# 摘 要

在线电影种类繁多、数量庞大，人们在搜索感兴趣的电影时会浪费大量时间。推荐系统能够根据用户已有的偏好信息，给用户推荐符合他们偏好的物品，这能节省用户的时间，并且增强用户粘着性。另一方面，随着时间推移，很多好的电影作品逐渐被人遗忘，推荐系统能够根据已有的评价信息，将电影推荐给真正欣赏它的人，对于艺术作品而言，这是极具现实意义的。

推荐系统的应用已经相当广泛了，在电商、新闻、音乐方面都有很好的表现。推荐系统的算法繁多，理论成熟，但是仍然存在很多不足，比如推荐质量和实时性。算法是对数据敏感的，任何算法都不能普遍适应各种业务场景。而对于电影推荐系统而言，以前的电影往往有很多评分数据都是以前的，而人的兴趣是随时间变化的，并且人们很少会对看过的电影打分，因此就导致数据的时间跨度大，稀疏性问题严重。

本文对电影推荐系统的推荐质量问题进行了探讨，首先介绍了立论的相关背景，然后介绍了相关计算平台以及数据采集平台，接着详细介绍了基于概率矩阵分解模型的协同过滤算法，并对其在Spark中的实现做了分析。将数据的时效性融入到算法的目标优化函数中，对算法进行改进。最后，使用Spark、Redis等技术实现了一个简易的电影推荐系统。

本文的主要成果：利用数据的时间戳，改良了Spark中的协同过滤算法，并在MovieLens数据集上进行了交叉验证，与原有的算法进行实验对比，发现其在时间跨度较大的数据集上，误差有明显减小。

**关键词**：矩阵分解、协同过滤、推荐系统、电影、Spark

# Abstract

Online movies are so diverse and numerous that people waste a lot of time searching for movies they are interested in.The recommendation system can recommend items that meet users' preferences based on their existing preference information, which can save users time.On the other hand, as time goes by, many good films are gradually forgotten. The recommendation system can recommend films to people who really appreciate them based on the existing evaluation information.For works of art, this is of great practical significance.

Recommendation system has been widely used in e-commerce, news and music.There are many algorithms and mature theories in the recommendation system, but there are still many shortcomings, such as recommendation quality and real-time performance.Algorithms are sensitive to data, and no algorithm can universally adapt to various business scenarios.For the film recommendation system, a lot of rating data of previous films are always the same, and people's interest changes with time, and people seldom rate the films they have seen, so the data span is large and the sparsity is serious.

This thesis discusses the recommendation quality of the film recommendation system, first introduces the relevant background of the theory, then introduces the relevant computing platform and data acquisition platform, then introduces the collaborative filtering algorithm based on the probability matrix decomposition model in detail, and analyzes its implementation in Spark.The timeliness of the data is integrated into the objective optimization function of the algorithm to improve the algorithm.Finally, using Spark, Redis and other technologies to achieve a simple film recommendation system.

The main achievements of this thesis: improved the collaborative filtering algorithm on Spark by using the time stamp of data, and cross-verified the algorithm on MovieLens. Compared with the original algorithm, it was found that the error was significantly reduced in the data set with a large time span.

**Keywords**：Matrix Factorization, Collaborative Filtering, Recommended System, Movie, Spark

# 目 录

[摘 要 I](#_Toc9661925)

[Abstract II](#_Toc9661926)

[目 录 III](#_Toc9661927)

[图目录 VI](#_Toc9661928)

[表目录 VII](#_Toc9661929)

[第一章 引言 1](#_Toc9661930)

[1.1 项目背景 1](#_Toc9661931)

[1.2 国内推荐系统的发展概况 2](#_Toc9661932)

[1.3 本文主要的研究工作 5](#_Toc9661933)

[1.3.1 协同过滤系统 6](#_Toc9661934)

[1.3.2 推荐系统应用 6](#_Toc9661935)

[1.4 本文的组织结构 7](#_Toc9661936)

[第二章 技术综述 8](#_Toc9661937)

[2.1大数据相关技术概述 8](#_Toc9661938)

[2.1.1 Spark分布式计算引擎 8](#_Toc9661939)

[2.1.2弹性分布式数据集RDD 10](#_Toc9661940)

[2.1.3键值数据库Redis 10](#_Toc9661941)

[2.2推荐算法 11](#_Toc9661942)

[2.2.1相关术语 11](#_Toc9661943)

[2.2.2矩阵分解模型 12](#_Toc9661944)

[2.2.3交替最小二乘拟合 13](#_Toc9661945)

[2.2.4算法的通信开销 14](#_Toc9661946)

[2.3 本章小结 17](#_Toc9661947)

[第三章 电影推荐系统的需求分析 18](#_Toc9661948)

[第四章 电影推荐系统的设计 25](#_Toc9661949)

[4.1系统总体设计 25](#_Toc9661950)

[4.1.1系统架构 26](#_Toc9661951)

[4.1.2逻辑视图 27](#_Toc9661952)

[4.1.3物理视图 28](#_Toc9661953)

[4.1.4开发视图 29](#_Toc9661954)

[4.1.5进程视图 30](#_Toc9661955)

[4.2功能模块设计 30](#_Toc9661956)

[4.2.1权限验证 30](#_Toc9661957)

[4.2.2个性化推荐 33](#_Toc9661958)

[4.2.3热门推荐 36](#_Toc9661959)

[4.2.4评分功能 37](#_Toc9661960)

[4.2.5社区功能 38](#_Toc9661961)

[4.3数据库设计 40](#_Toc9661962)

[4.3.1 E-R图 40](#_Toc9661963)

[4.3.2 数据库表设计 40](#_Toc9661964)

[4.4算法设计 43](#_Toc9661965)

[4.5 本章小结 44](#_Toc9661966)

[第五章 电影推荐系统的实现 45](#_Toc9661967)

[5.1影评系统的实现 45](#_Toc9661968)

[5.1.1权限验证 45](#_Toc9661969)

[5.1.2热门推荐 46](#_Toc9661970)

[5.1.3综合评分 48](#_Toc9661971)

[5.2推荐引擎的实现 50](#_Toc9661972)

[5.2.1模型训练 50](#_Toc9661973)

[5.2.2TopK推荐 51](#_Toc9661974)

[5.3算法实现与分析 52](#_Toc9661975)

[5.3.1数据集 52](#_Toc9661976)

[5.3.2算法实现 52](#_Toc9661977)

[5.3.3评估标准 53](#_Toc9661978)

[5.3.4实验分析 54](#_Toc9661979)

[5.4本章小结 57](#_Toc9661980)

[第六章 系统测试 58](#_Toc9661981)

[6.1系统测试环境 58](#_Toc9661982)

[6.2功能测试 58](#_Toc9661983)

[6.2.1鉴权功能测试 58](#_Toc9661984)

[6.2.2热门推荐功能测试 59](#_Toc9661985)

[6.2.3评分功能测试 60](#_Toc9661986)

[6.2.4综合评分功能测试 61](#_Toc9661987)

[6.3本章小结 62](#_Toc9661988)

[第七章 总结与展望 63](#_Toc9661989)

[7.1 总结 63](#_Toc9661990)

[7.2 进一步工作展望 63](#_Toc9661991)

[参 考 文 献 65](#_Toc9661992)

[致 谢 69](#_Toc9661993)

# 

# 图目录

[图 2.1 Spark运行架构 9](#_Toc9766929)

[图 2.2 ALS的分布式算法伪代码 15](#_Toc9766930)

[图 2.3 BlockRatings逻辑存储结构 16](#_Toc9766931)

[图 2.4 计算特征向量 16](#_Toc9766932)

[图 3.1 影评系统用例图 18](#_Toc9766933)

[图 4.1 推荐系统功能模块划分 25](#_Toc9766934)

[图 4.2 系统架构图 26](#_Toc9766935)

[图 4.3 系统逻辑视图 27](#_Toc9766936)

[图 4.4 系统物理视图 28](#_Toc9766937)

[图 4.5 系统开发视图 29](#_Toc9766938)

[图 4.6 系统进程视图 30](#_Toc9766939)

[图 4.7 访问权限的验证流程 31](#_Toc9766940)

[图 4.8 权限验证模块类图 32](#_Toc9766941)

[图 4.9 个性化推荐时序图 33](#_Toc9766942)

[图 4.10 个性化推荐系统 33](#_Toc9766943)

[图 4.11 计算推荐项 35](#_Toc9766944)

[图 4.12 热门推荐时序图 36](#_Toc9766945)

[图 4.13 评分时序图 37](#_Toc9766946)

[图 4.14 用户点赞活动图 38](#_Toc9766947)

[图 4.15 评论列表的数据结构 39](#_Toc9766948)

[图 4.16 评论时序图 39](#_Toc9766949)

[图 4.17 影评系统E-R图 40](#_Toc9766950)

[图 5.1 AuthInterceptor类部分代码 45](#_Toc9766951)

[图 5.2 AuthActionInterceptor类部分代码 46](#_Toc9766952)

[图 5.3 非法操作跳转至登录页面 46](#_Toc9766953)

[图 5.4 热门推荐 47](#_Toc9766954)

[图 5.5 热门推荐列表 48](#_Toc9766955)

[图 5.6 评分功能的实现 49](#_Toc9766956)

[图 5.7 综合评分数据结构 49](#_Toc9766957)

[图 5.8 查询综合评分 50](#_Toc9766958)

[图 5.9 定时脚本 50](#_Toc9766959)

[图 5.10 模型训练 51](#_Toc9766960)

[图 5.11 TopK推荐 52](#_Toc9766961)

[图 5.12 算法伪代码 53](#_Toc9766962)

[图 5.13 误差随变化图 55](#_Toc9766963)

[图 5.14 误差随变化图 55](#_Toc9766964)

[图 5.15 误差对比图 56](#_Toc9766965)

[图 6.1 Redis控制台的热门排序 60](#_Toc9766966)

[图 6.2 评分功能运行效果 62](#_Toc9766967)

[图 6.3 Redis控制台综合评分信息 62](#_Toc9766968)

# 表目录

[表 2.1 Redis数据类型 11](#_Toc9662135)

[表 3.1 个性化推荐用例 19](#_Toc9662136)

[表 3.2 电影评分用例 20](#_Toc9662137)

[表 3.3 热门推荐用例 20](#_Toc9662138)

[表 3.4 发布影评用例 21](#_Toc9662139)

[表 3.5 修改影评用例 22](#_Toc9662140)

[表 3.6 删除影评用例 22](#_Toc9662141)

[表 3.7 评论用例 23](#_Toc9662142)

[表 3.8 点赞用例 23](#_Toc9662143)

[表 3.9 权限验证用例 24](#_Toc9662144)

[表 3.10 综合评分用例 24](#_Toc9662145)

[表 4.1 电影信息表film 41](#_Toc9662146)

[表 4.2 用户信息表user 41](#_Toc9662147)

[表 4.3 评分信息表rating 42](#_Toc9662148)

[表 4.4 影评信息表blog 42](#_Toc9662149)

[表 4.5 评论信息表comment 43](#_Toc9662150)

[表 5.1 ALS参数与均方误差 54](#_Toc9662151)

[表 5.2 ALS-T参数与均方误差 56](#_Toc9662152)

[表 6.1 系统测试环境 58](#_Toc9662153)

# 第一章 引言

## 1.1 项目背景

## 1.2 国内文件系统的发展概况

## 1.3 本文主要的研究工作

本文所研究的是个性化推荐系统，而个性化推荐并不能独立地存在，它以一个基于大规模数据分析计算引擎的网络日志发布系统为上下文，依赖于特定用户行为数据。结合具体的业务，从用户的角度看这就是一个电影社区。既然是一个社区，那必然需要有登录这册功能、发布日志、讨论话题的功能等等。游客可以在这个系统中浏览电影的资讯、搜索信息，浏览某一部电影相关的讨论点评情况，获取热门电影相关资讯。注册登录后的用户才算是加入了社区，可以发表电影观后感，给电影打分。有了这些基础，个性化推荐系统才有研究的条件。推荐系统会根据用户行为反馈情况预测用户兴趣偏好，为这些用户过滤出推荐项，计算用户对电影的评分的预测值。

本系统将分为数据采集层和推荐引擎两个部分来实现。数据采集层也可以称作应用层，负责数据的展示，收集用户的交互数据，包含了所有业务逻辑。推荐引擎本质上就是一个协同过滤系统，对数据进行挖掘计算分析。

本文包括的主要工作如下。

理论知识储备：查阅项目开发中所使用的计算平台技术以及推荐算法的国内外专著文献，特别是底层原理和技术的优缺点，做到知识储备完善。

项目背景描述：描述了当前国内在线影院的发展现状，分析了产生的数据过载问题，阐述了开发个性化推荐系统的意义。

项目技术的描述：详细介绍了Spark平台的运行架构和RDD的设计思想，适当展开底层原理，并且结合算法特点说明使用Spark能高效地运行算法，适合处理大规模数据。

项目需求及设计的构架和阐述：通过用例图和用例描述列表阐述了系统的功能需求，使用UML对软件总体结构进行设计建模，通过类图、顺序图对软件各个模块的详细设计进行说明，并阐述了设计的思路和与这样设计的优点。

