**基于大数据平台的电影推荐系统**

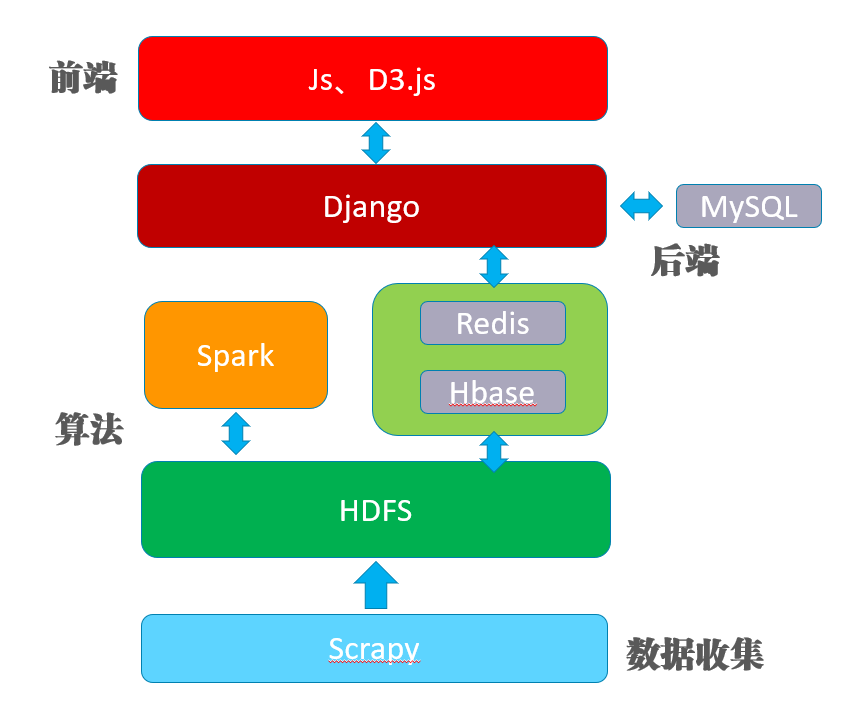
茶园小分队（陈蒙，王尧，杨一帆、林殷年、黄港）

# 项目/产品介绍

在信息爆炸的今天，获取信息已经不是什么难事，如何在众多信息中寻找自己感兴趣的内容变得越来越重要，推荐系统就应运而生。本项目着眼于对海量的电影数据进行挖掘和分析，并从数以万计的电影中给新老用户推荐其感兴趣的内容。

