XXXX 大 学

实习（实训）报告

名称 数据可视化实训

——B 站热门视频弹幕的分析系统

2022 年 12 月 26 日至 2022 年 12 月 31 日共 1 周

学院(部) 计算机工程学院

班　　级 21 大数据技术 1

姓　　名 张三丰

学　　号 21XXXXXX

学院(部)负责人 XXX

系　主　任 XXX

指 导 教 师 XXXXX

目录

1 前言

1.1 研究背景与意义

各类网络视频资源是互联网发展事业发展的基础，也是其娱乐、学习、创新创作等各项工作顺利开展的重要物质保障，近几年来，随着社会和经济的大力发展，国家对互联网建设的投入越来越多，致使互联网建设的资产规模也越来越大。通过数据分析和可视化的实现可以使 B 站视频数据情况更加直观地展现出来，实现数据共享，更好地发挥资产数据对互联网视频平台公司的决策支持作用。

互联网建设是时代进步，科技发展的产物，它丰富了人们的娱乐生活，同时也为企业提供了发展的机会。当前，视频创作产业有力促进了我国的经济增长。而 B 站作为互联网视频平台产业中增长最快的一部分，理应得到更多关注与引导，形成更加可持续发展的产业。B 站是在全球范围内极具影响力和知名度的视频平台项目。

1.2 本文研究内容

本文的研究是通过设计和实现一个基于 Flask 框架的实时分析 B 站热门视频数据的分析系统，用于数据可视化和分析热门视频的趋势，通过调用 B 站 API 接口来获取数据，以获 TOP20 热门视频的关键信息。然后对数据进行预处理，数据清洗与分析，将结果展示到网页前端。B 站热门视频数据的分析系统的前端是基于 BootSharp5 前端框架，摒弃了传统网页的繁杂的 CSS 的编写。

本课题的研究内容包括：

（1） 对 B 站 TOP20 热门视频进行分析，确定一些重点的元素与数据，然后获取 API 接口来获取数据。

（2） 利用 jieba 分词工具对弹幕内容与评论内容进行去除停用词与分词操作，对主要关键词进行分析与统计，构建出词云图。

（3） 利用 Flask 框架将所有功能封装成一套完善的系统，包含数据图表的展示，数据的存储功能，数据的情感分析功能。

1.3 本文组织结构

论文具体的章节安排情况如下：

第 1 章 前言。简单介绍了研究背景及意义，然后通过文献综述法分析数据可视化分析平台的发展现状。最后交代本文的研究内容和文章组织结构。

第 2 章 相关理论与技术。对系统的关键技术进行介绍，主要是对 Flask 框架于 Echarts 框架的详细介绍。

第 3 章 系统需求分析。从功能性需求和非功能性需求对 B 站弹幕可视化分析系统进行分析。

第 4 章 系统的具体设计。本章节先对开发环境进行简单的描述，然后介绍分析系统的主题框架，主要的功能模块设计。

第 5 章 系统的具体实现与测试。对 B 站弹幕可视化分析系统的各项主要的模块进行详细的实现，并给出具体的代码过程。

第 6 章 总结与展望。对论文的工作进行总结，并对工作中遇到的困难与系统功能上的不足提出一些改进的设想。

2 核心技术介绍

2.1 Python 语言介绍

Python 是一种解释型，面向对象的高级程序设计语言，功能强大，具有很多区别于其他语言的个性化特点。

(1) 语法简洁，易于上手，程序可读性强。

(2) 既支持面向过程的函数编程，也支持面向对象的抽象编程。

(3) 可移植性好，python 程序可以在任何安装解释器的环境中运行。

(4) 可扩展性好，程序可以集成如 C,C++,Java 等语言编写的代码，这样就可以让核心算法不公开，也可以通过内嵌的代码提高运行速度。

(5) 开源本质，使任何用户都有可能成为代码的改进者。

(6) Python 解释器提供了数百个内置库和函数库，开源社区的程序员们还在源源不断地贡献第三方函数库，几乎覆盖了计算机应用的各个领域。

(7) 提供了安全合理的异常退出机制。

Python 语言具有以下缺点。

(1) 由于 python 是解释型语言，因此运行速度稍慢。这里是指与 C 和 C++ 相比。Python 开发人员尽量避开不成熟或者不重要的优化。一些针对非重要部位的加快运行速度的补丁通常不会被合并到 Python 内，若对速度有特殊要求的话，可考虑用 C++改写关键代码。

(2) 构架选择太多，没有像 C#那样的官方 .NET 构架。

2.2 Flask 框架介绍

2.2.1 Flask 框架的发展

受 SinatraRuby 框架启发，基于 Werkzeug 和 Jinja2，Flask 框架由此诞生，它由 Armin Ronacher 在 pocoo 开发。它可在 BSD 许可下获取。与大多数 Python 框架相比，虽然 Flask 相当年轻，但它具有很好的前景，并且已经在 PythonWeb 开发人员中流行起来。

2.2.2 Flask 框架的特点

Flask 的设计易于使用和扩展。它的初衷是为各种复杂的 Web 应用程序构建坚实的基础。从那时起，您就可以自由地插入任何扩展。您也可以自由构建自己的模块。Flask 适合各种项目。它对原型设计特别有用。Flask 依赖于两个外部库：Jinja2 模板引擎和 Werkzeug WSGI 工具包。Flask 框架具有以下特点：

(1) 内置开发服务器和快速调试器

(2) 集成支持单元测试

(3) RESTful 可请求调度

(4) Jinja2 模板

(5) 支持安全 cookie（客户端会话）

(6) 符合 WSGI1.0

(7) 基于 Unicode

因此本项目将利用 Flask 框架来设计一套前后端系统，来帮助 Echarts 图表更好的呈现，并满足实时处理数据与实时分析的需求。

2.2.3 Jinja 模板引擎

Jinja 模板只是一个文本文件，可以基于模板生成任何基于文本的格式（HTML、XML、CSV、LaTeX 等），一般用在前端的项目中，渲染 HTML 文件。

作为网络工程师，可以将其用来批量生成网络设备的配置。或者其他需要批量生成文本的场景中。模板包含变量或表达式，这两者在模板求值的时候会被替换为值。模板中还有标签，控制模板的逻辑。模板语法的大量灵感来自于 Django 和 Python。

2.3 SQLite 数据库介绍

SQLite 是一款轻量级数据库，是一个关系型数据库（RDBMS）管理系统，它包含在一个相对小的 C 库中，实现了自给自足的、无服务器的、零配置的、事务性的 SQL 数据库引擎。在很多嵌入式产品中使用了它，它占用资源非常的低，在嵌入式设备中，可能只需要几百 K 的内存就够了。

它能够支持 Windows/Linux/Unix/Android/iOS 等等主流的操作系统，同时能够跟很多程序语言相结合，比如 C#、PHP、Java 等，更重要的是 SQLite 文件格式稳定，跨平台且向后兼容，开发人员保证至少在 2050 年之前保持这种格式。SQLite 运行速度极快，目前是在世界上最广泛部署的 SQL 数据库，SQLite 源代码不受版权限制，任何人都可以免费使用于任何目的。

本项目将引入 SQLite 数据库来存储实时采集的数据，方便系统的组织与管理。同时数据库存储弹幕与评论数据来作为情感分析系统的数据来源。

2.4 BootSharp5 前端框架介绍

2.4.1 BootSharp 框架的发展

Bootstrap 是全球最受欢迎的前端组件库，用于开发响应式布局、移动设备优先的 WEB 项目。Bootstrap5 目前是 Bootstrap 的最新版本，是一套用于 HTML、CSS 和 JS 开发的开源工具集。它支持 Sass 变量和 mixins、响应式网格系统、大量的预建组件和强大的 JavaScript 插件，助你快速设计和自定义响应式、移动设备优先的站点。

2.4.2 BootSharp 框架的特点

Bootstrap 框架的流行，得益于它非常实用的功能和特点。主要核心功能特点如下。

(1) 跨设备、跨浏览器。可以兼容所有现代浏览器，包括 IE7、8 浏览器。

(2) 响应式布局。不但可以支持 PC 端的各种分辨率的显示，还支持移动端 PAD、手机等屏幕的响应式切换显示。

(3) 提供的全面的组件。Bootstrap 提供了实用性很强的组件，包括：导航、标签、工具条、按钮等一系列组件，方便开发者调用。

(4) 内置 jQuery 插件。Bootstrap 提供了很多实用性的 jQuery 插件，这些插件方便开发者实现 Web 中各种常规特效。

(5) 支持 HTML5、CSS3。HTML5 语义化标签和 CSS3 属性，都得到很好的支持。

(6) 支持 LESS 动态样式。LESS 使用变量、嵌套、操作混合编码，编写更快、更灵活的 CSS。它和 Bootstrap 能很好的配合开发。

本项目利用 BootSharp 框架来辅助 Jinja 模板引擎，来美化网页的设计。，并减少开发时间。

2.5 Echarts 图表介绍

ECharts 全称 EnterpriseCharts，商业级数据图表，使用 JavaScript 实现的开源可视化库，涵盖各行业图表，满足各种需求，能够流畅的运行在 PC 以及移动设备上，兼容当前绝大部分浏览器。Echarts 具有以下特点：

