目录

[一、 背景 4](#_Toc16692)

[（一） 这个体系能帮助回答哪些问题？ 4](#_Toc5109)

[（二） 谁会使用和维护这个知识体系？ 5](#_Toc17251)

[二、 知识图谱体系构建 5](#_Toc19576)

[（一） 专家本体构建 5](#_Toc32721)

[（二） 文献类构建 6](#_Toc31726)

[（三） 领域数据属性 7](#_Toc32587)

[（四） 数据联系构建 7](#_Toc8070)

[三、 数据来源与预处理 8](#_Toc19351)

[（一） 数据来源 9](#_Toc15883)

[（二） 数据类型 10](#_Toc27622)

[（三） 数据清洗 11](#_Toc22306)

[四、 Neo4j知识图谱构建 11](#_Toc27064)

[（一） CSV文件导入介绍 11](#_Toc23817)

[（二） 导入流程 12](#_Toc18952)

[五、 Neo4j专家关系网络分析 16](#_Toc5539)

[六、 问答系统 17](#_Toc1213)

[（一） 1. 建立与Neo4j的连接 17](#_Toc25754)

[（二） 定义用户意图模式和响应模板 17](#_Toc5296)

[（三） 用户意图识别 20](#_Toc20444)

[（四） 提取专家名称 20](#_Toc19649)

[（五） 数据库查询与响应生成 21](#_Toc22723)

[七、 前后端页面搭建 22](#_Toc17116)

[（一） 设置防火墙端口 23](#_Toc22431)

[（二） 部署后端 23](#_Toc26589)

[（三） 部署neo4j 24](#_Toc13443)

[（四） 调试 24](#_Toc23458)

[（五） 页面展示 25](#_Toc28544)

[八、 附录 25](#_Toc18147)

基于肉鸡领域研究专家知识图谱

农业科技的不断发展，肉鸡类农业研究在提高生产效率和质量方面发挥着重要作用。肉鸡养殖作为现代农业的重要组成部分，涵盖了遗传育种、饲养管理、疾病防控等多个研究领域。近年来，大量研究成果的涌现为行业提供了丰富的数据和信息资源。然而，这些成果往往分散且缺乏系统化的组织和管理，难以充分发挥其潜在价值。知识图谱是一种通过节点和边的方式来表示实体及其相互关系的技术，能够系统化、结构化地展示数据之间的复杂关系。通过构建知识图谱，可以将分散在各处的科研数据进行整合，并通过可视化工具展示出来，使得科研人员能够直观地理解和分析数据之间的关系，发现潜在的知识和规律，从而提高科研效率和质量。

课设的主要目标是构建一个全面的肉鸡类农业专家知识图谱，通过系统化的知识组织和分析，促进农业科研的信息共享与知识发现，提升科研效率。具体来说，我们将整合从知网上爬取的十位肉鸡类农业专家的论文信息，结合专家信息、研究论文、期刊信息、研究主题和研究单位等多方面的数据，利用Neo4j图数据库构建一个全面的知识图谱。该知识图谱不仅能够揭示农业科研领域中的重要规律和趋势，还能够为科研人员提供有力的支持和参考，推动肉鸡类农业研究的持续发展和创新。通过本研究构建的肉鸡类农业专家知识图谱，科研人员可以通过分析专家之间的合作关系，揭示研究团队和合作网络，发现领域内的重要合作伙伴和研究群体；通过分析论文的研究主题分布，了解当前的研究热点和趋势，识别未来的研究方向；通过分析期刊的影响因子和论文发表情况，评估期刊在特定领域的影响力，帮助研究人员选择合适的投稿期刊。此外，利用Neo4j的可视化工具，还可以直观地展示知识图谱的结构和内容，如专家与其研究成果的关系网络图、期刊之间的引用关系图等，从而更好地理解和分析数据。

# 背景

在肉鸡领域的研究历程中，农业科技的发展起到了至关重要的推动作用。从19世纪中叶中国的丝羽乌骨鸡、北京鸭、九斤黄、狼山鸡等品种被输出到英国、美国等地，与当地品种结合培育出新品种，到上世纪初中国开始引进国外鸡种，再到改革开放后引进、培育白羽肉鸡的过程中，中国在农业，特别是家禽业领域与国际间进行了密切的遗传物质交流，取得了巨大的产业进步。经过40年的发展，中国在肉鸡及家禽产业的产学研领域与欧美国家的水平差距逐渐缩小。

在全球和中国白羽肉鸡产业的发展前景中，技术和人才的竞争优势显得尤为重要。世界家禽学会主席、中国农业大学教授杨宁在接受《国际家禽》杂志记者专访时，对这些因素进行了深入分析和阐释。在改革开放40年的中外家禽业交流史上，2016年在北京举行的第二十五届世界家禽大会具有里程碑的意义。杨宁教授指出，中外家禽业的交流史源远流长，在推动家禽业的进步和创新中发挥了重要作用。早在南北朝时期就有中国鸡品种输出到日本的记录，19世纪中叶，中国的“九斤黄”从上海输出到英国，后又输出到美国，美国利用九斤黄与当地品种杂交培育出了著名品种洛克鸡。中国的丝羽乌骨鸡、狼山鸡、北京油鸡也在这一时期输出到英国和美国，为培育肉用型和观赏型鸡种提供了重要素材。

在现代家禽养殖和科研领域，中国在全球范围内的地位逐步提升。中国农业大学教授杨宁表示，近年来中国在家禽科研领域的发展速度很快，研究队伍不断壮大，研究广度和深度全面发展。在国际知名的家禽科学杂志《Poultry Science》上的投稿论文数量，中国已超过美国，成为世界第一。然而，由于试验设计和语言问题，中国学者论文的接受率还需进一步提高。杨宁教授指出，尽管中国在家禽科研领域取得了显著进展，但在试验设计和预见力等方面仍有改进空间。此外，中国的家禽科研力量在大学和研究所相对集中，而企业中的科研力量较小，需要通过机制创新和优惠政策，鼓励更多高水平的博士生和硕士生进入企业发展，将科研实力转化为产业发展所需的力量。

在知识图谱构建的初期我们思考了几个问题

## 这个体系能帮助回答哪些问题？



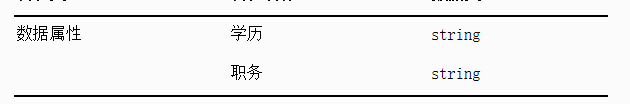
## 谁会使用和维护这个知识体系？



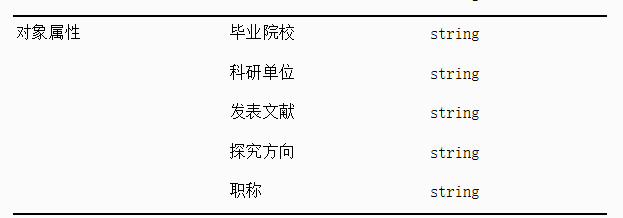
# 知识图谱体系构建

## 专家本体构建

本体专注于肉鸡领域的专家，旨在整合专家信息、研究成果和相关学术产出。

确定本体中的主要实体类型。对于专家本体，其数据属性为

实体定义对象属性为：

1. 职务 (position): 字符串类型，表示专家的职位。
2. 毕业院校 (almaMater): 对象属性，链接到Institution实体。
3. 科研单位 (researchInstitute): 对象属性，链接到Institute实体。
4. 发表文献 (publications): 对象属性，链接到Publication实体。
5. 探究方向 (researchInterests): 对象属性，链接到ResearchField实体。
6. 职称 (jobTitle): 字符串类型，表示专家的职称。

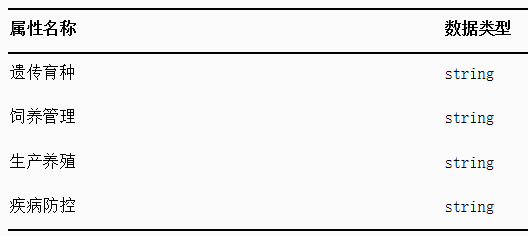
## 文献类构建

文献数据的核心元数据，为文献的检索、分类和分析提供了基础信息。在构建知识图谱或进行文献管理时，可以帮助标准化和组织文献信息。在构建文献类时主要选取了以下的属性构建文献类:

图 1 文献数据类型

1. 发表时间 (publicationDate): 文献发表的日期，数据类型为日期（data）。
2. 关键词 (keywords): 文献的关键词，用于描述文献的主要内容或研究领域，数据类型为字符串（string）。
3. 摘要 (abstract): 文献的摘要部分，通常包含研究目的、方法、结果和结论的简短概述，数据类型为字符串（string）。
4. 题名 (title): 文献的标题，是文献的名称或名称的一部分，数据类型为字符串（string）。
5. 编号 (id): 文献的唯一标识符，通常用于引用或索引，数据类型为整数（int）。
6. 研究内容 (researchContent): 文献的研究内容或主题，详细描述了文献的研究范围或研究问题，数据类型为字符串（string）。
7. 所属单位 (affiliation): 文献作者所属的单位或机构，数据类型为字符串（string）。

## 领域数据属性

描述农业科学领域内肉鸡研究论文和项目的重点领域。构建知识图谱时，重点领域分类属性有助于对相关领域的文献和专家进行分类和关联，进而促进信息的检索和分析。根据肉鸡领域专家论文的预分析,将肉鸡研究领域分为了以下四大领域:

1. 遗传育种 (genetic\_breeding): 涉及遗传学原理和育种技术，用于改良和优化畜种的特性，数据类型为字符串（string）。
2. 饲养管理 (feeding\_management): 关于动物饲养的日常管理实践，包括饲养方法、饲料配方等，数据类型为字符串（string）。
3. 生产养殖 (production\_farming): 指动物生产过程中的养殖技术和管理，可能涉及养殖规模、环境控制等，数据类型为字符串（string）。
4. 疾病防控 (disease\_prevention\_control): 包含动物疾病预防、诊断和治疗的策略和方法，数据类型为字符串（string）。

## 数据联系构建

根据实体之间的关系我们可以分类如下:

1.专家与研究领域：专家进行研究的领域如遗传育种、饲养管理、生产养殖和疾病防控等。

2.专家与文献：专家撰写并发表了与其研究相关的文献，这些文献详细记录了专家的研究成果。

3.文献与研究领域：文献的内容涉及特定的研究领域，通过文献的关键词、摘要和研究主题，可以明确文献所覆盖的研究内容。

可进一步抽取为如下关系:

1.专家实体:

研究（Research）：专家与研究领域之间的关系，表明专家在特定研究领域内从事研究工作。

发表（Publication）：专家与文献之间的关系，表明专家是某些研究文献的作者或共同作者。

2.文献类

发表（Authored by）：文献与专家之间的关系，表明某文献是由哪些专家撰写的。

研究内容（Related Content）：文献与研究领域之间的关系，表明某文献涉及哪些研究领域。

3.领域节点

研究（Research）：研究领域与专家之间的关系，表明哪些专家在该研究领域内开展工作。

相关内容（Related Content）：研究领域与文献之间的关系，表明哪些文献涉及该研究领域的内容。

# 数据来源与预处理

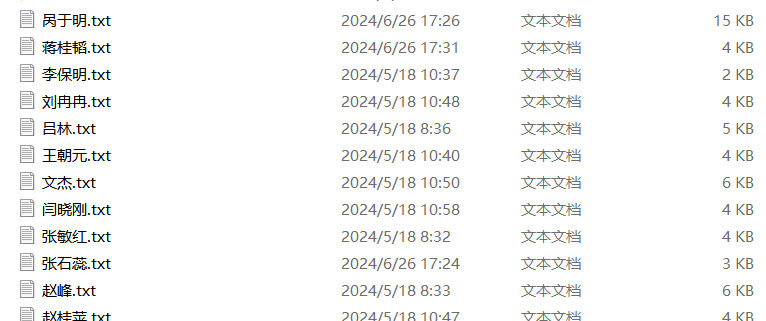
我们对中国农业科学院白羽肉鸡研究中心的团队核心成员进行了深入的学术背景和研究成果的调研。通过在线研究，收集了大量非结构化数据，通过GPT大模型进行实体和关系抽取，识别和提取关键信息，系统化地整理成表格形式，以便于后续的信息整合和知识发现。根据名字和肉鸡的研究主题，定向地从中国知网（CNKI）上爬取了相关的学术论文，内容包括论文标题，作者，期刊名，发表时间论文类型（期刊，会议），引用次数，下载次数等。

## 数据来源

我们的数据来源于中国农科院白羽肉鸡研究中心的团队核心成员，包括李保明、刘冉冉、吕林、王朝元、张敏红、文杰、赵峰，以及湖南农业大学教授张石蕊、湖南畜牧兽医研究所研究员蒋桂韬、中国农业大学教授呙于明。



我们查找了7位专家，保存到专家.txt文档中，用于后续的实体和关系抽取。

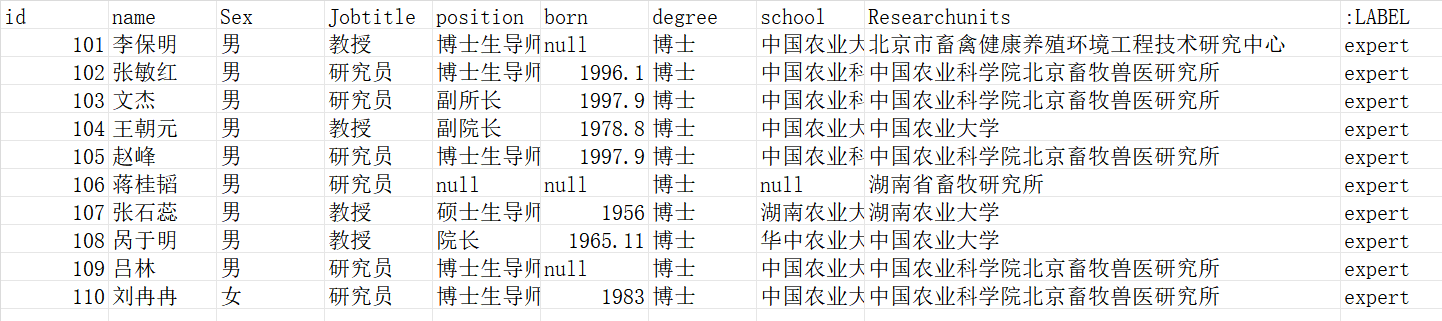


接着我们通过知识爬取知网，谷歌上该专家在肉鸡领域上的研究论文，内容包括论文标题，作者，期刊名，发表时间论文类型（期刊，会议），引用次数，下载次数等。



图 3 知网爬取

## 数据类型

数据收集覆盖了多个维度，确保了知识图谱的丰富性和深度。主要数据来源包括：

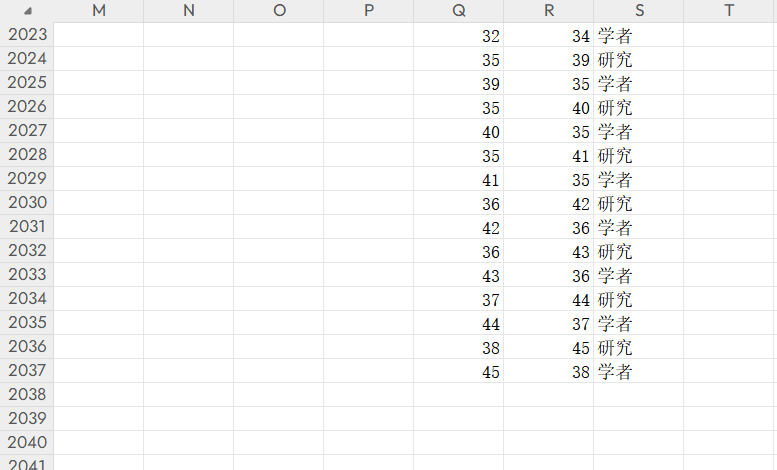
农业专家信息 (expert.csv)：包含肉鸡领域权威专家的基本信息，如姓名、职称、学位、工作单位、性别等。