项目重要模块的详细实现的阐述：对各模块的重点部分给出了代码实现，难点部分使用流程图，阐述了代码的思路和算法流程。

软件系统的测试：给出影评系统的功能测试的测试过程和结果分析。

项目总结：对项目开发过程进行总结，反思其中的不足和可以改进的地方。

## 1.4 本文的组织结构

本文的组织结构如下：

第一章 引言部分。介绍了长尾现象，以及在线影院的大规模数据产生的信息过载，以及国内在个性化推荐系统方向的研究现状。

第二章 技术综述。将项目所要涉及的大数据相关的技术和软件栈做了介绍，包括Spark数据分析引擎、缓存数据库Redis。此外还研究了Spark.mllib中的个性化推荐算法的原理。

第三章 电影推荐系统的需求分析。通过用例图和用例描述分析了系统的功能需求，阐述了系统各个模块的职责。

第四章 电影推荐系统的设计。对系统的总体设计思路进行了概述，对项目模块进行了划分、对系统数据结构、功能模块设计进行了详细描述。对推荐算法的优化函数进行了改良设计。

第五章 推荐系统的具体实现。在需求分析和系统设计的基础上，实现了推荐引擎层，并且详细描述了业务层的电影评分、个性化推荐、热门推荐等功能的实现，对算法进行了实验与评估。

第六章 系统测试。对影评系统的主要功能进行测试，意在验证功能的正确性，确保系统的可用性。

第七章 总结与展望。总结论文期间所做的工作，并且就该电影推荐系统的未来扩展作了进一步展望。

# 第二章 技术综述

本章分为三部分，第一部分介绍了文件系统的基本概念

## 2.1 文件系统概述

文件系统是操作系统中最重要的部件之一。

本节主要介绍了文件系统的概念，以及Linux的文件系统设计思想，为下文的网络文件系统以及分布式文件系统打下理论基础。

### 2.1.1 文件的本质

### 2.1.2 Linux文件系统

开放式系统之间的通信遵守了一系列的规则，这些规则定义了系统发送和接收的消息的格式，

### 2.2 网络文件系统

分布式系统中的通信的核心是进程间通信。

## 2.3 分布式文件系统

网络应用的交互性越来越强，服务器消息推送技术

本节主要介绍了

## 2.3 本章小结

本章第一节介绍了

# 第三章 文件系统的分析与设计

Sophon文件系统将集群分布式文件系统、网络文件系统两种文件系统整合在了一起，实现了数据的共享与访问透明。

## 3.1系统边界

## 3.2需求分析

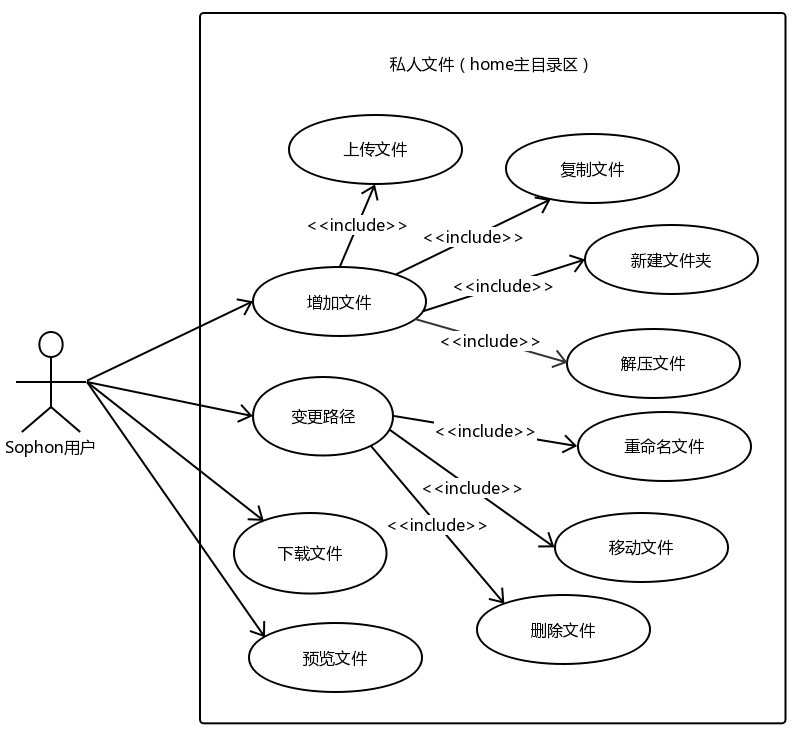
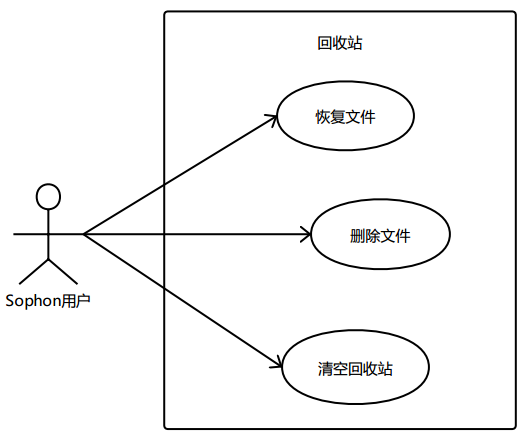


图 3.1 主目录文件管理用例

基于Sophon的Notebook开发环境需求，分析得到Sophon用户的主目录文件管理用例（图3.1）。



在主目录或公共目录被删除的文件会暂存在回收站。

表3.1文件上传用例

| **ID** | UC1 | **名称** | 文件上传 |
| --- | --- | --- | --- |
| **描述** | 用户将本地文件上传到集群或者服务器的指定位置 | | |
| **参与者** | Sophon用户 | | |
| **前置条件** | 用户已经完成CAS身份验证 | | |
| **后置条件** | 无 | | |
| **优先级** | 高 | | |
| **正常流程** | 1. 用户选择项目，点击进入。 2. 用户进入文件管理界面，选择目标路径，点击上传文件，弹出本地文件视图窗口。 3. 选择要上传的文件，点击确定。 4. 系统弹出上传进度条，上传完毕时文件列表中出现新上传的文件。 | | |
| **扩展流程** | 1. 目标路径下已经存在同名文件。 2. 自动在新上传的文件名中加上数字“1”。 3. 如果依然有名称冲突，数字递增。 4. 用户可以在文件上传时进行其他的操作。 5. 用户可以在进度条未达到100%时点击取消。    1. 文件终止上传。 | | |

# 第四章 文件管理系统的实现

本章分为四个部分，总体结构设计、功能模块设计以及数据库设计。第一部分采用UML对整体的软件系统进行建模，分析软件的体系结构以及行为模式；第二部分着重描述与推荐系统密切相关的几个功能模块设计，结合时序图、类图等，详细描述其设计细节；第三部分则对系统的底层数据结构进行设计，结合外在表现特征定义系统的内模式，第四部分对矩阵分解模型的目标函数进行设计，并给出具体解法。

## 4.1系统总体设计

图 4.1 推荐系统功能模块划分

本系统由影评系统（本质上是网络日志发布系统）和推荐引擎两个部分组成。图4.1展示了推荐系统的各个功能模块。其中影评系统包括个性化推荐、评分系统、热门推荐、权限验证，以及社区功能。权限验证包括用户的登录注册，以及页面的访问控制。社区模块包括了影评的点赞和评论功能。推荐引擎模块包括数据抽取、协同过滤、个性化推荐算法、以及模型的训练与存储。

影评系统的作用是收集用户个人信息，以及对电影的评分数据，收藏记录，搜索记录，对电影的评论信息，以及对影评的评论信息。推荐引擎是项目的核心模块，它通过挖掘这些收集到的评分信息，对已有的评分数据进行分析计算，预测用户对每一部电影的评分，然后过滤出用户可能感兴趣的电影。

### 4.1.1系统架构

图 4.2 系统架构图

图4.2展示的是推荐系统的软件架构。整体可以分成三层，分别是视图层，后台服务层，计算层。系统采用MVC模式，对请求进行拦截处理，将后台数据通过FreeMarker模板引擎动态地将数据模型映射到视图层，通过Restful接口加载视图资源并输出到浏览器。

前端采用JQuery进行视图的渲染以及交互逻辑的实现，评分组件Raty通过Ajax异步请求的方式与后台的评分子系统进行交互。

后台服务通过Jedis和Mybatis命令模式与数据库交互，使用Hikari数据库连接池与Mysql建立高效稳定的连接，并且通过缓存数据库获取计算层的数据分析结果。其中个性化推荐模块依赖于推荐引擎的计算结果，同时也会根据业务逻辑对计算结果进行过滤。

推荐引擎基于Spark分布式计算平台，它包括数据预处理模块、DBUtil、Recommender实体。DBUtil收集MySql数据库中的数据作为算法的输入，预处理模块会根据缓存中的数据对算法的输入进行修正，然后Recommender将数据发送到多个节点进行并行计算，最后将计算结果输出到缓存中。

### 4.1.2逻辑视图

如图4.3所示是系统的逻辑视图，其中包含了部分系统后台的实现类以及接口，以及它们之间的依赖关系、层次关系、组合关系。

图 4.3 系统逻辑视图

Service包中的实现类处理后台的业务逻辑，其中RecommendService提供了个性化推荐业务接口，实现了热门推荐的功能，FilmService实现了电影评分、查询电影评论列表、查看电影详情的功能，BlogService实现了发布影评、影评详情等功能，UserService实现了用户登录注册等功能。

Controller的职责通常是处理客户端请求，将数据模型和对应的视图组合起来，其中的类的方法通常返回的是rest风格的字符串，这些字符串会被解析成相关视图的访问定位。其中HomePageController处理主页各个功能点的请求，包含个性化推荐功能的入口。BlogController处理与影评相关的请求，包括发布、评论、点赞、列表展示、详情页。FilmController处理与电影相关的请求，包括评分、关注、列表查询等。UserController处理用户相关的请求，包括登录、注册、重置密码、修改个人信息、查看个人点赞列表、维护个人的社区信息等。

Model抽象出了系统功能需求中的实体，对软件系统进行建模，是整体设计中最核心最基础的模块，所有业务功能都是基于特定的Model展开。其中Blog类代表影评，包含影评内容、发布时间、作者id等成员变量。Rating类代表评分记录，包含用户id、电影id、评分值、评分时间。User类代表用户，包含帐户名密码、电话号码等信息。

Mapper包中是一系列的接口，定义了数据访问层的通信规范。其中包括了UserMapper、BlogMapper、FilmMapper、RatingMapper，分别与Model层的类User、Blog、Film、Rating对应。

### 4.1.3物理视图

图 4.4 系统物理视图

本系统中的Spark采用YARN模式安装，在这种模式下Spark无须部署多个节点，实际运行程序的是YARN。Spark作为客户端将任务提交给YARN的资源管理器。

图4.4是系统的物理视图。Web服务端采用了Redis和MySql数据库作为数据持久化层，处理浏览器请求的Web服务器采用的是Tomcat，这是以嵌入的方式配置在SpringBoot应用中的。部署时，只需要将应用打包到目标节点，然后使用命令行运行主方法即可。

在推荐引擎的节点中，安装了Spark、HDFS。推荐引擎实际上就是基于Spark构建的一个程序模块，这个程序模块会被提交到Spark计算引擎，程序会实例化一个SparkContext，此外，需要导入JDBC和Jedis的依赖Jar包，并且必须在Spark中配置这层依赖。如果只是在程序中导入Jar包，运行时会出现无法找到Jedis和JDBC的错误，因为程序并不是本地直接运行，而是在Executor中运行，Executor运行时会先去下载Task的运行时依赖，而如果不在Spark Context中配置，执行到import这条命令的时候，它只能扫描本地依赖，必然会报错。

### 4.1.4开发视图

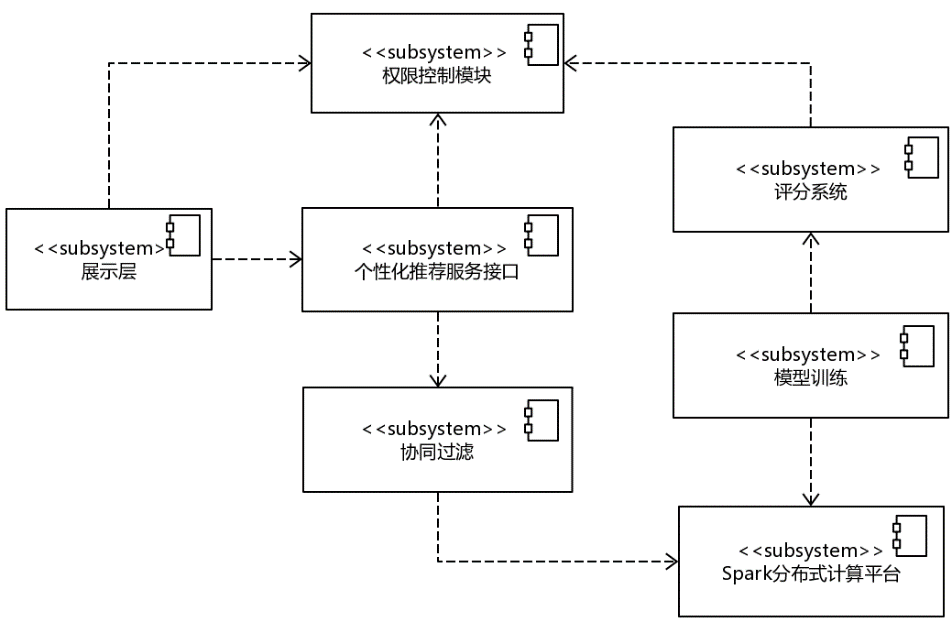


图 4.5 系统开发视图

开发视图如图4.5所示，电影推荐系统主要由权限控制模块、个性化推荐服务模块、协同过滤子系统模块、模型训练模块、评分系统模块和推荐展示层组成。

权限控制模块，任务是区分来自不同用户的功能点操作和页面请求，设置展示页面的访问权限。

评分系统是一个包含前端交互组件和后台持久化数据结构的子系统，负责采集用户对电影的评分数据。评分系统设置了用户访问权限，依赖于权限控制模块。

协同过滤子系统依赖于Spark平台，负责抽取训练好的模型数据，计算每个用户的候选推荐项，并且将计算出的结果持久化到缓存中。

个性化推荐服务是Web后台服务，负责处理业务数据加载、业务数据的推送，向展示层提供服务调用接口。个性化推荐服务依赖于协同过滤子系统中更新的计算结果，以及权限控制模块中的用户信息。

