

基于图数据库和自然语言实现的甜菜论文知识图谱

摘要

本文介绍了一个基于图数据库和自然语言处理技术的甜菜论文知识图谱系统的设计与实现。系统采用Spring Boot、Vue.js、Neo4j等技术栈，实现了前后端分离的架构设计。后端使用Spring Boot框架实现业务逻辑、数据存储和接口设计，前端基于Vue 3构建用户界面，提供了信息展示、智能化操作等功能。通过整合自然语言处理（NLP）技术和图数据库Neo4j，系统能够有效地处理和展示甜菜相关论文的知识图谱。本文详细阐述了系统的关键技术，包括界面设计、图数据库处理和自然语言处理，并提供了系统的部署文档。最后，通过视频和图片展示了系统的实现效果。

关键词：neo4j 图数据库 自然语言 spring boot vue nlp

1. 引言

背景

随着大数据时代的到来，知识图谱作为一种有效的知识表示和管理方式，被广泛应用于各个领域。与此同时，信息化社会的快速发展带来了知识总量的爆炸式增长，特别是在学术研究领域，大量的论文和文献包含了丰富的知识。然而，这些知识分散在不同的数据源中，传统的信息检索方法难以满足复杂语义理解和多维度分析的需求。如何有效地组织和利用这些知识，成为一个亟待解决的重要问题。

因此，“甜菜”问答系统应运而生。通过结合图数据库与自然语言处理（NLP）技术，该系统以知识图谱的方式直观呈现学术论文中的关键知识点及其关联，并利用分词和语义分析技术实现智能问答功能。系统采用Spring Boot、Vue.js、Neo4j等现代技术栈，搭建了前后端分离的架构，为用户提供高效、智能的信息查询和展示平台。

