软件工程概论报告

作品名称：基于协同过滤算法的电影推荐系统的设计与实现

填写日期：2025.5.24

目 录

[引言 1](#_Toc31210)

[1.选题背景与意义 1](#_Toc2601)

[2.小组分工情况 2](#_Toc22430)

[3.项目进度计划与安排 2](#_Toc25560)

[4.整个开发过程中涉及到的工具说明 3](#_Toc11960)

[1 需求分析 5](#_Toc9098)

[1.1 软件需求分析 5](#_Toc25219)

[1.1.1系统的功能需求 5](#_Toc701)

[1.1.2系统非功能需求 6](#_Toc10834)

[1.2用户需求 8](#_Toc31671)

[1.2.1用例模型 8](#_Toc19530)

[1.2.2主要用例的描述 8](#_Toc29890)

[1.2.3用户需求的实现优先级 15](#_Toc13171)

[2 软件设计 16](#_Toc31489)

[2.1系统总体架构设计 16](#_Toc21923)

[2.2界面设计 18](#_Toc15691)

[2.3 用例的设计 20](#_Toc15712)

[2.4类的设计 25](#_Toc32490)

[2.5.数据库设计 28](#_Toc32)

[3 编码实现 31](#_Toc18800)

[4 测试 43](#_Toc27484)

[5 安装及部署 44](#_Toc7669)

[6 项目总结（团队） 45](#_Toc10618)

引言

## 1.选题背景与意义

随着互联网技术的迅猛发展以及流媒体平台（如Netflix、爱奇艺等）的广泛普及，电影数量呈现出爆炸式增长的态势。这些平台所包含的内容库多达数万部影片，使得用户在面对海量选择时常常产生“信息过载”问题，难以高效、准确地发现符合个人偏好的内容。传统的基于分类浏览或关键词搜索的内容获取方式在应对如此规模的信息时已显得力不从心。因此，推荐系统作为信息过滤的重要技术手段，成为解决该问题的关键途径。

协同过滤（Collaborative Filtering, CF）作为推荐系统领域中最经典且应用最为广泛的方法之一，其核心思想是基于用户的历史行为数据（如评分、点击、收藏等），挖掘用户之间或物品之间的潜在相似性，从而预测用户可能感兴趣的内容。传统的协同过滤方法主要分为两类：基于用户的协同过滤（User-Based CF）和基于物品的协同过滤（Item-Based CF）。尽管这些方法取得了显著的应用成效，但在实际场景中仍面临一系列挑战，如评分数据稀疏、冷启动问题以及用户评分标准不一致等问题。

为提升推荐系统的准确性和鲁棒性，本文提出了一种改进型的基于用户的协同过滤算法，并在多个关键技术环节进行了优化设计。其主要创新点包括：

（1）引入皮尔逊相关系数：相较于传统的余弦相似度，皮尔逊相关系数能够更有效地度量用户之间评分趋势的相似性，从而减弱因评分尺度不同而导致的误差；

（2）采用用户评分中心化与缺失值处理机制：将评分数据进行去中心化处理，同时保留真实评分为0的情况，并对缺失数据进行合理处理，避免对未评分项的误判；

（3）邻居筛选机制优化：仅选择相似度大于零的 Top-K 相似用户作为活跃邻居，排除无关用户，从而提高预测评分的相关性与有效性；

（4）基于加权平均的评分预测模型：结合相似用户的评分数据，采用加权平均方式对目标用户未评分的电影进行预测，增强个性化推荐效果；

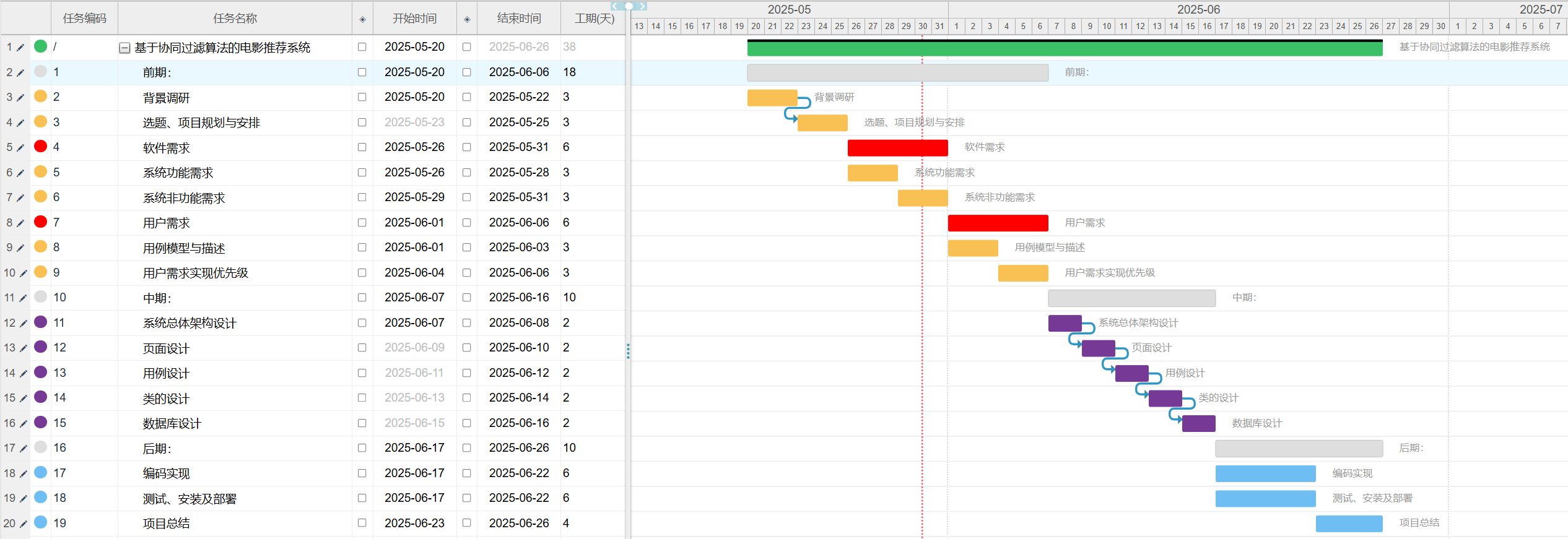
（5）生成个性化 Top-N 推荐列表：在排除用户已评分内容的基础上，筛选预测评分最高的若干电影，为用户提供具有实际价值和新颖性的推荐结果。

该改进方法不仅在精度上优于传统User CF模型，还提升了对评分矩阵稀疏性和用户行为差异的适应能力，特别适用于电影推荐这类具有长尾特征和高用户个性化需求的应用场景。因此，基于此技术的推荐系统在解决信息过载问题、提升用户满意度和平台活跃度方面具有重要的实际意义和应用价值。

## 2.小组分工情况

|  |  |  |
| --- | --- | --- |
| 成员 | 主要任务 | 小组成员自评 |
| 组长  姜佳辰 | 1.背景调研、需求分析、进行选题、项目的规划与安排  2.进行系统的总体架构设计、页面设计、用例设计、类设计、数据库设计、UML图的绘制 3.进行项目的代码编程  4.报告填写 | 100 |
| 组员  陈栋睿 | 1.软件的测试以及部署  2.报告填写以及PPT的制作 | 99 |

## 3.项目进度计划与安排



## 

## 4.整个开发过程中涉及到的工具说明

|  |  |  |  |
| --- | --- | --- | --- |
| **阶段** | **工具** | **任务** | **完成人** |
| 需求分析阶段 | 文献调研、平台使用体验分析、互联网检索 | 明确系统目标与核心功能，包括用户个性化推荐、评分数据处理等需求；分析现有推荐系统的不足与改进方向 | 姜佳辰 |
| 设计阶段 | Draw.Io、Process ON、Visio、PlantUML、boardmix | 设计系统的整体架构，包括数据流、模块划分、数据库结构设计、UML图、前后端交互逻辑，以及推荐算法流程建模 | 姜佳辰 |
| 编码实现 | PyCharm  Flask HTML/CSS/JavaScript  SQL | 数据的爬取，推荐算法实现，前端页面的展示、后端路由、数据库的建立 | 姜佳辰 |
| 测试 | 手动测试、白盒测试、黑盒测试 | 对系统各模块进行功能测试和性能测试，确保系统逻辑正确、推荐结果合理、前后端交互顺畅、无重大异常或漏洞 | 陈栋睿 |

5.项目自评描述

|  |  |
| --- | --- |
| **软件项目要素** | **自评描述** |
| 软件系统主要功能及特色 | 本系统实现了用户/管理员身份认证、数据管理、电影推荐、AI助手和后台统计分析等核心功能 |
| 软件系统是否集成了其他系统（如云服务）或物理设备（如机器人、可穿戴设备、智能手机等），如有请列出 | 软件系统集成了 MySQL 数据库系统/Redis数据库 |
| 描述软件系统的部署情况 | 系统通过本地Flask服务器部署 |
| 软件开发所用的编程语言 | 后端主要采用Python语言，使用Flask框架开发API和业务逻辑；前端部分使用HTML、CSS和JavaScript进行界面开发和交互实现。数据库操作使用SQL（MySQL）语言。 |
| 软件开发所用的支撑软件工具 | 代码编辑器/IDE：PyCharm  数据库管理工具：Navicat  数据库软件：Mysql/Redis |
| 软件系统的代码总量(LOC)（包含开源软件代码） | 7000+ |
| 代码的质量情况（如SonarQube的质量分析结果、是否遵循编码规范、代码的注释情况等） | 代码整体质量良好，遵循PEP8编码规范 注释较为充分，重要模块和函数均有详细说明 |
| UML模型（如提交了哪些UML模型、质量如何） | 用例图（Use Case Diagram）：描述系统的功能需求和用户交互。类图（Class Diagram）：描述系统中的类及其属性、方法和关系。顺序图（Sequence Diagram）：展示对象之间的消息传递和交互顺序。活动图（Activity Diagram）：描述业务流程或操作流程。部署图（Deployment Diagram）：描述系统的硬件部署环境。架构图（Architecture diagram）描述各部分的主要部件及部件之间的关系。 |
| 软件文档（质量如何） | 本项目的软件文档编写规范、内容完整，涵盖了系统架构设计、功能说明、接口文档、使用说明，能够有效支撑开发、测试与运维工作。文档结构清晰，语言简洁易懂，并根据不同使用者需求进行了分类整理，具备良好的可读性与可维护性。 |
| 列举访问过的开源社区，包括软件开发知识分享社区和开源软件托管社区 | CSDN —— 中文开发者技术分享社区  GitHub —— 代码托管和开源项目交流平台 |

# 1 需求分析

## 1.1 软件需求分析

## 1.1.1系统的功能需求

## 未命名绘图.iodraw (1)

AI功能：导入DeepseekAPI结合本地数据可以帮助用户或管理员进行搜索电影或者相关推荐

用户登录与身份识别：用户可通过选择或输入用户ID登录系统，系统据此加载用户相关数据，为后续操作提供个性化支持。

热门电影浏览：首页展示平台最受欢迎的热门电影，用户可快速浏览高分佳作，了解最新口碑动态。

个性化推荐：系统基于协同过滤推荐算法，根据用户的历史评分和偏好，为其推荐可能喜欢的电影。

已评分电影管理：用户可在“我的已评分”页面，直观查看自己过往评分过的电影，浏览详细信息，方便回顾与管理。

电影筛选与高级搜索：系统支持多条件组合筛选，包括类型、年份、评分区间等，用户可通过条件栏灵活检索并浏览目标电影，提升查找效率。

电影评论区：用户可通过“评论”页面搜索目标电影，查看全部用户评论，发布自己的影评、观点，也可删除自己发布的评论，实现自由交流与内容管理。

成就系统：根据用户的评分、评论行为等定制相应的成就系统。

管理员登录与权限控制：后台仅允许授权管理员账户访问，保障后台操作的安全性。管理员可通过专属入口登录后台，进行后续管理操作。

电影列表浏览：可查看平台全部电影的详细信息，包括标题、年份、类型、评分、排名等。

电影信息管理：支持查看、添加、修改、删除电影的相关信息。

用户列表浏览：可查看全部注册用户的信息和操作记录。

用户评分记录管理：支持查看、添加、删除指定用户的历史评分数据。

反馈：用户可以向管理员反馈，管理员可以向用户发通知。

演职人员统计：对相关出演人员职业、年龄等进行分析。

电影评分分布分析：支持对平台电影的评分分布、类型分布等进行可视化分析。

系统数据总览：快速查看平台内总电影数、用户数、评论数等核心指标。

搜索与检索：支持对用户和电影的快速检索，方便提升管理效率。

评论管理：可以按照内容或电影ID来查看和删除用户的评论

票房信息、影视资讯榜：查看电影票房，相关新闻。

## 1.1.2系统非功能需求

|  |  |  |
| --- | --- | --- |
| **模块** | **类别** | **要求 / 说明** |
| **1. 开发工具 / 运行环境** | 开发语言 | Python 3.12.0 |
|  | Web 框架 | Flask 2.0.3 |
|  | 数据库 | MySQL、Redis、CSV 文件存储 |
|  | 前端技术 | HTML5、CSS3、JavaScript、Chart.js |
|  | 开发工具 | PyCharm |
|  | 运行环境 | Windows 11 |
| **2. 部署要求** | 硬件 | 最低 2 核 CPU，4GB 内存，10GB 存储 |
|  | 软件 | Python 3.12+、MySQL、Redis |
|  | 网络 | HTTP/HTTPS 访问 |
|  | 部署方式 | 单机部署 |
| **3. 系统接口** | 用户界面 | Web 浏览器 |
|  | 数据接口 | 支持电影、评分、用户、评论、反馈等数据的批量导入导出 |
| **4. 数据要求** | 数据格式 | 数据文件符合通用CSV标准（字段名、编码、分隔符统一） |
|  | 数据量 | 支持 10,000 + 电影和 300,000 + 评分记录 |
|  | 数据更新 | 支持前后台动态添加、编辑、删除（无需重启服务） |
|  | 数据备份 | 定期备份 CSV 文件 |
| **5. 性能要求** | 响应时间 | 页面加载 < 1s，推荐计算 < 3s |
|  | 数据加载 | 初始化数据加载 < 5s |
|  | 推荐效率 | 每秒处理 10 + 推荐请求 |
| **6. 质量体系** | 可靠性 | 系统可用性 > 99%，错误率 < 0.1% |
|  | 安全性 | 管理员权限校验，防SQL注入，密码/Token保护 |
|  | 可维护性 | 模块化设计、清晰代码结构 |
|  | 可扩展性 | 支持推荐算法和数据规模扩展，便于后续接入AI与新功能 |

