**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（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估信息收集和分析系统。 在本文中，我们提出了一种基于 NLP 的系统的设计和实现，该系统利用 Spacy 进行文本分析，并利用数据可视化库 Echarts 进行数据呈现。

拟议的系统由两个主要部分组成：数据收集模块和分析模块。数据收集模块使用网络抓取技术从社交媒体平台和电商平台等各种来源收集数据。分析模块使用 Spacy 处理文本数据以提取有价值的信息，例如情绪、产品特性和用户偏好。 然后使用Echarts提供的可视化工具（例如条形图、饼图和折线图）呈现提取的信息。

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并将其标记以供审查。

为了评估所提出系统的性能，我们对我们系统收集的文本数据集进行了实验。 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 可视化呈现，为一些应用提供有价值的见解。

总的来说，所提出的系统提供了一个强大的工具，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抓取以及用于数据分析的 Echarts 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。

**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 evaluation information collection and analysis system. In this paper, we propose a design and implementation of an NLP-based system that utilizes Spacy for text analysis and matplotlib, a data visualization library, for data presentation.

The proposed system consists of two main components: the data collection module and the analysis module. The data collection module uses web scraping techniques to gather data from various sources such as social media platforms and online marketplaces. The analysis module processes the textual data using Spacy to extract valuable information such as sentiment, product features, and user preferences. The extracted information is then presented using Echarts and matplotlib provided visualization tools such as bar charts, pie charts, and line graphs.

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 and flag it for review.

To evaluate the performance of the proposed system, we conducted experiments on a dataset of textual data collected by our system. The results show that the system can accurately extract information from textual data and present it using Echarts visualizations, 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 visualizations for data analysis, which can help organizations make better decisions and improve their services.

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

**目 录**

[1 概述 2](#_Toc133244897)

[1.1 开发背景 2](#_Toc133244898)

[1.2 开发意义 3](#_Toc133244899)

[1.3 课题开发的工具及技术 4](#_Toc133244900)

[2 系统分析 7](#_Toc133244901)

[2.1 需求分析 7](#_Toc133244902)

[2.2 可行性分析 8](#_Toc133244903)

[2.3 数据库选择 9](#_Toc133244904)

[2.4 系统目标 10](#_Toc133244905)

[3 系统的概要设计 10](#_Toc133244906)

[3.1 总体设计 10](#_Toc133244907)

[3.2 系统架构 11](#_Toc133244908)

[3.3 系统的模块功能划分 13](#_Toc133244909)

[3.4 系统的接口设计 14](#_Toc133244910)

[3.5 系统涉及的数据结构 14](#_Toc133244911)

[3.6 系统的错误处理 15](#_Toc133244912)

[4 系统的详细设计 16](#_Toc133244913)

[4.1 系统接口设计 16](#_Toc133244914)

[4.2 系统登录模块设计 17](#_Toc133244915)

[4.3 系统数据收集模块设计 20](#_Toc133244916)

[4.3 普通用户模块设计 24](#_Toc133244917)

[5 系统实现 25](#_Toc133244918)

[5.1 系统接口的实现 25](#_Toc133244919)

[5.2 系统数据收集模块的实现 25](#_Toc133244920)

[5.3 系统数据展示模块的实现 25](#_Toc133244921)

[5.4 系统数据分析可视化模块的实现 25](#_Toc133244922)

[5.5 系统通知模块的实现 25](#_Toc133244923)

[5.6 系统Docker部署的实现 25](#_Toc133244924)

[6 系统部署与测试 25](#_Toc133244925)

[6.1 系统部署 25](#_Toc133244926)

[6.2 系统登录模块测试 25](#_Toc133244927)

[6.3 系统数据收集模块测试 25](#_Toc133244928)

[6.4 系统数据展示模块测试 25](#_Toc133244929)

[6.5 系统数据分析可视化模块测试 25](#_Toc133244930)

[6.6 系统通知模块测试 25](#_Toc133244931)

[7 系统的发展方向及应用前景 26](#_Toc133244932)

[8 结束语 26](#_Toc133244933)

[参考文献 28](#_Toc133244934)

[致 谢 29](#_Toc133244935)

[附 录 30](#_Toc133244936)

[附录1 源程序清单 30](#_Toc133244937)

[附录1.1 系统登录模块 30](#_Toc133244938)

[附录1.2 接警人模块 30](#_Toc133244939)

[附录1.3 普通用户模块 30](#_Toc133244940)

**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：

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。

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。

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：

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。

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。

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：

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。

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。

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。

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。

智能化爬虫技术的发展将会进一步推动网络爬虫技术的发展和应用。随着数据量的不断增加，网络爬虫技术将会在许多领域得到广泛的应用，例如市场调研、舆情监测和数据挖掘等。同时，网络爬虫技术也需要注意数据安全和隐私保护等问题，以保证用户的数据安全和隐私不受侵犯。

## 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：

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。

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。

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。

推动相关技术的研究和应用：该课题涉及到文本处理、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。

服务于社会公共利益：通过对于政府公共服务的评价信息进行分析，可以帮助政府更好地了解公众对于政府服务的需求和反馈，从而提高公共服务的质量和效率，为社会公共利益服务。

## 1.3 课题开发的工具及技术

### 1.3.1 开发工具

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。

MongoDB Compass

MongoDB Compass 是 MongoDB 的图形用户界面工具，MongoDB 是一种 NoSQL 面向文档的数据库系统。 它允许开发人员以可视化和直观的方式与 MongoDB 数据进行交互，从而更轻松地创建、修改和分析数据。 借助 MongoDB Compass，开发人员还可以查询和分析数据，以及创建和管理索引。

Visual Studio Code

Visual Studio Code 是微软开发的一款流行的开源代码编辑器。 它支持包括 Python 在内的多种编程语言，并提供语法高亮显示、代码完成、调试和 Git 集成等广泛的功能。 它还支持扩展，方便开发者自定义和扩展编辑器的功能。

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。

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，并将数据存储到 MongoDB 中。

NLP 模块使用 Spacy 来处理评价信息，提取关键信息并将其存储到 Elasticsearch 中。

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 和 Pandas 形成了一个强大的数据可视化和分析工具。

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。 另一方面，

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

Scrapy

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。使用Scrapy，开发者可以定义爬虫规则和提取数据的方式，将抓取的数据存储到本地或者远程数据库中，同时支持多种爬虫任务调度策略。

MongoDB

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。MongoDB具有简单易用的API和查询语言，同时支持全文搜索和地理位置查询等功能。在基于NLP的可视化评价信息收集与分析系统中，MongoDB可以作为数据存储和管理平台，将从爬虫中获取的评价信息存储到数据库中。

Spacy

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。它使用Cython编写，性能优异，同时提供了简单易用的API和预训练的模型，可以方便地进行文本处理和分析。在基于NLP的可视化评价信息收集与分析系统中，Spacy可以作为NLP模块的核心，对爬虫获取的评价信息进行清洗、处理、分类和实体识别等任务，并将处理结果存储到数据库中。

Elasticsearch

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。在基于NLP的可视化评价信息收集与分析系统中，Elasticsearch可以作为可视化模块的数据源，将处理后的评价信息进行可视化展示，并提供数据查询和聚合等功能。

Vue.js

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。在基于NLP的可视化评价信息收集与分析系统中，Vue.js可以作为可视化模块的前端界面，实现数据展示和用户交互等功能。

FastAPI

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。在基于NLP的可视化评价信息收集与分析系统中，FastAPI可以作为后端模块的核心，实现用户认证、数据查询和分析等功能，并提供API文档和测试工具。

Celery

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。在基于NLP的可视化评价信息收集与分析系统中，Celery可以作为任务调度和通知模块的核心，实现爬虫任务调度、数据处理和通知等功能。

RabbitMQ

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。在基于NLP的可视化评价信息收集与分析系统中，RabbitMQ可以作为消息代理和任务队列的核心，实现爬虫任务调度、数据处理和通知等功能。

Matplotlib

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。

Pandas

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。

Redis

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。在基于NLP的可视化评价信息收集与分析系统中，Redis可以作为缓存和消息代理，缓存一些常用的数据以加速访问，并作为RabbitMQ的消息代理。

Docker

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。在基于NLP的可视化评价信息收集与分析系统中，Docker可以作为部署和管理平台，将每个模块打包成Docker镜像，并使用Docker Swarm进行部署和管理，可以方便地进行应用程序的扩展和迁移。

Docker Swarm

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。在基于NLP的可视化评价信息收集与分析系统中，Docker Swarm可以作为集群管理和应用程序自动伸缩的平台，可以实现高可用性和可扩展性。

