



中级软件实作

院系： 软件学院

专业： 软件工程

小组成员姓名：

杨富广 20212005014

蔡钊20213332026

罗熹 20212005035

陈伟创20211131048

叶昕廷20213332019

2024年 1月 20 日

目录

[需求分析 3](#_Toc156761618)

[1.1目的 3](#_Toc156761619)

[1.2 需求调查分析 3](#_Toc156761620)

[1.3 功能性需求 4](#_Toc156761621)

[1.4 非功能性需求 6](#_Toc156761622)

[1.5技术框架 7](#_Toc156761623)

[项目可行性分析 9](#_Toc156761624)

[2 可行性研究的前提 9](#_Toc156761625)

[3 对现有系统的分析 11](#_Toc156761626)

[4 所建议的系统 11](#_Toc156761627)

[项目设计 12](#_Toc156761628)

[电影网站概要设计 12](#_Toc156761629)

[电影网站组织结构图的数据流向 14](#_Toc156761630)

[电影网站的详细设计 16](#_Toc156761631)

[项目实现 29](#_Toc156761632)

[开发工具和环境 29](#_Toc156761633)

[模块与组件实现 31](#_Toc156761634)

[首页（index） 31](#_Toc156761635)

[搜索模块（search） 32](#_Toc156761636)

[三、排行榜(rank) 34](#_Toc156761637)

[四、电影详情+播放(detail) 35](#_Toc156761638)

[五．注册 35](#_Toc156761639)

[六.登录 36](#_Toc156761640)

[七．用户中心 36](#_Toc156761641)

[八．管理员页面 37](#_Toc156761642)

[项目分工说明 39](#_Toc156761643)

# 需求分析

### 1.1目的

本项目旨在开发一个电影网站，通过提供丰富、便捷的观影服务，满足人们日常的娱乐需求。  
电影作为一种流行的娱乐方式一种强大的文化载体，它不仅展示了各种文化背景和历史时期，还能促进不同文化之间的理解和交流。通过电影，观众可以了解并体验到不同国家和地区的风土人情、传统习俗和生活方式，从而拓宽视野，增进对世界的认知。同时电影产业是一个重要的经济部门，涉及制作、发行、放映等多个环节，为社会创造大量的就业机会和经济收入。我们期望通过此网站实现以下几个核心目标：

1.为用户提供一个集电影信息查询、观影推荐及社交互动于一体的平台。

2.提高观影体验，让用户能够更方便地发现和观看他们感兴趣的电影。

3.推动电影文化的普及与传播，让更多的人享受到电影带来的乐趣。

### 1.2 需求调查分析

背景分析

随着数字媒体时代的到来，电影资源变得日益丰富，但同时也面临着资源分散和信息混乱的问题。市场调研显示，用户在寻找和观看电影时，常常感到困惑和不便，主要原因在于现有的电影资源分布在不同的平台，缺乏有效的整合。这不仅影响了用户的观影体验，也制约了电影行业的进一步发展。

用户需求概述

通过对用户反馈的分析，我们发现以下几点需求尤为突出：

资源整合：用户希望能在一个平台上访问到广泛的电影资源，包括不同类型、时期、国家的电影。

便捷的检索功能：用户需要能快速找到他们想看的电影，这要求网站具备强大的搜索和分类功能。

高质量的内容：用户期望网站提供的电影资源质量高，包括高清晰度的视频和准确的电影信息。

个性化推荐：用户希望根据自己的观影历史和偏好，获得个性化的电影推荐。

开发目标

构建全面的电影数据库：集成多种来源的电影资源，建立一个全面、丰富的电影数据库，覆盖各种类型、时期、语言的电影。

优化资源归纳与分类：开发高效的电影分类系统，便于用户根据类型、年份、地区等不同维度快速检索电影。

提供高效的搜索引擎：实现一个强大的搜索引擎，使用户能够通过关键词、演员名、导演名等多种方式快速找到目标电影。

1.3 功能性需求  
1.3.1 用户账户管理

**会员注册**：用户通过提供邮箱地址、用户名和密码来注册账户。系统通过数据库的账号密码信息进行比对以确保账户安全。。

**登录和退出**：用户登录时需输入用户名和密码，退出功能应清除所有会话信息，确保账户安全。

**会员等级**：分为VIP会员和普通账号。VIP会员享受可观看高级影片资源，高清播放等特权。

1.3.2 影片展示

**按热播排行展示**：展示当前最受欢迎的电影，基于观看次数和用户评分计算热度。

**按类型展示**：将电影分类（如动作、喜剧、科幻等），用户可以选择感兴趣的类型浏览。

**按地区展示**：根据电影的制作地区（如美国、中国、欧洲等）进行分类。

按电影基因展示：基于电影的特定元素（如导演、演员、主题等）分类。

1.3.3 电影排行榜

本周/本月/全部排行：根据时间范围显示不同的排行榜，提供用户不同的选择视角。

**按好评排行**：根据用户评分和评论数量进行排名，鼓励高质量内容的生产。

1.3.4 主创作品检索

**演员检索功能**：用户可以通过输入演员名字，快速找到该演员参与的电影。系统应提供自动补全功能，优化用户体验。

1.3.5 权限控制

**播放权限**：区别VIP会员和普通用户的权限，如VIP会员可观看高清或独家内容。

[[1]](#footnote-0)图1.1 电影网站用例图

### 1.4 非功能性需求

1.4.1性能需求

响应时间：界面响应时间不超过2秒，确保用户体验流畅。

并发处理：系统应能支持至少1000名并发用户，保持稳定的响应时间。

优化加载：Vue前端开发，应用代码分割以减少初始加载时间。

1.4.2.可用性需求

用户界面：简洁直观的界面设计，确保用户易于导航和操作。

适应性：前端设计应适应不同大小和分辨率的屏幕，包括移动设备。

国际化：支持多语言界面，以服务不同国家和地区的用户。

1.4.3. 安全性需求

数据加密：使用HTTPS协议加密用户数据传输。

1.4.4. 可维护性和扩展性需求

代码质量：确保代码具有良好的注释和文档，便于维护和更新。

模块化设计：采用模块化设计，使得未来添加新功能或修改现有功能更为容易。

API文档：提供详细的API文档，便于后端服务的扩展和维护。

1.4.5. 兼容性需求

浏览器兼容：确保网站在主流浏览器（如Chrome, Firefox, Safari等）上的兼容性。

数据库兼容：后端应能够与不同的数据库系统（如MySQL, PostgreSQL等）兼容。

1.4.6. 备份和灾难恢复

数据备份：定期备份数据库，以防数据丢失。

恢复计划：制定灾难恢复计划，以快速恢复服务于意外情况。

### 1.5技术框架

前端技术：Node.js 和 Vue 的应用与优化。

后端技术：Spring Boot 的结构和配置

#### 1.6数据模型和数据库设计

actor（演员）表

字段：movie\_id（电影ID，整型），main\_actor（主演，字符串）。

主键：movie\_id 和 main\_actor 的组合。

描述：关联电影和其主要演员。

lunbo（轮播）表

字段：lunbo\_id（轮播ID，整型），movie\_id（电影ID，整型），picture（图片，字符串）。

主键：lunbo\_id。

描述：展示电影的轮播图片。

member（会员）表

字段：email（电子邮件，字符串），password（密码，字符串），pm（整型），user\_type（用户类型，整型）。

主键：email。

描述：存储会员的账户信息，包括登录凭证和会员类型。

movie（电影）表

字段：movie\_id（电影ID，整型），movie\_name（电影名称，字符串），genre（类型，字符串），production\_place（制作地点，字符串），date（日期），rating（评分），week\_hits（周点击量），month\_hits（月点击量），all\_hits（总点击量），picture（图片，字符串），introduction（介绍，字符串），video\_url（视频URL，字符串），video\_vip（VIP视频，字符串）。

