"C++程序设计与训练"课程大作业
项目报告

项目名称：
简易铁路票务系统
（TicketSystem）

姓名：yfreedomliTHU
学号：12345678
班级：wu 52
日期：2016.9.12

目录

1 系统功能设计

项目需求分析

本项目是实现简易的铁路票务系统，铁路票务系统的目标是实现票务管理的信息化，自动化，安全可靠，提高票务管理的效率。铁路票务系统主要面向两类人，票务管理员和旅客。列车管理员可以通过系统对车次信息进行维护，如开通列车，变更发车时间，到达时间，途经站台等。而旅客则主要进行在线选票，代他人买票，退票等一系列操作。在具体的实现过程中需要注意良好的界面交互性以及一定的容错性。通过票务管理系统，能实现对大量数据的精确高效管理，很大程度上方便管理员和旅客。

1.1 总体功能描述

根据项目要求，结合实际生活，我们组通过讨论决定在票务系统中实现以下功能：

1. 旅客可以实现账号的注册和登录购票，管理员可以登录（由于管理员内置数据库，不能进行注册）进行车次管理；
2. 旅客根据起始车站、到达车站、出行时间等条件选择符合的车次，车次信息显示始达车站、始达时间、余票数量等等供其参考；
3. 旅客可以进行购票（限制列车发车前半小时不能购票），退票（退票会扣去一定比例的金额，只返回票价 80%）等操作；
4. 订票成功后系统会进行提示，且用户可以随时查看自己的个人车票信息；
5. 旅客可以为他人购票；
6. 旅客可以给自己的账户进行充值，买票时会扣去相应金额，如余额不足，则会提醒采用其他方式购票（在此做了简化处理，提醒后就默认以其他方式成功购票）；
7. 实现一一对应，限制一个人同一车次只能购买一张车票，且一个座位不能卖给多人；
8. 列车增加途经车站，能实现从起点不同里程不同票价；
9. 选座功能，图形界面能看到具体车次每个座位的售卖情况，并选择自己想要的空闲座位；
10. 通过数据库保存用户信息。
11. 管理员账户密码均为（admin），对于车辆信息的管理主要体现在：a.更新列车的基本信息（票价，座位数，出发时间，运行时间等）b.添加

和删除列车车次 c.添加和删除列车车站信息。

1.2 功能点说明

为了体现充分 OOP 的设计思想，采用了微缩版三层架构和工厂模式
说明一

功能类型	功能点名称	实现方式	功能点描述
数据层面：数据库支持	数据库对数据的储存	根据要求建立一系列有联系的表格，通过 SQL 语句进行数据的读写及写入	通过数据库实现对数据的储存能防止系统断电后数据丢失
逻辑层面：将对象特点进行总结，抽象出对应的 Model 类和 Logic 类	Model 类结合数据库和现实对象实现对象参数的定义（如 ModelUser）	C++编写类	实现与 Logic 类的对接，连接界面与数据的联系，体现 OOP 思想
	Logicl 类对应对应的 Model 类，主要是通过 SQL 语句实现对数据库数据进行增删改查，所有 Logicl 类 内嵌到 Factory 内中统一管理	C++编写 采用了 SQL 语句实现对数据库的操作 调用了 QT 的一些自带函数，库类	该设计使数据与界面板块完全分离，在 Model 类和界面部分不会见到 SQL 语句出现，单独模块化更有利于程序的拓展和管理
界面部分: 界面的布局以及提供数据对接的接口（在此简要概括，后面分界面具体说明特点及实现）	登录界面： 登录功能， 具有容错性，	界面的编写主要用到了 QT 的一些自带类库，以及,布局管理器，信号与槽,QGroupBox, QLable, QLineEdit ,QPushButton 等器件，调用前面的 Model 和 Logic 类，实现与数据库的对接	用户登录，如果输入密码，账户在数据库中没有，提示先注册
	注册界面 注册，一键置空		用户在此进行注册，同时提供了一键置空的功能
	管理界面： 管理车次，途经站		管理员能对车次信息和途经站台，票价等进行更改
	用户界面：		显示用户信息，同时链接子界面，实现具体的功能
	购票界面： 通过条件筛选和界面选座		个人购票或为他人购票，用户除了能根据条件筛选购票外，还可以界面选具体的座位
	充值界面： 账户充值		实现对账户的充值
	车票信息界面		用户可以通过此界面查看自己的已购车票信息

说明二

功能类型	功能点名称	实现方式 A.自己编写 C++代码 B.使用 C++标准库 C.使用第三方库 D.使用 SQL 语句	功能点描述
基本信息处理	旅客信息读写以及更改	A.C.(Qt 库)D	旅客注册信息后，构建旅客类对象，通过 c++程序写入数据库文件； 通过数据库查询功能找到旅客信息，用 qt 库<Qsqldatabase>实现数据对类对象的赋值，用类对象实现数据的读出。
	车票信息读写及删除	A.C(Qt 库).D	车票信息读写同上。 车票信息删除通过 sql 的 delete…where…语句完成车票的删除。
	列车信息删除及更新	A.D.	通过 logictrain 等自编库实现对列车信息等删除以及更新。自编库用到 sql 语句。
	车票信息简单查询	D.	Sql 语句内嵌到 logic 类库中，实现对数据库的查询功能。
	容错特性	A.B.D	旅客密码输入错误后弹出提示并返回登录页面；旅客查询车票信息错误时报空白，表示无此结果。
数据库支持	数据库读写	C(QT 库)，D.	将 sql 语句嵌入到 c++的类库中实现。
信息管理	车票信息管理	A.B.C(Qt 库).D	通过自编库 LogicUserTicket 实现，内嵌 sql 语言。可以实现对车票的读写，删除等。
	车次信息管理	A.B.C(Qt 库).D	通过自编库 LogicTrainNumber 实现，内嵌 sql 语言。可以实现对车次的读写，更新、增删等。
	用户信息管理	A.B.C(Qt 库).D	通过自编库 LogicUser 实现，内嵌 sql 语言。可以实现用户信息的读写，增加。

