

**电 子 科 技 大 学**

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

**学 士 学 位 论 文**

BACHELOR DISSERTATION

**论文题目 直播平台弹幕过滤系统开发**

**学生姓名 张智烁**

**学　　号 2014220104003**

**学　　院 信息与软件工程学院**

**专　　业 软件工程（信息工程）**

**指导教师 汪俊**

**指导单位 中石化沧州分公司**

**2018年 5 月 30 日**

摘 要

本文主要研究了直播平台弹幕信息智能分析与识别系统的设计与开发。针对目前流行的网络直播弹幕质量的问题，主要实现了对广告弹幕、语言暴力弹幕、色情弹幕等的过滤，对网络信息进行净化，维护网络直播环境，让观看者能有更好的体验，更愉快的观感，降低网络暴力、欺诈等事件发生的可能性。也在设计开发过程中学习到了关于Python、爬虫、Socket通信等知识。

**关键词：直播平台，弹幕，过滤，Python，爬虫**

ABSTRACT

This thesis mainly studies the design and development of the intelligent analysis and identification system of projectile screen information on the live platform. In order to solve the problem of the current popular webcast projectile quality, it mainly realizes the filtering of advertisement screen, language violence screen, pornographic screen and so on, purifies the network information, maintains the network live broadcast environment, so that the viewer can have a better experience. More pleasant perception, reducing the likelihood of cyber violence, fraud and other incidents. Also in the design and development process to learn about Python, spider, Socket communication and other knowledge.

**Keywords:** **Live platform，Barrage，Filter，Python，Spider**

目 录

[第一章 毕业设计（顶岗实习）概况 1](#_Toc516153605)

[1.1 选题背景，意义及价值 1](#_Toc516153606)

[1.2 国内外对弹幕过滤系统的研究现状 2](#_Toc516153607)

[1.3 本文的主要工作 2](#_Toc516153608)

[1.4 论文的章节安排 2](#_Toc516153609)

[第二章 开发环境及所用关键技术介绍 4](#_Toc516153610)

[2.1 开发环境PyCharm介绍 4](#_Toc516153611)

[2.2 抓包工具Fiddler和WireShark介绍 4](#_Toc516153612)

[2.3 Python语言介绍 4](#_Toc516153613)

[2.4 Nodejs介绍 5](#_Toc516153614)

[2.5 python requests爬虫库介绍 6](#_Toc516153615)

[2.6 Django框架介绍 6](#_Toc516153616)

[第三章 系统需求分析 8](#_Toc516153617)

[3.1 项目任务概述 8](#_Toc516153618)

[3.2 系统可行性分析 8](#_Toc516153619)

[3.2.1 技术可行性分析 8](#_Toc516153620)

[3.2.2 协议可行性分析 8](#_Toc516153621)

[3.2.3 经济可行性分析 9](#_Toc516153622)

[3.2.4 社会实现可行性分析 9](#_Toc516153623)

[3.3 系统总体业务功能分析 9](#_Toc516153624)

[3.3.1 系统顶层逻辑用例分析 9](#_Toc516153625)

[3.3.2 系统顶层数据流分析 10](#_Toc516153626)

[3.4 系统模块功能分析 11](#_Toc516153627)

[3.4.1 前端展示模块功能分析 11](#_Toc516153628)

[3.4.2 后端响应模块功能分析 11](#_Toc516153629)

[3.4.3 弹幕处理进程处理流程分析 11](#_Toc516153630)

[3.4.4 弹幕爬取模块功能分析 12](#_Toc516153631)

[3.4.5 弹幕过滤模块功能分析 12](#_Toc516153632)

[3.5 系统非功能性需求 13](#_Toc516153633)

[3.5.1 运行环境要求 13](#_Toc516153634)

[3.5.2 设计约束要求 13](#_Toc516153635)

[3.5.3 性能要求 13](#_Toc516153636)

[第四章 系统设计 14](#_Toc516153637)

[4.1 总体概要设计 14](#_Toc516153638)

[4.2 总体时序设计 14](#_Toc516153639)

[4.3 模块详细设计 15](#_Toc516153640)

[4.3.1 系统模块组成设计 15](#_Toc516153641)

[4.3.2 网站架构模块详细设计 16](#_Toc516153642)

[4.3.3 弹幕爬取模块详细设计 16](#_Toc516153643)

[4.3.4 弹幕过滤模块详细设计 17](#_Toc516153644)

[4.4 AC多模匹配过滤算法设计 18](#_Toc516153645)

[4.4.1 算法原理描述 18](#_Toc516153646)

[4.4.2 算法实际应用设计 21](#_Toc516153647)

[4.5 基于敏感字符出现次数过滤算法设计 22](#_Toc516153648)

[第五章 系统实现 23](#_Toc516153649)

[5.1 弹幕爬取模块实现 23](#_Toc516153650)

[5.1.1 斗鱼tv弹幕爬取模块实现 23](#_Toc516153651)

[5.1.2 熊猫tv（不包含星秀和星颜模块）弹幕爬取模块实现 29](#_Toc516153652)

[5.1.3 熊猫tv（星秀和星颜模块）弹幕爬取模块实现 35](#_Toc516153653)

[5.1.4 战旗tv弹幕爬取流程实现 41](#_Toc516153654)

[5.1.5 虎牙tv弹幕爬取流程实现 44](#_Toc516153655)

[5.1.6 全民tv弹幕爬取流程实现 47](#_Toc516153656)

[5.1.7 bilibili直播模块弹幕爬取流程实现 49](#_Toc516153657)

[5.2 弹幕过滤模块实现 53](#_Toc516153658)

[5.2.1 弹幕过滤预处理模块实现 53](#_Toc516153659)

[5.2.2 弹幕过滤AC算法模块实现 56](#_Toc516153660)

[5.3 网站架构模块实现 58](#_Toc516153661)

[5.3.1 首页输入框实时校验功能实现 58](#_Toc516153662)

[5.3.2 首页点击开始过滤按钮后台处理逻辑实现 60](#_Toc516153663)

[5.3.3 弹幕实时滚动效果的实现 63](#_Toc516153664)

[5.3.4 前后端WebSocket弹幕信息传输功能实现 64](#_Toc516153665)

[5.4 系统实现中存在的主要问题和解决方案 67](#_Toc516153666)

[5.4.1 存在的主要问题 67](#_Toc516153667)

[5.4.2 解决方案和可行性研究 68](#_Toc516153668)

[第六章 系统测试 72](#_Toc516153669)

[6.1 测试环境 72](#_Toc516153670)

[6.2 测试方法 72](#_Toc516153671)

[6.3 测试结果 72](#_Toc516153672)

[6.3.1 斗鱼tv弹幕爬取结果展示 72](#_Toc516153673)

[6.3.2 熊猫tv弹幕爬取结果展示 73](#_Toc516153674)

[6.3.3 熊猫tv（星秀星颜模块）弹幕爬取结果展示 73](#_Toc516153675)

[6.3.4 战旗tv弹幕爬取结果展示 73](#_Toc516153676)

[6.3.5 全民tv弹幕爬取结果展示 74](#_Toc516153677)

[6.3.6 bilibili直播模块弹幕爬取结果展示 74](#_Toc516153678)

[6.3.7 网站架构模块与弹幕处理子进程集成测试展示 74](#_Toc516153679)

[6.4 测试小结 75](#_Toc516153680)

[第七章 工程能力与职业素养培养 76](#_Toc516153681)

[7.1 知识技能学习情况 76](#_Toc516153682)

[7.2 对软件工程实践以及软件工程领域发展的认识 76](#_Toc516153683)

[第八章 结束语 77](#_Toc516153684)

[8.1 全文总结 77](#_Toc516153685)

[8.2 不足与下一步工作 77](#_Toc516153686)

[参考文献 78](#_Toc516153687)

[致 谢 79](#_Toc516153688)

[外文资料原文 80](#_Toc516153689)

[外文资料翻译 82](#_Toc516153690)

1. 毕业设计（顶岗实习）概况
   1. 选题背景，意义及价值

网络直播是新兴的高互动性视频娱乐，根据相关报告显示，第一家视频直播网站出现于2005年，而该行业于2016年呈现井喷式发展趋势。网络直播不断介入人们的视野和生活，乃至成为年轻人必不可少的娱乐方式。中国互联网络信息中心(CNNIC)2017年8月在北京发布第38次《中国互联网络发展状况统计报告》显示，截至2017年6月，中国网民规模达7亿1000万，其中，网络直播用户规模达到3亿2500万，占网民总体的45.8%。直播已经成为不可逆的潮流。

网络直播不同于优酷，土豆，腾讯视频等以往的视频播放网站，它具有如下几个特殊的特点和性质：

第一，直播内容多样化，有秀场类，户外类，颜值类，舞蹈类，猎奇类，热门游戏类，棋牌游戏类等；多样的直播内容为直播乱象不好监管埋下了伏笔。

第二，主播门槛要求低，纵观各大网络直播平台，网络主播水平参差不齐是现阶段直播平台发展所面临的问题之一，导致这样现状就是直播门槛低。一个摄像头一台有网络连接的电脑一个麦克风就可以进行网络直播。一部智能手机一个自拍杆一个wifi环境就可以进行手机直播。这些在生活中看似再平常不过的物件此时却成为网络直播的必需品，而这样的配置几乎每个人都可以毫无费力地得到并可以良好的使用。正是这样低门槛的设备要求促使了主播人数大大增加。伴随着主播人数的大幅度增加新的问题又呈现出来：主播文化素质及个人修养良莠不齐。由于各平台主播在申请直播间过程中，并没有复杂的审核环节，以大多数平台为例，只要你在网上填写个人信息绑定银行卡就可以完成注册。由于没有年龄限制，没有职业的要求，更没有对学历进行说明，这也就促使了文化层次分明，个人素质差距大的现状存在。这也为直播平台乱象埋下伏笔【1】。

第三，直播具有较强的时效性、娱乐性和互动性。，主播的直播视频流上传到直播平台的流服务器上，在网络稳定的情况下，传输到观众的客户端上的时间不会超过10s。也就是意味着观众可以实时观看主播的一举一动，网友在观看的过程中可以发送文字或表情与主播进行互动，这种传播方式更符合大众传播的发展趋势。相比于微信朋友圈，微博评论等方式，网络直播可以满足受众实时发表看法的需要，而弹幕的内容也可以随时变化直播的主题内容，实现了两者的完美互动过程。而另一方面，弹幕的内容由于对所有人可见，产生的影响不容小视。虽然网络直播最受年轻人的追捧，但随着内容、领域、门类的不断扩充其市场占有量在各年龄阶段均有提升。尽管受众数量增长趋势良好，但层次划分严重。大部分网民上网的目的是分散精力和压力，因此会出现煽动性强、内容露骨等不雅观的评论同主播的低俗内容互动。这些言语会使意志力不强“随风倒”的网民随之附和，而这种激进的言论更会促使主播“秀下限”。同时“网络喷子”“键盘侠”“黑粉”数量大幅度增加，他们游走于各大直播平台，大规模发布虚假充值信息、不良网站链接、引起粉丝间谩骂，这也使网络环境变得污浊不堪【2】。

所以，净化弹幕是净化直播环境的第一步也是最重要，最迫在眉睫的一步。只有将弹幕净化好，直播间的网络氛围才会变得轻松有序，观众与主播间的交流也会变得健康和谐，轻松愉悦。也能从根本上抵制不良信息对辨别是非能力不强的未成年人侵害。

* 1. 国内外对弹幕过滤系统的研究现状

据调查，国内外直播平台对弹幕系统存在监管不严的状况，大多数直播平台主要还是依靠管理员来排查非法弹幕，主要依靠人员来识别非法弹幕主要有两大问题：第一，在弹幕流量很多的情况下不容易找出非法弹幕的位置，第二，会浪费人力物力在这些无关劳动上。只有很少一部分平台会建立自己的脏字弹幕库来识别脏字弹幕，但是也存在算法不精准，效率不够高，不能实时过滤掉非法弹幕等问题，只能后来用算法来分析数据库中存有的弹幕历史信息来发掘脏字弹幕进行处理。所以，使用合理高效的识别算法来识别非法弹幕，做到实时过滤非法弹幕对于直播行业的发展和社会的文明和谐稳定是必要的，也是迫在眉睫的【3】。

* 1. 本文的主要工作

分析斗鱼，虎牙，哔哩哔哩，全民，战旗，熊猫六大直播平台的弹幕协议，并用python分别重写握手包和心跳包，与各大平台的弹幕服务器进行连续，完成对弹幕的实时获取，并将获取到的弹幕信息用AC算法和敏感字符匹配次数算法进行过滤并添加过滤标记。最后将整个系统架设在由Django和Django channels共同搭建的异步服务器上，为用户提供网页操作接口。

* 1. 论文的章节安排

第二章介绍了项目所用的开发环境和关键技术。第三章介绍了系统的需求分析，包括总体的用例分析，总体的数据流分析和各个模块的用例数据流分析等。第四章介绍了系统的设计阶段，包括系统的总体的概要设计和各个模块的详细设计情况。第五章介绍了系统的实现，分成各个模块分别进行实现，包括实现的主要代码。第六章介绍的是系统测试情况，包括测试环境的介绍和功能测试性能测试情况的介绍。第七章是工程能力与职业素养培养，主要包括知识技能学习情况和对软件工程领域发展的认识。第八章是结束语，是对全文内容的总结。

1. 开发环境及所用关键技术介绍
   1. 开发环境PyCharm介绍

PyCharm是一种Python IDE，带有一整套可以帮助用户在使用Python语言开发时提高其效率的工具，比如调试、语法高亮、Project管理、代码跳转、智能提示、自动完成、单元测试、版本控制。另外它自带的HTML，CSS和 JavaScript编辑器 ，用户可以更快速的通过Djang框架进行Web开发。它最大优势是自带python调试功能，用户可以用其自带的功能全面的调试器对Python或者Django应用程序以及测试单元进行调整，该调试器带断点，步进，多画面视图，窗口以及评估表达式。总之，这是一款全能的python开发IDE

* 1. 抓包工具Fiddler和WireShark介绍

Wireshark是非常流行的网络封包分析软件，功能十分强大。可以截取各种网络封包，显示网络封包的详细信息。wireshark是开源软件。 可以运行在Windows，Linux和Mac OS上。为了安全考虑，wireshark只能查看封包，而不能修改封包的内容，或者发送封包。wireshark能获取HTTP也能获取HTTPS，但是不能解密HTTPS，所以wireshark看不懂HTTPS中的内容。

Fiddler是一个http协议调试代理工具，它能够记录并检查所有你的电脑和互联网之间的http通讯，设置断点，查看所有的“进出”Fiddler的数据（指cookie,html,js,css等文件。

总结来说，分析HTTP,HTTPS WebSocket 协议 用Fiddler抓包,其他协议比如TCP,UDP。就用Wireshark抓包。

* 1. Python语言介绍

Python （英国发音：/ˈpaɪθən/ 美国发音：/ˈpaɪθɑːn/）, 是一种面向对象的解释型计算机程序设计语言，由荷兰人Guido van Rossum于1989年发明，第一个公开发行版发行于1991年。

Python是纯粹的自由软件， 源代码和解释器CPython遵循 GPL(GNU General Public License)协议。Python语法简洁清晰，特色之一是强制用空白符(white space)作为语句缩进。

Python具有丰富和强大的库。它常被昵称为胶水语言，能够把用其他语言制作的各种模块（尤其是C/C++）很轻松地联结在一起。常见的一种应用情形是，使用Python快速生成程序的原型（有时甚至是程序的最终界面），然后对其中有特别要求的部分，用更合适的语言改写，比如3D游戏中的图形渲染模块，性能要求特别高，就可以用C/C++重写，而后封装为Python可以调用的扩展类库。

Python在执行时，首先会将.py文件中的源代码编译成Python的byte code（字节码），然后再由Python Virtual Machine（Python虚拟机）来执行这些编译好的byte code。这种机制的基本思想跟Java，.NET是一致的。然而，Python Virtual Machine与Java或.NET的Virtual Machine不同的是，Python的Virtual Machine是一种更高级的Virtual Machine。这里的高级并不是通常意义上的高级，不是说Python的Virtual Machine比Java或.NET的功能更强大，而是说和Java 或.NET相比，Python的Virtual Machine距离真实机器的距离更远。或者可以这么说，Python的Virtual Machine是一种抽象层次更高的Virtual Machine。基于C的Python编译出的字节码文件，通常是.pyc格式。

除此之外，Python还可以以交互模式运行，比如主流操作系统Unix/Linux、Mac、Windows都可以直接在命令模式下直接运行Python交互环境。直接下达操作指令即可实现交互操作【4】。

* 1. Nodejs介绍

Node.js是一个Javascript运行环境(runtime environment)，发布于2009年5月，由Ryan Dahl开发，实质是对Chrome V8引擎进行了封装。Node.js对一些特殊用例进行优化，提供替代的API，使得V8在非浏览器环境下运行得更好。

V8引擎执行Javascript的速度非常快，性能非常好。Node.js是一个基于Chrome JavaScript运行时建立的平台， 用于方便地搭建响应速度快、易于扩展的网络应用。Node.js 使用事件驱动， 非阻塞I/O 模型而得以轻量和高效，非常适合在分布式设备上运行数据密集型的实时应用。

V8引擎本身使用了一些最新的编译技术。这使得用Javascript这类脚本语言编写出来的代码运行速度获得了极大提升，又节省了开发成本。对性能的苛求是Node的一个关键因素。 Javascript是一个事件驱动语言，Node利用了这个优点，编写出可扩展性高的服务器。Node采用了一个称为“事件循环(event loop）”的架构，使得编写可扩展性高的服务器变得既容易又安全。提高服务器性能的技巧有多种多样。Node选择了一种既能提高性能，又能减低开发复杂度的架构。这是一个非常重要的特性。并发编程通常很复杂且布满地雷。Node绕过了这些，但仍提供很好的性能。

Node采用一系列“非阻塞”库来支持事件循环的方式。本质上就是为文件系统、数据库之类的资源提供接口。向文件系统发送一个请求时，无需等待硬盘（寻址并检索文件），硬盘准备好的时候非阻塞接口会通知Node。该模型以可扩展的方式简化了对慢资源的访问， 直观，易懂。尤其是对于熟悉onmouseover、onclick等DOM事件的用户，更有一种似曾相识的感觉【5】。

虽然让Javascript运行于服务器端不是Node的独特之处，但却是其一强大功能。不得不承认，浏览器环境限制了我们选择编程语言的自由。任何服务器与日益复杂的浏览器客户端应用程序间共享代码的愿望只能通过Javascript来实现。虽然还存在其他一些支持Javascript在服务器端 运行的平台，但因为上述特性，Node发展迅猛，成为事实上的平台。

* 1. python requests爬虫库介绍

Requests 是用Python语言编写，基于 urllib，采用 Apache2 Licensed 开源协议的 HTTP 库。它比 urllib 更加方便，可以节约我们大量的工作，完全满足 HTTP 测试需求。被公认为是最好的python爬虫库

* 1. Django框架介绍

Django是一个基于MVC构造的框架。但是在Django中，控制器接受用户输入的部分由框架自行处理，所以 Django 里更关注的是模型（Model）、模板(Template)和视图（Views），称为 MTV模式。它们各自的职责如下：

表2-1 Django MTV模式所代表的含义

|  |  |
| --- | --- |
| 层次 | 职责 |
| 模型（Model），即数据存取层 | 处理与数据相关的所有事务： 如何存取、如何验证有效性、包含哪些行为以及数据之间的关系等。 |
| 模板(Template)，即业务逻辑层 | 处理与表现相关的决定： 如何在页面或其他类型文档中进行显示。 |
| 视图（View），即表现层 | 存取模型及调取恰当模板的相关逻辑。模型与模板的桥梁。 |

从以上表述可以看出Django 视图不处理用户输入，而仅仅决定要展现哪些数据给用户，而Django 模板 仅仅决定如何展现Django视图指定的数据。或者说, Django将MVC中的视图进一步分解为 Django视图 和 Django模板两个部分，分别决定 “展现哪些数据” 和 “如何展现”，使得Django的模板可以根据需要随时替换，而不仅仅限制于内置的模板。

至于MVC控制器部分，由Django框架的URLconf来实现。URLconf机制是使用正则表达式匹配URL，然后调用合适的Python函数。URLconf对于URL的规则没有任何限制，你完全可以设计成任意的URL风格，不管是传统的，RESTful的，或者是另类的。框架把控制层给封装了，在写程序的时候，只要调用相应的方法就行了，很方便。程序员把控制层东西交给Django自动完成了。 只需要编写非常少的代码就可以完成很多的事情。所以，它比MVC框架考虑的问题要深一步，因为我们程序员大都在写控制层的程序。现在这个工作交给了框架，仅需写很少的调用代码，大大提高了工作效率【6】。

1. 系统需求分析
   1. 项目任务概述

本项目的任务是完成一款能对直播平台弹幕信息智能分析与识别的一个数据爬取分析网站，该网站针对各大著名直播平台，编写爬虫程序对数据进行爬取，将数据进行整合和再处理，对弹幕信息进行识别过滤，包括广告识别以及情色信息识别等功能。

由使用者在前端输入需要访问的直播间的URL，点击获取弹幕按钮执行过滤弹幕功能。弹幕及过滤出的问题弹幕都将实时输出到页面中，所有协议及后台处理隐藏在服务器运行逻辑中，对外部使用者不可见。