主键：movie\_id。

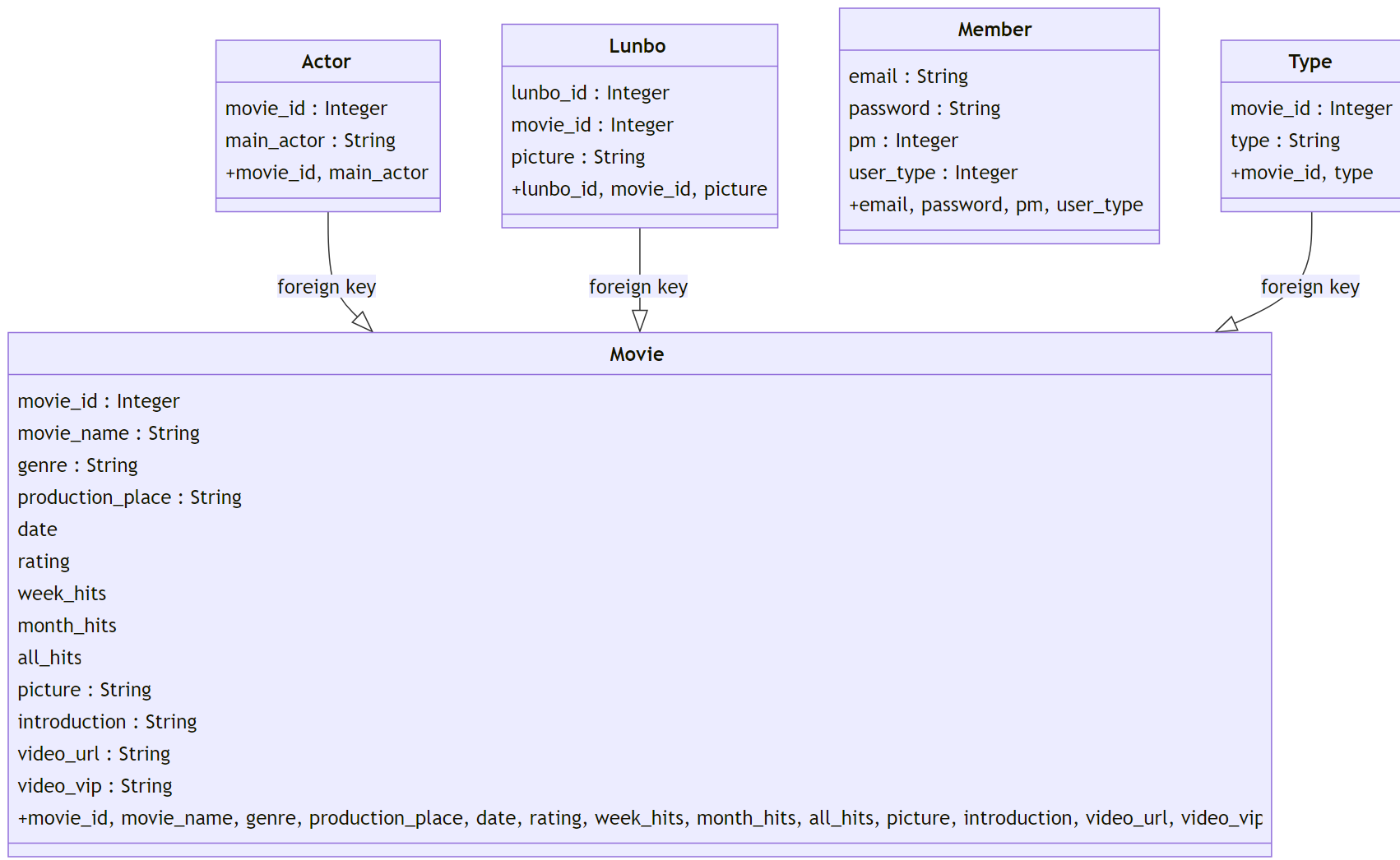
描述：电影的详细信息，包括名称、类型、评分和相关媒体资料。

type（类型）表

字段：movie\_id（电影ID，整型），type（类型，字符串）。

主键：movie\_id 和 type 的组合。

描述：电影的分类信息，一个电影可能有多个分类。

[[2]](#footnote-1)图1.2 数据库设计类图

# 项目可行性分析

### 2 可行性研究的前提

.2.1 **要求**

功能：包括用户账户管理、影片展示、电影排行榜、主创作品检索和权限控制。

性能：响应时间不超过2秒，支持至少1000名并发用户。

输出：生成用户行为报告、电影排行数据报告，每日更新，供管理员和系统分析师使用。

输入：用户数据、电影数据，来源包括用户输入和外部电影数据库，数据量大，结构化。

处理流程和数据流程：用户操作触发数据请求，系统处理后返回相应结果，涉及前端交互和后端数据处理。

安全与保密：强化用户数据保护，确保交易和个人信息安全。

相关系统：与电影数据库、支付网关等外部系统交互。

完成期限：预计项目周期为12个月。

2.2 **目标**

费用减少：通过自动化和优化操作减少人力成本。

处理速度提高：优化后台算法和数据库操作，提高处理速度。

管理信息服务改进：提供更精准的用户数据分析和电影推荐。

自动决策系统改进：利用AI算法优化电影推荐系统。

人员利用率改进：通过自动化减少日常维护工作，提高人员效率。

2.3 **条件、假定和限制**

运行寿命：至少5年。

经费限制：总预算不超过100万美元。

法律和政策：遵守相关版权法和网络安全法。

硬件和软件条件：需要高性能服务器，使用现有开发框架和工具。

最晚投入使用时间：预计在12个月后上线。

2.4 **进行可行性研究的方法**

方法：结合市场调研、技术评估、财务分析。

评价：基于市场需求、技术实现可能性、投资回报率进行评价。

2.5 **评价尺度**

费用：项目总成本控制在预算内。

功能优先次序：优先实现核心功能，如用户管理、影片展示。

开发时间：确保在12个月内完成。

使用难易程度：用户界面友好，易于操作。

### 3 对现有系统的分析

3.1 **处理流程和数据流程**

现状：现有系统可能功能单一，用户体验不佳。

流程：当前处理流程可能过于复杂或不够高效。

3.2 **工作负荷**

现有系统：可能承担了基本的电影信息展示和搜索功能。

3.3 **费用开支**

现状：维护现有系统可能导致较高的运营成本。

3.4 **人员**

需求：需要技术人员维护现有系统，但可能缺乏专业知识。

3.5 **设备**

现有：可能使用过时的服务器和存储设备。

3.6 **局限性**

问题：处理速度慢，功能有限，无法满足市场需求。

### 4 所建议的系统

4.1 **对所建议系统的说明**

系统描述：一个综合性的电影网站，提供高效的搜索、分类、推荐和社交互动功能。

满足要求：通过使用现代化技术栈和高效的数据库管理。

4.2 **处理流程和数据流程**

设计：更高效的数据处理和用户交互流程。

4.3 **改进之处**

改进：用户体验、数据处理速度、推荐系统的准确性。

4.4 **影响**

设备：需要更高配置的服务器和存储设备。

软件：需要更新的操作系统和数据库管理软件。

用户单位机构：可能需要额外的人员培训和技术支持。

运行过程：简化用户操作，优化数据存储和恢复过程。

4.5 **局限性**

现有局限：预算限制可能导致某些高级功能无法实现。

4.6 **技术条件方面的可行性**

可行性：在当前技术和预算条件下，大部分功能目标可实现。

开发时间：团队有能力在规定时间内完成开发。

# 项目设计

### 电影网站概要设计

1. 系统架构

前端：使用Node.js和Vue构建。负责用户界面展示和用户交互。

后端：使用Spring Boot。处理业务逻辑、数据库交互和外部API调用。

数据库：存储用户数据、电影信息、排行榜等。

2. 主要模块

用户账户管理模块

功能：用户注册、登录、退出。

关联：直接与数据库交互，管理用户信息。

电影展示模块

功能：展示电影列表，支持按热度、类型、地区等分类。

关联：连接到电影数据库，获取电影数据。

电影排行榜模块

功能：展示不同类别（如本周、本月、全部、好评）的电影排行。

关联：基于用户行为和评分数据生成排行。

搜索和推荐模块

功能：提供电影搜索功能；根据用户偏好推荐电影。

关联：使用搜索算法和推荐系统算法。

权限控制模块

功能：区分VIP和普通用户的权限，控制对高级内容的访问。

关联：与用户账户管理模块紧密结合。

管理员后台

功能：内容管理、用户管理、数据分析等。

关联：提供网站维护和监控功能。

3. 数据流

用户操作：用户在前端进行操作（如浏览、搜索、播放）。

请求处理：后端接收请求，处理业务逻辑。

数据库交互：后端与数据库交互，获取或存储数据。

响应返回：后端处理完毕后，将结果返回给前端展示。

4. 安全性和性能

采用HTTPS协议保护数据传输。

对重要功能进行性能优化，确保快速响应。

5. 界面设计

简洁直观的用户界面。

适应不同设备和屏幕尺寸。

### 电影网站组织结构图的数据流向

用户账户管理

流入：用户提供的注册信息和登录凭证从用户界面流入。

流出：验证和状态更新信息流向数据库管理模块，以及用户状态信息（如登录成功或失败）回馈到用户界面。

电影展示

流入：用户的浏览请求从用户界面流入。

流出：电影数据从数据库管理模块流出到用户界面，供用户浏览。

搜索和推荐

流入：用户的搜索查询和历史观影记录从用户界面流入。

流出：搜索结果和推荐列表从数据库管理模块流出，显示在用户界面。

电影排行榜

流入：用户对电影的评分和观看数据从用户界面流入。

流出：排行榜信息（基于用户行为和评分计算）从数据库管理模块流出到用户界面。

管理员后台

流入：管理员的操作请求（如内容更新、用户管理）从管理员界面流入。

流出：系统反馈和数据报告流回管理员界面，同时管理操作的结果更新到数据库。

数据库管理

集中流入：来自用户账户管理、电影展示、搜索和推荐、电影排行榜的数据。

集中流出：将处理后的数据流向相应的模块，如更新的用户信息、电影数据、搜索结果等。

[[3]](#footnote-2) 图1.3 电影网站组织结构图

### 电影网站的详细设计

#### 用户登录模块

类：MemberController

职责：管理与用户认证相关的HTTP请求。

依赖关系：依赖于 MemberService 类来执行具体的业务逻辑。

方法描述

isLogin(HttpServletRequest, HttpSession)

