微博用户情感分析系统的设计与实现

学　　院　　计算机学院

专　　业：　　软件工程

姓　　名：　　黄少鹏　　学　　号：　　150202102464

指导老师：　　蔡培茂　　职　　称：　　讲师

中国·珠海

二〇一九年 五 月

诚信承诺书

本人郑重承诺：本人承诺呈交的毕业设计《微博用户情感分析系统的设计与实现》是在指导教师的指导下，独立开展研究取得的成果，文中引用他人的观点和材料，均在文后按顺序列出其参考文献，设计使用的数据真实可靠。

本人签名：_____

日期：_____年___月____日

微博用户情感分析系统的设计与实现

摘　要

2018 年 8 月 20 日，中国互联网络信息中心（CNNIC）在北京发布第 42 次《中国互联网络发展状况统计报告》，报告显示截至 2018 年 6 月 30 日，我国网民规模达 8.02 亿。根据 2018 年度新浪微博第三季度财报显示，截至 2018 年 9 月，新浪微博月活跃用户高达 4.46 亿。可以得出结论：每两个中国网民之中，就会有一个人使用新浪微博。这说明新浪微博已经渗透入我们的日常生活之中，人们喜爱在新浪微博平台发表自己的日常生活的大小事，表达喜怒哀乐的情绪，在大数据的时代，用数据说话最有说服力，因此研究新浪微博的数据具有重大意义。

本系统主要通过 python 语言对新浪微博进行爬虫，然后利用开源中文情感分析库 SnowNLP 训练出自己的情感分析模型，然后将情感分析结果展示到用前端页面。技术框架后端主要使用 Django+Xadmin，前端主要使用 Vue+Vuex+ElementUI+V-charts。系统分为五大模块：微博用户信息及用户每条微博爬虫分析模块、单条微博及评论爬虫分析模块、日均十万条数据的微博多用户信息和用户微博及每条微博评论爬虫模块、独立的文本情感分析模块、已爬虫用户展示模块。

微博用户情感分析系统的设计与实现，能够快速分析个人微博情感倾向，一定程度上能反应出一个人的情感态度；能够对某一热门微博进行分析，一定程度上人们对该热门微博的态度；同时对完善情感分析提供更多的语料，对构建完善的中文情感分析库也有一定的意义。

关键词：微博爬虫；中文分词；情感分析；Vue；Django

Design and Implementation of WeiBo User Emotion Analysis System

Abstract

On August 20, 2008, China Internet Network Information enter (CNNIC) released the 42nd statistical report on Internet development in China in Beijing, which showed that by June 30, 2018, the number of Chinese netizens had reached 802 million. According to the 2018 sina weibo in the third quarter results show that as of September 2018, sina weibo monthly active users up to 446 million. May safely draw the conclusion that every two of China's netizens, there will be a people use sina weibo. Suggesting that sina weibo has penetrated into our daily life, people like published in sina weibo platform own necessities of daily life, expressing emotions, the joys and sorrows in the era of big data, using data to talk most persuasive, so the data of sina weibo is of great significance.

This system mainly through the python language to crawler sina weibo and use open source sentiment analysis in Chinese library SnowNLP training out their own emotional analysis model, and then the analysis results show affection from the front-end page. Django+Xadmin is mainly used in the Backend technical framework, while Vue+Vuex+ElementUI+ v-charts are mainly used in the front-end. System is divided into five modules: user information and user each post crawler analysis module, a single weibo and comment on the crawler data analysis module, the daily average of one hundred thousand weibo user information and user weibo and each post comments crawler module, independent text sentiment analysis module, display module has the crawler users.

The design and implementation of the emotion analysis system for microblog users can quickly analyze the personal emotion tendency of microblog, and to a certain extent can reflect a person's emotion and attitude. To some extent, people's attitude towards a popular microblog can be analyzed. At the same time, it provides more linguistic materials for the improvement of emotion analysis and has certain significance for the construction of a complete Chinese emotion analysis database.

Keywords: WeiBo Crawler; Chinese Word Segmentation; Emotion Analysis; Vue; Django

目　录

1 前言

1.1 项目开发背景

在大三的时候第一次接触 Python 这门语言，是利用开源库微信跳一跳跑分程序进行微信跳一跳跑分，那时候觉得 Python 很神奇，后面陆续学会一些爬虫的技术，在大四的时候又对很火的 AI 感兴趣就跑去学了点 AI 的知识，本科毕业设计是最重要也是最后一个大学的项目实践，回顾整个大学，除了大三跟着老师做了一个人口密度流动热力图预测系统之外，实在没有完整做出点有意义的项目，于是我想着要把大学学过的有意思的爬虫和 AI 结合起来，写个有意义的程序，于是就有了今天的微博用户情感分析系统。

1.2 项目开发的意义

随着互联网浪潮的激起，社交平台的迅速流行，社交平台开始占据了人们越来越多的时间。新浪微博从 2009 年 8 月推出至今，经历了近 10 年的时间，其用户数量已经占据了中国网民的一半多，可见新浪微博对网民的影响力有多深。人们已经习惯了在微博上与人沟通、表达自己的想法和记录自己的日常点滴，同时微博每天都在产生新的信息量，如果可以挖掘微博用户的微博文本信息，研究微博文本的情感倾向，获得微博用户的心情，对某个热点事件的看法，那不仅可以得到巨大的商业价值，还能对社会的稳定做出贡献。

本系统的目的正是为了可以获取新浪微博文本的信息，然后对其文本信息进行情感分析、最后得到微博用户的心情，得到用户对某一个热点事件的看法，最后将这些信息展示在网页上，让人们可以直观的得到有意义的信息。从而实现该系统本身的价值。

1.3 本章小结

微博用户情感分析系统是我觉得非常有意思的项目，在网上搜了很多资料，我没有看到有人用我所选的技术完整做出来这个项目，都是做出一点零零散散的一部分，因此我觉得很有挑战性，同时也符合自己对毕业设计的期待。

2 可行性分析

本节主要从四个方面进行可行性分析，分别是：技术可行性、经济可行性、操作可行性、法律可行性。

2.1 技术可行性

在技术上采用 Python 成熟的 Django 框架作为后端主要框架，采用 Vue 作为前端主要框架，爬虫模块采用 Python 的插件 requests 和 etree，持续爬虫采用成熟的 Scrapy 框架运行在 Scrapyd 服务上爬虫。

项目上难点：Django 和 Vue 的前后端联调存在跨域问题、Scrapy 部署在 Scrapyd 上通过 Http 实现请求启动爬虫问题、Vuex 状态管理不熟悉问题、Vue 请求是 json 格式 Django 不会收到该请求、Vue 解决了跨域问题后请求 Django 是无法成功请求问题、requests 爬虫新浪微博存在封号问题等。

开发人员有一定的开发经验，拥有爱学习的心态、以上难题均可解决，因此技术层面可实现。

2.2 经济可行性

硬件方面系统不需要用到太大的资源，虽说爬虫过程，情感分析过程的速度取决于硬件，但普通的服务器也能应付的过来。主要采用 Python+Vue 以及一些开源框架辅助开发，所以不需要支付任何费用。

2.3 操作可行性

本系统主要采用 B/S 的架构进行开发，因此用户只需要一个装有浏览器的电脑即可访问本系统。在操作可行性上简易便捷。

2.4 法律可行性

爬虫的基本操作就是模拟人到处点击页面获取数据，包括我们每天用的百度搜索，抢票软件，均是基于爬虫技术，《网络安全法》没有规定说明爬虫犯法，同时本系统的爬虫获取的都是公开的信息，因此不触碰任何法律。同时本系统均是利用开源软件开发，用于学习研究用途。在法律上没有任何问题。

2.5 本章小结

经过上面四个层面的分析，最后得出结论，该系统的开发是可行的。

3 系统需求分析

本系统的设计目标是为了对微博用户进行情感分析，需要先获取微博的文本信息，然后对这些微博信息稍微处理，接着再利用情感分析库对文本信息进行分析，最后在前端展示分析结果，除此之外，用户还可以登录后台 Xadmin 管理系统查看爬虫数据，也可以下载数据。

3.1 系统参与者

分析系统之后，系统有情感分析员和后台系统管理员两个参与者。

3.1.1 情感分析员

情感分析员打开系统页面之后，输入微博 id 爬虫并查看信息。输入微博 id 以及 Cookie 运行持续爬虫。输入任意文本进行情感分析，查看数据库内爬虫过的用户。

3.1.2 管理员

管理员打开系统页面之后，输入微博 id 爬虫并查看信息。输入微博 id 以及 Cookie 运行持续爬虫。输入任意文本进行情感分析，查看数据库内爬虫过的用户。同时管理员还可以查看后台爬虫的所有信息，并且可以修改单用户爬虫的 Cookie，导出所有爬虫的信息。

3.2 功能需求分析

本系统将划分为五大功能模块：微博用户模块、单条微博模块、持续爬虫模块、中文情感分析模块、已爬虫用户模块。

微博用户模块：该模块的主要通过输入微博用户 id 传给后台，同时后台通过 requests+etree 爬取微博用户并解析数据存入数据库，经过一系列的处理后的微博内容数据进行情感分析，同时存入数据库，然后返回数据库的信息给前端，前端展示微博用户的信息，用户所发的微博，每条微博的关键字、词性、情感分析结果，同时生成词云。

单条微博模块：该模块主要通过输入单条微博的标识 id 传给后台，同时后台通过 BeautifulSoup 解析微博内容，通过 lxml 解析所有的评论，然后存入数据库。同时对所有评论进行情感分析，把关键字、词性、情感分析结果存入数据库，同时生成词云。

持续爬虫模块：该模块主要通过前端输入微博 id、微博的 Cookie 两个信息然后启动运行在 6800 端口的 Scrapyd 服务器上的 Scrapy 持续爬虫，本爬虫可爬取多人信息，并展示，同时可从后台管理系统查看爬虫信息。

