

**课 程 实 验 报 告**

**课程名称： 射频识别技术原理及应用**

**专业班级： 物联网工程1601**

**学 号： U2016148989**

**姓 名： 潘翔**

**指导教师： 甘早斌**

**报告日期： 2019.4**

**计算机科学与技术学院**

目 录

[1 实验一 低频读写器实验 4](#_Toc1089144420)

[1.1 实验目的 4](#_Toc1089520888)

[1.2 实验内容与要求 4](#_Toc2003266999)

[1.3 实验过程与结果 5](#_Toc499586533)

[1.4 核心源码说明 9](#_Toc1283983327)

[1.5 思考题 17](#_Toc220371342)

[1.6 实验体会与总结 18](#_Toc703050852)

[2 实验二 高频读写器实验(ISO14443A) 19](#_Toc1954935420)

[2.1 实验目的 19](#_Toc1437857)

[2.2 实验内容与要求 19](#_Toc1020371344)

[2.3 实验过程与结果 20](#_Toc2146620975)

[2.4 核心源码说明 24](#_Toc2070518708)

[2.5 思考题 31](#_Toc1290743051)

[2.6 实验体会与总结 32](#_Toc905597753)

[3 实验三 高频读写器实验(ISO15693) 34](#_Toc1455729821)

[3.1 实验目的 34](#_Toc1568408128)

[3.2 实验内容与要求 34](#_Toc1868315406)

[3.3 实验过程与结果 35](#_Toc634758583)

[3.4 核心源码说明 47](#_Toc701608155)

[3.5 思考题 60](#_Toc1849951103)

[3.6 实验体会与总结 60](#_Toc1968403559)

[4 实验四 超高频读写器实验 62](#_Toc838942716)

[4.1 实验目的 62](#_Toc974881249)

[4.2 实验内容与要求 62](#_Toc542792153)

[4.3 实验过程与结果 63](#_Toc1680618658)

[4.4 核心源码说明 71](#_Toc98417840)

[4.5 思考题 75](#_Toc1441932717)

[4.6 实验体会与总结 76](#_Toc1805129550)

[5 实验五 基于RFID技术的图书管理系统设计与实现 77](#_Toc1524568930)

[5.1 实验目的 77](#_Toc1416226539)

[5.2 实验内容与要求 77](#_Toc684769381)

[5.3 实验环境 78](#_Toc466229703)

[5.4 系统架构 79](#_Toc358263779)

[5.5 系统实现 80](#_Toc540552732)

[5.6 系统功能测试 94](#_Toc965816236)

[5.7 核心源码说明 99](#_Toc1642247107)

[5.8 实验体会与总结 112](#_Toc760924074)

[6 参考文献 114](#_Toc1668867089)

[7 程序源码 115](#_Toc1449698879)

[7.1 Qt Project Profile 116](#_Toc762361931)

[7.2 BooksManage 118](#_Toc541754785)

[7.3 Borrow\_Return 132](#_Toc1448836206)

[7.4 Sqlite 146](#_Toc685396991)

[7.5 Stat\_dialog(统计图表绘制) 162](#_Toc1832497836)

[7.6 Tools工具类 164](#_Toc206950311)

[7.7 UHF\_Thread 171](#_Toc2141126812)

[7.8 Usermanager 181](#_Toc1253422316)

[7.9 Widget 194](#_Toc2075265717)

# 实验一 低频读写器实验

## 实验目的

1. 通过本次实验了解博创科技 RFID 读写器的结构组成，熟悉各个模块的功能，掌握 试验箱的连接和操作方法。掌握串口命令参数的意义和设置方式。 了解低频读写器的基本原理，学会如何使用实训软件对低频读写器进行读卡操作（验证性实验）。
2. 学习和掌握在低频读写器的编程操作，对标签进行读操作，了解低频读写器的工作机理，并完成一个示例程序。

## 实验内容与要求

### 实验内容

1. 完成低频读写器的标签读取试验；
2. 熟悉低频 LF-125K 开发实例，完成低频读写器编程实验，熟悉和了解低频读写器API 函数;分析应用 demo 存在的问题，进而完善应用 demo 的功能。完善的功能包括： 记录保存进出的历史记录、停留时间。

### 实验要求

1. 学会通过试验箱对低频标签的读取，通过反复循环读取十张低频电子标签，记录 在读取过程中遇到的问题或发现的现象，并分析遇到的这些问题或现象的原因；
2. 掌握低频读写器 API 函数的调用方法，编写程序打开串口，建立连接，读取标签信息；
3. 将标签移进移出低频模块天线，能够记录标签读取的次数。

## 实验过程与结果

### 环境配置

考虑在自身PC进行相关实验，故进行环境配置。使用x86交叉编译器进行编译。

1. 平台环境

OS: Manjaro 18.0.4 Illyria

Kernel: x86\_64 Linux 5.0.7-1-MANJARO

CPU: Intel Core i7-6700HQ @ 8x 3.5GHz

GPU: GeForce GTX 965M

RAM: 7865MiB

1. 软件环境

Qt Creator 3.5.1 (opensource)

Based on Qt 5.5.1 (GCC 4.9.1 20140922 (Red Hat 4.9.1-10), 32 bit)

### 实验过程

1. 发现问题

对系统进行测试，发现只能使用单卡进行相应操作，考虑纪录更新函数进行更改，同时希望添加时间记录。

1. 更改代码



图1.1 代码修改图

|  |
| --- |
| /\*\*   \* @brief MainWindow::readData   \* 读取串口数据   \*/  **void** MainWindow::readData()  {  **if**(serialPort->bytesAvailable() < 5)  **return**;      QByteArray data = serialPort->readAll();  **if**(m125dll->LF125K\_FrameAnalysis((uint8 \*)(data.data())) == 0)      {          QString tagId = CharStringtoHexString(tr(" "),data.data(),data.length());//获取标签ID          QString **time** = CurrentDateTime();//获取时间  **int** index = model->findRecord(tagId);//查询此标签记录  **if**(index >= 0 )          {              QString text = model->record(index).value(2).toString();  **if**(text == tr("进"))                  model->updateRecord(index,tagId,**time**,tr("出"));  **else**                  model->updateRecord(index,tagId,**time**,tr("进"));          }  **else** {              model->addRecord(tagId,**time**,tr("进"));          }      }  } |

1. 测试



图1.2 系统进出测试

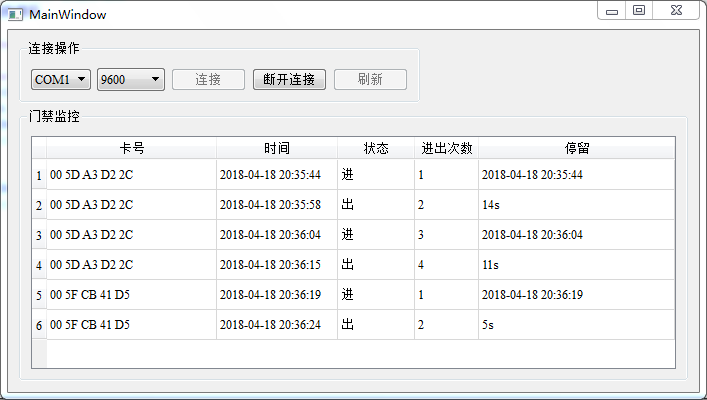


图1.3 系统停留时间测试

## 核心源码说明

### Qt事件相应机制

事件是由窗口系统或 qt 本身对各种事务的反应而产生的。 当用户按下、 释放一个键或鼠标按钮， 一个键盘或鼠标事件被产生； 当窗口第一次显示， 一个绘图事件产生， 从而告知最新的可见窗口需要重绘自身。

大多数事件是由于响应用户的动作而产生的， 但还有一些， 比如定时器等， 是由系统独立产生的。

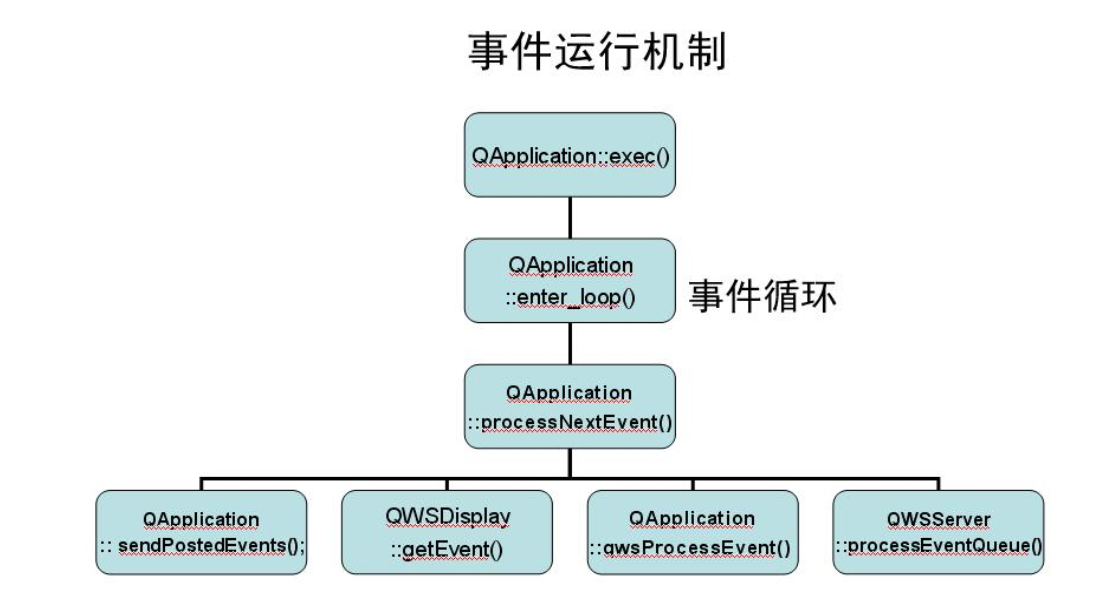


图1.4 Qt事件相应机制

### 数据库操作源码

|  |
| --- |
| #include "mainwindow.h"  #include "ui\_mainwindow.h"  #include "recordtablemodel.h"  #include <QMessageBox>  #include <QDebug>  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   \*作者: jianghj@up-tech.com   \*日期: 2016-09-30   \*描述: 125K演示程序主要代码,此处模拟的人员通道,进出需要刷卡,   \*      125K在实际应用中主要也是这个功能,比如小区的门禁卡.   \*      注意:人为主动刷卡,2.4G是被动刷卡  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  MainWindow::MainWindow(QWidget \*parent) :      QMainWindow(parent),      ui(**new** Ui::MainWindow)  {      ui->setupUi(**this**);  **this**->fillPortsParameters(ui->baudRateBox);//波特率填充  **this**->serialPort = **new** QSerialPort(**this**);      db = **new** Database(**this**);//连接数据库      model = **new** RecordTableModel(**this**);      ui->tableView->setModel(model);      ui->tableView->resizeColumnsToContents();      ui->tableView->horizontalHeader()->setStretchLastSection(**true**);      intValidator = **new** QIntValidator(0, 4000000,**this**);      ui->btn\_connect->setEnabled(**true**);      ui->btn\_refresh->setEnabled(**true**);      ui->btn\_disconnect->setEnabled(**false**);  **this**->on\_btn\_refresh\_clicked();      m125dll = **new** M125Dll();      //关联相关槽函数      connect(ui->baudRateBox, SIGNAL(currentIndexChanged(**int**)),**this**, SLOT(checkCustomBaudRatePolicy(**int**)));      connect(serialPort, SIGNAL(error(QSerialPort::SerialPortError)), **this**, SLOT(handleError(QSerialPort::SerialPortError)));//收到串口错误信息      connect(serialPort, SIGNAL(readyRead()), **this**, SLOT(readData()));//收到串口信息  }    MainWindow::~MainWindow()  {      model->submitAll();  **delete** model;  **delete** db;  **delete** m125dll;  **delete** intValidator;  **delete** serialPort;  **delete** ui;  }  //Baudrate parameter init  **void** MainWindow::fillPortsParameters(QComboBox \*box)  {      box->clear();      box->addItem(QStringLiteral("9600"), QSerialPort::Baud9600);      box->addItem(QStringLiteral("19200"), QSerialPort::Baud19200);      box->addItem(QStringLiteral("38400"), QSerialPort::Baud38400);      box->addItem(QStringLiteral("57600"), QSerialPort::Baud57600);      box->addItem(QStringLiteral("115200"), QSerialPort::Baud115200);      box->addItem(tr("Custom"));  }  /\*\*   \* @brief MainWindow::on\_btn\_connect\_clicked   \* 连接串口   \*/  **void** MainWindow::on\_btn\_connect\_clicked()  {      QString name = ui->serialNameBox->currentText();      QString baud = ui->baudRateBox->currentText().trimmed();  **if**(baud.isEmpty())      {          QMessageBox::critical(**this**, tr("Error"), "波特率输入错误！");  **return** ;      }      serialPort->setPortName(name);      serialPort->setBaudRate(baud.toInt(),QSerialPort::AllDirections);  **if** (serialPort->open(QIODevice::ReadWrite)) {          ui->btn\_connect->setEnabled(**false**);          ui->btn\_disconnect->setEnabled(**true**);          ui->btn\_refresh->setEnabled(**false**);      } **else** {          ui->btn\_connect->setEnabled(**true**);          ui->btn\_refresh->setEnabled(**true**);          ui->btn\_disconnect->setEnabled(**false**);          QMessageBox::warning(**this**,tr("提示"),tr("初始化%1失败！请检查串口是否已经被占用？").arg(name),QMessageBox::Yes);      }  }  /\*\*   \* @brief MainWindow::on\_btn\_disconnect\_clicked   \* 断开连接   \*/  **void** MainWindow::on\_btn\_disconnect\_clicked()  {  **if**(!serialPort->isOpen())  **return** ;      serialPort->close();      ui->btn\_connect->setEnabled(**true**);      ui->btn\_refresh->setEnabled(**true**);      ui->btn\_disconnect->setEnabled(**false**);  }  /\*\*   \* @brief MainWindow::on\_btn\_refresh\_clicked   \* 刷新按钮点击事件   \*/  **void** MainWindow::on\_btn\_refresh\_clicked()  {      QStringList list = getSerialName();      ui->serialNameBox->clear();      ui->serialNameBox->addItems(list);  }  /\*\*   \* @brief MainWindow::checkCustomBaudRatePolicy   \* @param idx combox被选中的索引值   \* 设置自定义波特率   \*/  **void** MainWindow::checkCustomBaudRatePolicy(**int** idx)  {      QComboBox \*box = **dynamic\_cast**<QComboBox\*>(QObject::sender());  **bool** isCustomBaudRate = !box->itemData(idx).isValid();      box->setEditable(isCustomBaudRate);  **if** (isCustomBaudRate) {          box->clearEditText();          box->setValidator(intValidator);      }  }    /\*\*   \* @brief MainWindow::readData   \* 读取串口数据   \*/  **void** MainWindow::readData()  {  **if**(serialPort->bytesAvailable() < 5)  **return**;      QByteArray data = serialPort->readAll();  **if**(m125dll->LF125K\_FrameAnalysis((uint8 \*)(data.data())) == 0)      {          QString tagId = CharStringtoHexString(tr(" "),data.data(),data.length());//获取标签ID          QString **time** = CurrentDateTime();//获取时间  **int** index = model->findRecord(tagId);//查询此标签记录  **if**(index >= 0 )          {              QString text = model->record(index).value(2).toString();  **if**(text == tr("进"))                  model->updateRecord(index,tagId,**time**,tr("出"));  **else**                  model->updateRecord(index,tagId,**time**,tr("进"));          }  **else** {              model->addRecord(tagId,**time**,tr("进"));          }      }  }  /\*\*   \* @brief MainWindow::handleError   \* @param error SerialPortError枚举类,详细请看SerialPortError的定义   \* 处理错误信息   \*/  **void** MainWindow::handleError(QSerialPort::SerialPortError error)  {  **if** (error == QSerialPort::ResourceError) {          QMessageBox::critical(**this**, tr("Critical Error"), serialPort->errorString());  **this**->on\_btn\_disconnect\_clicked();      }  } |

### 125K接口

进行验证读取的数据是否符合 125K 卡号格式， 在 API 的头文件中有介绍分析 125K 帧接口的方法， 头文件中代码如下：

//125K模块上报卡号的起始字节

#define LF125K\_SOF    0x00

//成功定义为0

#define SUCCESS      0

//失败定义为-1

#define FAILURE     -1

**typedef** unsigned **char**   uint8;

//125K模块上报卡号的结构

**typedef** **struct**

{

    uint8 sof;

    uint8 data[3];

    uint8 fcs;    // xor verify

}LF125K\_RspFrame\_t;

**class** M125DLLSHARED\_EXPORT M125Dll

{

**public**:

    M125Dll();

    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    \*函数名        ：LF125K\_FrameAnalysis

    \*描述     ：解析一帧数据是否有误

    \*输入参数   ：

    \*   @frame  const uint8 \* 125K的数据帧

    \*输出参数   ：无

    \*返回值        ：正确返回0，失败返回-1

    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**int** LF125K\_FrameAnalysis(**const** uint8 \*frame);

};