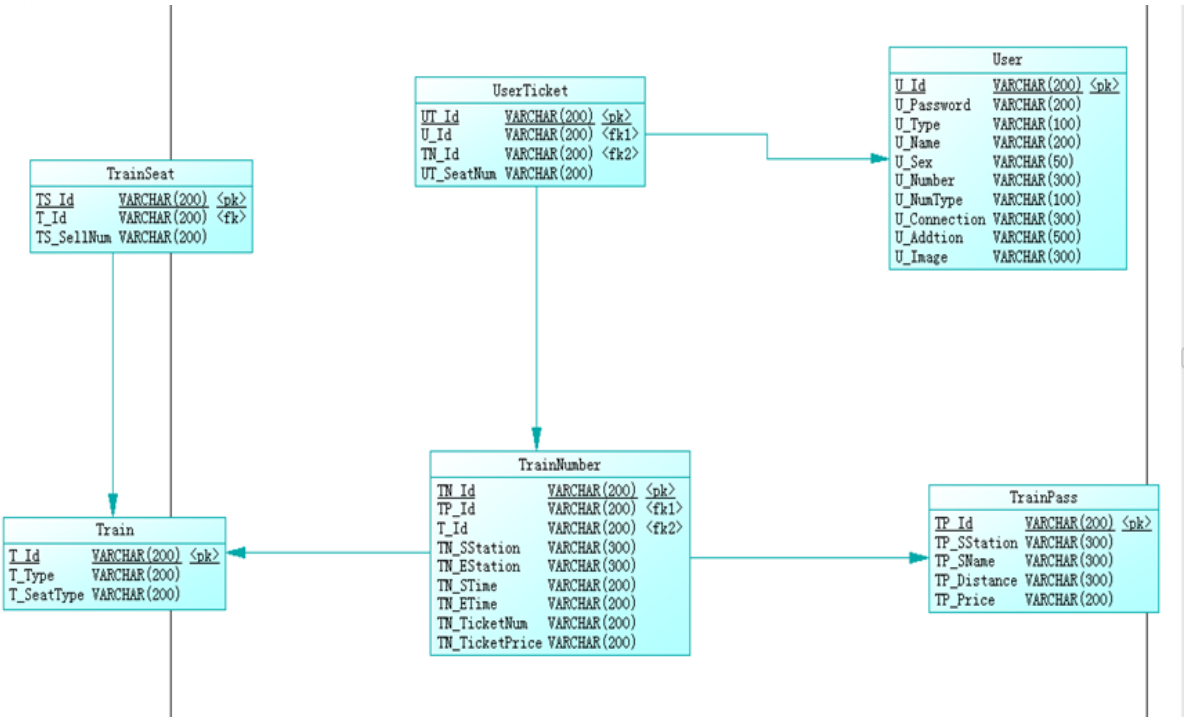
复杂查询功能	起始车站式查询	A.C.D	用数据库的联结语句实现条件查询功能。即用户输入起始站和终点站，在数据库中查询包含这两个车站的所有列车车次，并输出相关信息。
	列车车次站台显示	A.C.D	旅客搜寻到一趟列车，可以显示该列车的全部站点信息。
统计功能	车票票价计算	A.C.D	站台信息包含该站台到起始站到距离和票价，可以选择不同终点从而对应不同票价
	分别设计车次表和车站表	D.	在运行效率上，用两个表代替一个表，数据量很大时可以明显提高效率。
其它功能	列车选座功能	A.D	在购票界面，用户可以实现选座，已选座位变暗，且不可选

（具体分析会在后面进行）

2. 系统总体结构

程序架构采用了微缩版三层架构和工厂模式，分为数据库，逻辑层，以及界面三个模块。

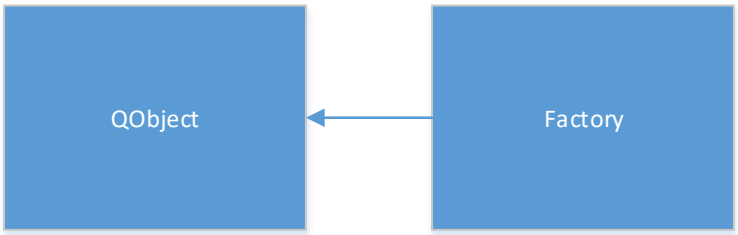
2.2 数据库 UML



数据库部分共建立了 TrainSeat . Train. TrainNumber. TrainPass(途经站). UserTicket. User 六张表来储存相关数据。（主键（pk）及外键(fk)图中已标出）

