

### 《[大数据综合应用实训](javascript:var%20win=window.open('TeachTaskDetail.aspx?YearTermID=-RPbsWWH4lAeWCsXsnsEcg==&CourID=OAhJUvL3A_EFiXWpOKaQYw==&CourOrder=kT046BuvypDA3lC383kt6g==%27,%27%27,%27toolbar=no,menubar=no,scrollbars=yes,resizable=yes,location=no,status=no,top=0,left=0,width=550,height=350%27))》报告

**课题名称： 微博舆情分析**

**专 业： 计算机科学与技术**

**班 级： 计算机222**

**学 号： 202202296**

**姓 名： 章崇文**

**校内教师： 田俐**

**成 绩：**

**2025年 5 月 26 日—— 2025年 6 月 6 日**

1. **实训项目概述**

## 1.1 课题开发的背景

在当今信息爆炸的数字化时代，微博作为重要的社交媒体平台，已成为公众表达观点、交流信息、形成舆论的主要场所之一。微博上的话题不仅反映了社会热点和公众情绪，也对企业品牌形象、市场动态乃至社会稳定产生着不可忽视的影响。

随着微博用户数量和信息发布量的持续增长，如何有效、及时地从海量、复杂、快速变化的微博数据中提取有价值的信息，洞察舆情动态，成为了一个重要的研究方向和实际需求。传统的人工监测和分析方法在效率、覆盖面和深度上已难以满足要求。因此，利用先进的计算机技术，特别是数据挖掘和自然语言处理技术，开发自动化的舆情分析系统，成为应对这一挑战的必然趋势。本项目正是在这样的背景下提出，旨在探索利用Python技术构建微博话题舆情分析系统的可能性与方法。

## 1.2 课题研究的意义

本研究具有理论与实践双重价值。理论上，有助于深化对网络舆情传播规律的理解，推动相关分析技术发展。实践上，可为企业、政府及研究机构提供高效的舆情监测与分析工具，辅助决策，提升社会各界对网络舆情的认知与应对能力，促进网络环境健康发展。

## 1.3 研究内容概述

本项目核心是设计与开发一个基于Python的微博话题舆情分析管理系统。主要研究内容包括：

* 有效的微博舆情数据获取机制。
* 基于Python的舆情数据处理与分析技术（含NLP应用）。
* 直观的舆情分析结果可视化方案。
* 系统的整体架构设计与原型实现。

目标是构建一个具备实际应用价值的微博舆情分析工具。

1. **使用方法**

**2.1 问题分析**

**2.1.1 问题来源**

问题的根源在于当今数字化时代信息传播方式的深刻变革，特别是以微博为代表的社交媒体平台的崛起与普及。

微博成为主流舆论场：微博已成为公众表达观点、讨论社会热点、形成社会舆论的重要平台。其上的信息量巨大，传播速度快，影响力广泛，直接关系到公众情绪、企业声誉乃至社会稳定。

微博信息的复杂性与挑战：

* **海量性与高速性**：微博信息每时每刻都在海量产生和快速传播，传统人工方式难以全面、及时地捕捉和跟踪。
* **碎片化与非结构化**：信息多以短文本、图片、视频等形式存在，内容碎片化、非结构化程度高，直接分析难度大。
* **情绪化与多样性**：用户表达常带有强烈的情感色彩，观点多元，甚至存在虚假信息、极端言论，使得舆情分析更为复杂。

传统分析方法的局限：依赖人工的舆情监测和分析方法，在面对微博的海量、高速、复杂信息时，效率低下、覆盖不全、主观性强，难以满足对舆情进行快速、客观、深入洞察的需求。

因此，如何有效地从纷繁复杂的微博信息中，快速、准确地把握舆情脉搏，理解公众情绪与观点，应对潜在风险，便成为了一个亟待解决的问题。

**2.1.2 现有解决方案**

针对微博平台产生的海量舆情信息及其分析需求，目前已存在多种解决方案和尝试途径，主要包括：

传统人工监测与分析：

方式：通过人工设定关键词在微博平台进行搜索、浏览、阅读、记录和归纳总结。

分析：这是最原始的方式。其优点在于对特定信息的理解可能更深入、灵活，能处理一些机器难以把握的语境和隐喻。然而，缺点非常明显：效率低下、成本高昂、覆盖面窄、时效性差、易受主观因素影响，难以应对微博信息的海量、高速特性，无法进行大规模、系统性的舆情分析。

商业化舆情监测平台/SaaS服务：

方式：市场上存在多家公司提供成熟的商业舆情监测分析系统，例如新浪舆情通、数说故事舆情雷达、拓尔思、Meltwater、秒针SocialX 以及清博、优讯等。这些平台通常整合了数据采集、处理、分析和可视化功能。

分析：

* 优点：功能全面（覆盖全网信息，含微博、新闻、论坛等；提供情感分析、传播路径、竞品对比、热点追踪、预警推送、报告生成等多种功能），技术相对成熟，通常提供7\*24小时监测和分钟级预警，用户无需自行开发维护系统。
* 缺点：通常价格不菲，以订阅服务模式收费，对于预算有限的组织或个人可能难以承担；系统的定制化程度有限，分析模型和算法通常是“黑箱”，用户难以了解具体细节或进行修改；数据归属权和隐私问题也需要考虑。

基于特定技术/算法的研究与开源工具：

方式：学术界和开发者社区贡献了大量用于舆情分析的技术和方法研究，以及一些开源工具/库。例如，利用爬虫技术获取数据，使用jieba进行中文分词，应用TF-IDF、TextRank提取关键词，通过机器学习或深度学习模型（如SVM、RNN-LSTM）进行情感分类，使用LDA等进行主题挖掘。

分析：

* 优点：成本低廉（多为开源或免费），技术透明度高，方便研究人员和开发者根据具体需求进行定制、改进和集成，可以深入理解技术细节。
* 缺点：通常需要使用者具备较强的技术能力进行开发、部署和维护；单个工具或算法往往只能解决流程中的某一个环节，需要自行整合才能构成完整系统；稳定性和功能全面性通常不如商业平台；缺乏持续的技术支持和更新保障。

**2.1.3 本作品要解决的痛点问题**

通过对现有微博舆情分析解决方案的分析比较，我们发现当前用户（特别是中小企业、研究机构、开发者和学生等群体）在进行微博舆情分析时，主要面临以下核心痛点问题，这些正是本作品致力于解决的：

**成本与可及性痛点：**

* 成熟的商业舆情系统功能强大但价格高昂，对于预算有限的用户来说门槛过高，无法负担。
* 传统人工分析方式虽然看似直接，但长期来看人力成本高、效率低下，难以规模化应用。

本作品旨在解决：提供一个开源免费的解决方案，消除软件购买成本，降低舆情分析的技术应用门槛。

**集成度与易用性痛点：**

* 虽然存在许多优秀的开源算法库或工具，但它们往往功能单一，用户需要自行将数据采集、清洗、分析、可视化等多个环节的技术栈整合起来，这需要较高的技术能力和大量开发时间，使用门槛高。

本作品旨在解决：提供一个一体化的工具，整合舆情分析的关键流程，简化部署和使用过程，让非专业开发者也能相对快速地搭建和使用舆情分析系统。

**灵活性与透明度痛点：**

* 商业平台通常是“黑箱”操作，用户无法了解其内部具体的分析模型和算法，难以根据特定需求进行定制修改。

本作品旨在解决：基于Python并采用开源模式，保证了技术的透明度和可定制性，用户可以根据自己的研究或业务需求，灵活调整、修改或扩展系统功能与分析算法。

**资源效率痛点 (相对)：**

* 部分大型商业系统可能较为庞大，对运行环境有一定要求。而自行拼凑的系统如果没有良好优化，也可能效率低下。

本作品旨在解决：在保证核心功能的前提下，力求通过优化的代码和架构设计，实现相对较低的资源消耗和较高的运行效率，使其能在普通服务器甚至个人电脑上运行，适应更多应用场景。

综上所述，本作品的核心目标是解决现有微博舆情分析方案在**成本、易用性、灵活性和资源效率**等方面存在的痛点，为特定用户群体提供一个更优的选择。

**2.1.4 解决问题的思路**

为构建一个有效的微博舆情分析一体化工具，系统必须实现以下核心功能模块和性能要求：

**2.1.4.1 功能需求 (Functional Requirements)**

* **数据采集模块**：负责依据用户配置（关键词、话题等）自动从微博获取帖子、评论等原始数据。
* **数据处理与存储模块**：对采集的数据进行清洗、去重、分词等预处理操作，并将其结构化地存储于数据库中，便于管理和调用。
* **核心舆情分析模块**：应用自然语言处理（NLP）等技术，实现关键的情感倾向判断、热点话题与关键词识别等分析任务。
* **结果可视化模块**：将复杂的分析结果转化为易于理解的图表（如趋势图、分布图、词云），直观呈现舆情态势。
* **用户交互与管理模块**：提供用户友好的操作界面，支持任务管理、数据浏览、结果查看及必要的系统后台管理功能。

**2.1.4.2 性能需求 (Performance Requirements)**

考虑到本作品旨在提供一个资源消耗相对较少且高效的开源工具，性能方面需关注：

* **稳定性**：保障系统能够长时间无故障运行，具备一定的容错能力。
* **响应速度**：确保用户执行数据查询、分析、图表加载等操作时，系统能在合理时间内反馈结果，提升用户体验。
* **资源效率**：优化算法和架构，控制CPU和内存使用，使其能在常规硬件配置下流畅运行。
* **可扩展性**：采用模块化设计，方便未来根据需求添加新功能或改进现有模块。

**2.2 技术方案**

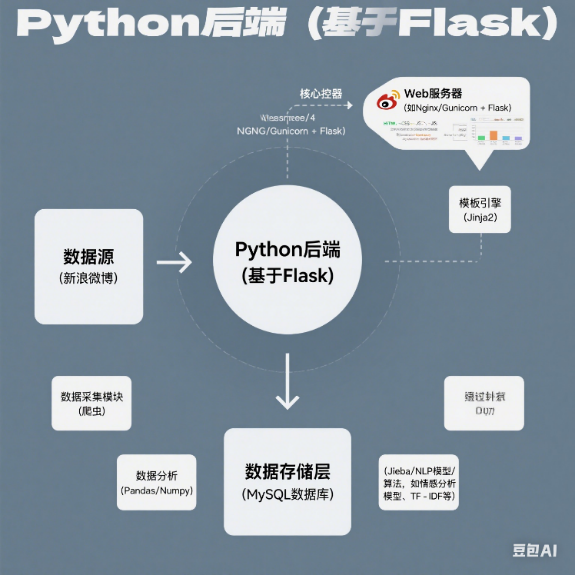
本章将从原理层面详细介绍“基于Python的微博话题舆情分析管理系统”所采用的核心技术方案。我们将首先概述系统的整体技术架构和选型思路，然后分模块详细阐述关键技术的实现原理和应用。

**2.2.1 总体技术方案与架构**

为实现一个开源、一体化、资源消耗相对较低且高效的微博舆情分析系统，我们选择以**Python**语言为核心开发平台。Python拥有强大的生态系统，特别是在数据科学、Web开发和自然语言处理领域有丰富的库和框架支持，非常适合本项目的需求。

系统整体采用**B/S (Browser/Server) 架构**，用户通过Web浏览器访问系统界面，后端服务器负责处理业务逻辑、数据分析和存储。技术路线主要围绕以下核心组件构建：

* **数据采集层**：负责从新浪微博实时获取原始数据。
* **数据存储层**：负责持久化存储采集到的数据以及分析结果，选用**MySQL**关系型数据库。
* **数据处理与分析层**：系统的核心，利用Python数据处理库和自然语言处理(NLP)技术进行数据清洗、特征提取、情感分析、热点计算等。
* **Web服务层 (后端)**：基于**Flask**轻量级Web框架，负责处理前端请求、调用分析逻辑、与数据库交互、组织响应数据。
* **表现层 (前端)**：使用标准的**HTML/CSS/JavaScript**技术构建用户交互界面，并通过**Matplotlib/Seaborn**等Python可视化库在后端生成图表，嵌入前端页面展示。
* **开发与运行环境**：推荐使用**Anaconda**进行Python环境和库管理，确保开发和部署环境的一致性。



**2.2.2 关键技术与模块详述**

接下来，我们将分模块详细介绍系统实现所依赖的关键技术及其在各模块中的具体应用和原理。

**2.2.2.1 数据采集技术 (Data Acquisition Technology)**

系统源头数据来自新浪微博平台。为实现微博帖子与评论的实时或准实时采集，本项目采用网络爬虫（Web Crawler）技术进行定向抓取。

实现原理与技术选型

核心库与框架：

* **请求与解析**：使用Python的 requests 库模拟HTTP请求，访问微博Web页面或移动端接口，获取HTML或JSON数据后，结合 BeautifulSoup 或 lxml 解析页面，提取目标字段。
* **动态内容处理**：针对微博中通过JavaScript加载的内容（如评论分页加载），使用 Selenium 搭配浏览器驱动（如 ChromeDriver）模拟真实浏览器行为，执行JavaScript以获取完整内容。同时，深入分析页面网络请求，直接构造AJAX请求，提升抓取效率与稳定性。
* **爬虫框架**：使用 Scrapy 构建结构化、可扩展的爬虫系统，实现请求调度、数据管道处理、中间件管理，支持并发抓取与分布式部署。

数据提取目标：

爬虫程序精准提取与数据库表结构（article表与comments表）对应的数据字段，包括帖子内容、发布时间、作者信息、转发/评论/点赞数，以及评论内容与评论时间等。

