1)前端部分

总体设计

前端部分使用React编写,大致分为浏览、搜索、收藏三个主模块和注册、登陆两个二级模块.

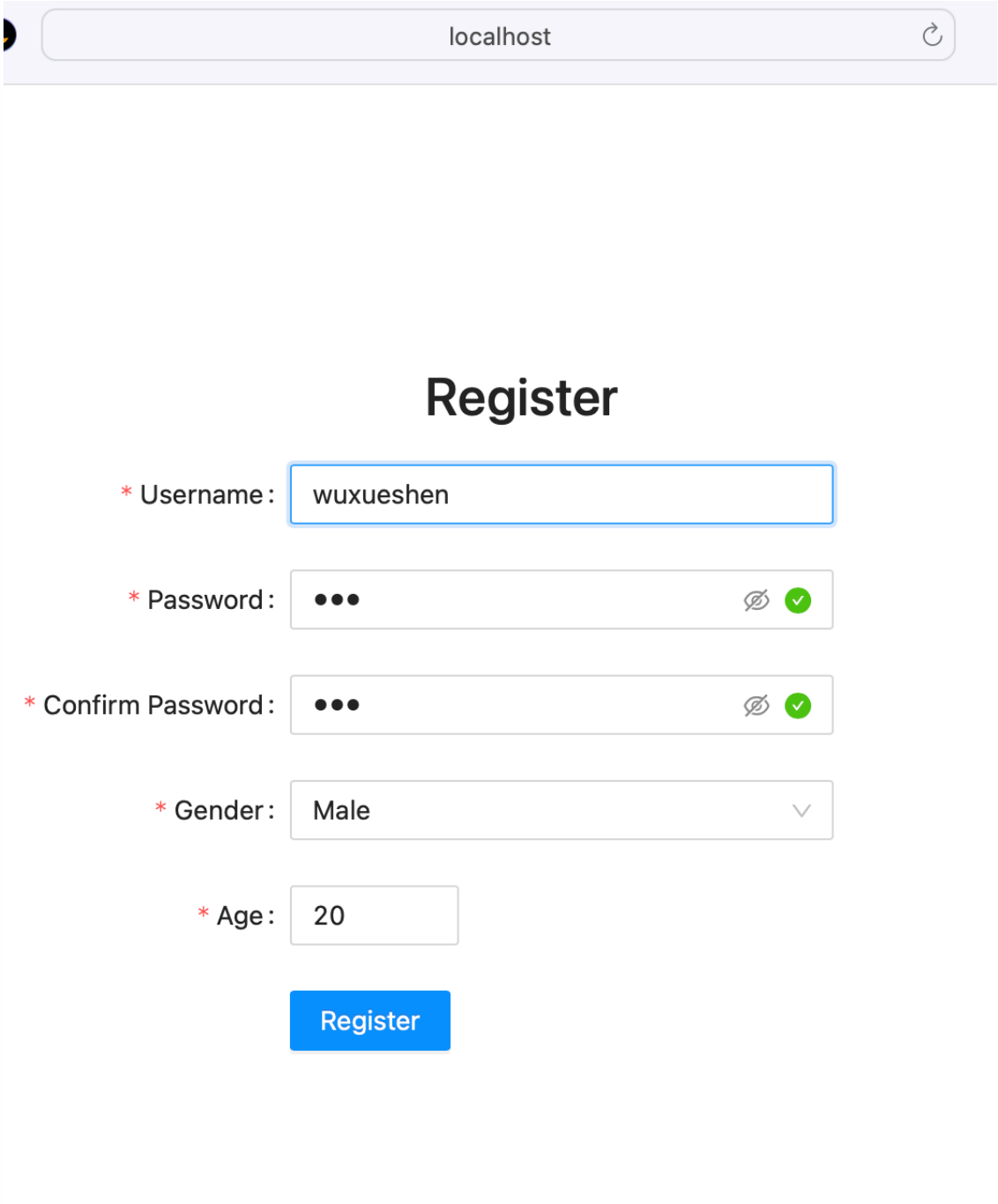
其中页面内容的获取与刷新由ajax实现,不同模块之间的跳转使用react的路由机制.

登陆注册

注册

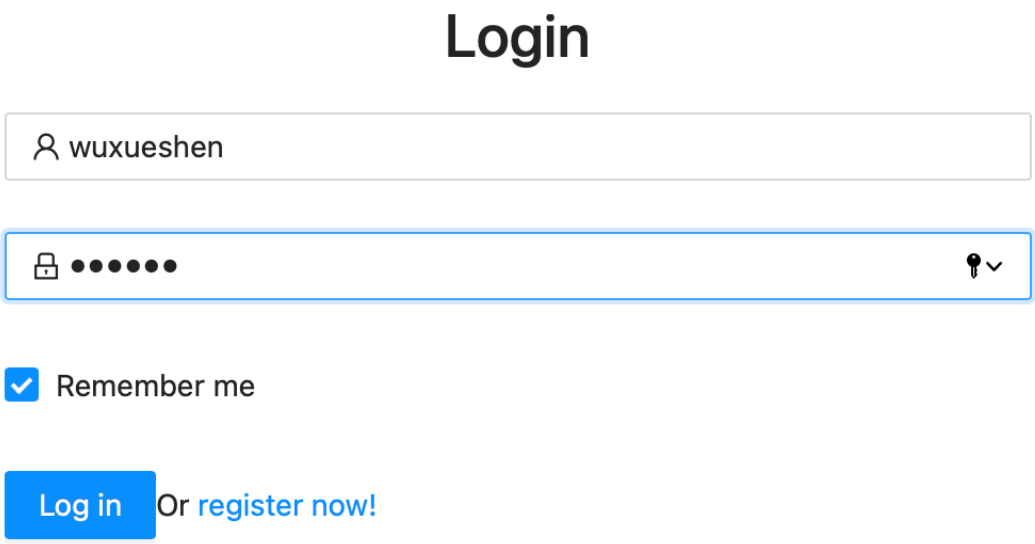
注册模块为一个表单,地址为/Register.在注册时收集除了填写用户名与密码外还需要填写性别与年龄,用户后期的推荐.

