剧场订票管理系统

一、课题内容和要求

(1) 预定 7 日内的座位，用图形表示已订座位。

(2) 管理影片类型，票价类型，折扣类型等功能。

(3) 区分管理员和客户 2 种角色。可预定今、明、后三天的票，以计算机的系统日期作为今天的日期；可修改观众的个人资料，包括会员升级管理等；可以取消已订票；预订票可以改订。

(4) 生成简单的报表，包括订单的查询与统计，个人消费的查询与统计，每个影片近期销售走向的统计，建议用可视化图表进行显示。

(5) 生成并动态管理热销影片，设计大数据分析和个性化影片推荐模块。

二、需求分析和总体设计

2.1 本课题的主要功能设计

包含该系统的用例图如下所示：

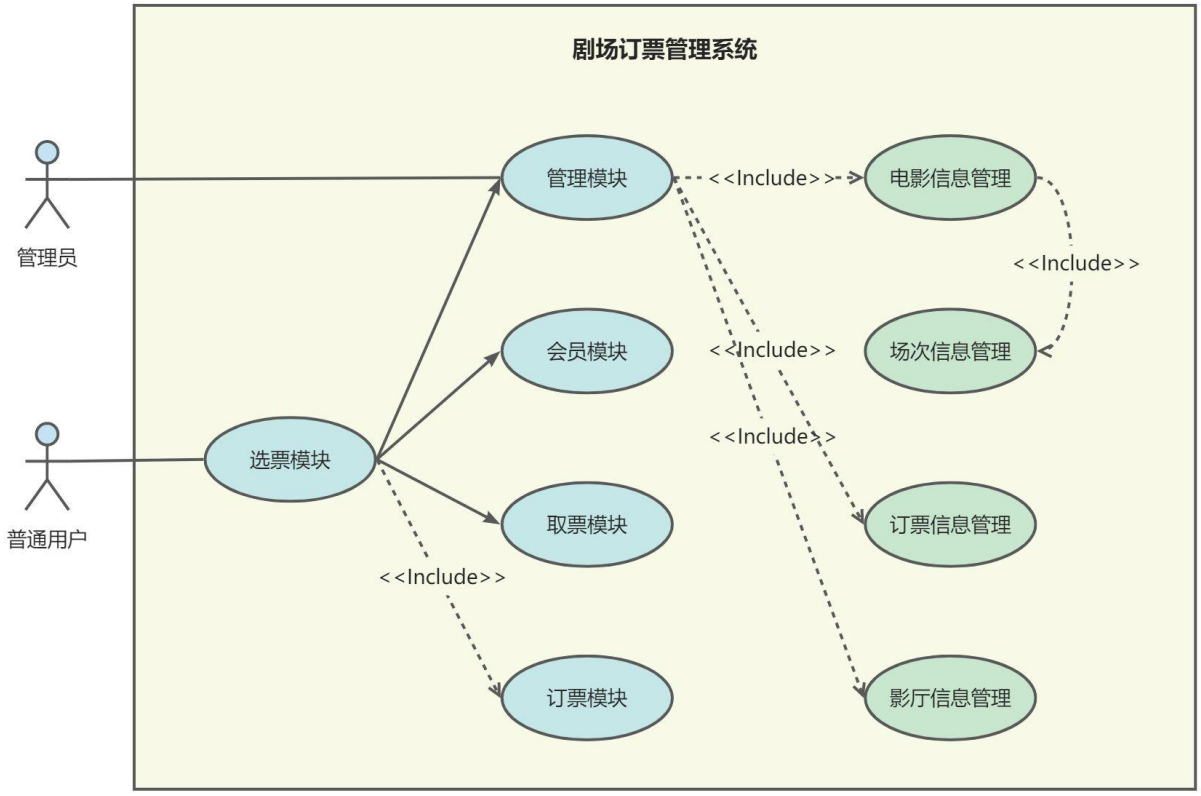

图 1 系统用例图

2.2 本课题的数据表单设计

包含该系统数据表单设计的 ER 图如下图 1 所示：

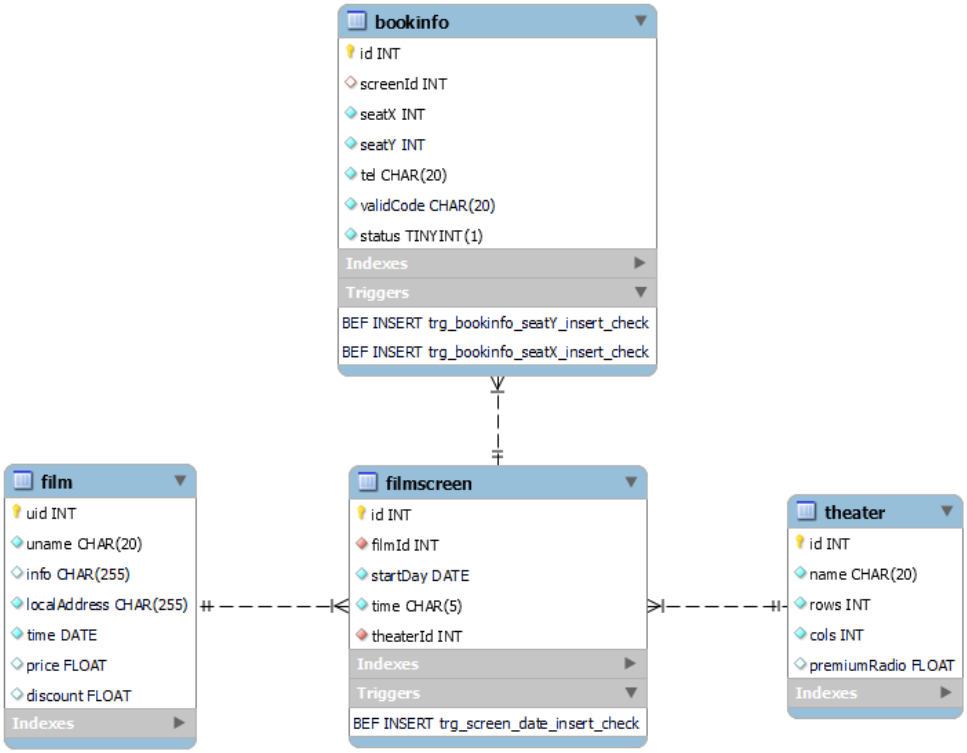

图 2 设计 ER 图

2.3 本课题的体系结构设计

1. 根据需求，将本项目体系结构分为以下 3 层：

   a) Dao 层：负责对数据库的连接与各种数据操作；

   b) Model 层：与数据库各表对应的实体类；

   c) Service 层：负责业务逻辑设计以及界面展示；

2. 项目结构

```
└src
    ├.idea
    ├action (Dao)
    ├entity (Model)
    ├lib
    ├res
└view  (Service)
```