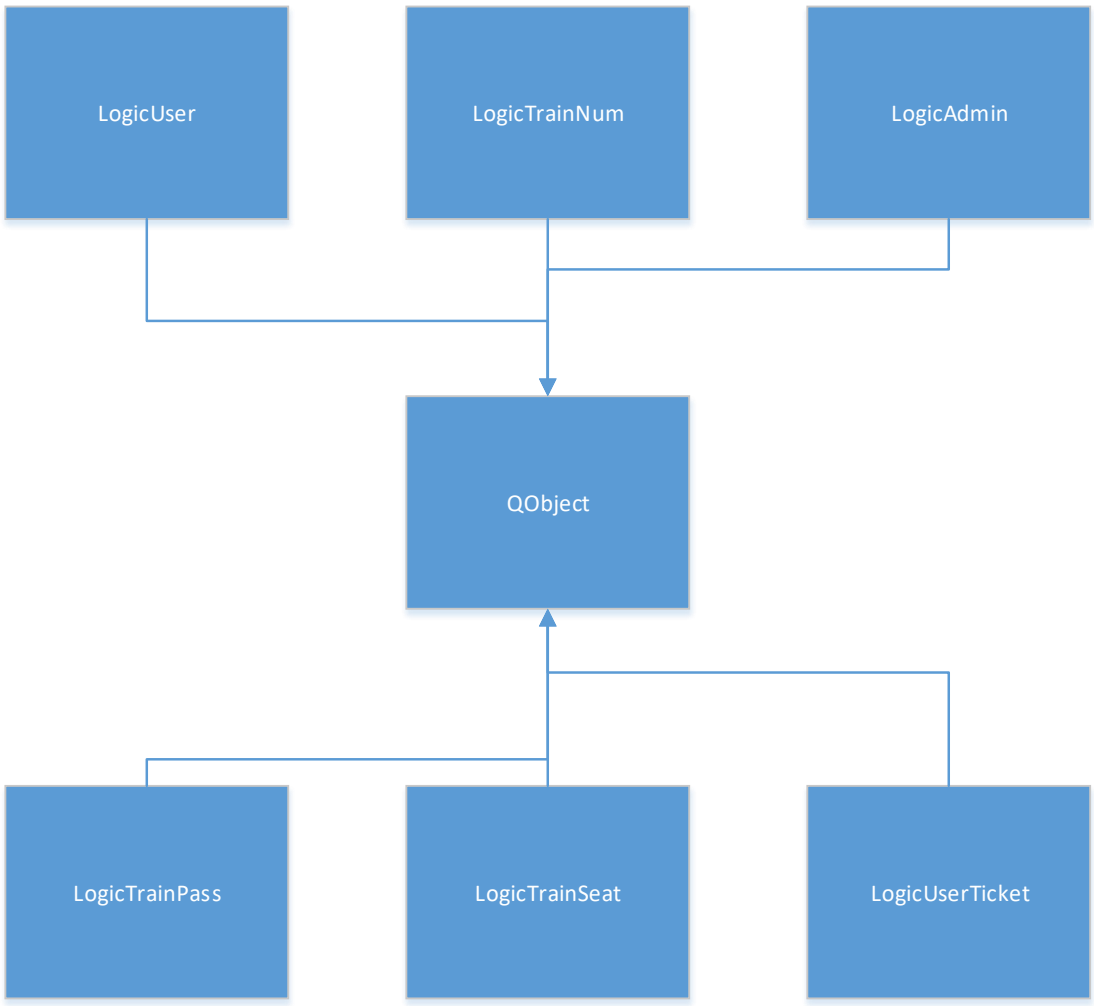
2.3 继承关系

数据库：



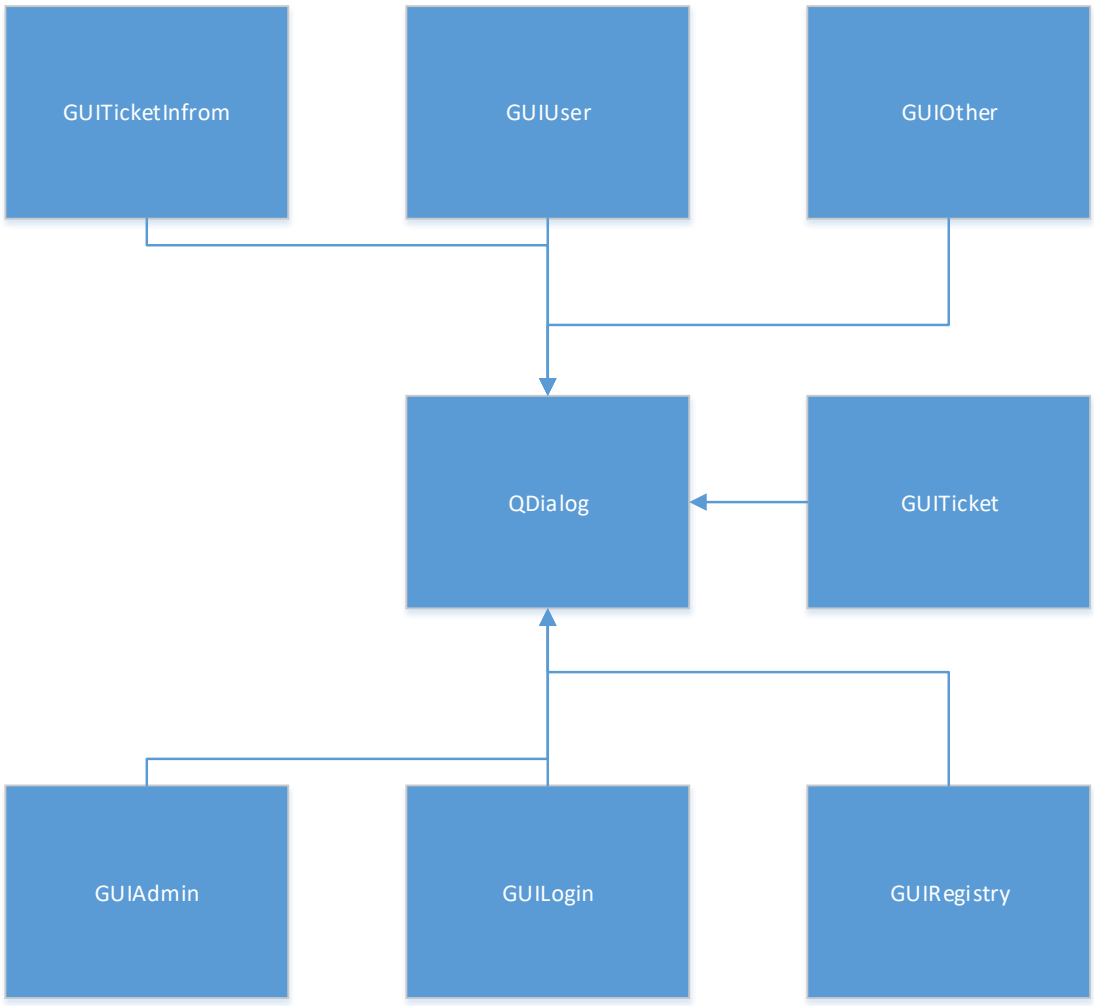
Factory 类继承自 Qobject

Logic:



类 LogicUser, LogicTrainNumber，LogicTrain，LogicTrainPass，LogicTrainSeat, LogicUserTicket,
对应 ModelUser ,ModelTrainNumber, ModelTrain, ModelTrainPass, ModelTrainSeat,ModelUserTicket.， 主要通过 SQL 语句实现了对数据库中数据的操作。

GUI(界面)：



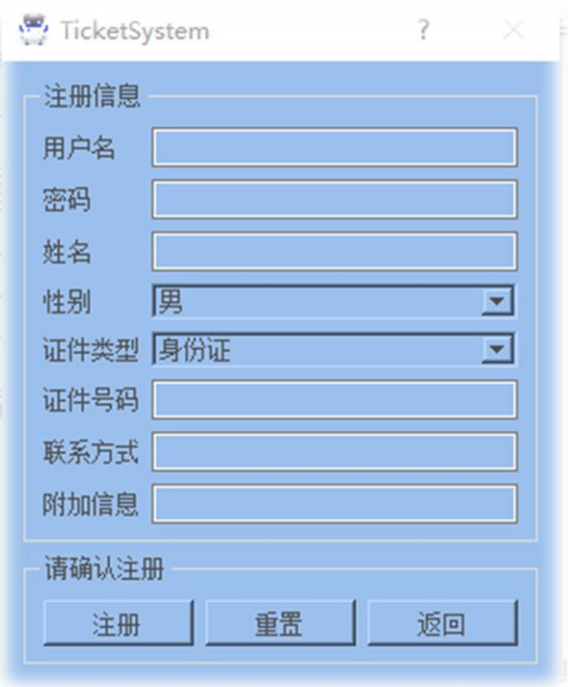
界面窗口均继承于 QDailog.

