上海电力大学

JavaEE 课程报告

题　　目：电影影评购票管理系统

学　　号：20231607

姓　　名：刘栩年

同组姓名：无

院　　系：计算机科学与技术学院

专业年级：计算机科学与技术专业 23 级

2026 年 1 月 11 日

目录

第一章 引言

1.1 项目背景

随着我国文化娱乐产业的蓬勃发展，电影已成为大众日常休闲娱乐的重要组成部分。根据国家电影局数据统计，近年来全国电影票房持续增长，观影人次屡创新高。在这一大背景下，传统的线下排队购票模式已无法满足用户对便捷性、实时性的需求，线上票务平台成为了电影产业链中不可或缺的一环。

然而，现有的部分票务系统在面对热门档期的高并发流量时，常出现系统响应缓慢、座位超卖（即“撞座”）、数据检索延迟等问题。此外，单纯的购票功能已难以留住用户，现代电影平台更需要融合影评社区、实时互动等社交属性，以构建完整的电影生态圈。

基于 JavaEE 技术体系，开发一套功能完善、性能优越的“电影影评购票管理系统”显得尤为重要。该项目旨在利用当前主流的微服务组件与互联网架构思想，解决传统购票系统中的痛点，如利用缓存技术解决高并发读写、利用消息队列处理订单超时与削峰填谷、利用搜索引擎优化海量数据检索等，从而构建一个集影片管理、排片调度、在线选座、影评互动于一体的综合性服务平台。

1.2 项目意义

本项目的设计与实现具有重要的现实应用价值与学术实践意义：

（1）现实应用价值

本系统致力于为影院方和消费者提供双向便利。对于消费者而言，系统提供了可视化的在线选座功能、便捷的支付与退款流程以及丰富的影评互动社区，极大地提升了观影决策效率和购票体验；对于影院管理者而言，系统实现了影片信息、排片计划、订单流水的高效数字化管理，并通过可视化报表实时监控票房数据，有助于优化运营策略，降低人工成本。

（2）学术与技术实践意义

作为 JavaEE 课程的结课项目，本系统不仅仅是业务功能的实现，更是对企业级开发技术的深度实践。

架构设计实践：通过 Spring Boot 搭建后端架构，实践了 RESTful API 设计规范与前后端分离的开发模式。

高并发解决方案：引入 Redis 实现分布式锁以防止座位超卖，利用 RabbitMQ 死信队列处理订单超时自动取消，模拟了真实互联网场景下的高并发挑战。

数据检索与存储优化：整合 Elasticsearch 实现毫秒级的电影全文检索与高亮显示，集成 MinIO 对象存储服务管理海量海报与用户头像，提升了系统的扩展性与性能。

全栈开发能力：通过本项目的开发，深入理解了从数据库设计（MySQL）、后端逻辑实现到前端交互的完整软件开发生命周期，巩固了 JavaEE 技术栈的综合应用能力。

第二章 需求分析

2.1 功能需求

（1）前台用户业务流程

本系统的前台子系统致力于为影迷提供一站式的观影服务体验。首先，在用户准入环节，系统需要提供安全的注册与登录机制，支持基于手机号的实名认证，并允许用户管理个人资料与收货信息。进入首页后，系统需通过高性能的缓存策略展示实时的“热门电影榜单”，为用户提供观影风向标。为了满足用户精准查找影片的需求，系统需构建一个智能检索引擎，支持根据电影名称、导演、演员或简介关键词进行模糊匹配，并具备对中文语义的精确理解能力，确保检索结果的准确性与高亮展示。

核心的购票流程是系统的重中之重。用户通过浏览排片信息进入选座页面时，系统需实时渲染影厅的真实座位分布图，准确区分普通座、情侣座及已损坏的不可售座位。在用户落座锁票的瞬间，系统必须在后端进行严格的库存校验，防止多人同时抢购同一座位的情况发生。订单创建后，系统应提供安全的钱包支付功能，要求验证用户的支付密码以确保资金安全。此外，系统还需包含完善的订单生命周期管理，支持用户查看历史订单状态，并允许在电影开场前规定时间内发起退款或取消订单操作。

除了购票功能，系统还旨在打造一个活跃的影评社区。用户观影后可以发表评分与评论，与其他影迷进行深度交流。系统支持对影评进行回复与点赞，为了保证交互的流畅性，点赞等高频轻量级操作需采用异步处理模式。同时，系统需具备实时的消息通知能力，当用户的订单支付成功、退款到账或有新片上映时，能够即时向客户端推送通知消息。

（2）后台管理业务流程

后台管理子系统主要面向影院运营人员，提供全方位的数据监控与业务配置能力。系统首页需呈现可视化的数据驾驶舱，实时统计总票房、今日流水、活跃用户数等关键指标，辅助管理者进行运营决策。在内容维护方面，管理员需要对

电影库进行增删改查，维护演职人员的详细档案，并上传高清海报与剧照，这些非结构化的图片资源需要统一存储以减轻数据库压力。

业务运营模块涵盖了影院与排片的管理。管理员需能够灵活配置影厅的座位模板，设置行列数及特殊座位标记。在制定排片计划时，系统需具备智能冲突检测机制，防止同一影厅在同一时间段内安排多场电影。此外，后台还需提供订单与用户的监管功能，支持管理员查询所有交易流水，处理异常订单，并对发布违规内容的用户进行账号封禁处理。为了保障系统的安全性与可追溯性，系统需自动记录管理员的所有关键操作行为，包括操作时间、IP 地址及具体动作，形成完整的审计日志。

2.2 技术要求

（1）高并发与高性能架构需求

鉴于互联网票务系统在热门档期可能面临的流量洪峰，本系统在架构设计上对性能提出了极高要求。为了减轻关系型数据库（MySQL）的读取压力，系统引入 Redis 作为高性能缓存中间件，对首页热榜、用户 Token 等高频访问数据进行缓存。针对海量电影数据的检索需求，传统的数据库模糊查询已无法满足性能标准，因此系统集成 Elasticsearch 搜索引擎，并配合 IK 中文分词器，实现了毫秒级的全文检索与复杂的聚合查询能力，确保用户在大数据量下依然拥有流畅的搜索体验。

（2）数据一致性与并发控制需求

在涉及资金与库存的核心业务中，数据的准确性至关重要。系统需解决多人同时选座导致的“超卖”问题，通过引入 Redis 分布式锁 机制，确保同一座位在同一时刻只能被一个用户锁定。同时，针对订单创建后未支付的场景，系统摒弃了传统的定时轮询方案，转而采用 RabbitMQ 消息队列 的“死信队列（DLX）”机制。订单创建时发送带有过期时间的消息，一旦用户超时未支付，消息过期进入死信队列触发自动关单逻辑，既保证了库存的及时释放，又极大降低了服务器的资源消耗。

（3）系统解耦与异步处理需求

为了提升系统的响应速度与扩展性，系统采用异步解耦的设计思想。对于点赞、记录浏览历史等非核心链路的操作，系统通过 RabbitMQ 进行异步流量削峰，避免因瞬间高并发写入导致数据库崩溃。此外，系统集成 Nacos 作为统一配置中心与服务发现组件，实现了配置文件的动态管理，无需重启服务即可调整系统参数，大大提升了运维效率与系统的灵活性。

