

《数据库课程设计报告》

——火车售票系统

目录

第一章、需求分析3

 一、项目概述3

 二、功能需求3

 1、出行方案查询3

 2、购票5

 3、购票记录查询7

 4、改签7

 5、退票8

 6、黑名单管理9

 7、车次管理9

 8、发布通知10

第二章、系统设计11

 一、系统开发平台及框架11

 二、系统整体架构及模块划分11

 三、界面设计12

第三章、数据库设计18

 一、数据库逻辑设计18

 1、ER图18

 2、数据库表设计18

 二、数据库物理设计21

 1、索引21

 2、视图22

第四章、系统实现23

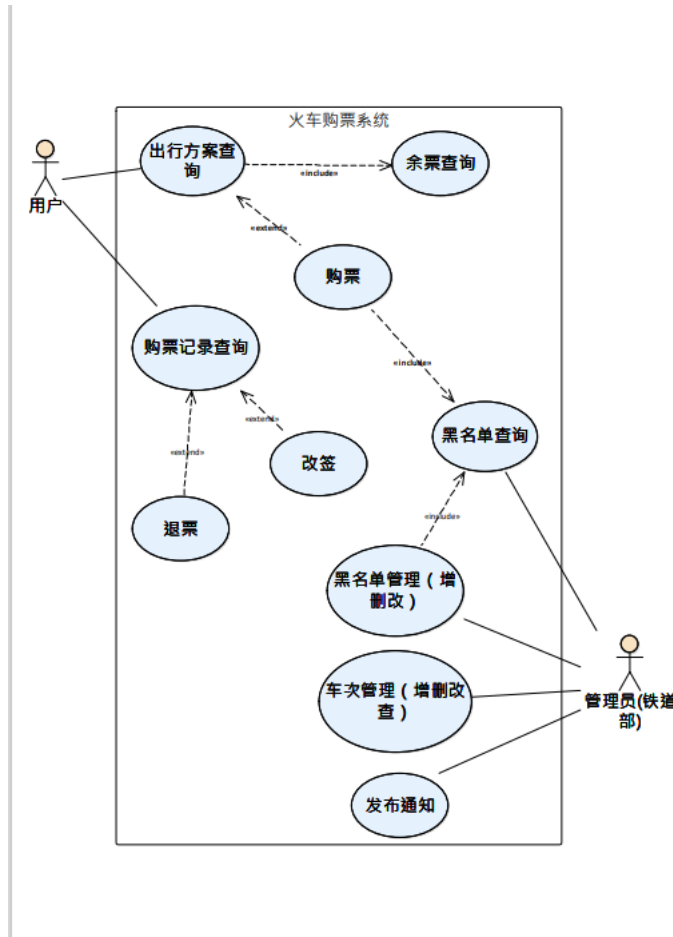
 前端：23

 后端：24

第一章、需求分析

一、项目概述

本系统为火车售票系统，应具有以下功能：查询余票、买票、退票、改签、查询订单，车次、坐席以及用户的管理等。



二、功能需求

1、出行方案查询

首先该功能是面向普通用户的。

对该功能，先从其输入输出来进行考虑。用户的输入应该包含以下几部分：出发地、目的地、出行的日期、是否换乘，换乘地；系统的输出就是查询到的可行的方案，其中每个可行的方案应该包含以下信息：车次号、该方案中车次的各种类型座位的余票数量及价格、出发时刻和到达目的地的时刻及经过的时间，从

起点站到终点站经停火车站的站名、到达时间、出发时间。

接下来是对输入输出的进一步的描述：

对用户输入中的出发地和目的地，不是用户随便输入的，是用户在可选的出发地和目的地中选择，这些可选的出发地和目的地是从数据库中查询得到的，另外，这些出发地和目的地不仅可以是某个火车站，也可以是某个城市

对用户输入中的出行日期，只能是只能是当天以及之后的日期，不能是已经过去的某一天的日期。

用户输入中的是否换乘，它是一个可选的输入，默认是不换乘，如果选择了换乘，只输出换乘一次的方案。

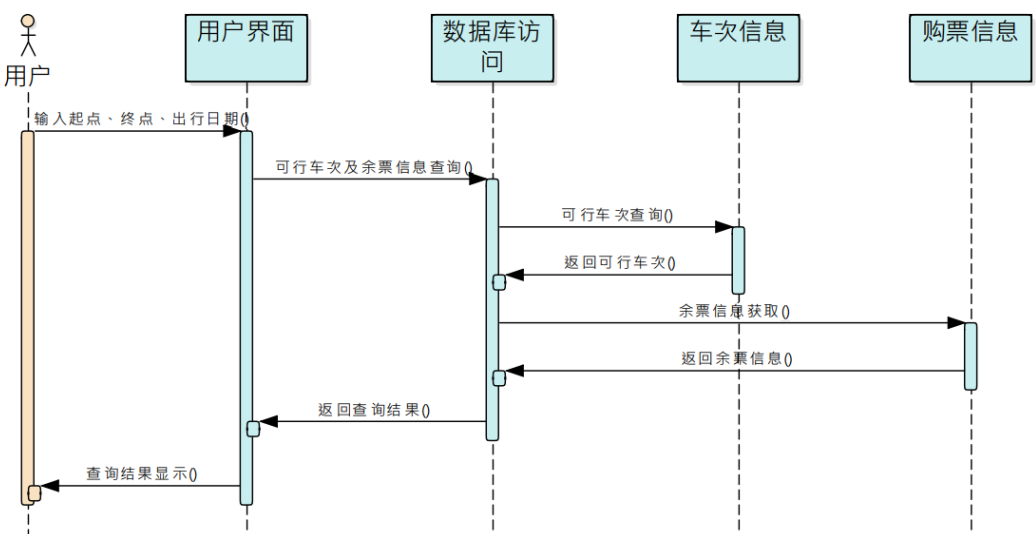
对用户输入中的换乘地，应该是当用户选择了换乘之后才能指定换乘地，同样换乘地可以是车站也可以是城市。

对于系统输出，可行的方案分两类，一类是直达的方案，那就只有一趟车次，包含的信息如上所述；另一类是换乘的方案，那就有两趟车次，每趟车次都应该包含如上的信息，除此之外，还应该有一些总的信息：出发站、中转站、终点站、出发的时间、中转等待的时间、到达时间、总历时。

另一方面，对系统输出的所有方案，系统应当支持如下排序的选项：按价格排序、按历时排序，如果是换乘还可以按中转等待的时间排序。

下面是对该功能的一些额外的说明：

- 1、系统应当是优先输出有余票的方案
- 2、对换乘来说，系统输出的方案应当合理，不能越走越远，比如从济南到沈阳的换乘，中转站是杭州，那就是先从济南到杭州，再从杭州到沈阳，这是不合理的。
- 3、出行方案查询分两种，一种是直达的查询，一种是换乘的查询，其中直达因为只有一趟车次，不需要考虑日期的问题，对换乘，因为有两趟车次，在特定的查询钟，两趟车次对应的出发日期可能是不一样的，这是需要特别注意的



