分类号TP31 编号

烟 台 大 学

毕 业 论 文（设 计）

**新浪微博的数据获取及分析**

**Acquisition and Analyzing of Sina Weibo Data**

申请学位： 工学学士

院 系：计算机与控制工程学院

专 业： 软件工程

姓 名： 王志扬

学 号： 201558503236

指导老师： 郭艳燕（讲师）

2019年 6月 1日

烟台大学计算机与控制工程学院

**新浪微博的数据获取及分析**

姓 名： 王志扬

导 师： 郭艳燕

2019年 6月 1日

烟台大学计算机与控制工程学院

**[摘要]** 随着计算机和信息技术的发展以及网络传输技术的更新换代，各行各业产生的数据均呈现指数级的增长，同时，近期企业、学术界对大数据的研究如火如荼。新浪微博正是最贴近生活最能反应当前热点并含有海量数据的社交平台之一，本项目旨在仿效企业级开发从新浪微博平台针对性的获取大量数据，对数据分析研究。得益于近期实习中的开发经验和对当前计算机界热门技术的学习和分析，本系统通过基于Scrapy框架的Python爬虫技术获取数据，基于训练和自优化的词库断词处理数据，基于VueJs、SpringBoot前后端分离框架配合MySQL关系型数据库做数据分析展示，基于训练线性回归模型预测热度走势。本系统获取数据和已处理的数据均达到百万量级，已具备数据分析的基础，并以此为基础分析并可视化展示了大V（拥有大量粉丝的微博用户，文中将这批用户均称作大V）热度排行及热度走势、大V塑造形象与粉丝反馈差异对比、话题热度排行及走势、全网用户信息分布、全网最常用字词/表情等可视化的数据模型。

**[关键词]** 新浪微博；爬虫；SpringBoot-Vue；前后端分离

**Abstract:** With the development of computer and information and the upgrading of network transmission technology, the data produced by all walks of life are increasing exponentially.The research on big data in the enterprise academia is in full swing recently,too. As we all know, sina weibo is one of the social platforms closest to life that can reflect the current hot issues and contain massive data. This project aims to imitate the enterprise development to obtain a large amount of data from the sina weibo platform, analyze and study the data, and make display and prediction with the visual interface. Benefited from the recent internshIP project development experience and the current computer industry hot technology learning and analysis, this system intends to obtain data through the Python crawler technology based on Scrapy framework, process data based on training and self-optimized word database, analyze and display data based on VueJs SpringBoot front-end framework and MySQL relational database, and predict heat trend based on training linear regression model. Up to now, the data acquired and processed in this system have reached millions, and the foundation of data analysis has been established. Based on this, we draw the heat ranking and heat trend of big V(Weibo users with a large number of fans are called big V in this paper) heat rankings and heat trends ,the big V shape differences between image and fan feedback contrast, the heat number and distribution situation, so the user information, so the most commonly used words/expressions such as visual data model.

**Key words:** SinaWeibo；Spider；SpringBoot-Vue； Front- rear Separation

目录

[1 绪论 1](#_Toc10744488)

[1.1 课题研究意义 1](#_Toc10744489)

[1.1.1 大数据研究的意义和新浪微博数据的价值 1](#_Toc10744490)

[1.1.2 Python爬虫技术与JavaWeb技术的研究意义 1](#_Toc10744491)

[1.2 国内外研究现状及发展趋势 1](#_Toc10744492)

[1.3 系统设计目的和思想 2](#_Toc10744493)

[2 系统分析 3](#_Toc10744494)

[2.1 系统功能特点 3](#_Toc10744495)

[2.2 系统需求分析 3](#_Toc10744496)

[2.2.1 功能需求分析 3](#_Toc10744497)

[2.2.2 非功能性需求分析 4](#_Toc10744498)

[2.3 系统可行性分析 4](#_Toc10744499)

[3 系统概要设计 6](#_Toc10744500)

[3.1 功能模块划分 6](#_Toc10744501)

[3.2 系统的设计风格和模式 7](#_Toc10744502)

[3.2.1 MVC的设计风格 7](#_Toc10744503)

[3.2.2 B/S的设计架构 8](#_Toc10744504)

[3.3 数据库设计 8](#_Toc10744505)

[3.3.1 数据库关键表设计 8](#_Toc10744506)

[3.3.2 基于InnoDb存储引擎的MySQL数据库优化探索 10](#_Toc10744507)

[4 系统框架的整合与设计 13](#_Toc10744508)

[4.1 设计思路及思想简述 13](#_Toc10744509)

[4.2 涉及技术及框架简述 13](#_Toc10744510)

[4.2.1 SpringBoot + SpringDataJpa后台框架 13](#_Toc10744511)

[4.2.2 Vue前端框架 13](#_Toc10744512)

[4.2.3 Axios及PromiseJs 14](#_Toc10744513)

[4.3 前后端完全分离思想 14](#_Toc10744514)

[4.3.1 实现方式 14](#_Toc10744515)

[4.3.2 为什么摒弃NodeJs平台？ 14](#_Toc10744516)

[4.3.3 前后端分离开发、交互逻辑 14](#_Toc10744517)

[4.4前后端交互的统一 15](#_Toc10744518)

[4.5后台合并控制逻辑层和业务逻辑层 16](#_Toc10744519)

[4.6界面扁平化和数据可视化UI设计风格 16](#_Toc10744520)

[4.7基于SeaJS的前端模块化设计 16](#_Toc10744521)

[4.8 框架搭建要点和难点 17](#_Toc10744522)

[4.8.1 公共模块的引入 17](#_Toc10744523)

[4.8.2 HttpVueloader转译Vue文件 17](#_Toc10744524)

[4.8.3 Vuex、Vue Router的实例化与配置 18](#_Toc10744525)

[4.8.4 导航守卫、动态路由、登陆状态 19](#_Toc10744526)

[4.8.5 Axios配置与跨域访问 20](#_Toc10744527)

[4.8.6 前后端分离导致的Session问题 21](#_Toc10744528)

[4.8.7 后台SpringBoot+SpringDatajpa框架的搭建 21](#_Toc10744529)

[5 系统详细设计及实现 22](#_Toc10744530)

[5.1 数据获取部分 22](#_Toc10744531)

[5.1.1 Scrapy框架搭建及应用 22](#_Toc10744532)

[5.1.2 数据获取详细设计 22](#_Toc10744533)

[5.1.3 与微博反爬虫机制的对抗 25](#_Toc10744534)

[5.1.4 数据获取广度与深度 26](#_Toc10744535)

[5.1.5 多线程与分布式爬虫的设想 27](#_Toc10744536)

[5.2 数据处理部分 27](#_Toc10744537)

[5.2.1 数据处理实现及流程 27](#_Toc10744538)

[5.2.2 最大正向匹配+最大正向查询断词算法 27](#_Toc10744539)

[5.2.3 词库的自扩展 29](#_Toc10744540)

[5.2.4 Snownlp简介 30](#_Toc10744541)

[5.3 数据分析、展示、预测部分 30](#_Toc10744542)

[5.3.1 登录、登出 30](#_Toc10744543)

[5.3.2 系统数据的总览 30](#_Toc10744544)

[5.3.3 大V热度及其人物画像分析 32](#_Toc10744545)

[5.3.4 话题热度分析及预测 36](#_Toc10744546)

[5.3.5 微博用户分布 37](#_Toc10744547)

[5.3.6 全网热词、热门表情 38](#_Toc10744548)

[5.3.7 中文数据处理 40](#_Toc10744549)

[5.3.8 词库管理、系统消息、配置与调度 41](#_Toc10744550)

[5.4 存储过程与事件 44](#_Toc10744551)

[6 系统测试 46](#_Toc10744552)

[6.1 数据获取速度测试 46](#_Toc10744553)

[6.2 数据处理负载测试 46](#_Toc10744554)

[6.3 数据分析展示功能点测试 47](#_Toc10744555)

[结束语 48](#_Toc10744556)

[致谢 49](#_Toc10744557)

[参考文献 50](#_Toc10744558)

[附录A功能测试部分Api表 51](#_Toc10744559)

[附录 一元线性回归模型原理及推导 52](#_Toc10744560)

# 1 绪论

## 1.1 课题研究意义

### 1.1.1 大数据研究的意义和新浪微博数据的价值

随着网络技术的不断发展和网络传输技术的更新换代，数据越来越“值钱”，同时越来越多的人意识到数据的重要性。但我们所常见的数据大多是海量的非结构化的数据“贫矿”，这需要我们来挖掘出来，通过有效的分析手段分析方法，提炼归类整合出其中有价值的“矿藏”将其模型化结构化，进而从中获取商业价值社会热点，同样大数据也是所有机器学习的基础，“No data, no learning. No data, no intelligence. Data is the blood. ”机器学习和统计分析永远也不会离开数据、离开有价值结构化的数据。

新浪微博正是最贴近生活最能反应当前热点并含有海量数据的社交平台之一，假如你是一名微博的用户，应该不难发现，微博是我们能够获取热点的最快捷途径，同样也是一个个热点事件发酵最快的平台之一，正如前段时间“崔永元效应”以及近期的“蔡徐坤事件”，几乎是一夜之间，响应全网。所以新浪微博大量数据具有一定的现实意义。

### 1.1.2 Python爬虫技术与JavaWeb技术的研究意义

Python是更注重可读性和效率的语言，更易于学习。同时Python拥有最成熟的程序包资源，这使得在爬虫获取数据并整理数据方面更方便更简单，本项目采用的Scrapy爬虫框架便是一个成熟的Python资源包。在商业应用中，Python的应用也越来越广，Python几乎能够完成其他开发语言实现的一切功能，学习Python并将其用精对深处IT行业的我们至关重要。在另一方面，本系统的数据展示使用JavaWeb技术，Java拥有足够多的成熟的web开源框架、技术上没有瓶颈和局限性等优势毋庸置疑是做web项目的首选，本项目将采用优秀的SpringBoot框架来进行后台开发，通过学习和应用这些新技术可以快速有效的提高我们的开发水平和掌握当前的先进前沿技术，以快速适应当前严峻的软件行业就业环境。

本系统采Python和Java两种开发语言开发，便是利用各自的长处完成其适用的模块，是提高系统开发效率最根本方式之一。另一方面，在完成不同开发语言开发的模块后，整合成“系统”，思考各模块的内在联系和相互调用监控方法，进而提高个人开发中的整合能力。

## 1.2 国内外研究现状及发展趋势

在数据爆发式增长的今天，移动终端和移动互联网的快速发展，数据增长速度远超想象。世界上越来越多的国家和组织认识到大数据的战略意义和应用价值，大多数国家也开始部署和发展大数据战略。在企业层面，有越来越多的企业也投入到大数据研发之中，像IBM、Google、百度、微软、Oracle等知名企业都参与其中。同样从数据中获取研究价值对于学术研究来讲也越来越热。

Python开发正变得越来越受欢迎，被广泛应用到网络爬虫和其他开发领域，从简单的个人数据研究到信息检索，再到全世界大型银行的数据挖掘、机器学习都有所涉及。但很多平台服务器对爬虫进行了严格的限制，以致于除Google crawler、Mercator几个爬虫示例外几乎找不到其他开源的商业爬虫示例[1]。在数据为王的时代，Python因其特性具有较高的研究价值。

另外，Java毋庸置疑是当前最主流的开发语言，是Web开发方向最主流的技术之一。基于其跨平台的特性，Java应用无处不在。全球每天有数十亿台服务器运行Java用于处理繁多的数据，像天猫、苏宁、京东等大部分企业都是选用Java作为基本架构来开发的。整个互联网在不断的发展，Java技术也在不断进步，Java应用会越来越广泛。

## 1.3 系统设计目的和思想

（1）本系统旨在通过Python爬虫技术，获取新浪微博平台大量数据，包括微博用户信息、用户的微博信息、微博评论信息等数据；通过对微博数据、评论数据处理获取文本的积极程度，统计分析出现的话题、表情、字词频次；之后通过JavaWeb技术，将获取的数据进行可视化的呈现，包括用户的热度及热度趋势预测、用户微博体现的情感程度与粉丝对其评价的情感程度分析、微博用户粉丝的地区年龄分布、热门话题排行、热门表情/热词排行及事件趋势等。通过这一系列的过程，实现对微博数据的获取、处理、分析、展示和预测。

（2）本系统将最大程度的遵循企业级项目开发规范，包括命名规范、目录结构规范、表结构规范、相关注释规范等，并使用git做版本控制，以期提高代码的可读性和可维护性，同时有助于代码审查，避免出现不该出现的bug。即使出现bug，优秀的符合规范的代码将大大降低维护成本。在实际的生产开发中，因其更注重团队合作，更应遵循这样的规范，来提高团队开发效率。另外，在本系统中对近期实习中遇到的不合理或不恰当的开发模式思想进行了创新性的改进，以期提高开发效率，减少本系统的开发周期。

# 2 系统分析

## 2.1 系统功能特点

（1）高效性

进行微博数据的分析展示和预测的前置条件均是拥有具有大量较高参考价值性的数据，所以高效的获取数据是本系统的关键之一。本系统通过优化爬虫速度，实现数据获取速度达到每小时4万条（环境：20Mbps宽带，I5-5200U处理器，KingSton SS300S37A硬盘），每日最大获取量接近百万级，因此作为个人研究本系统已具备高效性的特点。

（2）系统性

本系统包含完整的数据获取、数据处理、数据分析、数据展示和数据预测生态性功能，具备完整的系统性。Python数据获取及处理、Java后台、Vue前端“各司其职”，又能相互协调统一（具备完善的日志消息系统、优秀的交互逻辑）。因此本系统具备较为完整的系统性。

（3）研究性

截至目前，本系统获取数据和已处理的数据均达到百万量级，已具备数据分析的基础，并以此为基础绘出大V热度排行及热度走势、大V塑造形象与粉丝反馈差异对比、话题热度排行及走势、全网用户信息分布、全网最常用字词/表情等可视化的数据模型。因此本系统具有较高的研究性价值。

## 2.2 系统需求分析

### 2.2.1 功能需求分析

本系统拟在获取新浪微博平台大量数据，并予以处理分析加工，进而展示一定热点趋势，反映社会现状，预测热度趋势。因此本系统分为数据获取、数据处理、数据分析、数据展示与相关配置、数据预测五个模块，具体分析如下：

（1）数据获取：数据获取在本系统中是其他模块的前置条件，为有效全面的分析微博数据，要求Python爬虫程序必须高效广泛的获取数据。一方面需要优化爬虫效率，另一方面当爬虫效率优化到瓶颈时，亦需要权衡获取数据的广度和深度。在获取内容上需要获取微博的用户信息、用户发表的微博信息、微博对应的评论以及用户间关系[5]。

（2）数据处理：数据处理需要对获取数据中的文本（微博信息和评论信息）进一步分析加工，获取文段的情感积极程度，记录其中存在的话题标签、表情标签，清洗无价值字段数据信息，对清洗后的数据使用字词库做分词处理，获取记录关键词词频，并通过大量文本数据优化扩展词库。

（3）数据分析：由于获取数据和已处理数据分析都不可能达到全面，处理数据的时空复杂度远超数据获取，处理速度远慢于获取速度，因而不可能处理完所有已获取的数据，只能从一个点到一个随机的面获取随机数据、处理数据，所以应用统计学思想从一部分数据分析整体数据，对之后数据展示模块尤为重要。

（4）数据展示与相关配置：此模块是本系统与用户/访客交互最直接的模块，用户可以查看数据获取、分析、处理后的可视化数据模型，管理员则拥有除此之外的系统配置、词库管理、查看日志等管理权限。展示数据对象主要包括：对当前系统总览及运行状态的首页监控与展示，用户分布、大V热度分析及大V形象数据模型展示、话题的热度及分析的微博数据展示，全网热词、表情的展示，基于分词算法中文数据处理展示等。系统整体以简化操作、便用查看为基本原则，采用扁平化、数据可视的设计风格。

（5）数据预测：数据预测也是系统核心需求之一。通过筛选出核心数据，用恰当数据训练相应的预测模型以期最大程度减少误差，推算出数据、热度变化可能出现的趋势。在本系统中拟采用训练一元线性回归模型的方法实现。

### 2.2.2 非功能性需求分析

（1）交互易用性

界面设计合理，添加必要的提示性信息以及参考文档。同时界面UI统一，采用扁平化数据可视化的设计风格。

1. 可维护性

系统最大程度遵循企业开发规范，规范命名、规范代码缩进、规范目录结构、规范注释等，github做仓库使用git做版本控制，来保证系统的可维护性。

1. 跨平台性

系统采用的Java、Python程序开发语言，以Java的虚拟机（JVM）的运行方式和Python解释性语言的特点，二者均具备跨平台性，在不同的硬件或操作系统下都能运行。

1. 可靠性

设计系统必须考虑全面，对潜在的异常在程序上予以改进，对于复杂数据量大的查询修改进行优化避免超时，系统测试阶段测试样例要广泛、测试细致，保证程序的可靠性。

## 2.3 系统可行性分析

如课题研究意义所述，一个日活跃度超过2亿（出自新浪2018年财报）用户的社交平台，每天产生的几亿级数据。数据具有足够的广泛性，足够支撑起我们数据。另一方面，新浪微博平台反爬虫机制（验证IP封帐号机制）较为宽松，允许适量适速地获取数据。

Python拥有最成熟的程序包资源，这使得在爬虫获取数据并整理数据更方便更简单，本项目采用的Scrapy框架便是一个优秀的Python爬虫框架。本系统所获取的数据也具备一定的关联性，可形成数据爬取闭环，通过微博用户id可以获取到微博用户页面，用户页面可获取到用户信息、发表微博信息的url、粉丝列表关注列表的url，粉丝列表关注列表又可以获取到用户id…[5]获取流程参考图2.1数据获取的原理示意图。

获取的数据具有随机性和广泛性，当获取数据样本足够大时，获取的数据也具有足够的广泛性，足以通过样本来估计整体，分析数据。

所采用的SpringBoot-Vue前后端分离框架，具有优秀的交互逻辑，强大的数据处理请求能力，配合饿了么公司ElementUI前端UI组件库、基于百度eCharts的V-Charts UI组件库能够有效合理的展示数据。

获取的数据数据（微博、微博评论、处理的数据）都可获取相关信息时间分布集合，通过时间分布的数据集合可以训练出描述数据趋势的回归模型，通过该模型即可预测数据的走势，具有预测的可行性。