# 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：

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。

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。

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。

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。

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：

爬虫模块：实现自动抓取目标网站的评价信息，并将数据存储到MongoDB中。

NLP模块：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中。

可视化模块：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。

后端模块：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。

任务调度和通知模块：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：

爬虫模块需求：

（1）自动抓取目标网站的评价信息；

（2）支持多个目标网站的抓取；

（3）将数据存储到MongoDB中。

NLP模块需求：

（1）支持对评价信息的清洗、处理、分析等任务；

（2）将处理结果存储到Elasticsearch中。

可视化模块需求：

（1）支持对MongoDB中的评价信息进行可视化展示；

（2）支持数据查看和分析等功能；

（3）支持多种指标的可视化功能。

（4）将数据分析结果存储在Redis缓存中，以加速访问。

后端模块需求：

（1）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；

（2）完成对其他模块的支撑功能。

任务调度和通知模块需求：

1. 实现爬虫任务调度功能；
2. 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，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；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；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；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；Docker Swarm是一个容器编排和管理平台，可以使得系统的部署和管理更加方便和高效。

### 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

数据可行性考虑以下几个方面：

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。

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。

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。

安全可行性考虑以下几个方面：

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。

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。

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：

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。

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。

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。

查询能力：MongoDB 拥有强大而灵活的查询语言，可以实现复杂的查询和聚合操作。这对于需要对文本数据进行复杂分析和处理的系统来说非常重要。

与其他工具集成：MongoDB 与基于 NLP 的系统中常用的其他工具和技术很好地集成，例如 Python、NLTK 和 spaCy。

## 2.4 系统目标

本系统的目标是：

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评估（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。

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。

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。

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。

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。

不断优化和改进系统功能和性能，以满足用户需求和市场竞争。

# 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：

1）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。

2)处理：系统应使用自然语言处理技术来分析文本数据。

3)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

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。

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：

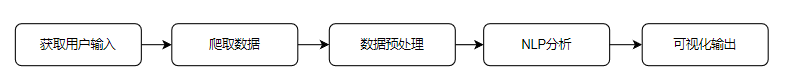


图3-1 系统处理流程

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下：

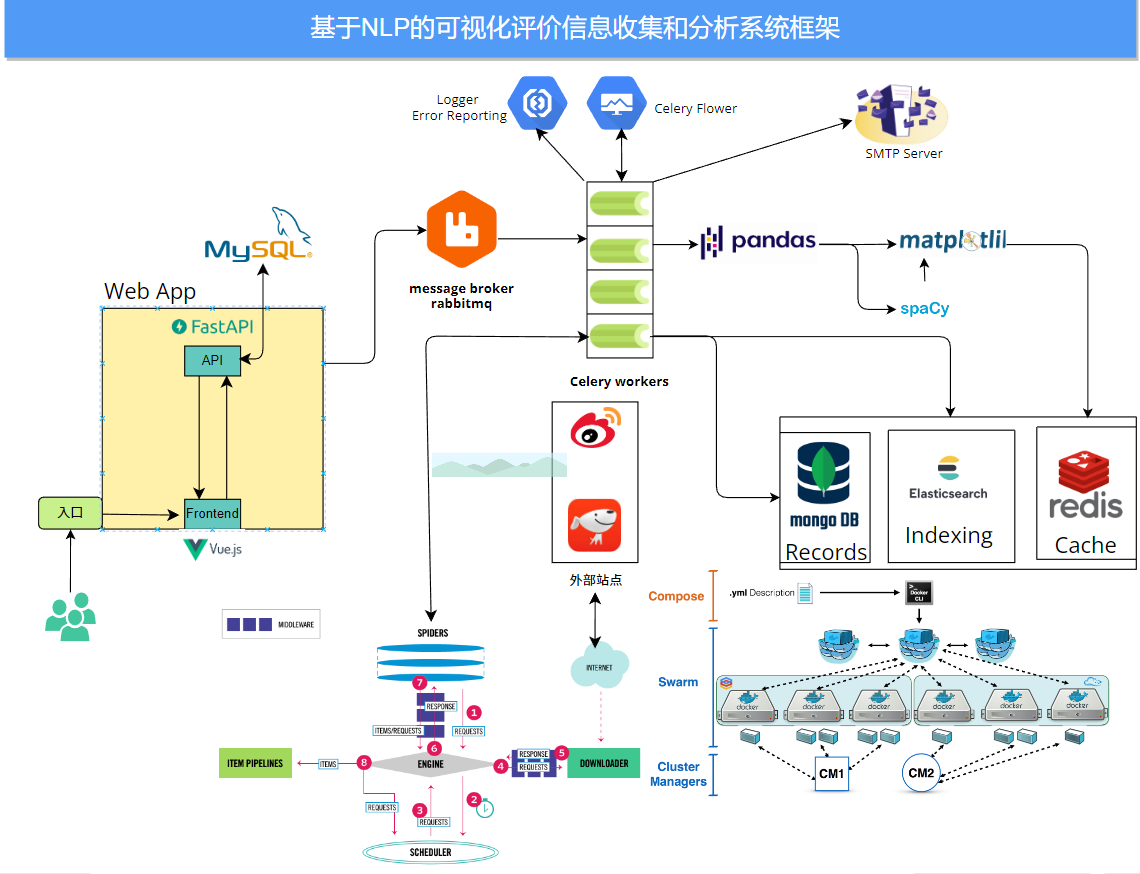


图3-2 系统架构图

基于NLP的可视化评估信息采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。系统的层次结构可以分解为若干层，每一层都有自己的一套功能和技术。

用户界面层：

用户界面层负责为用户提供与系统交互的基于网络的界面。它使用Vue.js 构建，并使用RESTful API 调用与后端API 层通信。

后端API 层：

后端API层提供了用户界面层与系统各种处理组件之间的接口。它使用FastAPI 构建，并利用Celery 和 RabbitMQ 进行任务管理和消息队列。

数据收集层：

数据收集层负责从各种来源收集评估数据，例如社交媒体或电商平台。它使用基于Python的网络爬虫框架Scrapy 实现，并利用RabbitMQ 进行消息队列。

数据处理层：

数据处理层负责使用自然语言处理和机器学习技术分析收集到的数据。它利用 spaCy 进行 NLP 处理，利用Pandas 进行数据处理。Redis用作缓存以提高性能，MongoDB用作存储处理后数据的主数据库。

数据可视化层：

数据可视化层负责以视觉上吸引人且易于理解的格式呈现分析后的数据。它利用 Elasticsearch 对处理过的数据进行实时搜索和索引，并使用matplotlib生成可视化效果，并且存储在Redis中，前端从Redis中获取。

基础设施层：

基础设施层负责管理系统组件的部署和扩展。它使用容器编排工具 Docker Swarm 实现，在需要的情况下可以利用 AWS 和 GCP 等各种云服务进行托管。

层次如下图所示：

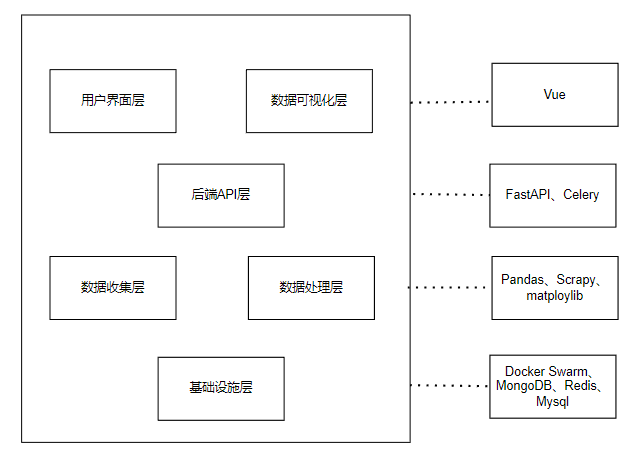


图3-3 系统层次图

系统各层之间的调用关系可以描述如下：

用户界面层调用后端API层发起处理任务，获取分析结果。

后端API层与数据采集层通信发起数据采集任务，与数据处理层通信发起数据处理任务。

数据处理层与数据可视化层通信以生成可视化效果，并与数据库层通信以存储处理后的数据。

基础架构层管理所有系统组件的部署和扩展。

实现技术类型：

在实现技术类型上，系统利用了多种开源工具和框架，如基于Python的Web框架（FastAPI）、网络爬虫框架（Scrapy）、任务队列系统（Celery、RabbitMQ）、大数据处理等框架（Pandas）、NLP 处理库（spaCy）、数据库管理系统（MongoDB、Redis、Elasticsearch）、前端库（Vue.js、D3.js）和容器编排工具（Docker Swarm）。

