**毕业设计说明书**

**基于NLP的可视化评价信息收集与分析系统的设计与实现**

|  |  |
| --- | --- |
| **专 业** | **网络工程** |
| **学 生 姓 名** | **左明** |
| **班 级** | **B网络191** |
| **学 号** | **190107151003** |
| **指 导 教 师** | **仲冰** |
| **完 成 日 期** | **2023年5月11日** |

**基于NLP的可视化评价信息收集与分析系统的设计与实现**

**摘 要**：自然语言处理（Natural Language Processing，NLP）是人工智能的一个重要领域，可以用来构建一个强大的可视化评论信息收集和分析系统。我们提出了一种基于 NLP 的系统的设计和实现，该系统使用 Spacy 进行文本分析，并使用 Echarts 和 Matplotlib 等数据可视化库来呈现分析数据。

拟议的系统包括三个主要部分：数据收集、分析和可视化。数据收集模块采用网络抓取技术从不同来源收集数据，包括社交媒体平台和电商平台。分析模块利用 Spacy 处理文本数据并提取有价值的信息，例如情绪、产品特性和用户偏好。最后，将提取的信息使用Echarts和Matplotlib提供的可视化工具呈现，包括条形图、饼图和折线图，增强了系统的可解释性。

为了评估所提出系统的效果，我们对我们系统收集的文本数据集进行了测试。结果表明，该系统可以准确地从文本数据中提取信息，并使用 Echarts 和Matplotlib可视化呈现，为各种应用提供有价值的见解。

总的来说，我们所提出的系统提供了一个工具，用于使用 NLP 技术收集和分析文本数据、用于数据收集的网络爬虫以及用于数据分析和展示的 Echarts 和Matplotlib可视化，这可以帮助组织做出更好的决策并改进他们的服务。

该系统可应用于电子商务、社交媒体监控和公共安全等各个领域。例如，在电子商务领域，该系统可用于收集客户对产品的反馈并分析情绪以提高产品质量。在社交媒体监控中，该系统可用于检测不当内容。

**关键词**：自然语言处理；数据可视化；文本分析；网页抓取；情感分析

**Design and Implementation of NLP-based Visualizable Evaluation Information Collection and Analysis System**

**Abstract**: Natural Language Processing (NLP) is a major field in artificial intelligence that can be leveraged to build a robust visualizable review information collection and analysis system. We propose a design and implementation of an NLP-based system that uses Spacy for text analysis and data visualization libraries such as Echarts and Matplotlib for presenting the analyzed data.

The proposed system comprises three main components: data collection, analysis, and visualization. The data collection module employs web scraping techniques to gather data from diverse sources, including social media platforms and online marketplaces. The analysis module utilizes Spacy to process textual data and extract valuable information, such as sentiment, product features, and user preferences. Finally, the extracted information is presented using visualization tools provided by Echarts and Matplotlib, including bar charts, pie charts, and line graphs, which enhance the system's interpretability.

To evaluate the effectiveness of the proposed system, we tested it on a text dataset collected by our system. The results show that the system can accurately extract information from text data and visualize it using Echarts and Matplotlib, providing valuable insights for various applications.

Overall, the proposed system offers a powerful tool for collecting and analyzing textual data using NLP techniques, web scraping for data collection, and Echarts and Matplotlib visualizations for data analysis, which can help organizations make better decisions and improve their services.

The proposed system can be applied in various domains such as e-commerce, social media monitoring, and public safety. For instance, in the e-commerce domain, the system can be used to collect customer feedback on products and analyze the sentiment to improve product quality. In social media monitoring, the system can be used to detect inappropriate content.

**Key words**: Natural Language Processing; data visualization; text analysis; web scraping; sentiment analysis

**目 录**

[1 概述 1](#_Toc134204512)

[1.1 开发背景 1](#_Toc134204513)

[1.2 开发意义 2](#_Toc134204514)

[1.3 课题开发的工具及技术 2](#_Toc134204515)

[2 系统分析 4](#_Toc134204516)

[2.1 需求分析 5](#_Toc134204517)

[2.2 可行性分析 6](#_Toc134204518)

[2.3 数据库选择 7](#_Toc134204519)

[2.4 系统目标 7](#_Toc134204520)

[3 系统的概要设计 8](#_Toc134204521)

[3.1 总体设计 8](#_Toc134204522)

[3.2 系统架构 8](#_Toc134204523)

[3.3 系统的模块功能划分 9](#_Toc134204524)

[3.4 系统的接口设计 10](#_Toc134204525)

[3.5 系统的错误处理 10](#_Toc134204526)

[4 系统的详细设计 11](#_Toc134204527)

[4.1 系统接口设计 11](#_Toc134204528)

[4.2 系统功能模块设计 13](#_Toc134204529)

[4.3 系统数据收集模块设计 15](#_Toc134204530)

[4.4 系统数据分析模块设计 16](#_Toc134204531)

[4.5 系统数据可视化模块设计 17](#_Toc134204532)

[4.6 系统任务调度与通知模块设计 19](#_Toc134204533)

[5 系统实现 21](#_Toc134204534)

[5.1 系统接口的实现 21](#_Toc134204535)

[5.2 系统登录与注册模块的实现 21](#_Toc134204536)

[5.2 系统数据收集模块的实现 22](#_Toc134204537)

[5.3 系统数据分析模块的实现 24](#_Toc134204538)

[5.4 系统数据可视化模块的实现 27](#_Toc134204539)

[5.5 系统任务调度与通知模块的实现 28](#_Toc134204540)

[5.6 系统Docker部署的实现 32](#_Toc134204541)

[6 系统部署与测试 34](#_Toc134204542)

[6.1 系统部署 34](#_Toc134204543)

[6.2 系统登录模块测试 35](#_Toc134204544)

[6.3 系统数据收集模块测试 37](#_Toc134204545)

[6.4 系统数据分析模块测试 39](#_Toc134204546)

[6.5 系统数据可视化模块测试 40](#_Toc134204547)

[6.6 系统任务调度与通知模块测试 42](#_Toc134204548)

[6.7 系统其他功能测试 44](#_Toc134204549)

[7 系统的发展方向及应用前景 46](#_Toc134204550)

[8 结束语 46](#_Toc134204551)

[参考文献 47](#_Toc134204552)

[致 谢 48](#_Toc134204553)

[附 录 49](#_Toc134204554)

[附录1 源程序清单 49](#_Toc134204555)

[附录1.1 系统功能模块 49](#_Toc134204556)

[附录1.2 数据收集模块 51](#_Toc134204557)

[附录1.3数据分析模块 53](#_Toc134204558)

[附录1.4数据可视化模块 55](#_Toc134204559)

[附录1.5任务调度与通知模块 58](#_Toc134204560)

[附录1.6 Dockerfile和docker-compose.yml 60](#_Toc134204561)

**基于NLP的可视化评价信息收集与分析系统的设计与实现**

# 1 概述

## 1.1 开发背景

随着互联网和社交媒体的普及，人们越来越依赖于在线评价来做出决策，如选择商品、预定旅游行程、选购服务等。这些评价通常包含了大量的信息和情感，然而，由于人工处理这些信息耗时且容易出现误判，因此自动化地分析和处理这些信息变得尤为重要。

基于NLP的可视化评价信息收集与分析系统是一种可以自动地收集和分析在线评价的工具，它可以将大量的文本数据转换成易于理解和分析的形式，并可以使用可视化技术来帮助用户更好地理解和分析这些数据。

基于NLP的可视化评价信息收集与分析系统的开发背景就是为了满足人们处理和分析在线评价的需求，帮助用户更好地理解和分析这些数据，为他们做出更加明智的决策提供支持。

### 1.1.1 NLP技术的发展

自20世纪50年代以来，自然语言处理（NLP）技术在计算机科学、人工智能和语言学等领域得到了广泛应用和研究。随着计算机性能的提高和数据量的增加，NLP技术也得到了快速的发展和普及。

NLP技术的发展主要经历了以下几个阶段：

a)基于规则的方法：早期的NLP技术主要采用人工编写规则的方式进行文本分析和语言处理。这种方法需要耗费大量的人力和时间，且规则的编写和维护十分复杂。

b)统计方法：20世纪90年代，随着机器学习技术的兴起，NLP技术也开始采用统计方法进行文本分析和语言处理。这种方法利用大量的语言数据进行训练，可以自动学习语言规律和模式。

c)深度学习方法：近年来，深度学习技术的发展也给NLP技术带来了重大的进展。利用深度学习算法，可以构建更加复杂和准确的NLP模型，例如文本分类、命名实体识别、机器翻译等。

### 1.1.2 可视化技术的发展

可视化技术的发展主要经历以下几个阶段：

a)基于手工绘图的方法：早期的可视化技术主要依靠人工绘制图表和图形来展示数据。这种方法需要耗费大量的人力和时间，且绘制的图表通常较为简单和粗糙。

b)统计图表方法：随着计算机技术的发展，出现了许多自动生成统计图表的软件工具，例如Excel和SPSS等。这种方法可以方便地生成各种类型的图表，但通常需要人工进行进一步的调整和优化。

c)交互式可视化方法：近年来，交互式可视化技术得到了快速发展。这种方法可以让用户通过交互操作来探索和分析数据，例如缩放、滚动和筛选等操作。交互式可视化技术通常采用Web技术和可视化库来实现。

### 1.1.3 爬虫技术的发展

爬虫技术是指利用程序自动获取和抓取互联网上的数据的技术。随着互联网的发展和数据量的增加，网络爬虫技术也得到了快速的发展和应用。

爬虫技术的发展主要经历以下几个阶段：

a)手工抓取：早期的爬虫技术主要是手工抓取网页数据，需要人工浏览和提取数据，工作效率低下。

b)基于规则的自动化抓取：随着计算机技术的发展，出现了一些自动化抓取工具，例如wget和curl等。这种方法需要用户自己编写规则来指定要抓取的网页和数据，存在一定的局限性。

c)基于爬虫框架的自动化抓取：近年来，出现了许多开源的爬虫框架，例如Scrapy和BeautifulSoup等。这种方法可以更加灵活和高效地抓取数据，且具有很好的可扩展性和稳定性。

d)智能化爬虫技术：最近几年，随着人工智能和机器学习技术的发展，智能化爬虫技术得到了快速的发展。这种方法可以通过机器学习算法来识别网页的结构和内容，以便更准确地抓取数据。

## 1.2 开发意义

本系统开发的意义主要体现在以下几个方面：

a)提高评价信息的收集和分析效率：通过使用NLP和可视化技术对于评价信息进行处理和展示，可以大大提高评价信息的收集和分析效率，从而帮助企业、政府和个人等更好地了解用户的需求和反馈。

b)优化产品和服务质量：通过对于评价信息进行分析，可以得到用户对于产品和服务的反馈和建议，帮助企业和政府优化产品和服务质量，提高用户的满意度和忠诚度。

c)促进企业竞争力的提升：通过分析竞争对手的评价信息，可以了解其产品和服务的优势和劣势，从而帮助企业制定更好的竞争策略，提高企业的竞争力。

d)推动相关技术的研究和应用：该系统涉及到文本处理、情感分析和可视化技术等方面，可以促进相关技术的研究和应用，从而推动人工智能和可视化技术的发展。

## 1.3 课题开发的工具及技术

### 1.3.1 开发工具

a)PyCharm 2023.1（专业版）

PyCharm 是专门为 Python 编程语言设计的集成开发环境 (IDE)。它提供了广泛的工具和功能，例如代码完成、语法突出显示、代码分析、调试和测试。 PyCharm 还支持各种 Python 框架，如 Django、Flask 和 FastAPI，以及各种版本控制系统。

b)Git

Git 是一种分布式版本控制系统，用于在软件开发过程中跟踪源代码的更改。它允许多个开发人员同时协作处理一个项目，同时跟踪更改和管理冲突。Git 提供了范围广泛的命令和工具，用于管理代码更改、合并代码以及在必要时回滚更改。

c)Chrome

Chrome 是谷歌开发的网络浏览器。它被开发人员广泛用于测试和调试 Web 应用程序。 它提供各种开发人员工具，例如开发人员控制台，允许开发人员查看和修改网页元素、运行 JavaScript 代码以及分析网络流量。 它还支持扩展，使开发人员可以轻松地向浏览器添加自定义功能。

### 1.3.2 技术选择

爬虫模块使用 Scrapy 来爬取评价信息。

NLP 模块使用 Spacy 来处理评价信息。

可视化模块使用 Vue.js 来设计和实现前端界面，用户可以在界面上查看评价信息和相关分析结果。

后端使用 FastAPI 来实现 API，包括用户认证、数据查询和分析等功能。

任务调度和通知模块使用 Celery 和 RabbitMQ 来处理任务调度和通知。当爬虫爬取到新的评价信息时，会将其发送到 RabbitMQ 的队列中，Celery 会从队列中取出任务并调用相应的函数进行处理。

Pandas 用于处理大规模数据，可以对MongoDB 中的评价信息进行分析和处理。Pandas 提供了一组丰富的数据操作和分析功能，可以轻松地为可视化准备数据，并将结果存储到 Redis中。

Matplotlib 允许用户创建美观且信息丰富的可视化，帮助他们理解数据并得出有意义的见解。

使用Matplotlib 和 Pandas 可用于创建可视化图表等，帮助用户理解其分析结果。例如，Matplotlib 可用于创建词频分布或情感分析分数的可视化，而 Pandas 可用于在可视化之前对数据进行操作和汇总。

Redis 用作缓存和消息代理，可以缓存一些常用的数据以加速访问，并作为 RabbitMQ 的消息代理。

Docker 和 Docker Swarm 用于部署和管理各个模块，可以将每个模块打包成 Docker 镜像，并使用 Docker Swarm 进行部署和管理。可以使用 Docker Compose 来定义和管理多个容器应用程序。

### 1.3.3 技术简介

a)Scrapy是一个Python的网络爬虫框架，可以自动抓取和处理网页数据。它提供了丰富的API，可以让开发者快速定义和运行爬虫，同时支持异步IO和并发处理等特性，可以高效地处理大量数据。

b)MongoDB是一个NoSQL数据库，采用文档型存储方式，可以快速处理大量数据。它支持高可用性和可扩展性，可以方便地进行分布式计算和数据处理。

c)Spacy是一个高性能的自然语言处理库，可以用于文本分类、实体识别等任务。它使提供了简单易用的API和预训练的模型，可以方便地进行文本处理和分析。

d)Elasticsearch是一种基于Lucene的搜索引擎，可以支持全文搜索和数据可视化。它采用分布式架构，可以高效地处理大规模数据，并提供了丰富的查询API和聚合函数等特性，可以方便地进行数据分析和可视化。

e)Vue.js是一种流行的JavaScript框架，可以快速构建复杂的单页应用程序。它具有简单易用的API和组件化开发方式，同时支持虚拟DOM和响应式数据绑定等特性，可以提高开发效率和应用程序性能。

f)FastAPI是一个快速构建API的Python框架，具有高性能、易用、文档友好等特点。它采用异步IO和类型注解等特性，可以提高API的性能和开发效率，同时支持OpenAPI和Swagger等文档自动生成工具，可以方便地进行API文档生成和测试。

g)Celery是一个Python的分布式任务队列，可以支持异步任务调度和消息队列。它采用消息传递和事件驱动的方式，可以方便地处理异步任务和并发计算，同时支持多种消息中间件和存储后端，可以快速地进行任务调度和分布式计算。

h)RabbitMQ是一个开源的消息代理软件，可以支持分布式计算和异步任务调度。它采用AMQP协议和消息队列方式，可以高效地处理消息传递和事件驱动，同时支持多种客户端和后端存储，可以方便地进行分布式计算和异步任务调度。

i)Matplotlib 是一个流行的 Python 库，用于在 Python 中创建静态、动画和交互式可视化。它提供了范围广泛的可定制图形和图表，包括折线图、散点图、条形图、直方图等等。

j)Pandas 是一个流行的用于数据操作和分析的 Python 库。它提供了用于高效存储和操作大型复杂数据集的数据结构。 它的一些主要功能包括数据对齐、数据清理、数据集的合并和连接以及时间序列分析。

k)Redis是一个内存数据库，可以作为高速缓存和消息代理使用。它采用键值存储和发布订阅模式，可以方便地进行数据缓存和消息传递，同时支持多种数据结构和存储方式，可以满足不同的应用场景。

l)Docker是一个开源的容器化平台，可以打包应用程序和服务到容器中，实现快速部署和管理。它采用轻量级的容器技术，可以快速地构建、运行和部署应用程序，同时支持多种操作系统和云平台，可以方便地进行应用程序的扩展和迁移。

m)Docker Swarm是Docker官方提供的容器编排和管理平台，可以支持集群管理和应用程序自动伸缩。它采用集群管理和负载均衡技术，可以方便地进行容器的部署和管理，同时支持服务发现和自动伸缩等特性，可以提高应用程序的可用性和性能。

# 2 系统分析

系统分析是任何系统设计和实现中必不可少的过程。以下是系统分析很重要的一些原因：

a)了解用户需求：系统分析涉及以包括最终用户在内的利益相关者的角度分析需求。此过程有助于确保系统满足其预期用户的需求。

b)识别系统约束：在系统分析期间，可以识别系统的潜在约束或限制，例如数据源、处理能力或用户界面的限制。然后可以在设计阶段解决这些约束，以确保系统可以按预期运行。

c)确定系统架构：系统分析有助于确定合适的系统架构以实现所需的功能。这包括选择合适的软件工具和技术、定义数据流和创建系统模型。

d)确保系统可扩展性：系统分析可以帮助识别潜在的可扩展性问题，例如处理大量数据或容纳许多用户。通过在设计阶段的早期解决这些问题，可以将系统设计为适当扩展。

e)降低开发成本：系统分析有助于在设计阶段的早期识别潜在问题和限制，从而减少在开发过程后期进行代价高昂的重新设计或更改的需要。

## 2.1 需求分析

### 2.1.1 系统实现的功能

基于NLP的可视化评价信息收集与分析系统的功能主要包括：

a)数据收集功能：实现自动抓取目标网站的评价信息，并将数据存储到MongoDB中。

b)数据分析功能：对爬虫获取的评价信息进行清洗、处理、分析等任务，并将处理结果存储到Elasticsearch和Redis中。

c)数据可视化功能：将MongoDB中的评价信息进行可视化展示，从Redis中获取后端生成的图片，并提供数据查看等功能。

d)系统功能：提供API接口，实现用户认证、发送任务、数据查看、分析、查看分析结果等功能。

e)任务调度和通知功能：实现爬虫任务调度、数据处理和通知等功能。

### 2.1.2 系统的功能需求

基于NLP的可视化评价信息收集与分析系统的功能需求包括：

A.数据收集模块需求

a)自动抓取目标网站的评价信息；

b)支持多个目标网站的抓取；

c)将数据存储到MongoDB中。

B.数据分析模块需求

a)支持对评价信息的清洗、处理、分析等任务；

b)将处理结果存储到Elasticsearch中。

C.数据可视化模块需求

a)支持对MongoDB中的评价信息进行可视化展示；

b)支持数据查看和分析等功能；

c)支持多种指标的可视化功能。

d)将数据分析结果存储在Redis缓存中，以加速访问。

D.系统功能模块需求

a)提供API接口，用户认证、发送任务、数据查看、分析、查看分析结果等功能；

b)完成对其他模块的支撑功能。

E.任务调度和通知模块需求

a)实现爬虫任务调度功能；

b)实现任务通知功能。

## 2.2 可行性分析

基于NLP的可视化评价信息收集与分析系统使用的技术包括FastAPI、Vue.js、Scrapy、Celery、RabbitMQ、Pandas、Matplotlib、Spacy、Redis、MongoDB、Elasticsearch和Docker Swarm等。

### 2.2.1 技术可行性

该系统使用的技术都是经过广泛应用和验证的成熟技术，可以满足系统的需求，并且具备足够的性能和可扩展性。

例如，Scrapy是一个Python的网络爬虫框架，可以自动抓取和处理网页数据；Celery和RabbitMQ可以协同工作，使得任务调度和消息队列变得简单；Spacy是一个高性能的自然语言处理库，可以用于文本分类、实体识别等任务；Redis和MongoDB是NoSQL数据库，可以快速处理大量数据等等。

### 2.2.2 经济可行性

这些技术都是开源的或者有免费版本，可以减少系统的开发和运营成本。同时，这些技术也有商业版本和服务，可以支持更高级别的功能和支持。

### 2.2.3 数据可行性和安全可行性

A.数据可行性

从以下几个方面考虑：

a)数据来源：系统需要从多个渠道收集评价信息，例如社交媒体、评论和问卷调查等。需要评估这些数据来源的可靠性和数据量是否足够，以及数据是否易于清洗和处理。

b)数据清洗和处理：系统需要对收集到的评价信息进行清洗和处理，例如去除重复数据和无效数据，以便更准确地分析数据。需要评估清洗和处理数据的成本和效率。

c)数据存储和管理：系统需要建立数据存储和管理机制，以保证数据的安全和可靠性。需要评估数据存储和管理的成本和效率。

B.安全可行性

考虑以下几个方面：

a)数据安全：系统需要采取合适的技术措施保护数据的安全，例如加密和访问控制等。

b)隐私保护：系统需要保护用户的隐私不受侵犯，例如匿名化处理和数据脱敏等。

c)系统安全：系统需要保护系统的安全，防止遭受黑客攻击和恶意软件侵入等，需要采取合适的安全措施，例如身份验证和防火墙等。

基于NLP的可视化评价信息收集与分析系统使用的这些技术在数据处理和安全方面都具有一定的优势。例如，MongoDB和Elasticsearch具有高性能、可扩展和数据可视化等特点，可以满足系统的数据处理需求；Redis可以作为高速缓存提高数据访问速度；Spacy可以支持文本处理和语言分析等任务。同时，这些技术也具有相应的安全特性，例如可以支持数据加密和身份验证等措施，保护用户的数据安全和隐私不受侵犯。