研究论文 (paper.csv)：记录了专家发表的论文详情，涉及论文标题、摘要、关键词、发表年份、引用次数等。

期刊信息 (periodical.csv)：提供了期刊的名称、影响因子、出版周期、研究领域等属性信息。

## 数据清洗

数据清洗是确保数据质量的关键步骤。主要包括去除缺失值、处理重复数据、标准化数据格式等。例如，将不同数据集中的日期格式统一，确保每个专家和论文都有唯一的标识符。为了适应Neo4j图数据库的需求，所有数据都需要转换成适当的格式。主要包括将CSV文件转换为Neo4j可识别的格式，并对数据进行规范化处理，确保数据的一致性和准确性，最终整理的得到2038条三元组数据。



# Neo4j知识图谱构建

## CSV文件导入介绍

Neo4j 是一种流行的图形数据库，它支持通过 CSV 文件导入数据，这是一种快速且灵活的方式来初始化或扩展数据库中的数据。CSV 文件导入可以通过两种主要方式完成：使用命令行工具 `neo4j-import` 或者在 Neo4j 的 Cypher 查询语言中使用 `LOAD CSV` 语句。

使用 `neo4j-import` 命令行工具是一种批量导入方法，适用于在 Neo4j 服务关闭状态下导入大量数据。此工具能够直接生成一个包含 CSV 数据的新数据库。在开始之前，需要确保 CSV 文件遵循特定的格式，字段通常以逗号分隔，并且文件必须使用 UTF-8 编码。导入过程中，节点和关系的 ID 必须是唯一的，以确保数据的准确性。另一种方法是使用 Cypher 语言中的 `LOAD CSV` 语句，这允许在数据库运行时动态地导入数据。`LOAD CSV` 可以指定文件的 URL，并在查询中使用数据。这种方法适合增量更新或导入较小的数据集。在导入 CSV 文件时，需要注意的是，CSV 文件的第一行应该是列标题，这有助于 Neo4j 识别字段名称和数据类型。此外，对于关系数据，CSV 文件应包含起始节点 ID、结束节点 ID 和关系类型。导入完成后，可能需要根据实际需要创建索引或约束，以优化查询性能和数据完整性

我们的数据较少，所以使用的是LOAD CSV语句方法，模板如下：

1. // 加载CSV文件，假设文件位于Neo4j的**import**目录下
2. LOAD CSV WITH HEADERS FROM "file:///import/数据.csv" AS row
3. // 合并节点，如果节点不存在则创建，并设置属性
4. MERGE (:NodeLabel {id: row.id})
5. // 设置节点属性
6. SET n += {property: row.propertyValue}
7. // 返回导入的节点数量
8. RETURN count(\*);

## 导入流程

1. 导入专家信息（expert.csv）

文件内容包含专家的基本信息，如姓名、性别、职称、工作单位、毕业院校等。在导入过程中，我们在Neo4j中创建了节点类型 Expert，每个节点代表一位专家，并为每个 Expert 节点添加了相应的属性，如 name（姓名）、sex（性别）、title（职称）、institution（工作单位）、alma\_mater（毕业院校）等。

1. LOAD CSV WITH HEADERS FROM 'file:///expert.csv' AS row
2. CREATE (:Expert {
3. id: row.id,
4. name: row.name,
5. sex: row.sex,
6. title: row.title,
7. institution: row.institution,
8. alma\_mater: row.alma\_mater,
9. degree: row.degree,
10. job\_title: row.jobtitle,
11. born: row.born
12. });

### 导入研究论文信息（paper.csv）

文件内容包含每篇研究论文的信息，如标题、发表时间、关键词、摘要、作者等。在导入过程中，我们在Neo4j中创建了节点类型 Paper，每个节点代表一篇研究论文，并为每个 Paper 节点添加了相应的属性，如 title（标题）、publication\_date（发表时间）、keywords（关键词）、abstract（摘要）等。此外，我们还创建了 Authored 关系，连接 Expert 节点和 Paper 节点，表示某专家是某篇论文的作者。

1. LOAD CSV WITH HEADERS FROM 'file:///paper.csv' AS row
2. CREATE (:Paper {
3. id: row.id,
4. title: row.title,
5. publication\_date: row.publication\_date,
6. keywords: row.keywords,
7. abstract: row.abstract,
8. topic: row.topic,
9. identifier: row.identifier,
10. content: row.content
11. });

### 导入期刊信息（periodical.csv）

文件内容包含每个期刊的信息，如期刊名称、影响因子等。在导入过程中，我们在Neo4j中创建了节点类型 Periodical，每个节点代表一个期刊，并为每个 Periodical 节点添加了相应的属性，如 name（期刊名称）、impact\_factor（影响因子）等。此外，我们还创建了 Published\_in 关系，连接 Paper 节点和 Periodical 节点，表示某篇论文发表在某个期刊上。

1. LOAD CSV WITH HEADERS FROM 'file:///periodical.csv' AS row
2. CREATE (:Periodical {
3. id: row.id,
4. name: row.name,
5. impact\_factor: row.impact\_factor
6. });

### 导入研究主题信息（theme1.csv 和 theme2.csv）

文件内容包含研究主题的信息，如主题名称、相关领域等。在导入过程中，我们在Neo4j中创建了节点类型 Theme，每个节点代表一个研究主题，并为每个 Theme 节点添加了相应的属性，如 name（主题名称）。此外，我们还创建了 Related\_to 关系，连接 Paper 节点和 Theme 节点，表示某篇论文涉及某个研究主题。

1. LOAD CSV WITH HEADERS FROM 'file:///theme1.csv' AS row
2. CREATE (:Theme {
3. id: row.id,
4. name: row.name
5. });
7. LOAD CSV WITH HEADERS FROM 'file:///theme2.csv' AS row
8. CREATE (:Theme {
9. id: row.id,
10. name: row.name
11. });

### 导入研究单位信息（unit.csv）

文件内容包含研究单位的信息，如单位名称、地址等。在导入过程中，我们在Neo4j中创建了节点类型 Research\_Unit，每个节点代表一个研究单位，并为每个 Research\_Unit 节点添加了相应的属性，如 name（单位名称）、address（地址）。此外，我们还创建了 Affiliated\_with 关系，连接 Expert 节点和 Research\_Unit 节点，表示某专家隶属于某个研究单位。

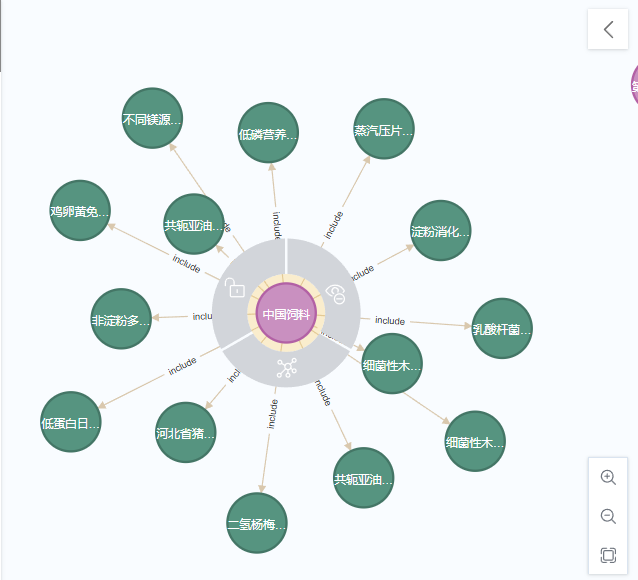
1. LOAD CSV WITH HEADERS FROM 'file:///unit.csv' AS row
2. CREATE (:Research\_Unit {
3. id: row.id,
4. name: row.name,
5. address: row.address
6. });