可行性及潜在风险

|  |  |  |
| --- | --- | --- |
| **模块** | **类别** | **评估 / 说明** |
| **1. 可行性分析** | 技术可行性 | 采用成熟 Python 技术栈，无技术障碍 |
|  | 经济可行性 | 使用开源技术，硬件成本低 |
|  | 操作可行性 | 界面友好，管理员易于使用 |
| **2. 潜在风险及应对措施** | 风险 | 数据丢失、性能瓶颈、安全漏洞、推荐质量 |
|  | 应对措施 | 定期备份 / 数据库冗余、算法优化 / 缓存机制、输入验证 / 权限控制、持续优化算法 |

界面要求

|  |  |
| --- | --- |
| **类别** | **要求** |
| **原型** | 基于现有 HTML 模板实现响应式布局，兼容主流 PC 与移动端浏览器 |
| **风格** | 采用现代化UI风格，配色简洁明快，突出电影内容和视觉沉浸感 |
| **功能区域** | 管理后台与普通用户界面明确区分，管理侧边栏与前台主导航各自独立，确保角色权限清晰，体验一致 |

**进度计划**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **周次** | **阶段** | **任务名称** | **开始时间** | **结束时间** |
| **第 1 周** | **前期准备** | 背景调研与项目规划 | 2025/5/20 | 2025/5/25 |
| **第 1 周** | **前期准备** | 软件需求分析（功能与非功能） | 2025/5/26 | 2025/5/31 |
| **第 2 周** | **前期准备** | 用户需求分析与优先级排序 | 2025/6/1 | 2025/6/6 |
| **第 2 周** | **中期设计** | 系统架构与页面设计 | 2025/6/7 | 2025/6/10 |
| **第 3 周** | **中期设计** | 用例与数据库设计 | 2025/6/11 | 2025/6/16 |
| **第 3 周** | **后期开发** | 核心功能编码实现 | 2025/6/17 | 2025/6/22 |
| **第 4 周** | **后期部署** | 界面优化、测试与部署 | 2025/6/23 | 2025/6/26 |
| **第 4 周** | **项目收尾** | 项目总结与验收 | 2025/6/23 | 2025/6/26 |

交付要求

|  |  |
| --- | --- |
| **交付物** | **形式** |
| **源代码** | GitHub 仓库 |
| **文档** | 结课报告（需求 / 设计 / 用户手册） |

验收要求

|  |  |
| --- | --- |
| **类别** | **要求** |
| **功能验收** | 完成所有需求功能 |
| **性能验收** | 满足性能指标 |
| **文档验收** | 完整的技术文档 |
| **用户验收** | 负责人确认 |

## 1.2用户需求

## 1.2.1用例模型

## 1.2.2主要用例的描述

|  |  |  |
| --- | --- | --- |
| 用例名称： | 获取个性化推荐电影 | |
| 用例ID： | UC001 | |
| 角色： | 用户 | |
| 用例说明： | 用户登录系统后，系统自动为其展示“为我推荐”页面，根据协同过滤算法，展示该用户可能喜欢的个性化电影推荐列表。 | |
| 前置条件： | 系统正常启动并加载所有电影、评分、用户数据  推荐算法已初始化  用户已成功登录系统 | |
| 基本事件流: | 参与者动作 | 系统响应 |
| 1. 用户登录后访问“为我推荐”页面  5. 页面显示推荐电影列表 | 2. 系统获取当前用户ID  3. 系统调用推荐器模块，基于用户 ID 计算推荐电影。 4. 系统将推荐结果返回前端页面 |
| 其他事件流: | 无 | |
| 异常事件流: | 参与者动作 | 系统响应 |
| 推荐数据加载失败或算法执行出错。  找不到指定用户 ID 的评分数据。  系统无法从数据库加载数据。 | 提示用户 “推荐生成失败，请稍后再试。”  提示用户 “该用户 ID 无历史评分数据，无法生成推荐。”  提示用户 “数据加载失败，请联系管理员。” |
| 后置条件： | 用户看到推荐电影列表 | |

|  |  |  |
| --- | --- | --- |
| 用例名称： | 电影评论 | |
| 用例ID： | UC002 | |
| 角色： | 用户 | |
| 用例说明： | 用户可在“评论”页面，搜索电影，查看评论区、发布自己的评论、删除自己发布的评论 | |
| 前置条件： | 用户已登录系统  评论功能正常加载  已有电影数据可搜索 | |
| 基本事件流: | 参与者动作 | 系统响应 |
| 1.用户进入“评论”导航页  3.用户输入电影名称搜索  5.用户选中电影  7.用户输入评论内容并提交  9.用户点击“删除”自己的评论 | 2.显示电影搜索框  4.系统返回匹配的电影  6.系统展示该电影的全部评论列表  8.系统保存评论并刷新评论区显示  10.系统删除该条评论并刷新评论区 |
| 其他事件流: | 无 | |
| 异常事件流: | 参与者动作 | 系统响应 |
| 电影名称不存在  评论内容为空  用户试图删除他人评论  数据库写入失败 | 系统提示“未找到相关电影”  系统提示“评论内容不能为空”  系统提示“无权删除”  系统提示“评论操作失败，请重试” |
| 后置条件： | 评论区实时显示用户新增/删除的评论 | |

|  |  |  |
| --- | --- | --- |
| 用例名称： | 按类型/条件筛选电影 | |
| 用例ID： | UC003 | |
| 角色： | 用户 | |
| 用例说明： | 用户可在“电影筛选”页，根据电影类型、年份、评分等条件筛选浏览电影。 | |
| 前置条件： | 用户已登录系统  系统已加载电影  数据筛选页可访问 | |
| 基本事件流: | 参与者动作 | 系统响应 |
| 1.用户点击“电影筛选”导航  3.用户选择筛选条件  5.用户可多次修改筛选条件 | 2.系统显示筛选条件栏（类型、年份、评分区间等）  4.系统动态更新展示符合条件的电影列表  6.系统实时刷新筛选结果 |
| 其他事件流: | 无 | |
| 异常事件流: | 参与者动作 | 系统响应 |
| 没有符合条件的电影  数据库读取失败 | 系统提示“暂无符合条件的电影  系统提示“电影数据加载失败” |
| 后置条件： | 用户浏览到匹配条件的电影 | |

|  |  |  |
| --- | --- | --- |
| 用例名称： | 管理电影信息 | |
| 用例ID： | UC004 | |
| 角色： | 管理员 | |
| 用例说明： | 管理员可以对所有电影的信息进行增删改查 | |
| 前置条件： | 系统已成功启动并加载电影数据。  管理员已成功访问后台管理界面。 | |
| 基本事件流: | 参与者动作 | 系统响应 |
| 1. 管理员导航到 “电影管理” 页面。  3. 管理员选择操作： a. 添加新电影 b. 编辑现有电影 c. 删除电影 d. 查看电影详情 e. 搜索电影  5. 管理员执行操作细节： a. 填写表单并提交 b. 修改表单并提交 c. 确认删除 d. 查看信息 e. 输入搜索关键词 | 2. 系统显示电影列表（支持分页）。  4. 系统根据选择执行操作： a. 显示 “添加电影” 表单。 b. 加载选定电影信息至 “编辑电影” 表单。 c. 提示确认删除。 d. 显示电影详情及评分分布。 e. 显示搜索结果列表。  6. 系统处理操作： a. 验证数据，保存新电影至 CSV 并重新初始化数据。 b. 验证数据，更新电影信息至 CSV 并重新初始化数据。 c. 从 CSV 删除电影及相关评分并重新初始化数据。 d. 显示电影详细数据。 e. 查找匹配项并显示搜索结果。  7. 系统反馈操作结果： a. 跳转至新电影详情页或电影列表页。 b. 跳转至编辑后的电影详情页或电影列表页。 c. 跳转回电影列表页。 d. 维持在电影详情页。 e. 维持在搜索结果页。 |
| 其他事件流: | 无 | |
| 异常事件流: | 参与者动作 | 系统响应 |
| 数据验证失败（如电影 ID 重复、格式错误）。  电影文件写入失败。 | 提示管理员 “数据输入无效，请检查。”  提示管理员 “电影数据保存失败，请重试。” |
| 后置条件： | 电影数据库已根据管理员操作更新。  管理员查看到操作结果（如新增、更新、删除电影或显示详情）。 | |

|  |  |  |
| --- | --- | --- |
| 用例名称： | 管理用户评分 | |
| 用例ID： | UC005 | |
| 角色： | 管理员 | |
| 用例说明： | 管理员可以对所有用户的评分信息进行增删改查 | |
| 前置条件： | 系统已成功启动并加载用户和评分数据。  管理员已成功访问后台管理界面。 | |
| 基本事件流: | 参与者动作 | 系统响应 |
| 1. 管理员导航到 “用户管理” 页面。  3. 管理员选择操作： a. 查看用户评分记录 b. 添加用户评分 c. 删除用户评分 d. 搜索用户  5. 管理员执行操作细节： a. 查看信息 b. 填写表单并提交 c. 确认删除 d. 输入搜索关键词 | 2. 系统显示用户列表（含评分数量、平均评分，支持分页）。  4. 系统根据选择执行操作： a. 显示选定用户的评分历史。 b. 显示 “添加评分” 表单。 c. 提示确认删除。 d. 显示搜索结果列表。  6. 系统处理操作： a. 显示用户所有评分记录（合并电影信息）。 b. 验证数据，保存新评分至 CSV 并重新初始化数据。 c. 从 CSV 删除指定评分并重新初始化数据。 d. 查找匹配用户并显示搜索结果。  7. 系统反馈操作结果： a. 维持在用户详情页。 b. 跳转至用户评分详情页。 c. 跳转至用户评分详情页。 d. 维持在搜索结果页。 |
| 其他事件流: | 无 | |
| 异常事件流: | 参与者动作 | 系统响应 |
| 数据验证失败（如评分值非法、用户 / 电影 ID 不存在）。  评分文件写入失败。  尝试删除不存在的评分记录。 | 提示管理员 “数据输入无效，请检查。”  提示管理员 “评分数据保存失败，请重试。”  提示管理员 “指定评分记录不存在。” |
| 后置条件： | 用户评分数据库已根据管理员操作更新。  管理员查看到操作结果（如新增评分、删除评分或显示评分历史）。 | |

|  |  |  |
| --- | --- | --- |
| 用例名称： | AI助手智能问答 | |
| 用例ID： | UC006 | |
| 角色： | 普通用户 / 管理员 | |
| 用例说明： | 用户或管理员可以通过AI助手自然语言查询系统内各类电影数据、用户数据和评论数据，AI助手会调用本地数据库和AI模型，自动给出友好回复 | |
| 前置条件： | **系统正常启动，用户/管理员已登录界面并可见AI助手悬浮按钮。AI助手后端可正常访问本地数据与API接口。** | |
| 基本事件流: | 参与者动作 | 系统响应 |
| 1. 用户点击右下角AI助手悬浮按钮。  3. 用户在对话框中输入查询内容（如“用户1有多少评论”“2023年评分最高的电影有哪些？”）。  8. 系统把AI回复显示在对话框中。 | 2. 系统弹出AI对话框，提示可输入查询内容  4. 系统接收消息，先用正则/NLP等方法解析用户意图。  5. 系统尝试用pandas等方法查询本地数据库（如查评论、评分、电影等）。  6. 若能查到，系统把查到的结果和原始问题一起发给AI大模型DeepSeek，生成友好回复  7. 若不能查到，则直接把用户问题发给AI模型，由AI自由回答。 |
| 其他事件流: | 无 | |
| 异常事件流: | 参与者动作 | 系统响应 |
| AI接口调用失败、API额度超限、网络异常  查询本地数据异常/数据文件丢失 | 系统提示“AI助手暂时不可用，请稍后重试。”  系统提示“系统数据暂不可用，请联系管理员。” |
| 后置条件： | 用户/管理员收到AI助手的自然语言答复，可以继续提问。若本地数据有变化，AI助手下次可查到最新数据。 | |