* 1. 系统可行性分析
     1. 技术可行性分析

在获取数据可行性分析方面，可以采用python爬虫来处理，python开源社区有成熟的爬虫框架可以简化请求构造的流程，我只需要将关注点放在直播网站弹幕协议的分析上。在非法字符串过滤方面，可以利用python字典构建字典树（TrieTree）选用AC多摸字符串匹配算法来匹配敏感字符串，根据弹幕敏感字符的特点，可以在AC算法上进行敏感字符概率匹配的算法优化。在数据展示可行性方面，可以采用服务器客户端架构，后端采用pythnon Django 框架，将过滤数据展示到前端页面。

总之，这个项目在技术方面完全是可行的，难度适中，既符合时代需求，也符合毕业设计工作量和工作难度。

* + 1. 协议可行性分析

前后端协议通信采用最新的HTML5 WebSocket协议，后端采用django-channels完成服务端WebSocket功能，前端JS自带WebSocket API，可以很轻松的建立起websocket通信。下面是对WebSocket与HTTP协议之间异同的简单介绍：相对于HTTP这种非持久的协议来说，首先，Websocket是一个持久化的协议。简单来说，HTTP的生命周期通过 Request 来界定，也就是一个 Request 一个 Response ，那么在 HTTP1.0 中，这次HTTP请求就结束了。在HTTP1.1中进行了改进，有一个keep-alive，也就是说，在一个HTTP连接中，可以发送多个Request，接收多个Response。但是Request = Response ， 在HTTP中永远是这样，也就是说一个request只能有一个response。而且这个response也是被动的，不能主动发起。但是Websocket只需要一次HTTP握手过程，之后服务端客户端的信息传送就和HTTP协议没有一点关系，全部建立在新型WebSocket协议之上，WebSocket协议像一条管道连接客户端和服务端，双方平等通信，服务端可以主动向客户端推送信息，客户端也可以像服务器传输命令，直到断开WebSocket连接为止，所以说整个通讯过程是建立在一次连接/状态中，也就避免了HTTP的非状态性，服务端会一直知道你的信息，直到你关闭请求，这样就解决了服务器要反复解析HTTP协议，反复查看客户端信息认证的信息的问题。同时，WebSocket提供了很好的协议拓展能力，企业可以在WebSocket协议之上自定义内部的通信协议，如熊猫的弹幕协议和bilibili的弹幕协议。所以弹幕过滤系统采取WebSocket通信是灵活的，稳定的，具有很高的可拓展性的【7】。

* + 1. 经济可行性分析

本项目采用完全开源库开发，不存在支付高额专利费用等问题，只需要租用云服务器进行测试即可。

* + 1. 社会实现可行性分析

社会因素上，从法律方面来看，本系统为独立开发，可以进行知识产权保护，使用的第三方代码库，也均为开源项目，所以在法律方面不存在侵犯专利权、著作权等问题。综上所述，实现这个系统，在社会因素方面是可行的。

* 1. 系统总体业务功能分析
     1. 系统顶层逻辑用例分析

用例图（Use case diagram），是由参与者、用例以及它们之间的关系构成的用于描述系 统功能的动态视图。从外部用户的角度观察，系统应该完成哪些功能，是对系统功能的宏观描述。

图3-1是对整个系统的用例分析，包括用户浏览器，Django后台，和弹幕处理进程之间关系的分析。

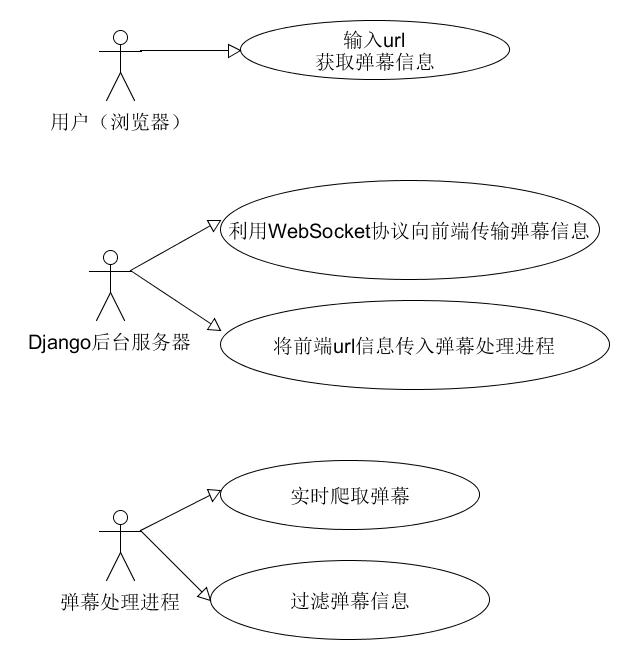


图3-1整个系统的用例分析

* + 1. 系统顶层数据流分析

数据流图（Data Flow Diagram）是展示系统中数据是如何加工处理和流动的，它采用输入-处理-输出的观点对数据流进行建模。下图3-2是系统总体数据流图，包含了弹幕信息获取处理流向等信息。

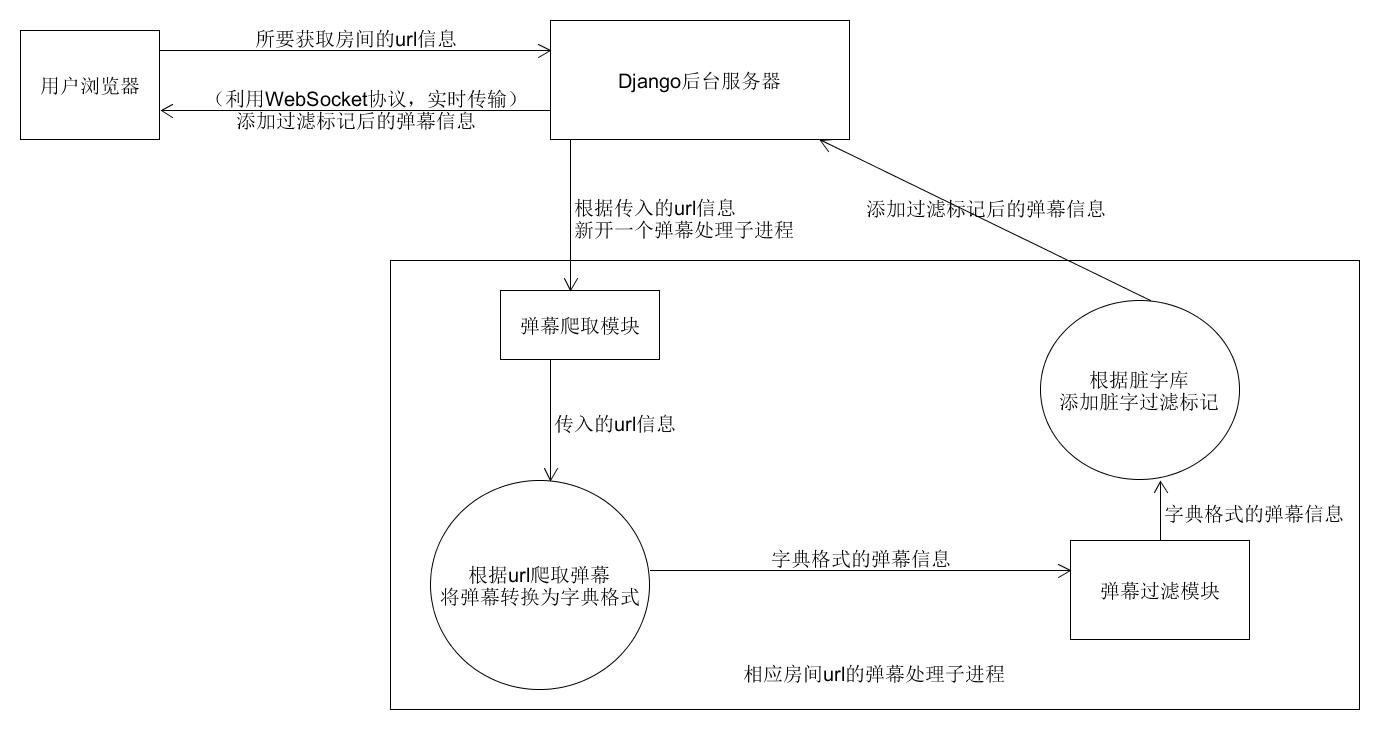


图3-2系统总体数据流图

* 1. 系统模块功能分析
     1. 前端展示模块功能分析

前端展示模块的主要功能如下：（1）提供用户输入url的输入框接口，（2）检验用户输入的url合法性和判定url的平台类型，（3）向后端传入合法的url和url所对应的平台类型（4）向用户展示实时弹幕信息。图3-4为前端展示模块的用例图。

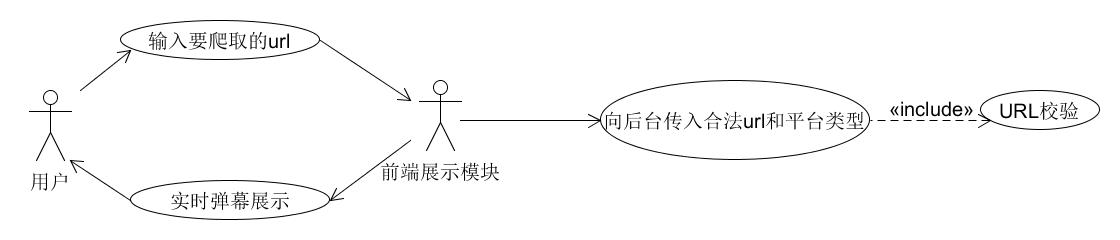


图3-3前端展示模块的用例图

* + 1. 后端响应模块功能分析

后端响应的主要功能如下：（1）接收前端传入的URL和平台类型，（2）根据相应的url获取到相应的直播房间信息（3）根据url和平台信息启动相应的弹幕处理进程（4）向前端模块利用WebSocket协议传输由弹幕处理进程传输过来的实时弹幕信息。图3-5为后端响应模块的用例图。

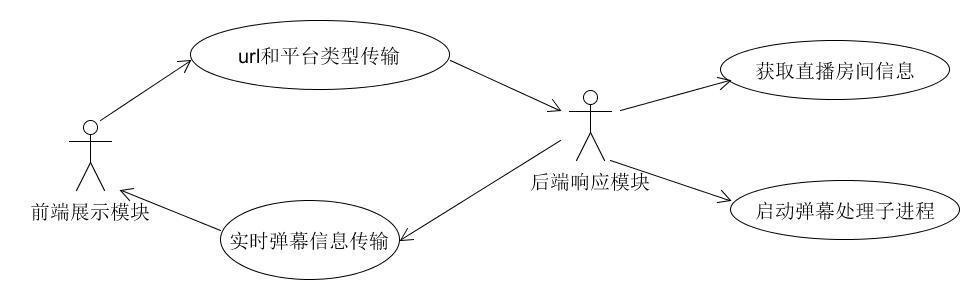


图3-4后端响应模块的用例图

* + 1. 弹幕处理进程处理流程分析

弹幕处理进程模块由如下两个子模块组成：（1）弹幕爬取模块（2）弹幕过滤模块。

弹幕处理的主要流程如下，第一步，根据后台响应模块传入的url和平台类型与对应弹幕服务器建立连接；第二步，等待socket变为可读状态，读取其中的信息；第三步，将读取到的信息格式化为python字典格式；第四步，将字典格式的弹幕信息传入过滤模块；第五步，过滤模块对传入的弹幕字典中弹幕内容域的弹幕内容进行预处理，过滤转化无关字符；第六步，将预处理之后的弹幕内容用相应算法进行检测，对检测出来的敏感字符串添加过滤标记；第七步，将过滤后的弹幕信息传入后端响应模块。

图3-5是弹幕处理模块的流程图。其中流程图（Flow Chart）图是流经一个系统的信息流、观点流或部件流的图形代表。主要作用是用来说明一个过程。

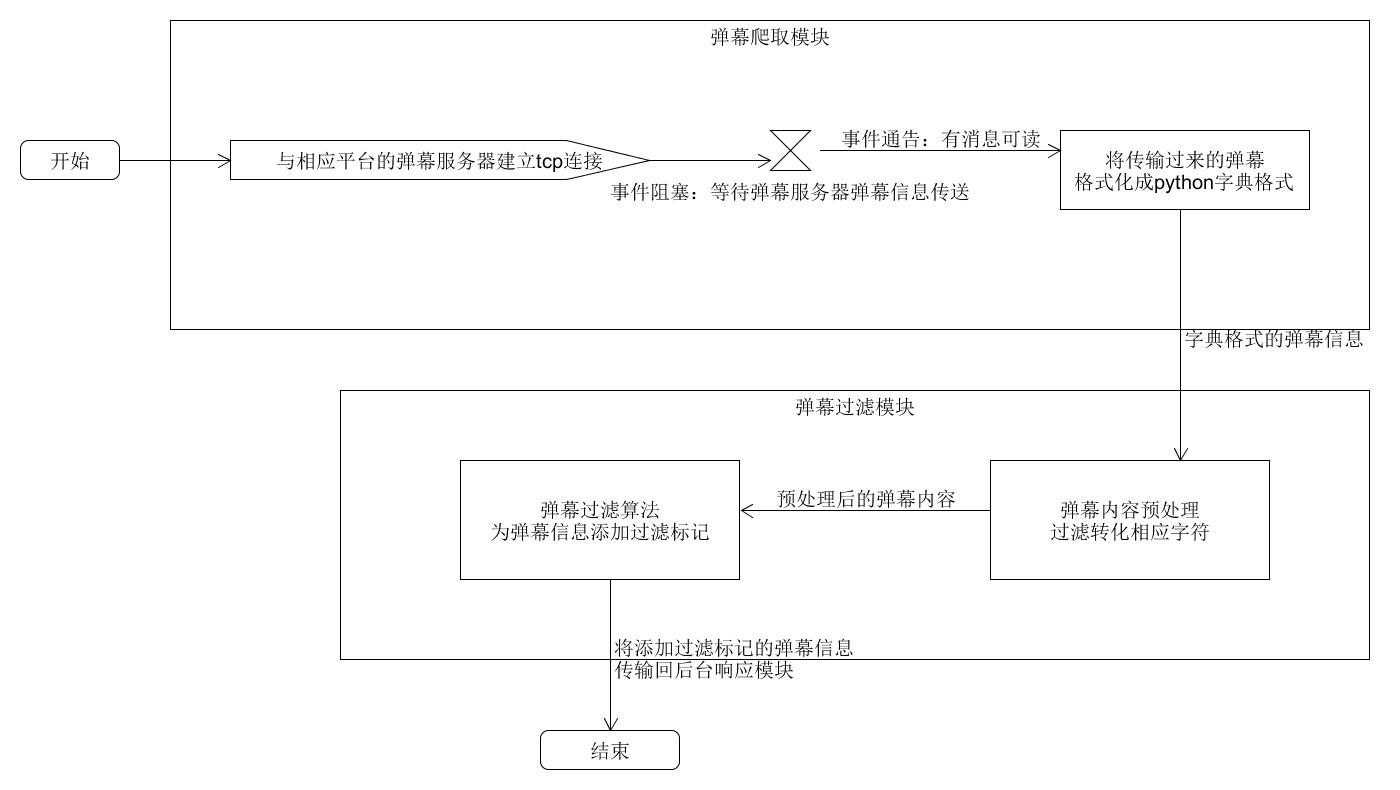


图3-5弹幕处理模块的流程图

* + 1. 弹幕爬取模块功能分析

弹幕爬取模块的功能主要有如下两点，第一点，与平台弹幕服务器建立持久化连；第二点，需要将获取到的弹幕信息利用正则表达式格式化为python字典格式。用例图如下图所示。

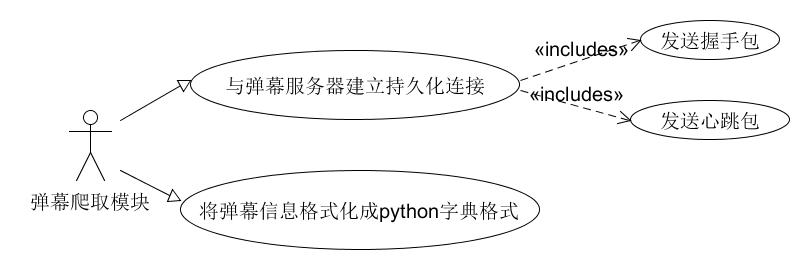


图3-6弹幕爬取模块的用例图

* + 1. 弹幕过滤模块功能分析

弹幕过滤模块的主要功能主要有如下两点，第一点，对弹幕内容字符串进行预处理，包括繁简体转换，大小写转换等功能；第二点，采用过滤算法对预处理之后的弹幕信息字符串进行算法层过滤操作。下图是弹幕过滤模块的用例图。

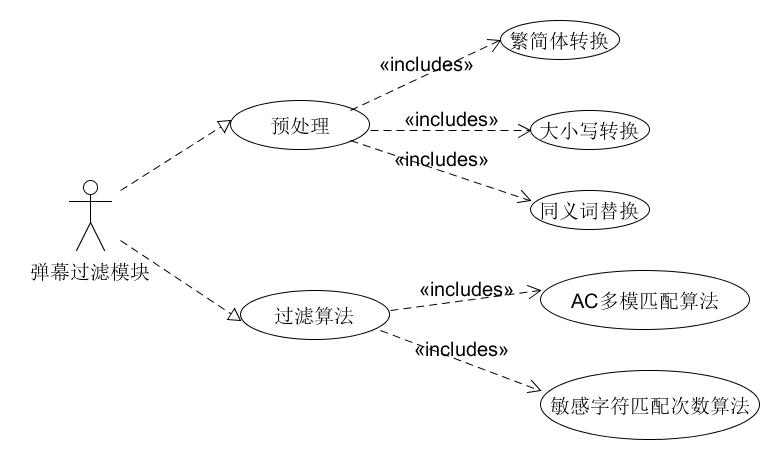


图3-7 弹幕过滤模块的用例图

* 1. 系统非功能性需求
     1. 运行环境要求

此项目搭建在服务器上，要求是跨平台的，后端只要操作系统支持python3.6及以上版本的python环境，和nodejs8.11及以上的node环境，就可以成功搭建服务器；前端需要支持所有chrome内核的浏览器，包括国内360浏览器和QQ浏览器等，支持兼容大多数用户的浏览需求。

* + 1. 设计约束要求

后端脚本语言采用Python开发，采用Django框架，前端页面采用原生js，不使用jquery等封装库，所有接口数据交换采用JSON格式交换。

* + 1. 性能要求

1G内存，单核处理器的服务器同时可以处理10个弹幕过滤请求，稳定工作一天以上，不出现服务器卡死，数据损坏等现象

1. 系统设计
   1. 总体概要设计

此弹幕过滤系统是一个多进程系统，第一个主进程是网站后端进程，子进程是弹幕处理模块的进程。网站后端进程负责与前端的WebSocket协议信息交互，弹幕处理模块子进程负责与直播平台弹幕服务器相连接，获取到直播平台弹幕服务器中的未过滤的原始弹幕信息，将未过滤的弹幕信息传递给过滤模块进行过滤，过滤模块添加过滤标记之后将含有过滤标记的弹幕信息传递回爬取模块，爬取模块将带有过滤标记的弹幕信息传递给服务器主进程，服务器后端主进程将含有过滤过滤信息的弹幕信息传递给前端页面进行渲染，最终，前端页面将渲染结果展示给用户。图3-1是系统整体架构图，显示了网站架构模块，弹幕爬取模块，弹幕过滤模块的整体关系和架构。

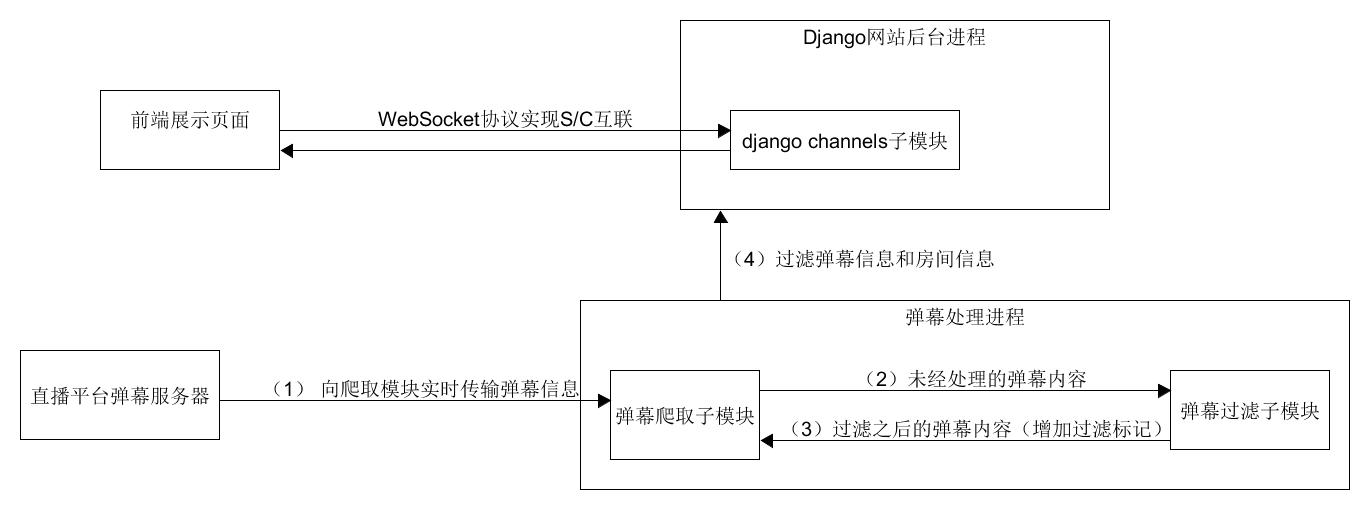


图4-1是系统整体架构图

* 1. 总体时序设计

时序图（Sequence Diagram），又名序列图、循序图、顺序图，是一种UML交互图。它通过描述对象之间发送消息的时间顺序显示多个对象之间的动态协作。它可以表示用例的行为顺序，当执行一个用例行为时，其中的每条消息对应一个类操作或状态机中引起转换的触发事件。下图是弹幕系统顶层时序图。