## 思考题

1. 通过试验箱，反复循环读取十张低频电子标签。在读取过程中可能会遇到哪些问 题或发生哪些现象，并分析遇到的这些问题或现象的原因；

发生只有一张卡进行记录，为单卡读取

1. 在利用低频读写器模拟门禁系统中，如何获取读写器发送过来的卡号？请写出相 应的函数体（含注释），并说明函数的调用方法。

|  |
| --- |
| /\*\*   \* @brief MainWindow::readData   \* 读取串口数据   \*/  **void** MainWindow::readData()  {  **if**(serialPort->bytesAvailable() < 5)  **return**;      QByteArray data = serialPort->readAll();  **if**(m125dll->LF125K\_FrameAnalysis((uint8 \*)(data.data())) == 0)                  // call function in m125k DLL      {          QString tagId = CharStringtoHexString(tr(" "),data.data(),data.length());   // get tag ID          QString **time** = CurrentDateTime();                                           // get cuttent time  **int** index = model->findRecord(tagId);                                       // findRecord by tagID  **if**(index >= 0 )          {              QString text = model->record(index).value(2).toString();                // set time  **if**(text == tr("进"))                  model->updateRecord(index,tagId,**time**,tr("出"));  **else**                  model->updateRecord(index,tagId,**time**,tr("进"));          }  **else**          {              model->addRecord(tagId,**time**,tr("进"));          }      }  } |

## 实验体会与总结

实验过程中，对试验箱进行了相应的熟悉，在系统的相应配置过程中出现了一些问题，试验箱对应的整体实验分为两部分，可以通过本地PC进行宿主机进行reader的相应操控，也可以使用试验箱板载的arm进行相应的操控。

如果使用ARM进行操控，需要使用交叉编译技术，实验环境为32bit的Qt Version。 Qt本身为跨平台，但是出于DLL限制，无法使用64bit Qt进行编译，但是Linux上32 Bit Qt已经停止维护，故从官网上的开源仓库获取较老的版本。

整个代码较为清晰，各个工具类分的比较详细，但是操作函数完全位于mainwindow类，较为混乱且拥挤，对于DLL封装切不提供多平台版本和源码，对于跨平台不友好。

整个试验箱更类似于一个教学平台，其中不同reader的相应切换没有提供相应的接口，且只能依靠物理方式切换，有些不方便。

实验过程中，查询了相关文档，对于系统串口通信方式和系统调用过程有了更深的理解。

# 实验二 高频读写器实验(ISO14443A)

## 实验目的

1. 通过本次实验了解高频读写器的基本原理，学会如何使用高频读写器，掌握 串口命令参数的意义和设置方式。
2. 阅读和了解 ISO14443A 协议的主要内容，进一步加深对 S50 卡的存储结构和ISO14443A 协议的理解，掌握 ISO14443A 协议的常用命令的含义和用法。
3. 通过高频读写器的实验，掌握对 S50 卡各个扇区数据的读写方法，并熟悉高频读写器（ISO14443A）API函数。

## 实验内容与要求

### 实验内容

1. 完成 ISO14443A 协议下标签寻卡、唤醒、休眠实验；
2. 完成 ISO14443A 协议下标签内存读写实验；
3. 完成 ISO14443A 协议下标签一卡通实验；
4. 熟悉和了解高频 HF1356M 14443A 开发实例，掌握高频读写器（14443A）API 函数，并通过编程实现对 S50 卡的指定内存进行数据读写、充值、减值。

### 实验要求

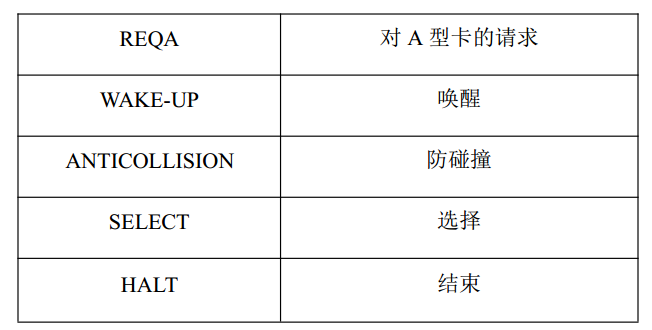
1. 通过试验箱，学会对 ISO14443A 协议下标签进行寻卡、唤醒、休眠操作， 掌握寻卡、唤醒、休眠的命令代码、命令功能和命令数据包构成，并分析唤醒、 休眠命令对标签寻卡所产生的影响。
2. 通过试验箱，学会对 ISO14443A 协议下标签指定内存的数据读写。掌握块 3 存取控制位对本扇区其他块的读写操作控制逻辑。掌握密码 A 和密码 B 的 加载方法以及对不同块的读写控制逻辑。
3. 学会通过试验箱对 ISO14443A 协议下标签进行加值减值操作；
4. 掌握高频读写器API函数的调用方法，并能够通过编程实现对ISO14443A 协议下标签数据的读写操作以及加值减值操作。

## 实验过程与结果

### ISO14443A协议理解

1. PICC状态集
   1. 调电状态： 由于没有足够的载波能量， PICC 没有工作， 也不能发送反射波。
   2. 闲置状态： 在这个状态时， PICC 已经上电， 能够解调信号， 并能够识别有效的 REQA 和 WAKE-UP命令。
   3. 准备状态： 本状态下， 实现位帧的防碰撞算法或其它可行的防碰撞算法。
   4. 激活状态： PCD 通过防碰撞已经选出了单一的卡。
2. 命令集

表2.1 PCD 用于管理与 PICC 之间通信的命令



1. 协议激活

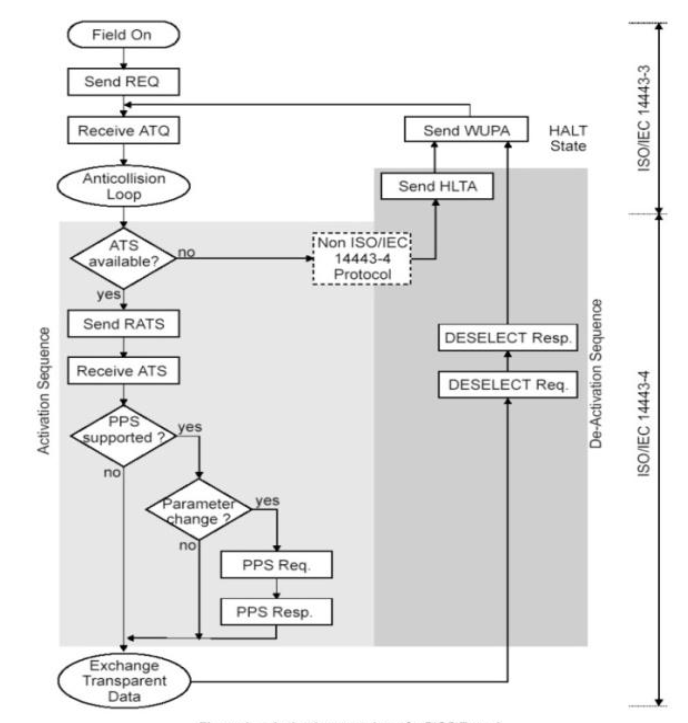


图2.1 A卡协议激活图

### ISO14443A寻卡、唤醒、休眠实验



图2.2 寻卡、唤醒、休眠实验图

### ISO14443A标签内存读写实验；

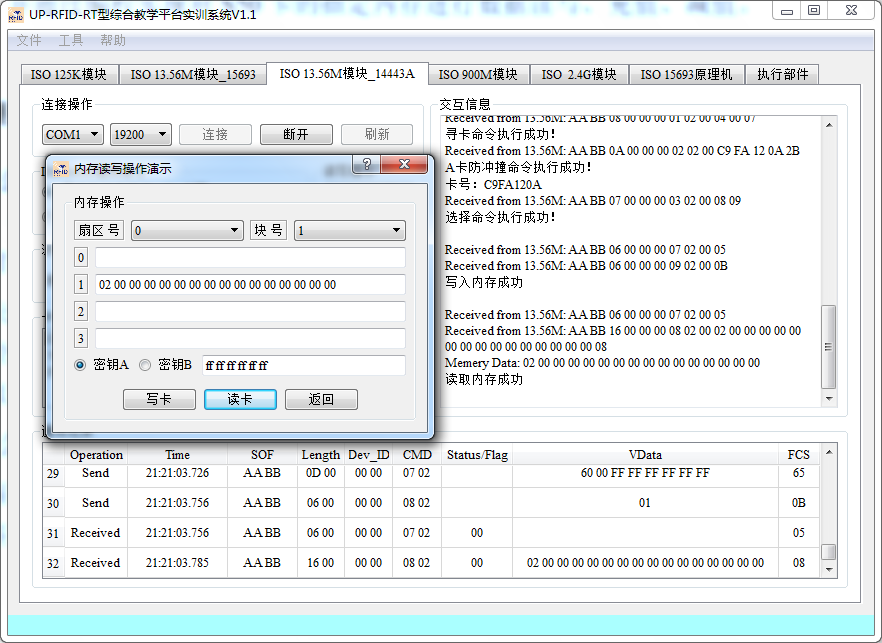


图2.3 读卡实验图

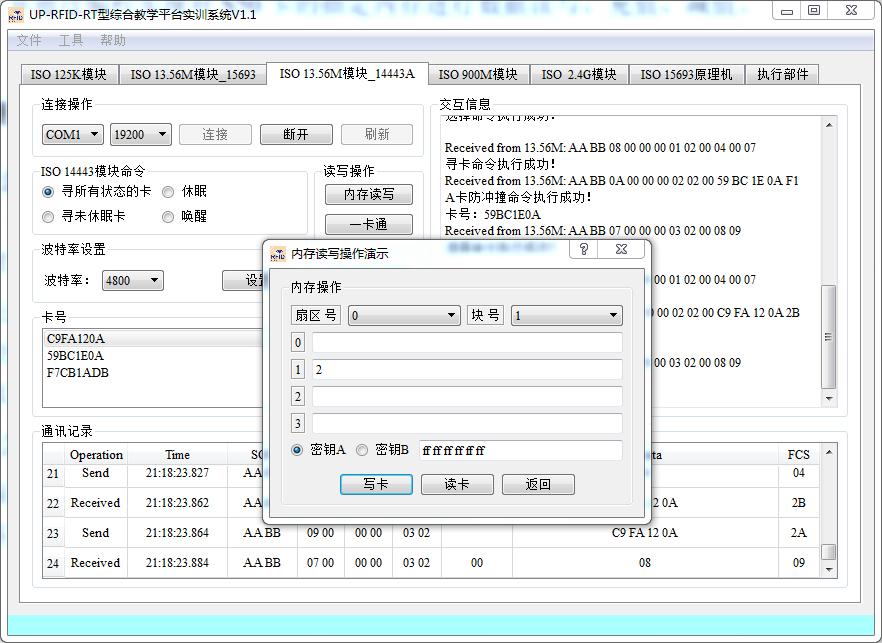


图2.4 写卡实验图

3、完成 ISO14443A 协议下标签一卡通实验；



图2.5 一卡通充值图

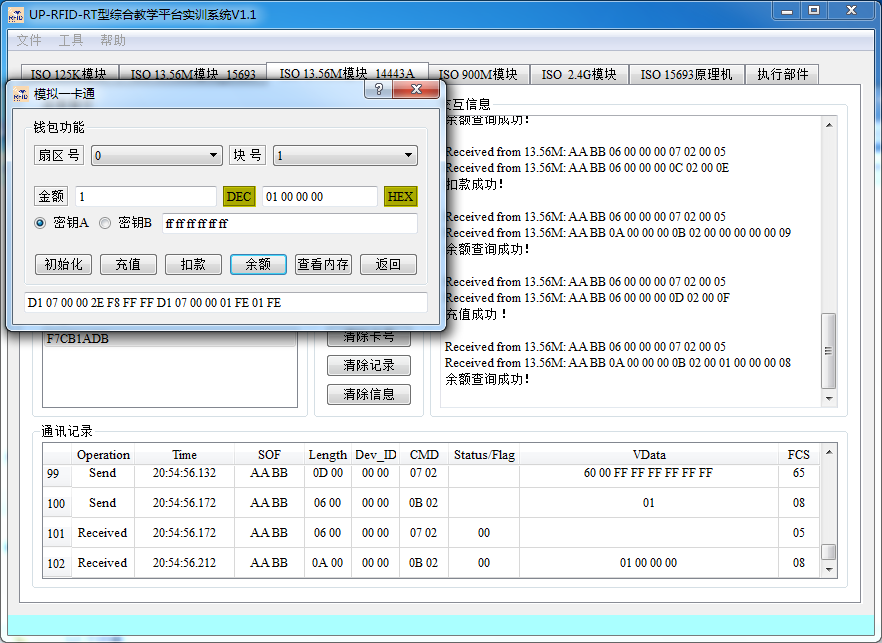


图2.6 一卡通查询余额图

## 核心源码说明

### 系统架构

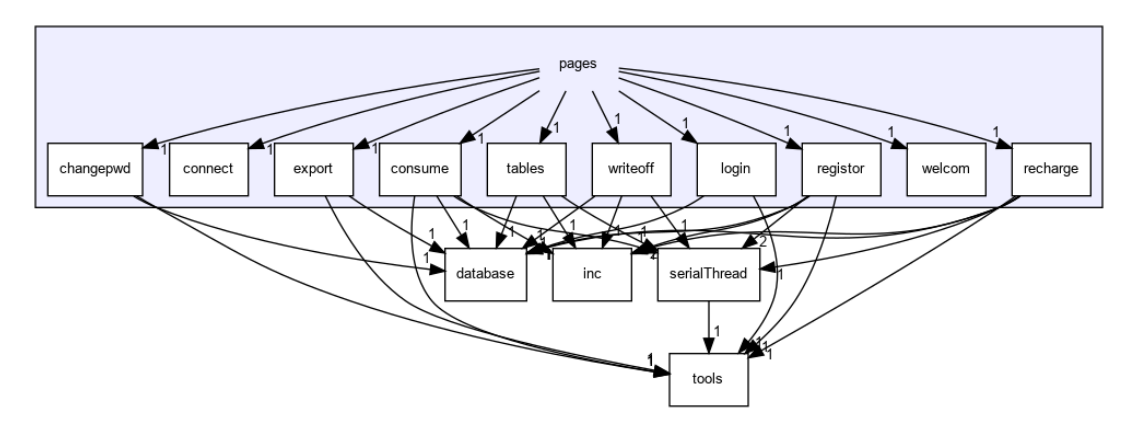


图2.7 工具类调用关系图

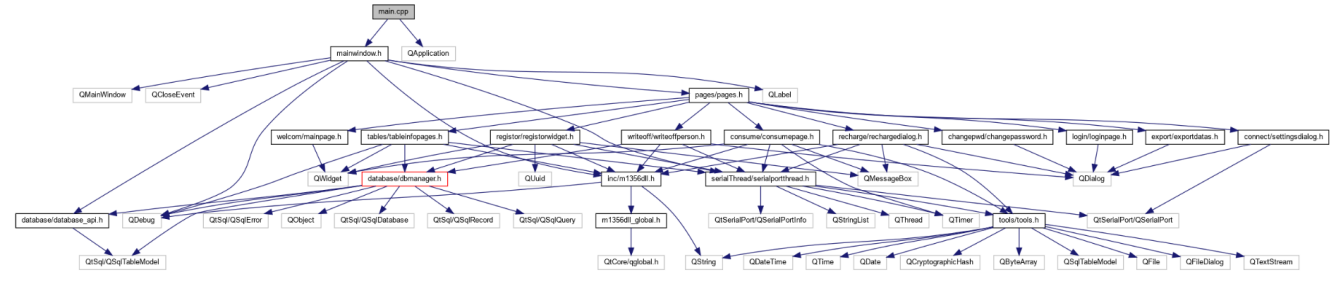


图2.8 系统架构图

### DBManager类型声明

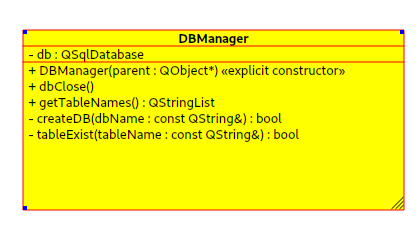


图2.9 DBManager接口声明图

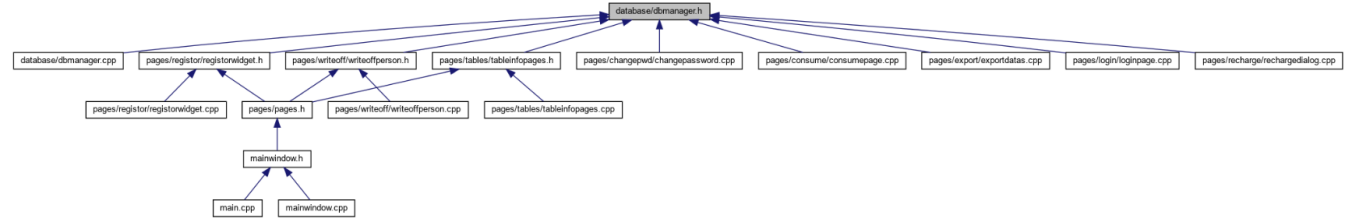


图2.10 DBManager接口调用图

**class** DBManager : **public** QObject

{

    Q\_OBJECT

**public**:

**explicit** DBManager(QObject \*parent = 0);

**void** dbClose(); //关闭数据库

    QStringList getTableNames();//获取所有表的表名

**private**:

    QSqlDatabase db;//sqlite database

    //创建数据库

**bool** createDB(**const** QString &dbName);

    //检测该表是否存在

**bool** tableExist(**const** QString &tableName);

signals:

**public** slots:

};

### DBManager方法实现

DBManager::DBManager(QObject \*parent) : QObject(parent)

{

**if**(**this**->createDB(DATABASE\_NAME))               //create DB by DB name

    {

**if**(!**this**->tableExist(TABLE\_NAME\_PERSON))    //op PersonTableModel

        {

            PersonTableModel p;

            p.createTable();

        }

**if**(!**this**->tableExist(TABLE\_NAME\_RECORD))

        {

            RecordTableModel r;

            r.createTable();

        }

**if**(!**this**->tableExist(TABLE\_NAME\_REGISTER))  //op RegisterTableModel

        {

            RegisterTableModel r;

            r.createTable();

        }

**if**(!**this**->tableExist(TABLE\_NAME\_WRITEOFF))

        {

            WriteOffTableModel w;

            w.createTable();