## 3.3 系统的模块功能划分

（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。

## 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 系统涉及的数据结构

设计要点：

基于NLP的可视化评价信息采集与分析系统利用多种数据结构对评价数据进行存储和处理。这些包括：

原始数据结构：该数据结构存储从各种来源收集的原始评价数据，例如基于文本的评论。此数据结构中的每条记录都包含数据项，例如数据源、时间戳和原始格式的实际数据。

处理后的数据结构：该数据结构存储使用 NLP 和 ML 技术分析后处理过的视觉评估数据。该数据结构中的每条记录都包含数据项，例如数据的来源、时间戳和结构化格式的分析数据。

可视化数据结构：此数据结构存储用于根据处理后的数据生成可视化的数据。此数据结构中的每条记录包含数据项，例如可视化类型、数据范围以及可视化所基于的处理数据。

物理结构要点：

系统中使用的每个数据结构中的每个数据项都有特定的存储要求、访问方法和物理关系。例如,原始数据结构中的数据项可能存储在MongoDB等面向文档的数据库中，具有CRUD操作等访问方式。 访问单元可以在文档级别，通过唯一标识符访问单个记录。该数据结构的存储区域可能是特定的数据库服务器，设计时考虑了数据分区和索引。

类似地，处理后的数据结构中的数据项可能存储在与原始数据结构相同的数据库服务器中，但具有不同的访问方法，例如用于复杂分析操作的 SQL 查询。 访问单元可以在记录级别，通过唯一标识符访问单个记录。该数据结构的存储区域可能是 MongoDB 数据库中的一个特定集合，设计时考虑了数据建模和索引。

可视化数据结构可以存储在单独的数据库中，例如Redis、Elasticsearch，具有搜索查询等访问方法。访问单元可能处于可视化级别，通过唯一标识符访问各个可视化。该数据结构的存储区域可能是 Redis、Elasticsearch 数据库中的特定索引，设计考虑了数据建模和查询性能。

与程序的关系：

每个数据结构都由系统中的软件程序通过适当的API和数据访问库进行访问和操作。例如，系统的数据收集组件可以访问原始数据结构，而数据处理和分析组件可以访问处理后的数据结构。可视化数据结构可以被系统的可视化组件访问，其基于处理的数据生成可视化结果。可以通过数据库查询和 API 调用来访问这些数据结构。

