**本 科 毕 业 论 文（设计）**

|  |  |
| --- | --- |
| 课题名称 | TAG电影推荐系统的设计与实现 |
| 学 院 | 计算机科学与网络工程学院 |
| 专 业 | 软件工程 |
| 班级名称 | 软件183 |
| 学生姓名 | 刘可佳 |
| 学 号 | 1806300040 |
| 指导教师 | 冯元勇 |
| 完成日期 | 2022年5月20日 |

教 务 处 制

**TAG电影推荐系统的设计与实现**

软件183 刘可佳(小四号宋体)

指导教师:冯元勇(小四号宋体)

**摘要** 随着时代变迁与全球化的逐渐进行，人类的文化浪潮不断交融，越来越多不同种类与风格的电影开始出现。而与此同时，科技的发展也致使电影制作技术地不断发展，因此越来越多的电影被积累在了互联网上。这其中不乏制作精良且内容优秀的好作品，但也充斥着粗制滥造的烂片。互联网上鱼龙混杂的电影池，让人难以寻找到他们心仪题材的优秀作品。虽然网上也有各种电影评分网站，但是这些评分网站也有很多水军混杂在其中滥竽充数，并且复杂的界面让用户较难使用。

本应用是TAG电影推荐系统PC端应用，专门为了想要找到心仪题材的优秀电影的人提供的Web应用，综合多家电影评分网站，提供简洁明了的UI界面，并按照一定权重打分为用户进行推荐，从而减少用户看到劣质电影的可能性，提升用户的生活质量。

**关键字** 电影；推荐；综合；权重打分；Web应用；

**Design and implementation of TAG movie Recommendation System**

**ABSTRACT** Throughout the time and the progress of globalization, the tide of human culture continue to blend. More and more movies of different kinds and styles began to appear. Meanwhile, the development of science and technology also leads to the continuous development of movie production technology. Therefore, more and more movies are accumulated on the Internet. There are many excellent movies with excellent production and excellent content, but it’s also full of shoddy movies. It is difficult for people to find some good movies on their favorite subjects. Although there are various movie scoring websites on the Internet, these scoring websites also have many unrealistic scores, and the complex interface makes it difficult for user to use.

This PC application is designed for people who want to find the movies with their favorite subjects. This application synthesize multiple movie scoring websites, rated according to a certain weight to recommend for users and provide a concise UI interface for users, so as to reduce the possibility of users seeing bad movies and improve users experience.

**KEY WORDS** movie; recommend; synthesize; weight scoring; Web application

**目录**

[第一章 前 言 1](#_Toc101708039)

[1.1 课题背景 1](#_Toc101708040)

[1.2 国内外现状 1](#_Toc101708041)

[1.3 研究内容 2](#_Toc101708042)

[1.4 可行性分析 2](#_Toc101708043)

[1.4.1技术可行性分析/技术路线 2](#_Toc101708044)

[1.4.2本课题的意义 3](#_Toc101708045)

[1.5 本课题的主要工作 4](#_Toc101708046)

[第二章 需求分析 6](#_Toc101708047)

[2.1 用例分析（功能需求详细分析） 6](#_Toc101708048)

[2.3 用例中的类 7](#_Toc101708049)

[2.4 用例主体行为（主要业务逻辑） 8](#_Toc101708050)

[2.4.1 用户打开程序 8](#_Toc101708051)

[2.4.2 用户选择TAG 9](#_Toc101708052)

[2.4.3 用户点击推荐 10](#_Toc101708053)

[2.5 用例异常行为（异常场景） 11](#_Toc101708054)

[第三章 概要设计 13](#_Toc101708055)

[3.1 开发环境及配置 13](#_Toc101708056)

[3.2 系统分解（体系结构） 14](#_Toc101708057)

[3.3 系统总体流程 14](#_Toc101708058)

[第四章 详细设计 16](#_Toc101708059)

[4.1 类（数据）设计 16](#_Toc101708060)

[4.2 关键逻辑、算法设计 18](#_Toc101708061)

[4.3 功能模块具体实现 20](#_Toc101708062)

[4.3.1 程序目录结构 20](#_Toc101708063)

[4.3.2 conf（配置解析模块） 21](#_Toc101708064)

[4.3.3 util（通用函数模块） 22](#_Toc101708065)

[4.3.4 proto（存储协议模块） 23](#_Toc101708066)

[4.3.5 crawler（爬虫模块） 24](#_Toc101708067)

[4.3.6 dao（数据库交接层模块） 25](#_Toc101708068)

[4.3.7 rank（电影数据排序模块） 25](#_Toc101708069)

[4.3.8 tag\_recommend\_system\_server（服务端核心模块） 26](#_Toc101708070)

[第五章 测试实施 27](#_Toc101708071)

[5.1 功能测试 27](#_Toc101708072)

[5.1.1 配置读取模块 27](#_Toc101708073)

[5.1.2 通用模块层 29](#_Toc101708074)

[5.1.3 存储协议模块 29](#_Toc101708075)

[5.1.4 爬虫模块 30](#_Toc101708076)

[5.1.5 数据库交接层模块 32](#_Toc101708077)

[5.1.6 电影数据排序模块 33](#_Toc101708078)

[5.1.7 服务端与客户端核心模块 34](#_Toc101708079)

[第六章 安装、使用说明 38](#_Toc101708080)

[6.1 安装说明 38](#_Toc101708081)

[6.1.1 安装前准备 38](#_Toc101708082)

[6.1.2 系统安装 38](#_Toc101708083)

[6.2 使用说明 38](#_Toc101708084)

[第七章 开发、维护环境设置说明 39](#_Toc101708085)

[7.1 软件开发环境概述 39](#_Toc101708086)

[第八章 结论 40](#_Toc101708087)

[8.1 总结 40](#_Toc101708088)

[8.2 展望 40](#_Toc101708089)

[参考文献 42](#_Toc101708090)

[致谢 44](#_Toc101708091)

# 前 言

## 课题背景

在此介绍课题的背景，引出课题所讨论的话题（课题所在的领域，比课题面对的问题略大）。

电影，英文为movie或者film，特点是运动的画面，是一种视觉艺术作品，它用来模拟通过动态图像来传达思想、故事、感知、感觉、美或氛围的体验。众所周知，电影现在已经成为了我们生活中不可或缺的文化传播载体。如今的时代，电影的拍摄随着科技发展已经变得越来越快捷化，成本也随之越来越低。在这种环境下有很多人通过低成本拍摄很多质量低劣的电影来盈利。在以前信息流通没那么便利的时候，通过观众之间信息差让不知情的观众买账。如今互联网发展迅速，人们的信息渠道也多来越多，各种电影点评打分网站也随着孕育而生。但是这也导致通过电影盈利的人能在这些点评网站上操纵网络水军[1]进行控评刷分的行为。

而如此多的电影点评打分网站，它们几乎都聚合了很多功能，对于只想找到自己想看的题材的电影的用户来说，不免有些麻烦复杂，而且如果可能会被充斥在其中的水军评分打分所误导。本文所作TAG电影推荐系统[2]由此孕育而生，通过简便使用的UI界面选取题材TAG、对现网不同电影点评网站以权重占比的方式来对电影进行排序推荐。

## 国内外现状

介绍与课题讨论内容相关的国内外理论、技术、工具、产品、效能等现状。

国内近年来质量低劣的电影越来越多，因此能用来查询电影评价的网站平台也是如此。在国内，著名的有豆瓣、猫眼等平台；而国外粗制滥造的电影相比于国内来说只多不少。在国外，也有烂番茄（Rotten Tomatoes）、互联网电影资料库（Internet Movie Database，简称IMDB）等等知名网站。他们都拥有一定数量的用户，因此能做到每部电影都有一定的评价基础数量。

## 研究内容

目前国内外的电影点评打分网站有很多，但是他们的UI界面由于混杂了其他不相关的功能以及内容，并且每家平台的用户群体几乎是不相同的，这就导致可能出现某些人通过控评刷分等手段，使得电影的点评打分可信度降低。因此本系统准备解决以上两个问题，实现简洁明了的UI界面，以及综合国内外多家平台的点评，推荐并提供给用户想观赏的优质电影。

* 提供给用户简洁明了的UI界面。
* 提供给用户大多数电影题材选取。
* 综合多家电影点评打分平台，并且按照一定网站权值进行排序。
* 通过记录用户历史选择的题材TAG，并按照一定物品多样性给予用户电影推荐。
* 基于上述功能，提供高效的请求答复。

