|  |  |
| --- | --- |
| 弯道和树的图片  课程设计报告  大数据软件与技术 课程设计 | 项目名称  基于Spark和Hive实现的电影推荐系统网站平台  组名  第18组  小组成员  张佳明（组长） 王昊 朱长昊 俞振杰 刘震宇 |

目录

[1 引言 1](#_Toc42461124)

[1.1 课程设计题目 1](#_Toc42461125)

[1.2 背景和意义 1](#_Toc42461126)

[2 小组成员及分工 1](#_Toc42461127)

[3 课程设计详述 2](#_Toc42461128)

[3.1 设计内容 2](#_Toc42461129)

[3.2 主要功能 2](#_Toc42461130)

[4 开发工具及环境 3](#_Toc42461131)

[4.1 数据爬取 3](#_Toc42461132)

[4.2 数据存储 3](#_Toc42461133)

[4.3 网站系统 3](#_Toc42461134)

[4.4 推荐系统 3](#_Toc42461135)

[5 系统架构 4](#_Toc42461136)

[5.1 整体系统架构 4](#_Toc42461137)

[5.2 爬虫系统架构 5](#_Toc42461138)

[5.3 网站系统架构 5](#_Toc42461139)

[5.4 推荐系统架构 6](#_Toc42461140)

[6 具体实现 7](#_Toc42461141)

[6.1 爬虫系统 7](#_Toc42461142)

[6.1.1 网页信息获取 7](#_Toc42461143)

[6.1.2 抓取需求信息 7](#_Toc42461144)

[6.1.3 批量存入数据库 11](#_Toc42461145)

[6.1.4 爬虫工作流程图 12](#_Toc42461146)

[6.2 网站系统 15](#_Toc42461147)

[6.2.1 前端 15](#_Toc42461148)

[6.2.2 后端 18](#_Toc42461149)

[6.3 推荐系统 20](#_Toc42461150)

[6.3.1 数据集加载 20](#_Toc42461151)

[6.3.2 离线统计服务 21](#_Toc42461152)

[6.3.3 离线推荐服务 22](#_Toc42461153)

[7 系统演示结果 26](#_Toc42461154)

[7.1 登陆注册 26](#_Toc42461155)

[7.2 首页-轮播图 27](#_Toc42461156)

[7.3 搜索界面 27](#_Toc42461157)

[7.3.1 电影详情 28](#_Toc42461158)

[7.3.2 热门推荐Top10 29](#_Toc42461159)

[7.3.3 个性化推荐 29](#_Toc42461160)

[8 项目实现重难点分析 31](#_Toc42461161)

[8.1 环境与硬件 31](#_Toc42461162)

[8.2 爬虫系统 31](#_Toc42461163)

[8.3 推荐算法 32](#_Toc42461164)

# 引言

## 课程设计题目

基于Spark和Hive实现的电影推荐系统网站平台。

## 背景和意义

随着经济的高速发展，人们的精神追求和娱乐生活日渐丰富。电影作为娱乐的中国要部分，也越来越多受到大众的喜爱和欢迎。在信息爆炸的今天，获取信息的途径和方式多种多样，人们花费时间最多的不再是去哪获取信息，而是要在众多的信息中寻找自己感兴趣的，这就是信息超载问题。如何在海量的电影中找寻到适合每个人口味的电影则是重点，不仅能够很大程度上减少我们寻找的时间，更能够让我们乐在其中。为了解决这个问题，推荐系统应运而生。

推荐系统在日常的网络应用中无处不在，比如网上购物、网上买书、新闻app、社交网络、音乐网站、电影网站等等等等，有人的地方就有推荐。根据个人的喜好，相同喜好人群的习惯等信息进行个性化的内容推荐。电影推荐系统则是可以通过用户的行为，比如用户对电影的打分情况，来判断用户的喜好，再通过使用推荐算法并结合用户的喜好来向用户进行推荐。

协同过滤是推荐系统应用较广泛的技术，该方法搜集用户的历史记录、个人喜好等信息，计算与其他用户的相似度，利用相似用户的评价来预测目标用户对特定项目的喜好程度。

# 小组成员及分工

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 学号 | 分工 | 职务 |
| 张佳明 | 1120173305 | 实现前端部分、撰写课程设计报告、录制demo视频 | 组长 |
| 王昊 | 1120171192 | 实现推荐算法、UI设计 | 组员 |
| 朱长昊 | 1120171227 | 实现前端部分、实现后端部分 | 组员 |
| 俞振杰 | 1120171218 | 实现爬虫部分 | 组员 |
| 刘震宇 | 1120172164 | 参与实现推荐算法 | 组员 |

# 课程设计详述

## 设计内容

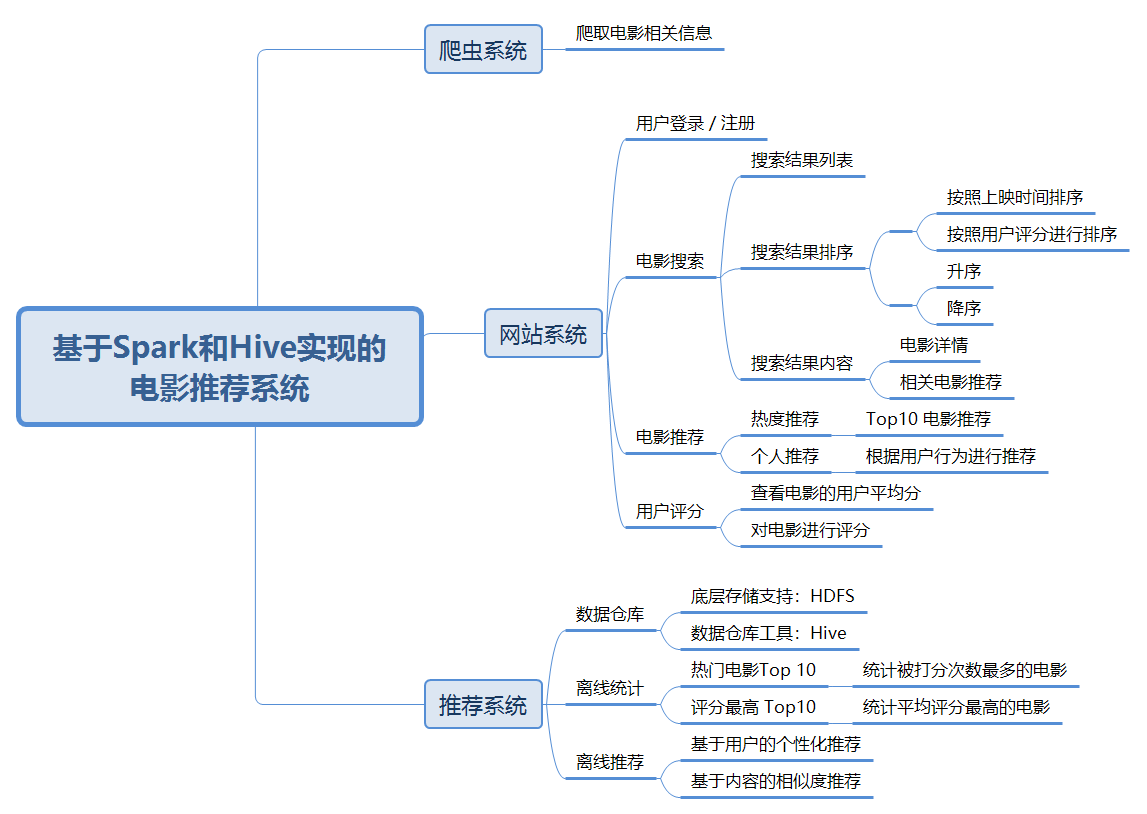
本项目是基于Spark处理引擎和Hive分布式存储实现的电影推荐系统网站平台。

在数据存储部分，首先通过爬虫实现对MovieLens数据集[[1]](#footnote-1)的爬取，同时将数据存储到HDFS分布式存储中，并使用Hive作为数据仓库工具进行离线存储服务。

在网站系统方面，我们实现基于Vue & Spring boot的推荐系统网站，用户可以注册账号后进行登录，可搜索电影、查看电影详情及其相关电影推荐，同时系统会对用户进行两方面电影推荐，一方面是系统通过统计所有的用户数据来进行热度Top的推荐；另一方面是系统将用户的打分行为进行记录，并根据用户日志进行个人推荐。

在推荐系统方面，主要基于离线推荐进行推荐，使用HDFS作为底层存储支持，通过Azkaban周期性调度离线统计和离线推荐服务，并将推荐结果写入Mysql数据库。后端负责从Mysql数据库中获取推荐结果并反馈至用户，其中离线统计服务包括热门电影统计与评分最高电影统计，离线推荐服务则包括基于用户的个性化推荐与基于内容的相似度推荐两部分。由于服务器性能与算力有限，本推荐系统没有实现实时推荐部分，而是采用了后端代替kafka模拟日志收集的模式，将用户行为记录存入Hive仓库，便于下一次离线计算推荐时使用，从而达到根据用户行为不断调整推荐模型的目标。

## 主要功能



# 开发工具及环境

## 数据爬取

* 操作系统：Deepin Linux 15.11
* 开发语言：Python
* 开发工具：vscode
* 开发环境：
* request
* BeautifulSoup
* pymysql

## 数据存储

* 操作系统：Ubuntu
* 开发语言：SQL
* 开发工具：IntelliJ IDEA
* 开发环境：
* Hadoop 2.7.1
* Mysql 5.1.39
* Hive 3.1.2