        }

**if**(!**this**->tableExist(TABLE\_NAME\_ADMIN))     //op AdminTableModel

        {

            AdminTableModel a;

            a.createTable();

        }

**if**(!**this**->tableExist(TABLE\_NAME\_RECHARGE))  //op RechargeTableModel

        {

            RechargeTableModel r;

            r.createTable();

        }

    }

}

/\*\*

 \* @brief RecordTableModel::createDB

 \* @param dbName 数据库名称

 \* 用于创建数据库

 \*/

**bool** DBManager::createDB(**const** QString &dbName) // createDB by name

{

    db = QSqlDatabase::addDatabase("QSQLITE");  //sqlite DB

    db.setDatabaseName(dbName);

**if**(db.open())

    {

        qDebug()<<dbName<<" create success";

**return** **true**;

    }

**else**

    {

        qDebug()<<dbName<<" create failed!";

**return** **false**;

    }

}

/\*\*

 \* @brief DBManager::tableExist

 \* @param tableName 表名

 \* @return  如果存在返回true，否则返回false

 \* 用于判断表是否存在

 \*/

**bool** DBManager::tableExist(**const** QString &tableName)    //judge DB exist

{

**int** count = 0 ;

    QString sqlText = QObject::tr("select count(\*) from sqlite\_master where type='table' and name='%1'").arg(tableName);

    QSqlQuery query;        // judge by select ans

    query.exec(sqlText);

**if**(query.next())

        count = query.value(0).toInt();

**if**(count > 0)

**return** **true**;

**else**

**return** **false**;

}

/\*\*

 \* @brief DBManager::getTableNames

 \* @return  数据库表集合

 \* 获取数据库所有表的表名

 \*/

QStringList DBManager::getTableNames()

{

    QStringList tables; // get table by name

    QString sqlText = QObject::tr("select name from sqlite\_master where type='table' order by name;");

    QSqlQuery query;

    query.exec(sqlText);

**while**(query.next())

    {

        tables << query.value(0).toString();

    }

**return** tables;

}

/\*\*

 \* @brief DBManager::dbClose

 \* 关闭数据库

 \*/

**void** DBManager::dbClose()

{

**if**(db.isOpen())

        db.close();

}

**static** DBManager \*dbManager;

/\*\*

 \* @brief DB\_Init

 \* 创建数据库管理对象实体，初始化数据库

 \*/

**void** DB\_Init()

{

    dbManager = **new** DBManager();

}

/\*\*

 \* @brief DB\_Close

 \* 关闭数据库

 \*/

**void** DB\_Close()

{

    dbManager->dbClose();

}

/\*\*

 \* @brief getTableNames

 \* @return QStringList型的表名集合

 \* 获取数据库所有表的表名

 \*/

QStringList getTableNames()

{

**return** dbManager->getTableNames();  // wrap for get table

}

### Model实现方式

对于不同的对象采用不同的Model进行设计实现，其实现方式如下

RegisterTableModel::RegisterTableModel(QObject \*parent) : QSqlTableModel(parent)

{

tableName = TABLE\_NAME\_REGISTER;

header<<"卡号"<<"用户编号"<<"时间"<<"信息备注";

}

### 串口通信

程序作为数据库和串口相应的交互，需要进行串口通信单独类的剥离

SerialPortThread::SerialPortThread(QObject \*parent) : QThread(parent)

{

    serialIsOpen = **false**;

    Stop = **false**;

    timer = **new** QTimer(**this**);

    connect(timer,SIGNAL(timeout()),**this**,SLOT(onError()));

    connect(**this**,SIGNAL(hasResponse()),**this**,SLOT(stopTimer()));

}

利用QThred创建线程，作为独立的串口通信线程，利用定时器机制进行轮训

## 思考题

1. S50 卡共有 16 个扇区，请问第 4 扇区的绝对块地址号是多少？请详细说明计算的方法和依据。

S50把1K字节的容量分为16个扇区(Sector0-Sector15)，每个扇区包括4个数据块(Block0-Block3，我们也将16个扇区的64个块按绝对地址编号为0~63)，每个数据块包含16个字节(Byte0-Byte15)，64\*16=1024。所以，第4扇区绝对地址号是：3\*4=12，为12。

1. S50 卡第 1 扇区第 0 块是否可读写？为什么？

S50从0开始起标，故第1扇区可以读写，因为只有第0扇区，第0块号为卡序列号，不可写(只读)。

1. “S50 卡共有 16 个扇区，每个扇区由 4 块组成，第 4 块为控制块，其余 三块为数据块，都可用于存储数据”。这句话正确吗？如果不正确，请改正。

不正确，控制块也可以用于存储数据。应该改正为S50卡除了0号扇区以外的每个扇区其他三块都为数据块，都可用于存储数据，0号扇区的第一块是用来存储卡的序列号。

1. S50 卡的数据块用于存储数据时，可以有哪几种用途？

所有扇区都由3个块组成，每个块由16字节用于存储数据(扇区0只有两个数据块，一个只读的厂商数据块)。数据块可以设置为：读写块，例如用于非接触门禁管理，有效命令： read, write;数值块，例如用于电子钱包，允许执行电子钱包功能(有效的命令是:读、写增量、减量、恢复、转移)。数值块有一个固定的数据格式允许错误检测和校正和备份管理。

1. 如何将一张空白的 S50 卡初始化成电子钱包？

可将卡放入读卡器，点击寻卡功能，找到该卡，执行读卡、写卡、初始化卡等功能，并进行相应的功能设置，就能将卡初始化为电子钱包，并将金额初始化为0。

## 实验体会与总结

实验过程中，对于ISO14443A协议有了更深的理解，由于实验提供了对应的一卡通管理系统，故可以直接将卡进行一卡通化，实际上，本质上仍然为定义了自己的数据存储方式，进行了相应的数据存储形式定义，与普通的信息管理系统无异，仅仅是存储形式在内存，硬盘还是RFID卡的区别。

此实验告诉我们，仅仅需要从数据结构的角度理解RFID卡，对于主键(ID)等进行保护控制，其余数据结构可以自己定义，同时满足相应的读写接口。

存取权限控制上，有硬件控制方式(reader/tag)设定权限，软件设置(界面仅仅提供允许用户更改的相应权限)。实际上，硬件仅仅能对TagID等进行控制，对于需要灵活更改的部分，需要考虑加密进行软件控制。

# 实验三 高频读写器实验(ISO15693)

## 实验目的

1. 通过本次实验了解高频读写器的基本原理，学会如何使用高频读写器，掌握 系统命令参数的意义和设置方式。
2. 进一步加深对ISO15693协议下标签的存储结构以及ISO15693协议的理解。 通过读写器试验箱，掌握对 ISO15693 协议下标签读写操作以及 ISO15693 协议 标签存储结构的功能，并熟悉高频读写器 API 函数。

## 实验内容与要求

### 实验内容

1. 完成 ISO15693 协议下的单标签和多标签手工寻卡和自动寻卡；
2. 根据标签内存地址，完成 ISO15693 协议下标签指定地址的数据读写实验；
3. 根据标签内存地址，完成 ISO15693 协议下标签指定地址范围的内存数据读取实验；
4. ISO15693 协议的命令，完成标签静默状态设置、重置到准备状态、标签 选择命令实验；
5. 完成 ISO15693 协议下标签 DSFID、AFI 的读写和块安全位的读取实验；
6. 熟悉和了解高频 HF1356M 15693 开发实例，掌握高频读写器 API 函数， 并通过编程实现 ISO15693 协议下标签的读写功能。

### 实验要求

1. 学会通过试验箱对 ISO15693 协议下标签指定内存地址的数据进行读写操作；
2. 理解和掌握应用族标识符（AFI）、数据存储格式标识符（DSFID）以及锁的基本概念和含义；
3. 掌握高频读写器 API 函数的调用方法，并能够通过编程实现对 ISO15693 协议下标签数据的读写控制。