意义

● 提升学术信息获取效率

构建“甜菜”论文知识图谱系统，可以帮助研究人员快速检索和理解特定领域的知识，降低信息筛选的时间成本，显著提高研究效率。

● 促进知识共享与传播

系统通过整合分散的学术资源，构建语义化的知识图谱，为科研工作者提供了一个知识共享的平台，推动知识在学术界的广泛传播与应用。

- **推动智能化技术应用**

借助自然语言处理与图数据库技术的结合，“甜菜”系统实现了复杂语义问题的自动解析与解答，为智能问答技术在学术研究中的应用开辟了新路径。

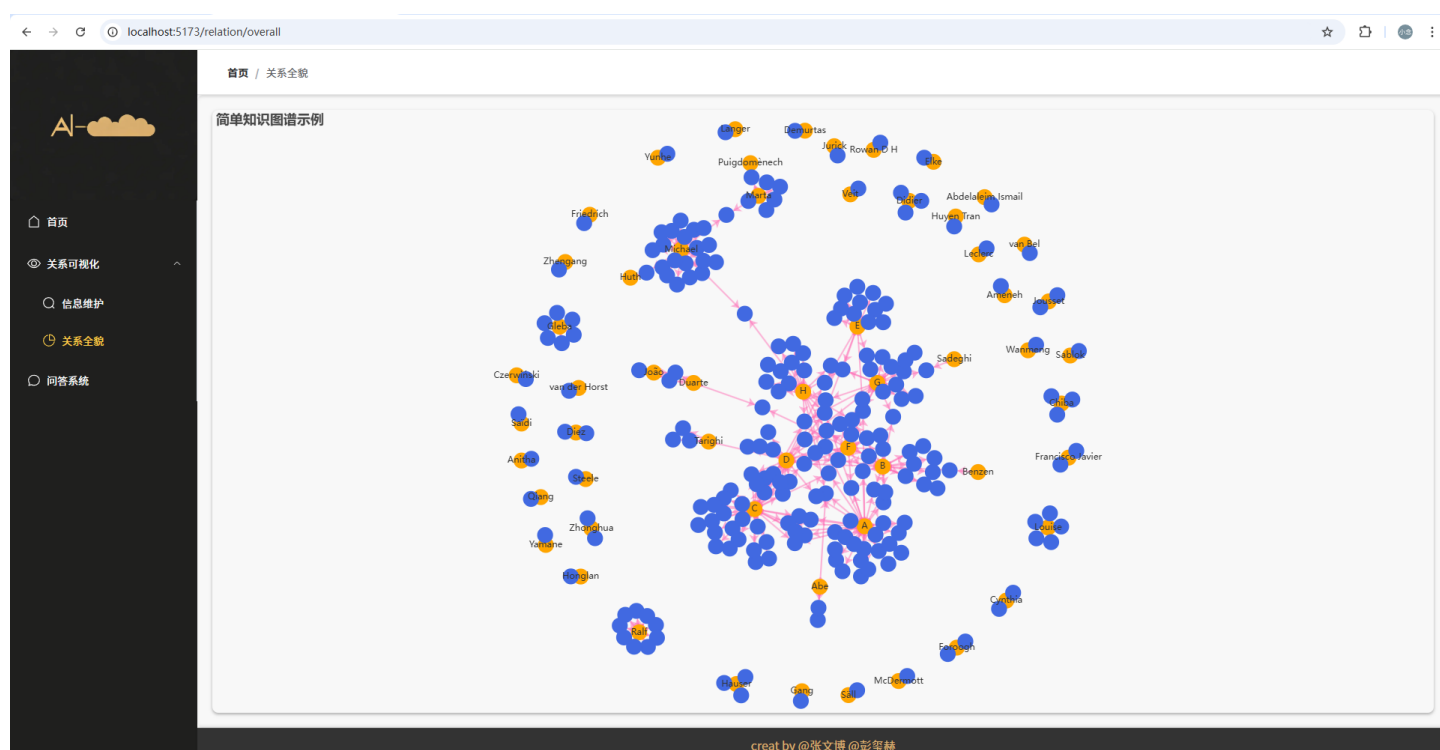
- **助力科研创新与决策**

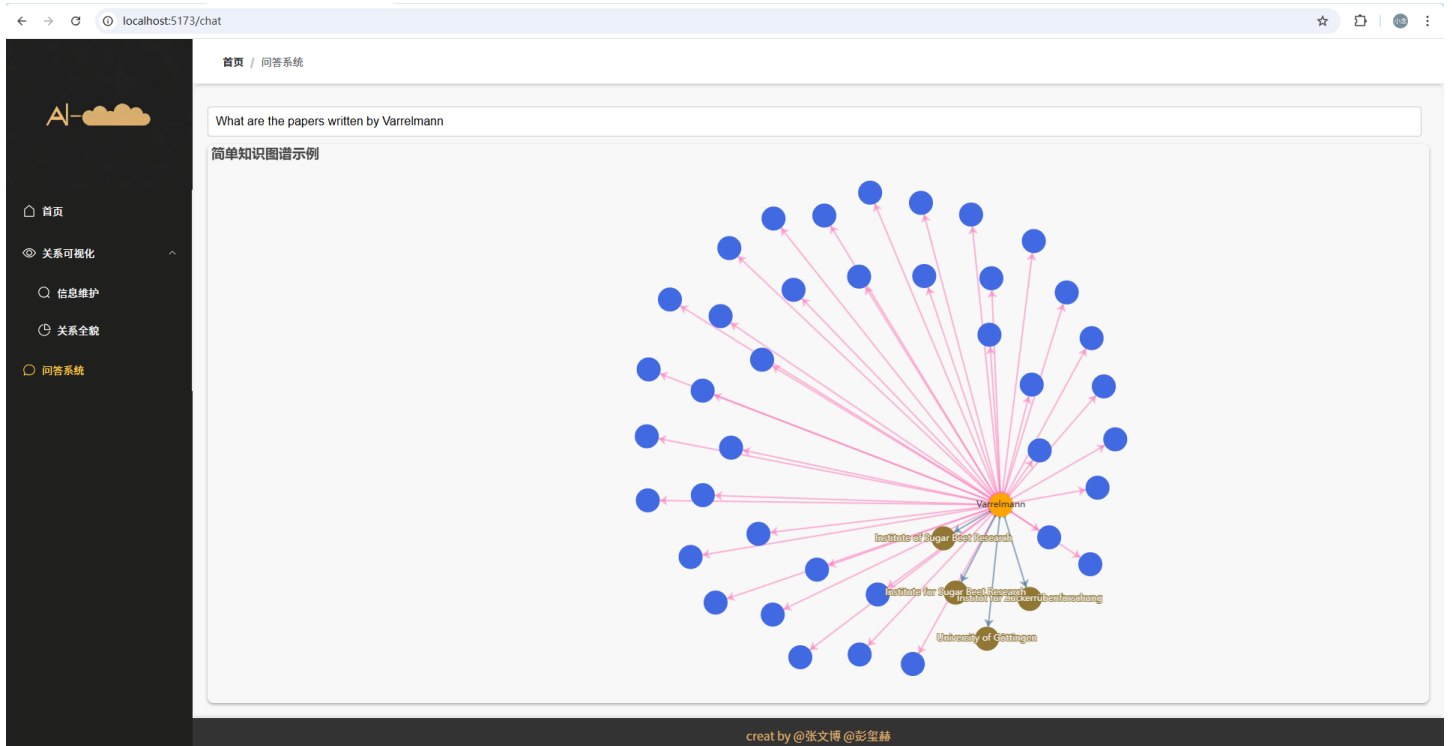
系统的知识图谱展示和智能问答功能能够帮助研究人员快速洞察研究热点和趋势，为科研创新提供启发，并支持更科学的决策。

- 为其他领域提供参考价值

“甜菜”问答系统的设计与实现方案具有一定的通用性,可作为其他领域知识管理和智能问答系统开发的重要参考,具有广泛的技术推广和应用价值。

2. 实现效果





3. 技术栈

- **Spring Boot**

Spring Boot，这一基于Spring框架的开源Java企业级应用开发平台，以其对配置和开发流程的简化而著称，极大地加速了应用程序的构建过程。正如《Spring Boot in Action》一书中所述，Spring Boot提供了丰富的自动化配置、嵌入式服务器（如Tomcat）以及与Spring生态系统的无缝集成，使开发者能够快速搭建高性能、可扩展的后端服务[1]。

- **Vue.js**

Vue.js，这一渐进式JavaScript框架，以其在构建用户界面方面的卓越表现而备受青睐。Vue.js的核心库专注于视图层，同时通过其丰富的生态系统支持复杂的单页应用（SPA）开发。正如Vue.js官方文档所强调的，Vue.js的轻量、灵活特点，以及其双向数据绑定和组件化设计，大大简化了前端开发工作。

- **自然语言处理（NLP）**

自然语言处理（NLP），作为计算机科学、人工智能和语言学的交叉领域，旨在使计算机能够理解、分析、生成和处理人类语言。正如Jurafsky和Martin在其著作《Speech and Language Processing》中所描述，通过分词、语义分析等核心技术，NLP在文本挖掘、机器翻译和智能问答等领域得到了广泛应用，为本系统的智能问答功能提供了技术支撑[2]。

- **Neo4j**

Neo4j，这一高性能的开源图数据库，以其在处理复杂的关系数据方面的卓越表现而备受推崇。正如Neo4j官方文档所强调，它以图结构存储数据，将数据表示为节点（Node）、关系（Relationship）和属性（Property），使其在关系查询和可视化分析方面具有显著优势。Neo4j的Cypher查询语言为开发者提供了简洁、高效的数据操作方式，非常适合构建知识图谱系统[3]。

4. 总体架构设计

系统架构是软件系统的核心组成部分，决定了系统的性能、可扩展性和可维护性。本系统采用前后端分离的架构设计，以提升系统的模块化程度和开发效率，实现前后端独立开发与部署。

前端架构基于 Vue.js 框架进行开发，采用组件化和模块化的设计思想，构建用户友好的交互界面。Vue.js 是一种用于构建用户界面的 JavaScript 框架，它基于标准的 HTML、CSS 和 JavaScript，并提供了一种声明性的、基于组件的编程模型，帮助开发者高效地开发任何复杂度的用户界面。通过 Vue Router 实现路由管理，支持多页面的动态加载；使用 Vuex 管理全局状态，确保数据在组件间的高效共享和一致性。前端通过 Axios 发送 HTTP 请求，与后端进行数据交互，同时提供实时动态的用户操作反馈。

后端架构基于 Spring Boot 框架开发，利用其内置的高效开发工具链和丰富的自动化配置功能，实现了快速搭建和开发。Spring Boot 是一个基于 Spring 框架的开源 Java 企业级应用开发平台，旨在通过简化配置和开发流程，加速应用程序的构建。它提供了丰富的自动化配置、嵌入式服务器（如 Tomcat）以及与 Spring 生态系统的无缝集成，使开发者能够快速搭建高性能、可扩展的后端服务。后端通过 RESTful 风格的接口设计，确保与前端的交互简单直观。

系统的数据存储使用 Neo4j 图数据库，通过图结构直观呈现复杂的节点与关系，适合知识图谱的构建与查询。后端通过与 Neo4j 的深度集成，使用 Cypher 查询语言高效地进行数据的读取与写入操作。Neo4j 是一个基于图结构存储数据的数据库，它以图结构存储数据，将数据表示为节点（Node）、关系（Relationship）和属性（Property），使其在关系查询和可视化分析方面具有显著优势。

系统引入了自然语言处理（NLP）技术，通过分词和语义分析等技术，对用户输入的问题进行解析，并结合知识图谱的数据进行匹配，从而实现智能问答功能。自然语言处理是计算机科学、人工智能和语言学的交叉领域，旨在使计算机能够理解、解释和生成人类语言。

通过前后端分离架构设计，系统实现了高效的开发流程和良好的可扩展性，同时利用现代化技术栈提供了流畅的用户体验与强大的数据处理能力。前后端分离架构将前端和后端开发解耦，使得前后端可以独立开发、测试和部署，提高了开发效率和质量。