## 网站系统

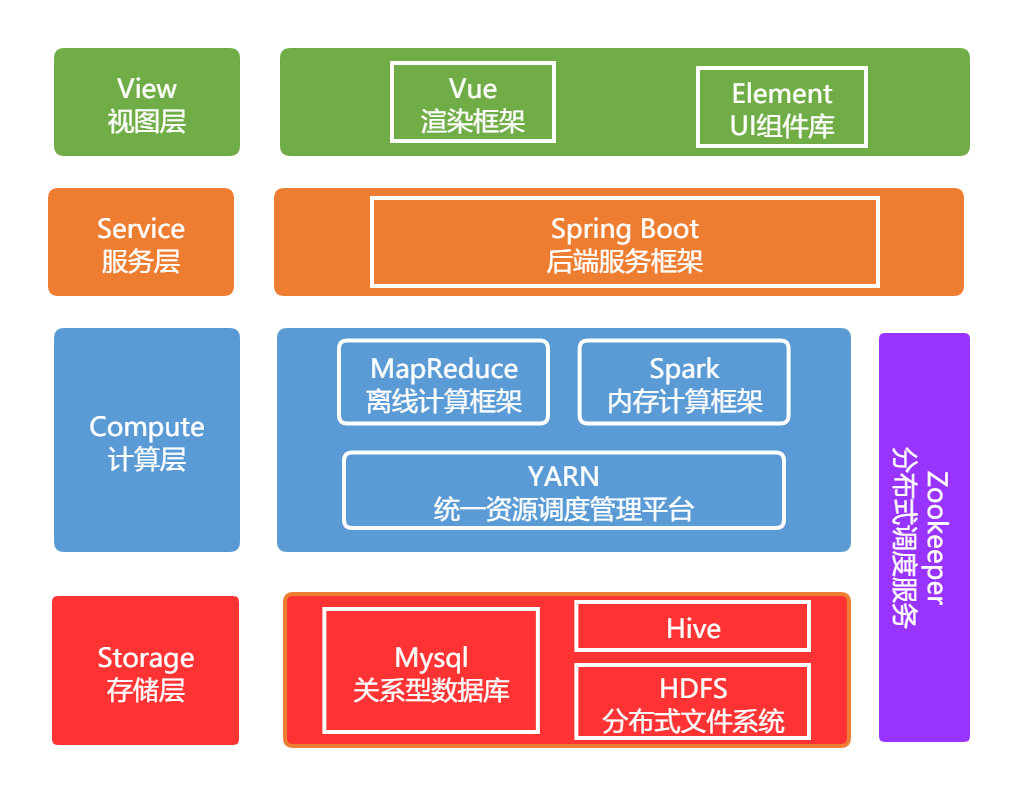
* 操作系统：Windows 10
* 开发语言：
* 前端部分：html + JavaScript
* 后端部分：Java
* 开发工具：IntelliJ IDEA
* 开发环境：
* 前端部分：Vue.js + Element
* 后端部分：Spring Boot

## 推荐系统

* 操作系统：Ubuntu 18.04
* 开发语言：scala 2.11.12
* 开发工具：IntelliJ IDEA
* 开发环境：Spark 2.4.5

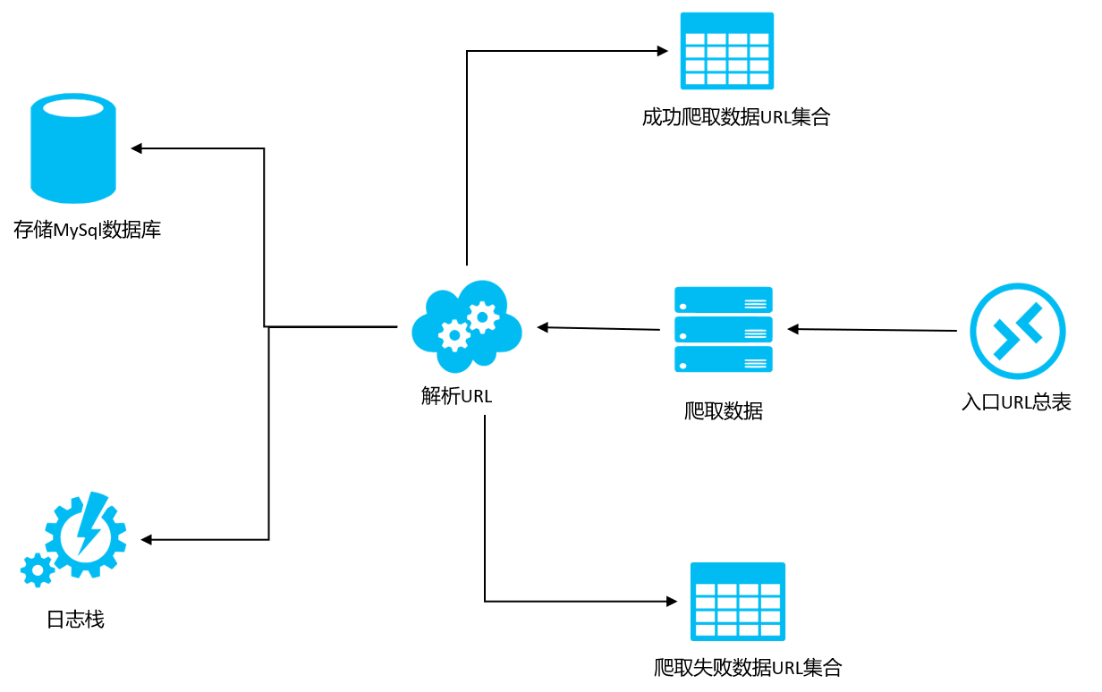
# 系统架构

## 整体系统架构



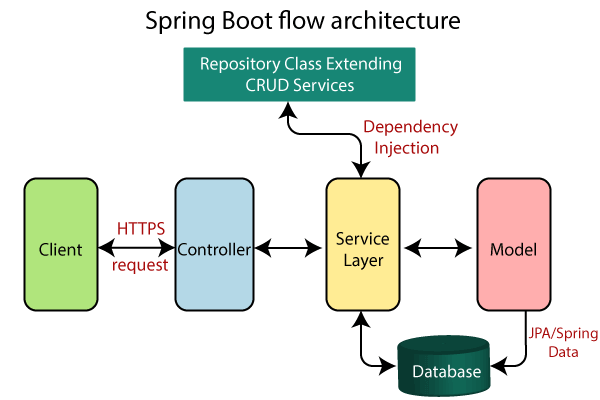
## 爬虫系统架构

通过读取本地URL总表，遍历提交URL，在request请求得到回应后，解析返回的网页源代码。根据BeautifulSoup对源码中的关键词进行分析，成功提取到信息后将URL插入数据提取成功集合，若失败则进行ErrorReport并且插入爬取失败URL集合。爬取成功后，累计一定的数据，一次批量的想MySql数据库提交插入数据申请，并保存日志。



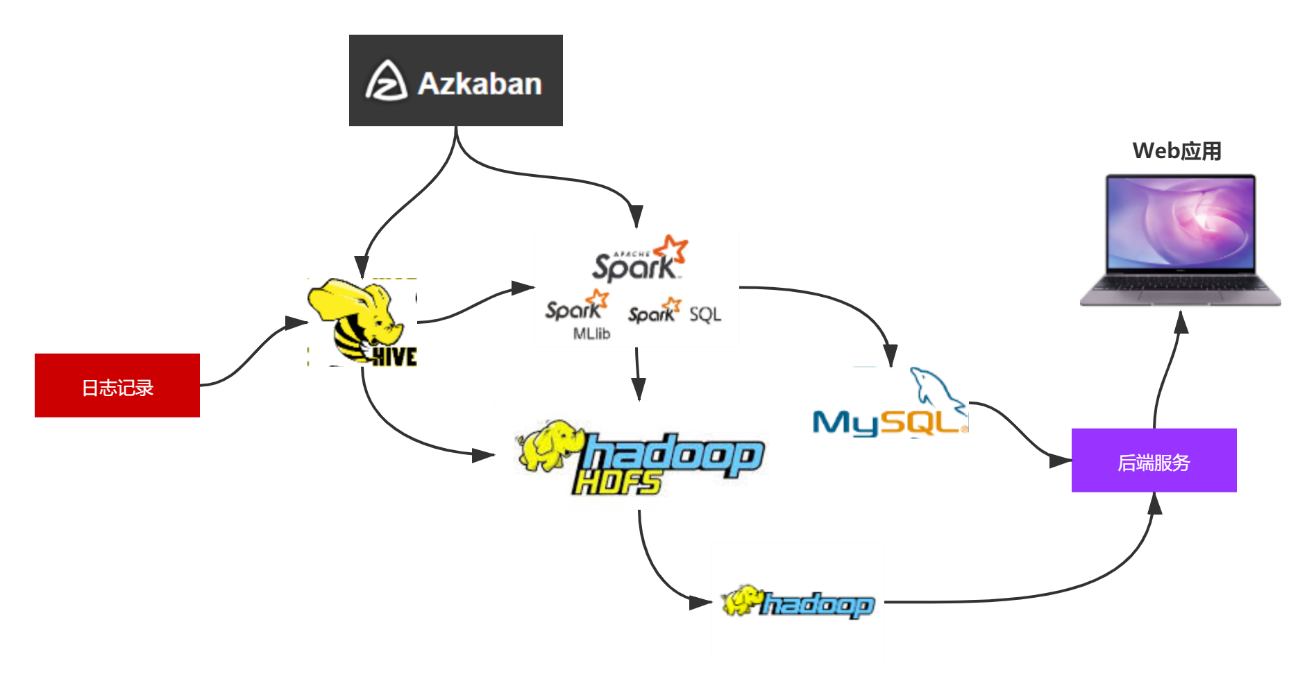
## 网站系统架构

网站系统采用B/S架构，前端部分使用Vue.js框架搭建，完成数据展示；后端部分使用Spring boot敏捷开发框架，采取MVC设计模式，读取前端请求，从MySQL数据库中获取相应数据，经处理后返回至前端显示。



## 推荐系统架构

推荐系统使用Hadoop HDFS作为底层存储支持，Hive作为数据仓库工具，通过Azkaban软件周期性调度Spark脚本运行离线推荐计算，并将计算结果存储至Mysql数据库中，后端服务则从Mysql数据库中查询推荐数据并推荐至Web端查看。



# 具体实现

## 爬虫系统

### 网页信息获取

爬虫爬取一个页面需要首先通过request请求访问网页以获取网页源代码。爬虫的工作原理就是模拟人为的访问网页以获取网页正常反馈的内容，通过对网站反馈的内容进行信息筛选已获得所需要的网页信息。

**def** get\_html\_text(self, url):  
 *# 获取网页text* r = requests.get(url, timeout=30)  
 r.raise\_for\_status()  
 r.encoding = r.apparent\_encoding  
 self.html = r.text  
 self.soup = BeautifulSoup(self.html, **'lxml'**)

### 抓取需求信息

通过request返回的是网页的初始信息，即网页源代码，里面掺杂这许许多多的我们所不需要的信息，因此我们需要通过特定工具对信息进行分析、处理，得到所需要的特定信息。

此处爬虫采用BeautifulSoup进行信息处理，BeautifulSoup是一个易于上手的爬虫分析库，在处理爬取信息上有着独特的工具。

因此定义一个爬虫类，在该类中进行爬虫功能的实现。

**class** Reptile:  
 **def** \_\_init\_\_(self):  
 super(Reptile, self).\_\_init\_\_()  
 self.html = **''** self.soup = BeautifulSoup(self.html, **'lxml'**)  
 self.rating\_list = {}  
 self.param = []  
 idx = 0

#### 获取电影名字、标签、发行时间以及国家

通过电影名字等对应信息所对应的网页标签以及class对应名字，定位到相应位置，获取该位置的信息值，将该信息返回。

**def** get\_movie\_name\_tags\_release\_nation(self):  
  
 soup = self.soup  
 messages = soup.find\_all(**'div'**, class\_=**'title ott\_false'**)  
 messages = BeautifulSoup(str(messages[0]), **'lxml'**)  
 time = messages.find\_all(**'span'**, class\_=**'release'**)  
 time\_string = time[0].string  
 time\_string = time\_string.strip(**'\n'**)  
 time\_string = time\_string.strip()  
 messages = messages.find\_all(**'a'**)  
 messages\_list = []  
 **for** message **in** messages:  
 messages\_list.append(message.string)  
  
