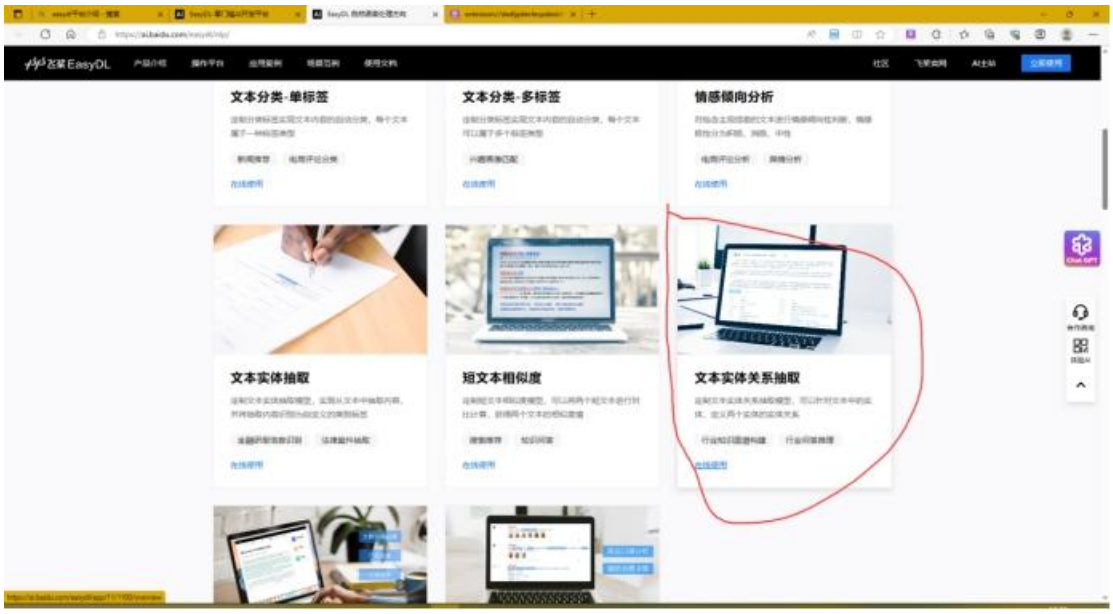


图 9　实体关系抽取功能页面

一，处理源文本数据

文本处理需要解决几个问题：

（1）easydl 中文本实体关系抽取功能中一个文本展示只能展示不多于 500 字

（2）该功能将换行符为一个分隔符，在原文件每个便签文本前无主语，需要加上主语才能进行标注

（3）多人标注必须先将实体，实体关系，属性等分好类再进行标注，本地标注可以边标注边进行添加实体或者关系，更便捷，不容易反工。

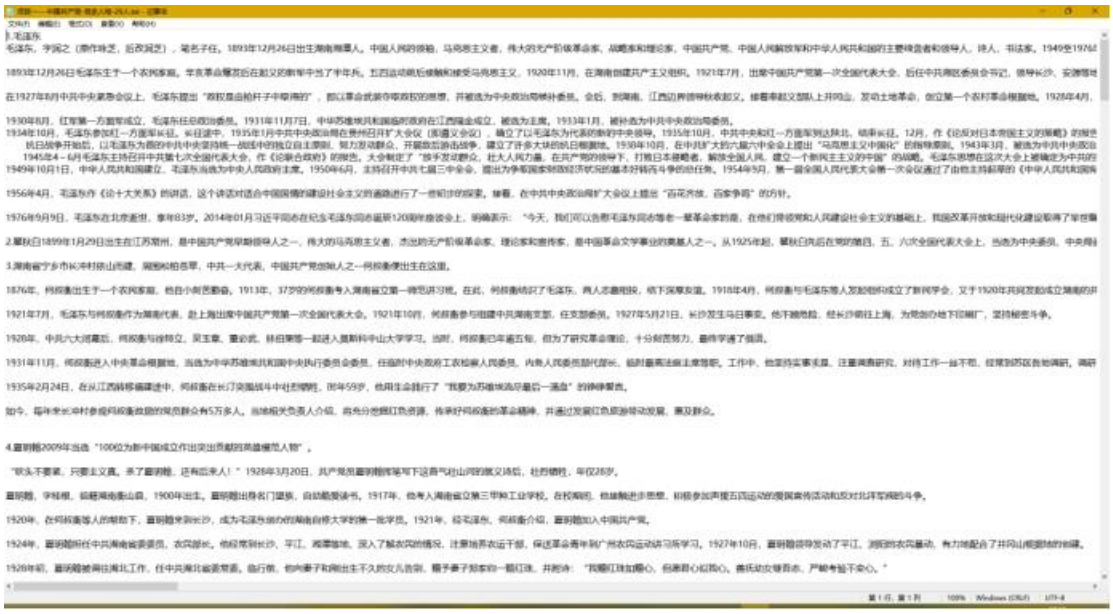


图 10　标注困难，杂乱（原文件）

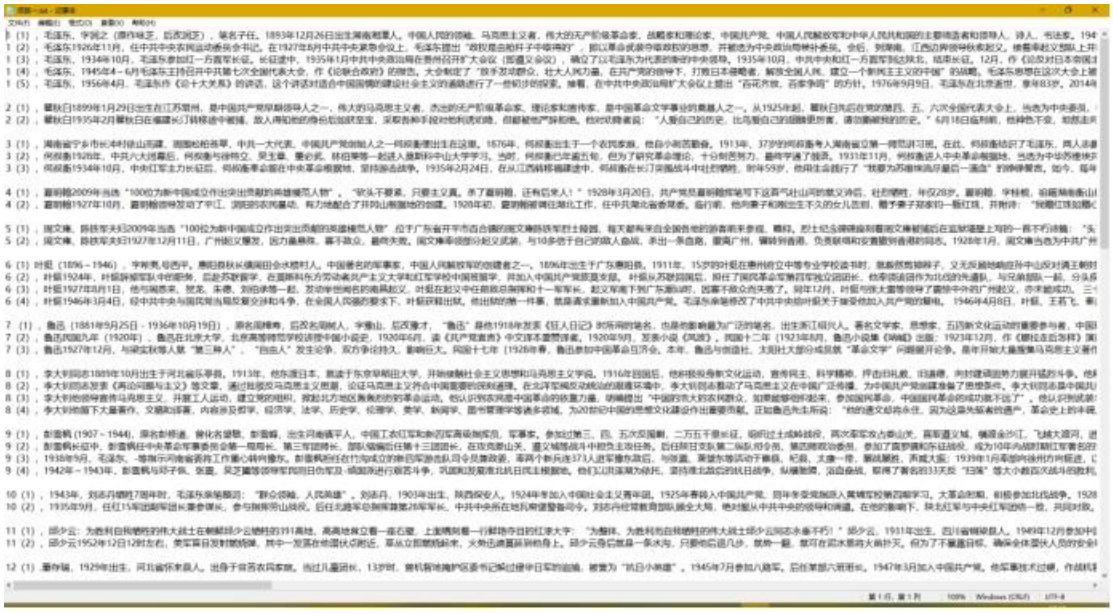


图 11　（修改之后）方便查找和标注

2.2 添加数据

点进“文本实体关系抽取”，得到下面的界面

点击数据总览，创建数据集

图 11　创建数据集界面

写入该数据集的名称，并导入

图 12 该数据集的名称操作截图

选择“本地导入”，“上传 TXT 文本”，分隔符选择“换行符”，数据去重不去重都可以，然后上传文本。

图 13 上传文本操作截图

2.3 进行实体关系的创建和标注

数据导入完成后，点击“标注”，进入到下一个页面。

图 14 标注选项页面

可以看出，文本清晰明了，有“人物”这个实体中心右上角添加“实体”以及实体之间的“关系”。

图 15 标注页面

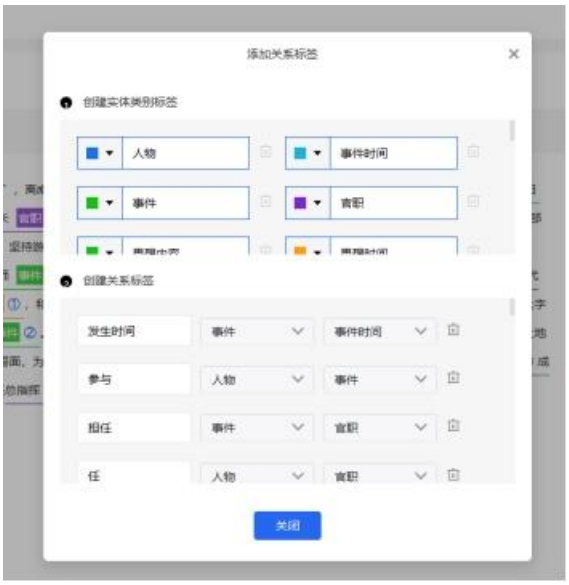


图 16 创建标注实体和关系页面

2.4 标注

非常枯燥无味的标注过程直接省去。

图 17 标注结果页面

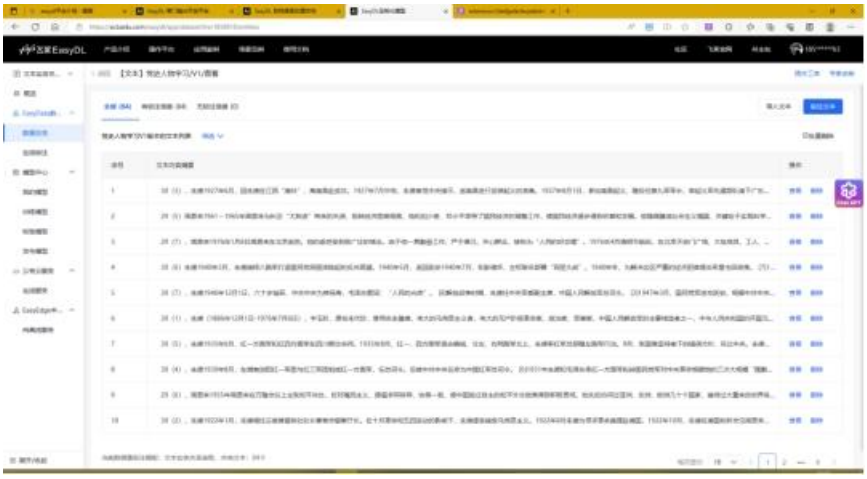


图 18 在线查看标注情况

2.5 导出数据

前往 EasyData 完整操作台，进入“数据集管理”，找到所标注的数据导出即可。

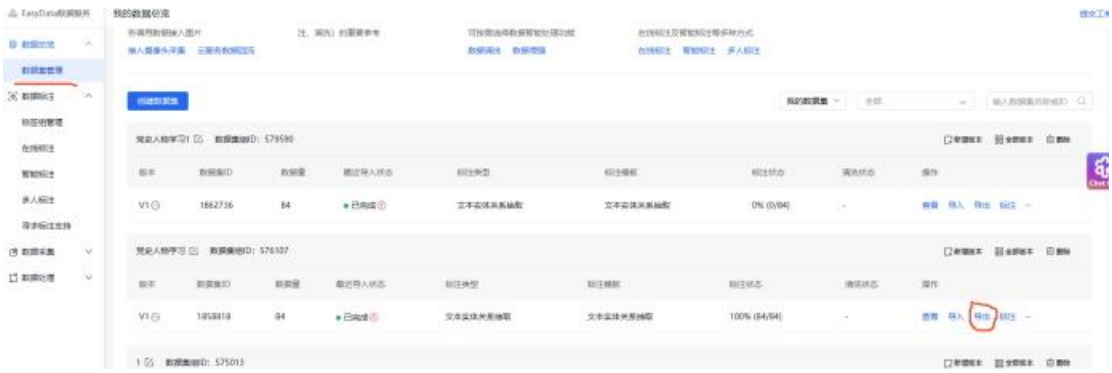


图 19 导出标注结果截图

下载即可完成。

图 20 下载标注结果

导出结果如下图所示

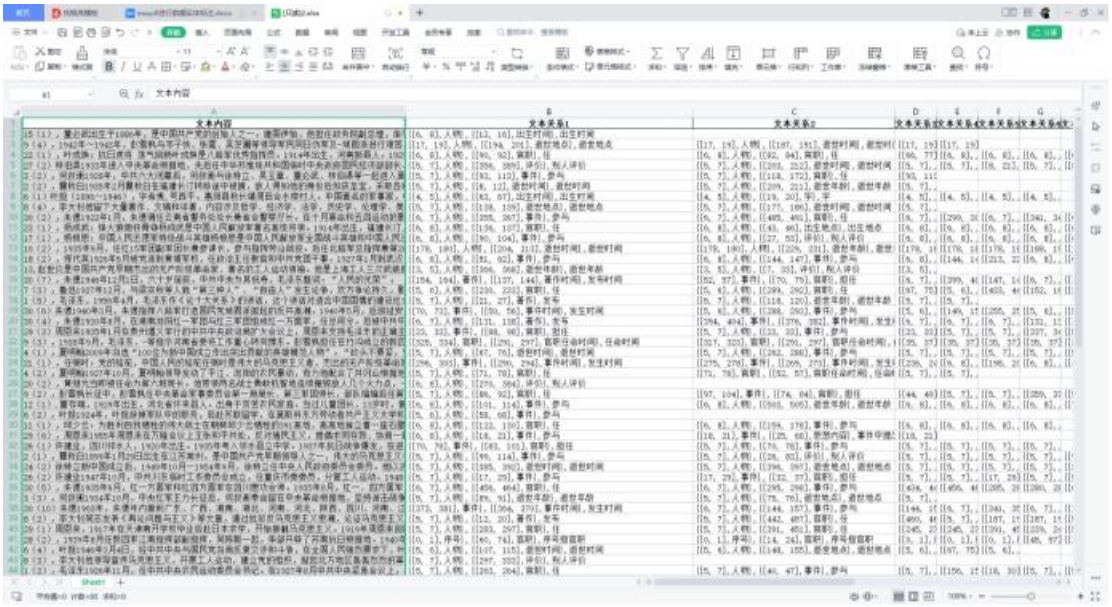


图 21 标注好的文本导出情况

2.6 任务总结

标记任务是个极其简单且枯燥的任务，所以该任务的重难点在数据处理上，由于第一次操作，原文本数据杂乱无章，经常出现，无人物无重点的情况。

图 21 标注出现无重点人物的情况

在一次多人标注之后，经过代码编译之后，效果不佳，需要重新进行“实体”“关系”的设定，打乱之前的设计。所以在一次次多人标注失效后，决定将此项目交给一个人做，本地标注可以边添加“实体”“关系”边进行标注，更加灵活。

为解决某一文本没有主体的问题，本组决定将原文本文档修改，因为以“换行符”为“分隔符”，所以讨论结果为将同一人物整合到一段话，理论上是可行的，所以进行实践。修改后文档如图

图 22 修改后的文档

因为只有三十个人物，所以一共 30 个需要打标签，但出现下图问题，短短几行，明显跟一个人物介绍的几千字大相径庭，更有些例子，在最后一段话戛然而止。可以看出，这种理论可行的方法并不适合这个平台（后续发现只能到 500 字）。

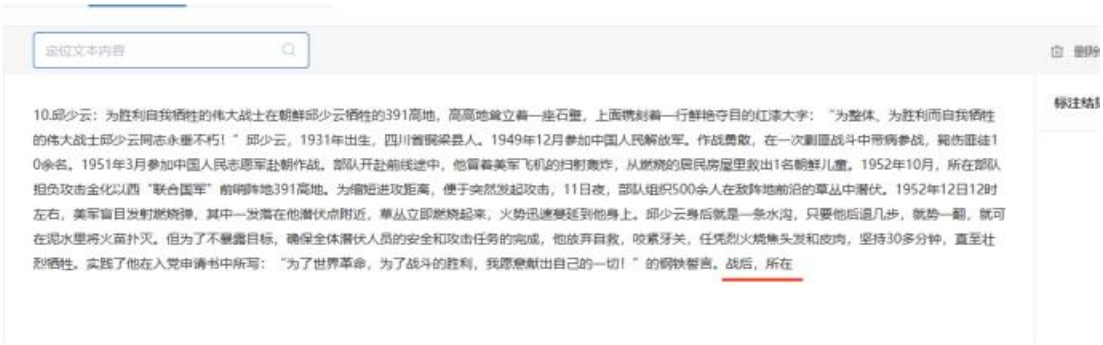


图 23 超过 500 字的内容被截断的现象

所以最终方案，是将每个人的事迹分为大小 400 字左右的事迹，并在每个人每段话之前加上人物名称，方便打便签时以人物为中心。

图 24 上图出现的问题解决方案

图 25 修改文本后的标注效果

3. 构建知识图谱

3.1 抽取实体和关系

首先，根据预先标注好的文件中的索引信息，需要找到相应实体名称在文本中的位置，并利用这些位置信息来提取出它们。下面是图 26，展示了使用 EasyDL 标注工具抽取出的文本内容和文本关系示意图。通过使用这些信息，可以有效地从文本中提取出特定实体的位置和名称。

图 26 文本内容和文本关系示意图

在这个任务中，首先要根据标注文件中的索引信息，定位出每个实体的起始位置和结束位置。然后，可以利用这些位置信息来在文本中准确地抽取出对应的实体名字。这样，就能够从文本中获取到我们感兴趣的实体信息。

这个过程是非常重要的，因为通过准确地提取实体的位置和名称，可以进一步分析这些实体之间的关系和上下文信息。这种信息提取和分析对于许多自然语言处理任务和应用非常关键，如信息抽取、文本分类、实体关系推理等。在本实验中，其作用是最后一个，实体关系推理。其核心代码如下所示：

```
row_content = data[i].split('},')
# print(row_content)
entity1,entity2,relation = row_content
#  节点 1 信息
B = entity1.split(',')[0][2:]
E = entity1.split(',')[1][0:-1]
label1 = entity1.split(',')[-1]
entity1 = data[0][int(B):int(E)+1]
result[label1] = entity1
if label1 == '人物' and entity1 not in person:
        person.append(entity1)
#节点 2 信息
B = entity2.split(',')[0][2:]
E = entity2.split(',')[1][0:-1]
label2 = entity2.split(',')[-1]
entity2 = data[0][int(B):int(E)+1]
```