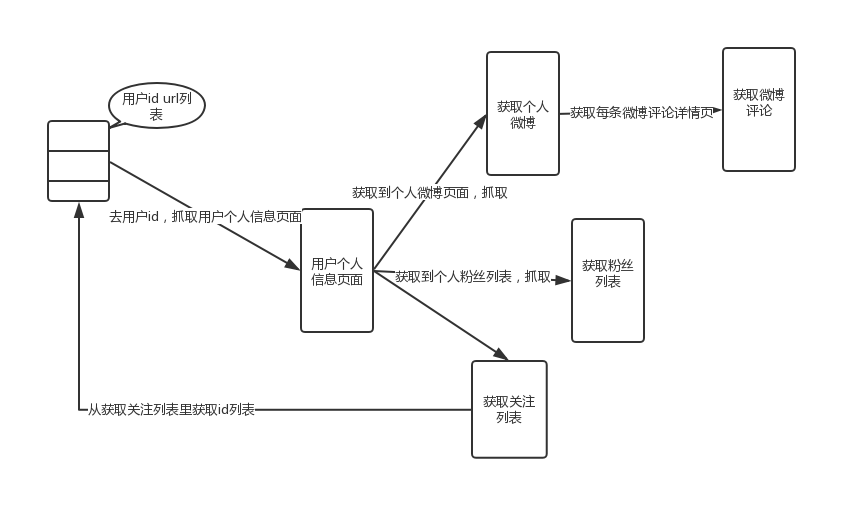


图2.1 数据获取的原理示意图

# 3 系统概要设计

## 3.1 功能模块划分

新浪微博数据获取分析系统主要包含基于Scrapy爬虫的数据获取模块、基于Python的数据处理模块、基于SpringBoot-Vue的数据分析数据展示数据预测模块，如图3.1所示。

数据获取模块主要包含：

（1）获取用户微博所有数据：通过不断通过关注列表的id爬取用户的全部信息。

（2）获取用户的用户信息：通过粉丝列表的id获取粉丝用户的用户信息。

数据处理模块主要包含：

（1）处理已获取的微博信息

（2）处理已获取的评论信息

数据分析展示预测模块主要包含：

（1）登录模块：访客无需验证即可登录，可查看平台各种数据；管理员需通过账号密码登录，除查看数据外，还拥有除查看数据之外的系统配置、词库管理、查看日志等管理权限

（2）系统首页：显示当前获取处理数据量的总览以及系统运行状态

（3）微博数据分析：包含大V热度分析、话题热度分析、微博用户分析等

（4）全网字词/表情分析：包含全网热词分析、全网表情分析等

（5）中文数据处理：用以展现断句算法和对Vue的一些最基本应用

（6）词库管理：包含基础词库管理，疑似新词词库管理

（7）系统消息：获取系统的提示消息，包含爬虫的状态，数据处理的状态

（8）配置与调度中心：管理存储过程/事件，设置爬虫爬取的“深度”等

各个模块详细设计见图3.2-3.4所示

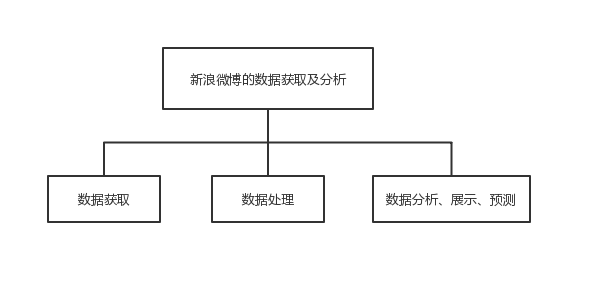


图3.1 系统3大模块

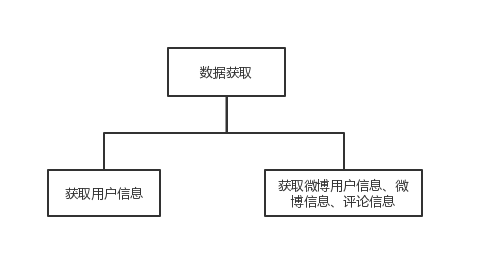


图3.2 数据获取模块图

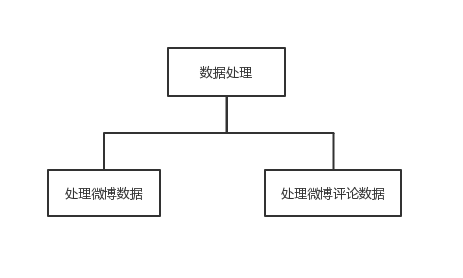


图3.3数据处理模块

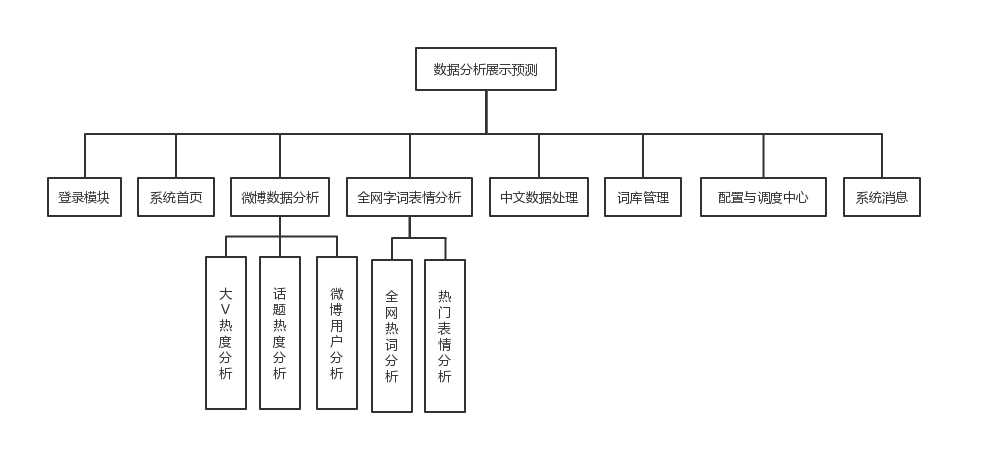


图3.4 系统后台模块图

## 3.2 系统的设计风格和模式

### 3.2.1 MVC的设计风格

MVC设计风格是一种被广为应用的系统设计风格，其将系统软件分为模型（Model）、视图（View）、控制器（Controller）三个基本部分，详见图3.5。而在本系统中，将Model、Controller在SpringBoot-SpringDataJap下封装为系统后台端，将Views视图层在Vue前端框架封装为系统前端，前后端分离，独立开发、独立部署、独立运行，框架详细搭建在第4章系统框架的整合和设计中详细描述。

### 3.2.2 B/S的设计架构

本系统数据展示数据分析预测模块为传统意义上的Web项目，因此必然遵循Bower-Server的设计架构。本系统的前后端分离设计的思想更符合这种设计模式。前后端独立运行，其体现便是Bower端浏览器运行项目前端程序，而Sever端便是本系统的后台端。

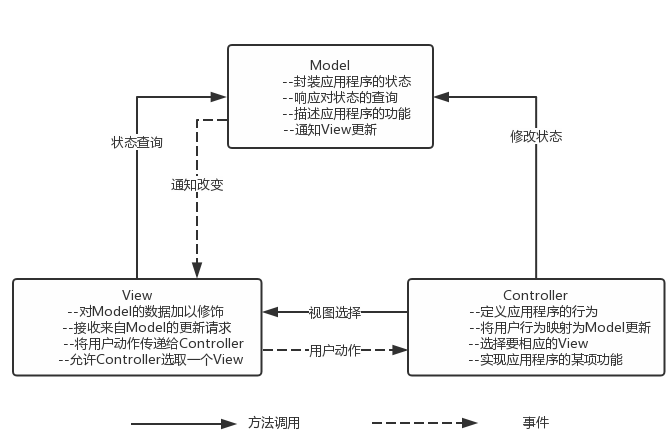


图3.5 MVC模型、视图、控制器关系图

## 3.3 数据库设计

### 3.3.1 数据库关键表设计

在此只介绍本项目中微博数据获取相关的表，其他系统表在“系统实现”的“框架搭建”章节有所描述。

（1）base\_word基础词库表，索引：id,word， 用于存储断词所需词库的数据表。

表3.1基础词库表

| 字段名称 | 数据类型 | 字段长度 | 备注 |
| --- | --- | --- | --- |
| Id | int | 4 | 自增主键，无实际意义 |
| Word | varchar | 12 | 字词 |
| Counts | int | 4 | 字词出现的次数 |
| Is\_show | int | 4 | 是否展示/断词是否入库 |
| Type | varchar | 10 | 类型 |
| Word\_length | int | 4 | 字词的长度 |
| Sentiments | int | 4 | 字词的情感色彩 |
| Pinyin | varchar | 15 | 字词的拼音 |

（2）weibo\_user\_info微博用户表，索引：无，用于存储获取用户的数据表。

表3.2微博用户表

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 字段长度 | 备注 |
| Id | varchar | 15 | 主键，用户id |
| nick\_name | varchar | 20 | 昵称 |
| brief\_introduction | varchar | 255 | 自述 |
| gender | varchar | 15 | 性别 |
| province | varchar | 15 | 省份 |
| birthday | varchar | 15 | 生日 |
| tweets\_num | int | 4 | 微博数量 |
| follows\_num | int | 4 | 关注量 |
| fans\_num | int | 4 | 粉丝量 |
| vip\_level | int | 4 | vip等级 |
| labels | varchar | 255 | 标签 |

（3）weibo\_info 微博数据表，索引：id,created\_at 用于记录微博数据。

表3.3微博数据表

| 字段名称 | 数据类型 | 字段长度 | 备注 |
| --- | --- | --- | --- |
| Id | int | 4 | 自增主键，无实际意义 |
| weibo\_url | varchar | 100 | 微博url |
| user\_id | varchar | 15 | 用户id |
| content | varchar | 255 | 内容 |
| created\_at | varchar | 15 | 发表时间 |
| repost\_num | int | 4 | 转发量 |
| comment\_num | int | 4 | 评论量 |
| like\_num | int | 4 | 点赞量 |
| isProcess | int | 4 | 是否已处理，0未处理 |
| sentiments | int | 4 | 情感状态 |

（4）weibo\_comment 微博评论表，索引：id, created\_at，用于存储微博评论。

表3.4微博评论表

| 字段名称 | 数据类型 | 字段长度 | 备注 |
| --- | --- | --- | --- |
| Id | varchar | 255 | 自增主键，无实际意义 |
| weibo\_url | varchar | 50 | 微博url |
| comment\_user\_id | varchar | 15 | 用户id |
| content | varchar | 255 | 内容 |
| created\_at | varchar | 15 | 发表时间 |
| isProcess | int | 4 | 是否已处理，0未处理 |
| sentiments | int | 4 | 情感状态 |

（5）weibo\_topic 微博话题表，索引：id，用于存储微博话题与用户关系。

表3.5微博话题表

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 字段长度 | 备注 |
| Id | Int | 4 | 自增主键，无实际意义 |
| user\_id | varchar | 15 | 用户id |
| topic | varchar | 50 | 内容 |

（6）weibo\_base\_data 微博基础数据表，索引：id，name用于存储字词/表情数据。

表3.6微博基础数据表

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 字段长度 | 备注 |
| id | int | 4 | Id自增主键，无实际意义 |
| user\_id | varchar | 15 | 用户id |
| name | varchar | 15 | 词/表情名 |
| counts | int | 4 | 出现次数 |
| type | varchar | 15 | 类型word/expression |
| date | varchar | 15 | 日期 |
| up\_user | varchar | 15 | 评论中的评论对象id |

### 3.3.2 基于InnoDb存储引擎的MySQL数据库优化探索

本系统使用基于InnoDb存储引擎的MySQL关系型数据库存储数据，其具备高效、轻便、开源等特点。但由于本系统个别表具有庞大的数据量以及个人电脑I/O存储速度上的制约，数据量达到百万级后查询效率明显变低，这也使得优化数据库查询成为系统关键之一。在此将从MySQL底层、原理详述本系统查询优化思路。

（1）InnoDb存储引擎的理解

InnoDb存储引擎区别于其他存储引擎最大的特点是其数据文件本身就是基于B+Tree构造的的索引文件（主索引、聚簇索引），并非一般意义上的索引、数据文件分离。在索引树上，其叶子节点域上直接存储数据，索引的“key”是数据表的主键[2]。示意图见图3.6。

其次，InnoDb存储引擎建立的data域索引（辅助索引）是用来记录主键信息而非数据域地址。该原理是优化count查询的关键。

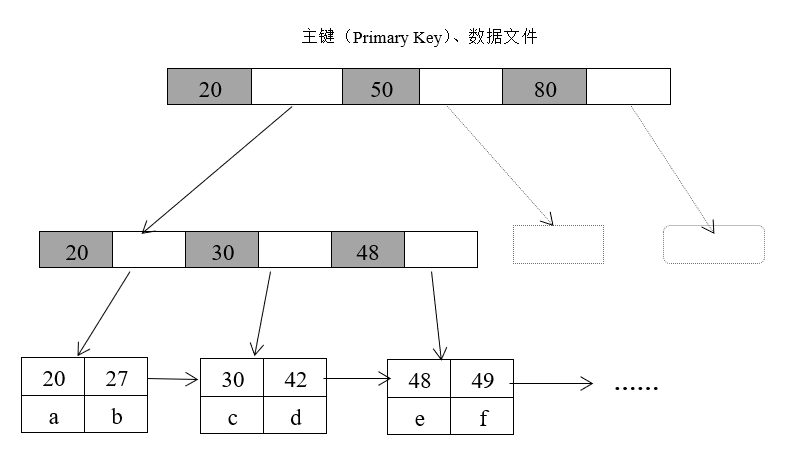


图3.6 InnoDb存储引擎原理示意图

（2）建立自增主键

通过上文的描述，数据库表的主键即为InnoDb引擎中的聚簇索引，数据在聚簇索引的叶子节点上有序排列。若设置表的主键为非自增主键，每次插入的数据的主键可近似于无序的、随机的，InnoDb引擎需要不断在数据存储的叶子节点间寻找合适位置甚至是移动已有的数据来插入新数据，原理如图3.7。若有大量数据插入，系统额外开销不言而喻；相反，采用自增的主键，每次插入数据都会是存储在前一条数据之后，其效率和系统开支都会大大优于前者。

但用Int型自增主键也存在关联查询时主键重复、不具备实际意义的缺点，在本系统开发中，在具有大量数据、数据读写频率大的表（如微博信息表、微博评论表、基础字库表）采用自增主键，在存储数据量较少的系统表均采用varchar型的具有实际含义的主键。

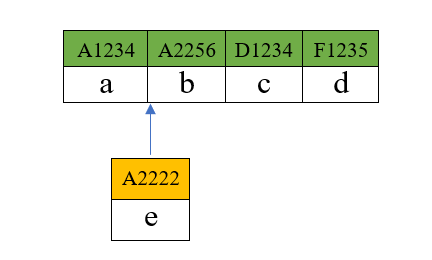


图3.7 采用非自增主键插入演示图

（3）建立辅助索引加快count查询

本系统中存在大量的count(\*)查询数据条数的操作，而在基于InnoDb存储引擎的MySQL数据库中count（\*）需要先从硬盘中将所有数据读入内存缓冲区中，之后通过整体行数或某字段行数计算数据量。尽管在这个过程中依靠聚簇索引从同一个节点获取数据能大大加快查询速度，但在大量数据、大量数据节点前提下，查询速度依旧不尽如人意。在前文Innodb存储原理中讲到辅助索引每一行记录都是每一列的主键列，这样添加辅助索引后，执行count(\*)操作会自动触发mySQL使用辅助索引即主键的值来统计数量的机制，这时存入内存缓冲区的数据只有主键一个字段，从而大幅减少系统的I/O读写，加快查询速度。另一方面，为进一步加快此机制速度，使用数据较小的自增主键必不可少。速度对比见图3.8、图3.9，同样的数据查询，220万条数据能有三倍的速度提升。

当然建立索引能够加快查询速度，但这并不意味着越多越好。添加索引后，每次数据插入和数据修改，MySQL都会为维持原有数据表的索引而耗费资源，进而降低插入、修改的速度。在本系统仅对数据量大、多用于查询的几个表适当添加辅助索引以优化查询速度。

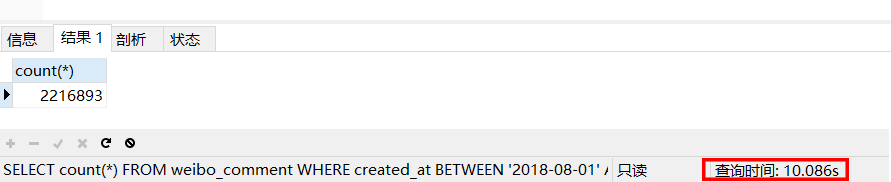


图3.8 仅使用聚簇索引

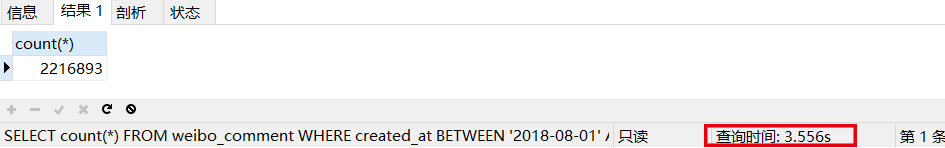


图3.9 使用聚簇索引和辅助索引

# 4 系统框架的整合与设计

## 4.1 设计思路及思想简述

在web的开发中，无论使采用何种框架，其核心思路、目的均是提高开发效率、降低开发成本、提高程序的健壮性和性能，当然，本系统框架设计的初衷也无不例外。本章主要介绍系统中关于JavaWeb框架的整合和设计。

热衷于框架的系统级改造，往往是经历了现有框架的摧残。在前段时间的实习中，应聘Java后台工程师，而实际从事的是包括前后端的全栈式开发，前端后台团队同时开发。整个项目下来，在前端展示质量、后台运行效率都存在严重的问题；另一方面前后端杂糅的开发框架极其不利于后期系统维护、bug解决。为解决上述问题，项目结束后,项目组开始了Bof-Vue框架（原公司架构师命名）的改造，本人有幸参与了项目改造的前期工作。

本系统参考原实习公司Bof-Vue框架改造项目思想，进一步实现前后端分离，提高开发效率，降低学习成本，整合设计出基于SpringBoot、SpringDataJpa、Vue、Axios、httpVueLoader等技术、摒弃nodeJs平台的SPA式前后端分离框架。下文将详细介绍该框架的整合和设计。

## 4.2 涉及技术及框架简述

### 4.2.1 SpringBoot + SpringDataJpa后台框架

SpringBoot是近几年相当火热的快速开发框架，其开发目是简化Web应用搭建及开发过程，摒弃了大量的XML配置操作，转而使用易于理解和使用的注解形式、实现约定大于配置的开发思路，同时集成了大量的内嵌应用服务器（如Tomcat）[3]。

SpringDataJpa 是基于Hibernate ORM框架、实现Jpa规范的一整套后台数据访问的解决方案，对于开发最实用的是提供了一套基于表字段映射的查询机制，极大简化了开发。SpringDataJpa也是SpringBoot官方推荐配合使用的ORM框架。