2.4 本课题的主要功能界面设计

界面内容展示如下图所示：

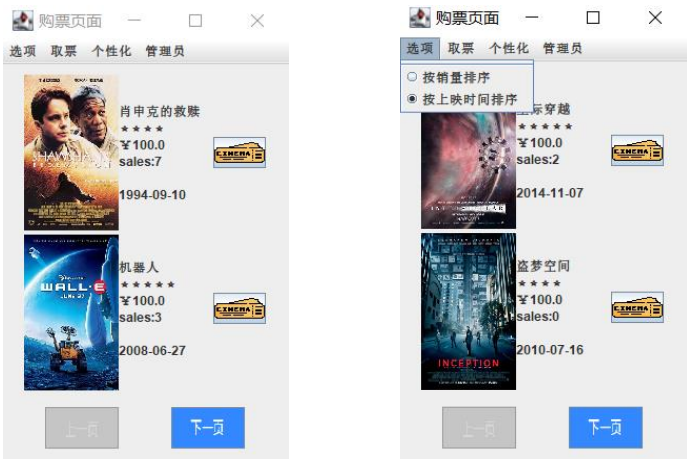

图 3 选票界面

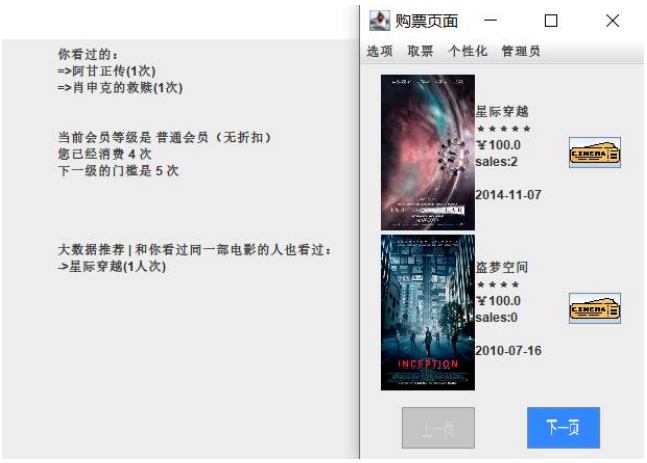

图 4 个性化界面（含会员统计与推荐）

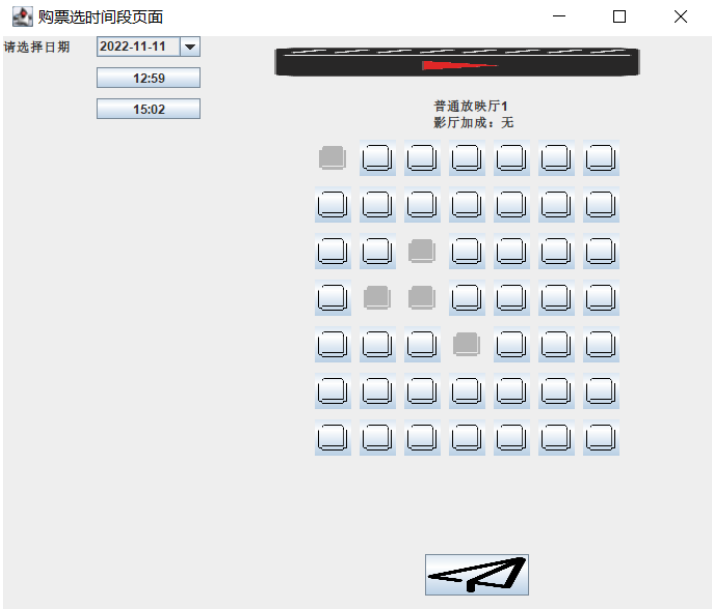

图 5 选座界面

图 6 影片信息管理界面与场次信息管理界面

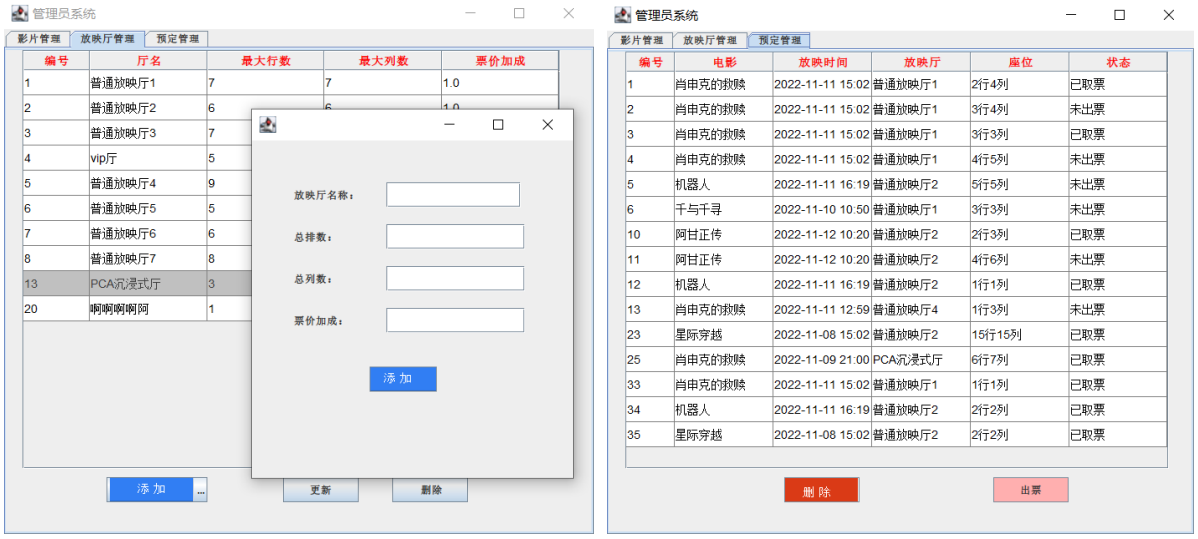

图 7 影厅信息管理界面与订票信息管理界面

界面操作流程如下图 8 所示：

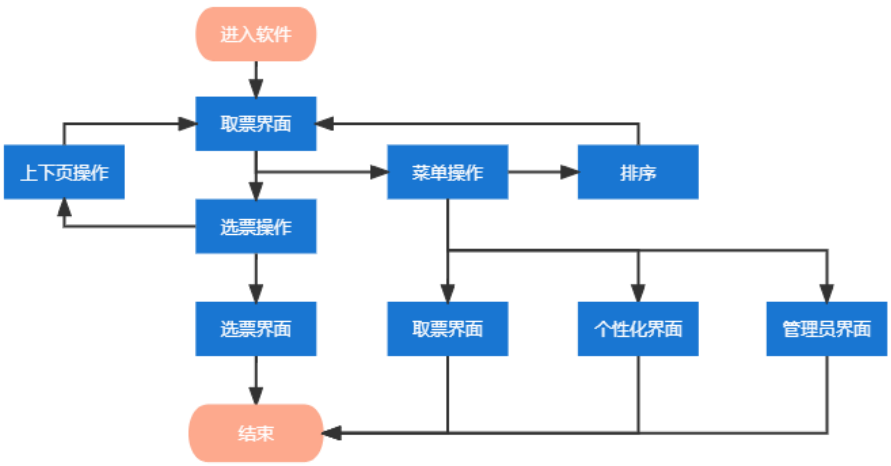

图 8 界面操作流程图

-

三、相关功能模块详细设计

3.1 数据库

film 电影信息表

id	int	PRIMARY AUTO_INCREMENT
filmName	char(20)	UNIQUE NOT NULL
posterUrl	char(255)	NOT NULL *注：本地地址以加载图片*
releaseDay	Date	NOT NULL *注：'YYYY-MM-DD'*
filmInfo	char(255)	
discount	float ~~*float(2,2)*~~ *MySQL 往后版本不再支持。*	DEFAULT 1 CHECK (0.0,1.0]
price	float ~~float(5,2)~~	DEFAULT 100 CHECK >0

screen 电影放映场次表

id	int	PRIMARY AUTO_INCREMENT