Register按钮监听点击事件,在点击事件回调函数中使用axios向服务器端口12000发送注册信息,并根据返回结果进行下一步.如果服务器返回成功则导航至登陆界面,否则提示注册失败.



登陆

登陆页面接收用户名和密码,使用POST方式请求服务器端口12000后根据结果进行判断是否登陆成功.若登陆成功则跳转至浏览模块,否则给出提示信息.



核心功能

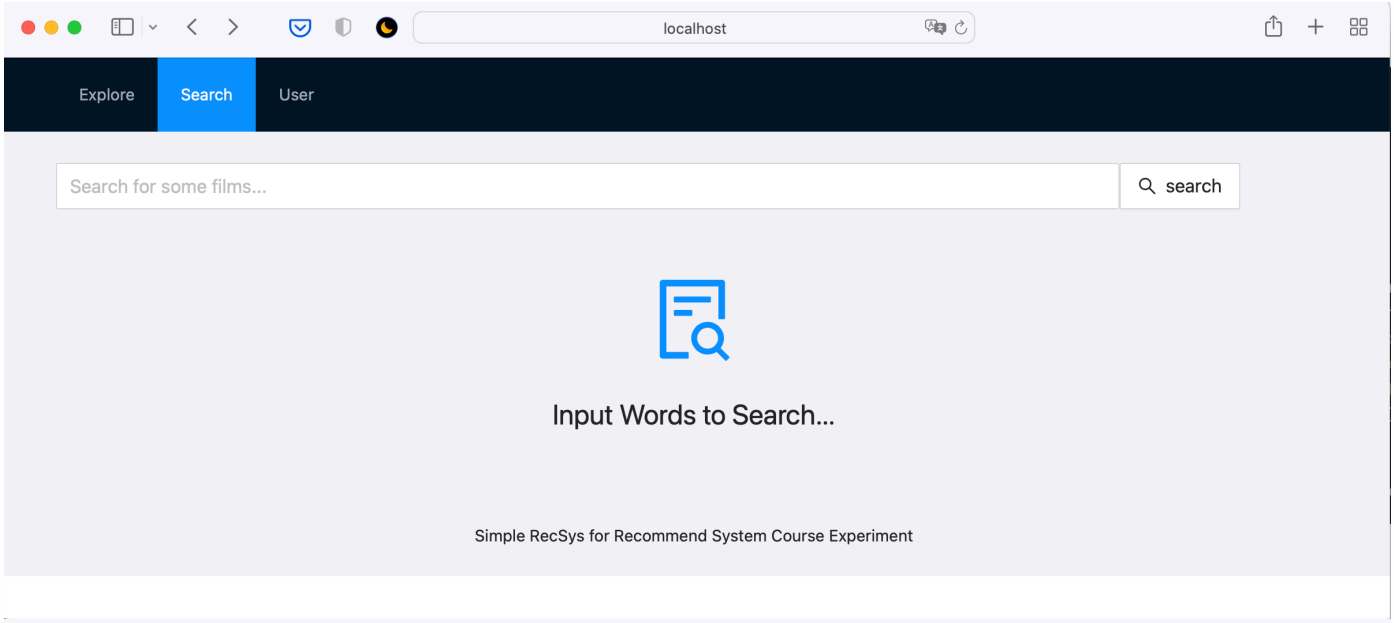
整体组件的关系如下所示

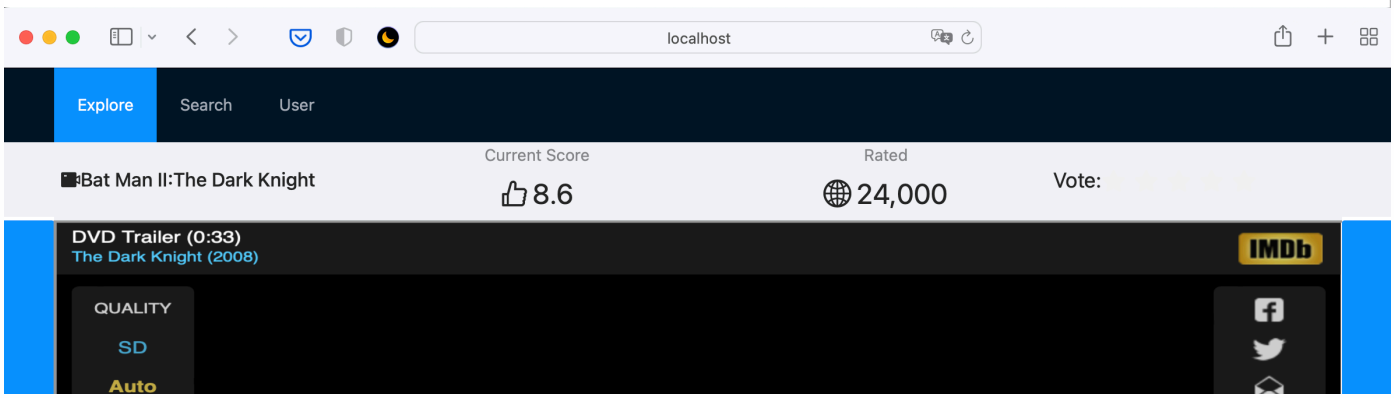
```
class App extends Component
{
    render()
    {
        return (
            <div>
                <BrowserRouter>
                    <Routes>
                        <Route path={"/"} element={<Main/>}>
                            <Route index element={<NewExplore/>}></Route>
                            {/*<Route path={'/Explore'} element={<ExploreWrapper/>}></Route>*/}
                            <Route path={'/Explore'} element={<NewExplore/>}></Route>
                            <Route path={"Search"} element={<Search/>}></Route>
                            <Route path={"User"} element={<User/>}></Route>