检查当前用户是否登录，返回JSON字符串表示登录状态。

toLogin(HttpServletRequest, HttpSession)

导航至登录页面，根据用户登录状态返回相应视图。

doLogin(String, String, int, HttpServletRequest, HttpSession)

处理用户登录请求，验证用户凭证，成功后存储用户会话。

getUserInfo(Model, HttpServletRequest, HttpSession)

获取当前登录用户的信息，返回JSON格式的用户信息。

deleteUser(HttpServletRequest, HttpSession)

登出用户，清除用户会话。

toRegister()

导航至注册页面。

doRegister(String, String)

处理用户注册请求。

checkEmail(String)

验证邮箱是否已被注册。

2. 数据模型和存储

类：Member

属性：userID, email, password, userType 等。

方法：包括密码加密、验证等。

数据库

表格：Member

字段：userID, email, password, userType 等。

3. 安全性和错误处理

密码处理：密码在服务器端进行加密处理，保证安全性。

输入验证：验证邮箱和密码的格式和有效性。

错误处理：处理登录失败、邮箱已注册等情况，返回适当的错误信息。

4. 接口设计

登录接口：接收用户邮箱和密码，返回登录状态。

注册接口：接收用户邮箱和密码，执行注册流程。

状态检查接口：检查用户登录状态。

5. 用户界面交互

登录页面：用户输入邮箱和密码。

注册页面：用户输入注册所需信息。

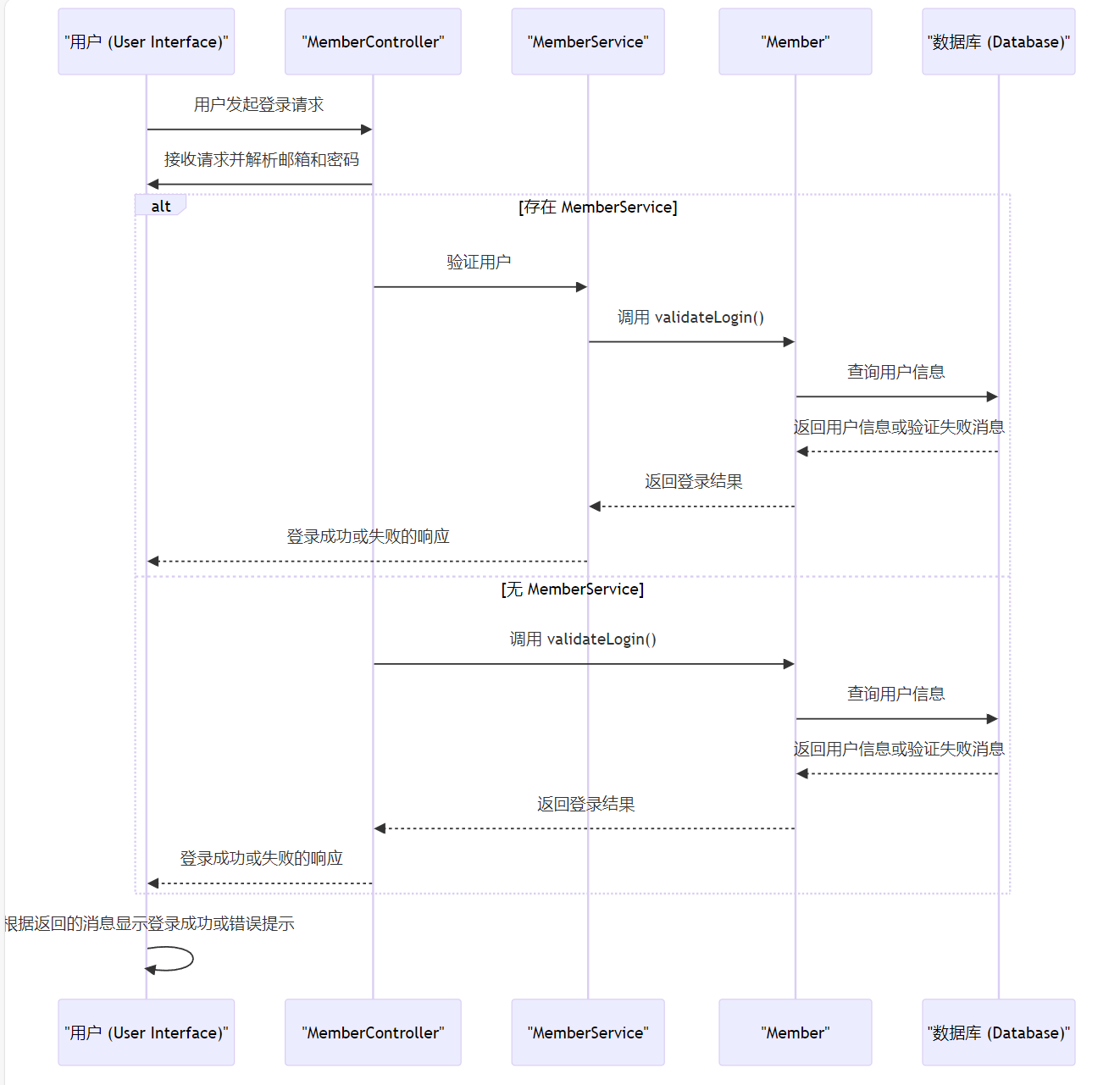
登录反馈：向用户展示登录成功或错误信息。

6. 业务流程

登录流程：用户提交凭证 → 验证凭证 → 存储会话/返回错误。

注册流程：用户提交信息 → 验证信息 → 注册用户/返回错误。

状态检查：查询当前会话 → 返回用户登录状态。

[[4]](#footnote-3) 图1.4 用户登录时序图

#### 搜索模块

类：SearchController

职责：管理电影搜索相关的HTTP请求。

依赖：依赖于 TypeService 和 MovieService 来获取电影类型、地区、基因和电影搜索结果。

方法描述

gettypes()

功能：获取电影类型列表。

返回：类型名称列表。

getareas()

功能：获取电影地区列表。

返回：地区名称列表。

getgenres()

功能：获取电影基因（特征）列表。

返回：基因名称列表。

index(Integer, String, Integer, Integer, String, String, String, String)