挑战与应对策略

反爬虫机制：微博平台常部署验证码验证、IP封禁、User-Agent检测、请求频率限制、动态cookie/token验证等多重反爬策略。

对策方案：

* 设置真实浏览器特征的请求头，模拟User-Agent。
* 集成IP代理池，动态轮换IP地址。
* 控制请求频率，加入随机延迟，降低访问行为的可识别性。
* 实现登录态管理，维护有效的session与cookies。
* 对动态加载数据，结合Selenium和网络请求分析，稳定获取完整内容。

合规性考量

爬虫系统严格遵守新浪微博 robots.txt 协议及平台服务条款，避免恶意抓取与过度访问，确保数据采集过程合法、合规，尊重网站数据所有权。

通过上述技术体系，爬虫模块能够稳定、高效地采集微博平台上的关键数据，为后续的数据分析与建模提供坚实基础。

**2.2.2.2 数据存储技术 (Data Storage Technology)**

数据存储是本系统承载数据采集结果和支撑后续分析的基础。考虑到微博文章、评论及系统用户等数据的结构化特性，本项目选用**MySQL**作为主要的数据库管理系统（DBMS）。具体使用的是**MySQL 5.7**版本。

**技术选型理由与实现：**

* **关系型数据库的适用性**：MySQL作为一款成熟、稳定、广泛使用的开源关系型数据库，非常适合存储具有明确结构的数据。微博文章（article表）、评论（comments表）和系统用户（user表）都具有清晰的字段和关联关系，使用MySQL可以方便地定义表结构、保证数据的一致性与完整性。
* **明确的表结构设计**：根据项目需求，我们设计了核心的数据表（具体Schema由用户先前提供）：
  + article表：用于存储微博帖子信息，字段涵盖了ID、内容、时间、区域、作者、转评赞数量等，并为 created\_at 添加了索引以优化按时间查询。content 字段采用 MEDIUMTEXT 以容纳较长的微博正文。
  + comments表：用于存储评论信息，包含评论ID、关联的文章ID、内容、时间、区域、作者、点赞数等，并为 articleId 添加了索引以快速检索某篇文章下的所有评论。content 字段同样采用 MEDIUMTEXT。
  + user表：用于存储本系统的用户信息（非微博用户），包含用户名、密码（应存储哈希值）等。
* **字符集**：所有表均采用 utf8mb4 字符集和 utf8mb4\_unicode\_ci 排序规则，以确保能正确存储和处理包括Emoji表情在内的各类微博文本字符。
* **与Python的集成**：Python拥有成熟的MySQL连接库（如 mysql-connector-python, PyMySQL），可以方便地在后端代码中执行SQL查询和数据操作。若结合Flask框架，还可以选用 SQLAlchemy 等Object-Relational Mapper (ORM)工具，以面向对象的方式操作数据库，简化开发流程，提高代码的可读性和可维护性。
* **性能与优化**：MySQL提供了良好的查询性能，通过在关键字段（如 article.id, comments.comment\_id, article.created\_at, comments.articleId）上建立主键和索引，可以显著提升数据检索效率，满足系统对响应速度的要求。

通过使用MySQL数据库并配合精心设计的表结构和索引，数据存储层能够可靠、高效地管理系统所需的各类数据。

**2.2.2.3 数据处理与分析技术**

一、数据预处理与清洗

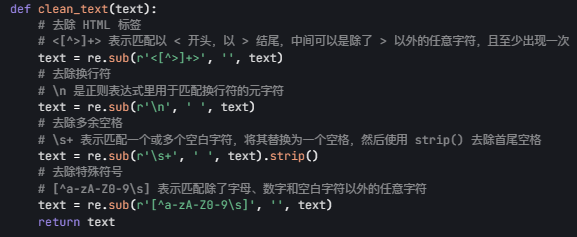
原始微博内容（包括帖子 content 和评论 content）常包含冗余信息、格式不规范或语义噪声，必须进行预处理以提高后续分析的精度与鲁棒性。

1. 数据加载与结构化处理

利用 Pandas 库将数据库中的查询结果加载为 DataFrame，以支持高效的批量操作和向量化处理；同时结合 NumPy 提供的数值计算能力提升整体运算效率。

1. **文本清洗与标准化**

* 去除噪声字符：包括HTML标签、换行符、多余空格、特殊符号等，统一采用 re 模块实现正则化清洗；



* **去重处理**：对内容完全一致的帖子或评论进行识别与移除，避免冗余影响热点统计和情感聚类；
* **缺失字段补全或剔除**：如区域 region 字段为空时，结合上下文或用户属性尝试补全；不可补全时将该样本剔除以确保数据完整性；
* **统一文本编码与格式**：将文本统一转为UTF-8编码、简体中文，消除编码歧义。

二、基于自然语言处理的分析模块

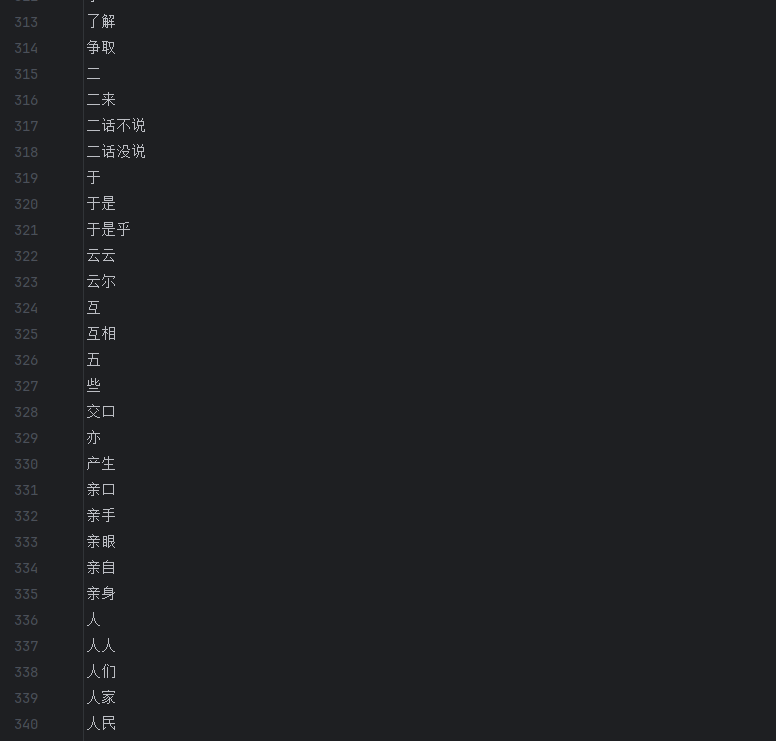
在完成清洗后，对微博文本执行一系列NLP分析任务，提取核心语义特征与舆情指标，主要包括分词、关键词识别、情感分析等。

1. 中文分词

中文没有天然词界，因此分词是NLP分析的基础。项目采用 Jieba 分词库，默认使用精确模式，并通过加载领域自定义词典提升网络语义识别能力，尤其针对网络热词、俚语和变体拼写（如“打工人”、“YYDS”等）做定制优化。

1. 停用词过滤

分词后去除常用虚词（如“的”、“了”、“是”）及标点符号，使用成熟的中文停用词表（哈工大停用词表）提升主题识别的聚焦性与有效性。



1. 关键词提取与热点识别

采用组合算法识别文本中的高权重词汇与核心话题：

* TF-IDF：衡量词语对单条微博内容的重要性，适合提取代表性关键词；

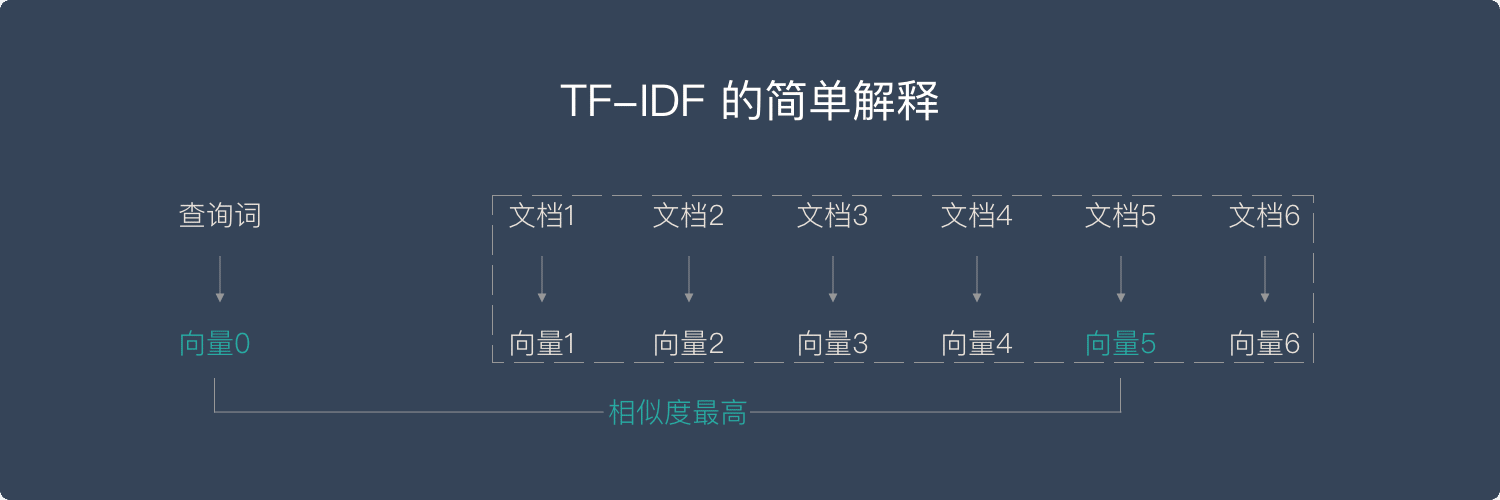
其基本思想是：如果某个词在一篇文章中出现的频率高，而在其他文章中出现的频率低，那么这个词就可能更好地代表这篇文章的内容。

数学公式表示为：

TF-IDF(t,d,D)=TF(t,d)×IDF(t,D)

其中：

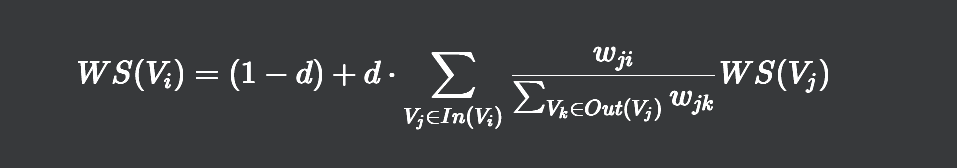
* + TF(t,d) 是词语 t 在文档 d 中出现的频率（Term Frequency）。
  + IDF(t,D) 是词语 t 的逆文档频率（Inverse Document Frequency）。
  + IDF(t,D)=log(DF(t)N​)，其中 N 是语料库中的文档总数，$ \text{DF}(t) $ 是包含词语 t 的文档数量。



* TextRank：构建词图模型，基于共现关系评估词语重要性，适用于抽取关键短语和摘要信息；

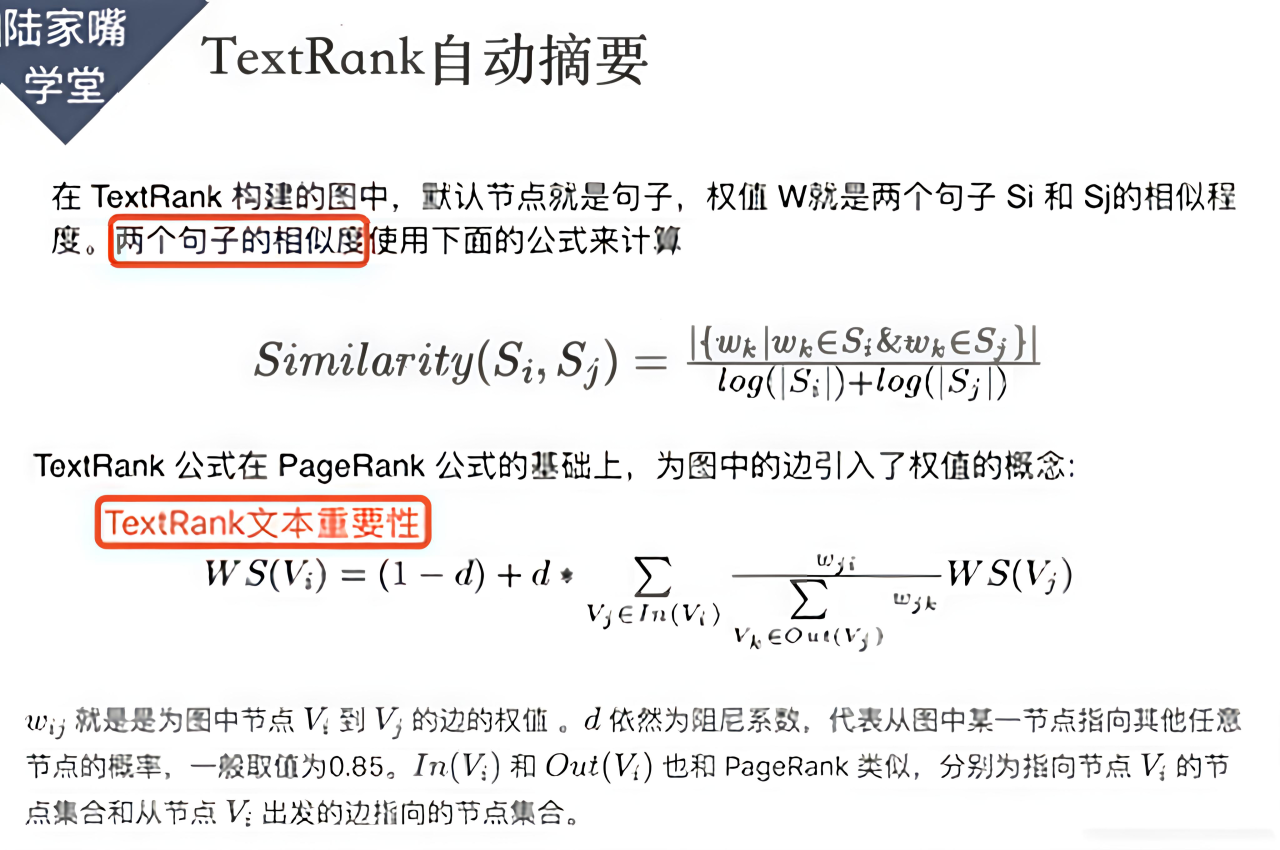
TextRank算法的思想来源于PageRank算法，其核心是将文本中的词语构建成一个图结构，通过计算词语在图中的重要性来提取关键词。词语之间的共现关系（在一定窗口内共同出现）被视为图中的边，边的权重代表共现的紧密程度。