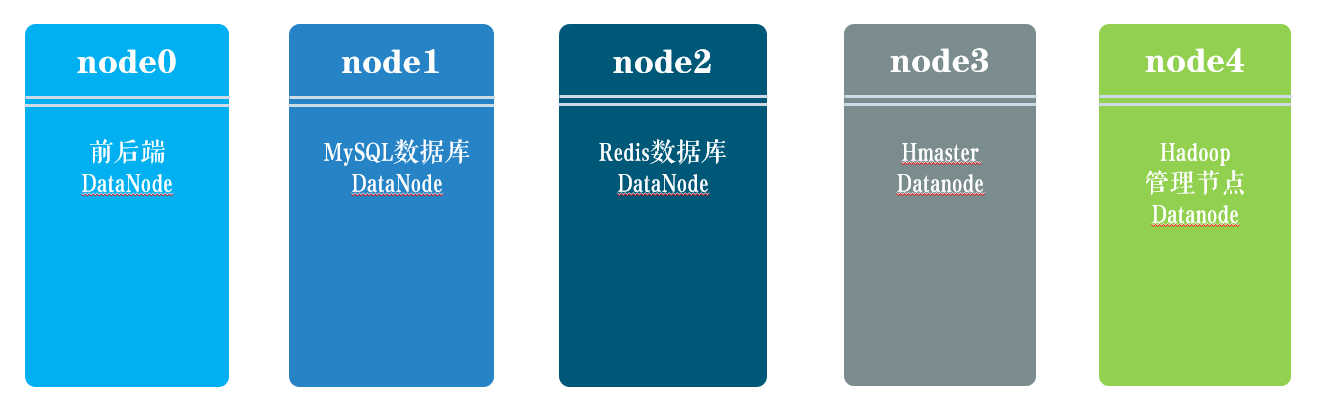
# 系统架构设计

图1. 系统架构示意图



整个系统可以分为数据收集、推荐算法、后端和前端四大模块，各部分的关系可见图1。本项目共有5台服务器，为了保证服务器负载均衡因此对各节点进行图2所示的分工。

图2. 节点分工示意图



# 环境搭建

（负责人：王尧）

每台服务器均使用名为hadoop的用户来进行运行相关任务。

系统中使用到python语言的环境用Anaconda进行包管理。

大数据平台方面，Hadoop、Spark、Zookeeper均将管理节点等配置在node4上，并且每台服务器在其中均作为Datanode存在。Spark采用Spark On Yarn的部署模式，由Yarn进行统一调度。为避免spark任务执行时，yarn过度从同一节点上分配内存，将node0-3的yarn使用内存上限设为了3g，node4设为了1g。Hbase配置时使用外部的Zookeeper，也是部署在了每台服务器上，另外为避免node4内存过于紧张，将Hbase的管理节点部署到了node3上。

数据库方面使用了MySQL、Redis、Hbase三种数据库，每种数据库的详细用途见后端服务开发。

# 数据采集

（负责人：陈蒙）

## 4.1模型数据

本项目采用MovieLens数据集（https://movielens.org）来训练电影推荐系统，数据集包含27278个电影、2000263个评分和465564个标签。

## 4.2 网页展示数据

网页展示数据包括电影名、导演、编剧、电影类型、制片国家、语言、上映日期、片长、电影简介、评论和海报，数据从豆瓣电影（<https://movie.douban.com>）中获取。

本项目采用Scrapy、selenium和FireFox批量从豆瓣电影中获取数据。先根据movie.csv中的电影名，用selenium模拟搜索以获取电影详细信息的页面地址，然后将地址传给scrapy来抓取具体格式化的数据，并进行保存。豆瓣会验证发起请求的是不是机器人，因此我从网上抓取了一些Header，每次从大量Header中随机选取一个来发起请求。由于数据量大，访问豆瓣电影过于频繁，IP会受到豆瓣封禁，我们采用设置等待时间和不停更换cookies的方法来避免IP被封。

在服务器上运行爬虫只需要运行movieServer中的bash脚本movie.sh（路径为~/cm/movieServer/movie.sh），带有两个参数，第一参数为开始电影序号，第二个参数为截止电影序号。 例如./movie.sh 0 10000，抓取0～10000电影的数据。可以修改movie.sh脚本中关于每批数量、休眠时间等参数。运行结束会在movirServer中创建posters文件夹和moviInfo.json文件，分别存储电影海报以及电影其他信息。

# 5 数据处理

只采用movieLen数据集中的特征来构造推荐算法，主要包括电影特征和用户评分特征。

## 电影特征

1. movieId：编号为1，2，3
2. tagId及relevance：movieId和所有tag的关联程度，一个movieId对应的所有tag的分数都有
3. genres：movie的类型，一个movie可能有多个类型，用|隔开

## 5.2 用户评分特征

* 1. rating：用户对电影的打分以及打分的时间
  2. tag：用户对电影的tag评价以及评价的时间

# 算法开发

（负责人：杨一帆）

## 6.1 技术栈

Spark的选用

1. 快速。和常用的Hadoop的MapReduce相比，Spark由于基于内存的运算、DAG计算模型、RDD的惰性求值等特性运算速度会快很多过滤观看人数过少的电影
2. 易用。支持Java、Python和Scala的api，还支持Python和Scala的Shell，方便快速开发和验证
3. 通用。有方便的Spark SQL库和Spark MLlib库供我们使用。可以使用Hadoop的YARN作为资源管理器。可以处理Hadoop支持的数据，比如HDFS和HBase

Scala的选用

1. 比起Java、Python，用Scala开发Spark应用代码将减少一个数量级，常用的reduce操作、aggregate操作的Scala代码相当精简
2. 可以用Spark-shell快速开发原型程序