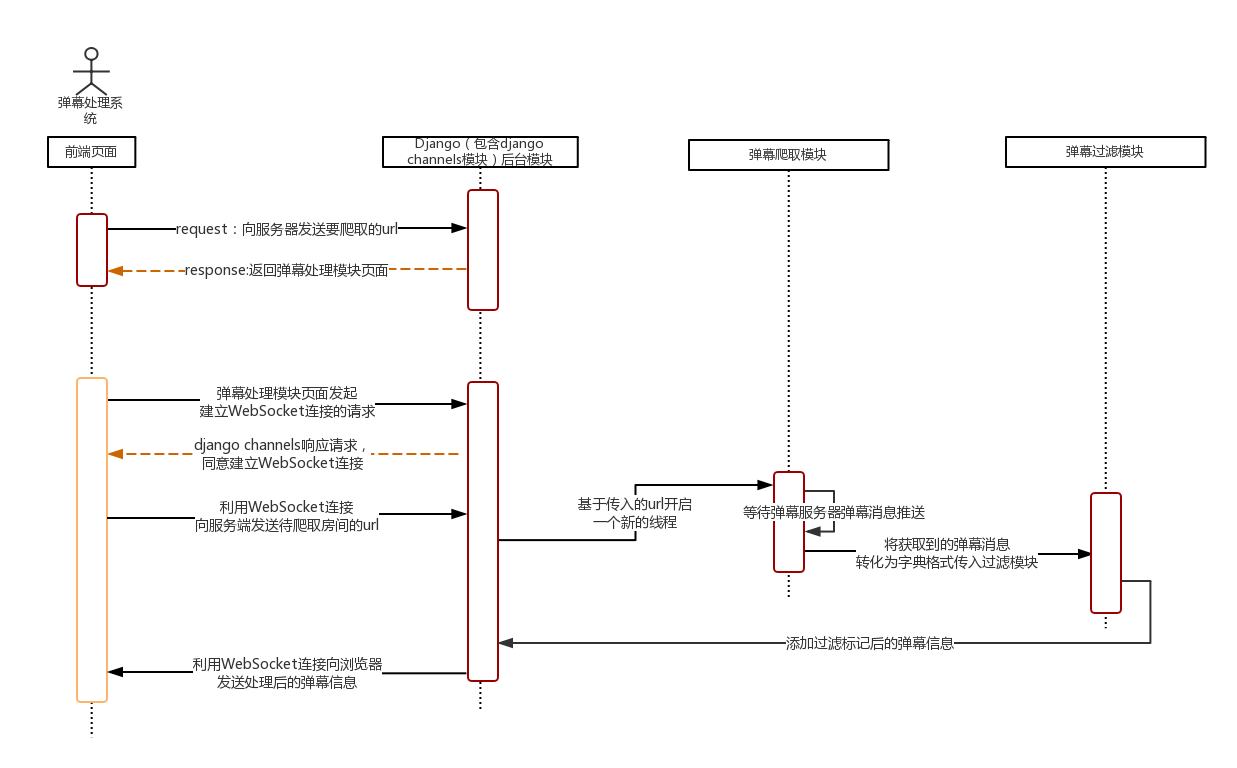


图4-2系统顶层时序图

* 1. 模块详细设计
     1. 系统模块组成设计

如图3-3所示，系统由如下功能模块组成：（1）前端展示展示模块:用于实时展示弹幕信息；（2）后台响应模块:用于响应前端传入的url信息并启动相应弹幕处理子进程；（3）弹幕爬取模块:用于连接相应平台的弹幕服务器，爬取对应房间的弹幕信息；（4）弹幕过滤模块:利用合适的敏感字符串过滤算法，为爬取到的弹幕信息添加过滤标记。

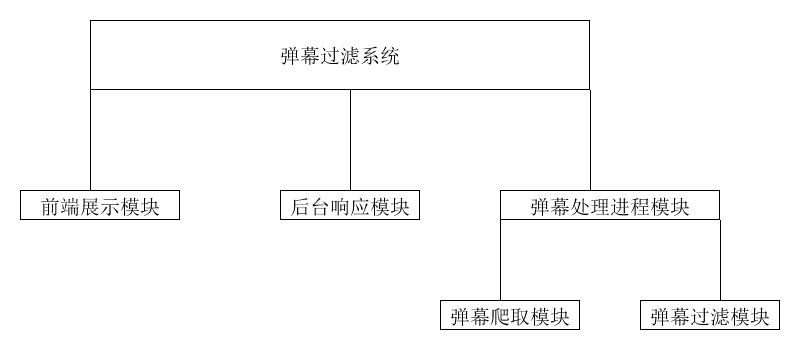


图4-3 系统模块组成

* + 1. 网站架构模块详细设计

首先，在网站主页上有一个输入框来获取用户输入的url，用户点击”开始过滤”按钮，前端页面通过href向后台发送一个get请求，url的query string段中包含了用户输入的合法的url和前端自动检测到输入url中的平台类型 typecode，后台django路由会处理处理这个get请求，将获取到这个get请求中的url参数和typecode参数传入房间信息获取函数，获取到直播间的相关信息，将相关信息渲染到django 模版中返回用户浏览器中，在返回的前端页面中既有通过过滤的滚动弹幕组，也有没有通过过滤的滚动弹幕组，在这个页面包含的JS脚本中，有一个向服务器请求建立的Websocket请求，django 的django channels会响应这个请求和前端页面建立WebSocket连接用于传输弹幕信息，并且，django channels对这个WebSocket请求的处理函数中，会调用弹幕处理子进程来获取其中的弹幕处理信息，将其通过这个WebSocket连接发送至前端页面。其中，django channels 是一款python的异步网络协议框架，它嵌入到django中，当做django的一个插件使用，它使用asgi服务器替换django的wsgi同步阻塞服务器，在django支持的http1.x协议基础上，使django服务端框架支持websocket和 http2协议，实现网站的升级。图4-2是网站架构的详细设计图。

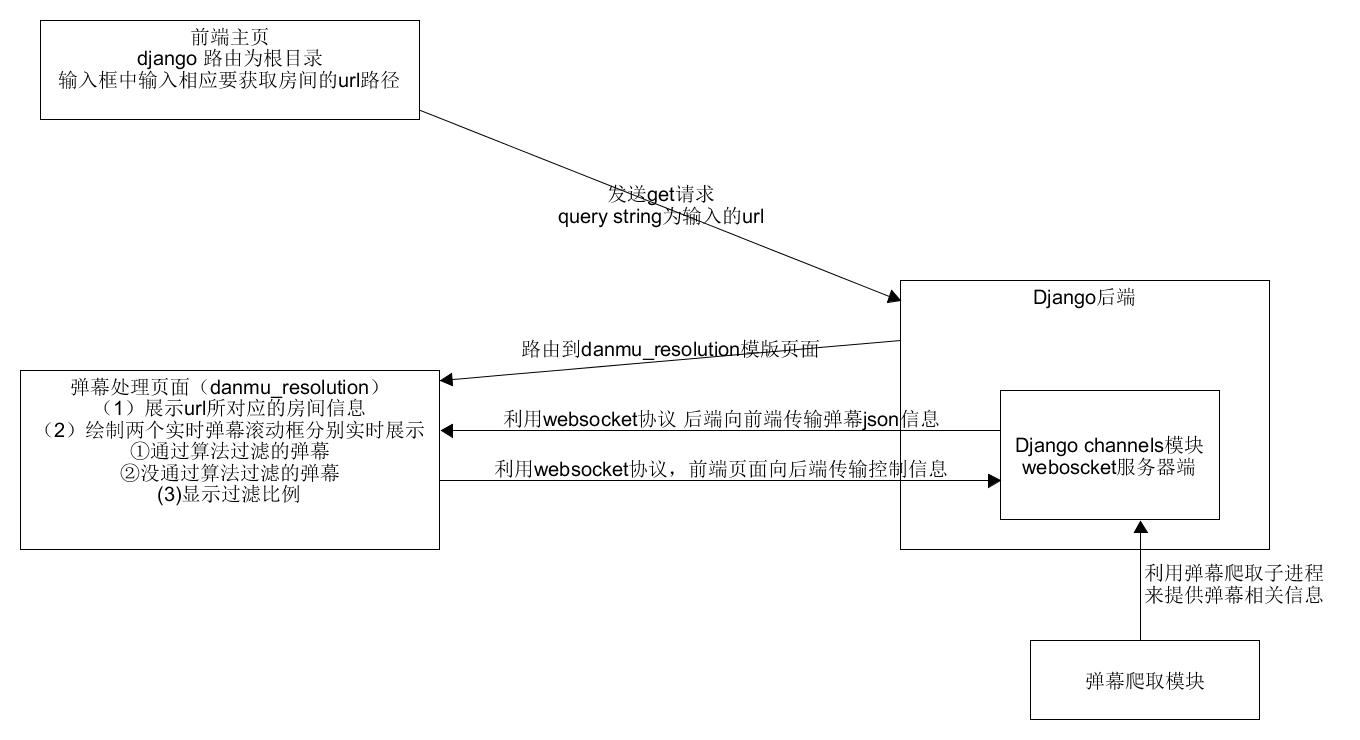


图4-4是网站架构的详细设计图

* + 1. 弹幕爬取模块详细设计

在弹幕爬取模块，第一需要完成对虎牙，斗鱼，熊猫，战旗，bilibili直播模块，全民六大平台的弹幕信息及所对应的主播房间信息的实时抓取。第二，需要对获取到的直播间信息以及实时弹幕信息进行实时格式化操作，格式化成python的字典格式，便于后来与前端网页的json数据交互操作，在python 弹幕信息字典中，将弹幕内容传入过滤模块来添加过滤标志信息实现弹幕的处理过程。

网站架构中弹幕处理页面会将直播间的相关信息通过与网站后端建立的WebSocket协议传入弹幕处理进程，弹幕处理进程中的爬取模块会利用传入的房间信息与平台弹幕服务器握手连接，获取到弹幕信息并交予过滤模块处理，过滤模块对弹幕信息加上过滤标志之后传回弹幕爬取模块，弹幕处理进程通过PIPE（管道）与服务器主进程通信，将处理好的弹幕弹幕通过stdout管道输出到主进程中，完成弹幕信息的交付。图4-3是弹幕爬取模块的详细设计图。

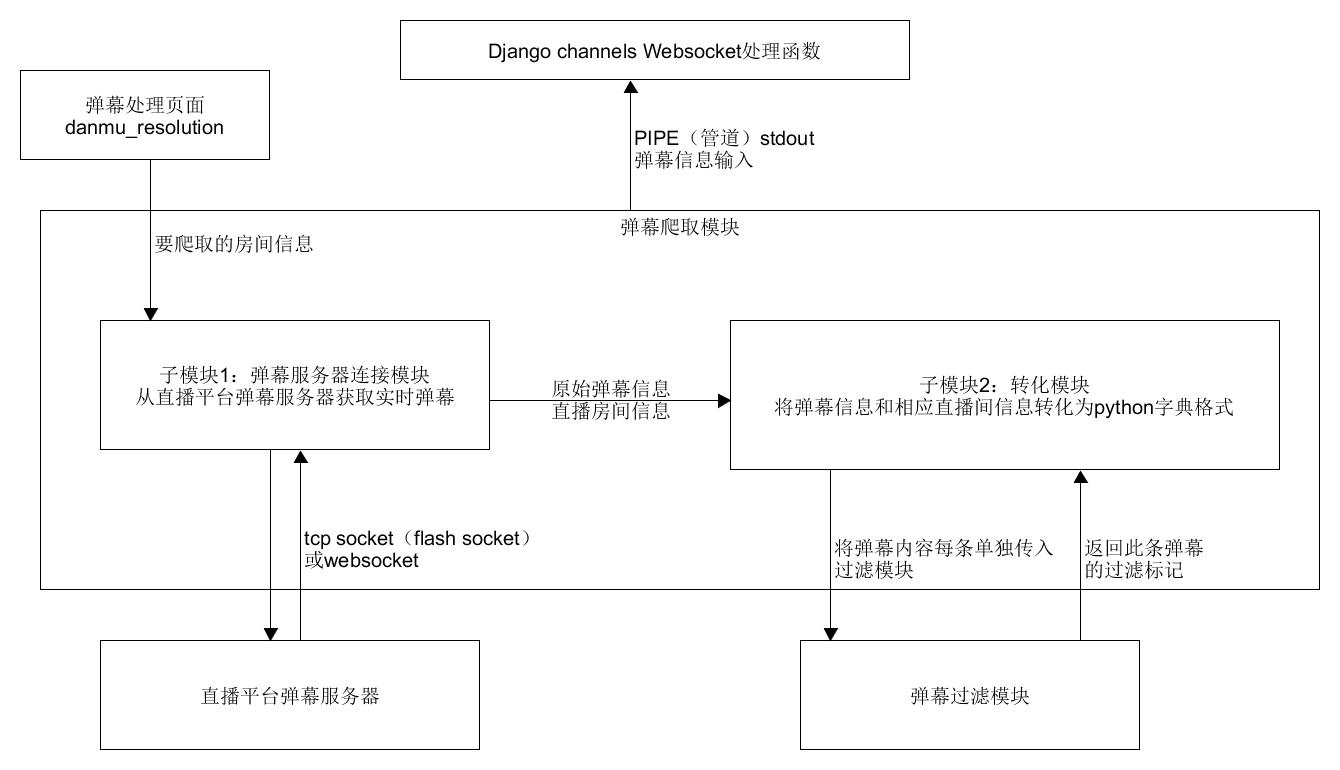


图4-5弹幕爬取模块的详细设计图

* + 1. 弹幕过滤模块详细设计

在敏感字过滤模块，需要对从弹幕爬取模块传入的一条弹幕内容做预处理，首先，将弹幕内容中的繁体转换为简体，其次，将一些对语义没有意义的特殊字符例如!"#$%&\'()\*,-./:;<=>?@等过滤掉，提取出对语义有影响的字符，第三，进行同义词替换，为的是减少脏字字库大小，减少AC算法所查询的时间，最终将预处理之后的弹幕内容利用AC多摸匹配算法和敏感字符命中率算法对字符串进行过滤筛选过滤，对匹配到的敏感字符串进行错误标记传输回弹幕爬取模块。图4-4是弹幕过滤模块的详细设计。其中，AC多摸匹配算法简单介绍如下：AC算法是在1975年贝尔实验室的两位研究人员Alfred V. Aho 和Margaret J. Corasick提出的以他们的名字命名的高效的匹配算法，它是一个经典的多模式匹配算法，可以保证对于给定的长度为n的文本，和模式集合P{p1,p2,…pm}，在O(n)时间复杂度内，找到文本中的所有匹配到的模式集合中的字符串，而与模式集合的规模m无关的高效率算法。

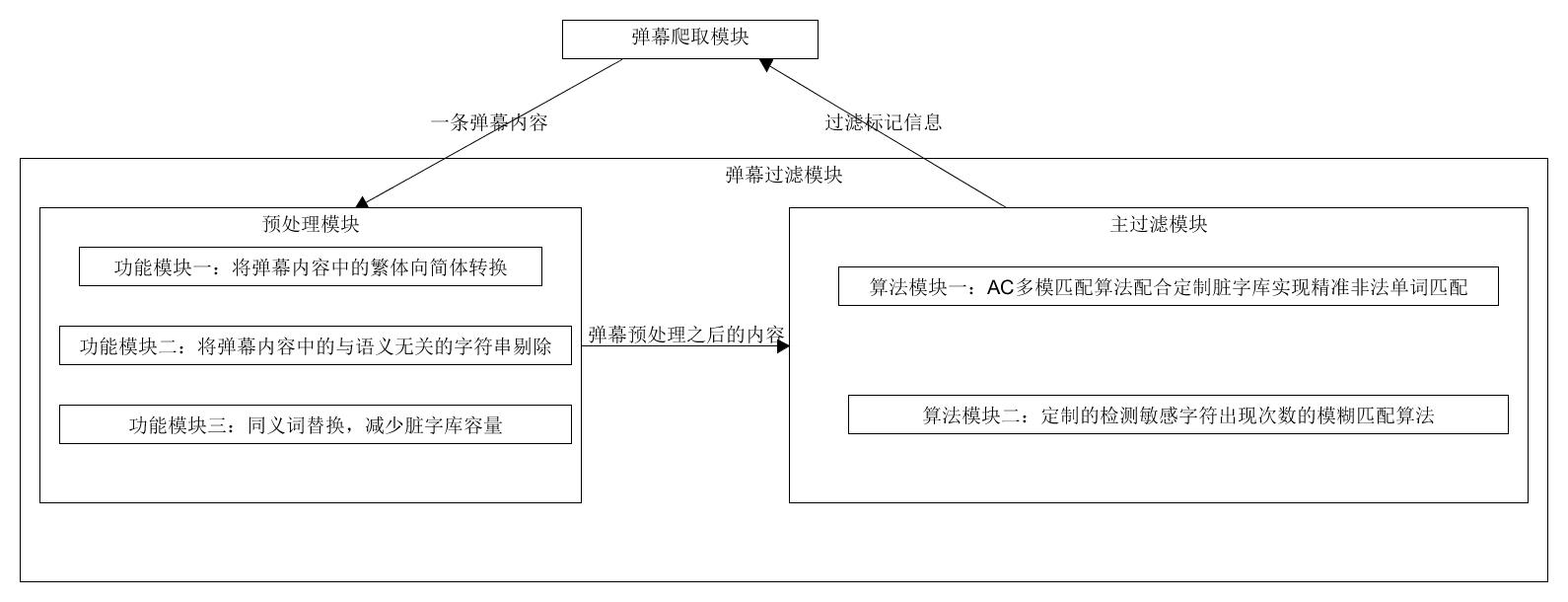


图4-6弹幕过滤模块的详细设计

* 1. AC多模匹配过滤算法设计
     1. 算法原理描述

对敏感字符串的匹配算法采用的是AC多模匹配算法。经典的AC算法由三部分构成，goto表，fail表和output表，共包含四种具体的算法，分别是计算三张查找表的算法以及AC算法本身。goto表是由模式集合P中的所有模式构成的状态转移自动机。failure表作用是在goto表中匹配失败后状态跳转的依据，这点与KMP中next表的作用相似。output表示输出，又称：emits，即代表到达某个状态后某个模式串匹配成功【8】。

实现AC算法的第一步是构造goto表，实际上是构造一棵Trie树。例如，对于模式集合P{he，she，his，hers}， goto表的构建过程如下：

1、PMM初始状态为0，然后向PMM中加入第一个模式串K[0] = “he”。下图是构造goto表的第一步。



图4-7 构造goto表的第一步

2. 继续向PMM中添加第二个模式串K[1] = “she”，每次添加都是从状态0开始扫描。下图是构造goto表的第二步。

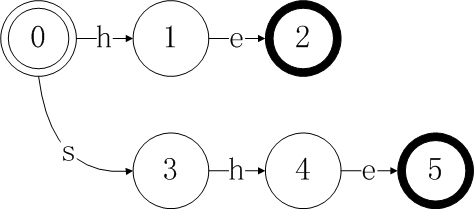


图4-8 构造goto表的第二步

3. 从状态0开始继续添加第三个模式串K[2] = “his”，这里值得注意的是遇到相同字符跳转时要重复利用以前已经生成的跳转。如这里的’h’在第一步中已经存在。下图是构造goto表的第三步。

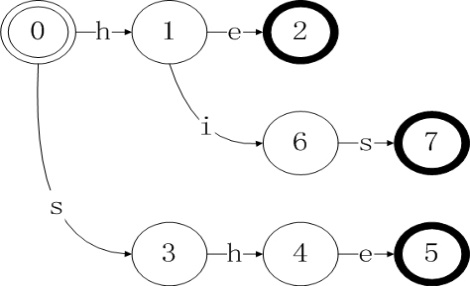


图4-9 构造goto表的第三步

4. 添加模式串K[3] = “hers”。至此，goto表已经构造完成。下图是构造goto表的最后一步。

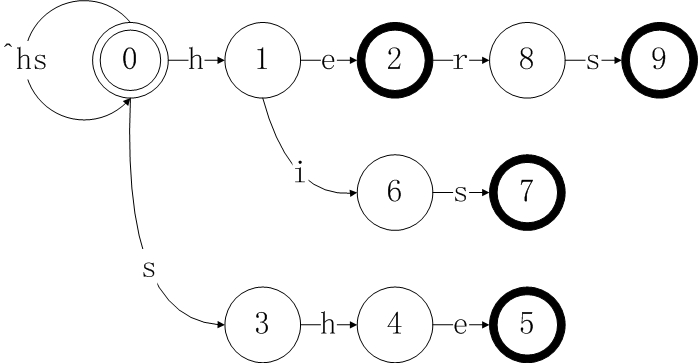


图4-10 构造goto表的最后一步

实现AC算法的第二步是构造failure表，failure表作用是在goto表中匹配失败后状态跳转的依据，这点与KMP中next表的作用相似。 首先说明什么状态，在上面goto表的图里，把圆圈里的数字记为状态。 再引入状态深度的概念，状态s的深度depth(s)定义为在goto表中从起始状态0到状态s的最短路径长度。如goto表中状态1和3的深度为1。

failure表的计算思路为：先计算所有深度是1的状态的失效函数值，然后计算所有深度为2的状态，以此类推，直到所有状态（除了状态0，因为它的失效函数没有定义）的失效函数值都被计算出。

failure的具体实现步骤如下【9】：

1、若节点s为root节点，则将其failure指针指向NULL；

2、若depth(s) = 1，则f(s) = 0；即与根节点（状态0）深度为1的所有状态的failure指针指向根节点（failure指针指向状态0）；

3、假设深度为d-1的所有状态r, 即depth(r) < d，已经计算出了f(r)；

4、那么对于深度为d的状态s：对每个使 g(r, a) = s成立的字符a，执行以下操作:

①使state = f(r)，即取到a字符节点（状态s）的父节点的failure指针指向的节点（状态）；

②重复步骤state = f(state)，直到g(state, a) !=NULL或者state为NULL，当g（state,a）不为NULL时，表示failure指针在回溯的过程中，有一个failure指针指向节点的子节点中包含字符a，这个a子节点的状态表示为g（state,a），所以此时将状态s的failure指针设置为g（state,a），即f（s）= g（state,a）；

当state为NULL时，表示没有在failure指针回溯的过程中找到一个合适的failure节点，此时state直接回溯到了root节点的failure节点，为NULL，所以此时将状态s的failure指针直接设置为root节点即可，即状态0的节点，

f（s）= 0；

将failure表用虚线表现，整合goto表，得到下图：

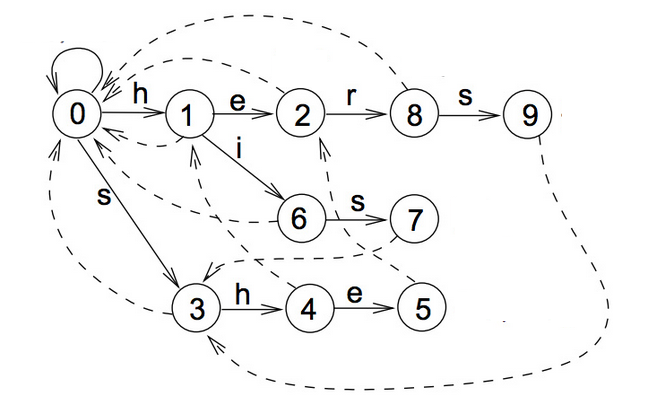


图4-11 对goto表中每个节点添加failure指针

实现AC算法的第三步是构造output表，output表示输出，即代表到达某个状态后某个模式串匹配成功。该表的构造过程融合在goto表和failure表的构造过程中。

1、在构造goto表时，每个模式串结束的状态都加入到output表中，也就goto表中的黑色加粗圆圈。得到 如下表所示。

表4-1 构造goto时得到的output表

|  |  |
| --- | --- |
| i | output(i) |
| 2 | {he} |
| 5 | {she} |
| 7 | {his} |
| 9 | {hers} |

2、在构造failure表时，若f(s) = s’，则将s和s‘对应的output集合求并集。如f(5) = 2，则得到最终的output表为：

表4-2 构造failure表后最终得到的output表

|  |  |
| --- | --- |
| i | output(i) |
| 2 | {he} |
| 5 | {she，he} |
| 7 | {his} |
| 9 | {hers} |

* + 1. 算法实际应用设计

对于这个算法的应用，我们首先要基于非法弹幕的模式建立一个自定义的模式库，里面存有所有脏字字符串的信息，例如，脏字库中有如下脏字模式集合模式集合P{sb，草你sb，cnm，草你大爷}，经过建立goto表，建立failure指针表之后，得到如下所示的AC树和相应的output表，虚线表示failure指针。

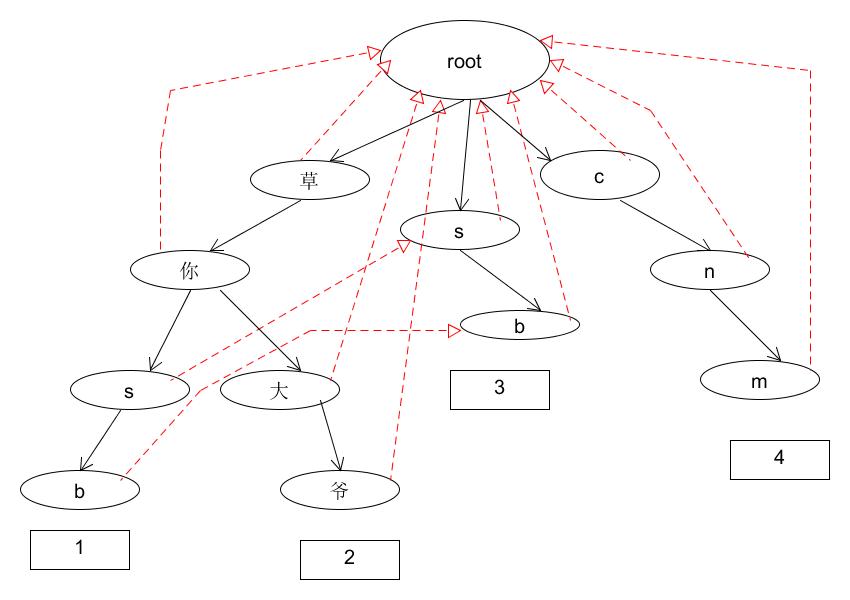


图4-12 AC算法配合脏字字库应用

根据图4-10的AC树，我们得到以下的脏字库output表

表4-3 图4-10的output表

|  |  |
| --- | --- |
| i | output(i) |
| 1 | {草你sb，sb} |
| 2 | {草你大爷} |
| 3 | {sb} |
| 4 | {cnm} |

* 1. 基于敏感字符出现次数过滤算法设计

AC多摸精准匹配算法有一个缺点，就是对掺杂着正常字符的非法字符串很难捕捉到，根据对非法弹幕结构的分析，我总结出了一个模糊匹配算法来捕捉掺杂着正常字符的非法字符串，通过设定敏感字符库来进行判断弹幕内容中命中敏感字符的个数来判断此弹幕内容是否为敏感字符串，此算法主要针对色情广告弹幕很有效，尤其是涉及到qq号，微信号等号码时尤为有效。

目前针对各平台总结出了两种过滤规则，两种过滤规则如下表所示。

表4-4 基于敏感字符出现次数过滤算法的两种过滤规则

|  |  |
| --- | --- |
| 序号 | 规则 |
| 1 | 一个字符串中出现定制脏字库中3个及以上字符则判定为脏字字符串 |
| 2 | 一个字符串中出现数字字符库中超过5个数字字符和1个及以上脏字库中字符则可以判定为qq号，微信号，手机号等色情广告和非法公告等信息 |

注：（1）脏字库和AC算法的脏字库为不同脏字库（2）数字字符库中包含的是各种数字的变体例如：0123456789一二三四五六七八九零①②〇Oo③④⑤⑥⑦⑧⑨ｏ１２３４５６７８９㈠㈡㈢㈣㈤㈥㈦㈧㈨⒈⒉⒊⒋⒍⒌⒎⒏⒐⑴⑵⑶⑸⑹⑺⑻⑼等

1. 系统实现
   1. 弹幕爬取模块实现

这个模块的难点是需要分析各个平台不同的弹幕传输协议，包括握手包的建立，心跳保持包的建立。根据计算机网络原理知识分析和调查得知，大多数的直播网站的弹幕协议一般是基于以下两种基本协议，第一个是flash socket协议，也就是基于flash的原生tcp socket连接，在tcp socket上自定义的弹幕应用层协议；第二个是新型WebSocket协议，在WebSocket协议上自定义弹幕传输协议。所以要基于这两个基本协议的方向来分析各个平台的弹幕协议。下面是各个平台的弹幕爬取分析实现过程。

* + 1. 斗鱼tv弹幕爬取模块实现

第一步，要编写一个获取主播房间信息的函数：斗鱼有两种房间，一种是普通房间，例如，url地址是https://www.douyu.com/dongzhu这种形式的；还有一种是特殊房间，用于斗鱼播放大型赛事活动及特殊主播使用的房间，例如，url地址是https://www.douyu.com/t/carniva l2018?roomIndex=1这种形式的，两种形式的房间获取对应房间信息的方法不同。第一种普通房间直接通过chorme开发者工具查看https://www.douyu.com/dongzhu网页的源代码，可以在javascript域中查看到相应的房间信息。图4-1 网页源代码中普通房间信息的存储位置。

C:\Users\47632\AppData\Roaming\Tencent\Users\476321012\QQ\WinTemp\RichOle\Q@0XLG}2T[E1_@FY1W7`IBW.png

图5-1 网页源代码中普通房间信息的存储位置

第一种的利用python只需要将网页源代码爬取下来进行简单的字符串匹配就可以找到对应的房间json信息，在python中需要用json.loads函数将匹配到的json字符串转换为python字典对象，下面是对应的获取第一种普通房间信息的代码，其中利用到了python的requests爬虫库【10】。

代码5-1 爬取普通房间信息的python方法

|  |
| --- |
| if self.douyu\_prefix not in self.url:  self.douyu\_logger.info("请确定是不是斗鱼直播平台！")  return False  else:  session = requests.session()  session.headers = {  "Accept": "text/html",  "Accept-Encoding": "gzip, deflate, sdch, br",  "Accept-Language": "zh-CN,zh;q=0.8",  "Connection": "keep-alive",  "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36"  }  douyu\_roomHtml=session.get(self.url).text  roomStateJson=json.loads(re.search('var\s\$ROOM\s=\s({.\*});',douyu\_roomHtml).group(1))  #print(roomStateJson)  self.douyu\_roomState["id"]=roomStateJson["room\_id"]  self.douyu\_roomState["roomName"]=roomStateJson["room\_name"]  self.douyu\_roomState["ownerName"]=roomStateJson["owner\_name"]  self.douyu\_roomState["ownerUid"]=roomStateJson['owner\_uid']  return True |

第二种特殊房间的房间信息不存在与网页源代码中，但是网页源代码中有此特殊房间的roomid，代码5-2是特殊房间网页源代码中隐藏roomid的html片段。

代码5-2 特殊房间网页源代码中隐藏roomid的html片段

|  |
| --- |
| <div data-component-id=room data-component-key=18 data-trunkid data-onlineid=288016 |

在python中需要用正则表达式将这个roomid提取出来，代码5-3是python提取特殊房间id的方法。

代码5-3 python提取特殊房间id的方法

|  |
| --- |
| if self.douyu\_special\_prefix in self.url:  headers = {  "Accept": "\*/\*",  "Accept-Encoding": "gzip, deflate, sdch, br",  "Accept-Language": "zh-CN,zh;q=0.8",  "Connection": "keep-alive",  "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36",  "X-Requested-With": "XMLHttpRequest"  }  res=requests.get(self.url,headers=headers)  #print(res.encoding)  res=res.text  result=re.search(r'<div data-component-id=.+?data-onlineid=.\*?(\d+)',res)  self.specialRoomId=result.group(1) |

可以利用这个roomid，向斗鱼服务器发送一个ajax请求来获取相应的房间信息，在chrome开发者工具中的NetWork一栏中，可以找到相应的获取房间信息的请求，下面两张图是从chrome开发者工具中获取到的发送的请求和返回的房间的信息

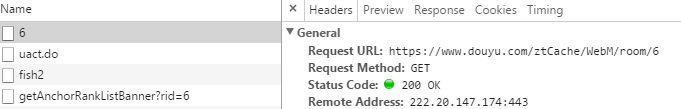


图5-2浏览器向对应地址发送的ajax请求

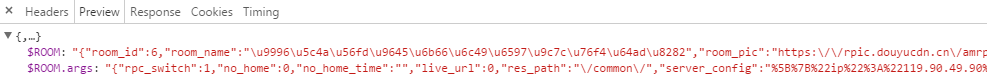


图5-3服务端返回的对应房间的信息

ajax请求的response是一个json对象，用requests库对应的json()函数将其解析为python的字典对象。下面是对应的代码。

代码5-4爬取特殊房间信息的python方法

|  |
| --- |
| if not self.url and self.specialRoomId:  res=requests.get('https://www.douyu.com/ztCache/WebM/room/%s'%self.specialRoomId,headers={  "Accept": "text/html",  "Accept-Encoding": "gzip, deflate, sdch, br",  "Accept-Language": "zh-CN,zh;q=0.8",  "Connection": "keep-alive",  "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36",  "X-Requested-With":"XMLHttpRequest"  }).json()  #print(json.loads(res["$ROOM"]))  self.douyu\_roomState["id"] = json.loads(res["$ROOM"])['room\_id']  self.douyu\_roomState["roomName"] = json.loads(res["$ROOM"])["room\_name"]#.replace(r"\\",'\\').encode('utf-8').decode('unicode\_escape')  self.douyu\_roomState["ownerName"] = json.loads(res["$ROOM"])["owner\_name"]#.replace(r"\\",'\\').encode('utf-8').decode('unicode\_escape')  self.douyu\_roomState["ownerUid"] = json.loads(res["$ROOM"])["owner\_uid"]  #print(self.douyu\_roomState)  return True |

第二步，根据相应的房间信息实时获取相应的弹幕内容。根据斗鱼开源的《斗鱼弹幕服务器第三方接入协议v1.6.2.pdf》中的信息交流格式，需要自己构造socket包与斗鱼弹幕服务器通讯，受 TCP 最大传输单元（MTU）限制及连包机制影响，应用层协议需自己设计协议头，以保证不同消息的隔离性和消息完整性。斗鱼后台协议头设计如下图：

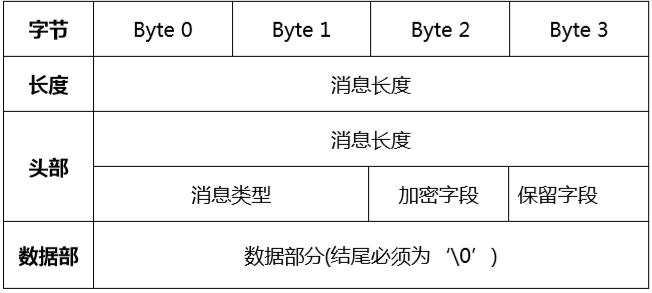


图5-4 斗鱼后台协议头

斗鱼消息协议格式如上所示，其中字段说明如下：

消息长度：4 字节小端整数，表示整条消息（包括自身）长度（字节数）。

消息长度出现两遍，二者相同。

消息类型：2字节小端整数，表示消息类型。取值如下：

689 客户端发送给弹幕服务器的文本格式数据。

690 弹幕服务器发送给客户端的文本格式数据。

加密字段：暂时未用，默认为0。

保留字段：暂时未用，默认为0。

数据部分：斗鱼独创序列化文本数据，结尾必须为‘\0’（所有协议内容均为UTF-8编码）。

下面的python代码是根据斗鱼协议写成的client socket包构造函数:

代码5-5 pyhon client socket包构造函数

|  |
| --- |
| def client\_packaging\_msg(msg):  """  douyu 格式  byte0 byte1 byte2 byte3  len  len  code 加密（0） 保留（0）  数据部分(结尾必须为‘\0’)  code：  689 客户端发送给弹幕服务器的文本格式数据  690 弹幕服务器发送给客户端的文本格式数据。  """  msg\_len = len(msg) + 8  code = 689  encryption = 0  left = 0  all\_the\_msg = msg\_len.to\_bytes(4, byteorder='little') + \  msg\_len.to\_bytes(4, byteorder='little') + \  code.to\_bytes(2, byteorder='little') + \  encryption.to\_bytes(1,byteorder='little') + \  left.to\_bytes(1, byteorder='little') +\  bytes(msg, 'utf-8')  return all\_the\_msg |

连接斗鱼弹幕服务器需要两个步骤，第一个步骤是向斗鱼弹幕服务器发送登陆请求信息。下图是斗鱼客户端登陆请求信息，该消息用于完成登陆授权，完整的数据部分应包含的字段如下【11】：

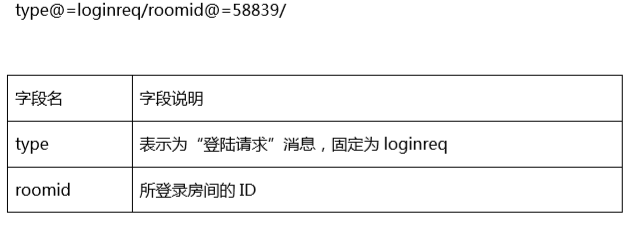


图5-5 斗鱼登陆请求信息规范

基于登陆协议，编写出相应的python代码，下面是对应的python代码：

代码5-6 pyhon 斗鱼登陆代码

|  |
| --- |
| def doLogin(self):  roomId = self.douyu\_roomState["id"]  login\_msg = "type@=loginreq/roomid@={0}/\0".format(roomId)  all\_the\_msg = client\_packaging\_msg(login\_msg)  self.transport.write(all\_the\_msg) |

第二个步骤是入组信息发送，该消息用于完成加入房间分组，完整的数据部分应包含的字段如下图所示：

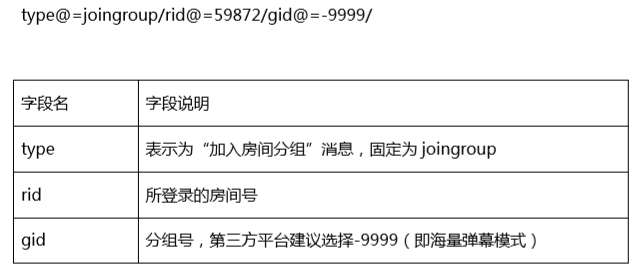


图5-6 斗鱼入组请求信息规范

基于入组协议，编写出相应的python代码，下面是对应的python代码：

代码5-7 pyhon 斗鱼入组代码

|  |
| --- |
| def doJoinGroup(self):  roomId = self.douyu\_roomState["id"]  joinGroup\_msg = "type@=joingroup/rid@={0}/gid@=-9999/\0".format(roomId)  all\_the\_msg = client\_packaging\_msg(joinGroup\_msg)  self.transport.write(all\_the\_msg) |

与服务器建立连接之后，还需要定期向服务器发送心跳信息来维持tcp连接，斗鱼弹幕心跳信息如下

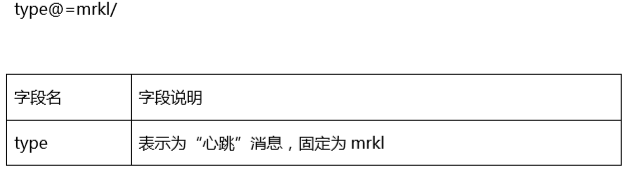


图5-7 斗鱼心跳信息包规范

相应的python代码如下所示：

代码5-8 pyhon斗鱼维持心跳代码

|  |
| --- |
| @classmethod  def keeplive(transport,loop):  keeplive\_msg = "type@=mrkl/\0"  all\_the\_msg = client\_packaging\_msg(keeplive\_msg)  transport.write(all\_the\_msg)  loop.call\_later(delay=30,callback=functools.partial(DouyuDanmuHook.keeplive,transport=transport,loop=loop)) |

弹幕爬取之后是二进制串，需要经过python字典化处理才能便于以后数据交互，下面的代码是斗鱼tv弹幕处理模式：

代码5-9 斗鱼tv弹幕处理模式

|  |
| --- |
| self.danmu\_format = re.compile('chatmsg/rid@=(\d+)/.\*uid@=(\d+)/.\*nn@=(.+)/.\*txt@=(.+)/.\*cid@=(\S+)/ic@=.\*level@=\d{1,3}')  each\_danmu=data.decode('utf-8','ignore')  useful\_segment = self.danmu\_format.search(each\_danmu)  if useful\_segment is not None:  self.each\_danmu\_json["roomId"] = useful\_segment.group(1)  self.each\_danmu\_json["userId"] = useful\_segment.group(2)  self.each\_danmu\_json["userName"] = useful\_segment.group(3)  self.each\_danmu\_json["danmuContent"] = useful\_segment.group(4)  self.each\_danmu\_json["danmuId"] = useful\_segment.group(5)  self.each\_danmu\_json["time"]=time.strftime('%Y-%m-%d %H:%M:%S',time.localtime(time.time())) |

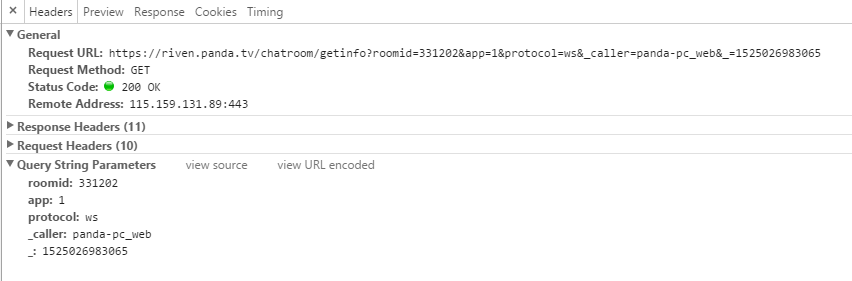
* + 1. 熊猫tv（不包含星秀和星颜模块）弹幕爬取模块实现

熊猫tv没有开源它的弹幕协议，利用一般直播网站的请求架构分析它的弹幕架构：第一步首先在ajax请求中获取弹幕服务器地址，第二步，利用自己网站的协议实现客户端向弹幕服务器发送请求信息，第三步，服务器接收请求信息并分析是否正确，如果正确，就向客户端推送弹幕消息【12】。

