目录

[一、实训目的 1](#_Toc43654758)

[1.1、需求分析 1](#_Toc43654759)

[1.2、实验要求 1](#_Toc43654760)

[1.3、实验背景 1](#_Toc43654761)

[二、实训原理 1](#_Toc43654762)

[2.1、用到的类 1](#_Toc43654763)

[2.2、用到的控件 2](#_Toc43654764)

[2.3、图形视图框架 6](#_Toc43654765)

[2.4、信号与槽 6](#_Toc43654766)

[2.5、线程同步 7](#_Toc43654767)

[2.6、进程通信 8](#_Toc43654768)

[三、实训设计思路 9](#_Toc43654769)

[3.1、总体设计 9](#_Toc43654770)

[3.2、设计流程图 9](#_Toc43654771)

[3.3、登陆与注册 10](#_Toc43654772)

[3.4、服务器框架 10](#_Toc43654773)

[3.5、客户机框架 11](#_Toc43654774)

[3.6、售票端功能详述 11](#_Toc43654775)

[四、实训内容 11](#_Toc43654776)

[4.1、设计ui界面 11](#_Toc43654777)

[4.2、数据库的创建 13](#_Toc43654778)

[4.3、头文件简易介绍 14](#_Toc43654779)

[4.4、售票端功能设计 14](#_Toc43654780)

[4.5、购票端、服务器简单介绍 27](#_Toc43654781)

[五、实训结果 29](#_Toc43654782)

[六、实训过程中出现的问题及解决方案 33](#_Toc43654783)

[七、实训总结 43](#_Toc43654784)

# 一、实训目的

## 1.1、需求分析

基于Qt和linux的简易飞机票售票系统模拟系统

1. 三类角色：服务器端、客户端、售票员端；
2. 网络通信使用C语言完成，体现进程通信、线程同步，数据库等知识的使用；
3. 界面部分使用QT编写。

## 1.2、实验要求

1. 熟练的掌握QT的常用类以及控件。
2. 熟练应用信号与槽机制。
3. 对于QT的三种窗口QMainWindow、QWidget、QDialog掌握。
4. 对于各种控件的应用以及适合哪一种情况下要做的能够掌握。
5. 客户机与服务器之间的关系，如何搭建一个服务器对多个客户机。

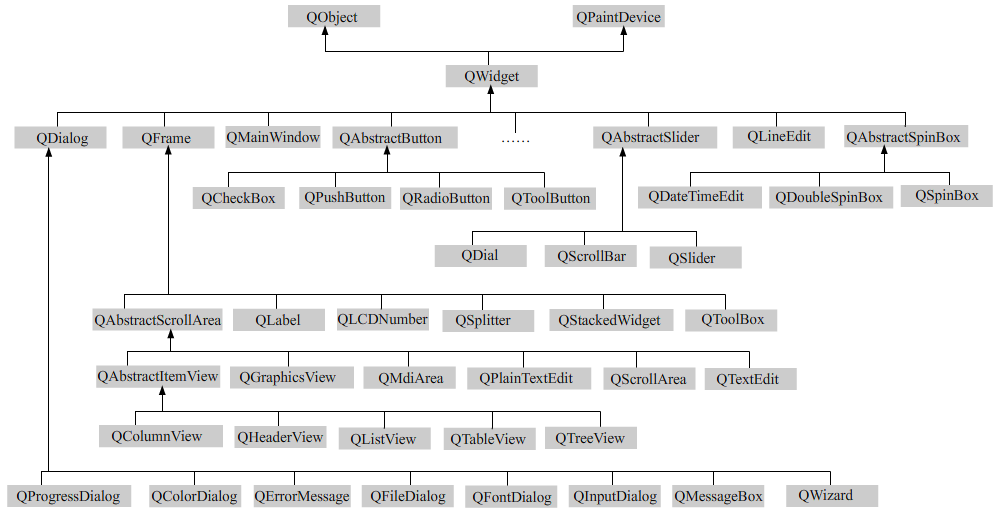
## 1.3、实验背景

本实验采用以春运时间段的飞机售票系统，添加了相关春运时的背景图来增添售票系统的丰富性。使用QT与linux c制作机票售票系统，该售票系统完成了后台服务器管理、管理员售票与普通用户购票的交互。在管理员模式下，可以进行机票的增添、删除、代理购票等操作，在普通用户权限下，用户可以进行买票、查询等功能。

# 二、实训原理

## 2.1、用到的类

Qt Creator提供的默认基类只有 QMainWindow， QWidget和 QDialog这3种。这3种窗体也是以后用得最多的，QMainWindow是带有菜单栏和工具栏的主窗口类， DIalog是各种对话框的基类，而它们全部继承自 Q Widget。不仅如此，其实所有的窗口部件都继承自 QWidget。类关系图如图2.1所示：



**图2.1 类关系图**

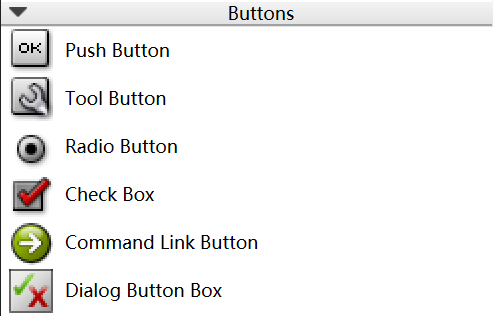
## 2.2、用到的控件

控件可以接收用户输入、显示数据和状态信息，并且在屏幕上绘制。有些也可以作为一个容器来放置其他部件。Qt中把没有嵌入到其他部件中的部件称为窗口，一般窗口都有边框和标题栏，就像程序中的widget和label一样。

在此次售票系统开发过程中，所有的界面都是以各类控件作为基本元素来开发的，例如欢迎界面用到的控件有： pushButton（按钮）、label（标签）；机票显示界面用到的控件有：label（标签）、lineEdit、pushButton、tableWidget。

**2.2.1、按钮组**

按钮组（Buttons）如图2.2所示：



**图2.2 按钮组**

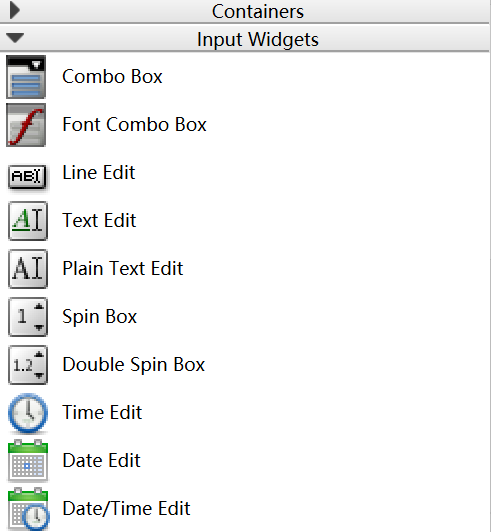
按钮组（Buttons）中各个按钮的名称依次解释如下：

Push Button：按钮。

Check Box：复选框。

**2.2.2、输入部件组**

输入部件允许你输入文字等内容，在此次开发的机票售票系统中用的最多的是textEdit部件和spin box部件。输入部件组中各个部件如图2.3所示，



**图2.3 输入部件组**

组中各个部件的名称依次解释如下：

 Combo Box：组合框。

 Text Edit：文本编辑。

 Spin Box：数字显示框（自旋盒）。

 Time Edit：时间编辑。

 Date Edit：日期编辑。

 Date/Time Edit：日期/时间编辑。

 Horizontal Scroll Bar：横向滚动条。

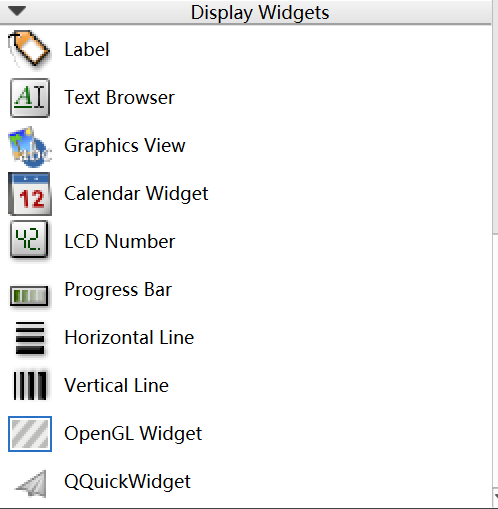
 Vertical Scroll Bar：垂直滚动条。

 Horizontal Slider：横向滑块。

 Vertical Slider：垂直滑块。

**2.2.3、显示控件组**

显示控件组用于显示相关文字信息，使开发者以及其他用户看到界面后能够一目了然的知道界面其他部件的功能。在此次售票系统开发过程中，有很多界面都用到了label部件。显示控件组包含的控件如图2.4所示：