## 2.3 数据库选择

在为本系统选择数据库时，需要考虑几个因素。基于系统的需求和特点，MongoDB是较适合的数据库选择,原因如下：

a)面向文档：MongoDB是一个面向文档的数据库,这意味着它将数据存储在类似 JSON 的文档中，这些文档可以具有不同的结构。这使得它非常适合存储非结构化和半结构化数据，这是基于NLP的系统的典型特征。

b)可扩展性：MongoDB具有高度可扩展性，可以轻松处理大量数据。它支持分片，可以跨多个服务器进行水平扩展。这对于需要频繁更新和实时分析数据的系统来说很重要。

c)灵活性：MongoDB在模式设计方面提供了高度的灵活性，允许随着需求的发展改变数据结构。这对于处理不断变化和演变的数据的基于NLP的系统特别有用。

## 2.4 系统目标

本系统的目标是：

a)使用自然语言处理（NLP）技术收集和分析评价信息。该系统的主要目标是通过从基于文本的数据（例如评论或评论）中提取有意义的见解并将其转换为可视化的指标来自动分析文本数据。

b)该系统可用于多种应用，例如分析客户对产品的反馈、评估用户对网站的参与度，或监控社交媒体对品牌或活动的情绪。通过自动分析文本数据，系统可以节省时间和资源。

c)提供一个集成化的平台，支持从多个渠道收集评价信息，进行数据清洗和处理，并进行可视化分析和展示。

d)提供多种数据可视化方式，例如词云图、柱状图、折线图等，以及文本可视化分析，例如情感图等，帮助用户更直观地理解和分析数据。

e)保证数据的安全和用户隐私的保护，采取合适的技术措施，例如身份验证、数据加密和访问控制等。

# 3 系统的概要设计

## 3.1 总体设计

### 3.1.1 需求规定

该系统应该能够从多个来源收集信息，比如基于文本的评论。然后，它应该使用自然语言处理技术来分析数据并以可视化形式输出。

功能性能要求：

a)输入：系统应该能够处理文本格式的输入数据。它还应该能够同时处理来自多个来源的输入。

b)处理：系统应使用自然语言处理技术来分析文本数据。

c)输出：系统应提供可量化的指标和可视化效果，以总结分析产生的结果。

### 3.1.2 操作环境

a)硬件：系统应该能够在标准计算机硬件上运行，包括台式机和笔记本电脑。

b)系统环境：系统应能在Windows、Linux上运行。它还应该为用户提供文档和支持资源。

### 3.1.3概念和处理流程

基于NLP的可视化评价信息采集与分析系统的基本设计理念是从多个平台收集文本数据，然后利用自然语言处理技术对文本数据进行自动化分析。然后，这些见解以可量化指标和可视化的形式呈现。

系统的处理流程如下图所示：

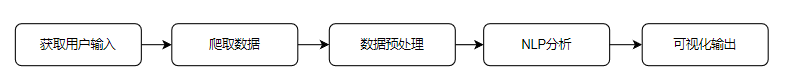


图3-1 系统处理流程

系统首先对输入数据进行预处理，以确保其采用适合分析的格式。然后它对文本数据执行 NLP 分析以提取有意义的信息。生成的信息随后用于生成可量化指标和可视化形式的输出。

在编程概念方面，系统利用模块和子程序将分析过程分解为更小、更易于管理的组件。它还利用并发性和并行性来优化性能，并在必要时跨多台机器分配处理任务。

## 3.2 系统架构

基于NLP的可视化评价信息收集与分析系统的架构图如图3-2所示。

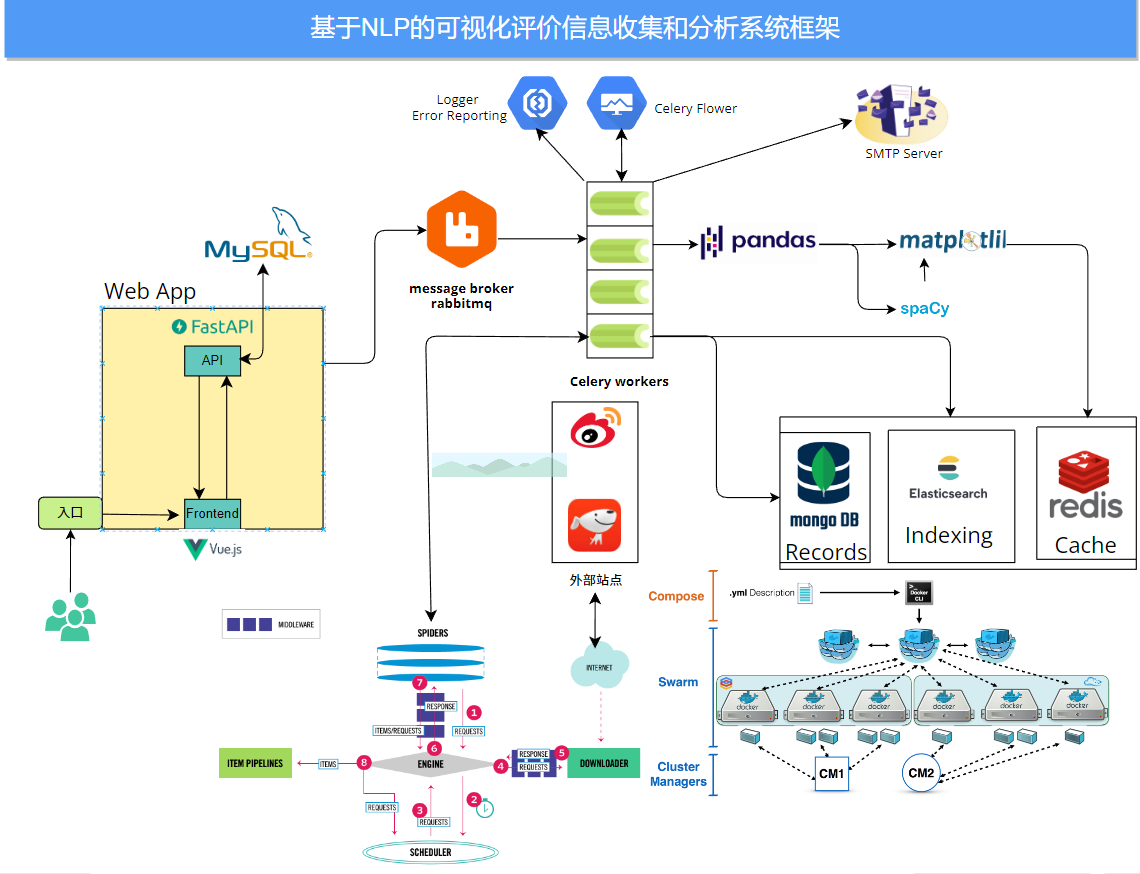


图3-2 系统架构图

基于NLP的可视化评价信息收集与分析系统采用了一系列不同的技术，包括FastAPI、Vue.js、Scrapy、Celery、RabbitMQ、matplotlib、Pandas、spaCy、Redis、MongoDB、Elasticsearch和Docker Swarm。

## 3.3 系统的模块功能划分

如图3-3所示为系统的功能图，主要包括以下几个模块：

（1）系统功能模块：包含登录及验证的简化模块，以及快捷菜单栏、系统日志等功能。

（2）数据收集模块：对指定站点的数据收集的功能。

（3）数据分析模块：对收集到的数据进行分析的功能。

（4）数据可视化模块：对数据分析的结果进行可视化展示的功能。

（5）任务调度与通知模块：对收集任务进行调度以及在任务完成时通知的功能。

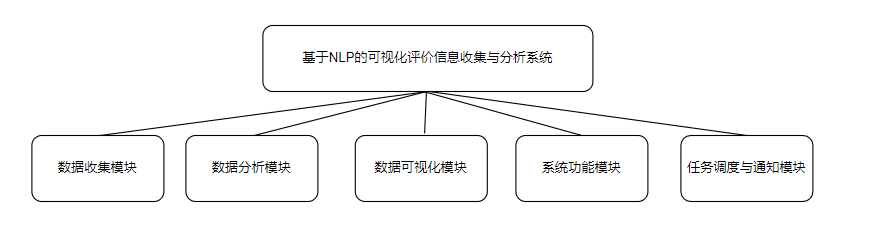


图3-3 系统功能图

## 3.4 系统的接口设计

用户身份验证：提供用户身份验证的API，接受用户的登录请求，检查用户提供的用户名和密码，验证用户身份的合法性。

数据收集：提供发送数据采集任务的API，接受用户的任务请求，由任务调度模块进行处理，调用相应的爬虫程序进行数据收集。

数据展示：提供返回对应任务所收集的数据，用于展示。

数据分析和处理：提供数据分析和处理的API，同样以任务的形式发送给workers,然后对数据进行预处理，再通过NLP等技术对系统收集的数据进行分析和处理，并返回分析结果，将其缓存在Redis中以加速访问。

用户设置和管理：提供用户设置和管理的API，包括用户信息的修改等操作，方便用户管理个人信息。

## 3.5 系统的错误处理

错误处理和错误信息输出：

基于NLP的可视化评价信息采集分析系统在数据采集、处理、可视化等各个运行阶段都可能遇到错误或故障。为确保用户能够有效地识别和解决这些问题，系统应提供清晰且信息丰富的错误消息以响应每个可能的错误或故障。

以下是系统可能输出的错误消息示例：

a)授权错误

表格：“登录失败：Invalid username or password”

含义：由于没有注册账户或者用户账号密码错误引起。

处理方法：系统会向用户显示错误信息。

b)注册错误

表格：“注册失败：Email already registered”

含义：该系统每一个邮箱只能注册一个账号，由于邮箱已经被注册，故无法注册。

处理方式：系统会向用户显示错误信息。

c)未授权错误

表格：“错误：无法生成可视化。请检查您的数据并重试。”

含义：由于数据或可视化设置存在问题，系统无法基于处理后的数据生成可视化。

处理方法：系统会向用户显示错误信息，提示用户检查数据和可视化设置后再试。

d)未授权访问错误

表单：“加载用户信息失败!请检查登录状态!”

含义：用户在未登录情况下尝试访问被保护的路由。

处理方法：系统会向用户显示错误信息。

e)数据加载错误

表格：“加载数据失败!”

含义：系统遇到意外错误或问题，导致其无法正常运行。

处理方式：系统会向用户显示错误信息。

一般来说，错误消息应该清晰、简洁且信息丰富，为用户提供足够的信息来理解问题并采取适当的措施。系统还应向用户提供有关他们为解决问题而采取的任何行动的反馈，例如在数据收集任务成功完成时显示成功消息。

# 4 系统的详细设计

详细设计提供了对每个组件应如何实现以及它们将如何相互交互的清晰理解。这有助于最大限度地减少错误并确保系统高效、可靠且易于维护。此外，详细的设计还可以帮助识别系统中的潜在问题或限制，这些问题或限制可以在系统部署之前解决。因此详细的设计对于任何复杂系统的成功开发和实施都是至关重要的。

## 4.1 系统接口设计

以下是系统的接口设计：

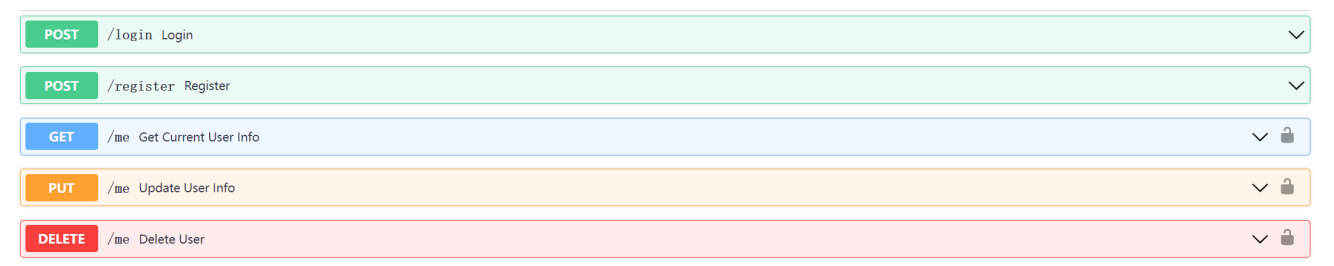


图4-1 用户登录注册及个人信息管理接口

如图4-1所示，共有5个接口。其中，/login用于用户登录，/register用于用户注册。/me的GET用于获取当前用户的信息，/me的PUT用于更新用户信息，/me的DELETE用于删除用户。

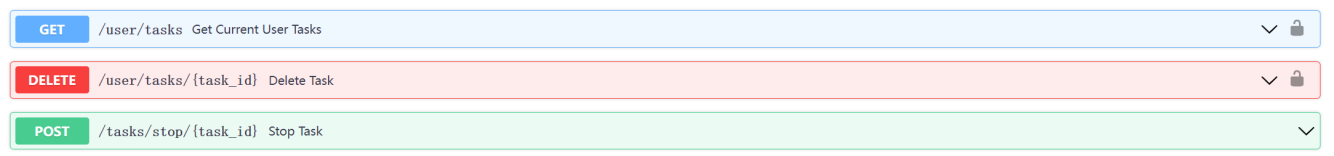


图4-2 任务管理接口

如图4-2所示，共有3个接口。其中，/user/tasks用于获取当前用户的任务列表，/user/task/{task\_id}用于删除任务。/tasks/stop/{task\_id}用于停止任务。



图4-3 微博爬虫任务接口

如图4-3所示，共有11个接口。其中，/tasks/stop/{task\_id}用于停止任务。/weibo/stop\_spider/{task\_id}用于停止爬虫任务，/weibo/resume\_spider/{task\_id}用于恢复爬虫任务。剩余的接口是用来发送任务给后端，每一个接口对应一种爬虫任务。



图4-4 微博爬虫数据接口

如图4-4所示，共有7个接口。其中，/weibo/data/user/{task\_id}用于获取收集到的微博用户信息。其余的接口功能与之相似，返回对应类型的数据。

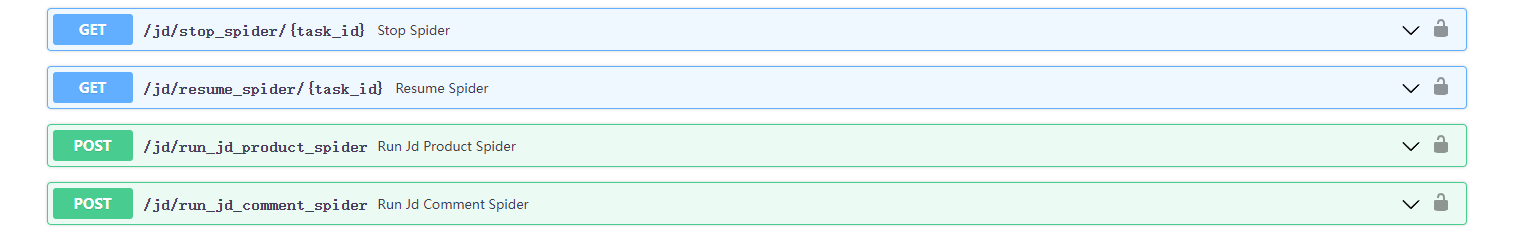


图4-5 京东爬虫任务接口

如图4-5所示，共有4个接口。其中，/jd/stop\_spider/{task\_id}用于停止爬虫，/jd/resume\_spider/{task\_id}用于恢复任务。剩余的2个接口用于发送京东爬虫任务。

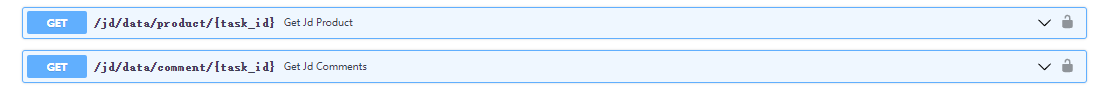


图4-6 京东数据接口

如图4-6所示，共有2个接口。分别用来获取京东商品信息和商品评价信息。

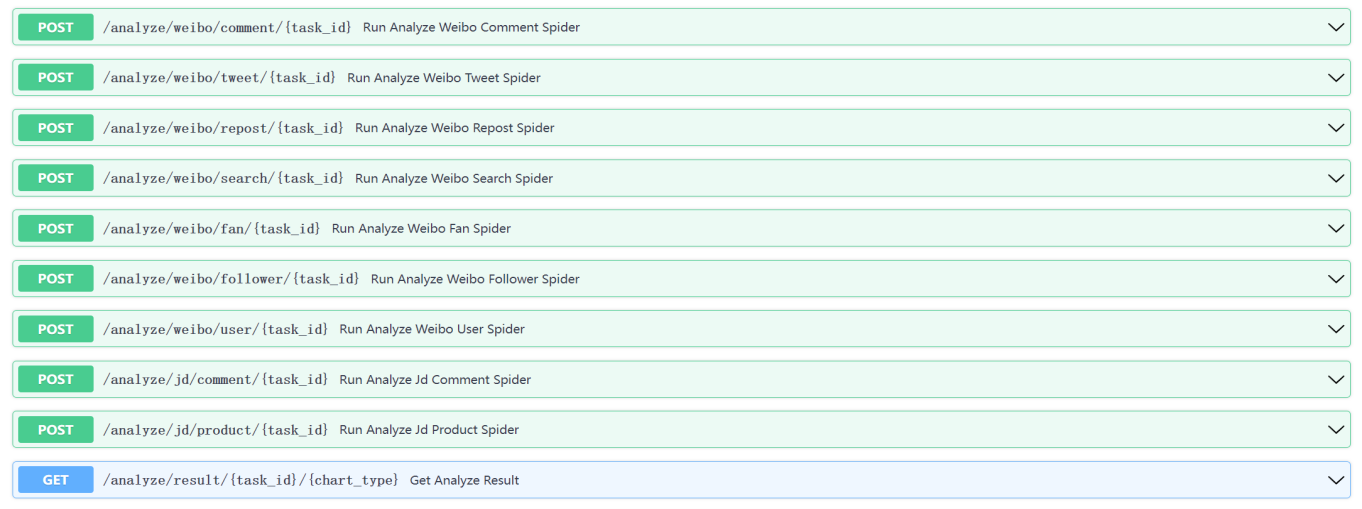


图4-7 分析任务接口以及分析结果返回接口

如图4-7所示，共有10个接口。其中，/analyze/result/{task\_id}/{chart\_type}用于获取某一任务号对应可视化类型的结果。剩余的9个接口用于发送分析任务，即将task\_id发送给后端，后端根据task\_id获取数据，然后进行分析处理。

## 4.2 系统功能模块设计

该部分分为用户登录、用户注册等模块。

### 4.2.1 用户登录模块设计



图4-8 用户登录界面

如图4-8所示，该登录页面包含了一个表单，其中包含了用户名和密码两个输入框，以及登录和注册按钮。当用户输入完用户名和密码后，点击登录按钮，触发submit事件，调用名为“login”的方法来进行登录操作。

### 4.2.2 用户注册模块设计



图4-9 用户注册界面

如图4-9所示，注册页面的模板代码也包含了一个表单，其中包含了用户名、邮箱和密码三个输入框，以及登录和注册按钮。当用户输入完注册信息后，点击注册按钮，触发submit事件，调用名为“register”的方法来进行注册操作。

### 4.2.3 其他系统功能设计

以下是系统的主界面：

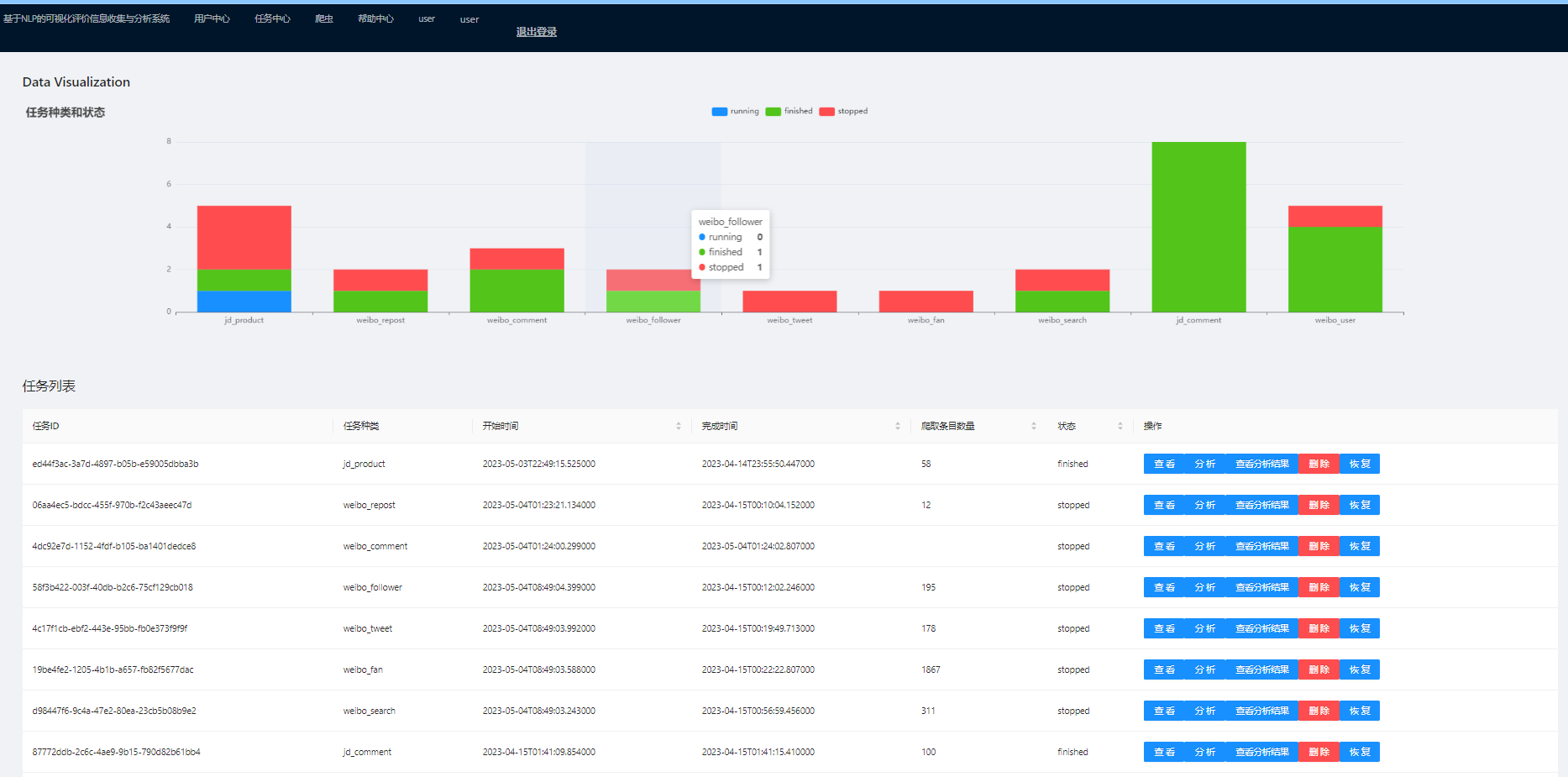


图4-10 主界面

如图4-10示，主界面包括以下部分：

导航栏：提供了主页、用户中心、任务中心、微博爬虫、京东爬虫和帮助中心等功能链接；

数据可视化：展示了通过数据可视化的方式分析评价信息的结果；

任务列表：展示了用户创建的所有任务列表，并提供查看、分析、查看分析结果和删除等功能按钮。其中，任务类型包括微博和京东两大类爬虫任务。这两大类又各自有多种爬虫任务。