(1) ECharts 遵循 Apache-2.0 开源协议，免费商用。ECharts 兼容当前绝大部分浏览器(IE8/9/10/11，Chrome，Firefox，Safari 等)及兼容多种设备，可随时随地任性展示。

(2) 为我们许多提供直观，生动，可交互，可高度个性化定制的数据可视化图表。能够支持折线图、柱状图、散点图、K 线图、饼图、雷达图、和弦图、力导向布局图、地图、仪表盘、漏斗图、事件河流图等 12 类图表，同时提供标题，详情气泡、图例、值域、数据区域、时间轴、工具箱等 7 个可交互组件，支持多图表、组件的联动和混搭展现。

因此本项目利用 Echarts 图形库生成一系列可视化图表，方便分析数据与观察效果。因此在设计图表应尽可能合理与美观。

2.6 本章小节

本章节主要介绍 B 站弹幕可视化分析系统用到的技术理论，Python 语言是该系统的主要编程语言。其次是介绍了 Flask 框架，这是构建 WEB 应用软件的基础。Jinja 模板引擎就是渲染前端网页的工具，方便该系统可视化的开发。SQLite 数据库用于存储 B 站弹幕数据与评论数据，用于后续进行基于自然语言处理的情感分析。BootSharp5 框架可以帮助构建美观实用的前端框架，以低代码量的模式完成系统的前端构建任务。最后利用 Echarts 生成各种数据分析图表，可以对信息进行直观的展示。

3 系统需求分析

在设计 B 站弹幕可视化分析系统之前，进行合理的分析各方面的需求。需求分析是对项目工程进行评估，大致给出项目的基本框架，总结出对平台与功能的要求，权衡各种技术的利弊并应用到项目中。

3.1 功能性需求

3.1.1 B 站数据实时获取功能

传统的数据获取方式，可以收集互联网公开的信息，通过下载或爬虫的方式获取，但是数据并不完整或者容易遇到反爬虫机制。大公司可能会采用购买数据库的方式或市场调研的方式来获取数据，这样时间成本与经济成本通常会比较高。在 B 站弹幕可视化分析系统的设计过程中，采用直接调用 B 站的 API 接口来实时获取数据，这样能做到数据的更新与分析处理及时，实时将结果展示在前端页面。为了能够满足实时处理数据，在调用 API 接口的同时，提出以下几点要求：

(1) 多线程调用 API。在后端框架中，接收到前端发送的获取弹幕与评论数据的请求，立刻创建合适数量的线程池，同时调用 B 站官方的 API 接口。并将数据库写入数据库作为备份，并保持数据库事务的一致性，适当为数据库增加写入锁，防止数据库死锁。

(2) 有效的应对反爬虫的措施。过多的请求 API 必然会被 B 站的后端拦截，并返回错误代码，因此设计好循环次数，线程数与延迟时间，是解决 B 站反爬虫策略的核心原理。

(3) 清理无效数据。调用 API 接口返回的数据，可能会存在特殊字符、SQL 语句或无意义字符串。因此需要对数据进行清洗。

3.1.2 B 站数据存储功能

为了方便 B 站弹幕可视化分析系统对海量数据的存储、查找、修改的功能，因此系统的数据存储功能需求如下：

(1) 数据存储功能。本项目优先采用 SQLite 数据库管理系统来存储数据，一是免费开源，二是 SQLite 存储性能优秀，不需要安装与部署。

(2) 系统日志存储功能。使用 SQLite 数据库方便存储日志，便于后期对数据进行分析与迁移。

(3) 数据查找。SQLite 在千万级别数据量上进行插入与查找还是有非常优越的性能。

3.1.3 数据分析与可视化功能

由于调用 API 接口来获取数据的 B 站弹幕可视化分析系统中会有重复数据与异常数据，因此 B 站弹幕与评论数据分析功能模块完成对数据的预处理。除了根据规则去除一些空白值与错误数据，还要利用 jieba 分词库对弹幕内容、评论内容与评论会员等级数据要求进行分割，按照等级划分并统计词频。B 站弹幕可视化分析系统对弹幕内容、评论内容进行数据分析，因此具体的需求如下：

(1) 弹幕内容、评论内容数据分词。得到数据后，需要对数据中的弹幕内容、评论内容进行预处理，包括按照规则分词与去除停用词。

(2) 数据挖掘与统计。对获取后的数据中弹幕内容、评论内容与评论会员等级等数据直接用 Echarts 模块生成词云图与饼图。进一步数据挖掘功能可以进行使用基于自然语言处理的文本情感分类、文本信息熵计算等手段。因此 B 站弹幕与评论数据分析需要数据挖掘的需求。

(3) 数据可视化功能。本项目主要采用 Echarts 来生成动态美观的图表，满足丰富的交互体验。并且可以适应各种各样的数据与大量数据的分析统计与展示。

3.1.4 基于自然语言处理的文本情感分析功能

3.2 非功能性需求

B 站弹幕可视化分析系统各项功能与使用的开相对较复杂，因此在设计系统时尽可能简化复杂的功能，封装好底层细节，对系统的鲁棒性与性能方面有一定要求，最终使得系统更加易用。因此在系统非功能性需求提出几点要求。

3.2.1 易用性

为满足普通的访客能够无障碍使用系统的所有功能，对网页前端的界面设计有一定的要求，满足前端界面的美观、简洁，对应的功能按钮有相应的提示。部分用户可能没有较高的电脑操作技巧，如果设计的过于复杂，步骤过于繁琐，导致用户对系统的部分功能的理解产生歧义。因此网页前端的简洁与数据可视化图表的直观展示很重要。本系统提供了功能强大的数据分析功能，用户可以使用系统，看到 TOP20 视频的数据分析结果。

3.2.2 可靠性

为了确保 B 站弹幕可视化分析系统能够稳定运行，并保证数据的完整性，我们设计了一套异常处理机制，以处理其他不可预见的客观因素，如程序内部错误、数据错误或数据库写入错误。为了追踪异常错误的原因，系统中增加了一个日志系统，记录调用 API 过程中所有数据记录和不可预见的错误，这样就可以通过日志文件迅速找出问题的根本原因并进行分析，从而改善系统可靠性。

3.2.3 可扩展性

可扩展性是指系统不仅能满足目前的需求，还要能够根据新出现的需求进行更新，以满足日益变化的需求。对于 B 站弹幕可视化分析系统来说，重点需要考虑下面的几点的可扩展性：

(1) 目前 B 站弹幕可视化分析系统的数据来源是 B 站 API 接口，数据来源不够多。由于开发系统的期限紧迫，因此只选取 B 站 API 接口的数据，因此后期需要接入抖音、快手、推特等网站，进行数据深度扩展。

(2) 系统的可视化功能依赖于 Echarts 的统计功能，但是不一定足够准确和充分体现数据背后的客观规律，因此以后添加基于机器学习的聚类模型，提升系统的分析效果。

3.3 本章小节

本章主要对 B 站弹幕可视化分析系统的总体需求进行评估，对相关业务逻辑进行了梳理，大致分为功能性需求与非功能性需求。其中功能性需求主要从

B 站数据实时获取功能、B 站数据存储功能与数据分析、可视化功能与基于自然语言处理的文本情感分析功能等角度进行分析并展开叙述，接着对系统的非功能性需求进行介绍，主要从系统的易用性、可靠性和可扩展性三个角度展开论述，提出一些合理的建议与要求。

4 系统的详细设计

4.1 开发环境介绍

B 站弹幕可视化分析系统采用传统的 B/S 架构，设计一套前后端不分离的框架，采用 Jinja 模板引擎与 BootSharp5 前端框架构建了一套网页前端界面，最终开发成软件的形式方便用户使用。业务逻辑代码采用 Python 编写，Python 丰富的类库，优秀的框架，帮助项目辅助地完成各种功能。例如 PyEcharts 模块用来统计数据并生成动态的图表。

根据分析 B 站弹幕可视化分析系统功能需求，开发环境如表 4.1.1 所示。

表 4.1.1　开发环境配置表

要求	配置
操作系统	Windows 10 64 位
开发平台	Visual Studio Code 1.65
工具环境	Flask 2.2.2
数据库系统	SQLite3
编程语言	Python 3.7
是否支持跨平台运行	是

