“C++程序设计与训练”课程大作业

项目报告

项目名称：餐厅服务与管理系统

姓名： 林嘉成

学号： 2016011498

班级： 自 66

日期： 2017.9.15

目　录

1 系统功能设计

1.1 总体功能描述

1.1.1 开发背景

随着信息时代的飞速发展，人们的生活水平不断提高，餐饮业的消费持续增长。传统的单纯靠人工管理日常运作、人工记录顾客的点菜、人工通知厨师所需做的菜品、人工结账等等的以人工为主导的方式早已无法满足新时代的需求。所以，一套基于互联网的高效的餐厅服务与管理系统应时而生。如此不但能提高工作效率，也避免了传统方式会产生的不必要的麻烦，同时也方便了管理者对餐厅的运营。

1.1.2 顾客操作的相关功能

用户若以顾客身份登录，如果有账号密码，可直接输入登录，或注册之后再进行登录。登录成功后，会进入到座位选择界面，选择座位结束后，进入到点餐界面。顾客可以浏览菜品的名称、单价、平均评价星数（5 星满）。点餐结束后，待服务员、厨师认领。认领成功后，顾客可发送催菜、加水等消息于服务员。用餐完毕后进行结账并离开桌位。确认离开座位后，顾客可以选择性对服务员和已点的菜品进行评价。

1.1.3 服务人员操作的相关功能

服务人员主要包括服务员和厨师。

用户以服务员身份登录后，直接进入到服务员界面。服务员可以实时得到待服务的桌号信息，以及正在服务的桌号信息。同时，也能实时收到由顾客发来的加水、催菜等消息，且能够接收菜品已就绪的消息（以便为顾客上菜）。

用户以厨师身份登录后，直接进入到厨师界面。厨师可以认领菜品。烹饪完成后，确认已完成。

1.1.4 运营者操作的相关功能

运营者主要包括管理员和经理。

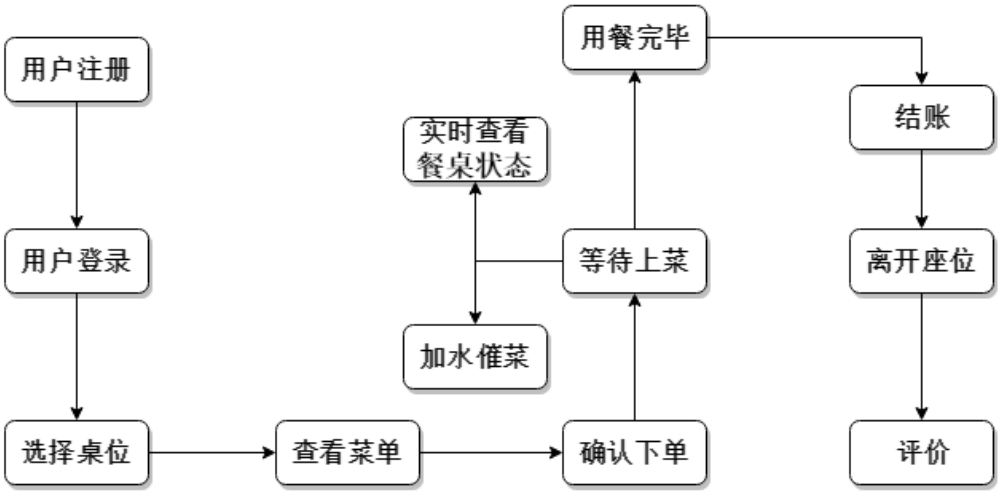
用户以管理员身份登录后，进入到管理员界面。管理员可以查看用户的信息以及菜品的信息。同时可以对其中的信息进行适当的修改，比如添加菜品类别、修改菜品单价、删除菜品，以及对用户身份的修改。

用户以经理身份登录后，进入到经理界面。经理可以实时查看服务员、厨师

的工作量以及平均评价，同时可以逐条浏览顾客对服务员以及菜品的评价。同时，经理也可以进入管理员界面进行管理员的一系列操作。

1.2 功能流程描述

1.2.1 顾客



顾客功能流程图

顾客功能如流程图所示，其中，结账之后，才能离开座位。确认离开座位之后，会弹出是否进行评价的窗口，根据顾客意愿判断是否进行评价。

注册 顾客初次使用该系统需要注册一个账号，输出手机号、密码、确认密码即可完成注册。其中，要求顾客输入规范的国内手机号码，密码不小于 7 位。注册界面实时对注册信息的规范性进行判断并反馈给正在注册的顾客。

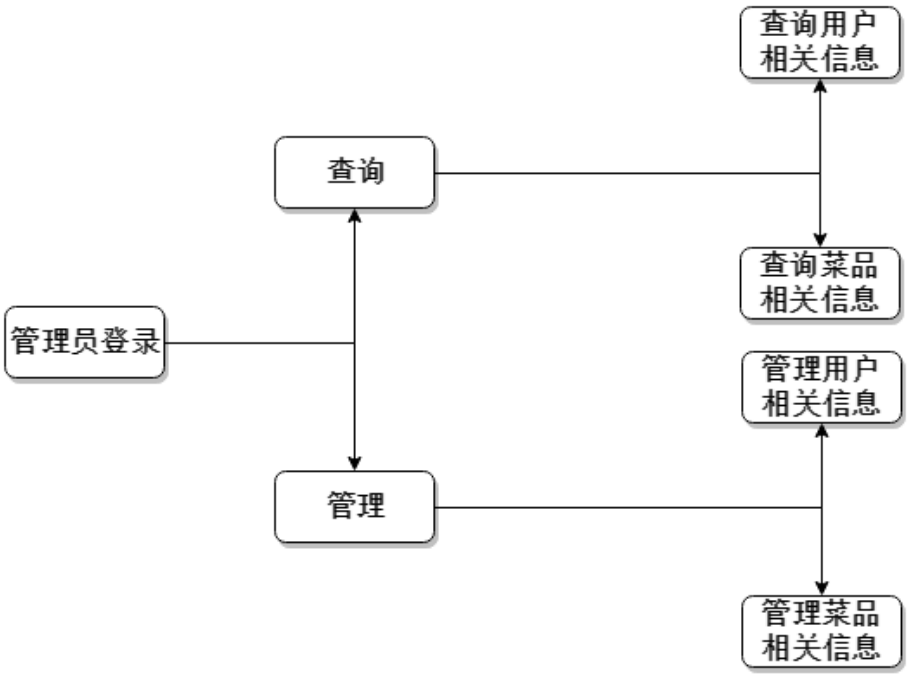
登录 用户输入已注册的账号及密码，待系统核实正确后，即可完成登录。

选择桌位 顾客登录之后，若之前未选择桌位，则进入选择桌位界面。

点菜 选择桌位之后，进入点菜界面。顾客可查看到所有菜品，当前桌号、当前服务员。同时也能查看顾客的购物车，下单之前可以随时增减菜品。确认下单后，等待服务员上菜。等待上菜期间，顾客可以实时查看菜品完成情况，且可以发送加水、催菜等消息给服务员。

