“C++程序设计与训练”课程大作业

项目报告

项目名称：酒店预订与管理系统

姓名：王思程

学号：2017011534

班级：自 71

日期：2018.8.2

目录

1 系统功能设计 3
1.1 总体功能描述 3
1.2 功能流程描述 4
2 系统结构设计 5
3 系统详细设计 6
3.1 类结构设计 6
3.1.1Customer 类 6
3.1.2Hotel 类 6
3.1.3Manager 类 6
3.1.4Room 类 7
3.1.5Order 类 7
3.1.6 数据库的数据表设计 7
3.2 界面结构设计 8
3.3 关键设计思路 8
4 项目总结 9
4.1 遇到的问题和解决方法 9
4.1.1 数据库的数据类型 9
4.1.2 像数据库存东西的语法问题 9
4.1.3 数据库键相同存数据的问题 9
4.1.4 信号与槽的作用域与构造函数导致有空指针出现异常退出问题 10
4.1.5 表格里选对象的问题 10
4.1.6 放图片的问题 10
4.1.7 哈希表排序问题 11
4.1.8 顾客查看房间时排序的各种要求代码重用问题 11
4.2.1 遵从最小信息量原则 11
4.2.2 使用 QHash 作为容器 11
4.2.3 界面设计简洁实用 11
4.2.4 界面美化 11
4.3 心得体会 12
4.3.1 提高代码重用 12
4.3.2 前期设计好再写 12
4.3.3 完成度与进度 12