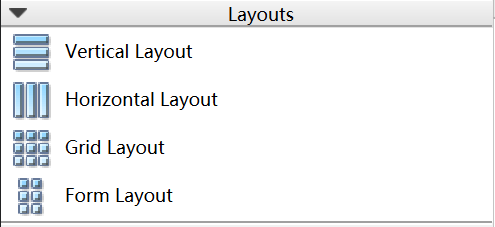
**图2.4 显示控件组**

显示控件组（Display Widgets）中几个重要的控件名称依次解释如下。

1. Label：标签。
2. Text Browser：文本浏览器。
3. Graphics View：图形视图。
4. Calendar：日历。

**2.2.4、布局管理组**

布局管理组使页面中各个部件以较为美观的方式排列。布局管理组如图2.5所示：



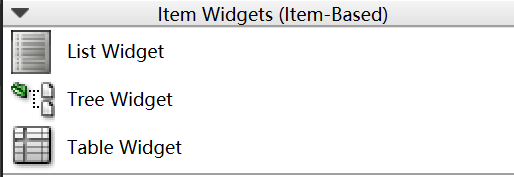
**图2.5 布局管理组**

布局管理组（Layouts）中各个控件的名称依次解释如下。

1. Vertical Layout：垂直布局。
2. Horizontal Layout：横向（水平）布局。
3. Grid Layout：网格布局。
4. Form Layout：表单布局。

**2.2.5、项目控件组**

项目控件组如图2.6所示：



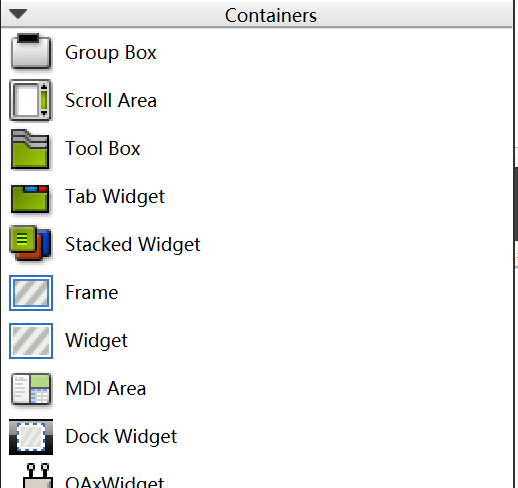
**图2.6 项目控件组**

项目控件组（Item Widgets）中各个控件的名称依次解释如下。

1. List Widget：清单控件。
2. Tree Widget：树形控件。
3. Table Widget：表控件。

**2.2.6、容器组**

在本次项目开发中，用到的最多的容器组部件为Group Box部件，其在登录界面及欢迎界面都有用到。容器组如图2.7所示：



**图2.7 容器组**

容器组（Containers）中几个重要控件的名称依次解释如下：

1. Group Box：组框
2. Scroll Area：滚动区域
3. Tool Box：工具箱
4. Tab Widget：标签小部件
5. Frame：帧
6. Widget：小部件
7. Dock Widget：停靠窗体部件
8. QAxWidget：封装Flash的ActiveX控件

## 2.3、图形视图框架

Graphics View框架结构的主要特点如下：

（1）Graphics View框架结构中，系统可以利用Qt绘图系统的反锯齿、OpenGL工具来改善绘图性能。

（2）Graphics View支持事件传播体系结构，可以使图元在场景（scene）中的交互能力提高1倍，图元能够处理键盘事件和鼠标事件。其中，鼠标事件包括鼠标按下、移动、释放和双击，还可以跟踪鼠标的移动。

（3）在Graphics View框架中，通过二元空间划分树（Binary Space Partitioning，BSP）提供快速的图元查找，这样就能够实时地显示包含上百万个图元的大场景。

**2.3.1、场景类**

场景类主要完成的工作包括提供对它包含的图元的操作接口和传递事件、管理各个图元的状态（如选择和焦点处理）、提供无变换的绘制功能（如打印）等。

事件传播体系结构将场景事件发送给图元，同时也管理图元之间的事件传播。如果场景接收到了在某一点的鼠标单击事件，场景会将事件传给在这一点的图元。

**2.3.2、视图类**

它提供一个可视的窗口，用于显示场景中的图元。在同一个场景中可以有多个视图，也可以为相同的数据集提供几种不同的视图。

QGraphicsView是可滚动的窗口部件，可以提供滚动条来浏览大的场景。如果需要使用OpenGL，则可以使用QGraphicsView::setViewport()将视图设置为QGLWidget。

**2.3.3、图元类**

QGraphicsItem主要有以下几点功能：

 1）处理鼠标按下、移动、释放、双击、悬停、滚轮和右键菜单事件。

 2）处理键盘输入事件。

 3）处理拖曳事件。

 4）分组。

 5）碰撞检测。

## 2.4、信号与槽

本次项目开发中，所有的界面之间的转换及各个按钮的动作都是通过信号与槽机制来完成的，可以说信号与槽是本次项目开发的核心中的核心。

QT中使用信号与槽机制来完成对象之间的协同操作。发射一个信号使用emit关键字，声明一个信号使用signals关键字；槽可以和很多信号相关联，其在信号发射后被调用来响应这个信号。槽可以有参数，但槽的参数不能有缺省值。槽函数分为三种类型，即 public slots 、 private slots 和 protected slots 。

实现信号与槽的连接方式有三种：

（1）在.ui设计界面中，在其中的Signals & Slots Editor选项框中，对signal和slot进行关联；

（2）使用connect()（手动关联）函数实现信号与槽的关联；

（3）在 .ui界面，右键点击某个能发出信号的部件（比如PushButton），选择Go to Slot（转到槽），即可转到一个名为 on\_btnClear\_clicked()的槽进行具体响应程序的编写。

## 2.5、线程同步

**2.5.1、线程基础介绍**

在线程里有这么一把锁——互斥锁，互斥锁是一种简单的加锁的方法来控制对共享资源的访问，互斥锁只有两种状态,即上锁和解锁。

【互斥锁的特点】

1. 原子性：把一个互斥量锁定为一个原子操作，这意味着操作系统(或pthread函数库)保证了如果一个线程锁定了一个互斥量，没有其他线程在同一时间可以成功锁定这个互斥量；
2. 唯一性：如果一个线程锁定了一个互斥量，在它解除锁定之前，没有其他线程可以锁定这个互斥量；
3. 非繁忙等待：如果一个线程已经锁定了一个互斥量，第二个线程又试图去锁定这个互斥量，则第二个线程将被挂起(不占用任何cpu资源)，直到第一个线程解除对这个互斥量的锁定为止，第二个线程则被唤醒并继续执行，同时锁定这个互斥量。

【互斥锁的操作流程】

1. 在访问共享资源后临界区域前，对互斥锁进行加锁；
2. 在访问完成后释放互斥锁导上的锁；
3. 对互斥锁进行加锁后，任何其他试图再次对互斥锁加锁的线程将会被阻塞，直到锁被释放。

**2.5.2、实验线程内容介绍**

实现线程的互斥与同步常使用的类有QMutex、QMutexLocker、QReadWriteLocker、QReadLocker、QWriteLocker、QSemaphore和QWaitCondition。

虽然类Key产生主键的函数creatKey()只有一条语句执行修改成员变量key的值，但是C++的“++”操作符并不是原子操作，通常编译后它将被展开成为以下三条机器命令：