filmId	int	FOREIGN => film(id) RESTRICT *注：对所有外键默认此属性，不再重复.* NOT NULL
theaterId	int	FOREIGN => theater(id) NOT NULL
screenDay	Date	NOT NULL CHECK < releaseDay *此处的判断应通过断言实现（课本 5.6 节 p167），然而事实上 MySQL 不支持断言，应改用触发器实现.*[1]
screenTime	char(5)	NOT NULL CHECK *(str_to_date(`time`,'%H:%i')<>null)*

theater 放映厅信息表

id	int	PRIMARY、AUTO_INCREMENT
theaterName	char(20)	UNIQUE
maxRows	int	NOT NULL
maxColumns	int	CHECK >0
premiumRatio	float	NOT NULL

[1] 触发器实现具体代码见标题四（部分核心代码），下同。

book 订票信息表

ID	int	PRIMARY AUTO_INCREMENT
SCREENID	int	NOT NUL FOREGIN => screen(id)
SEATX	int	NOT NULL CHECK >0 && <= maxRows *注：用触发器实现*
SEATY	int	NOT NULL CHECK >0 && <= maxCols *注：用触发器实现*
VALIDCODE	char(20)	NOT NULL
STATUS	Tinyint(bool)	DEFAULT 0 CHECK 0 || 1
TEL	char(20)	NOT NULL CHECK *(char_length()=11)* *注：电话号码具体格式不检查*