第1章 1 系统功能设计

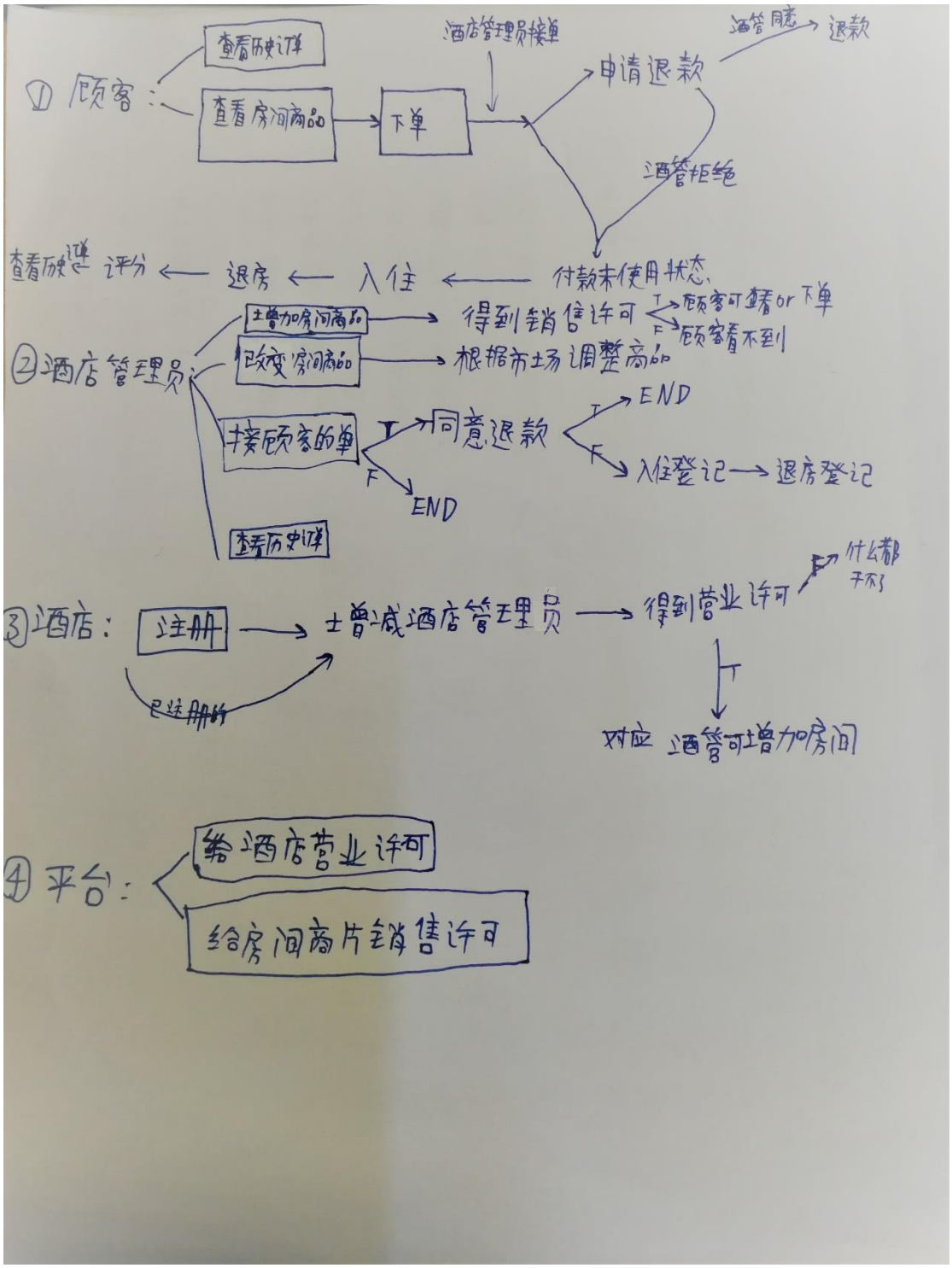
1.1 总体功能描述

描述软件系统总体功能

用户类型	功能名称	功能描述
顾客	注册账号	用手机号作为实名认证注册账号
	查看可预定房间	现实所有平台允许销售的种类的房间，可搜索某个酒店的房间，某个城市的房间，并可按酒店的得分或房间价格的升降序查看，且可以勾选仅查看有折扣的房间
	查看房间详细信息	在可预订房间的表中看到感兴趣的房间后可点击查看详细信息，看对应酒店和该房间种类的所有信息
	下单预定房间	可选择入住日期和天数去下单
	取消订单	在酒店接单前可直接取消订单
	申请退款	预定且被接单后，且在入住前，可向酒店提交退款申请
	评价待评价订单	正常住宿并退房后可对订单打分
	查看历史订单	看自己使用过的订单的各种信息
	修改个人信息	改密码，换手机后可改手机号
酒店管理员	更改已推出的房间商品的信息	改价格，折扣，总数量
	增加新的房间商品种类	增加一种房间，在平台给销售许可后顾客可以看到
	处理正在使用的订单	接单，拒绝接单，同意退款，拒绝退款，入住登记，退房登记
	查看历史订单	查看历史订单情况，便于对各种房间销售打分情况有更好的了解去做调整
酒店	看到酒店评分	酒店的董事会对自己的分数有了解
	增加减少管理员	招聘或开除管理员时使用
	更改酒店信息	改密码，地址，电话，图片
平台	审核酒店	酒店得到平台运营许可后，酒店管理员才能增加房间种类
	审核房间商品	销售许可后顾客才能看到