中文情感分析模块：该模块通过前端输入中文文本，后端接收文本信息，然后进行情感分析，返回词频统计、文本情感值、以及三个关键词，前端进行展示。

已爬虫用户模块：该模块主要显示已爬虫过的用户信息模块。

3.3 性能需求分析

为了保证软件能够顺利的运行，页面加载的及时性，系统的稳定性，各项功能能够顺利运行，我们需要保证本系统有最基本的硬件需求：

服务端基本要求：

√CPU：英特尔 i7-4710MQ

√内存：8G

√操作系统：window10 家庭版 64 位

√数据库版本：MySQL8.0

√后端服务器：Django2.1.5

√前台服务器：Node6.0

√系统开发语言：Python3.6、Vue2.5

√系统开发平台：Visual Studio Code1.32.3

√开发架构：B/S 架构

客户端基本要求：

√PC：带有浏览器的 PC 均可

√浏览器版本：FireFox、Chrome、IE11 及以上版本

3.4 主要用例

主要用例这里列出了系统中重要的五大功能模块的用例的详细叙述以及其用例图。

3.4.1 微博用户模块用例及用例图

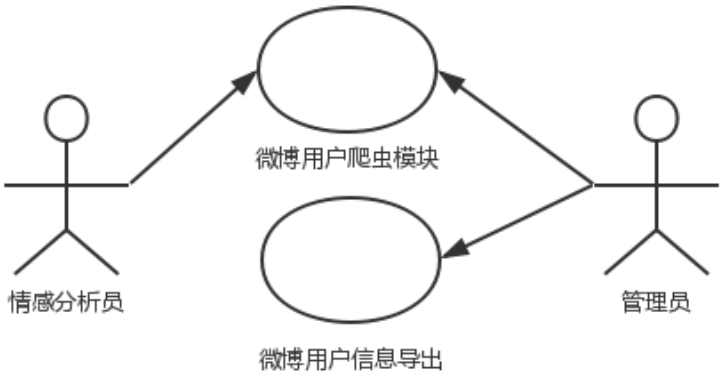

图 3.1 微博用户模块用例图

用例 1：微博用户模块

范围：微博用户管理系统

级别：用户目标

主要参与者：情感分析员

前置条件：情感分析员进入系统页面

后置条件：情感分析员输入需要爬虫的微博 id

主要流程：

1．情感分析员打开微博用户管理系统的主页；

2．情感分析员在界面中输入爬虫的微博 id；

3．系统检查情感分析员录入微博 id 的正确性与完整性；

4．系统保存输入的微博 id，开始启动爬虫，提示正在疯狂爬虫中；

5．爬虫完毕后，系统将显示微博用户信息，微博内容，微博的情感分析结果。

扩展：

2a 系统检查发现情感分析员输入错误的微博 id：

1．系统将不进行爬虫返回错误信息

3.4.2 单条微博模块用例及用例图

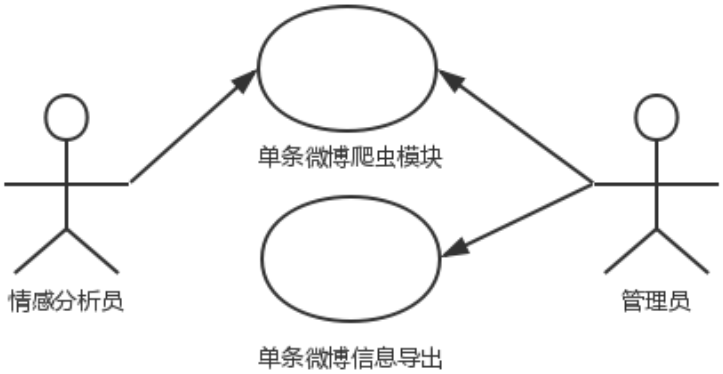

图 3.2 单条微博模块用例图

用例 1：单条微博模块

范围：微博用户管理系统

级别：用户目标

主要参与者：情感分析员

前置条件：情感分析员进入系统页面

后置条件：情感分析员输入需要爬虫的微博内容 id

主要流程：

1．情感分析员打开微博用户管理系统的主页；

2．情感分析员在单条微博爬虫界面中输入爬虫的微博内容 id；

3．系统检查情感分析员录入微博内容 id 的正确性与完整性；

4．系统保存输入的微博内容 id，开始启动爬虫，提示正在疯狂爬虫中；

5．爬虫完毕后，系统将显示微博用户信息，微博，微博的情感分析结果。

扩展：

2a 系统检查发现情感分析员输入错误的微博内容 id：

1．系统将不进行爬虫返回错误信息

3.4.3 中文情感分析模块用例及用例图

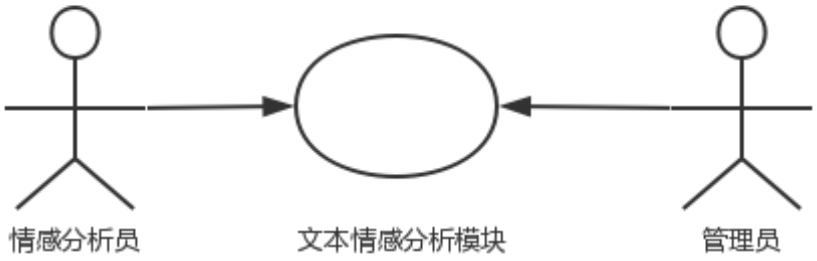

图 3.3 中文情感分析模块用例图

用例 1：中文情感分析模块

范围：微博用户管理系统

级别：用户目标

主要参与者：情感分析员

前置条件：情感分析员进入系统页面

后置条件：情感分析员输入需要输入一段中文

主要流程：

1．情感分析员打开微博用户管理系统的界面；

2．情感分析员在中文情感分析界面中输入任意一段文本；

3．系统检查用户录入文本内容发送给后台计算情感分析值；

4．系统计算完情感分析值实时返回词频、文本情感分析、文本关键词；

扩展：

2a 系统界面文本框上方存在随机一段文字：

1．点击该随机文字将随机产生一组中文文本并计算情感值展示

3.4.4 持续爬虫模块用例及用例图

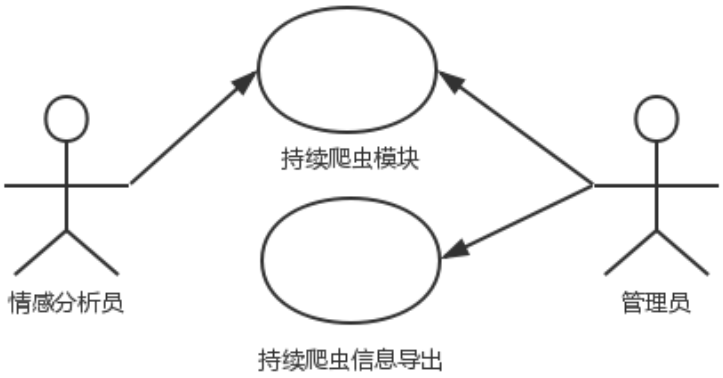

图 3.4 持续爬虫模块用例图

用例 1：持续爬虫模块

范围：微博用户管理系统

级别：用户目标

主要参与者：情感分析员

前置条件：情感分析员进入系统页面

后置条件：情感分析员输入需要输入持续爬虫 id 和 Cookie

主要流程：

1．情感分析员打开微博用户管理系统的界面；

2．情感分析员在持续爬虫界面中输入微博 id 和 Cookie；

3．系统检查爬虫 id 的正确性并保存持续爬虫和 Cookie；

4．后台调用 Scrapyd 服务开始启动爬虫；

扩展：

2a 系统界面点击按钮可进入 Scrapyd 页面：

1．在 Scrapyd 持续爬虫页面可以看到爬虫日志、正在进行的爬虫

3.4.5 已爬虫模块用例及用例图

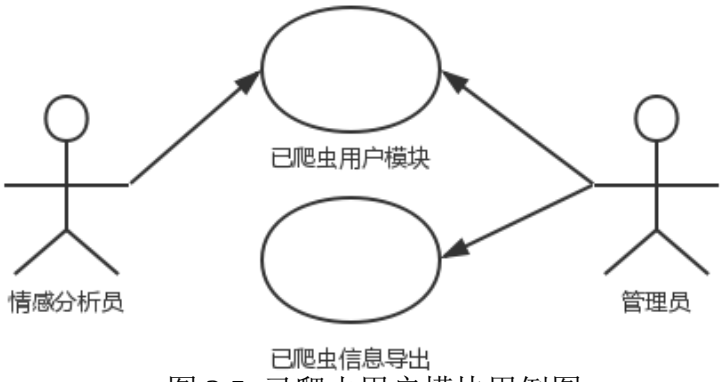

图 3.5 已爬虫用户模块用例图

用例 1：已爬虫用户模块

范围：微博用户管理系统

级别：用户目标

主要参与者：情感分析员

前置条件：情感分析员进入系统页面

后置条件：无

主要流程：

1．情感分析员打开微博用户管理系统的界面；

2．点击进入已爬虫界面；

3．系统从数据库拉取微博用户头像以及微博用户 id；

4．点击微博用户头像可以查看该用户信息；

3.5 本章小结

通过对系统的需求分析，明确了系统开发的流程，对后续系统正确开始起了一定的指导作用。

4 系统概要设计

本章节主要从五个方面来阐述系统的概要设计，分别是：系统整体流程图、系统活动图、系统领域模型、系统架构图、数据库设计。

4.1 系统整体流程图

如图 4-1 后台系统整体流程图所示，图中表述了后台管理系统的管理员的操作执行过程。首先是登录后台管理系统，登录正确以后就可以进入后台管理系统的界面，在后台管理系统的界面里面，可以对爬虫设置、用户信息、微博信息、微博评论、情感分析等数据的增删改查，还可以导出这些数据。

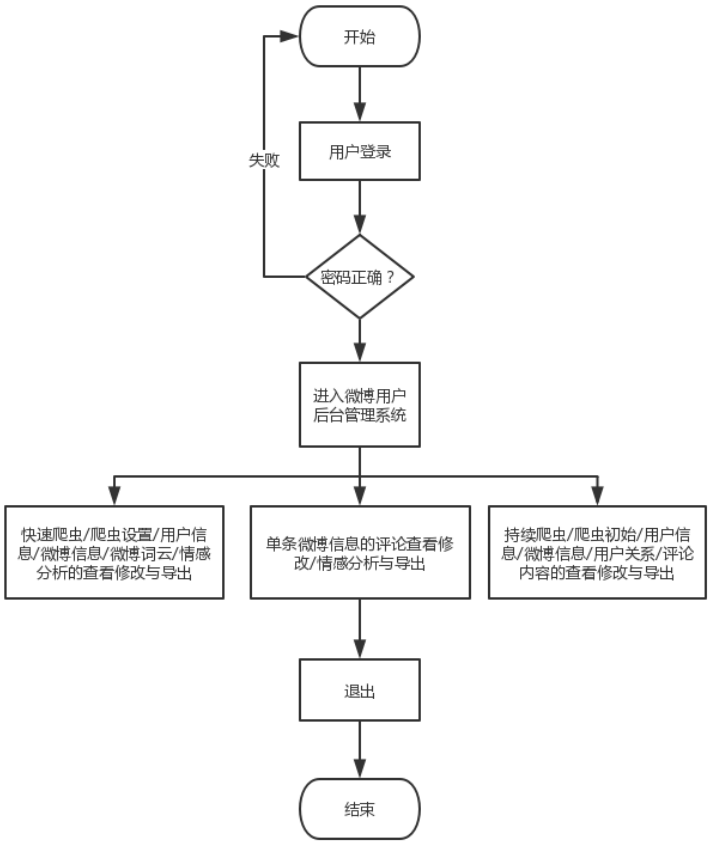

图 4.1 后台系统整体流程图

如图 4-2 前台系统整体流程图所示，图中表述了前台系统的用户的操作执行过程。首先打开网页进入前台系统，在前台系统可以看到五大模块：微博用户爬虫、单条微博爬虫、持续爬虫、中文情感分析、已爬虫用户。当进入微博用户爬虫、单条微博爬虫，输入正确微博 id、正确微博内容 id 进入分析页面；当进入持续爬虫模块，输入正确微博 id，启动 Scrapyd 爬虫；中文情感分析当输入文本，展示情感分析结果，词频和关键词；进入已爬虫用户，展示已爬虫数据。

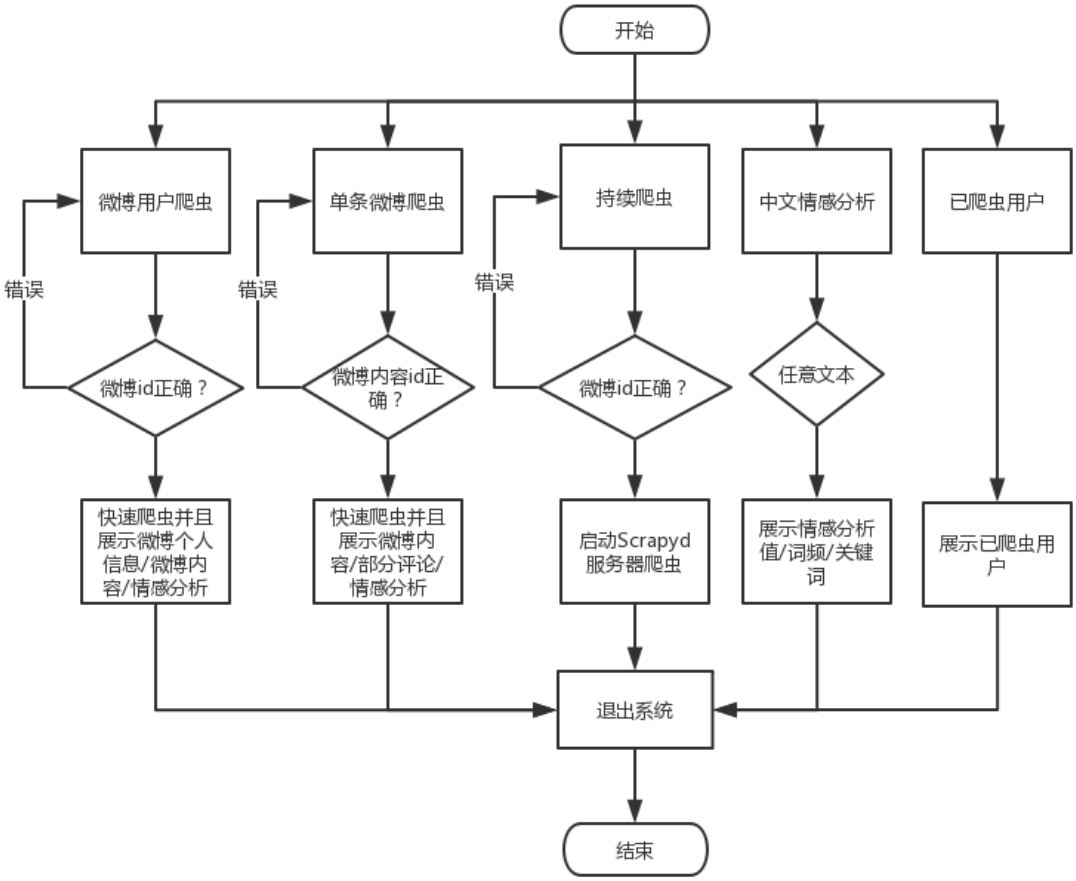

图 4.2 前台系统整体流程图

4.1.1 功能模块设计概述

前端功能模块设计图如下表所示：

表 4.1 前端功能模块设计

系统	功能类别	功能说明	描述
用户端	情感分析员	微博 id 爬虫	输入微博 id 得到爬虫结果
用户端	情感分析员	微博内容 id 爬虫	输入微博内容 id 得到爬虫结果
用户端	情感分析员	启动持续爬虫	输入微博 id 和 cookie 启动爬虫
用户端	情感分析员	中文文本情感分析	输入任意中文文本开始情感分析
用户端	情感分析员	查看已爬虫用户	进入已爬虫页面查看已爬虫用户
管理员端	管理员	查看微博 id 爬虫数据	管理员进入管理员端查看微博 id 数据
管理员端	管理员	增加微博 id 爬虫数据	管理员进入管理员端增加微博 id 数据
管理员端	管理员	删除微博 id 爬虫数据	管理员进入管理员端删除微博 id 数据
管理员端	管理员	修改微博 id 爬虫数据	管理员进入管理员端修改微博 id 数据

管理员端	管理员	查询微博 id 爬虫数据	管理员进入管理员端查询微博 id 数据
管理员端	管理员	导出微博 id 爬虫数据	管理员进入管理员端导出微博 id 数据
管理员端	管理员	查看微博内容 id 数据	管理员进入管理员端导出微博内容
管理员端	管理员	增加微博内容 id 数据	管理员进入管理员端增加微博内容
管理员端	管理员	删除微博内容 id 数据	管理员进入管理员端删除微博内容
管理员端	管理员	修改微博内容 id 数据	管理员进入管理员端修改微博内容
管理员端	管理员	查询微博内容 id 数据	管理员进入管理员端查询微博内容
管理员端	管理员	查看持续爬虫数据	管理员进入管理员端查看持续爬虫
管理员端	管理员	增加持续爬虫数据	管理员进入管理员端增加持续爬虫
管理员端	管理员	删除持续爬虫数据	管理员进入管理员端删除持续爬虫
管理员端	管理员	修改持续爬虫数据	管理员进入管理员端修改持续爬虫
管理员端	管理员	查询持续爬虫数据	管理员进入管理员端查询持续爬虫
管理员端	管理员	导出持续爬虫数据	管理员进入管理员端导出持续爬虫
管理员端	管理员	注册管理员	管理员进入管理员端注册管理员

后端功能模块设计图如下表所示：

表 4.2 后端功能模块设计

功能类别	功能说明	描述
角色管理	添加角色	管理员添加新的角色
角色管理	修改角色信息	管理员修改角色信息
角色管理	删除角色	管理员删除系统角色
微博 id 爬虫	爬新浪微博个人首页	爬取个人信息
微博 id 爬虫	爬新浪微博个人首页	爬取微博内容信息
单条微博 id 爬虫	爬新浪微博个人首页	爬取单条微博信息
单条微博 id 爬虫	爬新浪微博个人首页	爬取单条微博所有评论
持续爬虫	爬取新浪微博个人首页	爬取个人信息
持续爬虫	爬取新浪微博个人首页	爬取微博内容
持续爬虫	爬取新浪微博个人首页	爬取每条微博评论
文本情感分析	对文本情感分析	分析后返回词频
文本情感分析	对文本情感分析	分析后返回情感分析值
文本情感分析	对文本情感分析	分析后返回关键字

4.2 系统活动图

如图 4.3，是微博用户情感分析系统的活动图（以情感分析员进入页面输入微博 id

开始爬虫为例），在活动图中，比起用例增加了额外的其他一些细节，可以更加清晰、直接的看到情感分析员进入页面输入微博 id 之后整个系统所发生的活动。

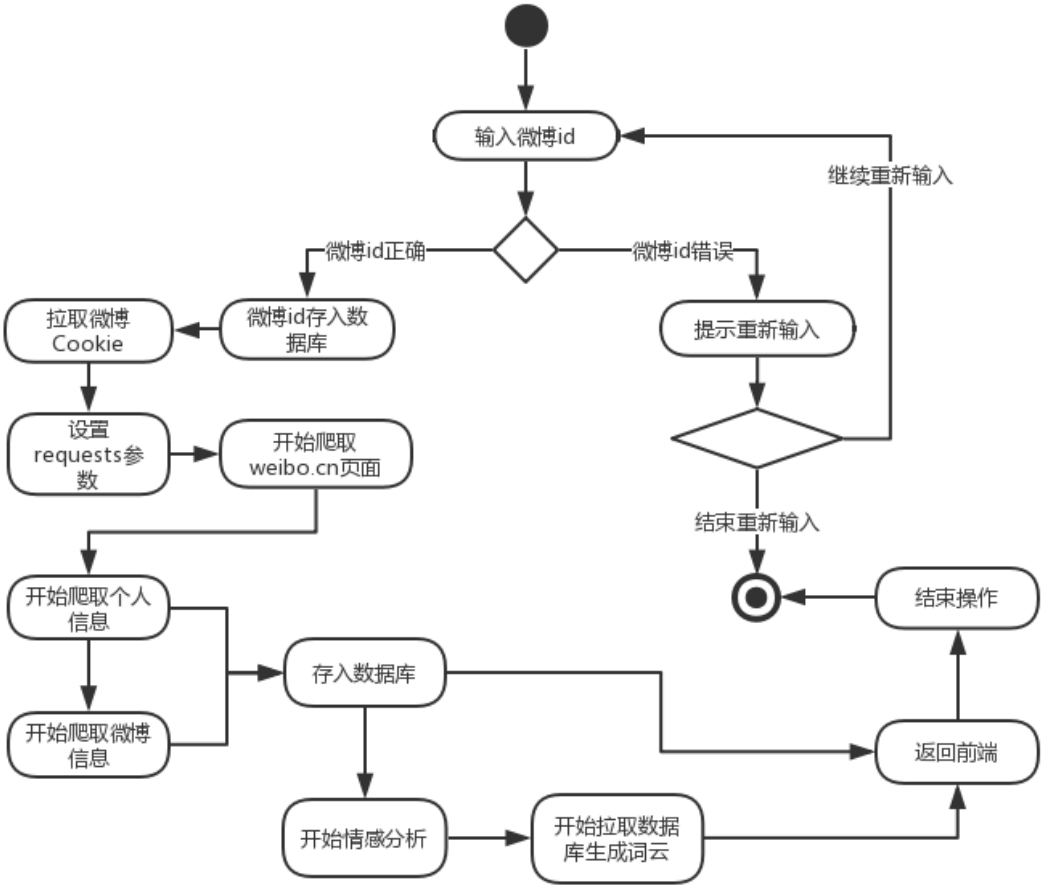

图 4.3 微博 id 爬虫活动图

4.3 系统领域模型

领域模型是指对现实世界的可视化抽象，建立一个领域模型可以增强用户理解，降低生活之中的概念和软件实体类之间的表示差异，如图 4.4，画的是微博用户情感系统的领域模型，在这个模型我们分析出了情感分析员和管理员之间的概念模型，各个模型之间存在着联系。