|  |  |  |
| --- | --- | --- |
| 用例名称： | 用户-管理员消息反馈与通知 | |
| 用例ID： | UC007 | |
| 角色： | 普通用户 / 管理员 | |
| 用例说明： | 普通用户可以通过消息中心页面，向管理员提交反馈、建议或投诉。管理员可在后台反馈与消息管理页面查看所有用户反馈，并主动向任意用户推送系统通知。用户和管理员均可通过消息中心实时收发消息，提升沟通效率。 | |
| 前置条件： | 系统正常启动，用户/管理员已登录界面，普通用户可见右上角“消息中心”入口，管理员可进入后台“反馈与消息管理”页面。 | |
| 基本事件流: | 参与者动作 | 系统响应 |
| 1. 用户点击页面右上角“消息中心”  3. 用户输入反馈内容并点击发送  5. 管理员登录后台，进入“反馈与消息管理”页面  7. 管理员可输入内容并选择用户ID，向任意用户推送通知 | 2. 系统展示消息对话框，显示历史消息与反馈提交入口  4. 系统将消息保存至本地数据库，并刷新消息列表  6. 系统展示所有用户消息和反馈列表，最新消息高亮显示  8. 系统保存并发送该通知，用户下次进入消息中心即可看到新通知 |
| 其他事件流: | 无 | |
| 异常事件流: | 参与者动作 | 系统响应 |
| 用户或管理员提交反馈/消息时，数据库写入异常或文件缺失  管理员尝试给不存在的用户发送通知 | 系统提示“消息服务不可用，请稍后重试。”  系统提示“用户不存在或已被删除。”，消息未发送 |
| 后置条件： | 用户和管理员均可在消息中心查看历史消息和反馈，沟通内容实时同步。新消息会在页面刷新或重新进入消息中心时立即可见。系统数据如有变化，消息中心展示内容也会随之更新。 | |

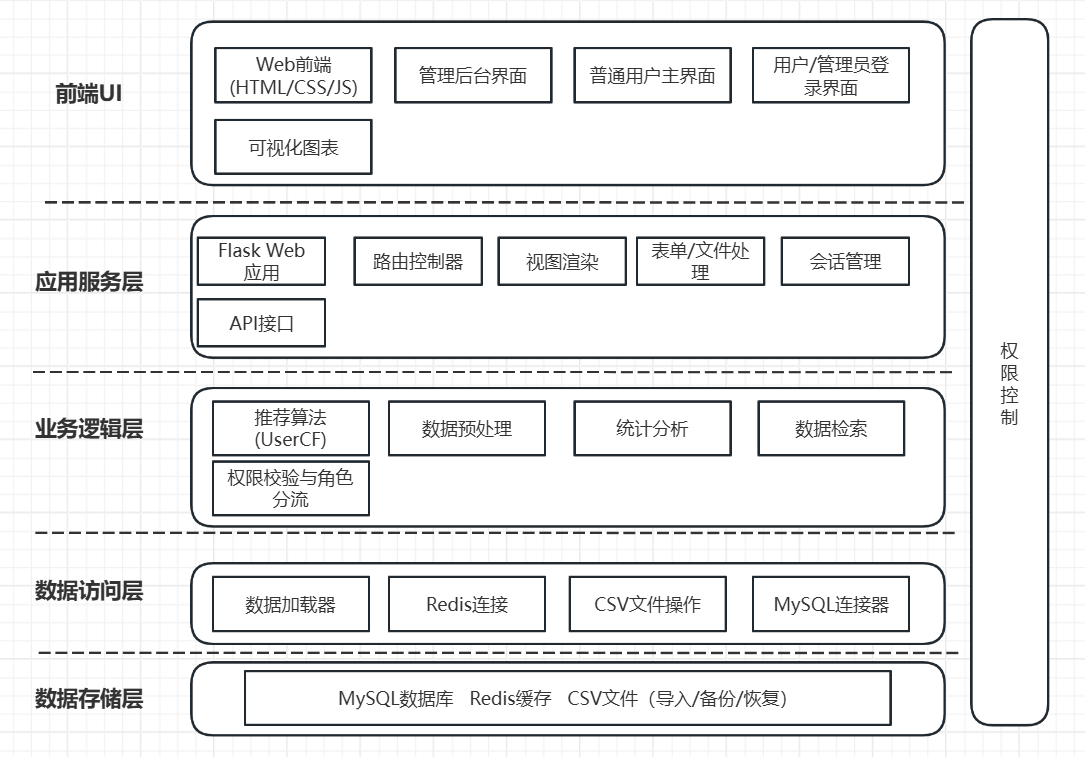
## 1.2.3用户需求的实现优先级

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **用例名称** | **核心 / 外围** | **关键 / 辅助** |
| **1** | 获取推荐电影 | 核心 | 关键 |
| **2** | 查看已评分电影 | 核心 | 辅助 |
| **3** | 电影类型筛选 | 核心 | 辅助 |
| **4** | 发表评论 | 核心 | 辅助 |
| **5** | 查看热门电影 | 核心 | 辅助 |
| **6** | AI助手智能问答 | 核心 | 关键 |
| **7** | 查看系统统计 | 外围 | 关键 |
| **8** | 电影管理 | 外围 | 关键 |
| **9** | 用户管理 | 外围 | 关键 |

# 

# 2 软件设计

## 2.1系统总体架构设计

**系统架构分层详细说明：**

本电影推荐系统采用分层架构设计，从底层数据存储到前端展示，各模块职责清晰，易于扩展与维护。整体系统通过权限控制机制实现多角色安全管理。

1. 前端UI层

负责所有用户可见界面的呈现与交互，支持普通用户与管理员两类角色。

Web前端（HTML/CSS/JS）：核心的页面结构与交互逻辑，负责数据展示和用户操作。

管理后台界面：供管理员管理电影、用户、评分、评论等信息，包含数据统计和图表。

普通用户主界面：普通用户浏览电影、评分、写评论、查看推荐、个人中心等功能入口。

用户/管理员登录界面：角色分离，确保不同权限用户进入对应操作区。

可视化图表：基于 Chart.js 等前端库展示统计数据、评分分布、类型占比等信息，提升数据可读性。

2. 应用服务层

处理核心业务流程和请求响应，所有前端请求最终都由本层管理。

Flask Web应用：承载全部业务路由，处理前后端数据交互。

路由控制器：定义各功能点的URL入口、处理请求分发。

视图渲染：基于Jinja2模板，将后端数据动态渲染为前端页面。

表单/文件处理：处理数据录入、文件上传/下载等表单型操作。

会话管理：管理用户登录状态、权限校验、Session信息。

API接口：为前端、AI助手等提供数据服务或消息/反馈、动态数据查询等接口。

3. 业务逻辑层

核心算法和业务流程实现。

推荐算法：用户协同过滤，分析用户行为，生成个性化推荐列表。

数据预处理：对原始数据清洗、特征工程处理，确保数据质量。

统计分析：用户活跃度、评分分布、热门类型等多维度统计。

数据检索：支持关键词、条件查询，提升搜索效率。

权限校验与角色分流：判断请求者角色（普通用户/管理员），动态授权或限制操作范围。

4. 数据访问层

为上层业务提供标准化的数据操作接口，支持多种数据源和缓存机制。

数据加载器：封装CSV文件、数据库等的数据读取与写入操作。

Redis连接：支持高并发下的用户相似度矩阵缓存管理。

CSV文件操作：实现本地文件的批量导入、导出、备份、恢复等功能。

MySQL连接器：通过SQL语句与数据库通信，实现结构化数据高效操作。

5. 数据存储层

系统底层数据的实际落地存储区域，支持多源、多副本。

MySQL数据库：持久化存储电影、用户、评分、评论、反馈等主业务表。

Redis缓存：存储高频访问数据、用户会话、临时消息等，提升系统响应速度。

CSV文件：作为数据导入导出的标准格式，同时承担日常数据的备份和恢复职责。

6. 权限控制（贯穿全局）

在所有层次实现多角色权限校验，保证数据安全、操作合规。

不同角色（普通用户、管理员）拥有不同访问、操作权限。

## 2.2界面设计

前台设计



用户功能导航栏：页面顶部设有横向主导航栏，集中展示了系统的核心功能入口，包括热门电影、为我推荐、我的已评分、电影筛选、评论和我的成就等。导航栏设计简洁明了，通过高亮和下划线实时指示当前所在页面，帮助用户快速在各大功能模块间切换，提高了操作便捷性和界面可用性。

内容显示区域：导航栏下方为核心内容展示区，根据当前用户选择的功能菜单动态切换显示不同内容。例如在热门电影页面，内容区域以大幅轮播图形式展示重点影片信息；在推荐页面则展现个性化推荐的电影卡片列表。该区域采用卡片式、分块布局，重点内容突出，界面层次分明，有效提升了用户的浏览效率和视觉体验。

总体逻辑切换：界面布局整体分明，各功能区域之间逻辑清晰。顶部导航栏与主内容区有明确的层级关系，所有功能切换与页面跳转均围绕主导航栏进行，保证用户在各大功能模块间流转时操作流畅、一致，避免迷失或重复操作。

悬浮AI助手：页面右下角设计有AI助手悬浮按钮，始终可见，方便用户随时唤起AI智能问答功能。用户点击后可与AI助手进行自然语言交互，获取电影推荐、数据查询等智能服务。悬浮设计不影响主要内容浏览，同时提升了系统的智能化和交互性。

后台设计



管理中心功能切换区：页面左侧为纵向排列的功能导航菜单，涵盖仪表盘、人物统计、电影管理、用户管理、评论管理、反馈/消息管理等各项核心管理功能。管理员可通过点击不同菜单项，实现后台各模块间的快速切换和功能跳转，极大提升了操作的高效性和系统的可扩展性。

内容显示区域：主操作区域位于页面中央和右侧，用于展示各类管理内容及统计报表。根据管理员选择的功能模块，内容区会动态呈现相关信息，如仪表盘展示系统关键数据总览及多种可视化图表，其他模块则显示用户信息、电影信息、评论详情等。内容区采用卡片式分块布局，重要数据醒目突出，界面层级清晰。

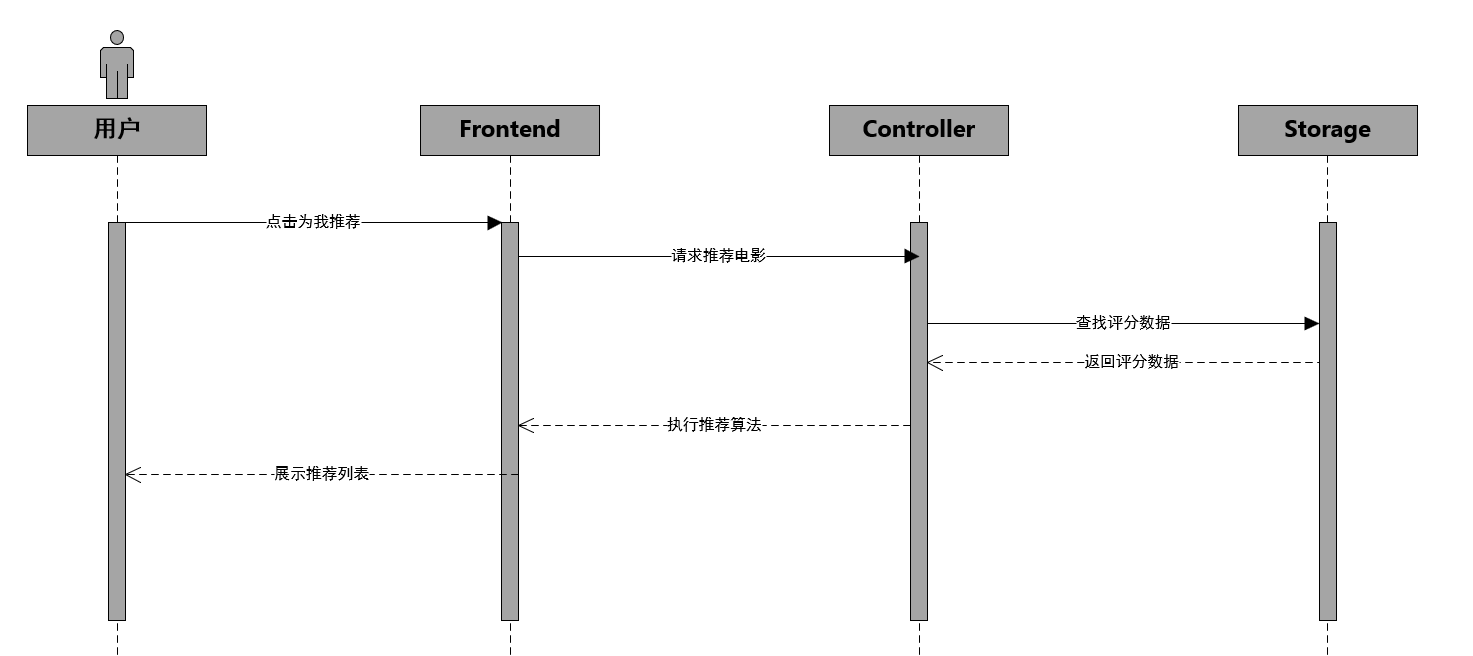
悬浮AI助手：页面右下角设有AI助手悬浮按钮，无论管理员浏览哪个功能模块，均可随时唤起AI助手进行智能问答、数据查询等操作。该设计兼顾了便捷性与功能拓展，增强了后台系统的智能化和交互性。