## 可行性分析

### 1.4.1技术可行性分析/技术路线

本课题主要运用到了Python作为前后端的主要编程语言。客户端，即前端可视化方面通过Python的Tkinter[3]作为GUI编程外部库来进行实现，这方面在Windows进行开发；而后端，即服务端方面，在以Linux为内核，Ubuntu为操作系统的服务端上开发；前后端的网络通信使用socket进行TCP连接交互；TCP通信协议[4]可以使用自定义简单通信协议；配置读取方面，配置文件使用yaml，引入Python的yaml解析库用于解析yaml文件；爬虫的原始html数据解析使用BeautifulSoup[5]的Python外部库；爬虫伪装代理使用UserAgent的Python外部库；存储方面在服务端部署Redis作为数据库；数据存储结构的协议方面可以使用Google Protobuf[6]，Google Protobuf目前支持C++、Python、Java等十种编程语言，可支持跨语言自动生成协议文件。通过Google Protobuf协议生成的结构体自带序列化与反序列化[7]函数，用于数据库存储结构体的生成与复原；服务端主程序使用多线程编程接收处理外部请求，使用Python的Thread即可实现。上述提到的Python外部库的API都能在官网、搜索网站或者教程网站中找到详细的说明。在前后端通信方面，运用网络编程的知识自定义一个简单的TCP通信协议也并不困难。对于Python爬虫，也能通过不断进行爬取尝试与解析来推进进度。在数据库方面，本课题数据库使用只需要通过对Redis的接口进行调用，无需对其内部的详细逻辑加以实现，而Python拥有与Redis进行通信的外部库可以调用，这些API也都在网上详尽无比。Python语言的学习成本是很低的，在一定的编程基础上，就能直接参考Python的基础语法和各种API接口一步步编写代码。且由于Python是脚本语言，不需要进行编译，因此相较于其他高级语言来讲，使得代码调试、功能测试的效率很高。除此之外，Linux服务器上的开发环境安装配置相较于Windows来说更加方便，因此本课题的技术可行性是非常高的。

### 1.4.2本课题的意义

#### 1.4.3 社会效益分析

在国家经济文化水平的发展下，电影的数量也在不断累计上涨，在如此鱼龙混杂的电影大环境下，拥有一款能提供优质电影内容推荐的应用是必不可少的。这对于喜欢看电影的人来说是这样，而对于只是偶尔看电影的人来说更加如此，因为他们只是偶尔不经常看电影，如果看的电影质量非常差的话，就会导致他们的时间白白浪费在偶然不多的情况之下。因此，这些群体都是该应用的潜在使用群体，这些使用群体会在自己内部同一圈子推广该圈子的相关应用，这就让本应用的推广更加简单快速。

#### 1.4.4 经济效益分析 done

在全国经济的发展下，购买一台个人家庭电脑花费已经越来越低，而大多数人观看电影都是使用个人家庭电脑，且系统大部分都是Windows。而开发一个Windows前后端交互的应用程序的经济成本也不高，只需要一台能够运行的电脑、在平台购买一台云服务器即可。而人力方面只需要一个拥有编程基础的人就可以很快的完成。因为上述也提到Python的学习成本其实非常低，拥有编程基础的人只需要掌握基本的语法就能一边查阅官网的API一边进行开发。而开发出来的应用可以通过开发者在知名平台发布，以及群体内部自我推广。一款实用的Windows应用，能够拥有成千上万、上十万百万用户使用都不是问题。而且Windows应用在其中添加广告推广也并非难事，因此经济上的效益还是可行的。

本课题意义主要是对目前市场上的电影评价平台的功能进行优化，在实现UI简化、功能效果几乎不变的情况下，添加电影评价平台做不到的功能。因为在热门的电影评价平台，可能会出现很多水军恶意刷分刷评论的情况。本应用结合多个网站，并按一定权重对每个网站的数据打分，再提供给用户。这就是平台做不到的功能。因此，本课题研究的应用主打UI简洁、功能优质的优点来吸引用户使用。

## 本课题的主要工作

本课题的想法其实在去年末的时候，就已经初有雏形，原因其一，是这个课题所研究的功能现象，是笔者自己在生活中实际遇到的需求之一。第二，是因为课题所使用到的技术，与笔者一直在实习的工作中所使用的技术几乎息息相关。笔者在工作中所在的部门是做关于内容推荐系统的后端基础架构，这就与本课题中所涉及与使用的技术大致相同，比如整体的推荐架构设计、前后端交互、服务器程序之间的相互通信、代码安全规范的保持以及代码模块解耦的设计等等。这就保证了笔者在研究对本课题设计的系统上，是有一定基础技能与知识的保证的，从而不至于无从下手的情况发生。这也是笔者为了验证自身在工作中所学习的知识的一个渠道与方法。如上述的课题背景也已经提到，本课题研究的应用主要是为广大观看电影的用户提供一个，能找到他们希望看到的电影，界面简洁的应用。在确保用户使用体验的同时，保持有一定的市面平台或者应用不具有的原创功能性。通过这个应用想法的基础上，再来设计本应用的需求分析、设计应用划分的模块，以及各模块应有的功能、模块之间的交互性。再对应这些模块的功能，从而来研究应该使用什么技术来加以实现，整体的流程就变得清晰明了。譬如，早期笔者的想法是使用C++来对后端进行数据处理、使用Python实现爬虫模块、前端则是使用Html的Vue框架来进行实现。因为笔者在大学三年以及实习期间几乎都是使用C++为主要编程语言。虽然上述想法确实是有实现性的。但是在中期，当笔者使用Python完成爬虫模块后，却发现其实Python用于数据处理的效率是要高于C++的。原因是Python能调用许多外部库，用较少的行数完成比较多的逻辑功能，而C++需要话更多的时间自己去实现。并且由于Python是脚本语言，能直接对某一部分代码进行Debug，而C++每次测试函数功能的时候，往往需要花费几分钟到十几分钟的时间编译完才能进行测试，这就导致使用Python的效率是要高于C++的。而C++的优势是在于程序运行的效率高，以编写速度、编译时间换取运行效率，但是本课题所研究的应用实际上涉及到的运算逻辑并不多，因此使用Python作为主要语言是最佳的。而就是这般边学习、边思考、边完成课题研究与完成应用实现的过程，便是本课题的主要工作。

本论文一共分为了七个章节：第一章介绍了本课题的一些相关背景和意义，第二章给出了应用需求分析；第三章介绍了应用的系统设计；第四章对应用的具体的实现方式作了详细的阐述；第五章介绍了应用的测试和运行的情况；第六章介绍应用安装与使用说明；第七章对应用的开发环境维护与再配置做说明；第八章对全文进行了总结。