 **return** {  
 **'movie\_name'**: messages\_list[0],  
 **'movie\_tags'**: messages\_list[1:],  
 **'movie\_release'**: time\_string.split(**' '**)[0],  
 **'movie\_nation'**: time\_string.split(**' '**)[1]  
 }

#### 获取电影简介

电影简介中，简介内容位于标签div以及class=’overview’中，通过对该信息的甄别定位，可以获取到标签值。

**def** get\_movie\_overview(self):  
 soup = self.soup  
 messages = soup.find\_all(**'div'**, class\_=**'overview'**)  
 messages = BeautifulSoup(str(messages[0]), **'lxml'**)  
 message = messages.find(**'p'**).string  
 **return** {  
 **'movie\_overview'**: message  
 }

#### 获取电影海报

电影海报中，存在着点击放大，因此海报分为简略图和大图两种情况，需要将这两种信息同时爬取下来。因为大图是隐藏在标签信息中的，不能直接通过获取url得到，需要通过字符串处理，获取大图url，通过标签获取url可以直接获取简略图。

**def** get\_movie\_poster(self):  
 soup = self.soup  
 messages = soup.find\_all(**'div'**, class\_=**'image\_content'**)  
 *# print(messages[0])* messages = BeautifulSoup(str(messages[0]), **'lxml'**)  
 picture = messages.img.attrs[**'data-src'**]  
 images = messages.img.attrs[**'data-srcset'**]  
 images = images.split(**','**)  
 image = images[len(images) - 1]  
 image = image.strip()  
 image = image.split(**' '**)[0]  
 *# print(image)* **return** {  
 **'movie\_picture'**: picture,  
 **'movie\_poster'**: image}

#### 获取电影制片人等信息

电影相关人员信息有多种，并且不同电影人员种类设置并不尽相同，因此对于不同的电影人员种类要根据网页上的相关信息进行命名，生成对应的信息。

**def** get\_movie\_people(self):  
 soup = self.soup  
 messages = soup.find\_all(**'li'**, class\_=**'profile'**)  
 people = {}  
 **for** message **in** messages:  
 message = BeautifulSoup(str(message), **'lxml'**)  
 *# people[message.find(  
 # 'p', class\_='character').string] = message.a.string* character = message.find(**'p'**, class\_=**'character'**).string  
 **if** len(character.split(**','**)) > 1:  
 character\_list = character.split(**','**)  
 **for** chara **in** character\_list:  
 chara = chara.strip()  
 **if** chara **in** people.keys():  
 people[chara].append(message.a.string)  
 **else**:  
 people[chara] = [message.a.string]  
 **else**:  
 **if** character **in** people.keys():  
 people[character].append(message.a.string)  
 **else**:  
 people[character] = [message.a.string]  
  
 *# print(people)* **return** people

#### 获取电影主演

每个电影网页都有各自对应的主演人员，且数量不尽相同，因此需要动态的进行插入主演数目，生成最后信息。

**def** get\_movie\_actors(self):  
 soup = self.soup  
 messages = soup.find\_all(**'li'**, class\_=**'card'**)  
 actors = []  
 **for** message **in** messages:  
 actors.append(message.find(**'p'**).string)  
  
 **return** {**'actors'**: actors}

#### 获取电影评分以及评分人数

电影评分信息在本地rating.csv文件中，但由于文件中是数据库转录的每个人的评分情况，因此需要对文件进行遍历操作以生成针对每个电影的总评情况以及评价人数。

**with** open(**'bigData/ratings.csv'**, **'r'**) **as** lines:  
 lines = lines.readlines()  
 lines = lines[1:]  
 *# print(len(lines))* **for** line **in** lines:  
 idx += 1  
 **if** idx % 100000 == 0:  
 print(**'idx:'**, idx, **' '**, line)  
 line\_split = line.split(**','**)  
 **if** len(line\_split) < 4:  
 print(**'error!'**, line)  
 **continue** movie\_id = line\_split[1]  
 rating = float(line\_split[2])  
 *# print(rating)* **if** movie\_id **in** self.rating\_list.keys():  
 *# print(self.rating\_list[movie\_id])* cur\_rating = self.rating\_list[movie\_id]  
 *# print('front:', cur\_rating)* cur\_rating[**'rating'**] = (  
 cur\_rating[**'rating'**] \* cur\_rating[**'num'**] + rating) / (cur\_rating[**'num'**] + 1)  
 cur\_rating[**'num'**] += 1  
 *# print('after:', cur\_rating)* self.rating\_list[movie\_id] = cur\_rating  
 *# break* **else**:  
 temp\_dict = dict()  
 temp\_dict[**'rating'**] = rating  
 temp\_dict[**'num'**] = 1  
 self.rating\_list[movie\_id] = temp\_dict

### 批量存入数据库

由于采用的是通过联网将数据插入服务器上的MySql数据库中，数据库为学生服务器，综合带宽原因以及配置原因，多次访问数据库将会导致时间的大大增加，因此需要进行批量处理，同时进行一批数据的插入工作，可以大大减小数据插入导致的损耗时间。

**def** sql\_executemany(self, db, cursor):  
 **if** len(self.param) == 0:  
 **return** sql\_formate = **'insert into movie\_info values (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)'  
 try**:  
 cursor.executemany(sql\_formate, self.param)  
 db.commit()  
 self.param = []  
 print(**'insert successfully!'**)  
 **except** Exception **as** e:  
 print(e)  
 db.rollback()

### 爬虫工作流程图



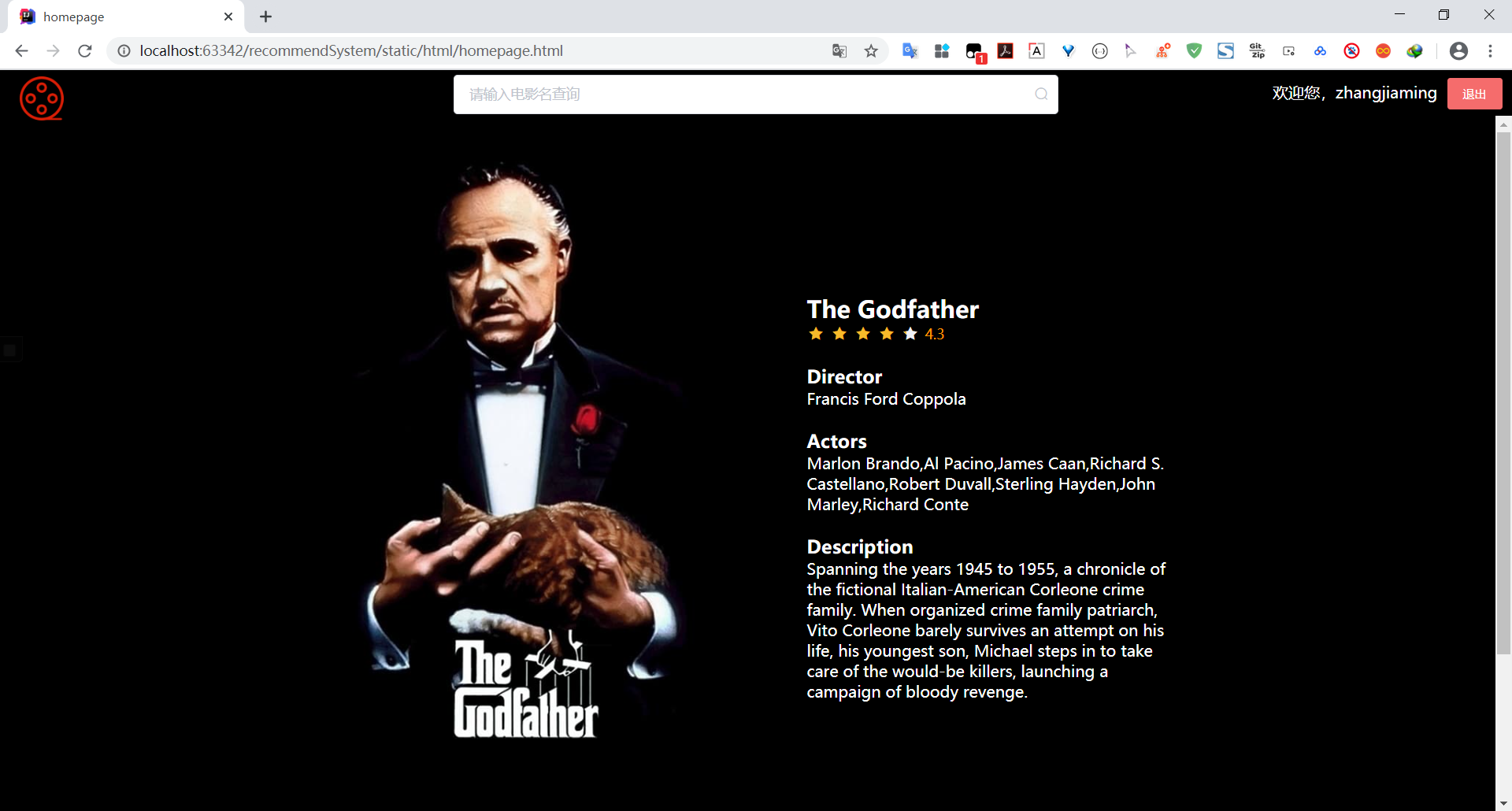
**def** auto(self, idx, movie\_id, url\_id, db, cursor):  
 self.get\_html\_text(  
 **'https://www.themoviedb.org/movie/{}'**.format(url\_id))  
 result = {**'movie\_id'**: movie\_id, **'tmdb\_id'**: url\_id}  
  