## 4.3 系统数据收集模块设计

以下是系统数据收集模块的UI设计：

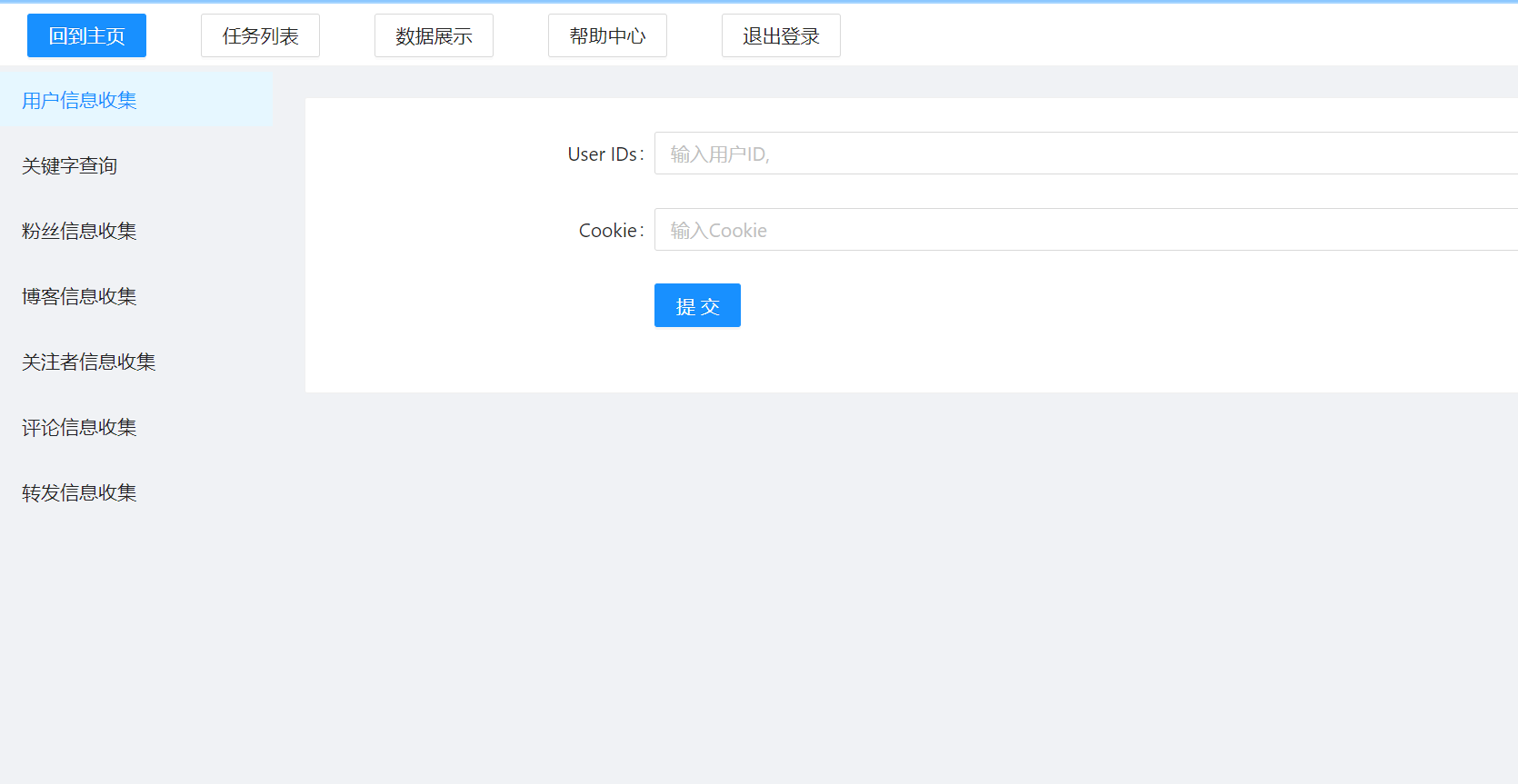


图4-11 微博爬虫界面

如图4-11示，实现微博数据收集模块的用户界面。该模板包含了一个顶部导航栏和一个左侧菜单栏，以及一个主要内容区域。左侧菜单栏提供了多个选项，分别对应不同的数据收集任务，主要内容区域根据所选任务显示不同的表单，用于输入相应的参数和提交任务。在主要内容区域下方，还有一个可选的数据展示组件，用于显示收集到的数据。该界面，用于配置和启动不同类型的爬虫任务。其中，每个菜单项对应不同的爬虫类型，如用户信息收集、关键字查询、粉丝信息收集等。每个菜单项下面的表单，用于配置对应爬虫任务的输入参数，例如指定要收集的用户ID、搜索关键字、起始时间、结束时间、爬取的页数等等。在表单中填入对应的参数之后，点击提交按钮即可启动对应的爬虫任务。



图4-12 京东爬虫界面

如图4-12示，该界面包含了一个顶部导航栏和一个左侧菜单栏，以及一个主要内容区域。左侧菜单栏提供了两个选项，分别对应产品信息收集和评价信息收集，主要内容区域根据所选任务显示不同的表单，用于输入相应的参数和提交任务。在主要内容区域下方，还有一个可选的数据展示组件，用于显示收集到的数据。整个模板使用了Ant Design Vue组件库提供的多个组件，包括导航菜单组件、布局组件、卡片组件、表单组件等。同时，使用了Vue.js模板语法的多种指令和事件绑定方式，实现了组件之间的动态交互。对于产品信息收集的表单，用户可以输入要搜索的产品名称，然后提交任务。对于评价信息收集的表单，用户可以输入要爬取的产品链接和需要爬取的页数，然后提交任务。提交任务的事件会触发爬虫程序开始工作，收集相应的数据。

## 4.4 系统数据分析模块设计

具体处理过程不在前端展示，在前端只展示分析结果，以及提交分析任务的接口。

该模块应包括以下设计：

数据预处理：在分析数据之前，需要对其进行预处理。这包括通过删除不相关的信息来清理数据、规范化文本数据以及将数据转换为易于分析的结构化格式。预处理模块还应包括检测和纠正数据错误的方法。

特征提取：特征提取涉及从将用于分析的预处理数据中识别最相关的特征。这包括词频分析、情感分析和主题建模等方法。

数据可视化：分析结果应以易于理解的格式呈现。可以使用图表、图形和热图等数据可视化工具来呈现结果。

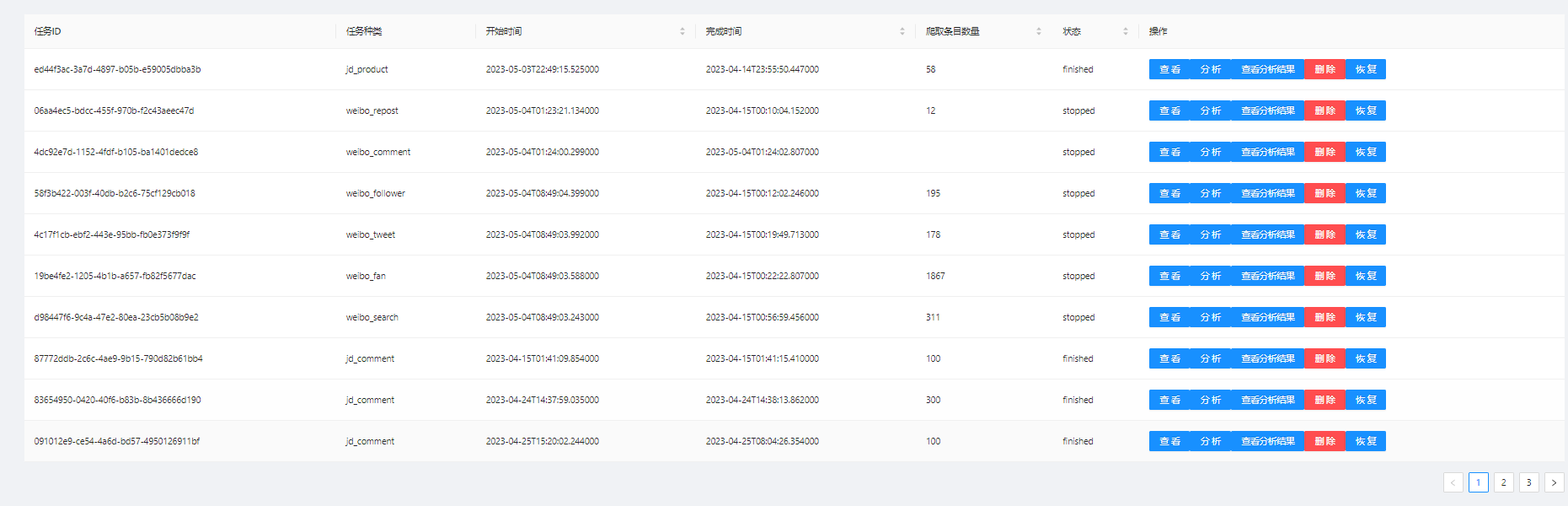


图4-13 数据分析模块的前端交互界面

如图4-13所示，任务类别中每一行有一个分析按钮，通过点击该按钮，前端将会通过api将分析任务发送给后端，由后端进行处理。点击查看分析结果按钮将会跳转到可视化界面。

## 4.5 系统数据可视化模块设计

如图4-13所示的界面，查看分析结果按钮用于跳转到对应任务类型的可视化模块。

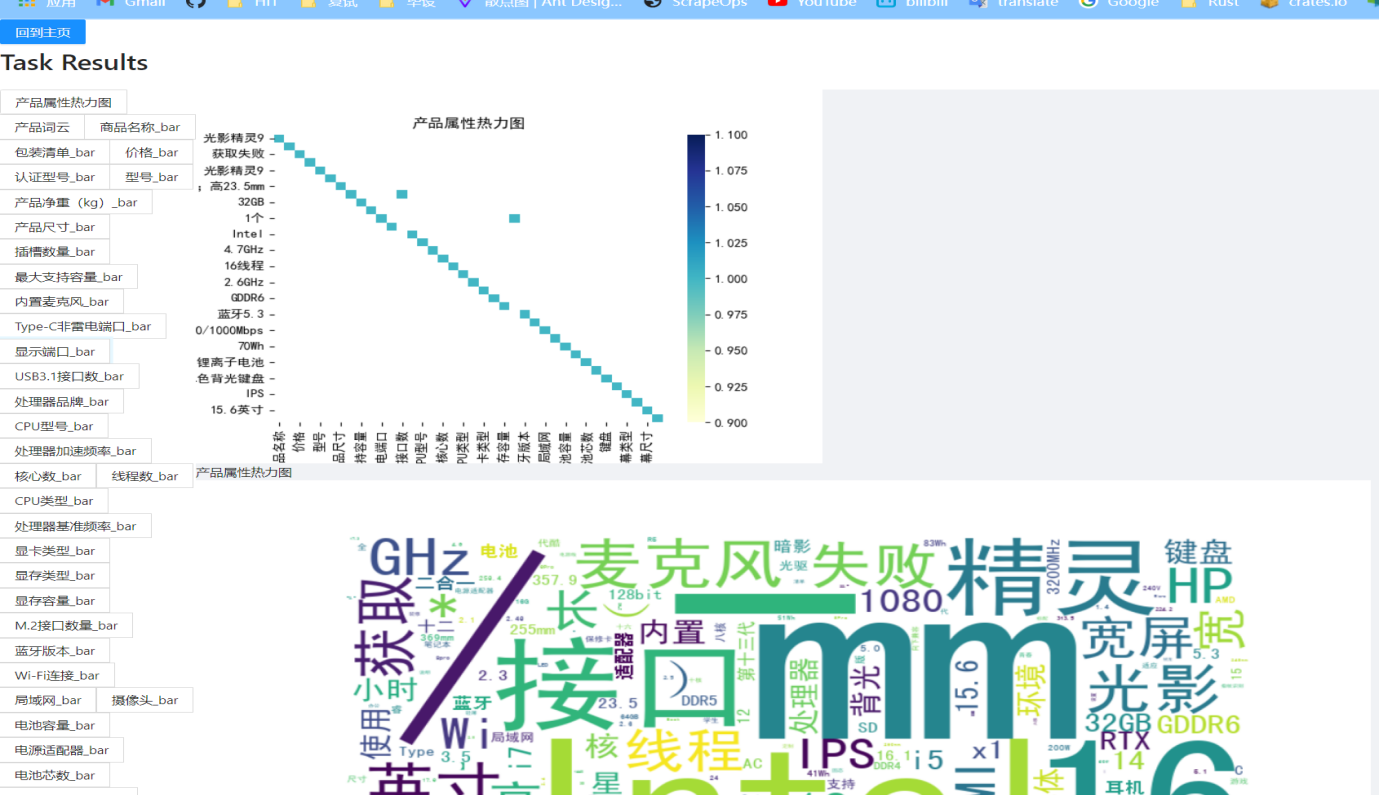


图4-14 京东商品信息可视化界面

如图4-14所示的界面，右端有多种选择，每点击一个，将会在右端显示对应结果。如产品属性热力图、产品词云等等。



图4-15 京东商品评价可视化界面

如图4-15所示的界面，右端有多种选择，每点击一个，将会在右端显示对应结果。如产品属性热力图、产品词云等等。

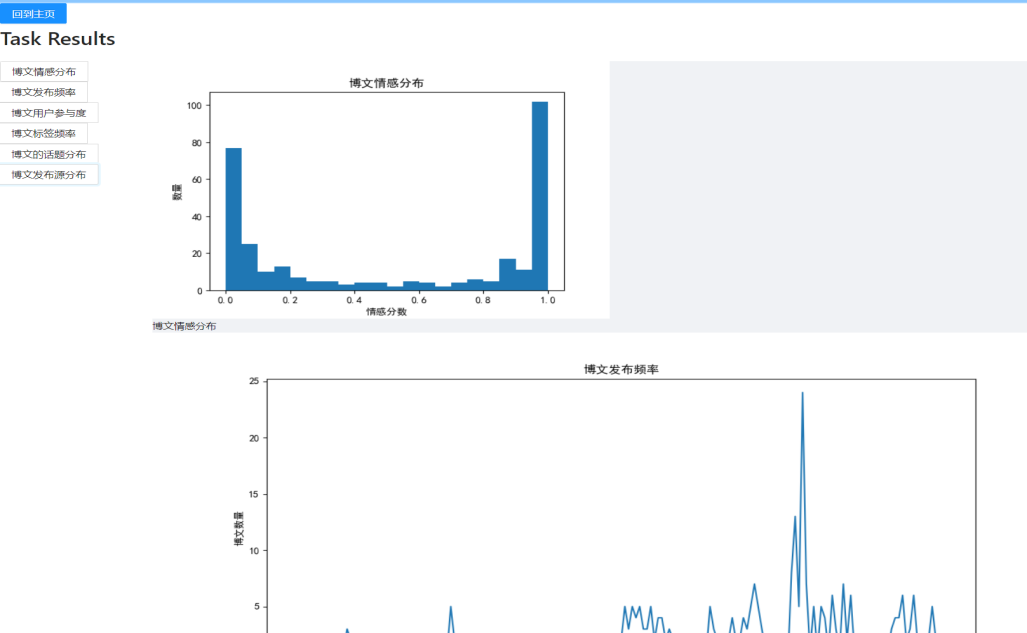


图4-16 微博博文信息可视化界面

如图4-16所示的界面，右端有多种选择，每点击一个，将会在右端显示对应结果。如产品属性热力图、产品词云等等。

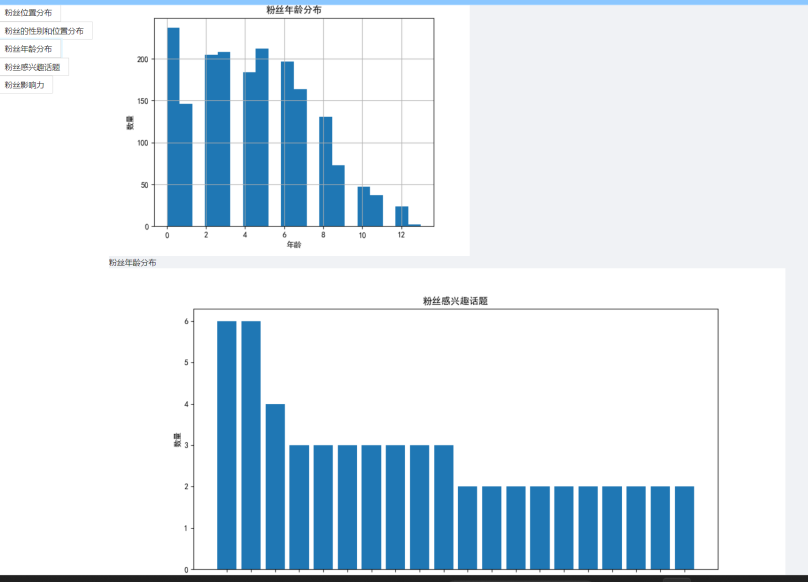


图4-17 微博粉丝信息可视化分析界面

如图4-17所示的界面，右端有多种选择，每点击一个，将会在右端显示对应结果。如产品属性热力图、产品词云等等。

在每个可视化界面中，都会展示任务结果。页面由一个标题和一个回到主页的按钮组成。其余部分是一个由选项卡和图片组成的布局。选项卡由一个a-radio-group组件实现，每个选项卡有一个标题。当选项卡切换时，会展示相应的图片。图片由一个v-for循环渲染，图片的地址和标题存储在Redis中，数据由请求后端得到。在每张图片的下方，有一个标签展示对应的选项卡的标题。

## 4.6 系统任务调度与通知模块设计

通知模块包含提示用户的各种信息。



图4-18 获取用户信息成功通知

如图4-18所示，当登录成功到主界面时，将会获取用户信息，因此对用户进行通知。



图4-19 更新用户信息成功通知

如图4-19所示，在更新用户信息成功后进行通知。



图4-20 登录成功通知

如图4-20所示，在登录成功时进行提示。



图4-21 加载数据成功通知

如图4-21所示，在主界面中的任务列表中点击某一已收集数据的任务的查看按钮时，将会跳转到数据显示界面，如加载数据成功将会通知。



图4-22 加载任务列表成功通知

如图4-22所示，加载任务列表成功时将会通知。



图4-23 产品信息收集任务提交成功通知

如图4-23所示，在爬虫任务界面中的产品信息收集选项时，点击提交，成功提交时将会通知。

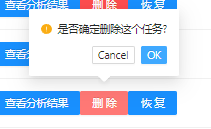


图4-24 任务删除提示

如图4-24所示，当在任务列表中点击删除按钮时，系统将会弹出该对话框，以防止用户误操作。

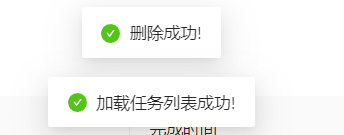


图4-25 任务删除成功提示

如图4-25所示，在确认删除后，系统将会重新加载任务列表，故系统会提示删除成功以及加载任务列表成功。

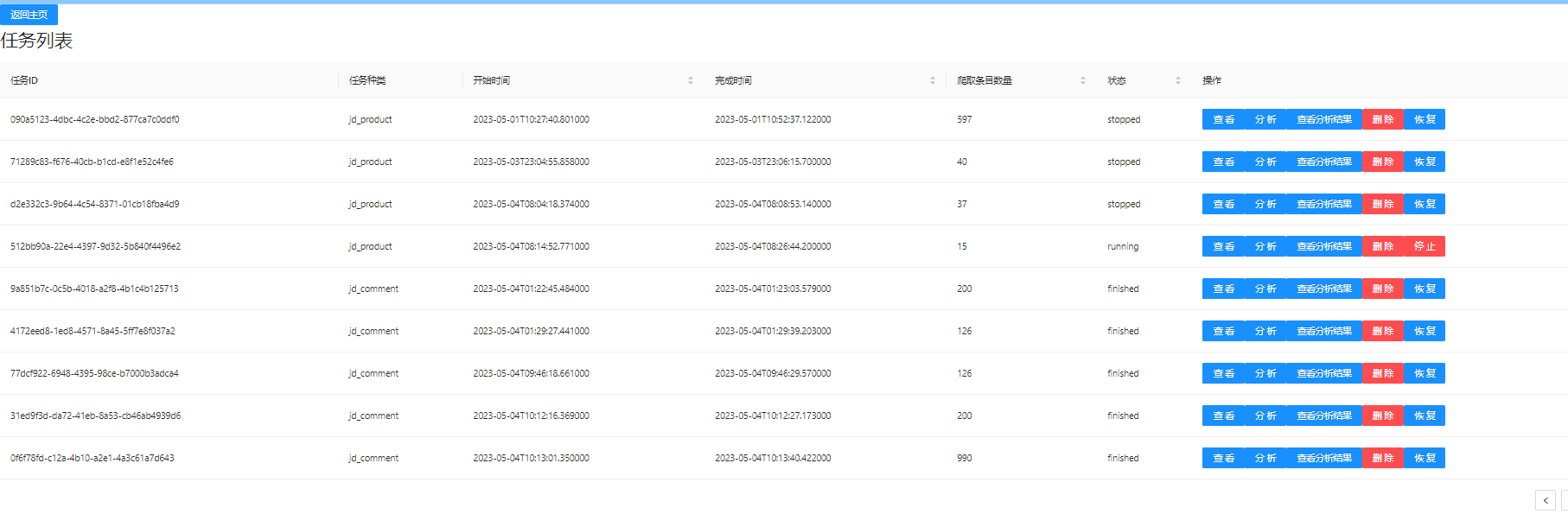


图4-26 任务管理界面

如图4-26所示的界面，可以在界面中进行多种操作以对任务进行管理。

具体的任务调度由后端调用Celery框架完成。

# 5 系统实现

## 5.1 系统接口的实现

系统后端实现FastAPI框架。

logging.basicConfig(level=logging.DEBUG)

app = FastAPI()

app.include\_router(auth\_router)

app.include\_router(user\_router)

app.include\_router(weibo\_router)

app.include\_router(jd\_router)

app.include\_router(analyze\_router)

app.include\_router(protected\_router)