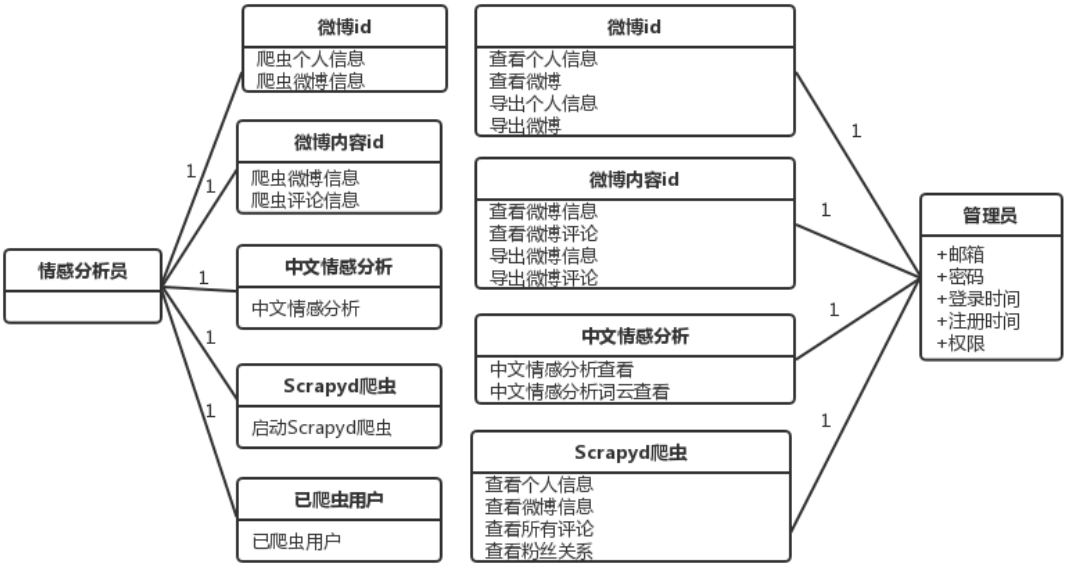

图 4.4 系统领域模型图

4.4 系统架构图

为了明确系统的开发流程，把元素与元素的关系图示化，明确定义、图例和说明的元素，规范化每一个接口和每一个行为，为系统设计了系统架构图，从图中可以明确看到每一层用什么样的视图和视图组合。Django 的模式是由 Models 抽象层、Templates 模板和 Views 视图组成，简称 MTV 模式，而 Vue 是由 View、ViewModel、Model 三部分组成，简称 MVVM，本系统采用两者合并的方式，简称 MMVVMV 的架构，画出如图 4.5 的系统架构图。

我们可以看到每一层的分工明确，前端由 ElementUI 等技术渲染视图层、通过 axios 的 GET 和 POST 的请求业务层、业务层又和数据层打交道，而数据层直接和数据库联系在一起，最后整个系统运行在 Django 的 wsgi 上。

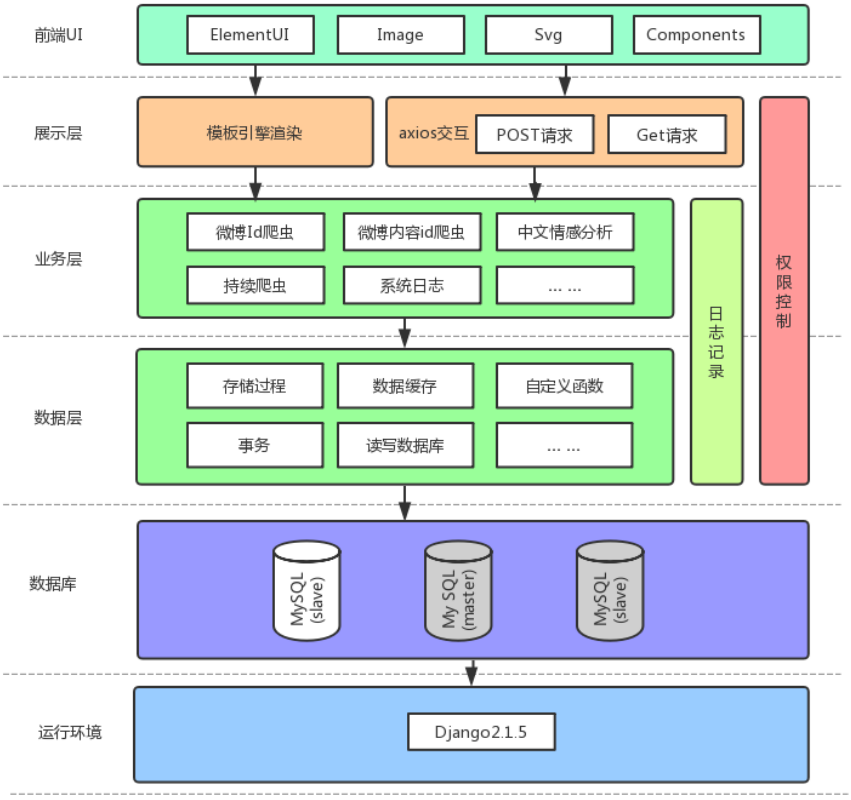

图 4.5 系统架构图

4.5 数据库设计

本小结主要从数据库环境说明、数据库命名规则、数据库 E-R 设计图以及数据字典四个方面来阐述系统的数据库设计。

4.5.1 数据库环境说明

数据库名称：MySQL

数据库版本：8.0.15

数据库：weiboanalysissystem

运行环境：window10 家庭版 64 位

4.5.2 数据库命名规则

如表 4.3 所示为数据库的命名规则，我们可以看到每一张表的命名都是由下划线-之间隔开，每一个 django 的模块都是一个 app 组成，每一个 app 的名称对应着下划线的前面的说明，每一个在 django 定义的表名对应着下划线后面的内容。由此可见，整个表

名是由安装的 app 名称和 models 里面定义的表名组合而成。这样的好处是这样一眼看清独立的 app，以及其独立的表，模块化开发。

表 4.3 数据库术语表

缩写、术语	解释
Weiboanalysissystem	微博用户管理系统的数据库名称
auth_group	认证组表
auth_group_permissons	认证组权限表
auth_permission	认证权限表
auth_user	认证用户表
auth_user_groups	认证用户组表
auth_user_user_permissions	认证用户用户权限表
django_admin_log	admin 日志表
django_content_type	内容类型表
django_migrations	orm 生成记录表
django_session	服务器缓存表
scrapydapi_commentinfo	持续爬虫评论信息表
scrapydapi_target	持续爬虫设置表
scrapydapi_tweetsinfo	持续爬虫微博内容表
scrapydapi_userinfo	持续爬虫用户信息表
scrapyapi_commentinfo	快速爬虫微博评论表
scrapyapi_commentweiboinfo	快速爬虫微博内容表
scrapyapi_imginfo	快速爬虫生成词云表
scrapyapi_target	快速爬虫设置表
scrapyapi_tweetsinfo	快速爬虫微博内容表
scrapyapi_userinfo	快速爬虫用户信息表
xadmin_bookmark	后台系统标签表
xadmin_log	后台系统日志表
xadmin_usersettings	后台系统用户设置表
xadmin_userwidget	后台系统用户自定义控件表

4.5.3 数据库 E-R 设计图

实体-联系图（Entity Relationship Diagram）即我们说的 E-R 图，E-R 图提供了表示实体类型、属性和联系的方法，其意思是用来表达现实世界的概念，为了描述现实世界的关系概念模型而创立的一种图。如图 4.6 表示的就是微博用户情感分析系统的现实概

念的一个结构模型。

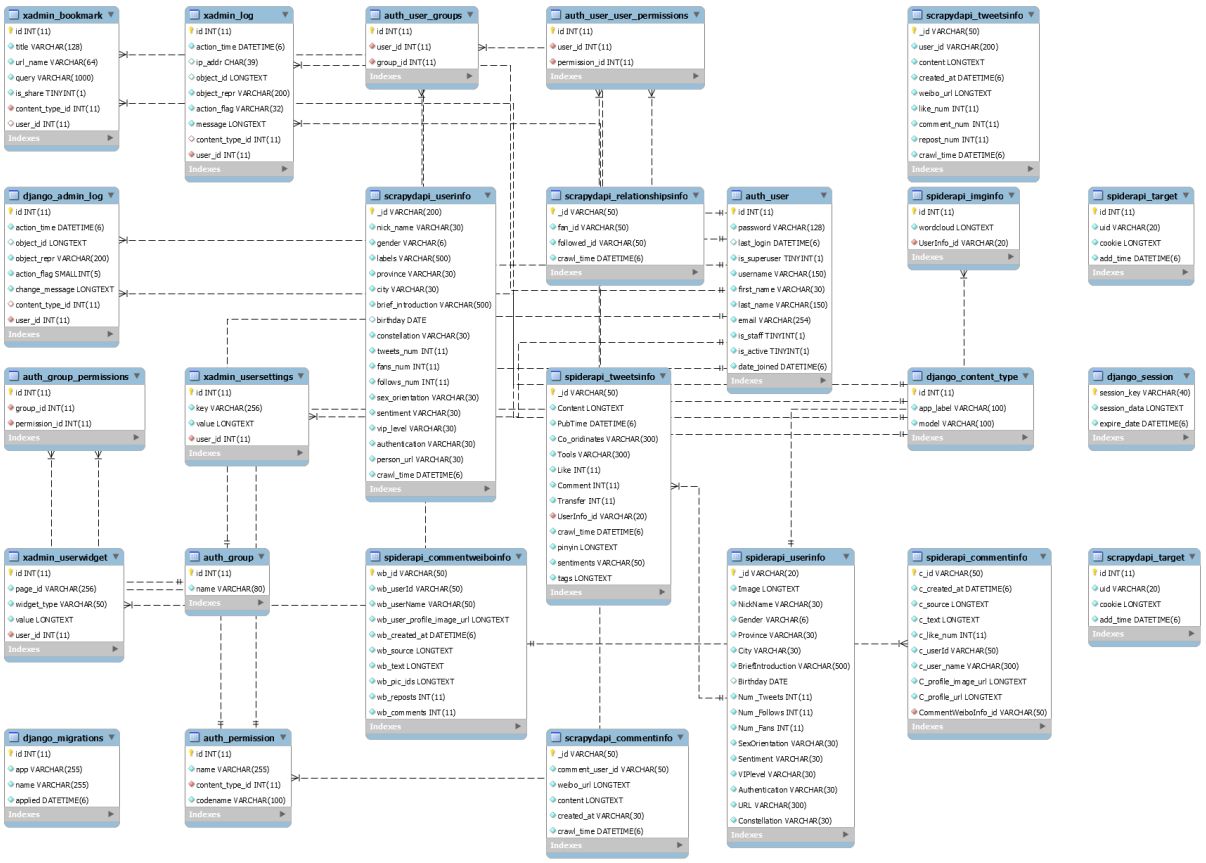

图 4.6 系统数据库 E-R 图

4.5.4 数据字典

数据字典表示的是数据库的数据项、数据结构、数据流和数据存储、处理逻辑等定义的描述，下路的所有表表示的是微博用户情感分析系统的各个表、表名、数据类型、非空情况、约束条件以及中文描述。

表 4.4 auth_group 表

表名	auth_group			
列名	数据类型	非空	约束条件	中文描述
id	int(11)	TRUE	主键	主键表示
name	varchar(80)	FALSE		名字

表 4.5 auth_group_ permissions 表

表名	auth_group_permissions			
列名	数据类型	非空	约束条件	中文描述
id group_id	int(11)	TRUE	主键	主键表示
id	int(11)	FALSE	外键	标识

permission_id	int(11)	FALSE		权限标识

表 4.6 auth_permission 表

表名	auth_permission			
列名	数据类型	非空	约束条件	中文描述
codename	varchar(100)	FALSE		权限代码名字
content_type_id	int(11)	FALSE		权限类型标识
id	int(11)	TRUE	主键	主键标识
name	varchar(255)	FALSE		名字

表 4.7 auth_user 表

表名	auth_user			
列名	数据类型	非空	约束条件	中文描述
id	int(11)	TRUE	主键	主键标识
password	varchar (128)	FALSE		密码
last_login	datetime(6)	FALSE		最后一次登录
is_superuser	varchar(80)	FALSE		超级用户
username	varchar(80)	FALSE		用户名
first_name	varchar(80)	FALSE		名
last_name	varchar(80)	FALSE		姓
email	varchar(80)	FALSE		邮件
is_staff	varchar(80)	FALSE		管理员
is_active	varchar(80)	FALSE		活跃
date_joined	datetime (6)	FALSE		创建时间

表 4.8 auth_user_group 表

表名	auth_group			
列名	数据类型	非空	约束条件	中文描述
id	int(11)	TRUE	主键	标识
user_id	int(11)	TRUE		用户标识
group_id	int(11)	TRUE		组标识

表 4.9 auth_user_user_permissions 表

表名	auth_user_user_permissions			
列名	数据类型	非空	约束条件	中文描述
id	int(11)	TRUE	主键	标识
user_id	int(11)	TRUE		用户标识

permission_id	int(11)	TRUE		权限标识

表 4.10 django_admin_log 表

表名	django_admin_log			
列名	数据类型	非空	约束条件	中文描述
id	int(11)	TRUE	主键	主键标识
action_time	datetime (6)	FALSE		活跃时间
object_id	longtext	FALSE		对象标识
object_repr	varchar(200)	FALSE		对象
action_flag	smallint(5)	FALSE		标志
change_message	longtext	FALSE		改变信息
content_type_id	int(11)	TRUE		文本标识
user_id	int(11)	FALSE	外键	用户标识

表 4.11 django content_type 表

表名	django_content_type			
列名	数据类型	非空	约束条件	中文描述
id	int(11)	TRUE	主键	主键标识
app_label	varchar(80)	FALSE		应用标签
model	varchar(80)	FALSE		模型层

表 4.12 django_migrations 表

表名	django_migrations			
列名	数据类型	非空	约束条件	中文描述
id	int(11)	TRUE	主键	主键标识
app	varchar(80)	FALSE		应用名称
name	varchar(80)	FALSE		姓名
applied	datetime(6)	FALSE		时间

表 4.13 django_session 表

表名	django_session			
列名	数据类型	非空	约束条件	中文描述
session_key	int(11)	TRUE	主键	主键标识
session_data	varchar(80)	FALSE		缓存数据
expire_date	varchar(80)	FALSE		到期时间

表 4.14 scrapydapi_commentinfo 表

表名	scrapydapi_commentinfo

列名	数据类型	非空	约束条件	中文描述
_id	varchar (50)	TRUE	主键	主键标识
comment_user_id	varchar(80)	FALSE		评论用户标识
weibo_url	longtext	FALSE		微博主页
content	longtext	FALSE		微博内容
created_at	varchar (80)	FALSE		微博创建时间
crawl_time	datetime(6)	FALSE		加入系统时间

表 4.15 scrapydapi_relationshipinfo 表

表名	scrapydapi_relationshipinfo			
列名	数据类型	非空	约束条件	中文描述
_id	varchar (50)	TRUE	主键	主键标识
fan_id	varchar(80)	FALSE		粉丝标识
followed_id	varchar(80)	FALSE		关注人标识
crawl_time	datetime(6)	FALSE		加入系统时间

表 4.16 scrapydapi_target 表

表名	scrapydapi_target			
列名	数据类型	非空	约束条件	中文描述
_id	int (11)	TRUE	主键	主键标识
uid	varchar(80)	FALSE		微博标识
cookie	longtext	FALSE		Cookie 标识
add_time	datetime(6)	FALSE		加入系统时间

表 4.17 scrapydapi_tweetsinfo 表

表名	scrapydapi_tweetsinfo			
列名	数据类型	非空	约束条件	中文描述
_id	varchar (50)	TRUE	主键	主键标识
user_id	varchar(80)	FALSE		用户标识
content	longtext	FALSE		微博内容
created_at	longtext	FALSE		微博创建时间
weibo_url	varchar (80)	FALSE		微博主页
like_num	int(11)	FALSE		点赞
comment_num	int(11)	FALSE		评论
repost_num	int(11)	FALSE		转发
crawl_time	datetime(6)	FALSE		加入系统时间

表 4.18 scrapydapi_userinfo 表

表名	scrapydapi_userinfo			
列名	数据类型	非空	约束条件	中文描述
_id	varchar (50)	TRUE	主键	主键标识
nick_name	varchar(80)	FALSE		用户昵称
gender	varchar (80)	FALSE		用户性别
labels	varchar (80)	FALSE		用户标签
province	varchar (80)	FALSE		用户所在省份
city	varchar (80)	FALSE		用户所在城市
brief_introduction	varchar (80)	FALSE		用户简介
birthday	varchar (80)	FALSE		生日
constellation	varchar (80)	FALSE		星座
tweets_num	int(11)	FALSE		微博数
fans_num	int(11)	FALSE		粉丝数
follows_num	int(11)	FALSE		关注数
sex_orientation	varchar (80)	FALSE		性取向
sentiment	varchar (80)	FALSE		情感状态
vip_level	varchar (80)	FALSE		会员等级
authentication	varchar (80)	FALSE		认证
person_url	varchar (80)	FALSE		主页
crawl_time	datetime(6)	FALSE		加入系统时间