1.2 功能流程描述

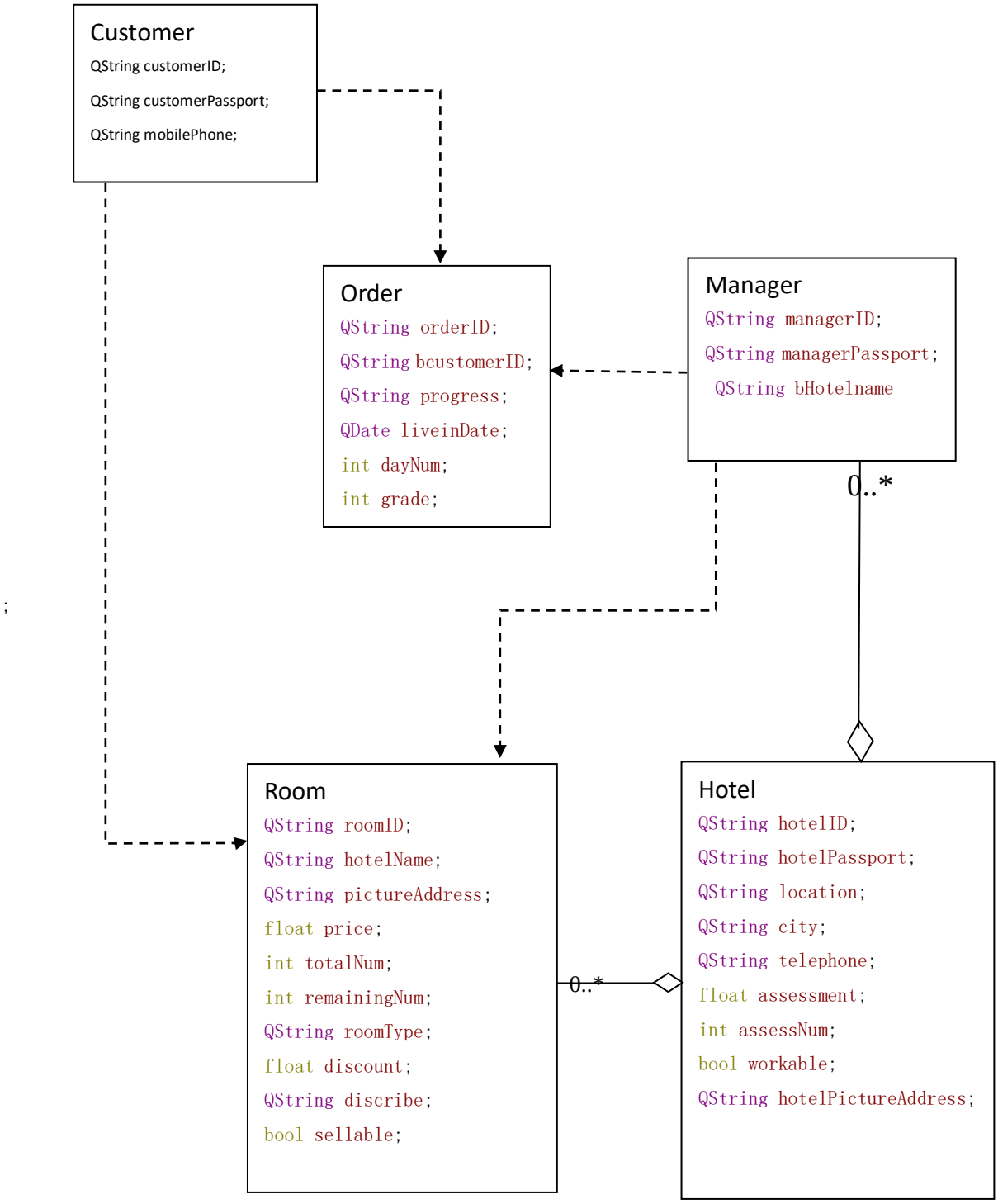
按不同用户描述对应的功能流程，可以用流程图配合文字来说明。



第2章 2 系统结构设计

这是进行复杂软件开发的第二步，即概要设计。此部分需要说清楚整个软件系统包含哪些模块（或功能部件），模块之间的关系和是怎样的；包含哪些主要的类，类之间的关系是怎样的。

此处仅列出类间关系和静态数据成员。每个类的构造函数，每个数据对应的 get 和 set 函数此处不再赘述。一切行为最终落到与静态成员有关时都通过 get 和 set 解决



第3章 3 系统详细设计

3.1 类结构设计

类的设计思路是找到每个对象属于自己的固有特性，得到一组静态数据成员，然后考虑该类和其他类之间有什么关系，若关系为利用或组合或聚集，则被使用的或作为组成的一份子的那方去存其他类的一个能唯一标识的静态成员作为互相间作用的钥匙（Order 存 Customer 和 Room 的，Room 存 Hotel 的，Manager 存 Hotel 的），钥匙也作为静态数据成员。类间在各种场合作用时都是通过类的静态数据成员的 get 和 set 函数实现。
遵从最小信息原则，没有任何存在的静态数据成员间存在冲突矛盾的问题。
注册时不允许重名。

3.1.1Customer 类

customerID 和 customer Passport 用于登陆，mobilephone 让酒店管理员可以和顾客联系，且是顾客身份识别的唯一标识（考虑现实手机号绑定到身份证号，唯一确定一个人）

3.1.2Hotel 类

hotelID 和 hotelPassport 登陆用，hotel ID 也就是酒店名，可以给平台和顾客看，city 用于顾客搜索时用，location 具体地址让顾客能知道在哪，telephone 让顾客可以电话酒店前台了解更多信息，assessment 和 assessNum 用于配合订单得分算出酒店的平均得分，workable 是否有运营许可，平台许可方可推出产品，hotelPictureAddress 模拟酒店上传图片，顾客查看商品房间信息时可看到。