模型训练模块基于Spark平台提供的算法库、数据结构以及资源管理，依赖于评分系统采集的数据，负责生成用户和电影的特征。

### 4.1.5进程视图

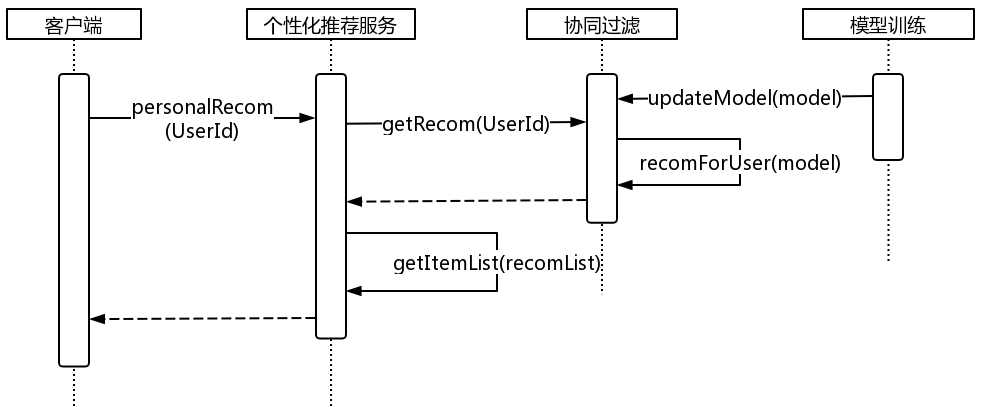


图 4.6 系统进程视图

图4.6为推荐系统的进程视图，个性化推荐服务运行在独立的服务器进程中，协同过滤模块是一个后台进程。一个典型的推荐系统交互由客户端发起，客户端的操作触发调用个性化推荐服务的事件。个性化推荐服务首先从协同过滤系统中查询出针对某一用户的推荐项ID列表，然后再根据ID列表装载业务数据，最后将数据返回给客户端。模型训练进程与其它进程独立，定时地更新推荐系统中的模型。协同过滤系统会根据用户模型和电影模型计算出所有用户的电影推荐项，个性化推荐服务会从中查询出特定用户的推荐项集合。

## 4.2功能模块设计

### 4.2.1权限验证

根据第三章的需求分析，有些页面需要登录后才能访问，也就意味着后台在处理每个需要登录才能发送的请求之前，需要先判断当前用户的登录状态。如果直接在业务逻辑处理的代码中写，一方面，每段代码都要写相同的判断逻辑，这会造成代码的冗余，增加重复的工作量；另一方面，当要给一些功能点增加或者调整权限验证的条件时，需要在业务代码中的多处进行修改，这违反了开闭原则。为了避免上述情况的发生，本文利用Spring的AOP的思想，即通过构造拦截器去处理权限验证问题，提高代码的复用率，并且能实现无侵入式代码扩展，使得权限验证和处理业务逻辑的代码完全解耦和。

如图4.7所示是权限验证的流程图。对于每个前端请求，服务器会通过这次请求对应的Session对象得到用户信息，然后在整个请求处理过程中都保存着这份信息，直到请求处理流程结束才释放。整个访问权限的验证流程中有两个条件判断环节，第一个条件判断是拦截需要登录才能访问的页面的请求，这个通过配置实现；第二个条件判断是拦截没有登录的用户的请求, 若用户未登录就要重定向到登录页面。

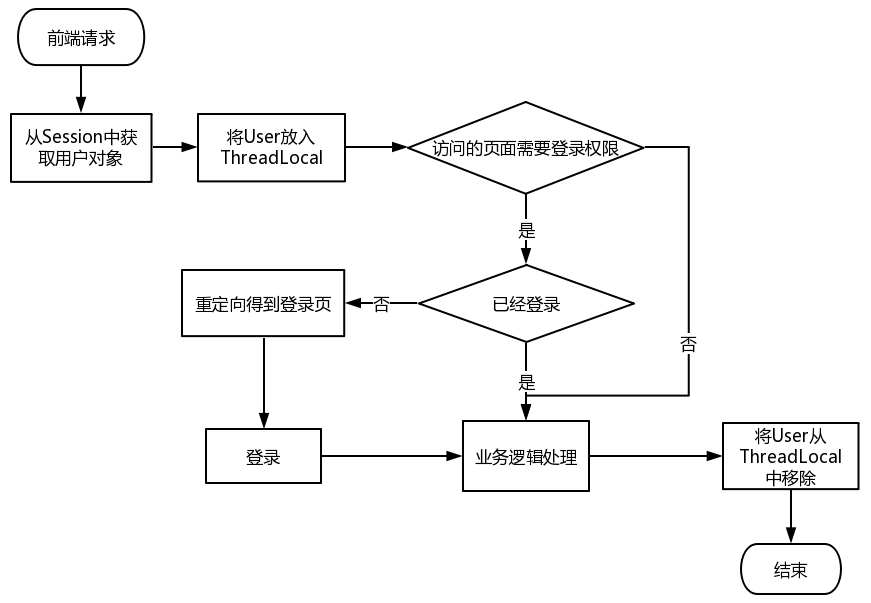


图 4.7 访问权限的验证流程

如图4.8所示，是权限验证模块的类图组织结构，上文图4.7中的程序是通过这些类去完成的。其中AuthActionInterceptor和AuthInterceptor是本文需要实现的类，它们都实现了HandlerInterceptor这个接口，HandlerInterceptor是Spring提供的一个拦截器接口，它的子类将具有拦截器的功能。WebMvcConfigurationSupport是在MVC模式中提供配置的类，本文通过扩展这个类去使用其内部的高级选项，这里需要关注addInterceptors这个方法，通过子类去重写这个方法能达到配置拦截器的目的。WebMvcConf也是本文需要实现的一个类，它继承了WebMvcConfigurationSupport，加上@Configuration注解后才可以真正具备JavaBean配置的功能。在WebMvcConf中，用依赖注入的方式得到AuthActionInterceptor和AuthInterceptor。

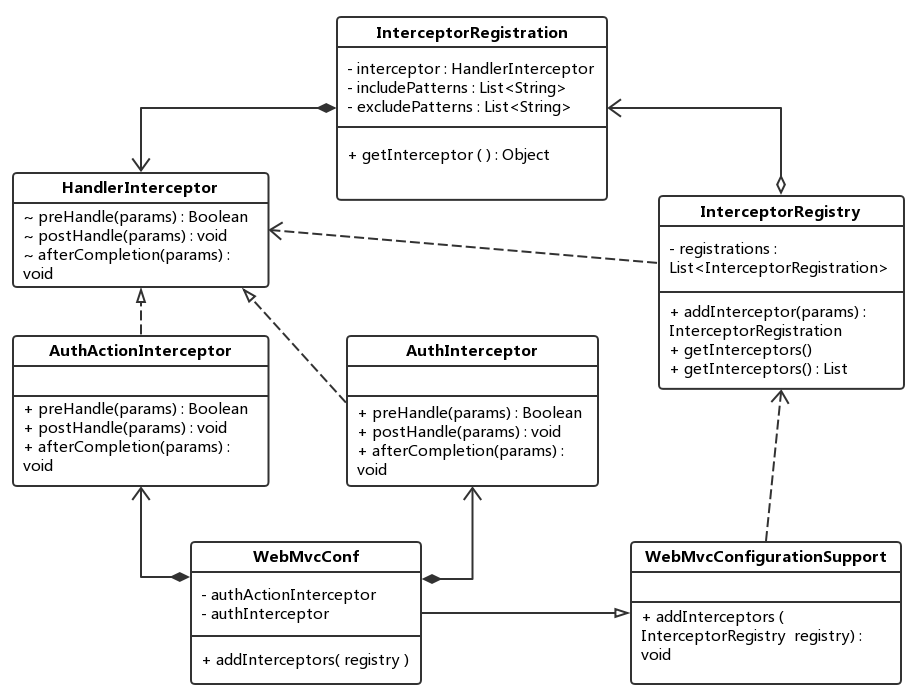


图 4.8 权限验证模块类图

从代理模式的角度看，AuthActionInterceptor和AuthInterceptor这两个类就是本文扩展的具体代理类，它们重写的preHandle方法实现了具体的代理逻辑。在编码时，代理逻辑和业务逻辑是解耦和的，可以相互独立地进行编写，这遵守了开闭原则，使代码具有良好的可维护性和扩展性。

从观察者模式角度看，InterceptorRegistry是主题对象，HandlerInterceptor是观察者。虽然从类图中分析，InterceptorRegistration似乎更像观察者，实际上在这里它只是对HandlerInterceptor的一层封装。当新的HandlerInterceptor的子类实例AuthActionInterceptor被发布到InterceptorRegistry中后，它的拦截器方法不需要在客户端显式调用，InterceptorRegistry起到广播作用。InterceptorRegistry依赖于抽象的HandlerInterceptor接口，与具体的拦截器AuthActionInterceptor和AuthInterceptor是松耦合的关系，这满足开闭原则。

WebMvcConf、AuthActionInterceptor和AuthInterceptor是本文需要实现的类。WebMvcConf充当配置文件的作用，通过InterceptorRegistry的实例将AuthActionInterceptor和AuthInterceptor加入到InterceptorRegistry的拦截器列表中。AuthActionInterceptor和AuthInterceptor都实现的preHandle方法，会在拦截器列表中依次被调用。得益于Spring面向切面编程的机制，本文能实现无侵入式代码扩展。

### 4.2.2个性化推荐

如图4.9所示，推荐引擎定时地、全量地读取MySql中的评分数据，存入HDFS中。接着从HDFS中构造评分数据的RDD，调用Spark的ALS算法训练模型并且将模型存入到HDFS中，最后根据模型数据计算候选推荐项，得到每个用户的推荐结果，并且将结果集存入Redis。影评系统通过Redis与推荐引擎交互，获取推荐项。

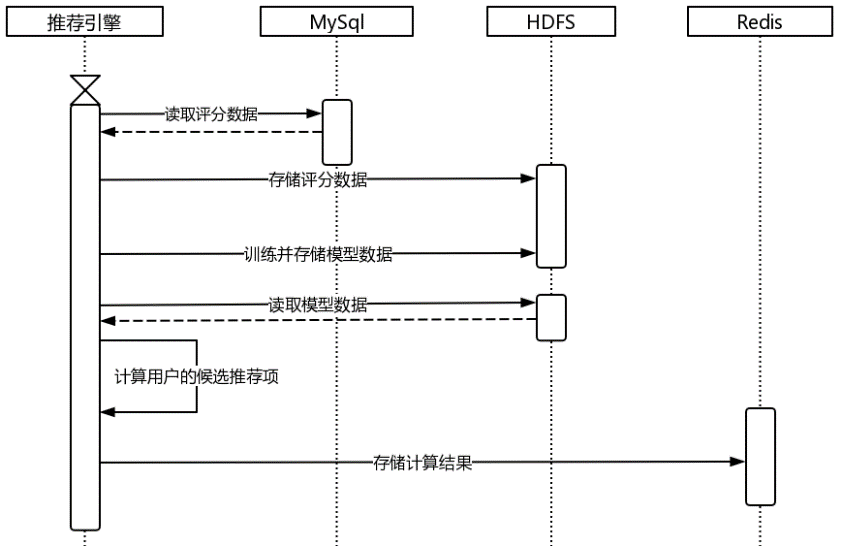


图 4.9 个性化推荐时序图

如图4.10所示是推荐引擎中的数据流图，评分数据从页面输入，存入MySql中，然后依次通过数据抽取、模型训练、产生推荐结果环节，最后通过Redis输出到影评系统。

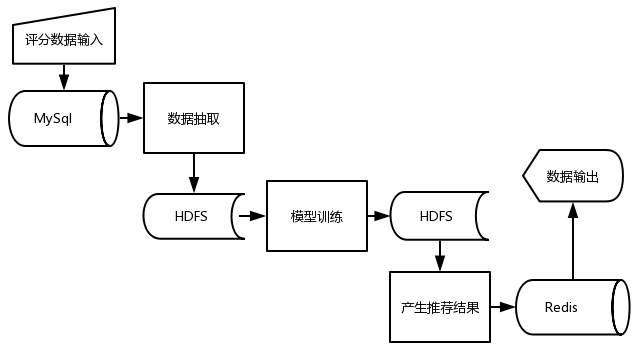


图 4.10 个性化推荐系统

数据抽取环节任务主要是从MySql中查询出评分数据，然后进行预处理，并且存入到HDFS中。原始的评分取值范围是[1,5]，而用户的打分范围是{1, 2, 3, 4, 5}，对于每个用户而言，在显示反馈中最多也只能将对电影的感兴趣程度分成5个等级。比如某个用户对多数电影打分都是4分，然而实际上用户对这些电影感兴趣程度依然是有差距的。因此，系统对用户的评分值的预处理步骤如下。