## 实验过程与结果

### 读取状态设置



图3.1 修改波特率为38400图

图3.2 修改波连接特率为38400图



图3.3 断电后38400连接失败图



图3.4 恢复19200连接成功图

### ISO15693单标签/多标签手工寻卡和自动寻卡



图3.5 单标签寻卡图



图3.6 多标签寻卡图

### ISO15693指定地址的数据读写实验

根据标签内存地址，完成 ISO15693 协议下标签指定地址的数据读写实验；

### ISO15693指定地址范围的内存数据读取实验

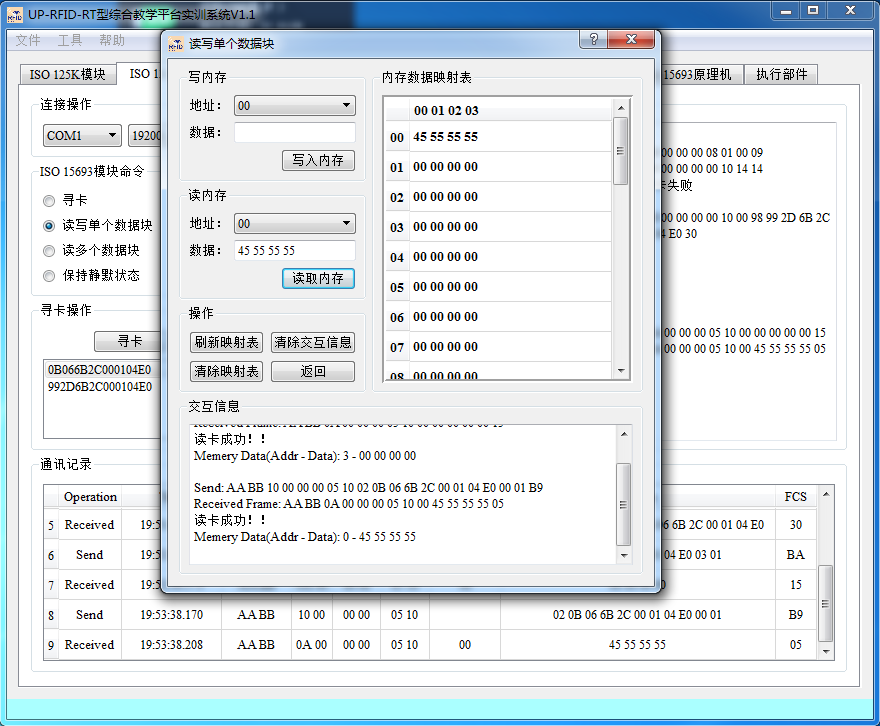


图3.7 指定地址范围读卡(00)实验图

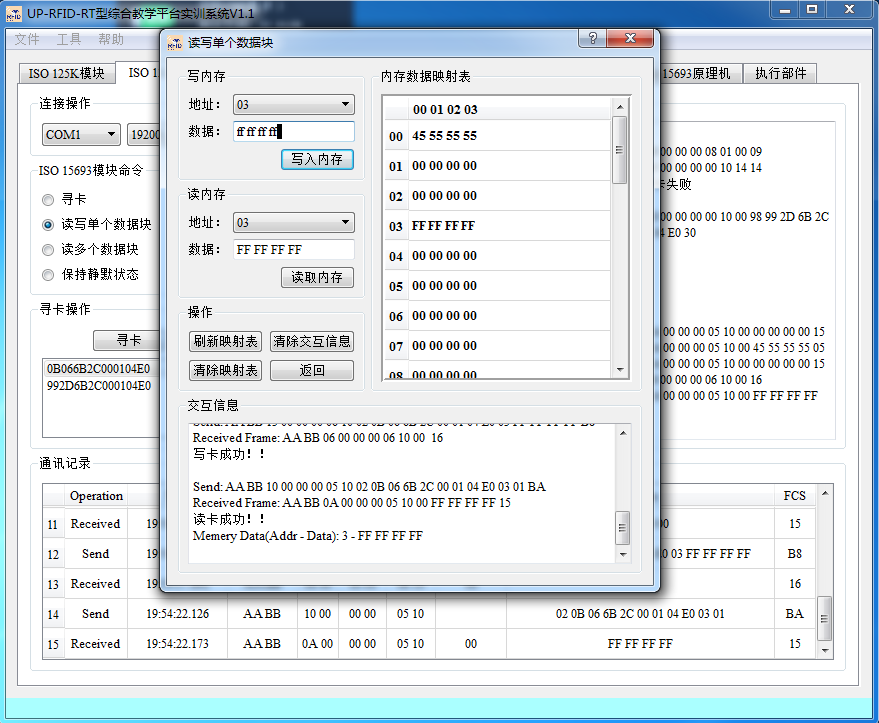


图3.8 读写(03)实验图

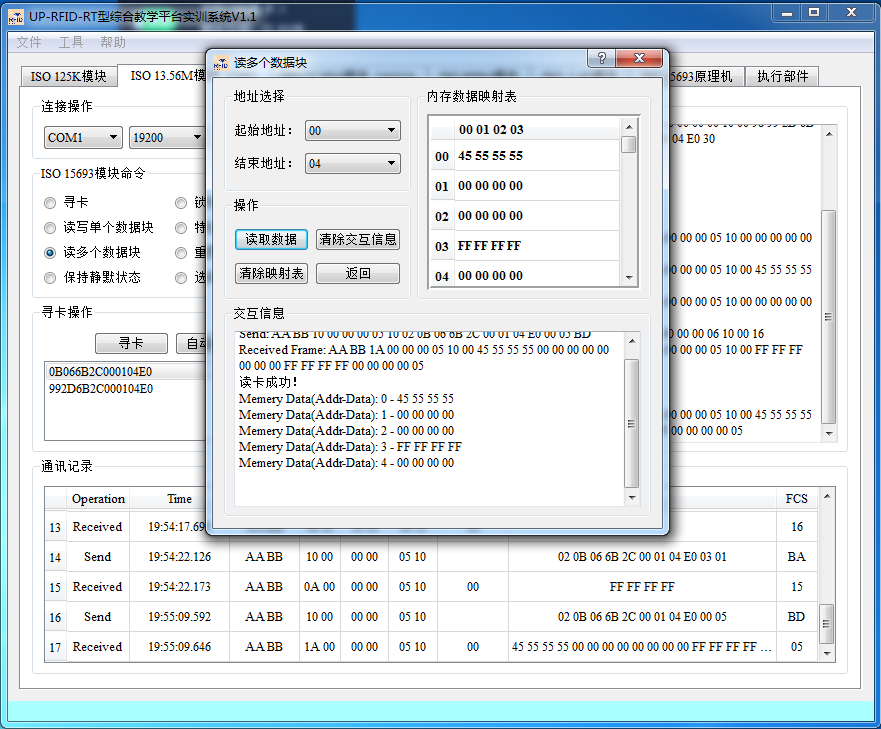


图3.9 读多个数据块实验图

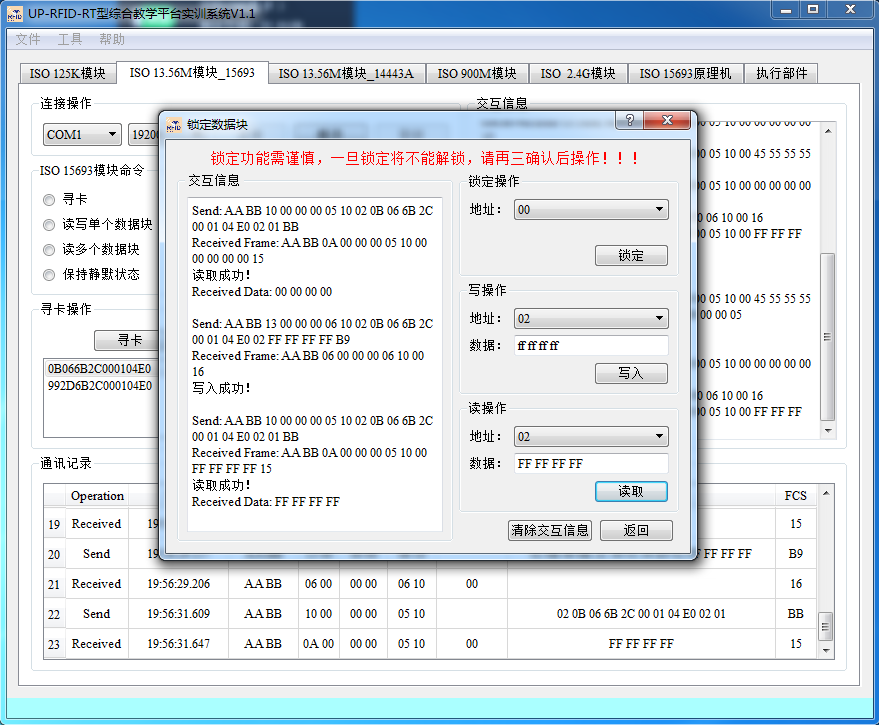


图3.10 读写(02)实验图

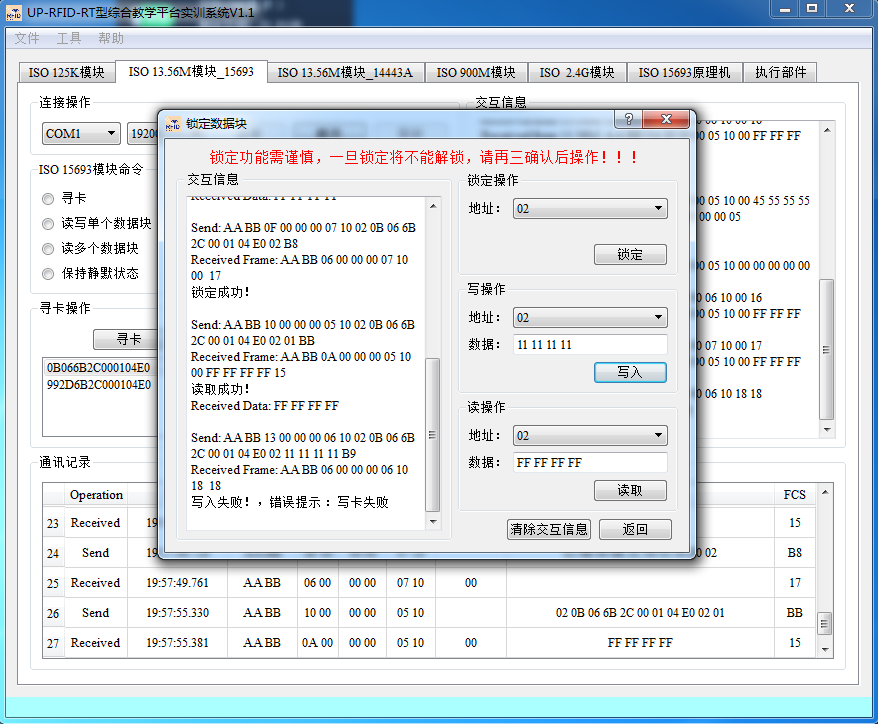


图3.11 锁定02后可读不可写实验图

### ISO15693协议的命令

1. 完成标签静默状态设置



图3.12 静默实验

1. 重置到准备状态



图3.13 恢复实验

### ISO15693 协议下标签 DSFID、AFI 的读写和块安全位的读取实验

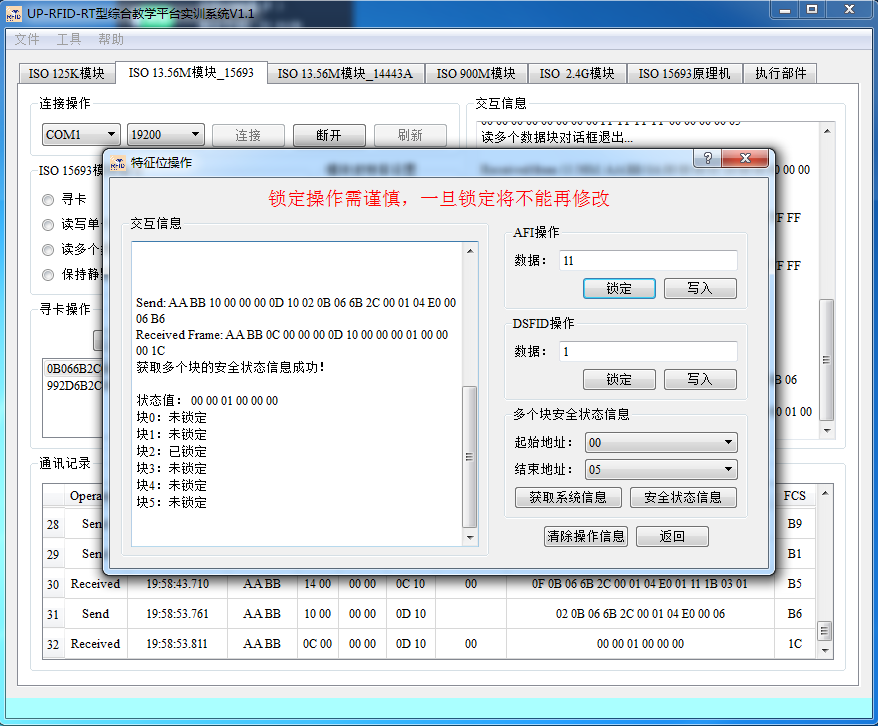


图3.14 安全状态图

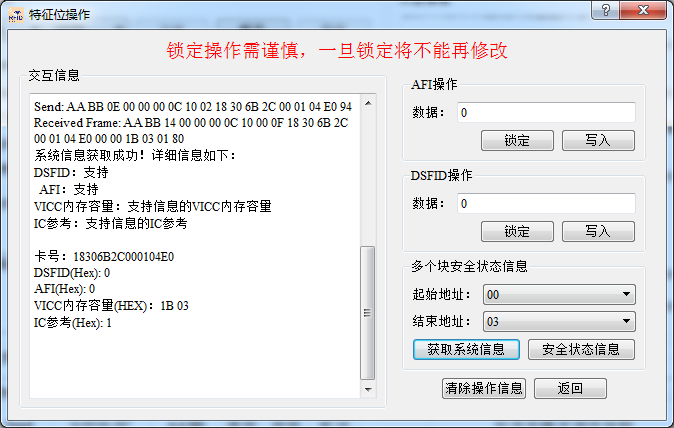


图3.15 DSFID、AFI 的读写和块安全位的读取实验

### ISO15693 协议下标签的读写功能

熟悉和了解高频 HF1356M 15693 开发实例，掌握高频读写器 API 函数，并通过编程实现 ISO15693 协议下标签的读写功能



图3.16 开发实例测试图

## 核心源码说明

### 系统架构图

系统在BooksManage中进行系统相应槽函数的刷新

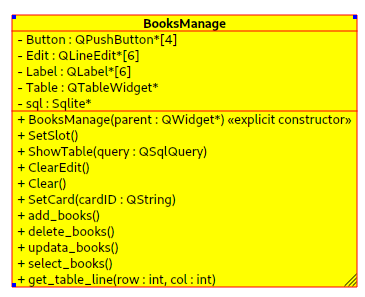


图3.17 BooksManage接口图

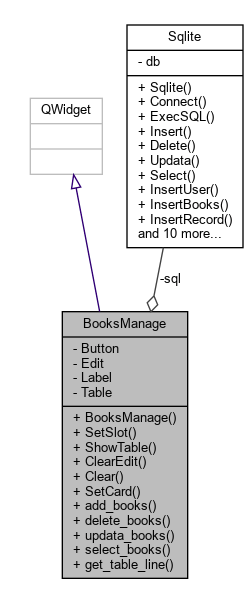


图3.18 BooksManage与DB交互图

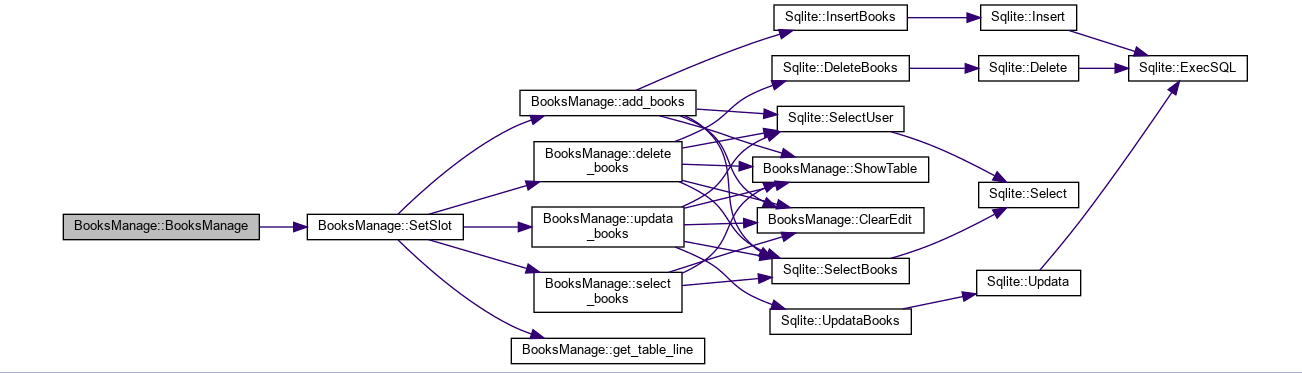


图3.19 BooksManage调用图

### BooksManage接口定义

DBManager::DBManager(QObject \*parent) : QObject(parent)

{

**if**(**this**->createDB(DATABASE\_NAME))               //create DB by DB name

    {

**if**(!**this**->tableExist(TABLE\_NAME\_PERSON))    //op PersonTableModel

        {

            PersonTableModel p;

            p.createTable();

        }

**if**(!**this**->tableExist(TABLE\_NAME\_RECORD))

        {

            RecordTableModel r;

            r.createTable();

        }

**if**(!**this**->tableExist(TABLE\_NAME\_REGISTER))  //op RegisterTableModel

        {

            RegisterTableModel r;

            r.createTable();

        }

**if**(!**this**->tableExist(TABLE\_NAME\_WRITEOFF))

        {

            WriteOffTableModel w;

            w.createTable();

        }

**if**(!**this**->tableExist(TABLE\_NAME\_ADMIN))     //op AdminTableModel

        {

            AdminTableModel a;

            a.createTable();

        }

**if**(!**this**->tableExist(TABLE\_NAME\_RECHARGE))  //op RechargeTableModel

        {

            RechargeTableModel r;

            r.createTable();

        }

    }

}

/\*\*

 \* @brief RecordTableModel::createDB

 \* @param dbName 数据库名称

 \* 用于创建数据库

 \*/

**bool** DBManager::createDB(**const** QString &dbName) // createDB by name

{

    db = QSqlDatabase::addDatabase("QSQLITE");  //sqlite DB

    db.setDatabaseName(dbName);

**if**(db.open())

    {

        qDebug()<<dbName<<" create success";

**return** **true**;

    }

**else**

    {

        qDebug()<<dbName<<" create failed!";

**return** **false**;

    }

}

/\*\*

 \* @brief DBManager::tableExist

 \* @param tableName 表名

 \* @return  如果存在返回true，否则返回false

 \* 用于判断表是否存在

 \*/

**bool** DBManager::tableExist(**const** QString &tableName)    //judge DB exist

{

**int** count = 0 ;

    QString sqlText = QObject::tr("select count(\*) from sqlite\_master where type='table' and name='%1'").arg(tableName);

    QSqlQuery query;        // judge by select ans

    query.exec(sqlText);

**if**(query.next())

        count = query.value(0).toInt();

**if**(count > 0)

**return** **true**;

**else**

**return** **false**;

}

/\*\*

 \* @brief DBManager::getTableNames

 \* @return  数据库表集合

 \* 获取数据库所有表的表名

 \*/

QStringList DBManager::getTableNames()

{

    QStringList tables; // get table by name

    QString sqlText = QObject::tr("select name from sqlite\_master where type='table' order by name;");

    QSqlQuery query;

    query.exec(sqlText);

**while**(query.next())

    {

        tables << query.value(0).toString();

    }

**return** tables;

}

/\*\*

 \* @brief DBManager::dbClose

 \* 关闭数据库

 \*/

**void** DBManager::dbClose()

{

**if**(db.isOpen())

        db.close();

}

**static** DBManager \*dbManager;

/\*\*

 \* @brief DB\_Init

 \* 创建数据库管理对象实体，初始化数据库

 \*/

**void** DB\_Init()

{

    dbManager = **new** DBManager();

}

/\*\*

 \* @brief DB\_Close

 \* 关闭数据库

 \*/

**void** DB\_Close()

{

    dbManager->dbClose();

}

/\*\*

 \* @brief getTableNames

 \* @return QStringList型的表名集合

 \* 获取数据库所有表的表名

 \*/

QStringList getTableNames()

{

**return** dbManager->getTableNames();  // wrap for get table

}

### 字符串转换函数

Qt采用UTF8编码，需要将QString转化成为HEXString

///普通字符串转为16进制字符串

QString CharStringtoHexString(QString space, **const** **char** \* src, **int** start, **int** end)

{

    QString hex = "";

**if**(space == NULL)

    {

**for**(**int** i = start ; i < end ; i ++)

        {

            hex += QString("%1").arg(src[i]&0xFF,2,16,QLatin1Char('0'));

        }

**return** hex.toUpper();

    }

**else**

    {

**for**(**int** i = start ; i < end ; i ++)

        {

            hex += space + QString("%1").arg(src[i]&0xFF,2,16,QLatin1Char('0'));

        }

**return** hex.right(hex.length() - space.length()).toUpper();

    }

}

### 发送请求帧

需要对帧长进行计算，并对相应的控制位进行设置，作为cmd帧请求

//发送请求帧

uint8\* M1356Dll::RC632\_SendCmdReq(uint16 cmd, **const** uint8 \*data, uint16 len)

{

    uint8 \*p;

    uint16 frameLen, uartdatalen;

    frameLen = len+9;   // 9 = sop(2)+len(2)+nc(2)+cmd(2)+fcs(1)

    uartdatalen = frameLen;

**for**(uint16 i=0; i<len; i++)

**if**(data[i] == 0xAA)  uartdatalen++;

    p = **new** uint8[uartdatalen+2];  // p[0] is used return frameLen

    p[0] = LO\_UINT16(uartdatalen);

    p[1] = HI\_UINT16(uartdatalen);

    p[2] = LO\_UINT16(RC632\_FRAME\_SOP);

    p[3] = HI\_UINT16(RC632\_FRAME\_SOP);

    p[4] = LO\_UINT16(frameLen-4);  // 4 = sop(2)+len(2)

    p[5] = HI\_UINT16(frameLen-4);  // 4 = sop(2)+len(2)

    p[6] = 0;  // NC

    p[7] = 0;  // NC

    p[8] = LO\_UINT16(cmd);

    p[9] = HI\_UINT16(cmd);

    //memcpy(p+10, data, len);

**int** k=0;

**for**(**int** j=0; j<len; j++, k++){

        p[10+k] = data[j];

**if**(data[j]==0xAA)  p[10+ ++k] = 0x00;

    }

    p[10+k] = RC632\_UartCalcFCS(p+5, uartdatalen-4);

**return** p;

}

### 解析返回帧

其中，对于不同返回帧的校验进行了处理，qDebug()用于返回值错误的输出，但此处并没有进行相应的错误处理

//解析某一帧数据

uint8 M1356Dll::RC632\_AnalysisFrame(uint8 \*frame, uint16 cmd)

{

    uint16 sof = BUILD\_UINT16(frame[RC632\_RPC\_SOP], frame[RC632\_RPC\_SOP+1]);

    uint16 len = BUILD\_UINT16(frame[RC632\_RPC\_LEN], frame[RC632\_RPC\_LEN+1]);

    uint16 rxCmd = BUILD\_UINT16(frame[RC632\_RPC\_CMD], frame[RC632\_RPC\_CMD+1]);

**if**(sof == RC632\_FRAME\_SOP)

    {

**int** tempLen = len,i=0,j=0;

**while**(tempLen > i){  //

            frame[RC632\_RPC\_NC+i] = frame[RC632\_RPC\_NC+j];

**if**(frame[RC632\_RPC\_NC+i] == 0xAA)  j++;

            i++; j++;

        }

**if**(rxCmd == cmd)

        {

            uint8 fcs = RC632\_UartCalcFCS(&frame[RC632\_RPC\_LEN+1], len);

**if**(fcs == frame[RC632\_RPC\_LEN+len+1])

**return** frame[RC632\_RPC\_DAT];

**else**{

                qDebug()<<"[ERR-RC632 Frame FCS ERR]";

                qDebug("%02X!=%02X", fcs, frame[RC632\_RPC\_LEN+len+1]);

            }

        }

**else**

        {

            qDebug()<<"[ERR-RC632 Frame CMD ERR]";

        }

    }

**else**

    {

        qDebug()<<"[ERR-RC632] Frame SOP ERR";

    }

**return** RC632\_FAULT255;  // frame fcs error

}

### 错误解析

/\*\*

 \* @brief GET\_ERROR\_STRING

 \* @param error 错误码

 \* @return  错误信息

 \* 获取错误码

 \*/

QString M1356Dll::GET\_ERROR\_STRING(QString error)

{

    QString result="未知错误";

**bool** ok = **true**;

**if**(error.length() == 2)

    {

**int** id = error.toInt(&ok,16);

**switch** (id) {

**case** RC632\_FAULT10:

            result = "通用错误";

**break**;

**case** RC632\_FAULT11:

            result = "不支持该命令";

**break**;

**case** RC632\_FAULT12:

            result = "无卡";

**break**;

**case** RC632\_FAULT13:

            result = "射频基站损坏";

**break**;

**case** RC632\_FAULT14:

            result = "射频基站损坏";

**break**;

**case** RC632\_FAULT20:

            result = "寻卡失败";

**break**;

**case** RC632\_FAULT21:

            result = "卡复位失败";

**break**;

**case** RC632\_FAULT22:

            result = "密码验证失败";

**break**;

**case** RC632\_FAULT23:

            result = "读卡失败";

**break**;

**case** RC632\_FAULT24:

            result = "写卡失败";

**break**;

**case** RC632\_FAULT100:

            result = "未响应";

**break**;

**default**:

            result = "未知错误";

**break**;

        }

    }

**return** result;

}

## 思考题

1. 什么是 AFI？AFI 如何编码？在通过编程对 AFI 进行读写、锁定时，其对应 ISO15693 的协议命令代码、上位机对读写器的命令代码和数据包分别是怎 么样的？

AFI是应用标示的简称，在ISO15693的电子标签中就有AFI写和锁。用户可以自己写一个关键字作为标签的类别。AFI被编码在一个字节里，由两个半字节组成，AFI的高半位字节，用于编码一个特定的或所有应用族，AFI的低半位字节用于编码一个特定的或所有应用子族。子族不同于0的，有其自己的所有权。

1. 什么是DSFID？在通过编程对DSFID进行读写、锁定时，其对应ISO15693 的协议命令代码、上位机对读写器的命令代码和数据包分别是怎么样的？

特殊功能 DSFID（数据存储格式标识符）可用来表示数据在存储器中的存储结构，具 体内容请自己查阅相关文档。数据存储格式标识符（DSFID）数据存储格式标识符指出了数据在内存中是怎样构成的。DSFID 被相应的命令编程和锁定。DSFID 被编码在一个字节里。DSFID 允许即时知 道数据的逻辑组织。假如标签不支持 DSFID 的编程，标签将以值“0”作为应答。