### 4.2.2 Vue前端框架

Vue是基于JavaScrIPt的一套渐进式的前端框架，Vue的核心只关注视图层操作而不像Jquery关注底层对Dom树（前端文本对象模型）结构操作。其数据双重绑定（v-bind）、条件渲染（v-if）、列表渲染（v-for）、类绑定（v-class）等操作都极大的简化了前端开发过程[4]。

同时Vue提供了Vue Cli、Vue loader等官方脚手架工具快速开发和运行Vue项目，VueX、Vue Router来适用于大型项目的系统状态保存、页面路由跳转。值得注意的是，在本项目中没有用到Vue Cli脚手架和 Vue Loader转而采用httpVueLoader来离线加载。

### 4.2.3 Axios及PromiseJs

PromiseJs是基于ES6开发规范的一部编程的解决方案，通过获取异步请求中的Promise对象的请求状态添加不同的回调方法。

Axios是基于PromiseJs用于浏览器请求的http客户端，用于本系统中的前后端数据交互，也是Vue官方推荐使用的异步请求工具。

## 4.3 前后端完全分离思想

### 4.3.1 实现方式

本框架区别于众多大型公司采用的基于NodeJs的前后端分离模式，采用前端负责View视图层和页面转发（路由跳转）、后端负责Model层和Controller层的数据交互的SPA（单页Web应用3）式的前后端分离框架。在数据交互的过程中，前端通过Axios请求后台获取数据渲染页面，通过路由的概念实现不同组件的渲染即完成“页面跳转”。构建“离线版”的Vue框架以完全摒弃NodeJs平台。

### 4.3.2 为什么摒弃NodeJs平台？

首先需要思考的是为什么不采用基于NodeJs的前端负责View层和Controller层、后台负责Model层的前后端分离框架。基于NodeJs前后端分离视图如图4.1所示，在原本前端后台之间再添加一层NodeJs控制层，其必然会带来性能、安全性、学习成本等问题。另一方面，传统认识中的Web项目，前端人员应专注于界面的设计、UI的交互、数据渲染等方面，后台人员专注于数据处理。将控制层完整下放到前端，虽使前端人员拥有了更大的数据控制权限，但也致使其工作量加大，不能专注于界面、视觉的设计。在本框架中认为，前后端分离本质上应是要前端后端回归定义的初衷，即前端人员回归到设计，后端人员专注于逻辑。

其次，在大多数企业开发中，开发人员使用的环境多为封闭（无网）环境，而Vue官方推荐的框架搭建必须依赖基于NodeJs的Vue Cli脚手架工具，致使在封闭环境下搭建框架异常复杂（NodeJS部署、安装、模块安装等）。故而在本框架中将完全摒弃NodeJs环境，单纯采用index.html做入口文件，引入并实例化（基于SeaJs模块化思想）Vue及其他组件（Vue Router、Vue X），使用httpVueLoader转译.Vue文件，构建“离线版”的Vue框架。基于上述构建的Vue前端框架无需基于NodeJs平台的复杂环境配置和部署，仅需加载到Apache服务下即可运行（该Vue前端框架本质上是仅含有html、CSS、JS的web页面）。应用本框架可极大提高部署、开发效率。

搭建基于“离线版”Vue SPA式的前后端分离web项目，去除其他繁杂的部署，正是回归到B/S设计模式的根本（在本框架中Bower-Service可映射为前端-后台）。

### 4.3.3 前后端分离开发、交互逻辑

如上述，搭建出的SPA式的前后端完全分离框架，前端和后台单独开发，亦可在不同的IDE（集成开发平台）下开发，前后端开发的系统可单独打包，单独部署在不同的服务器。在某种意义上，此时的前后端分别看作是两个系统。前端通过Axios异步请求后台，后台返回前端数据，前端将数据渲染到页面，交互图见图4.2本系统前后端交互逻辑。

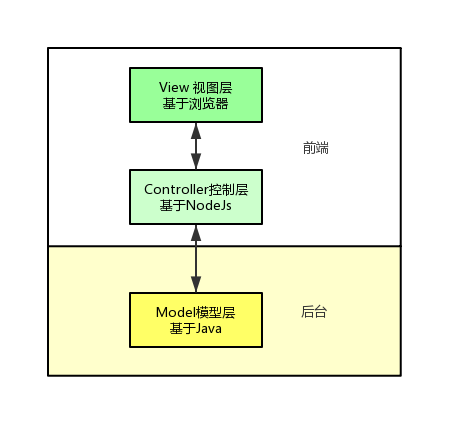


图4.1 基于NodeJs前后端分离视图

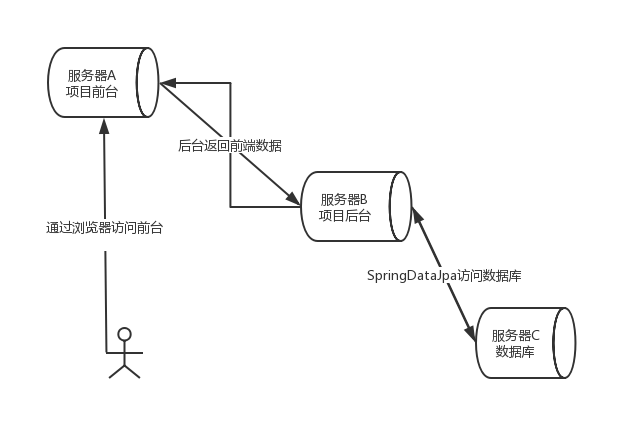


图4.2 本框架前后端交互逻辑

## 4.4前后端交互的统一

为简化前后台交互的开发过程，提高开发效率，前后台Axios请求参数和方式进行规范。如下：

（1）请求方式统一为Post方式

Post请求几乎可以代替所有的Get请求，另一方面post请求可上传的数据比Get大得多，统一使用Post请求可以有效的避免上传数据过大（url过长）引起的浏览器报错问题。故在本框架内均采用get请求。

（2）前端请求数据类型统一

在开发时前端统一向后台传输Js对象格式，后台通过 @RequestBody Map类型接收。Js对象和Map类型均可以接收任意数据，足以实现全部逻辑。

（3）后台返回类型统一

后台接收前端Axios请求处理逻辑后，返回数据统一用Results对象介绍，Results对象包括：String型的message接收消息、Boolean型的success接收请求事件是否成功，Object型data接收数据。

（4）前端统一存放api列表

前端将所有请求api放在目录Api文件下，前端入口文件统一引入，便于统一查看和更改。

代码统一形式

const BASE\_API = 'xxx'; //服务器的地址

const LoginApi = BASE\_API + '/login';

## 4.5后台合并控制逻辑层和业务逻辑层

在传统意义上JavaWeb后台分为控制逻辑层（Controller）、业务逻辑层（Service）、数据库操作层（Dao）。而控制逻辑层和业务逻辑层可以近似看作MVC设计风格里的Controller层，负责数据交互和页面转发。在早期的JavaWeb项目中Controller层需要处理复杂的逻辑，为降低系统层与层之间的耦合度、实现解耦的目的增设了业务逻辑层以共同实现数据交互和页面转发。

但随着JavaWeb框架的更迭，对数据库操作层（Dao）的封装越来越成熟，已具备处理一定数据逻辑的条件，致使在中小型项目中业务逻辑层几乎不会再额外处理数据，仅作为向控制逻辑层（Controller）传递数据的过度层；另一方面本框架Controller层已将页面转发权限下放到前端，只负责数据交互。同时，先进的前端Vue框架也可处理一定的数据逻辑，因而Controller实际上已无太多的逻辑处理。

本框架旨在降低层与层之间的依赖关系，将原控制逻辑层、业务逻辑层合并为控制逻辑层处理前端发来的请求。

## 4.6界面扁平化和数据可视化UI设计风格

本框架全局引入饿了么公司推出的ElementUI前端组件库，并以此为基础设计公有布局组件，公有布局组件见图4.3。本框架中建议基础组件均采用此组件库，以维持界面上的统一性。为方便数据的可视化展示，本框架引入适用于Vue的VChars组件库(基于百度公司的ECharts)，以上两个组件库均为扁平化设计风格，详见图4.4。在本框架设计和开发中，应避免再使用其他UI库致使设计风格上不统一。

## 4.7基于SeaJS的前端模块化设计

全局引入sea.js实现前端模块化开发，SeaJs实现并规定了Java scrIPt中模块定义、模块调用、模块启动的规范。Vue前端框架本身具有模块化的设计思想，引入sea.js主要实现了框架内原生Java scrIPt的模块化开发，与Vue组件设计思想上相一致。



图4.3 ElementUI组件设计的公有组件

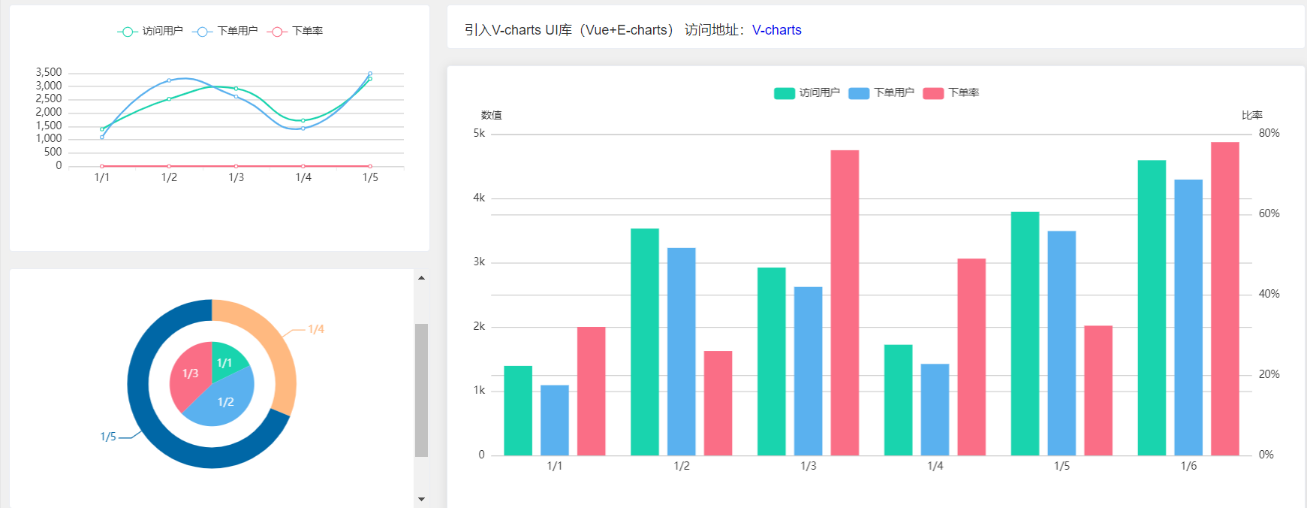


图4.4 基于Vcharts设计的Demo

## 4.8 框架搭建要点和难点

### 4.8.1 公共模块的引入

如前文所述，本框架摒弃了NodeJs平台，因而放弃了Nodejs提供的组件按需引入功能，只能采用全局引入，致使首次打开页面时较慢，这也是目前框架最不完善的功能之一。但深入思考，在单页面前端中组件全局引入一次都无需二次引入，亦不存在资源的浪费。在本框架index.html（入口文件）中，依次引入ElementUI、V-charts、SeaJs、httpVueLoader、Vuex、Vue Router等关键模块。

### 4.8.2 HttpVueloader转译Vue文件

在所有的Web项目中，浏览器作为客户端只会渲染加载html、jsp等文件类型，并不会识别渲染.Vue文件，上文也提到过本框架不再使用基于NodeJs的Vue Loader，转而使用HttpVueloader来转译Vue，其原理是依将Vue文件内的<template>、<script>、<style>标签转译为html、Js、CSS文件供浏览器识别渲染。

使用方法代码（Java script）：

var Login = httpVueLoader('./views/login/login.vue'); //将vue文件转译为“.html”文件

### 4.8.3 Vuex、Vue Router的实例化与配置

Vuex和Vue Router 通过在前文中讲到已通过全局引入，使用SeaJs规范将两者实例化方法定义为模块，在main.js模块中调用实例化方法，完成实例化。Vuex实例化过程中应将其必备的getter、action、mutaitions等属性方法注入，Vuex原理图见图4.5，值得注意的使其数据管理是单向的。Vue Router实例化过程应添加404界面和Login登陆界面的静态路由。

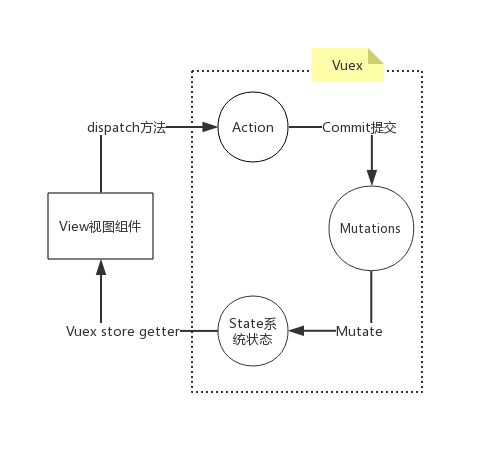


图4.5 Vuex管理系统状态图示

Vuex实例化核心代码（Java script）：

Vue.use(Vuex)

const store = new Vuex.Store({

modules: {

"user": user //app、user模块内封装action、mutaitions、state等改变系统状态的方法

},

"getters": getters

})

module.exports = store;

Vue Router实例化核心代码（Java script）：

module.exports = new VueRouter({

scrollBehavior: function () {

return ({y: 0})

},

routes: constantRouterMap //将404界面和Login界面添加静态路由

})

### 4.8.4 导航守卫、动态路由、登陆状态

导航守卫便是我们常讲的“路由钩子”，其原理是监控路由变化（导航）周期内的控制跳转（守卫导航）。本框架采用前置守卫，即在路由跳转请求后、路由实际跳转前予以控制，包括检测登陆状态、获取动态路由等功能。

在每一次的路由跳转中，首先检测前端Cookie缓存是否有数据（即前端是否是登陆状态），若没有数据（前端未登录状态）导航到登陆界面提醒登录；若Cookie中有数据，检测Vuex中是存有路由表、菜单表，若存在允许此次路由跳转，若不存在通过Dispatch方法调用Vuex Action中的getInfo方法，通过当前Cookie中的sessionId获取后台登录信息、用户对应权限的Menu列表和Router列表，若获取成功，路由中动态添加路由表，继续执行此次路由跳转，反之，即后台没有登录信息（登陆超时），导航到登录界面提醒登录，详见图4.6流程图。需要注意的是，在第一次登录系统后，向首页跳转的过程中也会调用此前置守卫，获取菜单表、路由表等信息，完成系统的初始化。

通过上述过程，实现了根据用户权限的不同赋予不同的路由表、检测用户登陆状态、控制路由跳转等大型项目必备的功能。

前置守卫核心代码（Java script）：

router.beforeEach(function (to, from, next) {

var token = getToken();//获取登录的token

if(to.path==='/login'){ //跳转到登陆界面的路由不予拦截

next();

}else{

if(token.length!=0){ //前端有登陆状态时

if(store.getters.menuList.length==0){ //前端菜单表为空时

console.log("---重新拉取用户数据---");

store.dispatch('GetInfo').then(function(res){ //拉取用户信息

router.addRoutes(store.getters.routerList);//动态添加路由

next({path:to.path});

}).catch(function(err){ //获取后台数据失败（后台超时）

next({path:'login?timeout'});

})

}else{ //前端菜单表不为空时。继续跳转

next();

}

}else{ //没有登录状态，跳转login

if(to.path==='/login'){

next();

}else{

var path = '/login?redirect=' + to.path;

next(path);

}

}

}

});

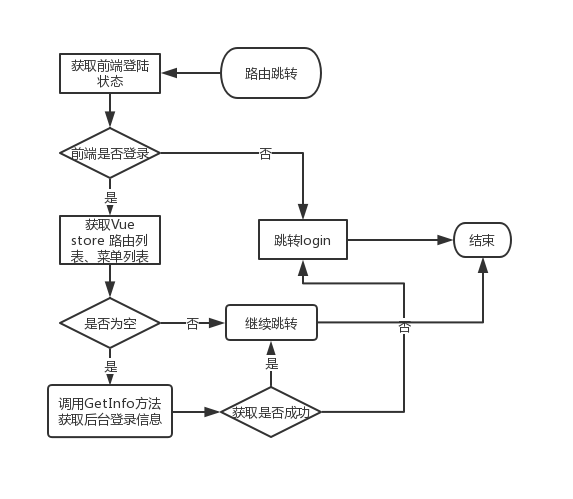


图4.6 前置导航守卫的流程图

### 4.8.5 Axios配置与跨域访问

如前文所述，Axios已通过全局引入，只需要在Vue中实例化。但与后台的交互中，出现了跨域访问的问题，本框架采用限定Axios请求头的格式、限定后台接收的格式来解决跨域请求、设置后台跨域请求过滤器来实现，详情见核心代码。

Axios前端实例化核心代码（Java script）：

axios.defaults.headers.post['Content-Type'] = 'text/plain'; //设置Axios请求数据的格式

Vue.prototype.$axios = axios; //在Vue中实例化

后台接收类型（Java）：

@RequestBody Map map //统一接收

后台设置跨域请求过滤器（Java）：

registry.addMapping("/\*\*").allowCredentials(true).allowedHeaders("\*").allowedOrigins("\*")

.allowedMethods("\*"); //设置跨域请求过滤器

经过上述处理以及前文中对请求前后端的统一规范后，前后端已经可以正常交互了，但前端若请求其他api（例如调用有道翻译的api）跨域请求，没法解决，毕竟本解决方案是通过前后端共同配置实现的。故其他第三方请求可通过引入基于JQuery的Ajax来实现，除此以外，此框架不适宜也不应该再引入JQuery（导致前端Java script框架的杂糅）。

### 4.8.6 前后端分离导致的Session问题

在本框架中，前后端分离采用Axios前后端交互，在这个过程中，每次前端的Request请求的头数据并非一致，这直接导致了本框架无法通过的传统的Request请求创建Session。因此本框架采用设置Session监听器的形式接管原Session机制，统一管理Session的id，前端可通过SessionId获取当前Session状态以实现对登录状态的监控。在SpringBoot框架中，在配置监听器、过滤器拦截器等的操作中无需繁琐的配置xml过程，仅需要添加相应的注解。Session监听器接管session如图4.7。

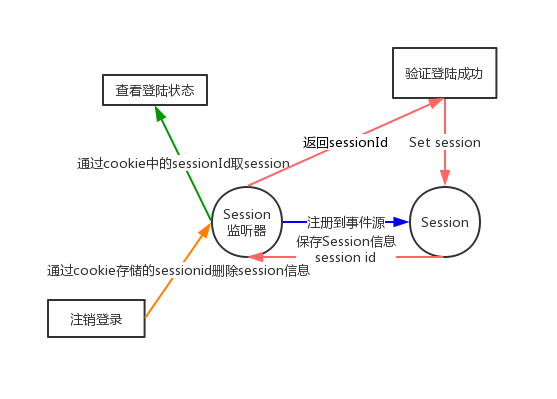


图4.7 Session监听器接管Session

### 4.8.7 后台SpringBoot+SpringDatajpa框架的搭建

在Idea集成环境下，通过Maven可以傻瓜式的搭建SpringBoot项目，在Maven Pom.xml配置文件引入spring-boot-starter-data-jpa SpringDataJpa依赖和对应的数据库依赖、在application.yml(或者对应的xml配置文件)配置数据库连接基本可以搭建完成。添加spring-boot-devtools 依赖可实现后台的热编译提高开发效率。