1. 加载用户隐式反馈数据，计算用户对电影的关注度*C*。具体的做法为查询用户对电影相关的影评的点赞和评论的情况，如果不存在隐式反馈，则*C* = 0；如果存在，设点赞数为*N*，评论数为*M*，点赞的权重为*w*，评论的权重为*y*，最终的关注度计算公式为
2. 遍历并且修改所有的评分数据，将用户对电影的评分值加上用户对电影的关注度，得到最终的评分值。

上述公式利用了关注度区分了相同评分的电影，而且不会破环原来的显示反馈结果，因为关注度的取值范围为[0,1)，评分值的变化量不会超过1，那么原来评分为4的电影即使关注度再高，最终评分值也不会超过5。

模型训练环节是利用Spark平台计算用户和电影的特征向量。主要的步骤包括从HDFS中加载算法的输入数据，运行ALS算法，将模型输出到HDFS中。

产生推荐结果的环节将输出每个用户的电影推荐列表。主要的步骤包括，将每个用户特征与电影特征相乘，得到用户所有评分的预测值，然后针对特定的用户，根据预测值对电影进行排序，得到用户评分最高的前K个电影。这是一个求TopK的问题。

如图4.11(a)所示，将用户特征矩阵和电影特征矩阵相乘，可以得到评分矩阵，矩阵中的某一列就代表某一用户的所有的评分。若要获取TopK，须根据用户对每部电影的评分对电影进行降序排序，返回前K个电影的ID。当然如果没有要求返回的前K个评分值的列表必须是有序的，而仅仅是前K个评分值最高的，可以使用快速排序的思想，找到第K大的评分值时，即可立即返回TopK，因为在快速排序中，轴点左右两边的值是严格大于或者小于轴点值的。

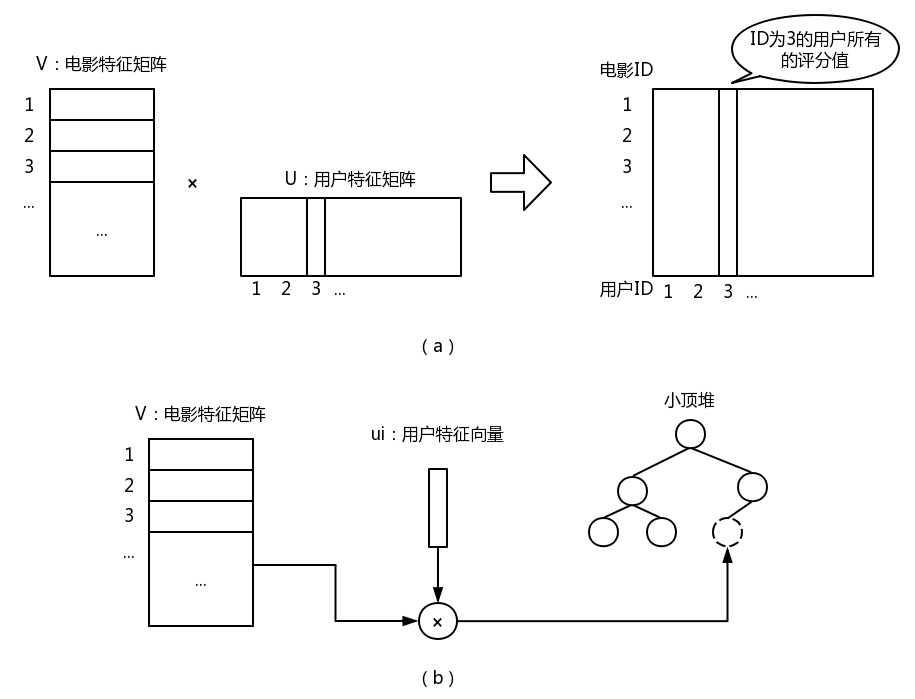


图 4.11 计算推荐项

将评分存储在矩阵中时，使用快速排序方法求TopK的时间复杂度为O(n\*logn)。另一种方法如图4.11(b)所示，为每一个用户建立一个小顶堆来存储评分数据，堆中的每个元素包含电影ID和评分值。具体的步骤如下。

1. 依次扫描电影特征向量；
2. 将电影特征向量与用户特征向量相乘得到评分值；
3. 当堆中元素数量小于K时，将评分封装成堆元素，加入到堆中，否则将评分值与堆顶的评分值比较，如果该元素评分值大于堆顶元素的评分值，则将该元素与堆顶元素替换，重新建立小顶堆；
4. 重复1、2、3步骤，直到扫描完所有的电影特征向量。

完成上述步骤之后，此时堆中的元素就是该用户的TopK个推荐项，再进行排序，时间复杂度为O(k\*logk)。

这两种存储评分值的方式各有优势，前者在计算并存储评分值时更加简单高效，后者取TopK项所需的时间复杂度更低。

个性化推荐系统采用管道过滤器的结构风格。数据抽取、模型训练、计算推荐项任务都是在彼此独立的进程中执行。MySql数据库、HDFS和Redis起到管道的作用，推荐引擎的各个模块是过滤器。使用这种结构有以下特点：

* 易于维护和拓展，若要提高推荐算法计算速度，只需要修改推荐引擎中某个模块即可，如果想要在推荐上做一些拓展，也只需要在推荐引擎的输出部分增加一个过滤器模块，同样的如果想再增加数据预处理操作，也只需要在推荐引擎的输入端增加过滤器。
* 方便系统分析，在测试阶段运行结果不正确可以单独测试每一个或几个模块的运行是否正常。

### 4.2.3热门推荐

热门推荐是解决推荐系统冷启动的一种方案。若要对影评列表进行排序，先为每个影评绑定一个分数代表热门程度。本文利用Redis的有序集合缓存影评或者电影的热门程度分数，通过统计影评的点击频率来计算一个影评的浏览量。

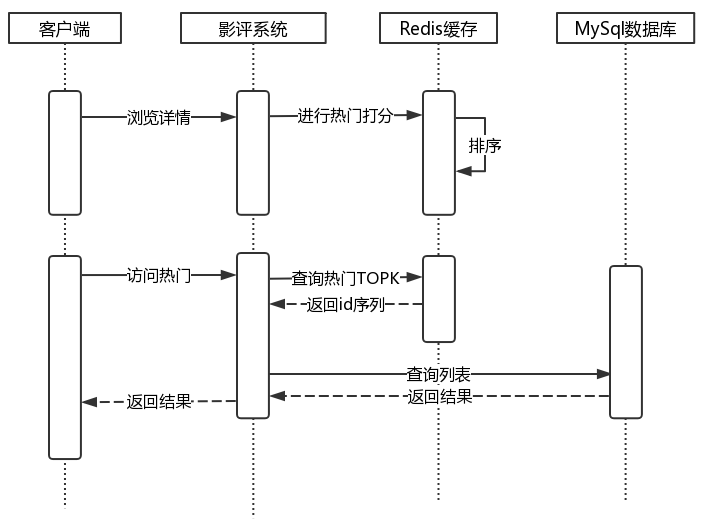


图 4.12 热门推荐时序图

如图4.12所示，若要得到热门推荐的列表，先要从Redis缓存中查到对应的实时排序结果，这个结果就是一个ID序列，然后通过这个ID序列从关系型数据库中查具体的内容。关键的步骤是进行热门程度打分，即估算或者说是刻画热门程度这一属性。

浏览量是一个重要的指标，浏览量越高，说明热度越高。因为热门是实时变化的，而且热门程度会随着时间的流逝慢慢消减，也就是说，如果要持续的在排序中处于较高的位置，那浏览量必须是持续增长的。所以只有总的浏览量是不够的，我们还需要知道浏览是在何时发生的。本文需要构建一个这样的程序，当接收到某个影评的一次浏览时，需要给该影评加分，而加的分数与时间成正比，也就是越靠近当前时间的浏览量，将得到更高的分值。

计算热门程度时，先通过Redis获取当前时间，得到的这个秒数必然是个很大的数，所以可以先用这个数除以一天的秒数86400，得到的是距离UNIX时间的天数。同一天的浏览对热度影响程度相同，将同一天的浏览量乘以当前天数，得到最终的热门程度评分值。假设当前影评的浏览量是*N*，当前时间是*T*，那么该影评的热度*H*可以通过公式计算得到。

### 4.2.4评分功能

对于基于矩阵分解模型的推荐系统，评分功能是很重要的功能，因为评分是显示反馈信息的来源。本文在4.3小节中将介绍数据库表的设计，其中有电影的综合评分和用户的个人评分。综合评分是实时更新的，通过每个用户的评分进行加权评分得到。

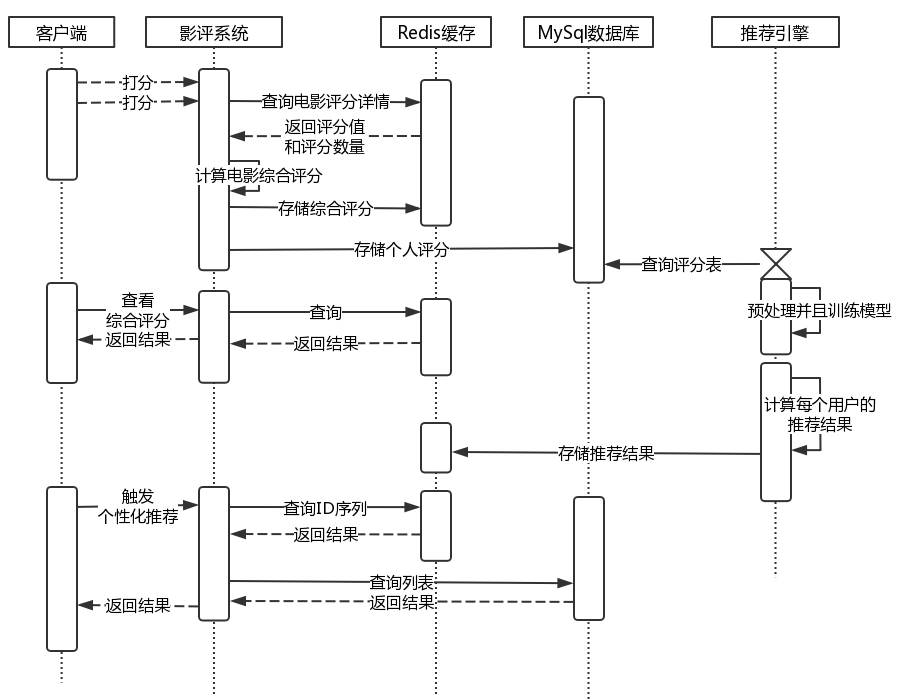


图 4.13 评分时序图

如图4.13所示，首先客户端通过异步请求的方式传递打分数据到影评系统。影评系统会从Redis缓存中先查出电影评分详情，即当前电影被哪些用户评论过，以及每个用户的评分值是多少，然后通过计算一部电影的所有评分的加权平均数，就可以得到这部电影的综合评分。当用户查询电影的综合评分时，直接从缓存中取数据，这并不意味着，MySql中的rating字段是多余的，在数据需要做导入导出的时候这个字段能派上用场。用户的个人评分会存入评分表中，数据的字段将在下文数据库表中给出。通过定时脚本运行推荐引擎，推荐引擎会从MySql数据库中抽取评分信息表，然后进行预处理得到用户-电影的二元矩阵，计算出推荐结果存入到缓存中，用来满足个性化推荐的需求。在图4.13中展示了个性化推荐与评分功能之间紧密的联系。

### 4.2.5社区功能

社区本质上是一个网络日志发布系统，本文的影评本质上就是日志。用户围绕影评的社交行为包括点赞、评论。

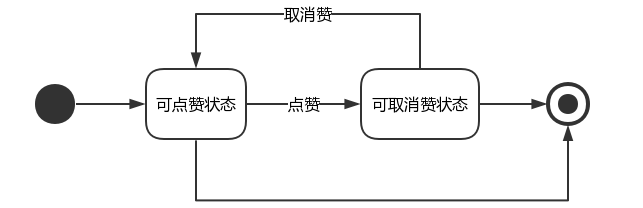


图 4.14 用户点赞活动图

如图4.14所示是点赞功能的活动图。用户对一个影评项至多只能点赞一次，对于每个用户的行为，可以利用前端组件的交互设计进行约束，在每个影评项的详情页面设置一个按钮，初始的状态是可点赞的状态，再次点击可以取消赞。对于每个影评，来自多个用户的点赞操作可以对影评的赞计数器累加，并且在用户的操作之后实时刷新。而在后台，需要设计一个数据结构记录这种活动状态。这种状态是实时变化的，而且这个状态是用要把用户和影评两者关联在一起的。