1.）将变量值载入寄存器。

2）将寄存器中的值加1。

3）将寄存器中的值写回主存。

**2.5.3、线程函数介绍**

（1）创建互斥锁

静态方式：POSIX定义了一个宏PTHREAD\_MUTEX\_INITIALIZER来静态初始化互斥锁，方法如下：

|  |
| --- |
| pthread\_mutex\_t mutex=PTHREAD\_MUTEX\_INITIALIZER; |

动态方式：采用pthread\_mutex\_init()函数来初始化互斥锁。API定义如下：

|  |
| --- |
| int pthread\_mutex\_init(pthread\_mutex\_t\*mutex,const pthread\_mutexattr\_t \*mutexattr) |

（2）互斥锁操作

锁操作主要包括加锁pthread\_mutex\_lock()、解锁pthread\_mutex\_unlock()这两个，不论哪种类型的锁，都不可能被两个不同的线程同时得到，而必须等待解锁。对于普通锁和适应锁类型，解锁者可以是同进程内任何线程；而检错锁则必须由加锁者解锁才有效，否则返回EPERM；对于嵌套锁，文档和实现要求必须由加锁者解锁，但实验结果表明并没有这种限制，这个不同目前还没有得到解释。在同一进程中的线程，如果加锁后没有解锁，则任何其他线程都无法再获得锁。

|  |
| --- |
| 加锁：int pthread\_mutex\_lock(pthread\_mutex\_t \*mutex)  解锁：int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex) |

## 2.6、进程通信

【售票端程序】

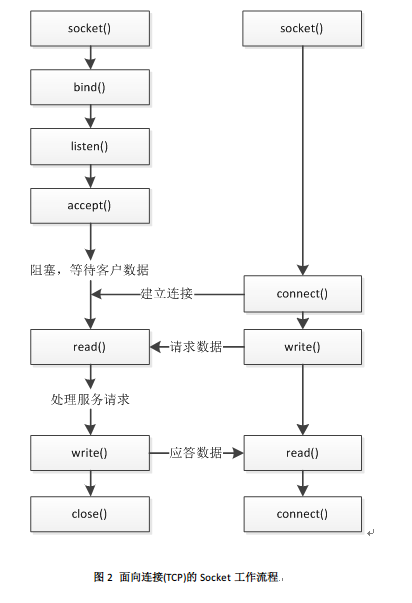
1. 创建套接字(socket)；
2. 设置接收、发送超时值；
3. 填写服务器的地址信息
4. 连接服务器，收发信息
5. 关闭套接字。

【服务器端程序】

1. 创建套接字(socket)；
2. 将套接字绑定到一个本地地址和端口上(bind)——本地计算机可能有多个IP，每一个IP有多个端口号，需要指定一个IP和端口进行监听；
3. 将套接字设为监听模式，准备接受客户请求(listen)；
4. 等待客户请求到来；当请求到来后，接受连接请求，返回一个新的对应于此次连接的套接字(accept)；
5. 用返回的套接字和客户端进行通信(send/recv)；
6. 关闭套接字。

【客户端程序】

1. 创建套接字(socket)；
2. 向服务器发出连接请求(connect)；
3. 和服务器端进行通信(send/recv)；
4. 关闭套接字。



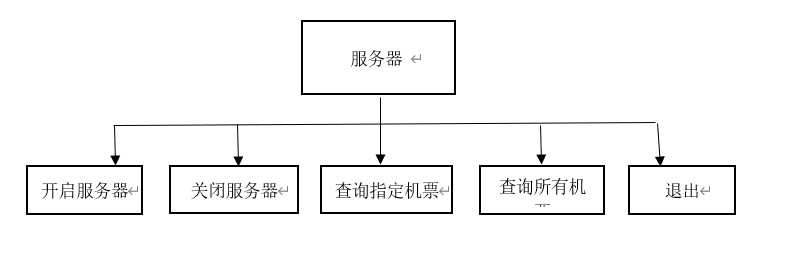
**图2.8 面向连接的Socket工作流程图**

# 三、实训设计思路

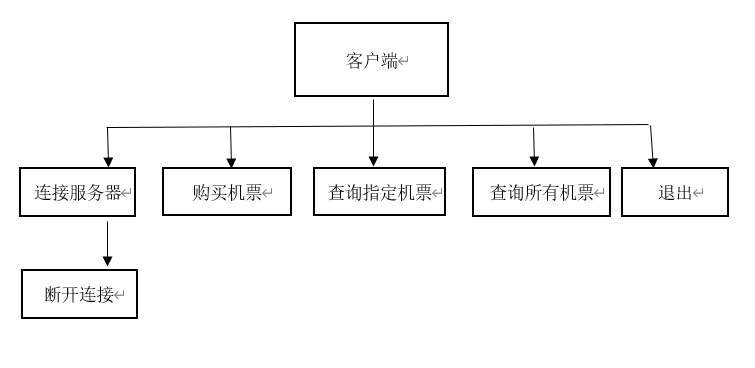
## 3.1、总体设计

1. 服务器端：开启、关闭服务器，单一、全部查询等
2. 客户购票端：连接、断开服务器，单一、全部查询，机票购买等
3. 管理员售票端：连接、断开服务器，单一、全部查询，机票购买、代理购票、增加、删除、退订等

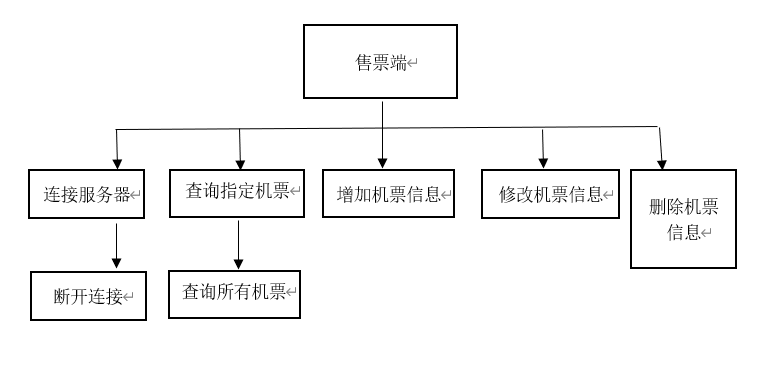
## 3.2、设计流程图



**图3.1 服务器流程图**



**图3.2 客户机流程图**



**图3.3 售票端流程图**

## 3.3、登陆与注册

1. 创建数据库，将数据存放在数据库中；
2. 登陆时进行循环查询操作，观察匹配情况进行登陆操作；
3. 注册时进行数据库插入操作。

## 3.4、服务器框架

使用多线程技术

主线程使用倾听套接字，接收新连接，当新连接接入后，主线程为新连接创建一个服务线程，并为新的服务线程分配一个线程缓冲区，将服务线程使用需要的信息保存在这个线程缓冲区中。

新的服务线程被创建后，将使用新连接接入后的连接套接字，来接受客户进程的请求报文，处理后并发送回应报文。

## 3.5、客户机框架

（客户机包括购票端以及售票端）

售票员可在不同的终端中开启多个终端进程；

线程获取售票员的输入，将信息组织成消息结构，发送给服务器，等待服务器响应；

若服务器在指定的时间没有送回处理结果，则客户机认为该连接已不可用，给出提示信息；否则，终端进程将服务器发送的回应信息进行数据接收并显示处理结果。

## 3.6、售票端功能详述

（1）开启服务器

在服务器端开启后，套接字处于监听状态，可接收来自客户端的请求；

在客户机端开启后，连接套接字，获取数据。

（2）请求购买机票

当指定航班的剩余票数小于请求票数，则回应记录中将填写该次剩余的票数和机票单价，同时消息类型为购票失败；

当指定航班的剩余票数大于或等于请求票数，则回应记录中填写购得的本次航班的票数和机票价钱总和，同时消息类型为购票成功。

（3）查询航班

单一航班查询，输入查询航班号，则回应报文中填写该航班的剩余票数和机票的单价，同时消息类型为查询成功；

所有航班查询，则回应报文中填写所有航班的剩余票数和机票单价，同时消息类型为查询成功。

（4）增加、删除、代购等

售票端通过模态产生新界面来对航班进行增删、代购等操作，通过调用函数来修改数据库内容，从而达到信息同步的功能设计。

（5）购票支付

通过请求购票命令后，购票成功，在信息显示下方存在一个购票支付选择页面，用于给客户方进行支付购票。

# 四、实训内容

## 4.1、设计ui界面

Ui界面设计与使用时界面大致相同，此处暂时略过，其他ui界面后面展示



**图4.1 登录ui界面**



**图4.2 注册ui界面**

****

**图4.3 售票ui界面**



**图4.4 支付界面**

## 4.2、数据库的创建

1. 创建数据库头文件

QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");

db.setDatabaseName("my");

if (!db.open()) {

QMessageBox::critical(0, "Cannot open database",

"Unable to establish a database connection.", QMessageBox::Cancel);

return false;