# 5 系统详细设计及实现

## 5.1 数据获取部分

### 5.1.1 Scrapy框架搭建及应用

Scrapy框架是一套集成网页请求、网页下载、网页解析、数据存储完整的体系化爬虫框架。Scrapy框架搭建极为简单，仅需要配置Python依赖，使用其集成的 Scrapy startproject 命令行工具即可。

### 5.1.2 数据获取详细设计

（1）数据源的选择

对于微博数据的获取，已知的可获取数据的数据源有weibo.cn（3g手机版）、m.weibo.cn（移动端）、weibo.com（PC端）三个[5]。接下来我将从微博个人信息页面DOM树结构上分析三个数据源的获取难度。 weibo.cn个人信息Dom结构如图5.1所示，获取微博用户的个人信息仅需要获取 class=”c” 标签中数据、通过正则表达分别匹配昵称、认证、性别、地区等即可获取完整的个人用户信息； m.weibo.cn个人信息Dom树结构如图5.2所示，要获取个人信息也较简单，从 class=”m-box”中获取即可，但我们仅可获得用户昵称、微博认证而无法获取用户详细资料，因而不采用此数据源；weibo.com个人信息Dom树结构如图5.3所示，很明显我们可以获取到用户的所有信息，但每个个人信息节点都在不同的标签下，我们需要对每个标签获取之后使用正则表达式匹配,开发量较大，另一方面PC端的数据源使用了大量的前端框架之时Dom层级过深，这使得页面下载和数据获取效率变低。三个数据源其他微博页面也存在上述差异，因此为方便开发，本系统采用weibo.cn作为数据源。



图5.1 weibo.cn个人信息Dom树结构



图5.2 m.weibo.cn个人信息Dom树结构



图5.3 weibo.com个人信息Dom结构

（2）url的确定

要想全面获取微博数据，必须从单个用户的信息着手做切入点，获取其个人信息、微博及其评论数据，之后从该用户与其他用户关系中获取下一个用户相关信息，这也是爬虫（Spider）最基本思想，以种子url做起点不断获取下一个url数据，以点结面，最终形成详尽、全面、复杂的数据关系网。在选定weibo.cn数据源后，不难发现我们仅需知道种子用户的id（例1042670620），就可以获取其用户资料（weibo.cn/1042670620/info）、微博信息（weibo.cn/1042670620/profile）、用户关注（weibo.cn/1042670620/follow）、用户粉丝（weibo.cn/1042670620/fans）的url，用户关注、用户粉丝中获取大量用户id，即可作为下一次数据获取的起点。

（3）数据获取主流程示意图

在确定数据源和url后，基于用户关注的对象往往是具有较大影响力的用户，本系统爬虫的下一节点为关注列表里的用户，并依照如图5.4数据获取流程示意图设计爬虫。

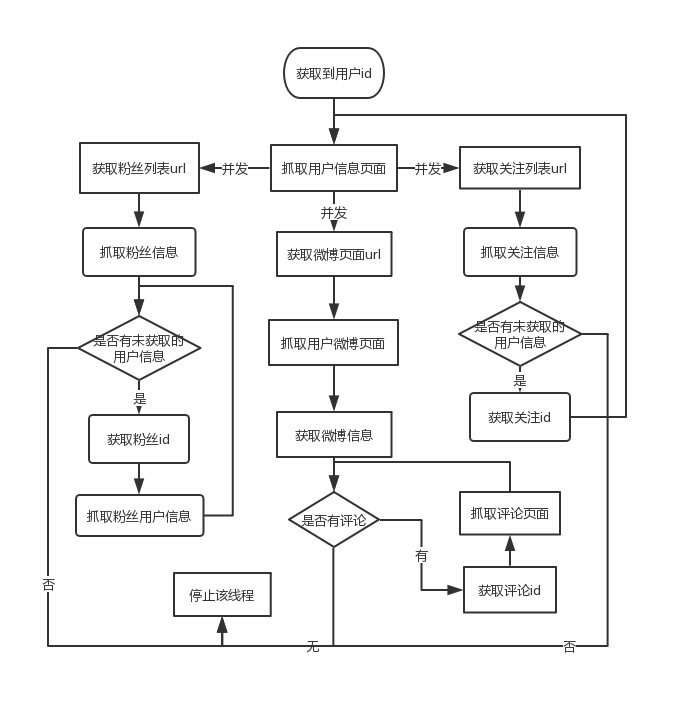


图5.4数据获取流程示意图

（4）数据解析

Scrapy框架官方提供了CSS选择器、xpath属性提取、正则表达式三种数据提取工具，在本系统设计基于weibo.cn Dom结构的特性，使用正则表达式、xpath属性提取为主的三种方式混用的提取方法。

正则表达式提取示例代码（Python）：

nick\_name = re.findall('昵称;?[：:]?(.\*?);', text1) #提取用户昵称

gender = re.findall('性别;?[：:]?(.\*?);', text1) #提取用户性别

xpath属性提取示例代码（Python）：

like\_num = tweet\_node.xpath('.//a[contains(text(),"赞[")]/text()')[-1] #xpath获取资源

tweet\_item['like\_num'] = int(re.search('\d+', like\_num).group()) #正则表达式简易获取点赞数量

repost\_num = tweet\_node.xpath('.//a[contains(text(),"转发[")]/text()')[-1] #xpath获取资源

tweet\_item['repost\_num'] = int(re.search('\d+', repost\_num).group()) #正则表达式简易获取转发数量

（5）数据存储

Scrapy框架官方提供了pipelines组件来快速保存Item（与实体相似），但在本系统中未使用该组件。主要原因有以下几点：在Python组件中基于MySQL的ORM框架SQLAlchemy对Scapy的pipelines组件不太友好，pipeline的最小单位Item和SQLAlchemy的实体并非完全一致需要一定的转换；使用ORM框架爬虫每获取一个Item便需要保存一个，本身ORM框架效率低于使用原生SQL，并且不能在Scrapy框架中不能实现多次执行SQL一次提交。为提高程序运行效率（一定程度牺牲开发效率），本系统使用基于pymySQL数据库驱动的原生SQL来保存数据。

### 5.1.3 与微博反爬虫机制的对抗

在任何大型的爬虫开发项目中，都充斥着爬虫、反爬虫、反反爬虫之间的较量。反爬虫即大型平台对爬虫、大量请求的涌入的反制机制。反反爬虫即爬虫开发者应对爬虫数据源反爬虫机制的方法。在本系统中，微博也具有一定的反爬虫机制，但可能作为一个开放平台，其反爬虫机制并不严格。

经过分析研究，发现新浪微博平台发爬虫机制（本系统以3g版的weibo.cn为基础做研究，其他数据源反爬虫机制或有差异）主要通过判断同一个IP地址同一个账号短时间内是否多次请求微博页面来实现，如果该IP访问过于频繁，微博平台会暂时封禁该微博账号，致使Cookie失效；或暂时封禁该IP（一般不会出现，出现停用该IP 3~5分钟自动解除）。

依据上述同时监控IP、账号以及封帐号不封（彻底）IP的反爬虫特性，本系统采用构建大量账号池的方式解决。Scrapy框架提供middlewares中间件组件，可以对每次的Request请求、每次Response返回进行处理。在本系统中，构建大量微博账号Cookie池，在每次Request请求中，从Cookie池中随机获取一个登录账号Cookie，模拟此账号请求下载微博页面，降低同一个账号的频率；在并发的请求中两次请求间隔设置为0.3秒，一方面降低IP访问频率，另一方面给数据处理、数据库I/O留足时间；在Respons返回结果后，若返回结果为418（微博平台对IP封禁返回码），程序暂时停止3分钟，不再进行任何请求。本系统反反爬虫方法详见图5.5示意图。

当前微博平台用户登录均需要进行人机验证，所以在本系统中构建账号池只能通过浏览器登录在开发者模式的NetWork视窗中获取，而无法使用Python模拟登陆。无法批量快速构建账号池，这也是本系统的一大遗憾。

Request中实现核心代码（Python）：

self.cursor.execute(

"SELECT \* FROM account\_pool WHERE status='success' ORDER BY rand() LIMIT 1") #随机获取

random\_account = self.cursor.fetchall()

request.headers.setdefault('Cookie', str(random\_account[0][3])) #设置cookie

Response中核心代码（Python）：

if http\_code == 302 or http\_code == 403: #账户cookie不可用时

#将账号池中此账号置为不可用，代码省略   
return request #重新请求该页面

elif http\_code == 418: #IP被封情况  
time.sleep(180) #延迟三分钟  
return request #重新请求该页面

else: #正常时返回  
return response

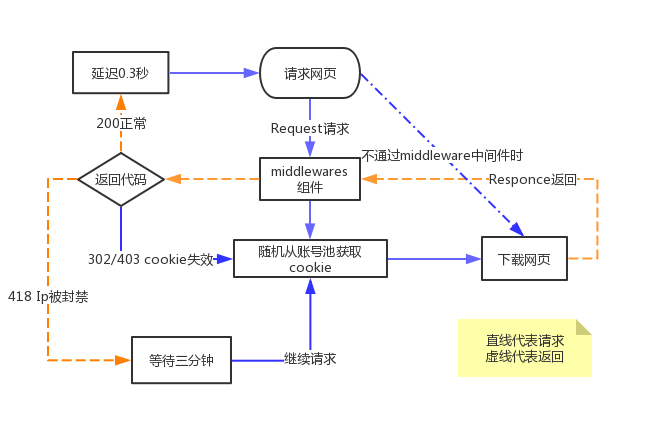


图5.5基于middlewares和账号池的反反爬虫示意图

### 5.1.4 数据获取广度与深度

在数据获取的过程中，数据的广度和深度永远是不会兼得的，权衡好获取数据的广度和深度尤为重要。基于个人电脑性能的局限性和网络带宽的限制，本系统对获取数据获取的深度做了较大的限制，主要从评论的深度、微博的深度、时间深度三个方面予以限制。评论深度上设置最多获取前5页数据（70-80条），对于大V来言，前5页数据几乎就是我们所说的热评，基本可代表多数网友的观点，具有较大的认同感。对于一般用户几乎全量获取评论，每个用户都具有可研究性；时间深度设置为仅获取2017.1.1到现在，基于现阶段的要求和设备的局限性，研究2017年以后的数据具有较强的可行性；微博深度设置为进获取前80页（800条），这是个宽泛的限制，具有较高影响力的大V一般情况下发表微博的频率不会太大，在时间深度确定的900天内的最多获取800条数据完全合理。

在本系统数据获取的过程中为确保数据的广泛性，将数据获取为三个阶段。第一阶段，通过顶级大V “人民日报”做种子url获取点数据，上述两个用户关注的用户也具有较大的影响力。第二阶段从上述用户关注列表随机选取10个用户id，获取所有数据，再从新的关注列表获取新的10个用户id……如此往复，以点成面获取一批具有高度影响力的大V用户集。第三阶段，从粉丝列表获取大量id，进获取这些粉丝的用户数据，进一步扩大用户数据的广度，覆盖更大、更具研究价值的面。

### 5.1.5 多线程与分布式爬虫的设想

Scrapy框架本身是支持多线程的，在本系统也应用到多线程，基于个人PC搭载Core I5-5200U 双核四线程处理器硬件基础，开辟4个线程获取数据。但基于本系统实际测验，4个线程比单个线程获取数据提升并不明显，基于最近的研究得出推测出以下几点可能原因：Python这种解释性语言对多线程支持不太友好；个人电脑对数据的I/O存储限制整体获取的性能；网络带宽限制网页下载速度，而Scrapy中的多线程是创建多个请求多个网页下载。以上设想均为得到想过论证，需要进一步研究。

基于Scrapy的分布式爬虫设计思想、案例屡见不鲜，可参考的项目较多，在本系统中建立更大的Cookie账号池（或许还需要构建IP代理池），建立Redis服务器，修改相应的配置设置文件即可完成分布式改造[6]。在本系统仅作此加快数据获取分布式构造的设想未作具体实现。

## 5.2 数据处理部分

### 5.2.1 数据处理实现及流程

数据处理是将已获取的微博信息和评论信息进一步加工处理，从中提取有价值的信息，删除无意义信息。在本系统中，主要在微博信息和评论数据中提取话题信息、存在表情信息、通过自有词库断词获取词频、训练优化词库，不断在未处理的微博信息、品论信息中获取数据，一条条的处理数据，处理数据的流程图见图5.6。

### 5.2.2 最大正向匹配+最大正向查询断词算法

提及断词算法，主流思想有基于字符串与词库匹配的分词思想、基于统计的分词思想、基于规则的分词思想。各种思想各有优劣，总体上基于统计思想的分词算法最为准确（Jieba分词应用此思想）。但基于本系统要统计词频（建词库），且要求快速断词，基于此，本系统采用自建词库、基于字符串匹配方式实现断词。由于个人PC设备的I/O性能的局限性和数据量庞大性，本系统设计算法思想为尽可能的减少与数据库的I/O、尽可能的提高断词速度。

在本算法中，首先将词句根据标点符号断句，依次处理子句str。子句str游标curcer从0开始遍历字符，添加入temStr，temStr最大长度为2，通过temStr查询内存中临时词典、正向最大查询，从词库查询到“正向”含有temStr所有字词，例如我们要查询“变化”会得到“变化”、“变化莫测”、“变化无常”等，从查询到的结果中匹配str中游标开始的字词，匹配到结果集中最长的word，存入分词集合，游标curcer增加匹配词的长度，直至curcer等于str长度。期间询到的结果集都添加到临时字典中，以减少数据库的I/O，临时词库达到一定量后释放。具体执行见图5.7流程图，每一流程均是为了减少不必要的匹配，减少数据库查询次数。

正向最大查询SQL：

#temstr是要匹配的字词，isShow用来判断是否添加到返回结果集

SELECT word,is\_show FROM base\_words WHERE word LIKE '%s' " % (temstr + "%")

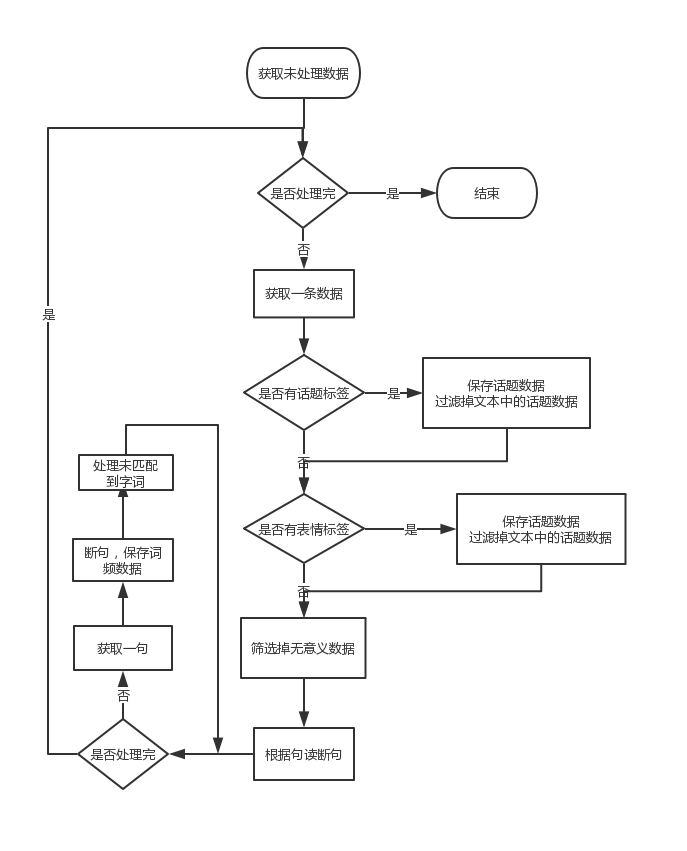


图5.6数据处理流程图

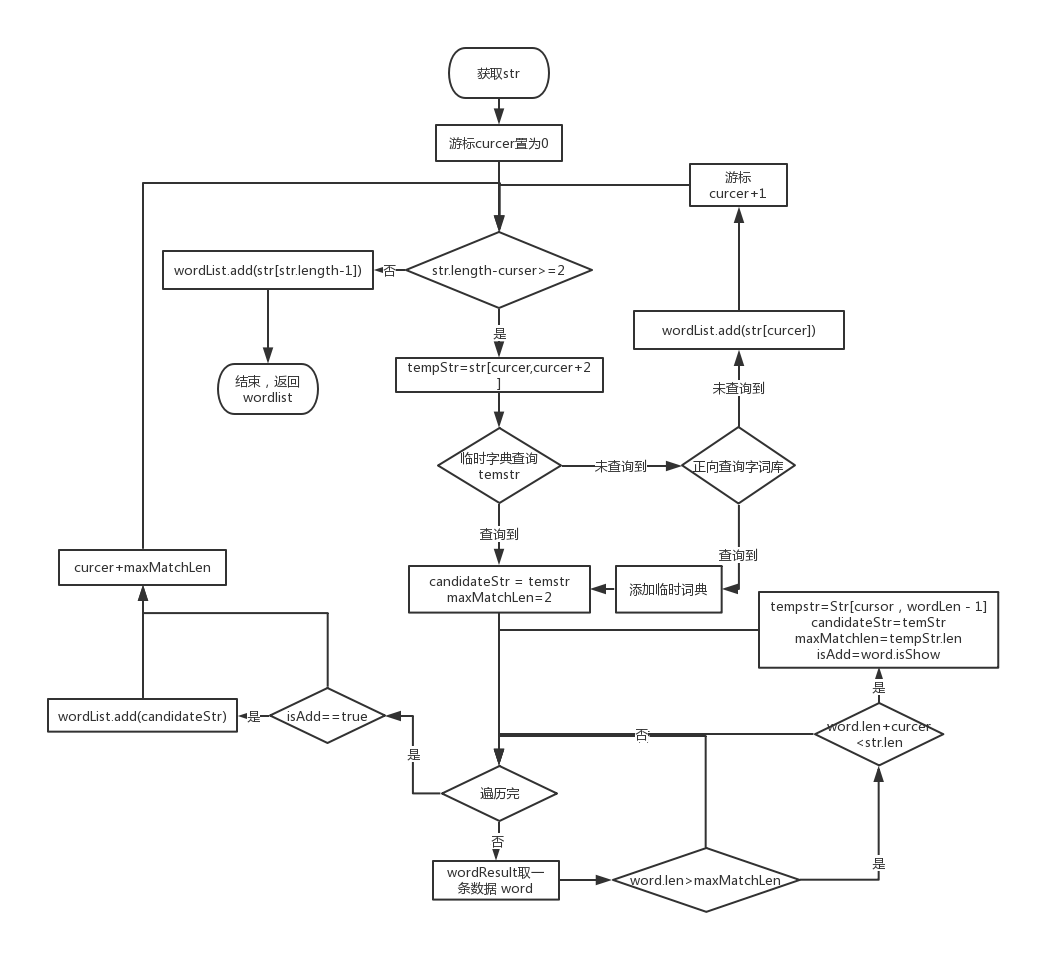


图5.7断词算法流程图

### 5.2.3 词库的自扩展

优秀的词库应当在基础词库基础上，通过不断的分析数据来不断扩展。本系统在数据处理断词过程中，不断对未匹配到的词尝试连结为新词，暂时存储在new\_word疑似新词表中，并记录出现的次数，当出现次数达到一定量时，将新词插入到base\_word基础词库中，通过此过程来不断扩充优化词库。