后端使用的依赖包括 Spring Boot 相关的依赖，如 spring-boot-starter-data-neo4j 用于与 Neo4j 图数据库集成，spring-boot-starter-thymeleaf 用于模板引擎，spring-boot-starter-web 用于构建 Web 应用程序，spring-boot-devtools 用于开发过程中的热部署等。此外，还包括了一些其他依赖，如 Apache POI 用于处理 Excel 文件，OpenNLP 用于自然语言处理，Gson 用于 JSON 数据处理等。

前端使用的依赖包括 Vue.js 相关的依赖，如 element-plus 用于构建界面组件，axios 用于发送 HTTP 请求，echarts 用于数据可视化等。此外，还包括了一些开发依赖，如 vite 用于构建和开发前端应用程序，@vitejs/plugin-vue 用于支持 Vue.js 组件等。

综上所述，本系统采用前后端分离的架构设计，前端基于 Vue.js 框架，后端基于 Spring Boot 框架，数据存储使用 Neo4j 图数据库，引入自然语言处理技术，实现了高效的开发流程和良好的可扩展性。

5. 系统关键技术

5.1 界面设计及实现

5.1.1 Element Plus 组件库

界面设计是软件系统的重要组成部分，它直接影响着用户的使用体验和系统的易用性。本系统采用 Element Plus 作为界面设计的组件库，这是一个基于 Vue 3 的桌面端组件库，提供了丰富的组件和灵活的布局功能。

Element Plus 组件库的设计理念是提供一套丰富、灵活且美观的组件，帮助开发者高效地实现现代化的界面设计。通过引入 Element Plus，开发者可以快速构建表单、对话框、表格等常用模块，从而减少开发工作量和提高开发效率。同时，Element Plus 的组件具有高度的扩展性和可定制性，开发者可以根据实际需求对组件进行二次开发，实现个性化的界面设计。

在界面人机交互设计方面，Element Plus 提供了丰富的交互组件和动画效果，使系统的界面更加生动和友好。例如，通过使用 Element Plus 的表单组件，用户可以轻松地输入和提交数据，同时系统可以实时验证数据的正确性并提供反馈。此外，Element Plus 的对话框组件可以用于显示提示信息、确认操作等场景，提供了良好的交互体验。

除了组件的功能和交互设计，Element Plus 还注重界面的美观性和用户体验的一致性。组件库采用了统一的视觉风格和配色方案，使整个系统的界面看起来更加协调和美观。同时，Element Plus 的组件具有响应式设计，可以适应不同屏幕尺寸和设备，提供良好的跨平台体验。

综上所述，本系统采用 Element Plus 作为界面设计的组件库，通过其丰富的组件和灵活的布局功能，实现了现代化的界面设计。同时，Element Plus 的组件具有高度的扩展性和可定制性，使开发者能够根据实际需求进行个性化的界面设计。界面人机交互设计方面，Element Plus 提供了丰富的交互组件和动画效果，使系统的界面更加生动和友好。整体而言，Element Plus 的引入提升了系统的美观性和用户体验的一致性。

5.1.2 echarts展示知识图谱

使用echarts的力导向图进行知识图谱构建，通过和后端请求获取节点和关系数据构建力导向图

规范化查询数据

1. 方法 `normalizedCypher` :

- **功能:** 这个方法的主要功能是将输入的记录列表（`List<Map<String, Object>>`）规范化为Cypher查询语言所需的格式。Cypher是Neo4j图形数据库的查询语言，用于描述图形模式和数据操作。
- **参数:** 方法接受一个名为 `records` 的参数，这是一个列表，其中每个元素是一个映射（Map），映射的键是字符串，值是对象。这些记录代表了要处理的数据。

- **实现:** 方法内部创建了两个 `ArrayList` 对象，分别命名为 `nodes` 和 `relationships`，用于存储节点和关系数据。接着，通过遍历 `records` 列表，对每个记录调用 `processNodeOrRelationship` 方法进行处理。这个处理过程涉及到数据结构的解析和转换。
- **输出:** 最后，使用 `ObjectMapper` 类将处理后的节点和关系数据转换为JSON格式的 `ObjectNode` 对象。这个对象可以用于生成Cypher查询，或者用于进一步的数据处理和分析。

```
public ObjectNode normalizedCypher(List<Map<String, Object>> records) { 2 个用法 🐼 Canary
    List<Map<String, Object>> nodes = new ArrayList<>();
    List<Map<String, Object>> relationships = new ArrayList<>();

    for (Map<String, Object> recordMap : records) {
        processNodeOrRelationship(recordMap, nodes, relationships);
    }

    ObjectMapper objectMapper = new ObjectMapper();
    return objectMapper.valueToTree(createJson(nodes, relationships));
}
```

1. 方法 `processNodeOrRelationship`:

- **功能:** 这个方法用于处理单个记录，并将记录中的节点和关系添加到相应的列表中。它是 `normalizedCypher` 方法中的一个关键步骤，用于解析和分类数据。
- **参数:** 方法接受三个参数：`recordMap`（记录映射）、`nodes`（节点列表）和 `relationships`（关系列表）。`recordMap` 是单个记录的映射表示，其中包含了节点和关系的数据。
- **实现:** 方法内部通过遍历 `recordMap` 的条目，使用 `instanceof` 操作符来判断每个值的类型。如果值是节点类型（`Node`），则将其添加到 `nodes` 列表中；如果值是关系类型（`Relationship`），则将其添加到 `relationships` 列表中。
- **作用:** 这个方法的作用是将原始数据记录中的节点和关系分离出来，为后续的数据处理和分析做准备。通过将节点和关系分类存储，可以更有效地生成Cypher查询，或者进行其他形式的数据操作。


```
private void processNodeOrRelationship(Map<String, Object> recordMap, List<Map<String, Object>> nodes, List<Map<String, Object>> relationships) { 1个用法 🐣 Canary
    for (Map.Entry<String, Object> entry : recordMap.entrySet()) {
        Object value = entry.getValue();
        if (value instanceof Node node) {
            if (containsNode(nodes, node.id())) {
                nodes.add(createNodeJson(node));
            }
        } else if (value instanceof Relationship relationship) {
            if (containsRelationship(relationships, relationship.id())) {
                relationships.add(createRelationshipJson(relationship));
            }
        } else if (value instanceof Path path) {
            for (Node node : path.nodes()) {
                if (containsNode(nodes, node.id())) {
                    nodes.add(createNodeJson(node));
                }
            }
            for (Relationship relationship : path.relationships()) {
                if (containsRelationship(relationships, relationship.id())) {
                    relationships.add(createRelationshipJson(relationship));
                }
            }
        }
    }
}
```