总体逻辑切换：后台管理页面顶部设置有系统名称栏和全局逻辑切换入口，包括首页、后台管理主页和退出登录等。该区域贯穿后台各页面，方便管理员进行系统级的导航与切换，确保整体操作流程流畅、体验一致。

## 2.3 用例的设计

顺序图

用户获取推荐电影



该顺序图展示了用户获取个性化推荐时的完整系统交互流程。主要操作包括输入用户ID、发起推荐请求、生成推荐结果，详细流程如下：

1.输入用户ID登录

用户在前端界面选择为我推荐，前端将该请求提交给控制器。

2.发起推荐请求

控制器接收到请求后，调用推荐模块，准备为用户生成个性化推荐列表；

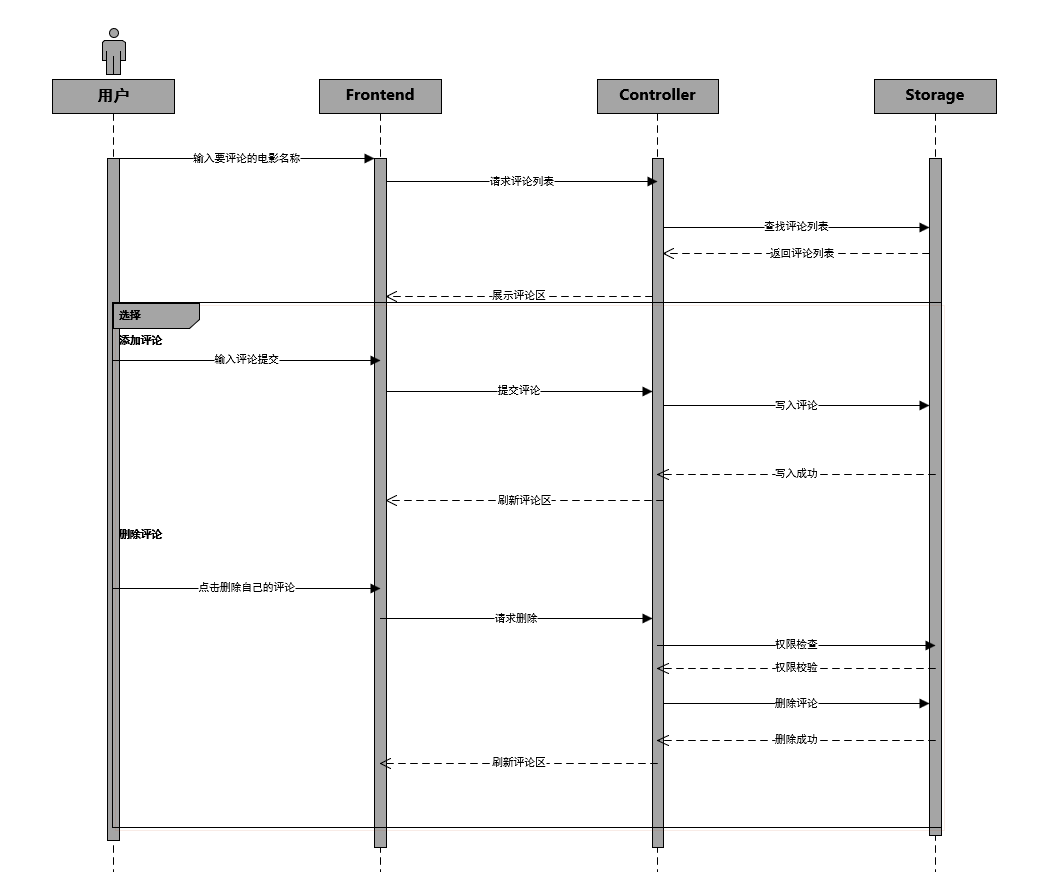
推荐模块需要访问数据库，从存储层获取该用户的历史评分数据。

3.生成并返回推荐结果

存储层返回评分数据，推荐模块基于用户历史行为和评分情况计算推荐结果；

控制器将推荐结果反馈至前端，前端页面展示个性化推荐内容或相应提示信息。

用户评论电影



该顺序图展示了管理员在系统中管理用户及用户评分信息时的完整交互流程。主要操作包括用户信息查询、用户评分查看与增删改、用户搜索等，详细流程如下：

1.用户信息管理

管理员进入用户管理界面，前端发起用户列表请求；

控制器从存储层读取所有用户信息并返回，前端显示用户信息列表；

管理员可选择具体用户，查看用户详细资料及评分记录，前端发起用户详情和评分请求，控制器查询并返回用户对应的数据，前端展示详细内容。

2.用户评分管理

管理员可为用户添加评分，输入评分信息后，前端将数据发送给控制器；

控制器调用存储层，写入新的评分数据，操作完成后前端同步刷新评分记录；

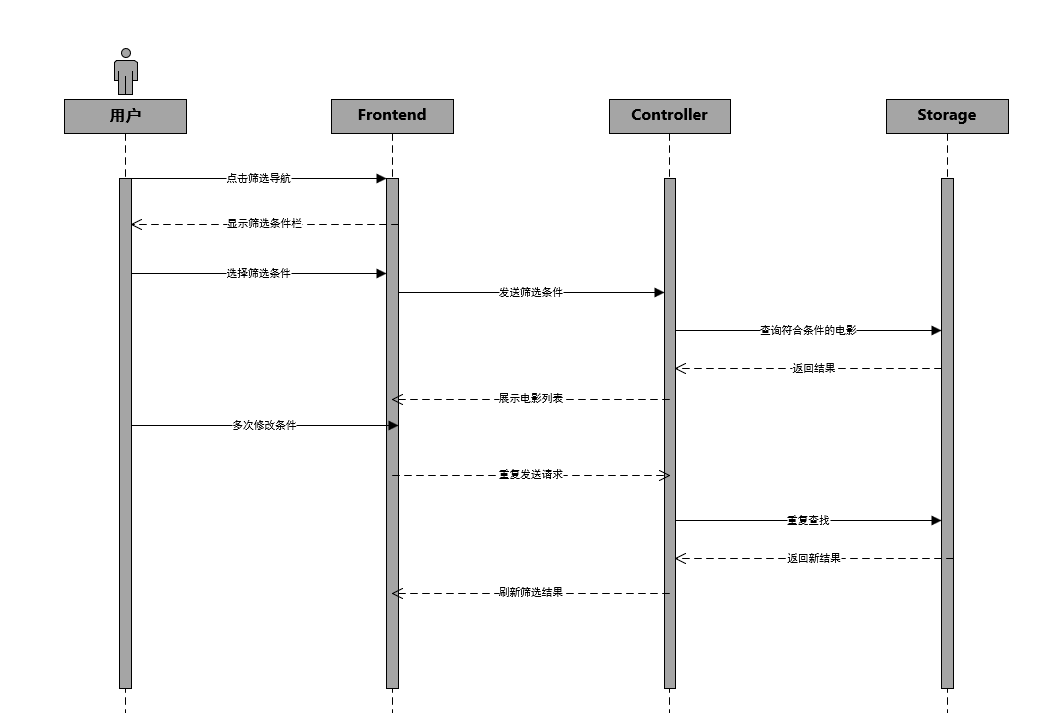
管理员也可编辑或删除用户的评分，前端发起修改或删除请求，控制器分别进行数据的更新或删除操作，存储层返回处理结果后前端实时展示最新评分信息。