未被匹配到字词拼接新词核心代码（Python）：

for i in range(len(wordList)):  
 if (len(wordList[i]) == 1 and self.ishan(wordList[i])):  
 temStr += wordList[i]  
 if (len(wordList[i]) > 1):  
 if (len(temStr) > 1):  
 temStrList.append(temStr)  
 temStr = ""  
if (len(temStr) > 1):  
 temStrList.append(temStr)

新词超过一定次数（100）添加入基础词库存SQL：

INSERT INTO base\_words (word,counts,is\_show,type) (SELECT word,count,1 AS is\_show,'newWord' AS type FROM new\_words WHERE count>100);

### 5.2.4 Snownlp简介

Snownlp是基于Python优秀的中文自然语言处理库，其内涵了中文断词、中文拼音、中文情感分析等模块。由于Snownlp断词词库的局限性和低速性，在本系统中没有使用其中文断词模块，而采用前文叙述的断词算法。在本系统中使用了Snownlp的情感分析模块来分析微博数据和评论数据的情感。Snownlp情感分析算法是基于贝叶斯模型训练实现的，通过对大量积极/消极词句训练，使用贝叶斯模型发现记录特征词[7]。分析语句中的特征词，取其加权平均值即为此语段的情感态度。

## 5.3 数据分析、展示、预测部分

### 5.3.1 登录、登出

本文已在4.8.4导航守卫、动态路由、登陆状态，4.8.6前后端分离导致的Session问题章节中详细阐述实现原理，在此不再详细赘述。在登陆入口、权限上设置有管理员和访客登录。管理员拥有系统配置和数据展示的所有信息，而访客仅拥有查看系统数据的权限。值得注意的是本系统中对访客登录中记录了包括IP、操作系统、浏览器版本等信息，用于后期访客信息的统计。管理员权限与访客权限对比见图5.8、5.9。

### 5.3.2 系统数据的总览

系统数据总览即本系统的首页，主要向访客和管理员展示系统的整体运行状态，展示内容主要包含微博数据用户获取总人数、获取微博总数、获取评论总数、数据处理量、数据待处理量、处理进度、账号池利用率、近10天的获取用户微博评论的日曲线图等功能，如图5.10、5.11所示。在此模块设计中大多采用数据库直接查询方式实现，应用SpringdataJpa框架对Count查询封装方法即可获取数据，在日曲线的数据依次获取最近十天每天的获取量来实现

代码示例（Java）：

userSum=weiboUserInfoReposity.count(); //微博总获取用户

weiboSum=weiboInfoRepository.count(); //微博总获取数

userDateSum=weiboInfoRepository.countByCrawlTimeBetween((int)datet1,(int)datet2) //两时间内获取数量

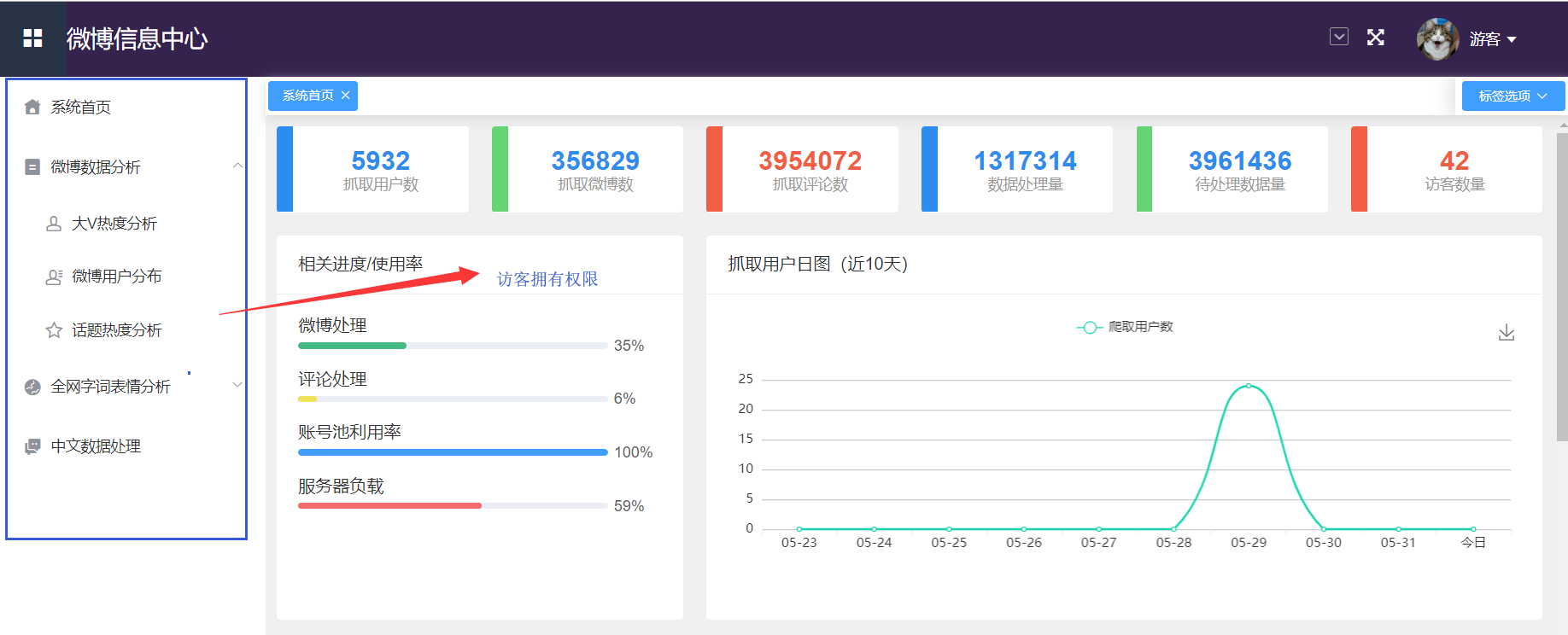


图5.8访客权限



图5.9管理员权限



图5.10 系统首页功能1



图5.11 系统首页功能2

### 5.3.3 大V热度及其人物画像分析

在此模块中，基于现有已获取的数据对大V(拥有大量粉丝)用户进行多维度的分析，主要包括一段时间内的热度排行、查询个人详细资料、人物画像、粉丝地区年龄性别分布、大V塑造形象与粉丝反馈情况对比、大Ｖ热度变化趋势及预测等功能。

1. 热度排行

在新浪微博平台上，毋庸置疑，粉丝（包括黑粉）与微博用户间的互动程度最能体现一个用户的热度，而用户发表的微博恰是该用户与其粉丝互动的桥梁。因此在本系统中用一段时间内的用户微博及其粉丝互动的程度反应用户热度。每一条微博均有点赞、评论、转发三个衡量互动程度的维度。给上述三个维度合适的权值在本系统中计算用户热度尤为重要。

在一般认识下，点赞数一般反映粉丝对其的好感度，转发数反应粉丝的认同度，回复数（包括争议回复）一般反映大V的流量带动能力。粉丝对于自己关注者的微博本能的会转化为自己认同的一种价值观，对于这种价值观的认同程度也表现在用户点赞、回复、转发上。点赞对于大多数人来说可能是个顺手而为的事情，在本系统中认为，点赞数对于热度影响的权值应该是最小的，但基于上述三种行为对于热度而言均是正促进的，点赞的权值不宜小于1，本系统中暂且设定为1；要点击回复按钮输入回复内容进行评论这种粉丝行为相较点赞更为繁琐，并且回复数的多少直接反应与热度挂钩的大V流量带动能力，在这里我把权值相较点赞的权值上浮20%为1.2；而转发这种行为应当是最具主观性的行为，最能体现真爱粉的数量，本应把权值调高于回复，但根据大量微博数据研究发现，存在大量粉丝的多次转发，许多微博的转发量都远高于其应有表现，在此将其权值暂定为1.1，较点赞量上浮10%。因此可得在本系统中计算大V热度公式（h为热度、n为t1~t2时间段大V发表的微博数、l为微博点赞数、r为转发量、c为回复数）如公示5.1

|  |  |  |
| --- | --- | --- |
|  |  | (公式5.1) |

在本系统为加快查询速度，对大V的定义略微提高门槛，即粉丝量达到500万时，认定为大V，查询的排行限制查询15条，即前15名。具体查询实现思路为将用户表和微博信息表相关联，按照大V表id分组计算热度。

核心SQL语句（t1,t2确定时间范围）：

SELECT a.\_id,a.nick\_name,a.authentication,

AVG(b.repost\_num\*1.1+b.like\_num\*1.0+b.comment\_num\*1.2) AS num from weibo\_user\_info AS a

LEFT JOIN weibo\_info AS b on a.\_id=b.user\_id

WHERE a.\_id IN

(SELECT \_id FROM weibo\_user\_info WHERE fans\_num>5000000 )

AND b.created\_at BETWEEN :t1 AND :t2 GROUP BY a.\_id ORDER BY num DESC LIMIT 15

具体热度排行界面见图5.13。在本系统中也添加了自定义搜索（模糊查询）获取热度排行。实现方式和上述整体热度排行类似，仅需将用户条件 fans\_num>5000000该换为nick\_name like ‘%name%’ 即可。实现界面如图5.12通过昵称关键词查询热度信息。

1. 个人资料与人物画像

在获取上述热度排行后，通过详情按钮可多维度查看大V画像，实现查看用户详细信息、大V词云、粉丝地区、年龄、性别分析，详细功能点见图5.14、5.15。词云是获取在本时间区段内微博、评论出现词的词频进行统计，根据词频绘制词云图，基于当前数据处理尚未处理完，此功能展示界面并非最终界面。粉丝的地区、性别年龄分布是根据，爬取的粉丝数据进行分析得出的。但由于微博权限问题，微博可获取的粉丝和关注者名单最多为70-80个，虽获取的这批粉丝具有广泛性和随机性，但由于样本容量过少，此功能的参考性有待商榷。

核心代码（Java）：

//获取t1-t2时间段内出现的所有热词

List<WeiboBaseData> weiboBaseDataList = weiboBaseDataRepository.findByUserIdOrUpUserAndDateBetween(id,id,t1,t2);

//热词统计数量核心代码

if(wordsCount.containsKey(weiboBaseData.getName())){  
wordsCount.put(weiboBaseData.getName(),(int)wordsCount.get(weiboBaseData.getName())+weiboBaseData.getCounts());  
}else{  
 wordsCount.put(weiboBaseData.getName(),weiboBaseData.getCounts());  
}

}

//获取所有粉丝

List<WeiboUserInfo> weiboUserInfos = weiboUserInfoReposity.findFansByFollowedId(id);

//对所有粉丝进行分析，统计数量代码类似于统计次词数代码

1. 大Ｖ塑造形象与粉丝反馈情况对比

前文中已经介绍到，在数据获取时我们已将微博数据、评论数据的情感状态一并获取了。这些情感状态预测有什么研究价值？大V发表微博的情感积极程度可以简单理解为该大V想要在公众中面前塑造的形象，而粉丝评论中的情感积极程度可以简单理解为粉丝对大V塑造形象的反馈，两条曲线趋势一致说明塑造形象比较成功，反之亦然；评论情感曲线两头高中间低说明人物争议比较大，反之则说明态度语言成中性，多是在陈述客观事实。在本系统内我们衡量情感积极程度范围为0~10，越接近0表示态度越消极，越接近1表示态度越积极。这两条曲线对比价值远不止这些，亟待继续发掘。实现功能如图5.16。

核心代码（Java）：

//统计出现次数 四舍五入

int []counts={0,0,0,0,0,0,0,0,0,0,0};

for(int i=0;i<weiboInfoList.size();i++){  
WeiboInfo weiboInfo = weiboInfoList.get(i);  
sum+=weiboInfo.getSentiments();  
int value = Math.round(weiboInfo.getSentiments());  
counts[value]++;

}

（4）大Ｖ热度变化趋势及预测

大V热度变化趋势，是指在一段时间内统计每天的热度值，按照每天的数据绘制变化曲线图。在实际计算中，常发现大V并不是每天都发微博的，假若某日大V没有发微博，在本系统中不认为大V在该日热度为0，相反认为此日大V的热度是昨日热度的延续，即与昨天热度保持一致（当然这也是一种模糊的认定方式，如果多日没有发表微博，无法展示其热度的趋势变化）。实现功能如图5.17所示。对于大V的热度预测具有较大的不可控性，因热度变化受太多人为因素的影响（比如蹭热度、热点大爆发），故而在本系统中仅使用一元线性回归模型对热度趋势进行简单预测。一元线性回归模型原理及推导详见附录B。



图5.12 大V热度排行条件查询



图5.13 大V热度排行

图5.14 用户详情及词云图

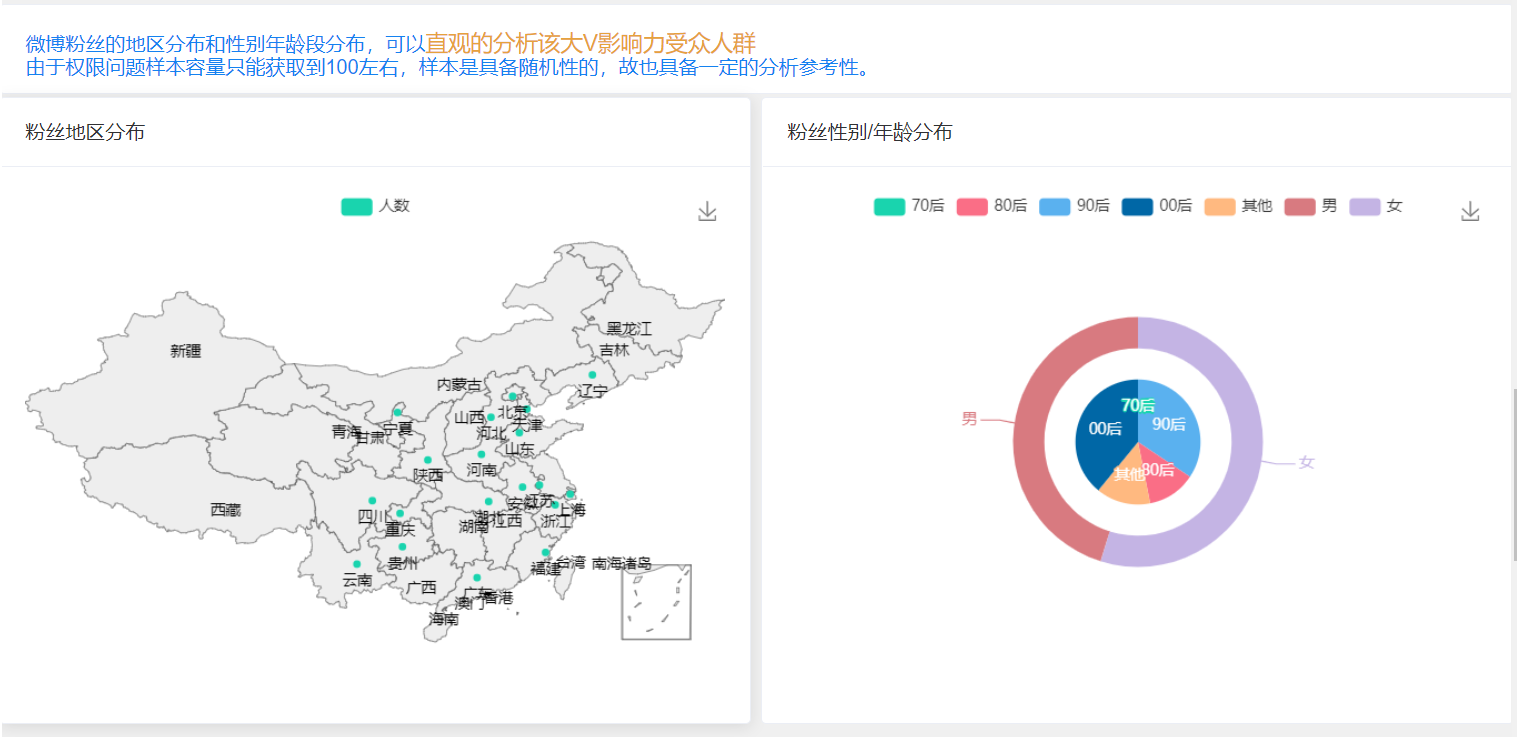


图5.15 用户粉丝地区、年龄、性别分布

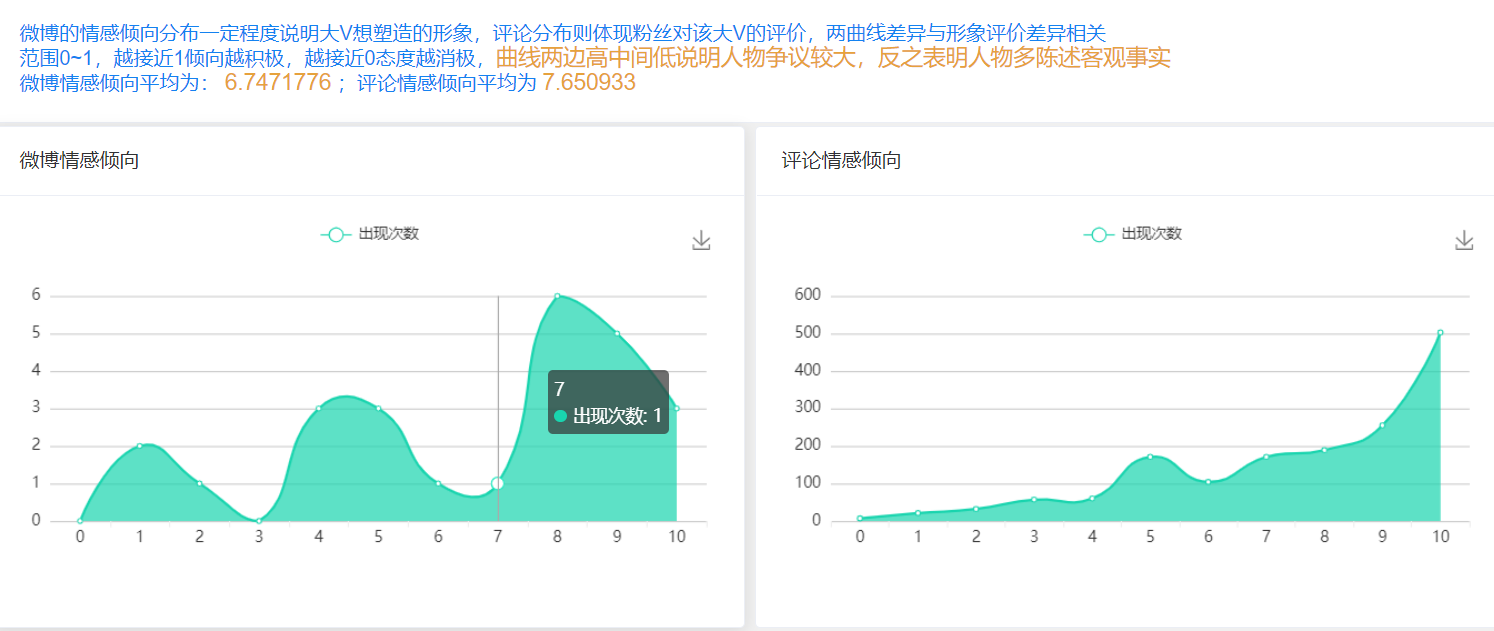


图5.16 大Ｖ塑造形象与粉丝反馈情况对比



图5.17 大Ｖ热度变化趋势

### 5.3.4 话题热度分析及预测

话题热度分析实现过程基本与用户热度分析一致，但由于前期建话题表时没有意识到话题和微博应该有一定的对应关系便于查询，当数据处理量超过百万后才意识到已为时已晚，致使当前获取数据时多加一步关联而效率变低。另一方面话题数据没有做单独的数据获取，仅能通过数据处理获取，截至目前未知，由于处理数据量较少，展示数据参考意义仍不大，功能界面见图5.18、5.19。