 sql = **"insert into movie\_info values('"**+movie\_id+**"','"** *# 包括 names tags，nation，release* name\_tags\_dict = self.get\_movie\_name\_tags\_release\_nation()  
 sql += name\_tags\_dict[**'movie\_name'**]+**"','"** year\_split = name\_tags\_dict[**'movie\_release'**].split(**'/'**)  
 year = year\_split[2]+**'-'**+year\_split[0]+**'-'**+year\_split[1]  
 sql += year + **"','"** + name\_tags\_dict[**'movie\_nation'**] + **"','"** *# 包括Director，Screenplay，Novel* people\_dict = self.get\_movie\_people()  
 director = **''** director\_list = []  
 **if 'Director' in** people\_dict.keys():  
 director\_list = people\_dict[**'Director'**]  
 director = director\_list[0]  
 director\_list = director\_list[1:]  
 **for** d **in** director\_list:  
 director += **','** + d  
 sql += director + **"','"** *# 包括leadactors* lead\_actors\_dict = self.get\_movie\_actors()  
 actors\_list = []  
 actors = **''  
 if 'actors' in** lead\_actors\_dict.keys():  
 actors\_list = lead\_actors\_dict[**'actors'**]  
 actors = actors\_list[0]  
 actors\_list = actors\_list[1:]  
 **for** a **in** actors\_list:  
 actors += **','** + a  
 sql += actors + **"','"** *# 调用上面的people\_dict* scream\_list = []  
 scream\_writers = **''  
 if 'Screenplay' in** people\_dict.keys():  
 scream\_list = people\_dict[**'Screenplay'**]  
 scream\_writers = scream\_list[0]  
 scream\_list = scream\_list[1:]  
 **for** scream **in** scream\_list:  
 scream\_writers += **','** + scream  
 sql += scream\_writers + **"','"** *# 包括picture以及poster* images\_picture\_dict = self.get\_movie\_poster()  
 sql += images\_picture\_dict[**'movie\_picture'**] + **"',"** *# 包括rating以及num* rating = 0.0  
 num = 0  
 **if** movie\_id **in** self.rating\_list.keys():  
 rating = self.rating\_list[movie\_id][**'rating'**]  
 num = self.rating\_list[movie\_id][**'num'**]  
 sql += str(rating) + **","** + str(num)+**",'"** *# 获取影片简介* overview\_dict = self.get\_movie\_overview()  
 sql += overview\_dict[**'movie\_overview'**] + **"','"** *# 获取tags* tags\_list = name\_tags\_dict[**'movie\_tags'**]  
 tags = tags\_list[0]  
 tags\_list = tags\_list[1:]  
 **for** tag **in** tags\_list:  
 tags += **','**+tag  
 sql += tags + **"','"** *# 获取大海报* poster = images\_picture\_dict[**'movie\_poster'**]  
 sql += poster + **"')"** self.param.append([movie\_id, name\_tags\_dict[**'movie\_name'**], year, name\_tags\_dict[**'movie\_nation'**], director, actors,  
 scream\_writers, images\_picture\_dict[**'movie\_picture'**], str(rating), str(num), overview\_dict[**'movie\_overview'**], tags, poster])

## 网站系统

### 前端

1. **页面搭建**

整体页面搭建使用Vue.js作为页面渲染框架，实现数据与表现分离；同时使用element UI作为前端组件库，实现响应式的页面布局并提供成熟页面组件。网站兼容Chrome、Firefox、以及IE9+等现代浏览器。



1. **数据通信**

数据通信模块使用AJAX技术，可以在不更新整个页面的前提下维护数据，使得Web应用程序更为迅捷地响应用户动作。

通信接口设计如下：

|  |  |
| --- | --- |
| 数据项 | 描述 |
| 描述 | 用户注册 |
| URL | /api/user/register |
| method | POST |
| 请求参数 | {  "userName":"userName",  "password":"password"  } |
| 返回结果 | true：注册成功  false：注册失败 |
| 备注 |  |

|  |  |
| --- | --- |
| 数据项 | 描述 |
| 描述 | 用户登录 |
| URL | /api/user/login |
| method | POST |
| 请求参数 | {  "userName":"userName",  "password":"password"  } |
| 返回结果 | "Success"：登录成功  "Wrong password"：密码错误  "User not exist"：用户不存在 |
| 备注 |  |

|  |  |
| --- | --- |
| 数据项 | 描述 |
| 描述 | 获取在首页展示的电影列表 |
| URL | /api/movie/getHomepageMovies |
| method | POST |
| 请求参数 | null |
| 返回结果 | [  {  "movieId":"movieId",  "movieName":"movieName",  "raitng":"raitng",  "picture":"picture url",  "bigPicture":"bigPicture url",  "director":"director",  "actors":"actors",  "description":"description",  "types":"types"  }  ] |
| 备注 |  |

|  |  |
| --- | --- |
| 数据项 | 描述 |
| 描述 | 获取个性化推荐 |
| URL | /api/movie/getRecommendMovies |
| method | POST |
| 请求参数 | "userName":"userName" |
| 返回结果 | [  {  "movieId":"movieId",  "movieName":"movieName",  "raitng":"raitng",  "picture":"picture url",  "bigPicture":"bigPicture url",  "director":"director",  "actors":"actors",  "description":"description",  "types":"types"  }  ] |
| 备注 |  |

|  |  |
| --- | --- |
| 数据项 | 描述 |
| 描述 | 获取热门电影推荐 |
| URL | /api/movie/getRecentTopMovies |
| method | POST |
| 请求参数 | null |
| 返回结果 | [  {  "movieId":"movieId",  "movieName":"movieName",  "raitng":"raitng",  "picture":"picture url",  "bigPicture":"bigPicture url",  "director":"director",  "actors":"actors",  "description":"description",  "types":"types"  }  ] |
| 备注 |  |

|  |  |
| --- | --- |
| 数据项 | 描述 |
| 描述 | 获取相似电影推荐 |
| URL | /api/movie/getRelatedMovies |
| method | POST |
| 请求参数 | {"movieId":"movieId"} |
| 返回结果 | [  {  "movieId":"movieId",  "movieName":"movieName",  "raitng":"raitng",  "picture":"picture url",  "bigPicture":"bigPicture url",  "director":"director",  "actors":"actors",  "description":"description",  "types":"types"  }  ] |
| 备注 |  |

|  |  |
| --- | --- |
| 数据项 | 描述 |
| 描述 | 获取电影搜索结果 |
| URL | /api/movie/getMoviesByName |
| method | POST |
| 请求参数 | {  "page":{  "searchKey":"searchKey",  "pageIndex":1,  "pageSize":12,  },  "movieName":"movieName",  "types":"type",  "orderBy":"orderBy",  "sortBy":"sortBy"  } |
| 返回结果 | [  {  "movieId":"movieId",  "movieName":"movieName",  "raitng":"raitng",  "picture":"picture url",  "bigPicture":"bigPicture url",  "director":"director",  "actors":"actors",  "description":"description",  "types":"types"  }  ] |
| 备注 |  |

### 后端

后端使用Spring boot框架，根据前端请求从数据库中获取相应信息回传至前端。前后端的数据传输主要以JSON格式，有利于数据的解析和使用。

基于安全的考虑，AJAX只能访问本地的资源，而不能跨域访问。而系统在运行中需要使用AJAX请求其他端口的请求，这个时候就会出现拒绝访问的情况（跨域问题）。故使用以下代码解决跨域问题。

1. **package** bit.ss.recommendSystem.common.utils;
2. **import** org.springframework.stereotype.Component;
4. **import** javax.servlet.\*;
5. **import** javax.servlet.http.HttpServletRequest;
6. **import** javax.servlet.http.HttpServletResponse;
7. **import** java.io.IOException;
9. @Component
10. **public** **class** CorsFliter **implements** Filter {
11. @Override
12. **public** **void** init(FilterConfig filterConfig) **throws** ServletException {
14. }
16. @Override
17. **public** **void** doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) **throws** IOException, ServletException {
18. HttpServletResponse response = (HttpServletResponse) servletResponse;
19. HttpServletRequest request = (HttpServletRequest)servletRequest;
21. String origin = request.getHeader("Origin");
22. response.setHeader("Access-Control-Allow-Origin","\*");
23. response.setHeader("Access-Control-Allow-Methods", "POST, GET, OPTIONS, DELETE");
24. response.setHeader("Access-Control-Max-Age", "3600");
25. response.setHeader("Access-Control-Allow-Headers", "Content-Type,Access-Token");
26. response.setHeader("Access-Control-Allow-Credentials", "true");
27. String method = request.getMethod();
28. **if**(method.equalsIgnoreCase("OPTIONS")){
29. servletResponse.getOutputStream().write("Success".getBytes("utf-8"));
30. }**else**{
31. filterChain.doFilter(servletRequest, servletResponse);
32. }
33. }
35. @Override
36. **public** **void** destroy() {
38. }
39. }

## 推荐系统

### 数据集加载

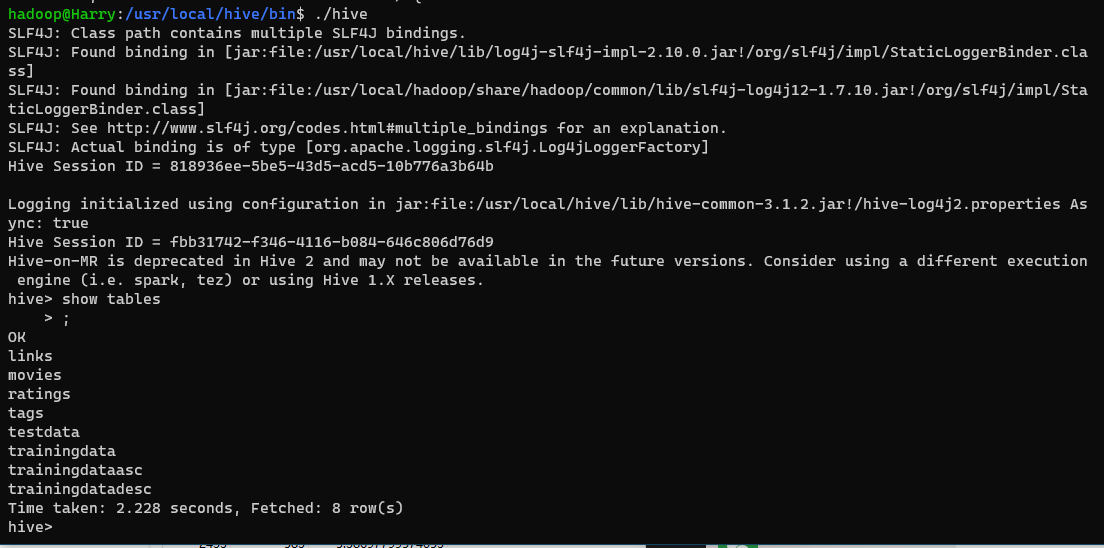
1. **建立数据仓库**

本系统数据集基于Movielens数据集，该数据集包含多个用户对多部电影的评级数据，包括对电影的评分，电影元数据（风格类型和年代）以及关于用户的人口统计学数据（年龄、邮编、性别和职业等）。

本系统使用HDFS作为底层存储支持，使用Hive作为本系统的数据仓库工具，在数据仓库中建立了如下表来进行离线存储服务：

|  |  |  |
| --- | --- | --- |
| 表名 | 字段列表 | 描述 |
| links | int movieId 每部电影的id  int imdbId 对应imdb网站的id  int tmdbId 对应tmdb网站的id | 通过网站id在对应网站上找到对应的电影链接 |
| movies | int movieId 每部电影的id  String title 电影的标题  String genres 电影的类别 | 存储每一部电影的id和标题，以及该电影的类别 |
| ratings | int userId 评分用户的id  int moiveId 每部电影的id  double rating 用户对该电影评分  int timestamp 时间戳 | 存储每一个用户对于每一部电影的评分 |
| tags | int userId 评分用户的id  int moiveId 每部电影的id  string tag 用户对电影的标签化评价  int timestamp 时间戳 | 存储每一个用户对于每一个电影的分类 |
| trainingdata | 同 ratings表 | 按8：2对ratings表进行拆分用以模型训练的训练集 |
| testdata | 同 ratings表 | 按8：2对ratings表进行拆分用以模型测试的测试集 |
| traindataasc | 同 ratings表 | 生成训练集的中间表 |
| traindatadesc | 同 ratings表 | 生成测试集的中间表 |