2.4 界面及关系

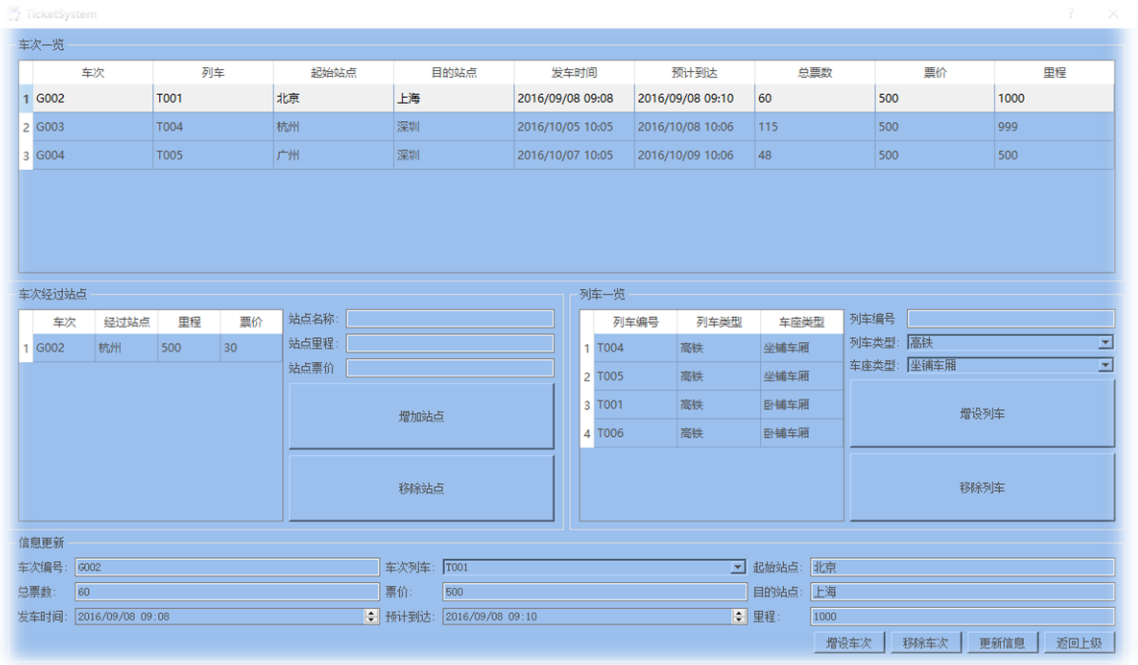
1. 登录


TicketSystem
欢迎使用火车订票系统
用户名:
密 码:
登陆
重置
注册

2. 注册


TicketSystem
注册信息
用户名
密码
姓名
性别
男
证件类型
身份证
证件号码
联系方式
附加信息
请确认注册
注册
重置
返回

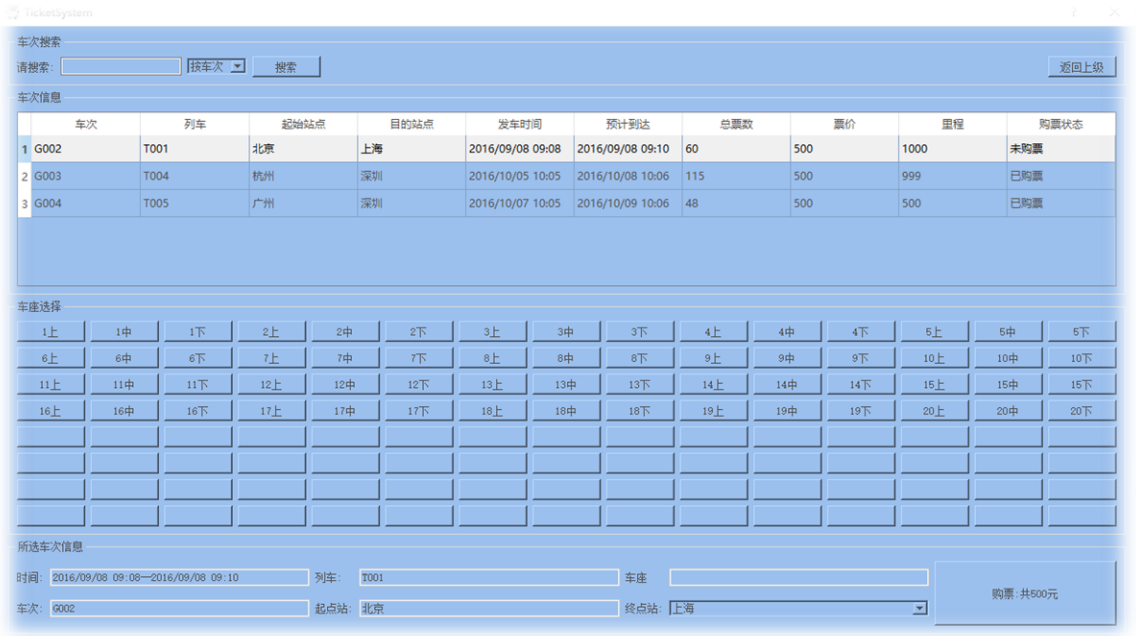
3. 票务管理


TicketSystem
车次一览
车次 列车 起始站点 目的站点 发车时间 预计到达 总票数 票价 里程
1 G002 T001 北京 上海 2016/09/08 09:08 2016/09/08 09:10 60 500 1000
2 G003 T004 杭州 深圳 2016/10/05 10:05 2016/10/08 10:06 115 500 999
3 G004 T005 广州 深圳 2016/10/07 10:05 2016/10/09 10:06 48 500 500
车次经过站点
车次 经过站点 里程 票价
1 G002 杭州 500 30
站点名称:
站点里程:
站点票价
增加站点
移除站点
列车一览
列车编号 列车类型 车座类型
1 T004 高铁 坐铺车厢
2 T005 高铁 坐铺车厢
3 T001 高铁 卧铺车厢
4 T006 高铁 卧铺车厢
列车编号
列车类型: 高铁
车座类型: 坐铺车厢
增设列车
移除列车
信息更新
车次编号: G002
车次列车: T001
起始站点: 北京
总票数: 60
票价: 500
目的站点: 上海
发车时间: 2016/09/08 09:08
预计到达: 2016/09/08 09:10
里程: 1000
增设车次
移除车次
更新信息
返回上级

4. 普通用户信息界面


TicketSystem
个人信息
用户名:freedom
姓名:基拉
性别:男
证件号码:524123564
证件类型:身份证
账户余额:98168
票务功能
个人购票
帮人购票
已购车票
账户充值
返回上级

5. 普通用户-个人购票



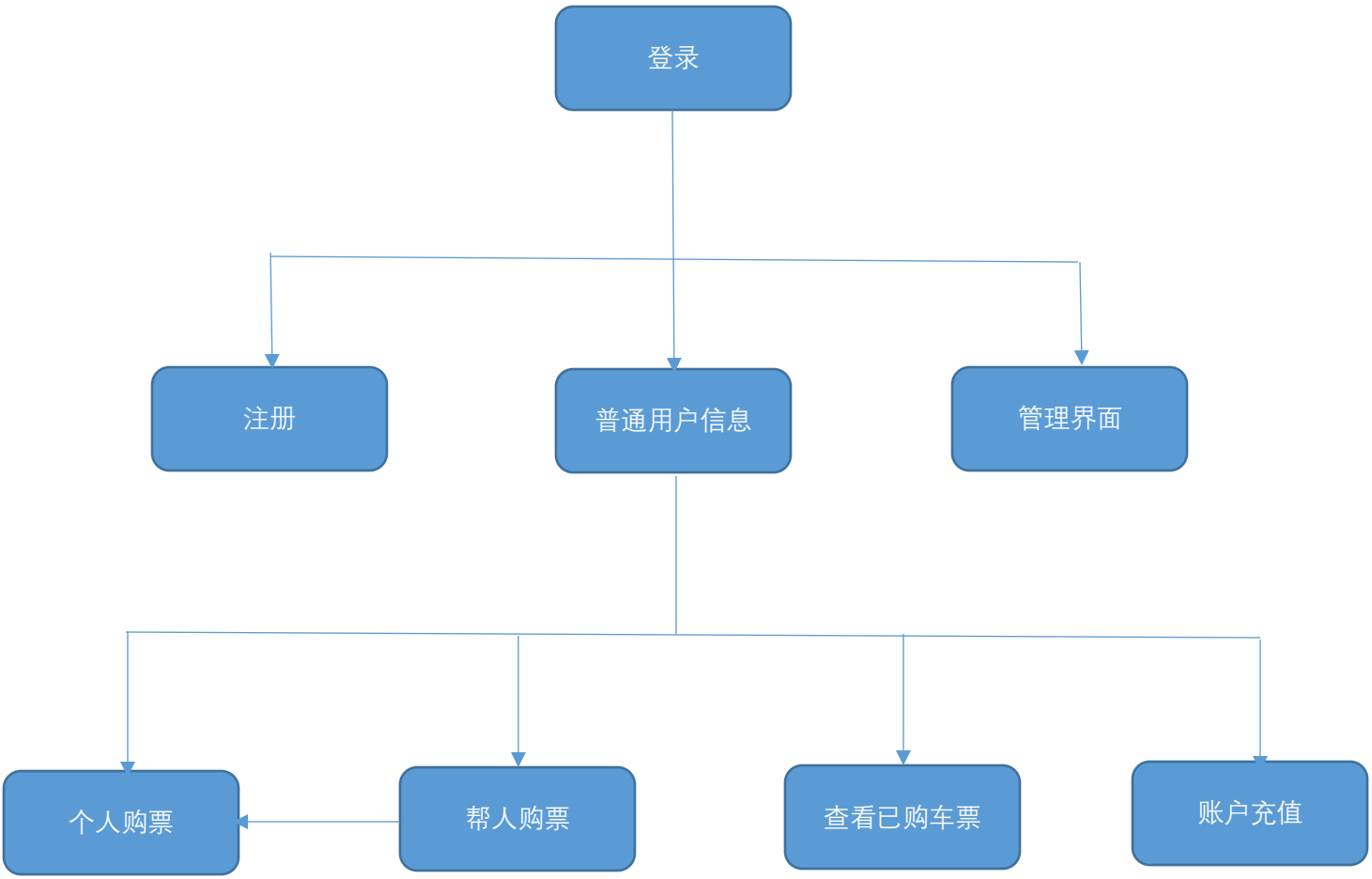
6. 代人购票

7. 查看已购车票

8. 账户充值

共八个界面实现具体的功能，其中界面进行了简单的半透明和背景颜色美化，加上了个性化的图标。

界面关系图：



主界面是登录界面，具体界面关系如图所示。

3 本人工作内容

我在本次大作业人中参与了数据库建立的讨论，逻辑部分模板的书写以及界面的开发和程序调试，对各个部分都有一定的了解，下面我将就各个界面的布局和功能实现细谈。

3.1 界面书写及布局

首先是界面书写及布局，最终的界面我选择的是用代码书写，一开始也想直接用 QTDesigner 直接画界面，但如果界面按钮多进行相关参数的配置，和布局也比较麻烦，并不比代码书写轻松。所以我开始在网上论坛查找，查阅书籍，开始尝试写登录界面，最终我发现了代码写界面有更高的自定义性。所有的界面类所有的界面类，都以 GUI 开头，继承自 QDialog，在构造函数中，均按顺序调用以下三个函数：

```
void InitWidgets();//初始化窗体控件
void InitWindowLayout();//初始化窗体布局
void InitSignalSlot();//初始化信号和槽连接
```