第一步，要在ajax请求中找到熊猫tv的弹幕服务器地址。在chrome开发者工具中NetWork一栏查看任一熊猫主播直播页面所发送的ajax请求，都会有一个getinfo的ajax请请求，下图便是这个请求



图5-8 getinfo请求

图5-9 getinfo请求的url和query string

上图中的query string 包含的参数 roomid是熊猫主播的房间号（唯一），这个在浏览器的url中就可以获取到，例如：https://www.panda.tv/331202 其中的33202就是roomid字段，“\_”参数的值是unix时间戳，这个请求返回的是一个json字符串，其中包含弹幕服务器的地址，包括ip地址和端口号，下图就是返回的json字符串，弹幕服务器chat\_addr\_list是一个列表，包含不止一个，这涉及到服务器端的负载均衡，所有服务器都是一样可以获取到全部弹幕。

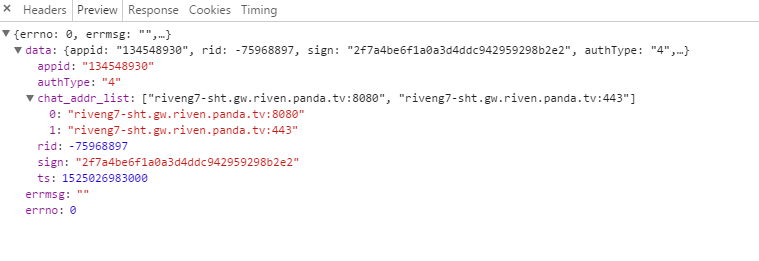


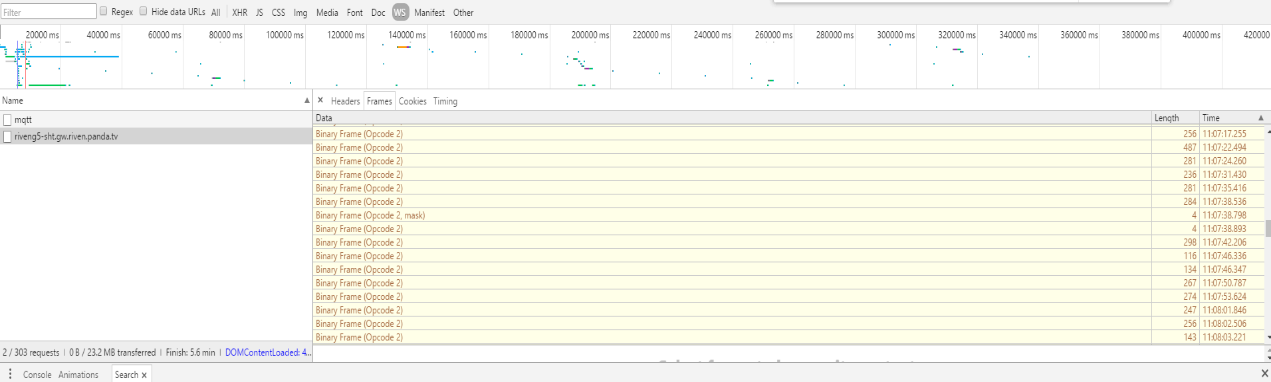
图5-10 getinfo请求返回的弹幕服务器地址chat\_addr\_list

下面的代码是获取熊猫弹幕服务器地址并进行提取的python代码：

代码5-10 pyhon 获取熊猫弹幕服务器代码

|  |
| --- |
| def get\_xiongmaoDanmuServerInfo(self):  roomId\_reg=re.search('https://www.panda.tv/(\d+)',self.url)  if roomId\_reg:  self.roomId=roomId\_reg.group(1)  print(self.roomId)  request\_header={  "Accept": "\*/\*",  "Accept-Encoding":"gzip,deflate,sdch,br",  "Accept-Language": "zh-CN, zh;q=0.8",  "Cache-Control": "max-age = 0",  "Connection":"keep-alive",  "Host" :"riven.panda.tv",  "Origin": "https://www.panda.tv",  "Referer": "https://www.panda.tv/{0}".format(self.roomId),  "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.104 Safari/537.36 Core/1.53.4882.400 QQBrowser/9.7.13059.400"  }  params={  'roomid': self.roomId,  'app': 1,  'protocol': 'ws',  '\_caller': 'panda-pc\_web',  '\_': math.floor(time.time())  }  req=requests.get(url="https://riven.panda.tv/chatroom/getinfo",params=params,headers=request\_header,verify=False)  #params 会转化为url 中的query string verify 表示不验证https协议的证书  #print(req.text)  self.\_\_data=json.loads(req.text)["data"]  self.\_\_danmuServer=self.\_\_data["chat\_addr\_list"][0]  #print(self.\_\_danmuServer) |

第二步，要分析熊猫tv弹幕的握手字段与所用协议，在chrome开发者工具中NetWork一栏查看websocket协议流量情况，发现有弹幕的时候就会有二进制的消息从服务器端传送过来，由此推断，熊猫tv弹幕流量是搭建在websocket协议之上的，下图是chrome websocket流量接收情况：

图5-11 熊猫平台chrome websocket流量接收情况

在熊猫tv的前端中所有js文件中使用在chrome开发者工具中按下Ctrl+Shift+F进入全部源文件搜索模式，搜索关键字符websocket得到如下结果：

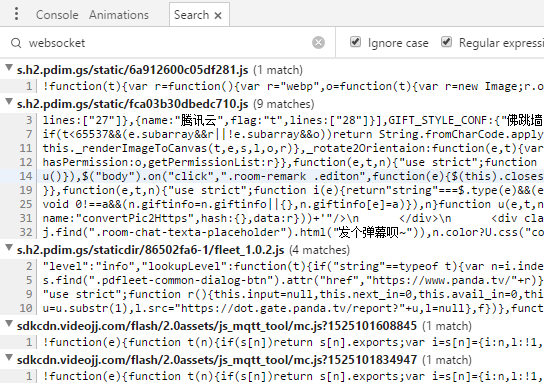


图5-12 chrome全局搜索结果

根据概率匹配原则，先分析关键字命中次数多的文件，将命中9次的文件下载下来并用vscode打开，格式化，寻找关键字websocket，因为弹幕服务器的都是riveng8-sht.gw.riven.panda.tv:443这种形式，所以要找到正确对应的websocket连接，找到onopen方法方法中客户端向服务器请求的初始数据包，下面的代码就是对应的主要代码段：

代码5-11 熊猫前端js代码websocket初始化请求包

|  |
| --- |
| c.prototype.\_initSocket = function () {  var e = this;  this.socket = new WebSocket(this.options.wsProtocal + "//" + this.\_currAddr),  this.socket.onopen = function () {  e.connected = !0, e.\_tryRequestAddrCount = 0, e.\_log("打开连接 riven websocket [" + e.\_currAddr + "]"), e.\_bindUser(), e.\_initHeartbeat(), e.trigger("open")  },  c.prototype.\_bindUser = function () {  var t = e.PANDA\_MONITOR ? e.PANDA\_MONITOR.util.getPlatform() : "",  n = o.cookie.match(/pdft=(\w+)/),  i = "u:" + this.\_chatAddrData.rid +  "@" +this.\_chatAddrData.appid +  "\nts:" + this.\_chatAddrData.ts +  "\nsign:"+this.\_chatAddrData.sign+  "\nauthtype:" + this.\_chatAddrData.authType +  "\nplat:jssdk" +  (t ? "\_" + t : "") +  "\nversion:" + this.version +  "\npdft:" + (n && n[1] ? n[1] : "") +  "\nnetwork:unknown\ncompress:zlib";  this.\_lastBindUser = i;  var a = new dcodeIO.ByteBuffer;  //00060002  a.writeShort(this.ve).writeShort(this.op.binduser).writeShort(i.length).writeUTF8String(i).flip(),  this.socket.send(a.toBuffer()), this.\_log("bind user [" + i.replace(/\n/g, " ") + "]") |

根据前端js与弹幕服务器握手初始化数据包，进行python重写，实现用python封装初始化包向弹幕服务器请求弹幕的功能

代码5-12 python重写熊猫websocket初始化请求包

|  |
| --- |
| def \_loginxiongmaoDanmubyWebSocket(self,ws):  loginmsg="u:{0}@{1}\nts:{2}\nsign:{3}\nauthtype:{4}\nplat:jssdk\_pc\_web\nversion:0.5.9\npdft:\nnetwork:unknown\ncompress:zlib".format(  self.\_\_data['rid'],  self.\_\_data['appid'],  self.\_\_data['ts'],  self.\_\_data['sign'],  self.\_\_data['authType']  )    loginHeader=int(6).to\_bytes(length=2,byteorder='big')+int(2).to\_bytes(length=2,byteorder='big')+len(loginmsg).to\_bytes(length=2,byteorder='big')  self.\_\_ws.send(loginHeader+loginmsg.encode('utf-8'),opcode=websocket.ABNF.OPCODE\_BINARY)  heartbeats = Thread(target=self.\_\_keepalive, name="Send-Heartbeats")  heartbeats.start() |

完成初始化包之后就可以从服务器接收弹幕，但是查看前端js代码，发现每隔30s需要一个心跳包来保持和服务器的连接，下面分别是前端js心跳信息的构造和python代码对心跳信息的重写：

代码5-13 熊猫前端js代码websocket心跳包

|  |
| --- |
| c.prototype.\_initHeartbeat = function () {  var e = this;  clearInterval(this.\_heartbeatTimer), this.\_heartbeatTimer = setInterval(function () {  e.\_heartbeatTimeout = setTimeout(function () {  e.\_report({  id: "105",  host: e.\_currAddr  })  }, 6e4);  var t = new dcodeIO.ByteBuffer;  //00060000  t.writeShort(e.ve), t.writeShort(e.op.heartbeat), t.flip(), e.socket.send(t.toBuffer())  }, this.options.heartbeatInterval) |

代码5-14 python代码重写保持心跳包

|  |
| --- |
| def \_\_keepalive(self):  while self.notDone:  time.sleep(30)  heartbeat = bytes.fromhex("00060000")#心跳信息  self.\_\_ws.send(heartbeat, opcode=websocket.ABNF.OPCODE\_BINARY) |

之后就可以永久实时获取熊猫平台（除星秀星颜模块外）的弹幕。

* + 1. 熊猫tv（星秀和星颜模块）弹幕爬取模块实现

观察星颜和星秀模块的网页中的连接请求，发现没有WebSocket协议流量，可以断定，熊猫平台的星颜和星秀模块不同于它的游戏模块，是采用Flash Socket 与弹幕服务器进行连接。

在网页客户端中使用Flash Socket 必须将.swf文件从服务器端传输到浏览器端，由浏览器运行ActionScript脚本来发起连接进行通信，（注：.swf文件是flash ActionScirpt编译过后的文件，类似于java的.class文件），只需将传输过来的所有.swf文件下载下来，利用ActionScript反编译工具JPEXS Free Flash Decompiler进行反编译，查看其中的源代码，找到建立socket连接的发包过程，就可以在python端进行重写。图5-13是chrome浏览器查看到的从服务器传输过来的所有swf文件。

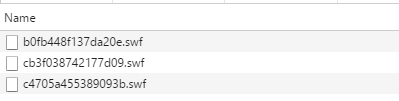


图5-13 chrome浏览器查看到的从服务器传输过来的所有swf文件

用Free Flash Decompiler对swf文件进行反编译，在b0fb448f137da20e.swf文件中发现了与弹幕服务器建立连接的相关反编译代码。代码5-15是反编译后flash socket 获取弹幕服务器地址的代码。

代码5-15 反编译后flash socket 获取弹幕服务器地址的代码

|  |
| --- |
| private function getSocketServer() : \*  {  var \_loc1\_:urlTools = new urlTools();  var \_loc2\_:Object = new Object();  \_loc2\_.guid = this.guid;  /\*gVars.guid = this.guid =  "7773" +  ("0000000000000000" +  now.getTime().toString(16)).substr(-16) +  ("000000000000"+  int(Math.random()\*1000000000).toString(16)).substr(-12);  \*/  \_loc2\_.time = Math.round(new Date().getTime() / 1000);  \_loc2\_.cluster = "v3";  \_loc2\_.plat = "pc\_web";  \_loc2\_.xid = this.xid;  /\*xid为对应的直播间id\*/  if(this.xid == null)  {  \_loc2\_.xid = "8352845";  }  var \_loc3\_:String = "uzY@H/C!N^G9K:EY" + \_loc2\_.guid + \_loc2\_.cluster + \_loc2\_.time;  \_loc2\_.sign = MD5.hash(\_loc3\_);  var \_loc4\_:Object = new Object();  \_loc1\_.addEventListener(cEvent.GOTDATA,this.goSocket);  \_loc1\_.requesst(this.protocol + "//online.panda.tv/dispatch",\_loc2\_);  } |

其中，xid为对应的直播间id，guid为熊猫用自己的算法生成的标识符，this.protocol为http,请求地址为//online.panda.tv/dispatch，用python重写获取弹幕服务器地址代码如下。

代码5-16 python重写获取弹幕服务器地址代码

|  |
| --- |
| def getSRInfo(self):  search\_result=re.search(r'https://xingyan.panda.tv/(\d+)',self.url)  if search\_result:  self.roomId=search\_result.group(1)  getPayLoad={}  getPayLoad["guid"]=self.guid  getPayLoad["time"]=int(time.time())  getPayLoad["cluster"]="v3"  getPayLoad["plat"]="pc\_web"  getPayLoad["xid"]=self.roomId  hmd5=hashlib.md5()  hmd5.update(("uzY@H/C!N^G9K:EY" + getPayLoad["guid"] + getPayLoad["cluster"] + str(getPayLoad["time"])).encode())  getPayLoad["sign"]=hmd5.hexdigest()  #print(getPayLoad)  res=requests.get(url="https://online.panda.tv/dispatch",params=getPayLoad,headers={  "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36"  }).json()  print(res)  self.serverInfo['ip']=socket.gethostbyname(res["addr"])  self.serverInfo['port']=res['port']  self.loginInfo['rnd']=res['rnd']  self.loginInfo['id']=res['id']  print(self.serverInfo,self.loginInfo) |

服务器返回的response是一个json字符串，json字符串如下。

代码5-17 服务器返回的json字符串

|  |
| --- |
| {'addr': 'v3b9.xingyan.panda.tv', 'port': '56002', 'baddr': 'v3b14.xingyan.panda.tv', 'bport': '80', 'timeout\_frozen': 7200, 'timeout\_break': 7, 'timeout\_ping': 60, 'rnd': '000000000000000083231700d1a7035b0628859f', 'id': '5b03a7d12fcf11e2', 'error': '', 'time': 1526966225} |

其中，字段addr port是第一个弹幕服务器的ip和端口；字段baddr，bport是备用弹幕服务器ip和端口，timeout字段是重连参数，rnd和id字段是服务器返回的唯一标识id。

获取到弹幕服务器信息之后，需要与弹幕服务器进行握手连接，反编译的ActionScript的函数中有一个是监听建立连接之后的函数，查看之后，发现是向弹幕服务器发送握手包的函数，它的代码如下：

代码5-18 ActionScript代码向弹幕服务器发送握手包

|  |
| --- |
| this.socket.addEventListener(Event.CONNECT,this.connectHandler);  public function connectHandler(param1:Event) : \*  {  var \_loc2\_:ByteArray = new ByteArray();  \_loc2\_.writeBytes(this.toArray(gVars.guid),0,16);  \_loc2\_.writeBytes(this.toArray(this.info.rnd),0,20);  this.sendBinary(2147483649,\_loc2\_);  this.connected = true;  this.connectCooldown.stop();  var \_loc3\_:Object = new Object();  \_loc3\_.type = "Log";  \_loc3\_.value = "socket sucess";  ExternalInterface.call("FLASH\_PROCESS.sendToJavaScript",JSON.stringify(\_loc3\_));  }  public function sendBinary(param1:uint, param2:ByteArray) : \*  {  var \_loc3\_:ByteArray = new ByteArray();  \_loc3\_.writeByte(3);  \_loc3\_.writeByte(0);  \_loc3\_.writeByte(0);  \_loc3\_.writeByte(0);  \_loc3\_.writeDouble(Math.random());  \_loc3\_.writeUnsignedInt(param1);  \_loc3\_.writeUnsignedInt(0);  \_loc3\_.writeUnsignedInt(param2.length);  this.socket.writeBytes(\_loc3\_);  this.socket.writeBytes(param2);  trace("send: header:" + this.fromArray(\_loc3\_));  trace("send: body:" + this.fromArray(param2));  this.socket.flush();  trace("sent");  }  private function toArray(param1:String) : ByteArray  {  param1 = param1.replace(/^0x|\s|:/gm,"");  var \_loc2\_:ByteArray = new ByteArray();  if((param1.length & 1) == 1)  {  param1 = "0" + param1;  }  var \_loc3\_:uint = 0;  while(\_loc3\_ < param1.length)  {  \_loc2\_[\_loc3\_ / 2] = parseInt(param1.substr(\_loc3\_,2),16);  \_loc3\_ = \_loc3\_ + 2;  }  return \_loc2\_;  } |

其中gVars.guid是上面已经生成的guid，this.info.rnd是上面请求服务器地址时返回的标识字符串，toArray函数的作用是将传入的字符串两个16进制字符字符并为一组，将它看做一个字节传入到一个数组中，而gVars.guid是32个字符，所以数组中有16个字节，this.info.rnd中有40个字符，所以数组中有20个字节。以字节流形式将握手包发送给弹幕服务器，服务器就可以返回字节流形式的弹幕信息。代码5-19是python重写向弹幕服务器发送握手包的函数。

代码5-19 python重写向弹幕服务器发送握手包的函数。

|  |
| --- |
| def toBytesList(self,str):  i=0  byteslist=[]  substrlist=[]  if len(str)&1==1:  str='0'+str  while i<len(str):  substrlist.append(str[i:i+2])  i+=2  for i,val in enumerate(substrlist):  byteslist.append(struct.pack('B',int(val,16)))  return byteslist  def makeLoginMsg(self):  params={}  loginBytes=bytearray()  loginBody=bytearray()  params["guid"]=self.toBytesList(self.guid)  params["rnd"]=self.toBytesList(self.loginInfo['rnd'])  loginBody+=b''.join(params['guid'])  loginBody += b''.join(params['rnd'])  paramslength=len(loginBody)  loginheader=struct.pack('>BBBBdIII',3,0,0,0,random.random(),2147483649,0,paramslength)  print(loginheader)  print(loginBody)  loginBytes=loginheader+loginBody  #print(loginBytes)  return bytes(loginBytes) |

其中，struct.pack函数是可以将整数，浮点数等打包成字节格式进行封包操作的函数。

弹幕爬取之后是二进制串，需要经过python字典化处理才能便于以后数据交互，下面的代码是熊猫tv星秀星颜模块的弹幕处理模式：

代码5-20 熊猫tv星秀星颜模块的弹幕处理模式

|  |
| --- |
| self.compile=re.compile(r'{"from":.+?"type":"chat",.+?"plat":".+?"}')  def data\_received(self, data):  #print(data.decode('utf-8','ignore'))  result=self.compile.search(data.decode('utf-8','ignore'))  if result:  usefuljson=json.loads(result.group(0))  self.eachDanmuJson['nick']=usefuljson['from']['nick']  self.eachDanmuJson['rid']=usefuljson['from']['rid']  self.eachDanmuJson['role\_name']=usefuljson['from']['role\_name']  self.eachDanmuJson['danmuContent']=usefuljson['data']['text']  print(self.eachDanmuJson) |

* + 1. 战旗tv弹幕爬取流程实现

第一步，获取弹幕服务器信息，查看战旗主播的房间网页源代码可以发现弹幕服务器列表，下面的代码就是战旗tv的网页源代码服务器列表部分：

代码5-21 战旗tv的网页源代码服务器列表部分

|  |
| --- |
| window.oPageConfig.oRoom = {"id":"268645","uid":"401578259","nickname":"暖芯真的不暖心","gender":"2","avatar":"https:\/\/img2.zhanqi.tv\/avatar\/5d\/6e5\/401578259\_1524190878.jpg","code":"873061670","url":"\/873061670","title":"战旗暖芯暖心，盲女溜五台",……. |

Server信息是base64编码过后的结果，需要爬取此部分代码后进行base64解码还原真正的弹幕服务器地址，下面是python对应功能的代码：

代码5-22 python解析战旗tv弹幕服务器地址

|  |
| --- |
| request\_headers={  "Accept": "text/html",  "Accept-Encoding": "gzip, deflate, sdch, br",  "Accept-Language": "zh-CN,zh;q=0.8",  "Connection": "keep-alive",  "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36"  }  res=requests.get(self.\_\_url,headers=request\_headers)  raw\_search=re.search('oRoom = (.\*);[\s\S]\*?window.',res.text)  if not raw\_search:raw\_search=re.search('aVideos = (.\*);[\s\S]\*?oPageConfig.', res.text)  if raw\_search:  serverInfo=json.loads(raw\_search.group(1))  if isinstance(serverInfo,list):serverInfo\_json=serverInfo[0]  else:serverInfo\_json=serverInfo  self.\_\_roomId=serverInfo\_json['id']  serverAddress=json.loads(base64.b64decode(serverInfo\_json['flashvars']['Servers']).decode('utf-8'))['list'][0]  self.serverIp=serverAddress['ip']  self.serverPort=serverAddress['port'] |