评价 用餐完毕后，进行结账。顾客确认离开座位后，会出现评价对话框，顾客根据个人意愿对服务员、所点菜品进行星数、文本的评价。

1.2.2 管理员



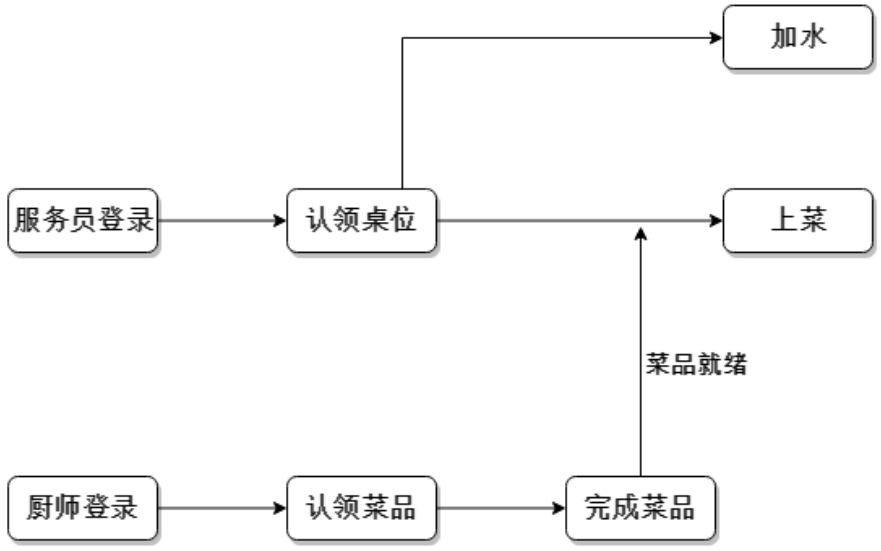
管理员功能流程

管理员功能如上流程图所示。管理员可以查看关于账户和菜品的信息，如菜品的总销量、平均评价。同时可以对信息进行管理，实现对菜品以及账户的增、删、改、查的功能。

管理员登录 管理员的账号需要经理或原有管理员进行授权，无法注册获得管理员权限。

查询管理 登录后，管理员可以查询账户及菜品的一系列信息，同时能够对相关信息进行管理，实现对账户和菜品的分类管理以及“增删查改”的功能。本餐厅系统的管理员同时拥有适当修改身份的功能，即厨师、服务员的权限无法注册得到，只能通过经理或服务员获得。

1.2.3 服务员、厨师



厨师、服务员
功能流程图

厨师、服务员功能流程如上图。其中，厨师认领由顾客下单后发送的点餐信息中的菜品，完成菜品之后，将信息发送给服务员，使得服务员收到正在服务的餐桌的菜品已就绪的消息并上菜。同时，服务员能够收到由顾客发来的加水、催菜的信息。

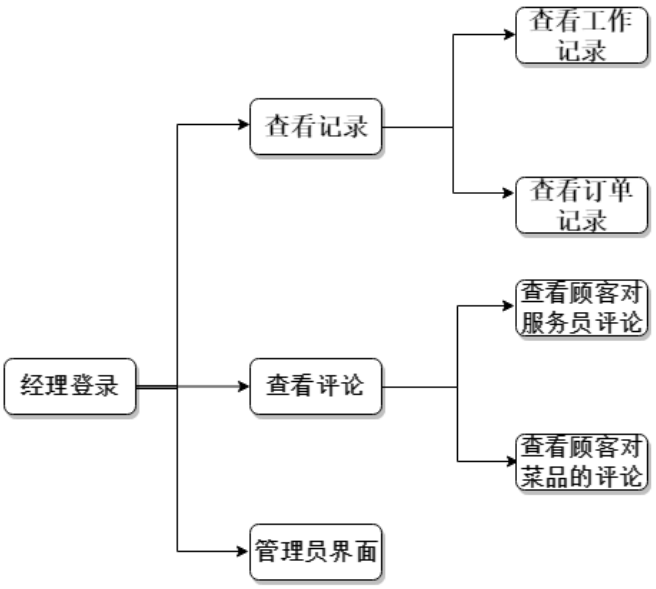
服务员：

认领服务桌位 进入服务员界面之后，服务员可以认领待服务的桌位，并能查看正在服务的桌号以及其发送的催菜、加水消息和其菜品的完成情况。

厨师：

认领、完成菜品 进入厨师界面后，厨师可以认领待做的菜肴，以及确认完成已认领的菜品，并自动将菜品就绪的消息发送给顾客及服务员。

1.2.4 经理



经理功能流程图

经理功能流程如上图所示。在此餐厅运作系统中，经理拥有最大的权限，即同时拥有管理员和经理本身的权限。经理可以查看服务员和厨师的工作记录以及顾客对服务员和菜品的评论。

查看记录 经理可以查看服务员的工作量、平均评价、厨师的工作量、顾客订单信息、厨师与服务员的工作记录。

查看评论 经理可以逐条查看顾客对服务员以及各个菜品的评论以了解餐厅经营的基本情况。

管理员权限 经理拥有管理员的所有功能，能够完成管理员权限的一系列操作。

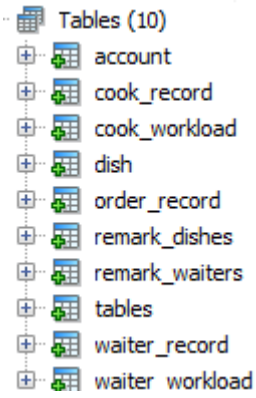
2 系统结构设计

2.1 模块说明概要

2.1.1 数据库的构造

该系统使用了 Qt 中的 Qt SQL 模块，底层选用了 SQLITE 数据库接口。

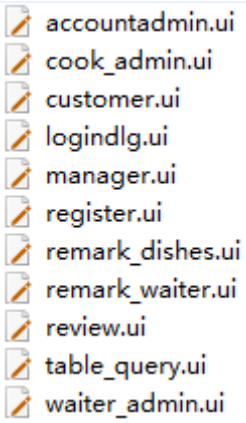
该系统共建立一个数据库，其中创建了 10 个表格。



其中 account 存储用户相关信息，cook_record，cook_workload 用来存储厨师工作记录，dish 用来存储菜品信息，order_record 用来存储订单信息，remark_dishes 、remark_waiters 存储顾客对菜品、服务员的评论，tables 用来存储桌号的相关信息，waiter_record，waiter_workload 用来存储服务员的工作信息。

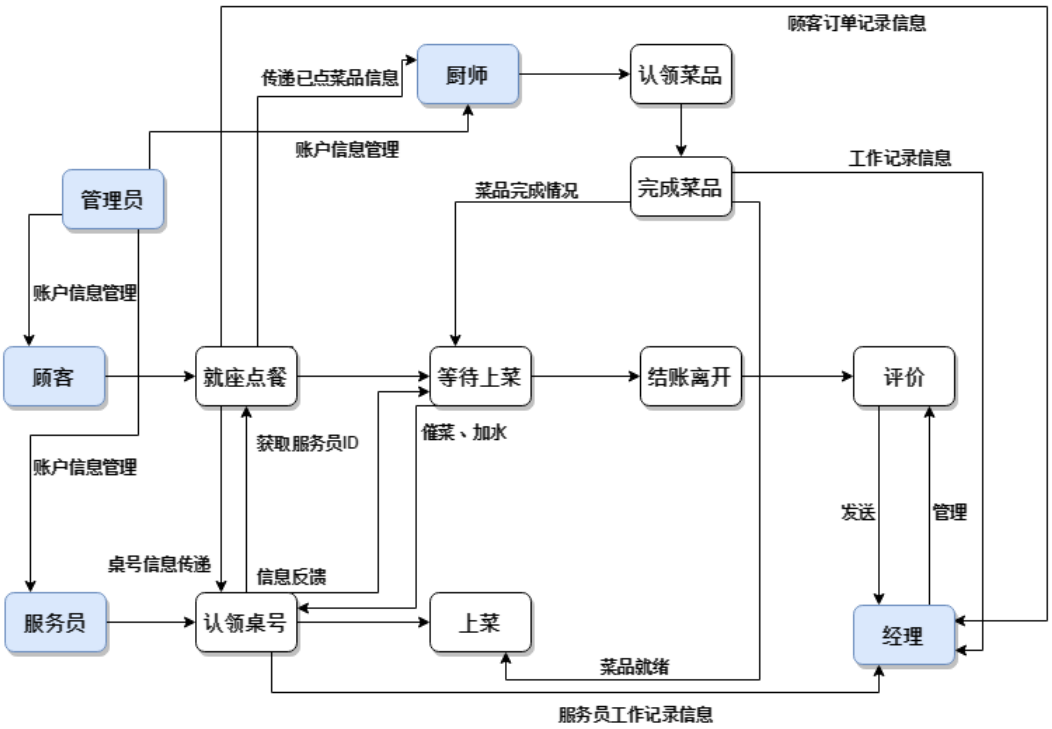
2.1.2 界面层

该系统共有 11 个 ui 界面，其中有登陆界面，注册界面，餐桌状态查询界面，查询菜品评论界面，查询服务员评论界面，管理员界面，服务员工作界面，厨师工作界面，顾客评论界面，经理界面。