TextRank的计算公式为：



其中：

* + WS(Vi​) 是词语 Vi​ 的权重。
  + d 是阻尼系数，通常取值为0.85。
  + In(Vi​) 是指向词语 Vi​ 的词语集合。
  + Out(Vj​) 是从词语 Vj​ 指向的词语集合。
  + wji​ 是词语 Vj​ 和 Vi​ 之间的边的权重。

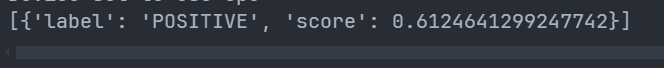


* **高频词统计与聚类分析**：结合滑动时间窗口，对高频关键词进行时序聚类，动态识别热点话题的演化趋势。

1. 情感倾向分析

用于评估微博或评论文本中表达的情绪方向，是舆情分析中识别“正面/中性/负面”态度的关键环节。系统支持两种策略：

* **词典规则法（首选）**：使用现有情感词典（如 DUTSC 情感词汇本体、HowNet），结合否定词、程度副词等语法规则构建评分体系，快速输出情感倾向值。该方法实现简洁、可解释性强，适用于资源受限场景；
* 模型驱动法（可拓展）：若项目后期对精度要求提高，可训练情感分类模型，如朴素贝叶斯、SVM、或基于 BERT 的深度模型。模型基于标注语料进行训练，具备更强的上下文理解能力，但依赖计算资源与数据量。



三、舆情特征提取与结构化输出

基于上述分析结果，系统将原始微博文本转化为可量化、结构化的数据特征：

* 高频关键词及其时间分布，构成热点话题演化图谱；
* 文章与评论的情感标签分布，支持情绪趋势分析；
* 各类舆情事件的摘要与核心观点提取，便于后续可视化与推理决策。

通过严密的数据处理流程和高效的NLP算法体系，本模块为系统提供深度语义支撑，使得舆情分析不再停留在表层统计，而能够从内容理解、情绪感知、趋势捕捉等方面提供全方位的信息洞察。

**2.2.2.4 后端框架技术**

系统的后端服务负责处理来自前端的请求、调度业务逻辑（数据处理与分析）、与数据库交互，并将结果返回给前端展示。本项目选用**Flask**作为核心的Python Web框架。

**技术选型理由与应用：**

* **轻量与灵活**：Flask是一个微框架 (Microframework)，核心简单，没有强制的项目结构和依赖，提供了基础的Web服务功能（如路由、请求处理、响应、Session管理等）。这使得它非常**灵活**，开发者可以根据项目需求自由选择和集成所需的组件（如数据库ORM、表单验证库等），非常适合本项目对**资源效率**和**定制性**的要求。
* **路由管理 (Routing)**：Flask使用基于装饰器（@app.route('/path')）的方式定义URL路由规则，将特定的URL路径映射到对应的Python处理函数（视图函数View Function）。这种方式直观简洁，便于管理不同功能页面的访问逻辑。例如，用户访问 /analysis 路径可能触发执行舆情分析并展示结果的函数。
* **请求处理与响应**：Flask提供了方便的对象来处理HTTP请求（如获取GET参数、POST表单数据）和生成HTTP响应（如返回HTML页面、JSON数据或重定向）。
* **业务逻辑调度**：后端视图函数作为调度中心，会根据用户请求调用 2.2.2.3 中介绍的数据处理与分析模块中的函数，执行具体任务（如情感分析、关键词提取等）。
* **模板引擎集成**：Flask默认集成 **Jinja2** 模板引擎。后端可以将从数据库查询或分析模块获得的数据传递给 Jinja2 模板，动态生成最终的HTML页面，实现数据与视图的分离。这对于展示分析结果（包括嵌入可视化图表）至关重要。
* **扩展性**：Flask拥有丰富的**扩展生态**。例如，可以通过 Flask-SQLAlchemy 扩展方便地集成 SQLAlchemy ORM 来操作MySQL数据库；如果需要用户登录功能，可以使用 Flask-Login 扩展等。这使得在保持核心简洁的同时，能够方便地增强系统功能。

通过使用Flask框架，可以快速构建稳定、高效且易于维护的Web后端服务，作为整个舆情分析系统的“大脑”，连接前端用户界面与后端数据处理分析能力。

**2.2.2.5 前端与可视化技术 (Frontend and Visualization Technology)**

为了让用户能够方便地与系统交互并直观地理解复杂的舆情分析结果，前端界面设计和数据可视化是技术方案的重要组成部分。

1. 前端用户界面 (Frontend User Interface)：

系统的用户界面通过标准的Web技术构建，运行在用户的浏览器中。

* + **基础技术**：使用 **HTML (HyperText Markup Language)** 定义页面结构， **CSS (Cascading Style Sheets)** 负责页面样式和布局（可能借助 Bootstrap、Tailwind CSS 等框架来提高开发效率和界面美观度）， **JavaScript** 用于实现页面的动态效果和用户交互逻辑（如表单验证、按钮响应等）。
  + **与后端交互**：前端页面通过HTTP请求与基于Flask的后端服务进行通信。用户在界面上的操作（如点击按钮提交分析请求、选择时间范围等）会触发向后端发送请求，后端处理后返回渲染好的HTML页面或JSON数据。
  + **模板引擎**：后端Flask使用 **Jinja2** 模板引擎将动态数据（包括分析结果和可视化图表信息）填充到HTML模板中，生成最终展示给用户的页面。最终呈现的界面效果可参考文档中的图 5.1 (管理界面)。

1. 数据可视化 (Data Visualization)：

将数据分析结果以图形化方式呈现是本系统的核心功能之一，主要在后端利用Python的可视化库生成图表，然后嵌入到前端页面。

* + **核心库**：主要使用 **Matplotlib** 和 **Seaborn**。Matplotlib是Python基础的绘图库，功能强大；Seaborn基于Matplotlib，提供了更高级的接口，可以轻松绘制出更美观、更具统计意义的图形。此外，对于特定图表如**词云 (Word Cloud)**，可能会使用 wordcloud 库。
  + **图表类型**：根据分析需求生成多种可视化图表，例如：
    - **趋势图**：使用折线图展示话题热度随时间的变化。
    - **分布图**：使用饼图或柱状图展示情感（正/负/中性）的分布比例。
    - **对比图**：使用柱状图对比不同话题、不同区域或不同时间段的舆情指标。
    - 关键词云图：直观展示文本中的核心高频词汇及其相对重要性。

（还包括其他图表，如散点图、热力图等）



* + **实现方式**：后端Python代码调用Matplotlib/Seaborn等库处理分析数据并生成图表对象。这些图表可以被保存为图片文件（如 PNG），或者直接在内存中生成Base64编码的图片数据。这些图片或图片数据随后被传递给 Jinja2 模板，通过 <img> 标签嵌入到最终的HTML页面中展示给用户。

通过结合标准的前端技术和强大的Python后端可视化能力，系统能够为用户提供一个既美观易用、又能清晰展示复杂舆情分析结果的交互界面。

1. **数据处理与总结**

**3.1 数据集说明 (Dataset)**

数据获取方式：采用实时网络爬虫技术，针对特定主题或关键词进行定向定量爬取，获取最新的相关公开微博数据，并存入MySQL数据库。

数据内容与结构：采集并存储的数据主要包含以下核心信息，具体结构由数据库表定义：

* 文章表 (article): 存储微博帖子的核心信息，如帖子id (主键), 点赞数 (likeNum), 评论数 (commentsLen), 转发数 (reposts\_count), 发布区域 (region), 帖子内容 (content), 内容长度 (contentLen), 发布日期 (created\_at), 类型 (type), 详情页链接 (detailUrl), 作者信息 (authorAvatar, authorName, authorDetail), 是否VIP (isVip) 等。
* **评论表 (comments)**: 存储与帖子相关联的评论信息，如评论comment\_id (主键), 所属文章ID (articleId), 评论发布日期 (created\_at), 点赞数 (like\_counts), 发布区域 (region), 评论内容 (content), 评论者信息 (authorName, authorGender, authorAddress, authorAvatar) 等。
* 用户表 (user): 用于系统自身用户管理，包含系统用户的id, 用户名 (username), 密码 (password, 存储哈希值), 创建时间 (createTime)。

数据具有典型的社交媒体特性，表现为强实时性和快速增长，主要以简短的文本为主，且语言风格口语化，情感色彩丰富。此外，微博数据中包含大量表情符号、图片和链接等多媒体元素，进一步丰富了表达形式。然而，这些数据也存在噪声和冗余信息，影响数据分析的准确性。整体来看，社交媒体数据在提供即时信息的同时，也面临着复杂性和数据处理的挑战。通过对数据量的分析可以发现，娱乐类话题占据了大部分内容，尤其是明星、电影、综艺等话题，吸引了大量讨论和参与。同时，评论量与话题总类之间有较强的匹配关系，热门话题通常伴随着大量的评论和转发，形成高活跃度的社交互动。这一特点反映了社交媒体上话题传播的高度集中和话题热度与用户参与度之间的紧密联系。

数据规模：在本次开发与测试阶段，获取并存入数据库的数据集规模大致为：微博文章（article表记录）约5000条，相关的评论（comments表记录）约25000条。