（4）安全性与存储优化需求

系统在安全性方面需严格遵循企业级标准。用户密码与支付密码均需经过加盐哈希（BCrypt）加密存储，杜绝明文泄露风险。在文件存储方面，考虑到海报、头像等图片资源占用空间大且增长快，系统集成 MinIO 高性能对象存储服务，将非结构化数据与业务数据分离存储，不仅提升了文件的读取速度，也方便了后续的扩容与备份。同时，系统利用 AOP（面向切面编程） 技术实现全链路日志记录，确保所有敏感操作均可追溯，进一步保障了系统的安全稳定运行。

第三章 系统设计

3.1 系统架构

本系统采用了前后端分离的开发模式，整体架构遵循经典的分层设计原则，旨在实现高内聚、低耦合的企业级应用标准。

（1）后端架构逻辑

后端基于 Spring Boot 3.2.5 框架构建，采用标准的 MVC 三层架构。接入层（Controller） 负责接收前端的 RESTful 请求，进行参数校验与权限拦截（基于 AOP 与 Interceptor）；业务逻辑层（Service） 是系统的核心，负责处理复杂的业务规则，如订单状态流转、并发锁座逻辑以及支付事务控制；数据访问层（DAO/Mapper） 基于 MyBatis Plus 框架，实现了与 MySQL 数据库的高效交互。此外，系统引入了 Nacos 作为统一配置中心，实现了配置文件的动态管理与服务发现，为后续扩展为微服务架构奠定了基础。

（2）中间件集成架构

为了应对复杂的业务场景，系统集成了多种中间件。Redis 充当缓存层与分布式锁管理器，横亘在业务层与数据库之间，拦截高频读取请求并保障数据一致性。RabbitMQ 作为消息总线，解耦了核心交易链路与辅助业务（如日志、通知），并通过死信队列机制处理延时任务。Elasticsearch 独立承担全文检索任务，通过数据同步机制与 MySQL 保持最终一致性。MinIO 则作为独立的文件服务器，接管所有的静态资源存储。

3.2 模块划分

系统在逻辑上划分为五大核心业务域，各域之间协同工作以支撑完整业务流程。

（1）用户与交易域

该领域主要包含用户管理与钱包支付两个子模块。用户模块负责处理注册、

登录鉴权（JWT）及个人信息维护；钱包模块则管理用户的虚拟资产，涉及余额充值、支付密码校验及交易流水记录。该域是系统的安全基石，所有涉及资金变动的操作均通过事务严格管控。

（2）电影内容域

这是系统的数据核心，涵盖了电影、演员、导演及影评管理。电影模块维护影片的基础元数据，并负责与 Elasticsearch 进行数据同步；演职人员模块管理明星档案；影评模块则处理用户生成内容（UGC），包括评论的发表、回复及点赞互动，该部分通过 Redis 计数器实现高性能的互动反馈。

（3）影院运营域

该域面向 B 端管理需求，主要包含影院管理、影厅配置及排片调度。影厅配置支持复杂的 JSON 格式座位布局，能够灵活定义行数、列数及损坏座位；排片调度模块则是连接电影与影院的纽带，核心逻辑在于自动检测时间冲突，确保同一影厅的排片时间段互不重叠。

（4）订单中心域

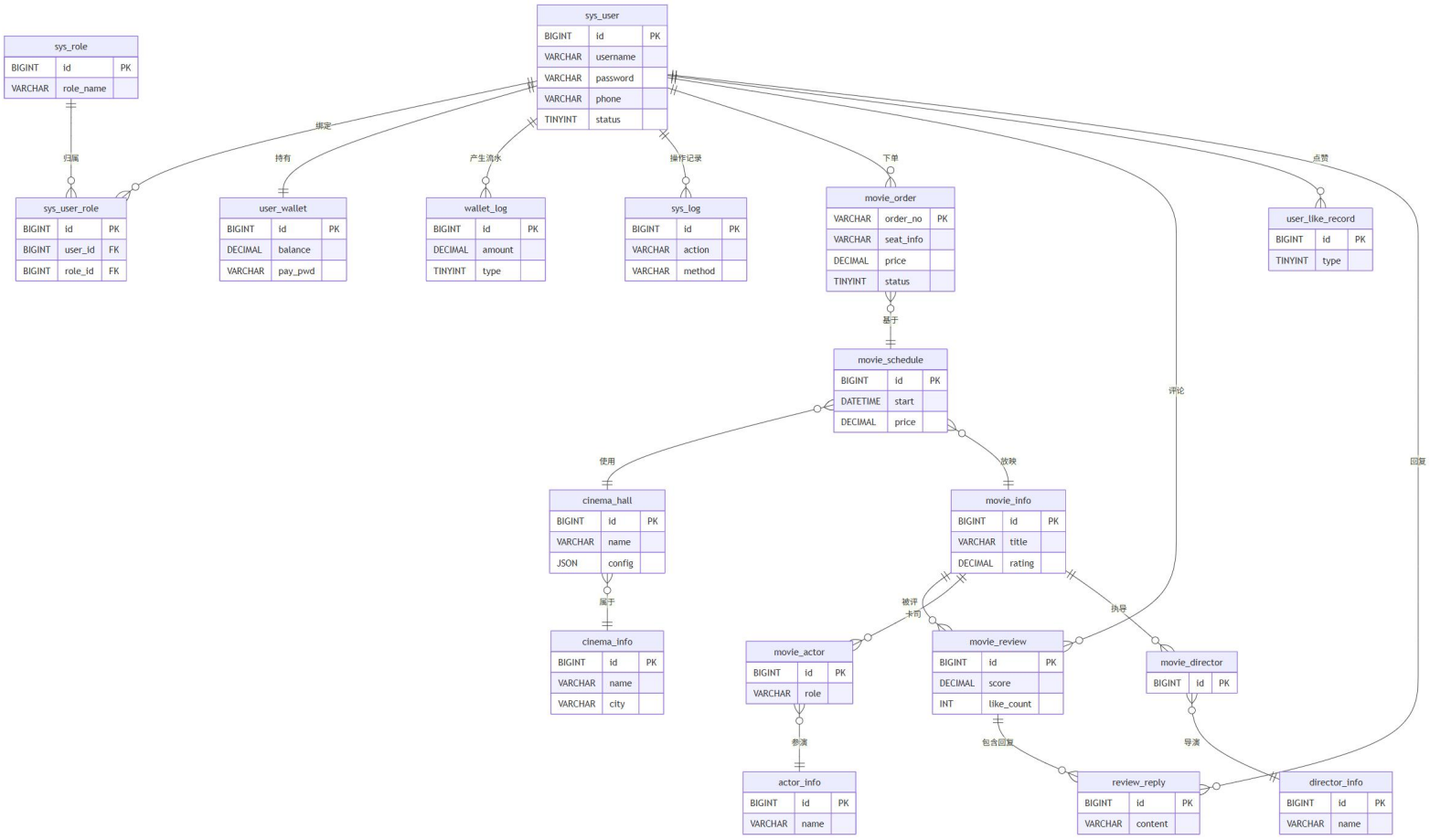
作为业务流转的中枢，订单中心负责从锁座、下单、支付到退款的全生命周期管理。该模块深度集成了 Redis 分布式锁以防止座位超卖，并结合 RabbitMQ 处理订单超时自动取消逻辑，是系统中并发逻辑最复杂的部分。

3.3 数据库设计

本系统使用 MySQL 8.0 作为主要持久化存储，数据库设计遵循第三范式（3NF），确保数据的规范性与完整性。

数据库设计包括以下主要表结构：

ER 图



表设计

1. 用户基础信息表（sys_user）

该表存储系统所有用户的登录凭证与基础档案。其中 password 字段存储经过 BCrypt 算法加密后的密文，status 用于控制用户是否被封号，is_admin 用于区分管理员与普通用户。