2.1.3 逻辑层

如下图所示



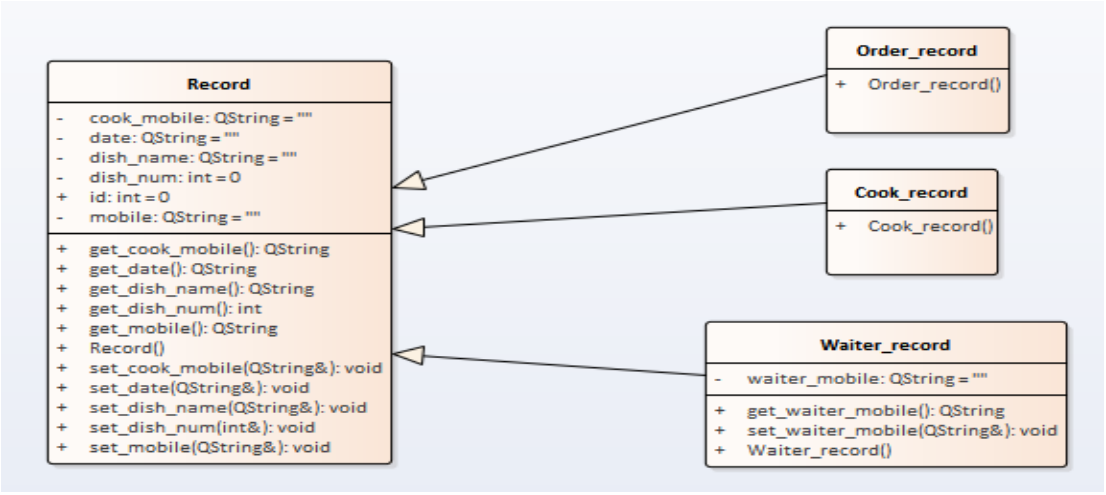
2.1.4 数据库与逻辑层的交互

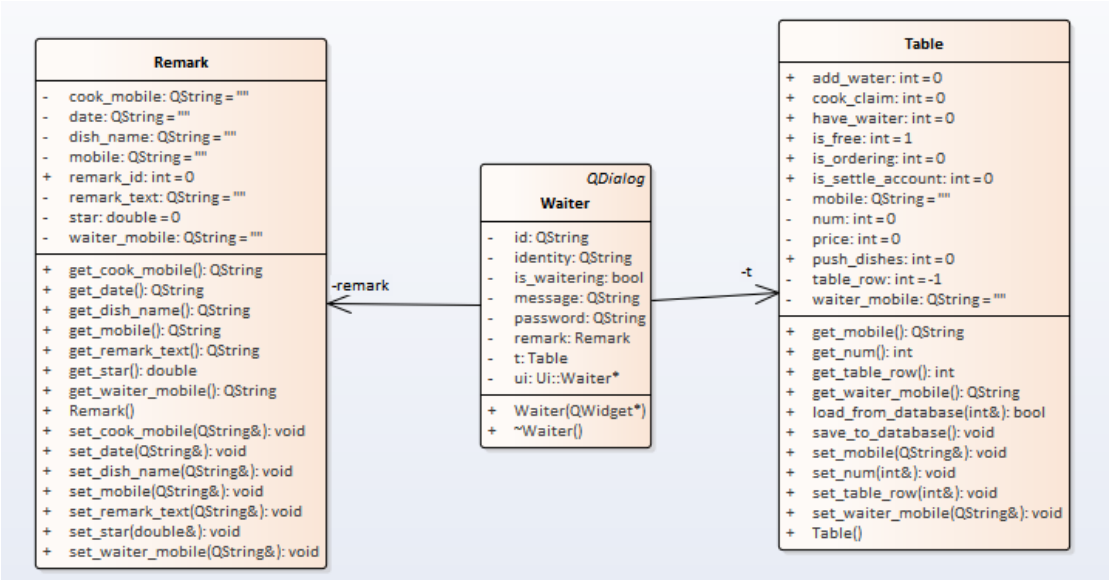
系统在程序运行初进行初始化，将数据库的内容读入内存，在逻辑层中对内存的数据进行操作。程序关闭之后，在调用主界面析构函数之前，将内存中的数据存入数据库加以保存。

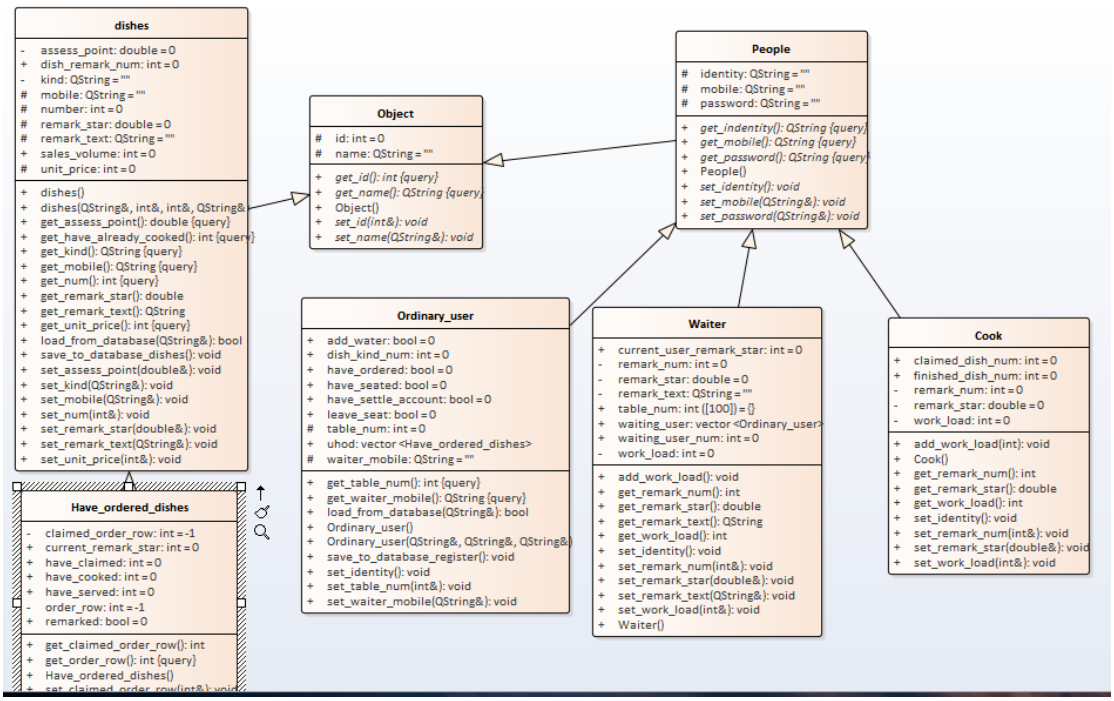
2.1.5 逻辑层与界面层的交互

逻辑层与界面层的交互主要借助了 Qt 中的信号和槽机制、QListWidget、QtableView、QTableWidget 等等 Qt 中的一系列功能，实现了界面间的切换以及将内存中的数据在界面中的显示，使得顾客、服务员、厨师、经理能够获取一定的信息。同时，数据库与 TableView 的组合，能够使管理员对餐厅信息进行适当的管理。