这段代码将会包含其他的路由，比如：auth\_router里是登录、注册接口的实现。user\_router里是用户信息管理和任务管理的接口实现。weibo\_router里是微博爬虫任务以及数据相关接口。jd\_router里是京东爬虫任务以及数据相关接口。analyze\_router里是分析微博和京东数据的接口。protected\_router里是受到保护的接口示例。

## 5.2 系统登录与注册模块的实现

下面这段代码是使用Python编写的FastAPI框架的后端代码。它实现了登录功能的API接口。

@router.post("/login")

async def login(form\_data: OAuth2PasswordRequestForm = Depends(), db: Session = Depends(get\_db)):

access\_token = create\_access\_token()

return {"access\_token": access\_token, "token\_type": "bearer"}

/login路由处理用户的登录请求。它从数据库中查询给定用户名的用户，如果用户不存在或密码验证失败，则抛出HTTP异常。如果验证成功，则为用户生成一个访问令牌(access\_token)并返回给客户端。

下面这段代码是使用Python编写的FastAPI框架的后端代码。它实现了注册功能的API接口。

@router.post("/register")

async def register(user\_create: UserCreate, db: Session = Depends(get\_db)):

if user: raise HTTPException(status\_code=400, detail="邮箱已被注册")

hashed\_password = hash\_password(user\_create.password)

new\_user = User(username=user\_create.username, email=user\_create.email, password=hashed\_password)

db.add(new\_user)

return new\_user

/register路由处理用户的注册请求。它首先检查给定的电子邮件地址是否已经在数据库中注册，如果是，则抛出HTTP异常。否则，它对用户的密码进行哈希处理并将新用户添加到数据库中。最后，它返回新用户的详细信息。

这些路由都需要一个数据库连接作为依赖项。get\_db()函数返回一个新的数据库会话(Session)。如果有必要，它将创建一个新的数据库引擎(Engine)和一个新的数据库会话(Session)，并在每个请求结束时关闭数据库连接。

## 5.2 系统数据收集模块的实现

### 5.2.1 后端实现

代码见附录1.2。

数据收集模块主要使用FastAPI、Scrapy、Celery框架。

在task.py中定义相关的任务，

@celery.task(name='tasks.run\_weibo\_search\_spider', bind=True)

def run\_weibo\_search\_spider(self):

def stop\_task(signum, frame):

reactor.callFromThread(reactor.stop)

signal.signal(signal.SIGTERM, stop\_task)

reactor.run()

return {'message': f'Spider search finished running.'}

例如这段代码，定义了一个名字叫做run\_weibo\_search\_spider的任务，此函数接受一些关键字参数，包括要搜索的关键词，起始和结束时间范围，是否按热度排序以及是否使用特定的时间范围进行搜索。函数会启动一个 Scrapy 爬虫并执行搜索任务。

该函数执行以下操作：

首先获取 Celery 任务的 ID，并初始化一个名为 SearchSpider 的 Scrapy 爬虫类实例。根据传递的参数设置爬虫的搜索关键词、时间范围、Cookie 等属性，配置 Scrapy 的设置，然后运行 CrawlerRunner 对象。启动爬虫，并等待爬虫运行结束。再添加一个信号处理器，以便在 Celery worker 终止时停止 Reactor。最后返回一个字典，指示爬虫搜索完成的信息。

对于其他类型的Scrapy爬虫代码相似，变化的是需要传递不同的参数。

在定义了Celery任务后，就可以在FastAPI的endpoint中发送任务。以下是一个示例：

@router.post('/weibo/run\_weibo\_search\_spider')

async def run\_weibo\_search\_spider(search\_data: dict = Body(...)):

task = celery.send\_task('tasks.run\_weibo\_search\_spider', kwargs={

'keywords': keywords,

})

return {'task\_id': task.id}

这段代码定义了一个路由，使用了 FastAPI web 框架，用于运行一个爬虫，从微博搜索中爬取数据。这个路由定义为一个 POST 请求，需要一个 JSON 类型的请求体，包含了搜索关键字、起始时间、结束时间、是否按热度排序、是否在指定时间范围内搜索和 Cookie 等数据。

代码接着将这些数据通过一个 Celery 任务队列异步地发送给一个名为 “tasks.run\_weibo\_search\_spider”的任务。任务逻辑在“tasks.py”文件中定义。

任务 ID 和其他任务相关信息会被存储在一个 MongoDB 数据库中，使用了 PyMongo 库来连接 MongoDB。任务 ID 会在响应中返回。

为了确保安全性，这段代码使用了一个名为“get\_current\_user”的依赖项，来获取当前用户的身份信息。然后代码将消息发送到 Celery 任务队列以异步执行任务。 该任务在单独的 Python 模块“tasks.py”中定义，其中包含用于抓取用户数据的逻辑。

任务 ID 和有关任务的其他信息使用 PyMongo 存储在 MongoDB 数据库中，PyMongo 是一个用于与 MongoDB 交互的 Python 库。任务 ID 作为对 POST 请求的响应返回。

### 5.2.2 前端实现

以下只给出Search Spider的前端组件的部分代码。

<a-input v-model:value="searchData.keywords" placeholder="输入关键字," />

<a-button type="primary" html-type="submit">提交</a-button>

<component v-if="showData === true" :is="Component"></component>

在具体的代码实现中，需要构建一个包含侧边栏和主要内容区域的页面布局。侧边栏包含一个菜单，其中有两个菜单项：用户信息收集和关键字查询。当用户点击菜单项时，会显示对应的内容在主要内容区域中。

当用户选择“关键字查询”菜单项时，一个表单会被渲染，其中包含关键字、起始时间、结束时间、是否根据热度排序、是否指定时间范围和Cookie等字段。当用户提交表单时，将会发送一个POST请求到后端的/weibo/run\_weibo\_search\_spider接口，并将表单数据作为JSON数据发送。如果成功，将会在控制台中输出响应。如果出现错误，将会在页面上显示错误消息。

## 5.3 系统数据分析模块的实现

代码见附录1.3。

本系统主要分为微博数据分析和京东数据分析模块。

分析功能同样由FastAPI将根据用户的请求发送任务给Celery。以下是一个示例：

@router.post('/analyze/weibo/comment/{task\_id}')

async def run\_analyze\_weibo\_comment\_spider(task\_id: str):

task = celery.send\_task('tasks.run\_analyze\_weibo\_comment', kwargs={

'task\_id': task\_id,

})

return {'task\_id': task.id}

这段Python代码创建了一个FastAPI路由器端点，用于分析微博评论。它使用了Celery分布式任务队列来运行后台任务。当发送一个包含任务ID的POST请求时，它会从MongoDB数据库中检索任务类型和数据，并将任务发送到Celery任务队列中执行。函数返回Celery任务的ID，可以用来监视任务的进度和检索结果。

对于其他的任务，代码类似。

以下是Celery一个task的代码：

@celery.task(name='tasks.run\_analyze\_weibo\_comment', bind=True)

def run\_analyze\_weibo\_comment(self, task\_id):

run\_weibo\_comment\_analyze(task\_id)

return {'message': f'Analyze Weibo Comment finished running.'}

这段代码定义了一个名为run\_analyze\_weibo\_comment的Celery任务，用于分析微博评论。当任务在Celery任务队列中运行时，它会调用run\_weibo\_comment\_analyze函数来执行分析。函数返回一个包含一个消息的字典，表示任务已完成运行。bind=True表示该任务将绑定到任务类实例，这使得可以在任务函数中访问任务实

run\_weibo\_comment\_analyze()等相似命名风格的函数都使用task\_id作文参数，隐藏具体的实现细节。

### 5.3.1 微博数据分析模块

代码见附录1.3.1。

微博评论预处理代码：

def remove\_emoticons(text):

return re.sub(r'\[.\*?\]', '', text)

def preprocess\_weibo\_comment\_data(data):

df = pd.DataFrame(data)

word\_counts = pd.Series(np.concatenate(df['content'].values)).value\_counts()

return df, word\_counts[:50]

以上这段代码是用来预处理微博评论数据的。它包含了两个函数，一个是用正则表达式删除文本中的表情符号，另一个是对微博评论数据进行处理的主要函数。

主要函数将数据转化为Pandas DataFrame，并对其中的created\_at列进行日期解析。接着，将like\_counts列中的字符串转化为整数。然后，对content列进行分词和词频统计，使用jieba库进行中文分词，并用Pandas中的value\_counts函数计算每个词语的出现次数。最后，返回处理后的数据和词频统计结果，其中只包含出现次数最多的前50个词语。

微博博文处理与分析代码：

def analyze\_weibo\_data(task\_id):

for d in weibo\_data:

if 'content' in d:

content = d['content']

blob = SnowNLP(content)

sentiment\_list.append(blob.sentiments)

sentiment\_df = pd.DataFrame(sentiment\_list, columns=['情感得分'])

这段代码是用来分析微博数据的。它会连接到数据库中的指定任务ID，获取微博数据，对微博内容进行情感分析。

代码首先连接到数据库中的指定任务ID，并获取微博数据。然后，对每条微博的内容进行情感分析，并将得分存入另一个Pandas DataFrame中。

年龄分布功能相关代码：

fan\_info\_df['account\_age'] = (datetime.now() - pd.to\_datetime(fan\_info\_df['created\_at'])).apply(lambda x: x.days // 365)

fan\_info\_df['account\_age'].hist(bins=20)

这段代码的功能是计算每个粉丝账号的年龄，并将其添加为一个新的列到fan\_info\_df数据帧中，名为account\_age。计算账号年龄的方法是使用当前日期时间与每个粉丝账号创建日期时间的差值，然后将差值转换为整数天数并除以365（取整数部分），这样就得到了账号的年龄（按整年计算）。

兴趣分布相关代码：

fan\_desc = fan\_info\_df['description'].dropna().str.cat(sep='|')

words = re.findall(r'\w+', fan\_desc)

word\_count = pd.Series(words).value\_counts()

top\_words = word\_count[:20]

这段代码的功能是从fan\_info\_df数据帧的description列中提取粉丝的兴趣，并统计出现频率最高的前20个单词。

首先，代码使用dropna()函数删除description列中的缺失值（NaN），并使用str.cat()函数将所有非缺失的description值连接成一个长字符串，其中用竖线符号'|'分隔。这样可以将多个描述合并为一个字符串进行处理。

然后，代码使用正则表达式的re.findall()函数从字符串中提取所有的单词，并将它们存储在一个列表words中。正则表达式\w+匹配一个或多个连续的字母数字字符（即单词），以提取字符串中的单词。

接着，代码使用pd.Series()函数将words列表转换为一个pandas系列，并使用value\_counts()函数计算每个单词出现的频率。最后，代码使用切片操作[:20]选取出现频率最高的前20个单词，并将它们存储在top\_words变量中。这个变量可以用于分析粉丝的主要兴趣领域，从而更好地了解他们的需求和偏好。

### 5.3.2 京东数据分析模块

代码见附录1.3.2。

A.京东商品数据分析代码

a)analyze\_attributes函数

b)get\_attribute\_counts函数

c)create\_wordcloud函数

以上代码是一个对京东商品的数据分析代码。包含了三个函数：analyze\_attributes()、create\_wordcloud()和analyze\_attribute\_bars()。这些函数的作用是提取商品的属性和评论信息，通过可视化的方式展示出来。

analyze\_attributes()函数用于提取商品的属性信息并为将其可视化为一个热力图进行分析工作。

create\_wordcloud()函数用于提取商品评论中的关键词。

analyze\_attribute\_bars()函数用于提取商品的属性值。

这些函数都使用了jd\_db中的数据，其中jd\_db是一个对MongoDB数据库的访问器。

B.京东商品评价分析代码：

def clean\_data(data):

for item in data:

content = re.sub('<[^<]+?>', '', content)

content = re.sub('[^\w\s]', '', content)

cleaned\_data.append(content)

return cleaned\_data

def tokenize\_data(cleaned\_data):

for content in cleaned\_data:

words\_list.append(' '.join(words))

return words\_list

这段代码会从数据库中获取数据，并对数据进行清理、分词和统计词频等预处理操作，然后使用WordCloud库生成词云，并将生成的图片保存到缓存中。

代码使用MongoDB连接库从指定任务ID的数据库中获取数据。接着，对数据进行清理，去除HTML标记和特殊字符。然后，对每条数据进行分词，并去除停用词和单个字符的词语，然后将所有分词结果存入一个列表中。接下来，使用Counter库统计每个词语出现的次数，并返回统计结果。

## 5.4 系统数据可视化模块的实现

本系统主要分为微博数据和京东数据可视化模块。

这些模块的功能依赖于数据分析模块。主要代码实现在前端。

前端会根据用户的输入请求存储在Redis缓存中的分析结果。

以下是FastAPI的一个endpoint:

@router.get("/analyze/result/{task\_id}/{chart\_type}")

async def get\_analyze\_result(task\_id: str, chart\_type: str):

chart\_type = unquote(chart\_type)

redis\_key = f"{task\_id}\_{chart\_type}"

image\_data = redis\_client.get(redis\_key)

return Response(content=image\_data, media\_type="image/png")

这段代码是一个Python的FastAPI框架的路由函数，定义了一个GET请求的接口。接口的路径为“/analyze/result/{task\_id}/{chart\_type}”，其中“{task\_id}”和“{chart\_type}”是路径参数，分别表示任务ID和图表类型。

在函数体内部，首先对“{chart\_type}”进行了解码操作，然后通过拼接“{task\_id}”和“{chart\_type}”生成了一个redis\_key。接着，代码判断redis数据库中是否存在该redis\_key对应的数据。如果不存在，则返回一个包含错误信息的字典。如果存在，则从redis中获取该redis\_key对应的图像数据，并将其封装成一个响应对象返回给调用者，响应对象的媒体类型为“image/png”。

### 5.4.1 微博数据可视化模块

以下代码是微博数据可视化模块的一部分：

<a-radio-group class="tabs" v-model="activeName" @change="(e) => handleTabClick(e.target.value)">

<a-radio-button v-for="tab in tabs" :key="tab.key" :value="tab.key">{{tab.title}}</a-radio-button></a-radio-group>

<div v-for="tab in visibleImageTabs" :key="tab.key" class="image">

<img :src="images[tab.key]" :alt="tab.title" />

<div class="image-label">{{ tab.title }}</div>

</div>

这段代码是一个 Vue 组件，它展示了一个任务的结果。代码中使用了 Ant Design Vue UI 库中的组件，如 a-button、a-layout、a-radio-group、a-radio-button 等。在 data 中定义了 images 和 tabs 两个变量，分别存储了选项卡的图像数据和选项卡的信息。在 visibleTabs 中，定义了哪些选项卡是可见的。在 methods 中，定义了 handleTabClick 方法，当用户点击某个选项卡时，该方法会检查该选项卡是否已加载，如果没有，则从服务器获取该选项卡的图像数据，并将其存储在 images 中。代码中使用了 Axios 库来从服务器获取数据，并使用了 Base64 编码将图像数据转换为字符串。

这段代码还使用了 Vue 3 的新特性 setup()，它用于组件的数据、方法等的初始化和声明。在 setup() 中，使用了 Vue 3 中的响应式 API ref 来定义了 activeName 变量，该变量用于控制当前选中的选项卡的值。

在组件的模板中，使用了 Vue 的模板语法来展示组件的内容，包括选项卡、图像等。在用户点击某个选项卡时，handleTabClick 方法会被触发，该方法会改变该选项卡的可见性，并从服务器获取该选项卡的图像数据。在组件的生命周期函数 mounted() 中，获取了该任务的 ID，并存储在组件实例中。

对于其他的部分，代码与其实现基本相似。有区别的地方在于tabs和visibleTabs变量的设置。

### 5.4.2 京东数据可视化模块

以下是数据可视化模块的主要部分，其与微博数据可视化模块较为相似。

该模块中有一个Vue组件，它包含了一个页面展示任务结果的布局。在页面中，有一个按钮用于回到主页，一个标题“Task Results”，以及一个由两个选项卡组成的布局。每个选项卡都对应一个图像和标签。用户可以点击选项卡切换图像和标签。

在组件中，定义了一些数据和方法。数据包括：图像、选项卡、可见选项卡、产品数据等。方法包括：处理选项卡点击事件、异步获取图像数据等。通过异步获取图像数据，可以实现延迟加载，提高页面加载速度。同时，使用了第三方库ant-design-vue中的message组件，用于在页面中显示加载成功的消息。

在mounted钩子中，使用了axiosInstance发送请求获取数据，然后根据获取的数据动态生成选项卡和可见选项卡。通过setup函数和ref函数，定义了一个响应式数据activeName，用于保存当前选中的选项卡。使用computed属性visibleImageTabs过滤出当前可见的选项卡，然后在页面中渲染图像和标签。

## 5.5 系统任务调度与通知模块的实现

### 5.5.1 系统任务调度模块的实现

代码见附录1.5.1。

系统任务调度模块主要由Celery框架负责。

Celery 通过代理 URL 与 RabbitMQ 通信，代理 URL 指定 RabbitMQ 服务器的位置和凭据。 当任务提交给 Celery 时，它会作为消息发送给 RabbitMQ。 然后根据任务的类型和其他标准将消息路由到特定队列。

Worker 是执行任务的 Celery 进程，监听特定队列上的消息。当工作人员收到消息时，它会处理任务并将结果作为另一条消息发送回 RabbitMQ。然后根据任务 ID 将结果消息路由到特定队列，客户端可以从队列中检索结果。

RabbitMQ 提供了多种特性，使其成为 Celery 的理想消息代理，包括支持多种消息交换模式、消息确认和持久队列。 这些功能使 Celery 能够在工作集群中高效可靠地分配任务，即使在出现故障和网络中断时也是如此。

Celery是一个Python分布式任务队列，可以实现任务调度的功能。Celery通过消息中间件，如RabbitMQ或Redis，进行任务调度和处理，支持异步任务和定时任务调度。以下是Celery任务调度的具体实现：

配置Celery

在使用Celery进行任务调度之前，需要先配置Celery。需要定义一个celery实例，指定任务队列和消息中间件等信息。例如：

from app.config.config import BROKER\_URL, BACKEND\_URL

celery = Celery('tasks', broker=BROKER\_URL, backend=BACKEND\_URL)

celery.conf.task\_routes = {'celery\_task.tasks.\*': {'queue': 'celery'}}

在上面的代码中，创建了一个名为celery的Celery实例，并指定了消息中间件为config中的BROKER\_URL，结果后端为config中的BACKEND\_URL。

定义任务

在Celery中，任务是通过Python函数来定义的。定义任务时，需要使用@app.task装饰器来将函数转换为Celery任务。例如：

@celery.task(name='tasks.run\_jd\_comment\_spider', bind=True)

def run\_jd\_comment\_spider(self, urls=None, pages=None):

settings = get\_project\_settings()

runner = CrawlerRunner(settings)

deferred = runner.crawl(JDcommentspider, \*\*spider\_kwargs)

signal.signal(signal.SIGTERM, stop\_task)

reactor.run()

return {'message': f'Spider JD Comment Spider finished running.'}

这段代码定义了一个Celery任务函数 run\_jd\_comment\_spider，用 @celery.task 装饰器修饰。这个任务函数接收两个参数 urls 和 pages，并返回一个包含任务执行结果的字典。在任务函数中，首先获取任务ID（task\_id = self.request.id）并定义爬虫类（spider\_cls = JDcommentspider）。接着，根据传入的参数，构建一个字典 spider\_kwargs，并将它作为关键字参数传递给爬虫类的构造函数。为Scrapy设置环境变量，指定Scrapy配置文件的位置（os.environ['SCRAPY\_SETTINGS\_MODULE'] = 'jdspider.settings'）。调用 get\_project\_settings 函数获取Scrapy项目的设置，然后创建一个 CrawlerRunner 实例（runner = CrawlerRunner(settings)）。通过 CrawlerRunner.crawl 方法启动爬虫，并传入爬虫类和关键字参数。在任务函数中定义了一个 stop\_reactor 函数，用来停止 reactor 事件循环。将 stop\_reactor 函数添加到爬虫运行结束后的回调函数中，以确保爬虫运行结束后停止 reactor 事件循环。在任务函数中定义了一个 stop\_task 函数，用来在收到终止信号时停止 reactor 事件循环。使用 signal 模块将 stop\_task 函数注册为Celery worker终止信号的处理函数。

最后，调用 reactor.run() 函数启动 reactor 事件循环，等待爬虫运行结束。在爬虫运行结束后，任务函数会返回一个包含任务执行结果的字典。

在上面的代码中，定义了一个名为add的任务，该任务接受两个参数x和y，并返回它们的和。

调度任务

在Celery中，可以使用apply\_async方法来调度任务。apply\_async方法可以接受多个参数，包括任务名、任务参数、任务选项等。例如：

from datetime import timedelta

run\_jd\_comment\_spider.apply\_async(args=(1, 2), countdown=3)

run\_jd\_comment\_spider.apply\_async(args=(3, 4), eta=datetime.now() + timedelta(seconds=10))