字段名	类型	长度/精度	约束	说明
id	BIGINT	20	PK, AI	主键 ID
username	VARCHAR	64	UK	用户名（唯一）

password	VARCHAR	128	NOT NULL	加密后的登录密码
nickname	VARCHAR	64	-	用户昵称
phone	VARCHAR	20	UK	手机号 (唯一,用于接收通知)
email	VARCHAR	64	-	电子邮箱
avatar_url	VARCHAR	255	-	头像地址 (MinIO 存储路径)
is_admin	TINYINT	1	Default 0	是否管理员: 0 否 1 是
status	TINYINT	1	Default 1	账号状态: 1 正常 0 禁用
create_time	DATETIME	-	Default NOW	注册时间
update_time	DATETIME	-	-	更新时间
is_deleted	TINYINT	1	Default 0	逻辑删除: 0 未删 1 已删

2. 角色表 (sys_role)

用于定义系统中的角色权限（如普通用户、超级管理员），配合 RBAC 模型使用。

字段名	类型	长度/精度	约束	说明
id	BIGINT	20	PK, AI	主键 ID
role_name	VARCHAR	32	NOT NULL	角色名称 (如 admin)
role_desc	VARCHAR	64	-	角色中文描述

3. 用户角色关联表 (sys_user_role)

多对多关联表，映射用户与角色的关系。

字段名	类型	长度/精度	约束	说明
id	BIGINT	20	PK, AI	主键 ID
user_id	BIGINT	20	UK_Group	用户 ID
role_id	BIGINT	20	UK_Group	角色 ID

4. 电影信息表 (movie_info)

系统的核心数据表。poster_url 存储海报图片路径，rating 字段存储基于影评计算出的平均分。

字段名	类型	长度/精度	约束	说明
id	BIGINT	20	PK, AI	电影 ID
title	VARCHAR	128	NOT NULL	电影名称 (加索引)
original_title	VARCHAR	128	-	英文原名
release_date	DATE	-	-	上映日期 (加索引)
duration	INT	11	-	电影时长(分钟)
genre	VARCHAR	128	-	类型 (如: 剧情,动作)
language	VARCHAR	64	-	语言
country	VARCHAR	64	-	制片国家/地区

synopsis	TEXT	-	-	剧情简介
poster_url	VARCHAR	255	-	海报图片地址
trailer_url	VARCHAR	255	-	预告片视频地址
rating	DECIMAL	3,1	Default 0.0	综合评分 (如 8.5)
review_count	INT	11	Default 0	评论总数
create_time	DATETIME	-	Default NOW	收录时间
update_time	DATETIME	-	-	更新时间
is_deleted	TINYINT	1	Default 0	逻辑删除标识

5. 演员信息表 (actor_info)

存储演员的个人资料。

字段名	类型	长度/精度	约束	说明
id	BIGINT	20	PK, AI	演员 ID
name	VARCHAR	64	NOT NULL	演员姓名
en_name	VARCHAR	64	-	英文名
gender	TINYINT	1	-	性别: 0 女 1 男 2 未知
birth_date	DATE	-	-	出生日期

nationality	VARCHAR	64	-	国籍
avatar_url	VARCHAR	255	-	演员照片 URL
create_time	DATETIME	-	Default NOW	创建时间
is_deleted	TINYINT	1	Default 0	逻辑删除

6. 导演信息表 (director_info)

存储导演的个人资料，结构与演员表类似。

字段名	类型	长度/精度	约束	说明
id	BIGINT	20	PK, AI	导演 ID
name	VARCHAR	64	NOT NULL	导演姓名
en_name	VARCHAR	64	-	英文名
gender	TINYINT	1	-	性别: 0 女 1 男 2 未知
birth_date	DATE	-	-	出生日期
nationality	VARCHAR	64	-	国籍
avatar_url	VARCHAR	255	-	导演照片 URL
create_time	DATETIME	-	Default NOW	创建时间
is_deleted	TINYINT	1	Default 0	逻辑删除

7. 电影演员关联表 (movie_actor)

多对多关联表，记录一部电影包含哪些演员以及饰演的角色名。

字段名	类型	长度/精度	约束	说明
id	BIGINT	20	PK, AI	主键 ID
movie_id	BIGINT	20	UK_Group	电影 ID
actor_id	BIGINT	20	UK_Group	演员 ID
role_name	VARCHAR	64	-	剧中角色名（如：钢铁侠）

8. 电影导演关联表（movie_director）

多对多关联表，记录一部电影由哪些导演执导。

字段名	类型	长度/精度	约束	说明
id	BIGINT	20	PK, AI	主键 ID
movie_id	BIGINT	20	UK_Group	电影 ID
director_id	BIGINT	20	UK_Group	导演 ID

9. 影院信息表（cinema_info）

存储影院的地理位置信息，uk_city_name 约束保证同一城市下影院名不重复。

字段名	类型	长度/精度	约束	说明
id	BIGINT	20	PK, AI	主键 ID
name	VARCHAR	64	NOT NULL	影院名称
address	VARCHAR	255	NOT NULL	详细地址

city	VARCHAR	64	NOT NULL	所在城市
create_time	DATETIME	-	Default NOW	创建时间

10. 影厅信息表 (cinema_hall)

存储影厅配置。seat_config 字段采用 MySQL 的 JSON 类型，用于灵活存储座位布局（行数、列数）及坏座坐标列表（如 ["3-4", "5-6"]）。

字段名	类型	长度/精度	约束	说明
id	BIGINT	20	PK, AI	主键 ID
cinema_id	BIGINT	20	UK_Group	所属影院 ID
name	VARCHAR	32	NOT NULL	影厅名 (如: 1 号 IMAX 厅)
seat_config	JSON	-	NOT NULL	座位布局 JSON 配置
create_time	DATETIME	-	Default NOW	创建时间

11. 电影排片表 (movie_schedule)

定义了具体的放映场次，是连接电影、影厅与时间的纽带。

字段名	类型	长度/精度	约束	说明
id	BIGINT	20	PK, AI	主键 ID
cinema_id	BIGINT	20	-	所属影院 ID
hall_id	BIGINT	20	-	放映影厅 ID

movie_id	BIGINT	20	-	放映电影 ID
start_time	DATETIME	-	NOT NULL	放映开始时间
end_time	DATETIME	-	NOT NULL	放映结束时间
price	DECIMAL	10,2	NOT NULL	票价
create_time	DATETIME	-	Default NOW	创建时间

12. 购票订单表 (movie_order)

记录用户的交易详情。order_no 为雪花算法生成的分布式 ID，seat_info 记录已选座位号。

字段名	类型	长度/精度	约束	说明
order_no	VARCHAR	32	PK	订单号 (非自增)
user_id	BIGINT	20	-	购买用户 ID
schedule_id	BIGINT	20	-	关联场次 ID
seat_info	VARCHAR	255	NOT NULL	座位号 (如 "5 排 6 座,5 排 7 座")
total_price	DECIMAL	10,2	NOT NULL	订单总金额
status	TINYINT	1	Default 0	0:待支付 1:已支付 2:已取消 3:已退款 4:已观影
pay_time	DATETIME	-	-	支付完成时间

create_time	DATETIME	-	Default NOW	下单时间

13. 用户钱包表 (user_wallet)