### 导入发表记录信息（发表.csv）

1. LOAD CSV WITH HEADERS FROM 'file:///发表.csv' AS row
2. MATCH (p:Paper {id: row.paper\_id}), (u:Research\_Unit {id: row.unit\_id})
3. CREATE (p)-[:PUBLISHED\_BY]->(u);

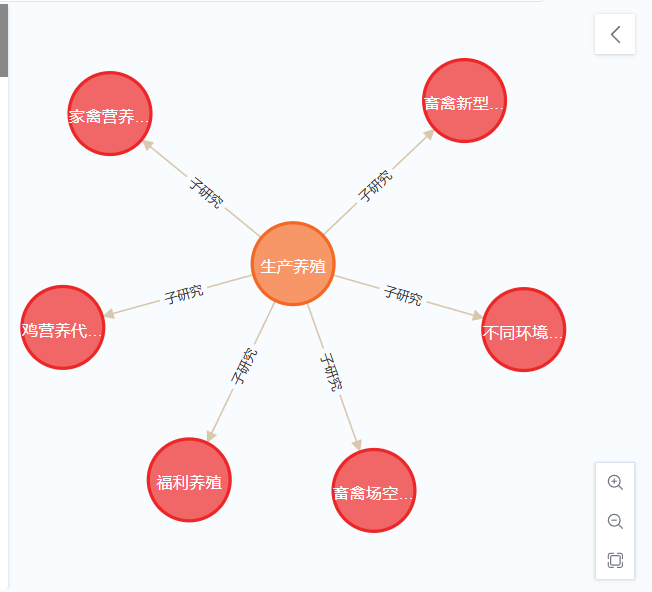
### 创建论文与期刊之间的关系（Published in）

1. LOAD CSV WITH HEADERS FROM 'file:///paper.csv' AS row
2. MATCH (p:Paper {id: row.id}), (j:Periodical {id: row.periodical\_id})
3. CREATE (p)-[:PUBLISHED\_IN]->(j);



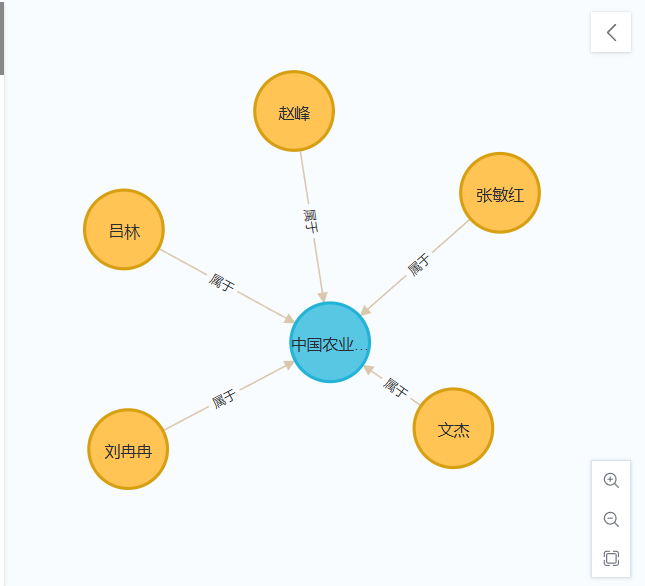
### 创建论文与研究主题之间的关系（Related to）

1. LOAD CSV WITH HEADERS FROM 'file:///paper.csv' AS row
2. MATCH (p:Paper {id: row.id}), (t:Theme {id: row.topic\_id})
3. CREATE (p)-[:RELATED\_TO]->(t);



### 创建专家与研究单位之间的关系（Affiliated with）

1. LOAD CSV WITH HEADERS FROM 'file:///expert.csv' AS row
2. MATCH (e:Expert {id: row.id}), (u:Research\_Unit {id: row.unit\_id})
3. CREATE (e)-[:AFFILIATED\_WITH]->(u);



# Neo4j专家关系网络分析

通过分析专家之间的合作关系，可以揭示研究团队和合作网络，帮助发现领域内的重要合作伙伴和研究群体。在知识图谱中，每个专家节点通过其撰写和合作的研究论文与其他专家节点相连。通过这种方式，可以识别出哪些专家经常合作，以及哪些专家在特定研究领域内具有较高的合作频率。这种分析不仅可以帮助我们理解当前的研究合作模式，还可以发现潜在的合作机会和新兴的研究群体。例如，通过分析专家之间的共同作者关系，可以发现哪些专家在某个研究主题上具有高度合作性，从而识别出核心研究团队和关键合作伙伴。

根据研究主题分布，可以了解当前的研究热点和趋势，识别未来的研究方向。知识图谱中，每篇论文节点都与一个或多个研究主题节点相连，通过这些连接关系，我们可以统计和分析不同研究主题的论文数量及其分布情况。这种分析可以揭示出哪些研究主题在某个时间段内受到更多关注，以及这些主题的发展趋势。例如，通过分析过去五年内的研究主题分布，可以发现哪些主题正在上升成为新的研究热点，哪些主题的关注度在下降。这样，研究人员和机构可以基于这些信息调整他们的研究方向和策略。

期刊的影响因子和论文发表情况，可以评估期刊在特定领域的影响力，帮助研究人员选择合适的投稿期刊。在知识图谱中，每篇论文节点与其发表的期刊节点相连，期刊节点包含影响因子等属性信息。通过统计和分析发表在不同期刊上的论文数量及其引用情况，可以评估各期刊在特定研究领域的影响力。例如，高影响因子的期刊通常发表了较多被引用次数较高的论文，通过这些分析，研究人员可以更好地选择合适的期刊进行投稿，从而提高他们的研究成果的影响力。

专家关系网络分析：通过分析专家之间的合作关系，可以揭示研究团队和合作网络，帮助发现领域内的重要合作伙伴和研究群体。

# 问答系统

## 1. 建立与Neo4j的连接

系统首先需要与Neo4j数据库建立连接，以便后续执行查询。通过py2neo.Graph类实现这一连接，具体代码如下：

1. **from** py2neo **import** Graph
2. graph = Graph("http://localhost:7474", auth=('neo4j', '123456'))