从 WorkBench 中导出的 ER 图如下图所示：

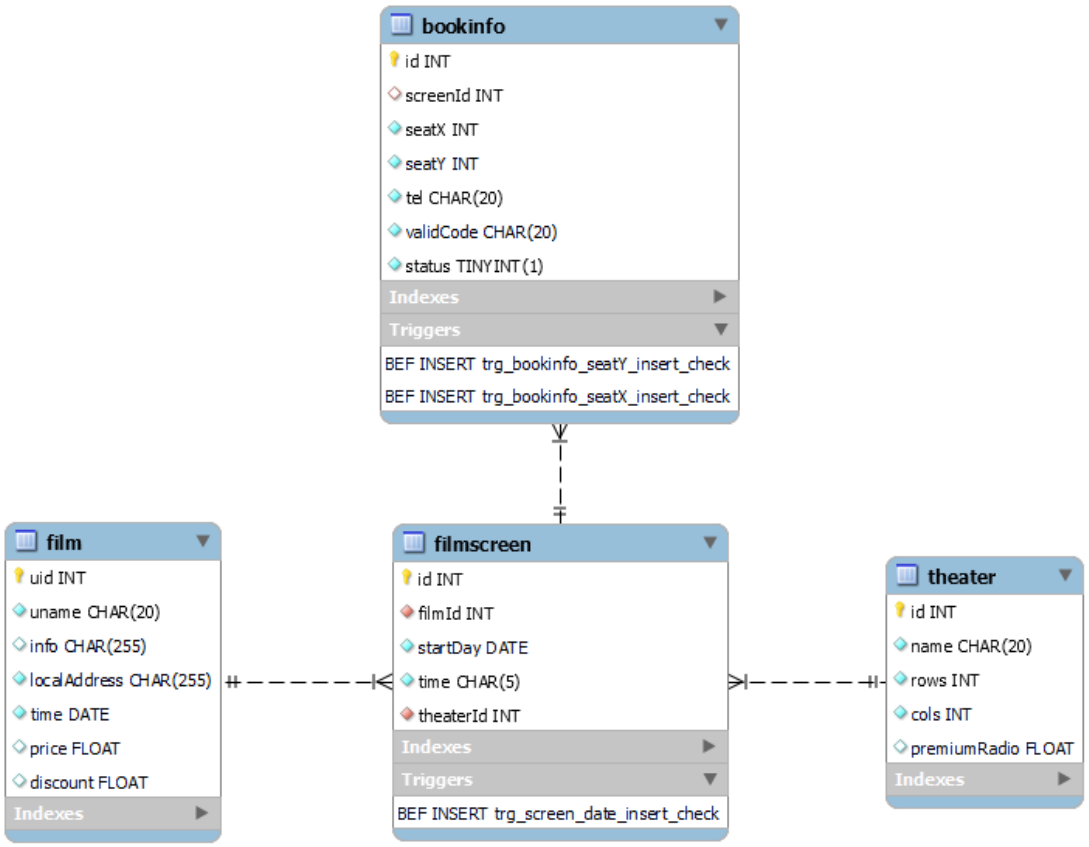

图 9 实现数据表后生成的 ER 图

3.2 Dao 层

数据操作类 Conn.class 主要负责数据库连接与数据操作。此类中的方法在初始化时，会先建立与数据库连接，之后完成对数据库的操作（数据的增删查改）。

访问数据库的操作模板如下所示：

```
Connection conn;
Statement st;
PreparedStatement pst;
ResultSet rs;

// 连接数据库
Class.forName("com.mysql.cj.jdbc.Driver");
conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/[scheme_name]",
"[user_name]", "[password]"); // 方括号`[]`中隐去了笔者的数据库模式名称、用户名、密码

int i;

// 查询方法 1
st = conn.createStatement();
String sql = "select startDay from filmscreen where id=" + i;
rs = st.executeQuery(sql);
rs.next();
rs.getDate(1);
// 查询方法 2
String sql2 = "select startDay from filmscreen where id=?";
pst = conn.prepareStatement(sql2);
pst.setInt(1, i);
rs = pst.executeQuery();
rs.next();
rs.getDate(1);

// 显然方法 2 的 sql 语句表示方法更适合增删改

// 接下来以这种方法给出插入的例子，删改只有 sql 语句内容的不同
String sql = "insert into filmscreen (filmId,startDay,time,theaterId)values(?,?,?,?)";
pst = conn.prepareStatement(sql);
pst.setInt(1, screen.getFilmId());
pst.setDate(2, screen.getStartDay());
pst.setString(3,screen.getTime());
pst.setInt(4, screen.getTheaterId());
pst.executeUpdate();
```

3.3 Model 层

本层的各个实体与数据库每张表各自对应，任何实例化的实体都对应于数据库中某张表上的一条数据（一个元组）；在 Dao 层我们通过建立与数据库的连接后，增删查改操作的方法接收的和返回的均是由本层所定义的实体；在 Service 层通过调用 Dao 层操作获得的数据形式是由本层定义的实体。