3.用户搜索功能

管理员输入用户ID等关键词进行检索，前端发起搜索请求；

控制器根据搜索条件从存储层查找相关用户，返回结果，前端显示匹配用户的列表或详情。

用户筛选电影



该顺序图展示了用户在系统中进行电影筛选时的完整交互流程。主要操作包括打开筛选界面、选择筛选条件、多次修改筛选条件，详细流程如下：

1.打开筛选界面

用户点击前端导航栏中的筛选入口，前端展示可选的筛选条件栏（如类型、年份、评分等）。

2.选择筛选条件

用户选择感兴趣的筛选条件后，前端将这些条件发送给控制器；

控制器接收请求，向存储层查询符合条件的电影数据；

存储层返回筛选结果，控制器将电影列表反馈至前端页面，前端展示符合筛选要求的电影信息。

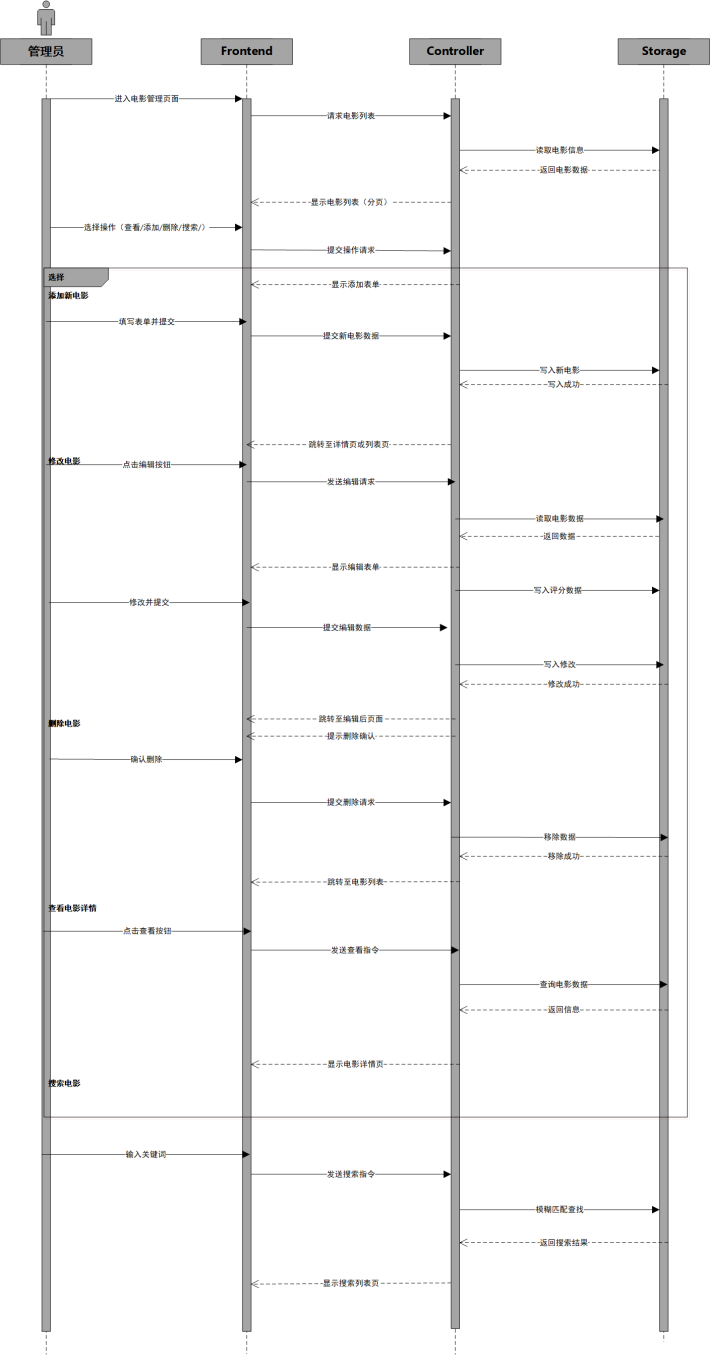
3.多次修改筛选条件

用户可多次修改筛选条件，前端会重复发送新的筛选请求；

控制器和存储层根据最新条件进行查询，返回更新后的筛选结果；

前端实时刷新显示最新的电影筛选列表，确保用户获取到最符合需求的影片信息。

管理员管理电影

该顺序图展示了后台管理员对电影数据进行管理时的完整系统交互流程。主要操作包括查看电影、筛选查询、添加电影、编辑电影和删除电影等，详细流程如下：

1.查看与筛选电影

管理员进入电影管理页面，前端发起电影列表请求至控制器；

控制器调用存储模块，从CSV文件中读取所有电影数据；

存储模块返回电影列表，控制器将数据传回前端，前端展示完整的电影信息表；

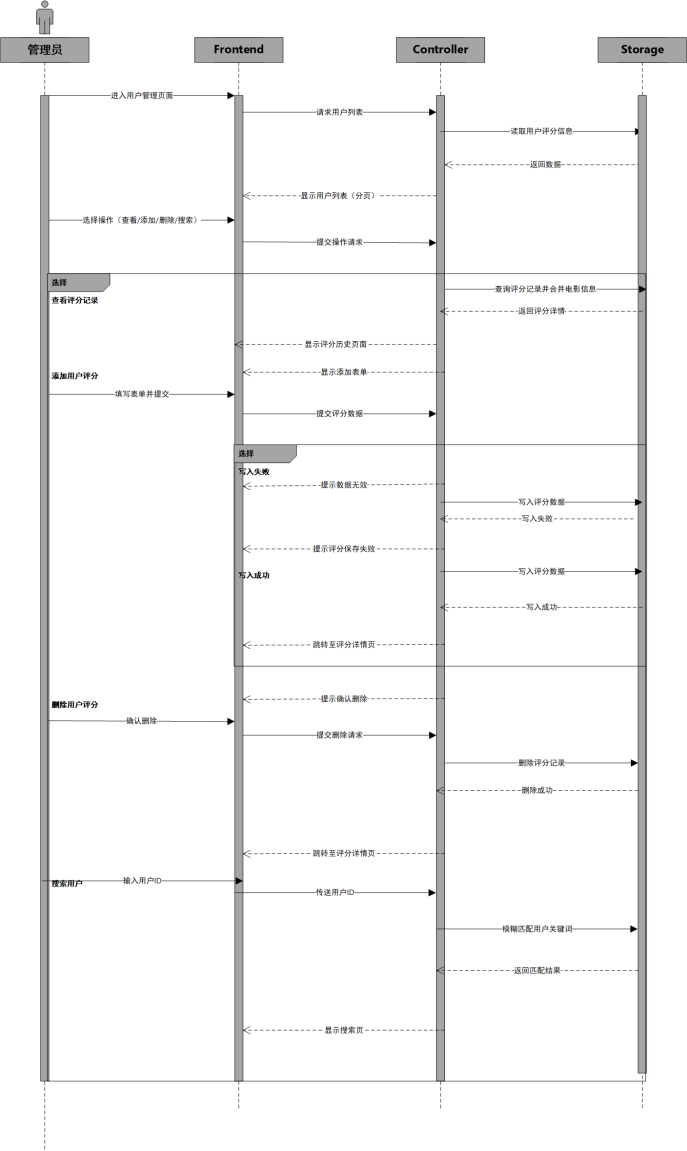
2.添加、编辑、删除电影

针对电影的新增、编辑或删除操作，前端会收集管理员输入的信息并发送给控制器；

控制器根据操作类型调用存储模块，对对应的CSV文件进行增删改写操作；

存储模块执行后返回处理结果，控制器将反馈信息传回前端，前端页面同步刷新，实时展现最新电影数据。

管理员管理用户评分

 该顺序图展示了管理员在系统中管理用户及用户评分信息时的完整交互流程。主要操作包括用户信息查询、用户评分查看与增删改、用户搜索等，详细流程如下：

1.用户信息管理

管理员进入用户管理界面，前端发起用户列表请求；

控制器从存储层读取所有用户信息并返回，前端显示用户信息列表；

管理员可选择具体用户，查看用户详细资料及评分记录，前端发起用户详情和评分请求，控制器查询并返回用户对应的数据，前端展示详细内容。

2.用户评分管理

管理员可为用户添加评分，输入评分信息后，前端将数据发送给控制器；

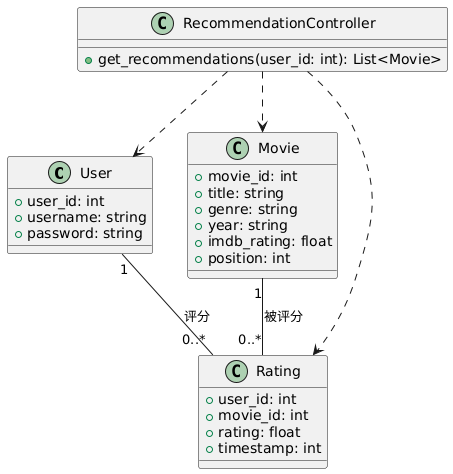
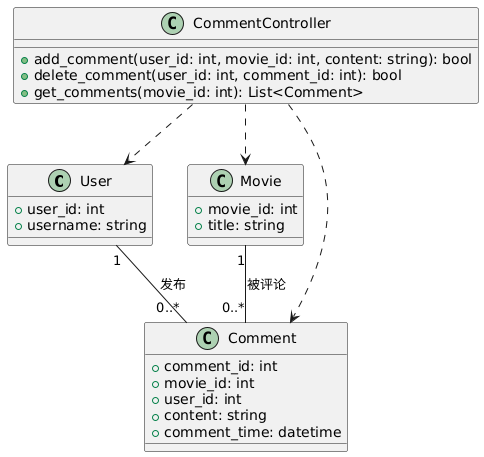
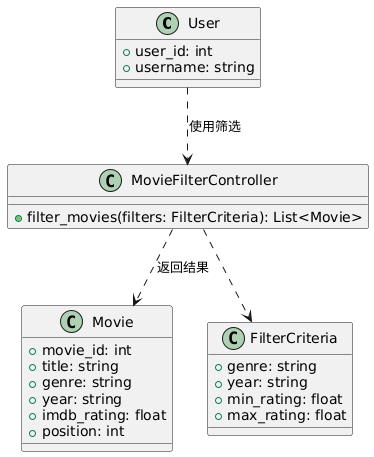
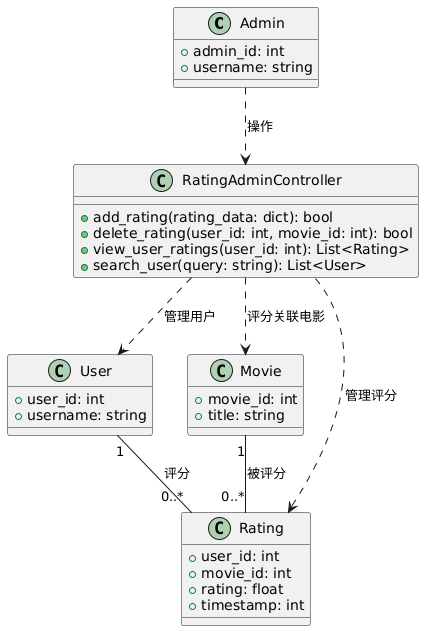
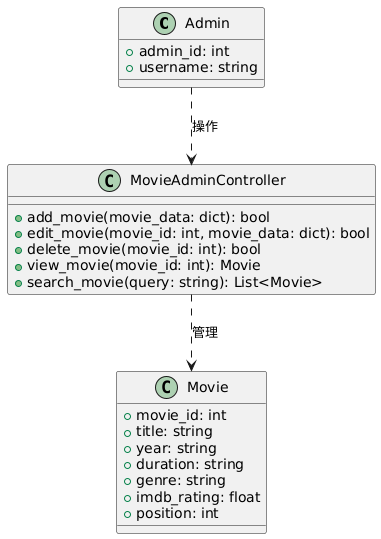
控制器调用存储层，写入新的评分数据，操作完成后前端同步刷新评分记录；

管理员也可编辑或删除用户的评分，前端发起修改或删除请求，控制器分别进行数据的更新或删除操作，存储层返回处理结果后前端实时展示最新评分信息。

3.用户搜索功能

管理员输入用户ID等关键词进行检索，前端发起搜索请求；

控制器根据搜索条件从存储层查找相关用户，返回结果，前端显示匹配用户的列表或详情。

用例实现的设计类图

1. 用户相关（User）普通用户可通过系统对电影进行评分、发表评论、筛选影片等操作。用户数据与评分（Rating）、评论（Comment）等模块关联，支持个性化推荐和活跃度统计。

2. 电影与评分管理（Movie, Rating, MovieAdminController, RatingAdminController）

Movie：电影基础信息，包括ID、名称、类型、年份、评分、排名等。

Rating：用户对电影的评分记录，支持按用户、电影维度统计。

MovieAdminController：提供后台电影的新增、编辑、删除、搜索等功能，实现对电影库的统一管理。RatingAdminController：实现评分的增删查、按用户查找评分记录，以及用户搜索等后台管理操作。

3. 评论系统（Comment, CommentController）

用户可对每部电影发布评论（Comment），评论与用户、电影均有关联。

## CommentController：负责评论的增删查，包括添加、删除和按电影查评论，支撑前台的互动功能和评论区展示。

## 4. 电影推荐与筛选（RecommendationController, MovieFilterController）

## RecommendationController：根据用户ID，调用推荐算法，返回个性化电影推荐列表，提升用户体验。

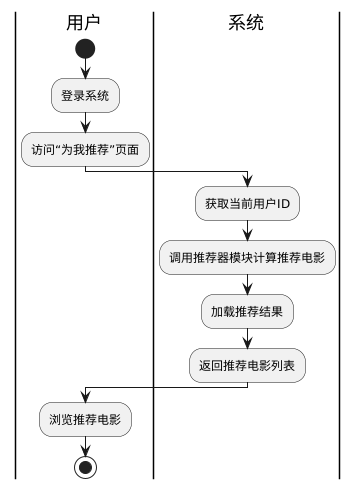
## MovieFilterController：根据用户指定的筛选条件（类型、年份、评分区间等），返回符合条件的电影集合，实现精准筛选与内容发现。