B 站弹幕可视化分析系统的开发与运行需要电脑硬件层面的支持，最低配置运行环境如下表 4.1.2 所示。

表 4.1.2　计算机硬件最低运行要求

要求	配置
操作系统	Windows 7（32 位或 64 位）以上（包含 Windows7）
CPU	奔腾 2.4GHz 双核
内存	2GB DDR3
显卡	显存 512M，支持 OpenGL2.0
硬盘	2GB 安装空间

4.2 系统整体架构介绍

系统的架构设计就是依据之前的需求分析提出的要求，将需求进行组织成一套框架，系统划分成多个层面，每个层面有具体的功能模块并设定方案进行详细的设计与完善。

B 站弹幕可视化分析系统主要分为四层，分别是前端界面、数据分析、数据存储与数据来源，如图 4.2.1 所示。

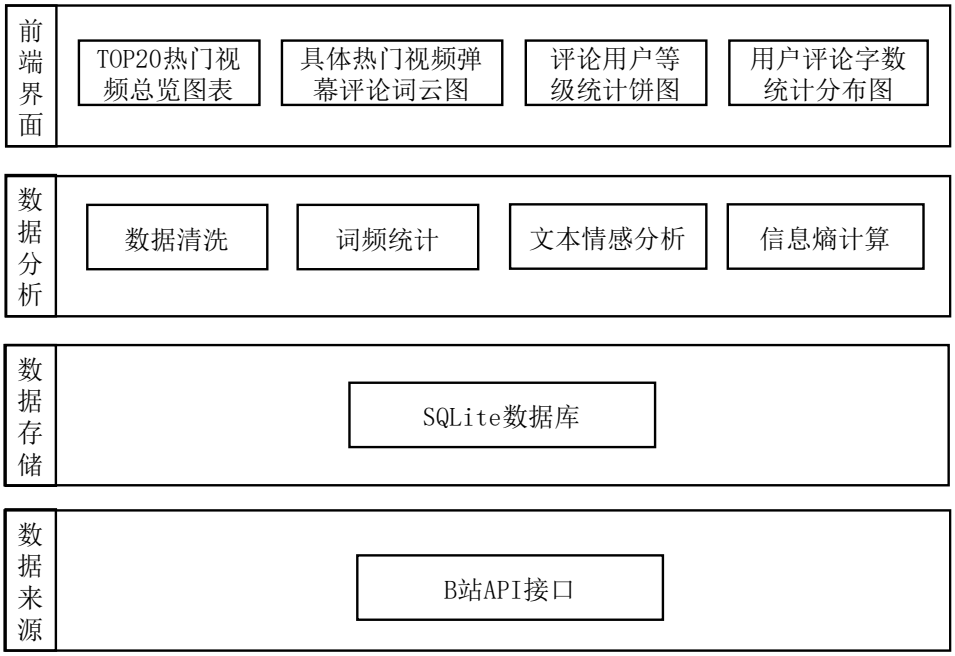

图 4.2.1B 站弹幕可视化分析系统架构

在设计本系统时，前端模块可使用 VScode 来进行代码的编辑、开发和优化，使用 Flask 基础架构，利用原生前端开发工具和 Bootstrap5 前端框架来进行前端页面的搭建，并使用 ECharts 可视化开发工具构建可视化图表，构建可视化系统，针对部分系统的数据，可使用 SQLite 数据库作为存储和交换的中间载体，系统设计技术路线直观表述。

4.3 系统功能模块设计与实现

4.3.1 数据实时获取与数据存储功能设计与实现

数据实时获取是通过获取 B 站的 API 来实现的，接下来重点介绍相关 API。

(1) 获取 TOP20 热门视频 API

GET 请求 Action：

```
https://api.bilibili.com/x/web-interface/popular?ps=20&pn=1
```

(2) 根据 BVID 获取视频弹幕 API

GET 请求 Action：

```
https://api.bilibili.com/x/player/pagelist?bvid=BVID&jsonp=jsonp
```

其中 BVID 可以根据 TOP20 热门视频 API 来获取。

返回的结果为 backdata，那么弹幕数据需要进行 Protobuf 格式解析。b 站弹幕传输的格式由原来的 XML 改为了 Protobuf，这个格式为二进制编码传输，其传输销量远高于原来的 XML，因此在移动端可以减小网络的压力具有一定的优势。但带来的一个问题就是，这个格式的弹幕解析起来变得十分困难，通常从 API 获得的数据直接看是一通乱码，需要特定的方式才能看到真正的内容。

ProtobufBuffers 是谷歌的语言中立、平台中立、可扩展的机制，用于序列化结构化数据例如 XML，但更小、更快、更简单。你只需定义一次你希望你的数据如何被结构化，然后你就可以使用特殊生成的源代码，轻松地从各种数据流和使用各种语言写入和读取你的结构化数据。

弹幕的 proto 定义并编译，protobuf 结构体。

```
syntax = "proto3";
package dm;
message DmSegMobileReply{
    repeated DanmakuElem elems = 1;
}
message DanmakuElem{
    int64 id = 1;
    int32 progress = 2;
    int32 mode = 3;
    int32 fontsize = 4;
    uint32 color = 5;
    string midHash = 6;
    string content = 7;
    int64 ctime = 8;
    int32 weight = 9;
    string action = 10;
    int32 pool = 11;
    string idStr = 12;
}
```

表 4.3.1 弹幕参数含义

名称	含义	类型	备注
id	弹幕 dmID	int64	唯一可用于操作参数
progress	视频内弹幕出现时间	int32	毫秒
mode	弹幕类型	int32	1 2 3：普通弹幕 4：底部弹幕 5：顶部弹幕 6：逆向弹幕 7：高级弹幕 8：代码弹幕 9：BAS 弹幕
fontsize	弹幕字号	int32	18：小 25：标准 36：大
color	弹幕颜色	uint32	十进制 RGB888 值
midHash	发送者 UID 的 HASH	string	用于屏蔽用户和查看用户发送的所有弹幕也可反查用户 ID
content	弹幕内容	string	utf-8 编码
ctime	弹幕发送时间	int64	时间戳
weight	权重	int32	用于智能屏蔽级别
action	动作	string	未知
pool	弹幕池	int32	0：普通池 1：字幕池 2：特殊池(代码/BAS 弹幕)
idStr	弹幕 dmID 的字符串类型	string	唯一可用于操作参数

解析之前需要先安装 python 的 probuf 包：

```
pip install protobuf
```

编译 proto 结构文件

```
protoc --python_out=. dm.proto
```

执行完成后会生成 dm_pb2.py，代码中引入这个 python 文件。

(3) 根据 CID 获取视频评论 API

GET 请求 Action：

```
http://api.bilibili.com/x/v2/dm/web/seg.so?type=1&oid=CID&segment_index=1
```

其中 CID 可以根据 TOP20 热门视频 API 来获取。

(4) 数据存储功能

SQLite 数据库驱动代码。

```
import sqlite3
import os
import threading
class DataBaseSqlite:
        DataBaseFileName="data.db"
        Connect=None
        Cursor=None
        lock = threading.Lock()
        def DataBaseIsExist(self):
                return os.path.exists("data.db")
        def Connect_DataBase(self):
                if self.DataBaseIsExist==False:
                return False
                self.Connect = sqlite3.connect(self.DataBaseFileName,check_same_thread=False)
                self.Cursor = self.Connect.cursor()
        def Query(self,SQL):
                try:
                self.lock.acquire(True)
                self.Cursor.execute(SQL)
                return self.Cursor.fetchall()
                finally:
                self.lock.release()
        def Execute(self,SQL):
                try:
                self.lock.acquire(True)
                self.Cursor.execute(SQL)
                self.Connect.commit()
                finally:
                self.lock.release()
        def Close(self):
                        self.Connect.close()
```

插入数据示例代码。

```
SQL="INSERT INTO main.record (id,BVID,type) VALUES ('%s','%s','%s')"%(uuid.uuid4
(),BVID,'danmaku')
Execute(SQL)
```


