火车票订票系统开发文档

毛潇涵 文舸扬 徐清瑶 罗雅婷

2018 年 6 月 15 日

1 开发分工

- 前端网页交互负责：文舸扬
- 前后端信息传递：毛潇涵
- 后端接口：罗雅婷
- 后端数据库存储：徐清瑶

2 前端网页交互

2.1 简介

前端使用网页进行构架，大体使用 bootstrap 框架，兼用自己改写的 JavaScript 和 css，实现网页的基本功能

2.2 网页功能介绍

1. 导航栏
   导航栏是在每个网页上都会出现的东西，通过左上角的按钮能返回首页，右上角则是购票以及用户界面的跳转，对于未登录的情况，会显示登录与注册界面的链接。

2. 界面设计

- 登录/注册
  登录和注册界面可以互相自由切换。注册成功后返回用户 ID 并自动登录
- 用户界面
  在用户界面可以查询个人的基本信息以及个人某日的订票情况，如果用户为管理员，可以从该界面通向管理员页面
- 管理员页面
  在其中可以进行对车次的基本操作和管理员的设置
- 购票页面
  在其中可以通过查询从 A 地到 B 地的车票来进行车票的购买，订票成功后返回成功信息
- 个人订票情况页面
  对于个人购买的车次进行查看以及退票
- 错误界面
  当操作非法时显示错误信息
- 服务器错误页面
  当服务器出问题时显示

3 前后端信息传递

3.1 简介

- 服务器使用 php 语言开发，使用 CodeIgniter 框架，通过 socket 与后端数据库进行通信和信息交换
- 用户提交信息后，会先访问 controller，controller 使用 model 进行运算以及查询数据库（如果有必要的话），然后加载相应的 view，最后将页面传递给用户显示

3.2 controller 介绍

1. Welcome < 网页主界面 >
   - index()
     直接加载 welcome 界面

2. Login < 登陆界面 >

   - index()
     加载 login 视图
   - check()
     接收从 login 视图传来的信息，使用 User_model 判断密码是否正确。如果数据库未开启导致查询失败，会加载 RE 视图，否则根据用户 ID 和密码是否正确选择加载 AC 或 WA 视图。如果登录成功，会将 username 记录在 cookie 里，将用户 id 记录在 session 中。

3. Register < 注册 >

   - index()
     加载 register/register_page 视图
   - check()
     接收信息，使用 User_model 进行注册，如果注册成功将会加载 register/success 信息，将用户 ID 告知用户

4. Logout < 登出 >

   - index()
     销毁 cookie 和 session，同时跳转至首页

5. Profile < 用户信息 >
   该界面的每个操作前都会先检查是否登录，如果不是将会加载 WA 视图，下面不再赘述。

   - index()
     根据 session 中的用户 id，使用 User_model 查询用户信息，将用户信息传递给 user/profile 视图显示。
   - change()
     接收 user/profile 视图中传来的信息，先检查原密码是否正确，如果不正确将加载 WA 视图，否则使用 User_model 修改用户信息，加载 AC 视图

- query_order()
  查询该用户的订单，接收 user/profile 视图中传来的信息，使用 Ticket_model 查询该用户的订票情况，并将查询结果传递给 query/order 视图显示

6. Admin < 管理员界面 >
   该界面的每个操作前都会先检查该用户是否为管理员，如果不是将会加载 WA 视图，下面不再赘述。

   - index()
     加载 user/admin 视图
   - set_admin
     使用 User_model 将指定 ID 的用户设置为管理员

7. Ticket < 车票相关 >

   - index()
     车票查询加载 ticket/query 视图
   - query()
     车票查询结果使用 Ticket_model 查询车票,将结果传递给 query/result 视图显示
   - book()
     订票该功能需要登录，使用 Ticket_model 来购买车票，根据是否成功选择加载 WA 或者 book/success 视图
   - book()
     退票该功能需要登录，使用 Ticket_model 来退订车票，根据是否成功选择加载 WA 或者 book/success 视图