第一个为初始化窗体控件，这个函数实例化界面上的所有控件，设置控件的初始名称，显示字符，以及父对象。Qt 的核心是对象的父子关系，通过这个关系，保证了分配的内存不会泄露。如果设置了父对象，那么子对象会随着父对象的析构而析构。

第二个是初始化窗体布局，这个函数必须在 InitWidgets()函数调用之后调用，作用是利用布局管理器规划界面布局，实现界面的布局。

第三个函数是初始化信号和槽链接。将界面上的控件的信号和槽相连接。

下面以登录界面为例阐述具体的布局过程。

1. 根据手绘的稿图确定需要哪些窗体控件
   根据界面可以看出有 QGroupBox, QLabel, QLinEdit, QPushButton,以及布局管理器 QVBoxLayouot,QHBoxLayout,QGridLayout

```
gBoxLogin=new QGroupBox(CN("欢迎使用火车订票系统"),this); //登录区域
    labelUserName =   new QLabel(CN("用户名:"),this);//用户名标签
    labelPassWord=        new QLabel(CN("密    码:"),this);//密码标签
    lEditUserName =   new QLineEdit(this);//用户名输入框
    lEditPassWord=        new QLineEdit(this);//密码输入框
    btnLogin=         new QPushButton(CN("登陆"),this);//登陆按钮
    btnReset=         new QPushButton(CN("重置"),this);//重置按钮
    btnRegistry=      new QPushButton(CN("注册"),this);//注册按钮

   //布局管理器
    vBLayoutMain=         new QVBoxLayout(this);//主要布局管理器
    hBLayoutButtons= new QHBoxLayout(this);//按钮布局管理器
    vBLayoutGroup=        new QVBoxLayout(this);//注册信息布局管理器
    gLayoutInfrom=        new QGridLayout(this);//登陆信息布局管理器
```

2. 对控件进行布局，在 InitWindowLayout()中实现

（网格布局）

（水平布局）

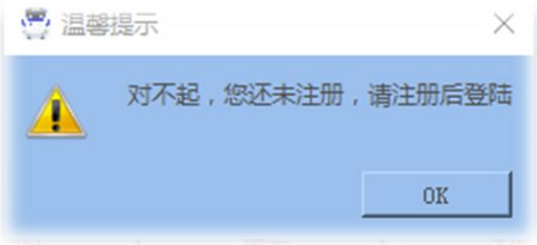
（将上面两部分垂直布局）

所有界面的布局都采用布局管理器得到，其间也加入了弹簧的使用和控件的自扩展。

3.2 界面功能的实现

1. 登录界面

登录界面比较简单，获取输入框的内容后，通过 Factory 类连接到数据库，判断当前账号的返回类型：1.普通用户 2.管理员 3.NULL，其中 NULL 表示登陆失败，会弹出未注册的警告框。如图：




```
connect(this->btnRegistry, SIGNAL(clicked()), this, SLOT(ShowRegistry()));
connect(this->btnLogin, SIGNAL(clicked()), this, SLOT(ShowLogin()));
connect(this->btnReset, SIGNAL(clicked()), this, SLOT(BtnClearClicked()));
```

通过信号与槽的对应，建立三个按钮与相应函数的对应，通过函数实现具体的功能。
登录界面根据实际实现登录密码隐藏，

```
//密码框，输入密码不可见
lEditPassWord->setEchoMode(QLineEdit::Password);
```

以及对界面的半透明化和背景颜色的设置等个性化处理。

```
//设置背景颜色及字体对比色
setStyleSheet ("background-color: rgb(156,192,237);color: rgb(80,80,80);");
setWindowOpacity(0.9);//设置透明度为 0.9
```

根据密码判断类型

```
void GUILogin::ShowLogin()
{
    Factory factory;
    QString userName = this->lEditUserName->text();
    QString passWord = this->lEditPassWord->text();
    if (factory.logicUser->Login(userName,passWord)=="admin")
    {
        guiAdmin->show();
        this->hide();
        return;
    }
    if (factory.logicUser->Login(userName, passWord) == "ordinary")
    {
        guiUser->SetUserInfrom(userName, passWord);

        guiUser->show();
        this->hide();
    }
    else{
        //弹出警告消息框
        QMessageBox::warning(this, CN("温馨提示"),CN("对不起，您还未注册，请注册后登陆"),QMessageBox::Ok);
    }
}
```

具体的实现是先由 Factory 类对象调用 logicUser,返回数据库中的类型，从而进行进一步的判断。

2. 注册界面

界面主要是 QLabel 和 QLineEdit 的堆砌，布局步骤与前面类似，不再加以说明，在注册界面中有了返回上级的函数，具体实现如下：

```
void GUIRegistry::ShowParent()
{
    //将父指针向下转型为QDialog
    QDialog *parentDialog = qobject_cast<QDialog*>(this->parent());
    this->hide();

    lEditUserName->clear();
    lEditPassWord->clear();
    lEditName->clear();
    lEditNumber->clear();
    lEditConnection->clear();
    lEditAddtion->clear();

    parentDialog->show();
}
```

在任意一个 GUI 类初始化的时候，需要窗体父窗体的指针：然后才能返回上级，思路就是隐藏当前窗体，显示父窗体。parent()可以获取当前类的父窗体指针，是一个 QObject 类，只要是继承自 QObject 类的类，都有 parent()。show()方法是 QWidget 类的基本方法，父窗体是一个继承自 QDialog(继承自 QWidget)的类，所以使用 qobject_cast 强制转换。

3. 管理界面

管理界面的实现是本程序的难点之一。

（1）容错性部分：在站台名，票价，里程数的输入用到了数据的校验知识，本

来打算在注册界面也使用，但考虑都外国人姓名，证件号，电话可能和中国不一样，未校验，但在管理界面中站台名，地名必须为中文，所以采用了数据的校验，采用了正则表达式的验证，通过网上查找资料得到，实现的效果是管理员输入站台名时必须是中文，里程数和票价也做了数值位数的规定。

发车时间和预计达到时间没有使用输入框 QLineEdit 使用的是时间输入框 QDateTimeEdit，设置格式如下

timeEditTrainNumSTime->setDisplayFormat("yyyy/MM/dd hh:mm");
timeEditTrainNumETime->setDisplayFormat("yyyy/MM/dd hh:mm");。

（2）布局部分，依旧是通过布局管理器进行布局，但本界面车次信息的显示使用了 QTableWidget 来实现已有车次信息以表格形式显示。

	车次	列车	起始站点	目的站点	发车时间	预计到达	总票数	票价	里程
1	G002	T001	北京	上海	2016/09/18 09:08	2016/09/19 09:10	60	500	1000
2	G003	T004	杭州	深圳	2016/10/05 10:05	2016/10/08 10:06	116	500	999
3	G004	T005	广州	深圳	2016/10/07 10:05	2016/10/09 10:06	48	500	500

该表的实现是构造了一个 GetTrainNumList()函数，具体分两步进行：

首先是建立 QVector<ModelTrainPass> trainList 这个以 ModelTrainNumber 为类模板的动态数组，通过 Factory 类调用数据库中的数据，具体用语句 trainList = factory.logicTrainNumber->GetList();通过调用 GetList()函数得到 trainList 的相关数据；

然后就是将该动态数组的数据显示在表格中，具体做法是用 QTableWidget 中的 setItem()和 setText()实现，采用 for 循环语句来遍历动态数组中的每一个元素，实现对车次的显示。