## 定义用户意图模式和响应模板

系统需要识别用户查询的意图，因此预定义了多种意图模式。这些模式包括询问专家所属学校、研究单位、职称、学位、职位、研究方向以及专家间的关系等。

意图模式定义示例：

1. # 定义用户意图模式和响应模板
2. school = [
3. '这位专家是哪个学校的研究人员？',
4. '这位专家是哪个学校的成员？',
5. '这位专家属于哪个学校？',
6. '请问这位专家是哪个学校的？',
7. '这位专家在哪个学校工作？'
8. ]
10. Researchunits = [
11. '这位专家是哪个单位的研究人员？',
12. '这位专家是哪个研究单位的成员？',
13. '这位专家属于哪个机构？',
14. '这位专家在哪个单位工作？',
15. '这位专家的研究单位是什么？'
16. ]
18. Jobtitle = [
19. '这位专家的职称有哪些？',
20. '这位专家的专业职称有哪些？',
21. '请问这位专家的职称是什么？',
22. '这位专家的学术头衔是什么？',
23. '这位专家的工作职位是什么？'
24. ]
26. degree = [
27. '这位专家的学位是什么？',
28. '这位专家的学历背景是什么？',
29. '这位专家拥有怎样的学位？',
30. '这位专家的最高学历是什么？',
31. '请问这位专家的学术学位是什么？'
32. ]
34. position = [
35. '这位专家的职位是什么？',
36. '请问这位专家的工作职位是什么？',
37. '这位专家担任什么职位？',
38. '这位专家的工作岗位是什么？',
39. '这位专家的职务是什么？'
40. ]
42. research = [
43. '这个专家的研究方向是什么',
44. '这个专家的研究方向是什么？',
45. '这位专家主要研究什么领域？',
46. '这位专家的研究领域有哪些？',
47. '这位专家关注的研究主题是什么？',
48. '这位专家的主要研究课题是什么？',
49. '这位专家在什么方面开展研究？',
50. '这位专家的研究重点是什么',
51. '这位专家致力于哪方面的研究？',
52. '这位专家专注于哪类研究？',
53. '这位专家的科研工作涉及哪些方面？'
54. ]
56. relationship = [
57. '这两位专家之间的关系是什么？',
58. '请问这两位专家合作过哪些项目？',
59. '这两位专家共同发表过哪些论文？',
60. '这两位专家之间的合作是什么？',
61. '请问这两位专家之间有何联系？'
62. ]
64. email = [
65. '这位专家的邮箱是什么？',
66. '请问这位专家的电子邮件地址是？',
67. '这位专家的电子邮箱是什么？',
68. '这位专家的联系邮箱是什么？',
69. '如何通过电子邮件联系这位专家？'
70. ]
72. phone = [
73. '这位专家的电话是什么？',
74. '请问这位专家的联系电话是什么？',
75. '这位专家的电话号码是多少？',
76. '这位专家的手机号码是多少？',
77. '如何通过电话联系这位专家？'
78. ]
80. address = [
81. '这位专家的地址是什么？',
82. '请问这位专家的工作地址是什么？',
83. '这位专家的办公地址是什么？',
84. '这位专家的联系地址是什么？',
85. '这位专家所在的地点是哪里？'
86. ]
88. publication = [
89. '这位专家的主要出版物有哪些？',
90. '请问这位专家发表过哪些论文或书籍？',
91. '这位专家的代表作是什么？',
92. '这位专家的学术出版物有哪些？',
93. '这位专家有哪些重要的出版物？'
94. ]
96. project = [
97. '这位专家参与了哪些项目？',
98. '请问这位专家负责或参与了哪些研究项目？',
99. '这位专家有哪些正在进行的项目？',
100. '这位专家过去参与了哪些项目？',
101. '这位专家的研究项目有哪些？'
102. ]
104. honor = [
105. '这位专家获得了哪些荣誉？',
106. '请问这位专家有哪些奖项或荣誉？',
107. '这位专家的学术荣誉有哪些？',
108. '这位专家获得过哪些表彰？',
109. '这位专家的荣誉和奖项有哪些？'
110. ]
112. teaching = [
113. '这位专家教授哪些课程？',
114. '请问这位专家的教学课程是什么？',
115. '这位专家在哪些课程中授课？',
116. '这位专家讲授什么内容？',
117. '这位专家的教学领域是什么？'
118. ]
120. experience = [
121. '这位专家的工作经历是什么？',
122. '请问这位专家的职业经历有哪些？',
123. '这位专家过去的工作经历是什么？',
124. '这位专家的学术经历是什么？',
125. '这位专家的职业背景是什么？'
126. ]

响应模板定义示例：

1. schoolResponse = '{}这位专家所在的学校是{}'
2. ResearchunitsResponse = '{}这位专家的研究单位是{}'
3. JobtitleResponse = '{}这位专家的职称是{}'
4. degreeResponse = '{}这位专家的学位是{}'
5. positionResponse = '{}这位专家的职位是{}'
6. researchResponse = '{}这位专家的研究方向是{}'
7. relationshipResponse = '{} 和 {} 之间的关系是: {}'

## 用户意图识别

通过模糊匹配技术，系统可以将用户的自然语言查询与预定义的意图模式进行匹配，识别出最接近的意图。

1. **from** fuzzywuzzy **import** fuzz
3. **def** AssignIntension(text):
4. stencilDegree = {}
5. **for** key, value **in** stencil.items():
6. score = 0
7. **for** item **in** value:
8. degree = fuzz.partial\_ratio(text, item)
9. score += degree
10. stencilDegree[key] = score / len(value)
11. **return** stencilDegree

## 提取专家名称

使用jieba分词库对用户查询进行分词，并识别出专家名称。系统加载了自定义的专家词典，以提高识别精度。

专家名称提取函数：

1. **import** jieba.posseg as pseg
2. **import** jieba
4. **def** getExpertName(text):
5. expertName = ''
6. jieba.load\_userdict('./expert.txt')
7. words = pseg.cut(text)
8. **for** w **in** words:
9. **if** w.flag == 'zj':
10. expertName = w.word
11. **return** expertName

## 数据库查询与响应生成

根据识别出的意图和专家名称，系统构建相应的Cypher查询语句，从Neo4j数据库中提取数据。根据查询结果，生成自然语言响应。

查询与响应生成函数：

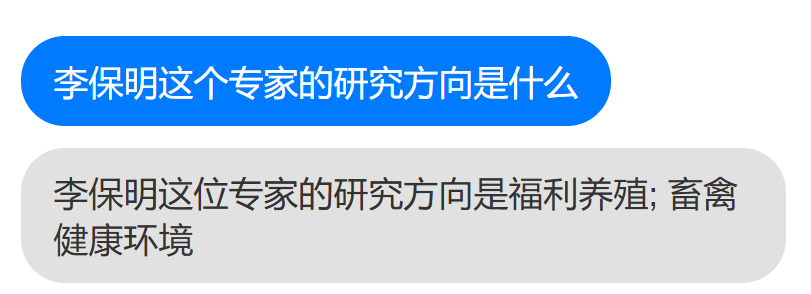
**def** SearchGraph(expertName, stencilDict={}, targetExpert=None):

1. classification = max(stencilDict, key=stencilDict.get)
2. **if** classification == 'research':
3. cypher = f'''''
4. result = graph.run(cypher).data()
6. **if** **not** result:
7. **return** classification, "No data available"
9. fields = [record['field'] **for** record **in** result]
10. data = '; '.join(fields)
11. **elif** targetExpert:
12. cypher = f'''''
13. result = graph.run(cypher).data()
15. **if** **not** result:
16. **return** classification, "No data available"
18. relations = [record['relation'] **for** record **in** result]
19. data = '; '.join(relations)
20. **else**:
21. cypher = f'MATCH (n:Expert) WHERE n.name = "{expertName}" RETURN n.{classification} AS result'
22. result = graph.run(cypher).data()
24. **if** **not** result:
25. **return** classification, "No data available"
27. data = result[0].get('result', 'No data available')
29. **return** classification, data