2.2 UML 类图


Record
- cook_mobile: QString = ""
- date: QString = ""
- dish_name: QString = ""
- dish_num: int = 0
+ id: int = 0
- mobile: QString = ""
+ get_cook_mobile(): QString
+ get_date(): QString
+ get_dish_name(): QString
+ get_dish_num(): int
+ get_mobile(): QString
+ Record()
+ set_cook_mobile(QString&): void
+ set_date(QString&): void
+ set_dish_name(QString&): void
+ set_dish_num(int&): void
+ set_mobile(QString&): void
Order_record
+ Order_record()
Cook_record
+ Cook_record()
Waiter_record
- waiter_mobile: QString = ""
+ get_waiter_mobile(): QString
+ set_waiter_mobile(QString&): void
+ Waiter_record()


Remark
- cook_mobile: QString = ""
- date: QString = ""
- dish_name: QString = ""
- mobile: QString = ""
+ remark_id: int = 0
- remark_text: QString = ""
- star: double = 0
- waiter_mobile: QString = ""
+ get_cook_mobile(): QString
+ get_date(): QString
+ get_dish_name(): QString
+ get_mobile(): QString
+ get_remark_text(): QString
+ get_star(): double
+ get_waiter_mobile(): QString
+ Remark()
+ set_cook_mobile(QString&): void
+ set_date(QString&): void
+ set_dish_name(QString&): void
+ set_mobile(QString&): void
+ set_remark_text(QString&): void
+ set_star(double&): void
+ set_waiter_mobile(QString&): void
-remark
QDialog
Waiter
- id: QString
- identity: QString
- is_waitering: bool
- message: QString
- password: QString
- remark: Remark
- t: Table
- ui: Ui::Waiter*
+ Waiter(QWidget*)
+ ~Waiter()
-t
Table
+ add_water: int = 0
+ cook_claim: int = 0
+ have_waiter: int = 0
+ is_free: int = 1
+ is_ordering: int = 0
+ is_settle_account: int = 0
- mobile: QString = ""
- num: int = 0
- price: int = 0
+ push_dishes: int = 0
- table_row: int = -1
- waiter_mobile: QString = ""
+ get_mobile(): QString
+ get_num(): int
+ get_table_row(): int
+ get_waiter_mobile(): QString
+ load_from_database(int&): bool
+ save_to_database(): void
+ set_mobile(QString&): void
+ set_num(int&): void
+ set_table_row(int&): void
+ set_waiter_mobile(QString&): void
+ Table()

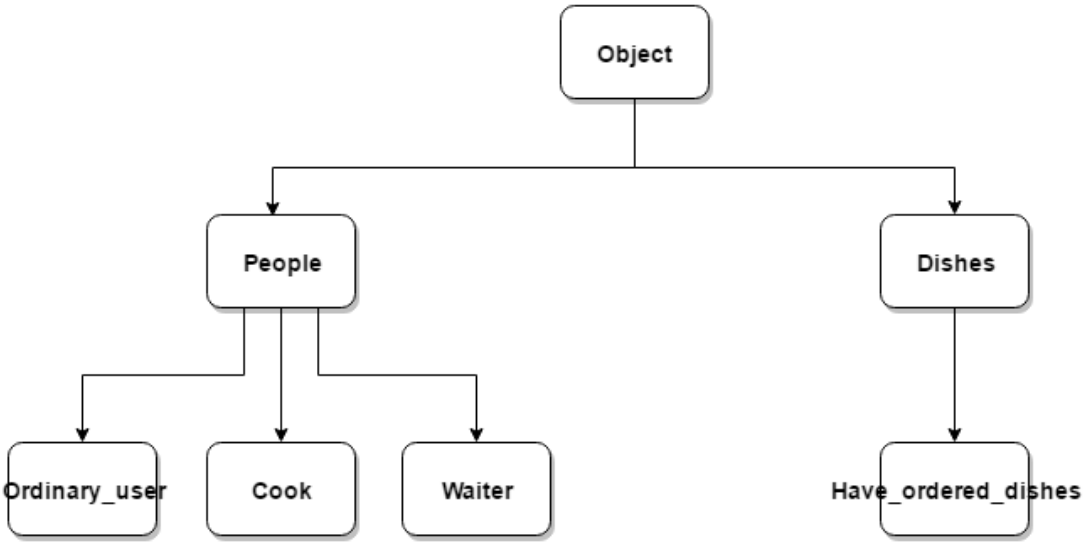

dishes
- assess_point: double = 0
+ dish_remark_num: int = 0
- kind: QString = ""
mobile: QString = ""
number: int = 0
remark_star: double = 0
remark_text: QString = ""
+ sales_volume: int = 0
# unit_price: int = 0
+ dishes()
+ dishes(QString&, int&, int&, QString&)
+ get_assess_point(): double {query}
+ get_have_already_cooked(): int {query}
+ get_kind(): QString {query}
+ get_mobile(): QString {query}
+ get_num(): int {query}
+ get_remark_star(): double
+ get_remark_text(): QString
+ get_unit_price(): int {query}
+ load_from_database(QString&): bool
+ save_to_database_dishes(): void
+ set_assess_point(double&): void
+ set_kind(QString&): void
+ set_mobile(QString&): void
+ set_num(int&): void
+ set_remark_star(double&): void
+ set_remark_text(QString&): void
+ set_unit_price(int&): void
Object
id: int = 0
name: QString = ""
+ get_id(): int {query}
+ get_name(): QString {query}
+ Object()
+ set_id(int&): void
+ set_name(QString&): void
People
identity: QString = ""
mobile: QString = ""
password: QString = ""
+ get_indentity(): QString {query}
+ get_mobile(): QString {query}
+ get_password(): QString {query}
+ People()
+ set_identity(): void
+ set_mobile(QString&): void
+ set_password(QString&): void
Ordinary_user
+ add_water: bool = 0
+ dish_kind_num: int = 0
+ have_ordered: bool = 0
+ have_seated: bool = 0
+ have_settle_account: bool = 0
+ leave_seat: bool = 0
# table_num: int = 0
+ uhod: vector <Have_ordered_dishes>
# waiter_mobile: QString = ""
+ get_table_num(): int {query}
+ get_waiter_mobile(): QString {query}
+ load_from_database(QString&): bool
+ Ordinary_user()
+ Ordinary_user(QString&, QString&, QString&)
+ save_to_database_register(): void
+ set_identity(): void
+ set_table_num(int&): void
+ set_waiter_mobile(QString&): void
Waiter
+ current_user_remark_star: int = 0
- remark_num: int = 0
- remark_star: double = 0
- remark_text: QString = ""
+ table_num: int ([100]) = {}
+ waiting_user: vector <Ordinary_user>
+ waiting_user_num: int = 0
- work_load: int = 0
+ add_work_load(): void
+ get_remark_num(): int
+ get_remark_star(): double
+ get_remark_text(): QString
+ get_work_load(): int
+ set_identity(): void
+ set_remark_num(int&): void
+ set_remark_star(double&): void
+ set_remark_text(QString&): void
+ set_work_load(int&): void
+ Waiter()
Cook
+ claimed_dish_num: int = 0
+ finished_dish_num: int = 0
- remark_num: int = 0
- remark_star: double = 0
- work_load: int = 0
+ add_work_load(int): void
+ Cook()
+ get_remark_num(): int
+ get_remark_star(): double
+ get_work_load(): int
+ set_identity(): void
+ set_remark_num(int&): void
+ set_remark_star(double&): void
+ set_work_load(int&): void
Have_ordered_dishes
- claimed_order_row: int = -1
+ current_remark_star: int = 0
+ have_claimed: int = 0
+ have_cooked: int = 0
+ have_served: int = 0
- order_row: int = -1
+ remarked: bool = 0
+ get_claimed_order_row(): int
+ get_order_row(): int {query}
+ Have_ordered_dishes()