使用hive命令行查看数据仓库中表结构，结果如下：



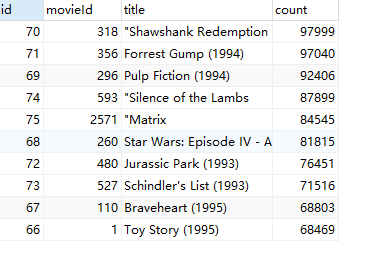
1. **拆分数据集**

由于ALS模型训练需要对数据集进行拆分，本系统中对ratings表按8:2比例进行训练集与测试集的拆分，通过对记录按时间戳进行排序，取升序前80%作为训练集，存入trainData表中，取降序前20%作为测试集，存入testData中。

### 离线统计服务

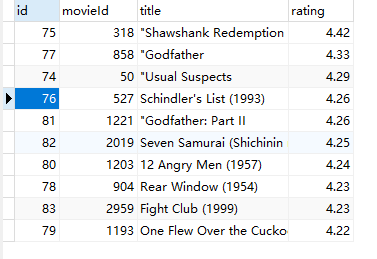
1. **热度最高Top10**

热门电影TOP10推荐基于Hive中的ratings表，使用spark统计一周内电影被打分次数最高的电影，并将结果存入mysql的top\_count\_movies表中，由于本系统未上线运行，目前统计的为全部时间内打分次数最多的电影，统计结果如下：



1. **评分最高Top10**

评分最高电影榜单同样基于Hive中的ratings表，使用spark统计电影平均评分最高的电影，这里进行了特殊化处理，只筛选评分次数大于1000的电影进行推荐，避免了某些小众电影的错误引导，并将结果存入mysql的top\_rating\_movies表中，运行后统计结果如下：



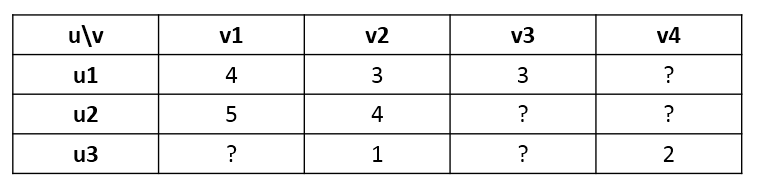
### 离线推荐服务

#### 基于用户的个性化推荐

在基于用户的个性化推荐模块中，在对用户历史打分内容进行学习后，系统可推荐出10个用户最喜欢的电影，本系统中采用了基于ALS的协同过滤算法，接下来将从算法原理、模型训练、模型推荐三个方面进行阐述。

1. **算法原理**

用户对物品的打分行为可以用一个矩阵（R）来表示：



矩阵中的打分值 表示用户对物品的打分，其中?表示用户没有打分，通过协同过滤算法预测该打分值，筛选出其中分值最高的物品对用户进行推荐。

根据协同过滤的思想，R矩阵的行向量对应每个用户U，列向量对应每个物品V。ALS的核心思想是：将用户和物品都投射到k维空间，也就是说假设有k个隐向量特征，至于这个k个隐向量是什么无需关心（可能是标签，年龄，性别等），将每个用户和每个物品都用k维的向量来表示，把他们的内积近似为打分值，这样便可以得到近似的评分。

其中：

* R为打分矩阵（m\*n，m表示用户个数，n表示物品个数）
* U表示用户对隐含特征的偏好矩阵（m\*k）
* V表示物品对隐含特征的归属矩阵（n\*K）

上述模型的参数就是U和V，求得U和V之后，就可以近似的得到用户对未评分物品的评分。

1. **模型训练**

本系统ALS算法实现基于Spark.Mllib,其中已经封装好了ALS算法，只需传入参数即可进行训练。训练中使用的数据为对ratings表按8：2拆分的训练集与测试集，通过不断调整正则化参数和迭代次数，使用均方根误差对模型进行打分，寻找模型最优参数，其中均方根误差公式如下：

由于算力有限，本次调参优化对正则化参数和迭代次数只进行了粗准调整，而将模型的隐藏因子数固定为50，其中正则化参数范围为(0.001,0.005,0.01)，迭代次数范围为[10,20],经过拟合训练，得出如下结果：



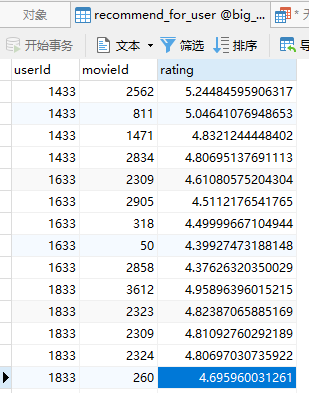
可见当正则化参数lambda = 0.001,迭代次数 = 20 时，模型表现最好，RMSE均方根误差最小，其值为0.532，保存该最优训练模型并进行内容推荐。

1. **模型推荐**

通过遍历用户id，调用ALS模型中的model.recommendProducts为每一位用户进行个性化推荐，并将推荐结果写入mysql中的recommend\_for\_user表中：



最终recommend\_for\_user保存了每位用户可能的评分最高的电影id与评分，通过多表联查即可对用户进行个性化推荐，后端通过查询Mysql数据库即可返回给前端离线推荐结果，表中部分内容如下：



#### 基于内容的相似度推荐

在基于内容的相似度推荐中，用户浏览某一电影之后，可根据浏览电影的内容进行相似推荐，本系统中采用了基于同现相似度算法的推荐算法，通过生成电影的相似度列表，用来提供查询相关电影服务。

1. **算法原理**

物品i和物品j的同相似度公式定义：

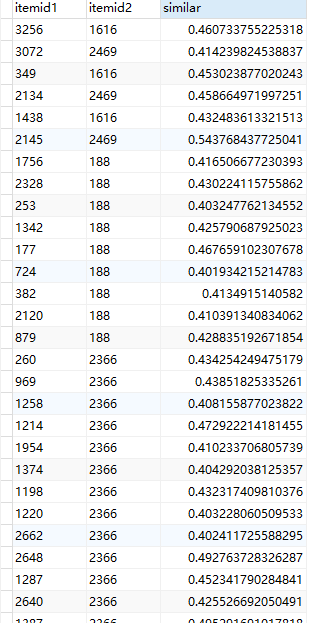
其中分母|N(i)|是喜欢物品i的用户数，而分子|N(i)∩N(j)|是同时喜欢物品i和物品j的用户数据。因此，上述公式可用理解为喜欢物品i的用户有多少比例的用户也喜欢j (和关联规则类似)。

但上述的公式存在一个问题，如果物品j是热门物品，有很多人都喜欢，则会导致Wij很大，接近于1。因此会造成任何物品都和热门物品交有很大的相似度。为此我们用如下公式进行修正：

这个格式惩罚了物品j的权重，因此减轻了热门物品和很多物品相似的可能性。

1. **推荐结果**

使用spark实现该公式，使用ratings仓库中的数据，当用户对某一电影进行打分，可以认为该用户喜欢该电影，并套入上方公式，计算每一对电影的相似度，将相似度>0.4的电影存入mysql的recommend\_for\_movie表中：