//此步骤代码如下：

```
for (int i = 0; i < trainListRow; i++)
    {

        this->tabWTrainNum->setItem(i, 0, new QTableWidgetItem());
        this->tabWTrainNum->setItem(i, 1, new QTableWidgetItem());
        this->tabWTrainNum->setItem(i, 2, new QTableWidgetItem());
        this->tabWTrainNum->setItem(i, 3, new QTableWidgetItem());
        this->tabWTrainNum->setItem(i, 4, new QTableWidgetItem());
        this->tabWTrainNum->setItem(i, 5, new QTableWidgetItem());
        this->tabWTrainNum->setItem(i, 6, new QTableWidgetItem());
        this->tabWTrainNum->setItem(i, 7, new QTableWidgetItem());
        this->tabWTrainNum->setItem(i, 8, new QTableWidgetItem());


        tabWTrainNum->item(i, 0)->setText(trainList[i].TN_Id());
```

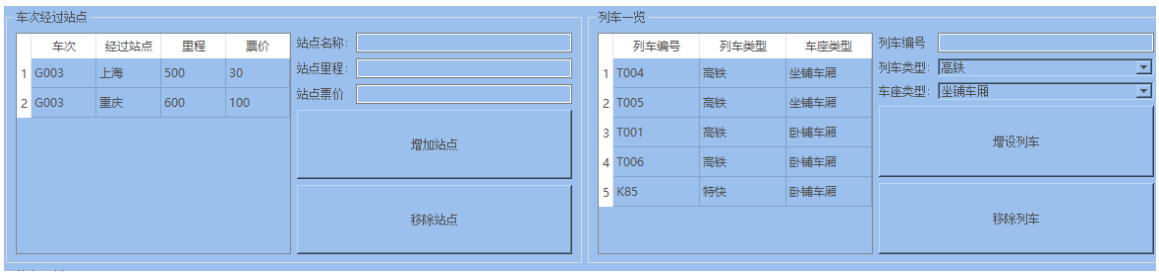
```
tabWTrainNum->item(i, 1)->setText(trainList[i].T_Id());
tabWTrainNum->item(i, 2)->setText(trainList[i].TN_SStation());
tabWTrainNum->item(i, 3)->setText(trainList[i].TN_EStation());
tabWTrainNum->item(i, 4)->setText(trainList[i].TN_STime());
tabWTrainNum->item(i, 5)->setText(trainList[i].TN_ETime());
tabWTrainNum->item(i, 6)->setText(trainList[i].TN_TicketNum());
tabWTrainNum->item(i, 7)->setText(trainList[i].TN_TicketPrice());
tabWTrainNum->item(i, 8)->setText(trainList[i].TN_Distance());
```

}

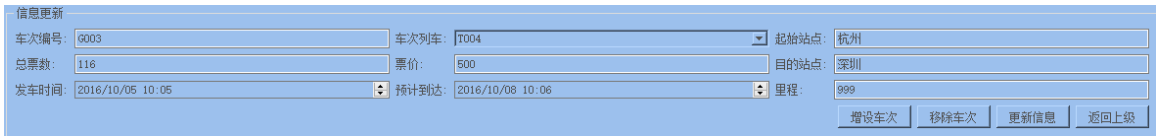
补充一点，表的自适应，主要是调用 QTableWidget 的自带的方法，如果不设置这些方法，那么表不会自动适应窗体，而且默认表会变得可以编辑，这和目标是不符合的具体代码如下：

```
tabTrainInfrom->horizontalHeader()->setHighlightSections(false);
//表头自适应大小
tabTrainInfrom->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);
//一次只能选择一行
tabTrainInfrom->setSelectionBehavior(QAbstractItemView::SelectRows);
//一次只能选择一个对象
tabTrainInfrom->setSelectionMode(QAbstractItemView::SingleSelection);
//单元格不能编辑
tabTrainInfrom->setEditTriggers(QAbstractItemView::NoEditTriggers);
```

采用相似的方式也可以实现该界面中站台和列车（结合实际情况，列车号和车次号在本项目中加以区分）的显示；



显示部分还有就是信息更新部分：



该栏的显示实现方法与前面有所不同，由 QLabel，QComboBox 和 QLineEdit 等控件，用布局管理器整理而成，对于数据的显示是采用 QLineEdit,一方面能实现数据的显示，另一方面又能进行数据的输入，从而实现数据更新的功能。

（3）建立关系

管理界面需要实现显示界面之间的关系，以及与按钮之间的关系，通过信号与槽来实现，具体建立关系是在函数 InitSignalSlot()中实现，通过调用相关函数来实现

具体功能，将各个板块联系起来，具体代码如下：

```
void GUIAdmin::InitSignalSlot()
{
    connect(this->btnTrainInfromAdd, SIGNAL(clicked()), this, SLOT(AddTrainClicked()));
    connect(this->btnTrainInfromDelete, SIGNAL(clicked()), this, SLOT(DeleteTrainClicked()));

    connect(this->btnAdd, SIGNAL(clicked()), this, SLOT(AddTrainNumClicked()));
    connect(this->btnDelete, SIGNAL(clicked()), this, SLOT(DeleteTrainNumClicked()));
    connect(this->btnUpdate, SIGNAL(clicked()), this, SLOT(UpdateTrainNumClicked()));
    connect(this->btnBack, SIGNAL(clicked()),this,SLOT(ShowParent()));

    connect(this->btnTrainPassAdd, SIGNAL(clicked()), this, SLOT(AddTrainPassClicked()));
    connect(this->btnTrainPassDelete, SIGNAL(clicked()), this, SLOT(DeleteTrainPassClicked()));

    //表
    connect(this->tabWTrainNum, SIGNAL(cellClicked(int,int)),this,SLOT(TabTrainNumClicked(int,int)));
}
```

各个具体功能函数的实现，其实就是在用户对界面做出改变后需要对数据库中的内容进行相应的增删改查，这就需要调用我们写的相应的 Logic 类，具体步骤是先声明一个 Factory 类，用于调用相应 logic 中的相关函数进行具体的 SQL 语句操作，这里需要注意的是由于 Logic 类返回的是 Model 类，所以先声明 Model 类，再调用相应的 Logic 类。具体如下面的代码：

```
ModelTrainNumber modelTrainNumber;
modelTrainNumber.TN_Id(tN_Id);
modelTrainNumber.T_Id(t_Id);
modelTrainNumber.TN_SStation(tN_SStation);
modelTrainNumber.TN_EStation(tN_EStation);
modelTrainNumber.TN_STime(tN_STime);
modelTrainNumber.TN_ETime(tN_ETime);
modelTrainNumber.TN_TicketNum(tN_TicketNum);
modelTrainNumber.TN_TicketPrice(tN_TicketPrice);
modelTrainNumber.TN_Distance(tN_Distance);


Factory factory;
if (factory.logicTrainNumber->Update(modelTrainNumber))
{
    GetTrainNumList();
}
```

（4）消息提示框：管理界面为了保证管理员在录入车次时信息填写完整，以及保证填写的座位数正确，所以设置了消息提示框。

提示输入完整信息：

提示列车座位数：

消息框的实现是通过 if 条件判断和 QMessageBox 的使用，具体见如下代码：

```
if (tN_Id.isEmpty()
    || t_Id.isEmpty()
    || tN_SStation.isEmpty()
    || tN_EStation.isEmpty()
    || tN_TicketNum.isEmpty()
    || tN_TicketPrice.isEmpty()
    || tN_Distance.isEmpty()
    )
{
    //弹出警告消息框
    QMessageBox::warning(this, CN("注意"),CN("开通车次请输入完整信息"),QMessageBox::Ok);
    return;
}
```