功能：根据给定的参数执行电影搜索。

参数：tag（标签）、text（文本）、pageNo（页码）、pageSize（每页大小）、type（类型）、area（地区）、g（基因）、p（偏好）。

返回：搜索结果的映射，包括电影列表和页码信息。

数据模型和存储

相关实体：Movie, Type, NewMovie 和 SearchMovieClass。

Movie：存储电影基本信息，如ID、名称、类型等。

SearchMovieClass：用于搜索结果的表示，包括电影详细信息和演员列表。

接口设计

提供了RESTful API端点进行搜索相关操作。

例如 /index/search 端点用于执行电影搜索。

安全和错误处理

跨域处理：使用 @CrossOrigin 注解处理跨域请求。

错误处理：合理处理搜索失败或无结果的情况，并返回适当的响应。

业务流程

用户提出搜索请求（包括类型、地区、特征等）。

SearchController 接收请求并调用相应的服务。

TypeService 和 MovieService 提供类型、地区、基因和电影数据。

结果返回给用户。

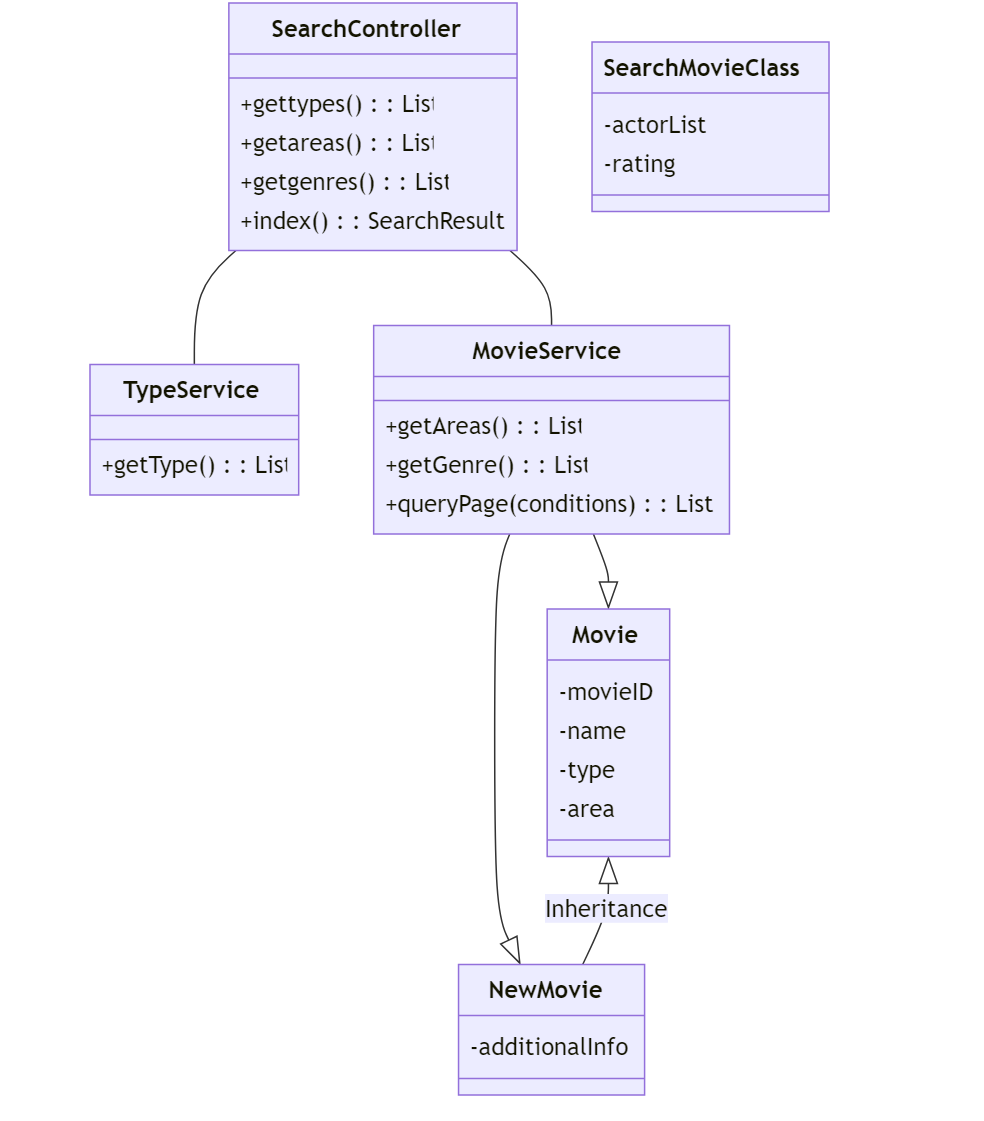


图 1.5搜索模块类图

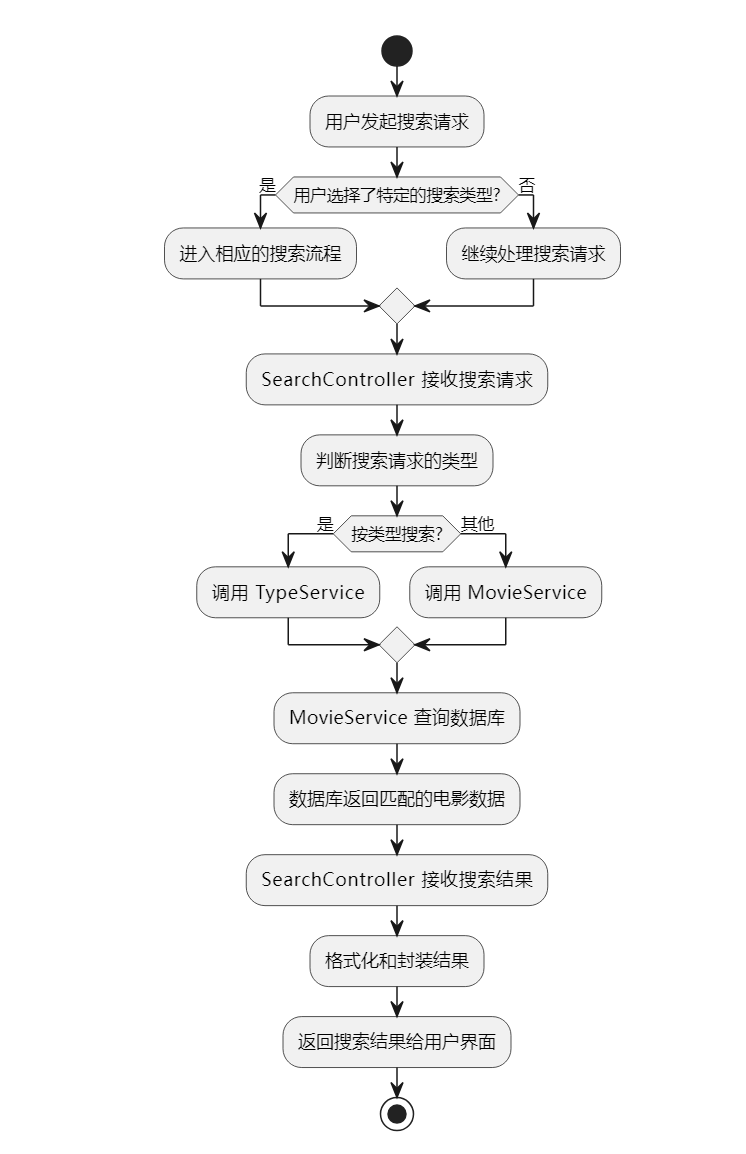


图1.6 搜索模块活动图

展示模块  
类设计：IndexController

职责：处理与电影展示相关的所有HTTP请求。

依赖：依赖于 MovieService、TypeService 和 LunboService 来获取电影数据。

方法描述

getlunboData

功能：获取轮播图数据。

返回：LunboInnerClass 数组。

getLatestMovieData  
功能：获取最新影片数据。

返回：MovieinnerClass 数组。

getweekHotMovieData

功能：获取每周热播影片数据。

返回：MovieinnerClass 数组。

getvipMovieData

功能：获取VIP影片数据。

返回：MovieinnerClass 数组。

getallMovieData

功能：获取所有影片数据。

返回：MovieinnerClass 数组。

gethotRankMovieData

功能：获取人气排行数据。

返回：HotRankMovie 数组。

getgoodRankMovieData

功能：获取好评排行数据。

返回：HotRankMovie 数组。

数据模型和存储

相关实体：LunboInnerClass、MovieinnerClass、HotRankMovie。

LunboInnerClass：用于表示轮播图数据。

MovieinnerClass：用于表示电影数据。

HotRankMovie：用于表示排行榜中的电影。

接口设计

提供了RESTful API端点进行电影展示相关操作。

例如 /index/latest 端点用于获取最新电影数据。

安全和错误处理

跨域处理：使用 @CrossOrigin 注解处理跨域请求。

错误处理：合理处理失败情况，并返回错误信息。

业务流程

用户提出特定的电影展示请求（轮播、最新影片、排行榜等）。

IndexController 接收请求并调用相应的服务层方法。

MovieService、TypeService、LunboService 提供电影相关数据。

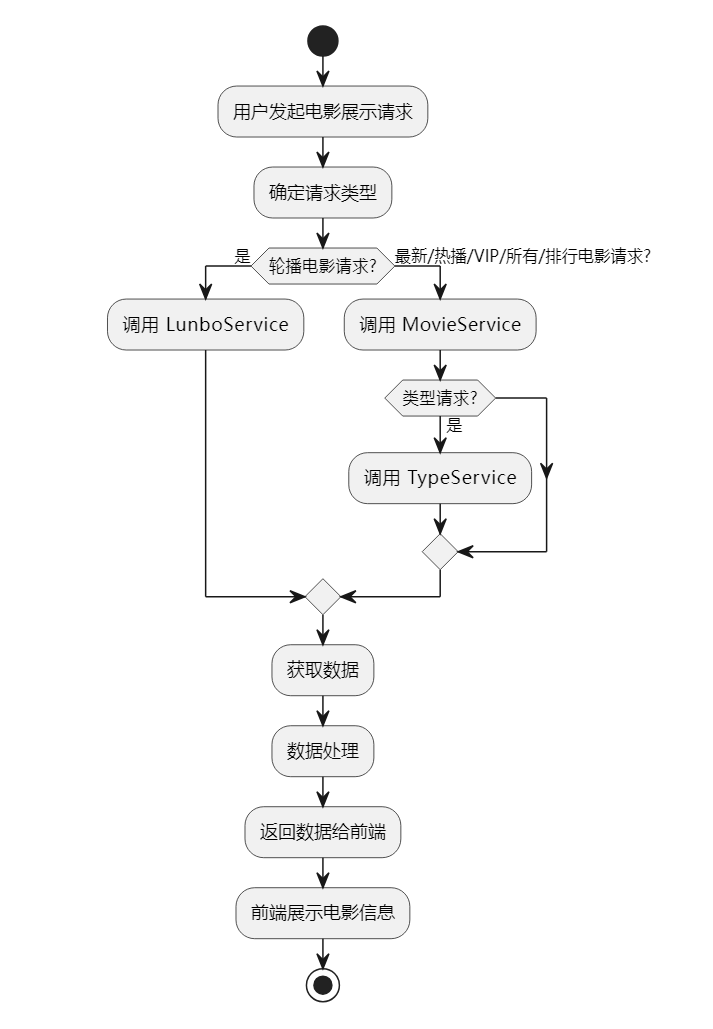


图 1.7 展示电影活动图

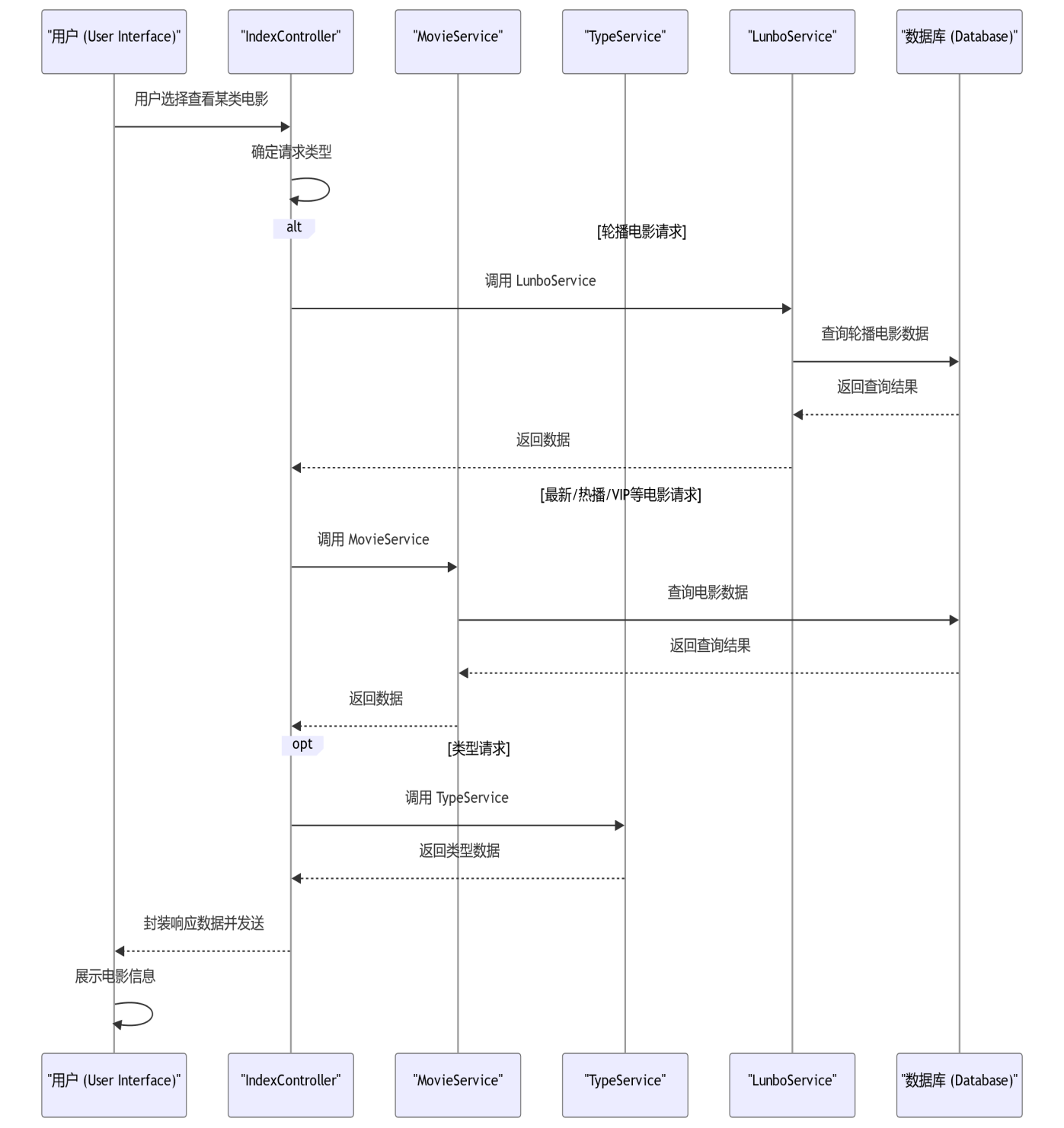


图1.8 轮播时序图

#### 排序模块

类：RankController

职责：管理电影排行相关的HTTP请求。

依赖：依赖于 MovieService 和 TypeService 来获取排序后的电影数据。

方法描述

rank

功能：根据指定的参数（类型、地区、排行标准等）获取电影排行数据。

参数：pageNo（页码）、pageSize（每页大小）、type（类型）、area（地区）、rank（排行标准）。

返回：包含电影排行数据和分页信息的映射（Map）。

数据模型和存储

相关实体：NewMovie, SearchMovieClass。

NewMovie：表示新电影的信息，包括ID、名称、类型等。

SearchMovieClass：用于搜索和排序结果的表示，包含电影的详细信息。

接口设计

提供RESTful API端点 /index/rank 进行电影排行的查询。

安全和错误处理