8. Train < 车次相关 >
   该界面的每个操作前都会先检查该用户是否为管理员，如果不是将会加载 WA 视图，下面不再赘述。

   - index()
     没用，为空
   - add()
     添加车次，使用 Train_model 来添加车次

- modify()
  修改车次，使用 Train_model 来修改车次
- sale()
  发售车次，发售指定的一辆未发售车次
- erase()
  删除车次，删除指定的一辆未发售的车次
- query()
  根据 train_id 查询指定车次的信息，将信息传递给 query_train 视图显示

3.3 model 介绍

model 使用 socket 与后端服务器进行短程连接，如果连接失败将返回-1，下面不再赘述

1. User_model < 用户管理 >
   用户的密码使用 md5 进行加密，下面不再赘述
   - register(name,psword,email,phone)
     注册用户，返回用户 id
   - login(id,psword)
     登录 0 表示失败，1 表示成功
   - query(id)
     查询用户信息返回 name,email,phone,privilege
   - modify_profile(id,name,psword,email,phone)
     修改用户信息，0 表示失败，1 表示成功
   - clean()
     删库，鉴于该命令后果严重，我们不打算在管理员界面实装这项功能
   - exit()
     数据库关闭，鉴于该命令后果严重，我们不打算在管理员界面实装这项功能
2. Ticket_model <>

- query_ticket(loc1,loc2,date,catalog)
  查询 loc1 到 loc2 的日期为 date 的 catalog 类车返回 num,Ticket*num
- query_transfer(loc1,loc2,date,catalog)
  查询 loc1 到 loc2 的日期为 date 的可中转一次的 catalog 类车返回 2,Ticket*2
- buy(id,train_id,loc1,loc2,date,ticket_kind,ticket_num)
  编号为 id 的用户订购 train_id 这列车中从 loc1 到 loc2 的 date 日的 ticket_num 张 ticket_kind 类票，返回 0/1 表示失败/成功
- refund(id,train_id,loc1,loc2,date,ticket_kind,ticket_num)
  编号为 id 的用户退订 train_id 这列车中从 loc1 到 loc2 的 date 日的 ticket_num 张 ticket_kind 类票，返回 0/1 表示失败/成功
- query_order(id,date,catalog)
  查询编号为 id 的用户订购的 date 日的 catlog 类车，返回 num,Ticket*num

3. Train_model < 车票管理 >

   - add_train(train_id,name,catalog,num_station,num_price,Price,Station)
     添加 train_id 这列车，名字为 name，类别为 catalog，有 num_station 个站，num_price 种票，票价信息为 Price，站点信息为 Station
   - modify_train(train_id,name,catalog,num_station,num_price,Price,Station)
     修改车次信息，同上
   - query_train(train_id)
     查询 train_id 这辆车的信息返回
   - sale_train(train_id)
     发售 train_id 这辆车
   - delete_train(train_id)
     删除 train_id 这辆车

3.4 数据库通信介绍

- 数据库使用 C++ 语言开发。使用 socket 与服务器通信
- 启动时会使用 socket 监听本机的 7000 端口，如果有请求就会建立连接。将请求及信息读入到 buffer 里，使用时从 buffer 里读取并解析信息，如果 buffer 空了则会再去 socket 中读取后续信息

- 请求完成后会将信息传输回服务器，同时断开连接，等待下一个请求

4 后端接口模块划分和各模块的具体实现

4.1 几个主要的 struct 说明

- USER: < 用户类的信息 >
  包括字符串类型的用户名、密码、邮箱、手机号和字符类型的用户权限。

- user_order_key: < 用户所订购的票的索引 >
  包括整型的用户 id、字符串类型的日期、车号、loc1、loc2 和字符类型的车类。

- tk_order: < 用户所订购的票的具体信息 >
  包括当前票的索引、字符串类型的出发时间、到达时间，票类名称、double 类型的各票类的价格、int 类型的票的种类数和 int 类型的每类票的订购数（初始值为 0）。

- tk_key: < 车票信息的索引 >
  包括字符串类型的 loc1、loc2、车号和字符类型的车类。
  注: 在重载的 < 运算符中, 先比较 loc1, 然后比较车类, 最后比较 loc2 的目的是为了方便查中转票的处理中的迭代器的操作。

- tk: < 车票信息 >
  包括当前车票的索引、bool 类型的 flag 记录车票是顺序的车还是反序的车（1 代表顺序表示真实存在的车票，0 代表反序表示为了方便查询添加的伪车票）、字符串类型的出发时间和到达时间。