4.3.2 数据库架构设计与实现

B 站热门视频的数据保存在 SQLite 数据库中，根据数据库创建相应数据表，如表 4.3.2 所示。

表 4.3.2 B 站热门视频数据表

字段名称	数据类型	字段大小	是否为主键	是否允许为空	说明
id	integer	N/A	TRUE	FALSE	索引
type	varchar	50	FALSE	FALSE	类型
BVID	varchar	50	FALSE	FALSE	视频唯一 ID
USER	varchar	50	FALSE	FALSE	操作用户
create_time	varchar	50	FALSE	FALSE	创建时间

contents	varchar	255	FALSE	FALSE	内容
score	varchar	255	FALSE	FALSE	情感得分

数据保存在数据库成功，部分条目图 4.3.1 所示，数据库里包含 ID，类型，BVID，用户，创建时间，内容与情感得分。

id	type	BVID	user	create_time	contents	score
2416064d-12b4-42a2-a8	danmaku	BV1AG411F7eF		1669463410	我的眼睛會了	
2287d357-b89a-433e-b9	danmaku	BV1AG411F7eF		1669463476	二次反射眩光	
1e1d5f63-2a14-4ff5-92b	danmaku	BV1AG411F7eF		1669463556	win号	
da697900-ba06-424a-bc	danmaku	BV1AG411F7eF		1669463618	太牛了	
ba519b4a-85b2-4ff3-98	danmaku	BV1AG411F7eF		1669463671	好强	
3fa87f54-0597-4fb2-93e	danmaku	BV1AG411F7eF		1669463801	玉米地 摇子	
cd604ed9-3623-4849-ba	danmaku	BV1AG411F7eF		1669463815	做噩梦的程度，哈哈哈哈哈	
438fdc90-dca7-4df6-a52	danmaku	BV1AG411F7eF		1669463946	这个房不是记得给爸妈租的	
3e758c2b-1cc6-4163-aa	danmaku	BV1AG411F7eF		1669463955	星际穿越配乐	
ef9ddc6c-da1a-4686-a5	danmaku	BV1AG411F7eF		1669463999	想跳下去。。。。。	
4da51ba6-550c-4d5b-ac	danmaku	BV1AG411F7eF		1669464160	贞子快乐桌	
95ed55b7-8f8a-4ca6-93	danmaku	BV1AG411F7eF		1669464210	我的世界矿洞	
4394bf24-3db0-451a-bc	danmaku	BV1AG411F7eF		1669464215	师傅是从事什么工作啊	
c136869a-da23-41ea-8f	danmaku	BV1AG411F7eF		1669464230	为什么连气钉都有.....	
2a3302d8-68ab-4f61-a2	danmaku	BV1AG411F7eF		1669464239	仿佛准备教会我	
ee89f9fb-b147-4966-8ef	danmaku	BV1AG411F7eF		1669464319	波导，光纤是吧	
dba1a0cb-c0d4-48ab-8c	danmaku	BV1AG411F7eF		1669464324	梯子还有细节	
c66a3dd9-3771-460f-97	danmaku	BV1AG411F7eF		1669464332	我是不敢在这茶几上放东西	
2a618b5e-1a43-452b-91	danmaku	BV1AG411F7eF		1669464403	内存溢出桌	
4f6d505e-dc65-4f82-892	danmaku	BV1AG411F7eF		1669464429	加贞子	
01640630-dfe4-4c5c-a6	danmaku	BV1AG411F7eF		1669464769	效果惊人	
1aa102f4-42d9-4cbf-bd	danmaku	BV1AG411F7eF		1669464864	灰色的里面不是空的啊	
16b0868a-2b6e-4aae-a4	danmaku	BV1AG411F7eF		1669465039	太酷了 也想做一个	
d9faf8d0-6ee3-4eca-8a6	danmaku	BV1AG411F7eF		1669465638	爸妈住就不是自己家吗	
a32f2174-d445-432f-9ec	danmaku	BV1AG411F7eF		1669466005	吓人	
6b1f7610-8100-4084-98	danmaku	BV1AG411F7eF		1669466344	好可怕	
2db0d0a9-de5c-46e9-9l	danmaku	BV1AG411F7eF		1669466358	太厉害了	
2f4527af-d4d6-4721-84	danmaku	BV1AG411F7eF		1669466438	加个贞子	
3eb858a2-766f-4f45-94f	danmaku	BV1AG411F7eF		1669466496	加个贞子	
4393b4fd-b39f-4098-ba	danmaku	BV1AG411F7eF		1669467029	怕不怕半夜贞子从里面爬出	
a922486e-135d-4648-84	danmaku	BV1AG411F7eF		1669467218	秒哇	
2b6ca64b-2b10-49df-9e	danmaku	BV1AG411F7eF		1669467924	遗传天赋	
71721926-d0be-44c7-a6	danmaku	BV1AG411F7eF		1669468337	■：摆手办！看着就像手办	

图 4.3.1 部分数据

4.3.3 Echarts 可视化模块设计与实现

参考 Echarts 官网的部分示例，绘图代码见附录。其中设置折线为虚线的代码如下代码所示。

```
itemStyle: {
    normal: {
        lineStyle: {
        width: 3, //  设置虚线宽度
        type: 'dotted' //  虚线'dotted'  实线'solid'
        }
    }
}
```