上述代码的作用是从一个字符串中提取实体和关系信息，并将它们存储在相应的变量中。首先，代码使用`split('},')`将字符串`data[i]`按照`},`进行分割，得到一个包含多个元素的列表`row_content`，每个元素代表一个实体和关系。接下来，代码将列表中的第一个元素赋值给`entity1`，第二个元素赋值给`entity2`，第三个元素赋值给`relation`，分别表示第一个实体、第二个实体和它们之间的关系。然后，代码通过字符串分割和切片操作，从`entity1`中提取出实体的起始位置`B`、结束位置`E`和标签`label1`。这些信息用于在原始文本数据中定位实体并抽取出实体名称。抽取的实体名称存储在变量`entity1`中，并将其与对应的标签`label1`一起存储在字典`result`中。如果`label1`的值为'人物'，并且`entity1`不在列表`person`中，代码将`entity1`添加到`person`列表中。这段代码的目的是筛选出标签为'人物'的实体，并将它们存储在`person`列表中。类似地，代码使用类似的方式

从`entity2`中提取出实体的起始位置、结束位置和标签，并将实体名称存储在变量`entity2`中。

3.2 创建实体和关系

在创建实体时，有一个关键的步骤是在创建之前先判断该实体是否已经存在。这是一个非常重要的步骤，因为如果实体已经存在，就无需再次创建，以避免在后续的查询中产生混乱。此外，通过这样的判断，可以使实体关系更加集中和一致，进一步优化数据库的结构。在处理实体时，我们需要考虑数据库中是否已存在具有相同属性的实体。通过先进行判断，我们可以检查数据库中是否已存在该实体，而无需重复创建。这样做的好处是避免了重复存储相同实体的信息，减少了数据冗余，并且确保了数据库的一致性。通过这种判断和避免重复创建实体的方式，我们可以确保实体关系更加集中和清晰。实体关系的集中性意味着相关实体都被正确地连接在一起，这有助于提高数据库的查询效率和准确性。此外，避免重复创建实体还可以节省存储空间并提高数据库的整体性能。

因此，在创建实体时，我们应该先进行判断，确保该实体是否已存在，并根据判断结果来决定是否需要创建新实体。这样做既可以避免查询混乱问题，也可以提高实体关系的集中性，从而优化数据库的结构和性能。其核心代码如下所示：

```
nodelist=list(matcher.match(label1,name=entity1))
# print(len(nodelist))
if len(nodelist) > 0:
    entity1 = nodelist[0]
else:
    entity1 = Node(label1, name=entity1)
    g.create(entity1)

nodelist=list(matcher.match(label2,name=entity2))
# print(len(nodelist))
if len(nodelist) > 0:
    entity2 = nodelist[0]
else:
    entity2 = Node(label2, name=entity2)
    g.create(entity2)

relation = Relationship(entity1,relation,entity2)
g.create(relation)
```

代码用于创建 Neo4j 图数据库中的节点（Node）和关系（Relationship）。

首先，代码使用`matcher.match(label1, name=entity1)`查询数据库中具有标签为`label1`且名称为`entity1`的节点。如果查询结果列表`nodelist`的长度大于 0，表示已存在符合条件的节点，代码将第一个节点赋值给`entity1`变量。如果查询结果为空（即`nodelist`长度为 0），则表示数据库中不存在符合条件的节点。在这种情况下，代码使用`Node(label1, name=entity1)`创建一个新节点，标签为`label1`，名称为`entity1`，并将其赋值给`entity1`变量。接着，代码使用类似的方式查询数据库中是否存在符合条件的第二个节点，即使用`matcher.match(label2, name=entity2)`来查询。如果查询结果列表`nodelist`的长度大于 0，表示已存在符合条件的节点，代码将第一个节点赋值给`entity2`变量。如果查询结果为空（即`nodelist`长度为 0），代码使用`Node(label2, name=entity2)`创建一个新节点，标签为`label2`，名称为`entity2`，并将其赋值给`entity2`变量。最后，代码使用`Relationship(entity1, relation, entity2)`创建一个关系对象，其中`entity1`和`entity2`分别表示关系的起始节点和结束节点，`relation`表示关系的类型。然后，通过`g.create(relation)`将该关系对象添加到图数据库中。创建结果如下图 27 所示。

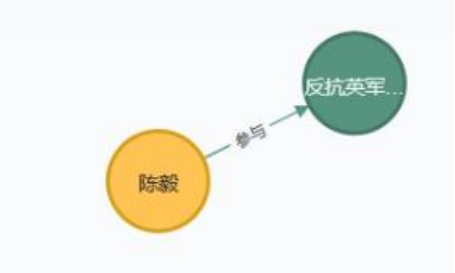


图 27 创建的实体和关系

3.3 自动化创建

通过将第 1 和第 2 部分的代码封装成一个创建函数，就可以方便地创建实体关系图谱中的节点和边。这个函数可以在遍历所有数据时被调用，每次调用都会创建一个数据对应的节点和边。当遍历完所有数据后，最终的知识图谱也就建立完成了。这个封装后的创建函数可以提供一个高效的方式来处理大量数据并构建图谱。通过在遍历过程中调用该函数，我们可以将每个数据映射到图谱中的节点和边，从而逐步建立起整个知识图谱的。通过这种方式，我们可以在遍历完所有数据后，实现了知识图谱的构建。知识图谱作为一个结构化的知识表示方式，能够将实体之间的关系和上下文信息有效地呈现出来。这样的知识图谱不仅能够提