1. ISO15693 协议的电子标签 ID 有何特点？

ISO/IEC 15693 协议标准的高频 RFID 无源 IC 卡，专为供应链与运筹管理应用所设计，具有高度防冲突与长距离运作等优点，适合于高速、长距离应用。

工作频率范围为13.56MHz±7KHz，工作磁场中工作场最小值0.15A/m,最大值5A/m，支持多种调制方式，支持防冲突和传输协议。

## 实验体会与总结

通过这次实验，通过试验箱对 ISO15693 协议下标签指定内存地址的数据进行读写操作，加深了对应用族标识符（AFI）、数据存储格式标识符（DSFID）以及锁理解，了解高频读写器 API 函数的调用方法。

实验代码部分，通过DLL进行GPIO串口通信，采用DB工具类进行数据库操作，同时将种不同身份的类别进行隔离进行权限管理。

实验代码对于错误处理上，并没有进行处理，而是直接将错误输出，系统的鲁棒性不够，将错误处理交给了用户，可以采取重新发送指令再次读取，或者提供相应的操作方法。

代码大量使用qDebug()函数，在实际过程中，系统最终无法呈现某些错误信息，可以考虑将qDebug()输出为log进行记录。

# 实验四 超高频读写器实验

## 实验目的

1. 通过本次实验了解超高频读写器的基本原理，学会如何使用超高频读写器， 掌握超高频读写器和标签参数的含义和设置方法。
2. 进一步加深对 Gen2 协议下标签的存储结构以及 Gen2 协议的理解。通过读写 器试验箱，掌握对 Gen2 协议下标签读写操作，并熟悉超高频读写器 API 函数的 调用。

## 实验内容与要求

### 实验内容

1. 超高频读写器的基本认知，完成超高频读写器频率和功率读取和设置实验；
2. 完成 Gen2 协议下单标签和多标签识别实验；
3. 执行 Gen2 协议下单命令操作实验，并分别对 EPC 标签各个存储区进行 读写擦除操作试验；
4. 熟悉和了解超高频 UHF-900M 开发实例，掌握超高频读写器 API 函数， 并通过编程实现 Gen2 协议下标签的读写功能。

### 实验要求

1. 学会通过试验箱对 Gen2 协议下标签指定存储区的数据读写；
2. 理解和掌握 Gen2 协议下标签存储器结构的特点及含义；
3. 通过多次设置读写器功率，不断移动电子标签与读写器之间的距离，分析理解读写器功率和频率对电子标签读写的影响；
4. 理解和掌握访问密码的用途和使用方法；
5. 掌握超高频读写器 API 函数的调用方法，并能够通过编程实现对 Gen2 协议下标签数据的读写控制。

## 实验过程与结果

### 读写状态设置



图4.1 读取版本号和功率图



图4.2 设置功率图

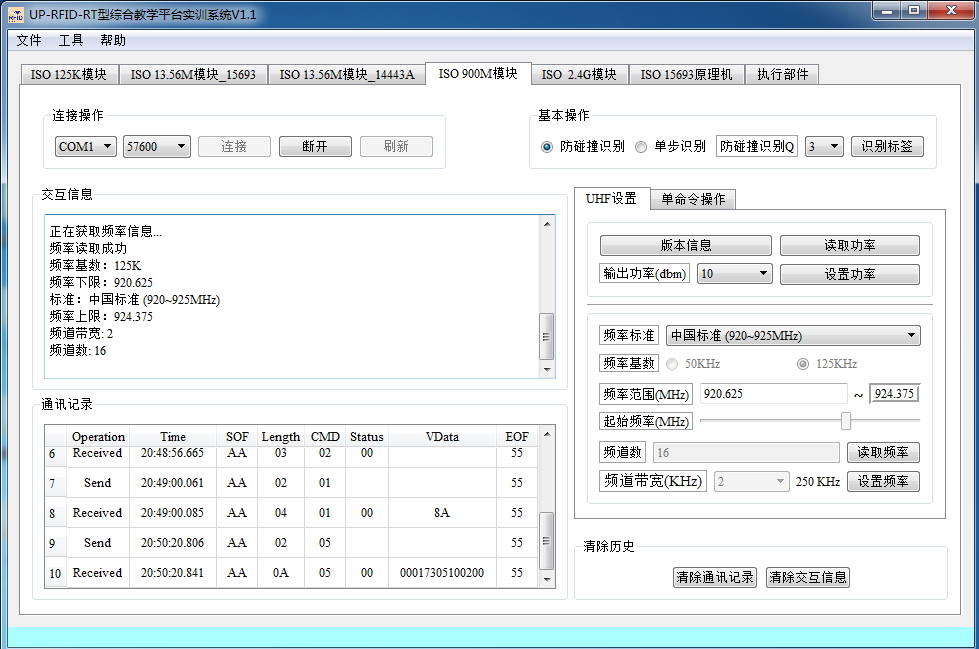


图4.3 读取频率信息图

### 单标签识别实验



图4.4 单标签单步识别识别图

### 多标签识别实验

对于卡重叠进行测试，当两张卡发生重叠的时候，只能读取到下面的卡。



图4.5 多标签单步识别图



图4.6 多标签防碰撞识别图

### 读写擦除操作实验



图4.7 读取内存数据图



图4.8 写入成功图

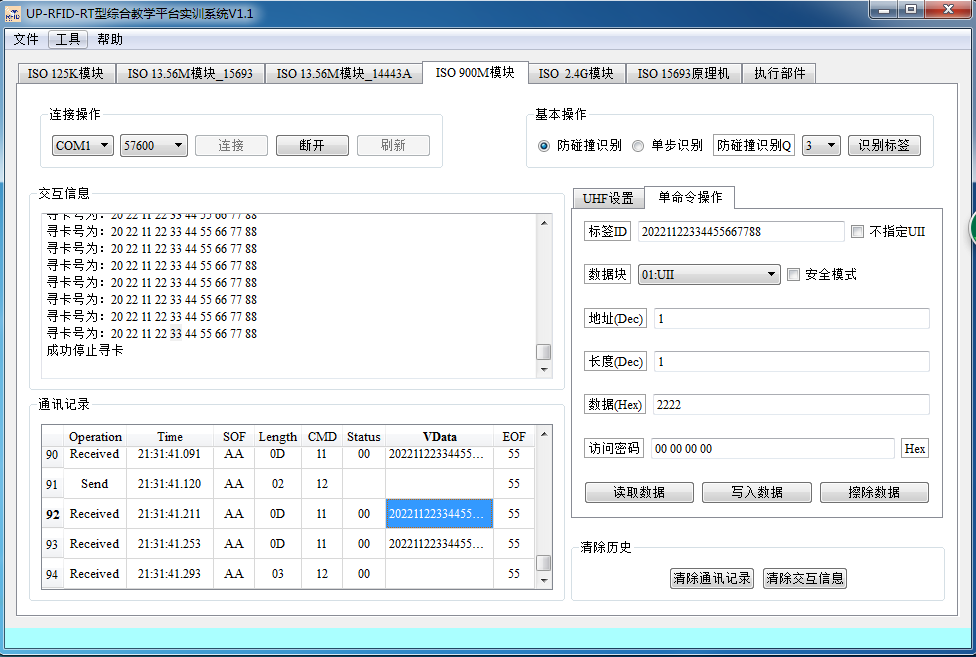


图4.9 卡号修改图

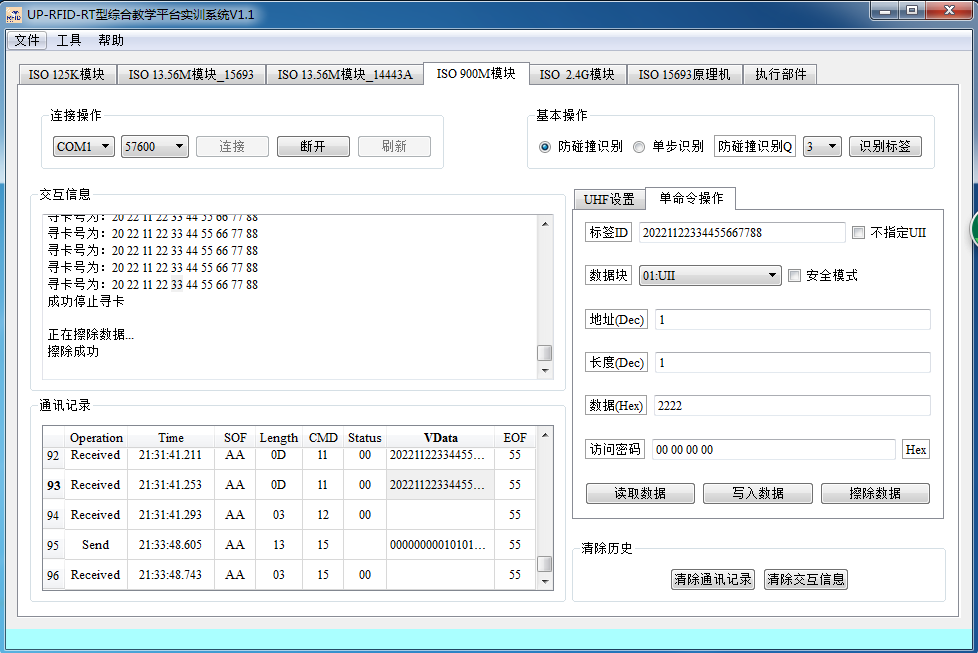


图4.10 擦除图



图4.11 修改0000图

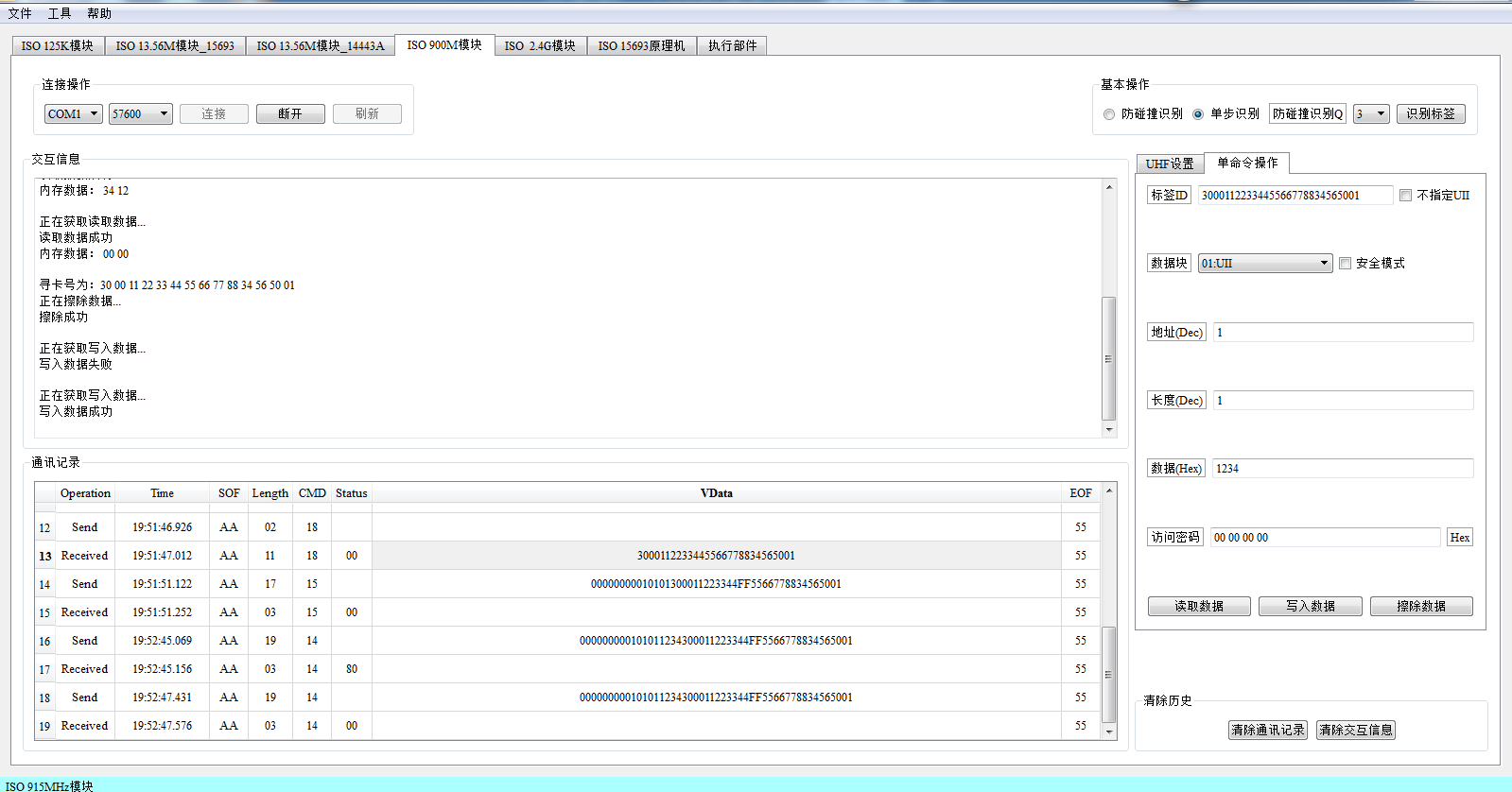


图4.12 读写擦除操作图

### **UHF-900M开发实例**



图4.13 UHF-900M图

## 核心源码说明

### 系统架构

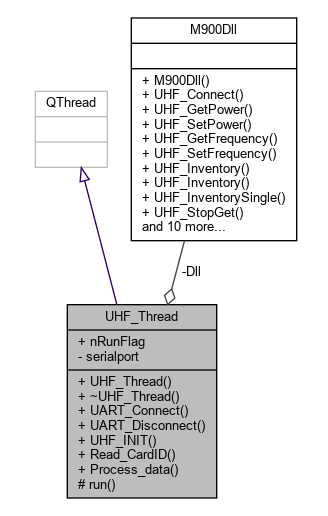


图4.14 UHF-900M接口图

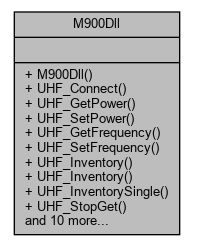


图4.15 M900DLL接口图

### UHF\_Thread接口定义

利用UHF\_Thread进行串口读写

#ifndef UHF\_THREAD\_H

#define UHF\_THREAD\_H

#include <QObject>

#include <QThread>

#include <QtSerialPort/QSerialPort>

#include <QtSerialPort/QSerialPortInfo>

#include "tools.h"

/\*添加dll链接库头文件

\* pro文件中添加：LIBS += -L D:\QtProject\900M\lib -l M900Dll\*/

#include "inc/m900dll.h"

#include "inc/m900dll\_global.h"

**class** UHF\_Thread:**public** QThread

{

    Q\_OBJECT

**public**:

**explicit** UHF\_Thread(QObject \*parent = 0);

    ~UHF\_Thread();

**bool** UART\_Connect(QString ComName, **int** Baudrate);

**bool** UART\_Disconnect();

**bool** UHF\_INIT();

**bool** Read\_CardID();

**void** Process\_data(**int** cmd, **char** data[], **int** length, **int** status);

**bool** nRunFlag;

**private**:

    M900Dll \*Dll;//dll链接库

    QSerialPort \*serialport;

signals:

**void** cardID(QString cardid);

**protected**:

**void** run();

};

#endif // UHF\_THREAD\_H

### 注册操作

判断是否文本框为空，此处进行字符串判断，并没有检测字符串的合法性，采用QMessageBox进行事件提醒。

同时对于数据库查询是否存在，如果存在进行更新，否则添加。未进行数据库异常处理。

**void** Regist\_Widget::Uhf\_Regist\_Button\_Click()

{

    //标签中显示的文本，用于在提示框中显示

**char** Label\_Name[][50] = {"卡号：","车牌号：", "金额：", "车型："};

    /\*如果文本框中存在空，则提示不能为空\*/

**for**(**int** i = 0; i < EDIT\_COUNT\_REGIST; i++)

    {

**if**(Edit[i]->text().isEmpty())

        {

**char** warning[256];

**sprintf**(warning,"%s不能为空",Label\_Name[i]);

            QMessageBox::warning(NULL, "warning", warning, QMessageBox::Yes, QMessageBox::Yes);

**return**;

        }

    }

    Sqlite sql;

    QSqlQuery Sqlite;

    /\*查询卡号是否已经存在\*/

**char** where[256];

**sprintf**(where,"cardID = '%s'",Edit[ID\_Regist]->text().toUtf8().data());

    Sqlite = sql.select("user", where);

**if**(Sqlite.next())

    {

        QMessageBox::warning(NULL, "warning", "已存在此卡", QMessageBox::Yes, QMessageBox::Yes);

**return**;

    }

    /\*将文本框中的信息添加到数据库\*/

    sql.add\_user(Edit[ID\_Regist]->text().toUtf8().data(), Edit[Plate\_number\_Regist]->text().toUtf8().data(), Types->currentText().toUtf8().data(), Edit[Balance\_Regist]->text().toInt());

    QMessageBox::warning(NULL, "warning", "注册成功", QMessageBox::Yes, QMessageBox::Yes);

    /\*清空文本框\*/

**for**(**int** i = 0; i < EDIT\_COUNT\_REGIST; i++)

    {

        Edit[i]->clear();

    }

}

## 思考题

1. 试从定性和定量两个方面分析读写器功率与标签读写距离的关系；
   1. 定性：功率越大读取标签的距离就越大，功率一定的情况下，距离越小越灵敏。
   2. 定量：采用dBm作为功率单位，所以功率的改变对读取的距离的影响为指数级别。
2. EPC 标签存储器分为哪几个区？各区有何功能？

分为Tag memory（标签内存）分为Reserved（保留），EPC（电子产品代码），TID（标签识别号）和User（用户）四个独立的存储区块（Bank）。

* 1. Reserved区：存储Kill Password（灭活口令）和Access Password（访问口令）。
  2. EPC区：存储EPC号码等。
  3. TID区：存储标签识别号码，每个TID号码应该是唯一的。
  4. User区：存储用户定义的数据。