## 3.6 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：

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。

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。

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。

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。

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的截图：

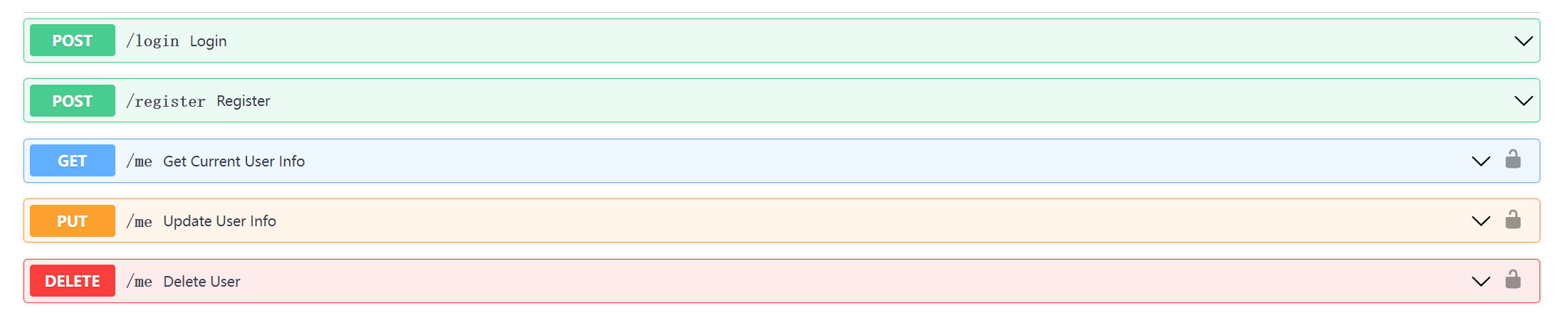


图4-1 用户登录注册及个人信息管理接口

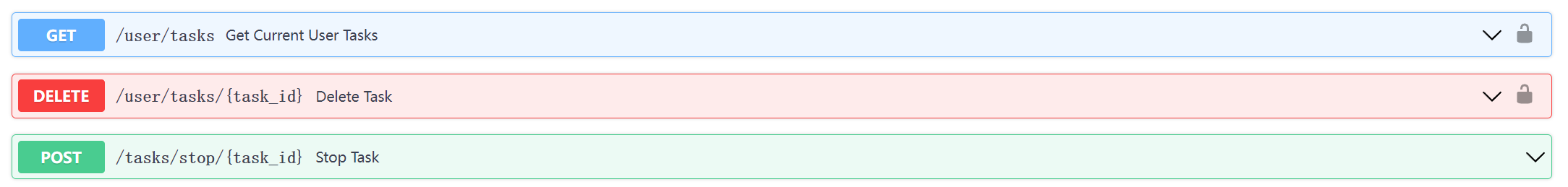


图4-2 任务管理接口

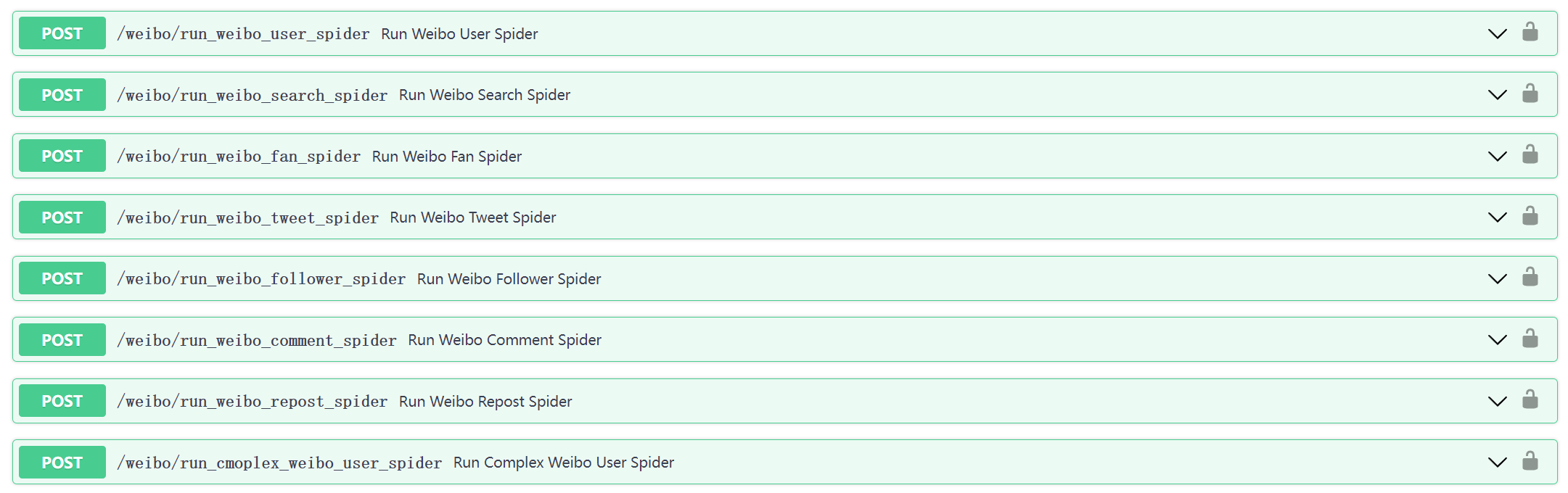


图4-3 微博爬虫任务接口

图4-4 微博爬虫数据接口

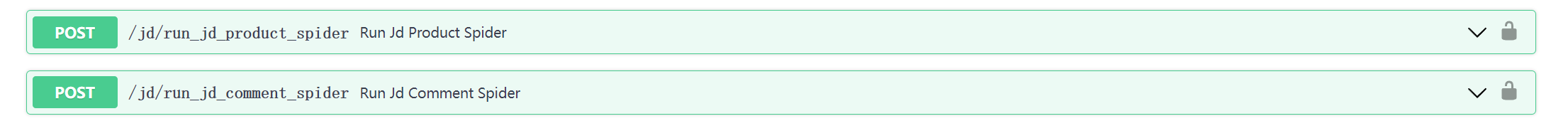


图4-5 京东爬虫任务接口

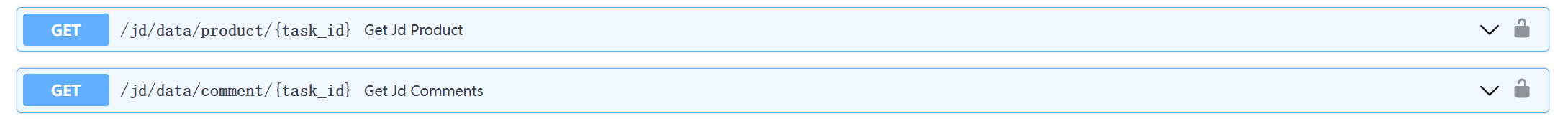
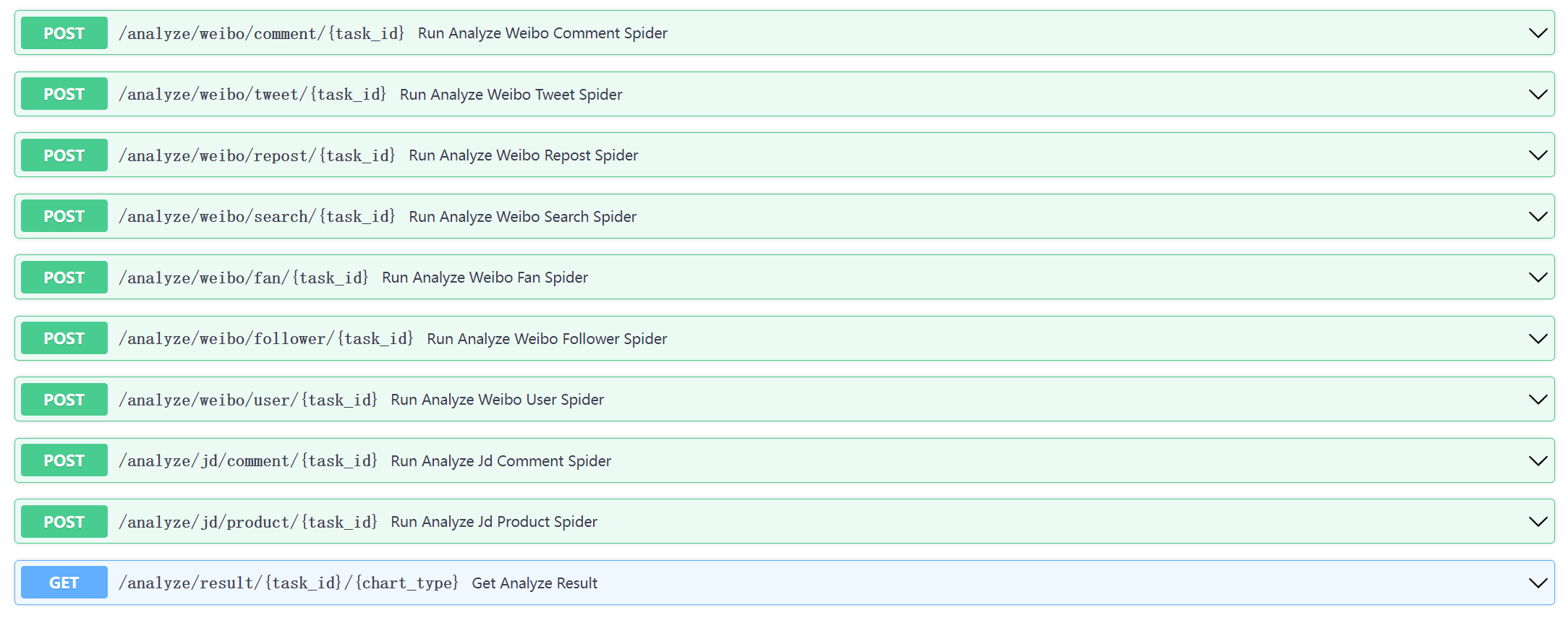