这里展示了一种数据解析和转换的方法，它将从Neo4j图形数据库中Cypher查询到的数据转化成所需的json格式文件用于前端展示图谱。

展示图谱

```
4
5 <script>
6 import * as echarts from 'echarts';
7
8 export default { 1个用法 🐣 Canary
9     name: 'EChartsGraph',
10    props: {
11        nodes: Array,
12        relationships: Array,
13    },
14    data() {
15        return {
16            chart: null,
17            // 定义节点类型的颜色映射
18            nodeCategoryColors: {
19                Author: '#FFA500', // 棕色
20                Paper: '#4169E1', // 蓝色
21                Country: '#3CB371',
22                Institution: '#917634'
23            },
24            // 定义关系类型的颜色映射
25            relationshipLabelColors: {
26                AUTHORED: '#FF69B4', // 粉红色
27                AFFILIATED_WITH: '#228B22',
28                LOCATED_IN: '#704512',
29                CITED: '#746594'
30            }
31        }
32    }
33 }
```

```

31     });
32   },
33   computed: {
34     computedNodes() {
35       return this.nodes.map(node => ({
36         // 使用 id 作为图表中节点的唯一标识符
37         id: node.id.toString(),
38         // 使用 name 作为图表中节点的显示名称
39         name: node.properties.name,
40         // 其他属性可以根据需要进行添加
41         category: node.category[0],
42         // 如果需要显示其他属性，可以在这里添加
43         nationality: node.properties.nationality,
44         // 注意，这里假设每个节点只有一个类别
45         itemType: {
46           color: this.nodeCategoryColors[node.category[0]] || '#0056b3', // 默
47         },
48       }));
49     },
50     computedRelationships() {
51       return this.relationships.map(relationship => ({
52         // 使用 source 和 target 作为关系图的起点和终点
53         source: relationship.source.toString(),
54         target: relationship.target.toString(),
55         // 如果需要显示关系的名称或标签，可以在这里添加
56         name: relationship.name,
57         label: relationship.label,
58         lineStyle: {
59           width: 2,
60           color: this.relationshipLabelColors[relationship.label] || '#e2e0e4',
61         },
62       }));
63     },
64   },
65   mounted() {
66     this.initChart();
67   },
68   beforeDestroy() {
69     if (this.chart) {
70       this.chart.dispose();
71     }
72   },
73   methods: {
74     initChart() {
75       if (!this.chart) {
76         this.chart = echarts.init(this.$refs.chart);
77       }

```



```

77     }
78     this.updateChart();
79 },
80 updateChart() {
81     if (!this.chart) return;
82     const option = {
83         title: {
84             text: '简单知识网络图示例'
85         },
86         tooltip: {},
87         series: [{
88             type: 'graph',
89             layout: 'force',
90             symbolSize: 45,
91             roam: true,
92             edgeSymbol: ['circle', 'arrow'],
93             edgeSymbolSize: [4, 10],
94             edgeLabel: {
95                 textStyle: {
96                     fontSize: 20
97                 }
98             },
99             force: {
100                 repulsion: 2500,
101                 edgeLength: [10, 50]
102             },
103             draggable: true,
104             label: {
105                 show: true,
106                 textStyle: {}
107             },
108             data: this.computedNodes,
109             links: this.computedRelationships,
110             categories: [
111                 {
112                     name: 'paper'
113                 },
114                 {
115                     name: 'author'
116                 },
117                 {
118                     name: 'country'
119                 },
120                 {
121                     name: 'institution'
122                 }
123             ]

```

```

124         }
125     };
126     this.chart.setOption(option);
127 }
128 },
129 match: {
130     // 监听props变化，当数据更新时重新渲染图表
131     computedNodes: {
132         handler() {
133             this.updateChart();
134         },
135         deep: true
136     },
137     computedRelationships: {
138         handler() {
139             this.updateChart();
140         },
141         deep: true
142     }
143 }
144 }
145 </script>
146

```

5.2 图数据库处理及实现

1. Neo4j数据库访问接口准备

在Spring Boot框架中，通过集成Neo4j数据库来支持图形数据库功能是一种常见实践。Neo4j是一个高性能的、NoSQL图形数据库，它非常适合处理大量连接的数据。在Spring框架中，通过使用 `spring-boot-starter-data-neo4j` 依赖项，可以方便地实现与Neo4j数据库的集成。

```

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-neo4j</artifactId>
</dependency>

```

在给出的XML代码片段中，`<dependency>` 标签定义了一个Maven依赖项，这是在Java项目中声明外部库的一种方式。具体来说，这个依赖项是Spring Boot的 `spring-boot-starter-data-neo4j`，它为Spring Boot项目提供了Neo4j支持。这个依赖项包括与Neo4j数据库交互所需的全部类库和配置。

- `<groupId>` 标签指定了依赖项的组ID，这里是 `org.springframework.boot`，表示这个依赖项是由Spring Boot官方提供的。
- `<artifactId>` 标签指定了依赖项的项目ID，即 `spring-boot-starter-data-neo4j`，这是Spring Boot为Neo4j提供的启动器。

在Spring框架中，`AuthorRepository` 作为一个接口，可以用来定义与Neo4j数据库交互的方法。通过Spring Data Neo4j的插件，开发者可以创建一个继承自 `Neo4jRepository` 的 `AuthorRepository` 接口，从而利用Spring Data的强大功能，如声明式查询和分页支持。

例如，如果我们要创建一个 `AuthorRepository` 来处理 `Author` 节点的CRUD（创建、读取、更新、删除）操作，我们可以这样定义：

```
1 package com.example.cloudhuntchartbackend.repository;
2
3 import com.example.cloudhuntchartbackend.entity.Author;
4 import org.springframework.data.neo4j.repository.Neo4jRepository;
5 import org.springframework.stereotype.Repository;
6
7 @Repository 4 个用法 🐣 Canary
8 public interface AuthorRepository extends Neo4jRepository<Author, Long> {
9     Author findById(long id); 0 个用法 🐣 Canary
10 }
11
```

在这个例子中，`AuthorRepository` 接口扩展了 `Neo4jRepository`，后者提供了基本的CRUD操作。通过这种方式，Spring框架能够自动实现这些操作，无需开发者编写具体的数据库访问代码。此外，开发者还可以在 `AuthorRepository` 中定义自定义的查询方法，以满足特定的业务需求。

2. 数据统一读入

a. nbib转规范化Data

本段代码实现了从多个文件加载科研文献数据并进行处理的功能，主要分为三个部分：加载数据文件、提取和映射数据、以及错误处理机制。

◦ 加载文件（`loadFiles` 方法）

在 `loadFiles` 方法中，通过依次调用 `loadInstitutionsAndCountries`、`loadAuthorAndAffiliations` 等多个方法，加载不同类型的数据文件。这些文件分别涉及论文作者信息、机构信息、国家信息、论文标题等。每个文件的加载都通过特定路径传递给方法，以确保数据源的正确性。为了增强代码的健壮性，使用了 `try-catch` 语句来捕获IO异常，防止加载过程中出现的任何错误导致程序崩溃。如果发生错误，系统会输出错误信息，并返回空值。

- **数据映射（`getDataMap` 方法）**

在 `getDataMap` 方法中，数据被整理成一个 `Map` 数据结构，该结构包括论文、作者、机构和国家的映射关系。通过调用前面加载的各个数据集（如 `authorMap`、`institutionMap`、`countryMap` 等），创建了一个包含这些信息的 `HashSet`。这些集合将被用于构建最终的数据映射，以便于后续的数据分析和处理。