跨域处理：使用 @CrossOrigin 注解允许跨域请求。

错误处理：在查询过程中合理处理错误情况，并向前端返回适当的响应。

业务流程

用户通过前端发起电影排行查询请求。

RankController 接收请求并调用 MovieService 的 queryRank 方法。

MovieService 根据提供的参数（类型、地区、排行标准）从数据库查询电影数据。

查询结果返回给 RankController。

RankController 处理数据并返回给前端。

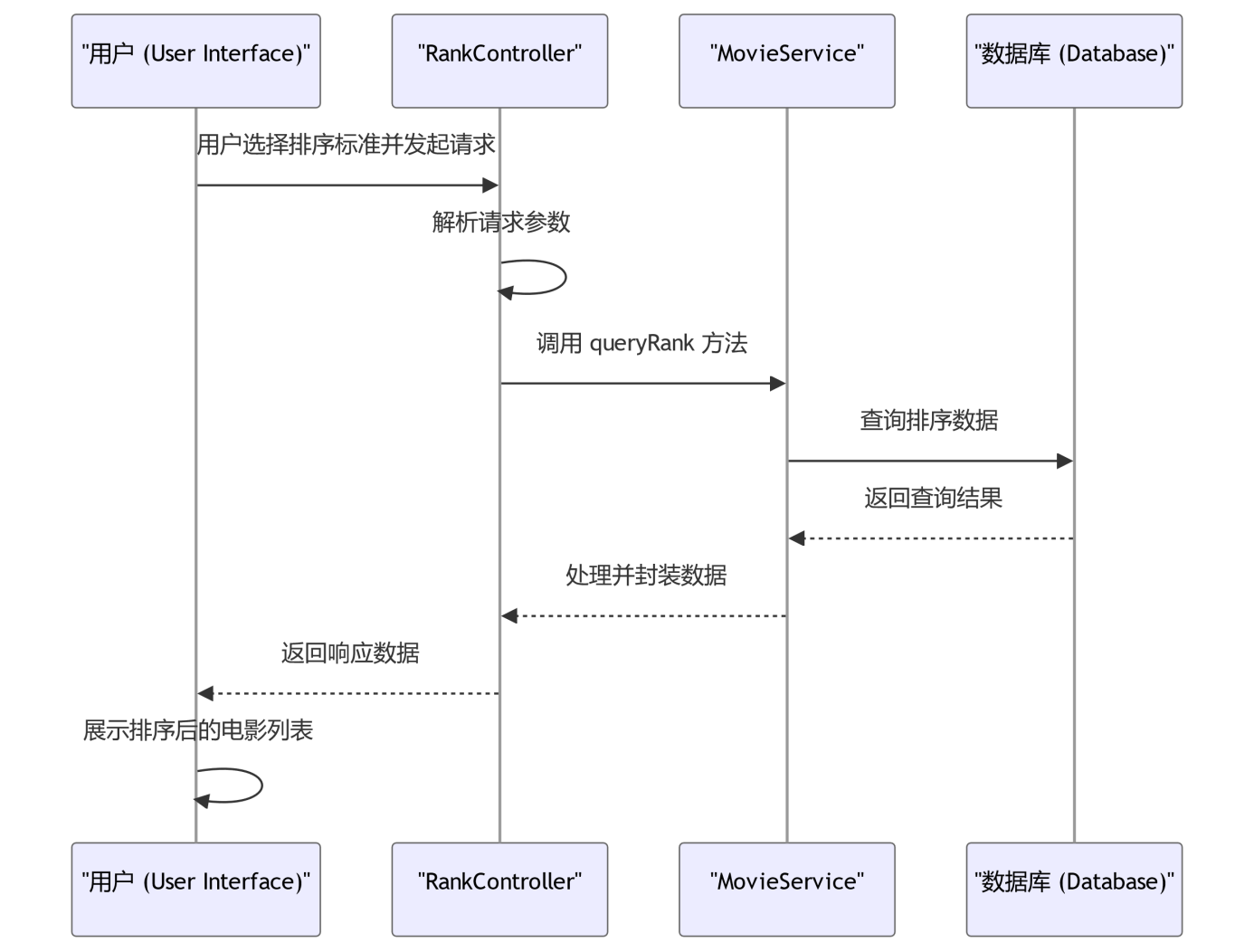


图1.9 排序模块时序图

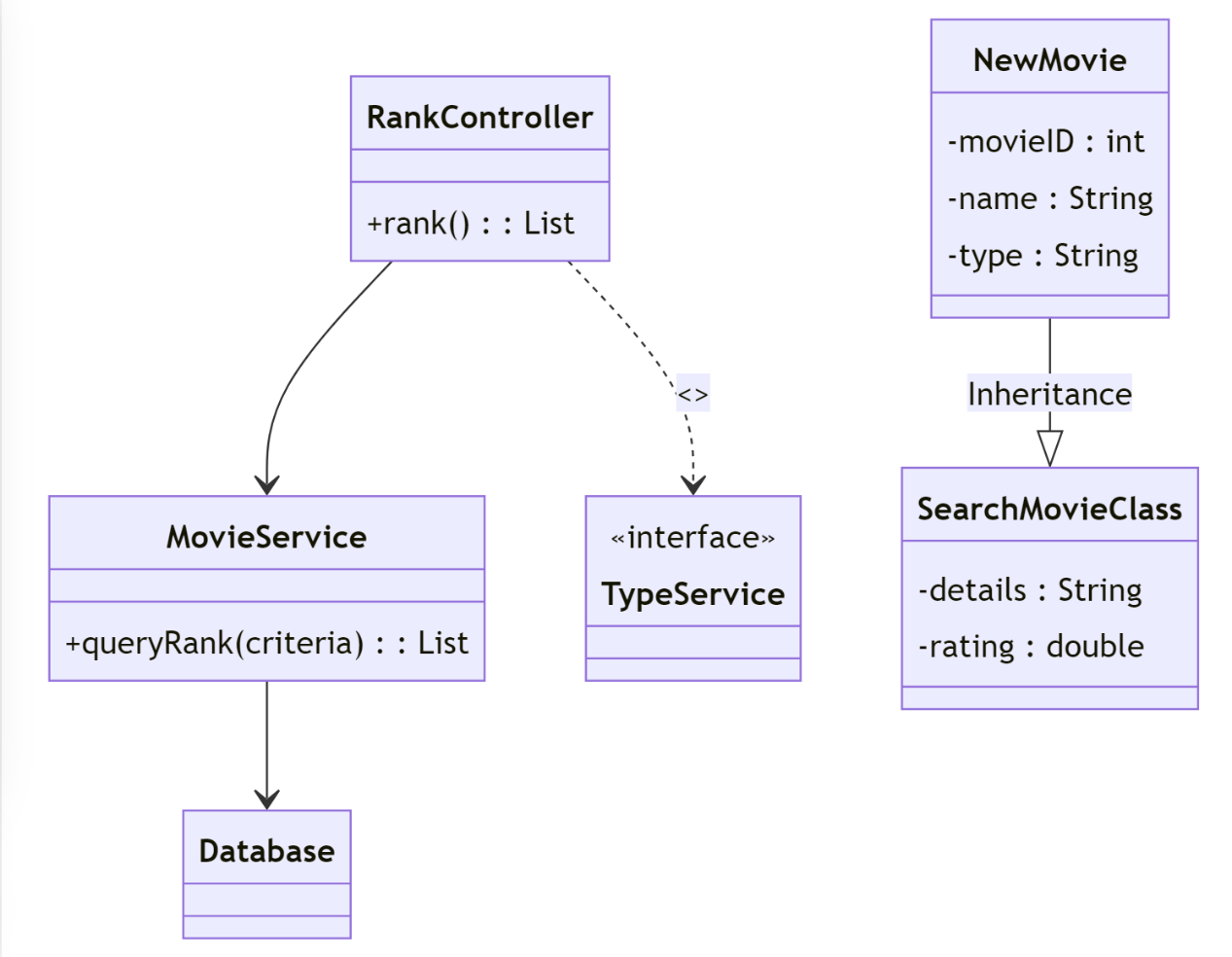


图 2排序模块类图

# 项目实现

### 开发工具和环境

1. 集成开发环境 (IDE)

前端 (Node.js + Vue.js)：

后端 (Spring Boot)：

IntelliJ IDEA：提供强大的 Spring Boot 支持，包括自动配置、快速导航、数据库工具等。

2. Java 开发工具包 (JDK)

JDK 21：最新版本的 Java Development Kit，用于编译和运行 Java 应用。

3. 构建工具

前端：

Node.js：运行时环境，用于执行 JavaScript 代码。

npm：JavaScript 包管理工具，用于管理前端项目的依赖。

后端：

Maven 或 Gradle：用于自动化构建 Spring Boot 应用，管理依赖，运行测试。

4. 数据库和缓存

数据库：

MySQL

Redis：

作为缓存和会话存储，安装 Redis 服务器。

在 Spring Boot 中配置 Redis 依赖和连接设置。

5. 版本控制系统

Git：用于代码版本控制，支持协作开发。

GitHub：提供代码托管，支持代码审查、问题跟踪等。

6. 其他工具

Postman：用于测试和文档化 RESTful API。

开发流程

使用 Git 进行版本控制，协调团队成员的开发工作。

开发过程中，前后端分别部署和测试，通过 API 接口进行交互。

利用 Redis 进行数据缓存，提高应用性能。

定期进行代码审查和合并，确保代码质量和项目进度。

### 模块与组件实现

### 首页（index）

* 1. 轮播

Json样例：

[  
    {  
        **"movieId"**:**1**,  
        **"picture"**:**"001.jpg"**  
    },  
    {  
        **"movieId"**:**2**,  
        **"picture"**:**"002.jpg"**  
    }  
]

返回参数

|  |  |
| --- | --- |
| **名称** | **说明** |
| **lunboList** | 轮播列表，5项数据 |

* 1. 最新影片

返回参数

|  |  |
| --- | --- |
| **名称** | **说明** |
| **latestList** | 最新影片列表，8项数据，按datetime降序排序 |

Json样例：

[  
    {  
        **"movieId"**:**1**,  
        **"movieName"**:**"爱情天梯"**,  
        **"rating"**:**8**,  
        **"picture"**:**"001.jpg"**,  
        **"videoVip"**:**"0"**,  
        **"actor"**:**"格桑 王世奎 胡歌 徐峥"**  
    },  
    {  
        **"movieId"**:**2**,  
        **"movieName"**:**"一起飞"**,  
        **"rating"**:**6.5**,  
        **"picture"**:**"002.jpg"**,  
        **"videoVip"**:**"0"**,  
        **"actor"**:**"格桑 王世奎 胡歌 徐峥"**  
    }  
]

1.3 每周热播

1.4 VIP影片

1.5 所有影片

1.6 人气排行

1.7好评排行

### 搜索模块（search）

#### 类型列表

返回参数：

|  |  |
| --- | --- |
| **名称** | **说明** |
| **typeList** | 所有类型的列表 |