                        </Route>
                        <Route path={'/Register'} element={<Register/>}/>
                        <Route path={'/Login'} element={<Login/>}/>
                    </Routes>
```

在程序根节点App中进行路由的注册,导航页为Main组件('路径为/'),Main组件中组册了二级路由,分别为Explore、Search和User,三个组件分别对应浏览、搜索/推荐、收藏夹/登陆/退出.

Main

Main组件的功能是提供一个常驻的NavBar,实现不同功能的切换.





上面的深蓝色条即为Main组件,而Explore、Search和User组件都作为其子组件进行渲染.

Explore

目的

为了设计能够实现Bandit算法,我曾经考虑过使用捎带推荐的方式,即在返回的推荐数据中插入若干条Bandit算法产生的推荐数据,但这么做的CTR概率显然较为低下,因为用户未必会去点击推荐的内容,更未必会点击其中来自Bandit算法推荐的内容,因而导致数据极其稀疏.因此,我打算采用类似刷短视频的方式在此专门使用强化学习进行推荐.

展示信息

Explore组件上方包含电影名称、当前评分、评分人数和投票按钮,下方包含电影介绍、电影标签和收藏按钮,中间为电影的预告片.

电影预告片使用了iframe捕获imdb数据库中的电影预告片,看似较为简单,但问题在于imdb的电影id和预告片id并不是一样的,因此我使用爬虫爬取了ml-ls中提供的9000多条电影的预告片id与剧情介绍.

使用iframe播放预告使得服务器从返回视频流变为返回一个字符串,能够极大减少服其开销,.

行为收集

- CTR

  在每次切换电影后,Explore会设定一个时长为20秒的计时器,在20秒结束后,启动一个回调函数将当前用户的id与电影id发送给服务器,代表该用户对电影表现出了兴趣.