绘制结果如图 4.3.3、图 4.3.4、图 4.3.5、图 4.3.6、图 4.3.7、图 4.3.8、图 4.3.9 与图 4.3.10 所示。

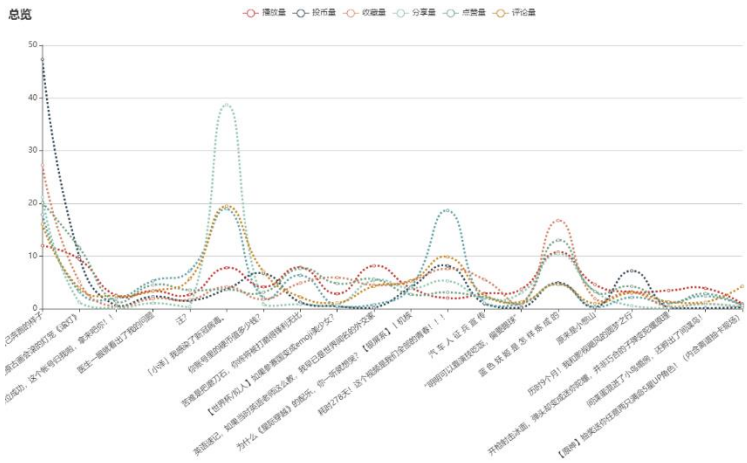

图 4.3.2 总览

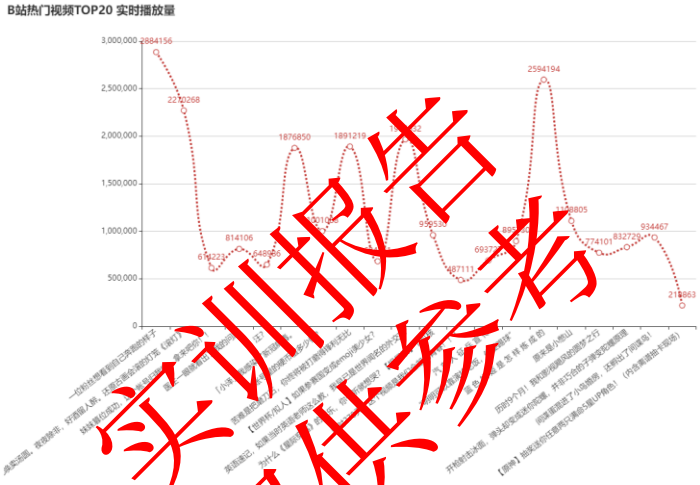

图 4.3.3 B 站视频 TOP20 实时播放量

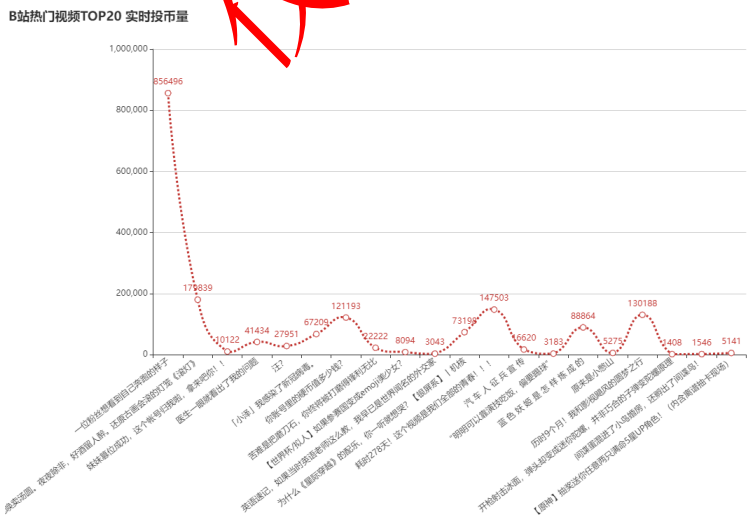

图 4.3.4 B 站视频 TOP20 实时投币量

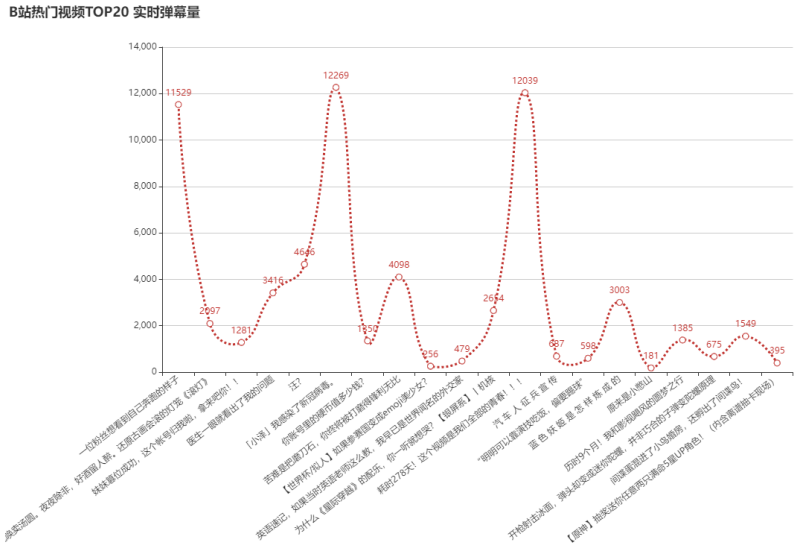

图 4.3.5 B 站视频 TOP20 实时弹幕量

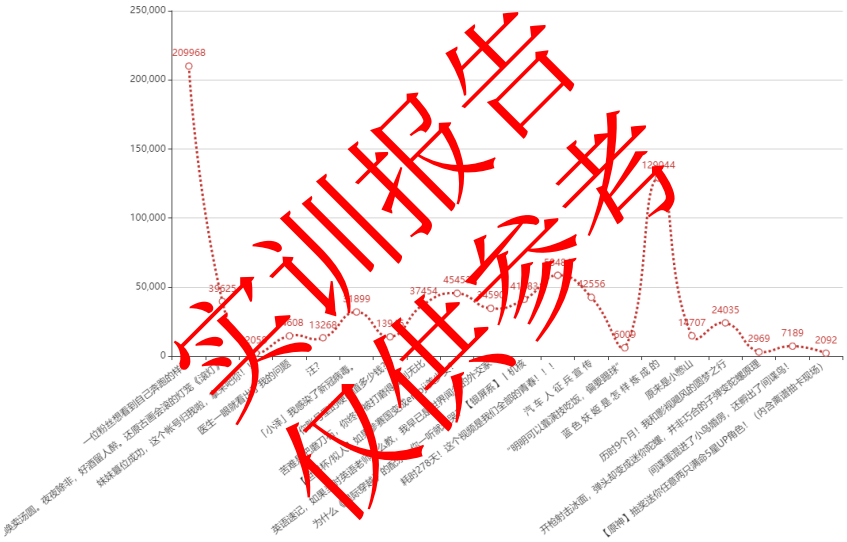

图 4.3.6 B 站视频 TOP20 实时收藏量

目前实时收集了980条弹幕数据进行分析

弹幕关键词TOP100

图 4.3.7 B 站视频弹幕关键词词云

图 4.3.8 B 站视频评论关键词词云

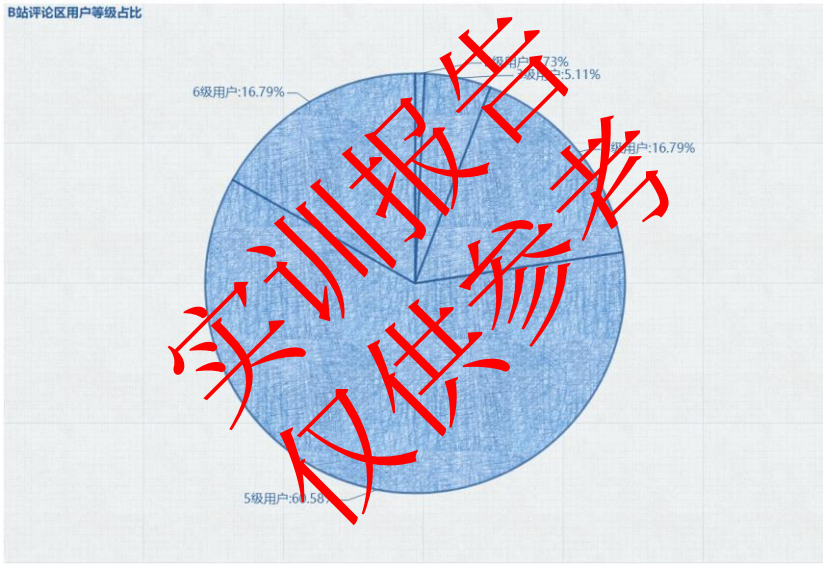

图 4.3.9 B 站评论区用户等级占比

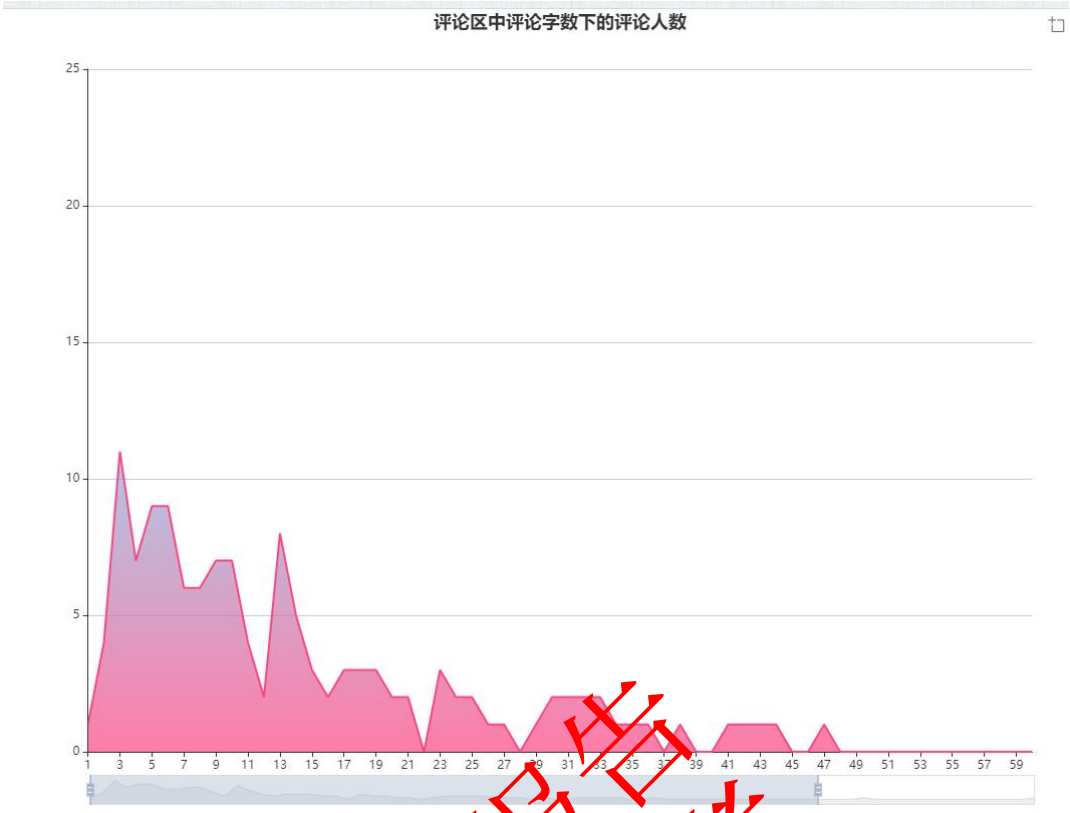

图 4.3.10 评论区中评论字数下的评论人数

4.3.4 Flask 后端多线程处理系统设计与实现

Flask 是 Python 中有名的轻量级同步 WEB 框架，在一些开发中，可能会遇到需要长时间处理的任务，此时就需要使用异步的方式来实现，让长时间任务在后台运行，先将本次请求的响应状态返回给前端，不让前端界面卡顿，当异步任务处理好后，如果需要返回状态，再将状态返回。

使用线程的方式，当要执行耗时任务时，直接开启一个新的线程来执行任务，这种方式最为简单快速。通过 ThreadPoolExecutor 来实现

```
from flask import Flask
from time import sleep
from concurrent.futures import ThreadPoolExecutor
#  创建线程池执行器
executor = ThreadPoolExecutor(2)
app = Flask(__name__)
@app.route('/jobs')
def run_jobs():
  #  交由线程去执行耗时任务
  executor.submit(long_task, 'hello', 123)
  return 'long task running.'
#  耗时任务
def long_task(arg1, arg2):
  print("args: %s %s!" % (arg1, arg2))
  sleep(5)
  print("Task is done!")
if __name__ == '__main__':
  app.run()
```