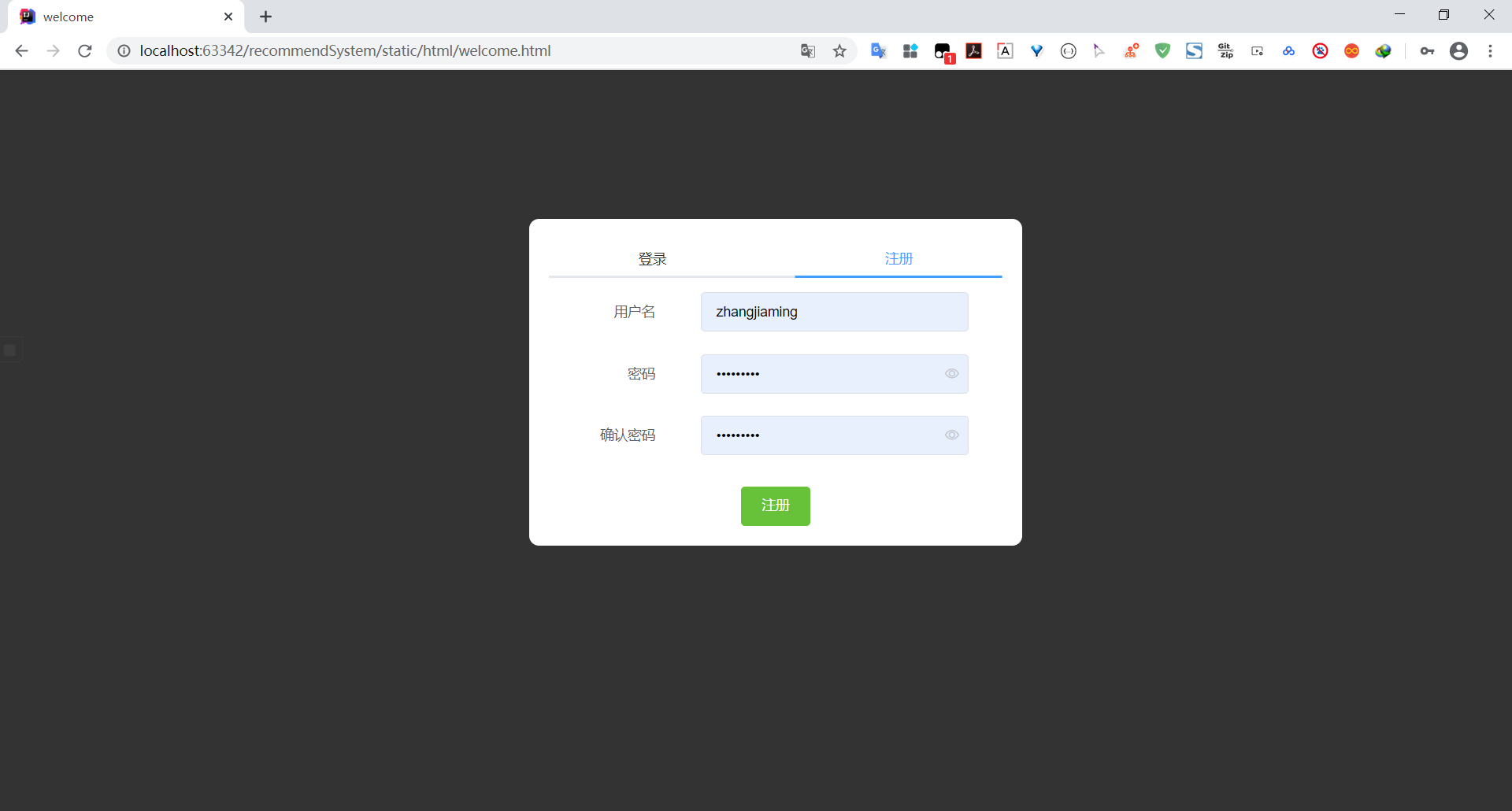


当用户浏览某一电影之后，系统会自动查找与改电影相似度最高的前5部电影并推荐给用户。

# 系统演示结果

## 登陆注册

通过用户名、密码、二次确认密码的方式进行注册：

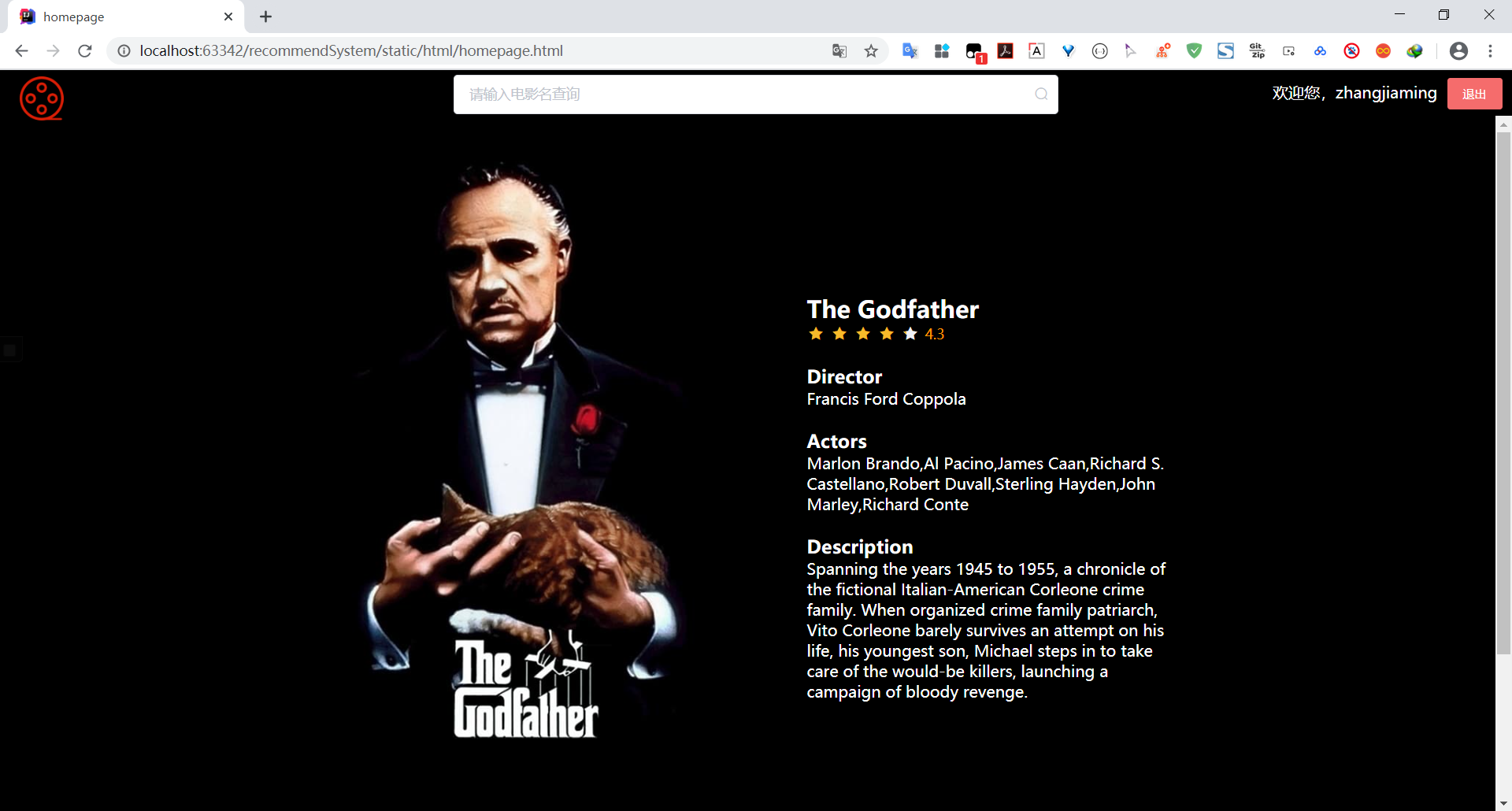


通过注册后的用户名、密码进行登录：



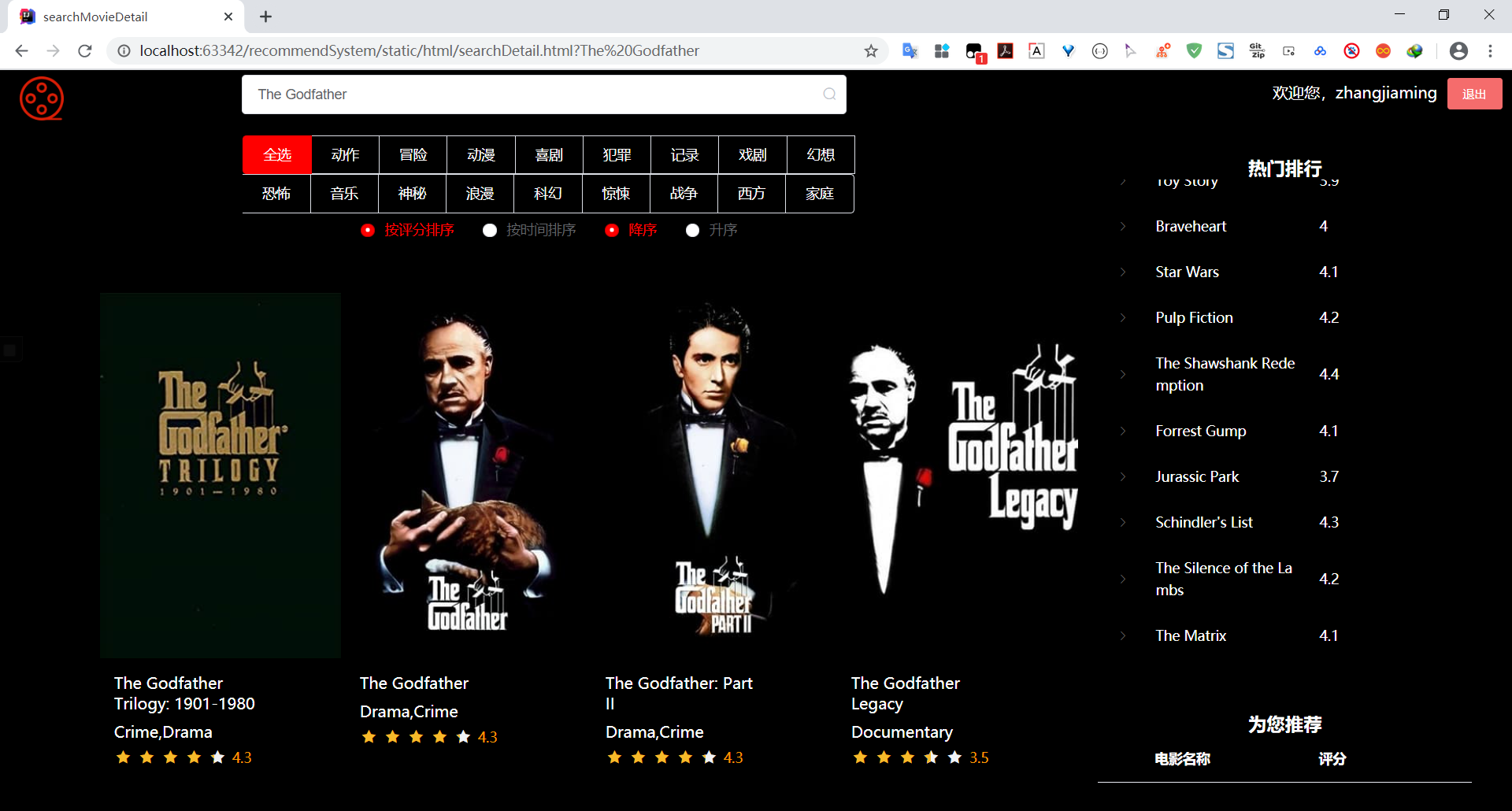
## 首页-轮播图

首页展示热度Top5的电影推荐，包括电影海报和电影详情部分，同时可点击电影海报进入该电影的搜索界面：



## 搜索界面

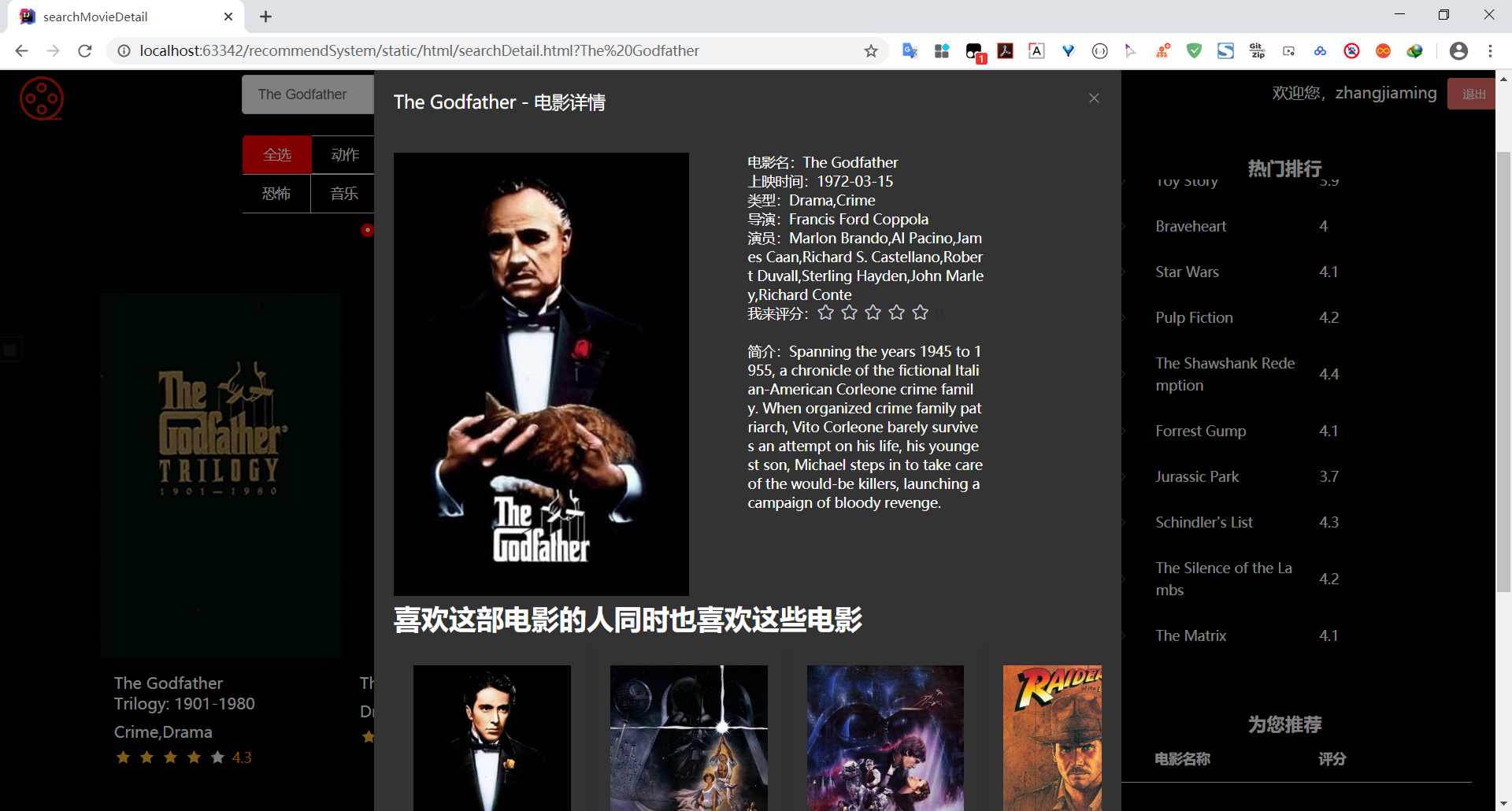
用户在搜索栏进行关键字搜索之后（或点击首页轮播图）可跳转到搜索界面，包括左侧的搜索结果和右侧的电影推荐，其中左侧的搜索结果可对电影进行筛选以及按照评分或上映时间对搜索结果进行升、降序的排列，右侧的推荐结果包括热门推荐和个性化推荐。