Data
+ admin: vector <Ordinary_user>
+ admin_num: int
+ cook: vector <Cook>
+ cook_num: int
+ cook_record: vector <Cook_record>
+ cook_record_num: int
+ cook_record_temp: int
+ dish: vector <dishes>
+ dish_num: int
+ order_record: vector <Order_record>
+ order_record_num: int
+ order_record_temp: int
+ remark_dish_id_record: int
+ remark_dish_num: int
+ remark_dishes: vector <Remark>
+ remark_waiter: vector <Remark>
+ remark_waiter_id_record: int
+ remark_waiter_num: int
+ tb: vector <Table>
+ user: vector <Ordinary_user>
+ user_num: int
+ waiter: vector <Waiter>
+ waiter_num: int
+ waiter_record: vector <Waiter_record>
+ waiter_record_num: int
+ waiter_record_temp: int
- Data()

3 系统详细设计

3.1 类结构设计

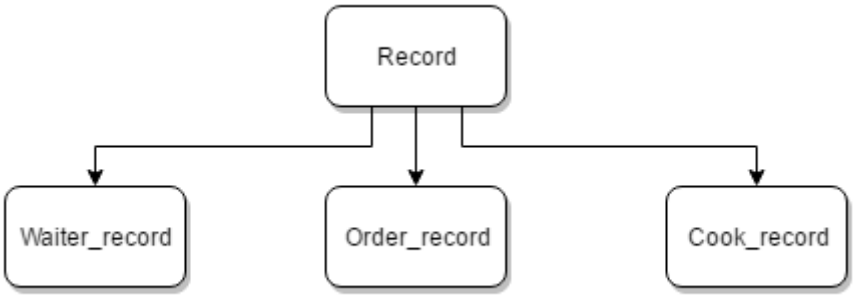
3.1.1 Object 类



由 Object 抽象类派生出 People 类和 Dishes 类，People 类派生出 Ordinary_user、Cook、Waiter 三个类，由 Dishes 派生出 Have_ordered_dishes 类。Object 作为基类，内包括 ID、name 等私有成员。Dishes 是菜品类，Have_order_dishes 是顾客已点菜品类，故由 Dishes 类派生。Ordinary_user 是顾客类，Cook 为厨师类，Waiter 是服务员类，继承了 People 中的 mobile、password、identity 等私有成员。People 类中声明纯虚函数，使得派生出的三个类根据自身需要进行定义，实现了**多态**。

该结构合理使用了**类的组合**。Waiter 类的定义中使用 Ordinary_user 类，用来记录服务员正在服务的用户的相关信息；Ordinary_user 类的定义中使用 Have_ordered_dishes 类，用来记录顾客已点的菜品。

3.1.2 Record 类



Record 为记录类，用来记录相关信息。由此派生出 Order_record、Cook_record、Waiter_record，分别用来记录订单信息、厨师工作信息、服务员信息。

3.1.3 Table 类

Table 类的成员变量多数为对餐桌状态的记录，如是否空闲、是否已下单、是否已结账、是否向服务员催菜加水、是否离开等等。

3.1.4 Remark 类

Remark 类是评论类，可以记录评论的评论者、被评论对象、评价星级、评论文本、评论 ID、以及评论日期。

3.1.5 Data 类

Data 类内定义的静态变量用来实现变量的跨文件使用，进而实现跨界面的信息传递。

3.2 数据层

3.2.1 表格的创建

具体代码如下：

```
query.exec("create table cook_workload (mobile varchar primary key,remark_star double"
                                    ",workload int,remark_num int)");
query.exec("create table waiter_workload (mobile varchar primary key,remark_star double"
                                    ",workload int,remark_num int)");

query.exec("create table remark_dishes (id int primary key, mobile varchar, dish_name varchar,point int , text varchar"
                                    ",date varchar)");

query.exec("create table remark_waiters (id int primary key, mobile varchar, waiter_mobile varchar,point int , text varchar"
                                    ",date varchar)");
query.exec("create table order_record (id int primary key, mobile varchar, dish_name varchar,num int"
                                    ",date varchar)");
query.exec("create table waiter_record (id int primary key,waiter_mobile varchar, mobile varchar"
                                    ",date varchar)");
query.exec("create table cook_record (id int primary key,cook_mobile varchar,dish_name varchar,num int,mobile varchar"
                                    ",date varchar)");
query.exec("create table dish (id int primary key,dish_name varchar,unit_price double ,remark_star double , remark_num int , sales_volume int"
                                    ",kind varchar)");
query.exec("create table tables (num int primary key,mobile varchar,price int,settle_account int,have_waiter int,cook_claim int,is_free int,is_ordering int"
                                    ",add_water int,push_dishes int)");
for(int i = 1;i <= 36 ;i++)
{
    query.exec(QString("insert into tables values(%1,'',0,0,0,0,1,0,0,0)").arg(i));
}
query.exec("create table account (id varchar primary key,identity varchar"
                                    ",password varchar)");
```

3.2.2 数据初始化

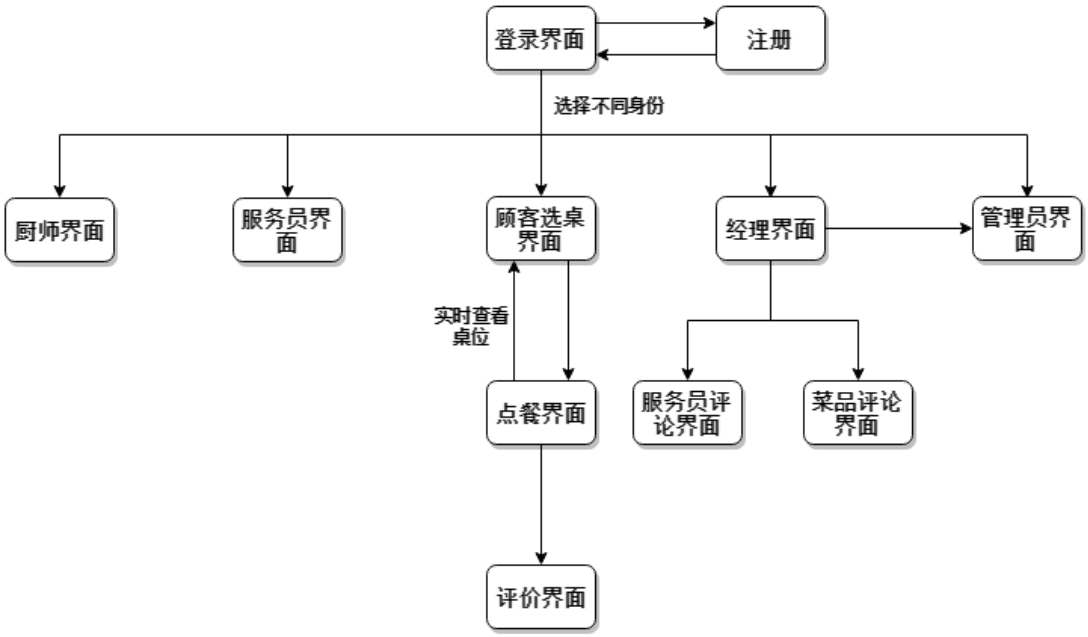
在 main 函数中，定义了 void initialize() 函数，将数据库的信息读取到内存，将相关变量进行初始化。