供更好的可视化和查询功能，还可以支持更高级的知识推理和分析。其核心代码如下所示：

```
from tqdm import tqdm

print('正在为你导入%d 条数据...'%len(df))
for i in tqdm(range(len(df))):
        data = df.iloc[i].values
        # print('第%d 条数据已入库！'%i)
        get_Node(data)
print('导入成功！')
```

上述代码调用 get_Node(data)函数，将 data 作为参数传递给 get_Node()函数。这个函数是自定义的用于创建实体关系图谱中节点和边的函数。最终的知识图谱如下图 28 所示。

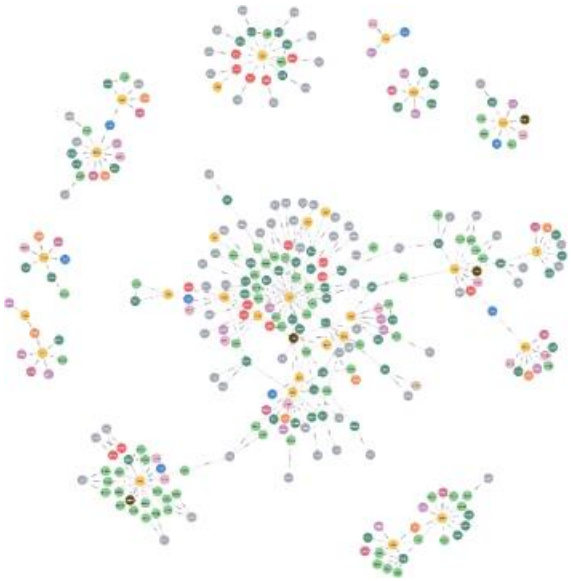


图 28 本实验建立的知识图谱

4. 知识图谱查询

4.1 定义查询内容

这里首先要定义样本标志变量、实体词典、关系词典、属性词典、规则列表和 Cypher 查询语句列表，这些信息可以用于后续的问题解析和图数据库查询。通过这些定义，可以根据具体的问题和数据定义，生成相应的查询语句，并从图数据库中检索实体的属性和关系信息。代码如下：

```
import re
sample = False
entity_dict = {'南昌起义': '事件', '解放战争时期': '事件', ...}
rel_dict = {'参与':'关系'}
```

```
prop_dict = {'出生时间':'属性','原名':'属性',.....}
rules=["人物的属性是？",
                "人物的关系是？",
                "人物的好友的是？",]
cypher_strs=["match (n:人物 {name:'<名称>'}) return n.<属性> ",
    "match (n1:人物 {name:'<名称>'})-[re:<关系>]->(n2) return n2.name",
     "match (n1:<人物> {name:'<名称>'})-[re:<关系>]->(n2) match (n2)-[re:<关系>]->(n3) return n3.name"]
#目标查询
# q="毛泽东的出生时间是?"
q='毛泽东和朱德有什么关系'
```

这段代码主要包含了一个样本标志变量、实体词典、关系词典、属性词典、规则列表和 Cypher 查询语句列表。首先代码定义了三个词典：`entity_dict`：实体词典，用于存储实体名称及其类型的映射关系。`rel_dict`：关系词典，用于存储关系名称及其类型的映射关系。`prop_dict`：属性词典，用于存储属性名称及其类型的映射关系。然后，代码定义了一个规则列表`rules`，包含了几个用于查询人物属性和关系的样本问题模板。最后，代码定义了一个 Cypher 查询语句列表`cypher_strs`，其中包含了一些针对不同问题模板的查询模板。这些查询模板中的`<名称>`、`<属性>`、`<关系>`、`<人物>`等占位符可以根据具体问题和实体关系词典来填充，用于构建查询语句。

4.2 获取实体内容

在步骤 4 中，需要查找到各种实体对应的标签，因此需要将其分类整理好。普通做法是通过 Cypher 语句查询实体，具体查询语句为：

```
match (n:`人物` ) return n
```

图 29 查询人物

之后可以导出 json 文件或者 csv 文件，复制粘贴到文本文件里面就能得到最

终的人物标签。但是这种方法太过于复杂，工作量大，在这里我们采用了 python 自动化运行。其代码如下：

```
from py2neo import Graph,Node,Relationship
# Graph()中第一个为 local host 链接，auth 为认证，包含 username 和 password
neo_graph = Graph('http://localhost:7474', auth = ('neo4j', 'gsl1234567890'))
for label in labels:
        cypher = "match (n:%s ) return n"%label
        data = neo_graph.run(cypher2)
        with open('%s.txt','a',encoding='utf-8'%label) as f:
                f.write(data)
```

这段代码使用了`py2neo`库来连接到 Neo4j 图数据库，并执行了一系列查询。首先，通过`Graph`类创建了一个连接到本地主机的图数据库实例`neo_graph`，其中指定了主机地址和认证信息。然后，通过一个循环遍历了`labels`列表中的每个标签。在每次迭代中，根据当前的标签构造了一个 Cypher 查询语句`cypher`，该查询语句用于匹配具有该标签的所有节点。接下来，使用`neo_graph.run(cypher)`执行查询，并将查询结果保存在`data`中。最后，将查询结果写入一个文本文件，文件名为当前标签的名称加上`.txt`后缀，使用追加模式打开文件并将结果写入。最终结果如下图 30 所示：

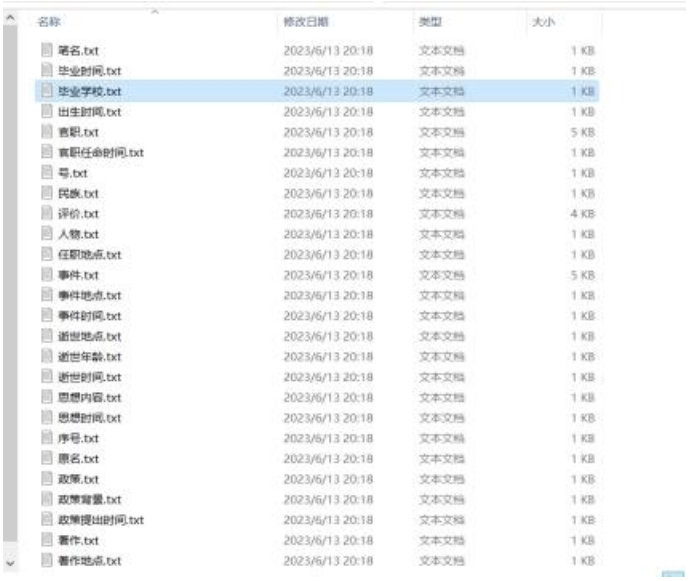


图 30 实体文本

4.3 正向匹配查询算法

所谓正相匹配查询算法，即从给定的查询字符串中抽取实体、关系和属性的关键词。首先，根据指定的词典，循环遍历查询字符串，并以每次最大长度进行

切割子串。然后，检查子串是否在词典中存在，如果存在，则将匹配到的词、起始位置和结束位置以及对应的类型添加到一个列表中。接下来，根据关键词在查询字符串中的起始位置，对列表进行排序。最后，将排序后的关键词列表保存在 obj_list 中。这些关键词可以用于后续的查询和分析。代码如下：

```
def get_kwords(obj_dict,q):
    max_len=5 #每次用于匹配的字符串的最大长度
    word_list=[]#保存匹配得到的词
    start=0 #用于匹配的字符串的起始位置
    while(start<len(q)):#当用于抽取词的句子长度大于 0
        #print("当前的 start:",start)
        end_cut_pos=min(start+max_len,len(q))
        current_s=q[start:end_cut_pos] #当前用于抽取词的子串
        #print("外 current_s:",current_s)
          is_cut_words=False
          for i in range(end_cut_pos-start):
              current_s=q[start:end_cut_pos]
              #print("内 current_s",current_s[start:end_cut_pos])
              if current_s in list(obj_dict.keys()):
                  word_list.append((current_s,start,end_cut_pos,obj_dict[current_s])) #保存格式为(匹配到的词，词的起始位置，词的最末位置
                  #print("内 current_s",current_s[start:end_cut_pos])
                  start=end_cut_pos
                  #print("jljkljkl",start,q[start])
                  is_cut_words=True
                  break
              else:
                  end_cut_pos=end_cut_pos-1
                  #print("end_cut_pos:",end_cut_pos)
        if is_cut_words==False:
              start=start+1
    return word_list
```

这段代码实现了使用正向最大匹配法从查询中抽取实体、关系和属性的关键词。首先，定义了一个名为`get_kwords`的函数，它接受两个参数：`obj_dict`表示进行匹配的词典，`q`表示待匹配的查询字符串。在函数内部，通过设定最大长度`max_len`和起始位置`start`，使用正向最大匹配法从查询字符串中抽取词。循环遍历查询字符串，每次取最大长度的子串进行匹配。如果子串在词典中存在，则将匹配到的词、起始位置和结束位置以及对应的类型添加到`word_list`列表中，并更新起始位置为结束位置。如果子串不在词典中，则将结束位置递减 1，重新

进行匹配，直到匹配成功或结束位置等于起始位置。然后，代码分别调用`get_kwords`函数对实体词典、关系词典和属性词典进行匹配，将匹配得到的词保存在`entity_list`、`rel_list`和`prop_list`列表中。接下来，将匹配得到的词列表按照词在查询中的起始位置进行排序，使用`sorted()`函数和`lambda`表达式来实现。最后，将排序后的词列表保存在`obj_list`列表中，其中包括了匹配到的实体、关系和属性的词信息。

4.4 模板替换

该过程是对查询字符串进行处理，其中包括将其中的关键词替换为预定义的槽位，并根据替换后的查询字符串匹配对应的 Cypher 语句模板，从而生成完整的 Cypher 语句。生成的 Cypher 语句可以用于在图数据库中进行查询操作，以获取与知识图谱相关的信息。首先，通过逐个遍历关键词列表，确定每个关键词在查询字符串中的起始位置和结束位置，并提取待替换的字符。然后，使用预定义的槽位来替换查询字符串中的关键词，将其转化为特定的占位符。接下来，根据替换后的查询字符串，与预先定义的 Cypher 语句模板进行匹配。找到与查询字符串匹配的模板后，使用正则表达式确定模板中的所有槽位的位置。依次遍历槽位列表，将关键词逐个替换到对应的槽位位置，形成最终的 Cypher 语句。生成的 Cypher 语句可以用于在图数据库中执行查询操作，以检索与知识图谱相关的信息。通过这种方式，我们可以利用提取的关键词和预定义的模板构建出具有灵活性和可扩展性的查询语句，从而获得所需的知识图谱数据。详细代码如下：

```
for obj in obj_list:
            start_pos=obj[1]
            end_pos=obj[2]
            o_qstr= q[start_pos:end_pos]#替换前的字符;
            #print("替换前的字符",o_qstr,"替换后的字符:",obj[3])
            q=q.replace( o_qstr,obj[3])#obj[0]也就是对应的词
            #print("q:",q)
            #
      print("变为模板之后的 q:",q)
        cypher=""
        for i in range(len(rules)):
            rule=rules[i]
            print("rule:",rule,"i:",i,"len(rules):",len(rules))
            if rule==q:
                cypher=cypher_strs[i]
```

```
                #采用正则表达式，确定查询语句模板中所有槽的位置 spans
                pattern="<[\u4e00-\u9fa5]+>"
                cypher=cypher_strs[i]
                spans= re.finditer(pattern,cypher)
                print("spans:",spans)
                j=0
                match_list=[]
                for match in spans:
                    #print(match.group(), match.start(), match.end())
                    match_list.append( (match.start(), match.end()))
                print(len(match_list),match_list)
                #对于每个槽位，依次用匹配到的词进行替换，形成最终的 cypher 语句
                for j in range(len(match_list)-1,-1,-1):
                    print("j=",j)
                    obj=obj_list[j]
                    start,end=match_list[j][0],match_list[j][1]
                    print("obj info:",obj,start,end)
                    #替换对应位置的字符
                    #cypher[start:end]=obj[0]
                    print(cypher)
                    cypher=cypher[: start] + obj[0] + cypher[end:]
                    print("中间 cypher",cypher)
        return cypher
```