核心SQL语句（t1,t2确定时间范围）：

SELECT a.topic,a.counts,AVG(b.repost\_num\*1.1+b.like\_num\*1.0+b.comment\_num\*1.2) AS hotData from (SELECT topic ,count(\*) as counts FROM weibo\_topic WHERE date BETWEEN :t1 AND :t2 GROUP BY topic ORDER BY counts DESC LIMIT 10) AS a

LEFT JOIN weibo\_info AS b ON INSTR(b.content,a.topic)>0

WHERE b.created\_at BETWEEN :t1 AND :t2

GROUP BY a.topic ORDER BY hotData DESC



图5.18 话题热度分析

图5.19 话题热度变化趋势

### 5.3.5 微博用户分布

基于减少数据库的I/O、现有数据的局限性，本系统基于简单随机取样的统计学方法，以随机的方式从获取的用户数据中抽取3000个用户样本，整体系统获取用户数据具有随机性，对这些样本进行数据分析以概括整体数据特性。对信息进行简单的分析，详细代码在此不再展示。具体实现了对用户粉丝数量分布、年龄分布、性别分布、用户地区分布及比重、会员等级及其分布等功能，祥见图5.20-5.23。

核心SQL：

SELECT \* FROM weibo\_user\_info ORDER BY RAND() LIMIT 3000 //随机获取3000样本

### 5.3.6 全网热词、热门表情

全网热词/表情是指基于全网数据而言，在一定时间段内统计出现次数。便于研究同时获取热词、热门表情的相关用户，对于某热词在时间段内出现次数变化进行统计以分析并预测可能出现的趋势(主要用来研究某一热点对新词的影响)。在全网热词分析中加入了最近两年的年度热词，由于要查询一年的数据，查询数据量过于庞大严重影响系统的性能，在本系统中采用数据库事件思想，定期执行存储过程将更新临时表的方法来实现。本功能数据来自前文中的数据处理，目前并未完全处理，数据展示目前参考性不足。数据相关功能展示见图5.24、5.25（热门表情和热词功能几乎一致，在此仅展示热词界面图）。

核心SQL：

//获取 t1到t2时间段内的热词/表情排行

SELECT name,sum(counts) as counts from weibo\_base\_data

WHERE type=:type AND date BETWEEN :t1 AND :t2 GROUP BY name ORDER BY counts DESC

//通过以userId分组查询weibo\_data\_base 表userid counts对应关系与user\_info表关联获取昵称

SELECT a.user\_id,b.nick\_name,a.counts FROM

(SELECT user\_id ,sum(counts) AS counts FROM weibo\_base\_data WHERE date BETWEEN :t1 AND :t2 AND name=:name AND type=:type GROUP BY user\_id ORDER BY counts DESC LIMIT 10)