### 电影详情

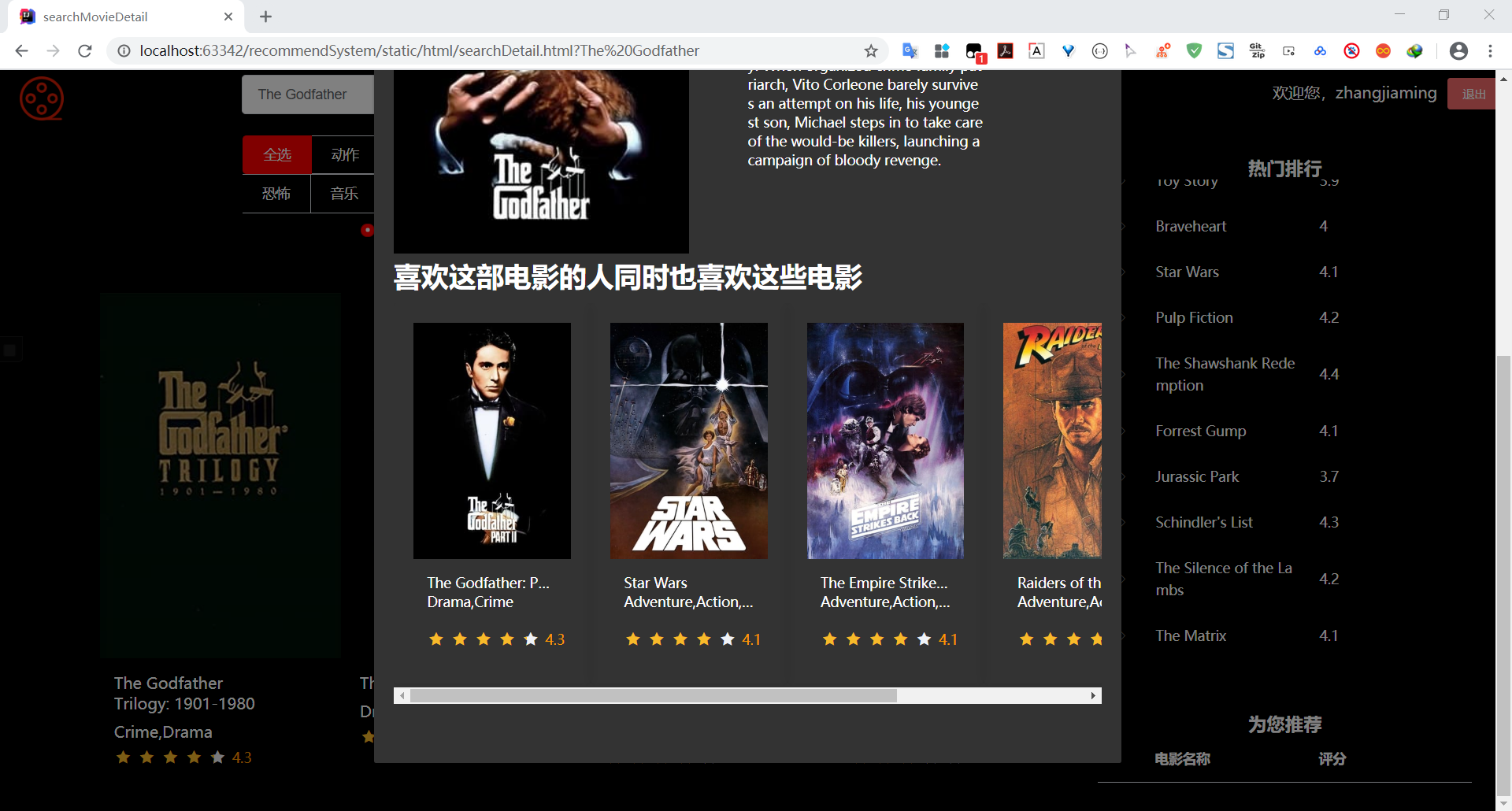
1. **电影详情部分**

电影详情部分包括海报及文字，同时用户可对该电影进行点击评分：



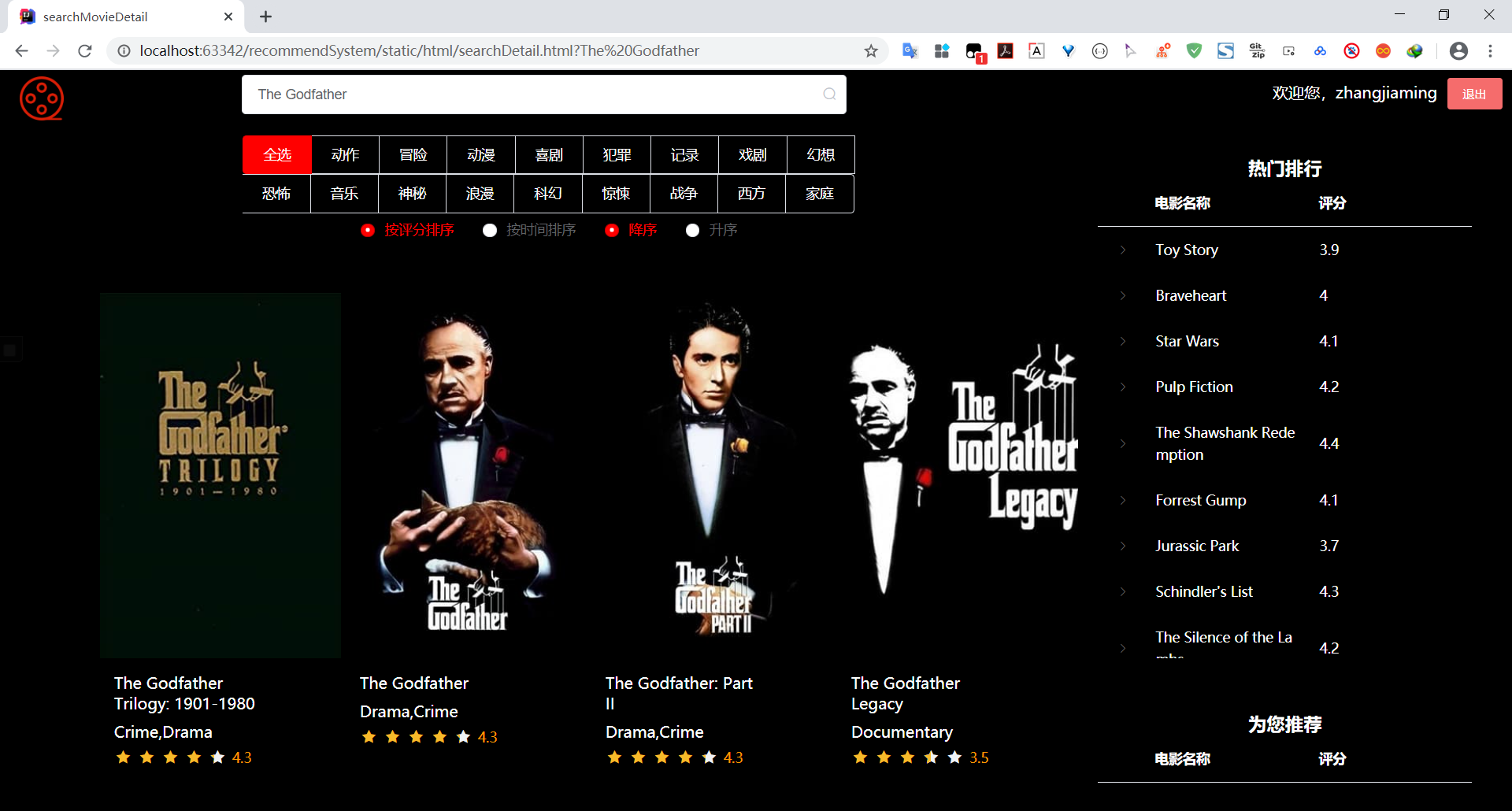
1. **相似电影推荐部分**

下方是相似电影推荐部分：

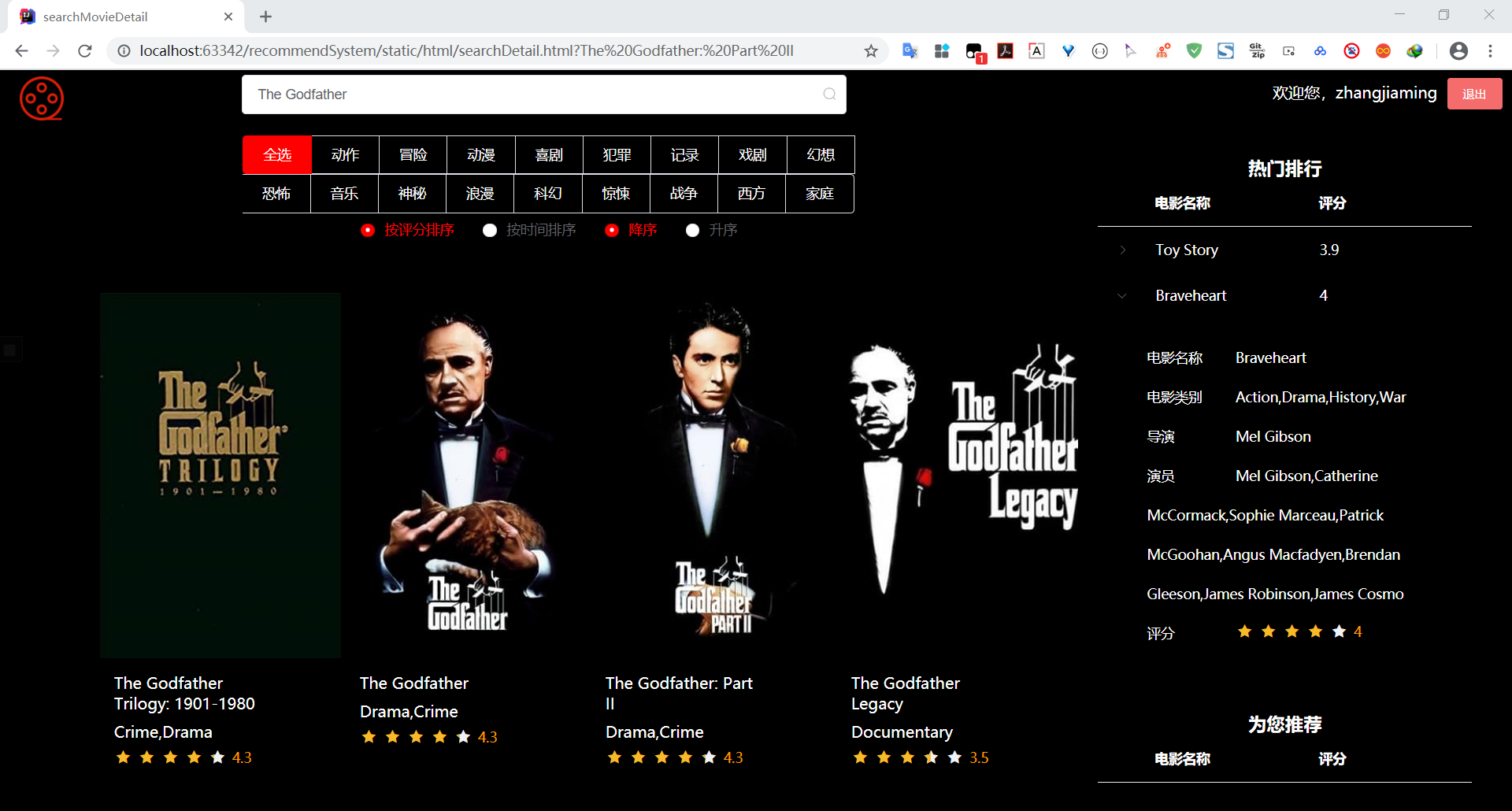


### 热门推荐Top10

右侧上半部分为热度电影推荐：

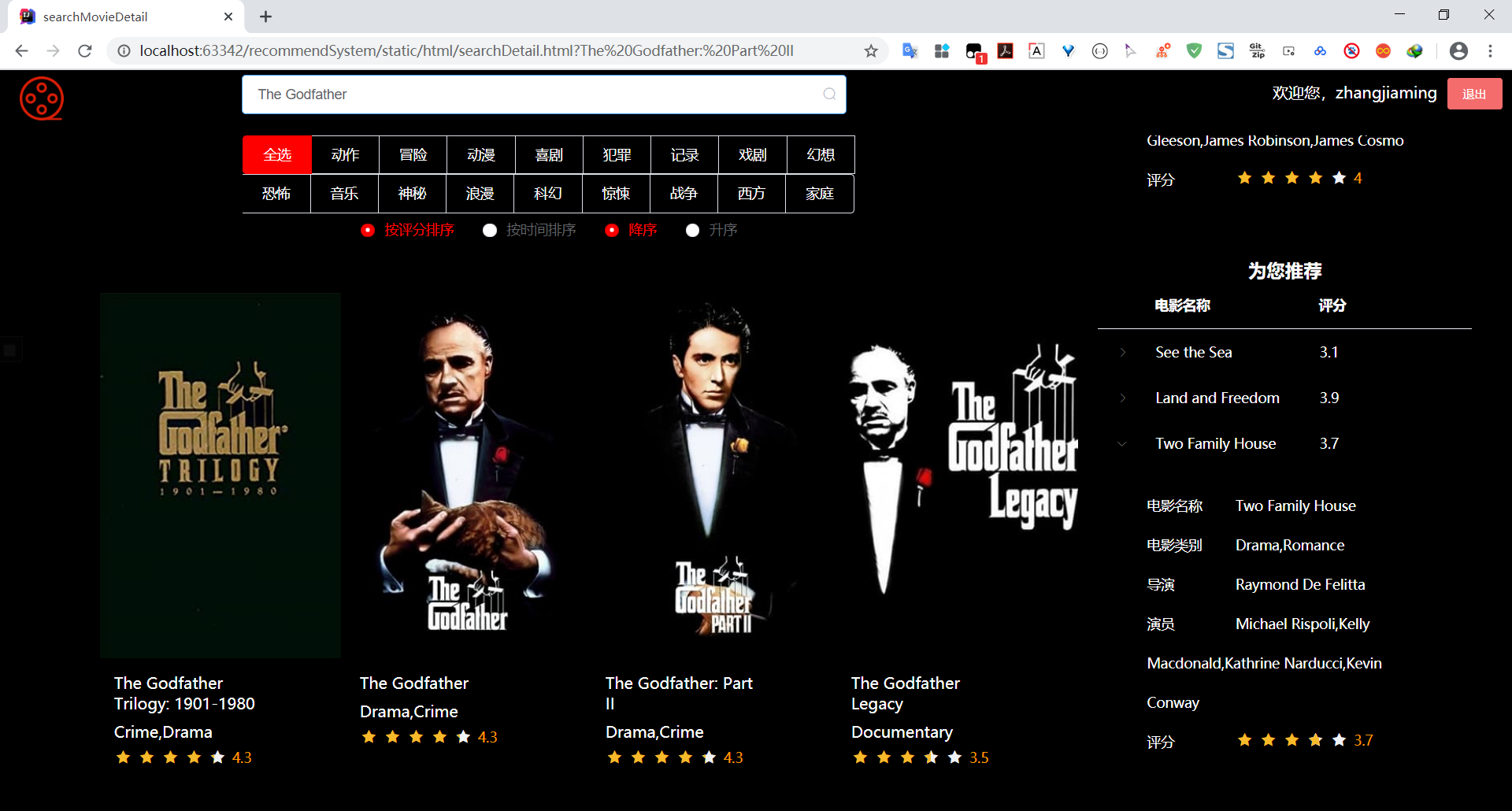


用户可点击展开按钮查看电影详情



### 个性化推荐

右侧的下半部分为个性化推荐，同样可点击展开查看电影详情



# 项目实现重难点分析

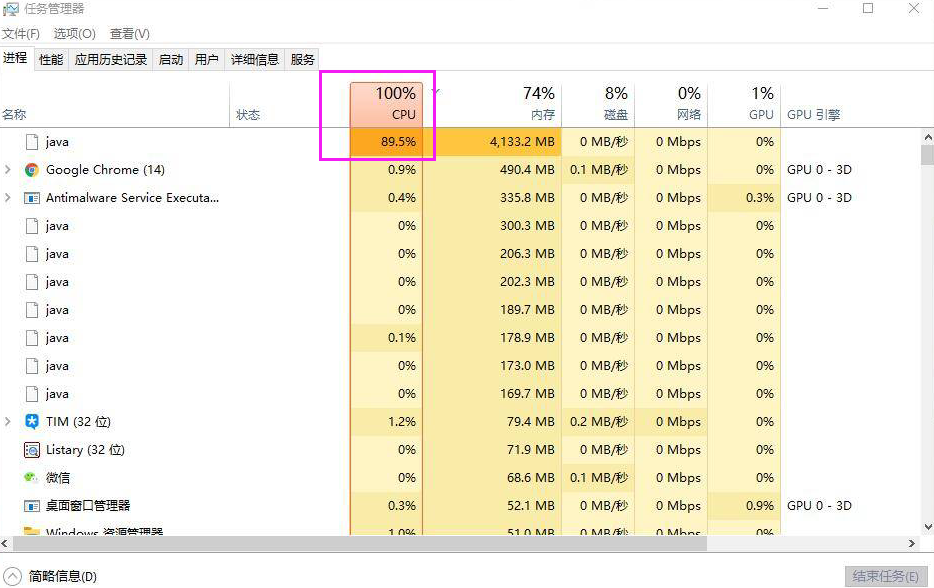
## 环境与硬件

* **环境搭建繁琐**

在本次实验过程中，用到了很多不同的软件、系统、平台和工具（详见： [4.开发工具与环境](#_开发工具及环境)）。其中一部分已经在课程实验中安装使用过（例如：HDFS、MongoDB等），但大部分还是较为陌生，需要我们查阅很多资料学习如何进行配置、安装和使用（例如： Spark、 Hived等）。在这些组件结合使用过程中，各种配置调试产生的报错是项目的难点之一。

* **硬件性能有限**