## 6.2 movieLens数据说明

1. genome-scores.csv

第一列为movieId，第二列为电影的tagId，第三列为该电影与该tag的相关性

1. movieId,tagId,relevance
2. 1,1,0.025000000000000022
3. 1,2,0.025000000000000022
4. 1,3,0.057750000000000024
5. 1,4,0.09675
6. 1,5,0.14675
7. 1,6,0.21700000000000003
8. 1,7,0.067
9. 1,8,0.26275000000000004
10. genome-tags.csv

第一列为tagId，第二列为tag的具体名称

1. tagId,tag
2. 1,007
3. 2,007 (series)
4. 3,18th century
5. 4,1920s
6. 5,1930s
7. 6,1950s
8. links.csv

第一列为movieId，第二列为该电影在imdb上的id，第三列为该电影在tmdb上的id

1. movieId,imdbId,tmdbId
2. 1,0114709,862
3. 2,0113497,8844
4. 3,0113228,15602
5. movies.csv

第一列为movieId，第二列为该电影的具体名称，第三列为该电影的类型

1. movieId,title,genres(类型)
2. 1,Toy Story (1995),Adventure|Animation|Children|Comedy|Fantasy
3. 2,Jumanji (1995),Adventure|Children|Fantasy
4. 3,Grumpier Old Men (1995),Comedy|Romance
5. 4,Waiting to Exhale (1995),Comedy|Drama|Romance
6. tags.csv

第一列为userId，第二列为movieId，第三列为该用户认为该电影所属的tag，第四列为贴tag的时间

1. userId,movieId,tag,timestamp
2. 18,4141,Mark Waters,1240597180
3. 65,208,dark hero,1368150078
4. 65,353,dark hero,1368150079
5. 65,521,noir thriller,1368149983
6. 65,592,dark hero,1368150078
7. 65,668,bollywood,1368149876

## 6.3 推荐策略概述

1. 热门内容推荐池

选用评分高的部分电影，对新用户和老用户同时进行推荐

1. ucf推荐池

利用基于用户的协同过滤算法生成，对老用户进行推荐

1. icf推荐池

利用基于物品的协同过滤算法生成，实现基于当前电影的相似电影推荐功能

1. 注意

会在上述三个推荐算法的实现过程中加入过滤逻辑和类别打散的策略

离线侧会根据新收集的用户特征和电影特征进行天级别的更新

## 6.4 代码实现

1. 热门内容推荐池

计算每部电影的平均分

1. val movieScoreRdd = userMovieScoreRdd.map(x => (x.\_2, x.\_3)).groupByKey()
3. .map(x => (x.\_1, x.\_2.toArray.reduce(\_ + \_) / x.\_2.size)).sortBy(\_.\_1)

取出每个类别下评分最高的5部电影，然后再拉通按分数进行排序

1. val hotMovie = movieScoreRdd.join(movieGenresRdd).map(x => (x.\_2.\_2, (x.\_2.\_1, x.\_1))) //genres,score,movieId
3. .aggregateByKey(mutable.PriorityQueue[(Double, Int)]()(Ordering[Double].reverse.on(\_.\_1)))(
5. (a, i) => {
6. a.enqueue(i)
7. **if** (a.size > 5) {
8. a.dequeue()
9. }
10. a
11. },
12. (a1, a2) => {
13. a1 ++= a2
14. **while** (a1.size > 5) {
15. a1.dequeue()
16. }
17. a1
18. }
19. ).map { **case** (genres, movieGroup) => movieGroup.toArray }
21. .flatMap(x => x)
23. .map(x => movieId2Name(x.\_2)).distinct()
24. icf召回池

特征提取。电影的特征取电影的tag列、导演、年代、主演、类型等等，后面的特征先置空，等待后续加入

1. val movieFeatures = movieTagRelevanceRdd.map(x => (x.\_1, x.\_3))
3. .groupByKey().map(x => (x.\_1, x.\_2.toArray))

用电影两两之间的余弦相似度来代表其相似性

1. def cosineSimilarity(x: Array[Double], y: Array[Double]): Double = {
2. **require**(x.size == y.size)
3. dotProduct(x, y) / (magnitude(x) \* magnitude(y))
4. }
6. def dotProduct(x: Array[Double], y: Array[Double]): Double = {
7. (**for** ((a, b) <- x zip y) yield a \* b) sum
8. }
10. def magnitude(x: Array[Double]): Double = {
11. math.sqrt(x map (i => i \* i) sum)
12. }