上述代码的功能是将查询字符串中的关键词替换为模板中的槽位，并生成最终的查询语句（Cypher 语句）。首先，通过遍历 obj_list 中的关键词信息，获取每个关键词在查询字符串中的起始位置和结束位置。然后，根据起始位置和结束位置，提取出待替换的字符 o_qstr。接下来，使用关键词的类型（obj[3]）来替换查询字符串中的关键词，将其替换为槽位。这样，查询字符串中的关键词就被替换成了对应的槽位。接着，根据替换后的查询字符串 q，遍历预先定义的查询语句模板列表 rules，并与 q 进行比较。如果找到匹配的规则，则获取对应的 Cypher 语句模板 cypher_strs[i]。然后，使用正则表达式，找到 Cypher 语句模板中所有槽位的位置（spans）。通过遍历 match_list，将每个槽位的起始位置和结束位置保存起来。接下来，对于每个槽位，从最后一个槽位开始（倒序遍历），依次用匹配到的关键词进行替换。使用关键词的文本（obj[0]）替换 Cypher 语句模板中对应位置的字符，形成最终的 Cypher 语句并返回。

4.5 实现问答

问答系统的实现过程涉及到连接数据库，并利用之前生成的 Cypher 语句来

查询实体和关系，或者判断两种实体之间的关系。以下是具体的步骤：首先，与图数据库建立连接，确保可以与数据库进行交互。然后根据问题，使用之前生成的 Cypher 语句模板库，匹配到与问题相对应的 Cypher 语句模板。接下来将匹配到的 Cypher 语句模板填充上具体的实体名称或关键词，形成完整的 Cypher 查询语句。接着使用完整的 Cypher 查询语句来查询图数据库，获取与问题相关的实体和关系信息。最后对查询结果进行分析和判断，根据需要提取有用的信息或进行进一步的推理。代码如下：

```
from py2neo import Graph,Node,Relationship
# Graph()中第一个为 local host 链接，auth 为认证，包含 username 和 password
neo_graph = Graph('http://localhost:7474', auth = ('neo4j', 'gsl1234567890'))
cypher=match(q, obj_list)
print(cypher)

cypher2="MATCH p=(n:`人物` {name:'%s'})-[r:`%s`]->(n1)  RETURN
  n1.name"%(obj_list[0][0],obj_list[1][0])
answer2 = neo_graph.run(cypher2)
print(answer2)
```

上述代码使用`py2neo`库来连接到 Neo4j 图数据库，并执行了两个 Cypher 查询。首先，使用`Graph`类创建了一个连接到本地主机的图数据库实例`neo_graph`，其中指定了主机地址和认证信息。然后，通过调用之前的函数`match(q, obj_list)`，生成了一个完整的 Cypher 查询语句`cypher`。接下来，该查询语句通过节点和关系的条件进行匹配，并返回符合条件的节点的名称。最后，通过`neo_graph.run(cypher2)`执行查询，并将结果保存在`answer2`中。结果可以通过打印`answer2`来查看。该段代码的目的是连接到 Neo4j 图数据库，并执行两个查询语句，以获取相关的知识图谱信息。

5. 搜索引擎框架前端

5.1 flask 框架简介

Flask 是一个基于 Python 的微型 Web 框架，它简洁而灵活，适用于构建小型到中型的 Web 应用程序和 API。它被设计成简单易用，具有可扩展性和可定制性，是许多开发者选择的首选框架之一。

Flask 的主要特点如下：

1.简单易学：Flask 采用了简洁的设计哲学，使得入门变得非常容易。它的

代码量较小，易于阅读和理解，提供了清晰明了的 API 接口。即使对于初学者，也可以快速上手并开始构建 Web 应用。

2.轻量级和灵活性：Flask 是一个轻量级的框架，不包含过多的冗余功能，这意味着您可以根据自己的需求来选择和集成其他的扩展模块。它提供了核心功能，并允许您根据项目的要求进行灵活定制，使开发变得高效且具有强大的控制力。

3.路由和视图：Flask 使用基于装饰器的路由系统，可以通过定义视图函数和 URL 规则来处理网页的请求。通过简单地编写装饰器，您可以将不同的 URL 映射到相应的视图函数，并在其中处理请求参数、渲染模板和返回响应。

4.模板引擎：Flask 集成了 Jinja2 模版引擎，使得在网页中使用动态内容变得非常简单。您可以通过定义模板文件，并在视图函数中传递参数来动态生成网页内容。Jinja2 引擎提供了丰富的模板语法，包括条件判断、循环迭代、过滤器等功能，使得构建复杂的页面变得更加容易。

5.数据库支持：Flask 可以与各种流行的数据库进行集成，例如 SQLite、MySQL、PostgreSQL 等。它提供了轻量级的对象关系映射（ORM）工具 Flask-SQLAlchemy，让您可以方便地进行数据库操作，并且保持良好的可扩展性。

6.扩展性和社区支持：Flask 拥有庞大的开发者社区，提供了许多可选的扩展模块，如身份认证、表单验证、缓存管理、REST API 等。这些模块提供了额外的功能和便利性，通过简单的导入和配置即可快速集成到您的应用程序中。

总之，Flask 是一个简洁而灵活的 Python Web 框架，它提供了基础的工具和功能，同时允许您根据项目需求进行定制和扩展。无论是构建小型网站还是大型 Web 应用，Flask 都能够帮助您快速搭建并提供高效、可靠的解决方案。它的简单性、灵活性和强大的社区支持使其成为众多开发者钟爱的框架之一。

5.2 html 简介

HTML（HyperText Markup Language，超文本标记语言）是一种用于创建网页的标准标记语言。它描述了网页的结构和内容，并通过标签（tags）来定义页面上的各个元素。

HTML 使用一系列标签来构建网页。每个标签都由尖括号包围，通常成对出现，比如<tagname></tagname>。标签由两部分组成：开始标签和结束标签。开

始标签用于定义元素的起始位置，而结束标签用于定义元素的结束位置。
除了常见的标签，如标题、段落、链接和图像，HTML 还提供了许多其他标签，用于创建表格、表单、列表等各种网页元素。通过嵌套不同的标签，可以创建复杂的页面结构和布局。

HTML 还支持使用属性为标签提供额外的信息。属性位于开始标签中，并使用键值对的格式表示。例如，<a href="https://www.example.com">定义一个链接，并使用 href 属性指定链接的目标 URL。

在编写 HTML 文件时，可以使用文本编辑器（如 Notepad++、Sublime Text 等）进行编辑，并将文件保存为.html 扩展名。之后，可以将文件通过浏览器打开，就可以看到所编写的页面效果。

总结来说，HTML 是一种用于创建网页的标记语言，通过使用标签来定义页面结构和内容。它是构建万维网的基础，为开发人员提供了丰富的工具和选项，以创建交互式和多媒体的网页体验。

5.3 CSS 简介

CSS（Cascading Style Sheets，层叠样式表）是一种用于控制网页布局和设计的样式表语言。它与 HTML 结合使用，可以为 HTML 文档中的各个元素定义样式和外观。

通过使用 CSS，可以将页面的表现和内容分离开来。HTML 主要负责描述页面的结构和语义，而 CSS 则负责描述页面的外观和样式。这样的分离使得网页的维护更加灵活且易于管理。
CSS 使用选择器来选择需要应用样式的 HTML 元素。选择器可以根据元素的标签名、类名、ID 等属性进行匹配。例如，可以使用标签选择器（比如 p）选择所有段落元素，或者使用类选择器（比如.myclass）选择具有特定类名的元素。

一旦选择了要应用样式的元素，就可以使用 CSS 属性来指定样式规则。每个 CSS 属性都有一个对应的值，用于定义元素的外观。例如，可以使用 color 属性设置文本颜色，font-size 属性设置字体大小，background-color 属性设置背景颜色等。除了基本样式属性，CSS 还提供了一系列高级功能。这包括盒模型（box model）用于控制元素的尺寸和边距，定位（positioning）用于精确布局元素，动画（animation）和过渡（transition）用于创建流畅的页面效果，以及响应式设计

（responsive design）用于适应不同屏幕尺寸的布局等。

CSS 样式可以通过内联样式、嵌入式样式和外部样式表来应用到 HTML 文档中。内联样式直接写在 HTML 标签的 style 属性中，嵌入式样式使用<style>标签放置在 HTML 头部或文档中的<head>标签中，而外部样式表则是将 CSS 代码保存在独立的.css 文件中，并通过链接 (<link>) 引用到 HTML 文档中。

总结来说，CSS 是一种用于控制网页样式和布局的样式表语言。它与 HTML 结合使用，为开发者提供了丰富的选项和灵活性，以创建各种精美的网页设计和用户界面。通过使用选择器和属性，可以定义元素的外观、排版、动画和交互效果，使网页更具吸引力和可视化效果。

5.4 问答系统 html 首页

在 templates 文件中创建一个 html 文件命名为 chat.1.html 下面是 html 文件的结构图 html 文件主要分为两个头部（<head></head>）和身体部分（<body></body>）

<head> 是 HTML 中的一个标签，用于定义文档的头部区域。它位于 <html> 元素内部，并包含了与文档相关的元数据和链接。

<body> 是 HTML 中的一个标签，用于定义文档的主体内容区域。它位于 <html> 元素内部，包含了网页上实际展示给用户的内容，如文本、图像、链接、表单等。如图 31。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>

</body>
</html>
```

图 31 代码 1

用 CSS 函数将路径下是 2.jpg 的图片定义为网页背景：如图 32

```
body{
        background-image: url('../static/2.jpg');
        background-size: cover;
    }
```

图 32 2.jpg 的图片定义

用 创建两个标签并用 CSS 来重定义标签的位置，
换行，标题占用网页的宽度为 100%，字体为水平居中，行高为 100px，段落高度为 100px，字体的大小尺寸为 60pt：如图 33 和图 34

```
<div class="chat">
    <h1>党史任务学习<br /></h1>
    <h1>第十四组<br /></h1>
</div>
```

图 33 字体设置

```
.chat h1{
        width:100%;
        text-align:center;
        line-height:100px;
        height:100px;
        font-size:60pt;
    }
```

图 34 位置设置

用 form 函数写一个输入框，输入框在 eva 类型里，输入内容为 message 发送的目的 URL 为“/chat.3.html”，请求方式为 post，<button>为定义按钮：

如图 35