表 4.19 scrapyapi_commentinfo 表

表名	scrapyapi_commentinfo			
列名	数据类型	非空	约束条件	中文描述
c_id	varchar (50)	TRUE	主键	主键标识
c_created_at	datetime(6)	FALSE		创建时间
c_source	longtext	FALSE		微博来源
c_text	longtext	FALSE		微博内容
c_like_num	varchar (80)	FALSE		微博点赞
c_userId	varchar (80)	FALSE		微博用户标识
c_user_name	varchar (80)	FALSE		微博用户名
C_profile_image_url	varchar (80)	FALSE		微博头像

C_profile_url	varchar (80)	FALSE		微博主页
CommentWeiboInfo_id	varchar (80)	FALSE	外键	微博评论标识

表 4.20 scrapyapi_commentweiboinfo 表

表名	scrapyapi_commentweiboinfo			
列名	数据类型	非空	约束条件	中文描述
wb_id	varchar (50)	TRUE	主键	微博标识
wb_userId	varchar(80)	FALSE		微博用户标识
wb_userName	varchar(80)	FALSE		微博用户名
wb_user_profile_image_url	varchar(80)	FALSE		微博头像
wb_created_at	datetime(6)	FALSE		微博创建时间
wb_source	longtext	FALSE		微博来源
wb_text	longtext	FALSE		微博文本
wb_pic_ids	longtext	FALSE		微博图片来源
wb_reposts	int(11)	FALSE		微博转发
wb_comments	int(11)	FALSE		微博评论

表 4.21 scrapyapi_imginfo 表

表名	scrapyapi_imginfo			
列名	数据类型	非空	约束条件	中文描述
id	int (11)	TRUE	主键	主键标识
wordcloud	varchar(80)	FALSE		词云
UserInfo_id	longtext	FALSE	外键	用户标识

表 4.22 scrapyapi_target 表

表名	scrapyapi_target			
列名	数据类型	非空	约束条件	中文描述
id	int (11)	TRUE	主键	主键标识
uid	varchar(80)	FALSE		用户标识
Cookie	longtext	FALSE		Cookie 标识
Add_time	datetime(6)	FALSE		添加时间

表 4.23 scrapyapi_tweetsinfo 表

表名	scrapyapi_tweetsinfo			
列名	数据类型	非空	约束条件	中文描述
_id	varchar (50)	TRUE	主键	主键标识
Content	longtext	FALSE		微博内容
PubTime	datetime(6)	FALSE		微博发表时间
Co_oridinates	varchar (80)	FALSE		微博定位
Tools	varchar (80)	FALSE		微博发布工具
Like	int(11)	FALSE		微博点赞
Comment	int(11)	FALSE		微博评论
Transfer	int(11)	FALSE		微博转发
UserInfo_id	varchar (80)	FALSE	外键	用户 id
crawl_time	datetime(6)	FALSE		加入系统时间
pinyin	varchar (80)	FALSE		词性
sentiments	varchar (80)	FALSE		情感状态
tags	varchar (80)	FALSE		关键词

表 4.24 scrapyapi_userinfo 表

表名	scrapyapi_userinfo			
列名	数据类型	非空	约束条件	中文描述
_id	varchar (50)	TRUE	主键	主键标识
Image	longtext	FALSE		头像
nick_name	varchar(80)	FALSE		用户昵称
gender	varchar (80)	FALSE		用户性别
labels	varchar (80)	FALSE		用户标签
province	varchar (80)	FALSE		用户所在省份
city	varchar (80)	FALSE		用户所在城市
brief_introduction	varchar (80)	FALSE		用户简介
birthday	varchar (80)	FALSE		生日
constellation	varchar (80)	FALSE		星座
tweets_num	int(11)	FALSE		微博数
fans_num	int(11)	FALSE		粉丝数
follows_num	int(11)	FALSE		关注数
sex_orientation	varchar (80)	FALSE		性取向

sentiment	varchar (80)	FALSE		情感状态
vip_level	varchar (80)	FALSE		会员等级
authentication	varchar (80)	FALSE		认证
person_url	varchar (80)	FALSE		主页
crawl_time	datetime(6)	FALSE		加入系统时间

表 4.25 xadmin_bookmark 表

表名	xadmin_bookmark			
列名	数据类型	非空	约束条件	中文描述
id	int (11)	TRUE	主键	主键标识
title	varchar(80)	FALSE		词头
url_name	varchar(80)	FALSE		姓名地址
query	varchar(80)	FALSE		查询
is_share	int (11)	FALSE		是否分析
content_type_id	int (11)	FALSE		内容类型标识
user_id	int (11)	FALSE	外键	用户标识

表 4.26 xadmin_log 表

表名	xadmin_bookmark			
列名	数据类型	非空	约束条件	中文描述
id	int (11)	TRUE	主键	主键标识
action_time	datetime(6)	FALSE		活动时间
ip_addr	varchar(80)	FALSE		ip 地址
object_id	varchar(80)	FALSE		对象标识
object_repr	varchar(80)	FALSE		对象转发
action_flag	varchar(80)	FALSE		活动标志
message	varchar(80)	FALSE		信息
content_type_id	int (11)	FALSE		内容类型标识
user_id	int (11)	FALSE	外键	用户标识

表 4.27 xadmin_usersettings 表

表名	xadmin_usersettings			
列名	数据类型	非空	约束条件	中文描述
id	int (11)	TRUE	主键	主键标识
key	vatchar(60)	FALSE		密钥
value	varchar(80)	FALSE		值

user_id	int(11)	FALSE		对象标识

表 4.28 xadmin_userwidget 表

表名	xadmin_userwidget			
列名	数据类型	非空	约束条件	中文描述
id	int (11)	TRUE	主键	主键标识
page_id	vatchar(60)	FALSE		页码
widget_type	varchar(80)	FALSE		部件类型
value	varchar(80)	FALSE		值
user_id	int(11)	FALSE		对象标识

4.6 本章小结

本章节主要是描述系统的概要设计，系统的概要设计主要是根据交互过程以及用户的需求来形成相互的框架和视觉框架的过程，其在系统的设计和研究之间无缝结合，为后续微博用户情感分析系统的开发提供了流程参考。

5 系统详细设计

本章节将会从程序开发视图、微博信息的获取与清理、中文文本情感分析、系统详细模块来阐述微博用户情感分析系统的详细设计。

5.1 系统开发视图

本系统采用 Django 作为后端开发系统，Vue 作为前端系统，实现前后端分离的开发模式，后端的 Django 主要分为三层：Model 层、View 层、Template 层，Model 带有 ORM（object relational mapping）模型，当用户发出请求的时候，django 的 url 层接收，view 层开始处理数据，然后通过 model 层和数据库交互。具体的开发视图如下图 5.1 所示：

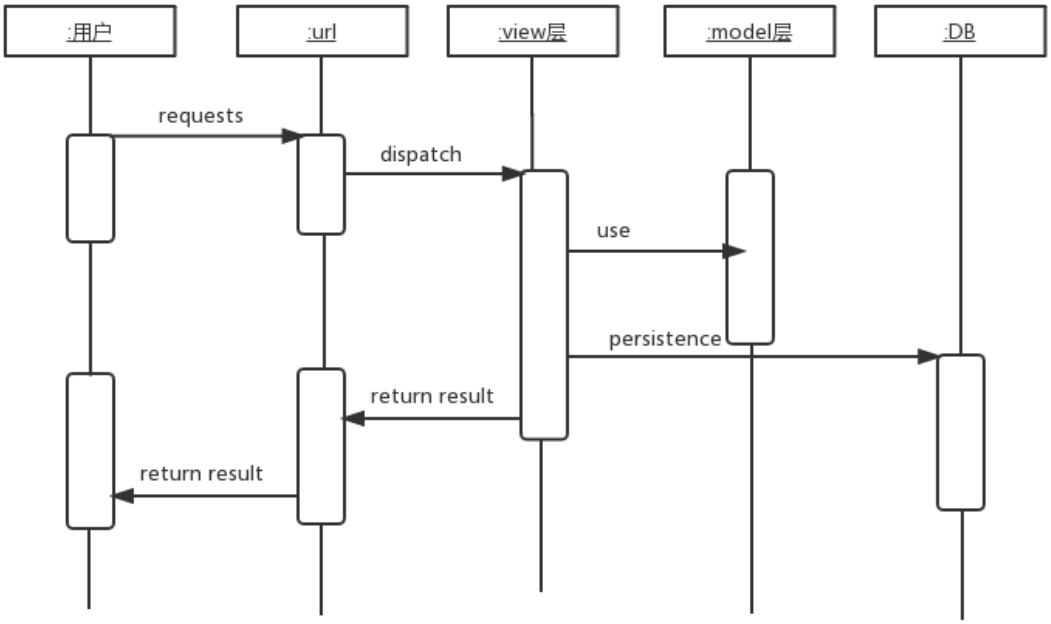

图 5.1 系统开发视图

5.2 系统逻辑架构图

如图 5.2 详细介绍了微博用户情感分析系统的逻辑架构图。从获取微博爬虫的数据到接口的接入。纵向介绍了本系统的逻辑。

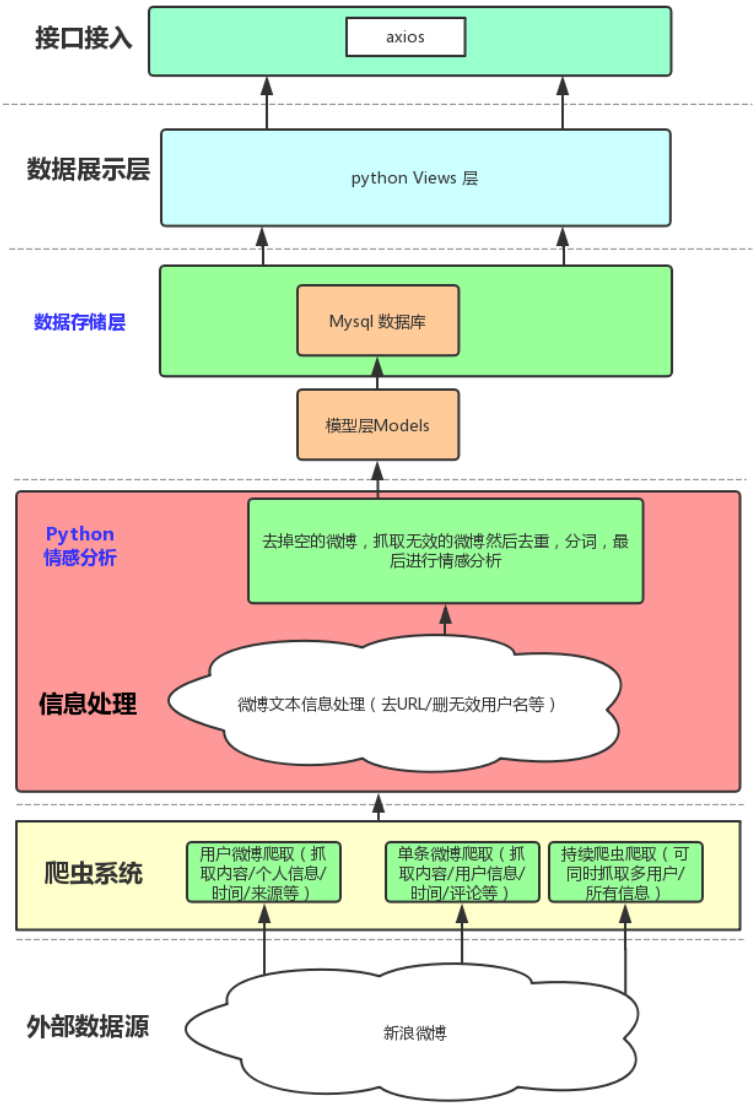

图 5.2 系统逻辑架构图

5.3 微博信息的获取与清理

从上述项目开发的意义中谈到微博已然成为社交平台非常突出的新媒体。每天在新浪微博上都会产生大量的数据，而且这些数据都是开放的，新浪微博俨然就像一个流动的大型数据库，同时作为开放的平台，新浪微博也以特定的方式来授权开发者可以获得一定量的微博数据，让开发者可以分析、处理这些数据，开发者可以深入研究这些数据，并且还开放了商业合作的权限。

但是针对新浪微博数据的获取方式，目前主要有两种方式：

（1）第一种是使用新浪微博官方提供的 API 接口，这些接口主要是基于 REST 实现的 HTTP 协议，主要以 JSON 的格式返回。但是获取这些数据也不是随便可以得到的，

需要一定的权限，同时接口的请求频率也有一定的限制，甚至对于接口有速率的限制。

（2）第二种则是基于网页的解析。对于这种方式解析网页，存在着一定的缺陷，如果网页的代码发生改变，相对应的解析代码也需要改变，对于大量的信息抓取，还需要破解新浪微博的反爬虫机制，可以通过使用代理 ip，增加 header，或者使用不同账号等。

如果使用第一种方式来获取微博的信息，那么首先需要去新浪微博开发平台注册账号，同时我们能得到的数据有限，只能拿到个人用户信息，连个人用户的微博都得不到，对于评论，只能得到前 200 页，所以第一种方式限制太多，不符合我们的开发需求。因此本系统采用第二种方式获取新浪微博的网页数据。如果采用第二种方式，那么我们就需要解决新浪微博的反爬虫问题。

5.3.1 反爬虫机制

理解什么是爬虫，我们才能知道什么是反爬虫。在上述的法律可行性曾经说到过，爬虫就是模拟用户的行为点击网页，因为是模拟的，所以可以在短时间内快速点击大量网页，对于恶意的爬虫，疯狂点击会造成微博的服务器压力过大，所以新浪微博为了解决这个问题，就会设置反爬虫机制。

为了防止爬虫，一般会从三个方面入手：第一个就是分析网页请求的 headers，用来监督用户访问网站的行为；第二个方式则是调整网站之中的目录和数据的加载方式；第三种则是应用 AJAX 技术来反爬虫。一般来说，前两种的方式是比较常见的，大部分网站都会从前两种方式的角度来进行反爬虫。

5.3.1.1 通过 Headers 反爬虫

目前网页一般的反爬虫机制是检测请求网页中的 HEADERS 和 USER-AGENT，有些严格的还会检查网站的 REFERER。如果遇到这类反爬虫机制的网页，我们可以在代码中添加 HEADERS 和 REFERER 来绕过此类检查。因此对于这些网站只要添加了 HEADERS 就能比较好的绕过。

5.3.1.2 基于用户行为的反爬虫

当用户访问网站的时候，用户的行为是目前主流网站的常用检测手段，比如：一个 IP 在短时间内访问一个网页很多次；或者是同一个账号在短时间内进行了多次的相同的操作。对于这类型的情况，我们可以通过代理 IP 来解决。目前网上 IP 代理有很多收费的或者免费的。如果需要，我们可以把一些免费 IP 代理存储起来，然后请求的时候每隔一段时间更换一次 IP。

5.3.1.3 动态页面的反爬虫

在上述微博的反爬虫机制，我们说过有一些网站会从 AJAX 的角度反爬虫，这些网

站的数据通过 AJAX 的请求生成，或者是通过 JS 的请求生成。我们不好拿到这些数据，但是如果我们能在网站的请求之中，找到请求的规律，分析其中请求接口的含义，调用这个请求，那么也可以得到相对于的数据。如果我们是在找不到请求的接口的规律含义，那么我们可以调用 selenium+phantomjs 的组合框架，调用其浏览器内核来模拟人的操作来触发 ajax 的请求或者 js 请求从而得到数据。

5.3.2 微博的反爬虫机制

上述说了三种反爬虫的机制，那么新浪微博是使用哪一种爬虫机制呢？经过验证，新浪微博是使用了上述的三种反爬虫机制。新浪微博首先验证客户端 HEADERS，同时对访问量大的 IP 进行了禁止访问的设置，同时也有使用 AJAX 进行数据传输。要破解上述的微博爬虫，我们可以使用 IP 代理，添加 HEADERS，对于同一账号不同时间进行访问。

5.3.3 微博的信息的获取

微博用户情感分析系统则是通过添加 HEADS 如下：User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.119 Safari/537.36 然后利用一个账号的 Cookie 登录网页得到数据，每次请求间断 3 秒的方式来解决新浪微博的反爬虫机制。

5.3.3.1 通过 Cookie 登录

如果要获取新浪微博的数据，那么我们需要登录新浪微博才能查看别人数据，既然要登录，那么就有两种方法：第一种就是使用代码访问登录然后得到返回的数据，第二种就是利用 Cookie 的方式直接登录。如果使用第一种方法，那么我们会存在一个验证码的问题，当然我们也可以调用图片打码的 API 接口来获取数据，但是还是过于麻烦，我们的目的是为了得到新浪微博的数据，当然是越快越好，所以在这里，我们采用第二种方法获取新浪微博的数据。如图 5.3 所示：