3.1.3Manager 类

managerID 和 managerPassport 用于登陆，bhotelname 表示所属酒店，与 Hotel 类间建立联系。

3.1.4Room 类

roomID 是为了给 Room 一个唯一标识，方便程序设计而增加的，否则要通过酒店和房型两个数据才能唯一确定，此数据不对顾客开放，hotelname 所属酒店名，与 Hotel 类建立联系，pictureAddress 模拟酒店管理员增加房间商品种类时上传图片，picture，totalNum，remainingNum，roomType，discount，discribe（商家对之描述）都为一种房间商品的固有属性，sellable 为是否通过平台销售许可，许可后可以让顾客看到。

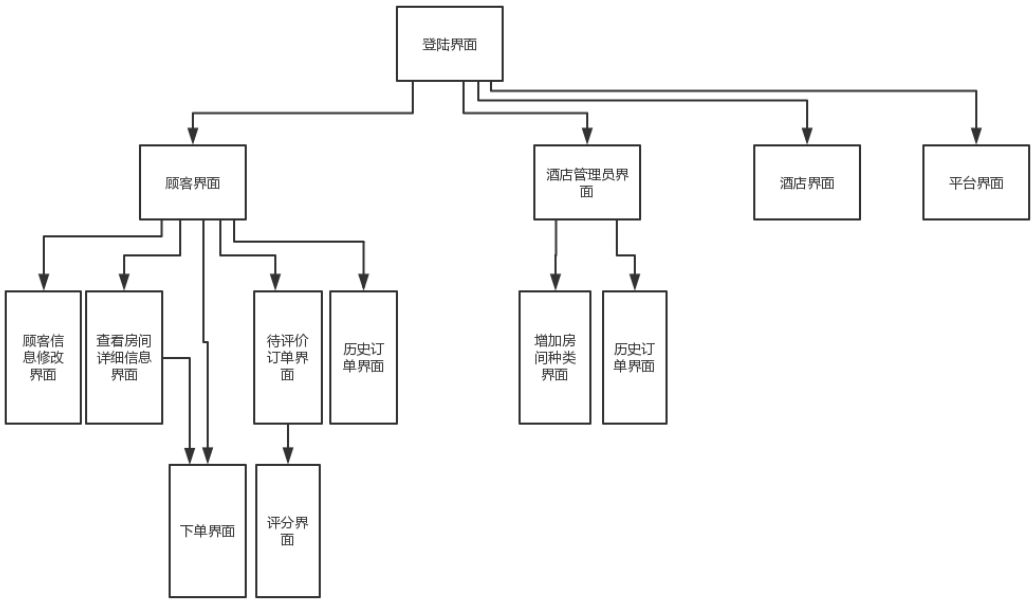
3.1.5Order 类

orderID 设计的 order 的唯一标识，在顾客下单时由系统自己不会重复地生成 bcustomerID 为下单顾客名，与 Customer 类建立联系，broomID 与 Room 类建立直接联系，与 Hotel 类间接联系，liveinDate 入住时间和 dayNum 预定天数为订单固有属性，progress 为订单状态，总共有 1 待接单 2 已接单 3 使用中 4 使用完毕 5 申请退款 6 已退款 7 拒绝接单 8 已取消订单八种

3.1.6 数据库的数据表设计

为以上 5 个类开 5 个 table 分别叫 customerInfoKu，orderInfoKu，roomInfoKu，managerInfoKu，hotelInfoKu，去存各个类的所有静态数据成员

3.2 界面结构设计



3.3 关键设计思路

内部结构：用 QHash 作为容器，有 customerInfo，hotelInfo，roomInfo,orderInfo,mangerInfo 这 5 个全局变量的容器去存这 5 个类的对象的所有静态数据成员，再加上一个全局变量 loginer 记录从登陆界面登陆的人。（☆有了这些数据一切唯一确定，类之间又通过 key 互相联系，作为各个部分理论上能获取的信息都能得到，那么问题就很简单了。）且这 5 个容器作为程序运行和数据库的中转站，开始运行程序时，从数据库读到容器，关闭程序前再从容器存到数据库。