# 需求分析

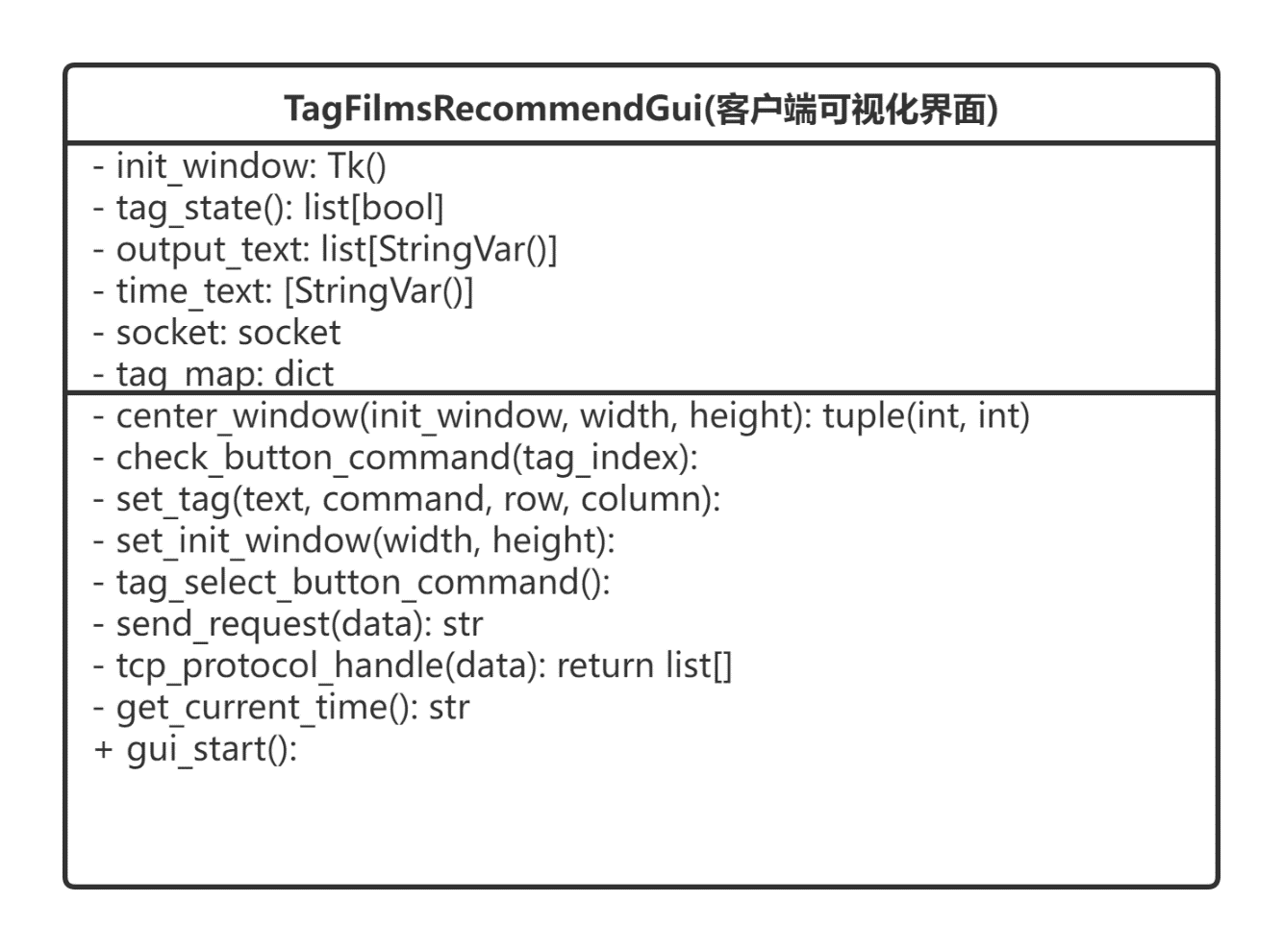
## 用例分析（功能需求详细分析）

客户端总体用例图

图2.1为该课题实现程序的客户端系统总体用例图，也就是给用户使用的应用的基本用例情况。本课题主要研究方向为服务端的开发，客户端为次要实现方向，客户端存在的意义仅是为了展示后端的可视化结果，因此用例情况较少。

对于客户端，首先考虑到的用例使用场景就是启动程序，即通过点击图标进入程序。该用例主要涉及到客户端的通信协议初始化、客户端界面初始化、以及界面选择框对应属性初始化。启动用例之后，客户端方面会出现多个电影标签给用户选择，该用例主要涉及到对应标签选择框的值的变更。当用户选择完电影标签之后，用户需要点击推荐，实现前后端通信，获取推荐结果，即上图的用例3，主要涉及到请求包的生成，前后端的网络通信协议、应答包的解析、推荐结果的可视化展示。

## 2.3 用例中的类

图2.2 TAG电影推荐系统客户端类图

上述图2.2为TAG电影推荐系统客户端的类图。

其属性包含init\_window：tkinter外部包窗口类、tag\_state：电影标签勾选框对应的状态、output\_text：绑定输出标签的可变值，用于输出服务端应答包解析结果、socket：前后端通信用的socket、tag\_map：存放中文标签与数字对应的映射。

其功能函数包括center\_window：用于计算窗口在全屏幕居中的坐标值，用于更改可视化窗口初始显示的位置、check\_button\_command：绑定电影标签勾选框的函数，用于更改标签对应的状态、set\_tag：抽象出在窗口中设置标签的函数，减少重复代码编写，作用是在窗口对应坐标设置标签勾选框、set\_init\_window：初始化客户端可视化窗口，主要是设置主题文本、标签勾选框、推荐按钮、tag\_select\_button\_command:推荐按钮绑定的执行函数，主要是构造请求包、接收应答包以及解析数据，再对展示文本进行更改、send\_request：通过私有变量socket来发送构建好的请求包给服务端、tcp\_protocol\_handle：对接收包使用自定义tcp协议进行解析的执行函数、get\_current\_time：获取当前时间，用于显示推荐请求发送时的时间、gui\_start：集成了客户端可视化展示的对外接口，调用时会通过socket与服务端建立TCP连接。

## 2.4 用例主体行为（主要业务逻辑）

### 2.4.1 用户打开程序

简述：用例1指用户启动程序，通过点击应用的图标进入应用程序。

图示

描述已自动生成

图2.2 用户打开程序泳道图

见上图2.2用例1泳道图，用例1的流程如下：用户打开客户端程序，客户端会执行对外接口函数，然后初始化客户端的可视化窗口、设置标签勾选框、推荐按钮。接着初始化socket与服务端建立TCP连接，显示可视化窗口提供给用户使用。

### 2.4.2 用户选择TAG

简述：用户选择自己需要的电影标签的勾选框，用于后续电影推荐。

图示

描述已自动生成

图2.3 用户选择TAG泳道图

见上图2.3用例2泳道图，用例2的流程如下：用户选择电影标签并点击勾选框，客户端会执行勾选框对应的函数，判断该勾选框的值，如果为True则设置成False、如果是False则设置成True，用于后续推荐请求包的构建。

### 2.4.3 用户点击推荐

简述：用户点击推荐按钮，客户端会通过用户选择给用户推荐对应标签的电影。

图图示

描述已自动生成2.4 用例3泳道图

见上图2.4用例3泳道图，用例3的流程如下：用户点击推荐按钮，客户端通过标签勾选框的状态值来构建请求包，通过socket将请求包发送给服务端。服务端部分处理完请求之后，将应答包发送回客户端，客户端再使用自定义TCP协议进行解析，最后进行结果展示。

## 2.5 用例异常行为（异常场景）

可能有以下异常场景出现：

1. 用户在打开应用程序之后，跳过了选择TAG的部分，直接点击推荐按钮。这种情况客户端会进行判断，如果所有标签的状态都为False，则不对用户请求进行处理。
2. 用户在选择TAG的部分，选择了非常多的TAG，导致筛选出来的电影列表为空，这种情况客户端不对信息进行展示。

③ 用户在短时间内发送大量请求给服务端。这种情况下，服务端方面有预防措施：即每次处理同一个IP端口发送的请求时，会进行sleep线程休眠，间隔一秒之后再接收请求，这样就能方式恶意请求的行为。

# 概要设计

## 开发环境及配置

本系统使用C/S模型[8]进行通信，因此开发分为两部分：

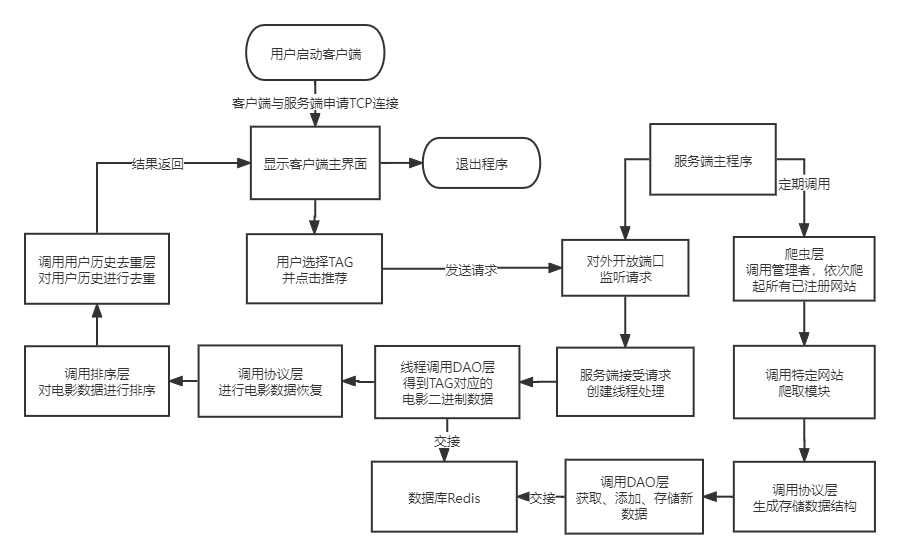
第一部分是服务端的开发，第二部分是客户端的开发。关于服务端的开发环境，我在国内某云服务器租赁平台购置了一台云服务器，其配置为CPU4核、内存4GB、云硬盘50GB、带宽1Mbps，系统为Ubuntu 20.04 LTS 64位。在Windows上使用VS code的扩展程序ssh连接云服务器进行开发。VS code的ssh[9]自带有机器目录显示功能，且能打开远程机器的文本在编辑器中开发，并且能通过选择函数或者变量名进行源头跳转，对于Python的安装以及Python各种外部库的安装，只需要使用pip命令即可完成，十分方便。