## 2.4类的设计

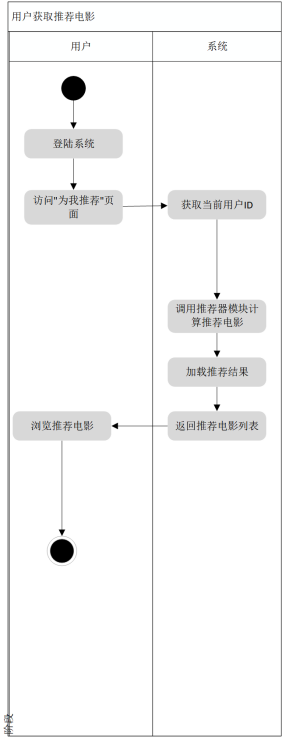
**用顺序图表示界面跳转**

活动图

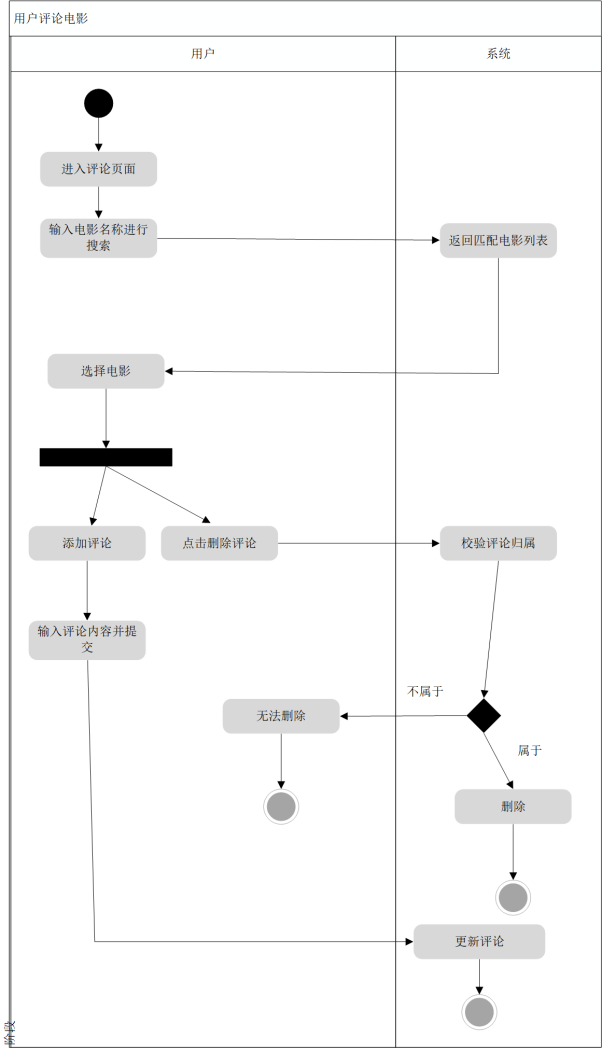
用户登录系统

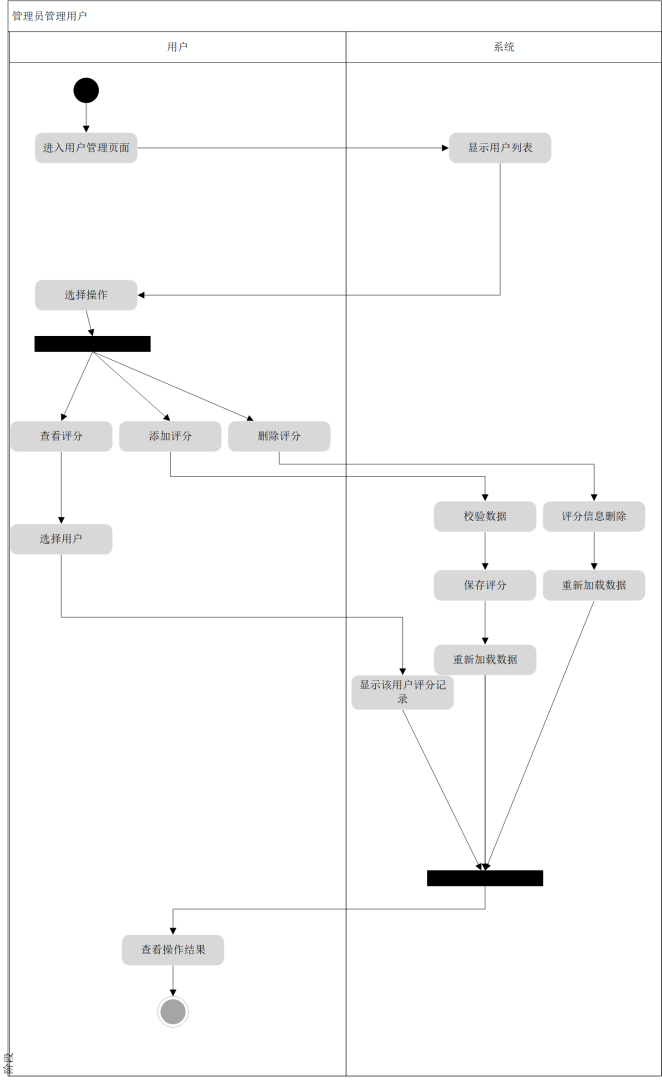
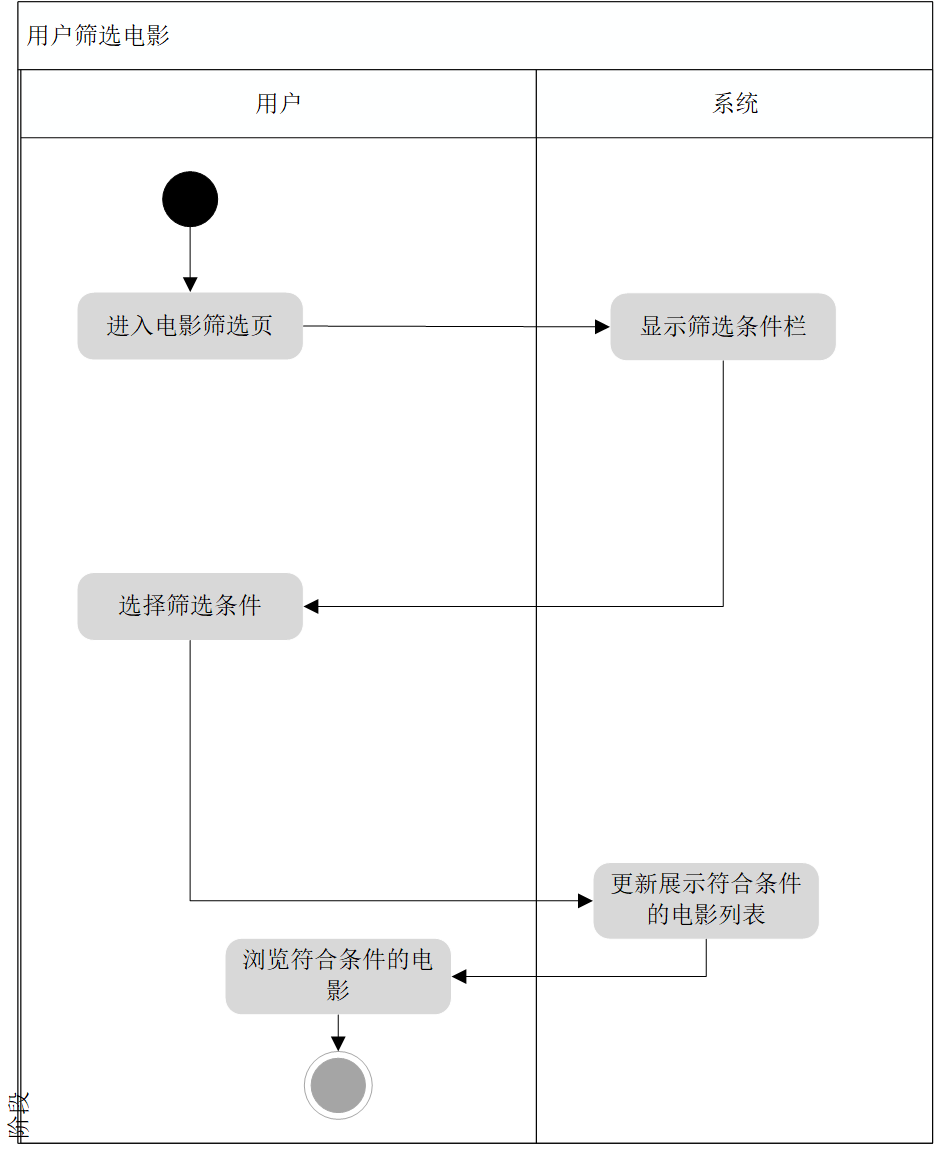


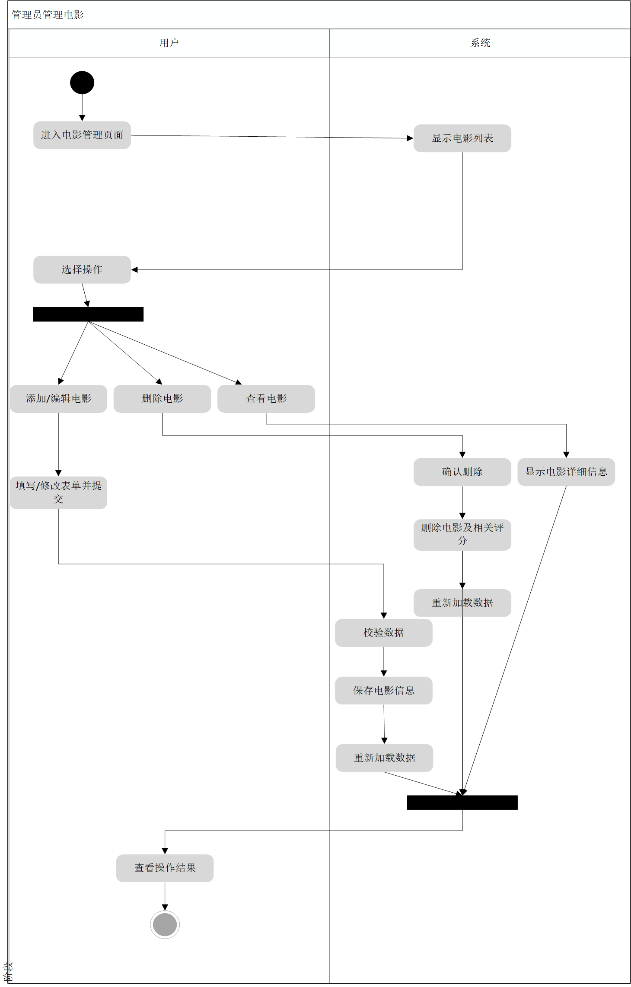
用户获取推荐电影



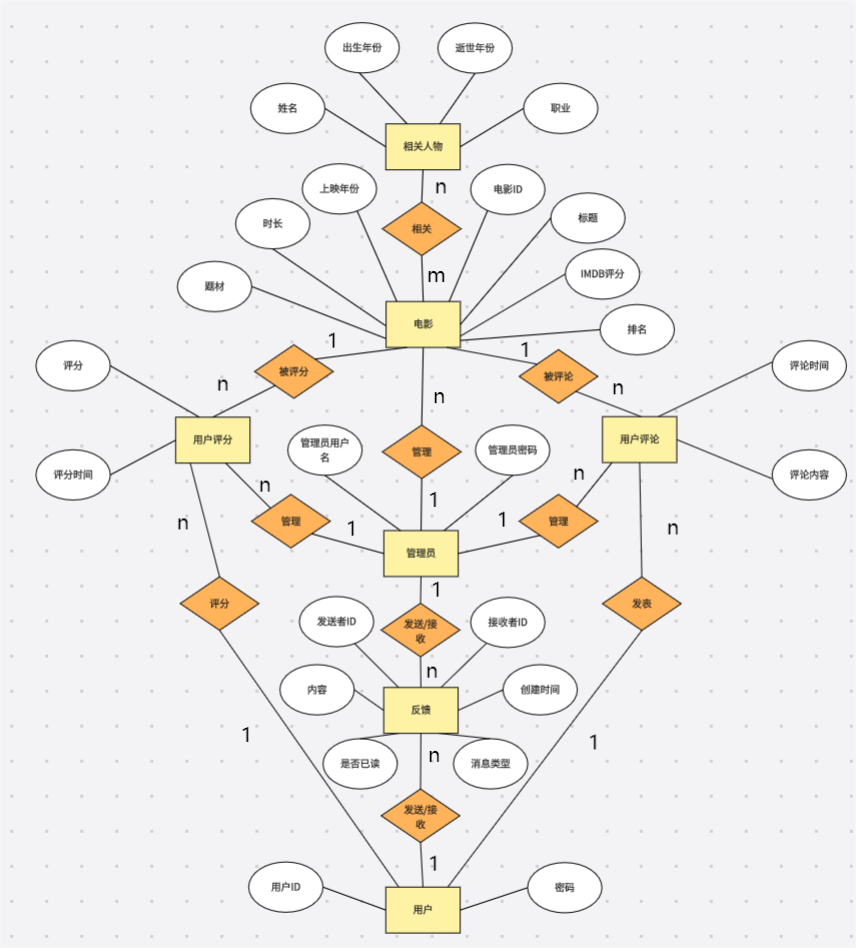
用户评论电影

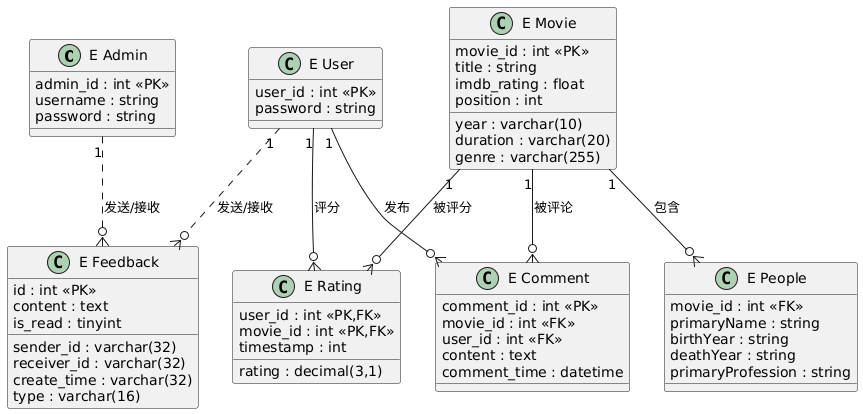
用户筛选电影

 管理员管理用户

 管理员管理电影

## 2.5.数据库设计

E-R图

表的设计类图

数据库表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **表名** | **字段名** | **数据类型** | **约束条件** | **说明** |
| **admins** | admin\_id | INT | PRIMARY KEY AUTO\_INCREMENT | 管理员唯一标识ID |
|  | username | VARCHAR(32) | UNIQUE, NOT NULL | 管理员用户名 |
|  | password | VARCHAR(64) | NOT NULL | 管理员密码 |
| **users** | user\_id | INT | PRIMARY KEY | 用户唯一ID |
|  | password | VARCHAR(64) | NOT NULL | 用户密码 |
| **movies** | movie\_id | INT | PRIMARY KEY | 电影唯一标识ID |
|  | title | VARCHAR(255) | NOT NULL | 电影标题 |
|  | year | VARCHAR(10) |  | 上映年份 |
|  | duration | VARCHAR(20) |  | 格式化时长 (如"2h 15m") |
|  | genre | VARCHAR(255) |  | 电影类型(逗号分隔) |
|  | imdb\_rating | FLOAT | CHECK (imdb\_rating BETWEEN 0.0 AND 10.0) | IMDB评分 (0.0-10.0) |
|  | position | INT | UNIQUE | 榜单排名位置 |
| **ratings** | user\_id | INT | PRIMARY KEY (user\_id, movie\_id), FOREIGN KEY | 用户ID |
|  | movie\_id | INT | PRIMARY KEY (user\_id, movie\_id), FOREIGN KEY | 电影ID |
|  | rating | DECIMAL(3,1) | CHECK (rating BETWEEN 0.0 AND 10.0) | 用户评分 |
|  | timestamp | INT |  | 评分时间戳 |
| **comments** | comment\_id | INT | PRIMARY KEY | 评论唯一ID |
|  | movie\_id | INT | NOT NULL, FOREIGN KEY REFERENCES movies(movie\_id) | 评论电影ID |
|  | user\_id | INT | NOT NULL, FOREIGN KEY REFERENCES users(user\_id) | 评论用户ID |
|  | content | TEXT | NOT NULL | 评论内容 |
|  | comment\_time | DATETIME | NOT NULL | 评论时间 |
| **people** | movie\_id | INT | FOREIGN KEY REFERENCES movies(movie\_id) | 所属电影ID |
|  | primaryName | VARCHAR(255) |  | 姓名 |
|  | birthYear | VARCHAR(10) |  | 出生年份 |
|  | deathYear | VARCHAR(10) |  | 逝世年份 |
|  | primaryProfession | TEXT |  | 职业（逗号分隔） |
| **feedback** | id | INT | PRIMARY KEY | 唯一反馈ID |
|  | sender\_id | VARCHAR(32) | NOT NULL | 发送者ID（用户/管理员） |
|  | receiver\_id | VARCHAR(32) | NOT NULL | 接收者ID（用户/管理员） |
|  | content | TEXT | NOT NULL | 内容 |
|  | create\_time | VARCHAR(32) | NOT NULL | 创建时间 |
|  | is\_read | TINYINT | DEFAULT 0 | 是否已读 |
|  | type | VARCHAR(16) |  | 消息类型 |