本文采用Redis的位图结构来存储点赞的状态，使用一个比特位来存储用户对影评的点赞情况，1代表点赞，0代表没有点赞。位图中的每一行代表影评，每一列代表用户。计算某一个影评的点赞数时，只需要计算该影评对应的一行二进制字符串中1的个数。

用户可以在影评下方发表评论，评论下方又能得到用户的回复信息，相当于评论又能被评论，所以影评的评论列表实际上是多个树状结构组成的，每一个评论是树的一个节点，对影评的直接评论处在树的根节点位置，回复的评论可以看成是子节点。一条评论记录不管是直接评论影评的还是回复其它评论的，本质的区别是展示出来的排列位置不同。

当系统根据一个影评的ID查询出该影评所关联的所有评论时，需要将这些评论以正确的拓扑结构组织成评论列表展示出来。

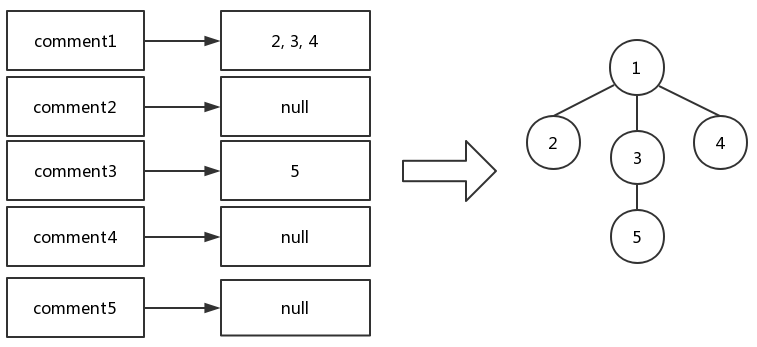


图 4.15 评论列表的数据结构

本文使用Redis存储这种拓扑结构信息，如图4.15所示是评论列表的键值结构，键由字符串“comment”和评论id拼接而成，值为评论的id列表。用深度优先搜索的方式生成每一棵多路树，当然在递归的访问列表中每个id值的子树时，不能忘了加上“comment”前缀去查找对应的键。最终的评论列表实际上是一个“森林”，为了加快查询效率，还可以为每一个影评构建一个列表，用来存储评论列表的每个树根节点。

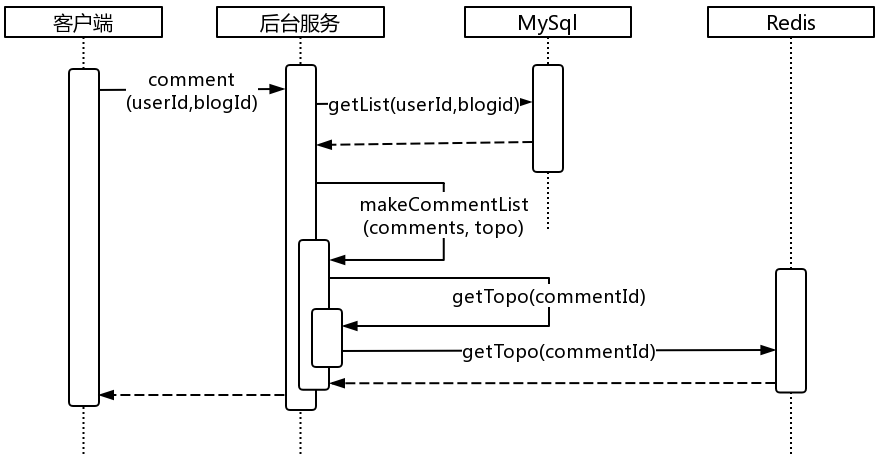


图 4.16 评论时序图

图4.16是评论功能的时序图。客户端调用评论服务，后台首先从数据库中根据用户ID和影评的ID查询出评论列表，然后调用makeCommentList生成评论列表，在该方法中调用getTopo方法递归地遍历出评论列表的拓扑结构。最后将拓扑结构与评论内容整合返回给客户端。

## 4.3数据库设计

### 4.3.1 E-R图

如图4.17所示是影评系统的实体关系模型图，图中忽略了实体的属性，而重点关注于实体之间的关系。尖括号中的名称代表了实体对应的关系型数据库表名，具体的字段描述可在3.4.2节中看到。

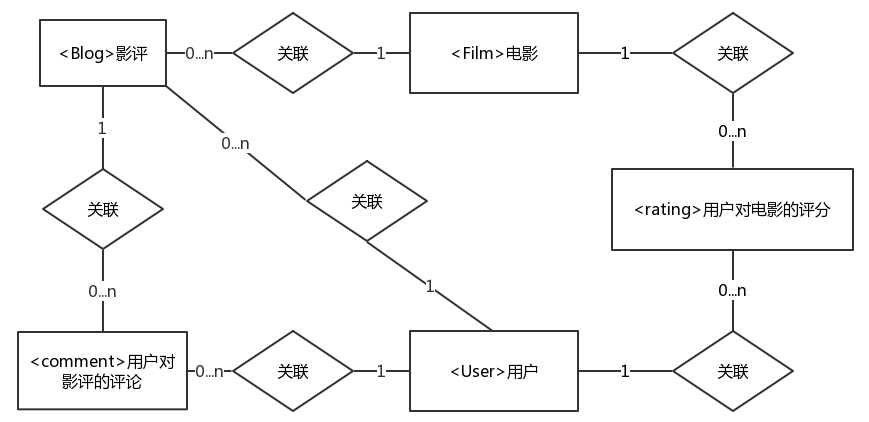


图 4.17 影评系统E-R图

一个用户可以发表多个影评，每个影评只针对某一部电影，而一部电影会被很多人评论，所以可以看到，用户和影评是一对多的关系，电影和影评也是一对多关系。同理，和影评一样，评分也是需要特定的用户和特定的电影唯一确定的实体；评论也一样，需要用户和影评唯一确定。用户和电影在整个实体关系模型中属于核心地位，影评以及评分等实体都在维护着这两者的多对多关联关系。

### 4.3.2 数据库表设计

关系型数据库表展示的是影评系统的数据的逻辑存储结构，与业务逻辑密切相关，良好的设计对于系统稳定性起到了关键作用。

如表4.1所示，在电影信息表中，存储了电影名称、综合评分、主演姓名、电影导演姓名、主题类型、上映年份等字段，其中id是一个自增的整型数字，是表的主键。rating字段作为电影的综合评分，只在电影信息收录进推荐系统后，第一次展示时用到。因为在本系统中，综合评分是一个实时变化的值，所以MySql数据库中的这个rating，在之后的系统运行过程中都不会再起到作用。电影海报image字段存储的是图片的资源定位符，它并不一定存储完整的图片路径，它可以只是一个区分图片的标记，这个标记作为完整路径的后缀，长度可以控制在很小的范围之内。

表 4.1 电影信息表film

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名称** | **数据类型** | **非空** | **备注** |
| id | INT(11) | 是 | 主键，自增 |
| name | VARCHAR(100) | 是 | 电影名称 |
| rating | DOUBLE | 否 | 电影的综合评分 |
| protagonism | VARCHAR(45) | 否 | 电影主演 |
| direct | VARCHAR(45) | 否 | 导演 |
| theme | VARCHAR(225) | 否 | 电影主题、类型 |
| release\_time | YEAR | 是 | 电影上映年份 |
| image | VARCHAR(45) | 否 | 电影海报 |

表4.2中展示了用户信息表，包含了用户姓名、登录密码、电话号码、邮箱号、注册时间等字段，其中id是一个自增的整型数字，是表的主键。邮箱用作登录帐号，在前端输入的时候需要做前置的格式检查。电话号码是定长字符串，在用户注册时需要用来检查其身份的真实性。

表 4.2 用户信息表user

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名称** | **数据类型** | **非空** | **备注** |
| id | INT(11) | 是 | 主键，自增 |
| name | VARCHAR(45) | 是 | 用户姓名 |
| passwd | VARCHAR(45) | 是 | 登录密码 |
| phone | CHAR(11) | 是 | 电话号码 |
| email | VARCHAR(45) | 是 | 邮箱号，用作登录帐号 |
| register\_time | DATE | 是 | 注册时间 |

表4.3展示的是评分信息表，存储了某个用户在某个时刻对某部电影的评分值，其中id是一个自增的整型数字，作为表的主键，user\_id和film\_id是表的外键，分别对应着用户信息表和电影信息表的主键。

表 4.3 评分信息表rating

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名称** | **数据类型** | **非空** | **备注** |
| id | INT(11) | 是 | 主键，自增 |
| user\_id | INT(11) | 是 | 用户信息表主键 |
| film\_id | INT(11) | 是 | 电影信息表主键 |
| value | DOUBLE | 是 | 评分数值 |
| timestamp | TIMESTAMP | 是 | 时间戳 |

如表4.4所示，影评信息表用自增的整型数字id作主键，外键是user\_id和film\_id，包含内容、时间戳、seen字段，其中seen是用于懒惰删除的。seen字段默认是True，表示可见。当用户删除影评的时候，后台只是把seen字段置为False，查询的时候对应的元组会被过滤掉，内容也就不可见了。

表 4.4 影评信息表blog

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名称** | **数据类型** | **非空** | **备注** |
| id | INT(11) | 是 | 主键，自增 |
| user\_id | INT(11) | 是 | 用户信息表主键 |
| film\_id | INT(11) | 是 | 电影信息表主键 |
| content | Text | 是 | 影评内容 |
| timestamp | TIMESTAMP | 是 | 发表时间 |
| seen | BOOLEAN | 是 | 是否可见，默认为TRUE |

如表4.5所示，评论信息表外键是user\_id和blog\_id，它和影评信息表的结构类似，都是用自增的整型数字id作主键，且都包含了内容、时间戳、seen字段，其中内容字段的数据类型不同，影评信息表采用Text类型存储内容，评论信息表采用varchar类型存储内容。

表 4.5 评论信息表comment

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名称** | **数据类型** | **非空** | **备注** |
| id | INT(11) | 是 | 主键，自增 |
| user\_id | INT(11) | 是 | 用户信息表主键 |
| blog\_id | INT(11) | 是 | 影评信息表主键 |
| content | VARCHAR(225) | 是 | 评论内容 |
| timestamp | TIMESTAMP | 是 | 评论时间 |
| seen | BOOLEAN | 是 | 是否可见，默认为TRUE |

## 4.4算法设计

本文采用基于矩阵分解模型的算法，根据用户的评分矩阵求解用户和电影的特征向量。考虑到用户的评分的时效性，本文通过函数将评分的时效映射在特定的数值范围内。

在上述公式中，是系数。*t*是评分的产生时间与当前时间的差值。当*t*值越大，那么对应的评分值在样本中的可信度就越小，表现在矩阵分解的目标优化函数中，即为对特征向量的范数产生削减的效果。

上述式子中，代表训练集中存在的评分项，正则化参数项与评分项的时效函数相乘，为正则化项系数，是用户特征向量，是电影特征向量，是第*i*个用户对第*j*个电影的评分项，即为产生的时间与当前时间的差。使用最交替最小二乘法求解目标函数，先固定电影特征矩阵，求用户特征，再固定用户特征矩阵，求电影特征。

首先对求导，可得：

令偏导数等于0，可得

同理也可以得到

注意，在上述公式中，表示或者对应的所有评分的集合，所以上述公式中的*U*和*V*都不是完整的原始特征矩阵，而是对应评分集合的关联特征集合。*E*是单位矩阵。

算法的详细过程即：

1. 随机数初始化特征矩阵u，v；
2. 固定u，使用上文得到的公式更新v；
3. 固定v，使用公式更新u；
4. 达到最大迭代次数时停止，否则重复2、3步骤。

## 4.5 本章小结

本章分成了三个部分对电影推荐系统进行了分析和设计，第一节对整体的结构进行了划分，大致介绍了系统包含的各个功能模块，详细描述了系统的整体架构、逻辑视图以及物理视图，制定了各个软件模块的行为规范与职责，分析了组件之间的依赖关系。第二节结合需求分析，对各个功能模块进行了分析设计，通过时序图以及类图提出了实现的思路，分析了设计的好处。第三节进行数据库设计，分析了应用层的实体关系模型，设计了MySql数据库中的表结构，对重要的字段做了解释，在第四节中设计了算法的目标优化函数，并且给出了具体解法。

# 

# 第四章 电影推荐系统的实现

## 5.1影评系统的实现

### 5.1.1权限验证

|  |
| --- |
| 1. … 2. HttpSession session = request.getSession(true); 3. User user = (User) session.getAttribute (CommonConstants.USER\_ATTRIBUTE); // 从session中得到用户信息 4. if (user != null) {UserContext.setUser(user);} // 将用户信息加入到线程本地变量中 5. return true; 6. } 7. public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) throws Exception { 8. UserContext.remove(); // 请求处理完成时，将user从ThreadLocal中移除   … |