3.3.3 数据的储存

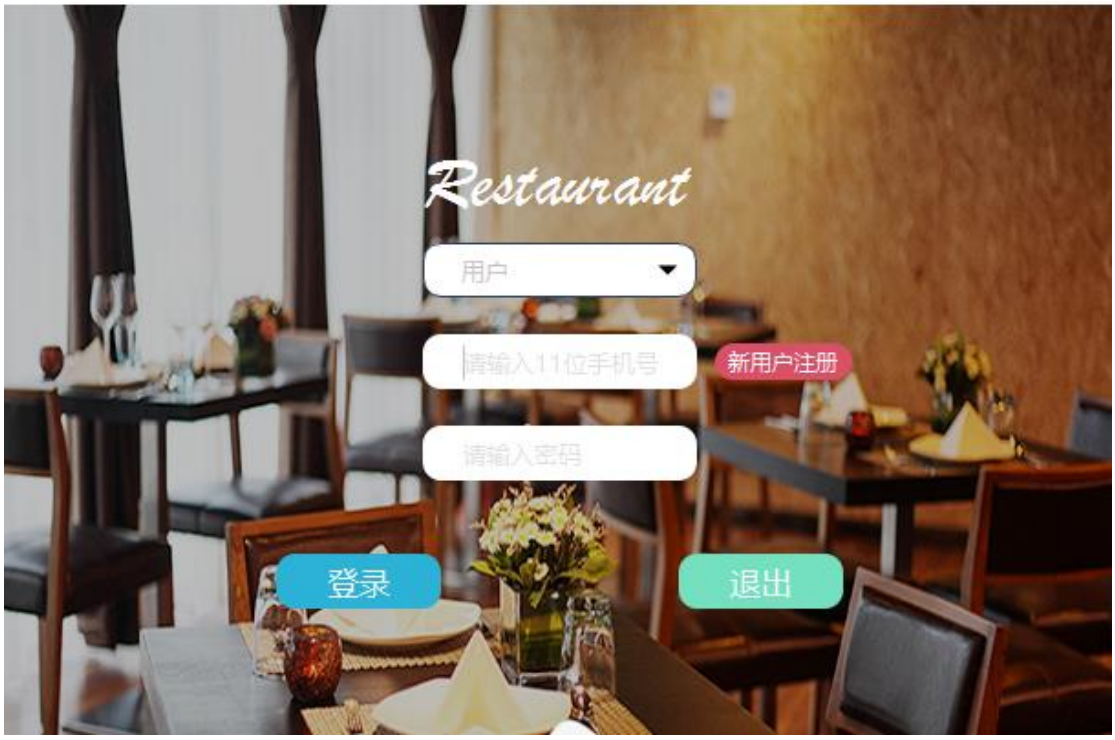
在主界面析构函数调用前，将相关数据存入数据库，使得在程序正常关闭之后，内存中的相应数据会存储到数据库中，实现对数据的储存。

3.3 界面层

3.3.1 界面结构



3.3.2 登录/注册界面



登录界面由 3 个 PushButton，1 个 ComboBox，以及一个输入文本为“Restaurant”的 Label 组成。

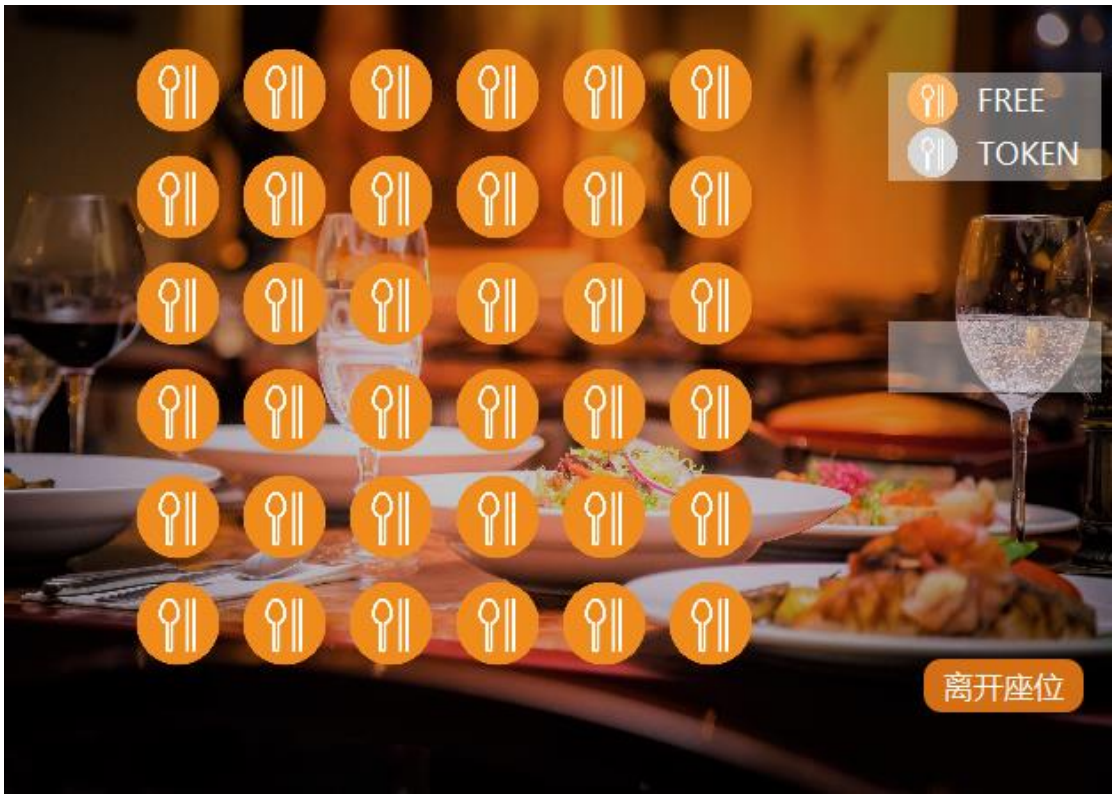
由于 ComboBox 的默认样式与整个登录界面不搭，故修改了其样式，代码如下。

```
ui->model_chosen->setStyleSheet
("QComboBox::drop-down {\
 subcontrol-origin: padding;\
 subcontrol-position: top right;\
 width: 30px;\
 border-left-width: 1px;\
 border-left-color: white;\
 border-left-style: solid; /* just a single line */\
 border-top-right-radius: 3px; /* same radius as the QComboBox */\
 border-bottom-right-radius: 3px;\
 }"
 "QComboBox::down-arrow { image: url(:/Image/下拉框图标3.png);}"
 "QComboBox{border: 1px solid #32435E;\
 border-radius:10% ;color:rgb(200, 193, 197); font: 10pt '微软雅黑'; padding-left:20%; }"
 );
```