图4-6 京东数据接口

图4-7 分析任务接口以及分析结果返回接口

## 4.2 系统功能模块设计

该模块分为用户登录、用户注册、帮助中心等模块。

### 4.2.1 用户登录模块设计



图4-8 用户登录界面

登录页面的模板代码包含了一个表单，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

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。

以下是系统的主界面：

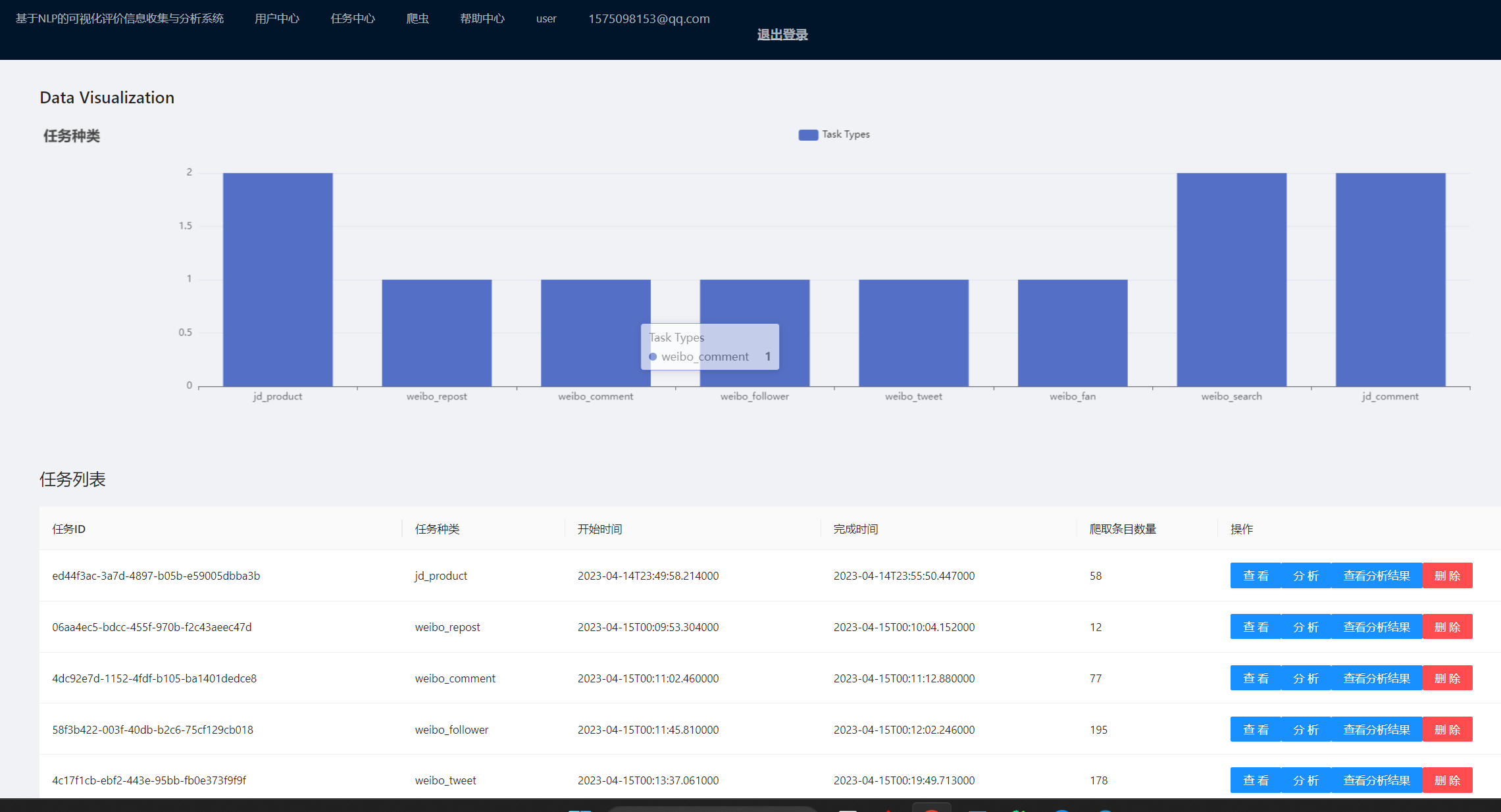


图4-10 主界面

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：

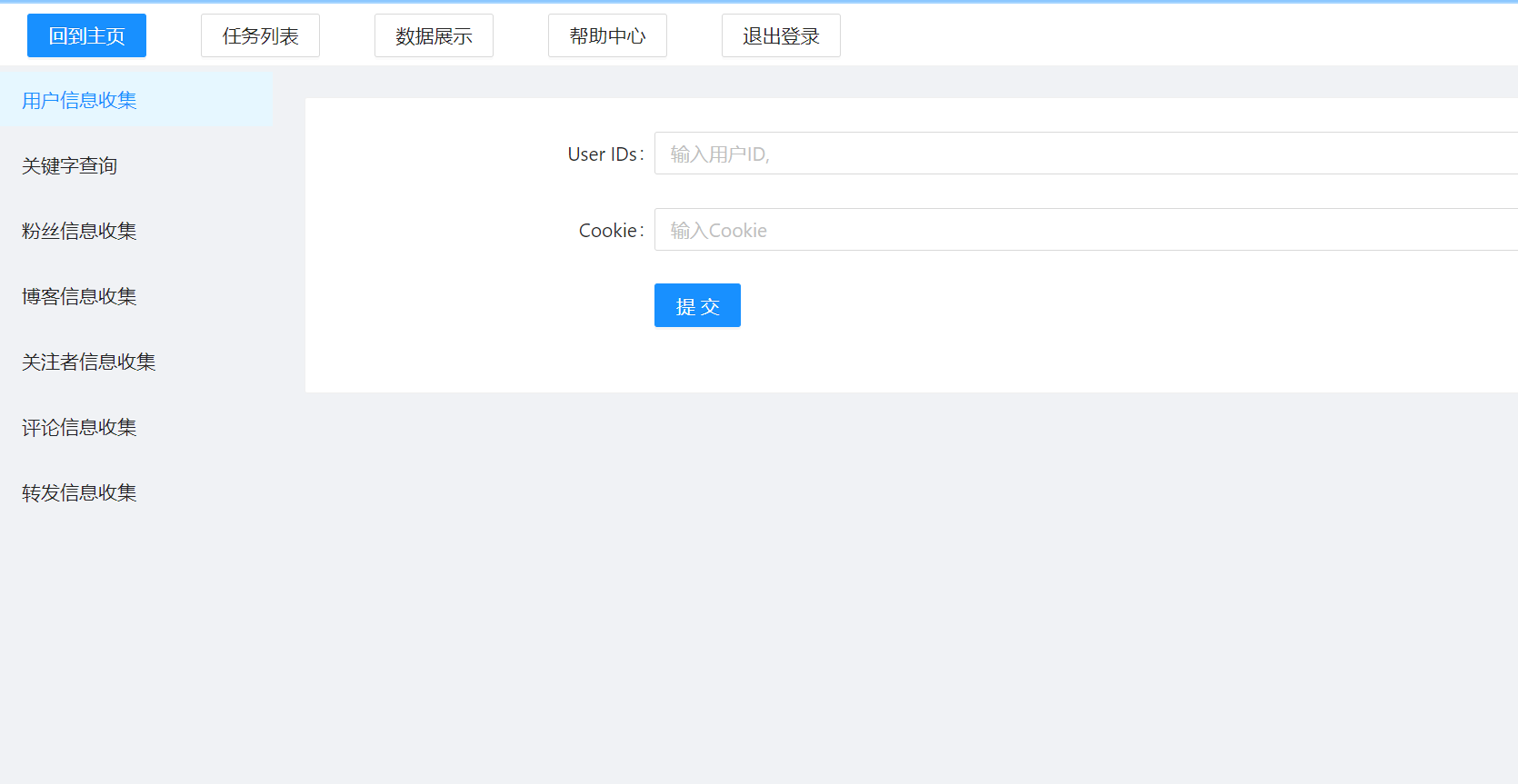


图4-11 微博爬虫界面

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

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。

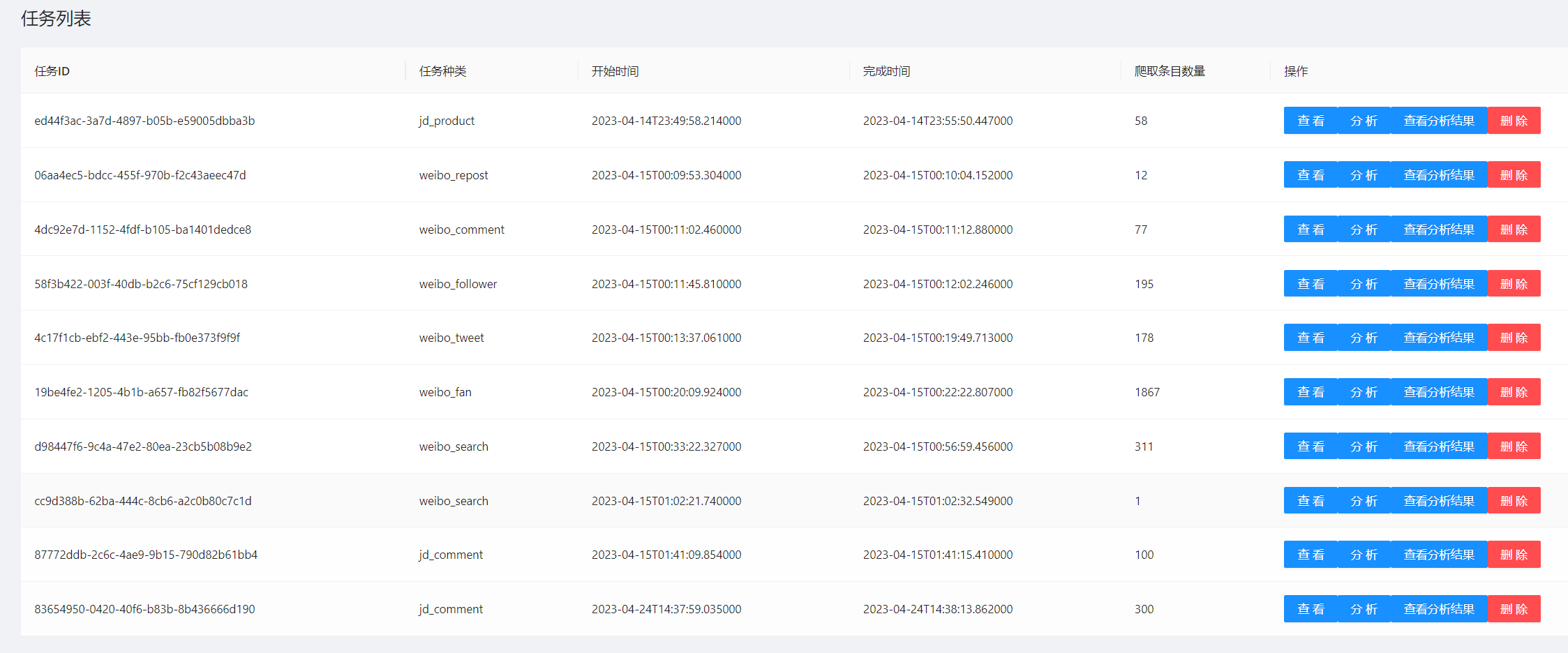


图4-13 数据分析模块的前端交互界面

## 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。

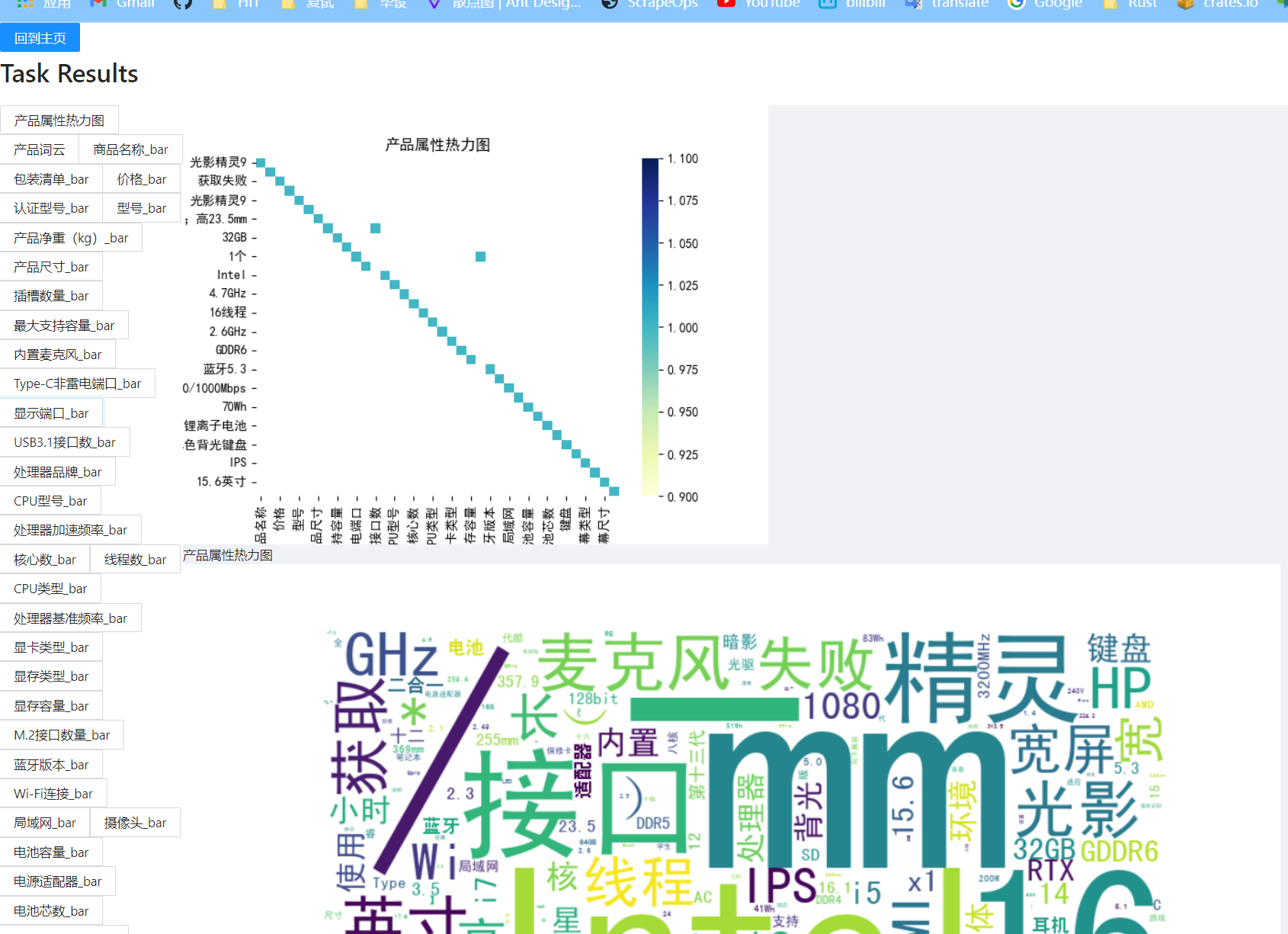


图4-14 京东商品信息可视化界面



图4-15 京东商品评价可视化界面

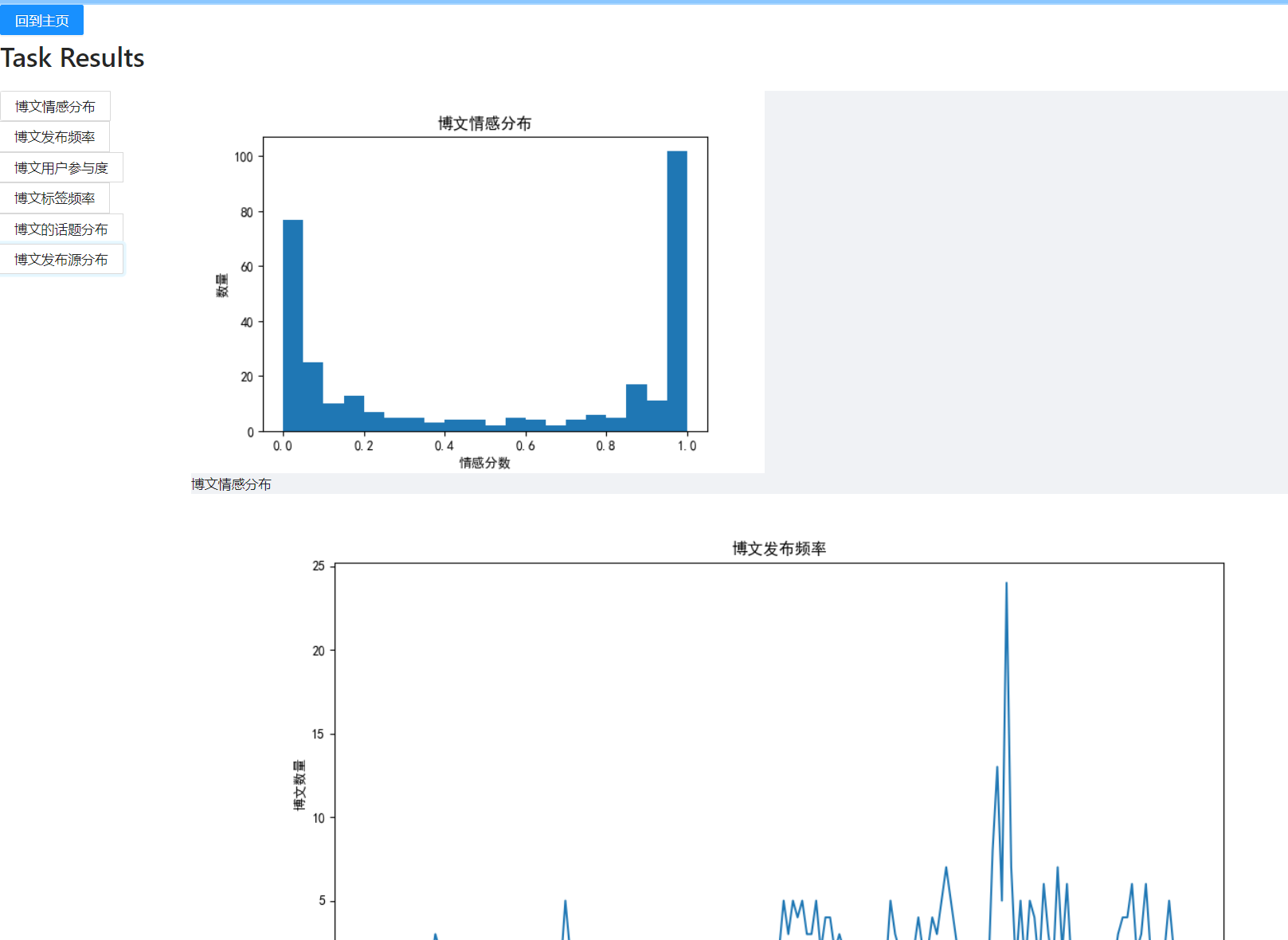


图4-16 微博博文信息可视化界面

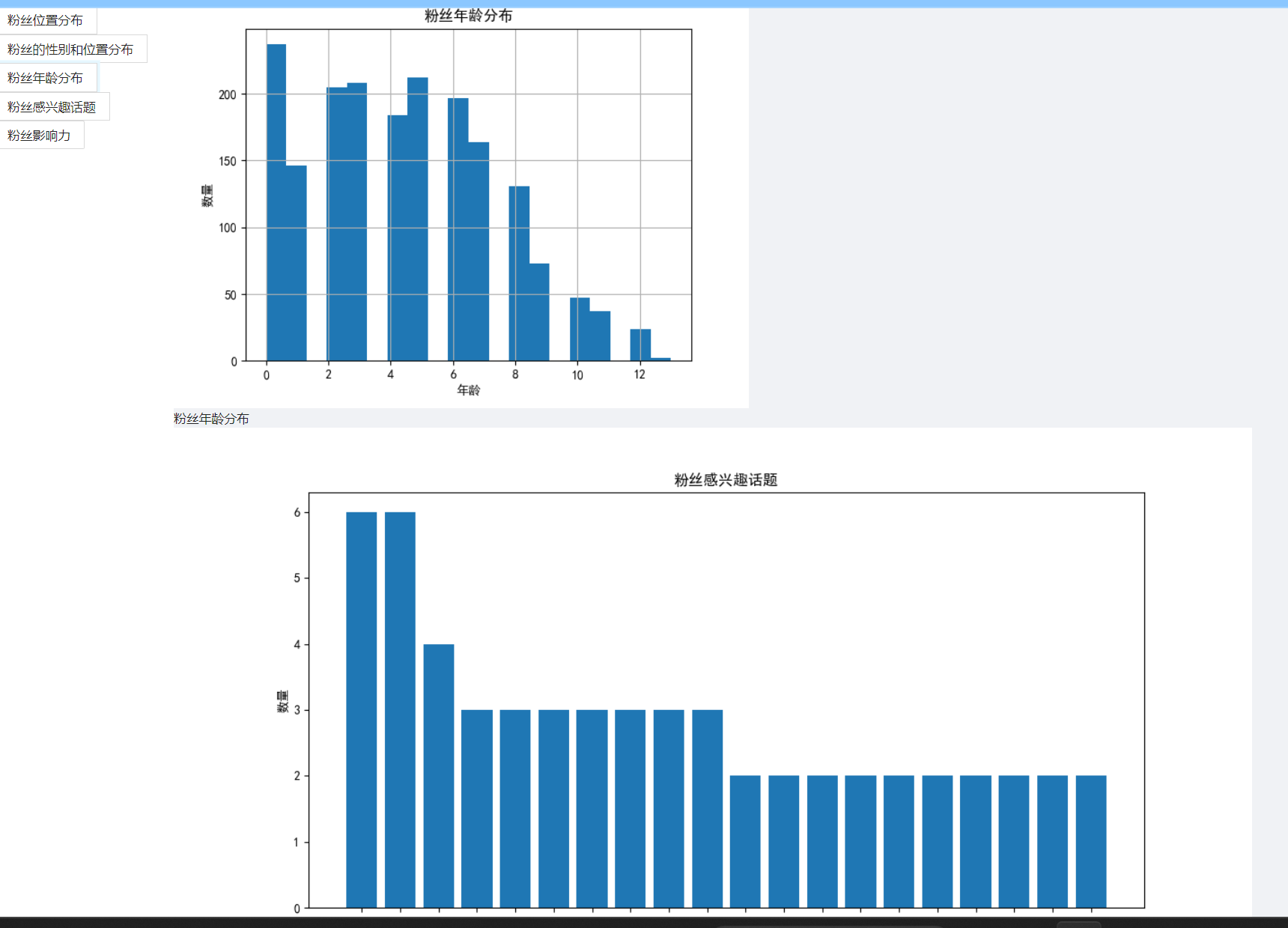


图4-17 微博粉丝信息可视化分析界面

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



图4-19 更新用户信息成功通知



图4-19 登录成功通知



图4-20 加载数据成功通知



图4-21 加载任务列表成功通知



图4-22 产品信息收集任务提交成功通知

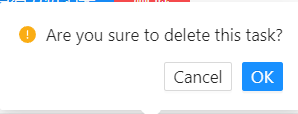


图4-23 任务删除提示

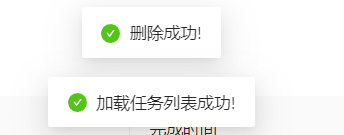


图4-24 任务删除成功提示

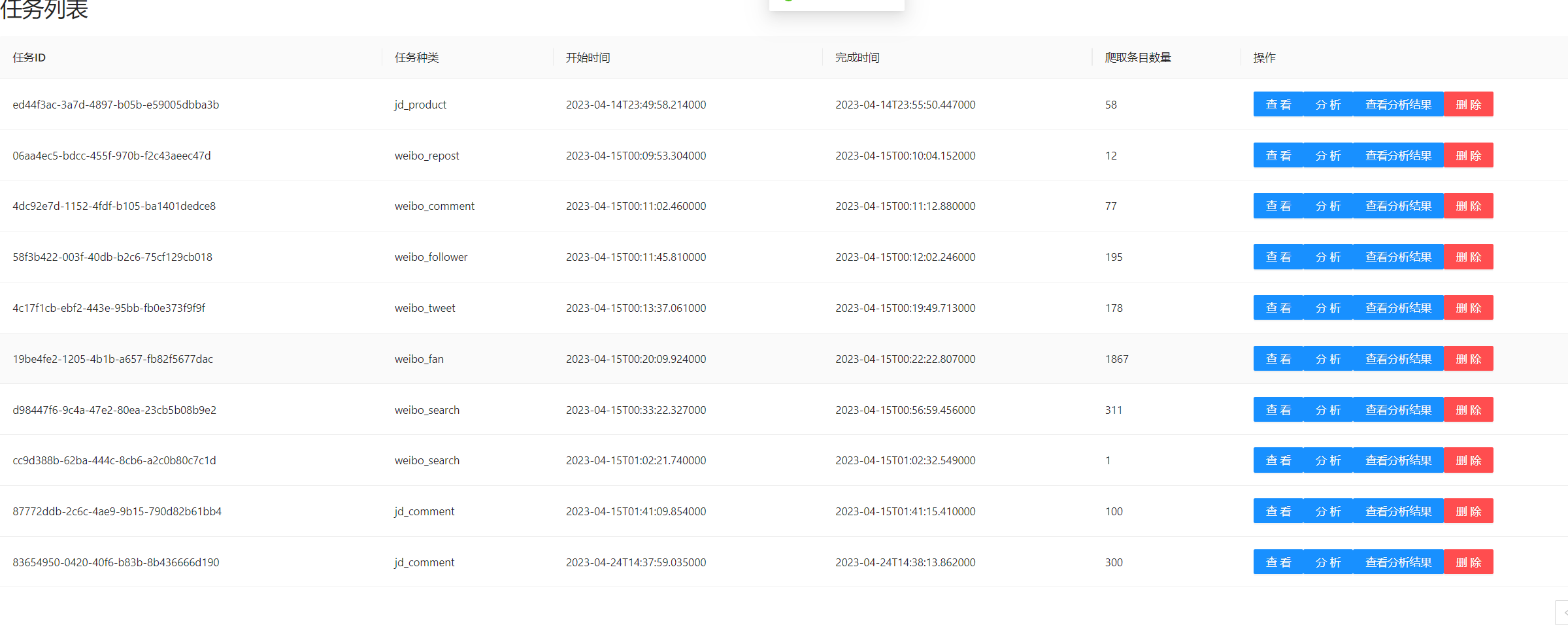


图4-25 任务管理界面

具体的任务调度由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。

main.py

import uvicorn

from fastapi import FastAPI

from fastapi.middleware.cors import CORSMiddleware

import logging

from app.routes.auth\_routes import router as auth\_router

from app.routes.protected\_routes import router as protected\_router

from app.routes.weibo\_routes import router as weibo\_router

from app.routes.user\_routes import router as user\_router

from app.routes.jd\_routes import router as jd\_router

from app.routes.analyze\_routes import router as analyze\_router