系统的主要视图函数views.py用于处理用户的HTTP GET请求，解析查询文本，调用上述函数进行意图识别、数据库查询和响应生成，并返回JSON格式的响应。

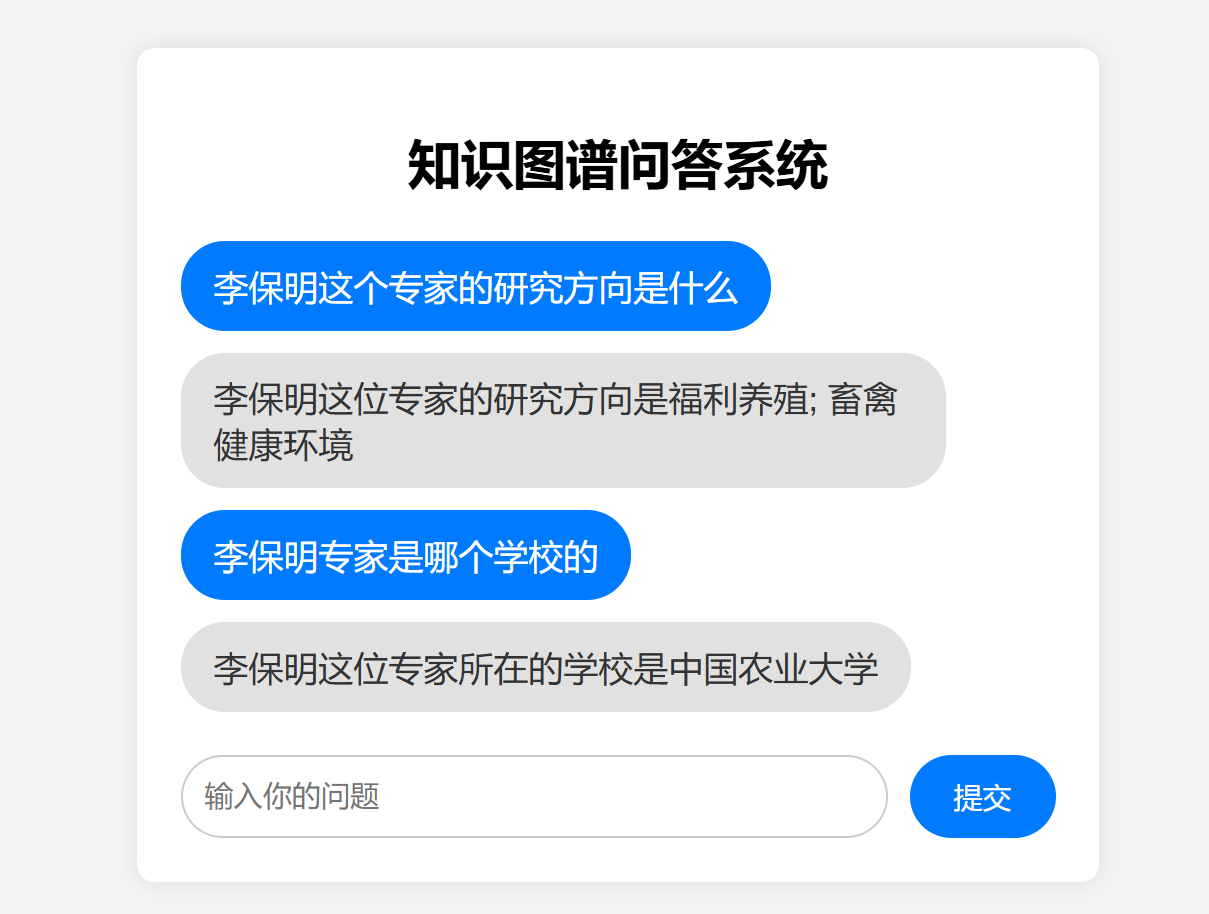
系统运行时，用户通过Web页面输入查询内容，系统通过上述流程进行处理，并在页面上显示结果。以下是一个完整的查询示例：