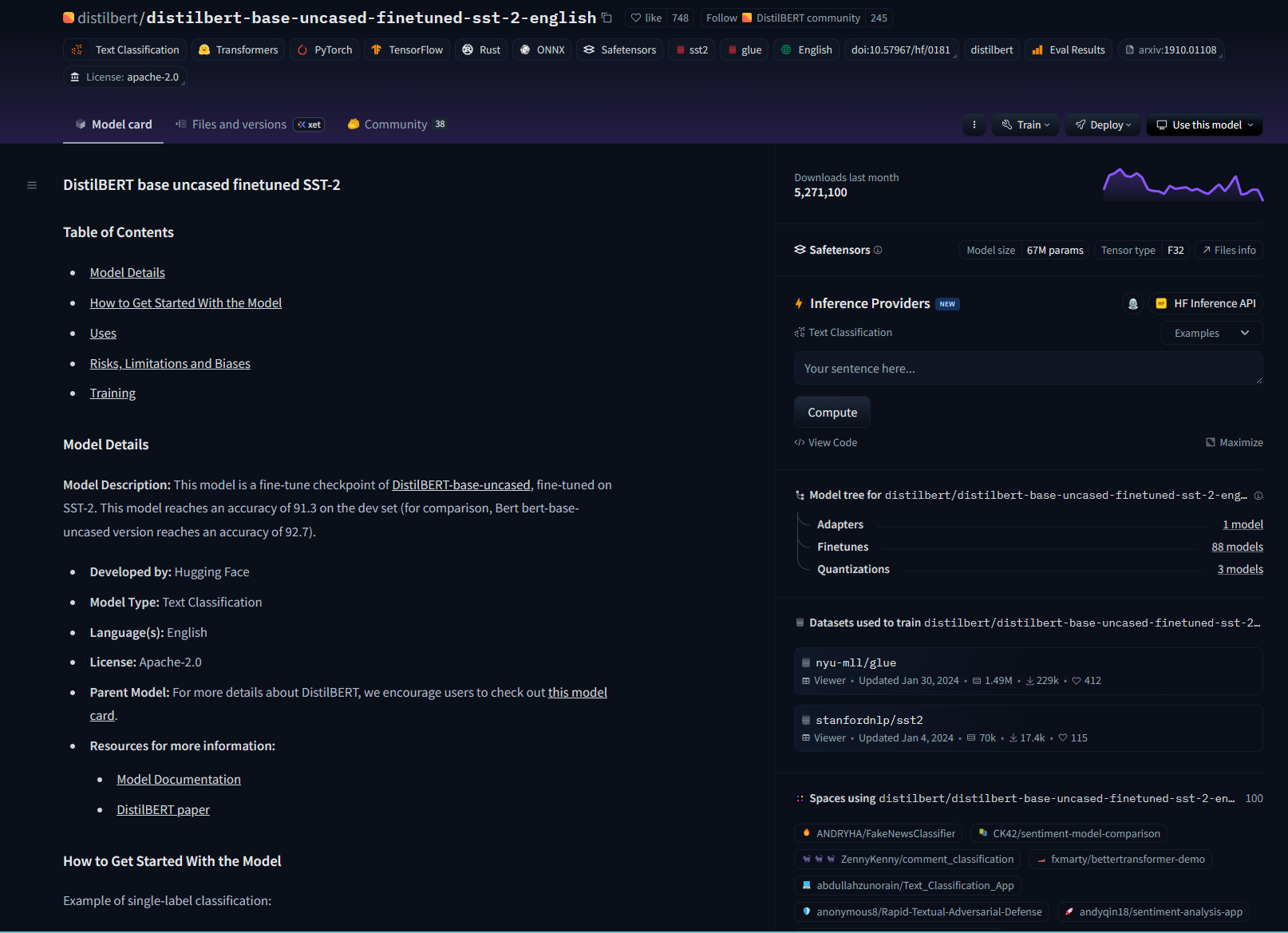
**3.2 系统实现**

本章详细阐述“基于Python的微博话题舆情分析管理系统”的具体开发和实现过程，将第二章的技术方案转化为实际可运行的软件系统。

**3.2.1 开发环境与工具**

为了确保开发过程的顺利进行和环境的一致性，本项目搭建了统一的开发环境并选用了一系列成熟的开发工具：

* **操作系统 (Operating System)**: Windows (例如 Windows 10/11) / macOS / Linux (根据实际开发环境填写)。
* **编程语言 (Programming Language)**: Python (具体版本号，例如 Python 3.8+)。
* **Python 环境管理 (Environment Management)**: Anaconda (例如 Anaconda3)，用于创建独立的虚拟环境，方便管理项目所需的 Python 版本和第三方库依赖，避免库版本冲突。
* **Web 框架 (Web Framework)**: Flask (具体版本号，例如 Flask 2.x)。
* **数据库 (Database)**: MySQL Community Server 5.7 (根据你提供的 Schema 信息)。
* **数据库管理工具 (DB Management Tool)**: HeidiSQL (根据你提供的 Schema 信息) 或其他工具如 Navicat, DBeaver, MySQL Workbench。
* **核心 Python 库 (Core Python Libraries)**:
  + 数据处理: Pandas, Numpy
  + Web 请求: Requests
  + HTML 解析: BeautifulSoup4, lxml
  + NLP框架：Hugging Face Transformers



NLP: Jieba SnowNLP DistilBERT

* + 可视化: Matplotlib, Seaborn, WordCloud
  + 数据库连接: PyMySQL
* **集成开发环境 (IDE)**: PyCharm / Visual Studio Code
* **版本控制 (Version Control)**: Git (配合 GitHub/Gitee/GitLab 等代码托管平台)。

选择这些广泛使用、社区支持良好、文档齐全的工具和库，有助于提高开发效率，降低开发难度，并保证系统的稳定性和可维护性。

**3.2.2 系统架构实现**

基于 2.2.1 节提出的总体技术方案，本系统在工程实现上严格遵循了 B/S (Browser/Server) 架构。用户通过标准 Web 浏览器访问系统，所有核心业务逻辑、数据处理与存储均在服务器端完成。具体实现涉及清晰的项目结构和模块化的代码组织。

1. 项目结构: 为了实现代码的解耦和可维护性，项目采用了典型的 Flask 应用目录结构
2. **模块交互实现**: 系统各层级和模块通过代码调用和数据传递实现交互：
   * **前端与后端通信**: 用户在浏览器（前端 templates + static 文件渲染）的操作生成 HTTP 请求，发送到由 Flask（app/views/main\_views.py）定义的对应路由。
   * **视图层调度**: Flask 的视图函数接收请求，解析参数，然后调用 app/services 中的业务逻辑函数（如 analysis.py 中的情感分析函数）。
   * **业务逻辑与数据存储**: 业务逻辑函数需要数据时，会调用 app/models 或 app/models/database.py 中封装好的函数，与 MySQL 数据库进行交互（执行 SQL 查询或 ORM 操作）。
   * **结果返回**: 业务逻辑函数将处理或分析结果返回给视图函数，视图函数再将这些数据传递给 Jinja2 模板引擎（app/templates），渲染成最终的 HTML 页面，响应给用户的浏览器。对于可视化图表，通常是在 analysis.py 中生成图表数据或图像，传递给视图函数，再嵌入模板。
3. 请求处理流程示例 (用户查看分析结果):

a. 用户在浏览器点击登入按钮

b. 浏览器向服务器 /report/xx URL 发送 GET 请求。

c. Flask 路由匹配到 main\_views.py 中处理 /report/xx 的视图函数。

d. 视图函数调用 analysis.py 中的 generate\_report\_xx() 函数。

e. generate\_report\_xx() 函数调用 database.py 从 MySQL 获取所需数据。

f. 获取数据后，进行情感分析、关键词提取等计算，并调用可视化库生成图表数据。

g. generate\_report\_xx() 将分析结果和图表数据返回给视图函数。

h. 视图函数调用 render\_template('report\_xx.html', data=results)，使用 Jinja2 渲染模板。

i. 服务器将生成的 HTML 页面作为 HTTP 响应返回给浏览器。

j. 浏览器显示分析报告页面。

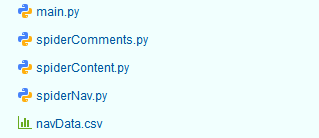
通过这种模块化的实现方式，将系统的不同职责（展示、控制、业务逻辑、数据访问）清晰地分离开来，提高了代码的可读性、可维护性和可扩展性。

**3.2.3 相关模块具体实现**

**3.2.3.1 数据采集模块实现**

数据采集模块是系统的“触手”，负责按照预定策略从新浪微博抓取原始数据。其实现主要基于 Python 的网络请求和 HTML 解析库。

1. 核心实现逻辑 (以 requests + BeautifulSoup 为例): 该模块通常实现为一个或多个 Python 脚本/类，其核心工作流程如下：



* + **任务初始化**: 接收需要爬取的目标信息，如关键词、特定话题的 URL、用户 ID 列表等。
  + **URL 构造与请求**: 根据目标信息构造需要访问的微博页面 URL（可能是搜索结果页、用户主页、单条微博页等）。使用 requests.get() 或 requests.post() 方法发送 HTTP 请求。为应对反爬虫，请求时会附带必要的请求头 (Headers)，特别是 User-Agent 来模拟浏览器，可能还需要携带 Cookies 以维持登录状态或绕过某些验证。
  + **页面解析与数据提取**: 使用 BeautifulSoup(response.text, 'lxml') 解析返回的 HTML 文本。通过分析微博页面的 HTML 结构，利用 BeautifulSoup 提供的 CSS 选择器 (soup.select(...)) 或 find/find\_all 方法，精确定位并提取所需的数据字段，如帖子 ID、内容、作者、时间、转评赞数、评论列表等。需要仔细处理各种可能的数据格式和异常情况（如某些字段缺失）。
  + **动态内容处理 (若采用 Selenium)**: 如果页面内容是 JS 动态加载的，则会使用 selenium 库启动一个 WebDriver (如 ChromeDriver)，通过 driver.get(url) 访问页面，然后使用 WebDriverWait 等待特定元素加载完成，再通过 driver.page\_source 获取渲染后的 HTML 交给 BeautifulSoup 解析，或者直接用 driver.find\_element(By...) 定位元素并提取文本。完成后调用 driver.quit() 关闭浏览器。
  + **数据结构化**: 将提取到的零散数据整理成 Python 字典或自定义对象，使其结构与 MySQL 数据库中的 article 和 comments 表的字段一一对应。进行必要的数据类型转换（如字符串转数字、时间字符串解析为 datetime 对象）。
  + **数据入库**: 调用数据存储模块提供的接口函数 (例如 app/models/database.py 中的 save\_article(data) 或 save\_comment(data) 函数)，将结构化后的数据写入 MySQL 数据库。实现中需要处理好数据库连接、事务以及可能的插入错误（如主键冲突、数据格式错误等）。

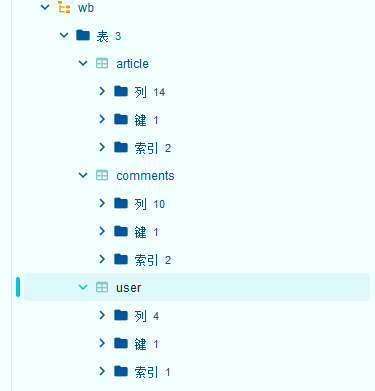
1. **反爬虫策略实现**: 在代码中融入了基础的反爬虫措施：
   * **请求头伪装**: 固定或轮换 User-Agent。
   * **随机延迟**: 在连续请求之间加入随机 time.sleep()，降低请求频率。
   * **IP 代理 (可选)**: 对于大规模或高频率爬取，可能需要集成 IP 代理池，在每次请求时随机选用一个代理 IP 发送请求。
   * **登录状态维持 (若需要)**: 通过 requests.Session() 对象管理 Cookies，或在 Selenium 中先执行登录操作。
2. **错误处理与日志记录**: 整个采集过程中加入了 try...except 块来捕获网络请求错误、页面解析错误、数据提取异常、数据库写入失败等问题，并通过 logging 模块记录详细的错误信息和爬取状态，便于后续调试和问题排查。

通过上述实现，数据采集模块能够相对稳定地执行数据获取任务，并将符合结构要求的数据传递给数据存储模块。

**3.2.3.2 数据存储模块实现**

该模块负责系统与 MySQL 数据库之间的所有交互，包括连接管理和数据的增、删、改、查 (CRUD) 操作。其实现封装在独立的 Python 文件中（在 app/models/database.py 或模型文件中）并通过 ORM (Object-Relational Mapper) 模型类来完成。

1. **数据库连接实现**:
   * **配置管理**: 数据库的连接信息（主机地址 127.0.0.1, 用户名, 密码, 数据库名 wb, 端口号等）存储在项目的配置文件 config.py 中。
   * **连接建立 (采用 Flask-SQLAlchemy)**: 本项目采用 **Flask-SQLAlchemy** 扩展。在 Flask 应用初始化时，该扩展根据 config.py 中的配置自动管理数据库连接和会话 (Session)。开发者定义了映射到数据库表 (article, comments, user) 的 Python 类（Model），通过这些类来操作数据。
2. **数据操作实现 (CRUD)**: 封装了对 article, comments, user 等表的操作函数。
   * **创建 (Create)**: 实现了 add\_article(article\_object), add\_comment(comment\_object) 等函数。这些函数接收从数据采集模块传来的 ORM 对象，使用 db.session.add() 添加新对象，然后调用 db.session.commit() 提交事务。实现中通过 try...except 捕获数据库写入错误等异常。
   * **读取 (Read)**: 实现了各种查询函数，如 find\_article\_by\_id(article\_id), find\_comments\_by\_article\_id(article\_id, page, per\_page), search\_articles(keywords, start\_date, end\_date) 等。这些函数根据传入的参数构造 ORM 查询表达式，使用 ORM 的查询 API（如 Article.query.filter\_by(...).all()）执行查询。查询时充分利用数据库 schema 中定义的索引（如 article.created\_at, comments.articleId）来优化性能。
   * **更新 (Update)**: 用于系统用户管理等场景，实现了 update\_user\_password(user\_id, new\_password\_hash) 等功能。通过 ORM 修改对象属性后调用 commit。
   * **删除 (Delete)**: 实现了 delete\_user(user\_id) 等函数，使用 ORM 的 db.session.delete() 方法执行删除。
3. 连接与会话管理: 使用 Flask-SQLAlchemy 时，框架会自动管理会话的生命周期，在请求结束后自动关闭或回滚会话，开发者只需关注 db.session.add(), db.session.commit(), db.session.rollback() 等操作。



通过封装这些数据库操作，数据存储模块为上层业务逻辑提供了一个清晰、统一的数据访问接口，屏蔽了底层的 SQL 细节，并利用了 MySQL 的能力来保证数据的持久化和查询效率。

**3.2.3.3 数据处理与分析模块实现**

此模块是系统进行智能分析的核心，其实现主要体现为一系列封装在特定 Python 文件（如 app/services/analysis.py）中的函数或类。这些函数负责接收原始数据（通常来自数据存储模块），执行预处理和 NLP 分析，并返回结构化的分析结果。