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)

# Enable CORS for all origins

app.add\_middleware(

CORSMiddleware,

allow\_origins=["\*"],

allow\_credentials=True,

allow\_methods=["\*"],

allow\_headers=["\*"],

)

ignore\_dirs = ["mongodb/data", "static/images"]

if \_\_name\_\_ == "\_\_main\_\_":

uvicorn.run(app, host="127.0.0.1", port=8080, reload\_dirs=[ignore\_dirs])

该启动文件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

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)):

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

if not user:

raise HTTPException(status\_code=400, detail="Invalid username or password")

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

raise HTTPException(status\_code=400, detail="Invalid username or password")

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"}

/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。

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("/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

/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

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, 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.'}

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。

具体来说，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(...), current\_user: User = Depends(get\_current\_user)):

keywords = search\_data.get('keywords')

start\_time = search\_data.get('start\_time')

end\_time = search\_data.get('end\_time')

is\_sort\_by\_hot = search\_data.get('is\_sort\_by\_hot')

is\_search\_with\_specific\_time\_scope = search\_data.get('is\_search\_with\_specific\_time\_scope')

cookie = search\_data.get('cookie')

print(is\_sort\_by\_hot)

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

'keywords': keywords,

'start\_time': start\_time,

'end\_time': end\_time,

'is\_sort\_by\_hot': is\_sort\_by\_hot,

'is\_search\_with\_specific\_time\_scope': is\_search\_with\_specific\_time\_scope,

'cookie': cookie

})

user\_id = current\_user.id

task\_time = datetime.now()

new\_task = {

"task\_id": task.id,

"user\_id": user\_id,

"task\_type": "weibo\_search",

"task\_time": task\_time,

}

task\_db["tasks"].insert\_one(new\_task)

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的前端组件代码。

<template>

<a-layout class="main-layout">

<a-layout-sider class="sider">

<a-menu :defaultSelectedKeys="['search']" :selectedKeys="[selectedTask]" @select="handleMenuSelect">

<a-menu-item key="search">关键字查询</a-menu-item>

</a-menu>

</a-layout-sider>

<a-layout-content style="padding: 23px" class="content">

<a-card>

<a-form :model="searchData" v-else-if="selectedTask === 'search'" @submit.prevent="submitSearchSpider">

<a-form-item label="Keywords" :label-col="{ span: 4 }" :wrapper-col="{ span: 20 }">

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

</a-form-item>

<a-form-item label="Start Time" :label-col="{ span: 4 }" :wrapper-col="{ span: 20 }">

<a-date-picker

v-model:value="searchData.startTime"

show-time

format="YYYY-MM-DD-HH"

placeholder="选择开始时间"

/>

</a-form-item>

<a-form-item label="End Time" :label-col="{ span: 4 }" :wrapper-col="{ span: 20 }">

<a-date-picker

v-model:value="searchData.endTime"

show-time

format="YYYY-MM-DD-HH"

placeholder="选择结束时间"

/>

</a-form-item>

<a-form-item label="根据热度优先收集" :label-col="{ span: 4 }" :wrapper-col="{ span: 20 }">

<a-switch v-model:checked="searchData.is\_sort\_by\_hot" />

</a-form-item>

<a-form-item label="是否指定时间范围" :label-col="{ span: 4 }" :wrapper-col="{ span: 20 }">