2、购票

首先该功能是面向普通用户的。

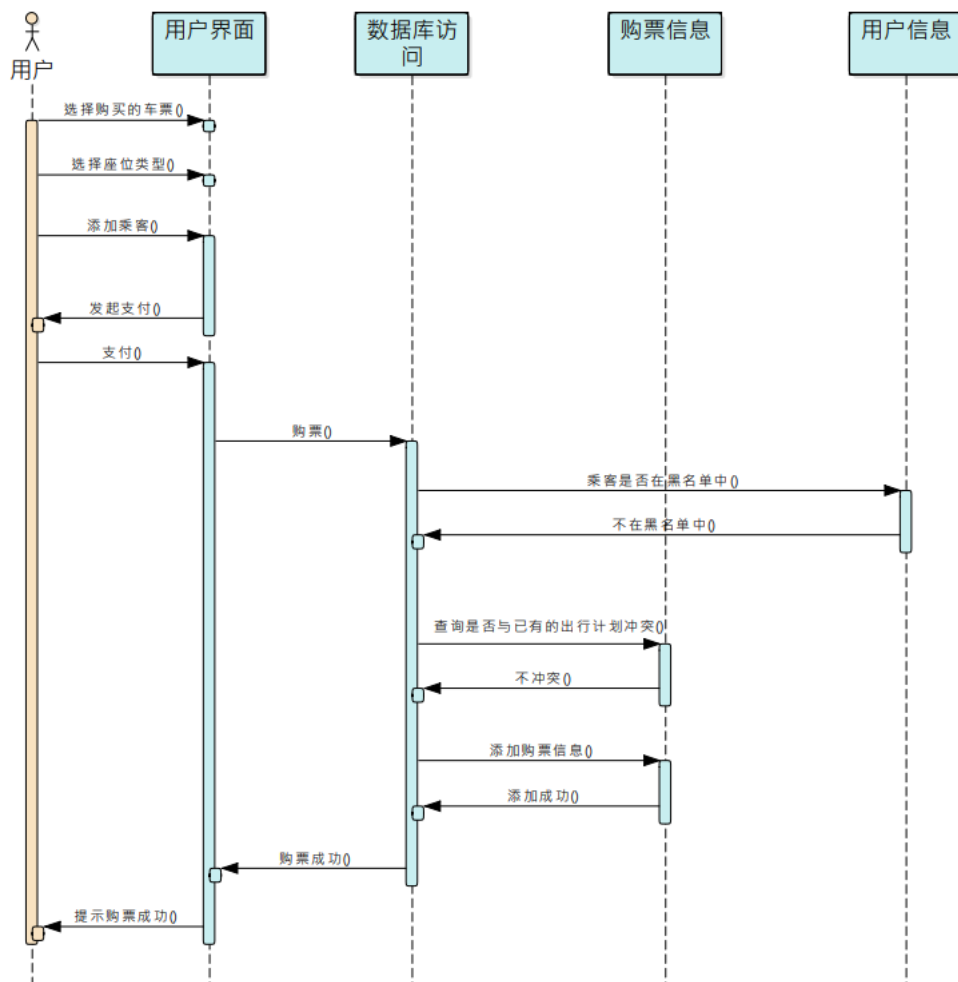
用户在出行方案查询后选中某一个方案，然后可以购票，可能是直达的方案也可能是换乘的方案，用户可以选择要购买的座位类型，系统要为用户分配可用的座位，然后在用户付款后则购票成功。

应该注意的是，用户不止能为他自己买票，用户可以添加多个乘车人，同时为这些乘车人都买票。

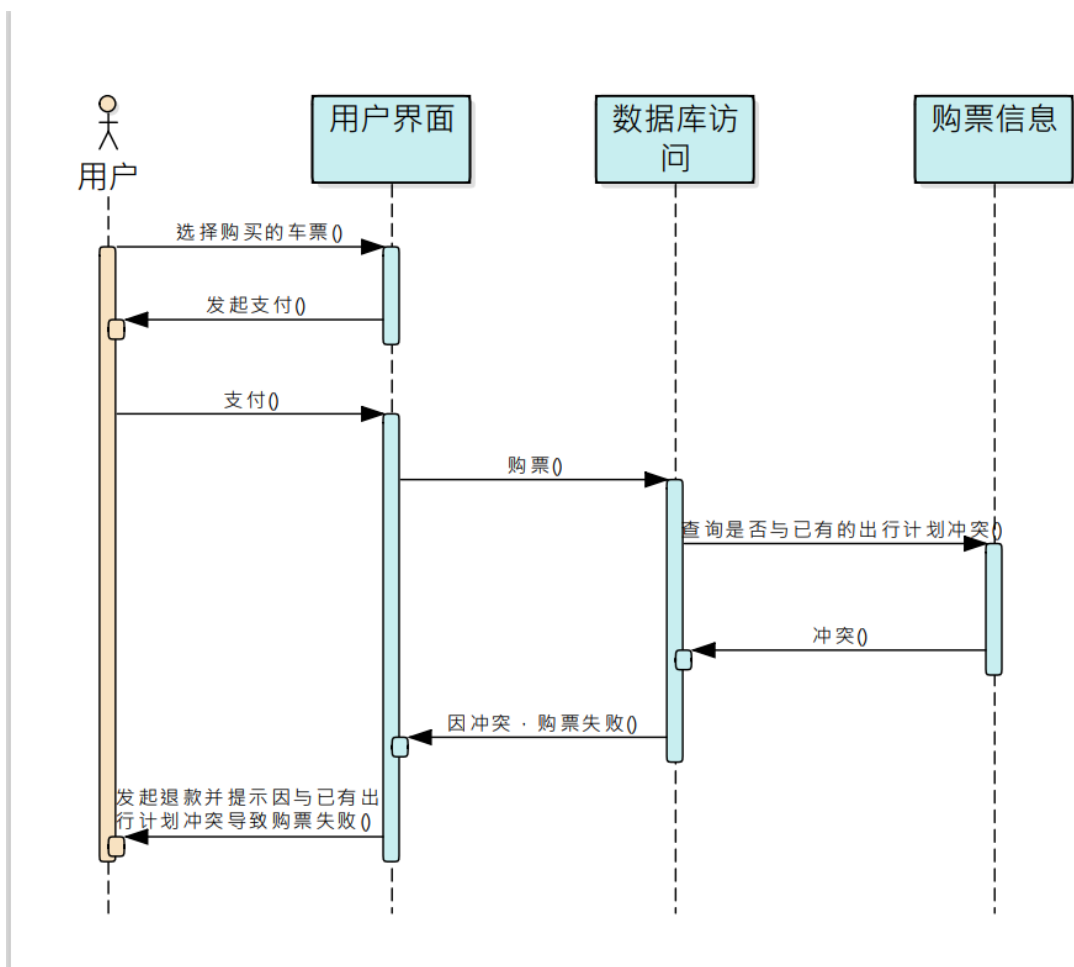
购票中系统要做两个额外的操作：

- 1、要查询乘车人是否在黑名单中，若乘车人在黑名单中，则禁止其买票
- 2、要查询乘车人是否与已有的出行计划冲突，例如：用户要购买这周日中午 12 点济南到沈阳的车票，但已经购买了这周日中午 12 点济南到北京的车票，则禁止其购买。

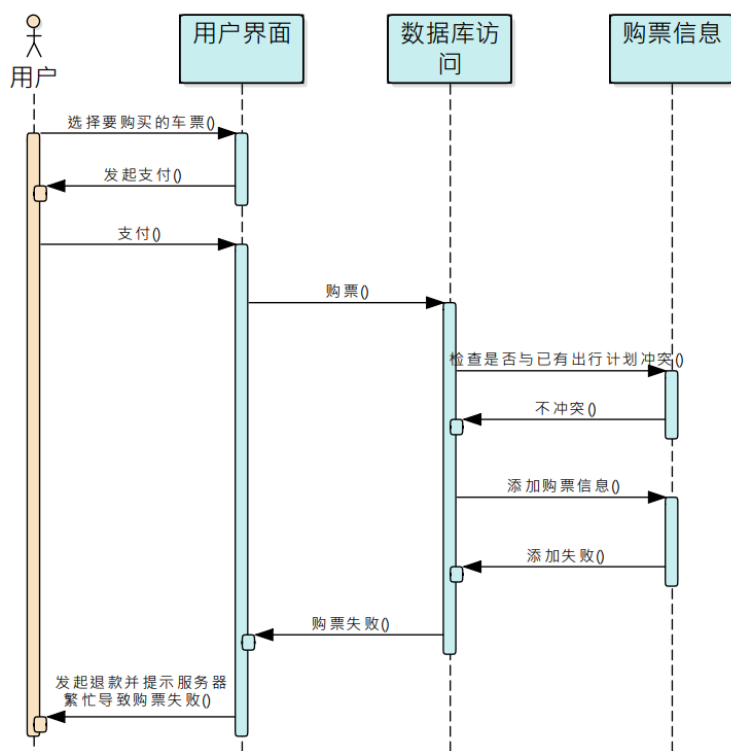
正常购票：



出行计划冲突导致购票失败：



购票添加失败：



3、购票记录查询

首先该功能是面向普通用户的。

用户可以看到已经购买的票，包含一些基本信息：票价、车次、车厢号、座位号、票价、出发地、目的地、出发时间、到达时间、出发日期。

已经购买的票分两种，一种是未过期的票，一种是已过期的票。未过期有直达的票，有换乘的票，要加以区分，且换乘的两张票应该放在一起；已过期的票有超过使用日期的，还有可能是被退票或被改签的。