当要执行一些比较简单的耗时任务时就可以使用这种方式，如发邮件、发短信验证码等。

但这种方式有个问题，就是前端无法得知任务执行状态。如果想要前端知道，就需要设计一些逻辑，比如将任务执行状态存储 SQLite 数据库中，通过唯一的任务 UUID 进行标识，然后再写一个接口，通过任务 UUID 去获取任务的状态，然后让前端定时去请求该接口，从而获得任务状态信息。

4.4 本章小节

本章节主要详细地介绍各项关键模块的具体实现。分别实现了数据实时获取与数据存储功能设计与实现、数据库架构设计与实现、可视化模块设计与实现、后端多线程处理系统设计与实现。系统的所有功能都符合需求，系统满足分析 B 站热门视频数据的可视化与统计功能。

5 实训总结与展望

5.1 实训总结

本文详细说明了基于 ECharts 的 B 站弹幕可视化分析系统热门视频 TOP20 数据的可视化分析，从数据采集到最后的可视化展示，进行了正式的前后端交互。在前端方面也用上了 ECharts；数据采集采用了 Request 来请求 API 获取数据，后端方面是 Flask、Ajax 结合使用。

5.2 展望

由于开发系统的时间紧促，B 站弹幕可视化分析系统仍然存在很多不足之处。例如分析的视频数据的数据量远远不够，目前获取 B 站 TOP20 热门视频数据。希望以后可以对系统增加多个版块的视频内容或抖音平台的视频数据，增加更多种分析功能。

参考文献

[1] 杨小芳. 网络数据的可视化研究与实现[D].北京邮电大学,2014.

[2] 郭欢欢. 基于大数据方法的精准招聘研究[D].北京工业大学,2020.DOI:10.26935/d.cnki.gbjgu.2020.000381.

[3] Violaine.人生苦短,我用 Python[J].书城,2022(03):48-54.

[4] 郭子豪,刘一林,田鑫裕,陈凤英,李灿苹.基于 ECharts 的新冠肺炎疫情实时监控系统的设计与开发[J].电脑与信息技术,2022,30(01):35-39.DOI:10.19414/j.cnki.1005-1228.2022.01.007.

[5] 敬国伟,黄大池.基于 ECharts 的数据可视化研究[J].西部广播电视,2022,43(20):227-230+234.

[6] 哈奔,包乌云毕力格.基于 Bootstrap 和 Echarts 的资产数据可视化实现[J].电脑编程技巧与维护,2022(05):107-109.DOI:10.16184/j.cnki.comprg.2022.05.009.

[7] 郑涛,陈婷婷,林国凤,何晶.基于 ECharts 的英雄联盟（LPL）数据可视化分析[J].数字技术与应用,2022,40(08):206-208.DOI:10.19695/j.cnki.cn12-1369.2022.08.65.

[8] 彭曙光,王梦梅,赵麒博,申刘宝,彭玉杰,陆雪艳.面向 ECharts 的疫情信息可视化系统[J].福建电脑,2022,38(04):80-83.DOI:10.16707/j.cnki.fjpc.2022.04.021.