两两组合进行召回

1. val movieMovieGroup = movieFeatures.map { x1 =>
3. val pairNewsScore = **new** mutable.PriorityQueue[(Int, Double)]()(Ordering[Double].reverse.on(\_.\_2))
5. movieFeaturesArrays.**foreach** { x2 =>
7. pairNewsScore.enqueue((x2.\_1, cosineSimilarity(x1.\_2, x2.\_2)))
8. **if** (pairNewsScore.size > 10) {
9. pairNewsScore.dequeue()
10. }
11. }
13. (x1.\_1, pairNewsScore.map(x => x.\_1).toArray)
14. }
15. ucf召回池

取每个用户评分最高的k部电影

1. val userMovieGroupRdd = userMovieScoreRdd.map(x => (x.\_1, (x.\_2, x.\_3)))
3. .aggregateByKey(mutable.PriorityQueue[(Int, Double)]()(Ordering[Double].reverse.on(\_.\_2)))(
5. (a, i) => {
6. a.enqueue(i)
7. **if** (a.size > 5) {
8. a.dequeue()
9. }
10. a
11. },
12. (a1, a2) => {
13. a1 ++= a2
14. **while** (a1.size > 20) {
15. a1.dequeue()
16. }
17. a1
18. }
19. ).map(x => (x.\_1, x.\_2.toArray.map(\_.\_1)))

每部电影召回相似电影并进行合并

1. val userMovieGroup = userMovieGroupRdd.map { **case** (user, movieGroup) =>
3. val userMovieGroup = movieGroup.map(x => (x, user))
5. userMovieGroup
7. }.flatMap(x => x) //movie, user
9. .join(movieMovieGroup).map(\_.\_2) //(user, movieGroup)
11. .aggregateByKey(ArrayBuffer[Int]())(
13. (a, i) => {
14. a ++= i
15. a
16. },
17. (a1, a2) => {
18. a1 ++= a2
19. a1
20. }
21. ).map(x => (x.\_1, x.\_2.toArray))

后续ucf召回池可以通过收集用户的性别、年龄、城市、机型、渠道号等个人信息，增加基于用户特征的多路召回

## 6.5 更新策略

1. 训练数据的收集和更新

* 用户自己填写的画像信息，性别、年龄、城市等，以及设备信息等等
* 更新的电影的信息，用户对电影的评论数、点赞数，电影的分数等等，以及新上映的电影
* 用户点击记录，包括点击时的环境信息、用户信息、电影信息

1. 推荐池的定时更新

* 目前是根据不断更新的用户的画像信息、电影的特征信息，对三大推荐池进行天级别的更新
* 后续服务器性能足够的话，可以以分钟级别的更新实现更加精准的推荐

## 6.6 结果

最终在推荐算法侧得到三个结果文件

1. hotMovie

热门内容推荐池，用于对未登录用户的推荐

1. userMovieGroup

ucf召回池，用于对每个已登录的userId，推荐一批movie

每行第一个为userId，后面为movieName，使用"\t"分割

1. movieMovieGroup

icf召回池，用于对每个movie，推荐一批相似的movie

每行第一个为当前的movieName，后面为推荐的movieName，使用"\t"分割

## 6.7 后续功能

1. 线上监控

* 对网页端的人均浏览时长、DAU、点击量、曝光量等数据进行监控
* 根据线上数据制作CTR实时变化曲线，便于实时观察算法效果

1. 用户分桶

* 将所有用户分为不同的桶，每次实验只上线到其中一个桶
* 观测实验效果良好之后再进行全量

1. 多路召回

* 目前的召回一共只有三路，对某些用户起作用的只有一路
* 增加更细粒度的召回方式，比如基于机型、基于用户渠道等用于新用户的专项召回

1. Ranking

* 目前的推荐系统中只有召回和粗排
* 收集环境特征、用户特征、物品特征等信息，离线侧训练Ranking模型（LR、DNN、DeepFM等）
* 实时推荐的时候，在线侧调用训练好的Ranking模型对电影进行打分和排序
* 涉及到featureMaker、featureCollector等多种策略