- Vote

  在用户进行评分操作后,会启动一个回调函数向后台发送电影评分信息(用户id、电影id、分数).再次点击会取消评分

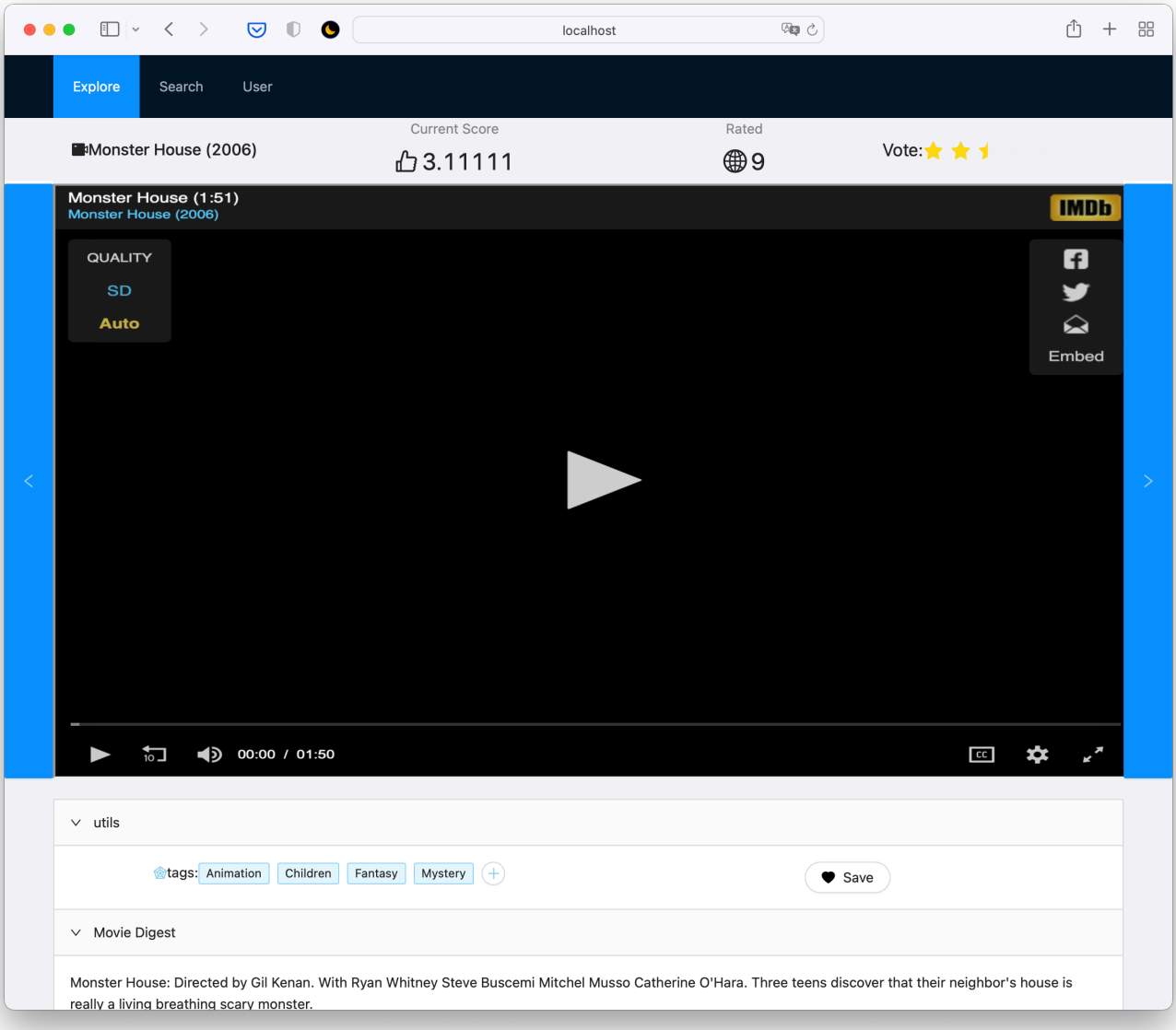
- Save

  用户点击收藏按钮后,向后台发送电影收藏信息(用户id、电影id),再次点击会取消收藏.

切换

在Explore组件中的电影切换通过Ajax请求实现.其中的所有组件均为受控组件,即数据依赖于Explore组件State的改变.

当监听到切换视频按钮的组件时,使用axios向服务器发送请求,请求数据包括电影基本信息、上个电影CTR状态(电影体裁和电影ID用于更新Bandit参数).同时根据缓存的计时器id将其清除,避免出现虚假的CTR反馈.



Search

目的

Search模块提供了搜索和推荐功能,当用户输入搜索信息并提交时,前端接收相应数据并展示.