图 5.1 AuthInterceptor类部分代码

AuthInterceptor类的相关代码如图5.1所示，用它来做了很多预处理，比如将URL请求中的参数置入请求属性键值对中，以及忽略对静态资源的拦截，还有必须要做的就是从Session中获取用户对象（这个对象是用户登录的时候就放入Session中的），如果获取到该用户对象就将其放入到线程局部变量ThreadLocal中。使用ThreadLocal的好处在于能够很高效地使用用户对象，而且能保证线程安全。Session是由SessionFactory负责创建的，SessionFactory的实现是线程安全的，多个并发的线程可以同时访问一个SessionFactory并从中获取Session实例，而Session并不是线程安全的。除此之外，在后续的很多操作中，都需要用到user对象，频繁的创建和销毁Session对象将会是低效的。在第4行的代码中，类UserContext中封装了ThreadLocal，向外提供了user的set和get方法，以及remove方法。

|  |
| --- |
| 1. public boolean preHandle (params…){ 2. User user = UserContext.getUser(); 3. if (user == null) { 4. String target = URLEncoder.encode(request.getRequestURL().toString(),"utf-8"); 5. String message = URLEncoder.encode("请先登录","utf-8"); 6. if ("GET".equalsIgnoreCase(request.getMethod())) { 7. response.sendRedirect("/signin?errorMsg=" + msg + "&target = " + target); 8. return false; 9. }else {response.sendRedirect("/signin?errorMsg="+msg); 10. return false;} 11. } 12. return true; |

图 5.2 AuthActionInterceptor类部分代码

类AuthActionInterceptor的相关代码如图5.2所示，在第2行中，程序从UserContext中取出用户信息，判断当前请求是否是在用户登录之后的请求，如果是，第3行中的判断语句会返回false，代表用户已经登录，程序进入业务逻辑处理环节，如果不是就要重定向到登录页面。

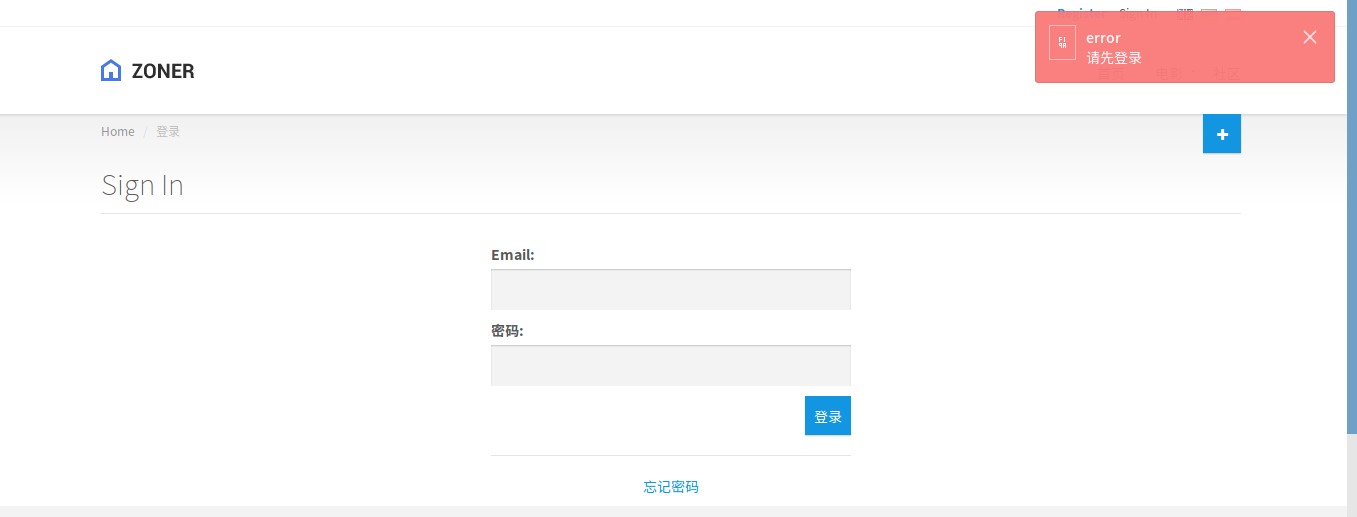


图 5.3 非法操作跳转至登录页面

如图5.3所示是用户未登录时的非法操作（比如访问个性化推荐信息或者使用社区的功能）的场景，页面跳转到登录页，并且在右上角弹出了提示信息。

### 5.1.2热门推荐

如图5.4所示是RecommendService的部分代码，RecommendService类提供了个性化推荐和热门推荐的功能接口。

|  |
| --- |
| 1. public class RecommendService { 2. private static final String HOT\_FILM\_KEY = "hot\_film"; 3. /\*\* 点击触发热门打分 \*/ 4. public void increase(Integer id) { 5. try { 6. Jedis jedis = CacheUtil.jPool.getResource(); 7. jedis.zincrby(HOT\_FILM\_KEY, 1.0D, id + "");//每次点击详情页会调用该方法,使得相应的电影或者影评对应的分值自增 8. … 9. } 10. /\*\* 热门推荐 \*/ 11. public List<Film> getHotFilms(Integer size) { 12. try { 13. Jedis jedis = CacheUtil.jPool.getResource(); 14. jedis.zremrangeByRank(HOT\_FILM\_KEY, 0, -1 - size); //去除有序集合中的除末尾size个元素之外的所有元素，即得到分值最高的size个元素。 15. Set<String> idSet = jedis.zrevrange(HOT\_FILM\_KEY, 0, -1); 16. jedis.close(); 17. List<Long> ids = idSet.stream().map(Long::parseLong).collect(Collectors.toList()); 18. List<Film> fSet = query(ids,size); 19. Film[] array = new Film[fSet.size()]; 20. for(Film film : fSet){ 21. final int idx = ids.indexOf(film.getId().longValue()); 22. array[idx] = film; 23. } 24. return Arrays.asList(array);   ... |

图 5.4 热门推荐

代码第4行的increase这个方法会在用户访问推荐项的详情页面的时候被调用，使得相应的推荐项在有序列表中的分值提升。因为在Redis的有序集合中，元素是按照分值升序排列的，所以若要获取分值最高的n个元素，只需要截取末尾n个元素。在代码第11行所示的getHotFilms方法中，根据传递的size值取元素，得到这些id序列后再去查MySql数据库，但是这时候查出来的数据列表是按照推荐项在MySql数据库中的id值大小排序的，所以在业务层还需要再根据Redis中的排序重新整理数据列表。在代码第19行到23行开始程序对列表进行重排序，首先初始化一个数组array，遍历数据列表中id在Redis排序中所在的位置idx，然后把该id对应的详细数据放入数组对应的下标为idx的元素中。最后得到整理好的数组array用List封装并返回结果集。

如图5.5所示是热门推荐的效果图。

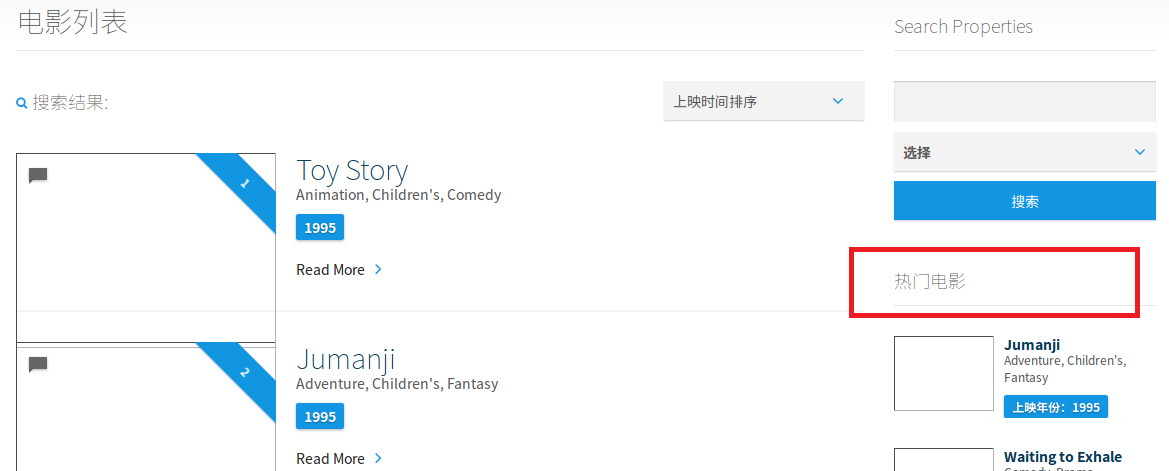


图 5.5 热门推荐列表

解决乱序问题还有另一种看起来更方便的方法，就是给关系数据库表中添加一个字段score，用这个存热门程度的分值，然后就不需要在业务层重新排序，只需要在SQL查询语句中添加Order By Score即可。但是本文不采取这种方法，原因有两点：

1. Score不应该成为业务数据建模的一部分。如果采取添加字段的方式，那么所有应用到热门推荐功能的数据库表都要添加这个字段，比如本文就有三个表用到热门推荐的逻辑，分别是Film、Blog和Comment。由于关系型数据库MySql物理存储并不是按列存储的，所以进行数据库结构改变在生产环境下通常是不可行的。本文没有改变数据库表的结构，也避免去改变已有的代码，只是添加了处理排序逻辑的代码。
2. 热门排序是实时变化的，每种推荐项的分值都是频繁改变的，每次都将新改变的分值插入到MySql中是没必要的。MySql实际上的排序操作也无非是这样在内存中进行，并没有本质区别，但是本文的做法减少了数据库连接的建立与撤销，以及一些磁盘开销，在高并发的或者大规模的请求中效率是更高的。

### 5.1.3综合评分

类FilmService完成了对电影的评分以及查询电影综合评分的功能，其中实现评分功能的部分代码如图5.6所示，第7行将用户对某部电影的评分数据存入此电影对应的Hash表中，以用户id为键，用户评分为值，然后在第8到11行中，先取出所有关于该电影的评分值，接着进行加权平均，得到综合评分值，存入“avg”对应的Hash表中。

|  |
| --- |
| 1. /\* 评分 \*/ 2. public void updateRating(Double rating, final Integer id) { 3. User user = UserContext.getUser(); 4. final Long uid = user.getId();//本文会对调用updateRating方法的请求进行鉴权操作，所以这里不需要判断用户是否为空，而只需要直接取值。 5. final String fk = "film" + id; 6. Jedis jedis = CacheUtil.jPool.getResource(); 7. jedis.hset(fk, uid.toString(), rating.toString()); 8. List<String> list = jedis.hvals(fk); 9. Double sum = list.stream().map(Double::parseDouble).reduce(0.0, (a, b) -> a + b); 10. Double avg = sum / jedis.hlen(fk); 11. jedis.hset("avg",fk,avg.toString());   … |

图 5.6 评分功能的实现

图5.7所示是Redis中“avg”对应的Hash表结构以及电影评分表的存储结构，为了与用户id区分开来，电影评分表将前缀“film”加上电影的id作为键来关联对应的Hash表。

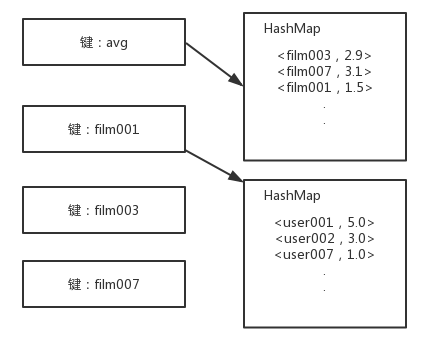


图 5.7 综合评分数据结构

相对于关系型数据库，使用Redis使得计算更简单，得益于散列表的存储结构，新增或者修改评分时不需要先查询原来的表中是否存在该用户的评分，因为散列表本身就保证了键（在这里是字段）的唯一性。某部电影所关联的散列表长度就代表对该电影进行过打分的用户的数量。若要得到特定电影的所有评分也很简单，不需要像关系型数据库那样进行表的连接操作或者遍历整个表，可以通过命令直接获取散列表中所有的值。如图5.6所示的代码中，第8行通过电影id可以直接得到该电影所有的评分值。

此外，为了使得详情页面直接从缓存中取得综合评分数据，本文在Redis中设置了avg键，代表由平均数得到的电影的综合评分值。

如图5.8所示，查询出综合评分的代码变得很简洁，第6行和第7行中只需要访问Redis的“avg”键即可，而不用遍历整个MySql数据库表去查找Rating表中关于特定电影的所有评分。调用detailFilm方法的开销在常数范围内。

|  |
| --- |
| 1. /\* 详情 \*/ 2. public Film detailFilm(Integer filmId) { 3. Film film = filmMapper.getOne(filmId);//先直接从数据库中查询 4. Jedis jedis = CacheUtil.jPool.getResource(); 5. final String fk = "film" + film.getId(); 6. if(jedis.hexists("avg",fk)){ 7. String avg = jedis.hget("avg",fk); 8. film.setRating(Double.parseDouble(avg));//如果缓存中存在avg对应的字段，优先使用缓存中的。 9. }   … |

图 5.8 查询综合评分

## 5.2推荐引擎的实现

### 5.2.1模型训练

本文采取了Spark实现模型的训练，由于算法进行的是全量式的计算，所以本文采用批处理的方式运行推荐引擎。在linux的环境下，编辑好运行程序的shell脚本，并把这个脚本加入到定时任务中。

如图5.9所示，向Spark提交一个通过maven编译打包好的jar文件recommender.jar，recommender是模块名称，运行其中的CollaborativeFilter类。由于代码中有引入jedis，所以在提交的时候需要在~/lib/目录下放入jedis的jar包，并且在脚本中加上--jars ~/lib/jedis.jar。

|  |
| --- |
| cd /usr/local/spark/bin/  ./spark-submit --jars ~/lib/jedis.jar --class "edu.graduate.recommend.cf. CollaborativeFilter" ~/workplace/graduate/recommender/target/recommender.jar |