管理用户资金。version 字段用于 MyBatis Plus 的乐观锁插件，防止高并发下的余额扣减冲突。

字段名	类型	长度/精度	约束	说明
id	BIGINT	20	PK, AI	主键 ID
user_id	BIGINT	20	UK	用户 ID
balance	DECIMAL	10,2	Default 0.00	可用余额
pay_password	VARCHAR	128	-	支付密码 (加密)
version	INT	11	Default 1	乐观锁版本号
update_time	DATETIME	-	-	余额变动时间

14. 交易流水表 (wallet_log)

记录钱包的所有变动记录，实现资金的可追溯。

字段名	类型	长度/精度	约束	说明
id	BIGINT	20	PK, AI	主键 ID
user_id	BIGINT	20	-	用户 ID
amount	DECIMAL	10,2	NOT NULL	变动金额 (+充值 -消费)

type	TINYINT	4	NOT NULL	类型: 1 充值 2 购票 3 退款
order_no	VARCHAR	32	-	关联的订单号
remark	VARCHAR	128	-	交易备注
create_time	DATETIME	-	Default NOW	发生时间

15. 影评表 (movie_review)

存储用户对电影的评分与文字评价。like_count 字段作为冗余字段，会通过 Redis 定时同步以保证数据最终一致性。

字段名	类型	长度/精度	约束	说明
id	BIGINT	20	PK, AI	主键 ID
user_id	BIGINT	20	-	用户 ID
movie_id	BIGINT	20	-	电影 ID
score	DECIMAL	2,1	NOT NULL	评分 (1.0-10.0)
content	TEXT	-	-	评论内容
like_count	INT	11	Default 0	点赞数 (Redis 同步)
create_time	DATETIME	-	Default NOW	发表时间
is_deleted	TINYINT	1	Default 0	逻辑删除

16. 评论回复表 (review_reply)

实现评论区的楼中楼功能。

字段名	类型	长度/精度	约束	说明
id	BIGINT	20	PK, AI	主键 ID
review_id	BIGINT	20	-	归属的主评论 ID
user_id	BIGINT	20	-	回复者 ID
target_user_id	BIGINT	20	-	被回复者 ID
content	VARCHAR	512	NOT NULL	回复内容
like_count	INT	11	Default 0	点赞数
create_time	DATETIME	-	Default NOW	回复时间
is_deleted	TINYINT	1	Default 0	逻辑删除

17. 用户点赞记录表 (user_like_record)

记录用户点过赞的对象，用于前端回显点赞状态，防止重复点赞。

字段名	类型	长度/精度	约束	说明
id	BIGINT	20	PK, AI	主键 ID
user_id	BIGINT	20	UK_Group	用户 ID
target_id	BIGINT	20	UK_Group	被点赞对象 ID
type	TINYINT	1	UK_Group	类型: 1 影评, 2 回复
create_time	DATETIME	-	Default NOW	点赞时间

18. 后台操作日志表 (sys_log)

利用 AOP 切面技术，记录管理员的所有操作细节。params 字段存储请求参数的 JSON 字符串。

字段名	类型	长度/精度	约束	说明
id	BIGINT	20	PK, AI	主键 ID
username	VARCHAR	64	NOT NULL	操作人用户名
module	VARCHAR	64	-	操作模块 (如: 电影管理)
action	VARCHAR	64	-	动作 (如: 删除电影)
method	VARCHAR	255	-	请求方法全路径
params	TEXT	-	-	请求参数 (JSON)
ip	VARCHAR	64	-	操作人 IP 地址
time	BIGINT	20	-	执行耗时 (毫秒)
create_time	DATETIME	-	Default NOW	记录时间

3.4 接口设计

本系统后端接口严格遵循 RESTful 设计规范，所有接口均通过 HTTP 标准动词（GET/POST/PUT/DELETE）区分操作语义。数据交互格式统一采用 JSON，响

应结果由 Result<T> 统一封装，包含状态码 code（200 成功/500 异常）、提示信息 message 及数据载体 data。

以下是系统所有功能模块的详细接口定义：

1. 用户与权限模块 (User & Wallet)
涉及用户注册登录、个人信息管理、管理员封号操作以及钱包资金管理。

接口名称	请求方法	接口路径	请求参数说明	功能描述
用户注册	POST	/user/register	Body: UserRegisterDTO	校验手机号唯一性并注册
用户登录	POST	/user/login	Body: UserLoginDTO	校验密码并返回 Token
修改个人信息	PUT	/user/update	Body: User	修改昵称、头像等(仅限自己)
获取当前用户	GET	/user/me	Header: Authorization	获取当前登录人的脱敏信息
修改用户状态	PUT	/user/status/{userId}/{status}	Path: userId, status	[管理员] 封号(0)或解封(1)
查询钱包详情	GET	/userWallet/info	-	获取余额及是否设置支付密码
设置支付密码	POST	/userWallet/se	Body:	首次设置钱包

		t-password	WalletPasswordDTO	支付密码
修改支付密码	POST	/userWallet/change-password	Body: ChangePayPwdDTO	验证旧密码后修改新密码
余额充值	POST	/userWallet/recharge	Body: RechargeDTO	增加钱包余额，记录流水
查询资金流水	GET	/wallet-log/my	Query: page, size	分页查询我的充值/消费记录

2. 电影内容模块 (Movie Content)

涉及电影元数据、Elasticsearch 搜索、热门榜单及演员导演管理。

接口名称	请求方法	接口路径	请求参数说明	功能描述
发布电影	POST	/movie/add	Body: MovieDTO	[管理员] 新增电影及演职员关联
全文检索	POST	/movie/search	Body: SearchDTO	基于 ES 的高亮分词搜索
修改电影	PUT	/movie/update	Body: MovieDTO	[管理员] 更新电影信息及关联
删除电影	DELETE	/movie/delete/{id}	Path: id	[管理员] 同步删除 MySQL

				和 ES
电影详情(简)	GET	/movie/{id}	Path: id	获取电影基础信息(用于回显)
电影详情(全)	GET	/movie/detail/{id}	Path: id	获取电影及完整的演职员列表
热门榜单	GET	/movie/top10	-	获取评分最高的 10 部电影(Redis)
电影列表	GET	/movie/list	Query: page, size, name	[管理员] 后台分页查询
添加演员	POST	/actorInfo/add	Body: ActorDTO	[管理员] 新增演员
修改演员	PUT	/actorInfo/update	Body: ActorInfo	[管理员] 修改演员信息
删除演员	DELETE	/actorInfo/delete/{id}	Path: id	[管理员] 删除演员
演员列表	GET	/actorInfo/list	Query: page, size, name	分页查询或搜索演员
添加导演	POST	/directorInfo/add	Body: DirectorDTO	[管理员] 新增导演

修改导演	PUT	/directorInfo/update	Body: DirectorInfo	[管理员] 修改导演信息
删除导演	DELETE	/directorInfo/delete/{id}	Path: id	[管理员] 删除导演
导演列表	GET	/directorInfo/list	Query: page, size, name	分页查询或搜索导演

3. 影院与排片模块 (Cinema & Schedule)