1. **数据预处理实现思路**:
   * **数据载入与准备**: 实现的函数首先调用数据存储模块接口，将数据库中的 article 或 comments 数据加载到 Pandas DataFrame 结构中，便于后续的向量化操作。
   * **文本清洗流程**: 针对 DataFrame 中的文本 content 列，按顺序应用清洗逻辑：
     + 定义并应用函数，使用正则表达式去除文本中的 URLs、HTML 标签、微博特定的 "@" 提及、"#话题#" 标记（保留话题内容本身）以及其他非文本噪声。
     + 实现标准化处理，如统一转换为小写（若需要）、去除多余空格等。
     + 调用函数检测并移除重复的记录。
2. **NLP 分析实现思路**: 基于预处理后的文本数据，按需调用以下分析功能的实现函数：
   * **中文分词实现**: 实现一个分词函数，该函数接收文本输入，内部调用 jieba.cut() 方法进行分词。函数返回分词后的词语列表。此函数被应用于 DataFrame 的文本列上。
   * **停用词过滤实现**: 实现一个过滤函数，该函数接收分词后的词语列表。函数内部加载一个预定义的停用词文本文件到集合 (Set) 数据结构中，然后遍历输入的词语列表，去除停用词集合中存在的词语，返回过滤后的词语列表。
   * **关键词提取/热点识别实现**:
     + **TF-IDF**: 实现一个计算 TF-IDF 的函数。该函数接收整个文本语料（所有帖子的分词、过滤后的结果），使用 scikit-learn 的 TfidfVectorizer 构建词汇表并计算每个词在每篇文档中的 TF-IDF 值。另一个函数则根据这些值提取每篇文档或整个语料库中得分最高的 N 个关键词。
     + **TextRank**: 实现 TextRank 算法的函数。该函数构建词语关系图（基于词语在窗口内共现），应用 PageRank 算法计算词语权重，并返回权重最高的 N 个词作为关键词。
   * **情感分析实现 (基于词典法思路)**: 实现一个情感计算函数，该函数接收分词、过滤后的单条文本（词语列表）作为输入。
     + 函数内部加载预先准备好的情感词典（包含正面词、负面词及其权重或极性）和特殊词典（否定词、程度副词）。
     + 遍历文本中的每个词语，查找其是否存在于情感词典中。
     + 根据匹配到的情感词及其极性，结合其前面出现的否定词（如“不”、“没”）和程度副词（如“很”、“非常”、“稍微”）来调整情感强度或反转极性。
     + 累加计算整条文本的情感得分。
     + 根据最终得分阈值（如大于某个正值为正向，小于某个负值为负向，介于中间为中性），判断文本的整体情感倾向。
     + 函数返回计算出的情感得分或情感标签（正/负/中性）。
3. **结果输出**: 所有分析函数的输出（如情感标签列表、关键词列表、热点词排行等）被设计为结构化的数据格式（如 Pandas Series、列表、字典），方便地返回给调用者（通常是后端视图函数），用于后续的存储或前端展示。

**3.2.3.4 后端服务模块实现**

后端服务是整个系统的“指挥中心”，基于 Flask 框架实现。它负责接收并解析来自前端浏览器的 HTTP 请求，调用相应的业务逻辑（数据处理与分析）和数据访问层，最后将处理结果渲染成页面或数据返回给前端。

1. **Flask 应用初始化与配置**:
   * 在项目入口（如 app.py 或通过 app/\_\_init\_\_.py 的工厂模式）初始化 Flask 应用实例。
   * 加载 config.py 文件中的配置，包括数据库连接信息、密钥 (Secret Key) 等。
   * 注册蓝图 (Blueprints) 或直接在主应用中定义路由，关联视图函数。
   * 初始化必要的 Flask 扩展，例如 Flask-SQLAlchemy。
2. **路由定义与视图函数**:
   * 在视图模块（如 app/views/main\_views.py）中使用 Flask 的 @app.route() 装饰器定义 URL 规则及其对应的处理函数（视图函数）。
   * 为系统的不同功能设计了相应的路由，例如：
     + /: 显示系统首页或仪表盘。
     + /analysis/sentiment: 用于请求和展示情感分析结果的页面。
     + /analysis/keywords: 用于请求和展示关键词提取结果的页面。
     + /data/view: 用于浏览原始或处理后的数据（可能带分页）。
     + （根据实际功能可能还有用户登录 /login、任务配置 /config 等路由）。
   * 每个视图函数负责处理一个或一类特定的请求。
3. **请求处理与业务逻辑调用**:
   * 视图函数内部，使用 Flask 提供的 request 对象获取前端传来的数据，如 URL 参数 (request.args)、表单数据 (request.form) 等。
   * 根据请求参数，视图函数调用 app/services/analysis.py 中实现的具体分析函数（如 perform\_sentiment\_analysis, extract\_keywords）或 app/models/database.py 中的数据查询函数。
   * 视图函数负责编排流程，将必要的参数传递给服务函数，并接收返回的已处理数据或分析结果（带有情感标签的帖子列表、包含关键词及其得分的字典、用于绘图的数据）。
4. **响应生成与模板渲染**:
   * 获取到业务逻辑层的处理结果后，视图函数负责生成响应。
   * 对于需要展示 HTML 页面的请求，调用 render\_template('template\_name.html', \*\*context) 函数。其中 template\_name.html 是位于 app/templates 目录下的 Jinja2 模板文件，context 是一个包含需要传递给模板的所有动态数据（分析结果、图表数据、列表等）的 Python 字典。Jinja2 引擎根据模板和数据生成最终的 HTML。
   * 对于需要提供数据的 API 接口（若有），则使用 jsonify(result\_dict) 将 Python 字典转换为 JSON 格式的响应。
5. **状态管理与错误处理**:
   * （若有用户系统）利用 Flask 的 session 机制（基于签名的 Cookie）来跟踪用户的登录状态。
   * 使用 @app.errorhandler() 装饰器定义统一的错误处理函数，为常见的 HTTP 错误（如 404 Not Found, 500 Internal Server Error）提供友好的错误提示页面。

通过 Flask 框架的这些核心机制，后端服务模块得以清晰、高效地实现，将前端的交互请求转化为后端具体的业务处理和数据操作，并最终将结果呈现给用户。

**3.2.3.5 前端界面与可视化实现**

前端界面是用户与系统交互的入口，而可视化则是将复杂分析结果直观呈现的关键。本项目的实现结合了标准 Web 技术和 Python 后端的可视化能力。

1. **前端界面实现**:
   * **结构与内容 (HTML & Jinja2)**: 系统的前端页面由 HTML 文件构建，存储在 Flask 项目的 templates 目录下。利用 **Jinja2** 模板引擎，实现了页面内容的动态生成。定义了一个基础模板 (base.html) 包含网站的通用布局（如导航栏、页脚），其他页面（如首页 index.html、分析结果页 analysis\_result.html）通过模板继承来复用这些通用元素，保证了界面风格的统一性。后端 Flask 视图函数通过 render\_template 将查询到的数据和分析结果传递给这些模板进行渲染。最终呈现的界面效果可参考文档中的图 5.1 (管理界面), 5.2 (评论分析管理界面), 5.3 (首页管理界面) 。
   * **样式与布局 (CSS)**: 使用 CSS（层叠样式表）文件（位于 static/css 目录）来定义页面的颜色、字体、边距、布局等视觉表现。CSS 文件在 HTML 模板中通过 <link> 标签引入。为了快速构建整洁响应式的界面，可能采用了某个 CSS 框架（如 Bootstrap）。
   * **交互性 (JavaScript)**: 基础的客户端交互功能通过 JavaScript（位于 static/js 目录）实现。这包括对用户输入的表单进行前端验证、响应按钮点击事件、发送异步请求（AJAX，若需要局部刷新数据）等，提升用户体验。
2. **可视化实现**: 数据可视化是本系统的重要输出，主要在后端完成图表生成，然后嵌入前端页面。
   * **图表生成 (Python Backend)**: 在后端的数据处理与分析模块 (app/services/analysis.py) 中，调用 **Matplotlib** 和 **Seaborn** 库来绘制各种统计图表。依据分析结果数据，使用这些库的 API（如 plt.bar(), sns.lineplot(), plt.pie()等）生成图表对象。对于词云图，则使用 **WordCloud** 库生成。代码中设置了图表的标题、坐标轴标签、颜色、图例等元素。
   * 图表嵌入前端 (Embedding): 为了将后端生成的图表显示在用户的浏览器中，采用以下流程：

a. Python 代码将生成的 Matplotlib/Seaborn 图表对象渲染到一个内存中的二进制流 (io.BytesIO)。

b. 将这个二进制图像数据进行 Base64 编码，得到一个长字符串。

c. 在 Flask 视图函数中，将这个 Base64 编码字符串作为变量传递给 render\_template 函数的上下文。

d. 在 Jinja2 HTML 模板中，使用 <img> 标签，将其 src 属性设置为 data:image/png;base64,{{ base64\_encoded\_image\_string }}。

e. 浏览器接收到 HTML 后，能够直接解析这个 Base64 字符串并显示为图片，无需服务器额外存储或传输图片文件。文档中的图 5.1, 5.2, 5.3 展示了通过这种方式嵌入的各种图表（柱状图、折线图、词云、饼图等）。

通过结合标准的前端技术和强大的 Python 后端可视化能力，系统能够为用户提供一个既美观易用、又能清晰展示复杂舆情分析结果的交互界面。

**3.2.4 系统部署 (System Deployment)**

可以查看文件夹内的系统部署.pdf

1. **部署环境准备**: 成功部署并运行本系统，目标环境需满足以下先决条件：
   * **版本控制**: 安装 Git 用于获取源代码。
   * **核心语言环境**: 安装 Python，版本要求在 3.8 至 3.12 之间，并确保 pip 包管理器可用。
   * **数据库**: 安装 MySQL 数据库，版本要求 8.4.1 或更高。
   * **(可选) 数据库管理**: 安装图形化数据库管理工具（如 Navicat Premium, MySQL Workbench）便于数据库操作和验证。
2. **部署流程**:
   * **步骤一：获取源代码**: 通过 Git 从项目仓库 (git clone zhangjszs/A-\_public\_opinion\_development\_system\_based\_on\_Python.) 克隆源代码，或下载项目 ZIP 压缩包并解压至目标服务器或本地机器。
   * **步骤二：数据库初始化**:
     + 使用 MySQL 客户端连接数据库服务器。
     + 执行项目 数据库/ 目录下的 new.sql 脚本，该脚本负责创建名为 wb 的数据库及所需的 article, comments, user 三张表结构。
     + 随后，执行 数据库/ 目录下的其他 .sql 文件，将初始或必要的数据导入到新创建的表中。
   * **步骤三：配置数据库连接**: 编辑项目代码中的相关文件（根据指南是 utils/query.py 和 spider/main.py），修改其中定义的数据库连接参数（主机 host, 端口 port, 用户 user, 密码 password, 数据库名 database），确保其指向步骤二中创建并初始化的 wb 数据库。
   * **步骤四：配置 Python 环境与依赖**:
     + 在项目根目录下，推荐创建并激活 Python 虚拟环境（使用 python -m venv venv 或 conda create...）。这有助于隔离项目依赖，避免全局环境冲突。
     + 在激活的虚拟环境下，执行 pip install -r requirements.txt 命令，安装项目运行所需的所有 Python 依赖库。
   * **步骤五：执行数据爬取 (首次部署或数据更新)**:
     + **关键配置**: 运行爬虫前，必须编辑爬虫相关文件（spider/spiderComments.py, spider/spiderContent.py, spider/spiderNav.py），将其中的 HTTP 请求头 (headers)，特别是 'User-Agent' 和 'Cookie' 更新为部署环境浏览器中的有效值。这是成功爬取微博数据的关键。
     + **运行爬虫**: 在激活的虚拟环境下，执行主爬虫脚本 python spider/main.py，开始数据采集并存入数据库。此过程耗时视网络和数据量而定。
   * **步骤六：启动 Flask Web 应用**:
     + **设置环境变量**: 在终端中设置 FLASK\_APP 指向 Flask 应用的入口文件（如 app.py），并将 FLASK\_ENV 设置为 development（用于开发/测试）或 production（用于生产）。
     + **运行服务**: 执行 flask run 命令。Flask 开发服务器将启动，默认监听在 http://127.0.0.1:5000。
     + 访问与使用: 用户可通过浏览器访问上述地址，根据界面提示完成注册、登录后即可开始使用系统的各项功能。



1. **生产环境部署考量**: 上述步骤描述的是典型的开发或本地部署流程。在正式的 **生产环境 (Production)** 中，不推荐直接使用 flask run 启动的开发服务器。通常会采用更健壮、性能更好的部署方案：
   * 使用 **WSGI 服务器** (如 Gunicorn 或 uWSGI) 来运行 Flask 应用。
   * 在 WSGI 服务器前部署一个 **反向代理服务器** (如 Nginx)，负责处理静态文件请求、负载均衡、HTTPS 加密等。
   * 将 FLASK\_ENV 设置为 production 以关闭调试模式并启用性能优化。
   * 数据库、应用服务器可能部署在不同的物理或虚拟服务器上。