4.用户信息界面

用户界面是已注册用户在输入正确的账号和密码后跳转进入的界面。

如图所示，用户信息界面更像是一个统筹几个子界面的主界面，左侧的 QGroupBox 里面显示了当前用户的信息，具体实现方法是用 QLabel 控件来对数据库提供的信息进行显示，用到了 setText()函数，具体代码如下：

```
void GUIUser::SetUserInfrom(QString userName, QString passWord)
{
    this->userName = userName;
    this->passWord = passWord;

    Factory factory;
    ModelUser modelUser;
    modelUser = factory.logicUser->GetUserInfrom(userName);


    labelUserName->setText(CN("用户名:") + modelUser.U_Id());
    labelName->setText(CN("姓名:") + modelUser.U_Name());
    labelSex->setText(CN("性别:")+modelUser.U_Sex());
    labelNum->setText(CN("证件号码:") + modelUser.U_Number());
    labelNumType->setText(CN("证件类型:") + modelUser.U_NumType());
    labelAddtion->setText(CN("账户余额:")+modelUser.U_Balance());

}
```

布局和前面的方法基本一致，通过信号与槽管理四个子界面，进入子界面的具体实现是在 InitWidgets()中包含子界面的类，在 InitSignalSlot()函数中通过信号与槽建立具体关系，在通过具体的函数实现。账户充值界面采用了 QInputDialog 实现了简单的窗体，充值界面代码如下：

int recharge=QInputDialog::getInt(this, CN("充值"),CN("请输入您要充值的金额O(∩_∩)O~"),0,0,100000,1);

该界面显示了账户余额，然而在用户购票或者为他人购票时候，会切换到子窗体，子窗体会对 User 表的余额列操作，用户的余额会变化，这时候需要通知主窗体，可以使用 emit，或者重写主窗体的 showEvent()方法，程序采用后者。需要注意的是，重写该事件后，需要将事件向上传递给父类 QDialog，具体代码实现如下：

```
void GUIUser::showEvent(QShowEvent *e)
{
    if (!this->userName.isEmpty())
    {
        Factory factory;
        ModelUser modelUser;
        modelUser = factory.logicUser->GetUserInfrom(userName);


        labelUserName->setText(CN("用户名:") + modelUser.U_Id());
        labelName->setText(CN("姓名:") + modelUser.U_Name());
        labelSex->setText(CN("性别:") + modelUser.U_Sex());
        labelNum->setText(CN("证件号码:") + modelUser.U_Number());
        labelNumType->setText(CN("证件类型:") + modelUser.U_NumType());
        labelAddtion->setText(CN("账户余额:") + modelUser.U_Balance());

    }

    QDialog::showEvent(e);
}
```

5.购票界面

（1）购票界面是本项目的又一个难点。首先是搜索，将搜索的类型放入下拉列表，不同的条件其实是不同的链接查询条件，例如按车次查询，就相当于知道车次的主键，查询车次的所有信息。

（2）相比于前面的界面，购票界面的一大困难点在于实现用户通过界面购票，题目要求用户可以自由选择车座，车座的实现有两种情况，坐铺和卧铺，数量分别为 120 和 60，可以看到 120 和 60 的最大公约数就是 60，那么可以将车座设计为 120 个按钮，某个按钮单击时，相当于选择了对应的座位，座位添加到界面，使用网格布局管理器，按 15*4 和 15*8 的方式排列，在根据车型，如果是坐铺就显示 120 个座位，如果是卧铺，就只显示前 60，后 60 取消显示：

设置 120 个按钮：

```
int btnIndex = 0;
for (int i = 0; i < 8;i++)
{
    for (int j = 0; j < 15;j++)
    {
        //120个按钮 信号和槽连接
        connect(&btnSeats[btnIndex], SIGNAL(clicked()), this, SLOT(SelectSeatNum()));
        btnSeats[btnIndex].setParent(this);
        gLayoutSeats->addWidget(&btnSeats[btnIndex],i, j);
        btnIndex = btnIndex+1;
    }
}
```

为每个按钮设置 objectName(),可以在槽函数中获取信号源对象，以此获取 objectName()就知道是哪个按钮被单击了，具体的分情况显示就是通过 if 条件句判断列车类型，如果为坐铺，就显示 120 个按钮，如果是卧铺，显示前 60 就行，具体代码实现如下：

```
if (CN("坐铺车厢") == currentTrainSeat)
{
    //1-120编号
    for (int i = 0; i < 120;i++)
    {
        this->btnSeats[i].setEnabled(true);
        this->btnSeats[i].setObjectName(QString("%1").arg(i+1));
        this->btnSeats[i].setText(QString("%1").arg(i+1)+CN("号"));
    }
}
else
{
    if (CN("卧铺车厢") == currentTrainSeat)
    {
        //1-60
        //前60显示
        int currentKey = 1;
        for (int i = 0; i < 60; i++)
        {
            this->btnSeats[i].setEnabled(true);
            this->btnSeats[i].setObjectName(QString("%1").arg(i+1));
            this->btnSeats[i].setText(QString("%1").arg(currentKey) + seatMap[(i+1)%3]);
            if ((i+1)%3==0)
            {
                currentKey++;
            }
        }
```

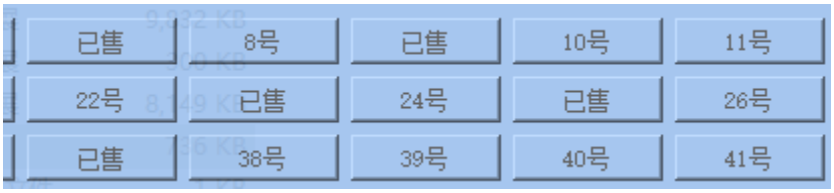
```
        //后60禁止
        for (int i = 60; i < 120; i++)
        {
            this->btnSeats[i].setEnabled(false);
            this->btnSeats[i].setText(CN(""));
        }
    }
}
```

从而实现了界面显示座位时坐铺和卧铺不同。接下来就是座位按钮上的信息显示，由于数据库中对于座位号的记录是数字，如果是坐铺车，可以直接显示座位号，没问题，但如果是卧铺，就得分上中下加以显示。这里使用了 QMap，将 对于数字转为 1 上， 3 下 之类的真实编号。这其实是简单的取模运算问题，座位号模 3 余 1 就为上，余 2 为中，余 0 为下，

```
QMap<int, QString> seatMap;
seatMap.insert(1, CN("上"));
seatMap.insert(2, CN("中"));
seatMap.insert(0, CN("下"));
```

```
//模 3 取余
int currentKey = 1;
for (int i = 0; i < 60; i++)
{
    this->btnSeats[i].setEnabled(true);
    this->btnSeats[i].setObjectName(QString("%1").arg(i+1));
    this->btnSeats[i].setText(QString("%1").arg(currentKey) + seatMap[(i+1)%3]);
    if ((i+1)%3==0)
    {
        currentKey++;
    }
}
```

座位，在单击某一列车次的时候，会通过数据库调用该车次(一个车次对应一列车)的所有已售出的座位列表，然后将对应的座位设置为不可用（已售）即可。



（3）实现分里程计费，选择到达站点，直接将车次对应的站点连接查询，返回到下拉列表中，同时要返回该站点的里程价格，将里程价格写入 itemData，这样单击某一个下拉列表选项时，可以获得当前里程的价格。

```
//填写 里程-价格键值对
this->comBoxTrainEStation->addItem(tabWTrainNum->item(row, 3)->text(), tabWTrainNum->item(row, 7)->text());
for (int i = 0; i < currentTrainNumPass.count();i++)
{
    this->comBoxTrainEStation->addItem(currentTrainNumPass[i].TP_SName(), currentTrainNumPass[i].TP_Price());
}
```