}

1. 添加数据

QSqlQuery query;

query.exec("create table stu (username varchar(20) primary key, password varchar(20))");

query.exec("insert into stu(username, password) values('q', '123')");

return true;

1. 查询

query.exec("select \* from stu");

while(query.next()){ }

1. 插入

QSqlQuery query;

QString sql = "insert into stu(username, password) values('" + username + "', '" + password + "')";

query.exec(sql);

## 4.3、头文件简易介绍

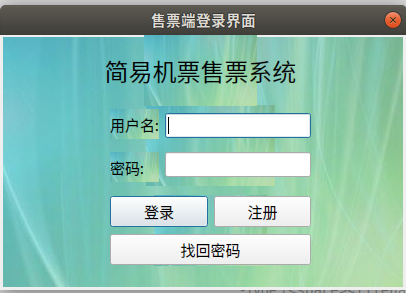
1. Servicethread.h：
2. 服务线程错误处理函数——多线程技术，简单的exit函数无法妥善处理，必须将使用的资源释放后调用pthread\_exit()退出，并且在需要进行线程同步的情况下，还需要做一些线程同步才能退出。
3. 线程函数——获取数据记录，根据数据中的消息类型进行相对应的处理，处理完毕后发送给客户端，等待客户端请求
4. Globalapi.h：
5. 函数库的头文件
6. 客户端与服务器端通信使用的消息格式
7. 通信前信息初始化函数
8. 服务器端的线程缓冲区的最大数量——可连接的客户机的最大数量
9. 有关于信息缓冲区的相关函数——初始化、分配、互斥锁防干扰、添加、释放
10. Ticket.h：
11. 简单定义机票的结构
12. 对读取数据库的函数声明与定义
13. 结构的初始化
14. Threadbuff.h：
15. 线程缓冲区的结构、使用状态
16. 定义用于线程缓冲区使用的互斥锁——主线程分配缓冲区时要检测使用状态，服务线程退出前改变使用状态，完成线程缓冲区的释放
17. 有关于线程缓冲区的相关函数——初始化、分配、释放、检测(检查线程缓冲区是否重复连接)

## 4.4、售票端功能设计

**4.4.1、登录界面**

登录界面主要由3个label部件、2个lineEdit部件和3个pushButton部件组成，它主要完成用户的登录。用户注册后，系统会将其注册的账号及密码保存在库里，用户登录时，系统将自动匹配用户名及密码，若匹配成功，则可正常登录系统，否则系统会出现账号或密码错误的对话框。

登录界面效果图如图所示：



**图4.5 登录界面**

登录界面实现的核心代码如下：

#include "login.h"

#include "ui\_login.h"

#include<QMessageBox>

Login::Login(QWidget \*parent) :

QDialog(parent),

ui(new Ui::Login)

{

ui->setupUi(this);

flag = 0;

}

Login::~Login()

{

delete ui;

}

//售票登录设计

void Login::on\_pushButton\_clicked()

{

flag = 1;

if((ui->lineEdit->text()=="lsh"&&ui->lineEdit\_2->text()=="12345")||\

(ui->lineEdit->text()=="cfc"&&ui->lineEdit\_2->text()=="12345")||\

(ui->lineEdit->text()=="lxm"&&ui->lineEdit\_2->text()=="12345")){

accept();

}

else{

QMessageBox::warning(this,tr("warning"),tr("user name or password error!"),QMessageBox::Yes);

}

}

void Login::on\_pushButton\_2\_clicked()

{

flag = 2;

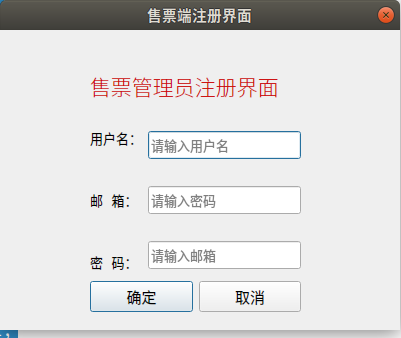
accept();

}

**4.4.2、注册界面**

注册界面主要由4个label部件、3个lineEdit部件、1个pushButton、1个buttonBox、一个groupBox（组容器）组成。它主要完成普通用户的注册，将注册后的信息保存到库里。

注册界面效果图如图所示



**图4.6 注册界面**

注册界面实现的核心代码如下：

#include "mainwindow.h"

#include "login.h"

#include <QApplication>

#include "login\_reg.h"

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

Login d1;

login\_reg d2;

d1.setWindowTitle("售票端登录界面");

d2.setWindowTitle("售票端注册界面");

w.setWindowTitle("售票端界面");

if(d1.exec()==QDialog::Accepted)

{

switch (d1.flag)

{

case 1:

w.show();

break;

case 2:

d2.show();

break;

default:

break;

}

}

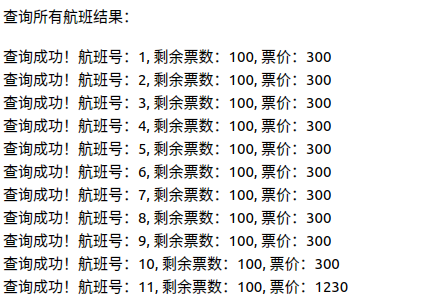
return a.exec();

}

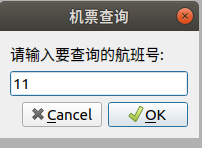
**4.4.3、机票查询界面**

查询界面主要由4个label部件、2个lineEdit部件、3个pushButton部件、1个dateEdit部件组成。在该界面，用户可以查询所需要的机票，输入出发地以及目的地，即可转入机票显示界面，显示符合其要求的所有车次。

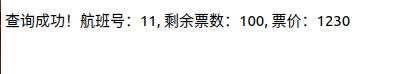
查询全部机票



**图4.7 查询所有航班界面图**



**图4.8 查询单独航班界面图**



**图4.9 查询单独航班成功图**

核心代码

void MainWindow::on\_action\_inquireone\_triggered(){

//ui->textBrowser->append("Inquire One\n");

// char\* ticket\_num\_temp = nullptr;

QDialog dialog(this);

QFormLayout form(&dialog);

dialog.setWindowTitle("机票查询");

QList<QLineEdit \*> fields;

QLineEdit \*ord = new QLineEdit(&dialog);

// update\_ticket\_number();

// sprintf(ticket\_num\_temp, "请输入要查询的航班号(1-%d):", numRows);

// form.addRow(new QLabel(QString(ticket\_num\_temp)));

form.addRow(new QLabel("请输入要查询的航班号:"));

form.addRow(ord);

fields << ord;

QDialogButtonBox buttonBox(QDialogButtonBox::Ok | QDialogButtonBox::Cancel, Qt::Horizontal, &dialog);

form.addRow(&buttonBox);

QObject::connect(&buttonBox, SIGNAL(accepted()), &dialog, SLOT(accept()));

QObject::connect(&buttonBox, SIGNAL(rejected()), &dialog, SLOT(reject()));

/\* 点击确认按钮 \*/

if (dialog.exec() == QDialog::Accepted) {

QString flight\_ord = ord->text();