4、改签

首先该功能是面向普通用户的。

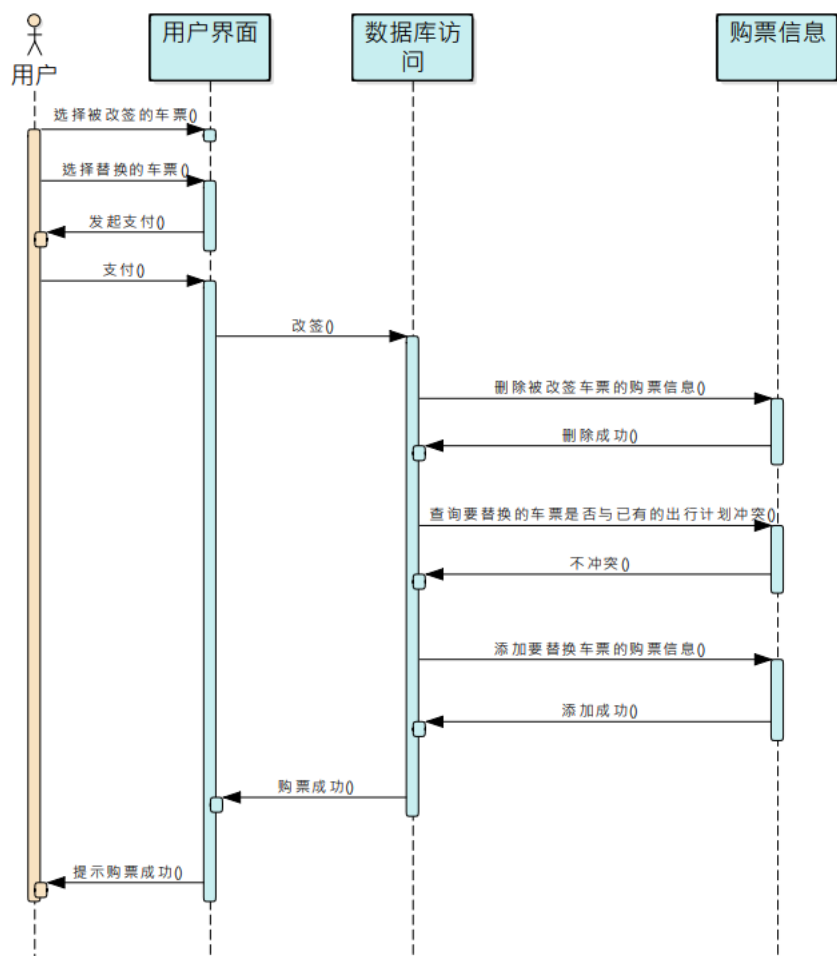
用户可以对已经购买的车票执行改签操作，但改签不能改变出发站和终点站只能改变出行的日期进行查询然后选择可选的出行方案。改签选择新的出行方案后仍然可以选择座位类型，但不允许添加乘车人。

一张票只限改签一次，即一张票在改签之后是不能再改签的。

改签一次只能改签一张车票。被改签的车票不应该从系统中消失，只是状态变成了不可用，改签后新得到的那张车票和被改签的车票还是有关联的，是改签和被改签的关系。

改签中系统要做三个额外的操作：

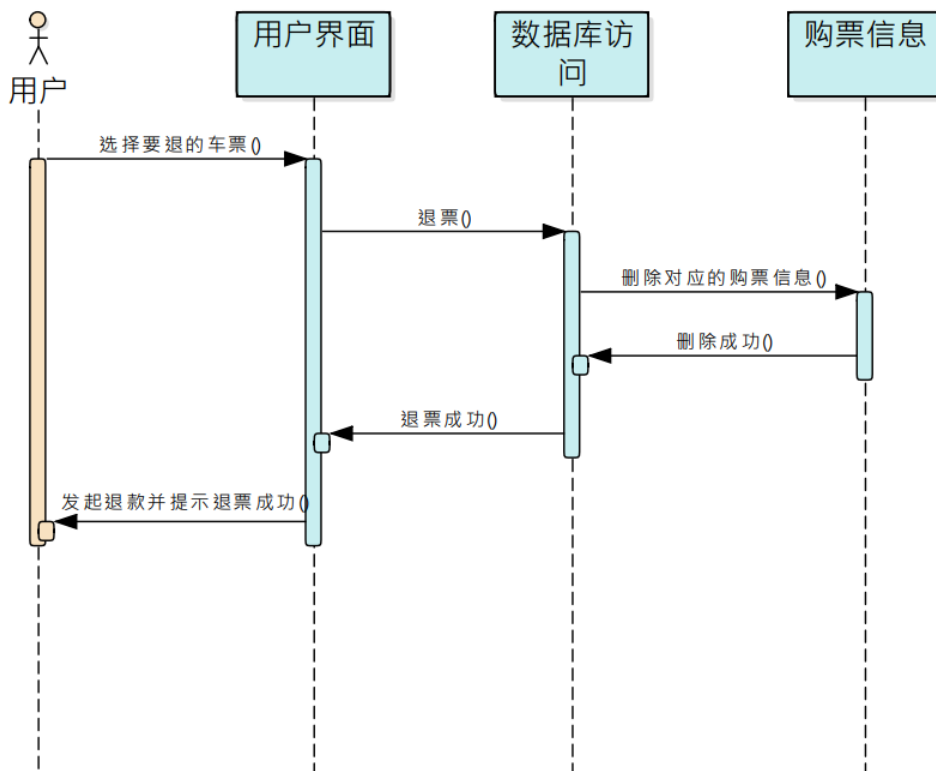
- 1、要查询乘车人是否在黑名单中，若乘车人在黑名单中，则禁止其改签，且原先购买的车票也被禁用
- 2、要查询乘车人是否与已有的出行计划冲突。
- 3、改签的车票要和原先的车票的票价比较，若新买的车票比原先车票的票价高，则用户要支付两张票的差价；若新买的车票比原先车票的票价低，则系统要发起退款的操作