- **数据处理（`loadInstitutionsAndCountries` 方法）**

`loadInstitutionsAndCountries` 方法负责加载具体的机构和国家信息。通过使用 `BufferedReader` 读取文件并逐行处理，每一行数据会被按特定的格式进行分割和解析。根据行内的标识符（如PMID、机构名称和国家名称），会创建相应的 `Country` 和 `Institution` 对象，并将其存入对应的映射集合中。此外，如果在加载过程中未能找到相关信息，程序会自动创建新的实体对象以保证数据的完整性和一致性。

- **错误处理**

错误处理是这段代码中的一个重要部分。通过在 `loadFiles` 和 `loadInstitutionsAndCountries` 方法中加入 `try-catch` 语句，确保在文件读取或数据处理过程中发生异常时，能够及时捕捉并输出错误信息。这种做法有效避免了由于单一文件或数据格式问题导致整个程序的失败。

```

21     public Map<String, Set<?>> loadFiles(String adPath, String depPath, String fauPath, String jtPath, String mhPath, String tiPath) {
22         try {
23             loadInstitutionsAndCountries(adPath);
24             loadPublicationDates(depPath);
25             loadAuthorsAndAffiliations(fauPath);
26             loadJournals(jtPath);
27             loadKeywords(mhPath);
28             loadTitles(tiPath);
29             return getDataMap();
30         } catch (IOException e) {
31             System.err.println("An error occurred while loading files: " + e.getMessage());
32             return null;
33         }
34     }
35
36     @private Map<String, Set<?>> getDataMap() { 1 个用法 🐼 Canary
37         Set<Paper> paperSet = new HashSet<>(paperMap.values());
38         Set<Author> authorSet = new HashSet<>(authorMap.values());
39         Set<Institution> institutionSet = new HashSet<>(institutionMap.values());
40         Set<Country> countrySet = new HashSet<>(countryMap.values());
41
42         // 将 Set 集合放入 map 中返回
43         Map<String, Set<?>> dataMap = new HashMap<>();
44         dataMap.put("paper", paperSet);
45         dataMap.put("author", authorSet);
46         dataMap.put("institution", institutionSet);
47         dataMap.put("country", countrySet);
48         return dataMap;
49     }
50
51     private void loadInstitutionsAndCountries(String adPath) throws IOException { 1 个用法 🐼 Canary
52         try (BufferedReader reader = new BufferedReader(new FileReader(adPath))) {
53             String line;
54             while ((line = reader.readLine()) != null) {
55                 // 使用正则表达式分割字符串，更准确地处理数据格式
56                 String[] str = line.split(regex: "[,\\.]");
57
58                 // 提取并处理 PMID，若格式不正确则跳过
59                 Integer pmid = Integer.parseInt(str[0].trim());
60                 String institutionName = str[1].trim();
61                 String countryName = str[2].trim();
62
63                 // 根据国家名获取或创建国家对象 // 根据机构名获取或创建机构对象 // 根据 PMID 获取或创建论文对象
64                 Country country = countryMap.computeIfAbsent(countryName, Country::new);
65                 Institution institution = institutionMap.computeIfAbsent(institutionName, Institution::new);
66                 Paper paper = paperMap.computeIfAbsent(pmid, Paper::new);
67
68                 country.getInstitutionSet().add(institution);
69             }
70         }
71     }

```

b. excel转规范化Data

本段代码实现了从 Excel 文件中读取数据并转换为程序内部的结构（如 `Map` 和集合）。主要分为两个部分：文件读取和数据处理。

◦ 文件读取（`loadFiles` 方法）

在 `loadFiles` 方法中，程序通过接收多个 Excel 文件路径（包括论文、作者、机构、国家等数据文件），依次调用 `LoadExcel` 方法将每个文件加载并解析为对应的数据结构。文件读取顺序严格按照论文、作者、机构、国家的顺序执行，确保数据依赖关系得到正确处理。每次文件加载时，都会传递相应的解析函数（如 `ExcelToData::LoadPaper`）进行处理。

◦ 数据解析（`loadExcel` 方法）

`loadExcel` 方法是实现 Excel 文件解析的核心。该方法通过 `FileInputStream` 读取文件，并使用 Apache POI 库的 `XSSFWorkbook` 解析 Excel 内容。首先，方法获取工作表中的数据，并将表头映射为 `headerMap`，以便后续数据的列索引映射。然后，程序遍历每一行数据，跳过空行并调用指定的解析函数（如 `LoadFunction.loadRow(row, headerMap)`）将每行数据提取出来。

- **数据存储（LoadFunction 接口）**

数据的具体提取逻辑通过 LoadFunction 接口实现。此接口允许根据不同的数据类型（如论文、作者、机构等）提供定制化的处理方法。每次读取到一行数据时，都会调用 LoadFunction 中的 loadRow 方法，将每个字段值提取并存入对应的对象或集合中。

```
20 @ public static Map<String, Iterable<?>> loadFiles(String paperPath, String authorPath, String institutionPath, String countryPath) { 1个用法 🐼 Canary
21     try {
22         //必须第一个加载
23         LoadExcel(paperPath, ExcelToData::LoadPaper);
24         //必须第二个加载
25         LoadExcel(authorPath, ExcelToData::LoadAuthor);
26         //必须第三个加载
27         LoadExcel(institutionPath, ExcelToData::LoadInstitution);
28         //必须第四个加载
29         LoadExcel(countryPath, ExcelToData::LoadCountry);
30         return getDataMap();
31     } catch (IOException e) {
32         System.err.println("An error occurred while loading files: " + e.getMessage());
33         return null;
34     }
35 }
36
37 private static void loadExcel(String path, LoadFunction function) throws IOException { 4个用法 🐼 Canary
38     try (FileInputStream fis = new FileInputStream(path);
39         Workbook workbook = new XSSFWorkbook(fis)) {
40         Sheet sheet = workbook.getSheetAt(0);
41         Map<String, Integer> headerMap = createHeaderMap(sheet);
42         for (int i = 1; i <= sheet.getLastRowNum(); i++) {
43             Row row = sheet.getRow(i);
44             if (row == null) continue;
45             function.loadRow(row, headerMap);
46         }
47     }
48 }
```

3. Data保存以及导出

- a. Data导出到Excel

本段代码实现了将论文数据从内存结构导出到 Excel 文件的功能。它包括四个主要步骤：数据处理、工作表创建、表头写入、数据写入。

- **数据处理（getExcel 方法）**

在 getExcel 方法中，程序通过接收一个包含论文数据的 Set<Paper> 集合，将数据转换为适合导出的格式。每篇论文数据会传递给 createPaperExcel 方法进行处理。该方法接受一个 Map<String, Set<?>> 类型的参数，用于按类别（论文、作者、机构、国家等）处理并输出数据。