Json样例：

[  
    **"爱情"**,  
    **"剧情"**,  
    **"热血"**  
]

#### 地区列表

返回参数：

|  |  |
| --- | --- |
| **名称** | **说明** |
| **areaList** | 所有地区的列表 |

Json样例：

[  
    **"内地"**,  
    **"港台"**,  
    **"欧美"**  
]

#### 1.3 基因列表

#### 1.4 搜索

//查找类型  
 @RequestMapping(path = "/index/type",method = RequestMethod.*GET*)  
 @ResponseBody  
 public List<String> gettypes(){  
 return typeService.getType();  
 }  
  
//查找地区  
 @RequestMapping(path="/index/area",method = RequestMethod.*GET*)  
 @ResponseBody  
 public List<String> getareas(){  
 return movieService.getAreas();  
 }  
  
 //查找基因  
 @RequestMapping(path = "/index/genre",method = RequestMethod.*GET*)  
 @ResponseBody  
 public List<String> getgenres(){  
 return movieService.getGenre();  
 }

### 三、排行榜(rank)

public Map<String,Object> rank(  
 @RequestParam(name = "pageNo", defaultValue = "1")Integer pageNo,  
 @RequestParam(name = "pageSize", defaultValue = "24")Integer pageSize,  
 @RequestParam(name = "type", defaultValue = "全部")String type,  
 @RequestParam(name = "area", defaultValue = "全部")String area,  
 @RequestParam(name = "p", defaultValue = "总")String rank,  
  
 Model model  
){  
 Map<String,Object> map=new HashMap<>();  
 Map<String, Object> result = movieService.queryRank(type, area, rank, pageNo, pageSize);  
 int count = (int)result.get("count");  
 int pageTotal = (count % pageSize == 0) ? (count / pageSize) : (count / pageSize + 1);  
  
 List<NewMovie> movieList= (List<NewMovie>) result.get("movies");  
 List<SearchMovieClass> searchMovieClassList=new ArrayList<>();  
 for (NewMovie movie : movieList) {  
 SearchMovieClass searchMovieClass=new SearchMovieClass(movie.getMovieId(),movie.getMovieName(),movie.getGenre(),  
 movie.getProductionPlace(),movie.dateToYear(),movie.getRating(),movie.getPicture(),movie.getIntroduction(),  
 movie.getVideoVip(),movie.getActorList()  
 );  
 searchMovieClassList.add(searchMovieClass);  
 }  
 map.put("data",searchMovieClassList);  
 map.put("pageTotal",pageTotal);  
 return map;