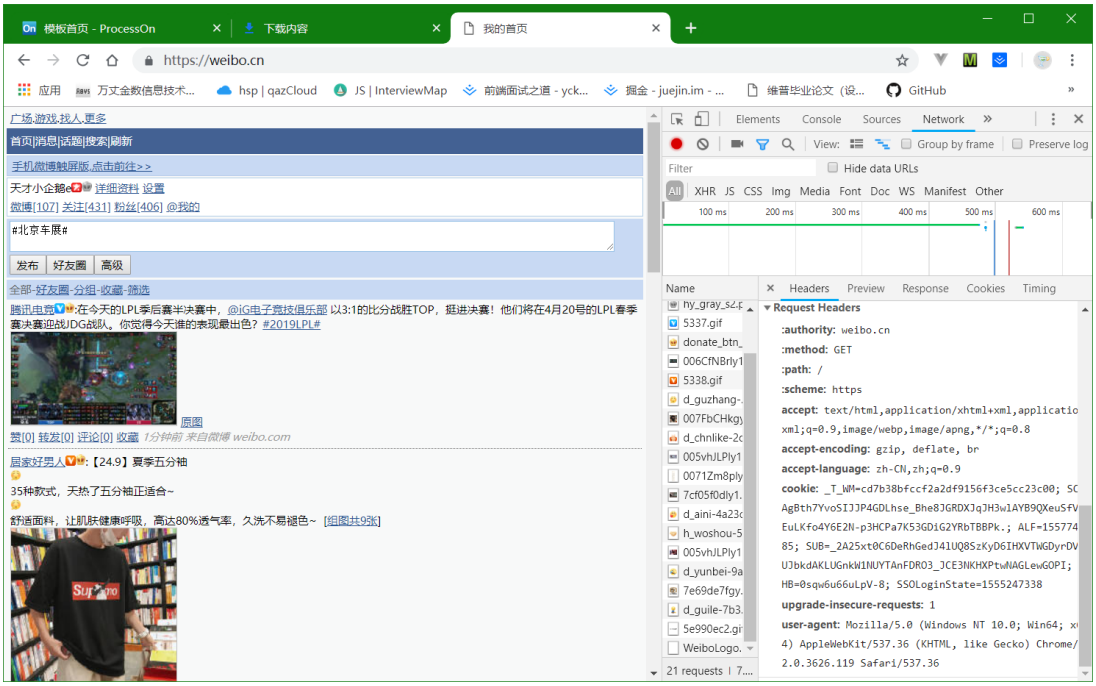

图 5.3 获取 Cookie

我们打开 weibo.cn 的网站，找到 Network，然后就登录了一个账号就可以得到 Cookie，我们只需要复制这段 Cookie 就可以在后续登录了。

5.3.3.2 通过 requests 抓取和存储

我们在上面说了如何登录了新浪微博，既然登录方法有了，那么就可以抓取数据了。

```
html = requests.get(url, cookies=self.cookie, headers=self.agent).content
selector = etree.HTML(html)
```

图 5.4 获取新浪微博网页

我们用这个图 5.4 的方法即可得到新浪微博的一个页面。然后就可以进行分析：

```
selector = etree.HTML(html)
info = ";".join(selector.xpath('body/div[@class="c"]//text()'))
# 获取信息
nickname = re.findall('昵称[：:]?(.*?);', info)
image = selector.xpath('body/div[@class="c"]//img/@src')
gender = re.findall('性别[：:]?(.*?);', info)
place = re.findall('地区[：:]?(.*?);', info)
briefIntroduction = re.findall('简介[：:]?(.*?);', info)
birthday = re.findall('生日[：:]?(.*?);', info)
sexOrientation = re.findall('性取向[：:]?(.*?);', info)
sentiment = re.findall('感情状况[：:]?(.*?);', info)
vipLevel = re.findall('会员等级[：:]?(.*?);', info)
authentication = re.findall('认证[：:]?(.*?);', info)
url = re.findall('互联网[：:]?(.*?);', info)
```

图 5.5 解析个人信息

如图 5.5 是截取了获取用户信息的代码，这样就能得到一个页面的用户信息并存入数据库了，可以看到本系统是通过 ORM 的方式操作数据库的，在此之前，还需要在 Models 页面定义数据，然后把上述的数据保存入 Models，保存方法如图 5.6。

```
#实例化
user_info = UserInfo()
user_info._id = self.user_id
if image:
    user_info.Image = image
if nickname and nickname[0]:
    user_info.NickName = nickname[0].replace(u"\xa0", "")
if gender and gender[0]:
    user_info.Gender = gender[0].replace(u"\xa0", "")
if place and place[0]:
    place = place[0].replace(u"\xa0", "").split(" ")
    user_info.Province = place[0]
    if len(place) > 1:
        user_info.City = place[1]
if briefIntroduction and briefIntroduction[0]:
    user_info.BriefIntroduction = briefIntroduction[0].replace(u"\xa0", "")
if birthday and birthday[0]:
    try:
        birthday = datetime.datetime.strptime(birthday[0], "%Y-%m-%d")
        user_info.Birthday = birthday - datetime.timedelta(hours=8)
    except Exception:
        user_info.Constellation = birthday[0]   # 有可能是星座，而非时间
if sexOrientation and sexOrientation[0]:
    if sexOrientation[0].replace(u"\xa0", "") == gender[0]:
        user_info.SexOrientation = "同性恋"
    else:
        user_info.SexOrientation = "异性恋"
if sentiment and sentiment[0]:
    user_info.Sentiment = sentiment[0].replace(u"\xa0", "")
if vipLevel and vipLevel[0]:
    user_info.VIPlevel = vipLevel[0].replace(u"\xa0", "")
if authentication and authentication[0]:
    user_info.Authentication = authentication[0].replace(u"\xa0", "")
if url:
    user_info.URL = url[0]
```

图 5.6 保存爬虫数据到 Models

至此，以抓取用户数据为例。就详细说明了从 requests 抓取网页数据，并且解析的数据，把保存到数据库的过程。

5.3.3.3 通过 Scrapyd 抓取和存储

Scrapy 是一个开源的专门用来爬虫的框架，因为其系统化的设置，可以节省很多操作，好处是可以便捷实现持续爬虫。但是也存在克服的难点，那就是把数据保存到 Django 的模型之中，同时把 Scrapy 部署在 Scrapyd 之上，并且通过 HTTP 的请求来启动爬虫。

Scrapyd 爬虫首先也是需要设置代理以及 Cookie，当代理和 Cookie 设置好了以后，在设置爬虫的事件间隔，一般是三秒的延迟时间就可以了，但是下载延迟需要设置 16 秒。然后是爬取的网页 Request，需要说的是，这里的 Request 跟上面的 requests 不一样，这里的 request 是经过 scrapy 的封装的，可以便捷实现更多需要的爬虫功能。

Request(url="https://weibo.cn/%s/info" % uid, callback=self.parse_information)

通过此方式我们进入爬虫的网页，然后解析这个网页的内容即可，解析方式如上图 5.5，但是写法不同，写法带有 Scrapy 的框架特色，基本原理是类似的，这里不再重复描述。问题是得到的数据如何存入 Django，因为这是两个不同的框架，部署在不同的服务器，要使得 Scrapy 的数据存入 Django 的 Models，需要在 Scrapy 的 Setting 设置引入 Django 的环境，然后再 Item 里建立联系就可以把 scrapy 爬虫的数据保存到 django 的 models，然后保存到数据库了。

再接着就是把部署到 scrapyd 上了，这个直接放上去即可，然后在 django 的 views 通过 http 启动即可。至此，我们就实现了 Scrapyd 持续爬虫并存储功能。

5.3.4 微博的分词与降噪

微博的分词和降噪这一内容下面将从五个方面来阐述：概念、中文分词、删除 URL、删除用户名、去除停用词。

5.3.4.1 概念

微博是一个社交平台，那么就拥有迅速传播的特点，我们平时看的热门新闻，基本上是第一时间从微博获取的，所以也成为了商家们重点开发的平台，在该平台上发布自家的广告，所以会存在大量的营销和广告。给微博情感分析造成了很大的困难。因此文本预处理在本系统显得很重要。

5.3.4.2 中文分词

中文分词就是把汉字变成一个独立的、有意义的词汇。文本挖掘首先以中文分词为前提。当前我们常用的中文分词工具有以下几种：

（1）NLPIR：中国科学院计算机研究所的一个产品、据说是世界上最好的中文分词工具。

（2）BosonNLP：性能强大的中文语言分词工具。

（3）结巴分词：github 开源的中文分词工具。

（4）IKAnalyzer：开源的 java 分词工具。

微博用户情感分析系统主要使用 SnowNLP 自带的分词工具，基于 Character-Based Generative Mode 算法实现的，其匹配程序像是 n-gram 和 TnT 的综合，也能达到不错的效果。因此采用此分词开发本系统，也能得到一个还算满意的结果。

5.3.4.3 删除 URL

使用分词如果遇到 URL，那么会把 URL 单独分开，但是我们一般看到的垃圾微博，都会看到一段文字之后接着一段链接。而且一般情况下，这个链接都是很长的字符串，如果要保持原来的链接，那么就会占用微博的资源，所以微博会转成短链。但是对于情感分析，这个短链可用的信息很少，一般就作为广告的导向和用户的定位。

由此可见，URL 在微博之中在进行情感分析的时候，要适当删除一定的 URL，对微博进行适当的清理，去除这些无用的 URL，从而降低 URL 对情感分析影响。

5.3.4.4 删除用户名

微博的用户名，一般是用来标识一个用户，而且大部分的用户名都是毫无规律，而且毫无意义的，比如：@上海羊毛衫只需九块九、@baby 永远像 18 岁的天空那样湛蓝、@rjeiorj_____pcyyyyyy 等，这些用户名对于分词来说，还是会有一定的影响的。因此我们为了减少构建词性特称造成的影响，我们需要适当删除用户名，也是极其重要的。

5.3.4.5 去除停用词

停用词指的是对于文本中表达意思起不了多大作用的词。在系统的 SEO 中，为了提高检索速度以及减少存储，也会适当忽略这些停用词。

停用词一般程度上，我们可以理解为过滤词，但是过滤词的范围太大了，通常是我们常见的包含色情、政治、暴力等敏感词汇。而停用词则是不一样，范围就小很多了。通常情况，停用词只分为两类：

（1）类似于“我”、“你”这些机会在很多地方都会出现，无法保证准确的搜索结果，还会降低效率。

（2）在文档之中使用频率很高的词，但是实际意义却不大，主要是包括语气助词、副词、介词等，这类词在文本表达中没有出现任何的意义，因此为了提高情感分析的准确性，我们需要适当去掉停用词。

5.4 微博文本情感分析

接下来从三个方面阐述微博的情感分析，分别是：情感分析的概念、基于 SnowNLP 的中文情感分析、利用 SnowNLP 训练情感分析模型。

5.4.1 情感分析的概念

情感分析：又称倾向性分析或者是意见挖掘。通常用带有感情色彩的词语来对文本进行判断、分析、处理、归纳、推理等流程。例如：我们在淘宝上买东西，可以通过看商品的评论来推断一件商品的好坏；我们可以在豆瓣上看影评，以及用户的评分来分析某部电影的好看程度；我们可以从播放量以及用户的评价中推断一首歌的受欢迎程度。

目前来说，情感分析一般采用机器学习来分类，我们一般看的电影的影评，可以用来标注，然后进行机器学习，使用机器学习来分类，最终构建出一个情感分类器，以后用户可以从这个情感分类器来对其他电影预测。

但是新浪微博这个开放的数据库拥有着基于互联网这种庞大的文本，如果我们对整个互联网的文本来标注，这显然是不可能的，我们只能用少量的微博文本人工标注，然后进行机器学习，构建出一个情感分类库。

目前来说，机器学习对于情感分析的分类制约还是存在的，因为中文的博大精深，一个词汇可以表达多个意思，存在着多种情感，所以还不能保证情感的准确度。需要构建出一个情感字典，然后在微博文本之中捕捉关键词，然后准确计算出情感倾向。本系统使用的 SnowNLP 中文情感分析正式基于这个原理。

5.4.2 基于 SnowNLP 的中文情感分析

SnowNLP 是一个利用 python 写的类库，在 Github 上开源，目前拥有者 3957 个 Star，这个开源库可以方便处理中文的内容，并且自带有训练好的词典，但是自带的词典是基于购物类词汇的，因此在后续中，我们需要构建自己的情感分析模型。

5.4.2.1 开源库 SnowNLP 的源码解读

SnowNLP 主要提供了 11 个接口，分别是：分词、提取句子、繁体字转简体字、中文转拼音、情感分析、词性分析、字数统、词频、逆文本频率、归纳、提取关键词。由于本文主要是获取情感分析，所以在此处重点说明情感分析原理。

查阅源码可知，SnowNLP 首先是调用 load 的方法，用于加载训练好的数据字典，然后调用 classify 的方法，在 classify 的方法中调用 Bayes 的方法进行情感分析，实际的原理就是利用朴素贝叶斯算法进行情感分析。

5.4.2.2 朴素贝叶斯算法进行情感分析

朴素贝叶斯算法，是数据挖掘领域经典算法之一。朴素的意思是：特征条件独立；贝叶斯则是基于贝叶斯定理。贝叶斯定理是一个拥有悠久历史的定理，因其有着坚实的理论基础，贝叶斯不仅在处理很多问题时高效直接，而且在很多高级语言里面，也有用它的模型演化。而朴素贝叶斯算法是一个监督学习的模型，实现简单，基于贝叶斯定理作为支撑，所以在样本量大的情况下的表现会比较好。朴素贝叶斯方法作为自然语言处理的切入口，其流程如图 5.7 所示：

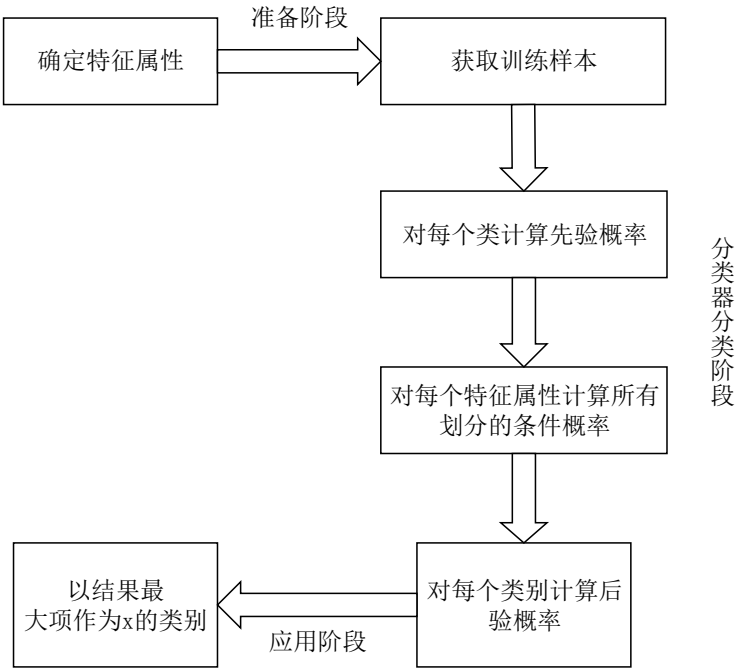

图 5.7 使用朴素贝叶斯定理进行情感分析

因贝叶斯定理作为数学的一个重要知识，从高中简单的数学概率问题就开始出现，它的公式如下：

$$P\left(\frac{Y}{X}\right)=\frac{P\left(\frac{X}{Y}\right)P(Y)}{P(X)} \quad \text{（式 5.1）}$$

其实就是我们高中接触的联合概率公式：

$$P(Y,X)=P(Y|X)P(X)=P(X|Y)P(Y) \quad \text{（式 5.2）}$$

其中 P(Y)叫先验概率，$P(\frac{Y}{X})$为后验概率，P(Y,X)一般称之为联合概率。

下面用本科生的角度举例来阐述贝叶斯定理如何进行情感分析：

从公式入手 P（Y）是指事件发生的概率，同理 P（X）是指 X 事件发生的概率。而 P（X|Y）是指 Y 发生后 X 发生的概率，同理 P（Y|X）是指 X 发生后 Y 发生的概率。了解了这个之后，我们来看一个例子：