附录

后端代码

```
import json
from typing import Iterator
import uuid
from flask import Flask
from time import sleep
from concurrent.futures import ThreadPoolExecutor
import DataBase
import re
import string
from flask import Flask, render_template
from flask import *
import json
import requests
from dm_pb2 import DmSegMobileReply
from google.protobuf.json_format import MessageToJson, Parse
import jieba
from pyecharts.charts import Geo,Bar,Pie,WordCloud,Boxplot,Line,Map
from pyecharts import options
from pyecharts.globals import SymbolType
class Data:
    DataBase=None
    data={}#弹幕与评论数据,请求 ID 为键，数据为值
#  创建线程池执行器
executor = ThreadPoolExecutor(10)
app = Flask(__name__)

#################################################################
#####################       Flask 接口        ###################
#################################################################
#POST 接口，提交任务——获取弹幕数据
@app.route('/submit_task_get_danmaku_data/<BVID>/',methods=["POST"])
def submit_task_get_danmaku_data(BVID):
    requestID=str(uuid.uuid4())
    executor.submit(long_task_of_get_danmaku_data,requestID,BVID)#创建异步获取弹幕长任务
    submit_task_record_to_SQLite('get_danmaku_data',requestID)#提交任务记录到数据库
    return json.dumps([{
        'requestID':requestID ,
        'status': 'create_danmaku_task_success'
    }])
#GET 接口，获取弹幕数据
@app.route('/get_danmaku_data/<requestID>/')
def get_danmaku_data(requestID):
    if str(requestID) in Data.data:
        return Data.data[str(requestID)]
    else:
        return json.dumps([{
        'requestID':requestID ,
        'status':"danmaku_data_don't_exist"
    }])
#################################################################
#POST 接口，提交任务获取 TOP 数据
@app.route('/submit_task_get_TOP20_data', methods=["POST"])
def submit_task_get_TOP20_data():
    requestID=str(uuid.uuid4())
    executor.submit(long_task_of_get_TOP20_data,requestID)#创建异步获取弹幕长任务
    submit_task_record_to_SQLite('get_TOP20_data',requestID)#提交任务记录到数据库
    return json.dumps([{
        'requestID':requestID ,
        'status': 'create_TOP20_task_success'
    }])
```

```
#GET 接口，获取 TOP 数据
@app.route('/get_TOP20_data/<requestID>/')
def get_TOP20_data(requestID):
    if str(requestID) in Data.data:
        return Data.data[requestID]
    else:
        return json.dumps([{
        'requestID':requestID ,
        'status':"TOP20_data_don't_exist"
    }])
################################################################
#POST 接口，获取评论数据
@app.route('/submit_task_get_reply_data/<aid>/', methods=["POST"])
def submit_task_get_reply_data(aid):
    requestID=str(uuid.uuid4())
    executor.submit(long_task_of_get_reply_data,requestID,aid)#创建异步获取弹幕长任务
    submit_task_record_to_SQLite('get_reply_data',requestID)#提交任务记录到数据库
    return json.dumps([{
        'requestID':requestID ,
        'status': 'create_reply_task_success'
    }])
#GET 接口，获取评论数据
@app.route('/get_reply_data/<requestID>/')
def get_reply_data(requestID):
    if str(requestID) in Data.data:
        return Data.data[requestID]
    else:
        return json.dumps([{
        'requestID':requestID ,
        'status':"reply_data_don't_exist"
    }])
################################################################
#POST 接口，提交任务保存所有数据
@app.route('/submit_task_save_all_data/<data>/', methods=["POST"])
def submit_task_save_all_data(data):
    data=str(data).split("@")
    bvid=data[0]
    requestID_danmaku=data[1]
    requestID_reply=data[2]
    requestID=str(uuid.uuid4())
    if str(requestID_danmaku) not in Data.data or str(requestID_reply) not in Data.data:
        return json.dumps([{
        'requestID':requestID ,
        'status': 'create_save_all_data_task_fail'
    }])

    executor.submit(long_task_of_submit_task_save_all_data,requestID,bvid,requestID_danmaku,requestID_re
ply)#创建异步获取弹幕长任务
    submit_task_record_to_SQLite('save_all_data',requestID)#提交任务记录到数据库
    return json.dumps([{
        'requestID':requestID ,
        'status': 'create_save_all_data_task_success'
    }])
################################################################
#GET 接口，获取长任务执行状态
@app.route('/get_task_status/<requestID>/')
def get_long_task_status(requestID):
    return json.dumps([{
        'requestID':requestID ,
        'status':query_task_record_to_SQLite(requestID)
    }])
#####################    网页模板 GET 接口    ##########################
################################################################
@app.route('/', methods=["GET"])
def home():
    return render_template('index.html')
```

```
@app.route('/TOP20', methods=["GET"])
def TOP20():
    return render_template('TOP20.html')
#############################################################
#GET 接口，获取弹幕词云数据
@app.route('/get_danmaku_wordCloud/<requestID>/', methods=["GET"])#得到词云
def get_danmaku_wordCloud(requestID):
    if str(requestID) not in Data.data:
        return "<h1>警告！弹幕任务未完成，或数据已丢失！</h1>"
    ####开始处理####
    dist=""
    for i in Data.data[requestID]:
        dist=dist+i["content"]+" "
    pattern = re.compile(u"([\u4e00-\u9fff]+)")#正则表达式去除非中文字符
    dist= pattern.findall(dist)
    dist="".join(dist)
    # print(dist)
    seg_list =jieba.lcut(dist)
    # print(seg_list)
    wordcount = {}
    for word in seg_list:
        if len(word)>1:
            wordcount[word] = wordcount.get(word, 0)+1
    sorted_words=sorted(wordcount.items(), key=lambda x: x[1], reverse=True)[:100]#排序后词语
    #######生成词云图##########
    wordCloud=(WordCloud()
    .add("",sorted_words,word_size_range=[10,100],shape=SymbolType.DIAMOND)
    .set_global_opts(title_opts=options.TitleOpts(title="弹幕关键词 TOP100")))
    wordCloud.render(path="danmaku_wordCloud.html")
    #######生成词云图##########
    return open("danmaku_wordCloud.html", "r").read()
    ####结束处理####
#############################################################
#GET 接口，获取评论词云数据
@app.route('/get_reply_wordCloud/<requestID>/', methods=["GET"])#得到词云
def get_reply_wordCloud(requestID):
    if str(requestID) not in Data.data:
        return "<h1>警告！评论任务未完成，或数据已丢失！</h1>"
    ####开始处理####
    dist=""
    for i in Data.data[requestID]:
        dist=dist+i["评论"]+" "
    pattern = re.compile(u"([\u4e00-\u9fff]+)")#正则表达式去除非中文字符
    dist= pattern.findall(dist)
    dist="".join(dist)
    # print(dist)
    seg_list =jieba.lcut(dist)
    # print(seg_list)
    wordcount = {}
    for word in seg_list:
        if len(word)>1:
            wordcount[word] = wordcount.get(word, 0)+1
    sorted_words=sorted(wordcount.items(), key=lambda x: x[1], reverse=True)[:100]#排序后词语
    #######生成词云图##########
    wordCloud=(WordCloud()
    .add("",sorted_words,word_size_range=[10,100],shape=SymbolType.DIAMOND)
    .set_global_opts(title_opts=options.TitleOpts(title="评论关键词 TOP100")))
    wordCloud.render(path="reply_wordCloud.html")
    #######生成词云图##########
    return open("reply_wordCloud.html", "r").read()
    ####结束处理####

#############################################################################
#####################       数据库操作            ##########################
```

```
#############################################################################
#提交记录写入到数据库
def submit_task_record_to_SQLite(task_name,requestID):
    SQL="insert into main.task_record(id,task_name,requestID,status)
values('%s','%s','%s','%s')"%(str(uuid.uuid4()),task_name,requestID,'waiting')
    Data.DataBase.Submit_For_Execution(SQL)
#到数据库查询提交记录
def query_task_record_to_SQLite(requestID):
    count=Data.DataBase.Excute("select count(*) as count from main.task_record where
requestID='%s'"%(requestID))[0][0]
    if count>=1:
        SQL="select status from main.task_record where requestID='%s' limit 1"%(requestID)
        return Data.DataBase.Excute(SQL)[0][0]
    else:
        return "no_task_record"
#更新记录状态到数据库
def update_task_record_status_to_SQLite(requestID,status):
    SQL="update main.task_record set status='%s' where requestID='%s'"%(status,requestID)
    Data.DataBase.Submit_For_Execution(SQL)
def Excute(SQL):
    return Data.DataBase.Excute(SQL)
def Submit_For_Execution(SQL):
    Data.DataBase.Submit_For_Execution(SQL)
#############################################################################
######################      长任务处理          ############################
#############################################################################
def long_task_of_get_danmaku_data(requestID,BVID):
    resp = requests.get("https://api.bilibili.com/x/player/pagelist?bvid=" + BVID + "&jsonp=jsonp")#根据
BVID 得到 cid
    data_dict = json.loads(resp.content.decode("utf8"))#结果解码
    cid=str(data_dict["data"][0]["cid"])
    resp = requests.get('http://api.bilibili.com/x/v2/dm/web/seg.so?type=1&oid='+cid+'&segment_index=1')
    DM = DmSegMobileReply()#解密弹幕数据
    DM.ParseFromString(resp.content)
    data_dict = json.loads(MessageToJson(DM))["elems"]#弹幕结果列表
    Data.data[requestID]=data_dict#保存数据

    update_task_record_status_to_SQLite(requestID,'get_danmaku_data_done')#更新任务状态为执行成功
    print("long_task_of_get_danmaku_data is done!")

def long_task_of_get_reply_data(requestID,aid):
    print("任务创建成功")
    reply_data=[]#评论容器
    for i in range(1):
        print(i)
        try:
            resp = requests.get("https://api.bilibili.com/x/v2/reply/main?csrf=&mode=3&next=" +str(i)+
"&oid=" + aid + "&plat=1&seek_rpid=&type=1")
            data_dict = json.loads(resp.content.decode("utf8")).get("data","none").get("replies","none")#结果解
码
            for index, value in enumerate(data_dict):#遍历评论，共 20 条
                # print("第",index,"条","共",len(data_dict))
                rpid = value.get("rpid","none");ctime = value.get("ctime","none");like =
value.get("clike","none");
                message =value.get("content","none").get("message","none");avater
=value.get("member","none").get("avatar","none");
                level =value.get("member","none").get("level_info","none").get("current_level","none");sex
=value.get("sex","none");
                uname =value.get("uname","none");sign =value.get("sign","none");

                if message.find(":") != -1 and message.find("https") == -1:
                    message = message.split(":")[1]
                pattern = re.compile(u"([\u4e00-\u9fff]+)")#正则表达式去除非中文字符
                message= pattern.findall(message)
                message="".join(message)
```

```
                # print(message)#################
                reply_data.append({"rpid": rpid,"评论": message,"时间": ctime,"点赞": like,"头像": avater,"等级": level,"性别": sex,"用户名": uname,"个性签名": sign})
                # print({"rpid": rpid,"评论": message,"时间": ctime,"点赞": like,"头像": avater,"等级": level,"性别": sex,"用户名": uname,"个性签名": sign})
                try:
                    resp_ = requests.get("https://api.bilibili.com/x/v2/reply/reply?csrf=&oid=262995792&pn=1&ps=10&root=" + str(rpid) + "&type=1")
                    data_dict_=json.loads(resp_.content.decode("utf8")).get("data","none").get("replies","none")
                    for index,value in enumerate(data_dict_):
                        # print("---------------",index,"-------------------------","总共",len(data_dict_))
                        rpid = value.get("rpid","none");ctime = value.get("ctime","none");like = value.get("clike","none");
                        message =value.get("content","none").get("message","none");avater =value.get("member","none").get("avatar","none");
                        level =value.get("member","none").get("level_info","none").get("current_level","none");sex =value.get("sex","none");
                        uname =value.get("uname","none");sign =value.get("sign","none");

                        if message.find(":") != -1 and message.find("https") == -1:
                            message = message.split(":")[1]
                        pattern = re.compile(u"([\u4e00-\u9fff]+)")#正则表达式去除非中文字符
                        message= pattern.findall(message)
                        message="".join(message)
                        # print(message)############
                        reply_data.append({"rpid": rpid,"评论": message,"时间": ctime,"点赞": like,"头像": avater,"等级": level,"性别": sex,"用户名": uname,"个性签名": sign})
                        # print({"rpid": rpid,"评论": message,"时间": ctime,"点赞": like,"头像": avater,"等级": level,"性别": sex,"用户名": uname,"个性签名": sign})
                except:
                    print("异常 02")
                    continue
            except:
                print("异常 01")
                continue

        # print("结束")
        # print(reply_data)
        # data_dict = json.loads(reply_data)#评论结果列表
        Data.data[requestID]=reply_data#保存数据

        update_task_record_status_to_SQLite(requestID,'get_reply_data_done')#更新任务状态为执行成功
        print("long_task_of_get_reply_data is done!")
def long_task_of_get_TOP20_data(requestID):
        resp = requests.get("https://api.bilibili.com/x/web-interface/popular?ps=20&pn=1")
        data_dict =json.loads(resp.content.decode("utf8"))["data"]["list"]
        Data.data[requestID]=data_dict#保存数据

        update_task_record_status_to_SQLite(requestID,'get_TOP20_data_done')#更新任务状态为执行成功
        print("long_task_of_get_TOP20_data is done!")
def long_task_of_submit_task_save_all_data(requestID,BVID,requestID_danmaku,requestID_reply):
        if requestID=="" or BVID=="" or requestID_danmaku=="" or requestID_reply=="":
            print("空参数异常")
            return
        print("------------------------------------ requestID=",requestID,"|BVID=",BVID,"|requestID_danmaku=",requestID_danmaku,"|requestID_reply=",requestID_reply)
############################保存弹幕数据#################################

        SQL="SELECT COUNT(*) as count from record where BVID='%s' and type='danmaku' "%(BVID)
        print("------------------------------------SQL=",SQL)
        print("------------------------------------结果=",Excute(SQL))
```

```
    if(Excute(SQL)[0][0]>=1):
        print("重复数据已拦截 danmaku")
        update_task_record_status_to_SQLite(requestID,'save_all_data_fail')#更新任务状态为执行失败
        return
    SQL="INSERT INTO main.record (id,BVID,type) VALUES ('%s','%s','%s')
"%(uuid.uuid4(),BVID,'danmaku')
    print("------------------------",1,"------------------------")
    Submit_For_Execution(SQL)
    for i in Data.data[str(requestID_danmaku)]:
        SQL="INSERT INTO main.data (id,type,BVID,user,create_time,contents,score) VALUES
('%s','%s','%s','%s','%s','%s','%s')"%(uuid.uuid4(),"danmaku",BVID,"",i["ctime"],i["content"],"")
        Submit_For_Execution(SQL)
        print(SQL)
    print("------------------------",2,"------------------------")
##############################保存评论数据################################
    SQL="SELECT COUNT(*) as count from main.record where BVID='%s' and type='reply' "%(BVID)
    if(Excute(SQL)[0][0]>=1):
        print("重复数据已拦截 reply")
        update_task_record_status_to_SQLite(requestID,'save_all_data_fail')#更新任务状态为执行失败
        return
    print("------------------------",3,"------------------------")
    SQL="INSERT INTO main.record (id,BVID,type) VALUES ('%s','%s','%s')
"%(uuid.uuid4(),BVID,'reply')
    Submit_For_Execution(SQL)
    for i in Data.data[str(requestID_reply)]:
        SQL="INSERT INTO main.data (id,type,BVID,user,create_time,contents,score) VALUES
('%s','%s','%s','%s','%s','%s','%s')"%(uuid.uuid4(),"reply",BVID,"",i["时间"],i["评论"],"")
        Submit_For_Execution(SQL)
        print(SQL)
    # Data.DataBase.commit()
    print("------------------------",4,"------------------------")
    update_task_record_status_to_SQLite(requestID,'save_all_data_done')#更新任务状态为执行成功
    print("long_task_of_save_all_data is done!")
if __name__ == '__main__':
    Data.DataBase=DataBase.DataBaseSqlite()
    Data.DataBase.Connect_DataBase()
    app.run(port="80")
```