涉及影院管理、影厅座位配置及排片冲突检测。

接口名称	请求方法	接口路径	请求参数说明	功能描述
添加影院	POST	/cinema/add	Body: CinemaDTO	[管理员] 新增影院
修改影院	PUT	/cinema/update	Body: CinemaInfo	[管理员] 修改影院信息
删除影院	DELETE	/cinema/delete/{id}	Path: id	[管理员] 删除影院(需无影厅)
影院列表	GET	/cinema/list	Query: page, size, city	分页查询影院
添加影厅	POST	/hall/add	Body: HallDTO	[管理员] 配置行、列及名称
修改影厅	PUT	/hall/update	Body: CinemaHall	[管理员] 修改座位布局(JSON)

删除影厅	DELETE	/hall/delete/{id}	Path: id	[管理员] 删除影厅
影厅列表	GET	/hall/list/{cinemaId}	Path: cinemaId	获取某影院下的所有影厅
添加排片	POST	/schedule/add	Body: ScheduleDTO	[管理员] 新增排片(含冲突检测)
删除排片	DELETE	/schedule/delete/{id}	Path: id	[管理员] 删除排片
排片查询	GET	/schedule/list/{movieId}	Path: movieId	查询某电影的未开场场次

4. 订单交易模块 (Order & Transaction)

核心交易链路，包含锁座、下单、支付、退款及订单管理。

接口名称	请求方法	接口路径	请求参数说明	功能描述
获取座位图	GET	/order/seats/{scheduleId}	Path: scheduleId	获取影厅配置及已售/锁定座位
创建订单	POST	/order/create	Body: OrderDTO	Redis 原子锁座并创建订单
支付订单	POST	/order/pay	Body: PayDTO	校验支付密码并扣款
申请退款	POST	/order/refund/	Path: orderNo	退还金额至余

		{orderNo}		额，释放座位
取消订单	POST	/order/cancel/{orderNo}	Path: orderNo	手动取消待支付订单
我的订单	GET	/order/my	Query: page, size, status	分页查询当前用户订单
所有订单	GET	/order/admin/list	Query: page, size, orderNo	[管理员] 查询全平台订单

5. 影评社交模块 (Review & Reply)

涉及评论发表、回复楼中楼及点赞互动。

接口名称	请求方法	接口路径	请求参数说明	功能描述
发表影评	POST	/review/add	Body: ReviewDTO	评分并评论(每人限一次)
影评列表	GET	/review/list/{movieId}	Path: movieId	分页获取影评及对应的回复
点赞影评	POST	/review/like/{reviewId}	Path: reviewId	异步点赞/取消点赞
删除影评	DELETE	/review/delete/{id}	Path: id	[管理员/用户] 删除影评
后台影评	GET	/review/admin/list	Query: keyword	[管理员] 搜索所有影评
发表回复	POST	/reply/add	Body:	回复某条影评或

			ReplyDTO	某个人
点赞回复	POST	/reply/like/{replyId}	Path: replyId	异步点赞回复
回复列表	GET	/reply/list/{reviewId}	Path: reviewId	分页获取某评论下的回复
删除回复	DELETE	/reply/delete/{id}	Path: id	[管理员] 删除回复

6. 系统辅助模块 (System & Tools)
包含文件上传、日志审计及数据报表。

接口名称	请求方法	接口路径	请求参数说明	功能描述
文件上传	POST	/file/upload	Form: file	上传图片至 MinIO，返回 URL
操作日志	GET	/log/list	Query: keyword	[管理员] 查询后台操作记录
运营报表	GET	/report/index	-	[管理员] 获取首页统计数据

3.5 技术选型

本系统采用当前主流的微服务技术栈进行设计与开发，虽然主体架构为单体应用，但引入了大量分布式中间件以解决高并发与大数据量的业务痛点。

以下是系统核心技术栈的详细选型说明及应用场景：

1. 后端核心框架

技术组件	版本	核心作用与应用场景
Java	17 (LTS)	开发语言。选用长期支持版本 Java 17，利用其新特性（如 Record、Switch 表达式）提升代码可读性与运行效率。
Spring Boot	3.2.5	基础框架。利用其自动装配（Auto-Configuration）特性快速搭建项目，集成 Web、AOP、Security 等模块，实现开箱即用。
MyBatis Plus	3.5.5	ORM 框架。替代传统 MyBatis，提供了通用的 CRUD 接口，极大减少了重复 SQL 的编写；利用其 代码生成器 快速生成 Entity/Mapper/Service 层代码；使用其 乐观锁插件 (@Version) 解决钱包扣款时的并发冲突问题。
Hutool	5.8.26	工具类库。使用了 IdUtil 生成雪花算法订单号，BCrypt 进行密码加盐加密，以及 BeanUtil 进行 DTO 与 Entity 的高效拷贝。

2. 数据存储与缓存

技术组件	版本	核心作用与应用场景
MySQL	8.0	关系型数据库。存储用户、订单、电影等结构化数据。利用 MySQL 8.0 的 JSON 数据类型 存储复杂的影厅座位配置（seat_config），避免了建立繁琐的座位关联表。
Redis	7.x	分布式缓存与锁。1. 缓存：缓存首页“热门电影榜单”及用户登录 Token，降低 MySQL 读取压力。2. 分布式锁：在用户下单选座时，利用 Redis 原子性操作（SetNX）实

现锁座功能，防止高并发下的“一票多卖”。3. 计数器：维护影评的点赞数，避免频繁更新数据库。

3. 中间件与分布式组件

技术组件	版本	核心作用与应用场景
RabbitMQ	Latest	消息队列。 1. 死信队列 (DLX)：实现订单超时自动关闭功能。订单创建时发送 TTL 消息，过期后转发至死信队列进行关单检查。 2. 异步解耦：将点赞、浏览记录等非核心业务通过 MQ 异步处理，提升接口响应速度。
Elasticsearch	7.17.10	全文搜索引擎。解决海量电影数据的模糊查询性能问题。配合 IK 中文分词器，实现了根据电影名、简介、演职员的混合检索，并支持关键词高亮显示。
MinIO	8.5.9	对象存储。用于存储电影海报、演员照片、用户头像等非结构化数据。相比将图片存入数据库或本地磁盘，MinIO 提供了更快的读取速度和更好的扩展性。
Nacos	2.2.0	配置中心与注册中心。虽然目前为单体部署，但引入 Nacos 管理 application.yml 配置文件，实现了配置的动态刷新与统一管理，为后续升级微服务架构做好准备。

4. 实时通信与安全

技术组件	版本	核心作用与应用场景
WebSocket	Spring Boot	全双工通信。实现了服务端向客户端的主动消息推送。例如：当用户支付成功或退款到账时，系统右下角会

	Starter	自动弹出通知消息；新片上映时向所有在线用户广播提醒。
AOP (AspectJ)	Spring Boot Starter	面向切面编程。用于实现系统的 操作日志记录 (@SysLog)。通过拦截 Controller 层的方法调用，自动记录操作人、IP 地址、执行方法及耗时，无需侵入业务代码。

5. 前端交互技术

技术组件	版本	核心作用与应用场景
Vue 3	3.5.25	前端核心框架。采用最新的 Vue 3.5 版本，全面使用 Composition API 进行开发，相比 Vue 2 具有更强的逻辑复用能力和渲染性能，构建单页面应用 。
Vite	7.2.4	新一代构建工具。选用了极新的 Vite 7 版本，利用其基于 ES Modules 的冷启动特性和毫秒级的热更新 ，极大提升了开发调试效率。
Element Plus	2.13.0	UI 组件库。基于 Vue 3 的企业级 UI 库，用于快速构建系统的前台选座界面、后台管理表格、表单弹窗等，统一了系统的视觉风格。
Pinia	3.0.4	状态管理库。替代了传统的 Vuex，作为 Vue 3 官方推荐的状态管理方案。用于在组件间共享 用户登录状态 (Token)、当前选座信息 等全局数据，且体积更小、API 更简洁。