早上，某医院来了六个病人。分别是：

（1）职业：警察；症状：打喷嚏；疾病：感冒。

（2）职业：程序员；症状：打喷嚏；疾病：过敏。

（3）职业：舞蹈演员；症状：头痛；疾病：脑震荡。

（4）职业：舞蹈演员；症状：头痛；疾病：感冒。

（5）职业：设计师；症状：打喷嚏；疾病：感冒。

（6）职业：设计师；症状：头痛；疾病：脑震荡。

问现在来了第七个病人，是一个打喷嚏的舞蹈演员。请问她患上感冒的概率是多少。根据贝叶斯定理解答如下：

P(感冒|打喷嚏 x 舞蹈演员) = P(打喷嚏 x 舞蹈演员|感冒) x P(感冒) / P(打喷嚏 x 舞蹈演员)

假定”打喷嚏”和” 舞蹈演员”这两个特征是独立的：

P(感冒|打喷嚏 x 舞蹈演员) = P(打喷嚏|感冒) x P(舞蹈演员|感冒) x P(感冒) / P(打喷嚏) x P(舞蹈演员) = 0.66 x 0.33 x 0.5 / 0.5 x 0.33= 0.66。

好了这样我们就算出了打喷嚏的舞蹈演员感冒的概率是 66%，这就是贝叶斯定理。

SnowNLP 的朴素贝叶斯算法流程执行过程是这样的：

（1）首先人工标记样本

（2）对于不同的分类样本进行中文分词

（3）去除样本垃圾词条

（4）整理后的词条作为特征组，分析词条频率

（5）根据词条的信息，计算先验概率

（6）读取训练的样本，分词降噪，然后形成样本特征

（7）测试样本带入朴素贝叶斯公式，计算先验概率和后验概率，得到情感概率值

了解清楚了这些，到这一步我们就清楚了 SnowNLP 进行情感分析的原理了。下面就要训练基于微博数据的情感分析模型。

5.4.3 训练微博数据情感分析模型

SnowNLP 的训练方式是遍历一个积极和消极的文本，给每条评论进行分词，同时加上分类标签得到训练模型。所以训练首先就是要获得数据，通过收集网上的 500 万条微博数据之后得到一个 2.34G 的.sql 数据库文件，然后是将这个数据库文件导入数据库，然后读取数据库分析文件，把情感概率大于 0.8 的放在积极的文本中，把情感概率小于 0.3 的文本，放在消极文本中。然后开始训练。训练流程图如下：

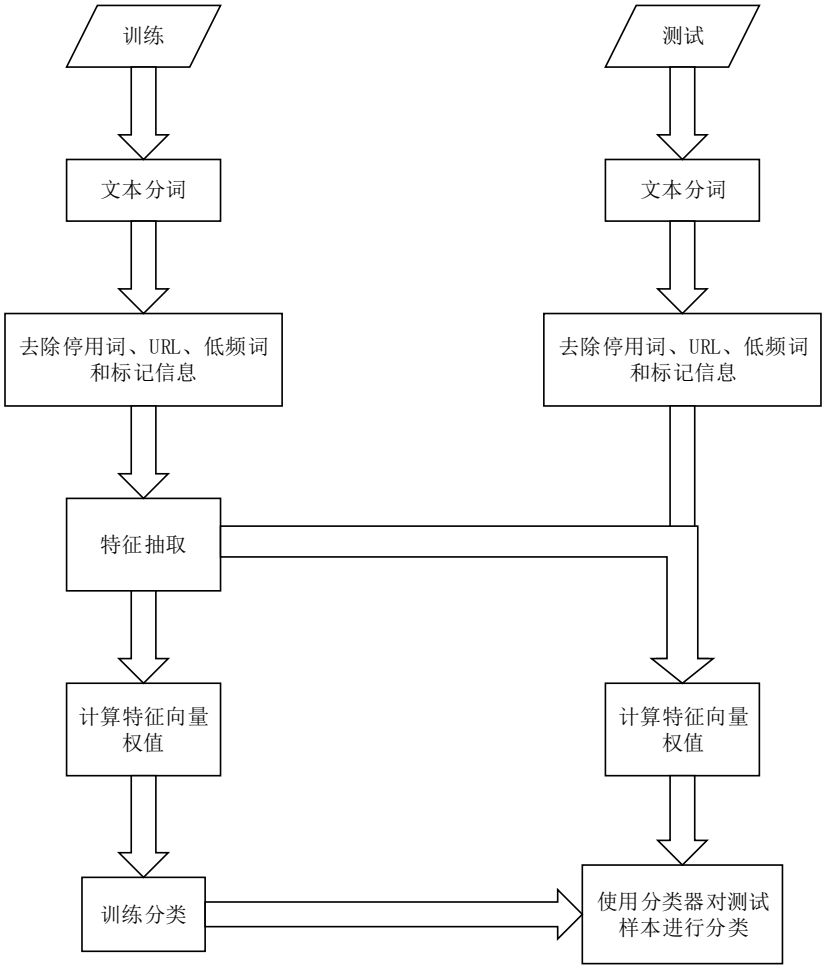

图 5.8 情感分析模型训练流程图

为了保证训练的数据有效并且准确率高，经过 500 万条数据三轮的测试，每次训练平均历时 48 小时，得到约 6 万条积极的文本和约 6 万条的消极文本，然后在对积极分本和消极分本就行训练两轮。总共进行了 5 轮的训练，最终得到训练模型 sentiment.marshal。

5.4.3.1 微博数据情感分析模型的准确率

经过上述的操作，最终得到了一个情感分析模型，但是得到模型还不能说是完事了，接下来就是要测试这个模型的准确率，随机从数据库拿出 300 条数据，然后人工甄别是否符合情感分析倾向值。经过 300 条数据的人工判断，然后比较情感分析模型测出的数值，最终的得到的结果如表 5.1 所示：