可将该层各实体类结构（见下图 12）与数据库表结构（见上图 9）进行对比：

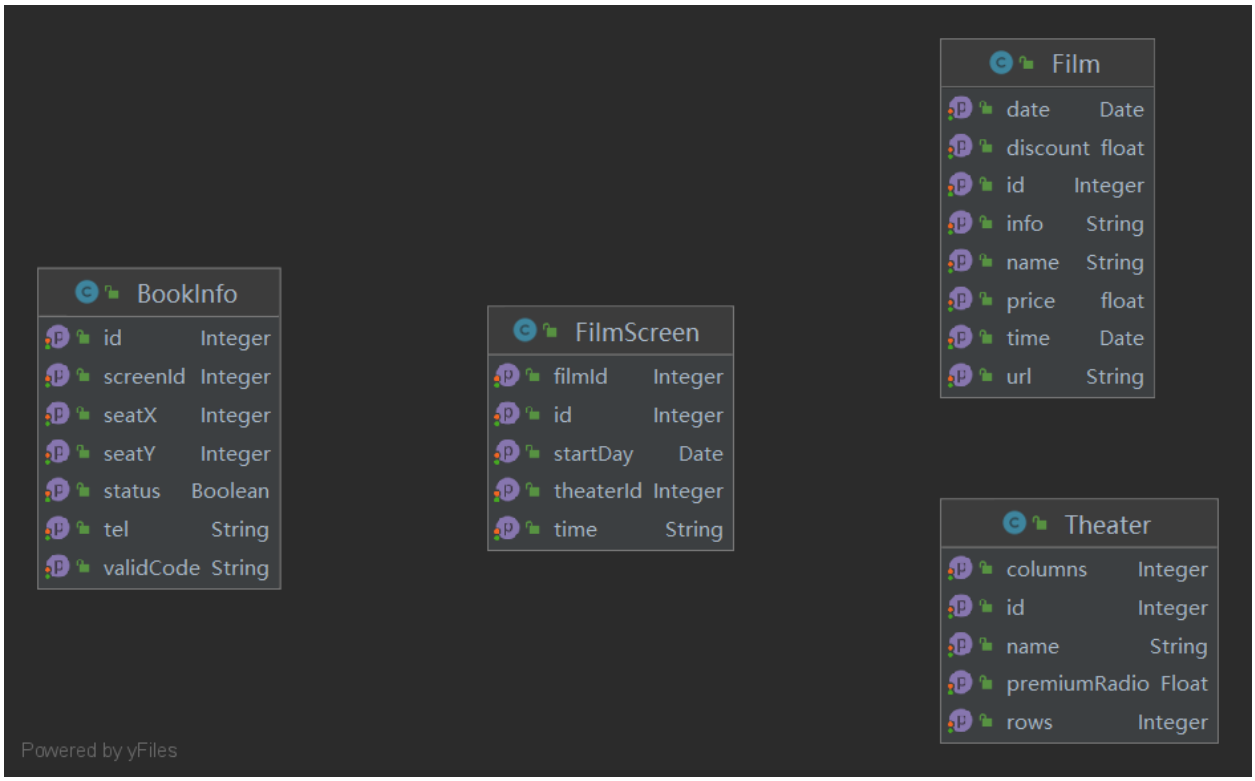

图 10 各实体类结构

3.4 Service 层

各界面业务逻辑关系如下图 11 所示：

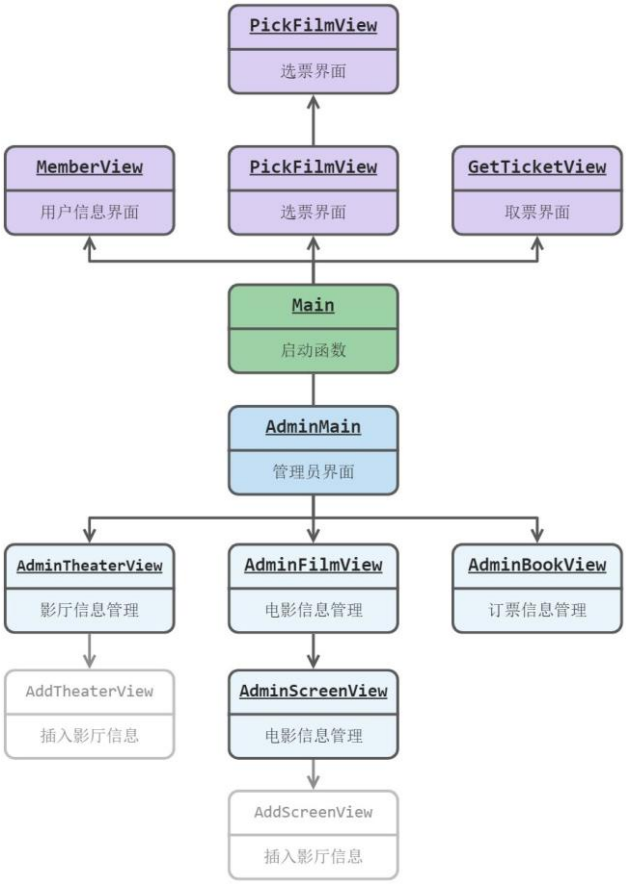

图 11 Service 层业务逻辑关系

需要再次强调，本项目中，所有涉及表中数据的条件查询，都通过对数据库的条件查询实现（即封装在 Conn.class 中的函数），代码本身主要负责业务逻辑构建。

采用 Java Swing 搭建界面的基本代码框架如下所示：

```
JFrame frame = new JFrame("窗口标题");
frame.setBounds(x,y,width,height);

JPanel panel = new JPanel();
panel.setLayout(null);
panel.setBounds(x,y,width,height);

    JLabel element = new JLabel("我可以是任何装于 panel 的页面元素");
    element.setBounds(x,y,width,height);
        /* ... */
    panel.add(element);

frame.add(jp);

frame.setVisible(true);
frame.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
```

这里额外阐述对于管理员界面的设计，管理员界面采用选项卡的形式（1 个 Frame 中包含 3 个 Panels）呈现不同板块的数据管理，如下所示：

```
// 创建 JFrame 实例
JFrame jf = new JFrame("管理员系统");
jf.setSize(720, 650);
jf.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
jf.setLocationRelativeTo(null);

    // 创建选项卡面板
    final JTabbedPane tabbedPane = new JTabbedPane();
    // 创建第 1 个选项卡（选项卡包含 标题、图标 和 tip 提示）
    tabbedPane.addTab("影片管理", new ImageIcon("bb.jpg"), new AdminFilmView().getPanel()," 影片信息");
    // 创建第 2 个选项卡
    tabbedPane.addTab("放映厅管理", new ImageIcon("bb.jpg"), new AdminTheaterView().getPanel(),"放映厅信息");
    // 创建第 3 个选项卡
    tabbedPane.addTab("预定管理", new ImageIcon("bb.jpg"), new AdminBookView().getPanel(), " 订票信息");
    // 添加选项卡选中状态改变的监听器
    tabbedPane.addChangeListener(new ChangeListener() {
        @Override
        public void stateChanged(ChangeEvent e) {
            System.out.println("当前选中的选项卡: " + tabbedPane.getSelectedIndex()+1);
        }
    });

// 配置 frame
jf.setContentPane(tabbedPane);
jf.setVisible(true);
```

在`2.4 本课题的主要功能界面设计`的图 6、图 7 中，向读者展示了管理员模式下的界面，显然各个信息管理模块拥有相同的构建结构，下面给出各个 panels 的结构，它们都属于图 11 中管理员界面下的各个子界面：

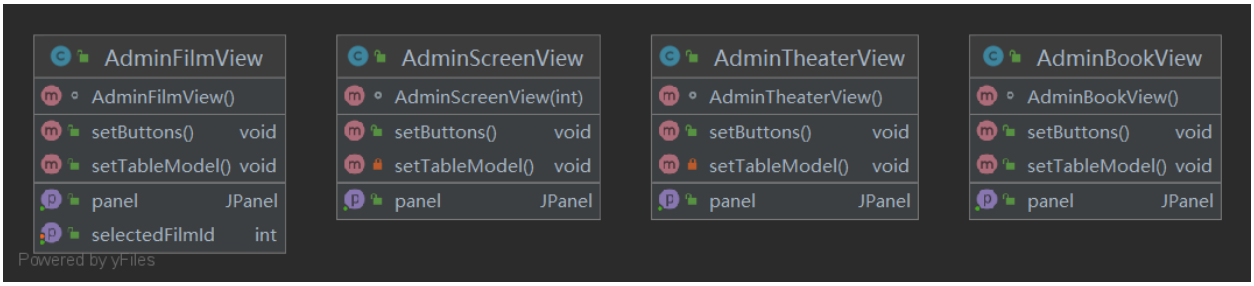

图 12 管理员界面（AdminMain）下各个界面结构

在他们的构造函数中，拥有相同的初始化操作：

```
// table
setTableModel();
table=new JTable(tableModel);
new Utils().setTableAttribute(table,tableModel);
// scroll panel
jsp = new JScrollPane();
jsp.setBounds(20,0,650,500);
jsp.setViewportView(table);
// panel
panel = new JPanel();
panel.setSize(600, 600);
panel.setLayout(null);
panel.add(jsp);
// button
setButtons();
```