外部界面：把待处理的和最常用的放在各个使用者进入后的第一个界面，实现最高效率

容错能力：一切错误操作用 QMessagebox 弹窗警告，程序不会错误运行不会崩（相当自信(∩_∩)）

4 项目总结

4.1 遇到的问题和解决方法

4.1.1 数据库的数据类型

本来以为数据库放进去什么直接取出来就是什么，设计好了类，房间和管理员是放在酒店里的，存酒店诗直接扔类里，读出来是都直接 query.value()然后就发现不行，然后查资料就发现扔库里的东西都变成 QVariant 了，取出来时有各种 to 某种类型，那么自定义的类型就没有 to 了，查了查有什么方法可以读自定义的，发现太难放弃了。然后就改类的结构，把 Room 和 Manager 类从 Hotel 里独立出来了，然后存一个 hotel ID 当钥匙，这样就没有自定义数据了。

4.1.2 像数据库存东西的语法问题

（始料不及啊，坑了我 1 天，晚上回来室友一句话就出来了）本来我是这样写的

```
query.exec("insert into orderInfoKu
values(i.value()->getorderID(),i.value()->getbcustomerID(),i.value()->getprogress(),i.value()->getbroomID(),i.value()->getliveinDate(),i.value()->getdayNum(),i.value()->getgrade())")
```

结果原来这样写那些

```
i.value()->get××
```

编译器都不会认，非要

```
QString str="insert into customerInfoKu values(?,?,?)";
  query.prepare(str);
   query.addBindValue( i.value()->getcustomerID());
   query.addBindValue( i.value()->getcustomerPassport());
   query.addBindValue( i.value()->getmobilePhone());
   query.exec();
```

这样写。浪费时间最久的一个很蠢的问题。

4.1.3 数据库键相同存数据的问题

本来想的每次从容器存数据库就把数据库全删了再存，但感觉操作量比较大，试了试不删直接存会怎么样，发现不会把同样的数据存两遍，于是就放心了，反正一样的就不存了，只有新的才存，所以只给管理员类加了先删表后存（因为考虑管理员被开除的问题），结果后面写创建房间时存数据库出问题了，因为房间商品构造函数 sellable 默认的是 false，需要平台去改成 true，但每次打开程序后就

发现有的我用平台改为 true 的又变成 false 了，当时很懵逼，看看从 true 变成 false 的和保持 true 的房间种类发现没啥特点啊，哼哼着玄学问题继续试，花了好久发现是存数据库不是数据完全一样才不存，而是键一样就不存了，所以那些酒店管理员增加房间种类后直接存的就是 false，
改成 true 后再存也不会改变，而那些增加房间种类并得到平台改为 true 后才点储存的就没问题。

4.1.4 信号与槽的作用域与构造函数导致有空指针出现异常退出问题

一开始遇到界面间传信息用的全局变量，后来感觉很 low 就去试信号与槽，本来我都是各个界面在符合条件的地方就直接声明并紧接着 exec，但这样有时连槽就说未声明，发现是作用域的问题，写 connect 的地方已经不在那个类的对象的作用域里了。然后我把类的声明先都写出来，在符合条件要打开时去 exec，结果开界面时程序异常结束根本打不开，就去 qdebug 看问题出在哪，最后发现是打开的界面有表格，声明这个界面类的对象时就调用了构造函数，去找表格信息的时候出问题。有一处我想酒店管理员发信号，顾客接受，所以顾客和酒店界面的声明都要在 mainwindow 里声明，登陆键的转到槽只负责 exec，但是登陆是有记录 loginer，所以登顾客时酒店管理员界面表格的刷新就用了顾客这个 loginer，然后就异常退出，然后我去在顾客和酒店管理员表格的函数里加上对 loginer 的判断，不符合身份就不执行，然后就出现了玄学问题，一直登顾客没问题，再等管理员就异常退出，一直登管理员页没问题，然后登的第一个顾客也没问题，再登第二个就异常退出，分析不了，放弃了这个地方的信号与槽。我觉得原理上我应该没什么问题了，这个没做出来很遗憾。一般的子界面发信号给父界面的槽还是用得挺好。