5、退票

首先该功能是面向普通用户的。

用户可以选择一张还未过期的车票进行退票操作，对于退票，系统要发起退款，并将该票的状态设置为已退票。



6、黑名单管理

首先该功能是面向管理员的。

管理员可以对黑名单进行添加、移除、查询的操作。

一旦管理员在黑名单中添加某一个用户，则同步禁用该用户所有的未出行的车票，且禁止其购票。当管理员移除某一个用户时，则同步恢复用户所有的未出行车票，允许其购票。

黑名单中添加和移除用户的操作不仅可以是单个用户的添加和移除，也可以是导入 excel 表进行添加或移除。

7、车次管理

该功能是面向管理员的。

管理员可以添加、移除、查询车次。车次的添加和移除都要给出相应的时间区间，是从哪一天到哪一天添加或移除该车次。

同样车次的添加和移除不仅可以是单个车次的添加或移除，也可以是 excel 表导入进行添加或移除。

8、发布通知

该功能是面向管理员的。

管理员可以发布通知，所有登录该系统的用户都会收到通知。

第二章、系统设计

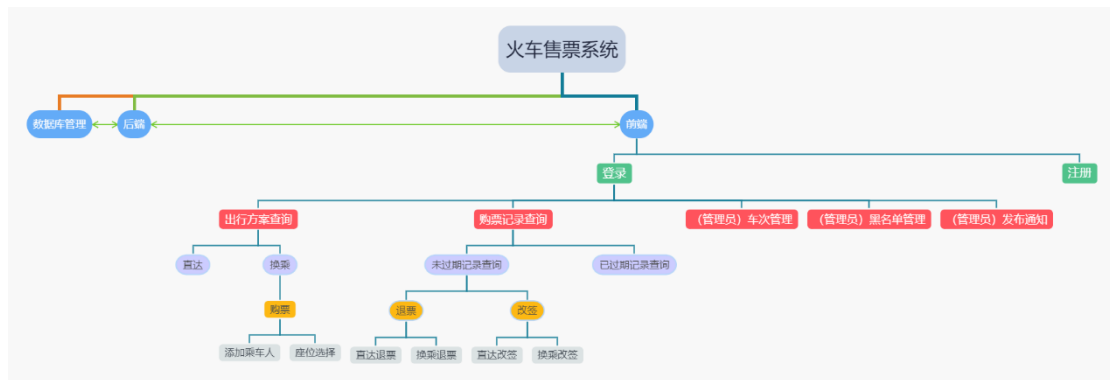
一、系统开发平台及框架

采用 C-S(client--server)架构,前端使用 vue 框架,后端使用 springboot 框架,前端中界面的部分部件使用 element-ui

开发工具: idea

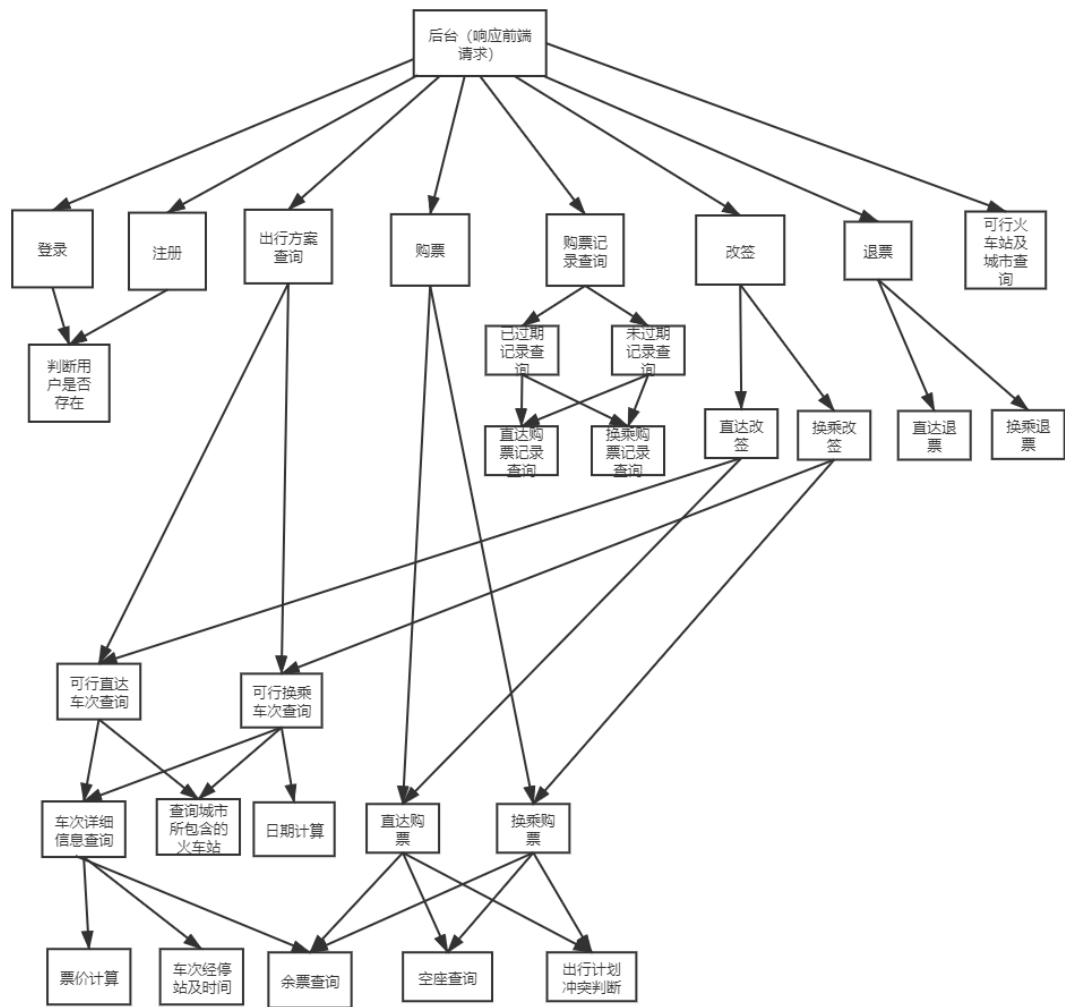
数据库: MySQL

二、系统整体架构及模块划分



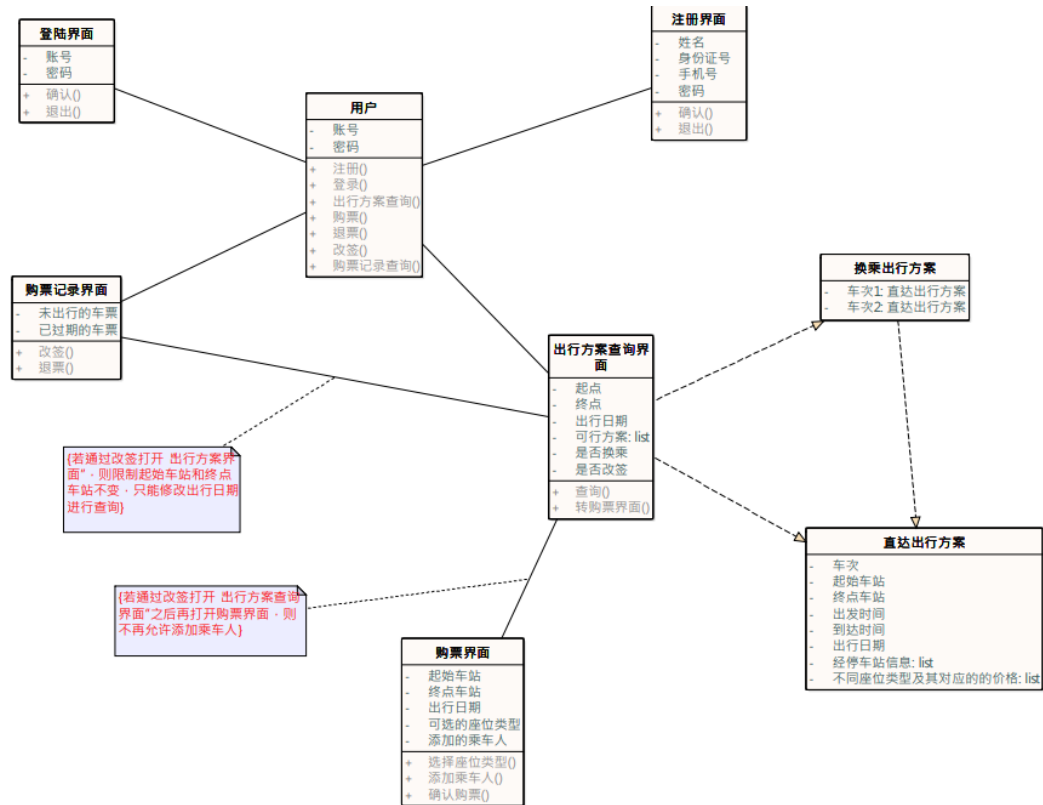
前端按功能划分为如上图的模块

后端主要是响应前端的请求,在数据库中查询,并返回给前端需要的数据,后端按前端请求的类型划分为如下的模块:



三、界面设计

界面类图：



1、登录界面：



2、主界面



1、方案查询界面：



4、查询结果显示界面： 直达查询结果显示：



换乘查询结果显示：

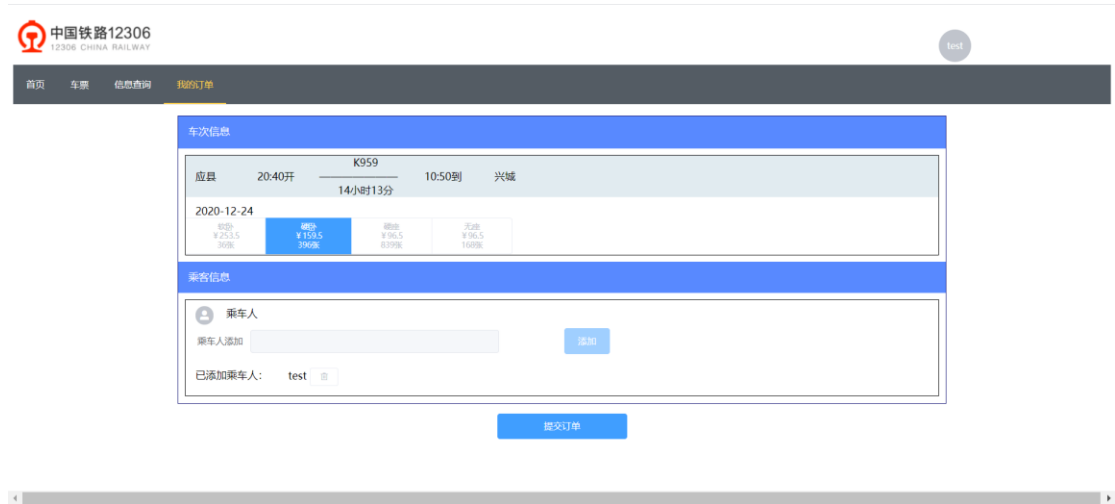
5、购票界面：
直达购票界面：

换乘购票界面：

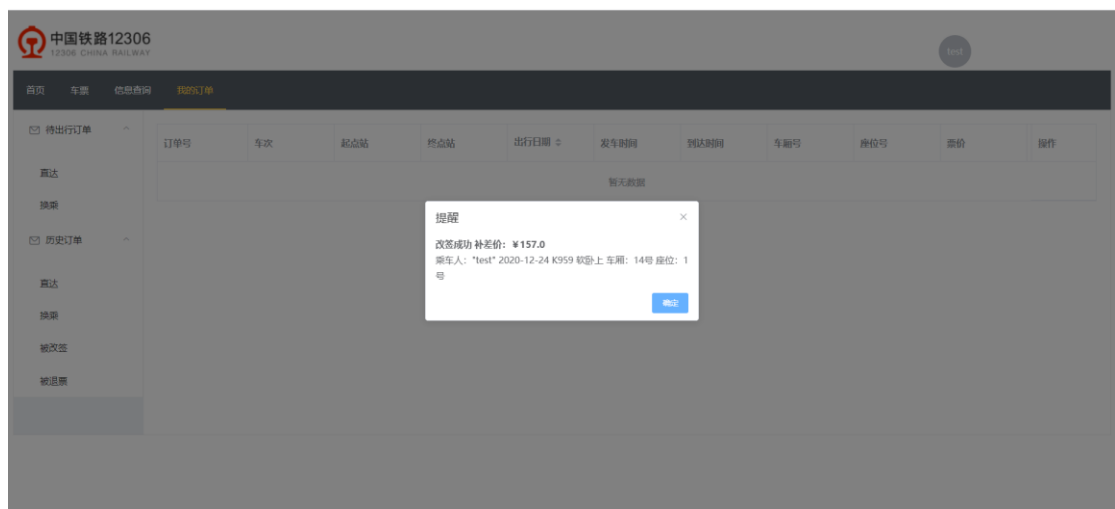
直达购票成功提示:



可以选座，但不能增加乘车人



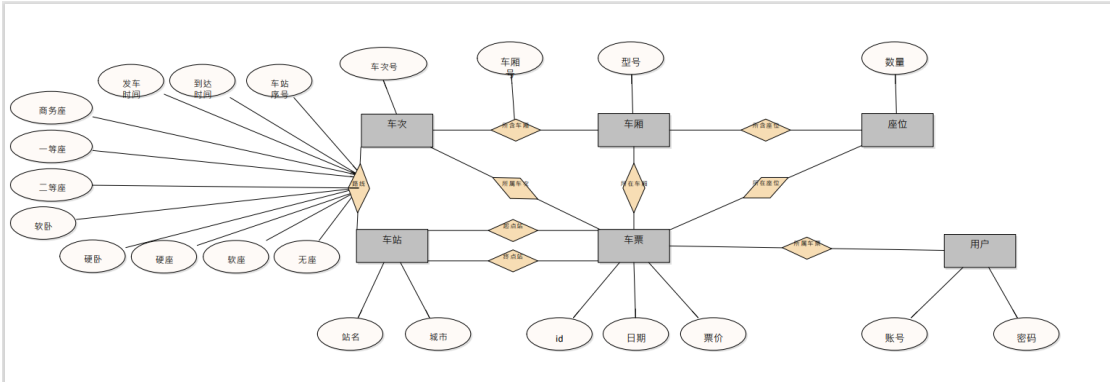
改签成功：



第三章、数据库设计

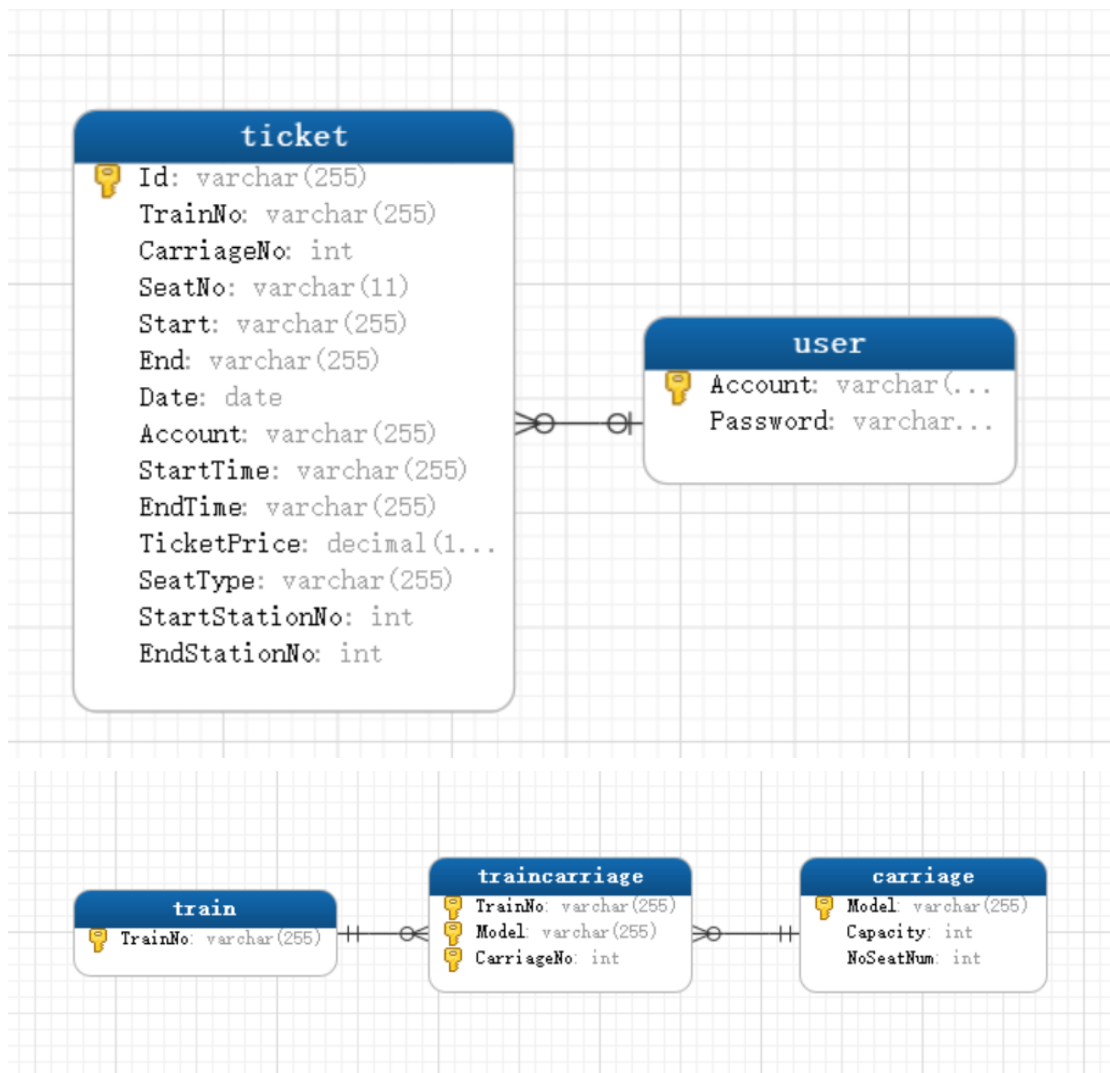
一、数据库逻辑设计

1、ER 图



2、数据库表设计

trainstation	oldticket	station
<ul style="list-style-type: none">TrainNo: varchar(255)Name: varchar(255)StationNo: intArrival: varchar(255)Departure: varchar(...)Business: doubleFirst: doubleSecond: doubleSoftSleeperUp: doubleSoftSleeperDown: do...HardSleeperUp: doubleHardSleeperMid: doubleHardSleeperDown: do...SoftSeat: doubleHardSeat: doubleNoSeat: doubleDuration: int	<ul style="list-style-type: none">Id: varchar(255)TrainNo: varchar(255)CarriageNo: intSeatNo: varchar(255)Start: varchar(255)End: varchar(255)Date: dateAccount: varchar(255)StartTime: varchar(255)EndTime: varchar(255)TicketPrice: decimal(1...SeatType: varchar(255)StartStationNo: intEndStationNo: intstatus: varchar(255)	<ul style="list-style-type: none">Name: varchar(255)Province: varchar...City: varchar(255)District: varchar...



详细信息:

Trainstation 表: 包含火车经停火车站的信息

名	类型	长度	小数点	不是 null	
▶ TrainNo	varchar	255	0	<input checked="" type="checkbox"/>	🔑 1
Name	varchar	255	0	<input checked="" type="checkbox"/>	🔑 2
StationNo	int	11	0	<input checked="" type="checkbox"/>	🔑 3
Arrival	varchar	255	0	<input type="checkbox"/>	
Departure	varchar	255	0	<input type="checkbox"/>	
Business	double	0	0	<input type="checkbox"/>	
First	double	0	0	<input type="checkbox"/>	
Second	double	0	0	<input type="checkbox"/>	
SoftSleeperUp	double	0	0	<input type="checkbox"/>	
SoftSleeperDown	double	0	0	<input type="checkbox"/>	
HardSleeperUp	double	0	0	<input type="checkbox"/>	
HardSleeperMid	double	0	0	<input checked="" type="checkbox"/>	
HardSleeperDown	double	0	0	<input type="checkbox"/>	
SoftSeat	double	0	0	<input checked="" type="checkbox"/>	
HardSeat	double	0	0	<input type="checkbox"/>	
NoSeat	double	0	0	<input type="checkbox"/>	
Duration	int	11	0	<input type="checkbox"/>	

Oldticket 表：包含已过期火车票的信息

名	类型	长度	小数点	不是 null	
▶ Id	varchar	255	0	<input type="checkbox"/>	
TrainNo	varchar	255	0	<input type="checkbox"/>	
CarriageNo	int	11	0	<input type="checkbox"/>	
SeatNo	varchar	255	0	<input type="checkbox"/>	
Start	varchar	255	0	<input type="checkbox"/>	
End	varchar	255	0	<input type="checkbox"/>	
Date	date	0	0	<input type="checkbox"/>	
Account	varchar	255	0	<input type="checkbox"/>	
StartTime	varchar	255	0	<input type="checkbox"/>	
EndTime	varchar	255	0	<input type="checkbox"/>	
TicketPrice	decimal	10	2	<input type="checkbox"/>	
SeatType	varchar	255	0	<input type="checkbox"/>	
StartStationNo	int	11	0	<input type="checkbox"/>	
EndStationNo	int	11	0	<input type="checkbox"/>	
status	varchar	255	0	<input type="checkbox"/>	

Station 表：包含所有火车站的站名及所在的省市

名	类型	长度	小数点	不是 null	
▶ Name	varchar	255	0	<input checked="" type="checkbox"/>	🔑 1
Province	varchar	255	0	<input type="checkbox"/>	
City	varchar	255	0	<input type="checkbox"/>	
District	varchar	255	0	<input type="checkbox"/>	


Ticket 表：包含所有的未过期的车票

名	类型	长度	小数点	不是 null	
▶ Id	varchar	255	0	<input checked="" type="checkbox"/>	🔑 1
TrainNo	varchar	255	0	<input type="checkbox"/>	
CarriageNo	int	11	0	<input type="checkbox"/>	
SeatNo	varchar	11	0	<input type="checkbox"/>	
Start	varchar	255	0	<input type="checkbox"/>	
End	varchar	255	0	<input type="checkbox"/>	
Date	date	0	0	<input checked="" type="checkbox"/>	
Account	varchar	255	0	<input type="checkbox"/>	
StartTime	varchar	255	0	<input type="checkbox"/>	
EndTime	varchar	255	0	<input type="checkbox"/>	
TicketPrice	decimal	10	2	<input type="checkbox"/>	
SeatType	varchar	255	0	<input type="checkbox"/>	
StartStationNo	int	11	0	<input type="checkbox"/>	
EndStationNo	int	11	0	<input type="checkbox"/>	


User 表：包含用户信息

名	类型	长度	小数点	不是 null	
▶ Account	varchar	255	0	<input checked="" type="checkbox"/>	🔑 1
Password	varchar	255	0	<input type="checkbox"/>	

Train 表：包含所有的车次

名	类型	长度	小数点	不是 null	
▶ TrainNo	varchar	255	0	<input checked="" type="checkbox"/>	 1

Traincarriage 表：包含所有车次车厢的信息

名	类型	长度	小数点	不是 null	
▶ TrainNo	varchar	255	0	<input checked="" type="checkbox"/>	 2
Model	varchar	255	0	<input checked="" type="checkbox"/>	 1
CarriageNo	int	11	0	<input checked="" type="checkbox"/>	 3

Carriage 表：各种型号车厢的信息

名	类型	长度	小数点	不是 null	
▶ Model	varchar	255	0	<input checked="" type="checkbox"/>	 1
Capacity	int	255	0	<input type="checkbox"/>	
NoSeatNum	int	11	0	<input type="checkbox"/>	

二、数据库物理设计

1、索引

在项目中创建如下索引，在保证整体的数据库效率基础上，能够提升查询数据的速度。

表名	索引
TrainStation	(TrainNo,StationNo)
oldTicket	Id
station	Name
ticket	Id
user	Account
train	TrainNo
trainCarriage	(TrainNo,Model)
carriage	Model

2、视图

创建多个视图，能够隐藏数据库的部分信息，但同时能够更高效的提供有用信息，使查询效率更高。但是应该考虑到视图创建的耗费，所以主要建立以下几个视图：

视图名称	视图内容
Capacity	每个车次各种座位类型及其数量，包括无座（计算余票时使用）
CityStation	每个城市所包含的火车站（用于在用户选择某个城市后，快速确认该城市包含的火车站）
Duration	每个车次从出发站到中间经过的火车站所需要的时间（用于计算可行方案时长）

第四章、系统实现

系统实现一方面考虑如何实现系统功能另一方面是要保证系统的性能

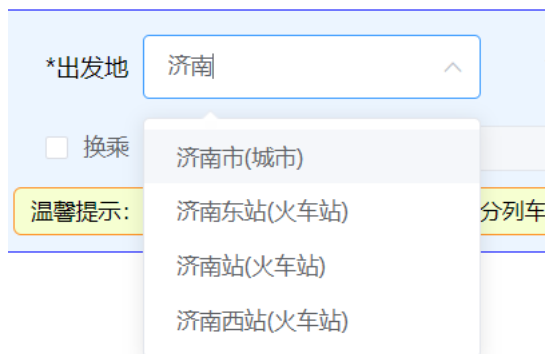
一、前端：

前端主要是和用户交互并将用户所需的数据展示给用户，前端要保证页面切换的流畅、不卡顿

- 1、可行方案查询需要用户提供有关起始站信息的输入，在这个地方需要查询数据库，将用户可选的火车站和城市展示出来以供用户选择。

基于以上场景，一个比较简单的方案是用下拉列表将后端返回的可选的火车站和城市全部展示出来，这是可以的，但因为数据量有几千条，前端渲染的压力比较大，会导致前端页面明显的卡顿，这样的用户体验是很不好的。在这个地方做出的优化是，在用户点击输入框但还没有输入的时候，下拉列表只展示数据的前几条，在用户输入后，则在已有的数据中查询，将包含用户输入的关键字的数据中的前十条展示出来，这样可以保证前端页面的流畅，而且也不会破坏用户体验。