第二部分是在客户端的开发，客户端是给用户使用的应用，因此在大众化的windows平台下进行开发，只需要在Python官网下载安装即可，客户端没有使用外部库，因此只需要原生Python即可使用。

上述提到的Python开发版本均为3.10。

## 系统分解（体系结构）

## 系统总体流程



系统总体流程图

上图3.1描述了整个系统的总体流程图。

用户启动客户端，客户端会与服务端申请建立TCP连接，然后显示出客户端的可视化主界面。此时用户可以选择退出程序，或者选择TAG并点击推荐。当用户点击推荐按钮后，会向服务端发送一个请求包，服务端主程序会创建一个线程对这个请求包进行处理，解析出请求包里的所有TAG，并且获得这些TAG对应数据库中的数据，加以排序去重，再构造出应答包返回给客户端，客户端对数据进行解析，然后显示在可视化界面中。服务端方面，服务端主程序会申请一个线程定期调用爬虫层的管理者，以此爬取在配置中已注册的所有网站，并且解析数据，存放到数据库中。

# 详细设计

## 类（数据）设计

系统总体类图

如上图4.1系统总体类图，囊括了系统所有的类设计。接下来会对上图所提到的类进行逐一简单介绍。

DataParser（配置解析器）：该类是一个配置解析集成器，字面意义上来说就是读取该服务的整个配置文件，并根据配置中的key进行分发解析，存入不同的公有成员变量供外界使用。

TagFilmDao（DAO层执行器）：DAO[10]是Data Access Object的简称，也就是数据访问对象，它是为某种类型数据库或其他持久化机制提供抽象接口的一种模式。通过将应用程序映射到持久性层，DAO层只是对外界提供一些特定的数据操作接口，而不会对外暴露出数据库的详细信息，从而支持单一责任原则，保证隔离性。因此外部可以通过生成该类，调用该类的接口来对数据库进行操作。

FilmInfo（PB协议生成电影信息结构体）：该类是使用PB协议生成的用于保存电影信息的结构体，其成员变量包含一部电影的重要评价信息。

TagFilms（PB协议生成电影列表结构体）：该类同样是使用PB协议生成的结构体，其属性包含repeat FilmInfo，表示许多部电影信息的集合，主要用于序列化后作为value存储到数据库中。

PbFilmStructCreator（PB结构体创造者）：该类包括了通过一定信息生成FilmInfo与TagFilms的方法提供给外界使用。

FilmRanker（电影排序者）：该类提供给外界一个电影排序[11]方法，可以传入一个TagFlims对象进行排序，返回一个排序后的对象。

CrawlerHandleManager（爬虫策略模式执行管理者）：爬虫模块使用策略模式[12]实现，该类为策略模式中的执行管理者，作用是对在配置中已注册的网站进行分发执行。

WebCrawler（爬虫执行者[13]）：该类为爬虫执行类，根据不同网站创建不同的爬虫执行类继承这个基类，再根据其接口重写函数。，爬虫策略模式执行管理者边能通过调用基类的函数执行所有子类重写的函数。

DataUpdater（数据更新器）：该类用于更新爬虫获取的最新数据到数据库。在4.4功能模块具体实现中会具体描述该更新器的具体实现流程。

Util（通用模块）：该类是整个服务端的通用模块，即抽象出整个服务端代码中可能复用的函数。

TagRecommendSystemServer（电影推荐系统服务端）：该类是系统服务端运行的核心代码，集成了服务端的所有功能，定期执行爬虫并更新数据、生成线程监听外部请求，处理并生成返回数据返送给客户端。

TagRecommendSystemClient（电影推荐系统客户端）：该类是提供给用户使用的应用客户端，主要提供可视化编程[14]UI界面，创建、发送请求包，处理返回数据并展示。在2.2用例分析中有对客户端功能与执行流程的全面说明。

## 关键逻辑、算法设计

关于数据库方面，本系统使用Redis[15]作为数据库，原因是Redis是基于内存的数据库，它存储的数据是半结构化的，就意味着其相较于其他结构化、多范式规则的数据库系统来说，具备更强的性能优势，因为在内存中进行的数据读或者数据写，比对磁盘进行的数据读或者数据写要快几十上百倍。本系统中在电影数据存储与用户历史数据去重中都使用Redis来实现，是基于效率方面的考虑。

在电影数据存储方面，本系统使用键值对来存储数据，其中Key是提供给用户选择的电影标签、Value是该电影标签下对应的所有电影的序列化后的二进制数据，这部分二进制数据是通过PB协议生成结构体的序列化与反序列化函数生成的。在爬虫获取到电影数据之后，会调用数据更新器将这部分新数据更新到数据库中，采取的流程是先从数据库中取出Tag对应的原有数据、反序列化为PB结构体、添加新数据、再序列化为二进制数据、再替换数据库中该Tag对应的二进制数据。这样就可以将一个电影列表结构体转变成一个字符串，从而实现数据库中键值对为string对string的存储，保证数据库的效率。

用户历史去重部分，是通过Redis的Bloom过滤器模块来实现的。Bloom过滤器是一种个二进制数组，主要用于判断某数据是否已经存在。在添加key的时候使用多个hash算法把字符串映射到多个位置值，将这些位置值设置为True。判断的时候同样通过这些hash算法算出对应位置值，如果存在一个位置不为True则这个数据必不存在。Bloom过滤器的优势就在于内存使用比Set要小得多，也支持扩容。本系统在Bloom过滤器的运用，就是将每个用户ID作为Key，对应生成一个Bloom过滤器。这里的扩容算法使用指数型提升，即给轻量用户分配较少的空间、给重度用户每次扩容满都分配双倍的空间，直到上限值。这样一来相较于给每个用户都分配相同空间的方法，会更节省内存，同时让Bloom过滤器的命中率更高。

爬虫部分的代码设计采用的是策略设计模式。策略设计模式是属于行为型模式、意图就是定义多个相关的执行算法，使它们可以互相替换，主要用于解决多种算法相似的情况下，使用多个判断逻辑带来的复杂性和难以维护性。在本系统的爬虫模块中，通过编写一个爬虫基类，设置一个通用性的爬取函数，让爬取不同网站的子类继承该基类，重写其爬取函数，从而实现调度管理类调用该通用性函数的时候，能爬取所有不同的网站。这样的优势就在于不需要通过多个判断逻辑来实现爬取不同网站，并且在后续需要添加新网站的爬虫类时，只需要在配置中注册新网站即可。

在前后端网络传输方面，采取TCP连接，其中网络协议采取自己包装的简单协议，即传输头部包含数据长度、后半部分为数据本体的设计。原因是TCP连接能保证了用户使用的客户端一定能接受到服务端发送的数据，保证应用的稳定性。

## 功能模块具体实现

### 4.3.1 程序目录结构

服务端程序目录结构

文本

描述已自动生成图4.2客户端程序目录结构

图4.1与图4.2展示了整个工程的目录结构，其中图4.1为服务端程序目录结构、图4.2为客户端程序目录结构。

如图4.2，客户端部分仅有三个文件，主要功能实现在client\_gui.py中，用于客户端可视化界面的设计、前后端交互请求包构建、TCP自定义简单协议解析、数据展示功能。客户端GUI编程所使用的库为python的tkinter，对其中的勾选框、按钮进行函数绑定，对勾选数据进行处理，再通过客户端socket与服务端socket进行TCP连接，从而实现前后端数据链路的流通。

如图4.1，服务端方面包括了七个模块，分别为配置conf（读取与解析模块）、crawler（爬虫模块）、dao（数据库交接层模块）、proto（PB协议模块）、rank（数据排序模块）、util（通用模块）与tag\_recommend\_system\_server（服务端主程序模块）。接下来的部分会对功能模块实现进行详细介绍。

### 4.3.2 conf（配置解析模块）

本应用所采用的配置文件格式为yaml，python中拥有对yaml文件的解析库‘yaml’，使用该库的load函数即可对读取的yaml文件字符串进行解析，得到对应字典与数组。