在上面的代码中，使用apply\_async方法调度了两个run\_jd\_comment\_spider任务，分别在3秒后和10秒后执行。

定时任务

Celery也支持定时任务调度，可以使用Celery Beat来实现。Celery Beat是Celery的一个内置组件，它可以定时地执行Celery任务。要使用Celery Beat，需要在Celery配置中指定调度器和调度任务。例如：

from celery.schedules import crontab

app.conf.beat\_schedule = {

'add-every-minute': {

'task': 'tasks. run\_jd\_comment\_spider ',

'schedule': crontab(),

'args': (1, 2),

},

}

在上面的代码中，定义了一个名为jd-comment-spider-every-minute的调度任务，该任务每分钟执行一次run\_jd\_comment\_spider任务，传递参数1和2。

以上就是Celery任务调度的具体实现。通过配置Celery、定义任务、调度任务和使用Celery Beat实现定时任务，可以实现各种任务调度需求。

### 5.5.2 系统通知模块的实现

代码见附录1.5.2。

系统通知模块的主要功能是向用户发送通知，包括系统事件通知、任务状态变化通知等。这些通知可以是文本消息等形式，通常会以弹窗、邮件等方式提醒用户。通过系统通知模块，可以让用户及时了解系统中重要事件的发生，提高用户的工作效率和体验。

邮箱通知功能的主要代码：

class EmailNotificationMiddleware(object):

def process\_spider\_exception(self, response, exception, spider):

self.send\_email\_notification(exception)

def send\_email\_notification(self, exception):

subject = 'Scrapy notification'

body = str(exception)

self.mailer.send(to=settings.get('MAIL\_RECIPIENTS'), subject=subject, body=body)

这段代码是一个Scrapy中间件，名为EmailNotificationMiddleware，用于在Scrapy爬虫程序中发生异常时发送电子邮件通知。它的功能包括：

在初始化时，从Scrapy的settings中获取邮件发送配置信息，并通过这些配置创建一个邮件发送器（即MailSender对象）。

当Scrapy爬虫程序中出现异常时，调用process\_spider\_exception方法。这个方法接收三个参数：响应对象response、异常对象exception和当前Spider对象spider。在这个方法中，会调用send\_email\_notification方法来发送电子邮件通知。

send\_email\_notification方法会将异常信息转化为字符串，并通过之前创建的邮件发送器发送电子邮件通知。通知的主题为“Scrapy notification”，收件人列表则从Scrapy的settings中获取（即MAIL\_RECIPIENTS配置项）。

其他通知功能在后端接口以及前端中实现。

比如通过 message 对象来实现通知功能。在这个功能中，当任务列表成功加载后，会弹出一个短暂的成功提示框，显示“加载任务列表成功!”这个信息，并且持续时间为0.3秒。如果加载任务列表失败，提示框会显示“加载失败!(当前用户token过期)”这个错误信息，并且持续时间为0.5秒。无论加载成功还是失败，都会在最终结束时将页面中的“loading”状态设置为false，表示加载状态已经结束。其他地方的通知功能与上面的代码相似。

## 5.6 系统Docker部署的实现

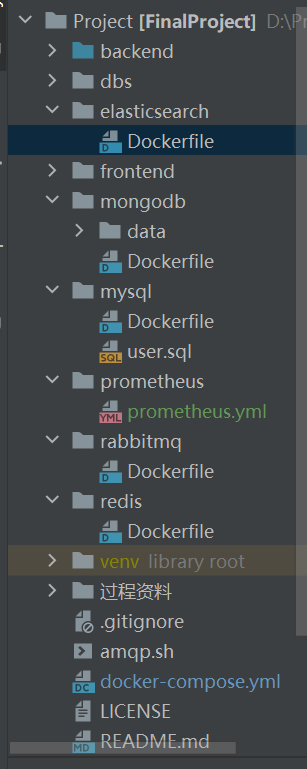


图5-1 系统目录结构图

如图5-1所示，每个子目录可能会有一个Dockerfile用来构建一个单独的Docker镜像。这些镜像可以在docker-compose.yml文件中定义为不同的服务，通过定义服务之间的依赖关系，将它们组合起来以形成一个完整的应用程序。

### 5.6.1 backend目录下的Dockerfile

代码见附录1.6.1。

以上配置的作用是将应用程序及其依赖打包为一个Docker镜像。该镜像包含了Python 3.10的运行环境和一些其他配置，使得应用程序可以在不同的机器上稳定运行。该Dockerfile将Python依赖包安装在Docker镜像中，并将应用程序代码复制到镜像中。最后通过运行uvicorn启动ASGI服务器，使得应用程序可以提供API服务，监听来自网络的请求。

### 5.6.2 frontend下的Dockerfile

代码见附录1.6.2。

以上配置的作用是将前端应用代码打包为一个Docker镜像。该Dockerfile使用Node.js 14作为基础镜像，然后在工作目录中安装了所有依赖包并构建了应用程序。最后通过运行npm run serve命令启动前端应用程序，并监听80端口提供服务。

### 5.6.3 rabbitmq目录下的Dockerfile

代码见附录1.6.6。

以上配置的作用是将RabbitMQ消息队列打包为一个Docker镜像。该Dockerfile使用最新版本的RabbitMQ作为基础镜像，并设置了默认的管理员账户和密码。然后通过rabbitmq-plugins enable命令启用了RabbitMQ管理插件。最后通过运行rabbitmq-server命令启动RabbitMQ消息队列服务，并暴露5672和15672端口以便与其他容器进行通信。

### 5.6.4 redis目录下的Dockerfile

代码见附录1.6.7。

以上配置的作用是将Redis数据库打包为一个Docker镜像。该Dockerfile使用最新版本的Redis作为基础镜像，并暴露6379端口以便与其他容器进行通信。最后通过运行redis-server命令启动Redis服务，使得其他容器可以通过6379端口连接到Redis数据库。

### 5.6.5 mysql目录下的Dockerfile

代码见附录1.6.5。

以上配置的作用是将MySQL数据库打包为一个Docker镜像。该Dockerfile使用MySQL 8.0作为基础镜像，并设置了root用户的默认密码。然后将用户自定义的user.sql文件添加到了容器初始化脚本docker-entrypoint-initdb.d中，以便在启动容器时自动运行该SQL脚本创建数据库、表和用户等。最后通过EXPOSE命令暴露MySQL数据库服务的3306端口以便与其他容器进行通信。

### 5.6.6 mongodb目录下的Dockerfile

FROM mongo:4.4.4-bionic

EXPOSE 27017

以上配置的作用是将MongoDB数据库打包为一个Docker镜像。该Dockerfile使用MongoDB 4.4.4-bionic作为基础镜像，并通过EXPOSE命令暴露了MongoDB数据库服务的默认端口27017以便与其他容器进行通信。

### 5.6.7 docker-compose.yml

代码见附录1.6.10。

这是一个docker-compose文件，用于启动多个容器来部署一个分布式应用程序。该应用程序包括一个前端、一个后端，以及一些辅助组件，如数据库、消息队列、监控等。该docker-compose文件定义了以下服务：

frontend：用于构建并启动前端应用程序的服务，通过端口映射将容器内的80端口映射到主机的80端口，并通过depends\_on关键字指定了依赖的backend服务，同时定义了环境变量API\_URL，用于指定后端API的地址。

prometheus：用于启动Prometheus监控服务的服务，使用官方的Prometheus镜像，并将容器内的9090端口映射到主机的9090端口，同时将宿主机上的./prometheus目录挂载到容器内的/etc/prometheus目录，用于指定Prometheus的配置文件。

backend：用于构建并启动后端应用程序的服务，通过端口映射将容器内的8080端口映射到主机的8080端口，同时定义了多个环境变量，用于指定后端应用程序的配置参数，如数据库、消息队列、缓存等。

celery：用于启动Celery任务队列的服务，通过构建后端应用程序的镜像，定义了多个环境变量，用于指定Celery的配置参数，如任务队列、结果后端、MongoDB等。

flower：用于启动Celery任务监控的服务，通过构建后端应用程序的镜像，定义了多个环境变量，用于指定Celery和Flower的配置参数，同时将容器内的5555端口映射到主机的5555端口，用于通过Web界面监控Celery任务。

mysql：用于启动MySQL数据库的服务，通过构建自定义的MySQL镜像，将容器内的3306端口映射到主机的3306端口，同时定义了环境变量，用于指定MySQL的root用户的默认密码。

mongodb：用于启动MongoDB数据库的服务，通过构建自定义的MongoDB镜像，将容器内的27017端口映射到主机的27017端口，同时将宿主机上的./mongodb/data目录挂载到容器内的/data/db目录，用于持久化MongoDB的数据。

elasticsearch：用于启动Elasticsearch搜索引擎的服务，通过构建自定义的Elasticsearch镜像，将容器内的9200端口和9300端口映射到主机的9200端口和9300端口，同时定义了环境变量，用于指定Elasticsearch的配置参数。

rabbitmq：用于启动RabbitMQ消息队列的服务，通过构建自定义的RabbitMQ镜像，将容器内的5672端口和15672端口映射到主机的5672端口和15672端口。

redis：用于启动Redis数据库的服务，通过构建自定义的Redis镜像，将容器内的6379端口映射到主机的6379端口。

除了以上内容，该docker-compose文件还使用了多个关键字来定义服务之间的依赖关系和环境变量。通过depends\_on关键字，可以定义服务之间的依赖关系，即某个服务依赖于其他服务，只有当其他服务启动完成后，才能启动该服务。通过environment关键字，可以定义服务的环境变量，用于指定服务的配置参数。