由于我们只能在自己的笔记本电脑上运行项目，而数据集的规模较大，因此运算耗时很久（例如进行模型训练和相似度计算时，耗时过长），且会出现电脑超负荷运转、宕机等情况，为项目开发带来了不小的问题。下图为运行项目代码时CPU满负荷运行的截图：



## 爬虫系统

* **爬虫错误处理**

爬虫服务在运行过程中会遇到很多异常和错误，例如：数据没有；访问网站不存在；网站未含有需要的信息。因此需要保证系统能经受这些错误后还能正常运行。虽然异常处理本身并不复杂，但单点在于繁琐，且难以考虑全面。

* **数据库中断**

爬虫数据写入数据库时，数据库可能会中断连接，需要设置数据访问中断自动重连机制

* **速度问题**

爬虫主要限制速度为网速，线性操作会导致数据插入缓慢，采用批量插入数据的方式减少用时。

## 推荐算法

* **离线计算使用RDD效率低**

原本离线计算部分使用的是RDD，但后来更换方法改为基于Spark DataFrame的实现。DF本身比RDD具有更多的默认优化，比如Tungsten运用了堆外内存，减少GC开销、序列化比RDD的Java默认序列化更加高效、全代码生成等优势。

* **难以对模型进行最优调参**

由于电脑运算能力有限，训练模型时无法调到最优参数，因此本次调参优化对正则化参数和迭代次数只进行了粗准调整，而将模型的隐藏因子数固定为50，其中正则化参数范围为(0.001,0.005,0.01)，迭代次数范围为[10,20]。

1. MovieLens数据集 <https://grouplens.org/datasets/movielens/> [↑](#footnote-ref-1)