第四章 关键功能实现

4.1 实现高并发选座与防超卖功能

（1）技术实现

在线选座是票务系统的核心业务。在热门电影上映时，可能会出现成百上千人同时点击同一个座位的并发情况。如果仅依赖数据库事务，容易导致“一票多卖”的严重事故。

本功能采用 Redis 分布式锁 机制与 MySQL 事务 相结合的方案。

原子锁座：在创建订单前，利用 Redis 的原子性操作，将用户选择的座位号作为 Key 尝试加锁。

双重校验：Redis 锁成功后，再次查询数据库确认座位状态，确保万无一失。

事务保障：订单落库与锁定逻辑在同一个事务中，一旦发生异常（如座位被抢、坏座校验失败），自动回滚并释放 Redis 锁。

（2）前端界面

前端采用 Vue 3 + Canvas/Div 渲染座位图，用户点击座位后触发 API 请求。

核心代码 (Customer/ScheduleAndSeats.vue)：

```
// 选座下单核心逻辑
const submitOrder = async () => {
  if (selectedSeats.value.length === 0) {
    ElMessage.warning('请至少选择一个座位');
    return;
  }

  // 1. 组装参数：排片 ID 和座位列表
```

```
  const orderForm = {
    scheduleId: route.params.scheduleId,
    seats: selectedSeats.value // 例如 ["5-6", "5-7"]
  };
  try {
    loading.value = true;
    // 2. 调用后端创建订单接口
    const res = await createOrder(orderForm);
    if (res.code === 200) {
      ElMessage.success('锁定座位成功，请在 15 分钟内支付');
      // 3. 跳转到支付页面，携带订单号
      router.push({ name: 'OrderPay', params: { orderNo: res.data } });
    } else {
      ElMessage.error(res.message || '选座失败，座位可能已被抢');
      // 刷新座位图
      refreshSeatMap();
    }
  } catch (error) {
    console.error(error);
  } finally {
    loading.value = false;
  }
};
```

代码说明： 前端负责收集用户选择的座位坐标，并通过 createOrder 接口发送

给后端。为了用户体验，请求过程中会有 loading 状态，防止重复点击。

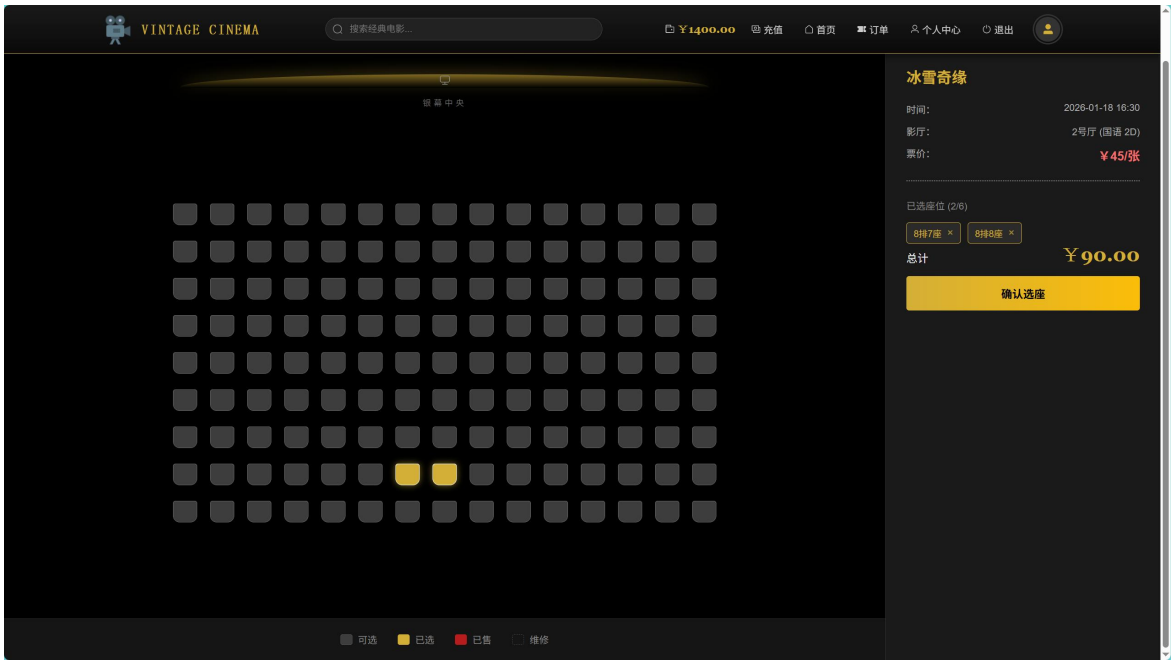


图 4-1 选座界面

（3）后端接口

后端通过 OrderController 接收请求，并委托给 OrderServiceImpl 处理复杂的锁座逻辑。

核心代码 (OrderServiceImpl.java)：

```
@Override
@Transactional(rollbackFor = Exception.class) // 开启事务，异常自动回滚
public String createOrder(OrderDTO dto, Long userId) {
    List<String> finalSeats = dto.getSeats();
    // 1. 尝试 Redis 原子锁座 (关键步骤)
    // 这里的 redisLockService 底层使用了 Lua 脚本或 SETNX 命令
    // 确保多个线程同时抢同一个座位时，只有一个能成功
    boolean lockSuccess = redisLockService.tryLockSeats(dto.getScheduleId(), finalSeats, userId);
```

```
if (!lockSuccess) {
    throw new RuntimeException("手慢了，部分座位已被锁定！");
}
try {
    // 2. 业务校验：排片是否存在、是否已开场、是否是坏座
    Schedule schedule = scheduleService.getById(dto.getScheduleId());
    if (schedule.getStartTime().isBefore(LocalDateTime.now())) {
        throw new RuntimeException("电影已开场，停止售票！");
    }
    // 3. 数据库兜底检查：确保这些座位没有在数据库里显示为“已支付”
    QueryWrapper<Order> checkQuery = new QueryWrapper<>();
    checkQuery.eq("schedule_id", dto.getScheduleId())
              .eq("status", 1) // 1-已支付
              .in("seat_info", finalSeats);
    if (this.count(checkQuery) > 0) {
        throw new RuntimeException("部分座位已售出");
    }
    // 4. 创建订单对象并保存
    Order order = new Order();
    String orderNo = IdUtil.getSnowflakeNextIdStr(); // 生成雪花算法 ID
    order.setOrderNo(orderNo);
    order.setUserId(userId);
    order.setScheduleId(dto.getScheduleId());
    order.setSeatInfo(String.join(",", finalSeats)); // 存入 "5-6,5-7"
```

```
            order.setTotalPrice(schedule.getPrice().multiply(new
BigDecimal(finalSeats.size())));
            order.setStatus(0); // 0-待支付
            this.save(order); //  写入数据库
            // 5. 发送延迟消息到 RabbitMQ (处理超时未支付)
            rabbitTemplate.convertAndSend(RabbitMQConfig.ORDER_TTL_QUEUE,
orderNo);
            return orderNo;
        } catch (RuntimeException e) {
            //  【关键】如果中间任何一步失败（如数据库报错、逻辑校验不通过）
            //  必须主动释放刚才锁住的 Redis 锁，否则座位会死锁
            redisLockService.releaseSeatLocks(dto.getScheduleId(), finalSeats);
            throw e;
        }
}
```