# 6 系统部署与测试

## 6.1 系统部署

经过测试，系统可以正常通过docker-compose.yml文件和Docker进行部署。

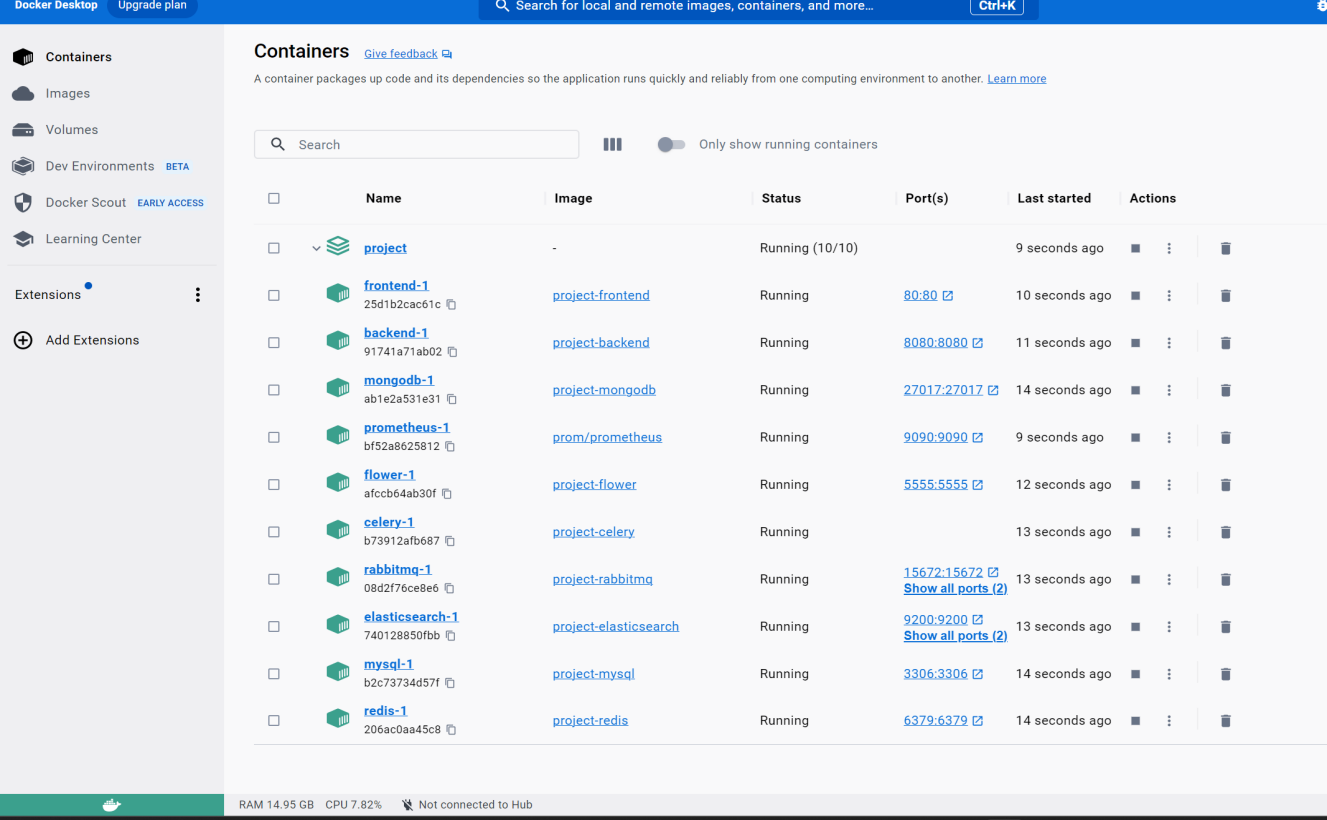


图6-1 系统部署测试

如图6-1所示，共10个服务，都能够正常运行，显示状态为running，并且可以正确映射端口。

## 6.2 系统登录模块测试

尝试使用已存在用户密码进行登录，测试结果如下：

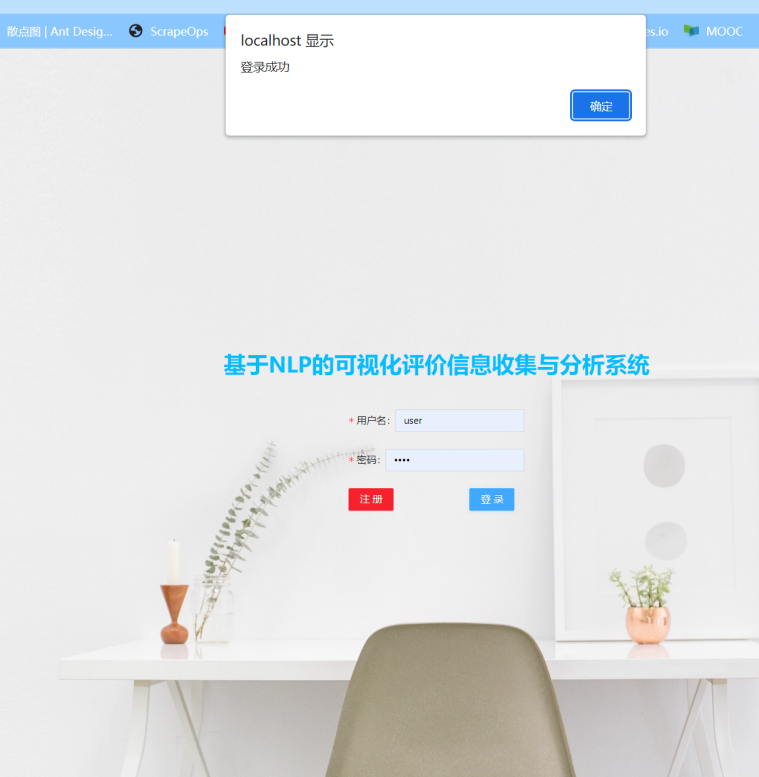


图6-2 登录成功结果

如图6-2所示，在输入正确的用户和密码后，提示登录成功。

尝试使用不正确的用户名和密码进行登录，结果如下：

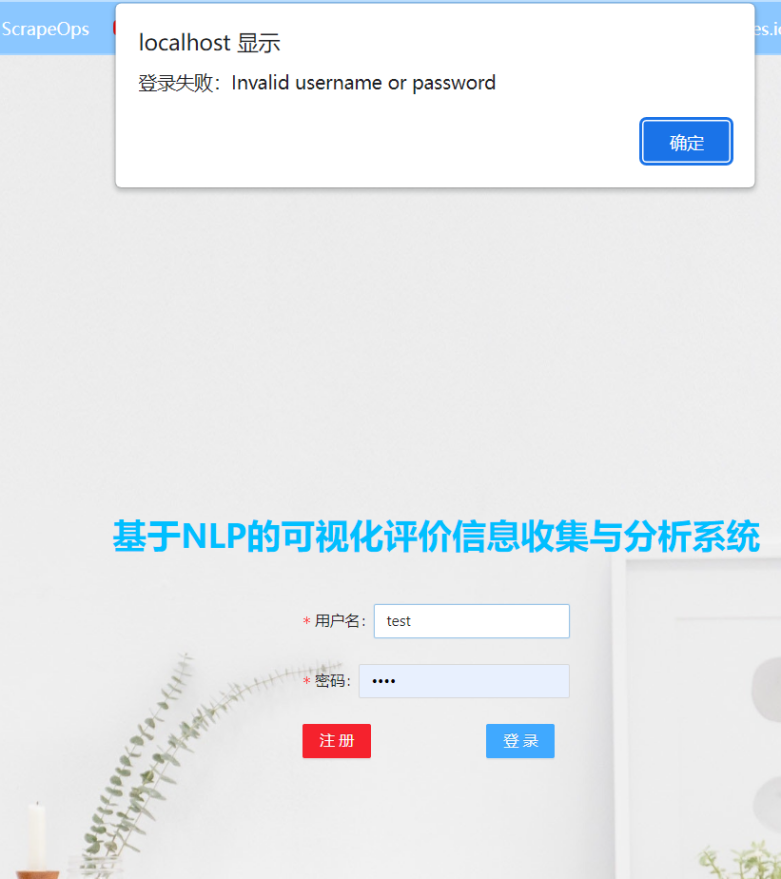


图6-3 登录失败结果

如图6-3所示，当输入错误的用户和密码后，提示登录失败。

使用未注册过的邮箱进行注册，测试结果如下：



图6-4 注册成功图

如图6-4所示，当输入信息注册后，提示注册成功。

再次使用相同邮箱注册，测试结果如下：

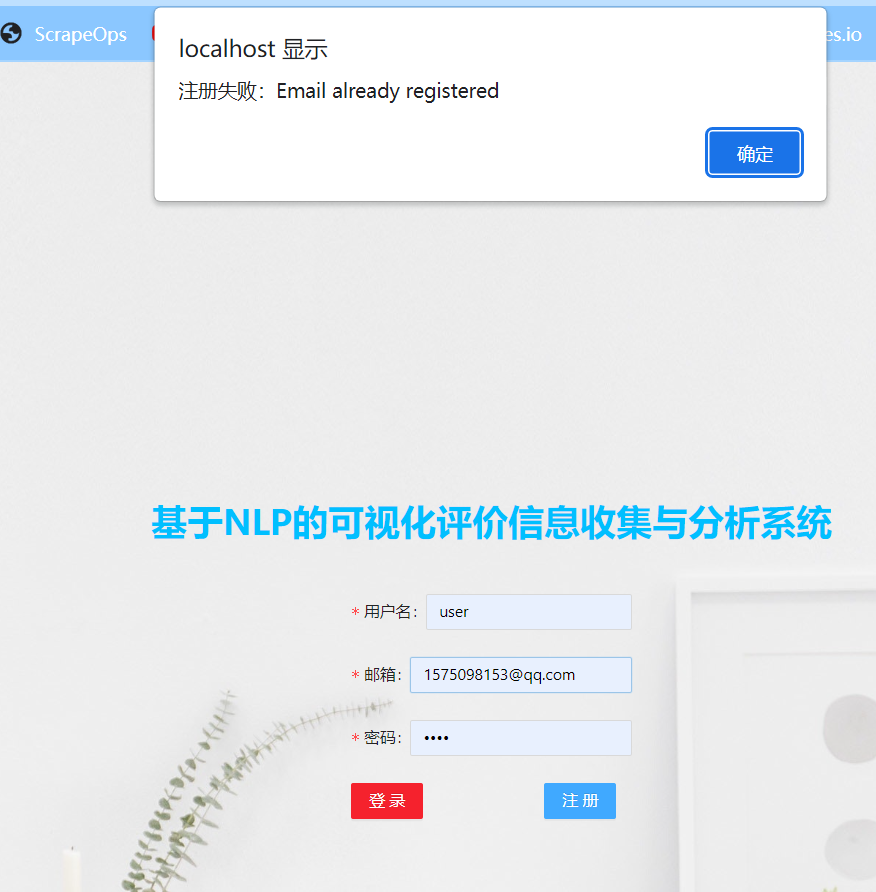


图6-5 注册失败图

如图6-5所示，当注册的邮箱已经创建过账号后，提示注册失败。

## 6.3 系统数据收集模块测试

尝试输入用户ID和Cookie后，点击提交按钮，测试结果如下：

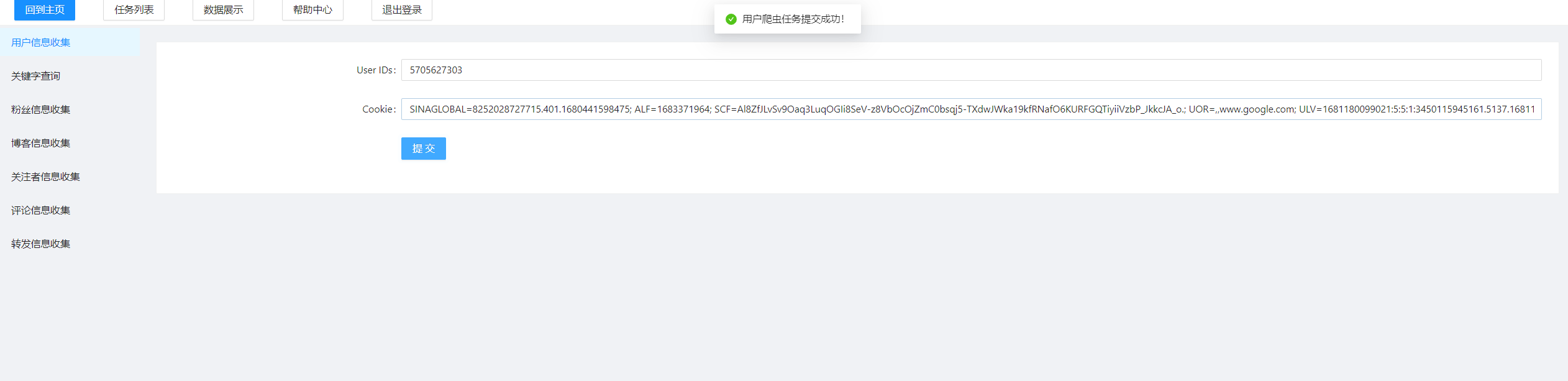


图6-6 成功提交用户信息收集任务

如图6-6所示，当输入运行爬虫必要的信息后，点击提交。提示提交成功。

尝试输入相应信息后，点击提交按钮，测试结果如下：

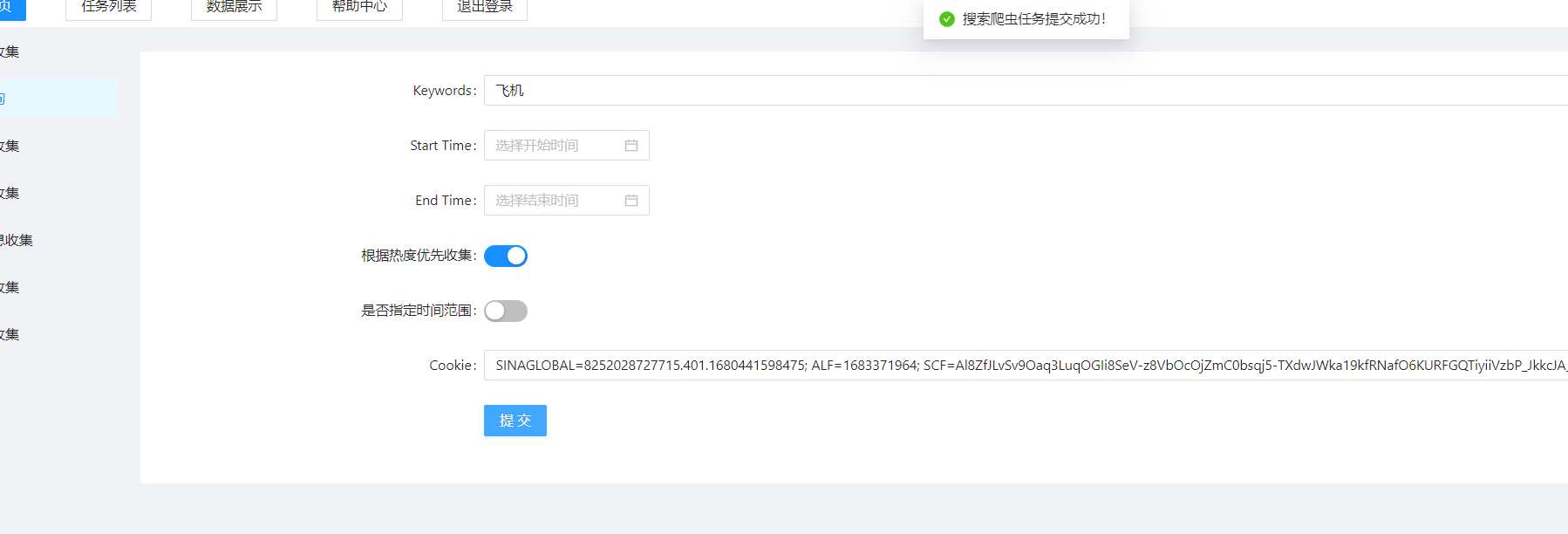


图6-7 成功提交关键词搜索任务

如图6-7所示，当输入运行任务必要的参数后，点击提交。提示提交任务成功。

尝试输入相应信息后，点击提交按钮，测试结果如下：

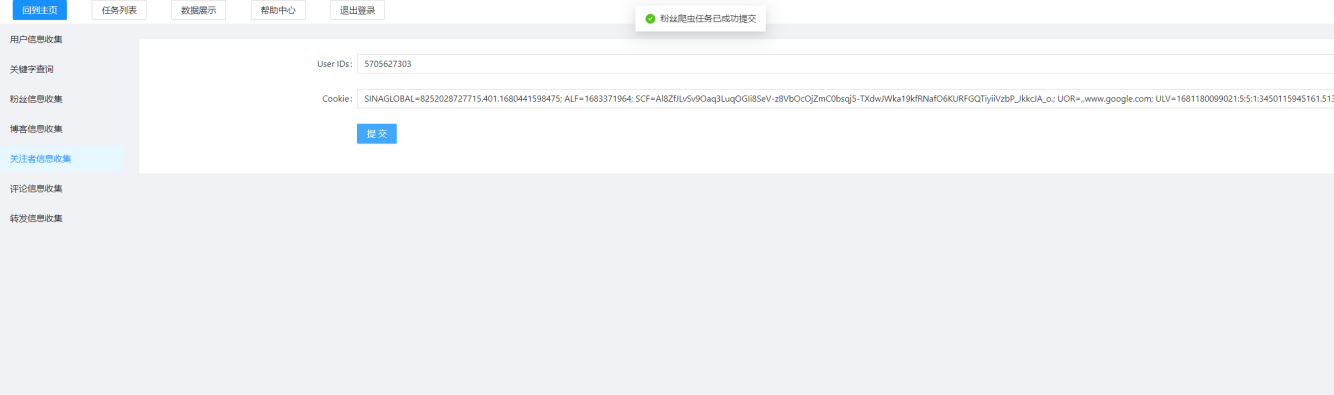


图6-8 成功提交关注者信息收集任务

如图6-8所示，当输入运行任务必要的参数后，点击提交。提示提交任务成功。

尝试输入相应信息后，点击提交按钮，测试结果如下：

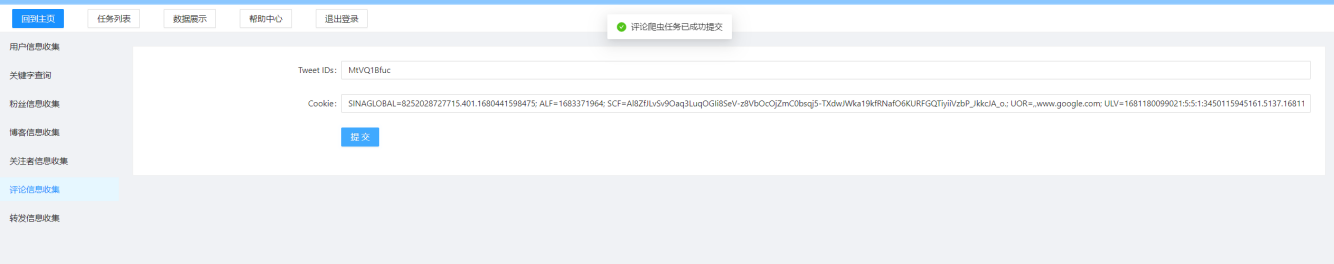


图6-9 成功提交评论信息收集任务

如图6-9所示，当输入运行任务必要的参数后，点击提交。提示提交任务成功。

尝试输入相应信息后，点击提交按钮，测试结果如下：

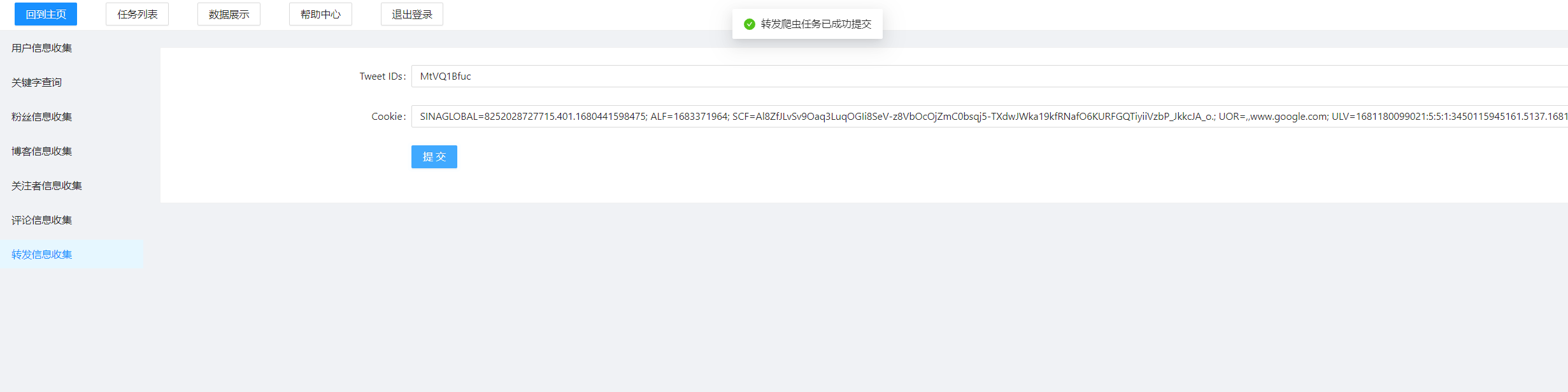


图6-10 成功提交转发信息收集任务

如图6-10所示，当输入运行任务必要的参数后，点击提交。提示提交任务成功。

尝试输入相应信息后，点击提交按钮，测试结果如下：

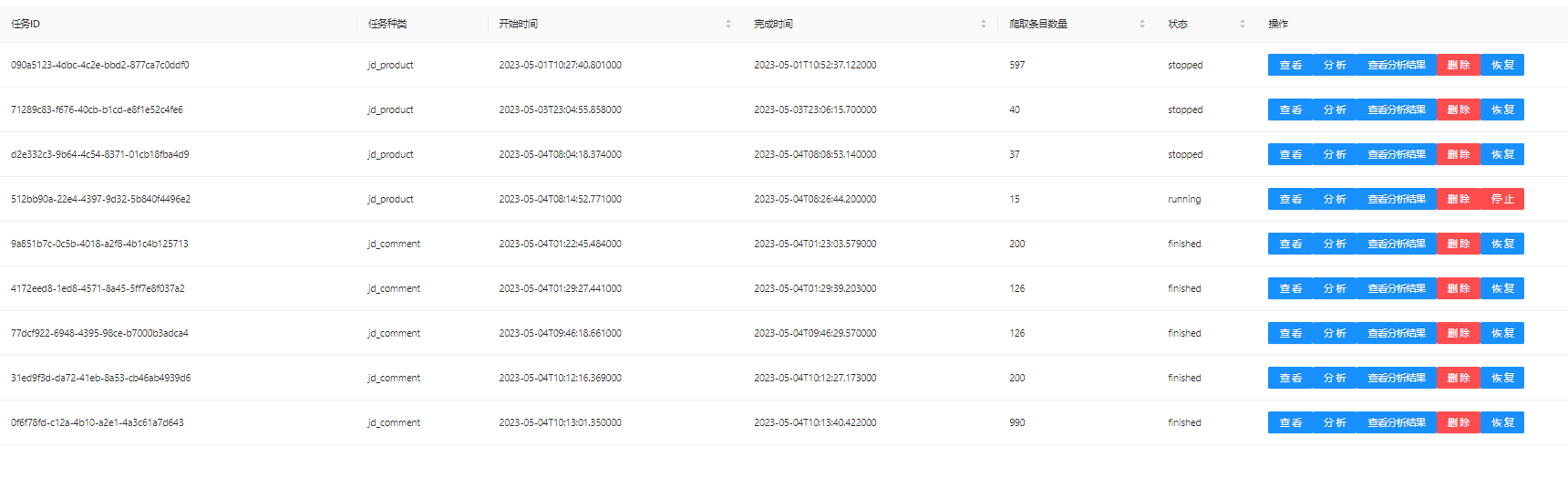


图6-11 任务列表显示任务状态

如图6-11所示，在点击了提交后，进入任务中心，可以看到相关信息，比如状态为running。

尝试输入相应信息后，点击提交按钮，测试结果如下：

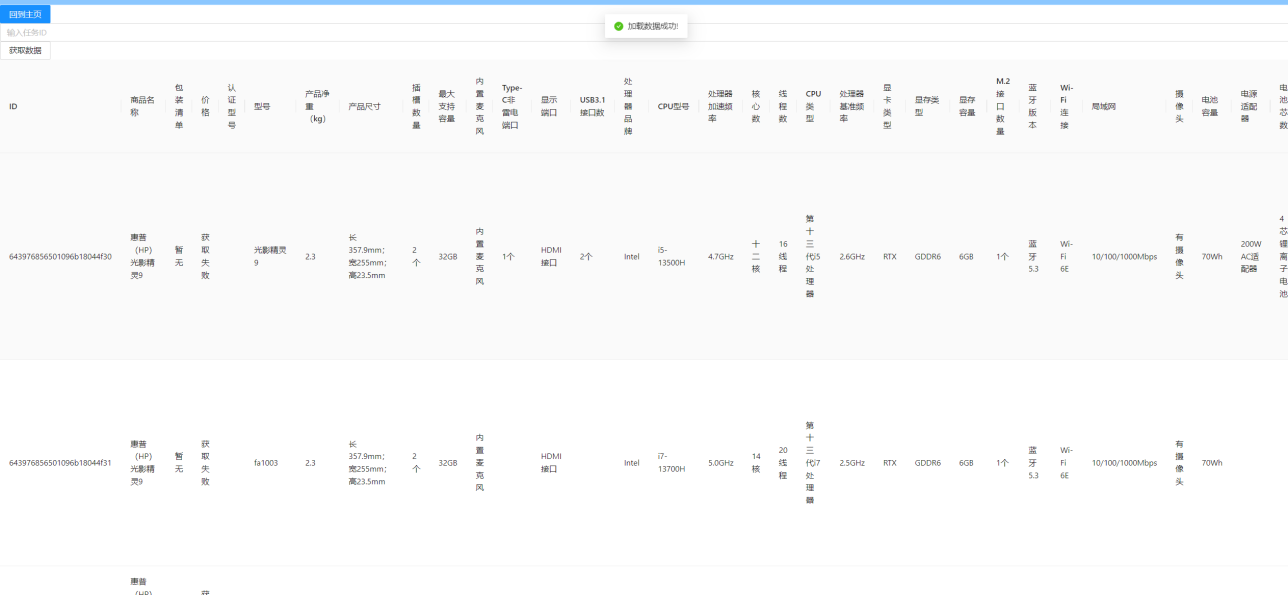


图6-12 爬取数据显示

如图6-12所示，成功显示了收集到的京东商品的相关信息。

## 6.4 系统数据分析模块测试

尝试点击任务列表中的任一项的分析按钮，测试结果如下：

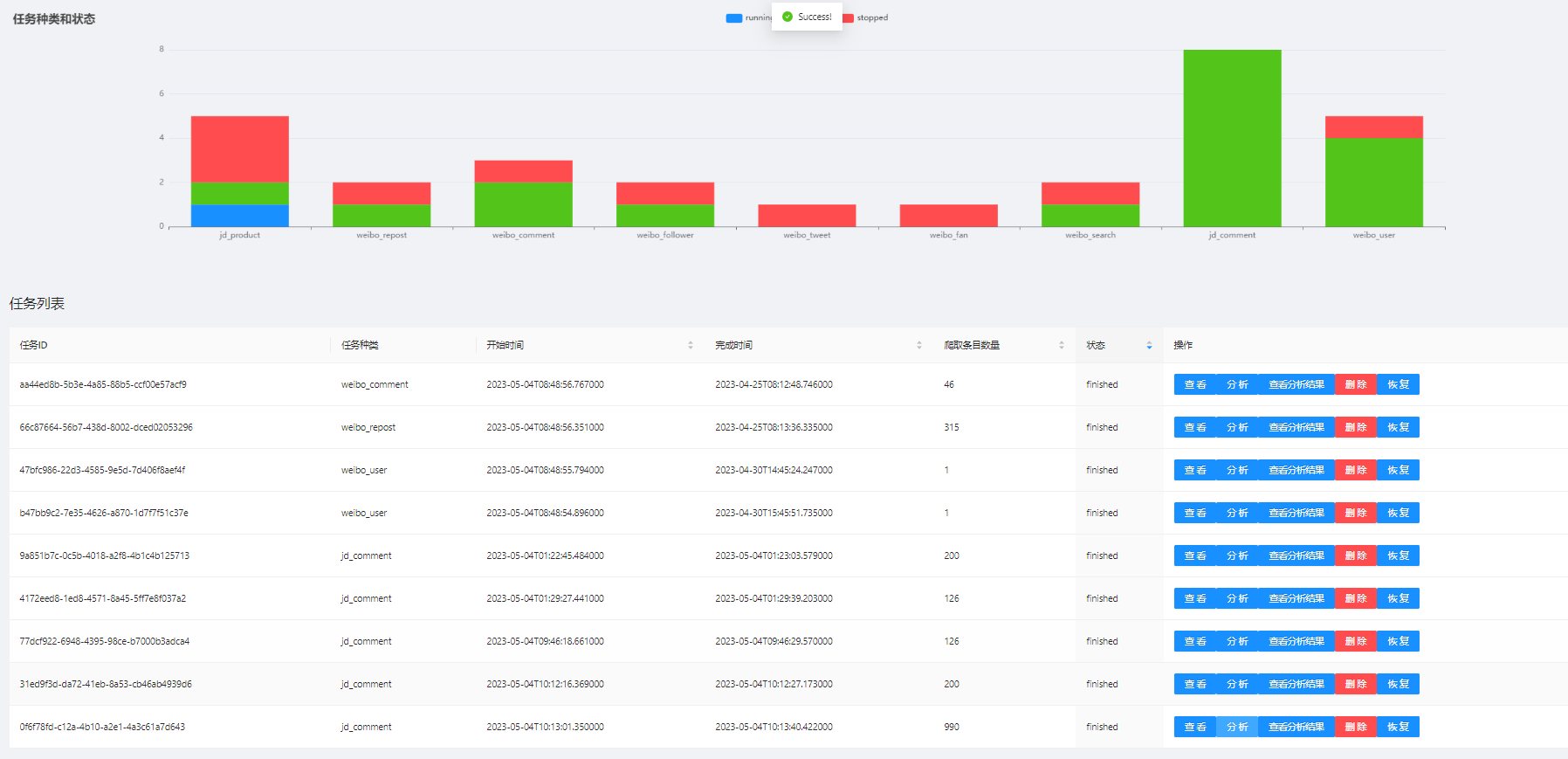


图6-13 成功提交数据分析任务

如图6-13所示，在界面中显示了提交成功。

## 6.5 系统数据可视化模块测试

在提交任务后，返回主页，测试结果如下：

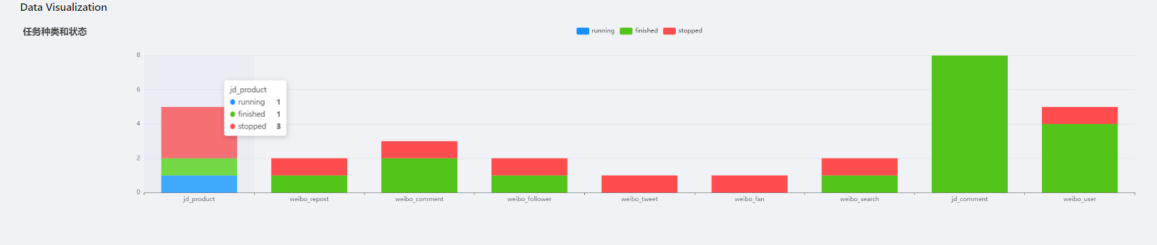


图6-14 任务统计信息

如图6-14所示，显示了每一种任务的数量，以及不同状态任务的数量，并且每一种状态的任务有不同颜色。其中，完成为绿色，运行中为蓝色，停止为红色。

选择京东商品任务条目，点击查看分析结果按钮，测试结果如下：

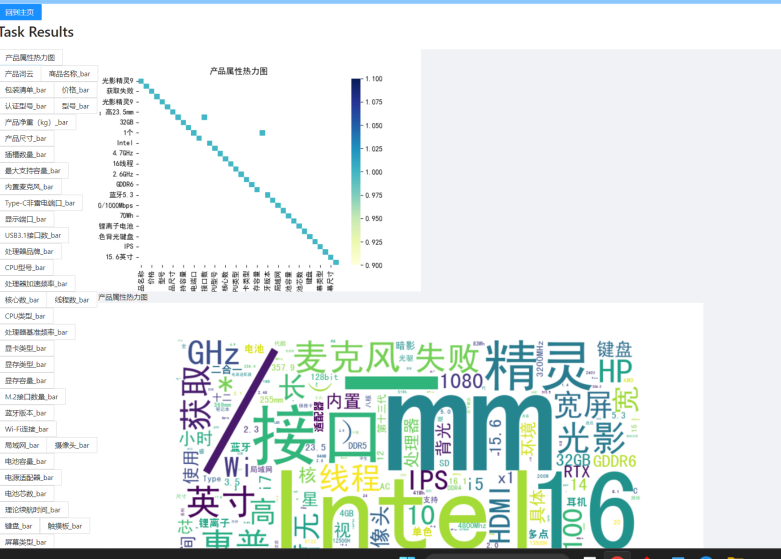


图6-15 京东商品可视化结果

如图6-15所示，显示了产品属性的热力图、产品属性词云等信息。

选择微博博文任务条目，点击查看分析结果按钮，测试结果如下：

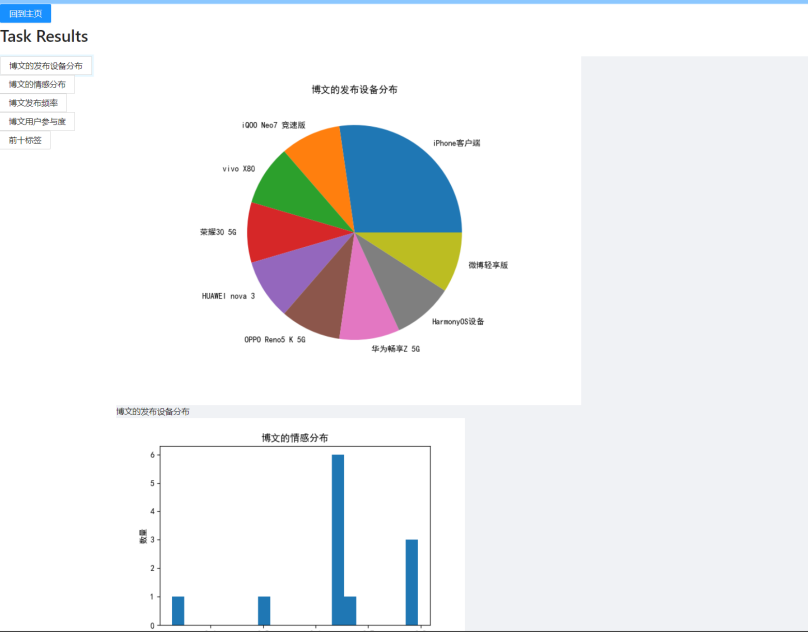


图6-16 博文可视化结果

如图6-16所示，页面显示了博文的发布设备分布、博文的情感分布等信息。

选择微博关注者任务条目，点击查看分析结果按钮，测试结果如下：



图6-17 关注者信息可视化结果

如图6-17所示，页面显示了关注者年龄分布、关注者的兴趣分布等等可视化结果。

选择京东商品评价信息任务条目，点击查看分析结果按钮，测试结果如下：



图6-18 京东商品评价信息可视化结果

如图6-18所示，可视化结果为一个词云，该词云统计了评价信息中关键词出现的频率，并且根据频率大小构造了词云。

## 6.6 系统任务调度与通知模块测试

在进行任务相关操作后，返回主页或者进入任务中心，测试结果如下：

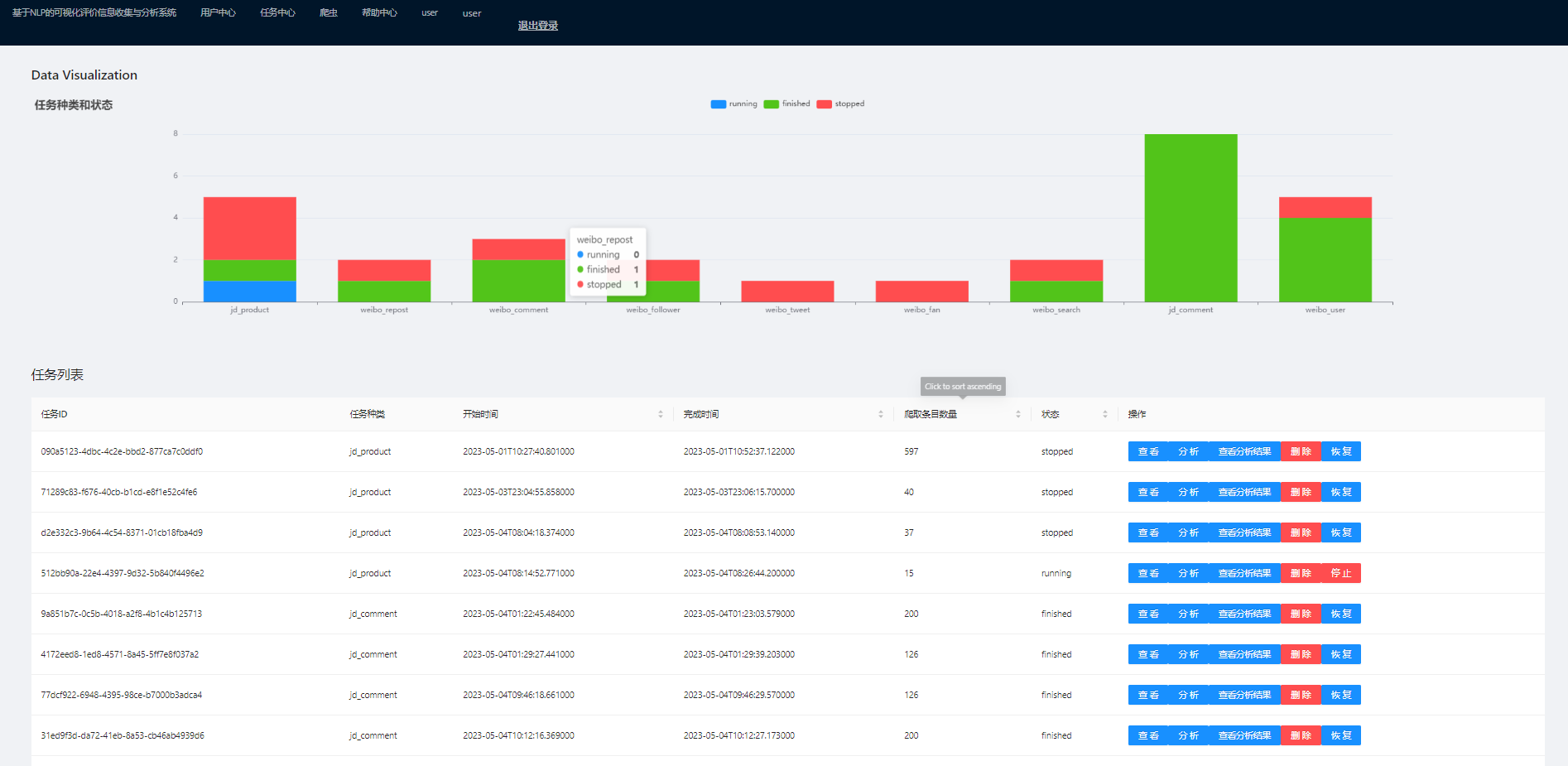


图6-19 任务中心成功显示任务相关信息

如图6-19所示，任务中心显示了历史任务的相关信息，比如状态、爬取条目数量、任务类型等等信息。

在尝试发送多个任务后，查看Celery worker的输出，测试结果如下：

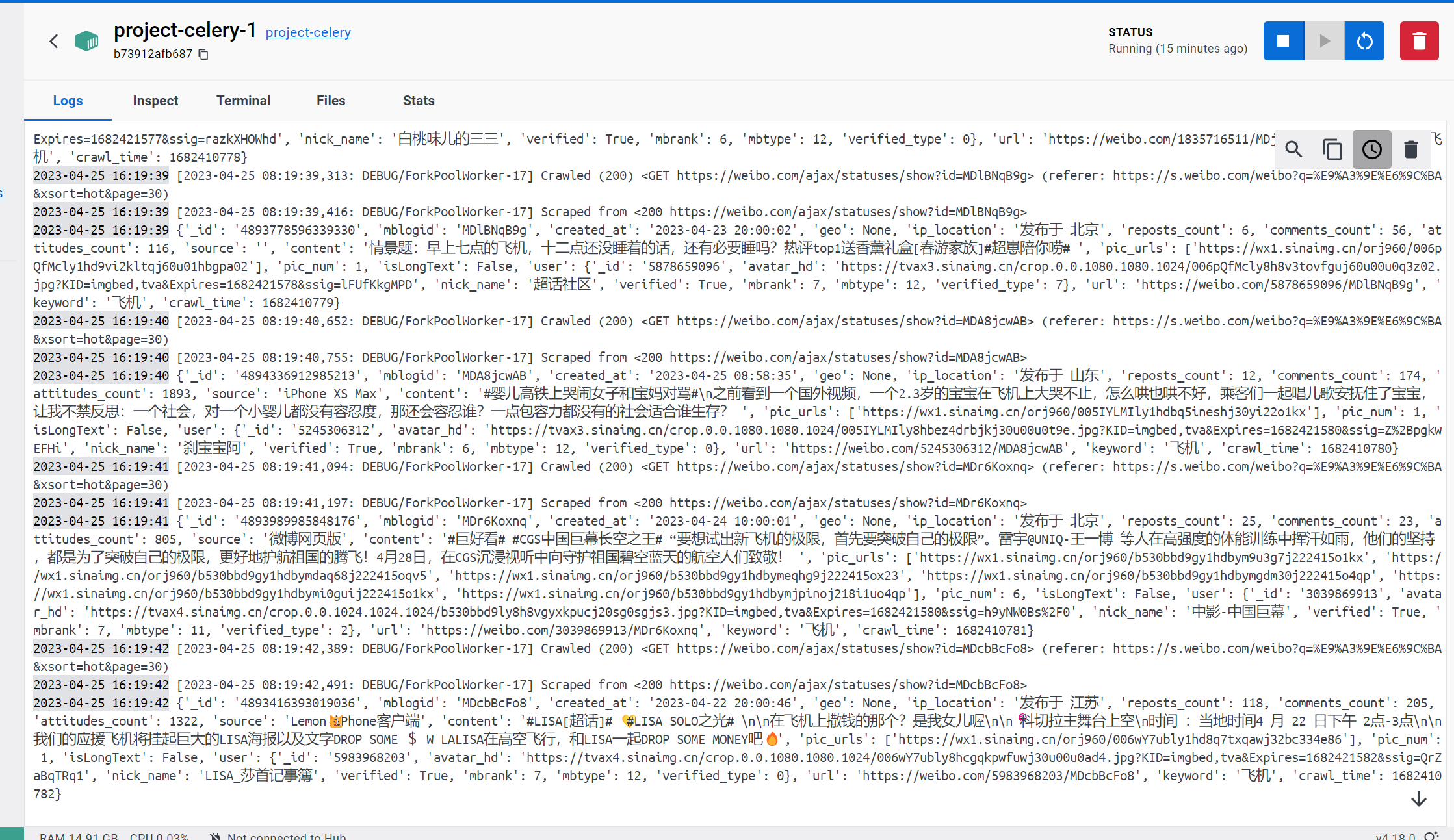


图6-20 Celery调度信息以及运行状况

如图6-20所示，Scrapy爬虫在Celery中运行，并且同时运行多个爬虫。

尝试点击任务列表中正在运行的任务的停止按钮，结果如下图。



图6-21 停止提示

图6-21所示，提示是否确定停止这个任务。

继续点击OK。结果如下：

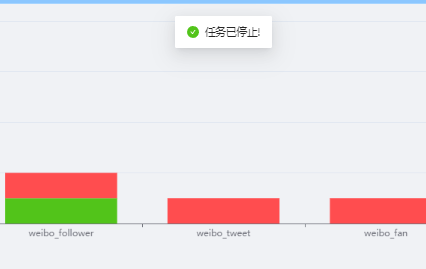


图6-22 停止任务成功

图6-22所示，提示任务已停止。

在任务完成后，检查邮箱。在代码中配置了邮箱的地址(248334369@qq.com)。发送者为1575098153@qq.com。结果如下：



图6-23 QQ邮箱通知结果

图6-23所示，248334369@qq.com邮箱中收到了1575098153的邮件。邮件中说明了任务的状态以及相关信息。

## 6.7 系统其他功能测试

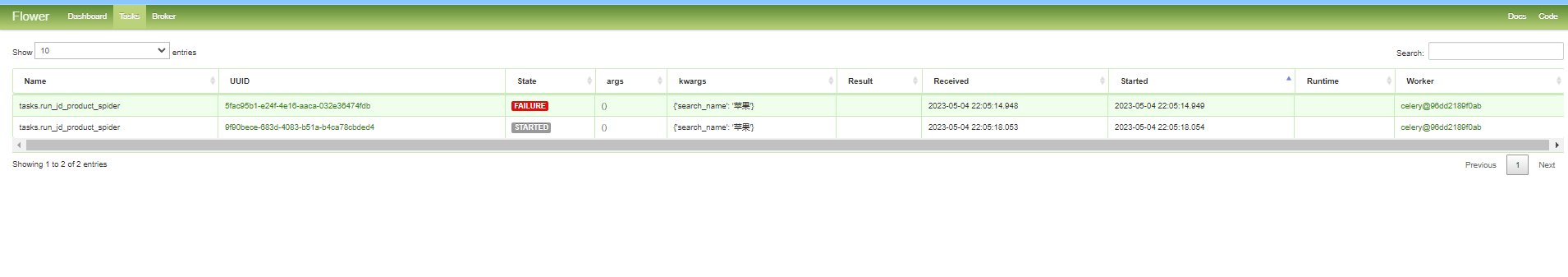


图6-24 Flower功能测试结果

如图6-24所示，Flower可以监测到任务发送的信息，比如图中发送的search\_name，以及其他的一些信息。

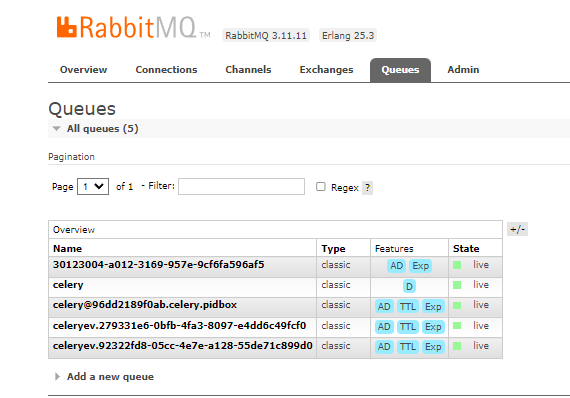


图6-25 RabbitMQ测试结果

如图6-25所示，RabbitMQ显示了任务队列中的任务，第一项即为发送的任务，Name为任务ID。

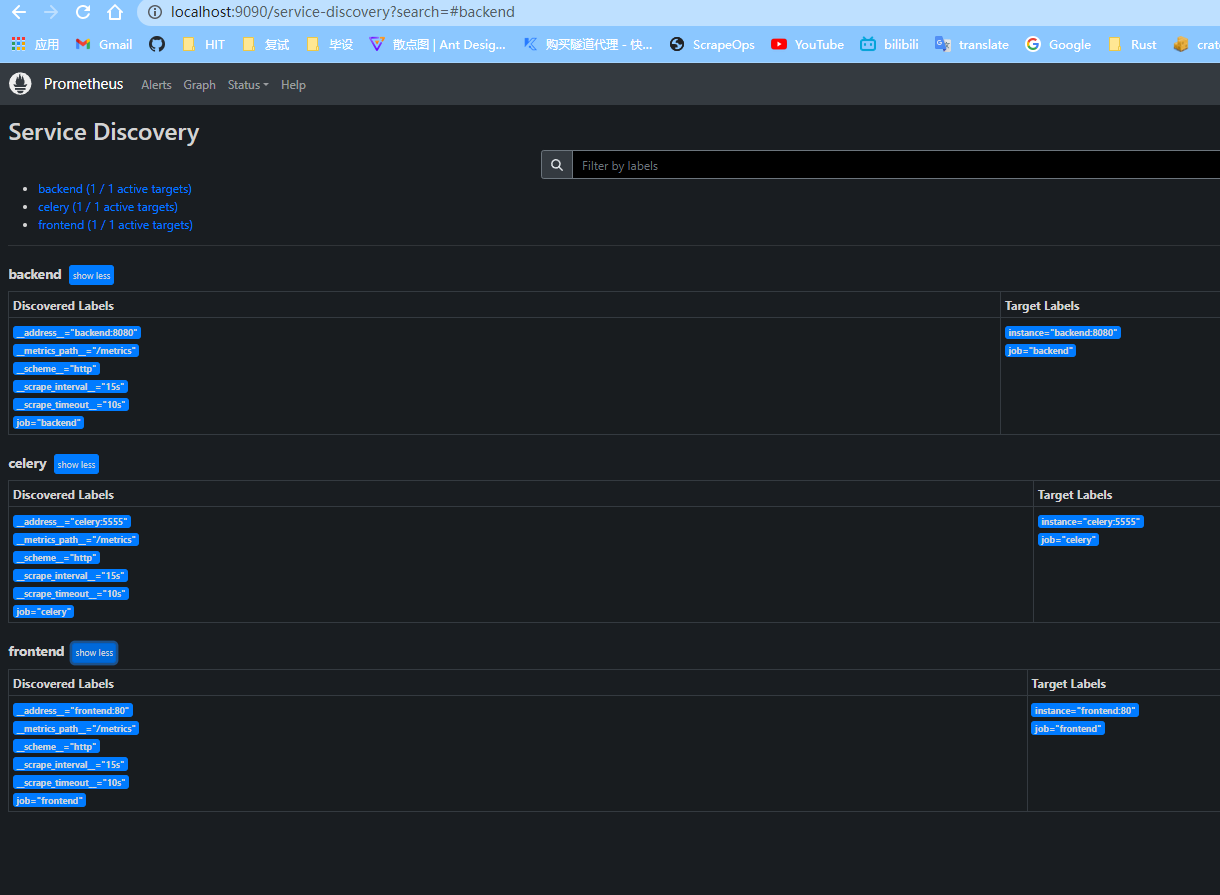


图6-26 Prometheus测试结果

如图6-26所示，Prometheus可以正确的发现backend、frontend和celery服务。

# 7 系统的发展方向及应用前景

数据分析和挖掘：随着数据规模的不断增大和数据分析需求的不断增加，数据分析和挖掘的市场需求也在不断扩大。该系统可以为用户提供更丰富的数据分析和挖掘功能，满足用户不同的分析需求，并可以应用于多个领域，如电商、金融、医疗等。

基于大数据的智能应用：随着人工智能技术的不断发展和应用，基于大数据的智能应用也将越来越受到关注。该系统可以为用户提供智能化的数据分析和挖掘功能，帮助用户更好地理解和利用评价信息。

大规模数据处理：随着数据规模的不断增大和数据处理需求的不断增加，大规模数据处理技术的市场需求也在不断扩大。该系统可以使用 Apache Spark 等大规模数据处理技术来处理评价信息和其他大规模数据，提高数据处理的效率和性能。

多模态数据分析：除了文本数据，还有图像、音频、视频等多种数据类型需要分析和处理。该系统可以扩展到多模态数据分析和处理领域，为用户提供更多样化的数据分析和挖掘功能。

# 8 结束语

设计和实现基于NLP的可视化评价信息收集与分析系统是一项具有挑战性的任务，需要学习很多没有接触过的内容。为了实现成功，必须明确系统的目标和要求，并考虑目标受众和数据源以确定最佳的数据处理技术。同时，在实现过程中，也遇到了很多问题，比如启动worker时celery不能用5.0及以上版本，因为已经弃用，需要使用一定的特殊技巧才可以使用，或者切换运行时系统。类似于这样的问题往往需要很长的时间才能找到解决方案，需要一定的耐心。由于技术水平有限，在实现过程中，经常会因为一些细节问题欠缺考虑，导致需要多次对代码进行重构，这也更加说明了提前对系统进行设计的必要性。

通关这次的毕业设计，我充分地认识到了学习能力的重要性，在未来的工作中，更多的是以项目为导向的学习，不断学习新技术，提高自己的个人能力。

# 参考文献

[1]Ma Xiaoju,Yan Min. Design and Implementation of Crawler Program Based on Python[J]. Journal of Physics: Conference Series,2021,2033(1).

[2]Feng Guilian. Implementation of Web Data Mining Technology Based on Python[J]. Journal of Physics: Conference Series,2021,2066(1).