本模块编写的类为DataParser，即配置解析器，对构造函数传入的配置文件名做解析处理。其中内部的原理是对解析yaml后的数据进行分发处理，分别分发到website（网站）配置解析、data\_base（数据库）配置解析、crawler（爬虫）配置解析、port（对外开放端口）配置解析对应的函数,然后这些函数对各自需要的数据进行提取，存放到DataParser公开成员变量中，这样就保证了各配置之间的解析的解耦性[16]，使之后新增配置解析功能更加方便。这样一来，当外接初始化生成DataParser类成员之后，就能直接通过DataParser的公开成员变量来拿到所需要的配置数据。并且这部分的解析过程是完全不对外开放的，从而保证了配置解析模块与外界的解耦性。如果需要新增配置解析的话，就直接在类中新增所需要的配置解析函数，并在分发函数中添加该函数即可。

### 4.3.3 util（通用函数模块）

本模块为通用函数模块，主要用于存放整个服务端可能会重复使用的函数，将这部分函数抽象出来，当需要使用其中的某个函数时，调用该模块即可。该模块的目的是为了减少代码的重复性，从而减少后续维护的时间成本与出错成本。

目前本模块存放的函数有两个，第一个函数是append\_project\_path（添加项目子目录到系统变量），Python对本地模块的引用，是需要本地模块存在于同级目录之下或者系统变量的目录之下。因此，几乎所有模块需要使用到其他模块的时候，都需要重复添加将其他目录添加到系统变量之下的步骤。本函数的功能，即为将项目的所有模块目录都添加到系统变量中，减少代码的重复性，降低修改维护的复杂度。本函数使用到了python的sys与os库，分别用于获取当前目录与添加目录到系统变量的功能。

第二个函数是get\_cur\_info（获取调用方函数名与堆栈的帧对象），这个函数的功能是获取目前系统时间、调用时的文件目录、文件名以及函数名，如下所示：

2022-04-20 02:35:54:/home/ubuntu/github/tag\_recommend\_system/server/dao/tag\_films\_dao.py:\_\_init\_\_[30]: pool init successfully!, host=localhost, port=6379

2022-04-20 02:35:54:/home/ubuntu/github/tag\_recommend\_system/server/proto/data\_updater.py:data\_restore[59]: data restore complete!

该函数的目的主要是用于服务端的代码debug，当某函数执行到一定进度的时候，服务端会在该行打印出这个函数返回的信息与自定义的debug信息，从而表明函数运行到了哪里。当服务端出错的时候，就能根据debug信息快速定位到出错的文件与运行行数。因此，该函数的功能也是非常常用且重要的功能之一，抽象出来便能大大减少代码的重复性。

### 4.3.4 proto（存储协议模块）

本模块为存储协议模块，主要功能是进行存储协议的声明、存储结构体构造、存储结构体转通用结构。

本应用采用的存储协议为Google protobuf协议。存储协议是用于序列化结构数据的灵活、高效、自动的方法。使用协议生成的源代码可以在各种数据流和各种高级语言之间读写结构体数据。在本应用中，协议主要用于序列化数据在Redis的存储。

模块包括tag\_recommend\_system.proto文件，该文件是用于PB协议结构体的定义，使用的是类C的结构语言编写，这里主要定义了film\_info（电影信息存储结构）、tag\_films（多部电影信息的集合）如下所示：

syntax = "proto3";

package tag\_recommend\_system;

// 电影信息

message film\_info {

  string film\_id = 1;               // 电影名

  string web\_id = 2;                // 网站名

  double web\_score = 3;              // 网站的打分

  double web\_full\_score = 4;         // 网站的满分

  int32 date = 5;                  // 上映年份

  repeated string tags = 6;         // 类型

  string areas = 7;                 // 制作国家

  string directors\_and\_actors = 8;  // 导演、演员

  string quote = 9;                // 短评

}

message tag\_films {

  // 一个tag对应的所有电影信息

  repeated film\_info film\_infos = 1;

}

film\_info结构体定义的存储信息包括电影名、网站名、网站打分、网站满分、上映年份、电影类型、制作国家、导演与演员信息、电影短评。tag\_films属性是一个film\_info数组。

本模块还包括film\_struct\_creator（电影PB协议结构体创造者）与rsp\_struct\_creator（PB协议结构体转换者）。前者实现的功能是将python原生数据转换成PB协议结构体，而后者实现的功能则是相反，将PB协议结构体转换成python原生结构体。除此之外，本模块还有data\_updater（数据更新器），该类则是用于数据库数据的更新，传入参数为需要更新的电影数据列表。其逻辑如下：将传入的所有电影数据的标签对应他们的电影名列表存放到字典中，然后遍历这些标签，以这些标签为Key从数据库中取出标签对应的电影二进制数据，使用PB协议结构体自动生成的反序列化函数生成PB结构体。将标签对应的所有电影名与这些PB结构体的电影名进行去重，然后再将新数据添加回PB结构体中，接着进行序列化，再使用对应的标签作为Key将数据覆盖回数据库原来的数据，这样就实现了数据库的数据更新。

### 4.3.5 crawler（爬虫模块）

爬虫模块使用策略设计模式进行设计，模块分为一个crawler\_core（爬虫核心调用部分）与多个[website\_name]\_crawler（网站爬取与解析部分，[website\_name]指不同网站的名称）。crawler\_core主要实现CrawlerHandleManager（爬虫调度执行管理者）类，当该类进行初始化的时候会传入一个DataParser（配置解析器）类对象，获得各网站的配置信息，包括网站权重、爬取url以及网站爬取开关。外部调用该类的handle（执行）函数，就会循环遍历所有网站配置，只有打开了网站爬取开关的网站才会对其进行爬虫的执行，然后会调度到dispatch（分发）函数，通过网站配置进行函数分发，从而调度到上述提到的[website\_name]\_crawler部分，进行网站的爬取与数据解析。这样设计的好处就在于，减少了各网站爬虫与解析的耦合性，并且减少了新增爬取网站代码的维护难度。当需要新增爬取网站的时候，只需要在配置中新增该网站的配置信息，然后在新增对应网站的爬取与解析部分的文件，在里面编写对应代码即可。

### 4.3.6 dao（数据库交接层模块）

DAO，在图4.1系统总体类图下方介绍TagFilmDao类的时候已经提及，就是用于对数据库进行直接操作，封装提供给外部对数据库进行操作的接口的模型层。他的优势就在于，将对数据库进行直接操作的部分对外界隐藏了起来，提高了数据库的安全性，并且减少了外部编写操作数据库的代码的重复性。这样一来，同时还减少了整个的系统模块与模块之间的耦合性。

本应用的DAO层封装了TagFilmsDao类作为对外部提供接口的类，初始化之后直接调用其公开接口就能通过key与value对数据库进行添加数据、获取数据等操作。使用的是python对redis进行操作的库‘redis’。

### 4.3.7 rank（电影数据排序模块）

本模块是电影数据排序模块，用于对电影数据列表进行排序，传参为电影数据列表、DataParser（配置解析器），列表中的数据类型则是上述提到的PB协议生成结构体film\_info。排序逻辑是先建立每部电影对应多个电影网站平台的字典，遍历这些电影，从配置解析器类对象中得到每个电影的权重，对进行计算：网站打分除与网站总分的值乘与网站占比权重，就能得到单个网站得分，然后将所有网站的得分相加，再进行排序，最终就能得到排序后的电影数据列表。

### 4.3.8 tag\_recommend\_system\_server（服务端核心模块）

服务端核心模块的功能主要有两个，第一个是建立与客户端之间的TCP连接，并接受客户端的请求进行处理，并发送回应包。第二个则是定期调用上述提到的爬虫模块的类CrawlerHandleManager的handle函数，更新数据库的数据。该间隔时间可以在配置中进行更改。

服务端核心模块运行的时候，会先调用配置解析模块，初始化一个DataParser类对象，然后再生成一个CrawlerHandleManager类对象，创建一个线程，定期调用CrawlerHandleManager类对象的handle函数。然后会创建一个socket，并对外进行请求监听。当接受到请求的时候，会创建一个线程用于执行请求处理函数，在请求处理函数中执行对数据的解码，并且执行应答包的构建。应答包的构建逻辑如下：获取请求中的所有电影标签，然后调用DAO层得到这些标签对应的二进制电影数据，并执行反序列化函数，得到PB结构体，然后使用set与电影名对电影数据进行去重处理，再调用排序模块对电影数据进行排序，而后将排名靠前的数据返回。线程对这些数据进行包装，将头部设置一个字节长度与间隔符号，在后文放置正文数据，生成应答包，将该包返送给客户端。

# 测试实施

## 功能测试

### 5.1.1 配置读取模块

将系统需要使用的配置数据集放入tag\_recommend\_system.yaml

yaml配置文件数据如图所示：

website:

  - name: 'douban'

    website\_weight: 60

    switch: 1

    url: 'https://movie.douban.com/top250'

  - name: 'maoyan'

    website\_weight: 50

    switch: 1

    url: 'https://maoyan.com/board/4?'