- **工作表创建（createPaperExcel 方法）**

createPaperExcel 方法负责创建一个新的Excel工作簿，并初始化一个名为“Data”的工作表。工作表的创建通过 workbook.createSheet 方法实现，确保数据可以被组织到表格中。为优化显示，使用 sheet.trackAllColumnsForAutoSizing() 方法，使得列宽自动适应内容，确保数据展示时不被截断。

- **表头写入（createHeaders 方法）**

在 createPaperExcel 方法中，首先通过 createHeaders 方法为 Excel 文件添加表头。表头由一组字段组成，包含论文的核心信息（如ID、标题、日期等）。该方法通过遍历预定义的 PAPER_HEADERS 数组创建每一列的名称，并为每一列单元格填充表头数据。

数据写入（createRowForPaper 方法）

数据写入通过 createRowForPaper 方法逐行进行。每篇论文的字段值会被提取，并写入到相应的单元格中。该方法首先为每篇论文创建一个新行，然后填充各列的具体数据。在写入过程中，如果某些字段为空（如关键词、期刊等），则会自动插入空字符串，避免表格出现空白单元格。每一行数据都由 row.createCell 进行处理，确保数据格式正确。

```
28 @ public void getExcel(Map<String, Set<?>> map, String paperFilePath, String authorFilePath, String institutionFilePath, String countryFilePath) {
29     createPaperExcel((Set<Paper>) map.get("paper"), paperFilePath);
30     createAuthorExcel((Set<Author>) map.get("author"), authorFilePath);
31     createInstitutionExcel((Set<Institution>) map.get("institution"), institutionFilePath);
32     createCountryExcel((Set<Country>) map.get("country"), countryFilePath);
33 }
34
35 @ private void createPaperExcel(Set<Paper> setData, String fileName) { 1个用法 新 *
36     try (SXSSFWorkbook workbook = new SXSSFWorkbook();
37         FileOutputStream outputStream = new FileOutputStream(fileName)) {
38
39         SXSSFSheet sheet = workbook.createSheet(sheetname: "Data");
40
41         sheet.trackAllColumnsForAutoSizing();
42
43         // 创建表头
44         createHeaders(sheet, PAPER_HEADERS);
45
46         // 填充数据
47         int rowNum = 1;
48         for (Paper data : setData) {
49             String citedPmid = data.getPaperSet().stream()
50                 .map(Paper::getName) // 返回 Integer
51                 .map(String::valueOf) // 将 Integer 转换为 String
52                 .collect(Collectors.joining(delimiter: ",", ""));
53             createRowForPaper(sheet, data, rowNum++, citedPmid);
54         }
55
56         // 自动调整所有列的宽度
57         for (int i = 0; i < PAPER_HEADERS.length; i++) {
58             sheet.autoSizeColumn(i);
59         }
60
61         // 写入文件
62         workbook.write(outputStream);
63     } catch (IOException e) {
64         e.printStackTrace();
65     }
66 }
67
68 private static void createRowForPaper(Sheet sheet, Paper data, int rowNum, String citedPmid) { 1个用法 新 *
69     Row row = sheet.createRow(rowNum);
70     row.createCell(0).setCellValue(data.getId() != null ? String.valueOf(data.getId()) : "");
71     row.createCell(1).setCellValue(data.getName());
72     row.createCell(2).setCellValue(data.getTitle() != null ? data.getTitle() : "");
73     row.createCell(3).setCellValue(data.getDate() != null ? data.getDate() : "");
74     row.createCell(4).setCellValue(data.getKeyword() != null ? data.getKeyword() : "");
75     row.createCell(5).setCellValue(data.getJournal() != null ? data.getJournal() : "");
76     row.createCell(6).setCellValue(""); // CITES
77     row.createCell(7).setCellValue(citedPmid);
78 }
79
80 private static void createHeaders(Sheet sheet, String[] headers) { 4个用法 新 *
81     Row headerRow = sheet.createRow(0);
82     for (int i = 0; i < headers.length; i++) {
83         Cell cell = headerRow.createCell(i);
84         cell.setCellValue(headers[i]);
85     }
86 }
87 }
```

b. Data存入数据库

本段代码通过将Data规范化之后使用Neo4j的访问器对不同节点以及关系进行保存操作，主要通过调用相应的 Repository 接口的 saveAll 方法批量存储数据到数据库。此功能支持将数据存入 Neo4j 图数据库，同时能够处理多种类型的数据（论文、作者、机构、国家）。

- **批量保存数据**

在 `addAllPaper`、`addAllAuthor`、`addAllInstitution` 和 `addAllCountry` 方法中，程序通过调用对应的 `saveAll` 方法将各类数据（论文、作者、机构、国家）批量存入数据库。每个方法接受一个 `Iterable` 类型的参数，确保可以处理大规模数据并提高写入效率。这些方法分别将数据传递给相应的 `Repository` 进行持久化，确保数据的正确存储。

- **数据加载与分发（`saveExcelToNeo4j` 方法）**

`saveExcelToNeo4j` 方法负责协调数据的加载与存储。首先，它调用 `ExcelToData.loadFiles` 方法，加载包含论文、作者、机构、国家的 Excel 文件，并将数据存入一个 `Map<String, Iterable<?>>` 中。然后，使用增强的 `forEach` 方法遍历 `Map`，根据不同的键（如"paper"、"author"、"institution"、"country"），将相应的数据传递给对应的存储方法（`addAllPaper`、`addAllAuthor`、`addAllInstitution`、`addAllCountry`）。通过这种方式，数据可以被自动分类并保存到数据库中。

- **异常处理与数据验证**

如果 `dataMap` 为空，表示数据加载失败或为空，程序没有进行任何存储操作。若遇到未知数据类型，代码会通过抛出 `IllegalArgumentException` 异常来处理，确保只有合法的数据类型会被存储，避免因类型错误导致的数据存储问题。

```
55 > public void addAllPaper(Iterable<Paper> papers) { paperRepository.saveAll(papers); }
58 >
59 > public void addAllAuthor(Iterable<Author> authors) { authorRepository.saveAll(authors); }
62 >
63 > public void addAllInstitution(Iterable<Institution> institutions) { institutionRepository.saveAll(institutions); }
66 >
67 > public void addAllCountry(Iterable<Country> countries) { countryRepository.saveAll(countries); }
70 >
71 > public void saveExcelToNeo4j(String paperPath, String authorPath, String institutionPath, String countryPath) { 2个用法 🐼 Canary
72 // 使用方法引用和Map的增强for循环来简化代码
73 Map<String, Iterable<?>> dataMap = ExcelToData.loadFiles(paperPath, authorPath, institutionPath, countryPath);
74 if (dataMap != null) {
75     dataMap.forEach((key, value) -> {
76         switch (key) {
77             case "paper":
78                 addAllPaper((Iterable<Paper>) value);
79                 break;
80             case "author":
81                 addAllAuthor((Iterable<Author>) value);
82                 break;
83             case "institution":
84                 addAllInstitution((Iterable<Institution>) value);
85                 break;
86             case "country":
87                 addAllCountry((Iterable<Country>) value);
88                 break;
89             default:
90                 // 可以在这里处理未知类型的逻辑或者抛出异常
91                 throw new IllegalArgumentException("Unknown data type: " + key);
92         }
93     });
94 }
95 }
```