```
<div class="eva">
    <ul id="messages"></ul>
    <form action="/chat.3.html" method="POST" class="parent">
        <input id="message" type="text"  name="message" placeholder="输入消息" >
        <button type="submit">提交</button>
    </form>
</div>
```

图 35 提交按钮设置

接下来用 CSS 对输入框进行修饰，设置输入框宽度为 380px，高度为 40px，输入框外宽度为 1px，总宽度为 42px，输入框内字体大小为 16px，首行缩进 10px：

如图 36

```
.parent>input:first-of-type {
            /*输入框高度设置为40px，border占据2px，总高度为42px*/
            width: 380px;
            height: 40px;
            border: 1px solid black;
            placeholder: 16px;
            outline: none;
            padding-left: 10px;
        }
```

图 36 输入框修饰代码

接下来定义鼠标点击输入框，输入框的颜色为#317ef3，输入的内容首行缩进10px：如图 37

```
.parent>input:first-of-type:focus {
            border: 1px solid #317ef3;
            padding-left: 10px;
        }
```

图 37 交互显示设置

接下来定义 button 按钮，高度为 42px，高度与输入框加上外边距的高度相同，用 position 定位位置临近输入框，定义颜色为#317ef3，定义按钮里字体的大小为16px：如图 38

```
.parent>button:last-of-type {
            /*button按钮border并不占据外围大小，设置高度42px*/
            width: 100px;
            height: 42px;
            position: absolute;
            background: #317ef3;
            border: 1px solid #317ef3;
            color: #fff;
            font-size: 16px;
            outline: none;
        }
```

图 38 按钮里字体的大小设置

做一个问答边框，高度为 400px，宽度为 600px，背景颜色为白色，位置水平居中，边缘凹进 10px：如图 39

```
.eva {
    width:600px;
    height:400px;
    background:white;
    margin:auto;
    border-radius:10px;
}
```

图 39 定义问答边框

5.5 问答系统 html 尾页

在 templates 文件中创建一个 html 文件命名为 chat.3.html，用< div >< /div >划分一个区域，用来接受 flask 框架返回的对象，并将其展示在区域内，< h1 >< /h1 >为标题，{{ key }}中 key 为返回值，< li >< /li >为创建一个 li 列表，将返回的数据对象进行整理，排版，完美的展示在 eva 区域里面。如图 40

```
<div class="eva">
    <h1>提交结果:</h1>
    <h1>{{ key }}</h1>
    <ul>
        {% for product in products %}
        <li>{{ key }}</li>
        {% endfor %}
    </ul>

</div>
```

图 40 接受后端返回值

用 CSS 对 eva 区域进行修饰，区域大小：宽为 600px，高为 400px，背景颜色为白色，区域在 html 页面水平居中，边缘向内凹进 10px，用 overflow-y:auto;语句设置一个滚动栏，用来防止文本过多，造成文本溢出，用 font-family: "Lucida Calligraphy", cursive, serif, sans-serif;来更改区域内的字体。如图 41

```
.eva {
    width:600px;
    height:400px;
    background:white;
    margin:auto;
    border-radius:10px;
    overflow-y:auto;
    font-family: "Lucida Calligraphy", cursive, serif, sans-serif;
}
```

图 41 修饰字体设置

在底部加一个超链接，为超链接设置一个区域，背景颜色为白色，宽度为200px，顶部距离为 280px，字体大小为 20pt，边缘凹进 10px，字体水平居中，设置字体的颜色为黑色，用 text-decoration:none;来取消超链接的下划线。如图 42 和图 43

```
<div class="hue">
    <div class="bsn">
        <a href="chat.1.html">点击返回上一级</a>
    </div>
</div>
```

图 42 加超链接

```
.hue {
    background-color:white;
    width:200px;
    margin-top:280px;
    font-size:20pt;
    margin:20px 670px;
    border-radius:10px;
    text-align:center;
}
a{
    text-decoration:none;
    color:black;
}
```

图 43 修饰代码

5.6 flask 框架实现

设置 flask 路由 如图 44

```
app = Flask(__name__)
```

图 44 设置 flask 路由代码

Flask 要用到 render_template 和 request 函数，添加路由@app.route（‘/’），用 GET 请求返回到 chat.1.html 页面 如图 45

```
@app.route('/', methods=['GET','POST'])
def index():
    return render_template('chat.1.html')
```

图 45 GET 请求返回到 chat.1.html 页面

再继续设置两个路由，一个路由通过 POST 请求接受 chat.1.html 函数通过 form 发送过来的数据对象，并将其保存在 message 中再返回到 chat.3.html 页面中，另一个路由用来实现 chat.1.html 与 chat.3.html 两个页面之间的输入与返回的循环。如图 46

```
@app.route('/chat.3.html', methods=['GET','POST'])
def send():
    message = request.form.get('message')
    return render_template('chat.3.html',key=message)
@app.route('/chat.1.html', methods=['GET','POST'])
def eta():
    return render_template('chat.1.html')
```

图 46 实现循环代码

5.7 结果展示

问答系统首页展示：如图 47

图 47 问答系统首页

问答系统尾页展示：如图 48

图 48 问答系统尾页

6. 问答系统前端

问答系统通过调用写一个对话式的 web 平台，在这个群平台里可以实现对话、查询和搜索功能。将项目打造成能够上线的查询平台。整个项目基于 flask 框架开发。

6.1 基于 flask 的 python 启动代码：

主要目的是创建一个简单的 Web 应用，当用户访问根路径时，会显示一个登录页面。它使用了 Flask 框架提供的功能来处理 HTTP 请求和生成动态内容，通过运行 Flask 应用，可以在本地开发服务器上部署该应用。其代码如下：

```
1. from flask import Flask, render_template, request, jsonify
2. import flask
3. app = Flask(__name__)
4.
5. from flask import Flask, render_template, request
```

```
6.
7.  app = Flask(__name__)
8.  @app.route('/')
9.  def hello_world():
10.         return render_template("login.html")
11.  if __name__ == '__main__':
12.         app.run()
```

导入所需的 Flask 模块和函数。创建一个 Flask 应用对象，命名为 app。定义了一个路由（route）为根路径 '/' 的处理函数 hello_world()。在 hello_world() 函数中，使用 render_template() 函数加载名为 "login.html" 的模板文件。将加载的模板作为响应返回给客户端，即在用户的浏览器上显示 "login.html" 的内容。最后，通过调用 app.run() 启动应用，监听客户端的请求并进行响应。

6.2 登陆页面

登录页面就是输入账号和密码的地方，当输入的账号密码匹配即登陆成功，否则不成功。若没有账号则可以点击注册。核心代码如下：

```
1.  <!DOCTYPE html>
2.  <html lang="en">
3.
4.  <head>
5.      <meta charset="UTF-8">
6.      <title>党宝！</title>
7.  </head>
8.
9.  <body>
10.     <div class="container-login100">
11.         <form class="login100-form validate-form">
12.             <span class="login100-form-title pb-5">登录</span>
13.             <div class="panel panel-primary">
14.                 <ul class="nav panel-tabs">
15.                     <li class="mx-0"><a href="#tab5" class="active"
    data-bs-toggle="tab">邮箱登录</a></li>
16.                     <li class="mx-0"><a href="#tab6" data-bs-toggle="tab">手机登录
    </a></li>
17.                 </ul>
18.                 <div class="tab-content">
19.                     <div class="tab-pane active" id="tab5">
20.                         <input class="input100 form-control" type="email"
    placeholder="邮箱">
21.                         <input class="input100 form-control" type="password"
    placeholder="密码">
```

```
                                <a href="t" class="login100-form-btn btn-primary">登录“党宝”</a>
                            </div>
                            <div class="tab-pane" id="tab6">
                                <input class="input100 form-control" type="text" placeholder="手机号">
                                <a href="javascript:void(0)" class="login100-form-btn btn-primary">登录</a>
                            </div>
                        </div>
                    </div>
                </form>
            </div>
</body>

</html>
```

上述代码定义了登录表单的 HTML 结构，包含了两个登录方式选项卡：邮箱登录和手机登录。用户可以通过点击选项卡来切换登录方式。在邮箱登录选项卡中，用户需要输入邮箱和密码进行登录；在手机登录选项卡中，用户需要输入手机号和验证码进行登录。结果如下图 49 所示：

图 49 登陆页面

6.3 主页

主页面提供了多个功能调用按钮，使用户能够方便地访问以下服务：搜索引擎、问答系统调用界面、知识图谱查看页面和人物介绍查看界面。这些按钮被设计成可点击的元素，用户可以通过点击页面上的任意位置来选择他们感兴趣的服务。这种设计使用户能够以直观的方式快速访问所需的功能，提高了用户体验和操作效率。无论用户想要进行信息检索、提问、浏览知识图谱还是查看人物介绍，

他们都可以通过简单的点击操作来实现。这种专业性的设计追求了用户友好性和功能可访问性，为用户提供了全面且高效的服务体验。本系统的主页面如下图50所示。

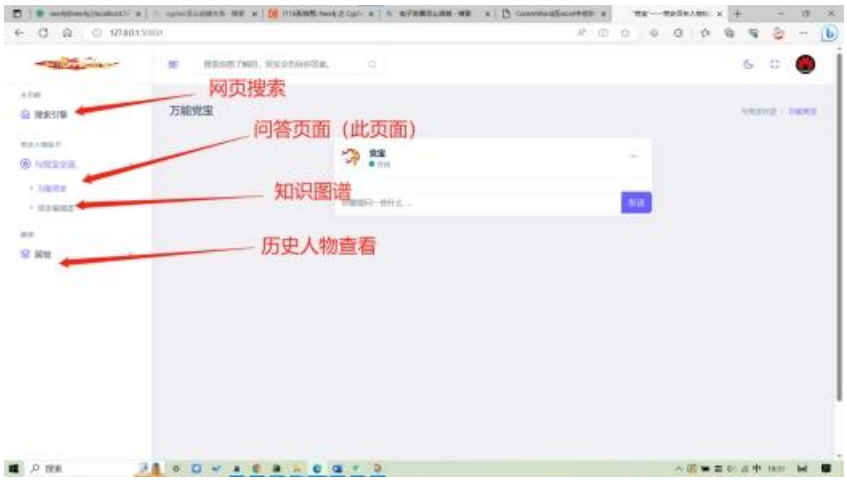


图 50 主页面

6.4 选择问答搜索

问答系统通过与后端的交互，能够将提问者所提出的问题传递到后端进行数据查询和处理（请参考实验4部分的代码）。后端对问题进行处理和分析，并返回处理后的结果，将其以流式数据的形式传递到前端进行展示。在前端界面上，经过问答系统处理后的结果将以直观的方式呈现给用户。

图 51 问答系统

这种问答系统的设计目标是通过与后端的无缝协作，实现对提问者的问题进行准确、高效的回答。后端利用丰富的数据资源和算法进行问题解析和搜索，从而提供具有深度和广度的回答内容。前端界面在接收到后端传递的流式数据后，通过合适的呈现方式将问题的处理结果展示给用户。这种专业性的问答系统设计旨在提供优质的用户体验，满足用户对知识获取和问题解答的需求，并为用户提

供全面而精准的信息支持。

6.5 选择网页搜索

当点击“搜索引擎“时，页面会跳到 chat.1.html 上，其定义了一个搜索页面，能够从前端获取所问的问题，并返回给后端，经过后端处理完毕后，将结果又返回到前端界面。其核心代码如下：

```
<!DOCTYPE html>
<html>
<head>
        <title>Chat</title>
        <meta charset="UTF-8">