1. Reranking

* 目前重排策略较为简单
* 可以通过一些运营规则（内容排版、大型热点等）和多样性规则，最终呈现每一刷的电影

# 7 后台服务开发

（负责人：林殷年）

## 7.1 后台功能描述

后台部分的主要功能是连通前端界面与后台推荐算法和爬虫生成的数据。对后台推荐算法和爬虫生成的数据，本部分对其进行批处理，存放入各自的数据库中，定义并实现数据读写接口。对前端界面的数据请求，后端调用对应的数据接口，响应前端界面的请求。

## 7.2 数据库概念抽象

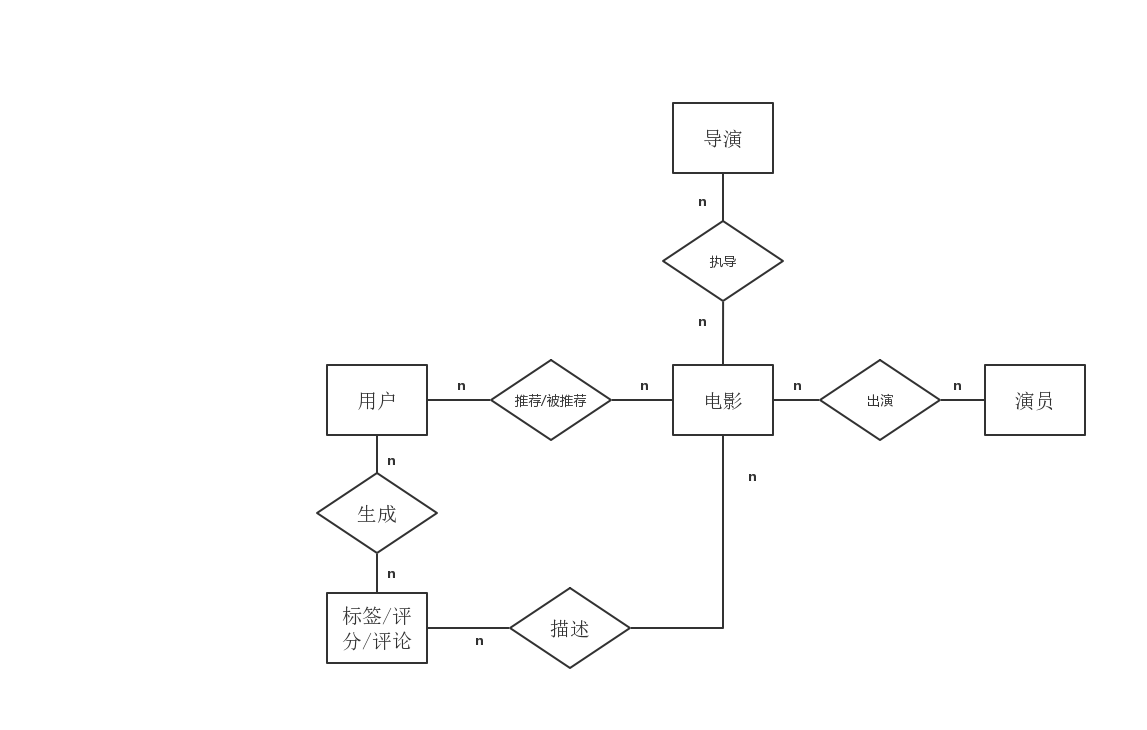


图3. 数据库概念设计图

电影推荐系统的数据概念图相对简单，在系统中主要的实体是“用户”和“电影”，它们之间是推荐与被推荐的关系，而在用户与系统交互的过程中，会产生各种纪律，例如用户给电影的自定义标签、评分、评论、赞等等，也算作一类实体，它们由用户生成，表达的是对电影的描述关系。另外，由于爬取的数据中还包括演员和导演的数据，所以在概念图中也表示了这些数据。

## 7.3 数据库技术选型

本系统结合了Hbase、MySQL与Redis三种不同架构的数据库，其主要考虑是：Hbase是列存储结构的数据库，存储结构灵活，适合处理只需要返回部分列的查询，因此适合用来存储电影的相关信息；同时，由于其写入性能佳，也适合用来维护用户反馈信息的数据。