5.3 自然语言处理及实现

自然语言处理（NLP）模块在本系统中扮演着至关重要的角色，它负责从非结构化的文本数据中提取出有价值的信息。为了实现这一目标，我们采用了Apache OpenNLP这一基于Java的开源NLP工具包。Apache OpenNLP支持多种NLP任务，包括分词、词性标注、命名实体识别、句子分割等，为我们的系统提供了强大的文本处理能力。

在具体实现上，我们利用OpenNLP中预先训练好的中文实体识别模型对论文文本进行深入分析与处理。实体识别是NLP中的一项基本任务，它旨在从文本中识别出具有特定意义的实体，如人名、地名、组织名、时间等。在本系统中，实体识别的目的在于从论文文本中提取出与知识图谱构建相关的关键实体，例如公司、国家、人名、机构名、时间等。

```
3 import opennlp.tools.namefind.TokenNameFinderModel;
4 import opennlp.tools.parser.ParserModel;
5 import opennlp.tools.sentdetect.SentenceModel;
6 import opennlp.tools.tokenize.TokenizerModel;
7 import org.springframework.context.annotation.Bean;
8 import org.springframework.context.annotation.Configuration;
9
10 import java.io.File;
11 import java.io.IOException;
12
13 @Configuration 0 个用法 🐦 Canary
14 public class OpenNLPConfig {
15
16     @Bean 0 个用法 🐦 Canary
17     public SentenceModel sentenceModel() throws IOException {
18         return new SentenceModel(new File( pathname: "bin/en-sent.bin"));
19     }
20     @Bean 0 个用法 🐦 Canary
21     public ParserModel parserModel() throws IOException {
22         return new ParserModel(new File( pathname: "bin/en-parser-chunking.bin"));
23     }
24     @Bean 0 个用法 🐦 Canary
25     public TokenizerModel tokenizerModel() throws IOException {
26         return new TokenizerModel(new File( pathname: "bin/en-token.bin"));
27     }
28
29     @Bean("personNameFinderModel") 0 个用法 🐦 Canary
30     public TokenNameFinderModel personNameFinderModel() throws IOException {
31         return new TokenNameFinderModel(new File( pathname: "bin/en-ner-person.bin"));
32     }
33
34     @Bean("locationNameFinderModel") 0 个用法 🐦 Canary
35     public TokenNameFinderModel locationNameFinderModel () throws IOException {
36         return new TokenNameFinderModel(new File( pathname: "bin/en-ner-location.bin"));
37     }
38     @Bean("institutionNameFinderModel") 0 个用法 🐦 Canary
39     public TokenNameFinderModel institutionNameFinderModel() throws IOException {
40         return new TokenNameFinderModel(new File( pathname: "bin/en-ner-institution.bin"));
41     }
42     @Bean("moneyNameFinderModel") 0 个用法 🐦 Canary
43     public TokenNameFinderModel moneyNameFinderModel () throws IOException {
44         return new TokenNameFinderModel(new File( pathname: "bin/en-ner-money.bin"));
45     }
46     @Bean("dateNameFinderModel") 0 个用法 🐦 Canary
47     public TokenNameFinderModel dateNameFinderModel () throws IOException {
48         return new TokenNameFinderModel(new File( pathname: "bin/en-ner-date.bin"));
49     }
50     @Bean("timeNameFinderModel") 0 个用法 🐦 Canary
51     public TokenNameFinderModel timeNameFinderModel () throws IOException {
52         return new TokenNameFinderModel(new File( pathname: "bin/en-ner-time.bin"));
53     }
54
55 }
56
```

提取到的实体被构建成键值对的形式，并保存到JSON文件中。这种结构化的数据格式便于后续处理，尤其是用于生成Cypher语句。Cypher是Neo4j图数据库的查询语言，它允许用户以声明式的方式

查询和操作图数据。通过将提取的实体转换为Cypher语句，可以高效地将这些信息插入到Neo4j数据库中，为知识图谱的构建提供数据支撑。

此外，通过NLP技术提取的关键信息还可以用于实现智能问答功能。系统可以根据用户的问题，从知识图谱中检索相关信息，并给出准确的答案。例如，用户可以询问关于某个特定研究领域的最新进展，系统通过分析知识图谱中的相关节点和关系，能够提供最新的研究论文和重要结论。

在代码实现方面，我们通过配置类（OpenNLPConfig）加载OpenNLP所需的各种模型，如句子检测模型、解析模型、分词模型以及各种命名实体识别模型。

```
public Map<String, Object> extractEntityAttributes(String sentence) { 1 个用法 🐦 Canary
    Map<String, Object> attributes = new HashMap<>();
    // 分句
    String[] sentences = sentenceDetector.sentDetect(sentence);
    for (String sentenceText : sentences) {
        // 分词
        String[] tokens = tokenizer.tokenize(sentenceText);
        // 识别实体
        Span[] personSpans = personFinder.find(tokens);
        Span[] institutionSpans = institutionFinder.find(tokens);
        Span[] dateSpans = dateFinder.find(tokens);

        // 处理每个实体并存储到属性 map 中
        for (Span span : personSpans) {
            String entity = tokens[span.getStart()];
            attributes.put("Author", entity);
        }
        for (Span span : institutionSpans) {
            String entity = tokens[span.getStart()];
            attributes.put("Institution", entity);
        }
        for (Span span : dateSpans) {
            String entity = tokens[span.getStart()];
            attributes.put("Date", entity);
        }
    }
    return attributes;
}
```

这些模型是预先训练好的，用于执行特定的NLP任务。同时，我们通过实体提取服务（EntityExtractorService）使用OpenNLP工具执行实际的NLP任务，如句子检测、分词和命名实体识别。在这个过程中，我们首先通过SentenceDetectorME检测输入文本中的句子，然后使用TokenizerME对每个句子进行分词。接着，利用不同的NameFinderME实例识别出句子中的各种实体，如人名、机构名、日期等。每个识别出的实体都被转换为键值对，并存储在Map中，最终这个Map可以被用于生成Cypher语句，以将提取的信息插入到Neo4j数据库中。

通过这种方式，我们的系统能够从非结构化的文本数据中提取出结构化的信息，为知识图谱的构建和智能问答功能提供了数据支撑。这不仅提高了信息处理的效率，也增强了系统的智能化水平，使得用户能够更加便捷地获取和利用学术资源。

6. 部署文档

在进行项目的开发以及使用之前，首先需要完成JDK17、node.js、neo4j的下载与安装

6.1 环境准备

在部署系统前，需要确保服务器或本地开发环境满足以下基础要求：

- **操作系统：**Windows
- **JDK17：**用于运行 Spring Boot 后端服务
- **Node.js (16.x 或以上版本)：**用于运行前端项目
- **Neo4j (5.x 或以上版本)：**图数据库，用于存储和管理知识图谱