4.1.5 表格里选对象的问题

本来看网上说不选默认行数是-1，我就那样写如果是-1 就用 QMessagebox 弹个框说一下，结果发现还是在操作第一行。后来发现是这样，如果设计了选中模式只能整个单行选择，不选择时 currentrow 才是-1，而我之前用的是 cuurentitem->row，没选时就会是第一行。

4.1.6 放图片的问题

先是像素大了就只能放一角，然后设置成铺满，结果就拉伸后看着不舒服，最后选择了这样的按比例缩放

```
    QImage Image1;
```

```
    Image1.load(roomInfo[chosingroom]->getpictureAddress());
    QPixmap pixmap1 = QPixmap::fromImage(Image1);
    int with1 = ui->label_6->width();
    int   height1 = ui->label_6->height();
 QPixmap fitpixmap1 = pixmap1.scaled(with1, height1,
Qt::KeepAspectRatio, Qt::SmoothTransformation);
    ui->label_6->setPixmap(fitpixmap1);
```

4.1.7 哈希表排序问题

QHash 里头是乱放的的，排序到表格上输出就很麻烦，先想的比如降序，先找一个最大的，以后每次遍历找小于等于上一个且不是上一个的里最大的，本来试的例子少没什么问题，后来测试时数据多了，有三个同值时就出问题了，想了拿个数列对每个数据都全部遍历记录比它小的，觉得麻烦，最后把 QHash 拷到 QMap 上用 QMap 直接冒泡法做得。

4.1.8 顾客查看房间时排序的各种要求代码重用问题

为了提高复用率，写了个函数 f（a，b，c）a 是通过各种筛选后剩下的 roomInfo，b 是按什么来排，c 是升降序。先通过城市啊，是否有优惠啊筛选后得到一个 roomInfo1，再扔到 f 里去。这样代码复用率高。

4.2 亮点

4.2.1 遵从最小信息量原则。各方获得必要的信息，各方信息间不会冲突，一切从能否唯一确定去思考，编程思路就很清楚。拿着 5 个容器只要理论上能做到的就都能做到

4.2.2 使用 QHash 作为容器。有着很快的查找速度

4.2.3 界面设计简洁实用。把使用最多的放在登陆后第一个界面，高效省时。

4.2.4 界面美化。颜色的配合非常和谐好看，没有用对比度过高的颜色，而

是选择那些柔和的淡蓝色，樱色，淡紫色与绿色，让使用者感到身心愉悦。详细信息界面无边框处理，配上合适的背景周边图，简约清新。用 qss 对各个控件进行美化处理，让之不再看着生硬

4.3 心得体会

4.3.1 提高代码重用。一方面要粘贴进行修改的代码会减少，另一方面以后再改起来也方便的多，不用到处找出现的依次修改。能重用写成函数的尽量都写成函数。

4.3.2 前期设计好再写。一路上写着写着就发现该有的静态成员少了，或者哪个地方为了以后功能要写什么忘写了，那么后期再在以前的代码改就很累，一个是拍改的地方有漏的，另一个是容易改了以前的东西又出问题，所以我写一半后意识到这个问题，就先把所有后面的七八个界面都在草稿纸上画出来，哪些地方需要写到那里之前准备什么，或是需要那些数据，把最初考虑漏的都加上后才继续写，就很顺。看着我室友后期写到才发现漏东西回头改的艰难，更加增加我对前期规划的重视度。

4.3.3 完成度与进度，我觉的应该先写好主线再锦上添花。前期写就力求完成度一方面会让主题思路不够清晰，另一方面前期总是难免考虑不了太全，有可能会与后期主线冲突，所以支线应为主线让路，写主线时想到的支线先标出来，主线实现后，再去写支线。这样保证主线较简单的实现，而不能让前期简单实现的支线导致后期主线功能要绕弯实现。

4.3.4 debug 的经验，一直没反应八成死循环，异常退出八成空指针，写成一部分功能快就运行，实现不了功能就分析经过了那些代码，用 qDebug 快速找到出错的地方，错误范围就缩到一定范围，再根据错误的数据想想，基本就能确定到少于 4 种可能的原因或地方了，所以调得还算快。

4.3.5 信号与槽。两个界面隔太远时真的不好用。子界面传父界面时倒是安全实用。