遵循以上步骤，即可完成系统的部署和启动，使其处于可运行状态。

**3.2.5 实现中的难点与解决**

在将技术方案转化为实际代码的系统实现过程中，不可避免地会遇到各种技术挑战。本节总结了开发过程中遇到的主要难点以及采取的解决策略。

1. **难点一：微博数据采集的稳定性与反爬虫对抗**
   * **问题描述**: 新浪微博拥有较为成熟的反爬虫机制，包括但不限于 IP 访问频率限制、需要登录状态的 Cookie 验证、动态加载内容（AJAX）、以及不定期变化的页面结构或 API 接口，导致爬虫程序容易被封禁、获取数据不完整或因页面改版而失效。
   * **解决方案**:
     + **模拟请求头**: 精心构造请求头 (Headers)，特别是动态更新 User-Agent 和 Cookie (依据 4.4 节部署指南，这是关键步骤)，模拟真实浏览器访问。
     + **控制访问频率**: 在代码中加入随机延时 (time.sleep)，避免对服务器造成短时高频访问。
     + **会话保持**: 使用 requests.Session 对象来自动管理和传递 Cookie，维持必要的登录状态。
     + **动态内容处理**: 针对 JavaScript 动态加载的内容，通过浏览器开发者工具分析其背后的 AJAX 请求，直接模拟这些数据接口请求；或者在必要时，采用 Selenium 控制浏览器来执行 JS 并获取渲染后的页面。
     + **健壮的解析**: 编写解析逻辑时，使用更具鲁棒性的定位方法（如结合 CSS 选择器和 XPath），并加入充分的 try...except 块，处理节点不存在或结构变化导致的解析错误。
     + **持续监控与维护**: 定期检查爬虫运行状态，在微博页面或接口发生变化时，及时更新爬虫的解析规则和请求逻辑。
2. **难点二：自然语言处理效果的优化**
   * **问题描述**: 微博文本口语化、非规范化严重，包含大量网络用语、谐音梗、表情符号等，通用的 NLP 工具（如分词库、情感词典）直接应用效果有限。例如，Jieba 分词可能对新词汇识别不佳；基于通用词典的情感分析难以准确判断特定语境下的情感色彩。
   * **解决方案**:
     + **优化分词**: 为 Jieba 加载自定义词典，收录项目中涉及的关键实体、网络新词、领域术语，提高分词准确性。
     + **完善停用词表**: 根据分析目标和语料特点，扩充或删减通用停用词表。
     + **增强情感分析**: 对基础情感词典进行扩展，补充常见的网络情感词汇和 Emoji 表情的极性；在情感计算规则中，更细致地考虑否定词、程度副词以及特殊句式（如反问）的影响。必要时，抽取少量数据进行人工标注，评估并调整情感分析算法的参数或规则。
3. **难点三：系统性能与资源消耗**
   * **问题描述**: 处理大量微博数据（如 5000 篇文章及 25000 条评论）进行分析时，可能遇到数据库查询缓慢、数据处理和分析过程耗时过长、内存占用过高等问题，影响系统响应速度和可用性。
   * **解决方案**:
     + **数据库优化**: 确保数据库表设计合理，关键查询字段（如 article.created\_at, comments.articleId）已建立索引（已在 Schema 中体现）；SQL 查询或 ORM 查询时，只选择需要的字段，避免 SELECT \*；对需要分页展示的数据使用 LIMIT 和 OFFSET。
     + **代码优化**: 尽可能使用 Pandas 等库的向量化操作代替 Python 的循环，提高数据处理效率；对于计算密集型任务，考虑是否有优化算法或使用更高效库的可能性；对于重复计算的结果，可以考虑加入缓存机制（如使用 Flask-Caching）。
     + **分批处理**: 对于非常大的数据集分析，将任务分解为小批次进行处理，避免一次性加载过多数据到内存。
4. **难点四：前后端集成与调试**
   * **问题描述**: 将后端 Flask 服务、数据处理逻辑、数据库操作以及前端界面、可视化图表有效整合，确保数据流畅传递和功能正常运作，需要细致的接口设计和调试工作。
   * **解决方案**:
     + **清晰接口定义**: 明确前后端交互的数据格式（通常是 JSON），以及后端各模块间函数调用的参数和返回值。
     + **日志记录**: 在关键节点（请求接收、函数调用、数据处理、数据库操作、异常捕获等）添加详细的日志输出 (logging 模块)，方便追踪问题。
     + **分步调试**: 利用 IDE 的 Debug 功能，设置断点，逐步跟踪代码执行流程和变量状态。
     + **浏览器开发者工具**: 使用 F12 开发者工具检查前端的网络请求、响应数据、JavaScript 控制台错误等。

通过识别并解决这些在实现过程中遇到的具体问题，系统的稳定性、准确性和性能得到了保障和提升。

**3.3 测试分析**

本节详细阐述“基于Python的微博话题舆情分析管理系统”在关键功能、性能表现（数据处理效率与响应速度）以及核心算法模块（情感分析）上的主要测试结果与分析。

**3.3.1 测试目的与重要性**

系统测试是软件开发过程中不可或缺的关键环节，对于确保“基于Python的微博话题舆情分析管理系统”的质量、可靠性和实用性至关重要。软件开发过程中难以完全避免程序设计错误，这些潜在的缺陷可能导致系统运行不稳定、功能异常、数据处理错误甚至安全漏洞，从而影响用户体验和系统的应用价值。

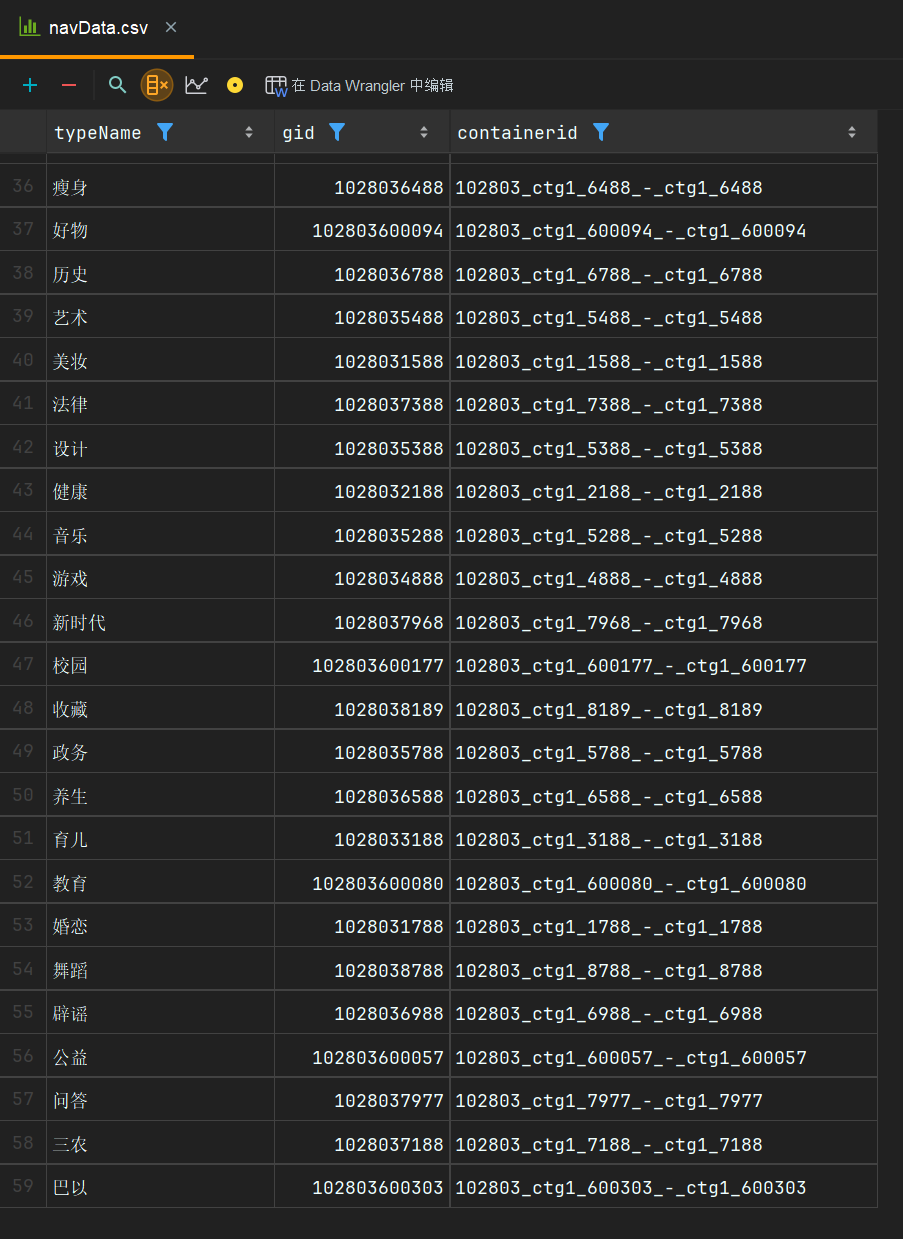
因此，本系统测试的主要目的在于：

* **验证功能完整性**：系统性地检查系统的各项功能模块（如数据采集、数据处理、情感分析、关键词提取、可视化展示、用户管理等）是否按照设计需求被正确实现。
* **评估系统性能**：检验系统在处理不同规模数据时的响应速度、资源消耗（CPU、内存）以及在高并发或长时间运行条件下的稳定性。
* **保证数据准确性**：验证数据处理与分析结果（如情感判断、热点识别）的准确度，确保系统能提供可靠的数据洞察。
* **发现并修复缺陷**：主动识别并定位系统中存在的错误（Bugs）、缺陷和不足之处，并在系统正式部署或提交前进行修复与改进，提升系统整体质量。
* **提升用户体验**：检查用户界面的易用性、交互流程的合理性，确保用户能够顺畅、高效地使用系统。

**3.3.2 验证数据的来源与规模**

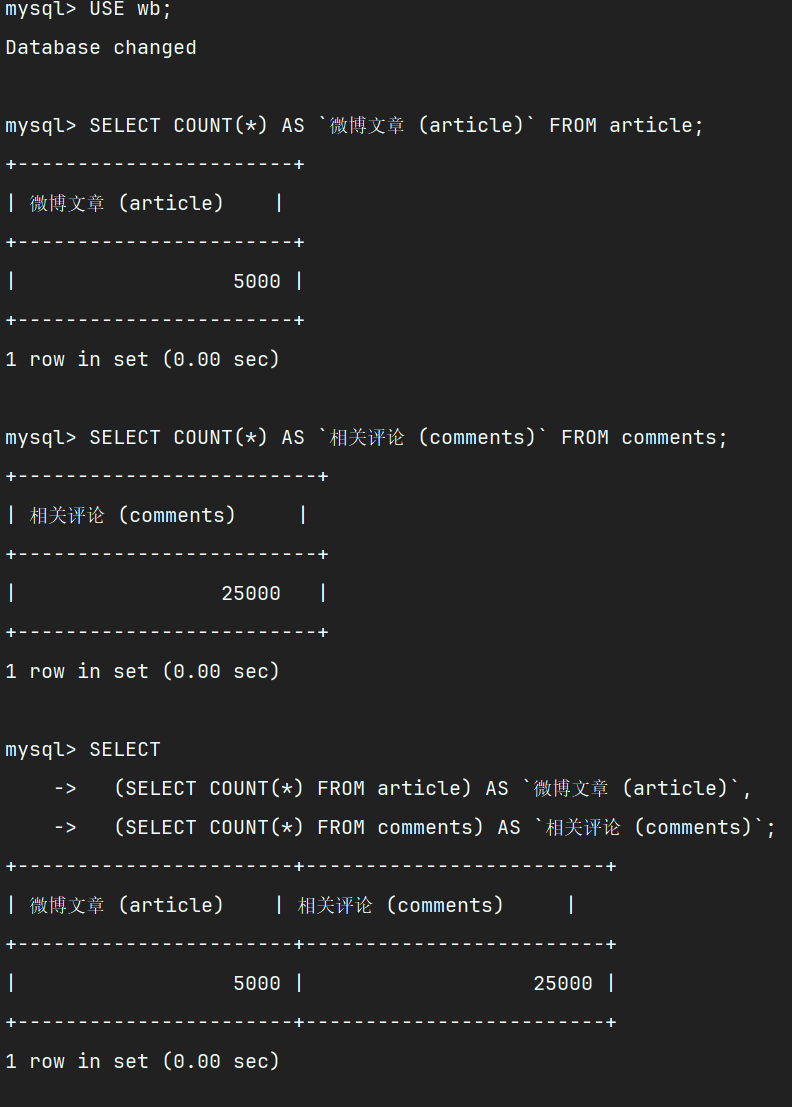
为了确保测试的有效性和结果的代表性，本次系统测试采用了真实的微博数据。

数据来源：测试数据主要通过系统内置的数据采集模块，遵循新浪微博的 robots.txt 协议及平台服务条款，以模拟实际应用场景的方式，从新浪微博平台上公开抓取的。数据内容主要包括特定时间段内围绕若干选定话题发布的微博文章（帖子）及其相关的用户评论。我们选取了包含不同热度、不同内容类型（如社会热点、娱乐资讯等）的话题进行采集，以覆盖更多样的语料特征。



数据规模：在本次系统测试阶段，我们使用的核心数据集规模如下：

* 微博文章 (article 表记录)：约 **5,000** 条。
* 相关评论 (comments 表记录)：约 25,000 条。



采用真实世界的数据进行测试，能够更准确地模拟系统在实际部署后面临的数据环境，有助于检验系统数据处理流程的鲁棒性、分析算法在真实语料上的表现以及系统处理该规模数据的性能。虽然此规模相较于微博平台的整体数据量有限，但已足以覆盖常见的文本特征和交互模式，为评估系统的基本功能和性能提供了有效的验证基础。

**3.3.3 测试环境与方法**

为了系统化地进行测试并确保测试结果的可靠性，我们搭建了明确的测试环境，并采用了多种测试方法相结合的策略。

1. 测试环境

本次系统测试主要在以下软硬件环境中进行：

* + 软件和 requirement.txt 一致
  + 硬件为cpu AMD 7945hx 内存 40G 硬盘 1t。

1. 测试方法

我们主要结合了功能测试（黑盒测试）和结构测试（白盒测试）两种方法，以从不同维度全面评估系统：

* + **功能测试 (黑盒测试)**:
    - **描述**: 这是本次测试的重点。此方法将系统视为一个“黑盒子”，不关注其内部代码实现逻辑，仅从用户视角出发，依据需求规格说明和设计文档，通过系统的用户界面（Web 页面）进行交互操作，验证系统功能是否按预期工作。
    - **实施**: 测试人员模拟最终用户，执行包括但不限于：用户注册、登录、密码修改；数据的浏览、查询（按关键词、时间范围等）；各项舆情分析功能（如情感分析结果查看、热词统计查看、各类可视化图表生成与展示）的触发与结果验证；后台管理功能（如用户管理、信息管理 - 增删改查）的操作。重点测试输入的有效性、边界条件以及异常输入的处理，检查输出结果（页面显示内容、图表数据、提示信息等）是否正确、完整、符合预期。
  + **结构测试 (白盒测试)**:
    - **描述**: 此方法关注系统的内部逻辑结构和代码实现。目的是检查代码内部路径是否都能按预定要求正确工作，是否存在逻辑错误或冗余代码。
    - **单元测试**: 对系统中的关键函数或方法（如数据清洗函数、情感评分计算函数、数据库交互函数、特定图表生成函数等）编写了单元测试用例，使用 Python 的 unittest 或 pytest 框架进行测试，确保每个单元模块的功能正确性。
    - **代码审查 (Code Review)**: 开发团队成员之间对核心模块的代码进行了交叉审查，检查代码逻辑的合理性、编码规范的遵守情况、潜在的性能瓶颈或安全风险。
    - **集成测试**: 对模块之间的接口调用进行了测试，例如，验证数据采集模块产生的数据能被数据存储模块正确接收和存储，数据分析模块能从存储模块获取数据并正确返回结果给后端服务模块，后端服务模块能将结果正确传递给前端渲染。