效果如下图：



关键代码：

```
filterMethod(query){//query 是输入的关键字
```

```
    if(query == '')
```

```
        this.options = this.sumoptions.slice(0,10)
```

```
    else{
```

```
        let result = []
```

```
//存储符合条件的下拉选项
```

```
        this.sumoptions.forEach(val=>{
```

```

        if((val.value+'('+val.label+')').includes(query))
result.push(val)
    })

    this.options = result.slice(0,10)//只取前 10 个
}
}

```

- 2、前端在展示可行方案时使用 element-ui 的表格组件，这也是一个需要注意的地方，如果使用大量 div 嵌套下来可以达到和表格相同的效果，但是会导致页面的卡顿，最好使用表格，与上一个问题类似，可行方案查询的结果有大量的数据，而且换乘方案所产生的数据量更是惊人，直接在前端展示出来不仅会导致页面卡顿，而且可能电脑直接死机，所以最好是只展示部分数据，即分页展示，更进一步，由于前端处理能力有限，把这么多数据放在前端，即使不展示只是放在那里出来也会导致页面卡顿，所以这里最好是分页请求，前端请求哪一页的数据，后端就返回哪一页的数据，这样可以保证页面流畅。
- 3、对于改签，如果简单来考虑就是退了一张票，然后又买了一张票，退票给用户一个按钮即可，对于再买一张票，还要进行可行方案的查询和选座、买票等，这部分其实和单纯的可行方案查询、购票其实是重复的，那这里自然最好是复用那部分代码（主要是前端界面的代码），所以用户改签可以直接跳转到之前已经写好的可行方案查询和购票界面，又由于改签是不允许改变起始站且不能添加乘车人的，所以要添加参数的限制，保证系统不会异常

二、后端：

后端就是响应前端的请求，然后在数据库中做对应的查询，并加以处理，按约定的格式将数据返回给前端，所以在大致的框架搭建好后，主要的问题就集中在 sql 的编写和数据的处理

- 1、用户提供的地点包含哪些火车站

因为本系统是允许用户在起点和终点输入城市的，在进行可行方案查询时，要把城市转为火车站，那就要查询该城市包含哪些火车站，这是进行可行方案查询的一项基本工作

代码：参数 para 就是前端返回的地址，该方法对它进行解析，并返回该地址包含的火车站，返回的结果是一个列表，列表中包含一个或多个火车站

```

public List<String> getStation(String para){
    List<String> ans= new ArrayList<String>();
    String[] p=para.split("\\|");
    if(p[2].equals("Name"))
ans.add(p[0].substring(0,p[0].length()-1));
    else if(p[2].equals("District")){
        String sql="SELECT Name from station where

```



```

District='"+p[0]+"';
        for(Map<String,Object> map :
jdbcTemplate.queryForList(sql)){
            String name=(String) map.get("Name");
            ans.add(name.substring(0,name.length()-1));
        }
    }else if(p[2].equals("City")){
        String sql="(SELECT Name from station where Province=City
AND City='"+p[0]+"') UNION (SELECT Name from station where
City='"+p[0]+" AND District LIKE '%区')";
        for(Map<String,Object> map :
jdbcTemplate.queryForList(sql)){
            String name=(String) map.get("Name");
            ans.add(name.substring(0,name.length()-1));
        }
    }
    return ans;
}

```

2、直达方案查询

对于直达，只涉及一趟车次，相对简单，对一趟车次，只要既包含起点火车站而且包含终点火车站（对用户提供城市作为起点和终点可以用上面的方法转为火车站），那这趟车次就是可行解，但要注意，这趟车次要先经过起点站再经过终点站才是可行解，反之是不行的

代码：下面的方法就是提供起点站和终点站，然后查询可行的车次

```

public List<String> getTrainNo(String StartPoint,String
EndPoint){
    List<String> ans= new ArrayList<String>();
    String sql="SELECT DISTINCT TrainNo FROM trainstation WHERE
TrainNo in (select DISTINCT TrainNo FROM trainstation as t1 WHERE
t1.`Name`='"+StartPoint+"' AND t1.StationNo<(SELECT StationNo
FROM trainstation AS t2 WHERE t2.TrainNo=t1.TrainNo and
t2.Name='"+EndPoint+"' ))";
    for(Map<String,Object> map : jdbcTemplate.queryForList(sql)){
        String TrainNo=(String) map.get("TrainNo");
        ans.add(TrainNo);
    }
    return ans;
}

```

3、获得车次的信息

更准确地说，是提供该车次的起点站和终点站然后获得这一段的信息，包括这一段各种座位类型余票的数量、各种座位类型的票价、经停火车站、这一段火车运行的时间等，这些也都是作为可行方案数据的一部分要返回给前端的（关于余票数量的计算在下一条进一步讨论）

代码：

```
public Map<String,Object> getInfo(String StartStation,String
EndStation,String TrainNo,String date){
    Map<String,Object> ans=new HashMap<String,Object>();
    String sql0="WITH t1 AS ( SELECT * FROM trainstation WHERE
TrainNo = '"+TrainNo+"' AND NAME = '"+StartStation+"'),t2 AS
( SELECT * FROM trainstation WHERE TrainNo = '"+TrainNo+"' AND
NAME = '"+EndStation+"' );
    String sql1="SELECT t1.StationNo as
StartStationNo,t2.StationNo as EndStationNo,t1.Departure AS
Depart,t2.Arrival,t2.Business - t1.Business AS BusinessPrice,t2.
FIRST - t1. FIRST AS FirstPrice,t2. SECOND - t1. SECOND AS
SecondPrice,t2.SoftSleeperUp - t1.SoftSleeperUp AS
SoftSleeperUpPrice,t2.SoftSleeperDown - t1.SoftSleeperDown AS
SoftSleeperDownPrice,t2.HardSleeperUp - t1.HardSleeperUp AS
HardSleeperUpPrice,t2.HardSleeperMid - t1.HardSleeperMid AS
HardSleeperMidPrice,t2.HardSleeperDown - t1.HardSleeperDown AS
HardSleeperDownPrice,t2.SoftSeat - t1.SoftSeat AS
SoftSeatPrice,t2.HardSeat - t1.HardSeat AS
HardSeatPrice,t2.NoSeat - t1.NoSeat AS NoSeatPrice FROM t2,t1";
    ans=jdbcTemplate.queryForMap(sql0+sql1);
    int
duration_time=getDuration_(TrainNo,ans.get("StartStationNo")+"" ,a
ns.get("EndStationNo")+"" );
    String duration=formatTime(duration_time);
    ans.put("duration",duration);
    ans.put("duration_time",duration_time);
    String sql3="SELECT
t3.TrainNo,t3.Name,t3.StationNo,t3.Arrival,t3.Departure FROM
trainstation AS t3,t1,t2 WHERE t3.TrainNo='"+TrainNo+"' AND
t3.StationNo>=t1.StationNo AND t3.StationNo<=t2.StationNo ORDER
BY t3.StationNo ASC";
    ans.put("detail",jdbcTemplate.queryForList(sql0+sql3));
    ans.put("StartStation",StartStation);
    ans.put("EndStation",EndStation);
    ans.put("TrainNo",TrainNo);
    ans.put("date",date);

    Map<String,Integer>
```

```

tp=getNumInfo(TrainNo,date,ans.get("StartStationNo")+"" ,ans.get("
EndStationNo")+"" );
ans.putAll(tp);

return ans;
}

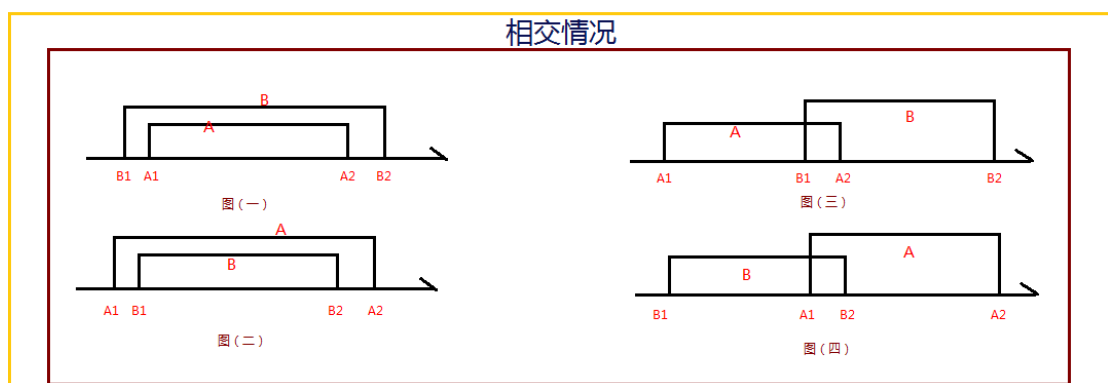
```