[3]Rameshbhai Chaudhary Jashubhai,Paulose Joy. Opinion mining on newspaper headlines using SVM and NLP[J]. International Journal of Electrical and Computer Engineering (IJECE),2019,9(3).

[4]方奇洲,程友清.基于Docker容器的分布式爬虫的设计与实现[J].电子设计工程,2020,28(08):61-65.

[5]闵江松. 基于Docker集群的分布式爬虫系统设计与实现[D].哈尔滨工业大学,2020.

[6]王予诺.基于API服务器的爬虫项目设计与实现[J].电子技术与软件工程,2020(02):201-203.

[7]许惠.基于NLP方法实现文本分类识别[D].大连理工大学,2021.

[8]王宝楠.基于NLP的商品在线评论研究[J].中国新通信,2020,22(17):23-24.

[9]王辛浩,单艳.探究Python语言下网络爬虫的技术特点及应用[J].数字技术与应用用,2022,40(10):85-87.

[10]王国华. 基于python的豆瓣电影网络爬虫设计与分析[C]//.第三十六届中国（天津）2022’IT、网络、信息技术、电子、仪器仪表创新学术会议论文集.,2022:212-215.

[11]杨孟姣,杜棋东.基于Python爬虫网站数据分析系统设计与实现[J].计算机时代,2022(11):81-83+88.

# 致 谢

千言万语，不如一笺谢意。

# 附 录

# 附录1 源程序清单

## 附录1.1 系统功能模块

### 附录1.1.1 用户登录与注册模块

from datetime import timedelta

from fastapi import APIRouter, Depends, HTTPException

from fastapi.security import OAuth2PasswordRequestForm

from sqlalchemy.orm import Session

from app.dependencies import get\_db

from app.models.user import User, UserCreate

from app.security import hash\_password, create\_access\_token, verify\_password

router = APIRouter()

ACCESS\_TOKEN\_EXPIRE\_MINUTES = 180

@router.post("/login")

async def login(form\_data: OAuth2PasswordRequestForm = Depends(), db: Session = Depends(get\_db)):

user = db.query(User).filter(User.username == form\_data.username).first()

if not user:

raise HTTPException(status\_code=400, detail="不合法的用户名或密码 ")

if not verify\_password(form\_data.password, user.password):

raise HTTPException(status\_code=400, detail="不合法的用户名或密码")

access\_token\_expires = timedelta(minutes=ACCESS\_TOKEN\_EXPIRE\_MINUTES)

access\_token = create\_access\_token(

data={"sub": user.username}, expires\_delta=access\_token\_expires)

return {"access\_token": access\_token, "token\_type": "bearer"}

@router.post("/register")

async def register(user\_create: UserCreate, db: Session = Depends(get\_db)):

user = db.query(User).filter(User.email == user\_create.email).first()

if user:

raise HTTPException(status\_code=400, detail="Email already registered")

hashed\_password = hash\_password(user\_create.password)

new\_user = User(username=user\_create.username, email=user\_create.email, password=hashed\_password)

db.add(new\_user)

db.commit()

db.refresh(new\_user)

return new\_user

### 附录1.1.2 系统功能其他模块

<template>

<div class="help-center">

<h1>帮助中心</h1>

<p>如果需要的话，可以点击下方按钮下载PDF版本的使用手册。</p>

<a-button type="primary" @click="downloadGuide">下载</a-button>

<p>If you have any questions or need further assistance, please <a href="mailto:248334369@qq.com">联系我</a>.</p>

<div class="github-link">

<a href="https://github.com/your-username/your-repo">GitHub</a>

<a-button type="dashed" shape="circle" icon="github" href="https://github.com/your-username/your-repo"></a-button>

</div>

</div>

</template>

<template>

<div>

<a-layout style="min-height: 10vh">

<a-menu mode="horizontal" theme="dark">

<a-menu-item>

<router-link to="/home">基于NLP的可视化评价信息收集与分析系统</router-link>

</a-menu-item>

<a-menu-item>

<router-link to="/user\_center">用户中心</router-link>

</a-menu-item>

<a-menu-item>

<router-link to="/task\_center">任务中心</router-link>

</a-menu-item>

<a-sub-menu title="爬虫">

<a-menu-item>

<router-link to="/weibo/run\_tasks">微博爬虫</router-link>

</a-menu-item>

<a-menu-item>

<router-link to="/jd/run\_tasks">京东爬虫</router-link>

</a-menu-item>

</a-sub-menu>

<a-menu-item>

<router-link to="/help\_center">帮助中心</router-link>

</a-menu-item>

<a-menu-item>

<router-link to="/user\_center">{{currentUser.username}}</router-link>

</a-menu-item>

<a-menu-item>

<a class="link" href="mailto:{{ currentUser.email }}">{{ currentUser.email }}</a>

</a-menu-item>

<a-menu-item>

<a class="button" href="#">退出登录</a>

</a-menu-item>

</a-menu>

<a-layout-content style="padding: 0 50px">

<div class="visualization">

<h2>Data Visualization</h2>

<div class="chart">

<v-chart ref="myChart" :option="chartOptions" autoresize></v-chart>

</div>

</div>

<div class="table">

<h2>任务列表</h2>

<a-table :columns="columns" :data-source="tasks" :loading="loading">

<template v-slot:action="{ record }">

<span>

<a-button type="primary" @click="handleView(record.task\_id, record.task\_type)">

查看

</a-button>

<a-button type="primary" @click="handleAnalyze(record.task\_id, record.task\_type)">分析</a-button>

<a-button type="primary" @click="handleResult(record.task\_id, record.task\_type)">查看分析结果</a-button>

<a-popconfirm

title="Are you sure to delete this task?"