</head>
<body>

        <div class="main">
            <div class="chat">
                <h1>党史任务学习<br /></h1>
                <h1>第十四组<br /></h1>
            </div>
            <div class="eva">
                <ul id="messages"></ul>
                <form action="/chat.3.html" method="POST" class="parent">
                    <input id="message" type="text"    name="message" placeholder="输入消息" >
                    <button type="submit">提交</button>
                </form>
            </div>
        </div>
```

上述代码用于创建一个聊天界面。页面中包含了一个标题和一个聊天消息显示区域，以及一个输入框和提交按钮用于发送消息。通过使用`<div>`、`<h1>`和文本内容，我们创建了一个聊天界面的标题部分，其中包含了"党史任务学习"和"第十四组"的文本。接下来，使用`<ul>`标签创建了一个无序列表，用于显示聊天消息。该列表的 id 属性设置为"messages"，以便后续的 JavaScript 代码可以操作和更新这个列表。然后，使用`<form>`标签创建了一个表单，其中的 action 属性指定了表单提交的目标页面为"/chat.3.html"，method 属性指定了提交方法为 POST。在表单中，使用`<input>`标签创建了一个输入框，用户可以在这里输入要发送的消息。输入框的 id 属性设置为"message"，name 属性设置为"message"，

用于后端处理时获取用户输入的消息。最后，使用`<button>`标签创建了一个提交按钮，用户点击该按钮可以提交消息。通过上述 HTML 元素的组合，我们实现了一个简单的聊天界面，用户可以输入消息并通过提交按钮发送，已发送的消息会显示在界面上。其页面效果如下图 52 所示。

图 52 搜索引擎问题界面

图 53 搜索引擎回答界面

6.6 选择查看人物

当用户点击"党史人物"链接时，将导航到"人物馆"页面，这个页面提供了对历史人物的详细介绍。用户可以在该页面选择任意一个历史人物，然后获取对该人物生平的全面讲解。这个功能满足了用户对党史英雄任务的深入了解的需求，并提供了对历史人物的全面了解和学习的机会。这样的设计使用户能够通过点击特定的人物来获得相关任务的生平背景、贡献和故事，为用户提供了一个细致而

全面的党史学习体验。

图 54 查看历史人物及介绍界面

小组任务分工

学号	姓名	任务分工
20201826	高树林	构建知识图谱和实现知识图谱的查询、搭建对话系统、测试各环节稳定性和准确率
202018525	郭彦溥	文本标注
20201827	黄士杰	前端搜索引擎代码实现
20201823	徐柏婷	通读文本，确定实体和关系类型

202018526 高树林完成任务的情况介绍

1.完成任务介绍(包括任务内容)

在这个实验中，我承担了多个重要任务。首先，我负责构建知识图谱，耗费了大量时间和精力进行数据搜集、整理和建模，确保知识图谱的准确性和完整性。其次，我实现了知识图谱的查询功能，设计和优化了查询算法，以提供高效且准确的搜索结果。我还搭建了一个强大的对话系统，通过深入研究自然语言处理和机器学习技术，使得系统能够智能地理解用户的问题并给出准确的答案。最后，我进行了大量的测试工作，检验各环节的稳定性和准确率，并根据测试结果不断优化和改进系统。我在这个实验中付出了巨大的努力和心血，以确保系统的质量和性能。

2.对于团队的贡献(包括任务在整个课题中的重要性等)

在这个实验中，我承担了多个重要的任务，对团队的贡献是不可忽视的。首先，我花费了大量的时间和精力来构建知识图谱，从数据搜集到建模，我投入了许多辛勤的努力，以确保知识图谱的准确性和完整性。我运用我的专业知识和技能，在这一过程中发挥了关键的作用。其次，我负责实现知识图谱的查询功能，设计并优化了查询算法，以使用户能够高效地检索和获取所需的信息。我发挥了我的技术能力和创新思维，为团队带来了有价值的功能和特性。此外，我搭建了强大的对话系统，通过深入研究自然语言处理和机器学习技术，使系统能够智能地理解用户的问题并给出准确的答案。我的工作为团队提供了核心的交互功能，使用户能够与系统进行有意义的对话和交流。

除了技术方面，我还进行了大量的测试工作，验证各个环节的稳定性和准确性。我始终保持严谨的态度和专注的精神，以确保系统的质量，并为团队提供了可靠的基础。

我的努力、专业知识和技能在这个实验中起到了至关重要的作用。我的贡献不仅体现在技术实现上，还体现在团队协作和项目推进中。我的工作对整个团队的成功和成果产生了积极的影响，为项目的顺利完成做出了重要贡献。

3.主要实现思路和使用的方法

获取实体关系信息：预先标注的索引信息提供了实体名称在文本中的位置。这

些索引信息可以是人工标注得到的，也可以通过一些自动化方法生成。索引信息可以包括实体名称、起始位置和结束位置等。根据预先标注的索引信息，可以在文本中找到对应实体的位置。可以通过匹配索引中的起始位置和结束位置来定位实体在文本中的位置，并提取出实体名称。通过标注工具生成的文本关系示意图可以帮助理解实体之间的关系。这可以是通过连接实体之间的边或箭头来表示关系，或者使用其他可视化方式来展示实体之间的关系。

避免创建重复节点：我通过查询数据库来检查是否已存在具有相同属性的实体。这可以通过执行数据库查询语句来搜索匹配条件的实体记录。在数据库设计中，可以为实体的某些属性设置唯一性约束。这意味着这些属性的取值在整个数据库中必须唯一。通过利用唯一性约束，系统可以自动检查实体属性的唯一性，避免重复创建相同属性的实体。在代码逻辑中，我会在创建实体之前先判断数据库中是否已存在相同属性的实体。这可以通过查询数据库返回的结果进行判断，如果存在匹配的实体记录，则判定实体已存在，无需重复创建。

高效的实体关系图谱构建：我使用循环结构或迭代器来遍历所有的数据。这可以确保每个数据都能被处理到，并在每次迭代中调用创建函数。在创建函数中，我根据数据的特征和属性来创建对应的节点。节点可以代表实体，其中包含实体的唯一标识和属性信息。除了创建节点，我还会创建节点之间的边。边表示实体之间的关系，可以用来建立实体之间的连接。根据数据的关联关系，我会在创建函数中添加相应的边，将不同节点连接起来。通过在遍历过程中反复调用创建函数，我逐步构建整个知识图谱。每次调用创建函数都会添加新的节点和边，逐步丰富图谱的结构。为了处理大量数据并提高效率，我可能会采用一些优化方法。例如，使用批量操作来批量创建节点和边，减少数据库操作的次数，从而提高性能。

正相匹配查询算法：首先，我创建了一个词典，其中包含实体、关系和属性的关键词。词典中的关键词应涵盖待查询字符串中可能出现的实体、关系和属性。我使用循环结构来遍历查询字符串。在每次迭代中，我以每次最大长度进行切割子串，以便逐个检查子串是否在词典中存在。对于每个子串，我检查它是否在词典中存在。如果存在，则将匹配到的词、起始位置和结束位置以及对应的类型添加到一个列表中。根据关键词在查询字符串中的起始位置，我对列表进行排序，

以确保关键词按照其在查询字符串中的顺序排列。最后，我将排序后的关键词列表保存在一个对象列表（obj_list）中，以便后续的查询和分析使用。