代码说明：

这是系统最核心的业务逻辑。代码首先调用 redisLockService 进行预占座，成功后再进行数据库层面的逻辑校验和写入。特别采用了 try-catch 结构，确保如果业务逻辑失败，能够执行 releaseSeatLocks 释放资源，保证了系统的健壮性。

（4）调用数据库

该功能最终会向 movie_order 表插入一条记录。

MyBatis Plus 调用代码：

```
// 使用 MyBatis Plus 提供的 BaseMapper 接口方法
```

对应 SQL: INSERT INTO movie_order (order_no, user_id, seat_info, ...) VALUES (?, ?, ?, ...)

this.save(order);

数据库表变化说明：

执行成功后，movie_order 表中新增一条记录，状态 status 为 0（待支付），seat_info 字段记录了锁定的座位号。此时虽然用户还没付钱，但座位已经被“占”住了。

（5）运行结果

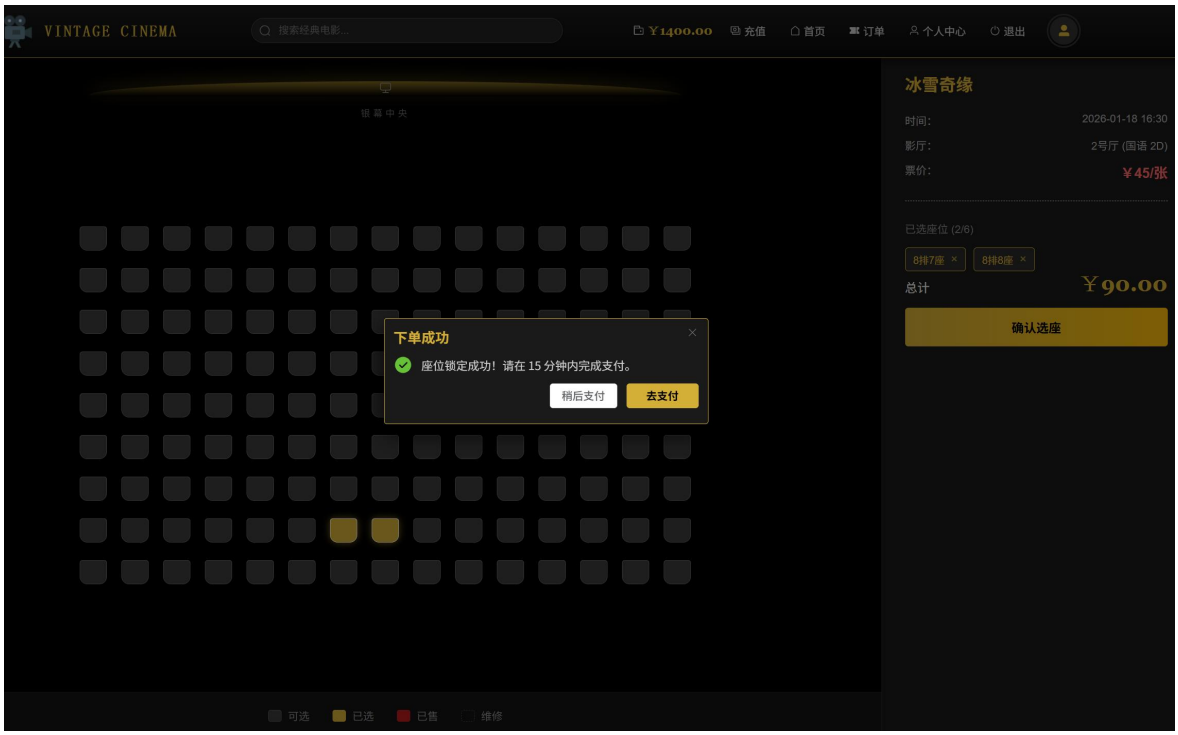


图 4-1 锁座成功截图

4.2 实现订单超时自动取消功能

（1）技术实现

为了释放长时间占用但不支付的座位资源，系统必须具备订单超时自动取消功能。传统的 Timer 轮询数据库方式效率低下且不仅实时。本系统利用 RabbitMQ 的 TTL (Time-To-Live) + DLX (Dead Letter Exchange) 机制实现。

订单创建时，发送一条消息到 延迟队列，设置过期时间为 15 分钟。消息

在队列中等待 15 分钟后，因过期被转发到 死信交换机。死信队列 的消费者收到消息，检查该订单是否仍未支付。若是，则执行关单和释放座位操作。

（2）前端界面

前端在“我的订单”页，展示订单状态的变化。

核心代码 (Customer/MyOrders.vue)：

```
// 获取订单列表
const fetchOrders = async () => {
  const res = await getMyOrders({ page: 1, size: 10 });
  orderList.value = res.data.records.map(item => {
    // 简单的状态映射展示
    if (item.status === 0) item.statusText = '待支付';
    else if (item.status === 1) item.statusText = '已支付';
    else if (item.status === 2) item.statusText = '已取消 (超时)';
    return item;
  });
};
```

代码说明： 前端通过轮询或刷新列表，展示从后端获取的最新状态。当后台 RabbitMQ 处理完超时逻辑后，用户刷新页面会看到状态变为“已取消”。

（3）后端接口

该功能由后端的消息监听器自动触发，无需前端主动调用接口。

配置类代码 (RabbitMQConfig.java)：

```
// 1. 定义延迟队列 (TTL Queue)
@Bean
public Queue orderTtlQueue() {
```

```
        return QueueBuilder.durable(ORDER_TTL_QUEUE)
                .ttl(900000) // 设置 TTL 为 15 分钟 (15 * 60 * 1000ms)
                .deadLetterExchange(ORDER_DLX_EXCHANGE) // 绑定死信交换机
                .deadLetterRoutingKey(ORDER_DLX_ROUTING_KEY)
                .build();
    }
    // 2. 定义死信队列 (实际消费队列)
    @Bean
    public Queue orderReleaseQueue() {
        return new Queue(ORDER_RELEASE_QUEUE, true);
    }
```

监听器代码 (OrderTimeoutListener.java)：

code

Java

```
@Component
public class OrderTimeoutListener {
    @Autowired
    private IOrderService orderService;
    @Autowired
    private StringRedisTemplate redisTemplate;
    // 监听死信队列，处理过期的订单消息
    @RabbitListener(queues = RabbitMQConfig.ORDER_RELEASE_QUEUE)
    public void listenOrderTimeout(String orderNo) {
```

```
            // 1. 查询订单最新状态
            Order order = orderService.getOne(new QueryWrapper<Order>().eq("order_no", orderNo));
            // 2. 如果订单存在且状态仍为 0 (待支付)
            if (order != null && order.getStatus() == 0) {
                // 3. 修改数据库状态为 2 (已取消)
                order.setStatus(2);
                orderService.updateById(order);
                // 4. 释放 Redis 中的座位锁
                // 拼接锁的 Key，例如 lock:seat:105:5-6
                String lockKeyPrefix = "lock:seat:" + order.getScheduleId() + ":";
                List<String> seats = List.of(order.getSeatInfo().split(","));
                List<String> lockKeys = seats.stream()
                        .map(seat -> lockKeyPrefix + seat)
                        .collect(Collectors.toList());
                redisTemplate.delete(lockKeys); // 删除 Redis Key，释放座位
                System.out.println("订单 " + orderNo + " 超时未支付，已自动取消并释放座位。");
            }
        }
    }
```