数据分析功能的设计与实现

```
function process_chart(json_data) {
    let view = {}//播放量
    json_data.forEach((item) => {
        view[item.title] = item["stat"].view
    });
    let coin = {}//投币数
    json_data.forEach((item) => {
        coin[item.title] = item["stat"].coin
    });
    let danmaku = {}//弹幕量
    json_data.forEach((item) => {
        danmaku[item.title] = item["stat"].danmaku
    });
    let favorite = {}//收藏量
    json_data.forEach((item) => {
        favorite[item.title] = item["stat"].favorite
    });
    let share = {}//分享量
    json_data.forEach((item) => {
        share[item.title] = item["stat"].share
    });
    let like = {}//点赞量
    json_data.forEach((item) => {
        like[item.title] = item["stat"].like
    });
```

```
let reply = {}//评论量
json_data.forEach((item) => {
    reply[item.title] = item["stat"].reply
});
////////////////////////////////////////////////////////////////////////////
let sum_view = 0;
Object.values(view).forEach((item) => {
    sum_view = sum_view + item;
});
let sum_coin = 0;
Object.values(coin).forEach((item) => {
    sum_coin = sum_coin + item;
});
let sum_danmaku = 0;
Object.values(danmaku).forEach((item) => {
    sum_danmaku = sum_danmaku + item;
});
let sum_favorite = 0;
Object.values(favorite).forEach((item) => {
    sum_favorite = sum_favorite + item;
});
let sum_share = 0;
Object.values(share).forEach((item) => {
    sum_share = sum_share + item;
});
let sum_like = 0;
Object.values(like).forEach((item) => {
    sum_like = sum_like + item;
});
let sum_reply = 0;
Object.values(reply).forEach((item) => {
    sum_reply = sum_reply + item;
});
//////////////计算百分比
let values_view = Object.values(view)
values_view.forEach((item, index, arr) => {
    values_view[index] = ((item / sum_view) * 100).toFixed(3);
});

let values_coin = Object.values(coin)
values_coin.forEach((item, index, arr) => {
    values_coin[index] = ((item / sum_coin) * 100).toFixed(3);
});

let values_danmaku = Object.values(danmaku)
values_danmaku.forEach((item, index, arr) => {
    values_danmaku[index] = ((item / sum_danmaku) * 100).toFixed(3);
});

let values_favorite = Object.values(favorite)
values_favorite.forEach((item, index, arr) => {
    values_favorite[index] = ((item / sum_favorite) * 100).toFixed(3);
});

let values_share = Object.values(share)
values_share.forEach((item, index, arr) => {
    values_share[index] = ((item / sum_share) * 100).toFixed(3);
});

let values_like = Object.values(like)
values_like.forEach((item, index, arr) => {
    values_like[index] = ((item / sum_like) * 100).toFixed(3);
});

let values_reply = Object.values(reply)
values_reply.forEach((item, index, arr) => {
    values_reply[index] = ((item / sum_reply) * 100).toFixed(3);
```

```
    });
        }
```

Echarts 可视化模块设计与实现

```
let chartDom = document.getElementById('main0');
let myChart = echarts.init(chartDom);
let option;
option = {
    title: {
        text: '总览'
    },
    tooltip: {
        trigger: 'axis',
        axisPointer: {
            type: 'cross',
            crossStyle: {
                color: '#999'
            }
        }
    },
    legend: {
        data: ['播放量', '投币量', '弹幕量', '收藏量', '分享量', '点赞量', '评论量']
    },
    grid: {
        left: '3%',
        right: '4%',
        bottom: '3%',
        containLabel: true
    },
    toolbox: {
        feature: {
            saveAsImage: {}
        }
    },
    xAxis: {
        type: 'category',
        boundaryGap: false,
        data: Object.keys(view),
        axisLabel: {
            interval: 0, //  坐标刻度之间的显示间隔，默认就可以了
            rotate: 38 //  调整数值改变倾斜的幅度（范围-90 到 90）
        },
    },
    yAxis: { type: 'value' },
    series: [
        {
            name: '播放量',
            type: 'line',
            // stack: 'Total',
            data: values_view,
            smooth: true,
            itemStyle: {
                normal: {
                    // label: {
                    //       show: true //  在折线拐点上显示数据
                    // },
                    lineStyle: {
                        width: 3, //  设置虚线宽度
                        type: 'dotted' //  虚线'dotted'  实线'solid'
                    }
                }
            }
        },
        {
            name: '投币量',
            type: 'line',
```

```
            // stack: 'Total',
            smooth: true,
            data: values_coin,
            itemStyle: {
                normal: {
                    // label: {
                    //       show: true //  在折线拐点上显示数据
                    // },
                    lineStyle: {
                        width: 3, //  设置虚线宽度
                        type: 'dotted' //  虚线'dotted'  实线'solid'
                    }
                }
            }
        },
        {
            name: '弹幕数',
            type: 'line',
            smooth: true,
            // stack: 'Total',
            data: values_danmaku,
            itemStyle: {
                normal: {
                    // label: {
                    //       show: true //  在折线拐点上显示数据
                    // },
                    lineStyle: {
                        width: 3, //  设置虚线宽度
                        type: 'dotted' //  虚线'dotted'  实线'solid'
                    }
                }
            }
        },
        {
            name: '收藏量',
            type: 'line',
            smooth: true,
            // stack: 'Total',
            data: values_favorite,
            itemStyle: {
                normal: {
                    // label: {
                    //       show: true //  在折线拐点上显示数据
                    // },
                    lineStyle: {
                        width: 3, //  设置虚线宽度
                        type: 'dotted' //  虚线'dotted'  实线'solid'
                    }
                }
            }
        },
        {
            name: '分享量',
            type: 'line',
            smooth: true,
            // stack: 'Total',
            data: values_share,
            itemStyle: {
                normal: {
                    // label: {
                    //       show: true //  在折线拐点上显示数据
                    // },
                    lineStyle: {
                        width: 3, //  设置虚线宽度
                        type: 'dotted' //  虚线'dotted'  实线'solid'
```

```
                    }
                }
            }
        },
        {
            name: '点赞量',
            type: 'line',
            smooth: true,
            // stack: 'Total',
            data: values_like,
            itemStyle: {
                normal: {
                    // label: {
                    //       show: true //  在折线拐点上显示数据
                    // },
                    lineStyle: {
                        width: 3, //  设置虚线宽度
                        type: 'dotted' //  虚线'dotted'  实线'solid'
                    }
                }
            }
        },
        {
            name: '评论量',
            type: 'line',
            smooth: true,
            // stack: 'Total',
            data: values_reply,
            itemStyle: {
                normal: {
                    // label: {
                    //       show: true //  在折线拐点上显示数据
                    // },
                    lineStyle: {
                        width: 3, //  设置虚线宽度
                        type: 'dotted' //  虚线'dotted'  实线'solid'
                    }
                }
            }
        }
    ]
};
        option && myChart.setOption(option);
```