数据类型

- 搜索结果

  后端搜索引擎根据搜索词返回的结果
- popular推荐

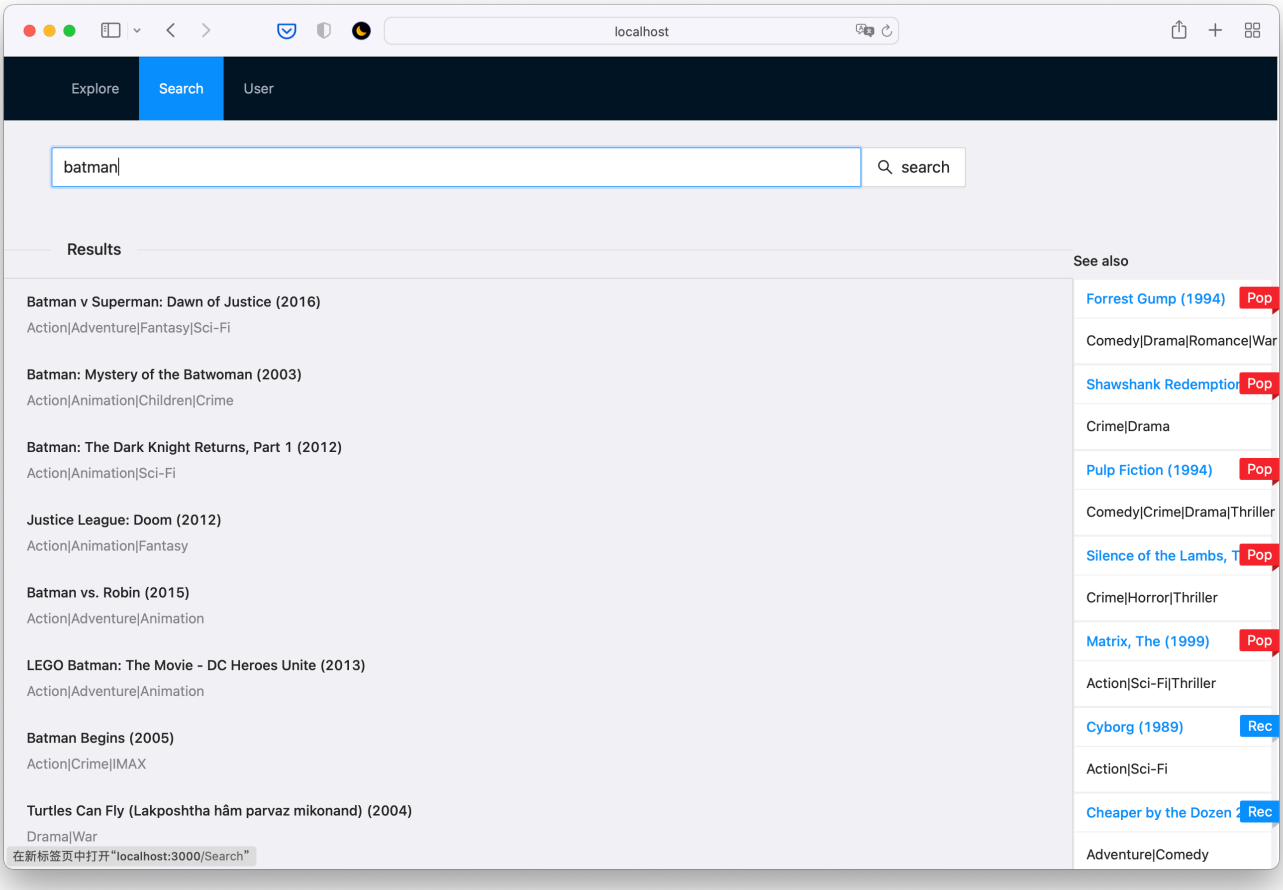
  返回评分较高/评分人数较多的电影条目,未登录用户返回10条
- 个性化推荐

  返回根据user历史行为而进行推荐的电影条目,具体推荐算法会在报告中详细描述.

搜索结果返回10条与搜索词最相关的条目,登陆用户返回5条popular推荐,5条个性化推荐,未登录用户返回10条popular推荐.