- STATION: < 车站信息 >
  包括字符串类型的车站名，到达时间，出发时间，停靠时间和 double 类型的与出发站的票价差。

- train_id_key: < 车次索引 >
  包括字符串类型的车号。

- TRAIN: < 车次信息 >
  包括字符串类型的车号、车名、票类名称、字符类型的车类，int 类型

的车站数和票的种类数，int 类型的剩余票数 tk_remain[i][j][k]（其表示始发站发车日期为 2018-06-i 的第 j 个车站到第 j+1 个车站的第 k 类票种的余票数）、STATION 类的车站和 bool 类的 isSale(记录是否公开，1 表示已公开不可进行删除或修改，0 表示未公开)

- tk_query: < 查票操作中用来记录需要输出的信息 >
  包括 tk_key 类的索引信息，字符串类的出发时间，到达时间，票类名称和 int 类型的票的种类数、剩余票数、dt1（记录从始发站到达 loc1 所跨的天数）、dt2（记录从始发站到 loc2 所跨的天数）

4.2 定义的 B+ 树名称

- 车次：train
- 车票：ticket
- 用户及其订购的票：User

4.3 用户模块

主要依靠 userlist 类实现用户注册，用户登陆，查询用户信息，修改用户信息，修改用户权限五个操作。

4.4 车次模块

- 新建车次：
  输入车次的信息 $\longrightarrow$ 处理字符串类的票价，以 double 类型存入车次信息中 $\longrightarrow$ 初始化剩余票数为 2000、初始化 isSale 为 0$\longrightarrow$ 把车次的信息插入到 train 这棵 B+ 树
- 公开车次：
  判断是否能够成功公开车次，如果可以成功公开车次，把车次相对应的相邻两站的票存进 ticket 这棵 B+ 树
- 查询车次：
  判断需要查询的车次是否存在 $\longrightarrow$ 根据索引在 train 这棵 B+ 树中找到对应的数据 $\longrightarrow$ 输出数据

- 删除车次：
  判断是否可以删除，根据索引在 train 这棵 B+ 树 erase 掉对应数据

- 修改车次：
  判断是否可以修改，根据输入的更新信息和索引在 train 这棵 B+ 树中 modify 对应数据

4.5 车票模块

- 查询车票
  ⇒ 根据输入的信息生成一个车票的索引 K，其中 K 的车号为空值
  ⇒ 声明一个迭代器，通过 lower_bound 函数指向 ticket 这棵 B+ 树中第一个比 K 大的索引位置
  ⇒ 通过 while 循环和迭代器的 ++ 操作把每一个符合条件的票的信息存入 tk_query 的数组 T 中
  ⇒ 遇到第一个不符合条件的票即 break 退出循环
  ⇒ 输出所有符合条件的票的信息

- 带中转查询车票
  ⇒ 根据输入信息生成两个车票的索引 K1，K2（K1, K2 的 loc2 均为空值）
  注：求 A 到 B 地的中转车票，即假设中转站为 C，K1 为 A 到 C 的索引所以对应数据的 flag 为 1，K2 为 B 到 C 的索引所以对应数据的 flag 为 0
  ⇒ 声明两个迭代器 it1,it2，分别通过 lower_bound 函数指向第一个比 K1 大的索引位置和第一个比 K2 大的索引位置
  ⇒ 比较两者的 loc2 向下排查找到每一个相同的 loc2（因为 loc2 是升序排列，所以如果一个迭代器 1 的 loc2 小于另一个迭代器 2 的 loc2，即可 ++ 迭代器 1 至 >= 迭代器 2, 重复这样的调整直到二者相同）
  ⇒ 把迭代器对应的满足条件的待定车票存进 T1 和 T2 中
  ⇒ 两层 for 循环找到 T1 和 T2 中对同一中转站存在的最短的时间的车票信息，存进 T_1 和 T_2 中
  ⇒ 分析 T_1 和 T_2 找到不同中转站最短的时间所对应的车票信息存入 TK1 和 TK2
  ⇒ 输出答案

- 订购车票
  ⇒ 根据 train 返回的信息判断是否可以购票成功
  ⇒ 购票成功将购票信息存入 User 这棵 B+ 树（如果已经存在此条购票信息则更新）
  ⇒ 购票成功更新 train 中所对应的余票数信息