/\* 获取输入的航班号 \*/

char msg[512];

char send\_buf[512],recv\_buf[512];

int flight\_ID = flight\_ord.toInt();

update\_ticket\_number();

if(flight\_ID<=0 || flight\_ID>numRows) { // 判断输入的航班号是否正确，不正确的话，给出提示信息，重新输入。

display\_info("输入的航班号错误！请重新输入！");

return;

}

init\_message(); /\* 将消息数据类型进行初始化 \*/

message.msg\_type=INQUIRE\_ONE;

message.flight\_ID=flight\_ID;

memcpy(send\_buf,&message,sizeof(message));

int ret=send(socket\_fd, send\_buf,sizeof(message),0);

/\* 发送出错 \*/

if(ret==-1) {

display\_info("发送失败！请重新发送！");

return ;

}

ret=recv(socket\_fd,recv\_buf,sizeof(message),0);

if(ret==-1) {

display\_info("接收失败！请重新连接服务器！\n");

return ;

}

memcpy(&message,recv\_buf,sizeof(message));

if(message.msg\_type==INQUIRE\_SUCCEED){

sprintf(msg,"查询成功！航班号：%d, 剩余票数：%d, 票价：%d\n",message.flight\_ID,message.ticket\_num, message.ticket\_total\_price);

}

else{

sprintf(msg,"查询失败！航班号：%d, 剩余票数：未知\n",message.flight\_ID);

}

display\_info(msg);

}

}

void MainWindow::on\_action\_inquireall\_triggered(){

//ui->textBrowser->append("Inquire All\n");

int i,pos;

char msg[512];

char send\_buf[1024], recv\_buf[1024];

init\_message();

message.msg\_type=INQUIRE\_ALL;

memcpy(send\_buf,&message,sizeof(message));

int ret=send(socket\_fd, send\_buf,sizeof(message),0);

/\* 发送出错 \*/

if(ret==-1) {

display\_info("发送失败！请重新发送！");

return;

}

ret=recv(socket\_fd,recv\_buf,sizeof(recv\_buf),0);

if(ret==-1) {

display\_info("接收失败！请重新发送！");

return;

}

pos=0;

sprintf(msg,"查询所有航班结果：\n");

display\_info(msg);

for (i=0;i<ret;i=i+sizeof(message)) {

memcpy(&message,recv\_buf+pos,sizeof(message));

if(message.msg\_type==INQUIRE\_SUCCEED){

sprintf(msg,"查询成功！航班号：%d, 剩余票数：%d, 票价：%d",message.flight\_ID,message.ticket\_num, message.ticket\_total\_price);

}

else{

sprintf(msg,"查询失败！航班号：%d, 剩余票数：未知",message.flight\_ID);

}

display\_info(msg);

pos+=sizeof(message);

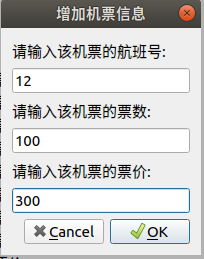
}

display\_info("\n");

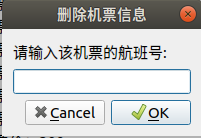
}

**4.4.5、售票功能设计**

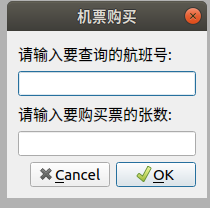
售票端在客户端基础上添加了购买、增加、删除、退订、代理购买机票等

****

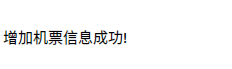
**图4.10 增加航班界面**



**图4.11 删除航班界面**



**图4.12 购票界面**



**图4.13 添加航班成功图**

核心代码

void MainWindow::on\_action\_buyticket\_triggered(){

ui->textBrowser->append("Buy Ticket....\n");

// char\* ticket\_num\_temp = nullptr;

QDialog dialog(this);

//布局:表单形式管理界面组件

QFormLayout form(&dialog);

dialog.setWindowTitle("机票购买");

QList<QLineEdit \*> fields;

QLineEdit \*ord = new QLineEdit(&dialog);

//addRow()添加表单项

form.addRow(new QLabel("请输入要查询的航班号:"));

form.addRow(ord);

QLineEdit \*cnt = new QLineEdit(&dialog);

form.addRow(new QLabel("请输入要购买票的张数:"));

form.addRow(cnt);

fields << ord;

fields << cnt;

QDialogButtonBox buttonBox(QDialogButtonBox::Ok | QDialogButtonBox::Cancel, Qt::Horizontal, &dialog);

form.addRow(&buttonBox);

QObject::connect(&buttonBox, SIGNAL(accepted()), &dialog, SLOT(accept()));

QObject::connect(&buttonBox, SIGNAL(rejected()), &dialog, SLOT(reject()));

/\* 点击确认按钮 \*/

if (dialog.exec() == QDialog::Accepted) {

QString flight\_ord = ord->text();

QString flight\_cnt = cnt->text();

/\* 获取输入的航班号 \*/

char msg[512];

char send\_buf[512],recv\_buf[512];

int flight\_ID = flight\_ord.toInt();

int ticket\_num= flight\_cnt.toInt();

update\_ticket\_number();

/\* 判断输入的航班号是否正确，不正确的话，给出提示信息重新输入 \*/

if(flight\_ID<=0 || flight\_ID>numRows) {

display\_info("输入的航班号错误！请重新输入！");

return;

}

/\* 判断输入的票数是否正确，不正确的话，给出提示信息，重新输入 \*/

if(ticket\_num<=0) {

display\_info("输入的票数错误！请重新输入！");

return;

}

/\* 购买机票 \*/

init\_message();

message.msg\_type=BUY\_TICKET;

message.flight\_ID=flight\_ID;

message.ticket\_num=ticket\_num;

memcpy(send\_buf,&message,sizeof(message));

int ret=send(socket\_fd, send\_buf,sizeof(message),0);

/\* 发送出错 \*/

if(ret == -1) {

display\_info("发送失败！请重新发送！");

return ;

}

ret = recv(socket\_fd,recv\_buf,sizeof(message),0);

if(ret==-1) {

display\_info("接收失败！请重新发送！");

return ;

}

memcpy(&message,recv\_buf,sizeof(message));

if(message.msg\_type==BUY\_SUCCEED){

sprintf(msg,"购买成功！航班号：%d, 票数：%d, 总票价：%d\n",message.flight\_ID,message.ticket\_num, message.ticket\_total\_price);

}

else{

sprintf(msg,"购买失败！航班号：%d, 剩余票数：%d, 请求票数：%d\n",message.flight\_ID,message.ticket\_num,ticket\_num);

}

display\_info(msg);

}

}

void MainWindow::on\_action\_add\_triggered()

{

//ui->textBrowser->append("Add Information...");

QDialog dialog(this);

QFormLayout form(&dialog);

dialog.setWindowTitle("增加机票信息");

QList<QLineEdit \*> fields;

QLineEdit \*id = new QLineEdit(&dialog);

form.addRow(new QLabel("请输入该机票的航班号:"));

form.addRow(id);

QLineEdit \*number = new QLineEdit(&dialog);

form.addRow(new QLabel("请输入该机票的票数:"));

form.addRow(number);

QLineEdit \*price = new QLineEdit(&dialog);

form.addRow(new QLabel("请输入该机票的票价:"));

form.addRow(price);

fields << id;

fields << number;

fields << price;

QDialogButtonBox buttonBox(QDialogButtonBox::Ok | QDialogButtonBox::Cancel, Qt::Horizontal, &dialog);

form.addRow(&buttonBox);

QObject::connect(&buttonBox, SIGNAL(accepted()), &dialog, SLOT(accept()));

QObject::connect(&buttonBox, SIGNAL(rejected()), &dialog, SLOT(reject()));

if (dialog.exec() == QDialog::Accepted) {

int flight\_ID = id->text().toInt();

int ticket\_num = number->text().toInt();

int ticket\_price = price->text().toInt();

char send\_buf[1024];

init\_message();

message.msg\_type=ADD\_TICKET;

message.flight\_ID = flight\_ID;

message.ticket\_num = ticket\_num;

message.ticket\_total\_price = ticket\_price;

memcpy(send\_buf,&message,sizeof(message));

int ret=send(socket\_fd, send\_buf,sizeof(message),0);