用户输入查询："李保明这位专家的研究方向是什么？"，系统识别出意图为"研究方向"，提取出专家名称并查询数据库，最终返回专家的研究方向信息。



# 前后端页面搭建

我们使用python的django后端框架以及HTML的前端，采用前后端不分离的模式搭建了一个简易的问答系统。

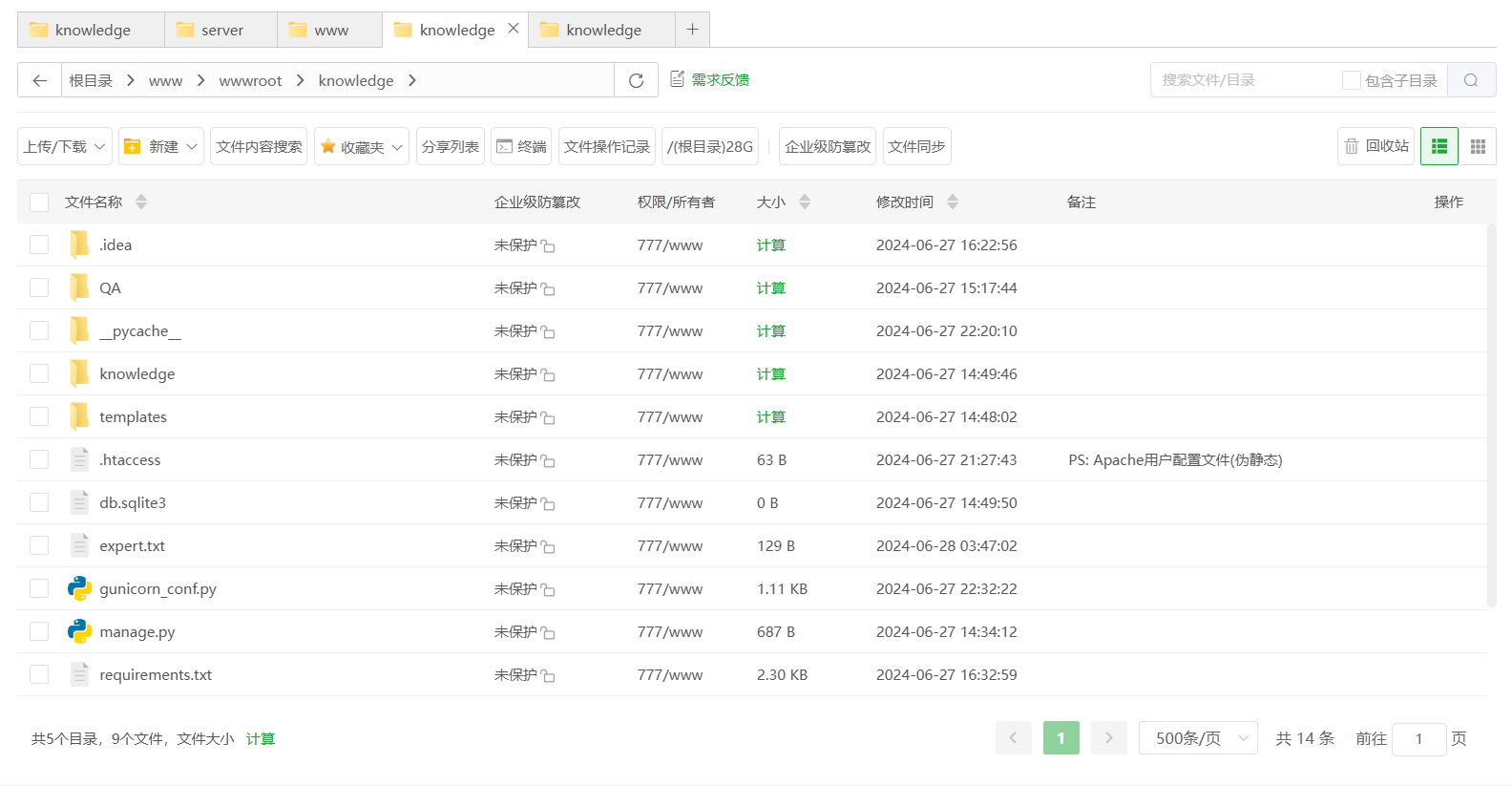


并将前后端以及Neo4j部署到腾讯云服务器中实现了远程调用。

## 设置防火墙端口



## 部署后端

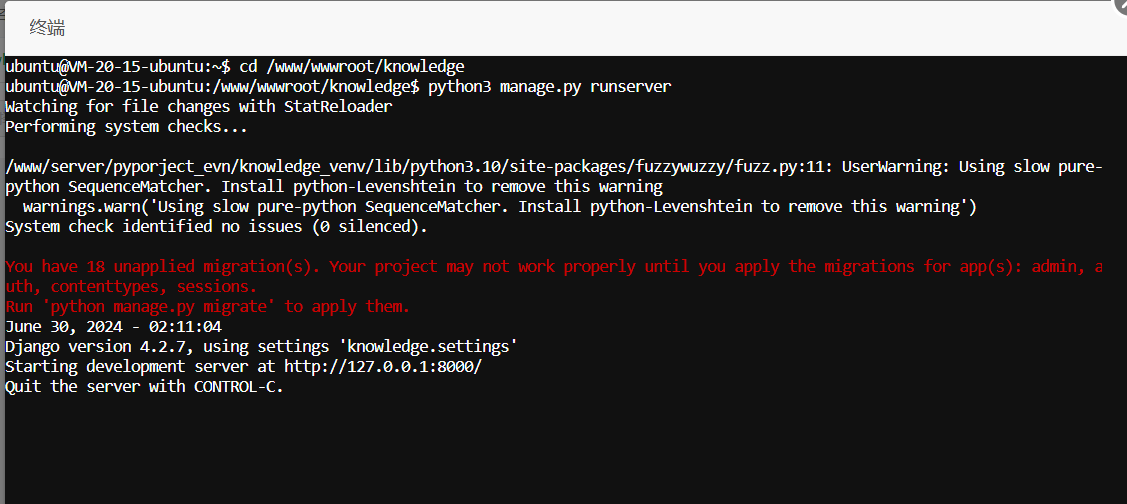




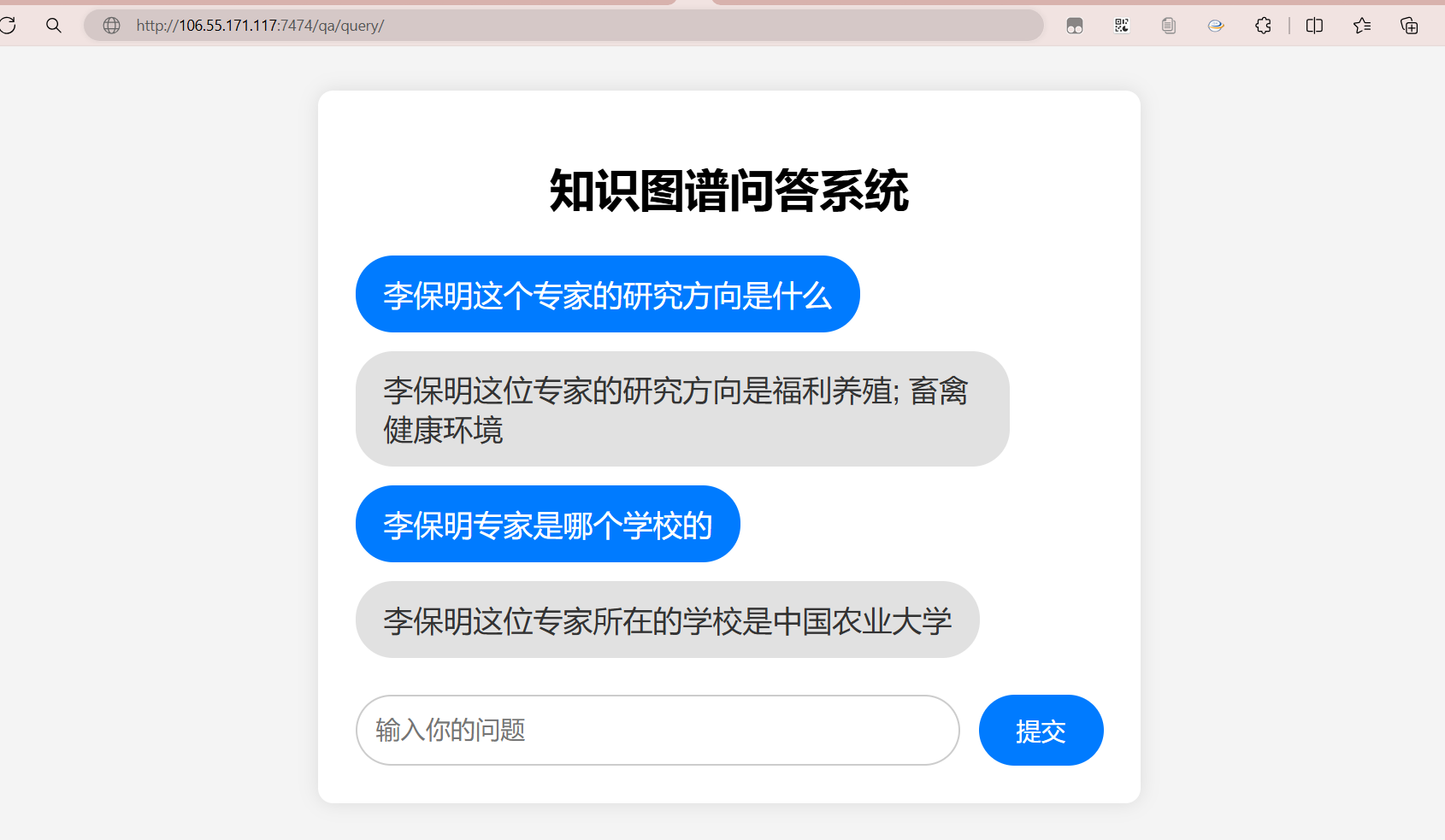
## 部署neo4j

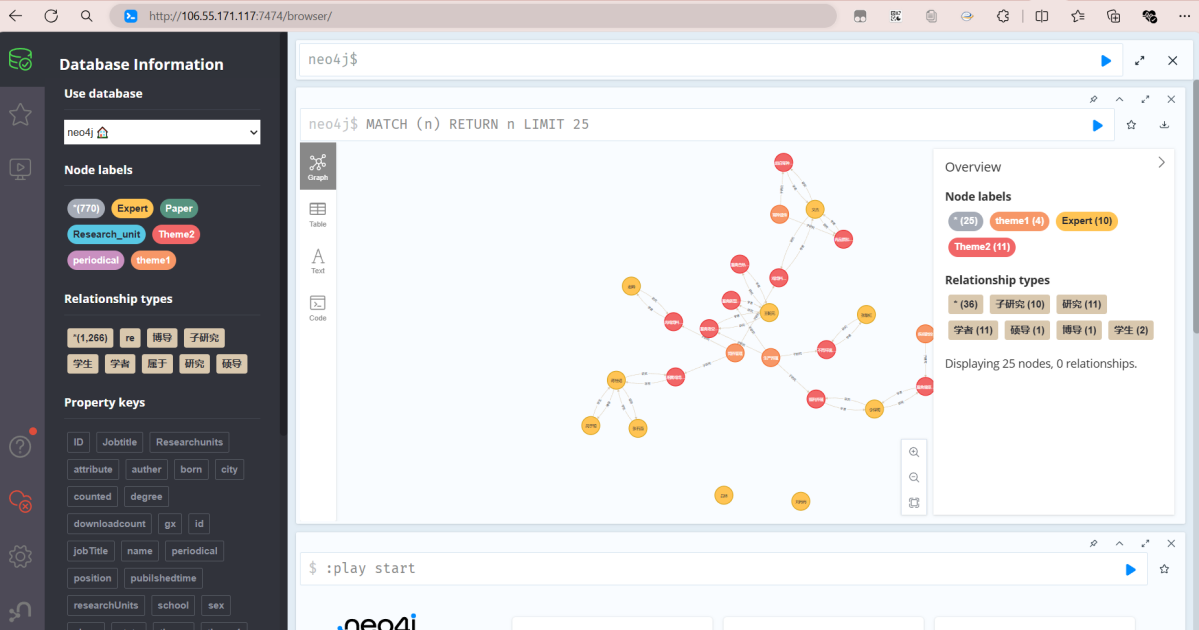


## 调试



## 页面展示





# 附录

前端代码query.html

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8">
5. <title>知识图谱问答系统</title>
6. <style>
7. body {
8. font-family: Arial, sans-serif;
9. background-color: #f4f4f4;
10. margin: 0;
11. padding: 0;
12. display: flex;
13. justify-content: center;
14. align-items: center;
15. height: 100vh;
16. }
17. .container {
18. background-color: #fff;
19. padding: 20px;
20. border-radius: 8px;
21. box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
22. width: 400px;
23. text-align: center;
24. }
25. h1 {
26. font-size: 24px;
27. margin-bottom: 20px;
28. }
29. form {
30. margin-bottom: 20px;
31. }
32. input[type="text"] {
33. width: calc(100% - 22px);
34. padding: 10px;
35. margin-bottom: 10px;
36. border: 1px solid #ccc;
37. border-radius: 4px;
38. }
39. button {
40. padding: 10px 20px;
41. border: none;
42. background-color: #007BFF;
43. color: #fff;
44. border-radius: 4px;
45. cursor: pointer;
46. }
47. button:hover {
48. background-color: #0056b3;
49. }
50. #response {
51. font-size: 16px;
52. color: #333;
53. }
54. </style>
55. </head>
56. <body>
57. <div **class**="container">
58. <h1>知识图谱问答系统</h1>
59. <form id="query-form">
60. <input type="text" id="query-input" placeholder="输入你的问题">
61. <button type="submit">提交</button>
62. </form>
63. <p id="response"></p>
64. </div>
66. <script>
67. document.getElementById('query-form').addEventListener('submit', function(event) {
68. event.preventDefault();
69. const query = document.getElementById('query-input').value;
70. fetch(`/qa/query/?query=${query}`)
71. .then(response => response.json())
72. .then(data => {
73. document.getElementById('response').innerText = data.response;
74. })
75. .catch(error => console.error('Error:', error));
76. });
77. </script>
78. </body>
79. </html>