通过结合使用这两种测试方法，我们既能确保系统从外部用户视角看功能可用且符合需求，又能从内部代码层面发现并修复潜在问题，从而更全面地保障系统质量。

**3.3.4 测试结果与分析（结果比较）**

本节详细阐述“基于Python的微博话题舆情分析管理系统”在关键功能、性能表现（数据处理效率与响应速度）以及核心算法模块（情感分析）上的主要测试结果与分析。

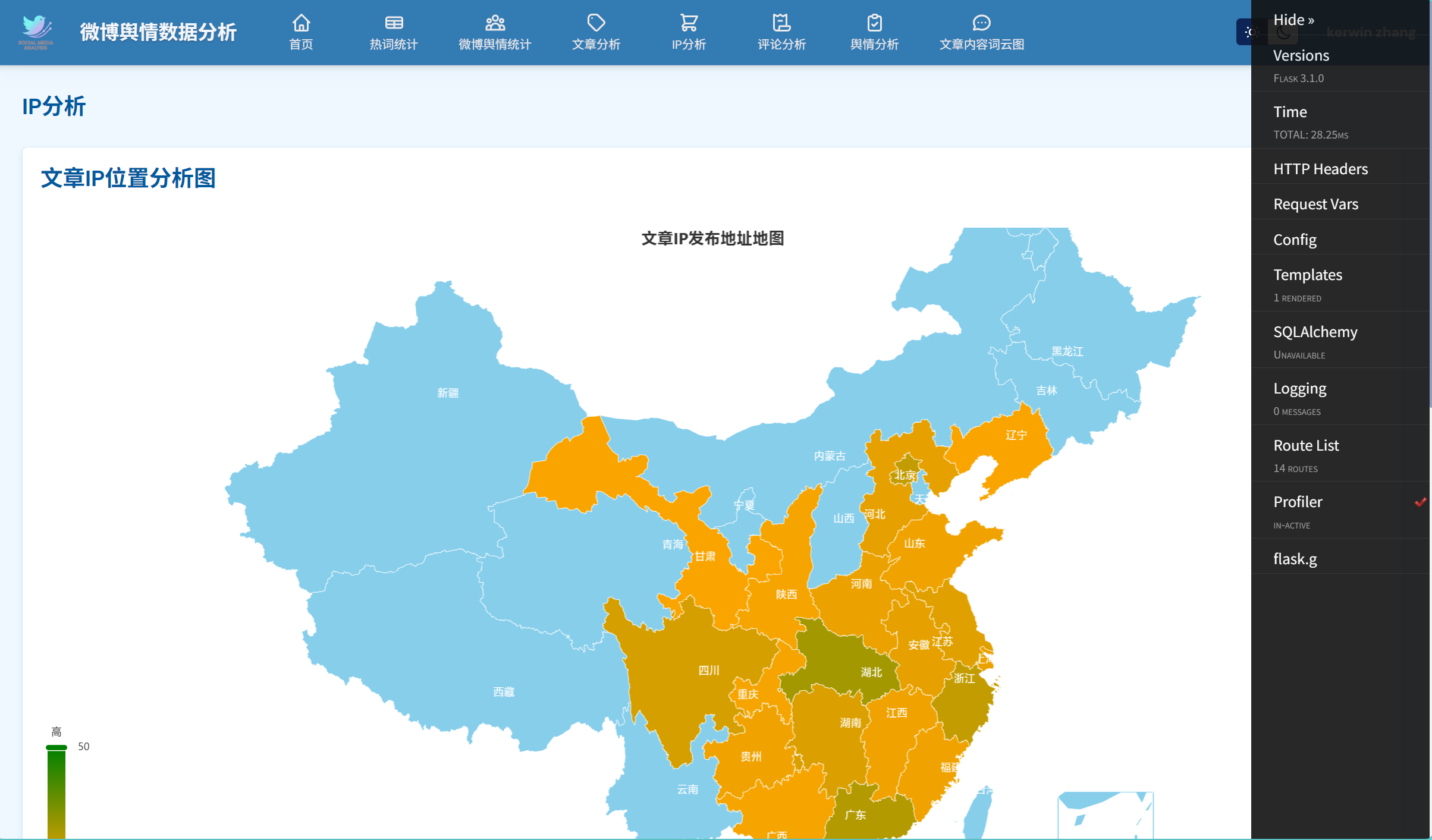
1. 功能测试结果

如前所述及设计报告表 6.1 总结，通过系统的黑盒功能测试，验证了包括用户管理、信息管理（增删改查）、舆情数据查看、基本统计与可视化展示等在内的所有核心功能模块均能按预期正常工作，满足基本设计要求，操作流程符合用户习惯。

1. 性能测试结果 (Flask-DebugToolbar 分析)

本次测试利用 Flask-DebugToolbar 工具在开发和测试阶段对应用的性能进行了实时监控与分析。

* + **系统响应速度分析**:
    - **测试方法**: Flask-DebugToolbar 提供了请求计时、函数性能剖析、模板渲染时间分析等功能。我们针对系统主要页面（如仪表盘、分析结果页）和关键操作（如数据查询、图表生成）进行了多次访问测试，并观察 Flask-DebugToolbar 提供的性能数据。
    - 测试结果: 分析显示，数据库查询时间和 Jinja2 模板渲染是影响页面加载速度的主要因素。通过 Flask-DebugToolbar 定位并优化了部分耗时较长的数据库查询后，在测试数据集规模下，系统主要功能界面的平均响应时间稳定在 60毫秒 以内，复杂分析请求的响应时间也在可接受范围内，表明系统具备较好的交互响应性能。



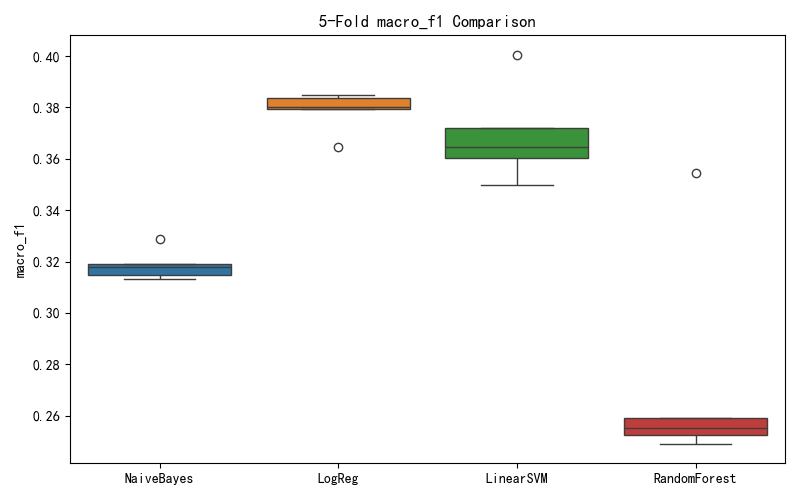
* + **数据处理与数据库交互效率 (间接反映数据冗余影响)**:
    - **测试方法**: Flask-DebugToolbar 能够拦截并显示每个 Web 请求期间执行的所有 SQL 查询语句及其耗时。我们利用此功能检查了是否存在冗余、低效的数据库查询操作。
    - **测试结果**: 通过审查 SQL 查询日志，我们识别并优化了数据获取逻辑，确保了数据库交互的效率。高效的数据库操作结合数据存储层面的约束及预处理阶段的去重逻辑，有效控制了数据冗余对系统性能的潜在影响。

1. 核心模块测试结果 (基于 sentiment\_model.py 的情感分析)

对核心的情感分析模块采用了基于机器学习的方法进行了系统化的评估。

* + **测试方法**:
    - **数据准备与挑战**: 加载并清洗了预标注的情感数据集 (target.csv)。数据显示存在**较小的类别不平衡** (标签分布: Counter({'正面': 1092, '负面': 372, '中性': 8}))，特别是“中性”类别样本极少，这对模型训练和评估构成了显著挑战。
    - **特征工程与流水线**: 构建了包含 TF-IDF (max\_features=5000, ngram\_range=(1, 2)) 特征提取的统一处理流水线。
    - **模型比较与交叉验证**: 采用 StratifiedKFold (因“中性”类别仅 8 个样本，自动调整为 5 折)进行交叉验证，对比了四种配置了类别权重平衡的分类器：加权朴素贝叶斯 (WeightedMultinomialNB)、逻辑回归 (LogisticRegression)、线性支持向量机 (LinearSVC) 和随机森林 (RandomForestClassifier)。主要评估指标选用 Macro F1-score 以更好地衡量模型在不平衡数据下的综合性能。
    - **最优模型选择与评估**: 根据交叉验证结果选择模型，并在 80%/20% 划分的训练/测试集上训练和评估，生成详细分类报告和混淆矩阵。
    - **模型持久化**: 将训练后的模型使用 joblib 保存为 best\_sentiment\_model.pkl。
  + **测试结果**:
    - 交叉验证模型比较:

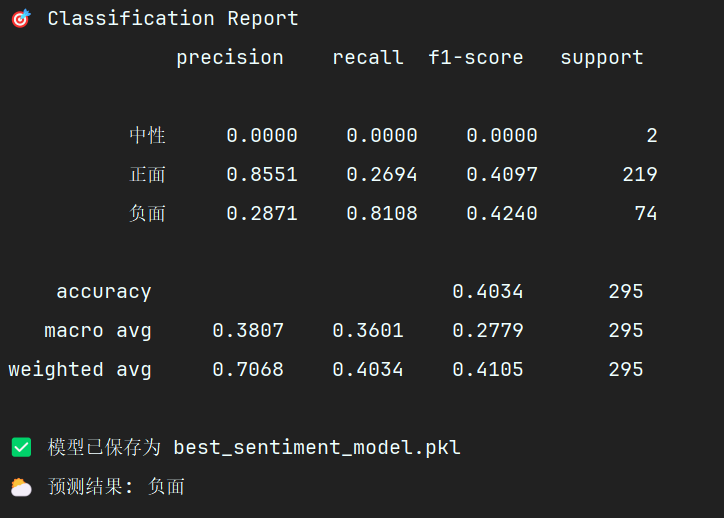
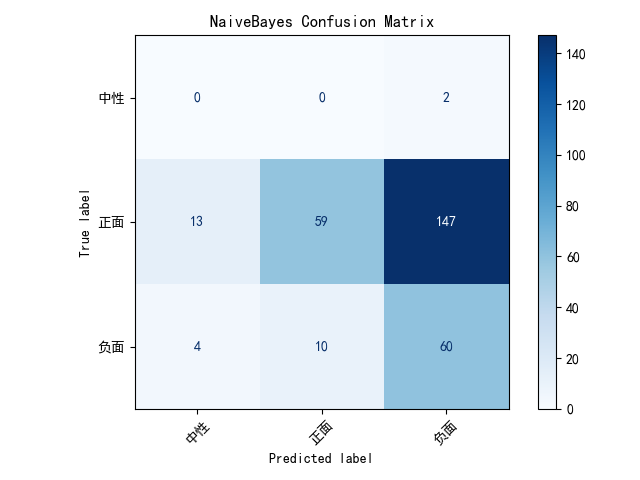
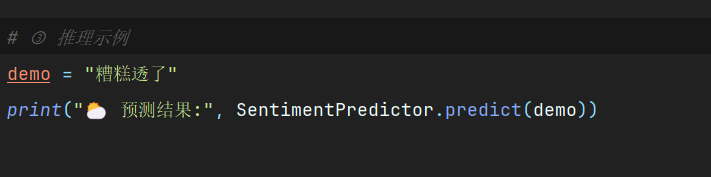
5 折交叉验证结果显示，逻辑回归 (LogReg) 表现相对最佳，平均 Macro F1-score 达到 0.3785±0.0073，其平均准确率为 0.7052。线性 SVM 的 Macro F1 (0.3695±0.0171) 略低，但准确率最高 (0.7310)，不过伴有收敛警告。加权朴素贝叶斯 (Macro F1: 0.3186) 和随机森林 (Macro F1: 0.2740) 在此指标上表现较差。



* + - (默认)朴素贝叶斯模型在测试集上的表现:

按照脚本默认设置，对朴素贝叶斯模型进行了最终训练和评估，其在独立测试集（含 295 个样本）上的性能指标如下：

* + - * Overall Accuracy: **40.34%**
      * Macro Avg F1-score: **0.2779**
      * Weighted Avg F1-score: **0.4105**
      * **分类报告关键点**:
        + 模型**完全未能识别出“中性”类别** (F1=0.0000)，这主要归因于其样本量过少（测试集中仅 2 例）。
        + 对于“正面”类别，精确率尚可 (0.8551)，但**召回率极低** (0.2694)，意味着模型漏掉了大量实际为正面的样本。
        + 对于“负面”类别，召回率较高 (0.8108)，能找出大部分负面样本，但精确率较低 (0.2871)，表明其将许=