在chorme开发者工具的NetWork面板的ws一栏没有发现websocket流量，可以断定，战旗tv采用原生flash socket 的方法与弹幕服务器进行握手交互操作，用wireshark显示过滤抓包操作，在相关端口过滤条件下可以发现如下图所示的由客户端发送的原生tcp套接字包：

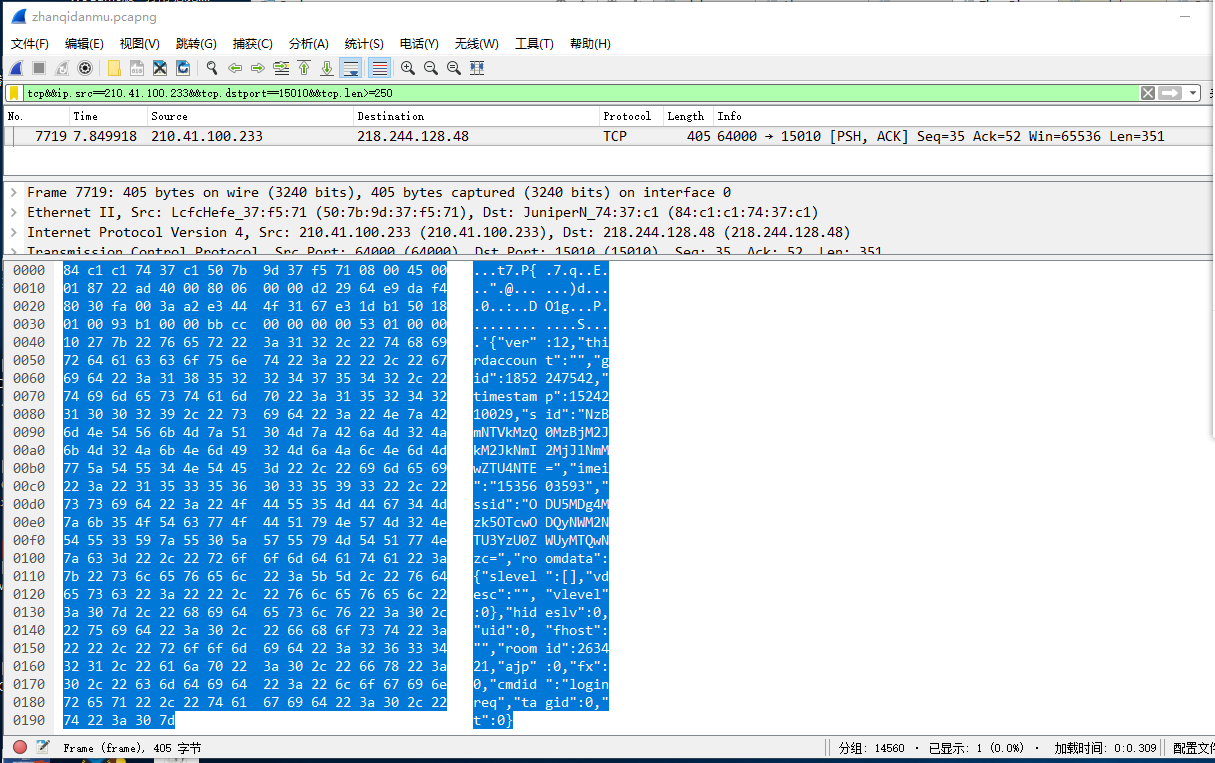


图5-14 wireshark抓取的客户端握手包

其中请求的相关字段可以在chrome开发者工具NetWork XHR一栏（ajax请求）的https://www.zhanqi.tv/api/public/room.viewer请求的response中获得，其中uid就是上面网页源代码部分中的uid值，\_t是unix时间戳除以60的值，图5-15,5-16就是这个请求的url,query string和response body：

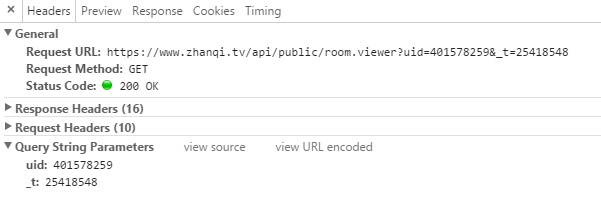


图5-15 请求的url,query string

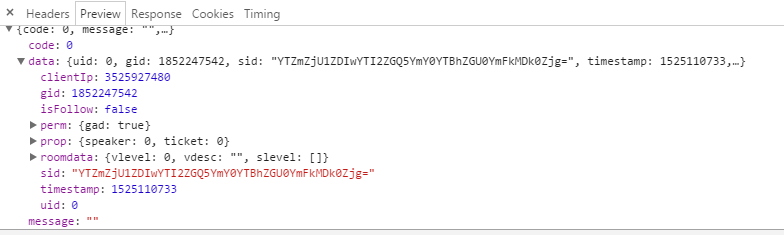


图5-16 请求的response body

用python来构造握手包如下代码所示：

代码5-23 python构造原生tcp握手包

|  |
| --- |
| res=requests.get('%s/api/public/room.viewer' % 'https://www.zhanqi.tv',params={  'uid':serverInfo\_json['uid'],  '\_t':int(time.time())/60,  },headers=request\_headers)  self.\_\_roomInfo=res.json()  def \_\_genLoginMsg(self):  content=json.dumps({  'nickname': '',  'roomid': int(self.\_\_roomId),  'gid': self.\_\_roomInfo['data']['gid'],  'sid': self.\_\_roomInfo['data']['sid'],  'ssid':self.\_\_roomInfo['data']['sid'],  'timestamp': self.\_\_roomInfo['data']['timestamp'],  'cmdid': 'loginreq',  'develop\_date': '2015-06-07',  'fhost': 'zhanqi.tool',  'fx': 0,  't': 0,  'thirdacount': '',  'uid': 0,  'ver': 2,  'vod': 0,  })  header=b'\xbb\xcc' + b'\x00'\*4+len(content).to\_bytes(length=4,byteorder='little')+b'\x10\x27'  return header+content.encode('utf-8') |

经过wireshark抓包分析，可以得到与弹幕服务器的心跳信息，下面是用python构造的心跳函数：

代码5-24 python构造原生tcp心跳包

|  |
| --- |
| @classmethod  def keepalive(cls,transport,loop):  transport.write(b'\xbb\xcc' + b'\x00'\*8 + b'\x59\x27')  loop.call\_later(delay=30, callback=functools.partial(cls.keepalive, transport=transport, loop=loop)) |

弹幕爬取之后是二进制串，需要经过python字典化处理才能便于以后数据交互，下面的代码是战旗tv弹幕处理模式：

代码5-25 战旗tv弹幕处理模式

|  |
| --- |
| self.compile=re.compile('"fromname":"(.+?)","content":"(.+?)","cmdid":"chatmessage"}')  compile\_obj=self.compile.search(data.decode('utf-8','ignore'))  if compile\_obj:  self.each\_danmu["userName"]=compile\_obj.group(1)  self.each\_danmu["danmuContent"]=compile\_obj.group(2)  self.each\_danmu["time"]=time.strftime('%Y-%m-%d %H:%M:%S') |

* + 1. 虎牙tv弹幕爬取流程实现

由于虎牙tv前端解析弹幕信息的代码过于复杂，所以采用Nodejs来在本地封装前端js解析代码，用nodejs来发送websocket握手协议和接收服务器推送过来的弹幕流量，之后采用python的subprocess来调用nodejs代码实现数据向python端传输的功能。

爬取虎牙弹幕第一步需要根据url获取虎牙直播间信息，以便构造相应的弹幕协议握手包，与弹幕服务器进行连接。下面代码的功能就是根据url获取虎牙直播间信息。

代码5-26 获取虎牙直播间信息

|  |
| --- |
| async \_get\_chat\_info() {  try {  let body = await r({  url: `https://m.huya.com/${this.\_roomid}`,  agent: this.\_agent  })  let info = {}  let subsid\_array = body.match(/var SUBSID = '(.\*)';/)  let topsid\_array = body.match(/var TOPSID = '(.\*)';/)  let yyuid\_array = body.match(/ayyuid: '(.\*)',/)  if (!subsid\_array || !topsid\_array || !yyuid\_array) return  info.subsid = subsid\_array[1] === '' ? 0 : parseInt(subsid\_array[1])  info.topsid = topsid\_array[1] === '' ? 0 : parseInt(topsid\_array[1])  info.yyuid = parseInt(yyuid\_array[1])  return info  } catch (e) {  this.emit('error', new Error('Fail to get info'))  }  } |

第二步与虎牙弹幕服务器建立WebSocket连接，发送相应的握手包即可完成连接。下面的代码是完成与弹幕服务器的握手连接过程。

代码5-27 与弹幕服务器建立WebSocket连接

|  |
| --- |
| \_start\_ws() {  this.\_client = new ws('ws://ws.api.huya.com', {  perMessageDeflate: false,  agent: this.\_agent  })  this.\_client.on('open', () => {  this.\_get\_gift\_list()  this.\_bind\_ws\_info()  this.\_heartbeat()  this.\_heartbeat\_timer = setInterval(this.\_heartbeat.bind(this), heartbeat\_interval)  this.\_fresh\_gift\_list\_timer = setInterval(this.\_get\_gift\_list.bind(this), fresh\_gift\_interval)  this.emit('connect')  })  this.\_client.on('error', err => {  this.emit('error', err)  })  this.\_client.on('close', async () => {  this.\_stop()  this.emit('close')  })  this.\_client.on('message', this.\_on\_mes.bind(this))  }  } |

python端使用subprocess模块 subprocess.PIPE管道定义nodejs进程的标准输入流和输出流实现和python进程的通信：

代码5-28 python进程通信代码

|  |
| --- |
| class HuyaDanmuHook():  def \_\_init\_\_(self,huyaUrl):  self.huya\_prefix="https://www.huya.com/"  self.roomId=huyaUrl.replace(self.huya\_prefix,"")  print(self.roomId)  self.jsSubprocess=subprocess.Popen(['node','test.js',self.roomId],  stdin=subprocess.PIPE,  stderr=subprocess.PIPE,  stdout=subprocess.PIPE,  shell=True,  encoding="utf-8"#加上这个参数之后stdout会把相应的bytes对象解码转换为str  #stdin会把相应的str对象 编码为相应的bytes对象  )  self.huyaDanmuDict={}  def readDamuLine(self):  while True:  jsonStr=self.jsSubprocess.stdout.readline()  print(jsonStr)  client.start() |

* + 1. 全民tv弹幕爬取流程实现

全民在url中的房间id不是房间弹幕组id，真正的id隐藏在网页源代码中，代码5-29就是真正的房间弹幕组id，其中的uid就是真正的弹幕组id:

代码5-29 全民网页源代码中隐藏的弹幕组id（uid）

|  |
| --- |
| new room($('.room\_p-room\_wrap'), {"uid":1939452458,"no":27760536,"noType":0,"title":"国服第一豹女，打野思路讲解","nick":"萧兮丶",……….. |

在获取到弹幕组id之后，需要找到全民tv弹幕服务器，经过查阅资料并进行验证，发现需要向http://www.quanmin.tv/site/route发送请求，返回的二进制串取前16位，每四位一体，采用Big Endian模式将其转换为四个int类型的数据，再与172异或，便能得到真正的全民弹幕服务器地址,弹幕端口经过查阅资料，均为9098端口，将弹幕组id封包发送给全民tv弹幕服务器就可以接收服务器向此弹幕组id的弹幕信息。代码5-30就是获取弹幕服务器和uid与封包的过程。

代码5-30 获取全民弹幕服务器，uid，封包代码

|  |
| --- |
| def createLoginMsg(self):  content = '{\n' \  ' "os" : 135,\n' \  ' "pid" : 10003,\n' \  ' "rid" : "%s",\n' \  ' "timestamp" : 78,\n' \  ' "ver" : 147\n}' % (int(self.roomInfo['roomId']))  data=pack('>i',len(content))+content.encode('utf-8')+b'\x0a'  return data  def get\_serverip(self):  headers = {  "Accept": "text/html",  "Accept-Encoding": "gzip, deflate, sdch, br",  "Accept-Language": "zh-CN,zh;q=0.8",  "Connection": "keep-alive",  "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36"  }  res=requests.get('http://www.quanmin.tv/site/route?time='  + str(int(time.time())),headers=headers).content  self.serverIp='.'.join([str(i^172) for i in unpack('>iiii',res[:16])])  res=requests.get(self.url,headers=headers).text  compile\_result=re.search(r'{"uid":(\d+),.+"title":"(.+?)","nick":"(.+?)",',res)  if compile\_result:  self.roomInfo["roomId"]=compile\_result.group(1)  self.roomInfo["roomTitle"]=compile\_result.group(2)  self.roomInfo["Nick"]=compile\_result.group(3)  print(self.roomInfo,self.serverIp) |

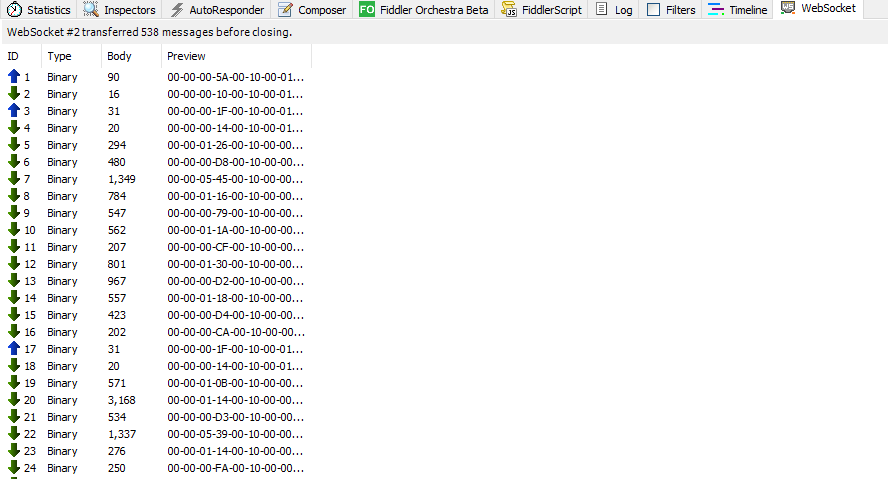
弹幕爬取之后是二进制串，需要经过python字典化处理才能便于以后数据交互，下面的代码是全民tv弹幕处理模式：

代码5-31 全民tv弹幕处理模式

|  |
| --- |
| self.compile=re.compile(  r'"chat":{"json":.+,\\"text\\":\\"(.+?)\\",\\"time\\":(\d+),.+\\"user\\":{\\"id\\":(\d+),.+?,\\"nick\\":\\"(.+?)\\"')#.+,\"text"')  def data\_received(self, data):  compile\_result=self.compile.search(data.decode('utf-8','ignore'))  if compile\_result:  self.each\_danmu["danmuContent"]=compile\_result.group(1).replace(r'\\','\\').replace(r'\\','\\').encode('utf-8').decode('unicode\_escape')  self.each\_danmu['time']=time.strftime('%Y-%m-%d %H:%M:%S')  self.each\_danmu['userId']=compile\_result.group(3)  self.each\_danmu['userName']=compile\_result.group(4).replace(r'\\','\\').replace(r'\\','\\').encode('utf-8').decode('unicode\_escape')  print(self.each\_danmu) |

* + 1. bilibili直播模块弹幕爬取流程实现

用Fillder抓取代理浏览器，从而可以截取浏览器的各种请求，在各种请求中，发现了一个Websocket连接，观察Websockt流量，发现在客户端发送一个初始化包之后，服务器就不断的向客户端推送信息，并且客户端每隔一段时间就发送一个keepalive的心跳包，可以断定，bilibili直播模块弹幕协议是架设在websocket协议之上的。下图是用Fillder抓取到的Websocket连接包：

图5-17 Fillder抓取到的Websocket连接包

用chrome浏览器开发者工具可以查看到此websocket连接所连接的服务器地址，端口的详细信息，确定了此弹幕服务器的地址：

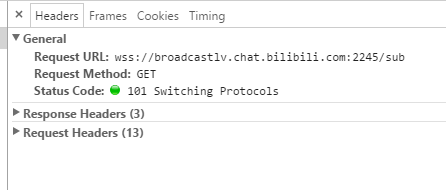


图5-18 chomre开发者工具查看到的弹幕服务器地址

在chrome浏览器中查看房间初始化信息，发现在一个ajax请求中初始化了房间信息，所传入的参数是url中的id，这个参数是房间的shortid ，在这个ajax请求的返回值中有真正的房间id，也是以后连接弹幕服务器的所需的房间id，图5-19，图5-20就是获取房间信息的ajax请求

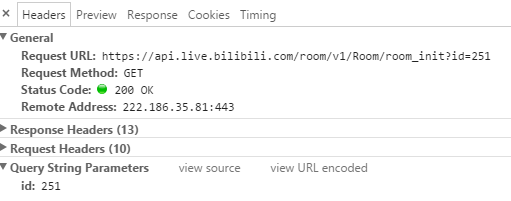
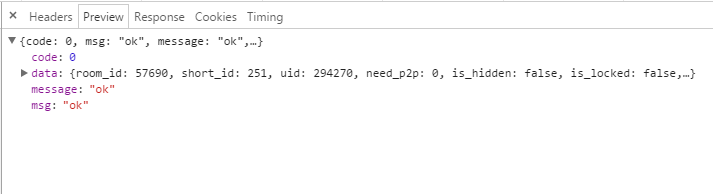
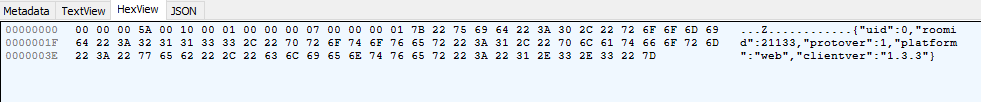


图5-19 ajax请求的url和query string

图5-20 ajax请求的返回值

获取到房间关键信息之后，在Fiddler中查看客户端向服务器发送的握手封包信息，发现第一个字段是握手包的整体长度，接下来的一串二进制码是bilibili握手规定码，后面的json字符串中的数据，只有roomid字段是可变的，其余都是固定不变的，下图是Fillder抓取到的整个二进制串：

图5-21 Fillder抓取到的整个二进制串

得到这些信息之后，用python模拟前端封包工作，利用python websocket-client库，模拟websocket请求向服务器发送握手包，就可以接收到服务器的弹幕推流。

代码5-32完成了模拟信息获取，模拟封包，和模拟发送的功能：

代码5-32模拟信息获取，模拟封包，和模拟发送的功能

|  |
| --- |
| def getRoomInfo(self):  id\_search=re.search('https://live.bilibili.com/(\d+)',self.url)  if id\_search:  id=id\_search.group(1)  request\_header={  "Host": "api.live.bilibili.com",  "Connection":"keep-alive",  "Cache-Control": "max-age = 0",  "Accept": "application / json, text / plain, \* / \*",  "Origin": "https: // live.bilibili.com",  "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.104 Safari/537.36 Core/1.53.4882.400 QQBrowser/9.7.13059.400",  "Referer": self.url,  "Accept-Encoding": "gzip, deflate, sdch, br",  "Accept-Language": "zh-CN, zh;q=0.8"  }  resjson=requests.get('https://api.live.bilibili.com/room/v1/Room/room\_init',params={"id":id},headers=request\_header ).json()  #print(resjson)  self.roomInfo['roomId']=resjson['data']['room\_id']  self.roomInfo['shortId']=resjson['data']['short\_id']  print(self.roomInfo)  def sendLoginMsg(self,ws):  print("WebSocket to %s is ok"%self.danmuServer)  self.getRoomInfo()  content={  "uid":0,  "roomid":int(self.roomInfo['roomId']),  "protover":1,  "platform": "web",  "clientver": "1.3.3",  }  header=bytes.fromhex("001000010000000700000001")  loginMsg=header+json.dumps(content).encode('utf-8')  self.\_\_ws.send((len(loginMsg)+4).to\_bytes(length=4,byteorder='big')+loginMsg)  heartBeatThread=threading.Thread(target=self.heartBeat,name='bilibiliHeartBeat')  heartBeatThread.start() |

每隔一段时间客户端还需要发送一个握手包，Fillder抓取的客户端的心跳包如下：

图5-22 Fillder抓取到的心跳包的二进制串

用python的fromhex函数可以比较方便的构造出这个二进制串，详细的构造方法如下所示：

代码5-33 python模拟心跳包

|  |
| --- |
| def heartBeat(self):  while True:  self.\_\_ws.send(bytes.fromhex("0000001f0010000100000002000000015b6f626a656374204f626a6563745d"))  time.sleep(30) |

弹幕爬取之后是二进制串，需要经过python字典化处理才能便于以后数据交互，下面的代码是bilibili弹幕处理模式：

代码5-34 bilibili直播模块弹幕处理模式

|  |
| --- |
| def receiveMsg(self,ws,message):  danmuList=re.findall(r'{"info":\[.+?\],"cmd":"DANMU\_MSG"}',message.decode('utf-8','ignore'))  for danmu in danmuList:  print(danmu)  danmnjson=json.loads(danmu)  self.danmuInfo['danmuContent']=danmnjson['info'][1]  self.danmuInfo['userId']=danmnjson['info'][2][0]  self.danmuInfo['userNick']=danmnjson['info'][2][1]  print(self.danmuInfo) |