表 5.1 测试准确率统计结果

统计项	结果
随机微博条数	300
人工判断正确条数	238

5.5 系统详细模块

下面将从挑两个模块来说明系统的详细设计，分别是：微博用户爬虫模块，文本情感分析模块说明。

5.5.1 微博用户爬虫模块

当用户进入网站首页，将会进入如图 5.9 的页面：

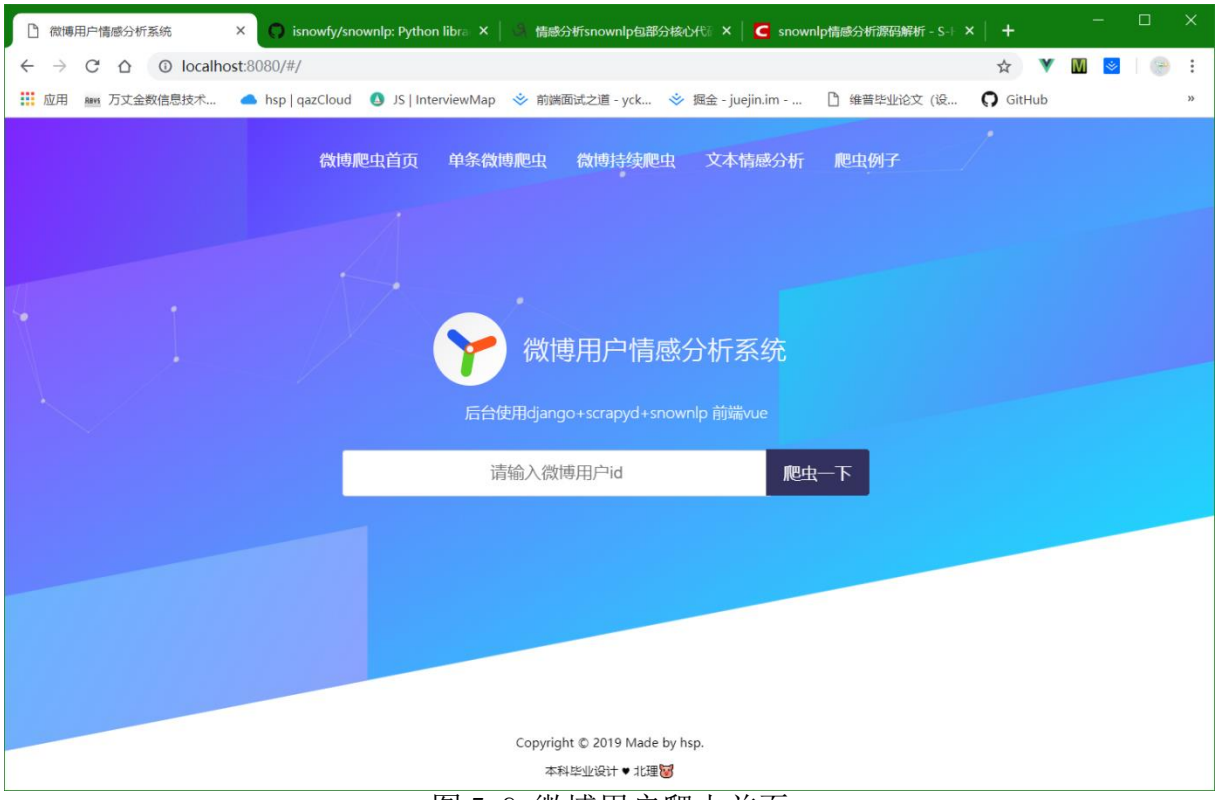

图 5.9 微博用户爬虫首页

用户输入微博用户 id，如果输入错误，那将会不进行爬虫，如果输入正确，那么将会启动爬虫，然后展示如图 5.10 和图 5.11 的界面。

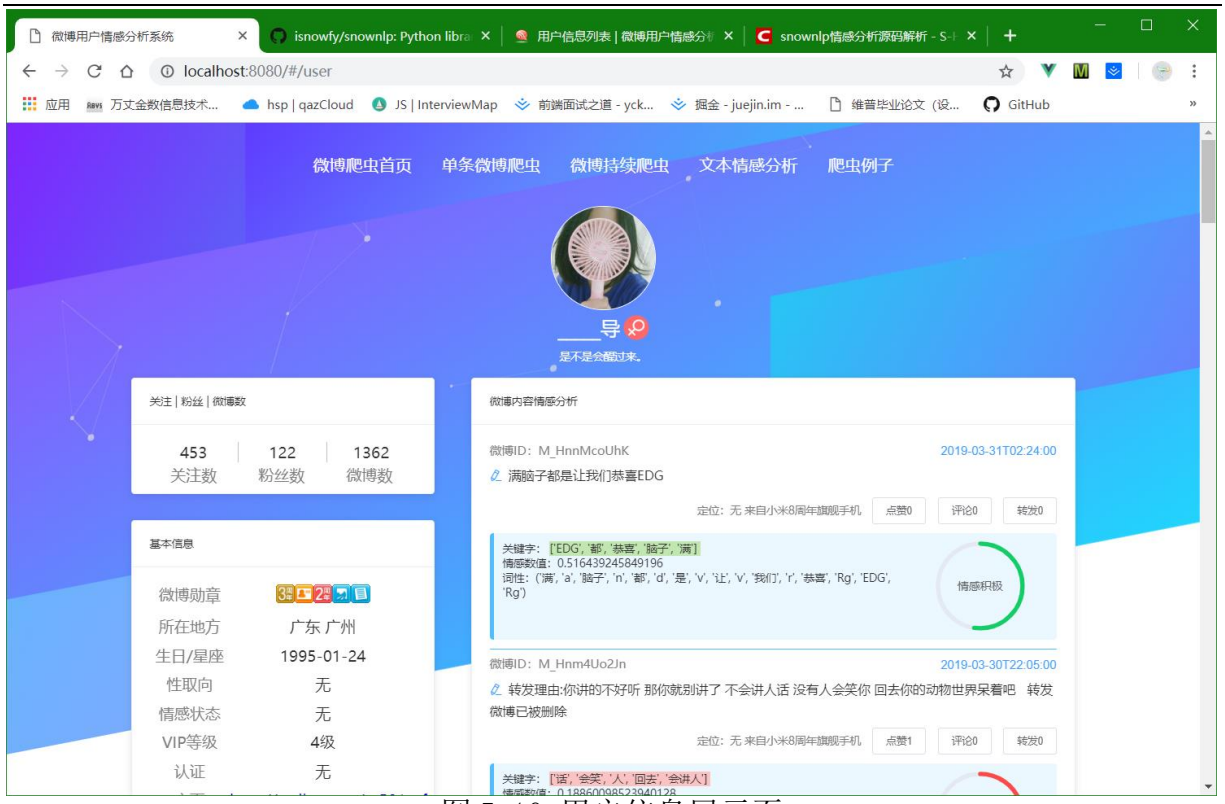

图 5.10 用户信息展示页

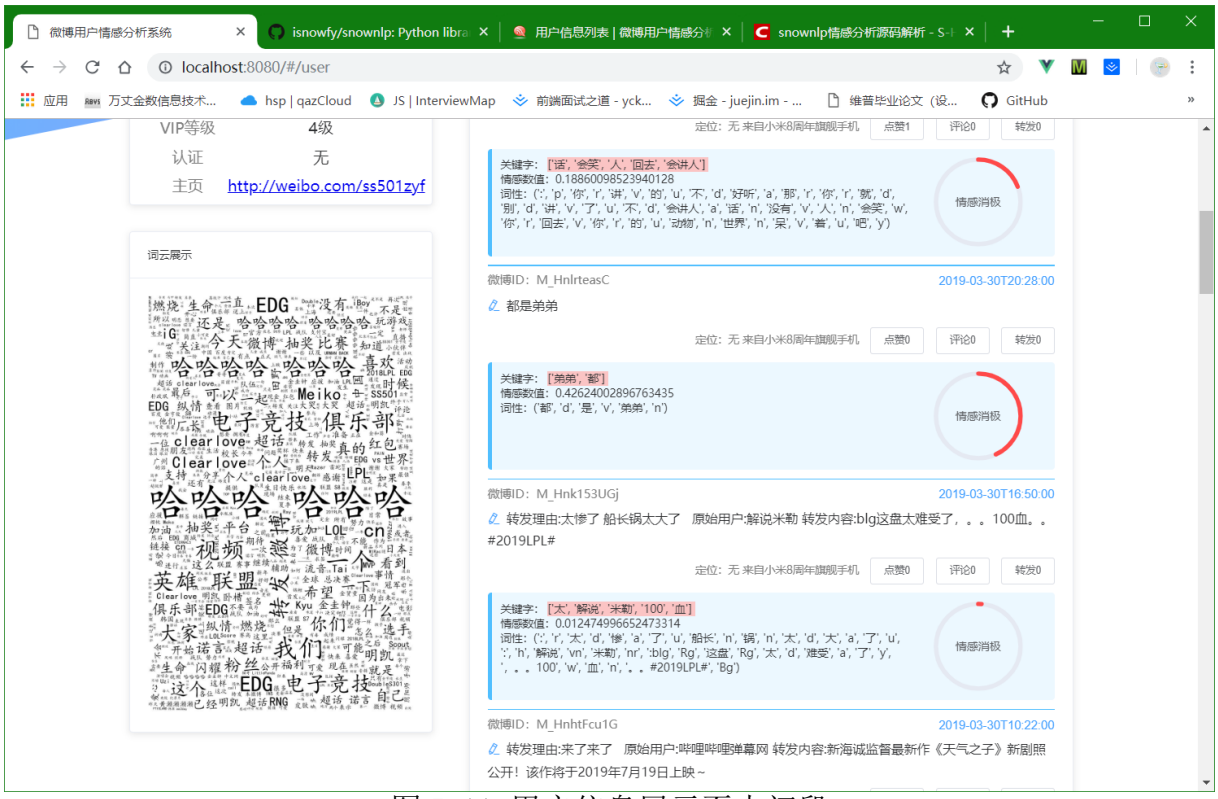

图 5.11 用户信息展示页中间段

当信息超过一定 20 条的时候，将会采用分页的方式重新请求数据。后台系统的数据可以下载，后台系统的界面如图 5.12 所示：

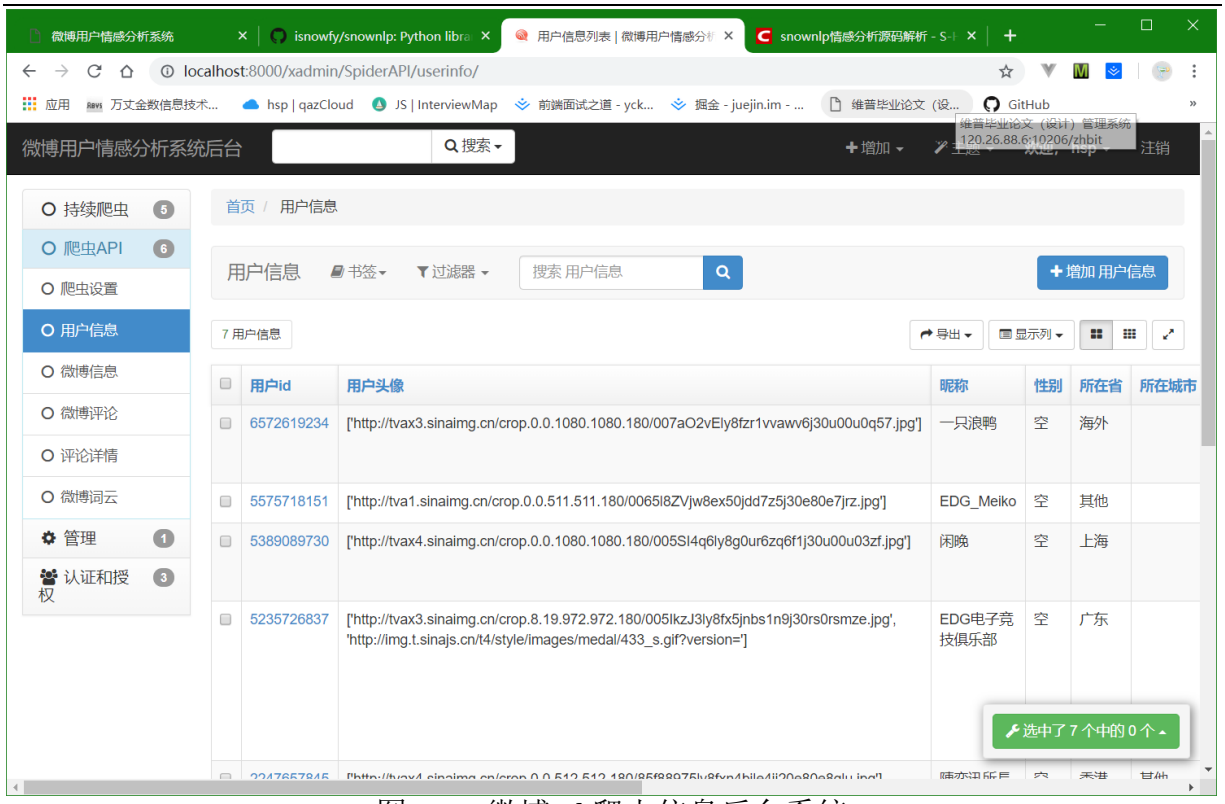

图 5.12 微博 id 爬虫信息后台系统

5.5.2 情感分析模块

当用户进入网站首页，点击文本情感分析，将会进入如图 5.13 的页面，在该页面只要输入任意一段文字，就可以如下所示，展示关键词，词频统计，以及情感分析数值，这里的情感分析数值采用了百分号的单位进行转换，0-50%表示消极，50%-100%表示积极。当用户的情感是消极的时候，圈圈的颜色将会是红色，当用户的情感是积极的时候，圈圈的颜色将会变成绿色。

此外，用户还可以随机一段系统自带的文字来查看情感分析值。右侧还有关闭按钮用于清空文本的内容。

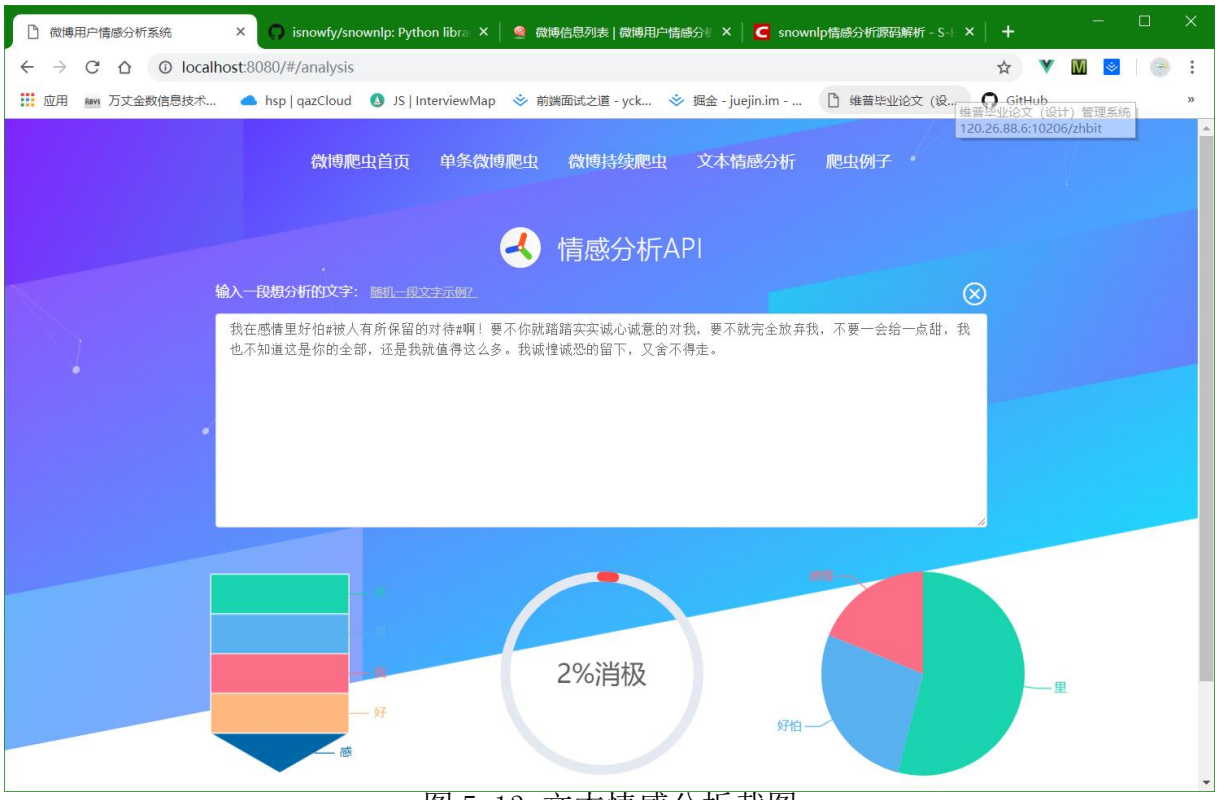

图 5.13 文本情感分析截图

6 系统测试

为了确定软件的质量，期望软件达成了期待的目标，必须对每一个软件进行测试，以及时发现软件潜在的问题，最后保证系统的正常运行。本章节将会从五个方面介绍系统测试，分别是：测试环境、测试方案与策略、测试准则、测试风险以及测试用例。

6.1 测试环境

测试环境分为两个分别是服务端的测试环境以及用户端的测试环境。

6.1.1 服务端测试环境

服务端测试环境如表 6.1 所示

表 6.1 服务端测试环境

服务器	
硬件	Cpu:Intel(R)Core(TM)i7-4710MQ 硬盘：GLOWAY WAR PRO T300 12G ATA Device 内存：8G
软件	操作系统：Windows 10 家庭版 64 位 浏览器：chrome 应用软件 Visual Studio Code1.32、MYSQL 8.0、WSGi
网络环境	校园网

6.1.2 用户端测试环境

用户端测试环境如表 6.2 所示

表 6.2 用户端测试环境

服务器	
硬件	Cpu:Intel(R)Core(TM)i7-4710MQ 硬盘：GLOWAY WAR PRO T300 12G ATA Device 内存：8G
软件	操作系统：Windows 10 家庭版 64 位 浏览器：chrome 应用软件 Node6.0、NPM5.51、Vue-cli2.5
网络环境	校园网

6.2 测试方案与策略

本系统的测试方案如表 6.3 所示

表 6.3 测试方案

测试功能模块	所采用的黑盒白盒用例设计方法
后台管理员登陆登出模块	等价类划分法
后台管理员修改密码	等价类划分法
后台录入用户微博 id	场景分析法

后台录入用户微博 Cookie	场景分析法
后台 Scrapy 爬虫录入微博 id	场景分析法
后台 Scrapy 爬虫录入微博 Cookie	场景分析法
后台导出 Eecel	场景分析法
后台搜索信息	场景分析法
后台添加信息	场景分析法
后台删除信息	场景分析法
微博用户爬虫模块	场景分析法
单条评论爬虫模块	场景分析法
持续爬虫模块	场景分析法
中文情感分析模块	场景分析法
已爬虫用户模块	场景分析法

6.3 测试准则

测试的准则分为开始测试准则和结束则是准则。

6.3.1 开始测试准则

开始测试：系统的模块功能开发完成。

6.3.2 结束测试准则

结束测试的准则如下：

（1）所写的测试用例必须全部被执行。

（2）严重级别为致命的，或者严重的缺陷必须全部解决掉，一般的缺陷和微小的缺陷和允许在 0-10%内。

（3）完成测试报告。

6.4 测试风险

测试风险如表 6.4 所示：

表 6.4 测试风险

风险名称	优先级	应对措施
没有详细设计说明书	3	在开发阶段对设计和需求进行分析，对模块功能进行分类，分析业务逻辑。
需求变更开发	1	建议将需求变更形成文档，对没有文档的需求变更，在测试过程中发现及时沟通变更
软件资源	2	Django2.0 的版本之后与 1.9 的版本存在巨大差别，开发软件版本不一致，导致冲突，需要整合相应的版本进行项目开发

6.5 测试用例

为了完成特殊目标而编制的一组数据，包含了测试输入，执行条件以及预期的结果

测试方案。在编写测试用例的时候，尽可能的在有限的测试用例之中发现比较多的缺陷。微博用户情感分析系统的测试用例如下。

6.5.1 后台管理员登录登出测试用例

表 6.5 登陆测试用例

测试用例编号	用户名	密码	预期结果	覆盖等价类
1	123456	123456	登陆成功	1、4
2	STUDENT	STUDENT	登陆成功	2、5
3	123456	STUDENT	登陆成功	1、5
4	STUDENT	123456	登陆成功	2、4
5	空	123456	登陆失败	7、4
6	123456	空	登陆成功	1、8
7	特殊字符	123456	登陆失败	3、4
8	STUDENT	特殊字符	登陆成功	2、6

6.5.2 后台管理员修改密码测试用例

表 6.6 修改密码测试用例

测试用例编号	密码	预期结果	覆盖等价类
1	123456	修改成功	1
2	STUDENT	修改成功	2
6	空	修改成功	4
8	特殊字符	修改成功	3

6.5.3 后台系统录入用户微博 id 测试用例

表 6.7 后台系统录入用户微博 id 测试用例

测试步骤	场景	测试步骤	预期结果	实际结果
STEP1	进入登录页面	输入正确登录密码	登陆成功	与预期结果一致

STEP2	进入选项卡	在选项卡点击爬虫 API	弹出次选项	与预期结果一致
STEP3	进入次选项卡	在次选项卡点击爬虫设置	进入爬虫设置页面	与预期结果一致
STEP4	爬虫设置页面	点击爬虫设置，输入爬虫用户，并保存	保存成功	与预期结果一致

6.5.4 后台系统录入用户微博 Cookie 测试用例

表 6.8 后台系统录入用户微博 Cookie 测试用例

测试步骤	场景	测试步骤	预期结果	实际结果
STEP1	进入登录页面	输入正确登录密码	登陆成功	与预期结果一致
STEP2	进入选项卡	在选项卡点击爬虫 API	弹出次选项	与预期结果一致
STEP3	进入次选项卡	在次选项卡点击爬虫设置	进入爬虫设置页面	与预期结果一致
STEP4	爬虫设置页面	点击爬虫设置，输入爬虫 COOKIE，并保存	保存成功	与预期结果一致

6.5.5 后台 Scrapy 爬虫录入微博 id 测试用例

表 6.9 后台 Scrapy 爬虫录入用户微博 id 测试用例

测试步骤	场景	测试步骤	预期结果	实际结果
STEP1	进入登录页面	输入正确登录密码	登陆成功	与预期结果一致
STEP2	进入选项卡	在选项卡点击持续爬虫	弹出次选项	与预期结果一致
STEP3	进入次选项卡	在次选项卡点击爬虫初始	进入爬虫设置页面	与预期结果一致
STEP4	爬虫设置页面	点击爬虫设置,输入爬虫用户，并保存	保存成功	与预期结果一致

6.5.6 后台 Scrapy 爬虫录入微博 Cookie 测试用例

表 6.10 后台 Scrapy 爬虫录入微博 Cookie 测试用例

测试步骤	场景	测试步骤	预期结果	实际结果
STEP1	进入登录页面	输入正确登录密码	登陆成功	与预期结果一致
STEP2	进入选项卡	在选项卡点击持续爬虫	弹出次选项	与预期结果一致
STEP3	进入次选项卡	在次选项卡点击爬虫初始	进入爬虫设置页面	与预期结果一致
STEP4	爬虫设置页面	点击爬虫设置，输入爬虫 COOKIE，并保存	保存成功	与预期结果一致

6.5.7 后台导出 excel 测试用例

表 6.11 后台导出 excel 测试用例

测试步骤	场景	测试步骤	预期结果	实际结果
STEP1	进入登录页面	输入正确登录密码	登陆成功	与预期结果一致
STEP2	进入选项卡	在选项卡点击持续爬虫	弹出次选项	与预期结果一致
STEP3	进入次选项卡	在次选项卡点击微博信息	进入爬虫设置页面	与预期结果一致
STEP4	微博页面	点击导出，选择导出 EXCEL	导出成功	与预期结果一致

6.5.8 后台搜索信息测试用例

表 6.12 后台搜索信息测试用例

测试步骤	场景	测试步骤	预期结果	实际结果
STEP1	进入登录页面	输入正确登录密码	登陆成功	与预期结果一致
STEP2	进入选项卡	在选项卡点击持续爬虫	弹出次选项	与预期结果一致
STEP3	进入次选项卡	在次选项卡点击微博信息	进入爬虫设置页面	与预期结果一致

STEP4	微博页面	输入用户 ID:5445610951	得到搜索结果	与预期结果一致

6.5.9 后台添加信息测试用例

表 6.13 后台添加信息测试用例

测试步骤	场景	测试步骤	预期结果	实际结果
STEP1	进入登录页面	输入正确登录密码	登陆成功	与预期结果一致
STEP2	进入选项卡	在选项卡点击持续爬虫	弹出次选项	与预期结果一致
STEP3	进入次选项卡	在次选项卡点击微博信息	进入爬虫设置页面	与预期结果一致
STEP4	微博页面	右侧点击添加信息	进入添加信息页面	与预期结果一致
STEP5	添加信息页面	填写添加信息，并保存	添加信息成功	与预期结果一致

6.5.10 后台删除信息测试用例

表 6.14 后台删除信息测试用例

测试步骤	场景	测试步骤	预期结果	实际结果
STEP1	进入登录页面	输入正确登录密码	登陆成功	与预期结果一致
STEP2	进入选项卡	在选项卡点击持续爬虫	弹出次选项	与预期结果一致
STEP3	进入次选项卡	在次选项卡点击微博信息	进入爬虫设置页面	与预期结果一致
STEP4	微博页面	选中删除信息，并点击删除	弹出确认删除页面	与预期结果一致
STEP5	确认删除页面	点击确认删除	删除成功	与预期结果一致

6.5.11 微博用户爬虫测试用例

表 6.15 微博用户爬虫测试用例

测试步骤	场景	测试步骤	预期结果	实际结果
STEP1	进入用户端页面	输入正确微博用户	进入用户信息页面	与预期结果一致
STEP2	用户信息页面	用户信息页面展示用户微博内容/用户信息/情感分析结果	显示成功	与预期结果一致

6.5.12 单条评论爬虫测试用例

表 6.16 单条评论爬虫测试用例

测试步骤	场景	测试步骤	预期结果	实际结果
STEP1	进入用户端页面	输入正确微博内容 ID	进入微博评论页面	与预期结果一致
STEP2	微博评论页面	微博评论页面展示微博内容/用户评论/情感分析结果	显示成功	与预期结果一致

6.5.13 持续爬虫测试用例

表 6.17 持续爬虫测试用例

测试步骤	场景	测试步骤	预期结果	实际结果
STEP1	进入用户端页面	输入正确微博内容 ID 以及 COOKIE	进入 SCRAPYD 爬虫页面	与预期结果一致
STEP2	SCRAPYD 爬虫页面	SCRAPYD 正在爬虫	爬虫成功	与预期结果一致