/\* 发送出错 \*/

if(ret==-1) {

display\_info("发送失败！请重新发送！");

return;

}

display\_info("增加机票信息成功!\n");

}

}

void MainWindow::on\_action\_update\_triggered()

{

//ui->textBrowser->append("Update Information...");

QDialog dialog(this);

QFormLayout form(&dialog);

dialog.setWindowTitle("修改机票信息");

QList<QLineEdit \*> fields;

QLineEdit \*id = new QLineEdit(&dialog);

form.addRow(new QLabel("请输入该机票的航班号:"));

form.addRow(id);

QLineEdit \*number = new QLineEdit(&dialog);

form.addRow(new QLabel("请输入该机票的票数:"));

form.addRow(number);

QLineEdit \*price = new QLineEdit(&dialog);

form.addRow(new QLabel("请输入该机票的票价:"));

form.addRow(price);

fields << id;

fields << number;

fields << price;

QDialogButtonBox buttonBox(QDialogButtonBox::Ok | QDialogButtonBox::Cancel, Qt::Horizontal, &dialog);

form.addRow(&buttonBox);

QObject::connect(&buttonBox, SIGNAL(accepted()), &dialog, SLOT(accept()));

QObject::connect(&buttonBox, SIGNAL(rejected()), &dialog, SLOT(reject()));

if (dialog.exec() == QDialog::Accepted) {

int flight\_ID = id->text().toInt();

int ticket\_num = number->text().toInt();

int ticket\_price = price->text().toInt();

char send\_buf[1024];

init\_message();

message.msg\_type=UPDATE\_TICKET;

message.flight\_ID = flight\_ID;

message.ticket\_num = ticket\_num;

message.ticket\_total\_price = ticket\_price;

memcpy(send\_buf,&message,sizeof(message));

int ret=send(socket\_fd, send\_buf,sizeof(message),0);

/\* 发送出错 \*/

if(ret==-1) {

display\_info("发送失败！请重新发送！");

return;

}

display\_info("修改机票信息成功!\n");

}

}

void MainWindow::on\_action\_delete\_triggered()

{

//ui->textBrowser->append("Delete Information...");

QDialog dialog(this);

QFormLayout form(&dialog);

dialog.setWindowTitle("删除机票信息");

QList<QLineEdit \*> fields;

QLineEdit \*id = new QLineEdit(&dialog);

form.addRow(new QLabel("请输入该机票的航班号:"));

form.addRow(id);

fields << id;

QDialogButtonBox buttonBox(QDialogButtonBox::Ok | QDialogButtonBox::Cancel, Qt::Horizontal, &dialog);

form.addRow(&buttonBox);

QObject::connect(&buttonBox, SIGNAL(accepted()), &dialog, SLOT(accept()));

QObject::connect(&buttonBox, SIGNAL(rejected()), &dialog, SLOT(reject()));

if (dialog.exec() == QDialog::Accepted) {

int flight\_ID = id->text().toInt();

char send\_buf[1024];

init\_message();

message.msg\_type=DELETE\_TICKET;

message.flight\_ID = flight\_ID;

memcpy(send\_buf,&message,sizeof(message));

int ret=send(socket\_fd, send\_buf,sizeof(message),0);

/\* 发送出错 \*/

if(ret==-1) {

display\_info("发送失败！请重新发送！");

return;

}

display\_info("删除机票信息成功!\n");

}

}

**4.4.6、支付界面**



**图4.14 支付方式界面**

核心代码：

void MainWindow::on\_pushButton\_clicked()

{

pay = new dialog\_pay;

pay->setWindowTitle("支付界面");

pay->setModal(false);

pay->show();