<a-switch v-model:checked="searchData.is\_search\_with\_specific\_time\_scope" />

</a-form-item>

<a-form-item label="Cookie" :label-col="{ span: 4 }" :wrapper-col="{ span: 20 }">

<a-input v-model:value="searchData.cookie" placeholder="输入Cookie" />

</a-form-item>

<a-form-item :wrapper-col="{ offset: 4 }">

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

</a-form-item>

</a-form>

</a-card>

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

</a-layout-content>

</a-layout>

</template>

<script>

export default defineComponent({

name: 'RunWeiboTasks',

data() {

return {

searchData: {

keywords: '',

start\_time: '',

end\_time: '',

is\_sort\_by\_hot: true,

is\_search\_with\_specific\_time\_scope: false,

cookie: ''

},

selectedTask: 'search',

componentMap: {

search: SearchList,

},

showData: true,

showTaskList: false,

}

},

methods: {

handleMenuSelect({ key }) {

this.selectedTask = key;

this.Component = this.componentMap[key];

},

submitSearchSpider() {

const formData = {

keywords: this.searchData.keywords.split(','),

start\_time: this.searchData.start\_time,

end\_time: this.searchData.end\_time,

is\_sort\_by\_hot: this.searchData.is\_sort\_by\_hot,

is\_search\_with\_specific\_time\_scope: this.searchData.is\_search\_with\_specific\_time\_scope,

cookie: this.searchData.cookie

}

console.log(JSON.stringify(formData))

axiosInstance.post('/weibo/run\_weibo\_search\_spider', JSON.stringify(formData), {

headers: {

'Content-Type': 'application/json',

},

})

.then(response => {

message.success('搜索爬虫任务提交成功！')

})

.catch(error => {

message.error('提交搜索爬虫任务时出错！')

})

},

}

})

</script>

上面这段代码，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 系统数据展示模块的实现

## 5.4 系统数据分析可视化模块的实现

## 5.5 系统通知模块的实现

## 5.6 系统Docker部署的实现

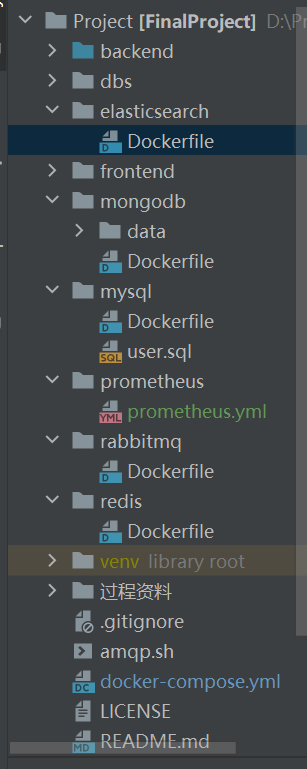


图5-1 系统目录结构图

### 5.6.1 backend目录下的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 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

它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。具体来说，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

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"]

它的作用是将前端应用代码打包为一个Docker镜像。具体来说，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

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"]

它的作用是将RabbitMQ消息队列打包为一个Docker镜像。具体来说，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

FROM redis:latest

EXPOSE 6379

CMD ["redis-server"]

它的作用是将Redis数据库打包为一个Docker镜像。具体来说，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

FROM mysql:8.0

MAINTAINER zm1575098153@gmail.com

ENV MYSQL\_ROOT\_PASSWORD Zm.1575098153

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

EXPOSE 3306

它的作用是将MySQL数据库打包为一个Docker镜像。具体来说，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

它的作用是将MongoDB数据库打包为一个Docker镜像。具体来说，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

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

- ELASTIC\_SEARCH=http://elasticsearch:9200

- REDIS\_HOST=redis

- FLOWER\_BASIC\_AUTH=admin:password

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

depends\_on:

- mysql

- 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"

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。具体来说，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

## 6.2 系统登录模块测试

## 6.3 系统数据收集模块测试

## 6.4 系统数据展示模块测试

## 6.5 系统数据分析可视化模块测试

## 6.6 系统通知模块测试

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。

# 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

经过这几个月的毕业设计的学习，我收获了很多。我充分体会到了一个系统的完成是需要不断的学习，不断的完善。经过这次对盐城市消防接警处理系统的设计，充分利用了软件工程的思想，使得我对大学四年所学到的知识有了实践性的提高。

通过这次毕业设计，我得出了以下几点体会：

a) 对于软件专业的学生来说，在实践中学习远比抱着书本死读书要收获的多的多。要在做的过程中，不断尝试，不断实践。虽然会遇到很多的难题，但是你每解决一个难题，就收获了一个知识点，遇到的困难越多，解决的越多，进步的就越快。

b) 对于一个软件的设计，一定要遵循软件工程的步骤。按正确的步骤，不仅能使软件的实现更容易，也是实现步骤更清晰。

c) 编写代码不仅要认真仔细，还要遵循代码编写原则，比如：

代码缩进。一般遵循的原则是，使用流程控制语句IF…ENDIF、SELECT CASE、 FOR…NEXT 、DO…LOOP时，流程控制语句之中的程序代码部分要用[Tab]键进行一次缩进，遇到嵌套时，亦遵循此规则。

代码换行。一行的程序代码如果太长，那么就需要换行以增加其可读性。换行时应尽量按照程序执行的逻辑进行换行，遇到标点时标点应放在前一行的末尾处，且所下移的行要尽量缩进。

代码模块化。对代码进行模块化处理，可以提高代码的复用性以及提高开发速度。代码的模块化处理，包括将需要进行复用的代码编写成一个子程序或者函数，或者一个类，以便使用时进行调用。将代码太长的函数分解成几个函数的调用，这样都会大大提高程序的可读性和可维护性。

总的来说，由于水平还不高，以及毕业设计时间较短，该系统只是基本达到毕业设计的要求，还有许多不尽如人意的地方，比如数据库的设计欠缺，用户界面不够美观，自动化程度不够等多方面问题。这些都有待进一步改善。

通过这次毕业设计，我发现自己的知识还很不够，还不能完全适应当前的需求。在今后的学习和工作中，我将努力弥补，不断地扩充各方面的知识，吸取失败的教训。通过不断的学习和改进，希望能够进一步提高自己的水平。

# 参考文献

[1] 张海藩.软件工程导论（第四版）[M].北京：清华大学出版社，2003.

[2] 周涛,吕伟臣,廉颖.SQL Server 2005 数据库企业级应用[M].北京：清华大学出版社，2008.4.

[3] 王敬栋,施伟.ASP.NET 2.0数据库通用模块开发与系统移植[M].北京：清华大学出版社，2007.5.

[4] 龙马工作室.Visual C# 2005 从入门到精通[M].北京：人民邮电出版社，2008.4.

[5] 李德奇.ASP.NET程序设计[M].北京：人民邮电出版社，2007.9.

[6] 启明工作室.ASP.NET+SQL Server网络应用系统开发与实例[M].北京：人民邮电出版社，2005.2.

[7] 蔡敏,徐慧慧,黄炳强.UML基础与Rose建模教程[M].北京：人民邮电出版社，2006.1.

[8] 郑霞,赵辉,徐慧.ASP.NET 2.0编程技术与实例[M].北京：人民邮电出版社，2007.7.

[9] 杨云,王毅.ASP.NET 2.0典型项目开发[M].北京：人民邮电出版社，2007.10.

[10]郑阿奇.SQL Server实用教程[M].北京：电子工业出版社，2005.7.

[11]ASP.NET页面事件：顺序与回传详解[DB/OL].www.csdn.net，2008.4.

[12]胡吉林,姜波,高丙朋.[存储过程在ASP．NET数据库系统开发中的应用](http://mag.shudoo.com/qk/96730X/200812/28859873.html)[J].电脑开发与应用,2008，（12）：66-68.

(>=10个参考文献，近3年的。书写按照我PPT中给的格式标准。)

# 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 系统登录模块

### 附录1.1.1 用户登录模块

using System;

using System.Data;

### 附录1.1.2 普通用户注册模块

using System;

using System.Data;

using System.Configuration;

## 附录1.2 接警人模块

### 附录1.2.1 接警记录模块

using System;

using System.Data;

### 附录1.2.2 出警安排模块

using System;

using System.Data;

using System.Configuration;

### 附录1.2.3 案件总结模块

using System;

using System.Data;

using System.Configuration;

### 附录1.2.4 案件查询模块

ntPlaceHolder1" Runat="Server">

<table style="width: 736px; height: 150px">

## 附录1.3 普通用户模块

### 附录1.3.1 普通用户案件查询模块

using System;

### 附录1.3.2 普通用户修改密码模块