1. EPC 标签可以通过哪些措施来保证各个存储区的信息安全？
   1. 状态安全

EPC标签通过有限状态机的思想来构建了七种状态，只有在指定的几个状态才能进行读写锁定等操作，同时进入操作的状态时，需要先经过Access Password的验证。

* 1. 数据安全

TID区：存储标签识别号码，每个TID号码应该是唯一的；User区：存储用户定义的数据。

1. EPC 和 TID 分别表示什么含义？二者结构有何特点？

EPC（Electronic Product Code），展品电子代码，对供应链中的对象进行全球唯一的标识；

TID，RFID电子标签识别号，是标签之间身份区别的标志，具有唯一性。

EPC码可以被复制，但TID是唯一的，具有一个Lock状态位，来防止出厂后被改写，才能被永久锁定。

## 实验体会与总结

通过试验箱对 Gen2 协议下标签指定存储区的数据读写，加深了对Gen2 协议下标签存储器结构的理解，了解了读写器功率和频率对电子标签读写的影响，了解了访问密码的用途和使用方法，了解了超高频读写器 API 函数的调用方法。

由于高频通信的硬件问题，造成许多卡不容易读出，此时可以考虑改变天线的朝向和相应角度进行读取。

对于Reader，尝试改变不同的读取参数(功率)，观察读取的距离，从而有了比较直观的体验。

实验代码对于不同的部分进行了解耦合和剥离，将工具类tools,sqlite等类进行了相应的区分，并创建线程进行串口读取，采用timer机制进行轮训。但整个界面UI采用动态添加，在高频读取的时候，如果时间相应较频繁，其开销会较大，可考虑采用组件池的机制进行相应的组件管理。

# 实验五 基于RFID技术的图书管理系统设计与实现

## 实验目的

1. 本次实验属于综合性应用实验，要求用户能够灵活应用 RFID 技术原理，解 决实际生活中遇到的应用问题，培养用户分析问题、解决问题的能力以及综合知 识的应用能力。
2. 由于 RFID 技术应用范围非常广泛，本次实验限定应用 13.56M 读写器、基于 ISO14443A 协议的电子标签、基于 ISO15693 协议电子标签开发两套综合应用 系统。

## 实验内容与要求

### 系统描述

采用北京博创 RFID 实验箱模拟图书管理系统的读卡设备、支持 ISO15693

协议的 S50 卡（5 张）模拟图书，一张卡作为用户身份的唯一识别卡，其他四张 卡与唯一的一本图书关联。

用户首次申请领用该卡（称用户卡）时，保存个人手机号，并对卡进行初始 化。将卡与个人手机关联、姓名关联起来（采用实名制，便于挂失）。

另外四张卡初始化与四本图书关联，标识图书的唯一性。将最近的 5 次借阅 信息同时存储在卡内和数据库中。

一旦用户卡丢失，马上通知管理员挂失,系统自动将该卡设置为未激活状态， 锁定消费。当然，每次消费时，一定要判断该用户卡是否处于激活状态。 假设不同的图书可供借阅的时间长短不一样，用户利用该用户卡借阅不同图书，在卡内记录最近五条借还明细，借还明细同时写入系统数据库表中。系统能 够规则，提示图书偿还时间。

假定读写器设备与上位机始终保持联系，上位机与数据库服务器始终保持联系。

### 系统功能

1. 用户发卡管理；
2. 图书与电子标签关联管理；
3. 用户毕业时的销卡管理，清除卡内借/还数据以及个人手机号；
4. 用户借或还图书时，在卡内和数据库中同时保存借阅记录信息，假定记录信息不超过 5 条；
5. 图书借/还明细查询。显示借/还明细时，必须同时显示用户手机号、电子标签 ID 号；

### 系统表结构

1. 用户基本信息表（卡号、姓名、手机号、…、是否激活);
2. 用户借/还信息表（卡号、姓名、电子标签 Id、借阅时间、还书时间、…);
3. 图书基本信息表（商品编号、电子标签 Id、图书名称、作者、出版社、出版时间…);

## 实验环境

### 平台环境

OS: Manjaro 18.0.4 Illyria

Kernel: x86\_64 Linux 5.0.7-1-MANJARO

CPU: Intel Core i7-6700HQ @ 8x 3.5GHz

GPU: GeForce GTX 965M

RAM: 7865MiB

### 软件环境

Qt Creator 3.5.1 (opensource)

Based on Qt 5.5.1 (GCC 4.9.1 20140922 (Red Hat 4.9.1-10), 32 bit)

## 系统架构

### 数据信息描述

1. 用户基本信息表（卡号，姓名，性别，年龄）；
2. 用户借/还信息表（卡号，姓名，电子标签Id，借阅时间）；
3. 图书基本信息表（电子标签Id，图书名称，作者，出版社，总本数，剩余本数，可借时间）；

### 数据字典

表5.1 用户基本信息表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 字段类型 | 主/外键 | 字段值约束 | 说明 |
| cardID | vchar | 主键 |  | 卡的编号 |
| name | vchar |  |  | 持卡人姓名 |
| gender | vchar |  |  | 持卡人性别 |
| age | int |  |  | 持卡人年龄 |

表5.2 用户借/还信息表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 字段类型 | 主/外键 | 字段值约束 | 说明 |
| booksID | vchar | 主键 |  | 书的编号 |
| name | vchar |  |  | 书的名称 |
| author | vchar |  |  | 书的作者 |
| publishing\_house | vchar |  |  | 书的出版社 |
| count | int |  |  | 书的总共数量 |
| residue | int |  |  | 书的剩余数量 |