图 5.9 定时脚本

如图5.10所示，是模型训练模块的核心代码。首先要从评分数据中剥离出评分矩阵和时间矩阵，然后对时间戳进行预处理。最后将评分和时差作为算法的输入进行模型的训练。将得到的结果存入HDFS中。第14行中*g*方法是评分的时效函数，w是时效函数的系数。第12行中的常量d将作为第14行代码中时差的除数，其作用是将时差的数值缩小，86400×60其含义是60天，因为86400是一天的秒数，那么最终带入时效函数*g*中的变量*t*就代表，评分时间距离当前有几个60天。第18行中的myAls方法是本文实现的ALS的改进版本，本文将在下一节介绍此算法的详细实现。

|  |
| --- |
| 1. //通过sc读取HDFS中的评分数据ratings.dat 2. val input = sc.textFile("data/ratings.dat").map(line => { 3. val fields = line.split("\t") 4. val rating = Rating(fields(0).toInt, fields(1).toInt, fields(2).toDouble) 5. val timestamp = TimeGap(fields(0).toInt, fields(1).toInt, fields(3).toDouble) 6. (rating, timestamp) // 从评分数据中得到评分矩阵和时间戳矩阵 7. }) 8. val ratings = input.map(x => x.\_1) // 评分矩阵 9. val times = input.map(x=>x.\_2)// 时间戳矩阵 10. val ct = system.current.time 11. // timeGap预处理 12. val d = 86400×60 // 由于时间戳数值过大，d值作为时差的除数 13. val timegap = times.map( t => { 14. t.\_3 = g( (ct - t.\_3) / d, w ) 15. t 16. }) 17. // 使用改进的算法进行模型训练 18. Modle m = myAls.train(ratings, timegap,lambda, maxiter, rank) 19. m.save(“/model“) |

图 5.10 模型训练

### 5.2.2TopK推荐

如图5.11所示，训练好模型后，从模型中的提取评分数据。其中第8行中recommend方法返回了特定用户user的TopK项最感兴趣的电影ID列表。第11行中根据用户ID和电影ID找到模型中对应的特征向量，相乘得到评分预测值。小顶堆heap的节点由电影ID和评分值组成。

|  |
| --- |
| 1. val users = ratings.map(x => x.user).distinct() // 得到所有用户ID 2. removeOld(model, ratings) // 将用户评分过的电影的特征向量剔除 3. for(u <- users){ 4. //遍历，给每个用户推荐5个最感兴趣的电影 5. val result = recommend (model,u,5).map(x => x.product.toString) 6. CacheUtil.persist(u,result) // 将结果存储到Redis中 7. def recommend(model,user,k): // 推荐TopK项 8. val films = ratings.map(x => x.product).distinct() // 得到所有的电影ID 9. for v in films: // 遍历每一部电影 10. r = model.find(v) \* model.find(user) // 评分值 11. node = new Node(r,v) // 堆节点 12. while(heap.size < k): 13. heap.add(node) 14. if(r > heap.root.rating): 15. heap.root = node // 堆顶用node替换 16. heapify() // 调整堆，O(logk) 17. sort(heap) 18. return heap.keys.toArray // 返回堆中的电影ID数组 |

图 5.11 TopK推荐

## 5.3算法实现与分析

### 5.3.1数据集

为了验证算法效果，本文采用MovieLens上的，最后更新时间为2018年9月的数据集进行算法的验证。数据集中包含10万条评分数据，是600多名用户对约9000部电影的评分以及评分时间，其中包含的最早的评分值的评分时间是2000年7月。

本文将原始数据集以8:2的比例划分为训练集和验证集。按照时间顺序划分，验证集为距离当前时间最近的20%的数据，剩余的为训练集。

### 5.3.2算法实现

如图5.12所示的伪代码是在Spark并行化环境中，基于矩阵分解模型改进后的算法。在原有的ALS算法的基础上进行修改，在没有改变原有算法主体框架的基础上，增加了时差矩阵Rtm作为算法输入。计算Rtm中第i行第j列元素值的步骤是，首先算出用户i对电影j最后打分的时间与当前时间之差，然后将差值代入时效函数，计算出值。

程序从第11行开始执行，第1行中定义了compute方法，遍历目标矩阵，每次循环中计算出一个目标向量。第8行中λ是正则化项系数，EquationSolver通过solve方法求解线性方程组，EquationSolver是一个代理类，在Spark中，默认采用Cholesky分解法求解线性方程组。第15行中maxiter是最大的交替求解的迭代次数。在第4行至第8行中，实际的代码里写得很冗长复杂，这是为解决算法运行效率而沿用Spark中解决通信开销的方法，这已在第二章中描述过。这里暂且抛开通信开销模型，把算法的主要实现思想表现出来。

|  |
| --- |
| // 输入：评分矩阵Ratings，评分时差矩阵Rtm  // 输出：矩阵分解模型，即用户特征矩阵U，电影特征矩阵V   1. def compute(des, src, Rb, Tb): 2. ne = new normalEquation // 初始化正规方程组 3. foreach fac in des: 4. (srcfs,ratings) = fac.join(src).join(Rb) // 得到计算目标的所有源特征值和评分值 5. ne.add(srcfs,ratings) 6. timesgap = ratings.join(Tb) // 得到评分项的时差集合 7. timeloss = timesgap.reduce(\_+\_) 8. des = EquationSolver.solve(ne, timeloss \* λ) // 调用BLAS解方程组 9. return des 10. partitions = N // RDD分片数量 11. sc = Spark.Context(partitions) 12. 用随机数填充U和V 13. 通过sc，Ratings、U、V、Rtm的RDD，分别为Rb、Ub、Vb、Tb 14. for t in range(1, maxiter): 15. Ub = compute(Ub, Vb, Rb, Tb) 16. Vb = compute(Vb, Ub, Rb, Tb) 17. return (Ub,Vb) |

图 5.12 算法伪代码

### 5.3.3评估标准

本文采用均方误差MSE来作为评测标准，通过计算预测值与真实值之间的误差反映评分预测的准确性。MSE越小，准确度越高。

上述公式中表示用户*u*对物品*i*的真实评分，是预测值，代表所有的评分，即评分项总数。

### 5.3.4实验分析

本文需要验证的是，在时间跨度较大的数据集中，根据时间对评分的影响，使得改进之后的算法能够训练出比原来更好的模型。为了叙述方便，本文将改进后的算法称为ALS-T。

首先，设置参数rank，maxiter，lambda，然后分别使用ALS和ALS-T对特定数据集进行训练。特别的，还需要对ALS-T中时效函数的系数进行设置。对于ALS算法而言，枚举参数的值，循环遍历计算，通过比较MSE即可得出最佳的模型。表5.1展示了误差最小的8组实验数据，其中记录了模型训练的参数设置以及均方误差。在第4条记录中，误差最小达到0.7882，其中 MSE保留4位有效数字。最大迭代次数maxiter统一设置成20，在实验过程中发现，rank和lambda参数不变的情况下，再把maxiter调整更大，MSE变化范围都在0.001左右，对误差影响较小。

表 5.1 ALS参数与均方误差

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Rank | Lambda | Maxiter | MSE |
| 1 | 10 | 0.1 | 20 | 0.8014 |
| 2 | 10 | 0.2 | 20 | 0.8329 |
| 3 | 10 | 0.3 | 20 | 0.9167 |
| 4 | 20 | 0.1 | 20 | 0.7882 |
| 5 | 20 | 0.2 | 20 | 0.8333 |
| 6 | 20 | 0.3 | 20 | 0.9168 |
| 7 | 30 | 0.1 | 20 | 0.7883 |
| 8 | 30 | 0.2 | 20 | 0.8333 |

然而对于ALS-T而言，首先需要对系数取值。如图5.13所示，是将rank、lambda、Maxiter分别设置为20，0.1，20的情况下，调整所得到的均方误差变化趋势，误差随着增大而增加。当把设置为0时，ALS-T退化为ALS算法，反而能得到最小的误差0.7882。而把分别设置成0.1，0.2，0.3，0.4时，误差却变成0.9964，1.214，1.512，1.547。分析其可能的原因，首先评分时效函数的特点，当和*t*大于零时，函数值小于1，再加上*t*的值本身也比较大，所以导致函数值是一个很小的小数，第一位有效数字大概在小数点后面的第6位，然后lambda一般也设置为0.1或者更小的数，两者相乘得到的数的第一位有效数字大概在小数点后面的第7位，这样的一个数作为正则化项系数相当于把正则化项的作用完全消除了。综合考虑以上因素，分析出导致这样的结果的原因可能是值设置过大。

图 5.13 误差随变化图

根据时效函数特征，以及上述实验的结果，将设置为0.01，0.02，0.03，0.04，将rank、lambda、Maxiter分别设置为20，0.1，20。如图5.14所示是调整所得到的均方误差变化趋势，效果有所改善。所以应该尽量将设置得小点，使得时效函数尽可能靠近1，但是又能对评分的时效性进行较好地映射。

图 5.14 误差随变化图

如表5.2所示，沿用ALS的较好的参数设置，对当前数据集进行训练，通过调整的值进行试验，发现将设置为0.006时，能得到较好的结果。

表 5.2 ALS-T参数与均方误差

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Rank | Lambda | Maxiter | MSE |
| 1 | 10 | 0.1 | 20 | 0.7350 |
| 2 | 10 | 0.2 | 20 | 0.7439 |
| 3 | 10 | 0.3 | 20 | 0.7882 |
| 4 | 20 | 0.1 | 20 | 0.7400 |
| 5 | 20 | 0.2 | 20 | 0.7406 |
| 6 | 20 | 0.3 | 20 | 0.7516 |
| 7 | 30 | 0.1 | 20 | 0.7444 |
| 8 | 30 | 0.2 | 20 | 0.7399 |

如图5.15所示，是8组实验结果的误差对比图，ALS-T的误差普遍比ALS缩小了0.1左右，有一定的优化效果。最小误差之间的比较，ALS-T的第一组实验结果的误差是0.7350，和ALS的0.7882相比，优化数值在0.05以上。

图 5.15 误差对比图

从上述的实验结果中，证明了在特定的数据集中，存在，使得ALS-T能训练出比ALS误差更小的模型。

## 5.4本章小结

本章围绕个性化推荐功能，通过代码以展示了业务层面的主要功能的详细实现，包括应用层的权限鉴定、热门推荐、综合评分，在计算层介绍了引擎运行方式，详述了模型训练过程，其中包括对评分以及时间戳进行预处理，此外还介绍了TopK项推荐，然后对ALS-T算法进行了详细的实现和实验分析。

# 第六章 系统测试

## 6.1系统测试环境

表 6.1 系统测试环境

|  |  |
| --- | --- |
| **设备与软件** | **备注** |
| 硬件环境 | 单核，2G内存，40G云盘，2M宽带 |
| 操作系统 | Ubuntu kylin14 |
| 数据库 | Redis，MySql |
| 安装软件 | JDK1.8，Spark，Hadoop，Crontab |
| 部署程序 | Spring Application，推荐引擎Jar包 |

系统环境如表6.1所示，在服务器上须安装配置好Redis、MySql、JDK、Hadoop、Spark等软件，其中Crontab是用于运行定时的shell脚本的。部署时，现在本地集成开发环境用Maven编译好Spring Application，以及推荐引擎Jar包，然后发送到云服务器上，推荐引擎放在定时脚本指定的路径中。Spring Application使用Java命令运行，内置的Tomcat、数据库连接池会一并启动。确保Redis、Hadoop、Mysql是运行状态，然后可以通过浏览器访问服务器。

## 6.2功能测试

### 6.2.1鉴权功能测试

1. 功能描述：对于配置了需要登录才能访问的URL，将会被拦截器处理。
2. 测试目的：验证拦截器的访问控制功能。
3. 测试场景：
   1. 不登录账户，在拦截器中去除个性化推荐的URL配置，重新运行系统,点击个性化推荐按钮；
   2. 不登录账户，在拦截器中加上个性化推荐的URL配置，重新运行系统，点击个性化推荐按钮；
   3. 登录账户，在拦截器中去除个性化推荐的URL配置，重新运行系统,点击个性化推荐按钮；
   4. 登录账户，在拦截器中加上个性化推荐的URL配置，重新运行系统，点击个性化推荐按钮；
4. 预计结果：

a页面和后台均抛出异常，b跳转到登录页面，并且出现提示信息，c、d均无异常。

1. a、b、c、d操作结果与预计结果一致。

### 6.2.2热门推荐功能测试

1. 功能描述：对于浏览次数更多的电影或者影评，在热门推荐列表中排在更高位。热门列表展示页面最多展示5部电影或者5篇影评。
2. 测试目的：验证热门推荐的功能。
3. 测试场景：
   1. 删除Redis数据库中的hot\_film键，随机访问ID = X的电影的详情页面，查看热门推荐列表；
   2. 删除Redis数据库中的hot\_film键，随机访问ID = X、Y、Z的电影的详情页面各一次，查看热门推荐列表；
   3. 删除Redis数据库中的hot\_film键，随机访问ID = X、Y、Z的电影的详情页面，访问X三次，Y两次，Z一次，查看热门推荐列表；
   4. 访问X三次，Y两次，Z一次之后，连续访问M四次，每次访问完查看热门推荐列表；
   5. 删除Redis数据库中的hot\_film键，随机访问十部电影，访问次数随机，查看热门推荐列表；
4. 预计结果：

a中热门推荐列表出现ID=X的电影，b中 X、Y、Z按照点击先后排序，c中列表排序依次是X、Y、Z，d中M依次出现在Z之后、Y之后、X之后，最后出现在表头，e中推荐列表中出现五部访问量最高的电影，并且按照访问量从高到低排序。