实现一人一票只需要用 if 条件句判断该用户的购票状态，如果当前用户在当前车次购买了票，那么就直接禁用购票按钮。实现出发半小时前才可以购票，将车次

的出发时间减去 30 分钟(加上-1800 秒)，然后与当前时间作比较，调用 QDateTime 的现成的方法。

（4）更新表：

User 表的余额按里程的票价减少，如果余额不足，按照题目要求弹出使用其他方式支付的对话框，然后继续购票，只是不减少余额。

TrainNum 表的车票数量-1

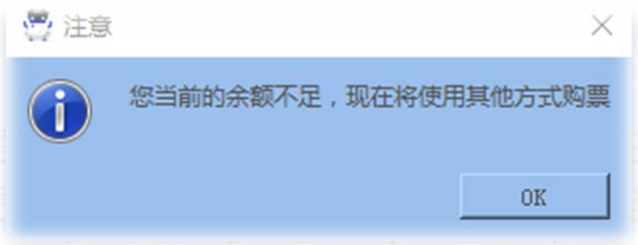
TrainSeat 表中增加当前车次的被售出的座位(用户选择的座位)

UserTicket 表中添加一条新的数据，代表新的车票诞生，主键直接使用 UUID(全球唯一标识码)代表车票的机打号码。这样保证了车票号的唯一性，采用了 QUuid。

（5）消息提示框：

购票成功

余额不足：



为了简单，已经默认提示后会以其他方式买票；

6.代人买票

他人购票的实现不复杂，只是扣款扣的是购买人而已，与个人购票只有这里有区别。由于时间有限，系统简化为为其他注册人员买票，但所买票只有票主才能看。

7.车票信息界面：

该界面列出了用户所买车票的信息，实现起来不难，方法前面已经讲了，就不再赘述，这里只是需要注意退票比例的计算，退票会扣除 20%的金额。

3.3 部分数据库及逻辑的工作

数据库：

数据库的建立在定了大致的表和元素后，初步建立了一个表，我为了方便 QT 的调用，采取了 SQLite 作为数据库，使用 Powerdesigner 建立数据库的概念模型，由概念模型生成逻辑模型和物理模型，最后生成数据脚本(ODBC3)。导入到 SQLite3 数据库中，数据库的管理程序有很多种，例如 sqlitestudio，如果是 Win10，必须要以管理员的权限，才能运行这类程序。

```
//数据库定义 使用 SQLite3
#ifndef TICKETSYSTEM_DATABASE
#define TICKETSYSTEM_DATABASE
//SQLite3 数据库文件
#define DATABASE_ADDRESS CN("TicketSystemDB.db")

#endif
```

注意：这种配置方法好处在于不需要去设置绝对路径，不过要将数据文件和 exe 文件放在同一文件夹下。

逻辑层面:Logic 和 Model 类的书写，这一部分我主要是给出界面需要调用那些参数，写了一个模板，由 bzp 根据我提供的参数调用要求完成具体的代码书写，最后由我进行微调。Logic 和 Model 类是为了实现对数据库的调用，是三层架构的中间部分，主要是通过 SQL 语句操作数据库数据并将其写到 Model 类中，返回具体的 Model 类。比如我写的其中的一个模板 LogicTrain 中的具体代码:

```
ModelTrain LogicTrain::GetTrainInfromById(QString T_Id)
{

    ModelTrain trainList;

    //判断是否创建数据库连接
    if (!this->DB.isOpen())
    {
        return trainList;
    }
    QString sql = CN("SELECT * FROM Train WHERE T_Id='%1'")
        .arg(T_Id)
        ;

    QSqlQuery query;
    query.exec(sql);
    while (query.next())
    {
        trainList.T_Id(query.value("T_Id").toString());
        trainList.T_Type(query.value("T_Type").toString());
        trainList.T_SeatType(query.value("T_SeatType").toString());
        break;
    }
    return trainList;
}
```

3.4 程序的最终拼接，调试及配置

1，字符集

字符集是 Qt 十分麻烦的地方，对于字符集的处理有两种情况:

(1)如果所有的.h 和 .cpp 文件已经保存为 UTF-8 编码格式，那么在 Qt5 以上版本，可以使用 QStringLiteral()来解析非 Latin1 字符集的字符，例如中文，QtCreator 和 VS 均可正常显示。如果使用 tr()那么 QtCreator 可以正常显示，部分 VS 不能。

(2).如果.h 和 .cpp 文件未保存为 UTF-8 编码格式，那么 VS 可以正常显示中文，QtCreator 默认不能正常显示。QtCreator 在不设置字符集的情况下，两种 CN()宏都失效。

经过网上查找资料，起初按照网上的说法定义宏，没有保存为 UTF-8 格式，在 QT 中中文部分显示乱码，后来将全部文件转化为 UTF-8，已解决乱码问题；

2. Factory 类

这是在网上查找的一种典型的做法，称之为工厂模式，将 Logic 类内嵌到该类中统一管理数据接口。

3.exe 图标

为了美化，将 exe 文件加了图标，具体做法参照网上教程，

4.调试及配置

QT 生成的 exe 文件要在其他电脑上运行，需要一些 dll 文件的移植

4 项目总结

回想整个项目的开发过程，其实是一个不断发现问题，查找资料，解决问题的过程。大作业开始前的自学准备，根据老师的建议，通过具体的例子确实是短时间学习 QT 的捷径。架构上采取三层架构的模式。在网上搜索相关资料后进一步加深了这一设计理念，同时也查到了工厂模式，采用这两种模式确实也提高了效率。在整个界面编写过程中都是一个边做边学的过程，查具体器件的使用方法，查数据校验的正则表达式等等，在整个过程中学到了很多东西。

在程序的 debug 阶段，由于是分工写的，调参数阶段特别头疼，好不容易使程序运行起来了，发现购票界面又出现了在线选票是的错位现象，后来发现才是数组下标忘了要减一。看着程序慢慢成型，虽然苦，还是挺开心的，也学到了很多知识。

将编译的 exe 文件放到其他电脑，发现不能运行，为了使程序能在其他电脑上运行，又在网上找寻解决方法，添加 dll 文件。

整个过程，从项目构思到编写程序再到调试，自己对 oop 的设计思想有了进一步的理解，同时也学到了很多知识，最重要的是锻炼了自学能力，学会了通过网络资源来解决问题。

5 相关问题的说明

开发环境 VS2012，QT5.5.1，项目在 VS2012 上编写，由 Qt Visual Studio Add-in.生成 QT 项目，数据库要放到 exe 文件旁(所需 dll 文件已复制到程序所在文件夹中)，需注意：由于 win10 系统权限管理问题，请右键 exe 文件，以管理员身份运行，数据库文件的打开可以使用 sqlitestudio。

参考资料：

1. Qt 快速入门系列教程：
http://bbs.qter.org/forum.php?mod=viewthread&tid=193
2. <QT>常见错误总结：
http://blog.sina.com.cn/s/blog_4ba5b45e0102e8h5.html
3. Qt 应用程序的发布,exe 文件图标设置
https://my.oschina.net/mjRao/blog/91049
4. QRegExp 解析
http://blog.csdn.net/c05170519/article/details/6873440
5. **【Qt5 开发及实例】正则表达式的验证**
http://blog.csdn.net/cutter_point/article/details/42009637
6. Qt 之操作数据库（SQLite）
http://blog.sina.com.cn/s/blog_a6fb6cc90101gx30.html
7. **Windows 如何打包 Qt 程序**

http://blog.csdn.net/gzshun/article/details/7495488
8. 讲解三层结构和抽象工厂模式的技巧
http://blog.sina.com.cn/s/blog_53dbe8cb0100n54w.html
9.通过百度百科，维基百科查找的相关资料。