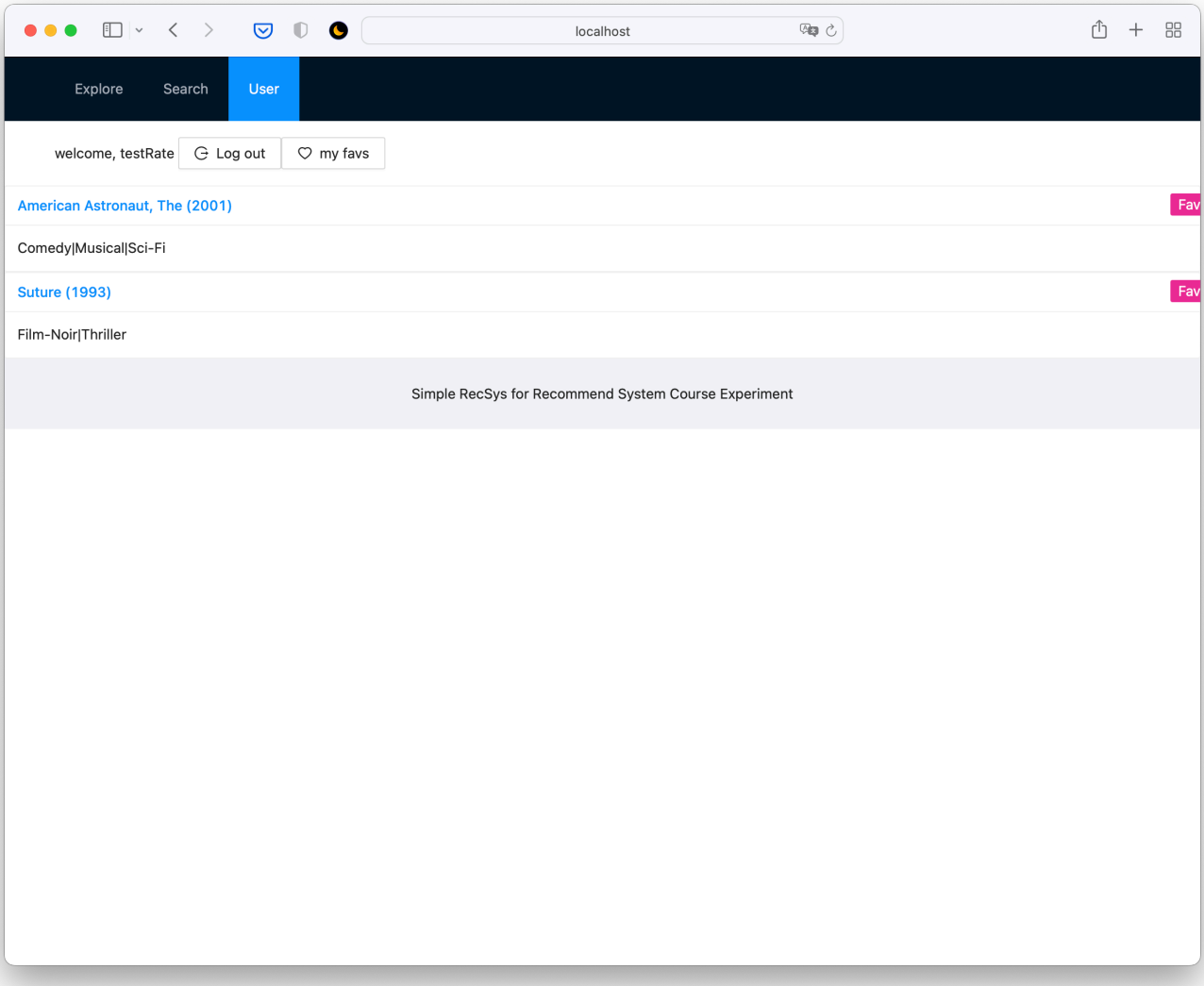
展示方式

对于每一种数据,在结果中使用名称+体裁的方式进行展示,当点击该条目时,会自动导航至Explore模块进行详情的展示,而这么做也能很好地将CTR的统计任务转移至此.



User

user模块可以对用户已经收藏的内容进行展示,可以进行登陆登出操作.



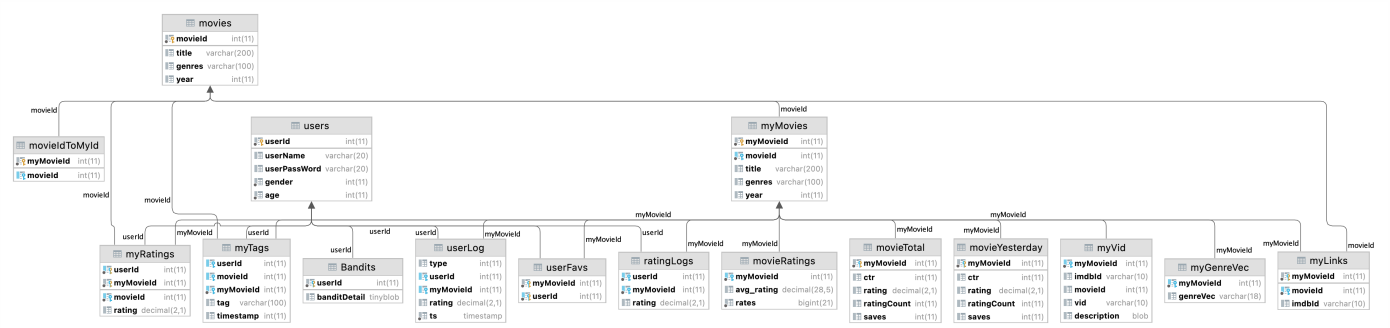
跨域问题

axios的ajax请求需要满足相同域名,但服务器和前端并没有运行在同一端口,因此我使用了proxyMiddleWare进行了代理配置,建立了3000端口到12000端口和8080端口的映射.

2)后端部分

数据库

使用mysql作为关系数据库,主要用于存放电影信息、用户信息等持久化数据,其依赖关系如图所示.



其中每个关系的用途如下:

- movies:ml-ls中的原始数据
- myMovies:将离散的movieId转换成了连续的从0到9723的myMovieId
- users:用户的注册信息
- myRatings:用户评分记录表
- myTags:用户添加标签表
- Bandits:用户Bandit偏好记录
- userLog:用户行为日志,包括ctr、收藏和评分
- userFavs:用户收藏表
- ratingLogs:测试用 用户评分记录表
- movieRatings:电影平均评分表,包含平均分与平均人数(已被下文的视图替代)
- myVid:myMovieId对应的预告片id与电影描述
- myGenreVec:对电影题材的稠密形式表示
- myLinks:myMovieId到imdbId的转换

此外,数据库中还包含两个视图:

- pop:用于计算当前评分最高的电影
- rating_stat:用于计算电影的平均评分,包含平均分与评分人数

我还使用了redis作为内存中的缓存数据库,主要用于缓存用户行为信息,避免关系数据库的频繁访问.具体方式在服务器处详细进行解释.

服务器

在服务器端,主要模块包括Svr、UserObj、DAO,辅助模块包括config、NumpyEncoder、parseGenres、ThompsonSampling,另外还有运行在Tomcat上的Lucene搜索服务.

每个模块的主要功能如下:

- Svr

  为前端开放网络API,直接与前端进行交互

  验证并管理用户生命周期,为登陆用户与非登陆用户提供不同级别的服务

  将前端请求结构,调用后端的不同业务逻辑并整合结果

- UserObj

  负责与内存键值数据库redis进行交互,主要用于用户信息的缓存与pop数据的缓存

- DAO

  负责与mysql关系数据库进行交互,主要用于封装各种粒度查询与验证服务,并对缓存冲刷逻辑进行了封装

- config

  config模块实际上为一个字典,用于进行各种变量的配置(并未广泛使用,因为目前为开发测试状态)
- NumpyEncoder

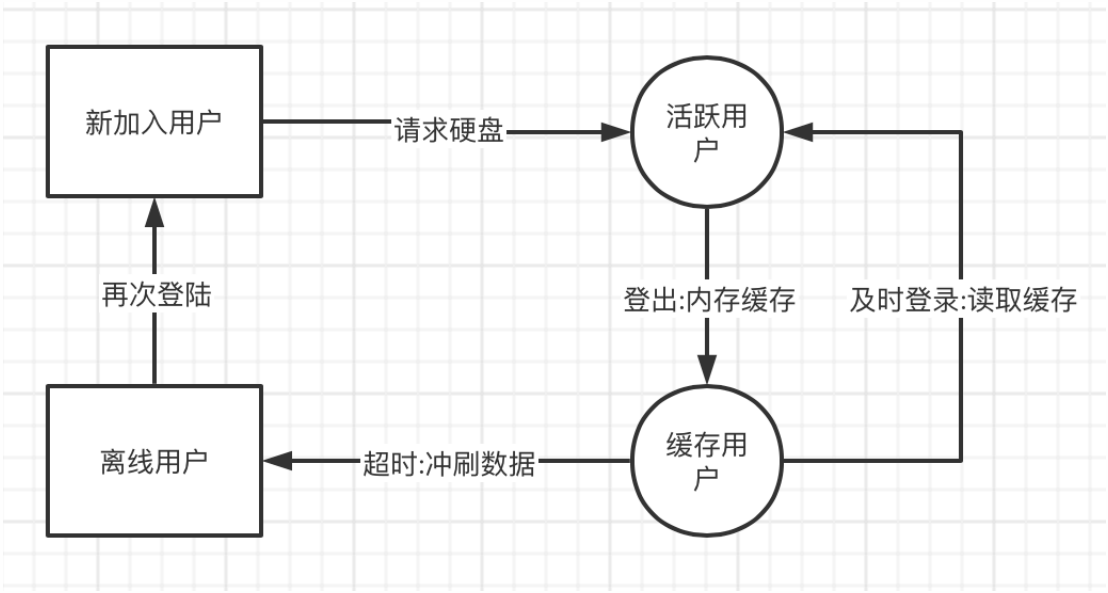
  为了解决json对numpy数据类型编码不支持问题而采取的方案,与核心业务无关
- parseGenres

  提供了电影体裁与向量的转换函数与速查表
- ThomsonSampling

  提供了进行汤普森抽样关于生成beta分布的方法,包括接受-拒绝采样与MCMC方法采样

用户生命周期

为了更好地对用户数据进行管理,我采用了一种基于用户生命周期的数据更新逻辑,具体流程如下:



对于具体的实现会在下面进行解释

用户字典

用户字典用户在内存中保留活跃用户的信息,其外层使用了一个UserObj实例进行包装.这么做的目的是为了进一步的灵活性,可以在UserObj中添加新的方法对userDict进行各种处理

内容

- userId:用户id
- userName:用户名
- userCtrDetail:用户Ctr信息,列表
- userSaveDetail:用户收藏信息,集合
- userUnsaveDetail:用户去掉收藏信息,集合
- userRateDetail:用户评分信息,字典:(电影id:电影评分)
- userFavs:用户收藏信息,字典:(电影id:电影详情)
- rateBuffer:用户评分缓存,字典:(电影id:电影评分)

之所以使用userSaveDetial和userUnsaveDetail两个集合是为了逻辑处理的简便,并将汇总工作在更底层进行完成.

userFavs中的电影详情为lazy loading方式加载,在需要时进行查询并缓存

rateBuffer为懒加载方式,在需要时进行缓存,并能够将关系数据库未命中的结果进行缓存避免连续访问磁盘但无匹配结果

Svr

启动

- 实例化Flask服务器对象app
- 初始化数据库访问对象DAO
- 配置logger写入问题
- 初始化用户数据表users:list
- 初始化用户推荐表usersRec:dict
- 读取并载入用户个性化推荐表mat:np.ndarray
- 初始化缓存工具UserObj类
- 启动服务器app

业务逻辑

1. login:登陆

   解析post请求并取得请求登陆用户的用户名与密码,使用dao进行登陆验证.

   根据用户生命周期状态决定请求dao或UserObj来获取数据.

   在响应报文中附带cookie,用于通知浏览器当前用户的名称.

2. logout:注销

   请求UserObj将用户信息缓存,并设置过期时间.

   请求dao设置定时缓冲任务,并得到任务句柄将其保存.

   删除该活跃用户信息

   在响应报文中请求删除cookie,用于通知浏览器用户注销

3. register:注册

   解析post请求取得用户的注册信息

   请求dao执行注册合法性验证与持久化存储

4. fetchFavs:请求收藏条目

   通过cookie取得用户名验证是否为登陆用户请求

   通过用户名查找session验证用户是否为合法的活跃用户并得到用户id

   通过用户id在活跃用户信息表中得到收藏条目,以json格式返回

5. movieDetail:电影详情

   解析post请求得到电影id

   请求dao对电影id对应的信息进行查询

   将查询结果以json格式返回

6. fav_status

   因为对于每个电影,需要判断用户是否已经喜欢,避免重复收藏或重复取消收藏.

   将电影id与活跃用户字典中的收藏id进行对比并将结果返回

7. changeFavStatus:更改收藏状况

通过cookie与session验证用户合法性

解析post请求得到动作(用户id,电影id,收藏/取消)

将其动作缓存在该活跃用户字典中

8. explore_next:下一条推荐

根据cookie验证用户是否登陆

若未登陆则进行随机推荐

若登陆则使用session验证合法性

对于已登陆用户,首先根据请求获取上一条推荐的反馈情况,并更新用户的bandit信息.