Redis是Key-value形式的数据库，基于内存，适合存储规模较小但是读写频繁的数据，因此我们考虑主要将其用作缓存数据库，存放经常需要读写的数据，比如用户反馈数据和活跃用户的推荐列表数据。

MySQL是传统的关系型数据库，适合管理需要一致性的数据，安全性和稳定性较好，适用于结构化数据，因此我们认为适合用来管理用户数据。

## 7.4 模块设计

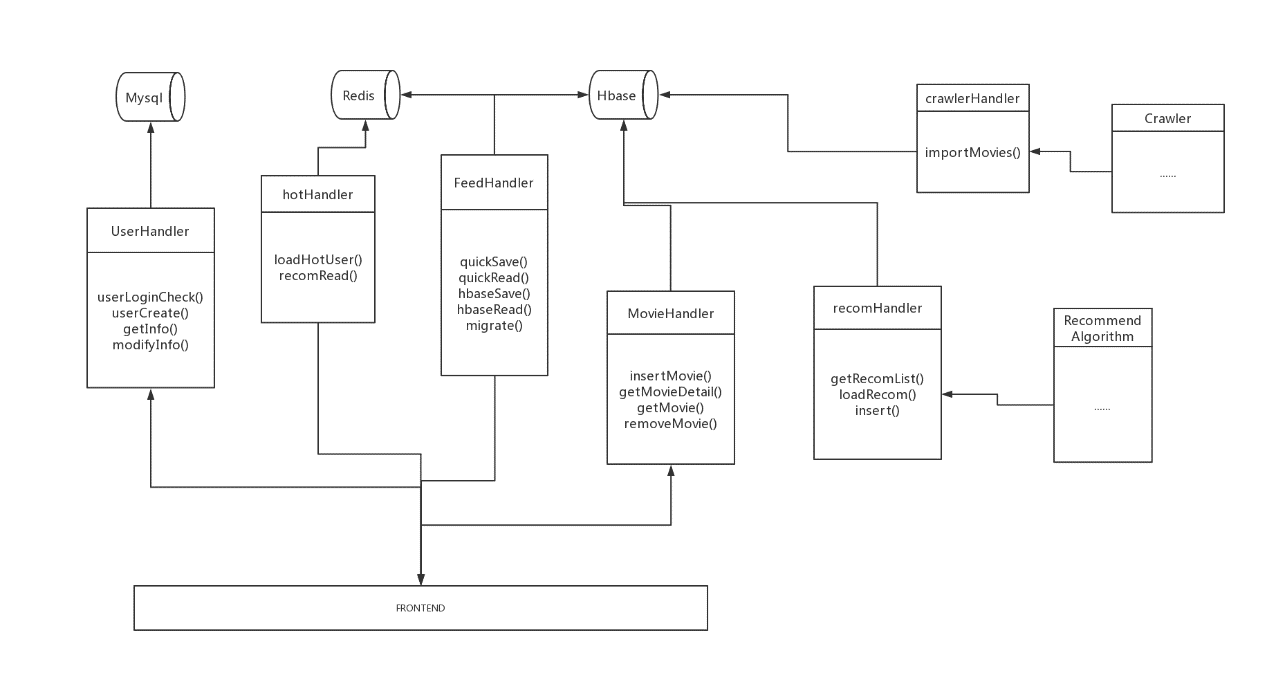


图4. 后端开发过程中的类与软件

图4描述了在后台服务开发过程中定义的各个类和用到的数据库软件。

UserHandler用于接收前端与用户相关的数据请求，包括登录、注册、查看用户个人信息、修改用户个人信息等。这部分数据主要存储在Mysql中，以确保安全性和稳定性。

hotHandler用于维护存储在Redis中的缓存数据，主要与Redis交互，缓存数据有两种，一种是前端页面返回的与用户行为和反馈有关的数据，另一种是我们会将活跃用户的电影推荐结果维护在redis中，以提高活跃用户的响应速度。

FeedHandler主要用于管理与用户反馈有关的数据，要与Redis以及Hbase交互。hotHandler将收集到的用户反馈信息存入redis以后，由FeedHandler读入，转换格式，最后在HBase中保存。