### 四、电影详情+播放(detail)

public MovieDetail getdetail(  
 Integer movieId,  
 HttpServletRequest request,  
 HttpSession session  
){  
 if(movieId==0)return null;  
 NewMovie movie=movieService.findMovieById(movieId);  
 boolean limitation=true;  
 MovieDetail movieDetail=new MovieDetail(  
 movie.getMovieId(),movie.getMovieName(),movie.getGenre(),  
 movie.dateToYear(),movie.getRating(),movie.getPicture(),  
 movie.getIntroduction(),movie.getVideoUrl(),movie.getVideoVip(),  
 movie.getActorList(),typeService.findMovieTypeByid(movieId),movie.getProductionPlace(),limitation  
 );  
 Member member=(Member)request.getServletContext().getAttribute("memberLogin");  
 if(member!=null){  
 if(member.getPm()<movieDetail.getVideoVip()){  
 movieDetail.setVideoUrl("");  
 movieDetail.setLimitation(false);  
 }  
 }else if(movieDetail.getVideoVip()>0){  
 movieDetail.setLimitation(false);  
 }  
 return movieDetail;  
}

### 五．注册

@RequestMapping("/index/checkemail")  
 @ResponseBody  
 public String checkEmail(String email){  
 Member member = memberService.findEmail(email);  
 if(member != null){  
 return "该邮件已被注册";  
 }  
 else{  
 return "该邮件可用！";  
 }  
 }  
}

### 六.登录

@RequestMapping("/index/doLogin")  
 @ResponseBody  
 public String dologin(String email,String password,int userType, HttpServletRequest request, HttpSession session){  
 Member member =memberService.login(email,password,userType);  
  
 if(member != null){  
 member.setPassword("");//去敏  
// session.setAttribute("memberLogin",member);  
 request.getServletContext().setAttribute("memberLogin", member);  
 return "{ \"result\":\"登录成功\" }";  
 }else{  
 return "{ \"result\":\"登录失败\" }";  
 }  
 }

### 七．用户中心

ublic String getUserInfo(Model model,HttpServletRequest request, HttpSession session){  
 Map<String, Object> map = new HashMap<>();  
 Member member = (Member) request.getServletContext().getAttribute("memberLogin");  
 if (member == null){  
 return "";  
 }  
 model.addAttribute("userEmail",member.getEmail());  
 model.addAttribute("pm",member.getPm());  
 model.addAttribute("userType", member.getUserType());  
  
 map.put("userEmail", member.getEmail());  
 map.put("pm", member.getPm());  
 map.put("userType",member.getUserType());  
  
 //返回json数据  
 String str = JSON.*toJSONString*(map);  
 return str;  
}

### 八．管理员页面

#### 8.1添加电影

public String addMovie(  
 @RequestParam(name="movieName")String movieName,  
 @RequestParam(name="genre")String genre,  
 @RequestParam(name="productionPlace")String productionPlace,  
 @RequestParam(name="date")Date date,  
 @RequestParam(name="picture")String picture,  
 @RequestParam(name="introduction")String introduction,  
 @RequestParam(name="videoUrl")String videoUrl,  
 @RequestParam(name="videoVip")Integer videoVip,  
 @RequestParam(name="actorList") List<String> actorList,  
 @RequestParam(name="typeList")List<String> typeList  
){  
 Movie movie = manageService.findMovieMaxId();  
 Integer movieId = movie.getMovieId()+1;  
 ManageMovie manageMovie = new ManageMovie();  
 manageMovie.setMovieId(movieId);  
 manageMovie.setMovieName(movieName);  
 manageMovie.setGenre(genre);  
 manageMovie.setProductionPlace(productionPlace);  
 manageMovie.setDate(date);  
 manageMovie.setRating(0);  
 manageMovie.setWeekHits(0);  
 manageMovie.setMonthHits(0);  
 manageMovie.setAllHits(0);  
 manageMovie.setPicture(picture);  
 manageMovie.setIntroduction(introduction);  
 manageMovie.setVideoUrl(videoUrl);  
 manageMovie.setVideoVip(videoVip);  
 manageMovie.setActorList(actorList);  
 manageMovie.setTypeList(typeList);  
 return manageService.addMovie(manageMovie);  
}

#### 8.2删除电影

public String deleteMovie(Integer movieId){  
 boolean flag = manageService.deleteMovieById(movieId);  
 if(flag){  
// manageService.deleteMovieById(movieId);  
 return "{ \"result\":\"删除成功\" }";  
 }  
 return "{ \"result\":\"删除失败\" }";  
 }

#### 8.3修改电影

public String updateMovie(  
 @RequestParam(name="movieId")Integer movieId,  
 @RequestParam(name="movieName")String movieName,  
 @RequestParam(name="genre")String genre,  
 @RequestParam(name="productionPlace")String productionPlace,  
 @RequestParam(name="date")Date date,  
 @RequestParam(name="rating")Float rating,  
 @RequestParam(name="weekHits")Integer weekHits,  
 @RequestParam(name="monthHits")Integer monthHits,  
 @RequestParam(name="allHits")Integer allHits,  
 @RequestParam(name="picture")String picture,  
 @RequestParam(name="introduction")String introduction,  
 @RequestParam(name="videoUrl")String videoUrl,  
 @RequestParam(name="videoVip")Integer videoVip,  
 @RequestParam(name="actorList") List<String> actorList,  
 @RequestParam(name="typeList")List<String> typeList  
 ){  
 ManageMovie manageMovie = new ManageMovie();  
 manageMovie.setMovieId(movieId);  
 manageMovie.setMovieName(movieName);  
 manageMovie.setGenre(genre);  
 manageMovie.setProductionPlace(productionPlace);  
 manageMovie.setDate(date);  
 manageMovie.setRating(rating);  
 manageMovie.setWeekHits(weekHits);  
 manageMovie.setMonthHits(monthHits);  
 manageMovie.setAllHits(allHits);  
 manageMovie.setPicture(picture);  
 manageMovie.setIntroduction(introduction);  
 manageMovie.setVideoUrl(videoUrl);  
 manageMovie.setVideoVip(videoVip);  
 manageMovie.setActorList(actorList);  
 manageMovie.setTypeList(typeList);  
 return manageService.updateMovie(manageMovie);  
}