表图书基本信息表

## 系统实现

基于 RFID 技术的图书管理系统设计与实现。

### 系统架构图

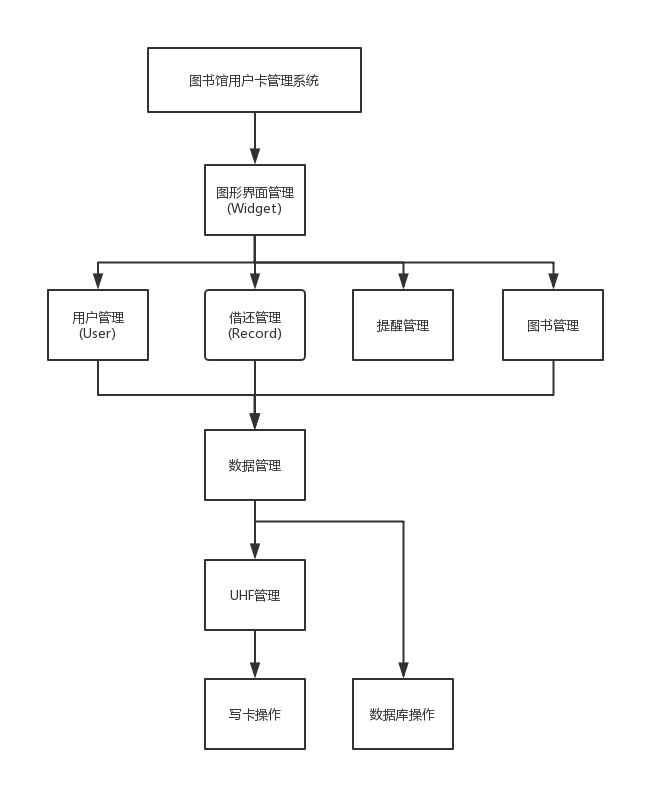


图5.1 系统逻辑架构图

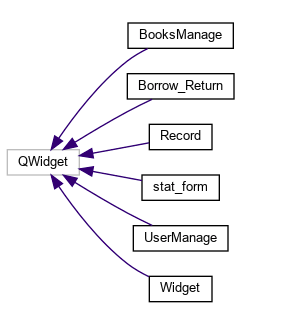


图5.2 系统实现架构图

### 项目类集成图

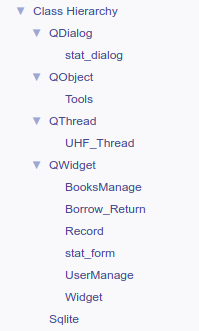


图5.3 项目类集成图

### 数据库E-R图

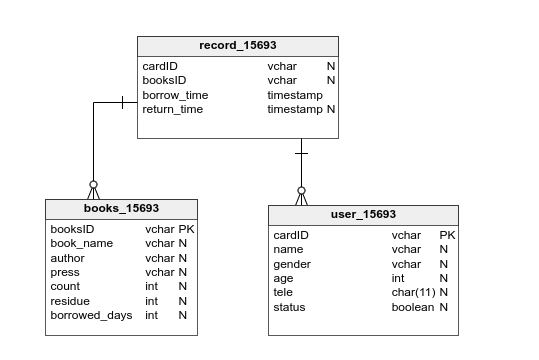


图5.4 数据库关系图

### 用户管理

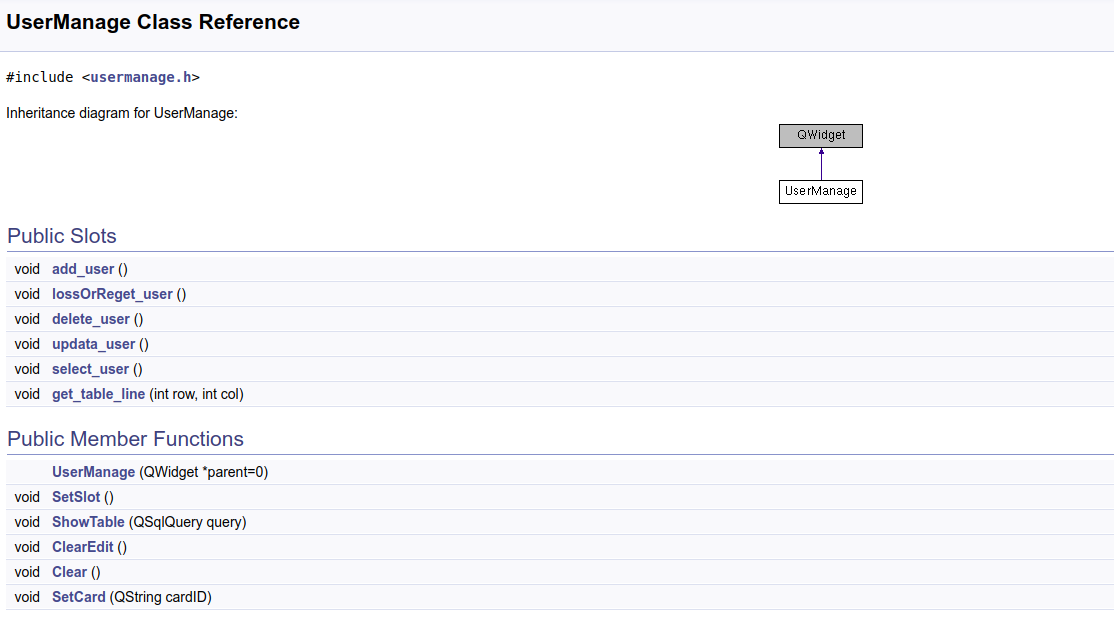


图5.5 用户管理

1. 用户注册

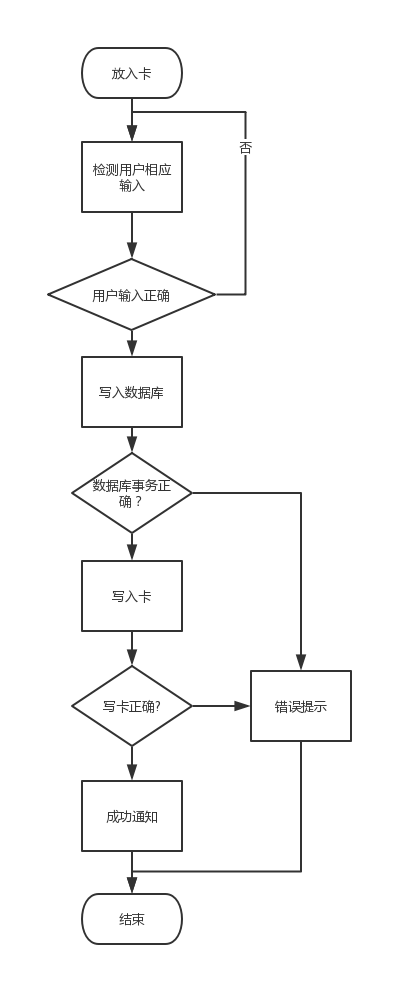


图5.6 用户注册流程

1. 用户搜索

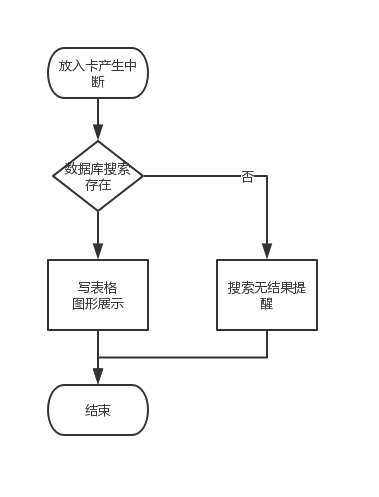


图5.7 用户注册流程

1. 用户挂失/激活

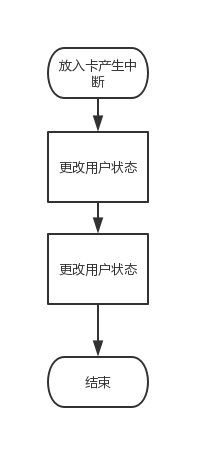


图5.8 用户挂失/激活流程

1. 用户删除流程

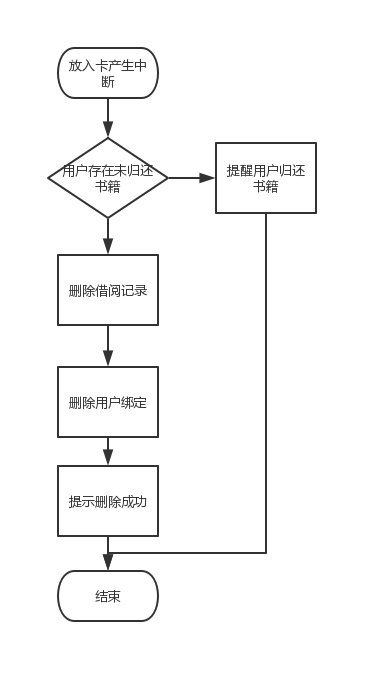


图5.9 用户删除流程

### 图形界面窗口管理



图5.10 图形界面窗口管理

由于图书管理过程中，由Widget进行用户相应的窗口管理，检测用户当前所在的窗口，进行相应的设置。

同时由于UHF由Widget持有，所有的写卡操作需要由Widget转发。

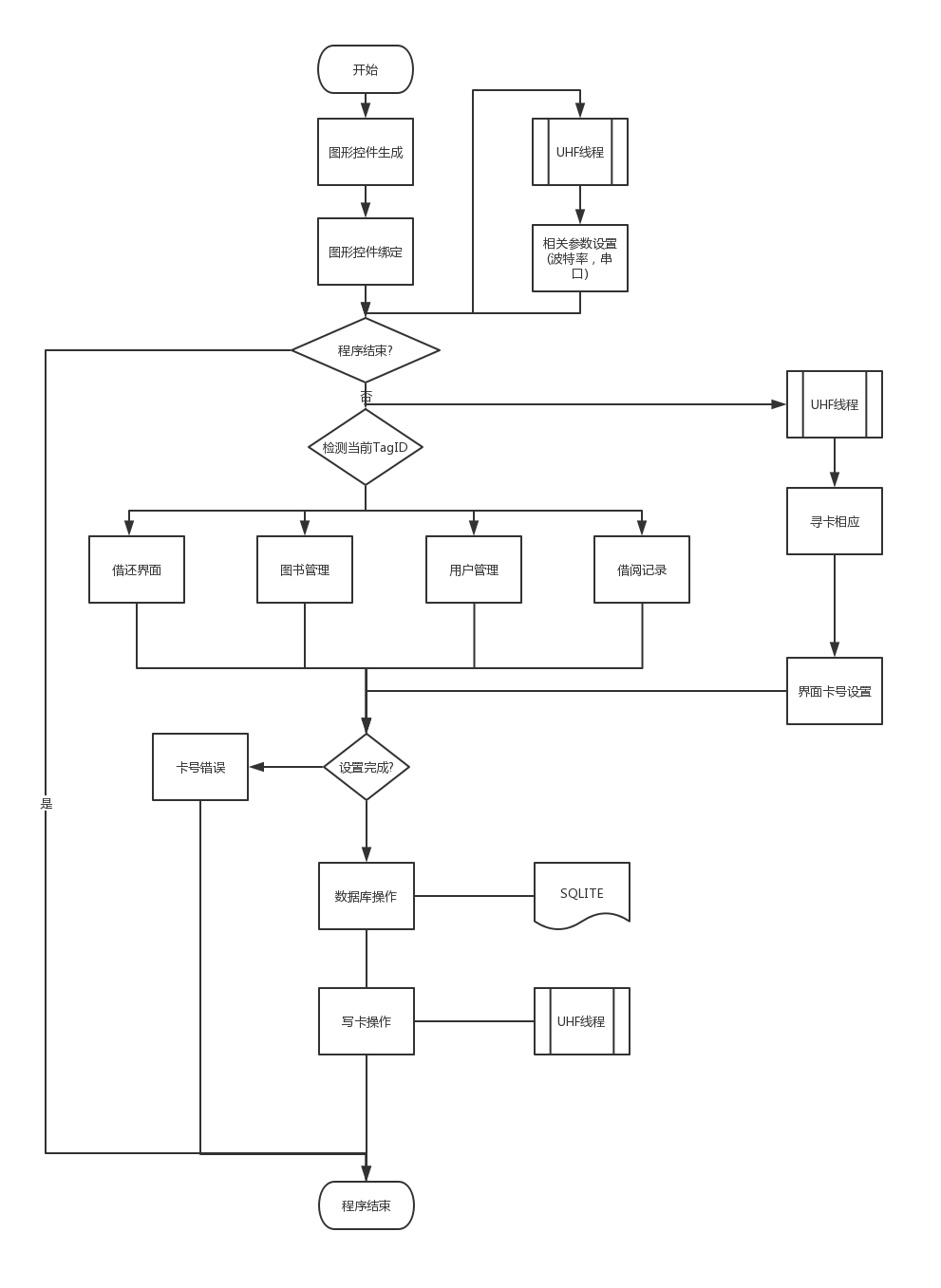


图5.11 Widget管理流程图

### 图书管理

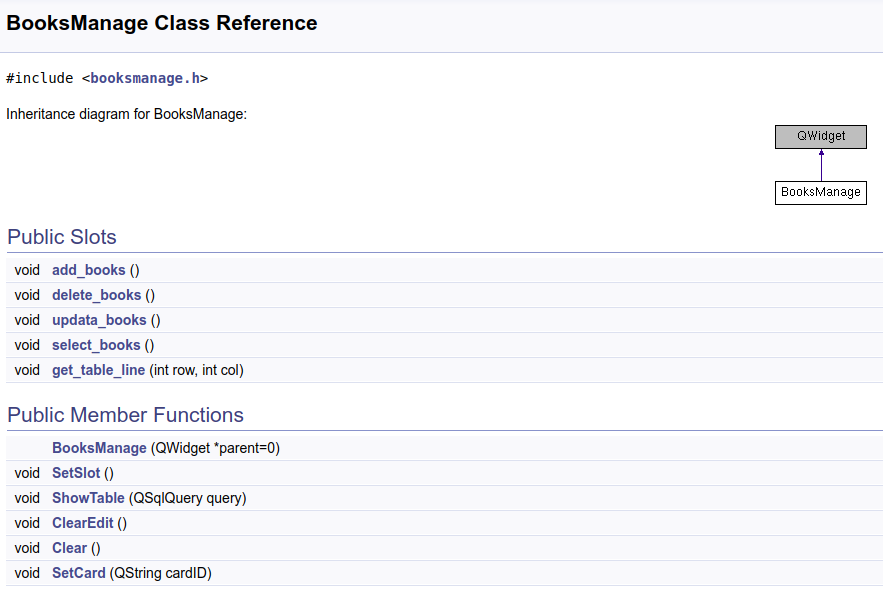


图5.12 图书管理方法类

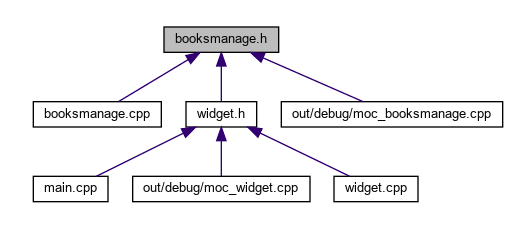


图5.13 图书管理调用关系

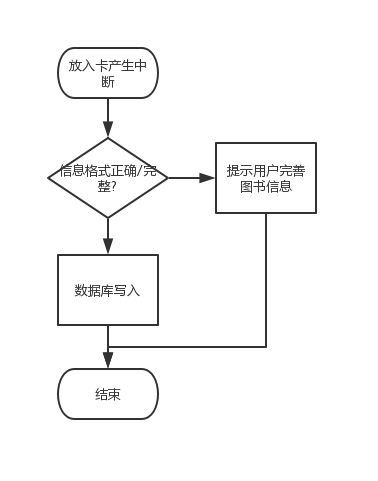


图5.14 图书绑定流程图

### 借阅管理

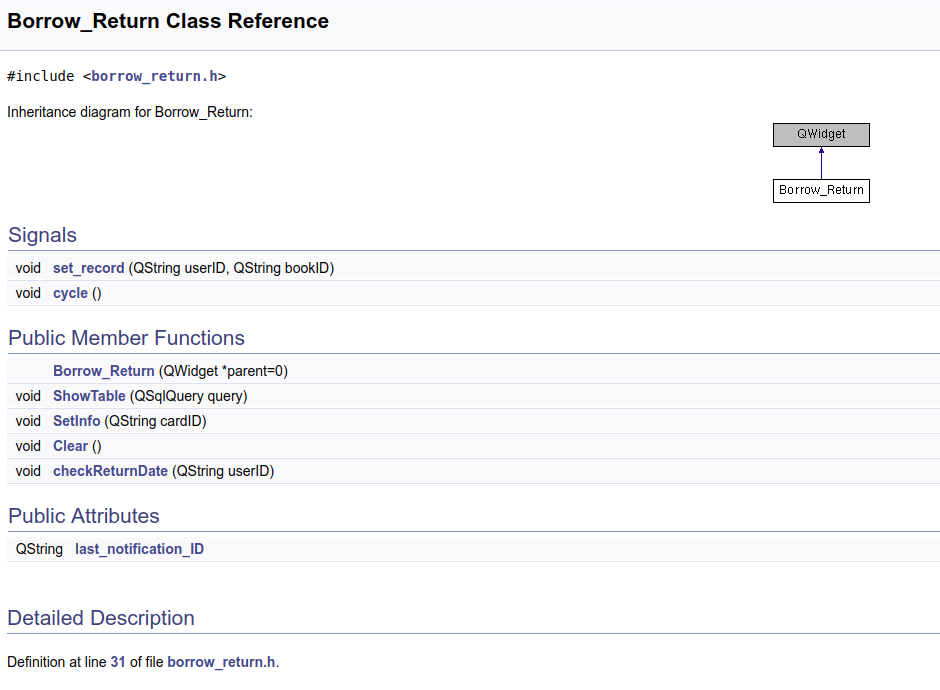


图5.15 借阅管理方法类

对于特定用户和特定图书进行管理，进行借阅记录的写入和提醒。

### 借阅记录

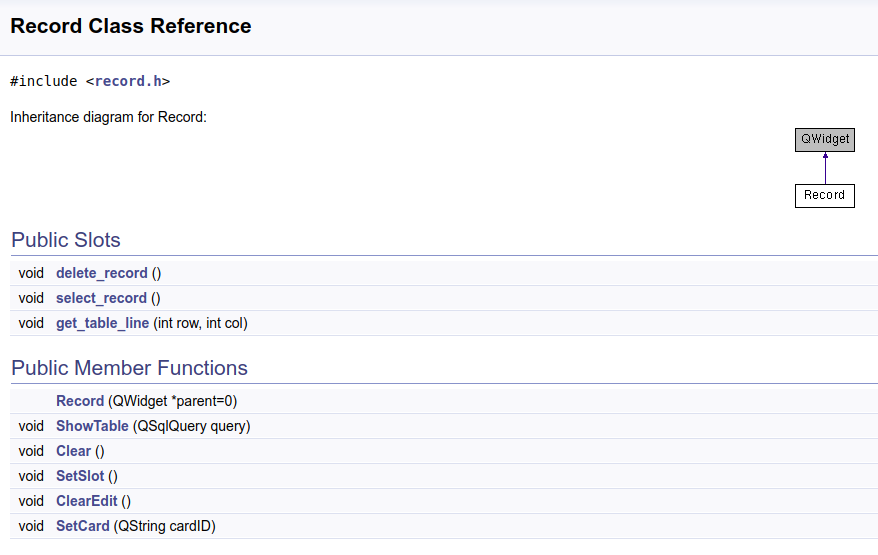


图5.16 借阅记录

存储所有的借阅记录，方便管理员进行查询。

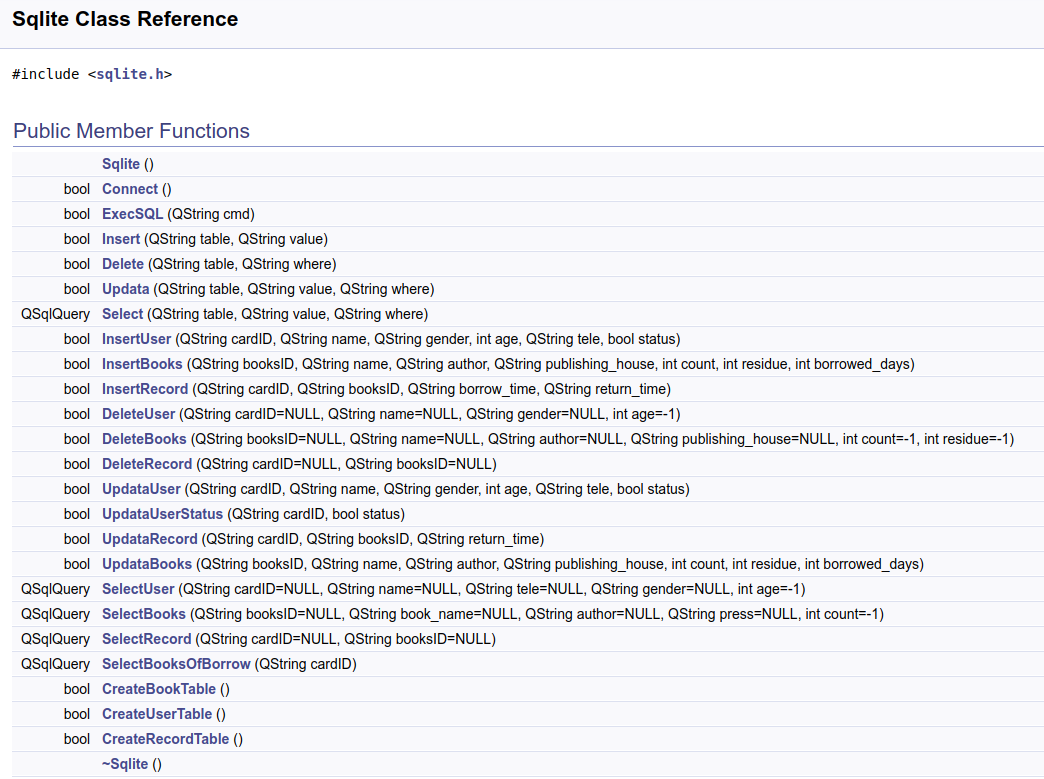
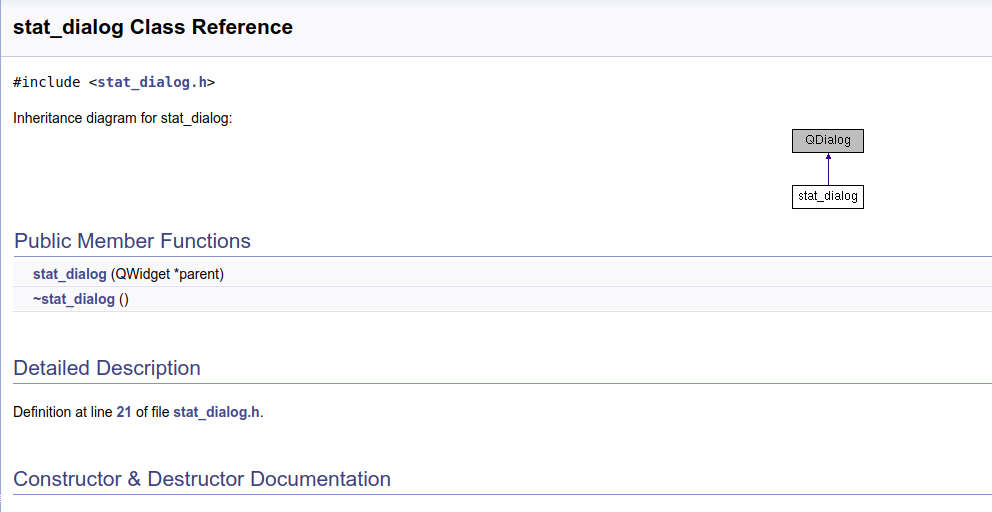