账号和密码的输出框设置了 placeholderText，以提示用户输入 LineEdit 的要求。

注册界面与登录界面比较类似。

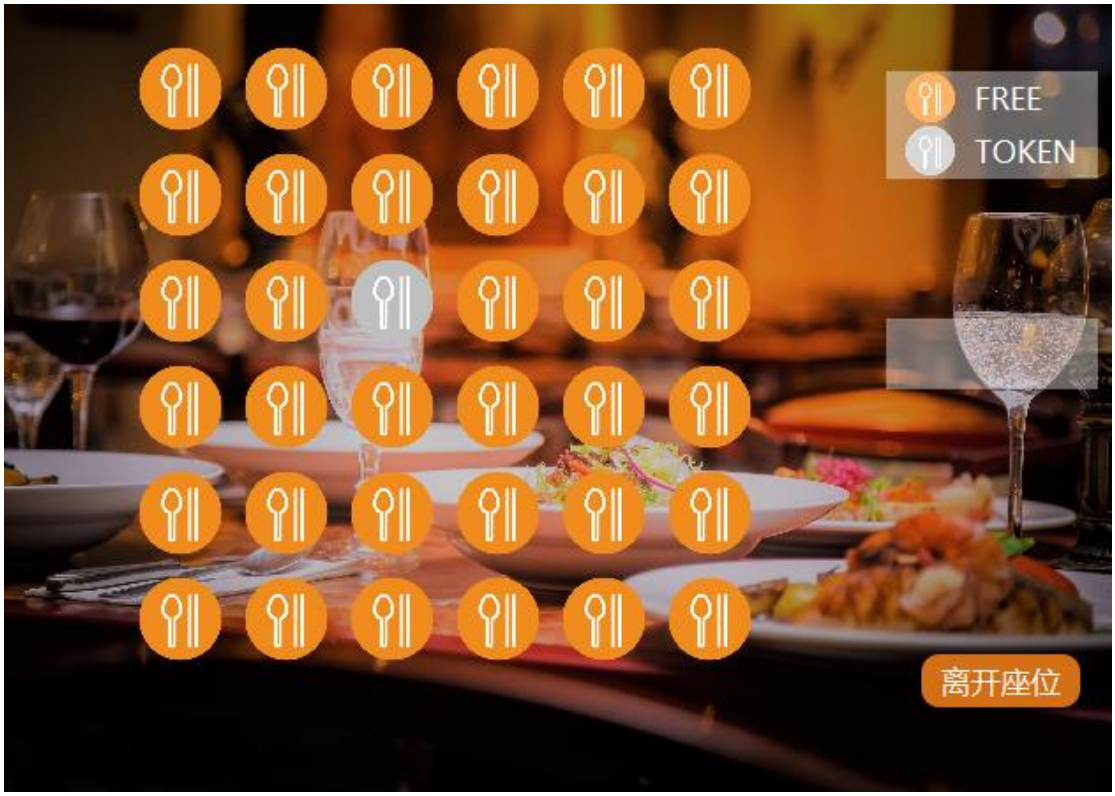
3.3.3 桌位界面



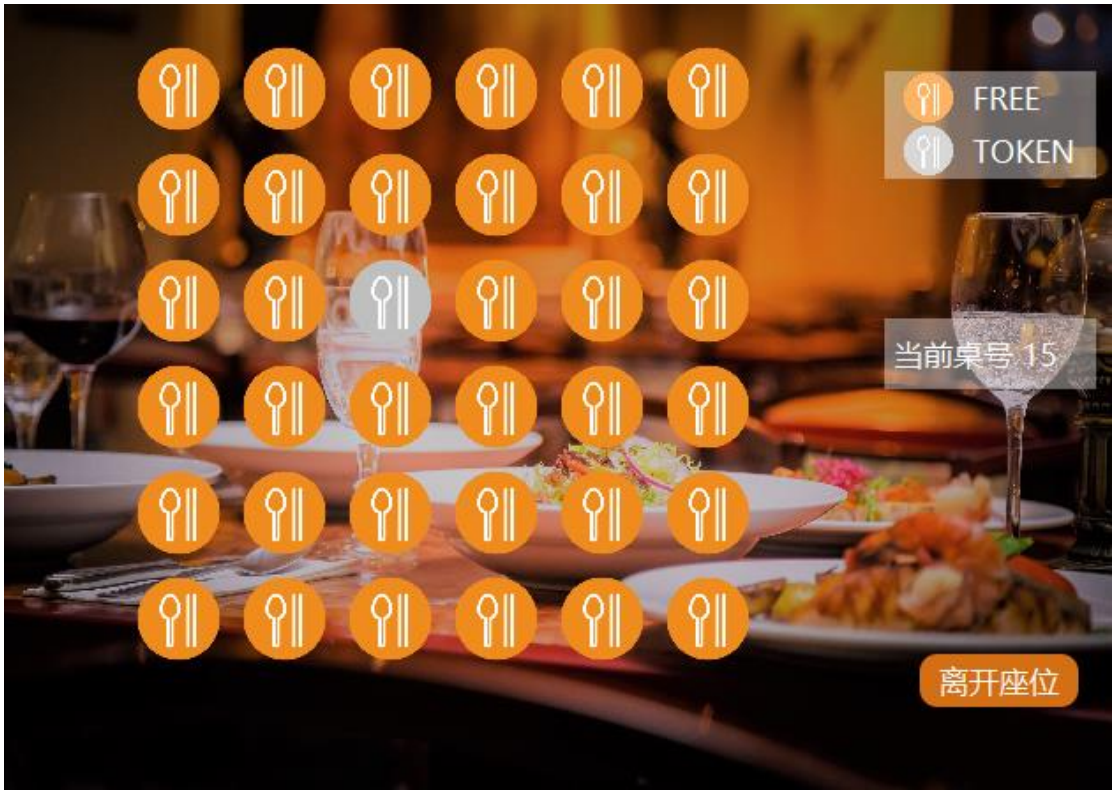
桌位界面主要由 5 个 Label，1 个 groupBox，1 个“离开座位”的 PushButton，以及在 groupBox 中动态生成的总桌数的 PushBuuton。

右上角为图例，当桌位上有人时，则桌位变为灰色，空闲时为橙色，如下图

所示。



当点击空闲桌位时，会出现对话框提示桌号并询问是否确认选桌。同时选桌成功后进入点餐界面可以实时查询餐桌状态。



3.3.4 点餐界面

点餐界面的主体是 ListWidget。

确认下单之前，界面如上图。顾客可以查询所有菜品，得知菜品的类别、名字、单价、评价。同时为了方便顾客查询已点，本系统将购物车和菜单放置在同一界面里。顾客可以实时查询餐桌状态，并可以得到当前服务员 ID 信息。右下方通过使用重载 QDailog 的 *paintEvent*，实时更新当前已点菜品的总价格。

确认下单后，如图所示。

顾客可以实时查看菜品烹饪状态，并可以给服务员发送催菜、加水等消息。

3.3.5 评价界面

评价界面主要由 TabWidget 构成，包括服务员评价界面和已点菜品评价界面。评价支持星级评价以及文本评价。星级评价由 5 个 PushButton 组成，根据鼠标点击不同位置的星星而调用相应的槽函数。

菜品界面通过 back 和 next 的 PushButton 依次对已点菜品进行评价。并且，顾客的操作记录都得到了保存。确认提交评价之后，相关信息会传送给经理。

3.3.6 服务员界面

服务员主要由 4 个 ListWidget 构成，依次获取待服务的桌号、正在服务的桌号及其菜品信息、催菜加水消息。

3.3.7 厨师界面

厨师的界面主要由 2 个 ListWidget 构成，依次获取待认领的菜以及需要完成

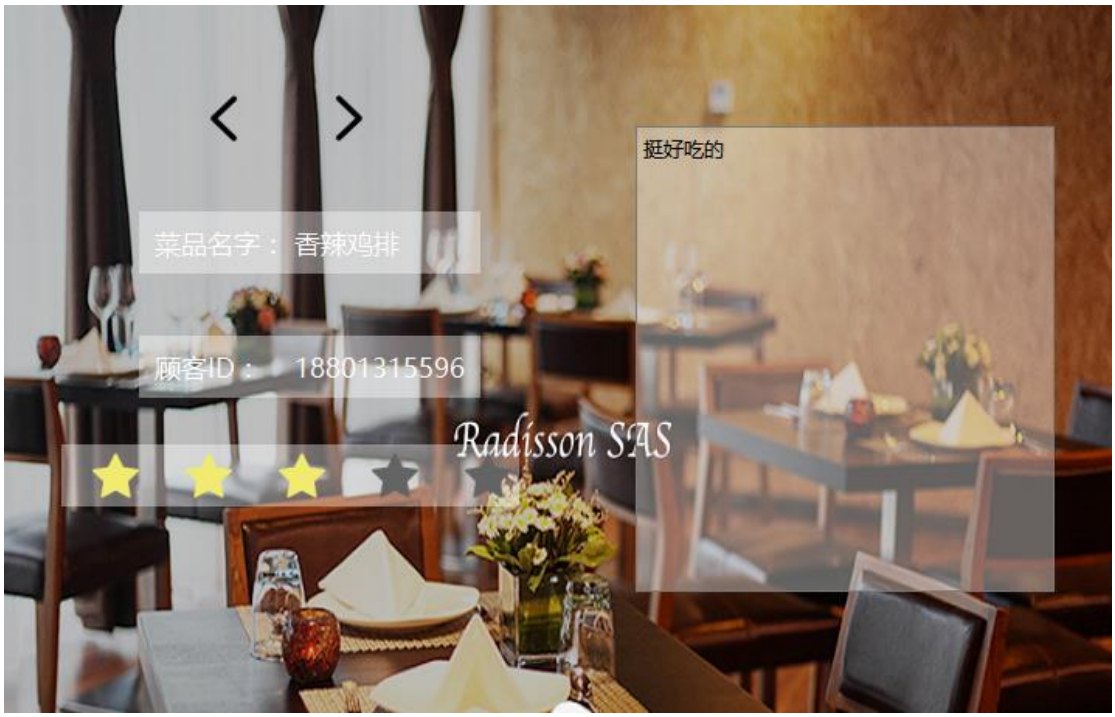
的菜。厨师通过“认领选中”、“完成选中”两个按钮，将菜品的烹饪进度消息发送给顾客及服务员。