# 测试计划

### 测试计划描述

根据电影网站的功能，设计详细的测试用例。这包括：

用户注册和登录。

电影搜索、排序、筛选功能。

电影详情查看。

用户评论、评分。

VIP功能测试（如果有）。

轮播展示功能。

排行榜功能。

#### .接口测试

对每个后端提供的RESTful API进行测试。包括：

GET 请求：用于获取数据，如电影列表、电影详情。

POST 请求：用于提交数据，如用户注册、评论。

PUT / PATCH 请求：用于更新数据。

DELETE 请求：用于删除数据。

#### 参数和边界条件测试

测试不同类型的输入参数对接口响应的影响。

包括正常值、边界值、异常值等。

#### 性能测试

测试系统的响应时间和承载能力。

包括API响应时间、并发请求处理能力等。

#### 安全性测试

验证安全机制，如身份验证、数据加密。

测试SQL注入、跨站脚本攻击（XSS）等常见的网络攻击。

#### 缓存和数据库测试

测试Redis缓存的有效性和效率。

验证数据库交互的准确性和性能。

#### 自动化测试脚本

在 Postman 中编写测试脚本以自动化测试过程。

使用 Postman 的测试套件和集合运行器进行批量测试。

#### 测试报告和分析

记录测试结果，包括成功、失败的用例和发现的问题。

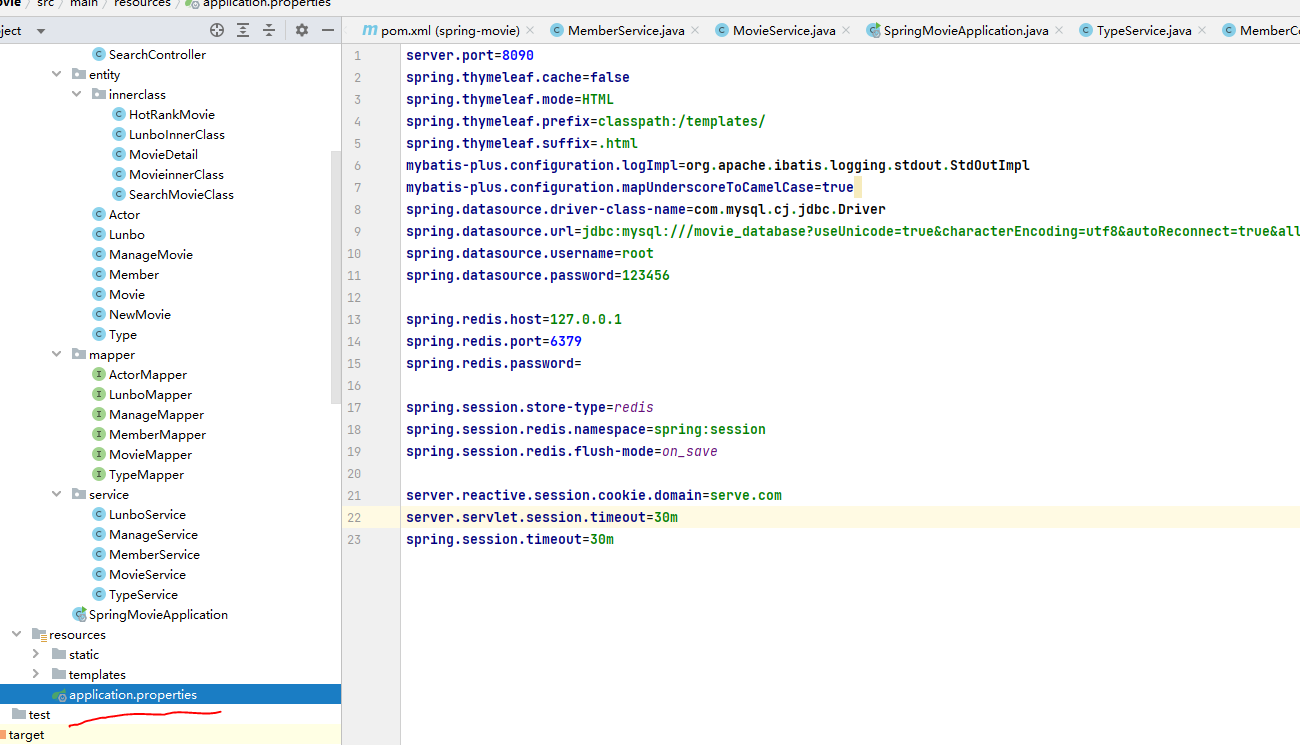


图 3测试结果

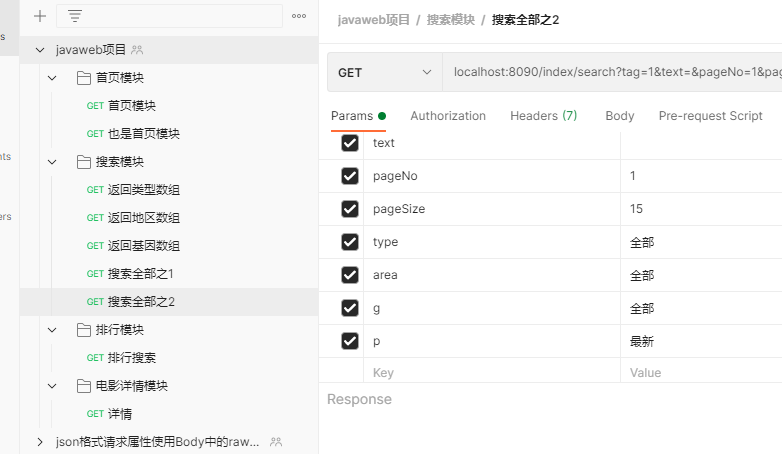


图 4测试结果

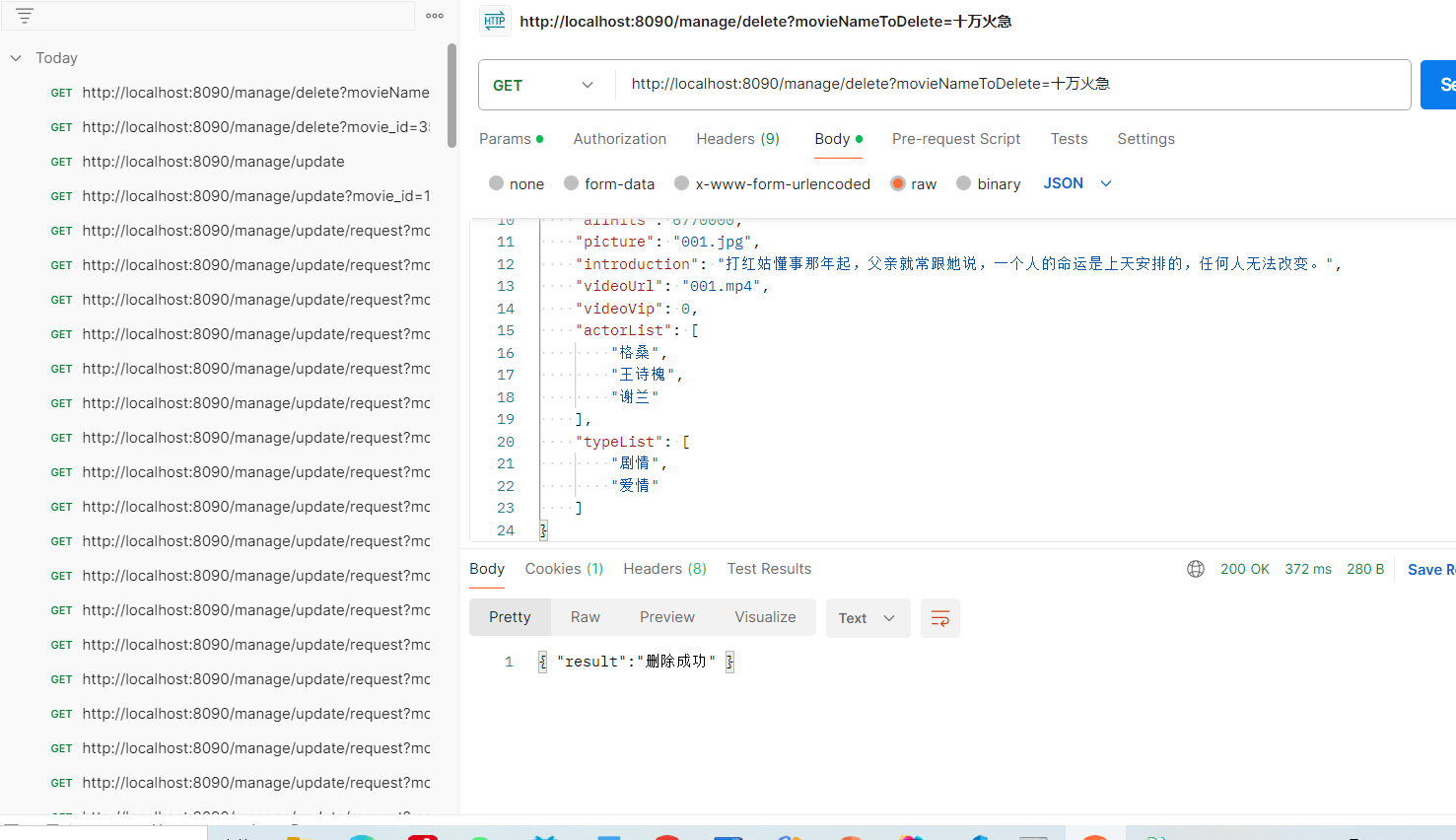


图 5测试结果

# 部署计划

#### 网站部署计划

1. 部署前准备

环境检查：确保生产环境的服务器硬件、操作系统、网络配置符合要求。

依赖安装：安装所有必要的软件依赖，包括 Node.js、Java JDK、数据库（如 MySQL）、Redis 等。

安全设置：配置防火墙规则、SSL证书等安全措施。

2. 代码部署

版本控制：从版本控制系统（如 Git）中拉取最新的稳定代码。

构建和编译：

前端：使用构建工具（如 Webpack）编译前端代码，并生成静态资源文件。

后端：使用 Maven 或 Gradle 构建和打包 Spring Boot 应用。

3. 数据库部署

数据库设置：在生产服务器上安装并配置数据库。

数据迁移：从开发环境迁移数据到生产数据库，可能涉及数据导出和导入操作。

数据验证：确保数据正确迁移，无丢失和错误。

4. 应用部署

后端部署：将 Spring Boot 应用部署到服务器，可能使用容器化工具（如 Docker）。

前端部署：将编译后的前端文件部署到Web服务器或内容分发网络（CDN）。

5. 配置和优化

配置文件：根据生产环境调整配置文件，包括数据库连接、缓存设置等。

性能优化：进行必要的性能调优，包括数据库索引优化、缓存策略设置等。

6. 测试和验证

功能测试：在生产环境进行全面的功能测试，确保所有功能正常运行。

压力测试：进行压力测试，确保系统在高负载下的稳定性。

7. 备份策略

数据备份：定期备份数据库和重要配置文件。

灾难恢复计划：制定应对故障的备份和恢复策略。

8. 监控和维护

日志记录：配置日志记录策略，用于问题追踪和性能监控。

监控系统：使用监控工具（如 Prometheus、Nagios）监控应用和服务器性能。

9. 发布和上线

软启动：初期可选择部分用户或内部用户进行访问，观察应用表现。

全面上线：在确保一切正常后，正式对外发布网站。

# 项目分工说明

由杨富广、罗熹合作完成前端代码，并制作接口文档给后端开发人员进行对接

由蔡钊、陈伟创根据接口文档合作完成后端代码,对数据的增删改查进行优化,尽可能实现数据持久

由叶昕廷对整个项目的前后端部分进行整合,代码修改,功能测试和对接口文档的修改,并承担数据的搜集录入工作  
具体地,  
由杨富广实现首页,搜索,排行,详情页的前端功能.  
由蔡钊实现首页,搜索,排行,详情页的后端功能.  
由罗熹实现登录,注册,管理员,个人中心的前端功能  
由陈伟创实现登录,注册,管理员,个人中心的后端功能.  
由叶昕廷整合前后端项目,对项目进行测试,对部分问题代码进行修改补充,并提供数据测试.

1. [↑](#footnote-ref-0)
2. [↑](#footnote-ref-1)
3. [↑](#footnote-ref-2)
4. [↑](#footnote-ref-3)