* 1. 弹幕过滤模块实现

弹幕过滤模块分为两个小的模块，一个是弹幕内容预处理模块，包括繁简体转换，同义词替换等预处理功能，另一个是算法过滤模块，包括AC多摸匹配算法和定制敏感字符次数匹配算法。下面对两个模块分别来进行分析实现。

* + 1. 弹幕过滤预处理模块实现

第一步.需要进行传入的字符串进行繁简体转换,将繁体转换为简体，有利于提高以后算法对敏感字符的感知力。进行繁简体转换的主要思路是根据繁简体对照字库来进行逐一匹配和替换，来完成繁简体转换的的工作。下面是部分繁简体对照字库，利用python字典格式存储。

代码5-35 部分繁简体转换字库

|  |
| --- |
| zh2Hant = {  '呆': '獃',  "打印机": "印表機",  '帮助文件': '說明檔案',  "画": "畫",  "板": "板",  "丑": "醜",  "淀": "澱",  "范": "範",  "丰": "豐",  "后": "後",  "伙": "夥",  "姜": "薑",  ……  } |

利用字典树来编写对照替换函数，下面是实现繁简体转换的功能函数。

代码5-36 实现繁简体转换的功能类

|  |
| --- |
| class Converter(object):  def \_\_init\_\_(self, to\_encoding):  self.to\_encoding = to\_encoding  self.map = MAPS[to\_encoding]  self.start()  def feed(self, char):  branches = []  for fsm in self.machines:  new = fsm.feed(char, self.map)  if new:  branches.append(new)  if branches:  self.machines.extend(branches)  self.machines = [fsm for fsm in self.machines if fsm.state != FAIL]  all\_ok = True  for fsm in self.machines:  if fsm.state != END:  all\_ok = False  if all\_ok:  self.\_clean()  return self.get\_result()  def \_clean(self):  if len(self.machines):  self.machines.sort(key=lambda x: len(x))  # self.machines.sort(cmp=lambda x,y: cmp(len(x), len(y)))  self.final += self.machines[0].final  self.machines = [StatesMachine()]  def start(self):  self.machines = [StatesMachine()]  self.final = UEMPTY  def end(self):  self.machines = [fsm for fsm in self.machines  if fsm.state == FAIL or fsm.state == END]  self.\_clean()  def convert(self, string):  self.start()  for char in string:  self.feed(char)  self.end()  return self.get\_result()  def get\_result(self):  return self.final |

第二步，需要将一些对语义没有意义的特殊字符例如!"#$%&\'()\*,-./:;<= >?@等过滤掉，提取出对语义有影响的字符。利用python的正则表达式的替换函数将无关字符替换成空字符就可以完成这个功能。下面是主要实现代码。

代码5-37 实现无关字符过滤的主要代码

|  |
| --- |
| def easy\_filter(self,text):  text=re.sub('[’!"#$%&\'()\*,-./:;<=>?@，。?★、…【】《》？“”‘’！[\\]^\_`{|}~\s]+',"",text)  #[]正则表达式中括号中不需要转义  #如果没有[] 且没有r 则 匹配\需要 \\\\--->(python 转义)\\---->(正则转义)\ r的作用是取消python转义只用正则转义就行了  #r \后边一定要有匹配的字符才会消除python转义字符的转义效果，所以最后一个不能是以\结尾  #print(r'[’!"#$%&\'()\*+,-./:;<=>?@，。?★、…【】《》？“”‘’！[\]^\_`{|}~\s]+')  return text |

最后一步，需要将一些同义词或者近义词还有形近字符进行替换，并且将字母全部转换为小写字母，减少脏字库的体积，可以提升过滤算法的效率。下面是替换功能主要实现代码。

代码5-38 替换功能主要实现代码

|  |
| --- |
| text = text.replace('+', '加')  text = text.replace('脫','脱')  text = text.replace('╅', '加')  text = text.replace('ĉ', 'c')  text = text.replace('Ǒ', 'o')  text = text.replace('Ṏ', 'o')  text = text.replace('M', 'm')  text = text.replace('Ｍ', 'm')  def lower\_transform(self,text):  return text.lower() |

* + 1. 弹幕过滤AC算法模块实现

下面的代码是利用python字典类型来构造Trie树的节点，即构造构造算法中的goto表，再利用列表类型构造待处理队列定位每个节点failure指针的指向，从而实现AC算法树的构造，而output表的构造隐藏在AC算法树构造的过程中：

代码5-39 原生python代码实现AC算法

|  |
| --- |
| class ACTreeNode():  def \_\_init\_\_(self,char):  self.value=char  self.children={}  self.fail\_pointer=None  self.tail=None  def get\_value(self):  return self.value  def get\_child(self,key):  if self.children.\_\_contains\_\_(key):  return self.children.get(key)  else:  return None  def set\_tail(self,tail):  self.tail=tail  def add\_child(self,ch):  if self.get\_child(ch):  return self.get\_child(ch)  else:  newNode=ACTreeNode(ch)  self.children[ch]=newNode  return newNode  class ACTree():  def \_\_init\_\_(self):  self.root=ACTreeNode('')  self.count=0  def add\_one\_strmode(self,line):  p=self.root  for i in line:  p=p.add\_child(i)  self.count+=1  p.set\_tail(self.count)  def setFailPointer(self):  queueList = [self.root]  while queueList:  temp=queueList.pop(0)  for child\_key in temp.children:  if temp==self.root:  temp.children[child\_key].fail\_pointer=self.root  else:  p=temp.fail\_pointer#父节点的回溯结点  while p:  if child\_key in p.children:  temp.children[child\_key].fail\_pointer=p.children[child\_key]  break  p=p.fail\_pointer#继续回溯，直到fail指针为空  if not p:#如果发现p为空，则将此结点的fail指针定为self.root  temp.children[child\_key].fail\_pointer=self.root  queueList.append(temp.children[child\_key])#向带处理列表中插入待处理项的父节点 |

利用AC算法树查询敏感匹配敏感字符串的实现代码如下：

代码5-40 敏感字符串匹配功能实现

|  |
| --- |
| def runkmp(self,mystr):  p = self.root  matchCount={}  for i in mystr:  while i not in p.children and p is not self.root:  p=p.fail\_pointer  if i in p.children:  p=p.children[i]  else:  p=self.root#没查询到，下一个字符继续从根节点开始查询  continue  temp=p  while temp is not self.root:  if temp.tail:  if temp.tail in matchCount:  matchCount[temp.tail]+=1  else:  matchCount[temp.tail]=1  temp=temp.fail\_pointer  return matchCount |

* 1. 网站架构模块实现
     1. 首页输入框实时校验功能实现

首页主要部分是一个输入框和一个开始过滤按钮，输入框有输入实时校验功能，实现方法是将输入框绑定 oninput事件，从而实现只要一改变输入框的值就会展示实时校验结果。下面是实现实时输入校验功能的主要js代码。

代码5-41 js实现实时输入校验功能

|  |
| --- |
| initValidate()  {  var self=this  self.inputElement.oninput=function (e) {  if(e)  {  e.preventDefault()  }  let i,result;  for(i=0;i<self.validateRules.rules.length;i++) {  result=this.value.match(self.validateRules.rules[i])  if(result)  {  self.typecode=i  self.msgp.innerHTML=self.validateRules.correctmsg(this.value);  self.msgp.className="correctp";  self.inputElement.className="correct"  self.inputLabel.classList.remove("incorrect")  self.inputLabel.classList.add("correct")  self.searchButton.classList.remove("disabled")  for(let j=0;j<self.searchButton\_a.length;j++)  {  self.searchButton\_a[j].setAttribute('href','/danmu\_resolution?'+  'url='+ encodeURIComponent(self.inputElement.value)+  '&typecode='+encodeURIComponent(self.typecode))  self.searchButton\_a[j].setAttribute('target','\_blank')  }  break;  }  }  //console.log(i)  if(i==self.validateRules.rules.length)  {  let j;  for(j=0;j<self.searchButton.classList.length;j++)  {  if(self.searchButton.classList[j]=="disabled")  {  break;  }  }  if(j==self.searchButton.classList.length)  {  self.searchButton.classList.add('disabled')  }  self.msgp.textContent=self.validateRules.errormsg;  self.msgp.className="errorp";  self.inputElement.className="incorrect"  self.inputLabel.classList.remove("correct")  self.inputLabel.classList.add("incorrect")  }  }  }  } |

* + 1. 首页点击开始过滤按钮后台处理逻辑实现

首页点击开始过滤按钮之后，会基于输入框中的输入的url的内容，动态构造一个get请求，http://localhost:8000/danmu\_resolution/?url=https%3A%2F%2Fwww.douyu.com%2Fgd008&typecode=0将url和前端基于url判定到的平台类型传入到querystring中，发送到django后台，Django后台会提取到query string中的参数然后调用获取房间信息的函数，将获取到的房间信息渲染到django 模版中来返回前端页面，代码5-41是django view层处理此get请求的逻辑，代码5-42是相应的django 模块文件。

代码5-42 django view层处理此get请求的逻辑

|  |
| --- |
| def danmu\_url\_handler(request):  url=request.GET.get("url")  typecode=int(request.GET.get("typecode"))  if typecode == 0 or typecode == 1:  danmuobj = douyuDanmuHook.DouyuDanmuHook()  danmuobj.room\_info\_hook(url=url)  return render(request,"resolution\_page.html",context={  "roomState":danmuobj.roomState,  "url": danmuobj.url,  "type":"douyu"  }) |

代码5-43 相应的django 模块文件主要代码

|  |
| --- |
| <div class="roomInfo">  <p id="url" data-type={{ url }}>  爬取的url:&nbsp;&nbsp;{{ url}}  </p>  <p id="hooktype" data-type={{ type }}>  爬取的平台：&nbsp;&nbsp;{{ type }}  </p>  <p>  爬取的房间信息——  {% if type == "douyu" %}  房间id: {{ roomState.id }}&nbsp;&nbsp;  房间名: {{ roomState.roomName }}&nbsp;&nbsp;  主播id: {{ roomState.ownerUid }}&nbsp;&nbsp;  主播名: {{ roomState.ownerName }}&nbsp;&nbsp;  {% endif %}  </p>  </div>  <div class="center-container">  <div class="good-danmu-component">  <p style="text-align:center;"><img style="vertical-align: middle" src="../static/imgs/绿色对号.png"><span style="color:#02FF84;font:lighter 1em/64px 'Helvetica Neue'">&nbsp;&nbsp;经过AC算法过滤的弹幕</span></p>  <div class="danmu-container">  <div class="danmu-content">  <ul class='danmu-ul'>  <li>  <p><img class='icon' src="../static/imgs/头像1.png"><span>用户：</span><span> 内容：</span><span> 时间：</span></p>  </li>  </ul>  </div>  </div>  </div>  <div class="dirty-danmu-component">  <p style="text-align:center;"><img style="vertical-align: middle" src="../static/imgs/错号.png"><span style="color:#FF2121;font:lighter 1em/64px 'Helvetica Neue'">&nbsp;&nbsp;未通过AC算法过滤的弹幕</span></p>  <div class="danmu-container">  <div class="danmu-content">  <ul class='danmu-ul'>  <li>  <p><img class='icon' src="../static/imgs/头像1.png"><span>用户：</span><span> 内容：</span><span> 时间：</span></p>  </li>  </ul>  </div>  </div>  </div>  </div>  <div class="danmu-calculate-component">  <p>处理时长：<span class="handle-time"></span></p>  <p>接收弹幕总条数：<span class="totality"></span></p>  <p>过滤掉弹幕条数：<span class="dirty-count"></span></p>  <p>过滤百分比：<span class="calculate-span"></span></p>  </div> |

* + 1. 弹幕实时滚动效果的实现

弹幕实时滚动效果类似与各个大直播平台的弹幕滚动框的实现方式为当有弹幕信息到来的时候向ul标签中添加一个li标签实现弹幕条数的添加，并将包含的div容器的overflow-y属性设置为scroll，实现滚动的效果。下面代码是添加li的js方法。

代码5-44 添加li的js方法

|  |
| --- |
| ;(function(global\_win){  "use strict";  var self  var MyUiChange=function(){  self=this  this.countimg=1;  };  MyUiChange.prototype.insertliforul=function(data,danmu\_container)  {  if(!window.$SIGN) {  danmu\_container.scrollTop = danmu\_container.scrollHeight - danmu\_container.clientHeight  }  let ul\_element=danmu\_container.querySelector(".danmu-content .danmu-ul")  let newli=document.createElement('li');  newli.classList.add("eachdanmu");  let newp=document.createElement("p");  let newicon=document.createElement('img');  newicon.classList.add("icon");  newicon.setAttribute("src","../static/imgs/头像"+this.countimg+".png")  newp.appendChild(newicon)  newp.appendChild(self.createSpan(data.userName+":","color:#FF6600"))  newp.appendChild(self.createSpan(data.danmuContent,""))  newp.appendChild(self.createSpan("("+data.time+")","color:#6699FF"))  newli.appendChild(newp);  //console.log(ul\_element.childElementCount)  if(ul\_element.childElementCount>=50)  {  ul\_element.removeChild(ul\_element.children[0])  }  ul\_element.appendChild(newli);  self.countimg++;  if(self.countimg>4)  {  self.countimg=1;  }  }  MyUiChange.prototype.createSpan=function(text,styleStr)  {  let newspan=document.createElement('span')  newspan.style=styleStr  newspan.innerHTML=text+"&nbsp;"  return newspan  }  global\_win.MyUiChange=MyUiChange;  })(this) |

* + 1. 前后端WebSocket弹幕信息传输功能实现

前端页面需要向asgi服务器发起一个WebSocket请求即ws://"+window.

location.host+"/danmu\_resolution/，asgi服务器会将这个请求根据请求的url交付给对应的django channels 中注册的自定义WebSocket请求处理函数进行处理，处理函数会向前端发送一个接受连接的信息，前端页面与django后端的WebSocket连接就建立起来了，前端页面收到服务器发过来的请求连接成功信息后，会将要处理弹幕的房间信息通过建立起来的WebSocket连接发送给服务器，服务器处理函数会通过subprocess库调用一个外部进程来爬取和过滤弹幕，通过将stdout设置为PIPE来将处理进程的弹幕处理结果传输回WebSocket服务端，处理函数通过WebSocket连接将json格式的弹幕信息传输回前端页面，前端将json格式的弹幕信息渲染到页面中展示给用户，代码5-44和5-45分别是前端JS WebSocket连接弹幕接受逻辑和后端django channels WebSocket连接弹幕传送逻辑。

代码5-45 前端JS WebSocket连接弹幕接受逻辑

|  |
| --- |
| initWebSocket(initData)  {  self=this  self.startTime=Date.now();  self.websocket=new WebSocket("ws://"+window.location.host+"/danmu\_resolution/")  self.websocket.binaryType="arraybuffer"  self.websocket.onopen=function (ev) {  //console.log(ev)  //console.log(initData)  console.log("Websocket to the danmu server is opening")  this.send(JSON.stringify(self.onopenData))  console.log("Send data: "+JSON.stringify(self.onopenData)+"to DanmuServer")  }  self.websocket.onmessage=function (ev) {  let danmu = JSON.parse(ev.data)  self.total\_count++  console.log(danmu)  if (danmu.error) {  self.dirty\_count++  window.$UIChangeFactory.insertliforul(  {  userName: danmu.userName,  danmuContent: danmu.danmuContent,  time: danmu.time  },  window.$DirtyDanmuContainer)  }  else {  window.$UIChangeFactory.insertliforul(  {  userName: danmu.userName,  danmuContent: danmu.danmuContent,  time: danmu.time  },  window.$GoodDanmuContainer)  }  self.calculate\_percent = (100 \* self.dirty\_count / self.total\_count).toString().slice(0, 4)  self.total\_span.innerHTML = self.total\_count  self.dirty\_span.innerHTML = self.dirty\_count  self.handle\_time.innerHTML=parseInt((Date.now()-self.startTime)/1000)+"秒"  self.calulate\_span.innerHTML = self.calculate\_percent + "%"  }  self.websocket.onerror=function (ev) {  console.log(ev)  }  self.websocket.onclose=function (ev) {  console.log(ev)  }  } |

代码5-46 后端django channels WebSocket连接弹幕传送逻辑

|  |
| --- |
| def dataHandler(self,data):  json\_data = json.loads(data)  self.type = json\_data['hooktype']  self.currentUrl = json\_data['url']  if self.currentUrl in workingStruct and workingStruct.get(self.currentUrl).get("count")>0:#如果工作目录中存在这个url 则直接入组接收消息即可，不用另开一个线程  workingStruct[self.currentUrl]["count"]+=1#此url用户数加一，到零时关闭这个url的处理线程  async\_to\_sync(self.channel\_layer.group\_add)(  workingStruct[self.currentUrl]["groupName"],  self.channel\_name,  )  else:  if self.currentUrl not in workingStruct:#创建一个新url的工作目录  workingStruct[self.currentUrl]={}  workingStruct[self.currentUrl]["count"]=1  self.groupName=re.sub('[’!"#$%&\'()\*,-./:;<=>?@，。?★、…【】《》？“”‘’！[\\]^\_`{|}~\s]+', "\_", self.currentUrl)  workingStruct[self.currentUrl]["groupName"]=self.groupName  async\_to\_sync(self.channel\_layer.group\_add)(  self.groupName,  self.channel\_name,  )  print("Group name is %s" %self.groupName)  workingStruct[self.currentUrl]["process"] = threading.Thread(target=self.threadrun,args=(self.type,self.currentUrl))#新增处理线程  workingStruct[self.currentUrl]["processRun"]=True  workingStruct[self.currentUrl]["process"].start()  print(workingStruct[self.currentUrl]["groupName"] + " thread: "+workingStruct[self.currentUrl]["process"].getName()+" start") |

* 1. 系统实现中存在的主要问题和解决方案
     1. 存在的主要问题

主要问题就是在熊猫平台弹幕获取之后如何对获取到的弹幕信息进行处理；

* + 1. 解决方案和可行性研究

针对这个问题，首先查看了前端js源代码中对弹幕数据的处理，发现弹幕信息在服务器端经过了zlib压缩处理，在客户端需要zlib解压处理之后才可以利用正则表达式进行提取操作，前端js处理代码如下：

代码5-47 熊猫前端js代码对弹幕信息的解压处理（利用pako库inflate方法进行处理）

|  |
| --- |
| c.prototype.\_processMessage = function (e, t) {  var n = this;  try {  if (r && e instanceof r || s && e instanceof s) this.\_readMessage(e, t);  else {  var i = new FileReader;  i.onload = function () {  try {  n.\_readMessage(i.result, t)  } catch (e) {  n.\_error("解析 socket 消息失败：", e)  }  }, i.readAsArrayBuffer(e)  }  } catch (a) {  this.\_error("解析 socket 消息失败：", a)  }  }, c.prototype.\_readMessage = function (e, t) {  var n, i, a = 0,  o = new dcodeIO.ByteBuffer;  if (o.append(e, "binary").flip(), n = o.readShort(a), this.ve !== n) throw new Error("Wrong value of ve: " + n);  if (a += this.lens.ve, i = o.readShort(a), a += this.lens.op, i === this.op.binduserack) this.\_readBindUserAck(o, i, a);  else if (i === this.op.heartbeatack) clearTimeout(this.\_heartbeatTimeout);  else {  if (i !== this.op.message) throw new Error("Wrong value of op: " + i);  this.\_handshaked && this.\_readChatMessage(o, i, a, t)  }  }  c.prototype.\_readChatMessage = function (e, t, n, i) {  var a, o, r, s = this;  if (a = e.readShort(n), n + this.lens.pl + a >= e.limit) throw new Error("Out of limit, read chat message offset is " + (n + this.lens.pl + a) + ", buffer limit is " + e.limit);  if (n += this.lens.pl + a, o = e.readInt32(n), n + this.lens.dl + o !== e.limit) throw new Error("Out of limit, read chat message offset is " + (n + this.lens.dl + o) + ", buffer limit is " + e.limit);  n += this.lens.dl, e.offset = n;  var l = new Uint8Array(e.toArrayBuffer()),  c = l.length;  this.\_compress && (l = pako.inflate(l));  var u = l.length,  d = new dcodeIO.ByteBuffer;  d.append(l).flip(), r = this.\_readChatMessageBody([], d, 0, u), r.forEach(function (e) {  s.\_debug("Receive message:", JSON.stringify(e)), i({  op: t,  data: e  })  }), this.\_compress && this.\_debug("Message count [" + r.length + "],", "compression ratio [" + (c / u \* 100).toFixed(0) + "%],", "origin zlib buffer length is [" + c + "],", "inflate buffer length is [" + u + "]")  }, c.prototype.\_readChatMessageBody = function (e, t, n, i) {  try {  n += this.lens.id + this.lens.appid;  var a = t.readInt32(n);  if (n += this.lens.bodyLength, n + a <= t.limit) {  var o = JSON.parse(t.readUTF8String(a, dcodeIO.ByteBuffer.METRICS\_BYTES, n).string);  if (e.push(o), n += a, i = i - this.lens.id - this.lens.appid - this.lens.bodyLength - a, i > 0) return this.\_readChatMessageBody(e, t, n, i)  }  } catch (r) {  this.\_error("读取消息体失败：", r)  }  return e |