  - name: 'rotten\_tomato'

    website\_weight: 70

    switch: 1

    url: 'https://editorial.rottentomatoes.com/guide/popular-movies/'

redis:

  ip: 'localhost'

  port: 6379

crawler:

  # 爬虫间隔时间，暂定12小时，单位毫秒

  crawl\_interval: 43200000

# 对外开放的端口

ports:

  - 9997

执行配置模块测试代码：

  config = config\_parser.DataParser('tag\_recommend\_system.yaml')

  print(config.website, config.crawler, config.redis, config.port\_list)

测试结果：输出配置解析器的公有成员变量，得到了解析后的配置数据。

/home/ubuntu/github/tag\_recommend\_system/server/conf/config\_parser.py:\_\_init\_\_[25]: yaml\_data parse complete!

[{'name': 'douban', 'website\_weight': 60, 'switch': 1, 'url': 'https://movie.douban.com/top250'}, {'name': 'maoyan', 'website\_weight': 50, 'switch': 1, 'url': 'https://maoyan.com/board/4?'}, {'name': 'rotten\_tomato', 'website\_weight': 70, 'switch': 1, 'url': 'https://editorial.rottentomatoes.com/guide/popular-movies/'}] {'crawl\_interval': 43200000} {'ip': 'localhost', 'port': 6379} [9997]

### 5.1.2 通用模块层

通用模块层共有两个函数，其中append\_project\_path的测试方式是执行该函数，然后尝试调用配置模块的解析函数并输出；get\_cur\_info的测试方式是直接输出查看数据即可。测试代码如下：

append\_project\_path()

import config\_parser

print(get\_cur\_info())

config = config\_parser.DataParser(now\_path + "/../conf/tag\_recommend\_system.yaml")

测试结果可以发现当调用配置解析的时候，也会使用到get\_cur\_info作为日志输出，可以发现执行成功。

2022-04-20 04:37:00:test\_util.py:<module>[10]:

2022-04-20 04:37:00:/home/ubuntu/github/tag\_recommend\_system/server/util/../conf/config\_parser.py:\_\_init\_\_[25]: yaml\_data parse complete!

### 5.1.3 存储协议模块

使用PB协议生成源代码直接使用生成命令：

protoc --python\_out=. tag\_recommend\_system.proto

可以发现在同级目录下会自动生成tag\_recommend\_system\_pb2.py的文件，再引用该py文件即可使用PB协议定义的结构体。

数据更新器模块、电影PB协议结构体创造者直接与爬虫模块进行联调测试。即爬虫数据解析完毕后，调用电影PB协议结构体创造者生成PB结构，再调用数据更新器更新到数据库中，进入数据库通过key查看数据是否更新成功即可。

### 5.1.4 爬虫模块

执行爬虫模块测试代码：

  config = config\_parser.DataParser(now\_path + "/../conf/tag\_recommend\_system.yaml")

  url\_handle\_manager = crawler\_core.UrlHandleManager(config)

  url\_handle\_manager.handle()

出现日志信息：

2022-04-24 03:36:01:/home/ubuntu/github/tag\_recommend\_system/server/dao/tag\_films\_dao.py:\_\_init\_\_[30]: pool init successfully!, host=localhost, port=6379

2022-04-24 03:36:01:/home/ubuntu/github/tag\_recommend\_system/server/proto/data\_updater.py:data\_restore[59]: data restore complete!

由于爬取并解析后的数据结果过长，此处近展示小部分数据：

{'title': '肖申克的救赎', 'rating': '9.7', 'dates': '1994', 'tags': ['犯罪', '剧情'], 'areas': '美国', 'directors\_and\_actors': '导演: 弗兰克·德拉邦特 Frank Darabont 主演: 蒂姆·罗宾斯 Tim Robbins /...', 'quote': '希望让人自由。'}

{'title': '霸王别姬', 'rating': '9.6', 'dates': '1993', 'tags': ['剧情', '爱情', '同性'], 'areas': '中国大陆 中国香港', 'directors\_and\_actors': '导演: 陈凯歌 Kaige Chen 主演: 张国荣 Leslie Cheung / 张丰毅 Fengyi Zha...', 'quote': '风华绝代。'}

{'title': '阿甘正传', 'rating': '9.5', 'dates': '1994', 'tags': ['剧情', '爱情'], 'areas': '美国', 'directors\_and\_actors': '导演: 罗伯特·泽米吉斯 Robert Zemeckis 主演: 汤姆·汉克斯 Tom Hanks / ...', 'quote': '一部美国近现代史。'}

进入Redis，执行命令（得到key为’恐怖’对应的value）

get ‘恐怖’

可以得到数据为序列化后的字节数据。将其提取出来再进行反序列化解析，便得到了上述原本的电影数据结构体。

"\n\xbb\x01\n\x0c\xe7\x94\xb5\xe9\x94\xaf\xe6\x83\x8a\xe9\xad\x82\x12\x06douban\x19ffffff!@!\x00\x00\x00\x00\x00\x00$@(\xd4\x0f2\x06\xe6\x82\xac\xe7\x96\x912\x06\xe6\x83\x8a\xe6\x82\x9a2\x06\xe6\x81\x90\xe6\x80\x96B\x06\xe7\xbe\x8e\xe5\x9b\xbdJW\xe5\xaf\xbc\xe6\xbc\x94: \xe8\xa9\xb9\xe5\xa7\x86\xe6\x96\xaf\xc2\xb7\xe6\xb8\xa9 James Wan \xe4\xb8\xbb\xe6\xbc\x94: \xe9\x9b\xb7\xc2\xb7\xe6\xb2\x83\xe7\xba\xb3\xe5\xb0\x94 Leigh Whannell / \xe5\x8a\xa0\xe5\x88\xa9\xc2\xb7\xe8\x89\xbe...R\x15\xe7\x9c\x9f\xe7\x9b\xb8\xe5\xb0\xb1\xe5\x9c\xa8\xe7\x9c\xbc\xe5\x89\x8d\xe3\x80\x82\n\xdb\x01\n\t\xe6\x83\x8a\xe9\xad\x82\xe8\xae\xb0\x12\x06douban\x19\x00\x00\x00\x00\x00\x00\"@!\x00\x00\x00\x00\x00\x00$@(\xa8\x0f2\x06\xe6\x82\xac\xe7\x96\x912\x06\xe6\x83\x8a\xe6\x82\x9a2\x06\xe6\x81\x90\xe6\x80\x96B\x06\xe7\xbe\x8e\xe5\x9b\xbdJ\\\xe5\xaf\xbc\xe6\xbc\x94: \xe9\x98\xbf\xe5\xb0\x94\xe5\xbc\x97\xe9\x9b\xb7\xe5\xbe\xb7\xc2\xb7\xe5\xb8\x8c\xe5\x8c\xba\xe6\x9f\xaf\xe5\x85\x8b Alfred Hitchcock \xe4\xb8\xbb\xe6\xbc\x94: \xe5\xae\x89\xe4\xb8\x9c\xe5\xb0\xbc\xc2\xb7\xe5\x8d\x9a\xe9\x87\x91\xe6\x96\xaf Antho...R3\xe6\x95\x85\xe4\xba\x8b\xe7\x9a\x84\xe5\x8f\x8d\xe8\xbd\xac\xe4\xb8\x8e\xe5\x8f\x8d\xe8\xbd\xac\xef\xbc\x8c\xe5\x88\x86\xe8\xa3\x82\xe7\x94\xb5\xe5\xbd\xb1\xe7\x9a\x84\xe5\xa7\x8b\xe7\xa5\x96\xe3\x80\x82"

### 5.1.5 数据库交接层模块

DAO模块执行器测试，主要测试单个数据存储、批量数据存储、单个数据读取、批量数据读取的接口，测试代码如下：

config = config\_parser.DataParser(os.getcwd() + r"/../conf/tag\_recommend\_system.yaml")

  tag\_films\_dao = TagFilmsDao(host=config.redis['ip'], port=config.redis['port'])

  print('set\_new\_value return=' + str(tag\_films\_dao.set\_new\_value('test\_set\_key', 'test\_set\_value\_1')))

  keys = ['t\_k\_1', 't\_k\_2']

  values = ['t\_v\_1', 't\_v\_2']

  exs = [-1, -1]

  print('set\_value\_batch return=' + str(tag\_films\_dao.set\_value\_batch(keys, values, exs)))

  print(tag\_films\_dao.get\_value('test\_set\_key'))

  print(tag\_films\_dao.get\_value('no\_key'))

  print(tag\_films\_dao.get\_value\_batch(['t\_k\_1', 't\_k\_2']))

测试结果：成功获取到了对应单个存储和批量存储的数据，说明存储与读取均成功。

/home/ubuntu/github/tag\_recommend\_system/server/dao/../conf/ add\_path has been added!