AS a LEFT JOIN weibo\_user\_info AS b ON a.user\_id=b.\_id

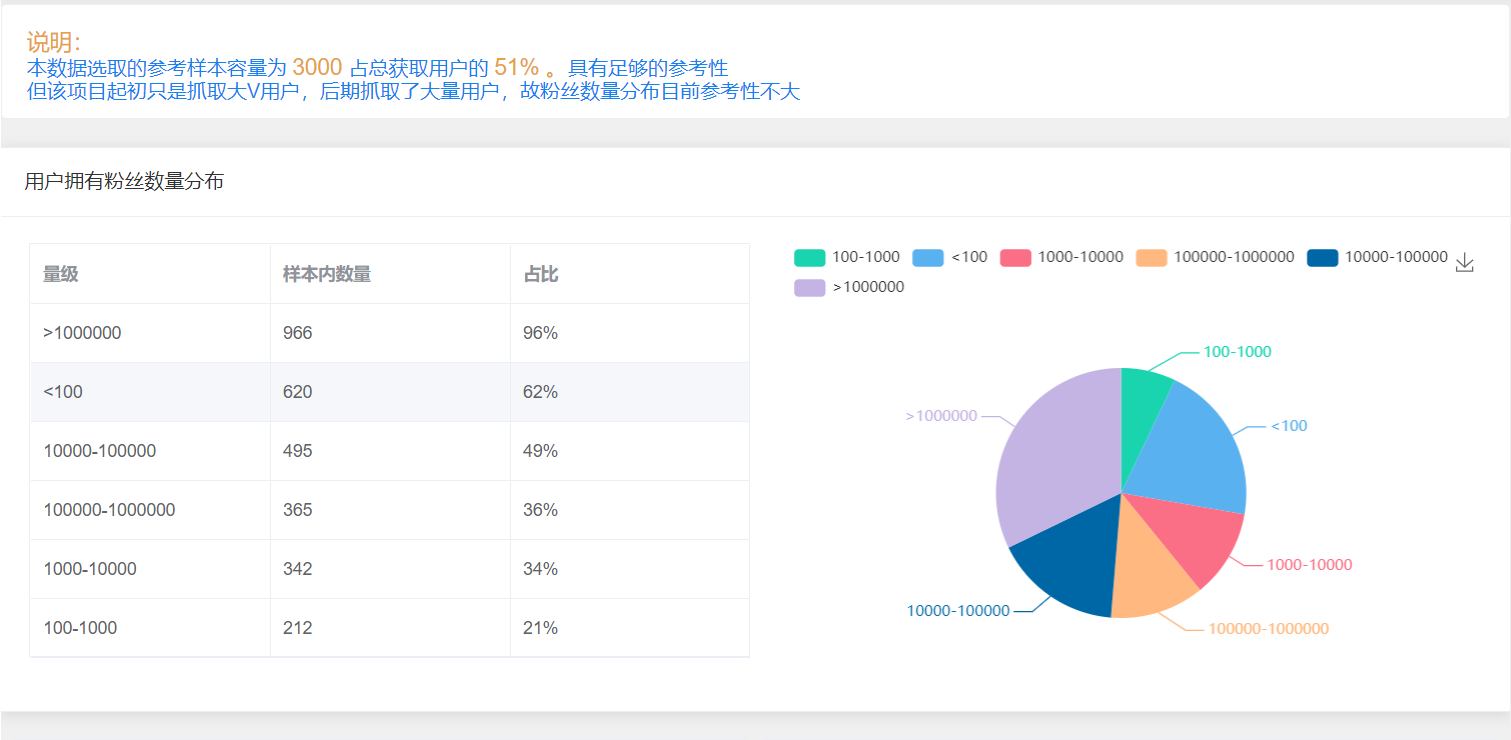


图5.20 用户粉丝数分布

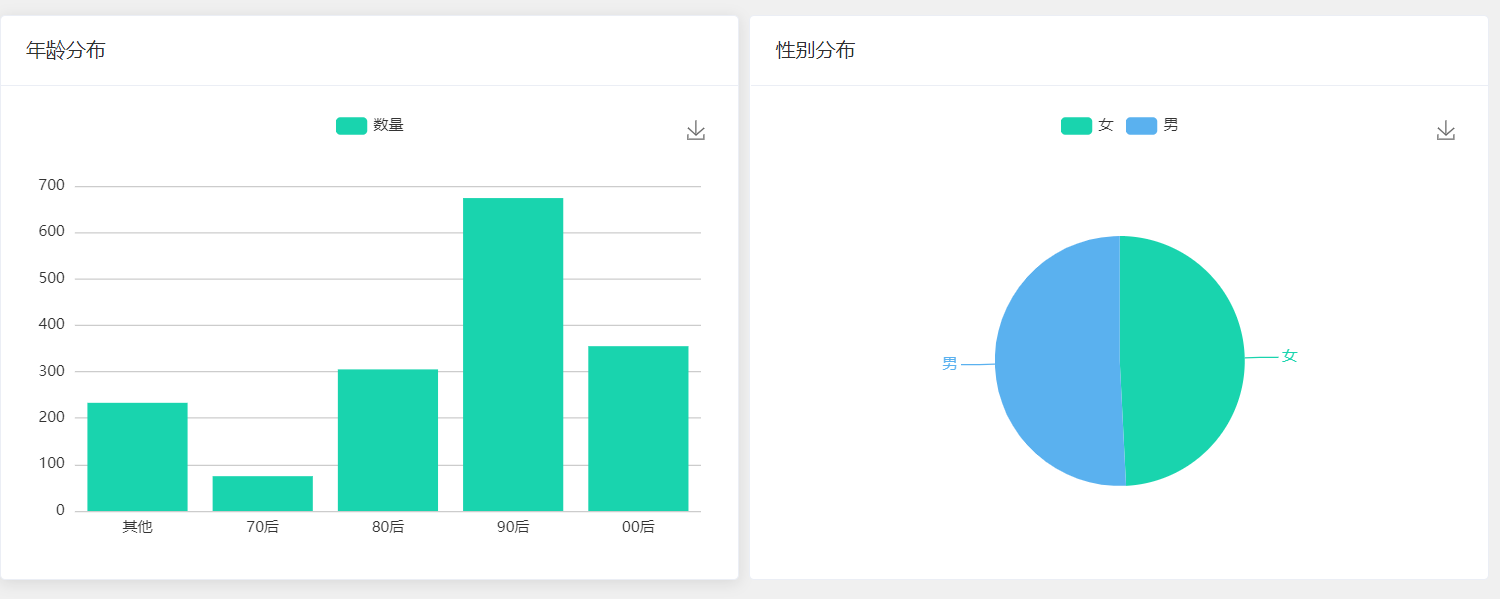


图5.21 用户年龄性别分布

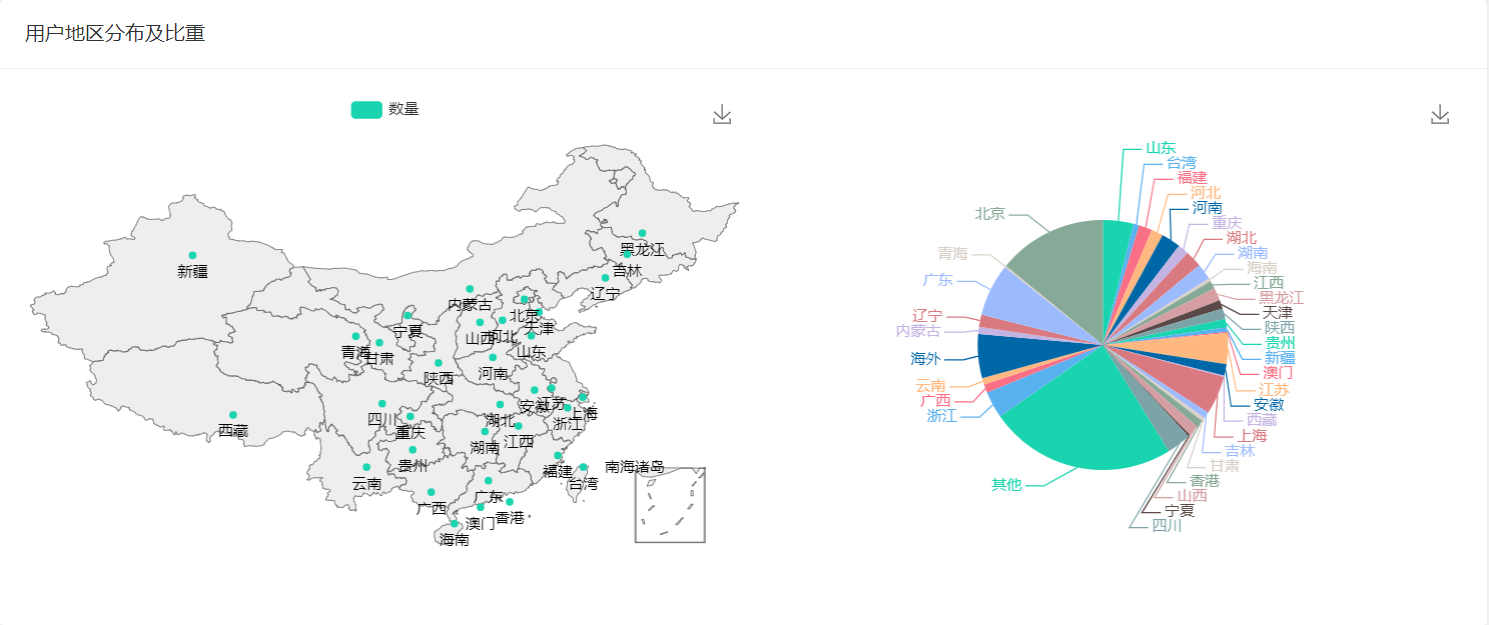


图5.22 用户地区及其比重分布

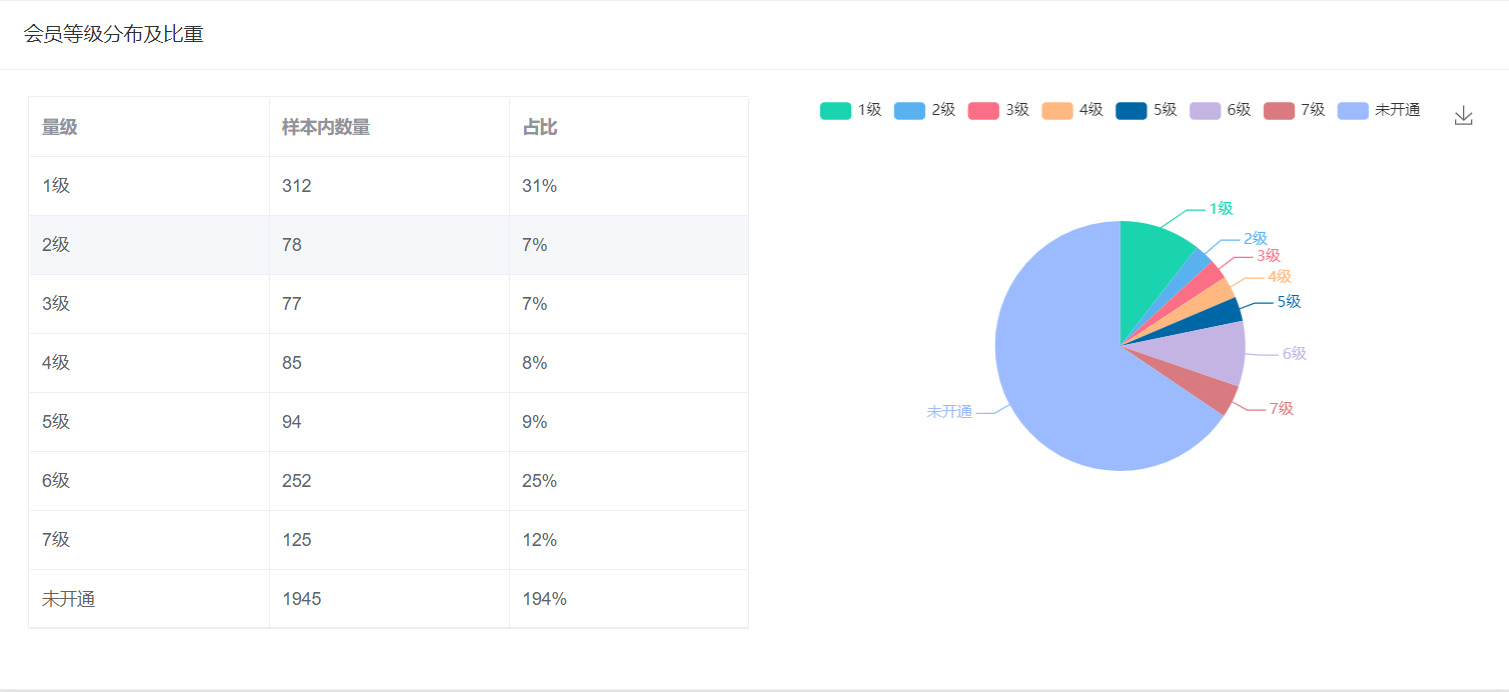


图5.23 用户地区及其比重分布

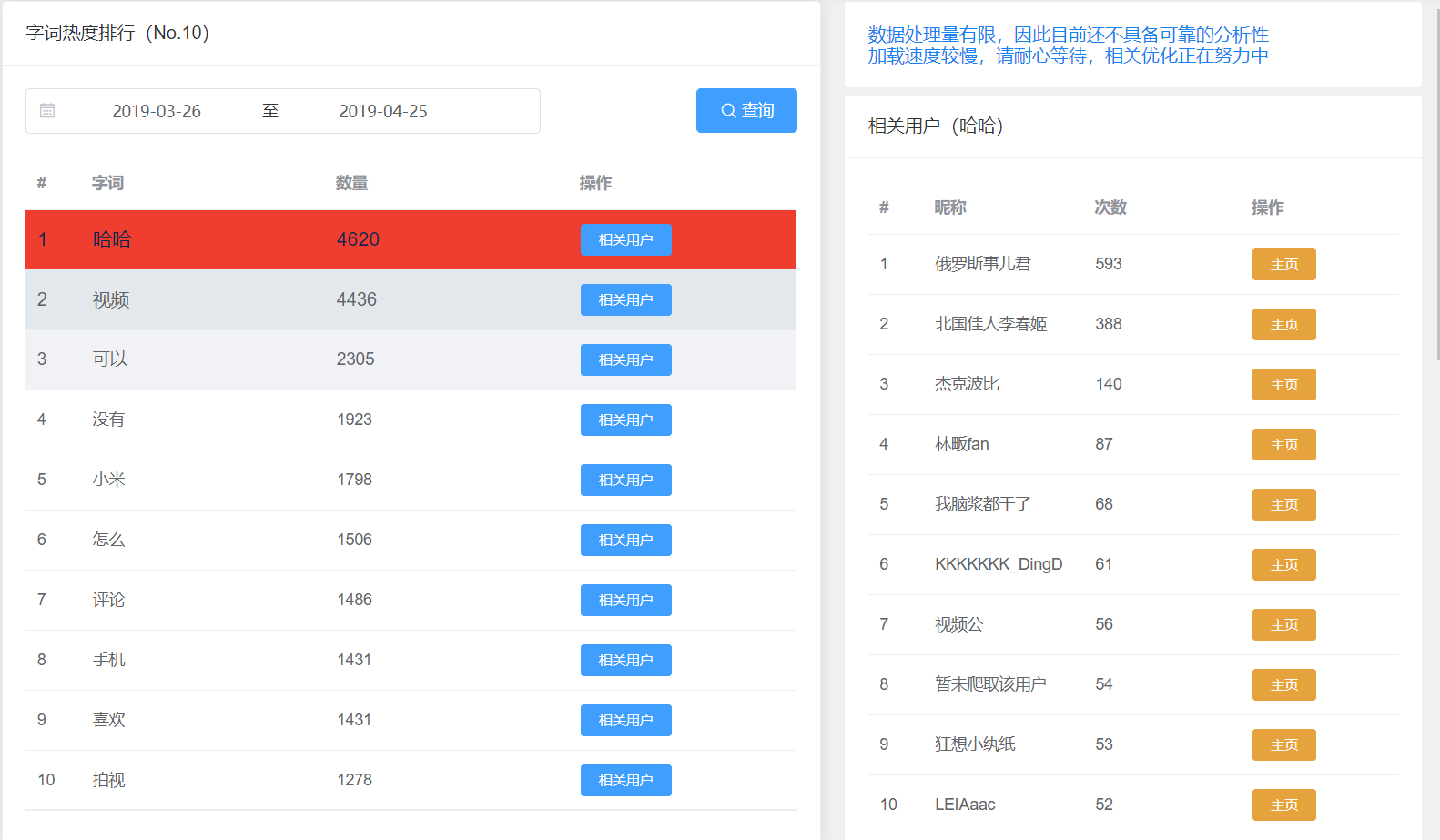


图5.24 字词热度排行1

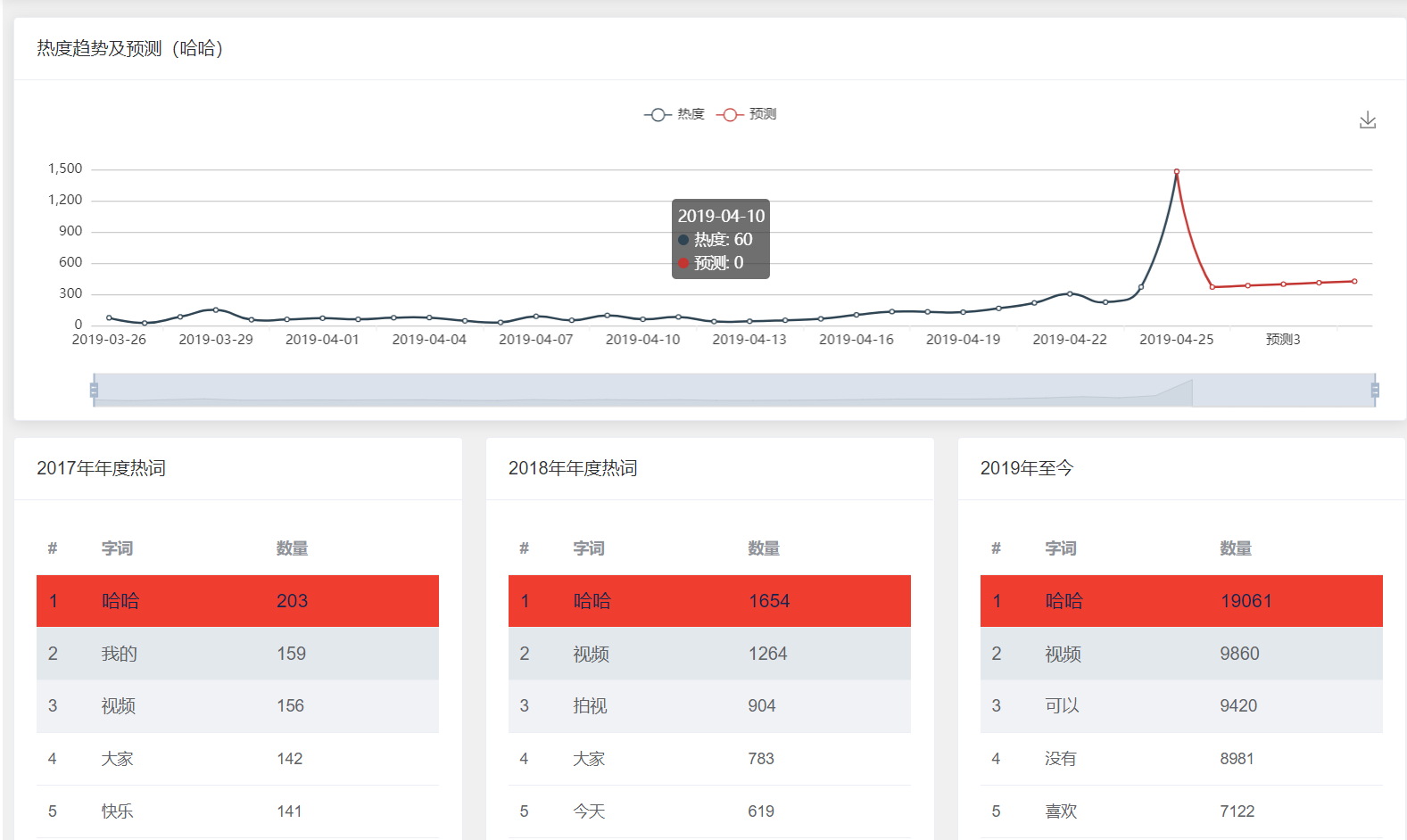


图5.25 字词热度排行2

### 5.3.7 中文数据处理

中文数据处理模块是本系统前端的第一个demo模块，设计初衷即是作为前端设计的基础、参照。在此模块中实现了分词、情感统计、词频统计等功能，知识点上几乎涉及了本系统Vue、Elements、Vcharts的所有核心用法，如v-bind数据绑定、v-if条件渲染、v-for列表渲染、v-class类绑定、el-tag标签、el-table表格、饼图、柱状图等，同样对本系统上述提及的Axios跨域请求问题做了解决——引入Ajax。翻译功能调用有道云api（Ajax请求），断词算法使用前文提及算法的Java实现版，具体实现功能见图5.26、5.27，在此章节中仅对分词界面呈现做代码说明，其他部分参照前文。

Tag列表渲染添加类动态改变状态核心代码（Vue）：

<el-tag :key="index" v-for="(word,index) in listWords" :class="{'tagSelected':word.selected}"

:disable-transitions="false" @click.native="handleTagClick(word)">{{word.word}}</el-tag>

<--!实现将所有断词 tag标签渲染出 点击（handleTagClick）时改变class状态 实现界面见图示分词部分--!>

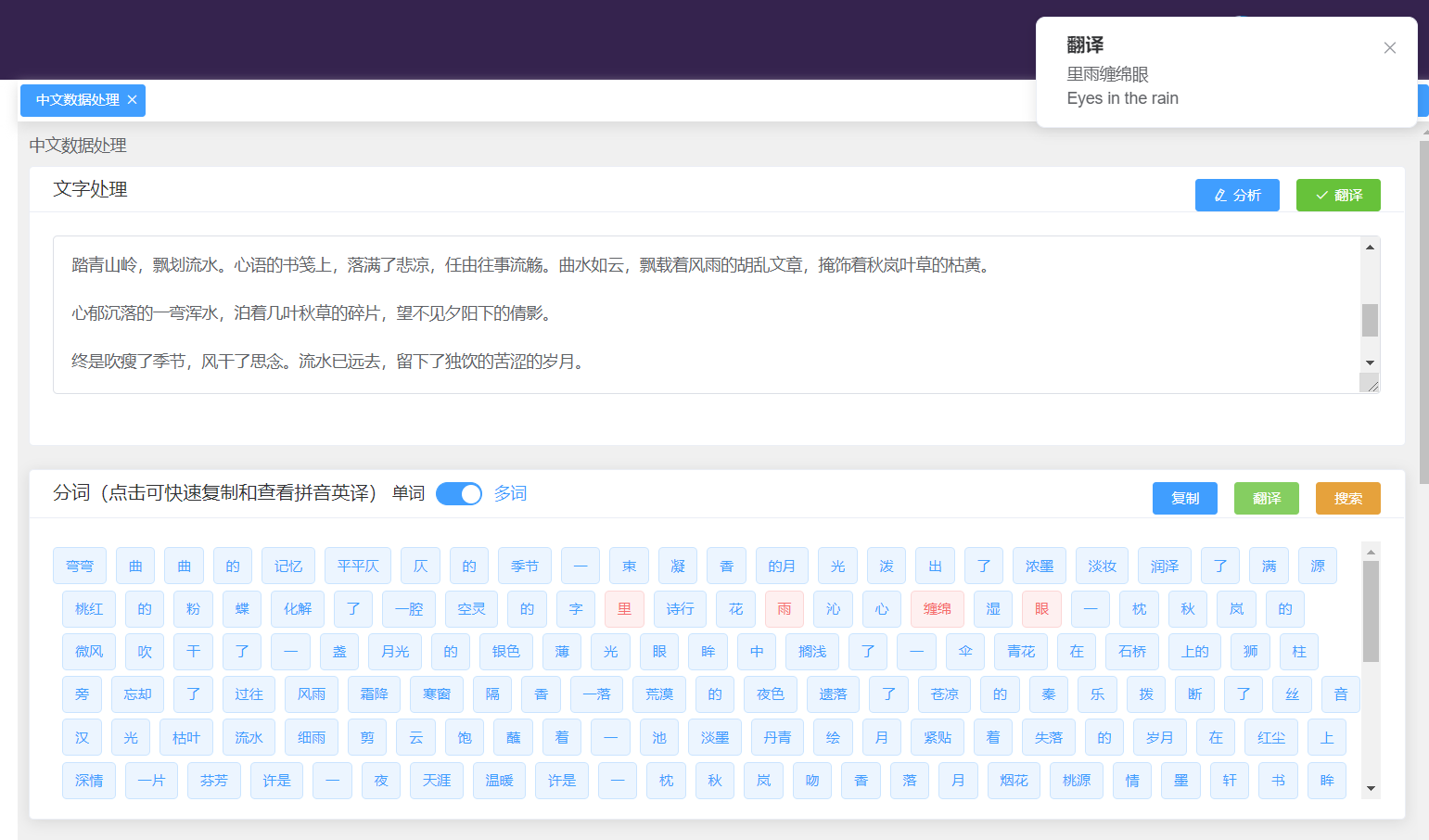


图5.26 中文数据处理1



图5.27 中文数据处理2

### 5.3.8 词库管理、系统消息、配置与调度

（1）词库管理

词库管理是对基础词库和疑似新词词库的管理。包括实现上传词库，导出词库（有问题），基础词查看、删除、置为不显示，新词库的导入删除功能。在此模块中应用到的所有方法均是对数据库的增删改查，均应用SpringDataJpa ORM框架封装好的方法，在此不再详细赘述，详细功能点参见图5.28、5.29。

SpringDataJpa分页排序查询示例（Java）：

Sort sort = new Sort(Sort.Direction.DESC,"counts"); //排序规则

Pageable pageable = new PageRequest(page, size,sort); //分页对象，page第几页、size每页数据规模

Page<BaseWords> pageList = baseWordsRespository.findAll(pageable);

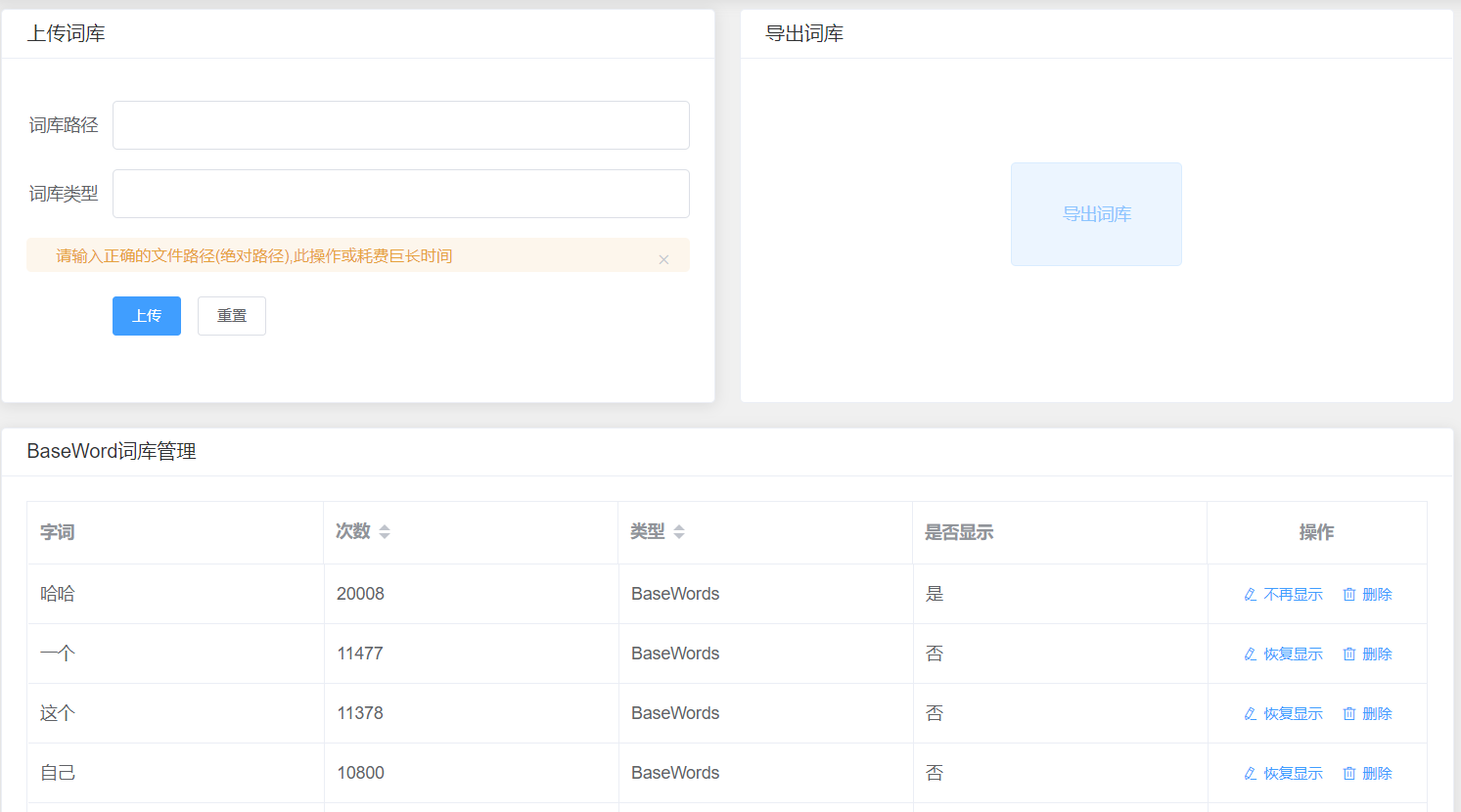


图5.28 词库管理1

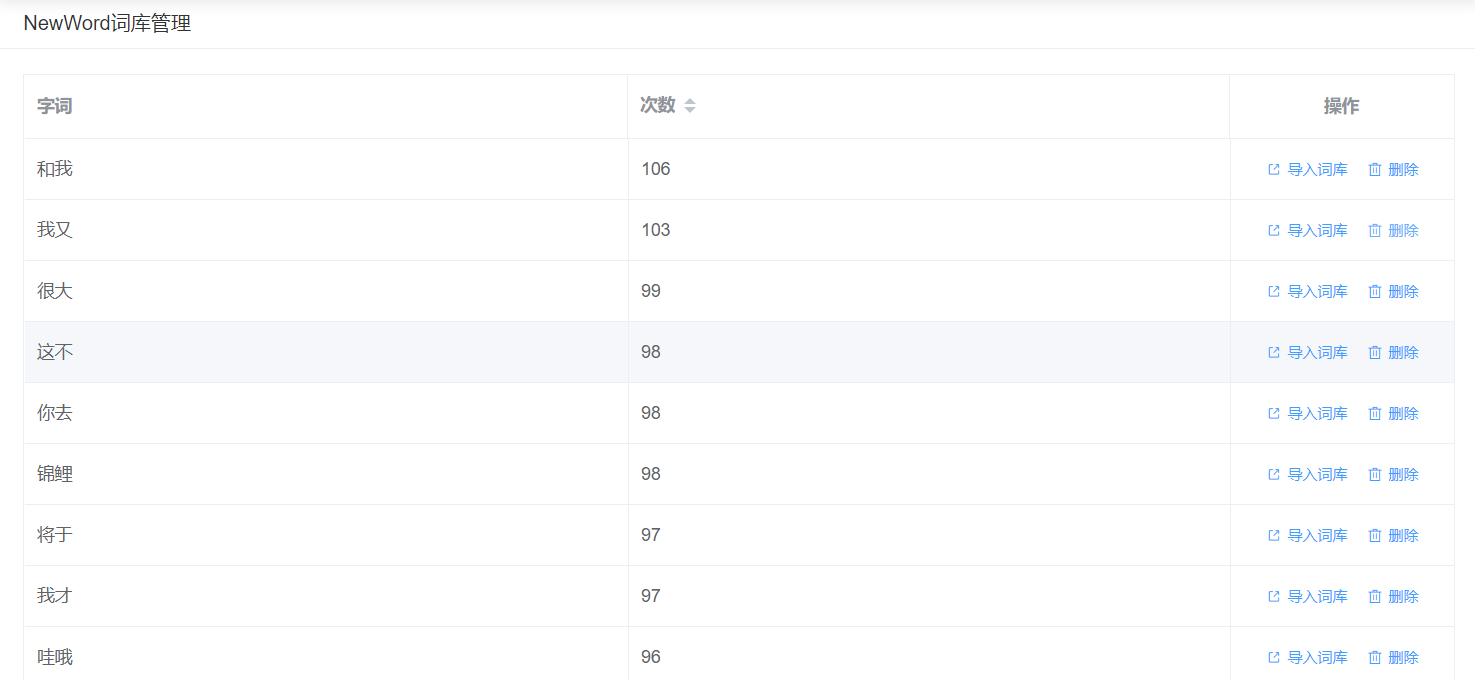


图5.29词库管理2

（2）系统消息

系统消息是全局的系统系统消息管理，包括接收存储过程的执行记录、数据获取的获取日志、数据处理进度等消息。功能实现主要是通过在执行完上述过程后插入消息表信息来实现。未读消息条数是在header中展示如图5.30，消息展示见图5.31。