代码说明： RabbitMQConfig 配置了消息的流转规则，OrderTimeoutListener 则是实际的执行者。它确保了只有“真正超时且未支付”的订单才会被取消，并同步清理 Redis 锁，让座位可以被其他用户购买。

（4）调用数据库

该功能涉及查询订单状态和更新订单状态。

MyBatis Plus 调用代码：

// 1. 查询

Order order = orderService.getOne(new QueryWrapper<Order>().eq("order_no", orderNo));

// 2. 更新 (对应 SQL: UPDATE movie_order SET status=2 WHERE id=?)

orderService.updateById(order);

数据库表变化说明：

当监听器执行完毕后，movie_order 表中对应记录的 status 字段会从 0 变为 2。

（5）运行结果

4.3 实现电影全文检索功能

（1）技术实现

为了让用户能通过模糊的关键词（如“动作”、“成龙”）快速找到电影，系统集成了 Elasticsearch。

分词索引：使用 IK Analysis 分词器，将电影标题、简介、演员名进行中文分词并建立倒排索引。

数据同步：在管理员新增或修改电影时，采用“双写”策略，MySQL 更新成功后同步更新 ES 索引。

高亮搜索：查询时对匹配的关键词进行 HTML 标签包裹，在前端显示高亮效果。

（2）前端界面

前端顶部搜索框，输入关键词后展示搜索结果列表。

核心代码 (HeaderSearch.vue)：

```
// 搜索方法
const handleSearch = async () => {
  const params = {
    keyword: searchKeyword.value,
    page: 1,
    size: 20
  };
  // 调用搜索接口
  const res = await searchMovies(params);
  if (res.code === 200) {
    // 渲染搜索结果列表
    searchResult.value = res.data.content;
  }
};
```

（3）后端接口

后端 MovieController 接收请求，调用 InfoServiceImpl 操作 ES。

核心代码 (InfoServiceImpl.java)：

```
@Override
public Page<MovieDoc> search(SearchDTO dto) {
    PageRequest pageRequest = PageRequest.of(dto.getPage() - 1, dto.getSize());
    // 1. 构建 ES 查询条件 (NativeQuery)
    Query query = NativeQuery.builder()
            .withQuery(q -> q
```

```
                .multiMatch(mq -> mq
                        .query(dto.getKeyword()) // 用户输入的关键词
                        // 指定搜索字段：标题、原名、演员、导演、简介
                        .fields("title", "originalTitle", "actors", "directors",
"synopsis")
                )
            )
            .withPageable(pageRequest)
            .build();
    // 2. 执行搜索
    SearchHits<MovieDoc> searchHits = elasticsearchOperations.search(query,
MovieDoc.class);
    // 3. 解析结果并返回分页对象
    SearchPage<MovieDoc> searchPage =
SearchHitSupport.searchPageFor(searchHits, pageRequest);
    return (Page<MovieDoc>) SearchHitSupport.unwrapSearchHits(searchPage);
}
```

代码说明：这里使用了 Spring Data Elasticsearch 的最新 NativeQuery 写法，multiMatch 表示在多个字段中同时搜索，只要任何一个字段匹配（比如演员名字匹配），该电影就会被搜出来。

（4）调用数据库/索引

此功能主要读取 Elasticsearch 索引 movie_index，不直接读取 MySQL。

ES 交互逻辑：

```
// 对应 REST API: POST /movie_index/_search
elasticsearchOperations.search(query, MovieDoc.class);
```

说明： 实际运行中，Spring 会将 Java 代码转换为 Elasticsearch DSL 语句发送给 ES 服务器。

（5）运行结果

图 4-3 es 搜索

第五章 系统总结

5.1 系统已实现的功能

本项目已成功构建了一个基于 JavaEE 架构的“电影影评购票管理系统”。系统前后端分离，部署架构采用了 Docker 容器化方案，各功能模块运行稳定，各项技术指标均达到了预期要求。

1. 前台用户端成果

极致的购票体验：实现了可视化的在线选座功能，支持实时查看座位状态（已售、锁定、维修）。支付流程集成了钱包与密码验证，配合 WebSocket 实现了毫秒级的支付结果通知。

高性能的内容检索：基于 Elasticsearch 构建了强大的搜索引擎，用户可以秒级检索海量电影数据，并享受关键词高亮显示的搜索体验。同时，首页的热门榜单通过 Redis 缓存支撑，极大提升了页面加载速度。

活跃的社区互动：构建了完整的影评生态，用户可以发表评分、撰写影评、回复他人以及点赞互动。系统通过异步消息队列处理点赞流量，确保了高并发下的交互流畅性。

个人中心闭环：实现了从用户注册、实名认证、钱包充值到订单管理的全流程闭环，保障了用户数据的完整性与安全性。

2. 后台管理端成果

数据可视化驾驶舱：集成了 ECharts 图表库，实时展示总票房、今日流水、用户增长趋势等关键运营指标，为管理者提供了直观的数据支持。

全维度的资源管理：实现了对电影、演员、导演、影院、影厅等基础数据的增删改查维护。特别是影厅配置模块，通过 JSON 数据结构灵活支持了各种复杂的座位布局。

智能化的业务逻辑：在排片管理中实现了自动冲突检测算法，有效避免了人为操作导致的排片时间重叠。

安全审计机制：完善的权限拦截与日志审计系统，确保了后台操作的安全性与可追溯性。

5.2 心得体会

本项目的开发过程是一次充满挑战与收获的技术探索之旅。通过从零开始构建一个企业级的互联网应用，我在以下几个方面有了深刻的体会：

1. 对分布式架构的深度理解

在项目初期，我曾认为引入 Redis 和 RabbitMQ 只是为了“炫技”。但在解决“座位超卖”和“订单超时取消”这两个具体痛点时，我深刻体会到了中间件的核心价值。Redis 的原子锁解决了数据库无法处理的高并发竞争，RabbitMQ 的死信队列则优雅地解耦了定时任务与业务逻辑。这些实践让我明白了架构设计不是堆砌技术，而是根据业务场景选择最合适的工具。

2. 全栈开发能力的提升

本项目采用了前后端分离模式，我不仅巩固了 Spring Boot、MyBatis Plus 等后端技术，还深入学习了 Vue 3、Vite 和 Pinia 等前端前沿技术栈。在对接接口的过程中，我学会了如何定义规范的 RESTful API，如何处理跨域问题，以及如何统一前后端的异常处理标准。这种全栈视角的建立，让我对软件工程的整体生命周期有了更宏观的认识。

3. 工程化思维的建立

在开发过程中，我引入了 Nacos 进行配置管理，使用 MinIO 存储非结构化数据，并编写 Docker Compose 文件进行容器编排。这些工作虽然繁琐，但极大地提升了系统的可维护性和可扩展性。我认识到，写出能运行的代码只是第一步，构建一个易于部署、易于扩展、代码规范的系统才是合格工程师的标准。

4. 不足与展望

虽然系统已基本完善，但仍存在优化空间。例如，目前系统仍为单体应用，未来可以利用 Spring Cloud Alibaba 将其拆分为“用户微服务”、“订单微服务”等，进一步提升抗压能力；此外，可以引入协同过滤算法，基于用户的观影历史推荐

电影，增强系统的智能化水平。

总而言之，这次课程设计不仅是一次学业任务的交付，更是一次宝贵的实战演练，为我未来从事 Java 后端开发工作打下了坚实的基础。