/home/ubuntu/github/tag\_recommend\_system/server/dao/../conf/config\_parser.py:\_\_init\_\_[25]: yaml\_data parse complete!

/home/ubuntu/github/tag\_recommend\_system/server/dao/tag\_films\_dao.py:\_\_init\_\_[30]: pool init successfully!, host=localhost, port=6379

set\_new\_value return=None

set\_value\_batch return=True

b'test\_set\_value\_1'

None

[b't\_v\_1', b't\_v\_2']

### 5.1.6 电影数据排序模块

排序模块的测试方式是使用数据库中key为‘科幻’的数据，对这部分数据进行打乱，再执行排序层的关键逻辑，输出返回的结果。

  config = config\_parser.DataParser(now\_path + "/../conf/tag\_recommend\_system.yaml")

  tag\_film\_dao = tag\_films\_dao.TagFilmsDao()

  data\_bin = tag\_film\_dao.get\_value('科幻')

  tag\_film\_list = tag\_recommend\_system\_pb2.tag\_films()

  tag\_film\_list.ParseFromString(data\_bin)

  film\_list = []

  # 打乱原本的顺序

  length = len(tag\_film\_list.film\_infos)

  for film in range(int(length/2), length):

    film\_list.append(tag\_film\_list.film\_infos[film])

  for film in range(0, int(length/2)):

    film\_list.append(tag\_film\_list.film\_infos[film])

  films\_after\_rank = films\_ranker.rank\_films(film\_list, config)

  file = open('films\_after\_rank', 'w')

  for film in films\_after\_rank:

    file.write(str(film) + '\n')

  file.close()

由于排序后的结果数据太长，此处仅展示部分结果数据：带有‘科幻’标签的电影按一定算法对所有网站权重的权值打分。

{'name': '盗梦空间', 'film\_weighted\_value': 56.4, 'film\_web\_list': ['douban得分:9.4/10.0'], 'film\_dates': 2010, 'film\_tags': ['剧情', '科幻', '悬疑', '冒险'], 'film\_areas': '美国 英国', 'film\_directors\_and\_actors': '导演: 克里斯托弗·诺兰 Christopher Nolan 主演: 莱昂纳多·迪卡普里奥 Le...', 'film\_quote': '诺兰给了我们一场无法盗取的梦。'}

{'name': '星际穿越', 'film\_weighted\_value': 56.4, 'film\_web\_list': ['douban得分:9.4/10.0'], 'film\_dates': 2014, 'film\_tags': ['剧情', '科幻', '冒险'], 'film\_areas': '美国 英国 加拿大', 'film\_directors\_and\_actors': '导演: 克里斯托弗·诺兰 Christopher Nolan 主演: 马修·麦康纳 Matthew Mc...', 'film\_quote': '爱是一种力量，让我们超越时空感知它的存在。'}

{'name': '楚门的世界', 'film\_weighted\_value': 55.8, 'film\_web\_list': ['douban得分:9.3/10.0'], 'film\_dates': 1998, 'film\_tags': ['剧情', '科幻'], 'film\_areas': '美国', 'film\_directors\_and\_actors': '导演: 彼得·威尔 Peter Weir 主演: 金·凯瑞 Jim Carrey / 劳拉·琳妮 Lau...', 'film\_quote': '如果再也不能见到你，祝你早安，午安，晚安。'}

{'name': '机器人总动员', 'film\_weighted\_value': 55.8, 'film\_web\_list': ['douban得分:9.3/10.0'], 'film\_dates': 2008, 'film\_tags': ['科幻', '动画', '冒险'], 'film\_areas': '美国', 'film\_directors\_and\_actors': '导演: 安德鲁·斯坦顿 Andrew Stanton 主演: 本·贝尔特 Ben Burtt / 艾丽...', 'film\_quote': '小瓦力，大人生。'}

### 5.1.7 服务端与客户端核心模块

由于服务端的特殊性，因此需要与客户端部分进行联调测试。

登录云服务器运行服务端部分代码，监听外部请求，得到如下日志，可以发现爬取各个网站数据成功，并解析存储成功，同时在监听外部请求中。

2022-04-21 10:56:35:/home/ubuntu/github/tag\_recommend\_system/server/conf/config\_parser.py:\_\_init\_\_[25]: yaml\_data parse complete!

2022-04-21 10:56:35:/home/ubuntu/github/tag\_recommend\_system/server/crawler/crawler\_core.py:\_\_init\_\_[32]: UrlHandleManager init complete!

2022-04-21 10:56:35:/home/ubuntu/github/tag\_recommend\_system/server/crawler/douban\_crawler.py:crawl\_douban[32]: crawl douban content!

2022-04-21 10:56:35:tag\_recommend\_system\_server.py:main[96]: Waiting for connection...

2022-04-21 10:56:38:/home/ubuntu/github/tag\_recommend\_system/server/dao/tag\_films\_dao.py:\_\_init\_\_[30]: pool init successfully!, host=localhost, port=6379

2022-04-21 10:56:38:/home/ubuntu/github/tag\_recommend\_system/server/proto/data\_updater.py:data\_restore[59]: data restore complete!

2022-04-21 10:56:38:/home/ubuntu/github/tag\_recommend\_system/server/crawler/maoyan\_crawler.py:crawl\_maoyan[33]: crawl maoyan content!

2022-04-21 10:56:38:/home/ubuntu/github/tag\_recommend\_system/server/dao/tag\_films\_dao.py:\_\_init\_\_[30]: pool init successfully!, host=localhost, port=6379

2022-04-21 10:56:38:/home/ubuntu/github/tag\_recommend\_system/server/proto/data\_updater.py:data\_restore[59]: data restore complete!

2022-04-21 10:56:38:/home/ubuntu/github/tag\_recommend\_system/server/crawler/rotten\_tomato\_crawler.py:crawl\_rotten\_tomato[32]: crawl rotten\_tomato content!

2022-04-21 10:56:41:/home/ubuntu/github/tag\_recommend\_system/server/dao/tag\_films\_dao.py:\_\_init\_\_[30]: pool init successfully!, host=localhost, port=6379

2022-04-21 10:56:41:/home/ubuntu/github/tag\_recommend\_system/server/proto/data\_updater.py:data\_restore[59]: data restore complete!

在Windows平台运行客户端代码，生成可视化界面，选择TAG：’恐怖’，并点击推荐，服务端部分接收到请求：

2022-04-24 11:01:09:tag\_recommend\_system\_server.py:tcp\_link[72]: Accept new connection from 113.108.77.72:10518

receive data:恐怖

2022-04-24 11:08:41:/home/ubuntu/github/tag\_recommend\_system/server/dao/tag\_films\_dao.py:\_\_init\_\_[30]: pool init successfully!, host=localhost, port=6379

2022-04-24 11:08:41:/home/ubuntu/github/tag\_recommend\_system/server/rank/films\_ranker.py:rank\_films[68]: tag\_films rank complete!