1.主要实体与说明：

E\_User（用户）

存储用户账号及密码，用户可对电影进行评分、评论，或与管理员/其他用户互发消息反馈。

E\_Admin（管理员）

系统管理者账号，可进行系统数据维护、接收/发送反馈信息。

E\_Movie（电影）

存储电影基本信息，包括ID、片名、评分、年份、时长、类型等，是系统核心资源。

E\_Rating（评分）

记录用户对电影的评分，关联用户和电影，支持推荐算法和数据统计。

E\_Comment（评论）

用户对电影的评论信息，包含评论内容、时间、发布者，关联用户和电影，支撑前台评论区功能。

E\_People（人物）

存储与电影相关的主创、演员等信息，为影片人物统计和展示提供支持。

E\_Feedback（反馈）

用户或管理员间的消息/反馈记录，包括内容、类型、已读状态、发送/接收人及时间等，提升平台互动性与服务质量。

2.主要关系：

用户可以评分、评论电影，并可以发送/接收反馈信息；

管理员可与用户进行消息/反馈互动；

电影可以被多个用户评分、评论，且可包含多位相关人物；

评分和评论都分别通过外键与用户、电影进行关联，保证数据溯源与完整性。

# 3 编码实现

3.1登录模块

本系统登录模块为用户和管理员提供了统一、便捷且安全的身份验证入口。用户通过登录页面输入角色、用户ID与密码，经过后端验证后可进入各自权限范围内的系统功能区。登录模块前端界面如图所示。  
 用户访问登录页面时，前端会展示角色选择（用户/管理员）、用户ID和密码输入框。用户填入信息后点击登录按钮，前端通过 HTTP POST 请求，将表单数据提交给后端 Flask Web 服务的 /login 接口。  
 后端接收到登录请求后，首先判断登录角色。如果为管理员，则校验账号和密码是否匹配管理员预设信息。若为普通用户，则根据输入的用户ID在评分数据（ratings.csv 或数据库）中查询该用户是否存在，并校验密码。用户ID与密码匹配成功后，系统将用户的登录信息写入 Session，包括角色标识、用户ID等，确保后续请求能够正确识别用户身份和权限。

关键代码

@app.route('/login', methods=['GET', 'POST'])

def login():

if request.method == 'POST':

role = request.form.get('role')

username = request.form.get('username')

password = request.form.get('password')

if role == 'admin' and username == 'admin' and password == 'password':

session['logged\_in'] = True

session['role'] = 'admin'

return redirect(url\_for('admin\_dashboard'))

elif role == 'user' and password == 'user':

try:

user\_id = int(username)

if user\_id in ratings['user\_id'].unique():

session['logged\_in'] = True

session['role'] = 'user'

session['user\_id'] = user\_id

return redirect(url\_for('recommend\_user'))

else:

return render\_template('login.html', error="用户ID不存在")

except ValueError:

return render\_template('login.html', error="用户ID格式错误")

return render\_template('login.html', error="账号或密码错误")

return render\_template('login.html')

3.2 AI助手模块



本系统AI助手模块为用户和管理员提供了智能化的自然语言问答与数据查询服务。用户可在系统主界面通过AI助手浮窗输入任意问题，包括电影推荐、数据统计、评论检索等，AI助手能够理解用户意图，并结合本地数据库和大模型推理，自动生成友好、个性化的回复。AI助手界面如图所示。  
 用户通过AI助手输入问题后，前端将消息通过HTTP POST请求发送到后端Flask Web服务的/chatbot\_ask接口。后端首先对用户输入的文本进行正则或NLP解析，判断是否可以通过本地数据库（如movies.csv、ratings.csv、comments.csv）直接查询得到答案。如果可以查询，系统将检索到的本地数据与原始问题一并整理，通过RESTful API调用接入的AI大模型（DeepSeek），生成更加自然、智能的答案并返回给前端。如果本地数据无法满足查询需求，则直接将用户问题提交给AI大模型，由其智能生成回复。  
 AI助手模块设计中，后端会根据不同的意图自动切换本地检索与云端推理，保证效率与智能性兼顾。所有AI助手交互均需用户登录后才可访问。系统对外请求大模型API时采用安全认证机制，防止数据泄露和接口滥用。

关键代码

@app.route('/chatbot\_ask', methods=['POST'])

def chatbot\_ask():

msg = request.json.get('msg', '').strip()

if not msg:

return jsonify({'reply': '请说点什么吧！'})

local\_answer = None

try:

from data\_loader import load\_data

movies, ratings = load\_data()

comments = load\_comments()

# 1. 多少部电影

if re.search(r'多少部电影', msg):

local\_answer = f"目前系统收录了 {len(movies)} 部电影。"

# ...无限扩展

except Exception as e:

print("AI助手查本地数据异常：", e)

local\_answer = None

# RAG部分

if local\_answer:

system\_prompt = (

"你是电影推荐系统的AI助手。"

"每次我会给你用户原始问题，以及根据数据库查到的事实。"

"请你用自然口吻，把事实融入对用户的友好回答中。"

)

user\_prompt = (

f"用户提问：{msg}\n"

f"系统查到的数据：{local\_answer}\n"

"请结合两者自然作答（如有评价可适当加鼓励/引导）。"

)

else:

system\_prompt = (

"你是电影推荐系统的AI助手，请尽量用自然语言回答电影相关问题。"

)

user\_prompt = msg

headers = {

"Authorization": f"Bearer {DEEPSEEK\_API\_KEY}",

"Content-Type": "application/json"

}

payload = {

"model": "deepseek-chat",

"messages": [

{"role": "system", "content": system\_prompt},

{"role": "user", "content": user\_prompt}

],

"temperature": 0.4,

"max\_tokens": 256

}

try:

resp = requests.post(DEEPSEEK\_API\_URL, headers=headers, json=payload, timeout=20)

resp.raise\_for\_status()

result = resp.json()

reply = result['choices'][0]['message']['content'].strip()

return jsonify({'reply': reply})

except Exception as e:

print("DeepSeek API error:", e)

return jsonify({'reply': '抱歉，AI接口出错了，请稍后再试～'})

3.3 协同算法推荐模块

本系统协同算法推荐模块以用户协同过滤（UserCF）为核心，为每位用户提供个性化的电影推荐服务。用户在系统中对电影进行评分或浏览后，系统会记录其行为数据，并与所有用户的历史评分数据一同用于构建用户兴趣模型。推荐模块界面如图所示。  
用户进入推荐页面时，系统会自动获取当前用户ID，并通过后端推荐算法计算出一组可能感兴趣的电影列表。后端接收到推荐请求后，首先提取当前用户已评分的电影集合，然后根据所有用户的评分矩阵，计算当前用户与其他用户之间的兴趣相似度。相似度通常采用皮尔逊相关系数或余弦相似度衡量。  
 系统根据用户相似度，筛选出与当前用户兴趣最接近的若干“邻居用户”，进一步统计这些邻居用户高度评价且当前用户未看过的电影。结合这些信息，系统为用户生成排序后的推荐电影列表，通过前端页面动态展示。  
 推荐模块的算法逻辑在保证推荐准确性的同时，兼顾系统响应速度与扩展性。所有计算均基于本地评分数据集，支持大规模用户和电影数据处理。模块采用缓存优化与分批计算。用户可以根据自己的观看和评分习惯不断优化推荐结果，系统也支持管理员对推荐算法参数进行调整和升级。

关键代码

def calculate\_similarity(self, force\_recompute=False):

"""

计算用户相似度矩阵（带Redis缓存支持）

参数:

force\_recompute (bool):

"""

# 如果强制重新计算，先清除缓存

if force\_recompute:

self.clear\_cache()

# 尝试从Redis加载缓存

if self.redis\_client and not force\_recompute:

try:

# 检查分块存储模式

if self.redis\_client.exists(f"{self.cache\_key}:meta"):

print("[Redis] 从分块缓存加载相似度矩阵...")

self.user\_similarity = self.\_load\_matrix\_chunks()

return

# 检查整体存储模式

cached\_data = self.redis\_client.get(self.cache\_key)

if cached\_data:

print("[Redis] 从缓存加载相似度矩阵")

self.user\_similarity = self.\_deserialize\_matrix(cached\_data)

return

except Exception as e:

print(f"[Redis] 缓存加载失败: {e}, 将重新计算")

# 无缓存或加载失败时重新计算

print("计算用户相似度矩阵...")

norm\_ratings = self.user\_movie\_ratings.sub(self.user\_movie\_ratings.mean(axis=1), axis=0)

filled = norm\_ratings.fillna(0)

# 计算皮尔逊相关系数

numerator = filled @ filled.T

denominator = np.sqrt((filled \*\* 2).sum(axis=1).values[:, None] @

(filled \*\* 2).sum(axis=1).values[None, :])

denominator[denominator == 0] = np.inf

self.user\_similarity = numerator / denominator

# 确保是NumPy数组（修复关键错误）

if isinstance(self.user\_similarity, pd.DataFrame):

self.user\_similarity = self.user\_similarity.values

# 将对角线设为零（排除自相似度）

np.fill\_diagonal(self.user\_similarity, 0)

# 存储到Redis

if self.redis\_client:

try:

n\_users = len(self.user\_similarity)

# 大型矩阵使用分块存储

if n\_users > self.chunk\_size:

print(f"[Redis] 分块存储相似度矩阵 ({n\_users}用户)...")

self.\_store\_matrix\_chunks(self.user\_similarity)

else:

# 小型矩阵整体存储

print("[Redis] 存储相似度矩阵到缓存...")

self.redis\_client.setex(

self.cache\_key,

86400, # 24小时TTL

self.\_serialize\_matrix(self.user\_similarity)

)

except Exception as e:

print(f"[Redis] 缓存存储失败: {e}")

3.4 用户其他模块

我的已评分



电影按条件筛选



电影评论

个人中心

成就系统



消息反馈/通知



3.5 数据分析模块



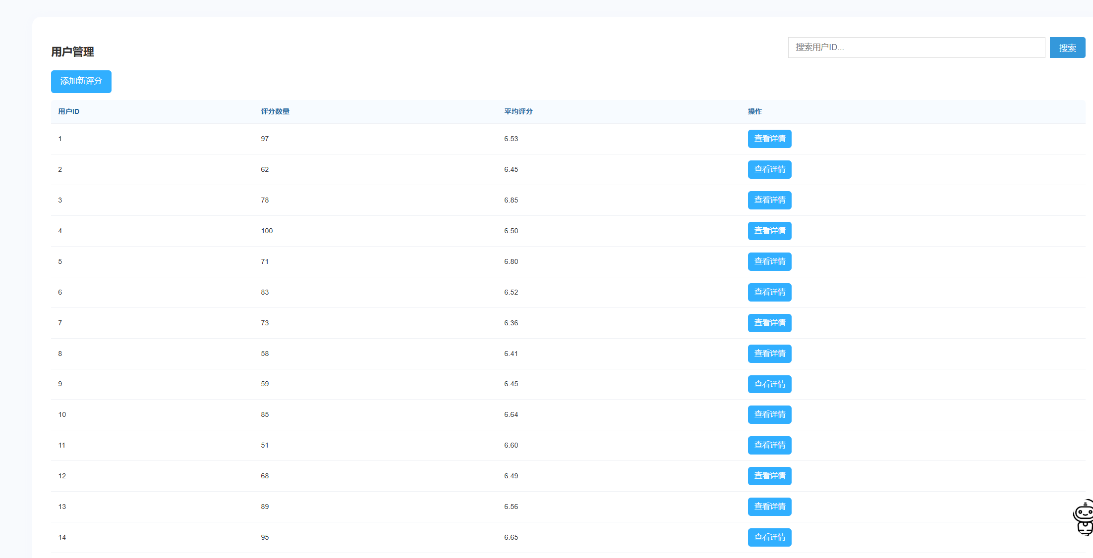
本系统的数据分析模块为管理员提供了丰富的可视化统计报表，帮助其全面掌握电影库、用户行为和系统活跃度等核心数据。仪表盘界面集成了电影总数、用户总数、评分数量、平均评分等关键业务指标，并通过图表形式动态展示电影时长分布、评分分布、活跃用户排名、热门类型分布及评分时间趋势等多维度数据。管理员可通过仪表盘直观了解系统运行状况，及时发现业务增长点与异常波动。  
 系统的数据分析功能还包括电影人物统计模块。该模块支持对电影相关人物的职业分布、出生年代、寿命等特征进行统计分析。管理员可以查看主要职业（如导演、演员、编剧）的人数分布，分析不同年代出生人物的数量变化趋势，统计电影人物的平均寿命、最大寿命等指标。系统同时提供每部电影涉及人物数量排行，帮助管理员发现重要作品与核心人物。  
 数据分析模块采用Chart.js等可视化工具，所有统计数据均由后端预处理后以JSON格式传递给前端，保证报表渲染效率和交互体验。管理员可通过筛选条件、时间段等灵活定制统计视图，实现对电影库和用户群体的深度洞察。