问答

1. **from** django.shortcuts **import** render
2. **from** django.http **import** JsonResponse
3. **from** py2neo **import** Graph
4. **import** jieba.posseg as pseg
5. **import** jieba
6. **from** fuzzywuzzy **import** fuzz
7. **import** os
9. # 建立与Neo4j的连接
10. graph = Graph("http://localhost:7474", auth=('neo4j', '123456'))
12. # 定义用户意图模式和响应模板
13. school = [
14. '这位专家是哪个学校的研究人员？',
15. '这位专家是哪个学校的成员？',
16. '这位专家属于哪个学校？',
17. '请问这位专家是哪个学校的？',
18. '这位专家在哪个学校工作？'
19. ]
21. Researchunits = [
22. '这位专家是哪个单位的研究人员？',
23. '这位专家是哪个研究单位的成员？',
24. '这位专家属于哪个机构？',
25. '这位专家在哪个单位工作？',
26. '这位专家的研究单位是什么？'
27. ]
29. Jobtitle = [
30. '这位专家的职称有哪些？',
31. '这位专家的专业职称有哪些？',
32. '请问这位专家的职称是什么？',
33. '这位专家的学术头衔是什么？',
34. '这位专家的工作职位是什么？'
35. ]
37. degree = [
38. '这位专家的学位是什么？',
39. '这位专家的学历背景是什么？',
40. '这位专家拥有怎样的学位？',
41. '这位专家的最高学历是什么？',
42. '请问这位专家的学术学位是什么？'
43. ]
45. position = [
46. '这位专家的职位是什么？',
47. '请问这位专家的工作职位是什么？',
48. '这位专家担任什么职位？',
49. '这位专家的工作岗位是什么？',
50. '这位专家的职务是什么？'
51. ]
53. research = [
54. '这个专家的研究方向是什么',
55. '这个专家的研究方向是什么？',
56. '这位专家主要研究什么领域？',
57. '这位专家的研究领域有哪些？',
58. '这位专家关注的研究主题是什么？',
59. '这位专家的主要研究课题是什么？',
60. '这位专家在什么方面开展研究？',
61. '这位专家的研究重点是什么',
62. '这位专家致力于哪方面的研究？',
63. '这位专家专注于哪类研究？',
64. '这位专家的科研工作涉及哪些方面？'
65. ]
67. relationship = [
68. '这两位专家之间的关系是什么？',
69. '请问这两位专家合作过哪些项目？',
70. '这两位专家共同发表过哪些论文？',
71. '这两位专家之间的合作是什么？',
72. '请问这两位专家之间有何联系？'
73. ]
75. # 响应模板
76. schoolResponse = '{}这位专家所在的学校是{}'
77. ResearchunitsResponse = '{}这位专家的研究单位是{}'
78. JobtitleResponse = '{}这位专家的职称是{}'
79. degreeResponse = '{}这位专家的学位是{}'
80. positionResponse = '{}这位专家的职位是{}'
81. researchResponse = '{}这位专家的研究方向是{}'
82. relationshipResponse = '{} 和 {} 之间的关系是: {}'
84. # 用户意图模式字典
85. stencil = {
86. 'school': school,
87. 'Researchunits': Researchunits,
88. 'Jobtitle': Jobtitle,
89. 'degree': degree,
90. 'position': position,
91. 'research': research,
92. 'relationship': relationship
93. }
95. # 响应模板字典
96. responseDict = {
97. 'schoolResponse': schoolResponse,
98. 'ResearchunitsResponse': ResearchunitsResponse,
99. 'JobtitleResponse': JobtitleResponse,
100. 'degreeResponse': degreeResponse,
101. 'positionResponse': positionResponse,
102. 'researchResponse': researchResponse,
103. 'relationshipResponse': relationshipResponse
104. }

107. **def** AssignIntension(text):
108. stencilDegree = {}
109. **for** key, value **in** stencil.items():
110. score = 0
111. **for** item **in** value:
112. degree = fuzz.partial\_ratio(text, item)
113. score += degree
114. stencilDegree[key] = score / len(value)
115. **return** stencilDegree

118. **def** getExpertName(text):
119. expertName = ''
120. jieba.load\_userdict('./expert.txt')  # 加载自定义词典
121. words = pseg.cut(text)
122. **for** w **in** words:
123. **if** w.flag == 'zj':
124. expertName = w.word
125. **return** expertName

128. **def** SearchGraph(expertName, stencilDict={}, targetExpert=None):
129. classification = max(stencilDict, key=stencilDict.get)
130. **if** classification == 'research':
131. cypher = f'''''
132. MATCH (e:Expert {{name: "{expertName}"}})-[:研究]->(r:Theme2)
133. RETURN r.theme AS field
134. '''
135. result = graph.run(cypher).data()
137. **if** **not** result:
138. **return** classification, "No data available"
140. fields = [record['field'] **for** record **in** result]
141. data = '; '.join(fields)
142. **elif** targetExpert:
143. cypher = f'''''
144. MATCH (n:Expert)-[r]->(m:Expert)
145. WHERE n.name = "{expertName}" AND m.name = "{targetExpert}"
146. RETURN type(r) AS relation
147. UNION
148. MATCH (n:Expert)<-[r]-(m:Expert)
149. WHERE n.name = "{expertName}" AND m.name = "{targetExpert}"
150. RETURN type(r) AS relation
151. '''
152. result = graph.run(cypher).data()
154. **if** **not** result:
155. **return** classification, "No data available"
157. relations = [record['relation'] **for** record **in** result]
158. data = '; '.join(relations)
159. **else**:
160. cypher = f'MATCH (n:Expert) WHERE n.name = "{expertName}" RETURN n.{classification} AS result'
161. result = graph.run(cypher).data()
163. **if** **not** result:
164. **return** classification, "No data available"
166. data = result[0].get('result', 'No data available')
168. **return** classification, data

171. **def** respondQuery(expertName, classification, item):
172. query = classification + 'Response'
173. response = responseDict.get(query, '{}该问题暂无答案')
174. **return** response.format(expertName, item)

177. **def** respondRelationshipQuery(expertName, targetExpert, relations):
178. response = responseDict.get('relationshipResponse', '{} 和 {} 之间没有关系数据')
179. **return** response.format(expertName, targetExpert, relations)

182. **def** query(request):
183. **if** request.method == 'GET':
184. queryText = request.GET.get('query')
185. **if** queryText **is** None:
186. **return** render(request, 'qa/query.html')
188. experts = queryText.split('和')  # 简单处理多个专家名字
189. **if** len(experts) == 2:
190. expertName = getExpertName(experts[0])
191. targetExpert = getExpertName(experts[1])
192. dict = AssignIntension(queryText)
193. classification, result = SearchGraph(expertName, dict, targetExpert=targetExpert)
194. response = respondRelationshipQuery(expertName, targetExpert, result)
195. **else**:
196. expertName = getExpertName(queryText)
197. dict = AssignIntension(queryText)
198. classification, result = SearchGraph(expertName, dict)
199. response = respondQuery(expertName, classification, result)
201. **return** JsonResponse({'response': response})
202. **return** render(request, 'qa/query.html')