四、部分核心代码

4.1 数据库设计

展示部分操作：

```
# 载入数据（以 bookinfo 表为例）
LOAD DATA INFILE 'D:\\Java\\Theater-booking-system-
requirements\\front\\src\\testData\\bookinfo.txt'
    INTO TABLE bookinfo
    FIELDS TERMINATED BY '\t'
    LINES TERMINATED BY '\r';
select * from bookinfo;

# 设置候选码
alter table filmscreen add unique `screen-info`(`filmId`,`startDay`,`time`);

# 放映日期约束
-- 此句使用 check 无法实现，需断言，而 mysql 不支持断言
alter table filmscreen add CONSTRAINT `time-check`
check(
    `startDay`>(select time from film where film.id=filmId)
);
-- 添加触发器以实现断言
CREATE TRIGGER trg_screen_date_insert_check BEFORE INSERT
ON filmscreen FOR EACH ROW
SET NEW.startDay = IF(
    (
```

```
        select time from film where film.uid=NEW.filmId
    ) < NEW.startDay,
    NEW.startDay,            -- true
    NULL -- false, 利用列自身约束 NOT NULL 阻止本次插入
);
drop trigger trg_screen_date_insert_check;
-- 测试 split value: 1994-09-10
insert into filmscreen values (0,1,'1994-09-07','12:00',3);
insert into filmscreen values (0,1,'1994-09-11','12:00',3);

# 选座 x 范围限制，必须在范围内
-- 方法 1,子查询
CREATE TRIGGER trg_bookinfo_seatX_insert_check BEFORE INSERT
ON bookinfo FOR EACH ROW
SET NEW.seatX = IF(
    (
        NEW.seatX <= (
            SELECT `rows` FROM theater
            WHERE theater.id=(SELECT theaterId FROM filmscreen WHERE
filmscreen.id=NEW.screenId)
       ) and NEW.seatX>0
    ),
    NEW.seatX,   -- true
    NULL     -- false, 利用列自身约束 NOT NULL 阻止本次插入
);
drop trigger trg_bookinfo_seatX_insert_check;
-- 测试
insert into bookinfo values (0,17,15,3,'11111111111','111111',0);
insert into bookinfo values (0,17,3,3,'11111111111','111111',0);

#选座 y 范围限制，必须在范围内
-- 方法 2,连接查询
alter table bookinfo add CONSTRAINT `seatY-check`
check(
    `seatY` >0 and
    `seatY` <= (
            select cols from (
                select filmscreen.id as sId, cols from filmscreen join theater on
filmscreen.theaterId=theater.id
            ) AS st
            where screenId=st.sId
    )
);
-- 添加触发器以实现断言
CREATE TRIGGER trg_bookinfo_seatY_insert_check BEFORE INSERT
```

```
ON bookinfo FOR EACH ROW
SET NEW.seatY = IF(
    (
        NEW.seatY <= (
            select cols from (
                select filmscreen.id as sId, cols from filmscreen join theater on
filmscreen.theaterId=theater.id
            ) AS st
            where NEW.screenId=st.sId
       ) and NEW.seatY>0
    ),
    NEW.seatY,   -- true
    NULL     -- false, 利用列自身约束 NOT NULL 阻止本次插入
);
drop trigger trg_bookinfo_seatY_insert_check;
-- 测试
insert into bookinfo values (0,17,15,15,'11111111111','111111',0);
insert into bookinfo values (0,17,3,3,'11111111111','111111',0);

# 操作表
-- 获取某影片(e.g.filmId=5)的放映日期列表
select startDay from filmScreen where filmId=5 group by startDay;
-- 获取某影片(e.g.filmId=3)某日(e.g.startDay='2022-12-21')的放映时间列表
select id from filmscreen where filmId=3 and Date(startDay)='2022-12-21';
-- 获取某影片(e.g.filmId=3)某日(e.g.startDay='2022-12-21')某时间(e.g.time='11:40')的放映厅
select theaterId from filmscreen where filmId=3 and Date(startDay)='2022-12-21' and
time='11:40';
-- 获取影片列表，按销量、上映时间、id 依次排序
select * from film left outer join (
    select filmscreen.filmId as fId, count(filmscreen.filmId) as sales
    from filmscreen, bookinfo
    where filmscreen.id=bookinfo.screenId
    group by fId
)As sales on sales.fId=film.uid
order by sales desc, `time` desc, uid;
-- 查询某部电影(e.g.id=14)销量
select count(filmscreen.filmId) as sales
from filmscreen, bookinfo
where filmscreen.id=bookinfo.screenId and filmscreen.filmId=2;
-- 测试：以 id=14 的日期为基准，调整 filmscreen 中的数据，从而实现未来 7 日内有数据可用
select startDay from filmscreen where id=14 -- 在下一条语句问号中应用这条语句结果
update filmscreen set startDay = date_add(startDay, interval datediff(now(),?) day)
-- 查询近 7 日的场次信息清单
select startDay from filmScreen where filmId=5 and datediff(startDay,now())>=0 and
datediff(startDay,now())<=7 group by startDay
```

4.2 Dao 层

各方法结构相似，已在`3.2 Dao 层`向读者介绍过了下面展示本项目中最复杂的查询方法：“你可能喜欢的影片”推荐算法:

```
/**
 * 根据看过相同影片的人们的观影情况，推荐本人还没看过的电影
 * @return 适应 label 内容与格式的一串字符串
 */
public String Recommend(String tel)
{
    // 建立连接
    getConn();
    String temp = "<br>";
    try {
        String sql = new String(Files.readAllBytes(Paths.get("src/res/ ")));
        PreparedStatement preparedStatement = conn.prepareStatement(sql);
        preparedStatement.setString(1, tel);
        preparedStatement.setString(2, tel);
        ResultSet resultSet = preparedStatement.executeQuery();
        while (resultSet.next()) {
            // 要注意这里二次调用了类内函数，意味着在本查询结果未完全输出的情况下，
            // 类成员变量 rs pst 会被所调用的函数使用，导致查询出错！
            // 报错内容："Cannot determine value type from string ’xxx‘"
            String str = this.getFilmById(resultSet.getInt(1)).getName();
            int per = resultSet.getInt(2);
            temp += "->" + str + "(" + per + "人次)<br>";
        }
        return temp;
    } catch (SQLException | IOException e) {
        e.printStackTrace();
        return "-数据库出错-";
    }
}
```