关键代码

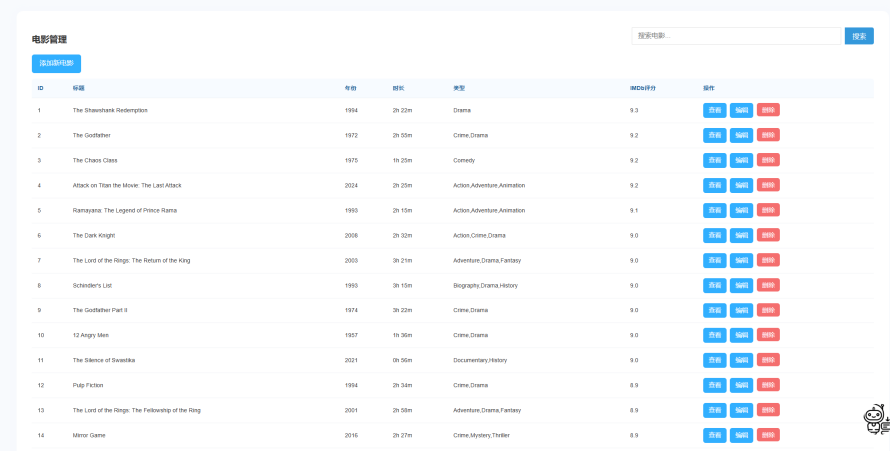
@app.route('/admin/movie/<int:movie\_id>')  
def admin\_movie\_detail(movie\_id):  
 *"""电影详情管理路由"""* if movies is None or ratings is None:  
 return render\_template('error.html', message="数据加载失败，请检查数据文件")  
  
 try:  
 # 获取指定电影  
 movie = movies[movies['movie\_id'] == movie\_id].iloc[0].to\_dict()  
  
 # 获取该电影的评分数据  
 movie\_ratings = ratings[ratings['movie\_id'] == movie\_id]  
  
 # 计算评分分布  
 rating\_dist = {}  
 if not movie\_ratings.empty:  
 rating\_dist = pd.cut(  
 movie\_ratings['rating'],  
 bins=[1, 3, 5, 7, 9, 10],  
 labels=['1-3分', '3-5分', '5-7分', '7-9分', '9-10分'],  
 right=False  
 ).value\_counts().sort\_index().to\_dict()  
  
 # 获取相关人物信息  
 people\_info = []  
 people\_path = os.path.join('data', 'people.csv')  
 if os.path.exists(people\_path):  
 people\_df = pd.read\_csv(people\_path)  
 movie\_people = people\_df[people\_df['movie\_id'] == movie\_id]  
  
 if not movie\_people.empty:  
 people\_info = movie\_people[[  
 'primaryName',  
 'birthYear',  
 'deathYear',  
 'primaryProfession'  
 ]].fillna('未知').to\_dict('records')  
  
 return render\_template('admin\_movie\_detail.html',  
 movie=movie,  
 rating\_dist=rating\_dist,  
 people\_info=people\_info)  
  
 except IndexError:  
 return render\_template('error.html', message="找不到指定的电影")  
 except Exception as e:  
 return render\_template('error.html', message=f"获取电影详情时出错: {str(e)}")

3.6管理模块

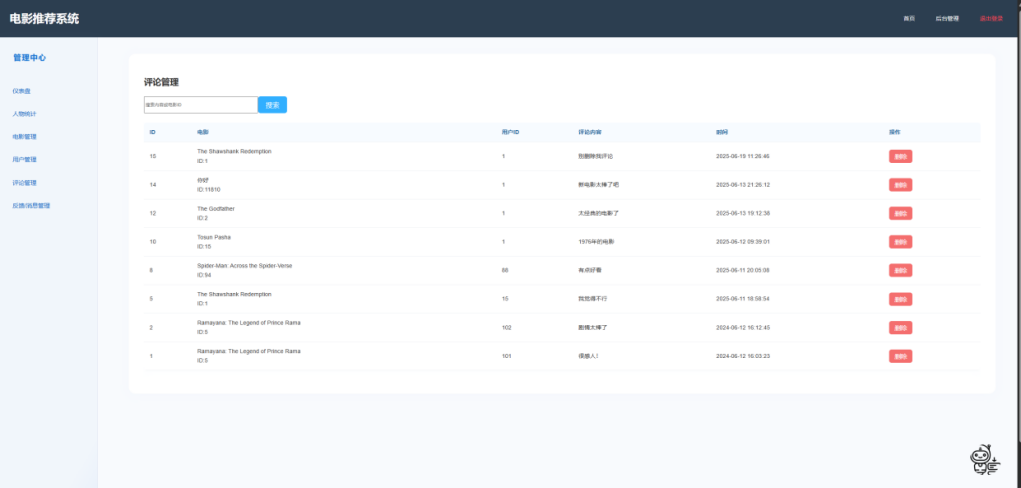
用户管理模块



电影管理模块



评论管理模块



 消息/反馈管理模块

本系统管理模块为管理员提供了对用户、电影、评论和反馈消息等核心业务数据的统一管理入口。管理员通过后台管理界面，可以对所有用户账号进行查询、分页浏览、评分统计、详情查看等操作，支持对用户信息的增删查改，便于实现用户分层管理和异常行为监控。  
 在电影管理方面，管理员可对电影库中的影片进行增删改查，包括录入新电影、编辑影片信息、删除失效或违规电影，支持按照影片类型、年份、评分等多条件筛选和批量操作。系统还为每部电影提供详细信息和关联人物展示，方便管理员进行内容质量控制。  
评论管理模块支持对全站所有用户评论的审核与删除，管理员可按电影或内容关键词检索评论，对违规、垃圾评论及时处理，维护社区秩序。评论数据支持与电影、用户信息的关联显示，实现多维度的内容监管。  
 反馈与消息管理功能实现了用户与管理员之间的即时沟通。普通用户可通过消息中心提交反馈或建议，管理员在后台可实时查看所有反馈信息，并主动向指定用户发送系统通知。所有消息按时间倒序排列，支持已读未读状态标记和多条件检索，提升沟通效率和响应速度。  
管理模块基于Flask后端和MySQL/CSV存储，采用权限校验机制确保所有敏感操作仅限管理员账号访问，有效保障系统数据安全和运维规范性。

# 4 测试

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **测试用例编号** | **测试功能** | **操作步骤** | **预期结果** | **实际结果** |
| **TC-01** | 用户登录 | 输入正确用户ID和密码登录 | 跳转至首页 | 通过 |
| **TC-02** | 管理员登录 | 输入admin/admin密码登录 | 进入后台管理页面 | 通过 |
| **TC-03** | 个性化推荐 | 普通用户登录后查看推荐页面 | 展示用户专属推荐电影 | 通过 |
| **TC-04** | 热门电影展示 | 登录后首页查看热门电影 | 各个部分展示清晰无误 | 通过 |
| **TC-05** | 电影筛选 | 按类型、年份、评分筛选电影 | 正确筛选并分页展示 | 通过 |
| **TC-06** | 电影评论 | 搜索电影并提交评论 | 评论显示在该电影评论区 | 通过 |
| **TC-07** | 后台增删电影 | 管理员新增/删除电影 | 数据同步更新、前后台同步显示 | 通过 |
| **TC-08** | 后台数据分析 | 后台查看各类统计图表 | 图表正常渲染并数据准确 | 通过 |
| **TC-09** | 反馈功能 | 测试用户和管理员之间的信息传递 | 用户管理员均能收到对方发送的信息 | 通过 |

# 

# 5 安装及部署

# PLLVJzjM57_tfx2OLBNIYCkglUm15T6YCe7625iVHXzCV1Co4XlPXWfDamX_IkZo9om4IWMKK6YpJiImjIHDml9XbdjjF_KhxFYU17ib3zOzzlcT-pk_yoVt3LFMpOLaGf0ZfgQBVVPsXQLhabqjsuLdTEl5j7gRVSWxRzDqhtAdJvGDSJWH9wef2Agc4B6FDhOzo7wMF (1)

# 

# 6 项目总结（团队）

**6.1 项目开发感悟与提升**

在本次电影推荐系统的开发过程中，团队成员密切合作，积极配合，按照既定计划高效地完成了各自承担的模块开发任务。在项目推进期间，我们不仅进一步提升了个人的技术能力，也在实际工作中不断锻炼了团队协作和沟通能力。

项目开发过程中，我们逐渐学会了将复杂任务细分为具体可执行的工作项，合理进行分工与资源分配。同时，针对项目过程中出现的各种难题，如功能需求变更、接口联调、系统集成等，团队成员能主动提出问题并协同攻关，有效保证了项目的进度和质量。

通过本项目，团队成员对 Python Web 开发、数据预处理、推荐算法实现等核心技术有了更深入的理解和实践体验，也更加熟悉了从需求分析、系统设计到代码实现与测试的完整开发流程。这为未来参与更大规模、更高复杂度的软件工程项目打下了坚实的基础。

此外，在项目推进过程中，我们体会到沟通和文档的重要性，学会了利用微信、在线文档等工具同步项目进度和变更，极大地提高了团队的协作效率。通过这次实践，大家的项目管理能力和自我驱动力均有了显著提升。

**6.2 主要问题与解决方法**

6.2.1 需求变更同步困难

项目开发初期，由于需求多次调整，成员一度对最新需求不甚了解，导致进度不同步和开发方向偏差。针对此问题，团队负责人及时制定了需求变更同步机制：每次讨论后立即整理变更要点，并通过微信和项目文档进行实时同步，确保所有成员能够随时查阅和理解最新需求。这一举措显著提升了团队内部的沟通效率和响应能力。

实际开发中，需求变更也会带来文档及代码的相应调整，尤其是功能模块的扩展。最初团队成员对流程不熟悉，但在组长的带领下，大家逐步掌握了如何在 Flask 应用中增加新路由、完善相应的 HTML 前端页面，并通过不断实践提升了系统性开发能力。

6.2.2 前后端接口对接不畅

项目进入联调阶段后，由于前后端成员对部分数据结构和接口格式理解不一致，导致初次对接时页面数据展示混乱。针对该问题，接口负责人重新梳理了接口规范，组织全员审阅并统一了数据传递格式。开发过程中还通过构造模拟数据和多轮测试，逐步完善了接口文档，最终实现了前后端的顺利对接和数据正确展示。

6.2.3 技术难点攻克

部分推荐算法、数据预处理和前端可视化等技术点对团队而言具有一定挑战。针对这些难点，团队成员查阅相关文献和技术资料，积极交流解决思路。最终通过反复测试和优化，保证了系统的功能完整性和性能达标。

**6.3 后续升级与展望**

针对当前项目的实际应用效果与用户反馈，团队对系统未来的优化和升级方向进行了初步规划：

算法优化：计划在现有协同过滤基础上，引入基于内容的推荐或混合推荐模型，提升系统的个性化推荐效果；

用户功能拓展：丰富用户端体验，增加“想看/已看”列表、短评、收藏等功能，增强用户活跃度与粘性；

后台管理增强：完善后台权限管理，支持更精细的角色划分和批量操作，便于大规模数据的高效管理和维护；

商业化及云端部署：如有进一步推广计划，可考虑对接第三方数据源，实现云端部署与横向扩展；同时探索接入广告推荐等商业模式，实现项目收益的可持续增长；

系统安全与稳定性：后续将加强数据安全与备份策略，提升系统抗风险能力，保障用户数据的安全与系统的长期稳定运行。