6.5.14 中文情感分析测试用例

表 6.18 中文情感分析测试用例

测试步骤	场景	测试步骤	预期结果	实际结果
STEP1	进入用户端页面	点击文本情感分析	显示文本情感分析页面	与预期结果一致
STEP2	进入文本情感分析页面	输入任意一段中文文本	得到情感分析结果/词频/关键词	与预期结果一致

6.5.15 已爬虫用户测试用例

表 6.19 已爬虫用户测试用例

测试步骤	场景	测试步骤	预期结果	实际结果
STEP1	进入用户端页面	点击爬虫例子	显示已爬虫页面	与预期结果一致
STEP2	进入已爬虫页面	查看所有已爬虫用户	得到所有已爬虫用户	与预期结果一致

6.6 本章小结

测试对于软件开发流程来说，有着重要的意义，是发现程序错误的重要手段，为后续提高软件的产品质量，验证软件是否满足需求提供了强有力的保障。同时有助于开发人员更精确的发现问题，为未来的合理工作提出合理的规划。

7 总结与展望

微博已然成为热点事件的始发地，针对微博的数据挖掘具有很广阔的前景，同时也有很重要的研究价值所在。随着互联网的快速发展，以后我们还会有更多的不规则、不规范的文本出现。也会有越来越多的网络流行词，中英结合的文字等。而本系统主要是结合了传统的文本分析方法，以及针对微博的特点做了一些处理，然后使用朴素贝叶斯进行情感分析。总体上把微博分为了两类：积极类和消极类。

7.1 本文主要内容总结

毕业设计作品的全称叫微博用户情感分析系统的设计与实现。很显然，重点在爬虫以及情感分析。本文主要使用 python 的语言进行爬虫，使用了 requests+etree 的组合以及 scrapy 的框架进行爬虫，增加了反爬虫的一些设置。在情感分析里面，根据微博的特点，适当对微博降噪处理。然后使用 SnowNLP 的带有的朴素贝叶斯算法对微博文本进行情感分析，最后得出情感分析数值，在前端界面展示。

7.2 存在问题及未来展望

在微博不断发展的情况下，本系统存在着很多方面需要进一步开展的研究。下面从两个大方面来说明：

第一个方面是爬虫的技术，爬虫是根据微博网页的 HTML 结构来进行解析文本的，如果微博更新了页面的结构，相对应的代码也需要更新。同时随着微博的发展，微博将会采用更先进的反爬虫技术，到时候爬虫这块也需要相对于的变化。

第二个方面就是情感分析方面，对于情感分析，本文采用的只是一个算法，针对于传统的文本，并且在数据量大的情况下，才会有一定的效果。因此还不完全的全面。同时情感分析还可以采用更好的情感分类策略，比如说更精确的分析用户的语言现象，分析程度副词进行综合建模等。除了情感分析的前提下，还可以根据热点事件的牵涉事件、地点、人物进行更深入的挖掘。从而做出更准确的预判。这样对社会的稳定性起到一定的作用。

综上所述，微博除了传统的文本，还加入了表情、图片、视频等众多因素，如果要深入研究，本系统做的还远远不够。同时微博海纳百川，不同领域又有不同的涵义，还需要针对不同领域采用不同的挖掘方法和策略，只有这样，才能够真正对社会产生巨大的价值。希望有一天，这一方面的挖掘能够对社会的稳定真正起到强大的作用。

参考文献

［1］中国网信网．第 42 次《中国互联网络发展状况统计报告》.

http://www.cac.gov.cn/2018-08/20/c_1123296882.htm,2018-8-20

［2］新浪科技．微博月活跃用户数.

https://weibo.com/1642634100/GtIy1uzIS,2018-8-8

［3］李彬，王丹，沙明瑞．基于 Django 的任务信息系统的设计[J].

科学技术创新, 2019, (5):70-71

［4］吴洁，朱小飞，张宜浩等．基于用户情感倾向感知的微博情感分析方法[J].

山东大学学报(理学版), 2019, (3):46-55.

［5］李彬，王丹，沙明瑞．基于 Django 的任务信息系统的设计[J].
科学技术创新, 2019, (5):70-71

［6］李勇敢，周学广，孙艳等．中文微博情感分析研究与实现*[J].
软件学报，2017，第 28 卷(12):3183-3205.

［7］S_H-A_N．SnowNLP 情感分析源码解析.

https://blog.csdn.net/lom9357bye/article/details/78565432，2017-11-17

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

［9］王珊．数据库系统概论 [M]．北京. 高等教育出版社，2010.

［10］钱乐秋，赵文耘，牛军钰．软件工程［M］．北京：清华大学出版社，2007.

［11］朱少民．软件测试方法和技术（第 2 版）[M]．北京：清华大学出版社，2010.

谢　辞

在项目开发背景说过，大学的毕业设计是一个很重要的最后一次实践，我本人非常希望能在大学搞点有意思的事情，无奈时光匆匆，大学四年平淡无奇。所以毕业设计想搞个有意思的系统，而不是我们常见的什么图书馆管理系统之类的系统，庆幸的是这次的毕业设计对我而言还算满意。因为真正的学到了很多新的知识，可以说在毕业设计之前，我对 python 的爬虫以及情感分析真的是一知半解的状态。当初选题的时候，我跟老师说了这个题目，但是我害怕自己最终不能完成这个系统。所以当时还犹豫过要不要改题这个问题，庆幸的是最后蔡培茂老师建议我选择这个系统，经过慎重的考虑，我还是选择开发微博用户情感分析系统。

这真的是一次磨练，虽然这个磨练的代价让我辞去已有的实习（但是我本身就打算离职的，因为在这个公司实在无法让我学习到更多的东西。）微博用户情感分析系统是我在接近半年内的时间挤出来的，虽然之前多少做过一点点，但是都是小样例，真正整合开发的时间是用了四到五个月。最后做的结果还算满意。虽然在本次毕业设计从每一个流程我都遇到各种各样的困难，比如部署到 Scrapyd 爬虫如何用 HTTP 启动、进行情感分析的原理是什么、Scrapy 怎么和 Django 的 model 联系在一起、Django 怎么和 Vue 进行前后端开发等等，但是，经过一步步的逐个解决问题，各个击破，最终将它们给解决了。

最后我要感谢蔡培茂老师对我们的认真指导、提醒，工作安排进度等通知到位，让我们清楚知道毕业设计的流程。感谢同学大学四年的关照。感谢辅导员认真细心的工作，站好了大学四年的最后一岗。

总而言之，此次的毕业设计让我收获很多，让我在接下来找工作和学习中，都有了一个不错的项目经验话题。

最后还要独立一段感谢张艺凡小可爱的陪伴，私はあなたを愛して。

附　录

附录 1

程序源代码

持续爬虫关键代码：

```
def parse_further_information(self, response):
    text = response.text
    information_item = response.meta['item']
    tweets_num = re.findall('微博\[(\d+)\]', text)
    if tweets_num:
        information_item['tweets_num'] = int(tweets_num[0])
    follows_num = re.findall('关注\[(\d+)\]', text)
    if follows_num:
        information_item['follows_num'] = int(follows_num[0])
    fans_num = re.findall('粉丝\[(\d+)\]', text)
    if fans_num:
        information_item['fans_num'] = int(fans_num[0])
    yield information_item

    # 获取该用户微博
    yield Request(url=self.base_url + '/{}/profile?page=1'.format(information_item['_id']),
                  callback=self.parse_tweet,
                  priority=1)

    # 获取关注列表
    yield Request(url=self.base_url + '/{}/follow?page=1'.format(information_item['_id']),
                  callback=self.parse_follow,
                  dont_filter=True)
    # 获取粉丝列表
    yield Request(url=self.base_url + '/{}/fans?page=1'.format(information_item['_id']),
                  callback=self.parse_fans,
                  dont_filter=True)
```

```
def parse_tweet(self, response):
    if response.url.endswith('page=1'):
        # 如果是第1页，一次性获取后面的所有页
        all_page = re.search(r'/> 1/(\d+)页</div>', response.text)
        if all_page:
            all_page = all_page.group(1)
            all_page = int(all_page)
            for page_num in range(2, all_page + 1):
                page_url = response.url.replace('page=1', 'page={}'.format(page_num))
                yield Request(page_url, self.parse_tweet, dont_filter=True, meta=response.meta)
```

```
    """
    解析本页的数据
    """
    tree_node = etree.HTML(response.body)
    tweet_nodes = tree_node.xpath('//div[@class="c" and @id]')
    for tweet_node in tweet_nodes:
        try:
            tweet_item = TweetsItem()
            tweet_item['crawl_time'] = datetime.now()
            tweet_repost_url = tweet_node.xpath('.//a[contains(text(),"转发[")]/@href')[0]
            user_tweet_id = re.search(r'/repost/(.*?)\?uid=(\d+)', tweet_repost_url)
            tweet_item['weibo_url'] = 'https://weibo.com/{}/{}'.format(user_tweet_id.group(2),
                                                                       user_tweet_id.group(1))
            tweet_item['user_id'] = user_tweet_id.group(2)
            tweet_item['_id'] = '{}_{}'.format(user_tweet_id.group(2), user_tweet_id.group(1))
            create_time_info = tweet_node.xpath('.//span[@class="ct"]/text()')[-1]
            if "来自" in create_time_info:
                tweet_item['created_at'] = time_fix(create_time_info.split('来自')[0].strip())
            else:
                tweet_item['created_at'] = time_fix(create_time_info.strip())

            like_num = tweet_node.xpath('.//a[contains(text(),"赞[")]/text()')[-1]
            tweet_item['like_num'] = int(re.search('\d+', like_num).group())

            repost_num = tweet_node.xpath('.//a[contains(text(),"转发[")]/text()')[-1]
            tweet_item['repost_num'] = int(re.search('\d+', repost_num).group())

            comment_num = tweet_node.xpath(
                './/a[contains(text(),"评论[") and not(contains(text(),"原文"))]/text()')[-1]
            tweet_item['comment_num'] = int(re.search('\d+', comment_num).group())

            tweet_content_node = tweet_node.xpath('.//span[@class="ctt"]')[0]
```

```
            # 检测由没有阅读全文:
            all_content_link = tweet_content_node.xpath('.//a[text()="全文"]')
            if all_content_link:
                all_content_url = self.base_url + all_content_link[0].xpath('./@href')[0]
                yield Request(all_content_url, callback=self.parse_all_content, meta={'item': tweet_item},
                              priority=1)

            else:
                all_content = tweet_content_node.xpath('string(.)').replace('\u200b', '').strip()
                tweet_item['content'] = all_content[1:]
                yield tweet_item

            # 抓取该微博的评论信息
            comment_url = self.base_url + '/comment/' + tweet_item['weibo_url'].split('/')[-1] + '?page=1'
            yield Request(url=comment_url, callback=self.parse_comment, meta={'weibo_url': tweet_item['weibo_url']})
```

```
def parse_all_content(self, response):
    # 有阅读全文的情况，获取全文
    tree_node = etree.HTML(response.body)
    tweet_item = response.meta['item']
    content_node = tree_node.xpath('//div[@id="M_"]//span[@class="ctt"]')[0]
    all_content = content_node.xpath('string(.)').replace('\u200b', '').strip()
    tweet_item['content'] = all_content[1:]
    yield tweet_item

def parse_follow(self, response):
    """
    抓取关注列表
    """
    # 如果是第1页，一次性获取后面的所有页
    if response.url.endswith('page=1'):
        all_page = re.search(r'/> 1/(\d+)页</div>', response.text)
        if all_page:
            all_page = all_page.group(1)
            all_page = int(all_page)
            for page_num in range(2, all_page + 1):
                page_url = response.url.replace('page=1', 'page={}'.format(page_num))
                yield Request(page_url, self.parse_follow, dont_filter=True, meta=response.meta)
    selector = Selector(response)
    urls = selector.xpath('//a[text()="关注他" or text()="关注她" or text()="取消关注"]/@href').extract()
    uids = re.findall('uid=(\d+)', ";".join(urls), re.S)
    ID = re.findall('(\d+)/follow', response.url)[0]
    for uid in uids:
        relationships_item = RelationshipsItem()
        relationships_item['crawl_time'] = datetime.now()
        relationships_item["fan_id"] = ID
        relationships_item["followed_id"] = uid
        relationships_item["_id"] = ID + '-' + uid
        yield relationships_item
```

```
def parse_fans(self, response):
    """
    抓取粉丝列表
    """
    # 如果是第1页，一次性获取后面的所有页
    if response.url.endswith('page=1'):
        all_page = re.search(r'/> 1/(\d+)页</div>', response.text)
        if all_page:
            all_page = all_page.group(1)
            all_page = int(all_page)
            for page_num in range(2, all_page + 1):
                page_url = response.url.replace('page=1', 'page={}'.format(page_num))
                yield Request(page_url, self.parse_fans, dont_filter=True, meta=response.meta)
    selector = Selector(response)
    urls = selector.xpath('//a[text()="关注他" or text()="关注她" or text()="移除"]/@href').extract()
    uids = re.findall('uid=(\d+)', ";".join(urls), re.S)
    ID = re.findall('(\d+)/fans', response.url)[0]
    for uid in uids:
        relationships_item = RelationshipsItem()
        relationships_item['crawl_time'] = datetime.now()
        relationships_item["fan_id"] = uid
        relationships_item["followed_id"] = ID
        relationships_item["_id"] = uid + '-' + ID
        yield relationships_item
```

```
def parse_comment(self, response):
    # 如果是第1页，一次性获取后面的所有页
    if response.url.endswith('page=1'):
        all_page = re.search(r'/> 1/(\d+)页</div>', response.text)
        if all_page:
            all_page = all_page.group(1)
            all_page = int(all_page)
            for page_num in range(2, all_page + 1):
                page_url = response.url.replace('page=1', 'page={}'.format(page_num))
                yield Request(page_url, self.parse_comment, dont_filter=True, meta=response.meta)
    selector = Selector(response)
    comment_nodes = selector.xpath('//div[@class="c" and contains(@id,"C_")]')
    for comment_node in comment_nodes:
        comment_user_url = comment_node.xpath('.//a[contains(@href,"/u/")]/@href').extract_first()
        if not comment_user_url:
            continue
        comment_item = CommentItem()
        comment_item['crawl_time'] = datetime.now()
        comment_item['weibo_url'] = response.meta['weibo_url']
        comment_item['comment_user_id'] = re.search(r'/u/(\d+)', comment_user_url).group(1)
        comment_item['content'] = comment_node.xpath('.//span[@class="ctt"]').xpath('string(.)').extract_first()
        comment_item['_id'] = comment_node.xpath('./@id').extract_first()
        created_at = comment_node.xpath('.//span[@class="ct"]/text()').extract_first()
        comment_item['created_at'] = time_fix(created_at.split('\xa0')[0])
        yield comment_item
```

附录 2

朴素贝叶斯进行情感分析：

```
class Bayes(object):

    def __init__(self):
        self.d = {}
        self.total = 0

    def save(self, fname, iszip=True):
        d = {}
        d['total'] = self.total
        d['d'] = {}
        for k, v in self.d.items():
            d['d'][k] = v.__dict__
        if sys.version_info[0] == 3:
            fname = fname + '.3'
        if not iszip:
            marshal.dump(d, open(fname, 'wb'))
        else:
            f = gzip.open(fname, 'wb')
            f.write(marshal.dumps(d))
            f.close()
```

```
def load(self, fname, iszip=True):
    if sys.version_info[0] == 3:
        fname = fname + '.3'
    if not iszip:
        d = marshal.load(open(fname, 'rb'))
    else:
        try:
            f = gzip.open(fname, 'rb')
            d = marshal.loads(f.read())
        except IOError:
            f = open(fname, 'rb')
            d = marshal.loads(f.read())
        f.close()
    self.total = d['total']
    self.d = {}
    for k, v in d['d'].items():
        self.d[k] = AddOneProb()
        self.d[k].__dict__ = v

def train(self, data):
    for d in data:
        c = d[1]
        if c not in self.d:
            self.d[c] = AddOneProb()
        for word in d[0]:
            self.d[c].add(word, 1)
    self.total = sum(map(lambda x: self.d[x].getsum(), self.d.keys()))
```

```
def classify(self, x):
    tmp = {}
    for k in self.d:
        tmp[k] = log(self.d[k].getsum()) - log(self.total)
        for word in x:
            tmp[k] += log(self.d[k].freq(word))
    ret, prob = 0, 0
    for k in self.d:
        now = 0
        try:
            for otherk in self.d:
                now += exp(tmp[otherk]-tmp[k])
            now = 1/now
        except OverflowError:
            now = 0
        if now > prob:
            ret, prob = k, now
    return (ret, prob)
```