- 查询购票信息
  ⇒ 根据输入信息生成用户订购车票的索引 K
  ⇒ 声明一个迭代器，通过 lower_bound 函数指向第一个比 K 大的索引位置
  ⇒ 通过迭代器 ++ 将所有符合条件的数据存入 tk_order t 中，遇到第一个不符合条件的数据即跳出循环
  ⇒ 输出 t 中结果

- 退订车票
  ⇒ 根据 User 中返回的信息判断是否可以退订成功
  ⇒ 如果退订成功将 User 中的购票信息更新
  ⇒ 如果退订成功更新 train 中所对应的余票数信息

5 后端数据库存储–Userlist 类和 B+ 树

5.1 Userlist 类的数据成员

- 存储用户信息的文件 iofile
- 该文件名 filename
- 用户数量 usernum

5.2 Userlist 的主要函数

- Userlist(filename)
  以 filename 为文件名构造一个 Userlist
- int Register(char *name, char *password, char *email, char *phone)
  注册用户，成功返回 id，失败返回-1 要求 name 数组长度 >40，password email phone 长度 >20，下同

- int Login(int id, char *password)
  登录，成功返回 1，失败返回 0
- int Query_profile(int id, char *name, char *email, char *phone, char &privilege)
  privilege 为 char，查询用户信息，查询成功返回 1，否则返回 0
- int Modify_profile(int id, char *name, char *password, char *email, char *phone)
  修改用户信息，成功返回 1，否则返回 0
- int Modify_privilege(int id1, int id2, char privilege)
  修改用户权限，成功返回 1，否则返回 0
- int size()
  返回用户总数

5.3 B+ 树的两个内嵌类

- 索引结点 idxNode
  它的第一个成员 type 表示子结点类型。当 type 为 0 时下一层为索引结点，当 type=1 时下一层为数据结点。key 和 idx 是两个数组，key 存一组关键字，idx 存一组位置信息。len 是结点中有效位置信息的数量。
- 数据结点 dataNode
  第一个成员 len 记录数据记录个数。数组 record 存储一组索引关键字项，而 value 存储一组要保存的数据。同时，为了方便查找，需将数据结点以链表的形式串起来，因此在结点中还保存了一个记录下一个 dataNode 位置的成员 nex。

5.4 B+ 树的数据成员

- 保存数据和保存索引项的文件
- 表示文件名称的 char 类型的指针。

5.5 B+ 树的主要函数

- void init(const char *i, const char *d)
  第一第二个参数分别为存储索引和存储数据的文件的名称。若文件不存在，则创建一个以其为名的空文件。

- void insert(const T &x, const U &v)
  第一个参数为插入元素索引值，第二个参数为插入元素数据值。该函数插入索引值为 x 的数据值为 v 的元素。

- void erase(const T &x)
  删除索引 x 对应元素。该函数不检查 x 的有效性。

- U* find(const T &x)
  返回索引 x 对应元素的数据值。该函数不检查 x 的有效性。

- bool check(const T &x)
  检查索引 x 对应元素是否存在。存在返回 1，否则返回 0。

- void modify(const T &x const U &v)
  查找索引 x 对应元素，将数据值更改为 v.

- void clear()
  清空文件

5.6 B+ 树的迭代器

为了表征 iterator 所在文件的位置，保存了存储数据的文件 ifdata 及文件名 data，数据成员 dcur 是当前 iterator 所指向数据结点在文件中的位置，而 dpos 则是 iterator 所指数据在当前 dataNode 结点中的位置。因为 iterator 的查找也需要 dataNode 类型的元素，还定义了与 B+ 树中一样的 dataNode 类型。

- 支持的操作：
  ⇒++ 向后遍历
  ⇒Record 函数返回迭代器对应的索引值
  ⇒Value 函数返回迭代器对应的数据值

- iterator at(const T &x)
  寻找索引值为 x 的位置作为迭代器返回
- iterator lowerbound(const T &x)
  找到文件中索引值大于等于 x 的最小位置作为迭代器返回
- bool isValid(iterator &i)
  判断当前迭代器 i 是否有效，有效返回 1，否则返回 0
- iterator begin()
  寻找指向第一个位置的迭代器返回