3.3.8 管理员界面

管理员主要负责用户和菜品信息的管理，可以实现对相关信息的增、删、添、查、改。对账户管理的主要信息为 ID、身份、密码；对菜品管理的主要信息为 ID、菜名、单价、评价星级、评价数量、菜品总销量以及菜品的种类。

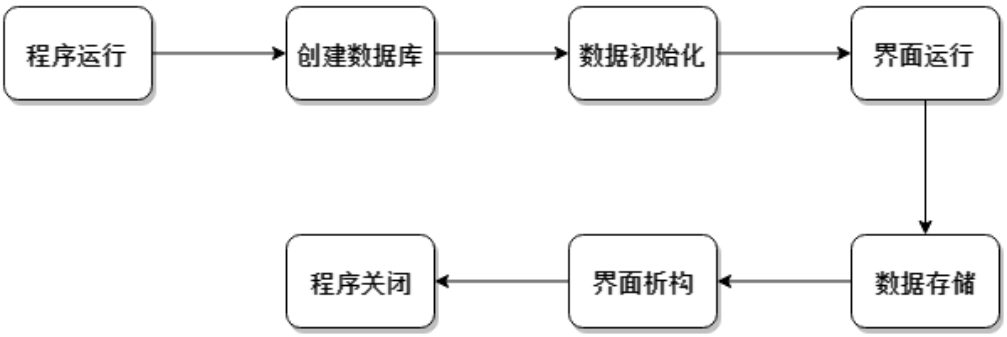
3.3.9 经理界面

经理界面主要由 4 个 ListWidget 构成。主要显示服务员工作信息、厨师工作信息、订单记录、厨师工作记录。经理可以逐条查看菜品评论及服务员评论。



3.4 程序运行流程

如下图所示



3.5 容错功能

3.5.1 数据库容错

管理员的修改账户或菜品信息时，为防止数据库出现丢失或其他问题，在将数据更新至数据库之前，对数据库的有效性进行相应的判断。

3.5.2 用户操作容错

注册 注册界面，该系统规定注册账号格式为中国大陆手机号格式，密码由字母和数字组成，至少七位。对 QDailog 的 *paintEvent* 重载，使得实时判断输出格式的规范性并进行提示。、

点餐界面 顾客登录选桌后，会通过成员变量记录顾客已选桌的状态。这样，即使顾客在确认订单前误操而退出，也能保存顾客已有的订单记录，再次登录时，仍能看到购物车内的菜品。在进行餐桌状态查询时，由于记录了已选桌状态，所以顾客在离开桌位之前不能重复选桌，而且该系统设置，顾客只有在结账之后才能离开桌位。

3.5.3 信息储存容错

该系统将信息储存的环节放在了主界面析构函数调用之前。为了防止误操将界面关闭，代码中重载了主界面的 *closeEvent*，在关闭事件发生前，会出现 QMessageBox 的问题对话框，信息在确认关闭之后才能进行存储到数据库。

3.5.4 输入机制容错

利用 Qt 的 QTextCodec 类，使得用户输入的所有文本都被视为 **UTF-8 字符串**，

避免由于用户输入不常见字符导致系统无法识别而崩溃的可能。同时，在注册界面和登录界面，利用了**正则表达式**，将账号输入框设置为只能输入数字，将密码输入框设置为只能输入数字和字母，以避免由于误输其他字符造成的不必要的麻烦。

3.6 关键设计思路

构建本系统框架时，关键的设计思路是数据储存以及多界面间的信息传递。为此，该系统合理利用了类的组合，Ordinary_user 内 Have_ordered_dishes 存储顾客已点的菜品的相关信息，Waiter 类 Ordinary_user 存储服务员服务的顾客的相关信息。在程序初始化时，数据库中的数据会读取到存储信息的 Data 类相应的 vector 类型的**静态变量**中，从而实现动态存储与读取。

4 项目总结

4.1 设计总结

该系统主要分为三个模块：逻辑层、界面层、数据层。三个层次彼此分明：逻辑层和数据层通过程序运行时的初始化和主界面析构之前的数据保存进行对接；逻辑层和界面层通过 Qt 相应功能进行对接。该系统合理应用 C++的 OOP 思想，合理利用类的继承与派生、多态性等知识点。

该系统的界面设计利用了适量 CSS 的知识，试图最大可能的将界面进行合理的美化。

为了避免因误操或其他原因而造成的错误，该系统在构建时考虑到适当的容错系统，如应用正则表达式来限制账号或密码输入框的输入字符的格式。

4.2 开发及调试工作中的问题及解决方法

在开发时，遇到了一个很久才解决的 bug：在一个类中添加新的类成员，在编译时，总是报错。后经上网查询相关资料，得出原因是 Qt 内部自身的 bug,解决方法是将项目 rebuilt 重新构建即可。

此外，在应用数据库初时，总会遇到 database not open 这样无法打开数据库的错误。经查阅书籍，得出错误原因是数据库连接代码出现了问题，解决方案是将数据库连接的代码进行修改。

4.3 难点与亮点

4.3.1 难点

难点一是**界面设计**，合理地设计界面格外重要，使得界面既看起来美观，使用起来也方便。这对于一个没有任何美工经验的人是一个很大的挑战。在设计好每个界面的框架之后，尽最大努力使界面变得尽可能美观。但由于一定原因，该系统的界面仍有很多优化的余地。

难点二是**跨界面信息的传递**，为了减少代码量，跨界面将数据传递成为其中一个难点。为此，把跨文件的变量设为静态变量使用从而实现了对跨界面信息的传递。

4.3.2 亮点

亮点一是**多界面合一的设计**。不同于多数的每个流程都做一个界面的想法，本系统将界面的个数进行压缩，将可合并的界面放在了一个界面中，实现了对界面数量的优化。在既不影响操作和美观的情况下，最大程度的将界面数量减少。

亮点二是一定的**容错能力。**为了避免由于误操而造成的不必要的麻烦，在逻辑层中，加入了一些容错的代码。同时运用正则表达式、QTextCode::setCodecForLocale 等知识，对用户输入的格式进行限制，增强了系统运行的安全性。

4.4 心得体会

4.4.1 自学能力的重要性

C++与程序设计小学期这门课是以实践为核心，编程能力的提升的根本途径无疑是多进行上机实践。Qt 在大作业中属于自学模块，除此之外，对于界面的优化还需要学习一些 CSS 的知识。在领悟知识的基础之上，更重要的就是合理的应用所学的知识：如何合理的构建类的框架、如何合理的构建界面的框架、如何合理的利用数据库合理的存储与读取数据。

4.4.2 框架构建的重要性

在正式写该项目之前，首要的把该项目的框架构建出来。框架主要分为三个层面：逻辑层、界面层、数据层。其中界面层主要应用 Qt 的相关知识，数据层主要应用数据库的相关知识，而逻辑层是该框架的核心，运用了面向对象的思想，将应有的类合理规划，理清继承和派等等类之间的关系。其次，合理设计每个层面之间的对接。在主要框架构建结束之后开始正式写此项目，会避免由于没有大体的框架而造成的反复的修改代码的结构的麻烦。

5 相关问题的说明