生成 Cypher 语句：通过逐个遍历关键词列表，确定每个关键词在查询字符串中的起始位置和结束位置，并提取待替换的字符。使用预定义的槽位来替换查询字符串中的关键词，将其转化为特定的占位符。这样做的目的是将查询字符串标准化，并为后续的模板匹配做准备。根据替换后的查询字符串，与预先定义的 Cypher 语句模板进行匹配。找到与查询字符串匹配的模板后，使用正则表达式确定模板中的所有槽位的位置。依次遍历槽位列表，将关键词逐个替换到对应的槽位位置，形成最终的 Cypher 语句。这样，每个关键词都被正确地插入到对应的模板槽位中，生成了完整的 Cypher 语句。生成的 Cypher 语句可以被用于在图数据库中执行查询操作，以检索与知识图谱相关的信息。通过执行 Cypher 语句，我们可以获取符合查询需求的结果集，从而获得所需的知识图谱数据。

实现问答系统：首先，与图数据库建立连接，确保可以与数据库进行交互。这一步骤通常涉及指定数据库的连接参数、验证身份和权限等。根据问题，使用之前生成的 Cypher 语句模板库，匹配到与问题相对应的 Cypher 语句模板。这些模板包含了特定的查询结构和关键词占位符。将匹配到的 Cypher 语句模板填充上具体的实体名称或关键词，形成完整的 Cypher 查询语句。这个过程可以利用之前提取的关键词或问题中的实体信息进行替换。使用完整的 Cypher 查询语句来查询图数据库，获取与问题相关的实体和关系信息。通过执行 Cypher 查询，可以获得符合查询条件的结果集。对查询结果进行分析和判断，根据需要提取有用的信息或进行进一步的推理。这可以包括从结果中提取特定属性、判断实体之间的关系、进行排序或过滤等操作。

flask 框架连接后端：在创建这个简单的 Web 应用的过程中，您采用了以下思路和方法：选择了 Flask 框架来创建 Web 应用。Flask 是一个轻量级的 Python Web 框架，它提供了处理 HTTP 请求和生成动态内容的功能。通过使用 Flask，您能够快速搭建起一个简单的 Web 应用。Flask 框架提供了路由功能，可以定义不同的路由路径和对应的处理函数。在这个应用中，当用户访问根路径时，我定义了一个处理函数来处理该请求，并生成登录页面的内容。通过使用 Flask 的模板引擎，可以在登录页面中生成动态内容。模板引擎允许在 HTML 页

面中嵌入动态数据，以便根据用户请求动态地生成页面内容。

4.所完成任务的效果展示与总结

图 1 完整的知识图谱

图 2 系统登陆页面

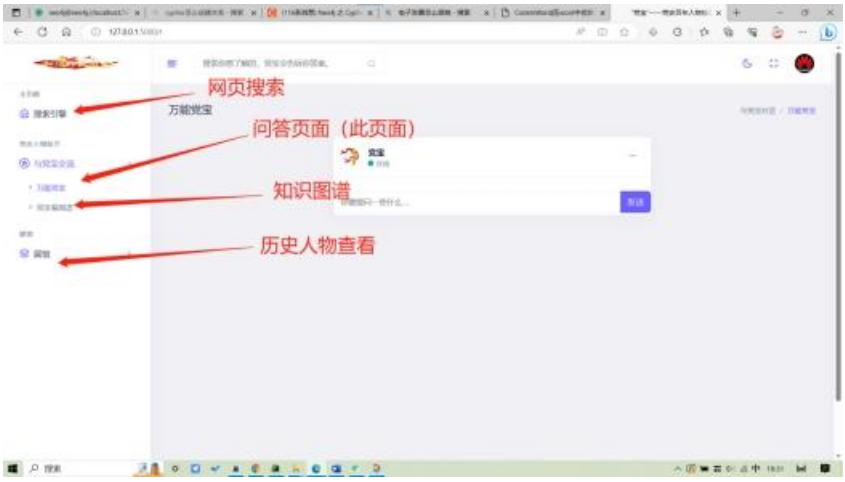


图 3 系统主页面

图 4 问答系统页面

图 5 提问界面

图 6 回答界面

图 7 查看历史人物界面

5.对于任务未来改进的思考

在这个实验中，我认为可以有以下改进方向：

1. 用户界面改进：可以进一步提升用户界面的设计和用户体验，使其更加直观、易用和美观。可以考虑增加一些交互元素、改进布局和配色方案，以及优化页面加载速度，提升整体的用户满意度。

2. 引入自然语言处理技术：目前的问答系统可能还有改进的空间。可以考虑引入自然语言处理技术，例如使用更高级的语义理解和问句匹配算法，以提高问答的准确性和灵活性。这样用户可以更自由地提问，系统能更好地理解和回答用户的问题。

3. 数据库性能优化：如果在实验过程中遇到了性能瓶颈，可以考虑对数据库进行优化。可以通过索引、分片、缓存等技术手段来提升数据库的查询和访问速度，从而提高整个系统的响应性能。

4. 扩展知识图谱：可以进一步扩展知识图谱的内容和覆盖范围，以提供更全面和丰富的知识。可以收集更多领域的数据，不断更新和扩充知识图谱，以满足用户对多领域知识的需求。

5. 引入机器学习算法：可以考虑将机器学习算法应用于问答系统中，例如使用机器学习模型进行问题分类、实体识别和关系抽取等任务，以提升系统的智能化水平和准确性。

202018525 郭彦溥完成任务的情况介绍

1.完成任务介绍(包括任务内容)

实体关系标注是指从文本中抽取出实体及其之间的关系，是自然语言处理中的一项重要任务。实体关系标注的任务介绍包括：实体识别、关系抽取和实体链接等。其中，实体识别是指从文本中识别出命名实体，如人名、地名、机构名等；关系抽取是指从文本中抽取出实体之间的关系；实体链接是指将命名实体与其对应的 ID 进行关联，以便于后续的查询和分析。

利用百度 easydl 平台的数据标注来进行实体关系标注，原文件处理和标注数据。

2.对于团队的贡献(包括任务在整个课题中的重要性等)

实体关系标注在知识图谱建立中是非常重要的一环。知识图谱是一个结构化的知识存储和表示形式，由实体（代表现实世界中的事物）和关系（描述实体之间的联系）组成。实体关系标注的目标是将自然语言文本中的实体和它们之间的关系映射到知识图谱中的实体和关系上。这个过程包括识别文本中的命名实体和实体关系，并为它们分配合适的类型和属性。

实体关系标注一般涉及到实体抽取和关系抽取两个任务。实体抽取的目标是从文本中识别出具有特定类型的实体，比如人名、地名、组织机构等。而关系抽取的目标是识别实体之间的关系，比如"作者"、"出生地"、"成立时间"等。

通过进行实体关系标注，可以将海量的非结构化文本转化为结构化的知识表示形式，进而支持知识图谱的建立和应用。对于知识图谱的构建来说，准确的实体关系标注能够提供丰富的关联信息，帮助我们理解和发现实体之间的关系，进而为知识推理、问题回答等任务提供支持。

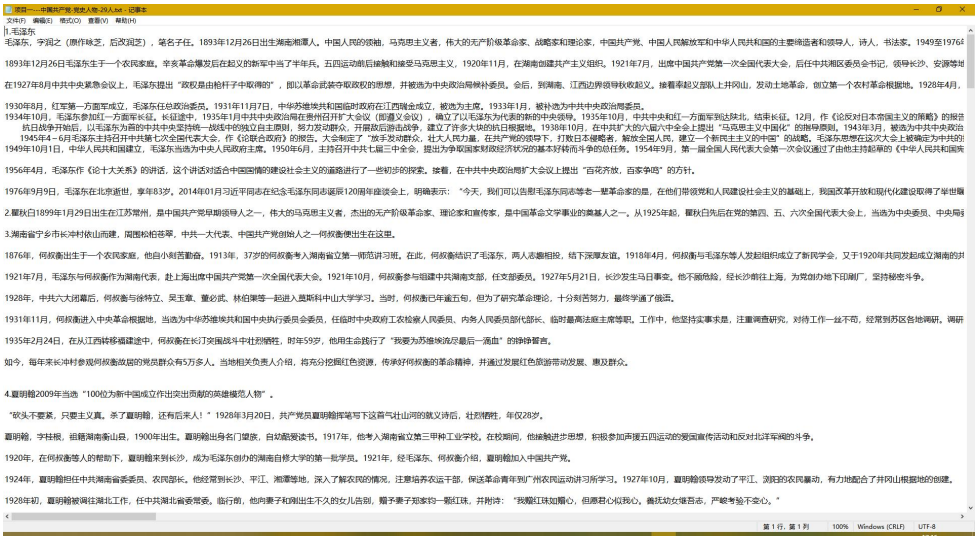
这一步的完整和完美程度，直接影响了小组在对之后代码实现的轻易程度，只有一个恰倒正确的实体关系抽取，才能在后面的代码中节省时间。

3.主要实现思路和使用的方法

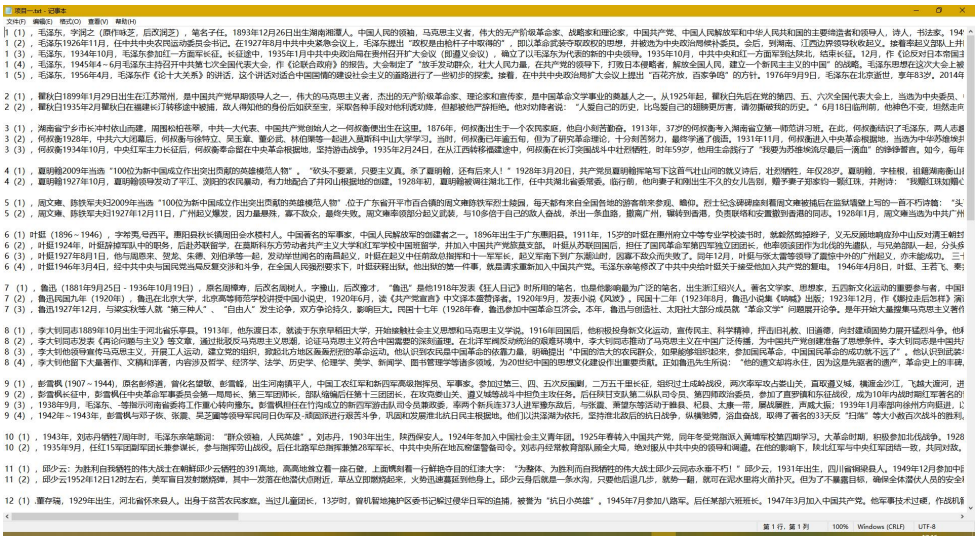
3.1，处理源文本数据

原文本数据杂乱无章，经常出现，无人物无重点的情况 标注困难，杂乱无章，先行处理源文本数据会让之后的标注更简单。图一为原文本文件，图二为修

改过后的。



图一（原文本文件）



图二（修改之后）

3.2 添加数据

点进“文本实体关系抽取”

点击数据总览，创建数据集

写入该数据集的名称，并导入

选择“本地导入”，“上传 TXT 文本”，分隔符选择“换行符”，数据去重不去重都可以，然后上传文本。

图四为创建过程

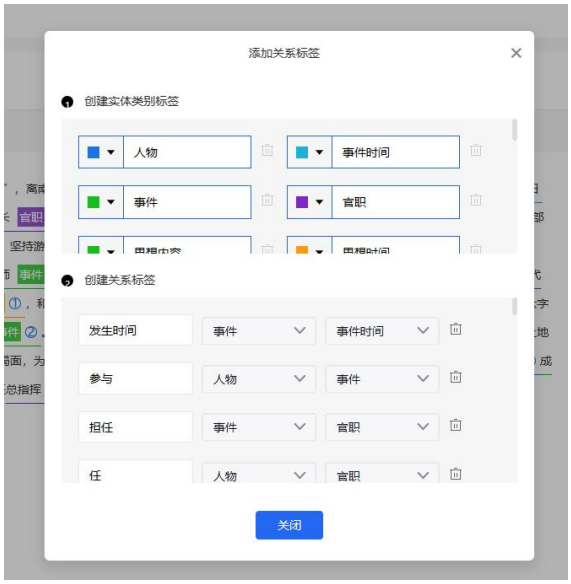
图三

3.3 进行实体关系的创建和标注。

数据导入完成后，点击“标注”，进入到下一个页面。

可以看出，文本清晰明了，有“人物”这个实体中心

右上角添加“实体”以及实体之间的“关系”



图四

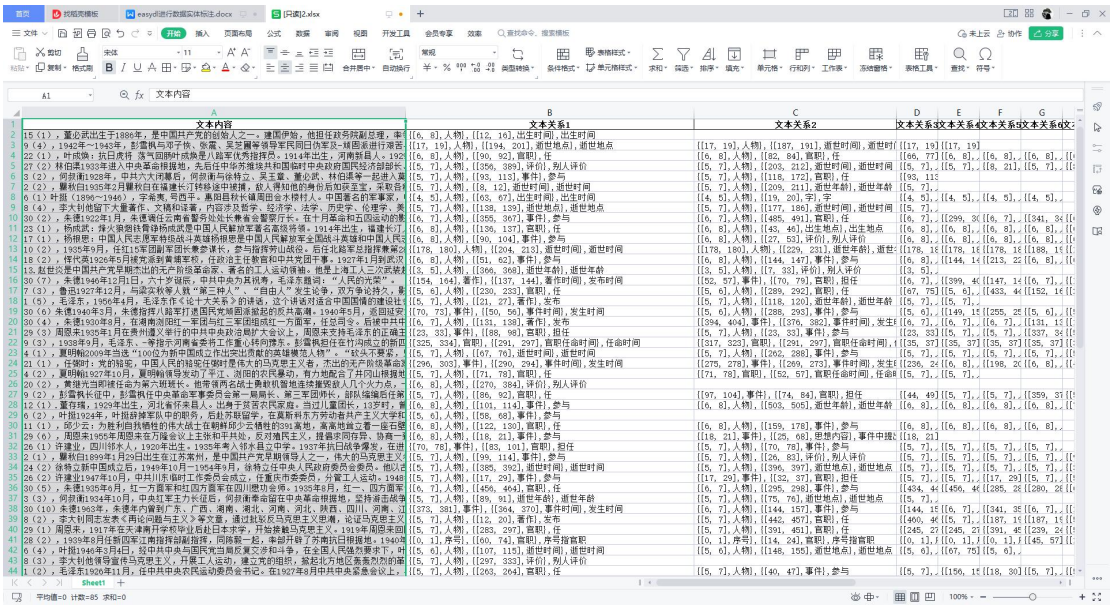
3.4 标注

枯燥的标注过程不多赘述

3.5 导出数据

导出即可

4.所完成任务的效果展示与总结



图五 实验结果

5.对于任务未来改进的思考

再修改的话，还是需要在原文本数据上修改，整体标注过程是一样的，标注的难易程度以及后续的代码实现，要的都是对原文本数据的修改，在之后，要修改的话，就需要对文章内容进行通读，然后精炼，并创建一个新的文本文件，并把精炼内容写到新文本文件上，文本文件还需要符合 easydl 文本关系提取功能的限制。

202018527 黄士杰完成任务的情况介绍

1.完成任务介绍(包括任务内容)

负责组内搜索引擎前端的制作，主要利用 CSS 和 HTML 代码生成两个交互网页，要求要有网页背景，完整的对话框，设置对话框的底色，设置对话字体格式，设置对话字体颜色，设置网页大小、布局、背景，设置超链接返回搜索框，实现搜索与返回的页面跳转，设置对话边框凹进效果。

运用 flask 框架对生成的两个网页进行数据传递，用@app 定义路由节点，用 render_template 来实现对 html 页面的返回和数据传递操作，用 request 实现 form 操作输入的接收，flask 主要实现一个接口的作用，通过将前端网络页面的输入传递到后端来进行数据的处理，最后再将 python 处理好的数据返回到前端 html 页面进行展示，最后定义一个路由，进行 html 的自动跳转返回的操作，这样实现了两个输入页面的切换操作。