可以发现请求处理成功，返回结果。客户端部分得到数据之后进行解码，得到部分数据如下：

["{'name': '惊魂记'", "'film\_weighted\_value': 54.0", "'film\_web\_list': ['douban得分:9.0/10.0']", "'film\_dates': -1", "'film\_tags': ['悬疑'", "'惊悚'", "'恐怖']", "'film\_areas': ''", "'film\_directors\_and\_actors': '美国'", "'film\_quote': '导演: 阿尔弗雷德·希区柯克 Alfred Hitchcock 主演: 安东尼·博金斯 Antho...'}", "{'name': '电锯惊魂'", "'film\_weighted\_value': 52.2", "'film\_web\_list': ['douban得分:8.7/10.0']", "'film\_dates': -1", "'film\_tags': ['悬疑'", "'惊悚'", "'恐怖']", "'film\_areas': ''", "'film\_directors\_and\_actors': '美国'", "'film\_quote': '导演: 詹姆斯·温 James Wan 主演: 雷·沃纳尔 Leigh Whannell / 加利·艾...'}"]

# 安装、使用说明

## 安装说明

### 6.1.1 安装前准备

本应用在windows 64位系统采用Pyinstaller进行打包生成exe，因此不需要安装任何前置软件，但是32位系统无法使用。

### 6.1.2 系统安装

将下载的软件zip包进行解压，其中的exe文件即为应用程序。

## 使用说明

打开exe文件，选择想要观看的电影标签，点击推荐按钮，等待推荐信息显示即可。

# 开发、维护环境设置说明

指导如何在全新的机器上在源代码细节级别上进行阅读、修改、重新编译、调试等开发维护工作。建议在VMWare等虚拟机上模拟该过程。

## 软件开发环境概述

服务端开发环境为Ubuntu 20.04，客户端开发环境为Windows10 64bits。使用Python版本3.10.1进行开发。

服务端Ubuntu python库安装

sudo pip3 install redis

sudo pip3 install UserAgent

sudo apt-get install Python-bs4

sudo pip3 install protobuf

服务端Ubuntu Redis数据库安装

sudo apt update

sudo apt install redis-server

# 结论

## 总结

由于在做毕设之前一直在进行实习，于是通过自己在实习期间总结的在线系统运行流程化的经验、网络通信相关方面的经验、以及关于代码结构设计、代码安全规范等等关于技术方面的学习之后、通过使用请假的一个多月，将自己总结的经验运用到该毕设项目，较为体系化地对整个系统进行了一定的设计。首先是完成了设计的模块图，划分出各个模块负责的功能、以及各个模块的代码类设计，然后根据模块图，逐步查阅Python官方API文档[17]对这些模块一个个进行实现，并每天对系统的更新过程加以记录，并且在完成所有功能实现之后，我对整个项目的代码进行了重构，重构主要在代码安全与命名规范、注释补全、代码函数抽取解耦设计。在代码实现期间，遇到了种种难以解决的难题：比如网络通信方面的中文编码、TCP通信自定义协议的问题、前端客户端设计的展示显示不全问题、WINDOWS系统编译环境配置与LINUX系统编译环境配置不一致等等难题。对于这些难题，我通过在各大搜索平台查找前人解决的方法，每个难题无一都花费了我巨大的精力才得以解决。一开始遇到难题的不知所措与焦头烂耳的心态，被解决难题过后的喜悦所冲刷，让我不断地成长。完成了这次的毕设，我不但在技术方面对整个系统设计、前后端各自实现与连通的流程更加熟悉了，也让我在非技术方面，比如面对难题心态、解决难题的方法上有很大的进步，这对我在之后的学习与工作上将会有很大的帮助。

## 展望

在前面的代码模块化设计期间、我使用实习时候学习到的代码结构设计方法，以及各种设计模式对模块进行设计。这就导致了我代码的解耦性强，使得系统的模块、函数功能可拓展性很高，在后续的维护、新功能添加上有很大的优势。总结中我提到在完成项目之后，对所有代码进行重构，其中对注释进行补全，就能让后续其他人接管、维护、开发代码有比较大的帮助。重构还包括对代码函数解耦设计，这也令每个函数不会过于耦合，导致新功能添加的困难。最后一项重构事项是代码命名以及安全规范，这个主要体现在整个项目使用同一套代码命名模式、以及代码设计安全化的方面，这对系统稳定性与维护上也有很大的帮助，后续进行升级使用同一套规范即可保证整个系统稳定迭代。

# 参考文献

1. [赵颐南](https://search.cnki.com.cn/Search/Result?author=%E8%B5%B5%E9%A2%90%E5%8D%97" \t "_blank). 中国网络水军对电影行业的危害与治理对策研究[D]. 西北师范大学, 2021
2. Shlomo Berkovsky, Ronnie Taib, Dan Conway. How to Recommend?: User Trust Factors in Movie Recommender Systems[C]. IUI '17: Proceedings of the 22nd International Conference on Intelligent User Interfaces, 2017.287-300
3. Fredrik Lundh. An Introduction to Tkinter[EB/OL]. (2003/03/29). [www.tcltk.co.kr/files/TclTk\_Introduction\_To\_Tkiner.pdf](http://www.tcltk.co.kr/files/TclTk_Introduction_To_Tkiner.pdf)
4. S. M. Bellovin. Security problems in the TCP/IP protocol suite[J]. [ACM SIGCOMM Computer Communication Review](https://dl.acm.org/toc/sigcomm-ccr/1989/19/2), [Volume 19](https://dl.acm.org/toc/sigcomm-ccr/1989/19/2), [Issue 2](https://dl.acm.org/toc/sigcomm-ccr/1989/19/2), 1989.
5. Vineeth G. Nair. Getting Started with Beautiful Soup[M]. Packt Publishing, 2014.65-82

1. [聂晓旭](http://www.cqvip.com/main/search.aspx?w=%e8%81%82%e6%99%93%e6%97%ad) [于凤芹](http://www.cqvip.com/main/search.aspx?w=%e4%ba%8e%e5%87%a4%e8%8a%b9) [钦道理](http://www.cqvip.com/main/search.aspx?w=%e9%92%a6%e9%81%93%e7%90%86). 基于Protobuf的数据传输协议[J]. 计算机系统应用 2015年第8期, 2015.112-116
2. 舒[尹](https://search.cnki.com.cn/Search/Result?author=%E8%88%92%E5%B0%B9" \t "_blank). 探究序列化与反序列化[J]. 通讯世界2019年01期, 2019.
3. HS Oluwatosin. Client-Server Model[J]. IOSR Journal of Computer Engineering (IOSR-JCE) e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 16, Issue 1, Ver. IX, 2014.67-71

1. [Daniel J. Barrett](https://www.google.com.hk/search?hl=zh-CN&tbo=p&tbm=bks&q=inauthor:%22Daniel+J.+Barrett%22), [Daniel J.. Barrett](https://www.google.com.hk/search?hl=zh-CN&tbo=p&tbm=bks&q=inauthor:%22Daniel+J..+Barrett%22), [Richard E. Silverman](https://www.google.com.hk/search?hl=zh-CN&tbo=p&tbm=bks&q=inauthor:%22Richard+E.+Silverman%22), [Richard Silverman](https://www.google.com.hk/search?hl=zh-CN&tbo=p&tbm=bks&q=inauthor:%22Richard+Silverman%22). SSH, the Secure Shell: The Definitive Guide[M]. "O'Reilly Media, 2001.2-10
2. 李晓东, 魏惠茹, 于景茹. DAO设计模式研究[J]. 软件导刊 2014年 第7期, 2014.36-37
3. 绝密伏击. 推荐系统-精排篇[EB/OL]. (2021/12/20). https://zhuanlan.zhihu.com/p/432817787, 2021.
4. 程杰. 大话设计模式[M]. 清华大学出版社, 2007.42-63
5. Xiaoju Ma, Min Yan. Design and Implementation of Crawler Program Based on Python[J]. [Journal of Physics: Conference Series](https://iopscience.iop.org/journal/1742-6596), [Volume 2033](https://iopscience.iop.org/volume/1742-6596/2033), [The Third International Conference on Electrical, Communication and Computer Engineering (ICECCE 2021) Kuala Lumpur,](https://iopscience.iop.org/issue/1742-6596/2033/1)  2021.12-13
6. 康计良. Python语言的可视化编程环境的设计与实现[D]. 西安电子科技大学, 2012.
7. 杨旭. Redis分布式缓存在影片推荐中的应用[J]. 现代电影技术, 2021.31-35
8. Jaymeng8848. 代码耦合和解耦[EB/OL]. (2020/04/10). https://blog.csdn.net/qq\_44836294/article/details/105440190, 2020.
9. Python3.10.4. 官方API文档[EB/OL]. (2022/04/24). http://docs.python.org/zh-cn/3/whatsnew/3.10.html

# 致谢

对那些没有在正文中提及，但帮助过完成该毕业设计课题的个人、团体和组织表达感激之意。