@confirm="handleDelete(record.task\_id)"

>

<a-button type="danger">删除</a-button>

</a-popconfirm>

</span>

</template>

</a-table>

</div>

</a-layout-content>

</a-layout>

</div>

</template>

## 附录1.2 数据收集模块

### 附录1.2.1 微博数据收集模块

from weibospider.settings import DEFAULT\_REQUEST\_HEADERS

class CommentSpider(Spider):

name = "comment\_spider"

tweet\_ids = []

cookie = []

headers = []

task\_id = ''

stats\_info = {}

def \_\_init\_\_(self, tweet\_ids=None, cookie=None, task\_id=None, \*args, \*\*kwargs):

super(CommentSpider, self).\_\_init\_\_(\*args, \*\*kwargs)

self.tweet\_ids = tweet\_ids

self.cookie = cookie

self.task\_id = task\_id

# Set cookie in default headers

if self.cookie is not None:

self.headers = DEFAULT\_REQUEST\_HEADERS.copy()

self.headers['Cookie'] = self.cookie

def parse(self, response, \*\*kwargs):

data = json.loads(response.text)

if data.get('max\_id', 0) != 0:

url = response.meta['source\_url'] + '&max\_id=' + str(data['max\_id'])

yield Request(url, callback=self.parse, meta=response.meta, headers=self.headers, cookies=self.cookie)

@staticmethod

def parse\_comment(data):

item = dict()

item['created\_at'] = parse\_time(data['created\_at'])

item['\_id'] = data['id']

item['content'] = data['text\_raw']

item['comment\_user'] = parse\_user\_info(data['user'])

return item

### 附录1.2.2 京东数据收集模块

import json

import re

import requests

import scrapy

import time

from jdspider.items import JdcommentItem

class JDcommentspider(scrapy.Spider):

name = 'JDcommentspider'

allowed\_domains = ['jd.com']

start\_urls = []

task\_id = ''

pages = ''

custom\_settings = {

'ITEM\_PIPELINES': {

'jdspider.pipelines.JDcommentPipeline': 300,

}

}

def \_\_init\_\_(self, urls, pages, task\_id):

super(JDcommentspider, self).\_\_init\_\_()

self.pages = int(pages)

self.task\_id = task\_id

if type(urls) == str:

self.start\_urls = [urls]

print(self.start\_urls)

elif type(urls) == list:

self.start\_urls = urls

print(self.start\_urls)

else:

raise RuntimeError("参数必须为字符串或者列表")

number = re.findall(r"com/(\d+)\.html", self.start\_urls[0])[0]

self.comment\_page\_baseurl = 'https://sclub.jd.com/comment/productPageComments.action?productId=' + number + '&score=0&sortType=5&page={0}&pageSize=10'

def parse(self, response):

comlist = response.xpath("//div[@id='hidcomment']/div[@class='item']//div[@class='o-topic']")

name = response.xpath("//div[@class='item ellipsis']/text()").extract()[0].strip()

for com in comlist:

item = JdcommentItem()

item['content'] = com.xpath(".//a/text()").extract()[0]

item['date'] = com.xpath(".//span[@class='date-comment']/text()").extract()[0]

item['url'] = response.url

item['name'] = name

yield item

page = 0

## 附录1.3数据分析模块

### 附录1.3.1 微博数据分析模块

def analyze\_weibo\_data(task\_id):

collection = weibo\_db[task\_id]

weibo\_data = list(collection.find())

reposts\_list = []

comments\_list = []

likes\_list = []

for d in weibo\_data:

if 'reposts\_count' in d:

reposts\_list.append(d['reposts\_count'])

if 'comments\_count' in d:

comments\_list.append(d['comments\_count'])

if 'attitudes\_count' in d:

likes\_list.append(d['attitudes\_count'])

engagement\_df = pd.DataFrame({'转发数': reposts\_list, '评论数': comments\_list, '点赞数': likes\_list})

plt.figure(figsize=(8, 6))

engagement\_df.boxplot()

plt.title('微博用户参与度分布')

plt.ylabel('数量')

buf = io.BytesIO()

plt.savefig(buf, format='png')

buf.seek(0)

redis\_client.set(f'{task\_id}\_engagement\_boxplot', buf.getvalue())

plt.close()

sentiment\_list = []

for d in weibo\_data:

if 'content' in d:

content = d['content']

blob = SnowNLP(content)

sentiment\_list.append(blob.sentiments)

sentiment\_df = pd.DataFrame(sentiment\_list, columns=['情感得分'])

plt.figure(figsize=(6, 4))

plt.hist(sentiment\_df['情感得分'], bins=20)

plt.title('微博情感分布')

plt.xlabel('情感得分')

plt.ylabel('微博数量')

buf = io.BytesIO()

plt.savefig(buf, format='png')

buf.seek(0)

redis\_client.set(f'{task\_id}\_sentiment\_histogram', buf.getvalue())

plt.close()

### 附录1.3.2 京东数据分析模块

import os

from app.db.jd\_db import db as jd\_db

import jieba

from wordcloud import WordCloud

import matplotlib.pyplot as plt

import re

from collections import Counter

import io

from app.redis\_client import redis\_client

plt.rcParams['font.sans-serif'] = ['SimHei']

script\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

font\_path = os.path.join(script\_dir, 'SimHei.ttf')

dirname = os.path.dirname(\_\_file\_\_)

stopwords\_path = os.path.join(dirname, 'common\_stopwords.txt')

with open(stopwords\_path, 'r', encoding='utf-8') as f:

stopwords = [line.strip() for line in f.readlines()]

def load\_data(task\_id):

collection = jd\_db[task\_id]

data = collection.find()

return data

def clean\_data(data):

cleaned\_data = []

for item in data:

content = item['content']

content = re.sub('<[^<]+?>', '', content)

content = re.sub('[^\w\s]', '', content)

cleaned\_data.append(content)

return cleaned\_data

def tokenize\_data(cleaned\_data):

words\_list = []

for content in cleaned\_data:

words = [word for word in jieba.cut(content) if word not in stopwords]

words = [word for word in words if len(word) >= 2]

words\_list.append(' '.join(words))

return words\_list

def count\_words(words\_list):

words\_count = Counter(' '.join(words\_list).split())

return words\_count

## 附录1.4数据可视化模块

### 附录1.4.1 微博数据可视化模块

<template>

<div class="container">

<a-button type="primary" @click="$router.push('/home')">回到主页</a-button>

<h1>Task Results</h1>

<a-layout>

<a-layout-sider :width="200">

<a-radio-group class="tabs" v-model="activeName" @change="(e) => handleTabClick(e.target.value)">

<a-radio-button v-for="tab in tabs" :key="tab.key" :value="tab.key">{{tab.title}}</a-radio-button>

</a-radio-group>

</a-layout-sider>

<a-layout-content>

<div class="image-container">

<div v-for="tab in visibleImageTabs" :key="tab.key" class="image">

<img :src="images[tab.key]" :alt="tab.title" />

<div class="image-label">{{ tab.title }}</div>

</div>

</div>

</a-layout-content>

</a-layout>

</div>

</template>

<script>

import axiosInstance from "@/api/axiosInstance";

import { message } from "ant-design-vue";

import { ref } from "vue";

export default {

name: "WeiboRepost",

setup() {

return {

activeName: ref('post\_sources'),

};

},

computed: {

visibleImageTabs() {

return this.tabs.filter(tab => this.visibleTabs[tab.key]);

}

},

data() {

return {

images: {},

tabs: [

{ key: 'post\_sources', title: '博文的发布设备分布' },

{ key: 'sentiment\_histogram', title: '博文的情感分布' },

{ key: 'posting\_frequency', title: '博文发布频率' },

{ key: 'user\_engagement', title: '博文用户参与度' },

{ key: 'top\_hashtags', title: '前十标签' },

],

visibleTabs: {

post\_sources: false,

sentiment\_histogram: false,

posting\_frequency: false,

user\_engagement: false,

top\_hashtags: false,

},

}

},

methods: {

async handleTabClick(tabKey) {

this.taskId = this.$route.params.taskId;

this.visibleTabs[tabKey] = !this.visibleTabs[tabKey]

if (!this.images[tabKey]) {

axiosInstance.get(`/analyze/result/${this.taskId}/${tabKey}`, {

responseType: 'arraybuffer',

headers: {'Accept': 'image/png'}

})

.then(response => {

const base64 = btoa(

new Uint8Array(response.data).reduce(

(data, byte) => data + String.fromCharCode(byte),

'',

),

);

this.images[tabKey] = `data:image/png;base64,${base64}`

message.success('加载成功',0.2)

})

.catch(error => {

console.error(error);

});

}

}

},

async mounted() {

this.taskId = this.$route.params.taskId;

}

}

</script>

### 附录1.4.2 京东数据可视化模块

<template>

<div class="container">

<a-button type="primary" @click="$router.push('/home')">回到主页</a-button>

<h1>Task Results</h1>

<a-radio-group class="tabs" v-model="activeName" @change="(e) => handleTabClick(e.target.value)">

<a-radio-button v-for="tab in tabs" :key="tab.key" :value="tab.key">{{tab.title}}</a-radio-button>

</a-radio-group>

<div class="image-container">

<img v-if="images['wordcloud'] && visibleTabs['wordcloud']" class="image" :src="images['wordcloud']" alt="词云">

</div>

</div>

</template>

<script>

## 附录1.5任务调度与通知模块

### 附录1.5.1 任务调度模块

using System; import os

import signal

from scrapy.crawler import CrawlerRunner

from scrapy.utils.project import get\_project\_settings

from twisted.internet import reactor

from celery import Celery

from app.config.config import BROKER\_URL, BACKEND\_URL

celery = Celery('tasks', broker=BROKER\_URL, backend=BACKEND\_URL)

celery.conf.task\_routes = {'celery\_task.tasks.\*': {'queue': 'celery'}}

import logging

from scrapy.utils.log import configure\_logging

configure\_logging(install\_root\_handler=False)

logging.basicConfig(

level=logging.DEBUG,

format='%(asctime)s %(levelname)s %(message)s',

handlers=[logging.StreamHandler()],

)

@celery.task(name='tasks.run\_weibo\_user\_spider', bind=True)

def run\_weibo\_user\_spider(self, user\_ids: list = None, cookie: str = None):

task\_id = self.request.id

task\_type = 'weibo\_user'

spider\_cls = UserSpider

spider\_kwargs = {}

if user\_ids:

spider\_kwargs['user\_ids'] = user\_ids

if cookie:

spider\_kwargs['cookie'] = cookie

if task\_id:

spider\_kwargs['task\_id'] = task\_id

os.environ['SCRAPY\_SETTINGS\_MODULE'] = 'weibospider.settings' # set the settings module for weibospider

settings = get\_project\_settings()

runner = CrawlerRunner(settings)

deferred = runner.crawl(spider\_cls, \*\*spider\_kwargs)

def stop\_reactor():

reactor.stop()

deferred.addBoth(stop\_reactor)

def stop\_task(signum, frame):

reactor.callFromThread(reactor.stop)

signal.signal(signal.SIGTERM, stop\_task)

try:

reactor.run()

except Exception as e:

return {'message': f'Spider {task\_type} failed to run.'}

# stats = runner.stats.get\_stats()

# tasks\_collection.update\_one({'task\_id': task\_id}, {'$set': {'stats': stats}}, upsert=True)

return {'message': f'Spider {task\_type} finished running.'}

@celery.task(name='tasks.run\_weibo\_search\_spider', bind=True)

def run\_weibo\_search\_spider(self, keywords: str = None,

start\_time: str = None, end\_time: str = None,

is\_sort\_by\_hot: bool = True,

is\_search\_with\_specific\_time\_scope: bool = False,

cookie: str = None):

task\_id = self.request.id

spider\_cls = SearchSpider

spider\_kwargs = {}

if keywords:

spider\_kwargs['keywords'] = keywords

if start\_time:

spider\_kwargs['start\_time'] = start\_time

if end\_time:

spider\_kwargs['end\_time'] = end\_time

if cookie:

spider\_kwargs['cookie'] = cookie

if task\_id:

spider\_kwargs['task\_id'] = task\_id

spider\_kwargs['is\_sort\_by\_hot'] = is\_sort\_by\_hot

spider\_kwargs['is\_search\_with\_specific\_time\_scope'] = is\_search\_with\_specific\_time\_scope

os.environ['SCRAPY\_SETTINGS\_MODULE'] = 'weibospider.settings' # set the settings module for weibospider

settings = get\_project\_settings()

runner = CrawlerRunner(settings)

deferred = runner.crawl(spider\_cls, \*\*spider\_kwargs)

def stop\_reactor():

reactor.stop()

deferred.addBoth(stop\_reactor)

# Add signal handler to stop the reactor when the Celery worker is terminated

def stop\_task(signum, frame):

reactor.callFromThread(reactor.stop)

signal.signal(signal.SIGTERM, stop\_task)

reactor.run()

return {'message': f'Spider search finished running.'}

### 附录1.5.2 通知模块

import smtplib

from email.mime.text import MIMEText

from weibospider import settings

def send\_email(subject, body):

msg = MIMEText(body)

msg['Subject'] = subject

msg['From'] = settings.MAIL\_FROM

msg['To'] = ','.join(settings.MAIL\_TO)

s = smtplib.SMTP(settings.MAIL\_HOST, settings.MAIL\_PORT)

if settings.MAIL\_TLS:

s.starttls()

if settings.MAIL\_USER and settings.MAIL\_PASS:

s.login(settings.MAIL\_USER, settings.MAIL\_PASS)

s.sendmail(settings.MAIL\_FROM, settings.MAIL\_TO, msg.as\_string())

s.quit()

def close\_spider(self, spider):  
 stats\_info = spider.crawler.stats.get\_stats()  
 task\_id = spider.task\_id  
 finish\_time = datetime.datetime.now()  
 stats\_info['finish\_time'] = finish\_time  
 print(f"Stats info in pipeline: {stats\_info}")  
 self.task\_db['tasks'].update\_one(  
 {'task\_id': task\_id},  
 {'$set': {'status': 'finished', 'stats': stats\_info}}  
 )  
 self.client.close()  
 subject = f"您的名称为{spider.name}的任务完成，任务号为 {task\_id}!"  
 body = f"数据收集任务完成，请前往您的任务中心进行下一步操作。 可以前往https://scrapeops.io/app/dashboard查看数据收集程序的状态. 运行状态:{stats\_info}"  
 send\_email(subject, body)

## 附录1.6 Dockerfile和docker-compose.yml

### 附录1.6.1 backend的Dockerfile

FROM python:3.10

ENV PYTHONPATH=/backend/app

ENV PYTHONPATH=/backend/jdspider

ENV PYTHONPATH=/backend/weibospider

ENV PYTHONPATH=/backend/pre\_process

WORKDIR /backend

COPY ./requirements.txt /backend/requirements.txt

COPY pre\_process/SimHei.ttf /usr/share/fonts/truetype/

RUN apt-get update && apt-get install -y tzdata

ENV TZ=Asia/Shanghai

RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone

RUN fc-cache -f -v

RUN pip install -U pip

RUN pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple

RUN pip install --upgrade -r /backend/requirements.txt

COPY ./app /backend/app

COPY ./jdspider /backend/jdspider

COPY ./weibospider /backend/weibospider

COPY ./pre\_process /backend/pre\_process

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8080"]

EXPOSE 8080

### 附录1.6.2 frontend的Dockerfile

FROM node:14

WORKDIR /frontend/app

COPY package\*.json ./

RUN npm install -g npm

RUN npm install

COPY . .

RUN npm run build

EXPOSE 80

CMD ["npm", "run", "serve"]

### 附录1.6.3 mongodb的Dockerfile

FROM mongo:4.4.4-bionic

EXPOSE 27017

### 附录1.6.4 mysql的Dockerfile

FROM mysql:8.0

MAINTAINER zm1575098153@gmail.com

ENV MYSQL\_ROOT\_PASSWORD Zm.1575098153

ADD user.sql /docker-entrypoint-initdb.d

EXPOSE 3306

### 附录1.6.5 mysql的user.sql

USE mysql;

CREATE DATABASE IF NOT EXISTS fast;

ALTER USER 'root' IDENTIFIED WITH mysql\_native\_password BY 'Zm.1575098153';

USE fast;

CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(50) NOT NULL,

email VARCHAR(100) NOT NULL UNIQUE,

password VARCHAR(100) NOT NULL

);

### 附录1.6.6 rabbitmq的Dockerfile

FROM rabbitmq:latest

ENV RABBITMQ\_DEFAULT\_USER=admin \

RABBITMQ\_DEFAULT\_PASS=admin

RUN rabbitmq-plugins enable --offline rabbitmq\_management

EXPOSE 5672 15672

CMD ["rabbitmq-server"]

### 附录1.6.7 redis的Dockerfile

FROM redis:latest

EXPOSE 6379

CMD ["redis-server"]

### 附录1.6.8 elasticsearch的Dockerfile

FROM docker.elastic.co/elasticsearch/elasticsearch:7.15.0

EXPOSE 9200

### 附录1.6.9 prometheus的配置

global:

scrape\_interval: 15s

evaluation\_interval: 15s

scrape\_configs:

- job\_name: 'backend'

static\_configs:

- targets: ['backend:8080']

- job\_name: 'frontend'

static\_configs:

- targets: ['frontend:80']

- job\_name: 'celery'

static\_configs:

- targets: ['celery:5555']

### 附录1.6.10 docker-compose.yml

services:

frontend:

build: ./frontend

ports:

- "80:80"

depends\_on:

- backend

environment:

API\_URL: http://backend:8080

prometheus:

image: prom/prometheus

ports:

- "9090:9090"

volumes:

- ./prometheus:/etc/prometheus/

command:

- '--config.file=/etc/prometheus/prometheus.yml'

- '--storage.tsdb.path=/prometheus'

- '--web.console.libraries=/usr/share/prometheus/console\_libraries'

- '--web.console.templates=/usr/share/prometheus/consoles'

depends\_on:

- backend

backend:

build: ./backend

ports:

- "8080:8080"

environment:

- CELERY\_BROKER\_URL=amqp://guest:guest@rabbitmq:5672/

- RABBITMQ\_DEFAULT\_USER=admin

- RABBITMQ\_DEFAULT\_PASS=admin

- RABBITMQ\_DEFAULT\_VHOST=/

- BACKEND\_URL=rpc://

- BROKER\_URL=amqp://admin:admin@localhost:5672/

- REDIS\_BACKEND\_URL=redis://redis:6379/0

- DB\_USER=root

- DB\_PASSWORD=Zm.1575098153

- DB\_HOST=mysql

- DB\_PORT=3306

- DB\_NAME=fast

- DB\_CHARSET=utf8

- DB\_AUTH\_PLUGIN=mysql\_native\_password

- CELERY\_BROKER\_URL=amqp://admin:admin@rabbitmq:5672/

- CELERY\_RESULT\_BACKEND=rpc://

- CELERY\_LOG\_LEVEL=debug

- ELASTICSEARCH\_URL=http://elasticsearch:9200

- REDIS\_HOST=redis

- FLOWER\_BASIC\_AUTH=admin:password

- MONGODB\_URL=mongodb://mongodb:27017

depends\_on:

- mysql

- mongodb

- flower

- elasticsearch

celery:

build: ./backend

environment:

- CELERY\_BROKER\_URL=amqp://admin:admin@rabbitmq:5672/

- CELERY\_RESULT\_BACKEND=rpc://

- CELERY\_LOG\_LEVEL=info

- MONGODB\_URL=mongodb://mongodb:27017

depends\_on:

- rabbitmq

command: celery -A app.celery\_task.tasks worker --loglevel=info

flower:

build: ./backend

environment:

- CELERY\_BROKER\_URL=amqp://admin:admin@rabbitmq:5672//

ports:

- "5555:5555"

depends\_on:

- celery

command: flower --app=app.celery\_task.tasks --broker=amqp://admin:admin@rabbitmq:5672// --port=5555

mysql:

build: ./mysql

command: --default-authentication-plugin=mysql\_native\_password

ports:

- "3306:3306"

environment:

MYSQL\_ROOT\_PASSWORD: Zm.1575098153

mongodb:

build: ./mongodb

ports:

- "27017:27017"

volumes:

- ./mongodb/data:/data/db

elasticsearch:

build: ./elasticsearch

environment:

- discovery.type=single-node

ports:

- "9200:9200"

- "9300:9300"

rabbitmq:

build: ./rabbitmq

ports:

- "5672:5672"

- "15672:15672"

redis:

build: ./redis

ports:

- "6379:6379"