1. a、b、c、d、e操作结果与预计结果一致。如图6.1所示，在随机地对ID为1、3612、3、4、2访问若干次后，在Redis的控制台中可以查到对应的排序信息。

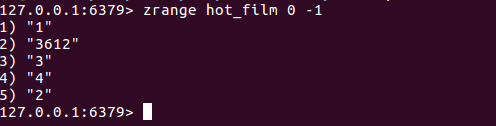


图 6.1 Redis控制台的热门排序

### 6.2.3评分功能测试

1. 功能描述：用户通过点亮星星，对电影打分，评分范围是1到5分。
2. 测试目的：验证用户的打分功能。
3. 测试场景：
   1. 登录账户U，随机选择电影X，点击进入详情页面进行评分，打分值为R1，查看数据库中评分信息表中的数据；
   2. 登录账户U，随机选择电影Y，点击进入详情页面进行评分，打分值为R2，查看数据库中评分信息表中的数据；
   3. 登录账户S，随机选择电影X，点击进入详情页面进行评分，打分值为R3，查看数据库中评分信息表中的数据；
   4. 登录账户S，随机选择电影Y，点击进入详情页面进行评分，打分值为R4，查看数据库中评分信息表中的数据；
   5. 登录账户U，随机选择电影Z，点击进入详情页面进行评分，打分值为R1，然后重新打分为R2，查看数据库中评分信息表中的数据；
   6. 登录账户U，随机选择电影Z，点击进入详情页面进行评分，连续点击打分按钮N次，查看数据库中评分信息表中的数据；
4. 预计结果：a中在film数据库中可以看到rating表中插入了用户U对电影X评分为R1，b中在film数据库中可以看到rating表中插入了用户U对电影Y评分为R2，c中在film数据库中可以看到rating表中插入了用户S对电影X评分为R3，d中在film数据库中可以看到rating表中插入了用户S对电影Y评分为R4，e中在film数据库中可以看到rating表中插入了用户U对电影Z评分为R2，f中在film数据库中可以看到rating表中插入了用户U对电影Z评分为最后一次的打分。
5. a、b、c、d、e、f操作结果与预计结果一致。

### 6.2.4综合评分功能测试

1. 功能描述：影片的综合评分等于该该影片的所有个人评分的平均数。
2. 测试目的：验证综合评分功能。
3. 测试场景：
   1. 登录账户U，选择电影X，点击进入详情页面进行评分，打分值为2，刷新详情页，查看电影综合评分；
   2. 登录账户U，选择电影Y，点击进入详情页面进行评分，打分值为2，登录账户S，选择电影Y，点击进入详情页面进行评分，打分值为3，刷新详情页，查看电影综合评分；
   3. 登录账户U，选择电影Z，点击进入详情页面进行评分，打分值为2，登录账户S，选择电影Z，点击进入详情页面进行评分，打分值为3，登录账户E，选择电影Z，点击进入详情页面进行评分，打分值为4，刷新详情页，查看电影综合评分；
   4. 登录账户U，选择电影Y，点击进入详情页面进行评分，修改打分值为4，登录账户S，选择电影Y，修改打分值为5，刷新详情页，查看电影综合评分；
4. 预计结果：

a中电影X综合评分为2，b中电影Y综合评分为2.5，c中电影Z综合评分为3，d中电影Y综合评分为4.5。

1. a、b、c、d操作结果与预计结果一致。如图6.2所示，点击评分之后再刷新页面，在电影详情的评分栏可以看到实时的变化。



图 6.2 评分功能运行效果

对比Redis数据库查询结果，综合评分值avg与页面显示一致。如图6.3所示，ID为1、2、3、4、3612、2765的电影有评分记录，评分来自随机的若干用户，评分的值与预计结果一致。

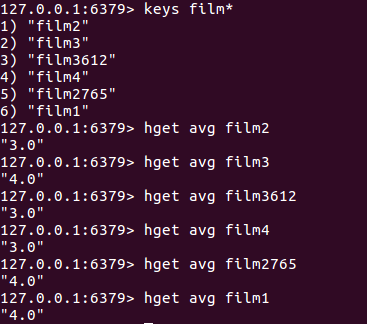


图 6.3 Redis控制台综合评分信息

## 6.3本章小结

本章的主要内容是对系统的主要功能进行的测试，意在验证功能的正确性，确保系统的可用性。第一部分介绍了测试开发的环境，第二部分描述了各个功能点的测试场景，预期结果以及实际测试结果。

# 第七章 总结与展望

## 7.1 总结

在线影院的流行使得电影信息过载，本文在这样的背景下采用Spark、Redis等技术栈实现了一个小型的电影推荐系统。Spark中提供了大规模数据协同过滤的工程化解决方案，本文基于其中的机器学习库，研究和分析了个性化推荐系统的实现原理，设计了一种个性化推荐应用方案，同时针对数据时效性设计了新的目标函数，实现了ALS-T算法。针对用户冷启动问题，实现热门推荐功能，解决新注册的用户获取推荐项的问题。此外，本文在推荐系统中引入了社会化推荐的思想，社区也是解决冷启动的一种方法，用户可以发布影评，互相点赞关注，通过社交网络使得新注册的用户可以在影评中有所发现。

## 7.2 进一步工作展望

本文基于Spark的ALS模型设计了评分系统，通过用户的显示反馈信息进行了协同过滤，兼顾用户和电影两个方面的因素来得到预测结果。然而，由于ALS算法本身的特点，推荐引擎只能进行批处理计算，无法做到实时、增量式的计算。

评分系统设计得过于理想化，通过用户评分得到的效用矩阵可能会是非常稀疏的，而且具有误导性，因为每个人的评分基准是不一样的，有的人可能多数评分都是一样的，偏好程度无法区分开来，而且两个人打同样的分实际上也并不是同等程度的喜好。

算法方面，推荐质量还不够好。时效函数的系数很难确定，需要调整目标函数的设计。

影评系统在社会化推荐的效果上发挥不足，社交数据没有参与到模型训练当中，而且也没有对效用矩阵起到修正作用。而且影评没有好的推送方式。

以上所述都是本文在研究之初考虑不周的地方，针对以上不足，在接下来的工作中将在以下几个方面做出改进：

1. 研究适合进行增量式计算的推荐算法，采用Kafka、Flume、Spark Streaming等流式数据处理技术实现数据抽取与挖掘。
2. 充分利用隐式反馈数据，比如点赞关注以及浏览情况，修正显示反馈数据（用户评分信息），或者设计矩阵分解的损失函数，根据用户的点赞关注情况得到用户间的相似度，在函数中加入社会正则化项[王雪婷，2014]。
3. 研究影评推送的算法，精准推送或者编排影评，比如采用LDA模型[Blei D M，2003]来分析影评的文本信息，得到该影评的主题，再挖掘其它的情景信息，然后分析用户在特定情境下感兴趣的主题，将相应的影评推送给目标用户。

# 参 考 文 献

**[**Blei D M，2003**]** Blei D M，Ng A Y，JonLan M I，Lantent D irichletA llocation[J]. Journal of Machine Learning Research，2003，3；993—1022.

**[C. Anderson,2006]** C. Anderson，The Long Tail: Why the Future of Business is Selling Less of More，*Hyperion Books*，New York，2006.

[Collaborative Filtering, 2019] [http://spark.apache.org/docs/latest/ml-collaborative-filtering.html](http://spark.apache.org/docs/latest/ml-collaborative-filtering.html%20), [**Collaborative Filtering**](http://en.wikipedia.org/wiki/Recommender_system#Collaborative_filtering) maintained by spark.apache.org, 2019.

[Freyne,2011] Freyne J．Anand SS，Recommender systems and the social Web，*Proc. of the ACM RecSys*，NewYork：ACM Press，2011，383-384.

[Jannach,2009] Jannach D，Geyer W，Dugan C，Freyne J，RecSys 09：Workshop on recommender systems and the social Web，*Proc. of the ACM RecSys*，New York：ACM Press，2009，421-422.

**[Josiah, 2015] Josiah L.Cartson，*Redis in action*，第二版，北京：人民邮电出版，2015。**

**[Jung,2009] Jung JJ，Contextualized mobile recommendation service based on interactive sociaI network discovered from mobile users，*Expert Systems with Applications*，2009，36(1)：11950-11956.**

**[Jure, 2015] Jure Leskovec, Anand Rajaraman, Jeffrey David Ullman，*Mining of Massive Datasets，*第二版，北京：人民邮电出版，2015。**

**[Kazienko,2011] Kazienko P．Musial K．Kajdanowicz T．Multidimensional social network in the social recommender system，*IEEE Trans on Systems*，*Man，and Cybernetics -- PartA：Systems andHumans*，2011，41(4)：746—759.**

[Koren,2008] **Koren Y. Factorization meets the neighborhood: A multifaceted collaborative filtering model. *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2008. 426–434.**

[Koren,2009] **Koren Y，Bell R，Volinsky C，Matrix factorization techniques for recommender systems，*Computer*，2009，42(8): 30-37.**

**[Levandoski,2012] Levandoski JJ, Sarwat M, Eldawy A, Mokbel M, LARS：A location—aware recommender system, *Proc．ofthe ICDE*, Washington：IEEE Computer Society，2012，450-461.**

**[Meng,2013] Meng XW，Hu X，Wang LC，Zhang YJ，Mobile recommender systems and their applications，*Journal of Software*，2013，24(1)：91-108.**

**[Min,2011] Min JK, Cho SB, Mobile human network management and recommendation by probabilistic social mining, *IEEE Trans．on Systems Man and Cybernetics*, Part B，2011，41(3)：76l-771.**

**[Mobasher,2013] Mobasher B，Jannach D，Geyer W，Freyne J，Recommender systems and the social Web，*Proc. ofthe ACM RecSys 2013*，ACM Press，2013，477-478.**

[Quijano,2013] **Quijano-Sanchez L，Recio-Garcia J，Diaz -- Agudo B Social factors in group recommender systems, *ACM Trans on Intelligent Systems and Technology*, 2013, 4(1)：Article No．8.**

[Schafer,2010] **Schafer JB, Dan F, Herlocker J, et al. Collaborative filtering recommender systems. *The Adaptive Web, Methods and Strategies of Web Personalizationk*, 2010，46-45.**

[Social Network,2007] <http://orgnet.com/sna.htmI>**, Social Network Analysis** maintained by **A brief introduction, 2007.**

**[Vozalis,2005] Vozalis MG, Margaritis KG, Applying SVD on item-based filtering, *International Conference on Intelligent Systems Design and Applications*, 2005, 464-469.**

[Wang,2013] **Wang Z，Sun LF，Zhu WW，Yang SQ，Li HZ，Wu DP. Joint social and content recommendation for user -- generated videos in online social network．*IEEE*** *Trans* *on Multimedia*，2013，15(3)：698-710.

[Zaharia,2010] **Zaharia M, Chowdhury M, Franklin MJ, et al, *Spark: Cluster computing with working sets*, Usenix Conference on Hot Topics in Cloud Computing, USENIX Association，2010，1765-1773.**

[Zaharia,2012] **Zaharia M, Chowdhury M, Das T, et al, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, *Usenix Conference on Networked Systems Design and Implementation*, 2012, 141-146.**

**[Zanda,2012] Zanda A, Eibe S, Menasalvas E, *SOMAR：A social mobile activity recommender Expert Systems with Applications*，2012，39：8423-8429.**

[梁冰, 2013] 梁冰，张玉忠，金铎，协同过滤中的矩阵分解算法研究，*广东通信技术*，2013， 8(16):76-79。

[刘建国, 2009] 刘建国，周涛，汪秉宏，个性化推荐系统的研究进展，*自然科学进展*，2009， 19(1):1-2。

[孟祥武, 2015] 孟祥武，刘树栋，张玉洁，胡勋，社会化推荐系统研究，*软件学报ISSN*，2015，26(6):1356-1372。

[王雪婷, 2014] 王雪婷，黄文亮，*基于社会正则化的推荐算法研究*，计算机与现代化，河海大学计算机与信息学院，2014。

[王振军，2017] 王振军，黄瑞章，基于Spark的矩阵分解与最近邻融合的推荐算法，*计算机系统应用*，2017，26(4):124-129。

[杨阳, 2012] 杨阳，向阳，熊磊，基于矩阵分解与用户近邻模型的协同过滤推荐算法，*计算机应用*，2012，32(2):395-398。

**[于寅, 2012] 于寅，*高等工程数学*，第四版，武汉：华中科技大学出版社，2012。**

**[张安站, 2015] 张安站，*Spark技术内幕：深入解析Spark内核架构设计与实现原理*，第一版，北京：机械工业出版社，2015。**

**[张宇, 2013] 张宇，程久军，基于MapReduce的矩阵分解推荐算法研究，*计算机科学*，2013，40(1):19-21。**

**[蔡皖东, 1997] 蔡皖东，HTTP协议的传输机制与超文本链的研究，*微电子学与计算机*，4(1997):48-51.**

# 致 谢