2. 对于团队的贡献(包括任务在整个课题中的重要性等)

在整个课题中，搜索引擎处于比较前列的地步，它可以将一个项目内容，与广大群众实现一个完美的对接，让每一个人都能感受到到科技带来的便捷，扩大项目的影响，增强项目的展示效果。搜索引擎前端的制作，丰富了课题内容，让团队对结果的检查进行的更顺畅，更简洁。Html 网页的引入，让团队的展示有一个完美的载体，团队可以对 html 页面进行随意的更改，增强了项目的展示效果，让项目变得更加多彩，flask 框架实现 html 跳转和数据传递，增加了项目的趣味性，让项目的进行不再是一行行代码，一个个报错，而是页面的一次次变化，结果的一次次轮回，搜索引擎的实现是整个课题的一个重大突破。

3.主要实现思路和使用的方法

用 python 里的 flask 框架（Flask 是一个基于 Python 的微型 Web 框架，它简洁而灵活，适用于构建小型到中型的 Web 应用程序和 API。它被设计成简单易用，具有可扩展性和可定制性，是许多开发者选择的首选框架之一）实现两个 html 页面的数据交互与跳转。首先在 templates 文件下面写入两个网页，chat1 和 chat3，chat1 里用 h1 写入了两个标题、用 form 写了一个输入框，再用 CSS 对标题和 form 输入框进行修饰，首先对标题用 width 实现行距大小，用 line—height 和 height 来定义行高和段落高度，用 font-size 定义字体的大小尺寸为 60pt，

form-action 实现将数据发往的地址，form-methon 定义是数据传递的类型，用 width 和 height 定义输入框的长度和宽度，对输入框的背景颜色，按钮的背景颜色，输入框外边距，首行缩进大小进行更改。

Chat3 相对于 chat1 少了输入的部分，多了数据接受的部分和超链接跳转的部分，用<a>写入一个超链接，URL 写 chat1 的 URL 这样实现页面的跳转，用{% for product in products %}和{% endfor %}来实现对数据的接收，并将数据完美的展示在页面上，后面用 CSS 对页面的布局，字体大小格式，进行整理。

Flask 框架，引用路由，用@app 定义路由节点，用 render_template 来实现对 html 页面的返回和数据传递操作，路由的请求有 GET 和 POST 两种类型，GET 请求不能发送数据，POST 请求可以是实现对数据的传递效果，用 POST 请求接受前端传递的数据并保存到 message 里，并用 python 后端调用它，用 render_template 来返回处理后的数据

4.所完成任务的效果展示与总结

图 1

图 2

在本次实验中，我们学习并使用了 Flask 框架进行 Web 应用程序的开发。以下是对实验过程和结果的总结：

1.了解 Flask 框架：我们首先对 Flask 框架进行了初步的了解，学习了其基本原理和用法。Flask 是一个简单而灵活的 Python 微型框架，适用于快速搭建 Web 应用。

2.搭建开发环境：为了开始使用 Flask，我们需要在本地搭建一个开发环境。通过安装 Python 和相关的依赖库，以及创建虚拟环境，我们成功地设置了 Flask 开发环境。

3.创建 Flask 应用：我们按照 Flask 的要求，创建了一个 Flask 应用，并编写了路由处理程序、视图函数和模板文件。这些组成部分共同实现了应用的基本功能。

4.定义路由和视图函数：我们定义了一些路由，用于将 URL 与相应的处理函数关联起来。通过在视图函数中编写业务逻辑，我们能够根据不同的 URL 请求返回不同的响应。

总体而言，本次实验使我们熟悉了 Flask 框架的基本使用方法和开发流程。我们能够创建简单的 Web 应用，并处理表单、用户输入等常见功能。通过这个实验，我们深入了解了 Web 应用开发的一些关键概念和技术，为今后的开发工作打下了坚实的基础。

5.对于任务未来改进的思考

1.可以增加一个登录页面，用户输入账号密码，将数据以 json 格式传递到服务器

或者浏览器本地，进行比对，增加项目的安全性。

2.增加动态的页面，可以添加动态的背景与视频，降低审美的疲劳，也能使整体看起来更形象美观。

3.增加本地保存功能，方便用户对自己搜到的结果进行下载保存，增加用户体验感。

4.增加问答页面，实现人机交互，增加用户体验感。

5.对数据输入进行过滤，加强后台防护，增加网站安全性，抵御不发分子的攻击，维护用户隐私安全。

202018523 徐柏婷完成任务的情况介绍

1.完成任务介绍(包括任务内容)

Protégé是一个本体建模工具软件，由斯坦福大学基于 java 语言开发的，属于开放源代码软件。具体来说，Protégé具有以下功能。

类建模。Protégé提供了一个图形化用户界面来建模类（包括概念）和它们的属性以及关系；实例编辑。根据创建的类型，Protégé会自动产生交互的形式，可以根据类之间的关系获得相应实例的约束，并对实例进行编辑；模型处理。Protégé有一些插件库，可以定义语义、解答询问以及定义逻辑行为；模型交换。最终的模型（类、实例、关系、属性等）能以各种各样的格式被保存和加载，包括 XML、UML、RDF、OWL 等。

该实验是用 Protégé软件进行实体关系标注的前提构想，构建好实体关系，为之后的实验步骤做铺垫。

2.对于团队的贡献(包括任务在整个课题中的重要性等)

使用 Protégé是为了让之后实体关系标注和抽取更方便，不用一遍一遍的工作，而是在标注和提取之前就构思好实体和关系，从而让后面的工作更轻易。

3.主要实现思路和使用的方法

3.1 下载与安装

3.11 软件安装

软件到官网 https://protege.stanford.edu/products.php 直接下载直接运行 Protege.exe 即可

3.12 可视化插件的安装

下载插件：http://www.graphviz.org/download/；选择 windows 的稳定版本。（graphviz-2.38）解压后放到 protégé安装包的目录下。（也可以放在其他路径）在 protégé的 file 引用插件，单击“Preferences”，新页面找到“OWLViz” tab，修改“Path to DOT”，为对应的地址。

3.13 添加环境变量

略

3.2 创建类，创建实体

打开 Classes 界面，owl:Thing 上的三个按钮从左至右，依次为“添加子目

标”“添加平级目标”和“删除目标”。其它界面同理。

3.3 创建实体关系属性

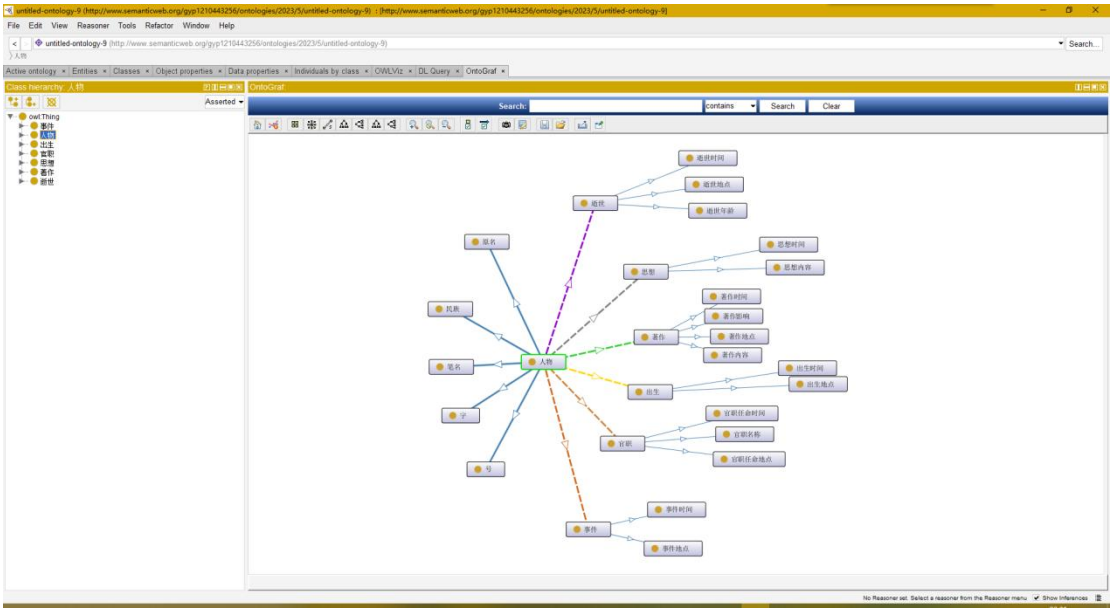
进入 Object Properties 功能界面，构建属性。右下角创建实体之间的关系 Domains 指主语，Ranges 为宾语，主题关系为“人物”“参与”“事件”。

3.4 创建实体属性

进入 Data properties 功能界面，创建实体的属性。例如人物的属性有“原名”“字”“号”“笔名”“民族”

4.所完成任务的效果展示与总结

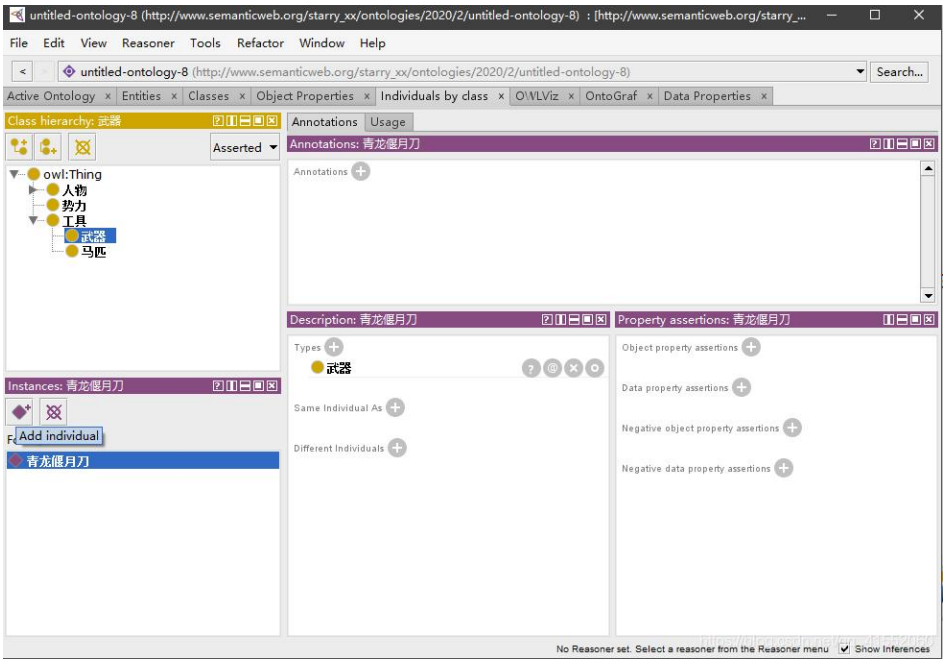
进入 OntoGraf 功能界面，点击即可出现结果



图一 展示结果

5.对于任务未来改进的思考

在未来，可以在未来步骤完成之后，进入 Individuals by class 功能界面，在 Instances 子面板中，添加具体实例。把实体数据填入到软件中，展示结果类似于 neo4j，但效果没有它好。例如：



图二 网上效果图

这个例子（网上效果图）

到最后，进入 OntoGraf 功能界面，双击可以展示实体里具体包含的数据，例如“人物”实体里有“毛泽东”“周恩来”“朱德”等。

小组内部自评排名

学号	姓名	排名
20201826	高树林	1
20201825	郭彦溥	2
20201827	黄士杰	3
20201823	徐柏婷	4