6.2 依赖安装

1. 安装JDK17

- 从 [Oracle JDK 官网](#) 或 [OpenJDK](#) 下载 JDK 17。
- [配置环境变量](#)

2. 安装Node.js

- 从 [Node.js 官网](#) 下载 LTS 版本。

3. 安装 Neo4j

- 从 [Neo4j 官网](#) 下载社区版。
- 解压安装包，在控制台中启动 Neo4j 服务：

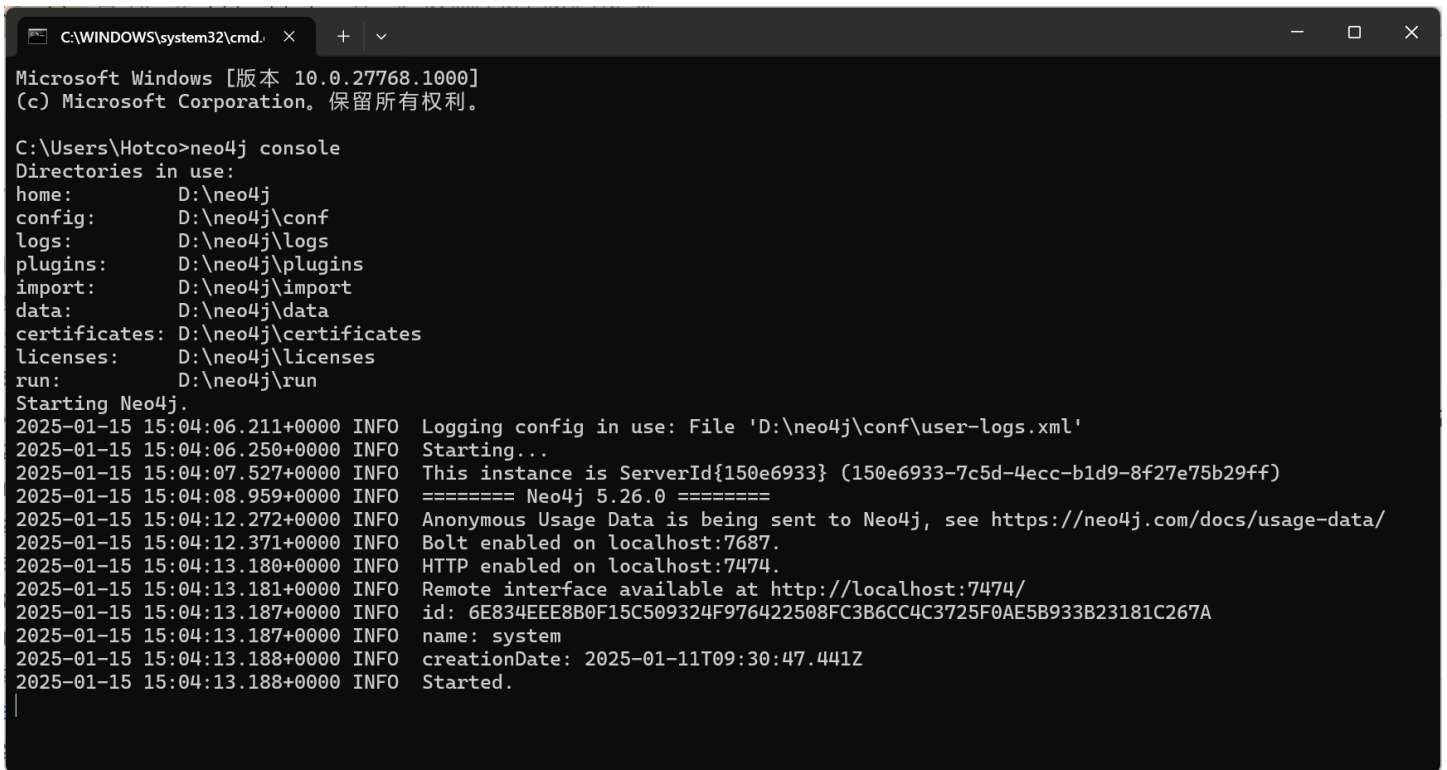
```
1 neo4j console
```

- 在浏览器输入网址：<http://localhost:7474/>，默认情况下，首次登录需要设置用户名和密码（默认用户名和密码均为 `neo4j`，密码需要在首次登录时重置）。

6.3 下载项目及启动

- 项目Github地址：<https://github.com/HotcocoaCanary/AI-Yunxun>
- 可以选择下载源文件也可以拉取项目
- 启动Neo4j

```
1 neo4j console
```



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [版本 10.0.27768.1000]
(c) Microsoft Corporation。保留所有权利。

C:\Users\Hotco>neo4j console
Directories in use:
home:          D:\neo4j
config:        D:\neo4j\conf
logs:          D:\neo4j\logs
plugins:       D:\neo4j\plugins
import:        D:\neo4j\import
data:          D:\neo4j\data
certificates:  D:\neo4j\certificates
licenses:      D:\neo4j\licenses
run:           D:\neo4j\run
Starting Neo4j.
2025-01-15 15:04:06.211+0000 INFO Logging config in use: File 'D:\neo4j\conf\user-logs.xml'
2025-01-15 15:04:06.250+0000 INFO Starting...
2025-01-15 15:04:07.527+0000 INFO This instance is ServerId{150e6933} (150e6933-7c5d-4ecc-b1d9-8f27e75b29ff)
2025-01-15 15:04:08.959+0000 INFO ===== Neo4j 5.26.0 =====
2025-01-15 15:04:12.272+0000 INFO Anonymous Usage Data is being sent to Neo4j, see https://neo4j.com/docs/usage-data/
2025-01-15 15:04:12.371+0000 INFO Bolt enabled on localhost:7687.
2025-01-15 15:04:13.180+0000 INFO HTTP enabled on localhost:7474.
2025-01-15 15:04:13.181+0000 INFO Remote interface available at http://localhost:7474/
2025-01-15 15:04:13.187+0000 INFO id: 6E834EEE8B0F15C509324F976422508FC3B6CC4C3725F0AE5B933B23181C267A
2025-01-15 15:04:13.187+0000 INFO name: system
2025-01-15 15:04:13.188+0000 INFO creationDate: 2025-01-11T09:30:47.441Z
2025-01-15 15:04:13.188+0000 INFO Started.
```

- 启动前端

- 在front-end文件夹下启动cmd终端顺序运行

```
1 npm install
2 npm run dev
```

- 启动后端

- 在IDE中运行启动类back-end/src/main/java/com/example/cloudhuntchartbackend/CloudHuntChartBackendApplication.java

[1] Walls C .Spring Boot in Action[J]. 2016.

[2] Bird, S., Klein, E., & Loper, E. (2009). *Natural Language Processing with Python*. O’ Reilly Media.

[3] Neo4j. (n.d.). Neo4j documentation. Retrieved [insert date], from <https://neo4j.com/docs/>