MovieHandler用于处理与电影信息有关的数据的增删改，主要与HBase交互。每部电影都有详细信息，其中片名、评分、海报链接这三项还需要当做缩略显示时的信息，因此MovieHandler既支持一次查询电影的所有信息，也可以只查询缩略显示需要的信息。

RecomHandler用于与电影推荐算法交互，将电影推荐算法的结果写入数据库中，并提供读取接口

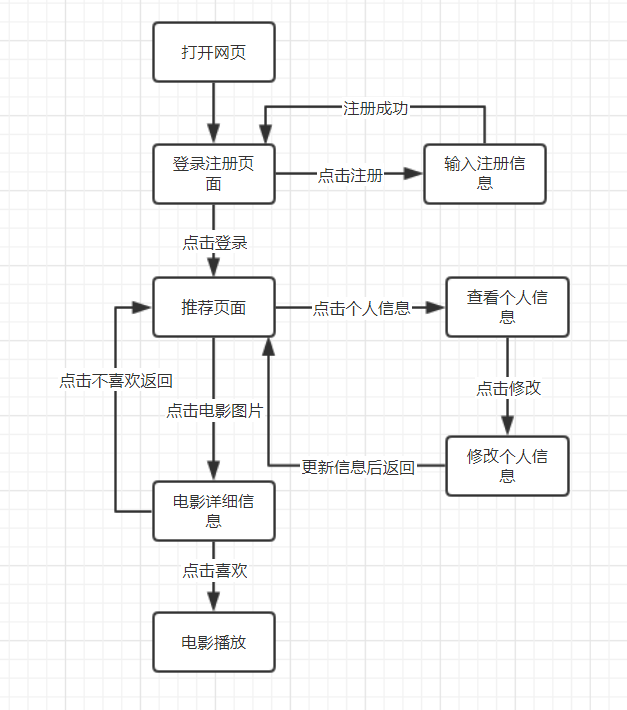
CrawlerHandler用于与爬虫部分交互，将爬虫爬到的结果存入Hbase中。

# 8 前端开发

（负责人：黄港）

## 8.1用户操作流程

图5. 用户操作流程



## 8.2用户注册

用户首先打开网页，进入主界面，点击注册和登录的选项卡，进行注册或登录；选择注册按钮，输入用户名和密码，点击注册，注册成功后返回登录页面；

## 8.3用户登录

选择登录按钮，输入用户名和密码，点击登录，如果登录用户名和密码正确就进入推荐电影的页面；

## 8.4 查看与修改个人信息

进入推荐电影的页面，如果是第一次登录则推荐当前热门电影，用户可以选择点击个人信息查看个人信息，如果有需要修改的地方点击修改按钮进入个人信息修改页面进行修改，修改完成后返回推荐页面；

## 8.5推荐电影

在修改个人信息完成后，系统有了用户的各个重要的信息，这样系统就会根据用户的个人信息情况进行电影的推荐

## 8.6电影详细信息

用户点击电影图片进入电影详细介绍页面，可以对该电影的所有信息，包括主演，导演，评分，他人评论，地点，语言等进行查看；

## 8.7用户反馈

用户选择可以在该页面选择喜欢或者无感按钮进行反馈，当选择喜欢的时候，会返回后台用户喜欢的数据，并跳转到电影播放的页面；如果选择无感的时候，会返回后台用户不喜欢的数据，并跳转到推荐页面，让用户选择其他可能喜欢的电影。

## 8.8前后端数据交互

登录注册过程中会有用户的账户名和密码还有userid进行前后端的交互；在查看与修改用户信息的过程中会有用户的userid，性别，所在地，生日等信息的交互；在推荐电影中会有movieid，电影图片，评分，电影名等信息的交互；在电影信息查看过程中会有movieid，电影图片，名称，导演，主演，主要内容，语言，评论的数据的交互；在用户点击反馈按钮会有userid、movieid、喜欢标志的交互。

# 9 项目地址

项目的github地址为：https://github.com/zero1997/movieRecommend

# 10 安装部署

安装部署内容较长，见文件 安装部署.md