}



**图4.15 支付界面**

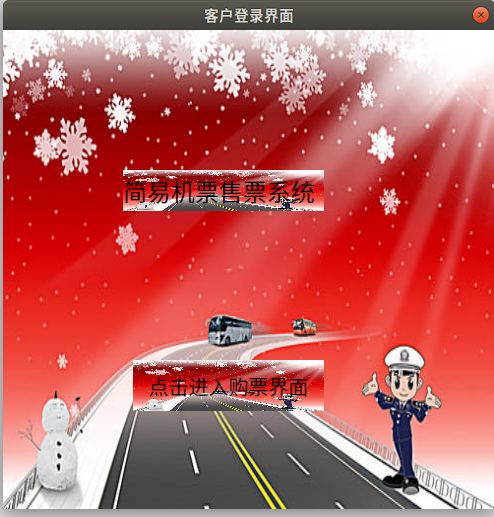
## 4.5、购票端、服务器简单介绍



**图4.16 服务器登录界面**



**图4.17 服务器界面**

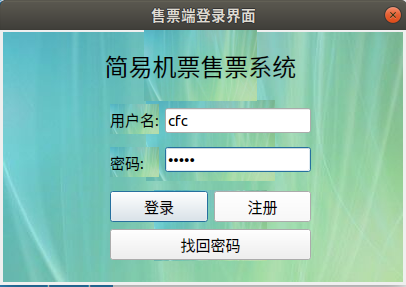


**图4.18 客户端登录界面**



**图4.19 客户端界面**

# 五、实训结果



**图5.1 售票登录界面**



**图5-2 售票界面**



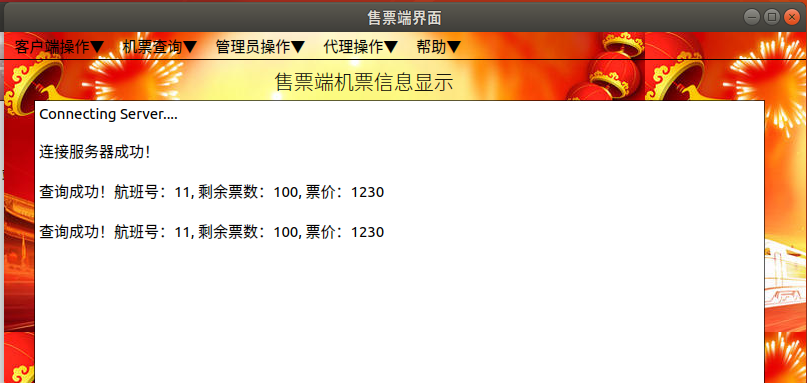
**图5.3 查询界面**



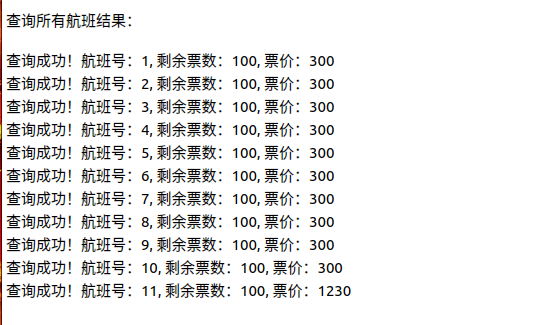
**图5.4 代理界面**



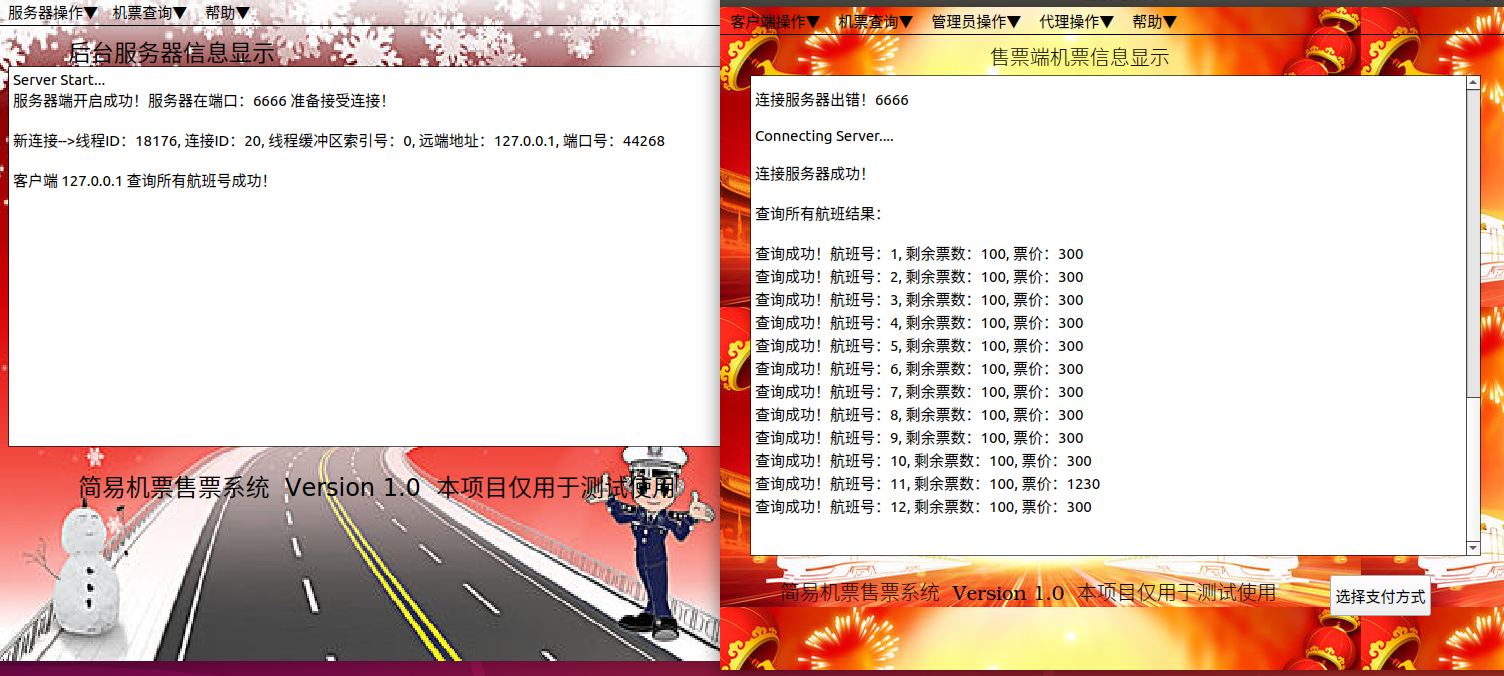
**图5.5 连接服务器成功图**



**图5.6 单独查询成功图**



**图5.7 查询所有航班成功图**



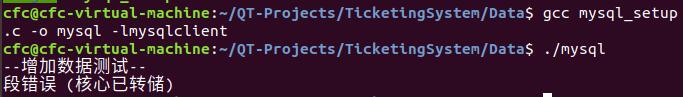
**图5.8 总框界面**



**图5.9 支付界面**

# 六、实训过程中出现的问题及解决方案

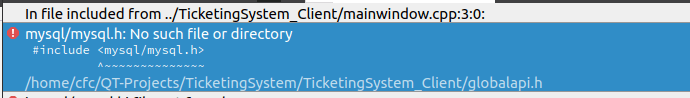
**6.1、段错误**



**图6.1 错误问题图1**

解决方法：数据库连接的信息有误，修改代码，使用正在使用中的数据库

**6.2、No such file or directory(1)**



**图6.2 错误问题图2**

解决方法：

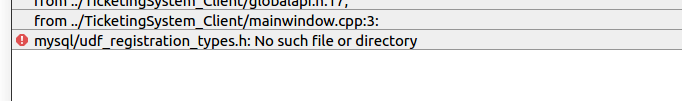
1、此处原因为缺少开发库

2、在linux c终端处进行测试，看看可以使用mysql

3、再执行一下sudo apt-get install libmysqlclient-dev下载开发库

4、然后再QT的.pro文件中，加入 LIBS+= -lmysqlclient

**6.3、No such file or directory(2)**



**图6.3 错误问题图3**

**解决方法：**



**图6.4 错误问题图4**

1、将mysql/udf\_registration\_types.h 这个头文件的尖括号<>修改为双引号””

由于此文件是只读文件，所以需要管理员权限进行修改

2、在linux终端，输入sudo su，进入root权限

3、然后一直cd，找到mysql\_com.h这个文件

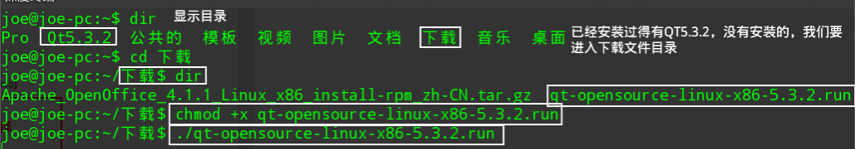
4、Vi编辑，一直下拉，由于代码过多，所以寻找比较麻烦

5、找到后将其修改保存，退出就可以了。

**6.4、Qt of linux的安装中存在的问题**

Qt of linux的安装：

1、去qt官网下载，官方的开源文件后，根据下图步骤进行安装



**图6.5 错误问题图5**

2、安装到最后一步要勾选gcc和source，这两个提供了后续qt使用所需要的资源。

因为第一次没有勾选，所以出现了问题，重新安装qt。

**6.5、运行c文件时，出现问题**



**图6.6 错误问题图6**

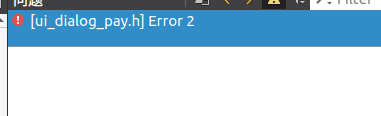
**解决方法：**

1、查看是否安装了mysql，并且是否可以运行

2、在运行编译时，添加数据库的动态库 -lmysqlclient

**6.6、出现 【ui\_dialog\_pay.h】Error 2**

qt中，出现问题，点击右键，查看详细错误信息



**图6.7 错误问题图7**

while building/deploying project TicketingSystem\_Server (kit: 桌面)

When executing step "Make"

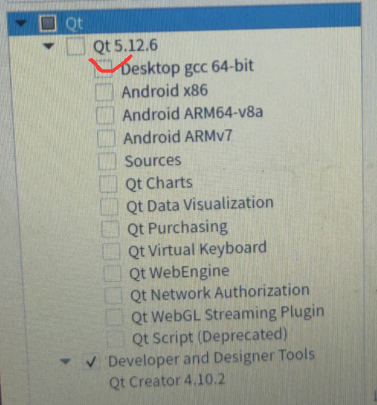
**解决方法**：项目->概要->shadow build中去掉对号

网站：

https://www.cnblogs.com/zhangjunwu/p/7417566.html

**6.7、kits中qt version中找不到qmake**

在linux下安装qt时，kits中qt version中找不到qmake  
原因：在安装过程中，少安装的一个模块



**图6.8 错误问题图8**

**6.8、QT与linux c 如何联合**

**解决方法**

将这两个联合是最困难的问题，最终解决方法是使用QT界面进行设计，在UI界面时，右击控件，转到槽函数，在槽函数里添加c函数，实现相应控件的功能。

**6.9、 父子窗口的交互**

子窗口与父窗口如何连接，使用了全局变量的方式，在主界面上声明全局变量并为全局变量赋值，在子界面上用extern声明外部变量。

**6.10、问题：**

**Sub-process /usr/bin/dpkg returned an error code (1)**  


**图6.9 错误问题图9**

解决方法：

网站  
https://blog.csdn.net/qq\_38375620/article/details/86707493  
https://blog.csdn.net/stickmangod/article/details/85316142

**6.11、在其他界面添加子界面**

1、通过按钮新建自定义窗口，新建一个对话框，qt->qt 设计师界面->选择一项界面， 起名为：dialog123 默认即可

2、在mainwindow.h中，加入dialog123的头文件 #include "dialog123.h"

在下面添加public：(private:)

dialog123 \*new\_dialog;

3、在需要按钮所在ui界面中（比如在mainwindow,h），添加按钮，右键选择【转到槽】，选择【clicked()】

在槽函数中，添加

void MainWindow::on\_pushButton\_clicked()  
{

new\_dialog = new dialog123;

new\_dialog->setWindowTitle(“ 标题”);

new\_dialog->setModal(false);

new\_dialog->show();