推荐算法的核心的 sql 语句由于太长，直接写入代码中会严重影响可读性，因此采用从外部 txt 文件导入的方式输入，`sql_recommend.txt` 中 SQL 代码如下：

```
select filmId, sum(amount) as per from (

    select filmId, count(*) as amount, tel from (
        select bookinfo.id as id,filmId,filmscreen.id as screenId,tel
        from filmscreen, bookinfo
        where filmscreen.id=bookinfo.screenId
        group by filmId, screenId
    ) as table1_3 group by filmId, tel
     -- 已经去同场电影的重复订票：每个人每个电影的观看次数 table2
```

```
) as table2_3 where tel in (

    select tel2 from (

        select table2_1.filmId as fid1, table2_1.amount as amount1, table2_1.tel as tel1,
            table2_2.filmId as fid2, table2_2.amount as amount2, table2_2.tel as tel2
        from (

            select filmId, count(*) as amount, tel
            from (
                select bookinfo.id as id,filmId,filmscreen.id as screenId,tel
                from filmscreen, bookinfo
                where filmscreen.id=bookinfo.screenId
                group by filmId, screenId
            ) as table1_1
            group by filmId, tel
             -- 已经去同场电影的重复订票：每个人每个电影的观看次数 table2

        ) as table2_1 join (

            select filmId, count(*) as amount, tel
            from (
                select bookinfo.id as id,filmId,filmscreen.id as screenId,tel
                from filmscreen, bookinfo
                where filmscreen.id=bookinfo.screenId
                group by filmId, screenId
            ) as table1_2
            group by filmId, tel
             -- 已经去同场电影的重复订票：每个人每个电影的观看次数 table2

        ) as table2_2

        ON table2_1.filmId=table2_2.filmId and table2_1.tel<>table2_2.tel

    ) as join_table_2
    where tel1='19850055597' --'19850055597'替换成? 作为变量
    -- 和某个人看过相同电影的人，找到了 table6

) and filmId not in (

    select distinct filmId
    from (
        select bookinfo.id as id,filmId,filmscreen.id as screenId,tel
        from filmscreen, bookinfo
        where filmscreen.id=bookinfo.screenId
```

```
        group by filmId, screenId
    ) as table1_4_1
    where tel='19850055597'       --'19850055597'是示例，在 txt 文件中应该替换成?
     -- 某个人看过的电影

) group by filmId
order by per desc;
```

推荐算法的原理是遍历订票信息表 bookinfo，首先查找与某个用户看过相同电影的用户群体，之后查询这个用户群体看过的电影，并统计各电影的看过的人次[2]，同时排除原用户已经看过的电影，返回电影编号及其统计人次（按观看人次进行排序）。

4.3 Model 层

实体类的结构十分简单，由和数据库同名表一一对应的属性作为类内成员，并设置实体构造函数（constructor）、Getter、Setter 函数，因此在此处略去。

4.4 Service 层

界面初始化在构造函数中完成，构造函数的内容见 `3.4 Service 层`最后一段代码，以下以电影信息管理页面（AdminFilmView）为例，实现了构造函数中的两个重要方法：

```
// 设置表头，数据稍后导入
public void setTableModel() {
    tableModel = new DefaultTableModel(Film.COLUMN_NAMES, 0){
        @Override
        public boolean isCellEditable(int row, int column) {
            return column != 0;
        }
    };
    tableModel.addTableModelListener(new TableModelListener() {
        @Override
        public void tableChanged(TableModelEvent e) {
            if(e.getType() == TableModelEvent.UPDATE){
                String currentValue =
tableModel.getValueAt(e.getFirstRow(),e.getColumn()).toString();
                // 下一行代码在各个 panel 中是不同的，以 Film 为例
                Film film = new Conn().getFilmById((Integer)
tableModel.getValueAt(e.getLastRow(),0));
                String originalValue = film.getVarByIndex(e.getColumn());
                if (!currentValue.trim().equals(originalValue.trim())) {
                    if(!film.setVarByIndex(e.getColumn(),currentValue)) {
                        JOptionPane.showMessageDialog(null, "请检查日期格式！","状态",
```

[2] 一个用户同一场次购买多张票仅算 1 人次，一个用户相同电影的不同场次可以累加人次。

```
JOptionPane.ERROR_MESSAGE);
                            tableModel.setValueAt(originalValue, e.getFirstRow(),
e.getColumn());
                        } else {
                            if (updateList == null) {
                                updateList = new ArrayList<>();
                                updateList.add(film);
                            } else if (updateFilmList.isEmpty()) {
                                updateList.add(film);
                            } else {
//                              if (checkDuplication(updateList, film)) {
//                                  updateList.add(film);
//                              }
                                updateList.add(film);
                            }
                        }
                    }
                }
            }
        });
        // 获取并插入数据
        filmList = new Conn().getFilmList(1);
        for (Film film : filmList) {
            Vector rowV = new Vector<>();
            rowV.add(film.getId());
            rowV.add(film.getName());
            rowV.add(film.getInfo());
            rowV.add(film.getUrl());
            rowV.add(film.getDate().toString());    // 日期形式则无法修改
            rowV.add(film.getPrice());
            rowV.add(film.getDiscount());
            tableModel.addRow(rowV);
        }
    }
    public void setButtons() {
        // 更新数据按钮
        btnUpdate = new JButton("更新");
       ImageIcon imgUpdate = new ImageIcon("src/res/icons/updateBatch.png");
       imgUpdate.setImage(imgUpdate.getImage().getScaledInstance(140, 30,Image.SCALE_DEFAULT));
       btnUpdate.setIcon(imgUpdate);
        btnUpdate.setBounds(120,510,120, 30);
        btnUpdate.addActionListener(new BtnUpdateAction());
        panel.add(btnUpdate);
        // 其余按钮类似，此处略
    }
```

```
class BtnUpdateAction implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (updateFilmList==null || updateFilmList.isEmpty()) {
            JOptionPane.showMessageDialog(null,"没有数据需要更新","状态",
JOptionPane.WARNING_MESSAGE);
            return;
        }
        boolean flag = true;
        for (Film item: updateFilmList) {
            if (!new Conn().updateFilm(item)){
                flag = false;
                break;
            }
        }
        JOptionPane.showMessageDialog(null, flag?"更新成功":"更新失败","状态",
JOptionPane.INFORMATION_MESSAGE);
    }
}
```