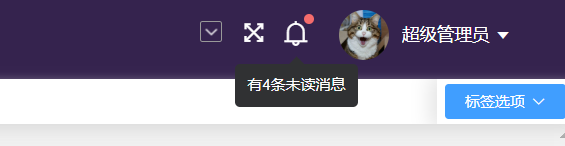


图5.30未读消息提示

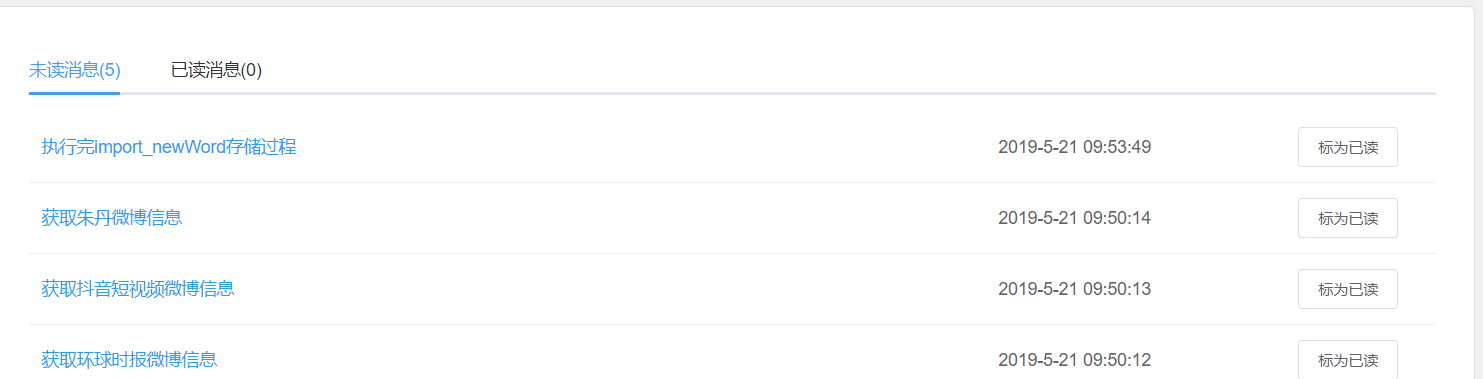


图5.31系统消息未读

(3)配置与调度

配置与调度是对系统的配置和存储过程调用状态，目前进实现对爬虫获取深度的配置,实现对事件状态的监控/日志的查看，具体功能见图5.32。

核心SQL:

//获取事件当前状态

SHOW VARIABLES LIKE'%event\_sche%';

//开启事件

SET GLOBAL event\_scheduler = 1;

//查看事件列表

SELECT name,interval\_value,interval\_field,created,comment FROM mySQL.event

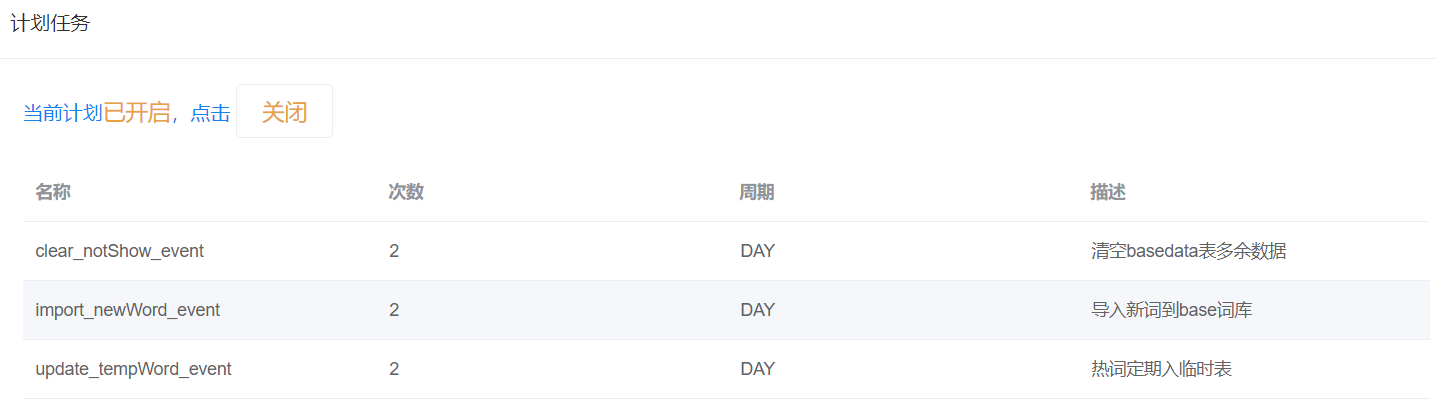


图5.33系统消息已读

## 5.4 存储过程与事件

在本系统中定义了清空不再显示词数据、将疑似词库导入基础词库、构建年度热词临时表等存储过程，这些存储过程都具有执行时间长、需定期执行的特征。因而在本系统中，通过部署相应的事件，周期性调用上述存储过程的方式实现数据的周期性的处理。存储过程定义如下：

（1）将疑似新词导入基础词库，实现疑似字词的new\_word词库表字词出现次数超过一定阈值（暂设100）时导入到基础表。

CREATE DEFINER=`wzy`@`localhost` PROCEDURE `import\_newWord`()

BEGIN

#将疑似新词插入baseword

INSERT INTO base\_words (word,counts,is\_show,type) (SELECT word,count,1 AS is\_show,'newWord' AS type FROM new\_words WHERE count>100);

COMMIT;

INSERT INTO `sbhdb`.`sys\_msg`(`message`, `message\_date`) VALUES ('执行完import\_newWord存储过程', NOW()); #写入消息

COMMIT;

INSERT INTO `sbhdb`.`pf\_dispatch`(`name`, `type`, `status`, `date`) VALUES ('import\_newWord', 'process', '1', NOW()); #写入事件调用日志

COMMIT;

END

清空数据表里的不再显示词。当词库管理中将字词设为不再显示时，应当删除weibo\_base\_data表中对应不再显示词，由于该过程过于缓慢，在本系统中在词库管理设为不再显示后将不再删除数据，转而采用定期执行存储过程方式执行。

CREATE DEFINER=`wzy`@`localhost` PROCEDURE `clear\_notShow`()

BEGIN

#清空weibo\_base\_data里对应的不再显示的内容

DELETE FROM weibo\_base\_data WHERE type='word' AND name IN (SELECT word from base\_words WHERE is\_show = 0);

COMMIT;

#省略写日志操作

END

重建热词数据临时表。在上文5.3.4话题热度分析及预测章节中已经提到。从weibo\_base\_data表中获取热词查询过程较为缓慢，且数据处理是一直进行中的，随着weibo\_base\_data表处理量的增加，年度热词是一直变化的直至数据处理完，因此通过定期调用存储过程的方式更新年度热词临时表、加快查询速度。

CREATE DEFINER=`wzy`@`localhost` PROCEDURE `update\_tempWord`()

BEGIN

DROP TABLE IF EXISTS `temp\_word`; #删除表

CREATE TABLE `temp\_word` ( #新建表 实现快速“清空表”操作

`id` int(11) NOT NULL AUTO\_INCREMENT,

`word` varchar(255) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL,

`counts` int(255) NULL DEFAULT NULL,

`name` varchar(255) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL,

PRIMARY KEY (`id`) USING BTREE

) ENGINE = InnoDB AUTO\_INCREMENT = 31 CHARACTER SET = utf8 COLLATE = utf8\_general\_ci ROW\_FORMAT = Compact;

COMMIT;

#获取2019年至今的数据

INSERT INTO temp\_word (word,counts,name) (SELECT name AS word,sum(counts) as counts,'2019年' AS name from weibo\_base\_data WHERE type='word' AND date > '2019-01-01' GROUP BY name ORDER BY counts DESC LIMIT 10);

#获取2018年度数据

INSERT INTO temp\_word (word,counts,name) (SELECT name AS word,sum(counts) as counts,'2018年度' AS name from weibo\_base\_data WHERE type='word' AND date BETWEEN '2018-01-01' AND '2019-01-01' GROUP BY name ORDER BY counts DESC LIMIT 10);

#此处省略添加2017年数据和写入日志操作

END

# 6 系统测试

## 6.1 数据获取速度测试

（1）测试方法

使用本机Pycharm集成开发环境测试，测试时关闭其他消耗资源大的进程，启动获取数据爬虫，运行一段时间后，记录获取数据增加量、数据获取量，计算数据获取速度。

（2）测试环境

表6.1 数据获取速度测试环境

|  |  |
| --- | --- |
| Python版本 | 3.7.2 |
| 数据库 | MySQL |
| OS | Win10 |
| IDE | Pycharm |
| CPU | Core I5-5200U |
| 内存 | 8G DDR3L |
| 数据库所在硬盘 | KingSton SS300S37A |
| 网络带宽 | 山东广电30Mbps |
| 运行参数 | -s JOBDIR=/data/jobdir |

（3）测试结果

测试所用时长6h50min，总获取数据量多于29万，获取数据速度每小时大于4万条。速度基本符合预期。

## 6.2 数据处理负载测试

（1）测试方法

使用本机Pycharm集成开发环境测试，关闭后台其他大量消耗资源进程，启动数据处理模块，使用window任务管理器记录内存、网络、I/O占用，多次实验，预估平均值。

（2）测试环境

表6.2 数据处理负载测试环境

|  |  |
| --- | --- |
| Python版本 | 3.7.2 |
| 数据库 | MySQL |
| OS | Win10 |
| IDE | Pycharm |
| CPU | Core I5-5200U |
| 内存 | 8G DDR3L |
| 数据库所在硬盘 | KingSton SS300S37A |

　　（3）测试结果

Python进程占用内存大致在500MB~600MB之间，符合程序涉及要求，但其对数据库的I/O占用过高，数据库I/O速率基本达到硬盘最高读写速度，数据库CPU占用率接近50%，资源消耗过大。数据获取模块亟需进一步优化。

## 6.3 数据分析展示功能点测试

（1）测试方法

使用本系统测试模块对附录Api表api做请求后台测试，测试后台获取数据是否有误；登陆系统对各个模块功能点进行测试，检查前端有无展示错误数据。

（2）测试环境

表6.3 数据展示功能点测试环境

|  |  |
| --- | --- |
| Java版本 | 1.8.2 |
| 数据库 | MySQL |
| OS | Win10 |
| IDE | IDEA、Hbuilder |

（3）测试结果

各个Api请求后台返回数据无误，前端未出现明显的错误，但部分界面数据加载过慢，需要优化。

# 结束语

经过几个月的系统设计开发，微博的数据获取、数据处理及数据分析展示系统统的各个功能基本实现。在系统设计的过程中出现过各种各样的问题，最终都一一解决。系统完工的那一刻，感到莫大的放松。不断的探索，不断的碰壁，又不断的找回思路，艰辛的过程也让自己开发水平得到了巨大的提升。在本系统开发中几乎应用到了在实习中学到的所有知识，对这些知识的认知高度也得到提升，算是对下一次就业打好基础。值得说的是在本系统框架的搭建中，能够做到真正从一个开发者角度思考问题、真正以效率为出发点思考框架的整合设计让我进步最快。

在系统获取的数据分析中也颇有感触。在统计结果的任何时间区段内热度排行的前15名中几乎全是演艺界的明星，几乎没有出现科技界、企业界大V，这也是我很不能理解的，难道说时代的风向标就是这些演艺界明星？在查看大多数大V发表微博的情感状态时，发现这些大V大多都会塑造积极向上的形象，值得欣慰，所有大V都树立起积极向上的旗帜，这号召力可是不容小觑的；从微博用户分析上，90后00后用户已接近50%，这大概体现了当今网民在迅速年轻化趋势吧，我想90后、00后已成为建设社会的最强生力军了。从用户的地区分布上，发现用户多少分布基本与当地经济水平成正比，在经济突出的北上广深用户所占比重极高，江苏浙江山东次之，不难推论，经济发展水平直接影响人们的网络社交发展程度。

系统设计至今仍存在许多不足之处，比如数据获取速度还不够快、数据处理速度较慢、目前数据处理量还较少、数据查询较慢、没有建立起Python与JavaWeb联系系统等问题，这些都是下一步应该解决完善的。

# 致谢

此次毕业设计或许是耗时最长、耗费精力最大的一次独立开发了，也算是对自己毕业毕业的最好献礼，在这个过程中，得到许多老师和同学的帮助，没有他们的帮助和支持就或许没有这个系统的完成了。

在实习时就开始了本系统的设计，感谢唐鸣泽同学在最初学习框架时的帮助；感谢父母在我辞职回家“不干正事儿”专注毕设时的理解和支持；感谢王立宏、董浩老师在开题报告、中期答辩中给予的意见和建议；感谢赵前同学在论文撰写时的相互学习相互支持相互促进；感谢毕远伟老师对这次毕设的支持；在最后，最最感谢从项目伊始到结束一直给予我信心、动力，提供建议，帮忙改正纰漏，帮忙修改论文的郭艳燕老师。

# 参考文献

[1]　马骁. 基于数据采集的电商评论分析系统的研究[D].青岛科技大学,2018．

[2]　牛一捷. 数据库决策支持测评系统的改造和InnoDB索引的分析[D].吉林大学,2005.

[3]　王丹,孙晓宇,杨路斌,高胜严.基于SpringBoot的软件统计分析系统设计与实现[J].软件工程,2019,22(03):40-42.

[4]　王鹏强.基于vue的MVVM框架的研究与分析[J].电脑知识与技术,2019(11):97-98+100.

[5]　王金峰,彭禹,王明,钟声,赵雪辉.基于网络爬虫的新浪微博数据抓取技术[J].中小企业管理与科技(上旬刊),2019(01):162-163.

[6]　严慧,彭绪富,朱小婉,熊旭辉,董叶豪.基于Scrapy-Redis分布式数据采集平台的设计与实现[J].湖北师范大学学报(自然科学版),2019,39(01):19-25.

[7]　赵志升,靳晓松,温童童,梁俊花.基于Python-Snownlp的新闻评论数据分析[J].科技传播,2018,10(18):104-105.

# 附录A功能测试部分Api表

|  |  |  |  |
| --- | --- | --- | --- |
| Api名称 | Api功能 | 参数 | 返回值 |
| LoginApi | 管理员登录 | Id,pwd | True/False |
| touristLoginApi | 访客登录 | 无 | True |
| LogoutApi | 登出 | Cookie | True/False |
| getInfoApi | 获取登录用户信息 | Cookie | Data（router，menu等） |
| getDataSizeApi | 获取首页数据 | 无 | Data（首页所有数据） |
| getCpuUsdApi | 获取服务器Cpu负载 | 无 | Num（百分数） |
| getUserlineDateApi | 抓取用户折线图数据 | 无 | Data(折线图数据) |
| getWeiboLineDataApi | 抓取微博折线图数据 | 无 | Data(折线图数据) |
| getMsgNum | 获取未读消息个数 | 无 | Num |
| updateMsgState | 改变消息状态 | Id | True/False |
| delMsgById | 删除消息 | Id | True/False |
| wordsImportApi | 导入词库 | Path、type | Num（导入数据量） |
| getUserHotDataApi | 获取热度排行 | T1、T2、name | Data（排行） |
| getUserPortraitapi | 获取用户词云 | T1、T2、Id | Data（词云数据） |
| getFansInfoApi | 获取用户粉丝分布 | Id | Data（饼图、柱状图） |
| getUserWeiboSetimentApi | 用户及粉丝情感分布 | Id | Data（柱状图） |
| getUserHotlineDateApi | 获取大V热度趋势曲线 | T1、T2、Id | Data（折线图） |
| getTopicOderDataApi | 获取话题排行 | T1、T2 | Data（排行） |
| getAllUserInfoApi | 抽样获取用户信息 | 无 | Data（饼图、柱状图） |
| getHotBaseDataApi | 获取表情、字词排行 | T1、T2、Type | Data(排行) |
| getBaseDataUserApi | 获取字词/表情常用用户 | T1、T2、Word | Data(排行) |
| getPassYearHotWordApi | 获取过去几年的热词 | 无 | Data(排行) |
| getConfigApi | 获取配置信息 | Name | Data(Bean) |
| getEventConfigDateApi | 获取数据库事件信息 | 无 | Data(table、status) |

# 附录 一元线性回归模型原理及推导

如图１散点分布图所示，一元线性回归模型即从样本集

若干散点中（即图示蓝点），寻找使每个点误差和最小的 线性方程（即途中红线）。

每个点的误差为：

即：

使得所有点误差和最小则：

利用微积分的对求偏导整理可得到：

通过ａ求得ｂ：

通过上述推导即可建立一元线性回归模型，在本系统中便是通过时间（ｘ）、热度（ｙ）数据集，训练求得误差最小的一元线性回归模型，进而预测走势。

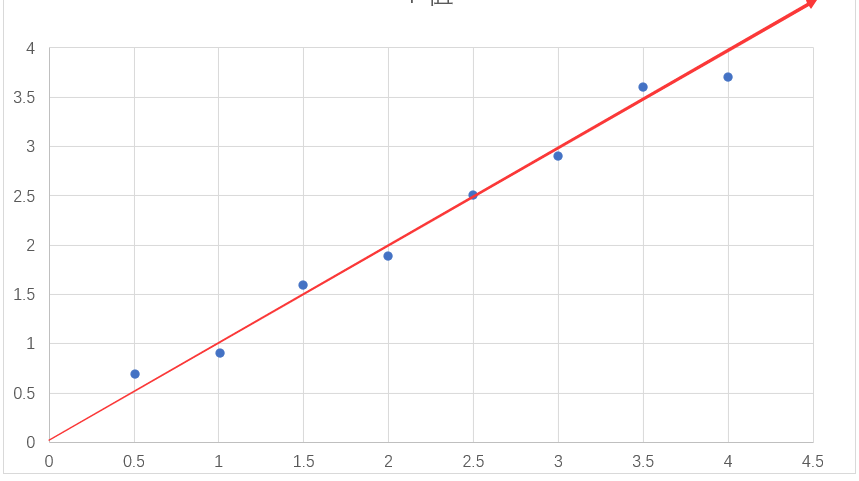


图１散点分布图