}

**6.12、加背景图**

注意，资源放在images文件夹中，再在resources文件中右键进行添加，在进行构建

**解决方法1：**

https://blog.csdn.net/kidults/article/details/80109738

1、先添加资源

2、在。cpp文件中，添加

#include <QPixmap>

#include <QPalette>

3、在构造函数中添加

**第二种代码**，在构造函数中添加

QPixmap pixmap = QPixmap(":/images/images/back.jpg").scaled(this->size());

QPalette palette(this->palette());

palette.setBrush(QPalette::Background, QBrush(pixmap));

this->setPalette(palette);

**解决方法2**

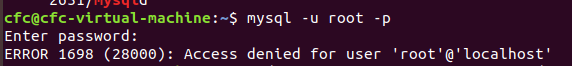
在ui界面中，进行设置，右键选择样式表->选择前缀->图片，选择apply进行查看

网站：

<https://blog.csdn.net/WU9797/article/details/78714650?tdsourcetag=s_pctim_aiomsg>

**6.13、**

**第一次使用mysql –u root –p时，可能会出现问题**



**图6.10 错误问题图10**

ERROR1698(28000):Access denied for user ‘root’@‘localhost’

**解决方法**

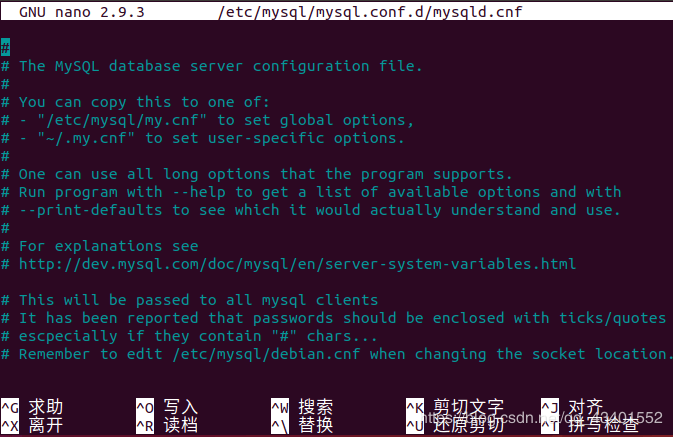
$ sudo nano /etc/mysql/mysql.conf.d/mysqld.cnf

在skip-external-locking后添加skip-grant-tables 之后

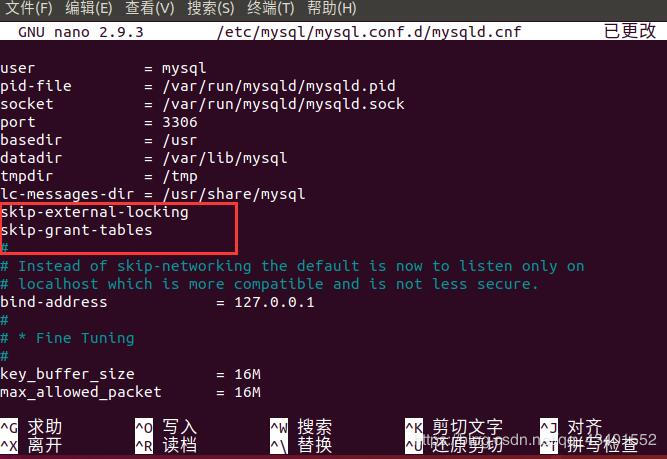
ctrl + s保存，ctrl +x退出

之后重新启动mysql

service mysql restart

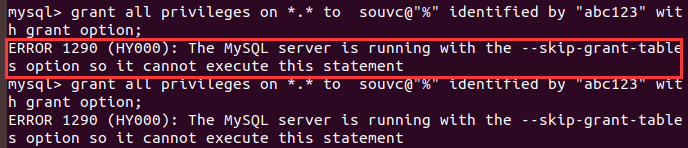


**图6.11 错误问题图11**



**图6.12 错误问题图12**

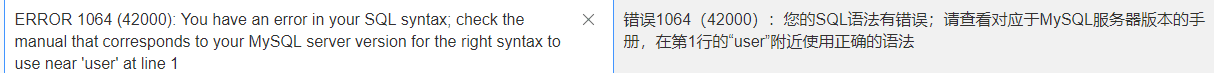
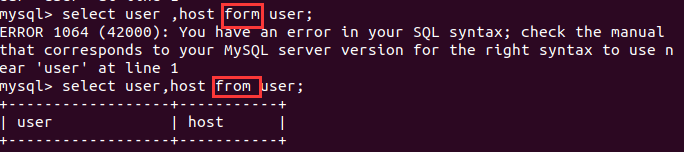
**6.14、安装5.7时，会出现的问题**

Linux：ERROR 1290 (HY000): The MySQL server is running with the --skip-grant-tables option so it cannot execute this statement  


**图6.13 错误问题图13**

解决方法：flush privileges;

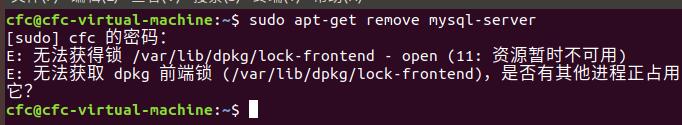
网站 https://blog.csdn.net/yongpole/article/details/82825417  
<https://www.cnblogs.com/hiit/p/11465470.html>

**6.15、**完美解决 ERROR 1064 (42000): You have an error in your SQL syntax … near …  
  


**图6.14 错误问题图14**

解决方法：存在sql语法问题

**6.16、**  
ubuntu提示E: 无法获得锁 /var/lib/dpkg/lock-frontend - open (11: 资源暂时不可用)

https://blog.csdn.net/dream\_follower/article/details/90311799  


**图6.15 错误问题图15**

原因:  
出现这个问题的原因可能是有另外一个程序正在运行，由于它在运行时，会占用软件源更新时的系统锁（以下称“系统更新锁”，此锁文件在“/var/lib/apt/lists/”目录下），而当有新的apt-get进程生成时，就会因为得不到系统更新锁而出现”E: 无法获得锁 /var/lib/apt/lists/lock - open (11: Resource temporarily unavailable)”错误提示！  
而导致资源被锁的原因，可能是上次安装时没正常完成，而导致出现此状况。  
因此，我们只要将原先的apt-get进程杀死，从新激活新的apt-get进程，就可以让软件管理器正常工作了！

解决方案：  
1.方法一：  
用这个命令查看一下apt-get的相关进程：

ps -e | grep apt  
显示：  
11669 ? 00:00:02 aptd  
25379 ? 00:00:00 update-apt-xapi

然后执行:

$sudo kill 11669  
$sudo kill 25379

但是用上面这个方法，好像没有用，所以我又使用了第二种方法，我只执行了第二句就可以了  
2.方法二：

sudo rm /var/cache/apt/archives/lock  
sudo rm /var/lib/dpkg/lock  
网站;https://blog.csdn.net/dream\_follower/article/details/90311799

**6.17、dpkg 被中断问题解决方法**

https://blog.csdn.net/qq\_14935437/article/details/77199615?utm\_source=blogxgwz3

linux系统安装软件是有时会碰到“dpkg 被中断,您必须手工运行 sudo dpkg –configure -a解决此问题”，然而按照提示运行却并没能很好的解决问题。其实导致这个问题的主要原因是因为/var/lib/dpkg/updates文件下的文件有问题，可能是其他软件安装过程或是其他原因导致的，这里删除掉然后重建即可。sudo rm /var/lib/dpkg/updates/\*

sudo dpkg --configure -a

sudo apt-get update  
sudo apt-get upgrade  
[*]sudo apt-get update指令会重新建立这些资料，所以不必担心删除后会出问题；  
[*]sudo apt-get upgrade会更新你的电脑里面已安装的软件的明细，根据软件的明细更新软件到最新版。  
PS：“dpkg ”是“Debian Packager ”的简写。为 “Debian” 专门开发的套件管理系统，方便软件的安装、更新及移除。所有源自“Debian”的“Linux ”发行版都使用 “dpkg”