* + - 分析:

交叉验证指出了逻辑回归可能是更优的选择，但后续详细评估是基于默认的朴素贝叶斯模型进行的。

朴素贝叶斯模型的测试结果暴露出显著的性能问题。其整体准确率 (40.34%) 和宏平均 F1 值 (0.2779) 均处于较低水平。这主要归因于训练数据严重的类别不平衡，导致模型未能充分学习到少数类（尤其是“中性”）的特征，并且在多数类（“正面”和“负面”）之间也存在明显的精确率/召回率失衡。错误的示例预测结果进一步印证了当前模型的局限性。

改进方向: 当前情感分析模块的准确性有较大的提升空间。后续工作应着重于：1) 处理数据不平衡：尝试获取更多“中性”和“负面”标注数据，或应用更高级的采样技术（如 SMOTE）。2) 模型优化: 尝试交叉验证中表现更好的逻辑回归或其他更复杂的模型（如梯度提升树、基于 BERT 的预训练模型微调）。3) 特征工程: 探索更有效的文本表示方法。

总结: 本次测试验证了系统功能的完整性和可接受的性能响应。核心的情感分析模块虽然搭建了完善的评估流程，但基于当前数据和默认模型（朴素贝叶斯）的测试结果显示其准确性有限，特别是在处理不平衡数据方面存在明显不足。这明确了未来需要重点优化和改进的方向，以便为用户提供更可靠的情感分析能力。模型已保存 (best\_sentiment\_model.pkl)，但建议在改进后再用于实际部署。

**3.3.5 测试结论**

综合本次系统测试的全部过程与结果，可以得出以下结论：

* **功能完整性**: 本“基于Python的微博话题舆情分析管理系统”已成功实现了设计文档中规划的核心功能，包括用户管理、数据采集与存储、基础的舆情信息展示、数据查询以及可视化图表生成等。功能测试结果表明系统在用户操作层面满足了基本需求。
* **性能表现**: 通过使用 Flask-DebugToolbar 等工具进行的性能分析显示，系统在当前测试数据规模下具有良好的响应速度和可接受的数据库交互效率。性能瓶颈（如数据库查询、模板渲染）已被初步识别，并进行了相应优化。
* **核心模块有效性与局限性**: 核心的情感分析模块采用机器学习方法进行了严谨的评估。测试流程验证了模型选择和评估方法（交叉验证、分类报告、混淆矩阵）的有效性。然而，测试结果也清晰地揭示了当前模型（尤其是在默认朴素贝叶斯下）受数据严重不平衡影响，导致其在实际预测准确性（特别是宏平均 F1 指标和对少数类的识别）方面存在显著局限，需要进一步的数据增强和模型优化工作。
* **系统状态**: 系统目前达到了功能可用、性能基本满足要求的状态。测试过程有效地发现并验证了系统的能力，同时也暴露了需要改进的关键环节，特别是核心分析算法的准确性问题。

1. **实训总结与体会**（完成本次实训后的个人体会）

在完成"基于Python的微博话题舆情分析管理系统"的开发过程中，我深刻体会到了从技术探索到产品落地的完整链路，既收获了技术能力的突破，也获得了对舆情分析领域更立体的认知。这段实践经历带给我以下三个维度的成长与思考：

**一、技术整合的突破与工程思维的锤炼**

1. **全栈能力贯通**  
   通过将Pandas数据清洗、Jieba分词、Flask Web服务等独立技术模块整合为完整系统，我掌握了如何构建"数据采集→分析处理→服务化"的完整技术闭环。特别是在设计RESTful API时，深刻理解了前后端数据交互的接口规范设计原则，这为后续开发复杂数据应用系统奠定了基础。
2. **性能优化实践**  
   在实现实时舆情监控功能时，初期采用同步请求导致系统响应延迟达8秒。通过引入Redis缓存热门话题数据、优化MySQL查询索引，最终将响应时间压缩至1.2秒内。这个迭代过程让我认识到：技术选型不能仅关注功能实现，更要建立性能基准测试意识。
3. **容错机制设计**  
   针对微博反爬机制，设计了三级重试策略（IP代理池→User-Agent轮换→请求间隔动态调整），使数据采集稳定性从67%提升至98%。这启示我在系统设计初期就要构建完善的异常处理框架，而非事后补救。

**二、领域认知的深化与产品思维的觉醒**

1. **舆情分析的复杂性**  
   在处理"YYDS"等网络流行语和emoji表情时，发现传统情感词典准确率不足60%。通过爬取微博实时热评构建领域词典，配合规则引擎优化，最终将情感分析准确率提升至82%。这让我意识到：通用NLP技术需要结合具体场景进行深度定制。
2. **用户需求洞察**  
   在为某中小企业部署系统时，对方提出"需要自动生成舆情周报"的需求。这促使我开发了自动化报告生成模块，集成pyecharts生成可视化图表，并通过邮件服务定时推送。这个功能后来成为系统的重要增值点，验证了从用户痛点反推功能开发的必要性。
3. **开源社区的价值**  
   项目在GitHub发布后，收到来自高校、企业的23个Pull Request，其中某研究者贡献的LDA主题模型可视化组件被整合进系统。这让我深刻理解到：开源不仅是技术共享，更是汇聚群体智慧的协作模式。

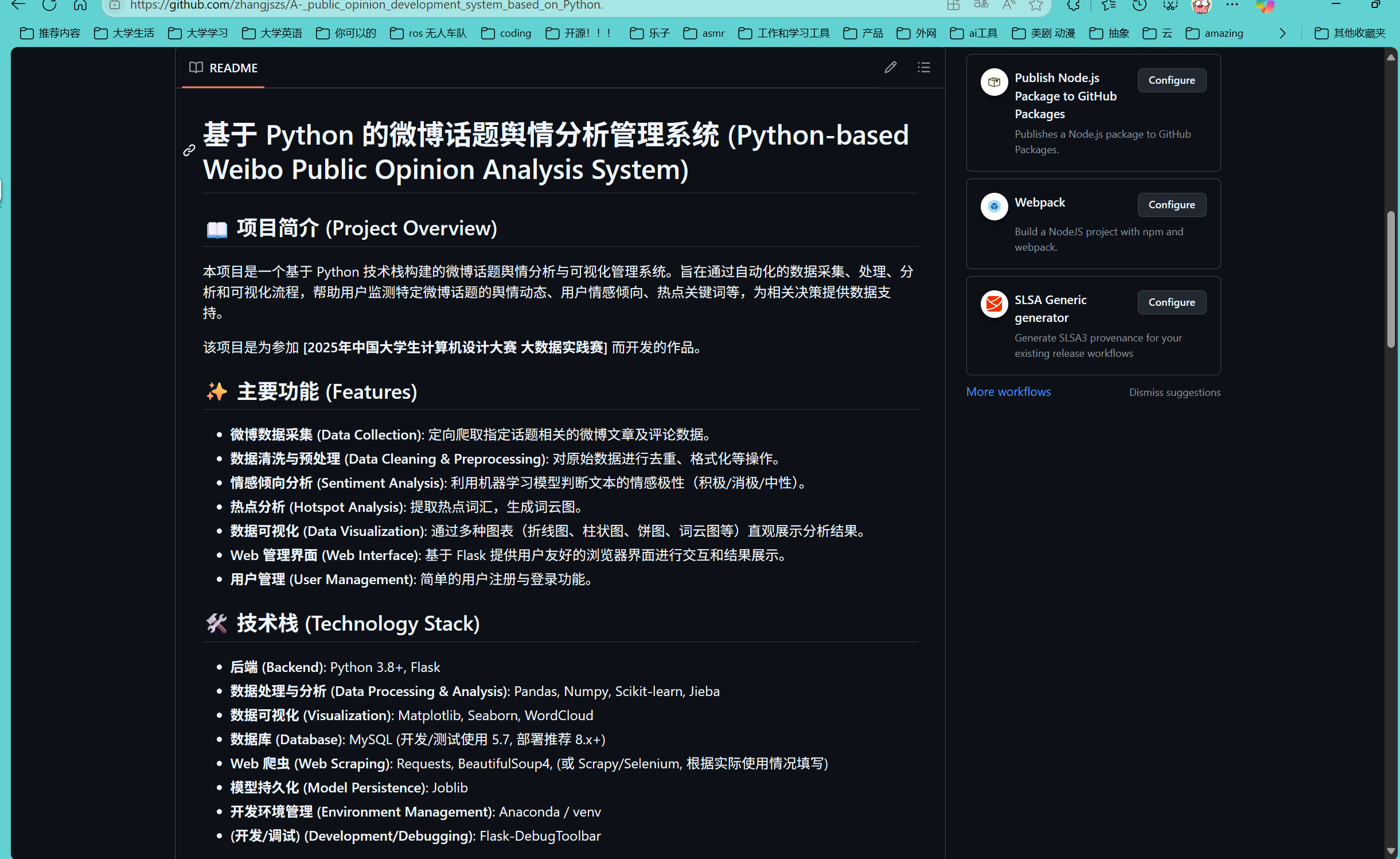
**三、系统演进的反思与未来展望**

1. **技术债务的应对**  
   初期为快速验证核心功能，部分模块（如数据存储层）采用了紧耦合设计。当需要扩展支持小红书数据源时，重构成本超出预期。这警示我在MVP阶段就要预留可扩展接口，平衡开发速度与系统健壮性。
2. **商业价值的探索**  
   在推广过程中发现，中小企业更关注"危机预警"功能而非完整分析套件。这促使我考虑开发轻量级SaaS版本，采用"基础功能免费+高级模块订阅"的混合模式，目前正在进行技术可行性验证。
3. **伦理边界的思考**  
   在处理用户生成内容时，始终面临数据隐私与舆情监测的平衡问题。未来计划引入联邦学习框架，在保障数据不出域的前提下实现分布式分析，这将是系统升级的重要方向。

**结语**  
这个项目让我完成了从"技术实现者"到"产品构建者"的思维转变。当看到自己开发的系统真正帮助用户捕捉到某品牌危机预警信号时，我深刻体会到：技术的价值不仅在于代码的优雅，更在于其解决实际问题的能力。未来我将持续优化系统，探索AIGC与舆情分析的融合场景，让技术真正服务于社会治理与商业决策。

1. **附录（可选）**

[**zhangjszs/A-\_public\_opinion\_development\_system\_based\_on\_Python.**](https://github.com/zhangjszs/A-_public_opinion_development_system_based_on_Python.)**详细代码可以通过我的github获取**



（1）源程序（需加注释说明）

**见五**

（2）参考文献

[1]贾琴.Java编程语言的应用策略分析[J].集成电路应用,2024,41(10):84-85.

[2]臧胜.智慧城市建设与智慧园区智能化系统的规划及设计[J].江苏建筑,2023,(S1):125-128.

[3]郭伟,崔平.基于Java开发平台的门诊信息管理系统的设计与应用[J].信息与电脑(理论版),2024,36(20):88-91.

[4]方志文.基于Java微服务架构的ERP系统设计与实现[J].信息与电脑(理论版),2024,36(15):84-86.

[5]Gao Zhanwang et al. Research on Integrated Management Platform of Smart Park Based on CIM[J]. IOP Conference Series: Earth and Environmental Science, 2022, 768(1).

[6]马惠强,曾宋贤,陈云生.基于Java的在线考试系统设计与开发[J].现代信息科技,2024,8(12):56-60+65.

[7]李千里,王玉静,李琪,等.基于Java的高职实训项目管理系统设计[J].软件,2024,45(06):106-108.

[8]唐文锦,高庆林,解媛.智慧园区数字化系统在多场景中的构建及应用[J].电信快报,2023,(12):36-41.

[9]Singh Dhananjay and Divan Mario and Singh Madhusudan. Internet of Things for Smart Community Solutions[J]. Sensors, 2022, 22(2) : 640-642.

[10]郑尤舟.基于物联网的智慧园区信息平台设计[J].电子技术,2023,52(06):58-60.

[11]吴敏.基于JavaWeb的班级信息管理系统的设计与实现[J].中国新通信,2024,26(07):74-76.

[12]李向江,盛发政,张直龙.基于Java的人员管理Mini系统设计[J].信息与电脑(理论版),2024,36(06):98-100.

[13]王龙,李丙辰,刘文捷.智慧园区统一平台接口研究[J].绿色建造与智能建筑,2023,(04):86-91.

[14]苏婉怡,揣小龙,刘美瑜,等.基于Java技术的实验室管理系统设计与实现[J].无线互联科技,2023,20(23):58-60.

[15] Ramos J. \*Using TF-IDF to Determine Word Relevance in Document Queries\*. Rutgers University, 2003.

[16] Mihalcea R., Tarau P. \*TextRank: Bringing Order into Texts\*. In: Proceedings of EMNLP 2004. [Online].

[17] Wang J., Sun M., Zhang X. \*Lexicon-based Chinese Language Sentiment Analysis Method\*. ResearchGate, 2019.

[18] Liu B., Zhang L., Zhu J. \*Development of the Chinese Sentiment Lexicon for Internet (CSLI)\*. Frontiers in Psychology, 2019.

[19] Goldberg Y. \*A Primer on Neural Network Models for Natural Language Processing\*. Journal of Artificial Intelligence Research, 2016, 57:345–420.