更新完毕后,使用新的bandit信息访问进行beta分布采样,得到最大的bandit编号.

使用该bandit编号,请求dao对该体裁电影进行抽样并返回电影id

9. increaseCTR:更新CTR信息

使用cookie与session验证用户合法性

将成功ctr的电影id保存在活跃用户字典中

0. rate:用户评分

根据cookie和session验证用户合法性

根据分值判断是进行评分还是取消评分

根据判断结果更新该用户字典

1. rateStatus

同fav_status,出于同样的原因,用户在浏览已进行过评分的电影后应当能看到当时的评价

根据cookie和session验证用户合法性

查询活跃用户字典中的本次会话评分寻找该电影是否进行过评价,有则返回

查询活跃用户字典中的评分缓存rateBuffer中寻找用户对该电影是否进行过评价,有则返回

使用dao查询数据库用户是否对该电影进行过评价

若无结果则返回-1代表无评价,并在rateBuffer中将该电影的评分设为-1

若有结果在返回对应结果,并在rateBuffer中缓存用户对该电影的评分

2. search:搜索+推荐

解析post请求获取查询词

请求UserObj查询是否存在对popular结果的缓存

若无则使用dao对popular视图进行查询,并对结果进行缓存(有过期时间)

调用handleSearch请求8080端口的lucene服务器返回查询结果

根据cookie与session验证用户是否登陆

若未登陆则返回搜索结果与pop推荐结果

若登陆则调用handleRec方法获取个性化推荐结果,与搜索结果和pop推荐结果一起返回

3. handleSearch:请求搜索

使用request向目标服务器发送请求并将结果json串解析成字典

4. handleRec:请求个性化推荐

查询usersRec中是否有缓存的推荐,若有则返回

若没有缓存则根据离线推荐算法生成的结果mat对该用户的电影偏好进行排序,并返回排名较高的若干部电影,并将结果返回

UserObj

UserObj为静态类,用于对user数据进行加工并作为工具与redis交互

工具方法

1. init:初始化

   连接redis数据库,将连接对象保存为类对象
2. shutdown:关闭

   释放与redis的连接
3. getPop:获取pop缓存

   从redis中取出popular推荐结果,并将其返回.结果可能为空.(exist,pop)
4. setPop:设置pop缓存

   将popular推荐结果进行缓存,并指定过期时间
5. fetch:获取缓存用户字典

   通过redis连接与用户id获取缓存的用户字典,并将其返回.结果可能为空.(exist,userDict)
6. cache:缓存用户字典

   将用户信息字典中的非必要信息进行清空后进行缓存,并设置过期时间

   清空的信息为rateBuffer:评分缓存数据

DAO

DAO为持有关系数据库连接的数据访问对象,用于缓存缺失时对持久数据进行查询

db

dao所在的db模块有两个全局函数,下面进行介绍

1. get_connection

   该方法用于连接mysql数据库并返回连接对象,参数可在config中更改
2. flushTask

   flushTask为冲洗缓存的业务逻辑,接收参数为dao对象和用户数据对象userObj(为UserObj实例)

   首先通过dao获取数据库执行游标cursor

   解析用户对象必要数据,即用户字典中所介绍除了buffer的所有条目

   根据CTR信息将数据保存至用户日志

   根据save信息将数据保存到用户日志、用户收藏

   根据rating信息将数据保存到用户日志、评分日志

   根据bandit信息将数据保存到bandit记录

   进行commit

DAO对象

1. init:初始化

   使用get_connetion方法建立到数据库的连接并保存

2. authenticate:登陆验证

   接收用户名和密码

   查询用户名,并比较密码,当且仅当用户存在且密码对应时成功,返回验证状态与用户id

3. register:注册

   接收用户注册信息并验证用户名是否重复

   完成后将数据插入数据库

   返回注册状态

4. close:关闭

   对事务进行提交并关闭连接

5. queryDetail:电影详细信息

   根据电影id查询电影的imdbId、预告片id、介绍、标题、体裁、平均评分和评分人数信息,以字典格式返回.

6. queryFavs:用户收藏信息

   根据用户id返回用户收藏电影id、体裁和名称

7. sample_in_genre:抽样查询

   随机返回一条给定体裁的电影id

8. queryBandits:查询bandit信息

   根据用户id查询用户bandit信息并将其解析成数组格式,若不存在则为其建立bandit数据并保存至数据库(会在算法部分详细解释)

9. getFlushTimer:生成持久化任务

   该方法为静态方法,接收dao实例、userObj对象和倒计时时间

   通过倒计时时间、数据库中的冲洗函数与dao和userObj数据实例化一个Timer对象,然后将其返回(并未开始任务)

0. queryGenres:查询体裁

   根据电影id查询电影体裁

1. queryRating:查询评分

   根据电影id,用户id查询用户对电影的评分,返回可能为空(status,rating)

2. queryPop:

   根据popular视图查询当前最热门的电影

3. queryBatchInfo:批量查询电影数据

   根据电影id批量查询电影信息