4、对某一车次，给定起始站和终点站，余票信息的计算

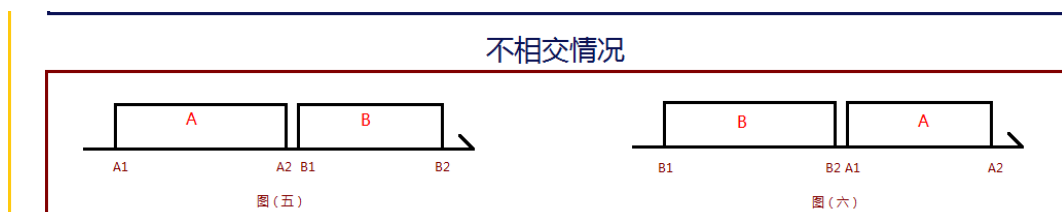
在本系统的数据库中，所有用户购买的车票的信息都包含在 ticket 表中，所以在计算出该车次各个座位所允许的乘客数量后，只要在 ticket 表中做查询即可得到余票的数量。

假设一趟车次经过 A,B,C,D,E 四个火车站，现在要查询该车次从 B 到 D 的余票的数量，首先要明并不一定只有其它用户买了从 B 到 D 的票才会导致这次查询的余票数量减一，其它用户购买从 A 到 C，从 A 到 E，从 C 到 E 的票都会导致这次查询的余票数量减少，那这个问题可以简单地抽象成区间重叠的问题，我们做这样的规定，我要查询一趟车次从 A1 到 A2 的余票数量，已知一个用户购买了该车次从 B1 到 B2 的票，那问题就转换成 (A1,A2) 和 (B1,B2) 这两个区间是否有重叠的部分，如果有重叠，那就会导致这次查询的余票数量减一，反之不会

区间重叠有如下四种情况，我们可以判断是否满足这四种情况的某一种来判断是否重叠，但这其实是比较复杂的



更好的办法是判断不相交的情况，不相交的情况只有两种，而且这两种情况都很好判断，一个是 $A2 < B1$ ，一个是 $B2 < A1$ ，比判断相交要简单的多，而且只要把不相交的情况取反就是相交



要注意的是，只需和 ticket 表中与要查询车次的日期车次相同的购票记录比较即可，而且因为一趟车次上每个类型的座位都是独立的，所以每种类型的座位都要求余票的信息

代码：

```

public Map<String,Integer> getTicketNumInfo(String
TrainNo,String date,String StartStationNo,String EndStationNo) {
    Map<String,Integer> ans=new HashMap<String,Integer>();
    String sql="SELECT SeatType,COUNT(*) AS num from ticket
WHERE TrainNo='"+TrainNo+"' AND Date='"+date+"' AND
(("+EndStationNo+"-1)>=StartStationNo AND
"+StartStationNo+"<=(EndStationNo-1)) GROUP BY SeatType";
    //      System.out.println(sql);
    for(Map<String,Object> map :
jdbcTemplate.queryForList(sql)){
        String type=(String) map.get("SeatType");
        int num=Integer.valueOf(map.get("num")+"" );
        ans.put(type+"Num",num);
    }
    return ans;
}

```

5、换乘方案查询

对于换乘，因为涉及两趟车次，所以会比较复杂，这个地方，我们先用最朴素的思想来解决这个问题，之后再逐步地优化。

对于换乘，和直达一样，同样是给定起始站和终点站（给定城市也可以转成火车站）然后找到可行的方案，但是换乘会有一个中转站，问题的关键其实是这个中转站的确定，如果确定了可行的中转站，那问题就和直达是差不多的，只是转化为求两个直达的车次。

那对于求中转的车站，比较简单的一个想法就是对所有经过起点站的车次找出它之后要经过的车站，对所有经过终点站的车次找出它经过该站之前走过的车站，两个车站取交集就是可行的中转站，之后就和求可行的直达方案是一样的。

上述是一个解决换乘问题的可行的方案，之后再讨论它的优化，在此之前要讨论换乘的一个问题，就是换乘可能会越走越远，比如要从济南到沈阳，但查询出的中转站的是南京，那就要从济南先去南京，再从南京到沈阳，那这显然是不合理的，在这里解决这个问题的方案是使用每个车站的经纬度，以起点站和终点站为“矩形”的对角顶点，可以画出一个“矩形”区域，只要中转站的经纬度在这个矩形区域内，那就一定不是越走越远。

在解决了越走越远的问题后，我们讨论查询换乘的可行方案的优化：

优化一、上述的方案是确定了中转站，然后用求直达的方法求换乘可行的方案，但这样就浪费了一部分有用的信息：车次。对所有经过起点站的车次找出它之后要经过的车站，在这步操作中，我们可以保留车次的信息，即查询出来的结果应该是两列：车次、火车站，对终点站的操作同样保留车次的信息，那就只要以车站为主键对两个中间表做连接就可以直接确定所有的换乘的方案：第一趟车次、第二趟车次、中转站，之后只要去查这些车次的余票信息、票价、经停站等信息即可。

优化二、如果去实际操作可以发现，对一些省会城市，中转的方案可能有好

几万，如果是北京上海这样的城市那中转方案数会更惊人，而且上万的方案展示给用户其实意义也是不大的，所以我们要尽可能地减少中转方案的数量，将更合理的方案保留下来。

首先要注意，其实这样的换乘方案是囊括了直达的方案，就是原本从 A 到 B 就是直达的，那现在对从 A 到 B 的换乘方案是会包括之前所有可行的直达的方案，这部分是应该去掉的，即在中间表连接时限制，两个车次不一样。

之后的一个很简单的想法其实就是按某种规则对查询出来的方案进行排序，然后选排序之后的前多少条方案展示给用户，这个地方问题不在于选取哪种规则，而是方案数太多，这样排序的代价太大，本身两个中间表连接就需要大量的时间，再这样排序，那前端需要很久才能得到需要的数据，响应时间要几十秒甚至一分钟多。对连接之后的数据做操作需要较大的代价，那最好就是对连接之前的数据就做处理，减少需要连接的数据，这样不论是连接还是之后的排序的压力都不会很大。

对连接前的数据，比如对找出的经过起点站的车次之后要经过的车站，我们首先假定一个合理的换乘是换乘站到起点站和终点站的距离是差不多的，那么我们就可以去掉那些到起点站的距离超过 0.5 倍起点站到中转站距离的车站，对经过终点站之前的车站做同样的处理，当然也不一定是 0.5，可以经过测试，找一个合理的值。这样可以减少两个中间表的数据量。

除了以上的优化，换乘还需要注意的一个问题是日期的问题，即两趟车次的日期可能是不一样的，这是需要额外计算的