在python的库中，有一个zlib库可以进行zlib，gzip的压缩解压操作，但是，我使用顶层方法zlib.decompress()方法并不能对所获取到的弹幕二进制信息进行解压操作，python解释器会报一个“解压流不完整”的错误，经过在google和csdn等专业博客的查询，了解到zlib压缩格式如下所示：zlib =zlib头（0×78 0x9c）+deflate编码的实际内容 + zlib尾（8字节：CRC32：4字节（BigEndian），原始(未压缩)数据的32位校验和 + ISIZE：4字节（BigEndian），原始(未压缩)数据的长度的低32位。）

我发现这个错误是由于解压目标没有zlib尾部（原未压缩数据adler32校验和）导致zlib.decompress()方法不能识别到zlib校验和尾部不能进行完整性校验，导致zlib.decompress()方法认为这是一个不完整的数据流，从而出现了错误。经过查验，发现可以使用zlib底层解压对象的解压方法zlib.decompressobj().decompress()，对所传入的数据不进行尾部校验和验证，从而可以避免校验和错误。但是要注意，解压流也必须从头部开始，即前两个字节必须是0×78和0x9c。还有一点结论，经过查验可以得出：对于底层解压对象，是“一次性的”，即每一次解压一个新的弹幕信息，都必须重新生成一个解压对象zlib.decompressobj(),不能重复利用。下面是熊猫弹幕信息解析的python代码：

代码5-48 熊猫弹幕信息解析

|  |
| --- |
| self.\_\_xiongmaoCompile = re.compile(  r'{"type":"1","time":(\d+),.+?"nickName":"(.+?)",.+?"rid":"(\d+)".+?"content":"(.+?)"')  self.xiongmaoDanmuJson={}  def \_receiveChatMsg(self,ws,message):  print(message)  if len(message)<5:  return  op=int.from\_bytes(message[2:4],byteorder='big')  if op==3:  usefulMsg=zlib.decompressobj().decompress(message[15:]).decode('utf-8','ignore')#解析块，可以没有尾，但必须有头 x\x9c 一个解析对象只能进行一次解析活动  print(usefulMsg)  useful\_segment=self.\_\_xiongmaoCompile.search(usefulMsg)  print(useful\_segment)  if useful\_segment:  #print(useful\_segment.group(0))  self.xiongmaoDanmuJson["time"]=time.strftime('%Y-%m-%d %H:%M:%S',time.localtime(int(useful\_segment.group(1))))  self.xiongmaoDanmuJson["userName"]=useful\_segment.group(2)  self.xiongmaoDanmuJson["userId"]=useful\_segment.group(3)  self.xiongmaoDanmuJson["danmuContent"]=useful\_segment.group(4)  self.xiongmaoDanmuJson["roomId"]=self.roomId  print(self.xiongmaoDanmuJson) |

1. 系统测试
   1. 测试环境

如表5-1所示，为系统测试时使用的windows版本，python版本、nodejs版本，django库版本，django channels版本，requests爬虫库版本等。

表5-1 测试环境数据表格

|  |  |
| --- | --- |
| 名称 | 值 |
| Windows版本 | windows10 |
| 主机分辨率 | 1920×1080 |
| cpu类型 | Intel(R)Core(TM)i7-4710MQ |
| 已安装内存（RAM） | 16.0GB |
| Python版本 | 3.6.2 |
| Nodejs版本 | v10.1.0 |
| Django版本 | 2.0.4 |
| Django-channels版本 | 2.1.1 |
| Requests爬虫库版本 | 2.18.4 |
| Daphne版本 | 2.1.2 |
| asgiref版本 | 2.3.2 |

* 1. 测试方法

（1）在项目文件夹DanmuServer下运行windows命令行。

（2）在命令行中输入 python manage.py runserver 8000,等待服务器启动。

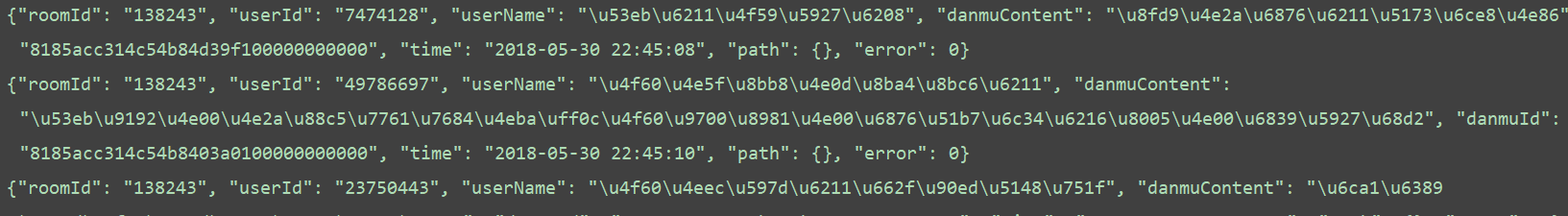
（3）在浏览器中输入localhost:8000访问网站主页。

（4）在url中输入正确的房间地址（在主播的网页地址栏中复制过来即可）。

（5）多开几个处理网页，用来进行网站的压力测试。

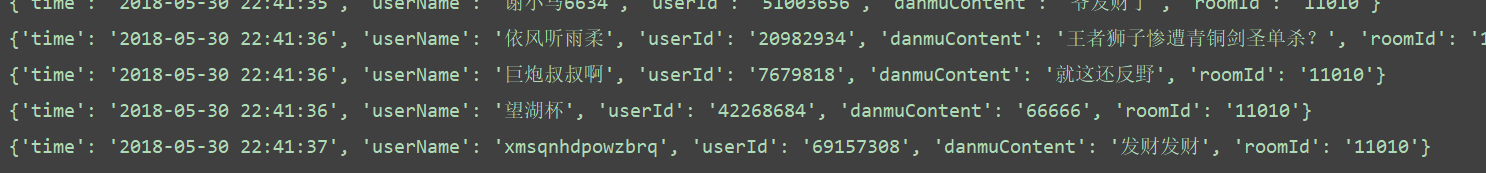
* 1. 测试结果
     1. 斗鱼tv弹幕爬取结果展示

url以https://www.douyu.com/dongzhu为例，来测试斗鱼爬取子进程的功能，测试结果如下图所示，输出为json格式。其中error字段为敏感字符串标记，error为0时表示通过过滤，为1表示未通过过滤。



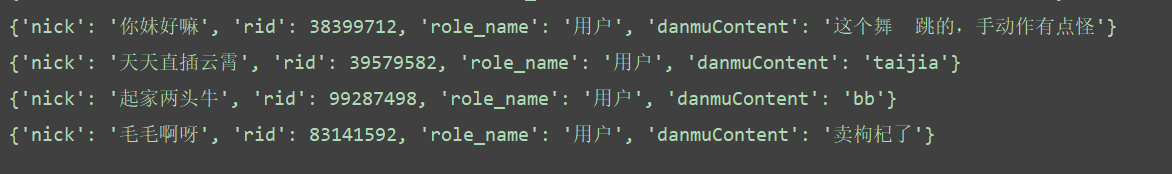
* + - * 1. 斗鱼tv弹幕爬取结果展示
    1. 熊猫tv弹幕爬取结果展示

url以https://www.panda.tv/11010为例，来测试熊猫爬取子进程的功能，测试结果如下图所示，输出为json格式。



* + - * 1. 熊猫tv弹幕爬取结果展示
    1. 熊猫tv（星秀星颜模块）弹幕爬取结果展示

url以https://xingyan.panda.tv/572475?channel=web\_xy-cate\_xingxiu-tj\_allcate为例，来测试熊猫爬取子进程的功能，测试结果如下图所示，输出为json格式。



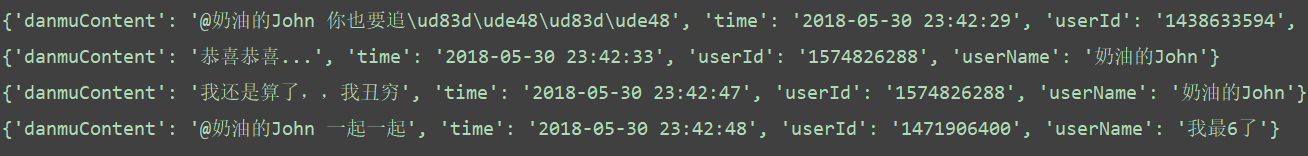
* + - * 1. 熊猫tv（星秀星颜模块）弹幕爬取结果展示
    1. 战旗tv弹幕爬取结果展示

url以https://xingyan.panda.tv/572475?channel=web\_xy-cate\_xingxiu-tj\_allcate为例，来测试熊猫爬取子进程的功能，测试结果如下图所示，输出为json格式。



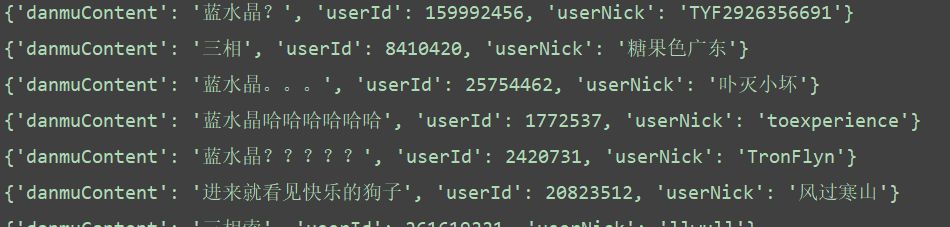
* + - * 1. 战旗tv弹幕爬取结果展示
    1. 全民tv弹幕爬取结果展示

url以https://www.quanmin.tv/28092450为例，来测试熊猫爬取子进程的功能，测试结果如下图所示，输出为json格式。



* + - * 1. 全民tv弹幕爬取结果展示
    1. bilibili直播模块弹幕爬取结果展示

url以https://live.bilibili.com/102?visit\_id=76ax8pz4vws0为例，来测试熊猫爬取子进程的功能，测试结果如下图所示，输出为json格式。



* + - * 1. bilibili弹幕爬取结果展示
    1. 网站架构模块与弹幕处理子进程集成测试展示

以斗鱼平台为例，在url输入框中输入https://www.douyu.com/dongzhu，点击开始过滤按钮，跳转到如下图所示界面，界面上顶部是url对应的房间信息，下部是过滤百分比用来分析直播间的弹幕环境，中部左侧部分是通过过滤的弹幕框，与直播平台上的弹幕框基本保持实时同步，中部右侧是被算法过滤掉的不良弹幕。



* + - * 1. 网站架构和弹幕爬取模块集成测试
  1. 测试小结

通过单独的模块测试和系统集成测试，可以看出,这款弹幕过滤系统有很强的健壮性和抗压性，能适应各种不同环境的使用

1. 工程能力与职业素养培养
   1. 知识技能学习情况

在工具应用方面，第一学习到了用wireshark抓取具有特定特征的网络数据包的方法，第二，学会了利用chrome开发者工具烂查看当前页面所发送的各类请求的信息并加以区分，第三，学习了利用Fillder代理浏览器抓包的操作，了解到Fillder的优点在于截取全部浏览器请求，将浏览器不能显示出来的二进制包的内容也可以在Fillder HexView面板中也可以轻松显示出来

在理论学习方面，第一，学习到了unicode编码，gbk编码，utf8编码的区别与异同，对二进制串，网络传输字节序等问题有了更深刻的理解。第二，了解了web端经常使用的base64编码和zlib压缩方式，能做到识别简单的base64编码。第三，学习了新型网络协议websocket，在编码工作中对websocket的协议组成，握手方式，服务端和客户端交互工作结合websocket协议规范，对各种网站架构的优化有了新的认识

在编码能力方面，熟练使用pyhton的reqeusts库，正则表达式re库，和python版websocket客户端websocket-client库和django框架。对python字符串处理，json字符串与python 字典对象的相互转换，二进制数据的提取有更为灵活的使用。

在算法学习方面，学会了AC多摸匹配算法进行字符串过滤的操作。

* 1. 对软件工程实践以及软件工程领域发展的认识

软件工程是一门研究用工程化方法构建和维护有效的、实用的和高质量的软件的学科。它涉及程序设计语言、数据库、软件开发工具、系统平台、标准、设计模式等方面。

软件工程的目标是，在在给定的时间和预算内，按照用户的需求，开发高质量（复杂）软件，在四个月毕业设计期间，我完完整整的从最初的需求，到概要设计，详细设计到最后的编码实现，参与了每一部分的工作。能够从工作中感受出软件工程的魅力。

1. 结束语
   1. 全文总结

（1）分析了斗鱼，虎牙，哔哩哔哩，全民，战旗，熊猫六大直播平台的弹幕协议；

（2）用python分别重写了握手包和心跳包，与各大平台的弹幕服务器进行连续，完成了对弹幕的实时获取；

（3）将获取到的弹幕信息用AC算法和敏感字符匹配次数算法进行过滤并添加过滤标记；

（4）最后将整个系统成功架设在由Django和Django channels共同搭建的异步服务器上，为用户提供了体验良好的网页操作接口。

通过这一段时间的对各种直播平台的弹幕爬取工作，发现脏字弹幕比例达到5%，弹幕净化工作迫在眉睫，需要找到一个既便捷又有效的方法来过滤敏感字符串。对网络安全知识和网络信息安全的架构设计也有了更深层次的理解。

* 1. 不足与下一步工作

可以增加数据库连接，将爬取到的所有弹幕存储在数据库中，通过过滤的弹幕存在一张表中，未通过过滤的弹幕存在一张表中，每个平台不同的数据表，每隔一段时间，可以通过数据表中的弹幕分析未通过过滤的弹幕所占全弹幕的百分比，从而得知各个平台的弹幕质量情况的优劣情况。还可以通过时间来观察哪个时间段弹幕数量，从而可以推断哪个时间段观看直播的人数数量情况。

参考文献

[1]李海峰，王炜.弹幕视频:在线视频互动学习新取向[[J].现代教育技术，2015,(6).

[2]新媒体时代的弹幕文化现象分析[J]. 付晶晶. 南京邮电大学学报(社会科学版). 2016(02)

[3]新媒介青年亚文化:弹幕族文化现象研究[J]. 徐媛. 学理论. 2015(34)

[4]情感分析及可视化方法在网络视频弹幕数据分析中的应用[J]. 郑飏飏,徐健,肖卓. 现代图书情报技术. 2015(11)

[5] 魏冬梅, 何忠秀 等, 基于Python的Web信息获取方法研究[J].软件导 刊 ,2018(01).

[6] 孙立伟 , 何国辉等 , 网络爬虫技术的研究 [J]. 电脑知识与技术 ,2010(05).

[7] 周中华, 张惠, 然谢江 ,基于Python 的新浪微博数据爬虫[J].计算机应用 2014(11).

[8] 张明杰 . 基于网络爬虫技术的舆情数据采集系统设计与实现 [J]. 现代计算机 (专业版)2015(06).

[9] 朱烨行 , 张明杰 数据采集的设计与实现[J]. 电脑编程技与维护 ,2017(09).

［10］第41次《中国互联网络发展状况统计报告》［Z］.中国互联网信息中心,2018．03［7］韦玮. 精通Python网络爬虫［M］．北京:机械工业出版社，2017．

［11］赵广辉．面向新工科的 Python 程序设计交叉融合案例教学［J］．计算机教育， 2017(8):27－31．

［12］刘春茂，裴雨龙．Python 程序设计案例课堂［M］．北京:清华大学出版社， 2017．

[13]基于Django的实验室设备管理系统设计与实现[J]. 王磊,危建国. 软件导刊. 2014(12)

[14]利用Django快速构建信息发布系统[J]. 刘班. 电脑编程技巧与维护. 2011(09)

[15]基于Django快速开发Web应用[J]. 刘班. 电脑知识与技术. 2009(07)

[16]基于Django和Python的Web开发[J]. 王冉阳. 电脑编程技巧与维护. 2009(02)

[17]基于开源的Web开发架构——ZOPE/Python[J]. 郭力子. 计算机时代. 2008(07)

[18] Learning Website Development with Django. Hourieh. . 2008

[19] Practical Django Projects. James Bennett. . 2008

致 谢

值此本科学位论文完成之际，首先要感谢我的导师杨珊老师。杨老师从一开始的论文方向的选定，到最后的整篇文论的完成，都非常耐心的对我进行指导。给我提供了大量数据资料和建议，告诉我应该注意的细节问题，细心的给我指出错误。他对软件工程领域的专业研究和对该课题深刻的见解，使我受益匪浅。杨老师诲人不倦的工作作风，一丝不苟的工作态度，严肃认真的治学风格给我留下深刻的影响，值得我永远学习。在此，谨向导师杨珊老师致以崇高的敬意和衷心的感谢!

外文资料原文

Python Network Source Automatic Evaluation System

**Abstract.**

With the rapid development of Internet and computer technology, the network information is exploding combined with the network capture technology and text analysis technology to achieve the evaluation of content resources has become hot research fields. The use of this method is significant to the resources evaluation. The topic of this paper comes from the project of the Ministry of science and technology project "Content Bank Evaluation System", this paper will make a detailed analysis of the evaluation mode based on the network information, and make the research and design of the network data capture and text analysis technology.

**Introduction**

The objective evaluation of network source is the key of network source management. As the construction of basic resource bank, it provides the orientation of construction of network source and it also reflects the preferences and demands of network searchers. We can optimize the network search engine, more importantly we can evaluate the network source in a more objective way to avoid one sidedness of subjective factors through the evaluation and arrangement of web resources. We can get a more comprehensive evaluation of network source information by analyzing a large data mining and data evaluation. It can collect network information timely and analyze mass Internet surfer behavior of audience and the evaluation opinions combining web information crawling technology and semantic analysis technology, then it can evaluate a content resources more accurate, provide the basis for selection of the audience and provide personalized push selection. Through the extraction of key words, emotional analysis and analyzing the key words. We can know the latest development in the field of scientific research and the latest direction of the development of advanced technology.

**Basic principle**

**Web crawler**. Web crawler is also called web spider. It is a program to automatically download web pages according to rules and it is an important part of the search engine grab system. The information on the Internet is scattered in the hundreds of thousands of web pages, and these pages are stored by millions of servers in every corner of the earth. Users who browse in the web world usually only get information via hyperlinks and they travel through the web pages.The crawler can collect the information of multiple sites and do the further analysis and mining through the online (web page is downloaded) or offline (after the page is stored).

**Natural Language Processing**. Natural Language Processing(NLP) is a theory and method to achieve the communication between human and computer in natural language (such as Chinese, English etc.), it is an important direction in the field of science and artificial intelligence. The earliest natural language understanding of the research work is Machine Translation. In 1949, Americans Weaver first puts forward the design scheme of Machine. In 1960s, foreigners did extensive research work and spent huge fees on machine translation, but they made a little progress because of underestimating the complexity of natural language and immature of the theory and technology of language processing. The main method is to store words and phrases of two languages, and do the translation according to Dictionary of translation and then just adjust the same order of the language technically. However, the language translation in our daily life is far from simple, it often needs to refer to the meaning of a sentence before and after

外文资料翻译

Python网络资源自动评估系统

摘要：随着Internet和计算机技术的飞速发展，网络信息爆炸，结合网络捕获技术和文本分析技术，实现对内容资源的评价已经成为研究的热点领域。该方法的使用对资源评价具有重要意义。本论文的课题来源于科技部项目“内容银行评价系统”，本文将对基于网络信息的评价模式进行详细分析，并对网络数据采集和文本分析技术进行研究和设计。

导言：网络资源的客观评价是网络资源管理的关键。作为基础资源库的建设，它为网络资源的建设提供了方向，也反映了网络搜索者的偏好和需求。通过对网络资源的评价和配置，可以优化网络搜索引擎，更重要的是可以更客观地评价网络资源，避免主观因素的片面性。通过对大型数据挖掘和数据评价的分析，可以对网络资源信息进行更全面的评价。它将网络信息爬行技术与语义分析技术相结合，能够及时收集网络信息，分析受众的大众上网行为和评价意见，更准确地评价内容资源，为受众选择提供依据，并提供个性化推送选择。通过关键词提取、情感分析和关键词分析。我们可以了解科学研究领域的最新发展和先进技术的最新发展方向。

基本原理：（1）网络爬虫：网络爬虫也被称为网络蜘蛛。它是一个按规则自动下载网页的程序，是搜索引擎抓取系统的重要组成部分。互联网上的信息分散在成千上万的网页中，这些网页被数百万服务器存储在地球的每一个角落。在Web世界中浏览的用户通常只能通过超链接获取信息，并通过网页进行访问。爬虫可以收集多个站点的信息，并通过在线(下载的网页)或脱机(存储网页后)进行进一步的分析和挖掘。（2）自然语言处理：自然语言处理(NLP)是用自然语言(如汉语、英语等)实现人与计算机之间交流的理论和方法，是科学和人工智能领域的一个重要方向。最早对自然语言理解的研究工作是机器翻译。1949年，美国人韦弗首次提出了机器的设计方案。20世纪60年代，国外学者在机器翻译方面做了大量的研究工作，花费了大量的费用，但由于低估了自然语言的复杂性和语言处理理论与技术的不成熟，使得机器翻译的研究进展不大。主要方法是存储两种语言的单词和短语，根据“翻译词典”进行翻译，然后对同一语序进行技术调整。然而，在我们的日常生活中，语言翻译远非简单，它往往需要参考一个句子前后的意义即可。