五、软件测试及其结果分析（手工）

1. **边界条件限制。**如不在 7 天内的场次信息不会显示：

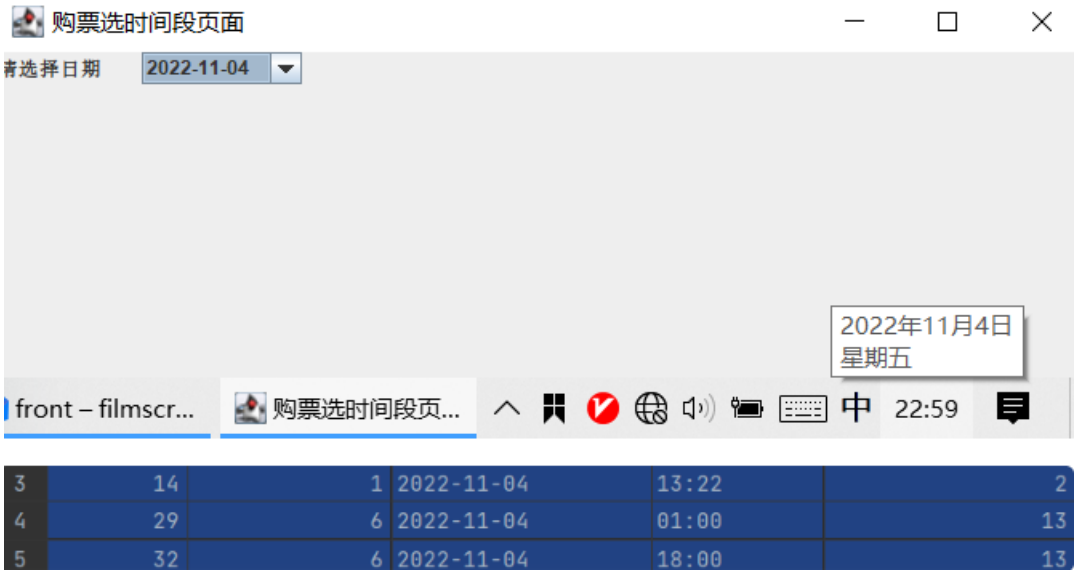

图 13 超过当天时间点的场次不予显示

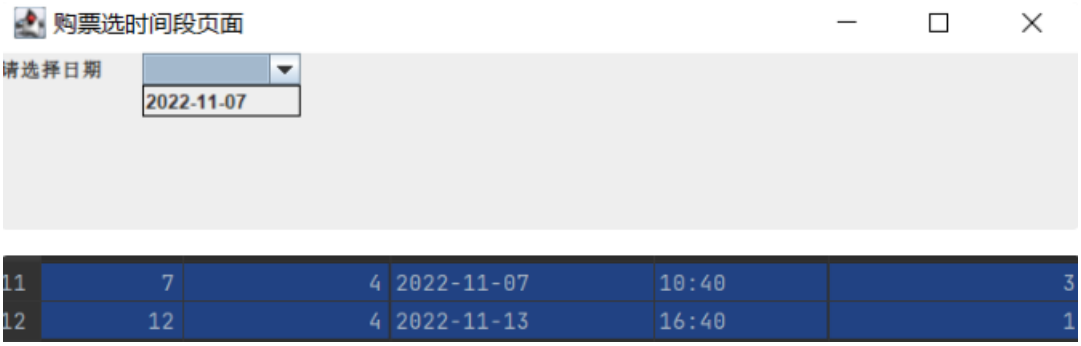

图 14 超过 7 天的场次不予显示

2. **已定座位不可选择**（见图 5 选座界面）

3. **信息管理时格式限制：**

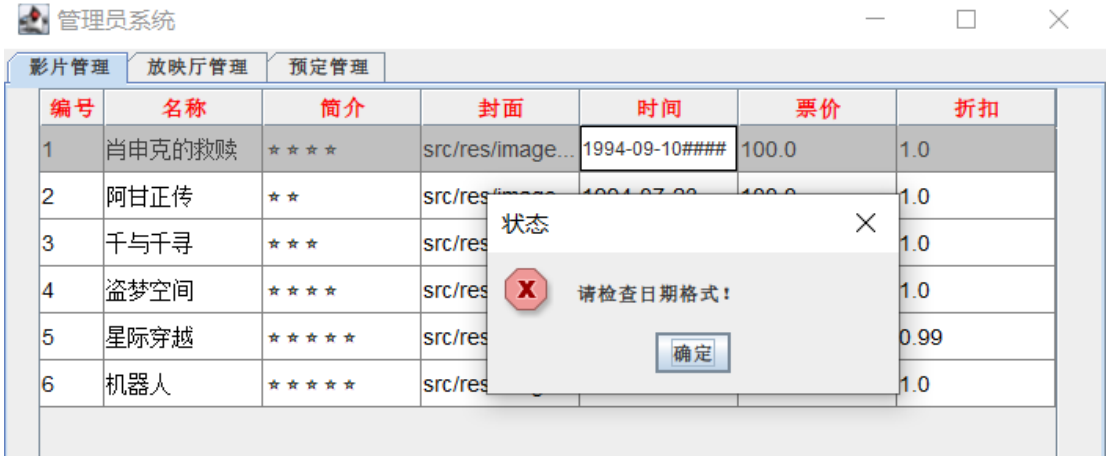

图 15 日期格式不符合不允许更新

图 16 作为外键的属性不可直接删除

图 17 不在要求范围内的值不允许更新

以上每种类型的错误只展示一次，实际上对于表中拥有 check 约束或 trigger 实现的断言的属性，都需要检查范围。

在业务逻辑层（Service）之外，笔者早在数据库构建是配置了相应的约束，即便此处不检查范围，更新也不会成功，数据库插入时也会报错。

4. **已取票的订单不能重复取票：**

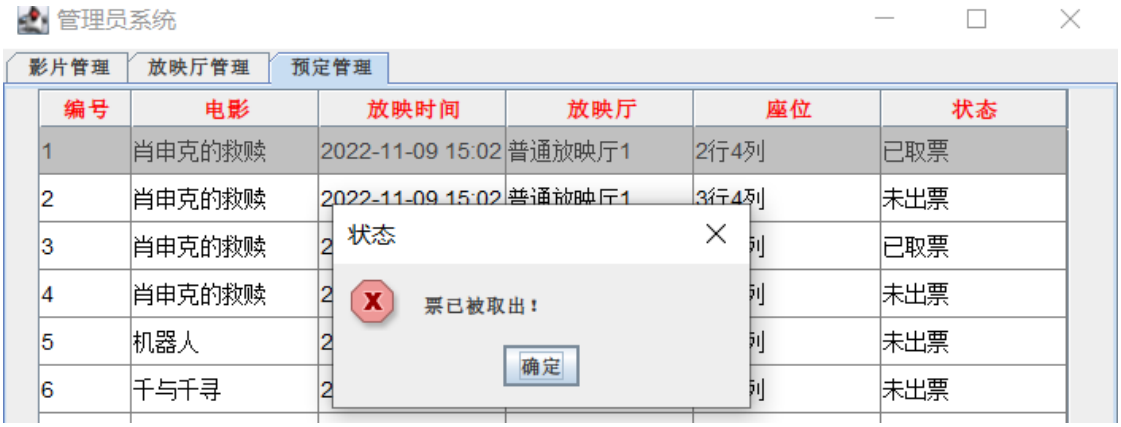

图 18 管理员取票约束

图 19 用户取票约束