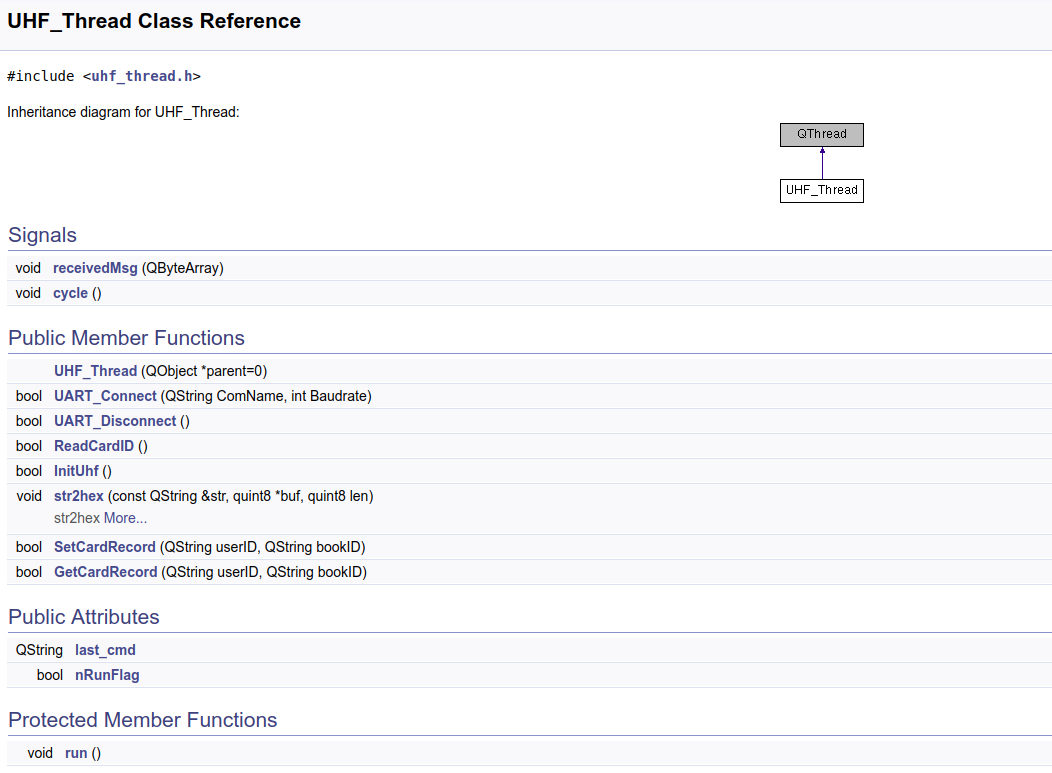
图5.17 SQL数据库操作方法

### 统计图表创建



查询数据库，绘制统计图表。

### UHF串口通信线程



用于创建线程，波特率串口设置，同时进行循环寻卡操作，如果响应，调用响应的处理函数。

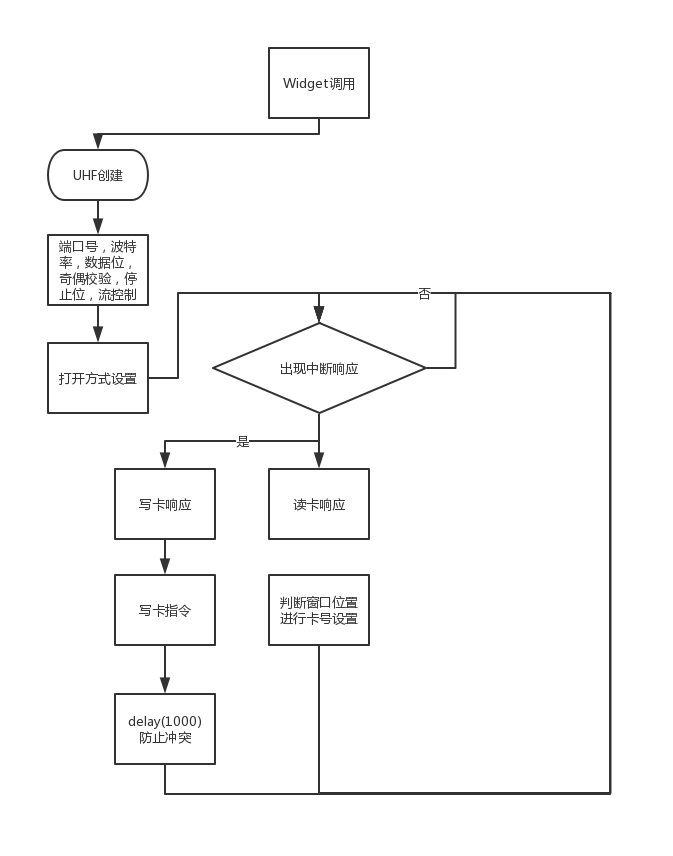


图5.18 UHF读卡/写卡方法

## 系统功能测试

### 用户发卡管理

刷卡获取卡号，填写用户姓名、性别、年龄，将信息添加到数据库表中，并将姓名信息写入卡内0块。

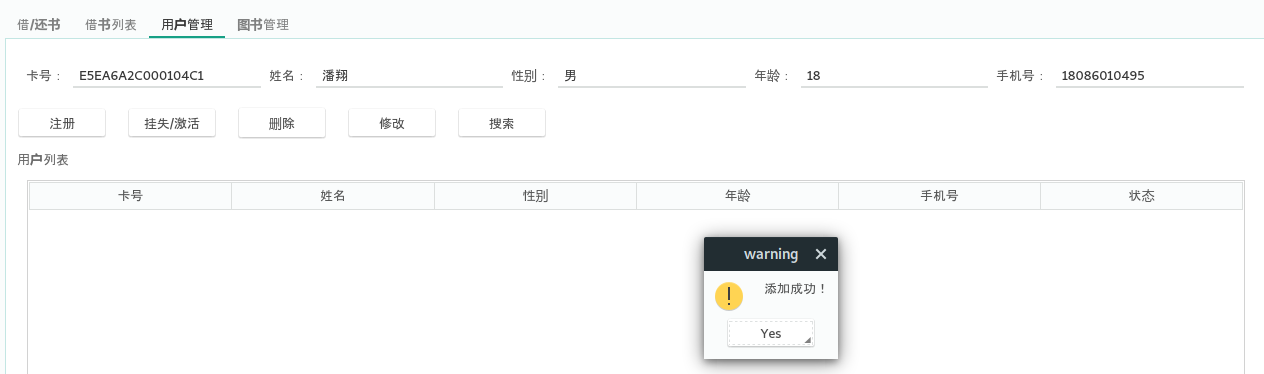


图5.19 注册用户



图5.20 用户搜索

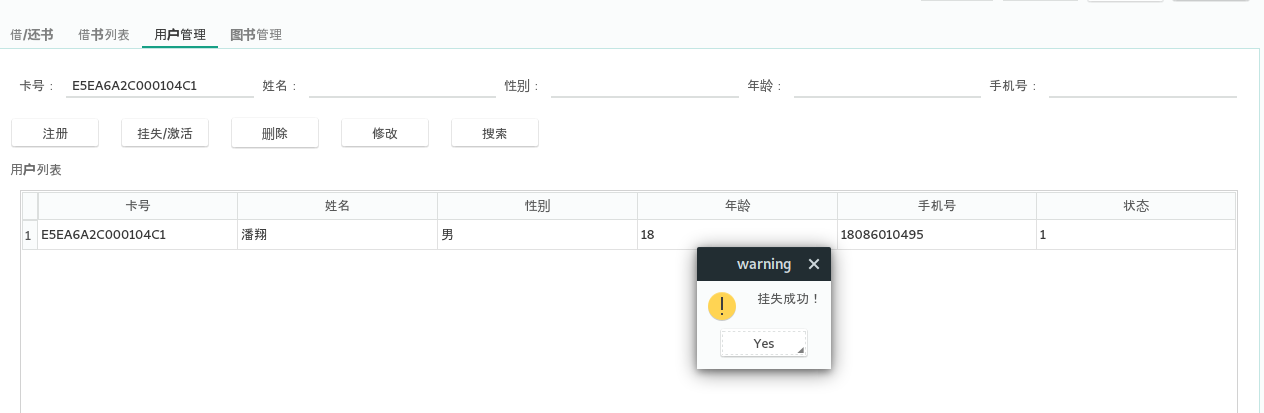


图5.21 用户挂失

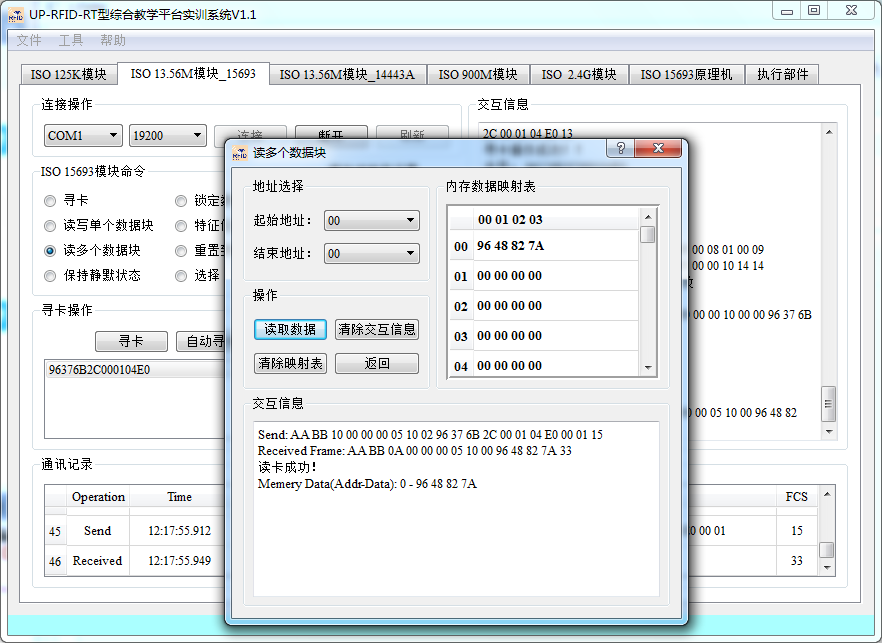


图5.22 成功写入图

### 图书管理

获取卡号，输入信息包括书名、作者、出版社、总本书、剩余本数和可借时长，将数据写入数据库表中。



图5.23 添加成功图

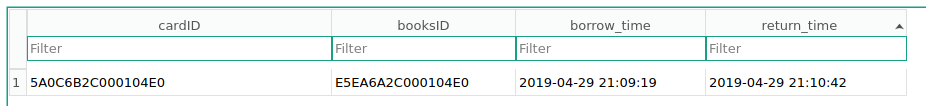


**图5.24 信息写入数据图**

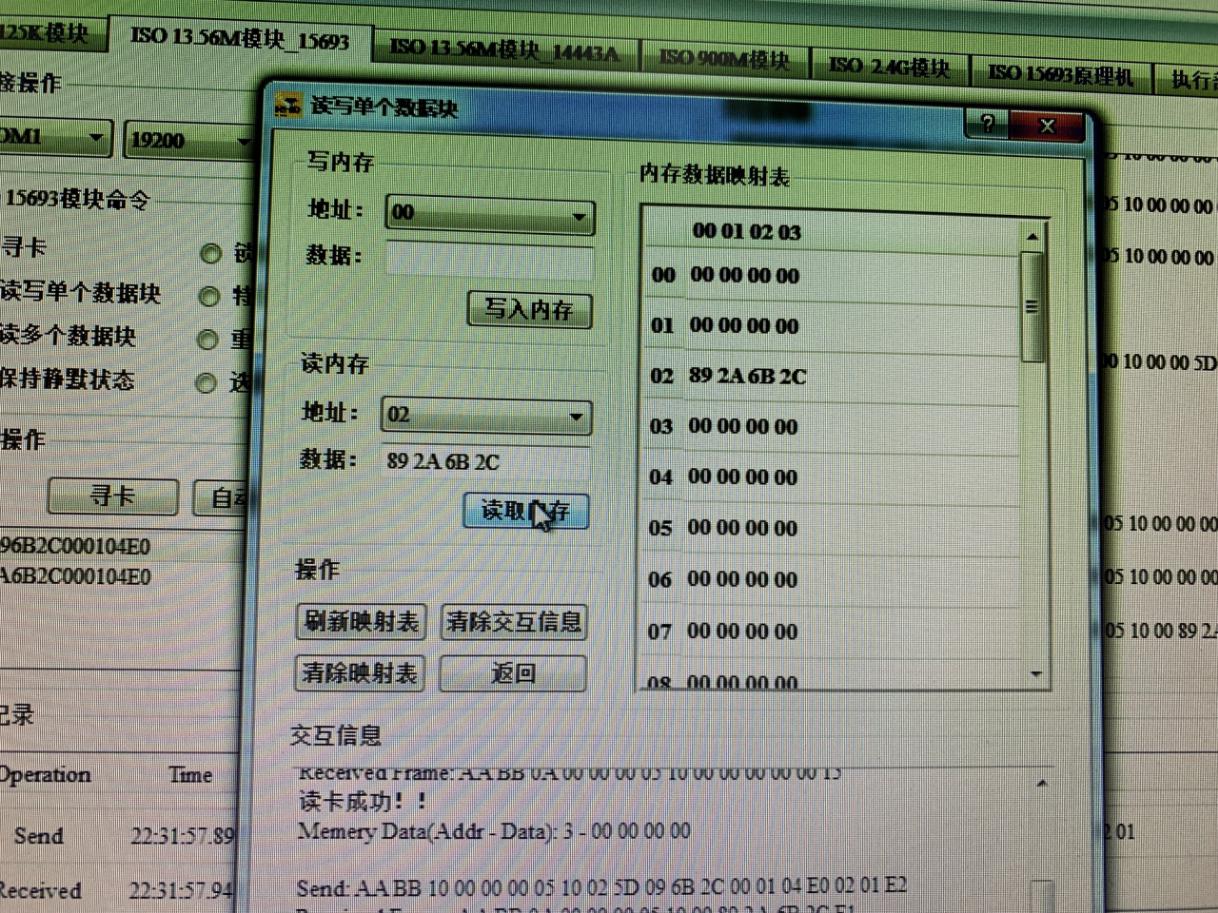
### 借阅管理



**图5.25 用户借阅查询-无借阅记录提醒**



**图5.26 用户借阅记录**



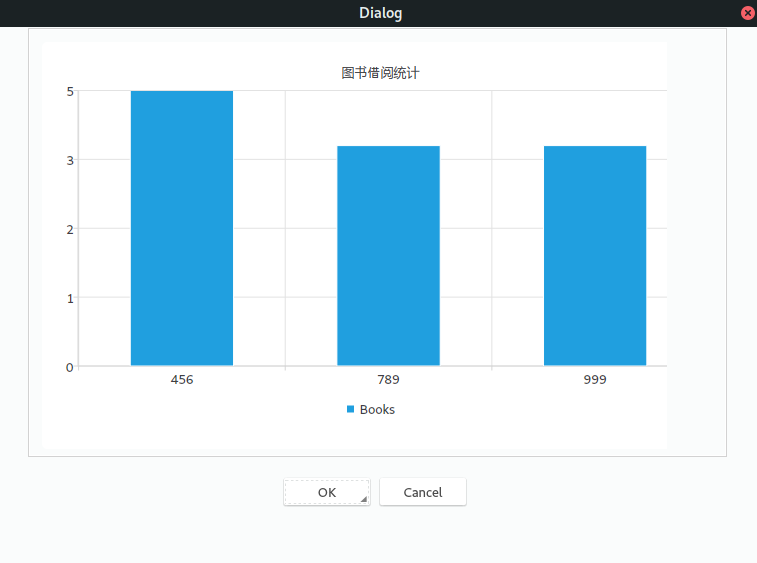
**图5.27 借阅写卡测试**

在从第二个扇区进行相应的借阅图书卡号写入。

同时提供用户的借阅相关记录，同时相关的借阅记录会提供借还日起提醒。(已进行演示，因时间相应操作，截图为模拟测试)。

### 统计图表测试

输出借阅次数统计图



**图5.28 统计图表测试**

### 提醒测试



**图5.29 挂失警告测试**



**图5.30 还书警告测试**

## 核心源码说明

### 用户管理

代码5.1 激活/挂失槽函数

//挂失或激活用户槽函数

**void** UserManage::lossOrReget\_user()

{

    //为了安全，需提供卡号、姓名、手机号

    QSqlQuery query = sql->SelectUser(Edit[ID\_User]->text(),Edit[Name\_User]->text(),Edit[Tele\_User]->text(),NULL,-1);

**if**(!query.next())

    {

        QMessageBox::warning(NULL, "warning", "卡号不存在！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

    }

**bool** willStatus = **false**;

    QString warn = "挂失";

**if**(query.value("status").toBool() == **false**)

    {

        willStatus = **true**;

        warn = "激活";

//        QMessageBox::warning(NULL, "warning", "该卡已注销！", QMessageBox::Yes, QMessageBox::Yes);

//        return;

    }

    //改变卡的状态

**int** ret = sql->UpdataUserStatus(Edit[ID\_User]->text(),willStatus);

**if**(!ret)

    {

        QMessageBox::warning(NULL, "warning", warn+"失败！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

    }

    QMessageBox::warning(NULL, "warning", warn+"成功！", QMessageBox::Yes, QMessageBox::Yes);

    ClearEdit();

    ShowTable(sql->SelectUser());

}

### 借阅管理

如果用户存在未归还的图书，则无法进行借阅，从而无法借阅，实现方法所检索数据库，判断是否有逾期书籍，如果有逾期，同时给出逾期警告。

代码5.2 能否借阅检测

**bool** canBorrow(QString userID)

{

    qDebug()<<"checkReturnDate"<<"borrow and return"<<endl;

    QSqlQuery query;

    QString cmd="select\

    record\_15693.booksID,\

    record\_15693.borrow\_time ,\

    books\_15693.borrowed\_days\

    from record\_15693\

    left join  books\_15693 on record\_15693.booksID = books\_15693.booksID\

    where record\_15693.cardID=='"+userID+"'   ";

    query.exec(cmd);

**if**(!query.next())

    {

        //no result

        qDebug()<<"no result"<<endl;

**return** **true**;

    }

    qDebug()<<query.size()<<endl;

    QString bookID;

    QDateTime borrowDate;

**int** borrowed\_days;

**int** warnning\_days=10;

    QDateTime dueDate;

    QDateTime nowDate=QDateTime::currentDateTime();

**bool** can\_borrow=**true**;

    query.first();//返回第一条数据

**do**

    {

        //check numd;

        qDebug()<<query.value(0)<<query.value(1)<<query.value(2)<<endl;

        bookID=query.value(0).toString();

        borrowDate = QDateTime::fromString(query.value(1).toString(), "yyyy-MM-dd hh:mm:ss");

        borrowed\_days=query.value(2).toInt();

        dueDate=borrowDate.addDays(borrowed\_days);

**int** ndaysec = 24\*60\*60;

        uint stime = nowDate.toTime\_t();

        uint etime = dueDate.toTime\_t();

        uint left\_days=(etime - stime)/(ndaysec) + ((etime - stime)%(ndaysec)+(ndaysec-1))/(ndaysec) - 1;

        qDebug() << "Day = " <<left\_days<<endl;

**if**(left\_days<0)

        {

            can\_borrow=**false**;

            QMessageBox::warning(NULL, "逾期警告！", "书籍："+bookID+"\n"+"还书日期："+dueDate.toString()+"\n"+"逾期时间："+QString::number(-left\_days, 10)+"Days\n", QMessageBox::Yes, QMessageBox::Yes);

        }

        qDebug()<<bookID<<borrowDate<<borrowed\_days<<endl;

    }**while**(query.next());

**return** can\_borrow;

}

对于同一个用户反复借阅谋一本书，原代码中未提供相应的警告，此处进行检测，同时给出警告。

代码5.3 重复借书警告

            QSqlQuery myquery=sql->SelectRecord(Edit\_User[CardId\_User\_Borrow]->text(), query.value(0).toString());

**while**(myquery.next())

            {

**if**(myquery.value(3).toString()=="")

                {

                    QMessageBox::warning(NULL, "warning", "借书错误,该书已借！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

                }

            }

用户进行对于同一本书进行借还操作的时候，会出现问题，因为记录始终存在，需要判断是否有还书时间为空的记录，判断能够还书。

否，则给出警告。

代码5.4 能否还书检测

            QSqlQuery myquery=sql->SelectRecord(Edit\_User[CardId\_User\_Borrow]->text(), query.value(0).toString());

**bool** can\_return;

**while**(myquery.next())

            {

**if**(myquery.value(3).toString()=="")

                {

                    can\_return=**true**;

                }

            }

**if**(!can\_return)

            {

                QMessageBox::warning(NULL, "warning", "还书错误,该书已还！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

            }

### 用户提醒管理

当寻卡产生中断的时候，调用此函数，判断是否有逾期书籍和提醒书籍。

通过int warnning\_days=10设置相应的提醒时间

计算采用转换为时间戳向上取整进行计算。

代码5.5 用户提醒

**void** Borrow\_Return::checkReturnDate(QString userID)

{

**if**(userID==last\_notification\_ID)

    {

**return**;

    }

    qDebug()<<"checkReturnDate"<<"borrow and return"<<endl;

    QSqlQuery query;

    //userID="999";

    QString cmd="select\

    record\_15693.booksID,\

    record\_15693.borrow\_time ,\

    books\_15693.borrowed\_days\

    from record\_15693\

    left join  books\_15693 on record\_15693.booksID = books\_15693.booksID\

    where record\_15693.cardID=='"+userID+"'   \

    and record\_15693.return\_time is NULL";

    query.exec(cmd);

**if**(!query.next())

    {

        //no result

        qDebug()<<"no result"<<endl;

**return**;

    }

    qDebug()<<query.size()<<endl;

    QString bookID;

    QDateTime borrowDate;

**int** borrowed\_days;

**int** warnning\_days=10;

    QDateTime dueDate;

    QDateTime nowDate=QDateTime::currentDateTime();

    query.first();//返回第一条数据

**do**

    {

        //check numd;

        qDebug()<<query.value(0)<<query.value(1)<<query.value(2)<<endl;

        bookID=query.value(0).toString();

        borrowDate = QDateTime::fromString(query.value(1).toString(), "yyyy-MM-dd hh:mm:ss");

        borrowed\_days=query.value(2).toInt();

        dueDate=borrowDate.addDays(borrowed\_days);

**int** ndaysec = 24\*60\*60;

        uint stime = nowDate.toTime\_t();

        uint etime = dueDate.toTime\_t();

        uint left\_days=(etime - stime)/(ndaysec) + ((etime - stime)%(ndaysec)+(ndaysec-1))/(ndaysec) - 1;

        qDebug() << "Day = " <<left\_days<<endl;

**if**(left\_days<0)

        {

            QMessageBox::warning(NULL, "逾期警告！", "书籍："+bookID+"\n"+"还书日期："+dueDate.toString()+"\n"+"逾期时间："+QString::number(-left\_days, 10)+"Days\n", QMessageBox::Yes, QMessageBox::Yes);

        }

**else** **if**(left\_days<warnning\_days)

        {

            QMessageBox::warning(NULL, "还书警告！", "书籍："+bookID+"\n"+"还书日期："+dueDate.toString()+"\n"+"剩余时间："+QString::number(left\_days, 10)+"Days\n", QMessageBox::Yes, QMessageBox::Yes);

        }

        qDebug()<<bookID<<borrowDate<<borrowed\_days<<endl;

//        bookID=query.value(0)

    }**while**(query.next());

    last\_notification\_ID=userID;

}

### 写卡操作

所有的写卡操作必须使用Widget进行转发，因为Widget持有UHF对象。

代码5.6 写卡转发函数

**void** Widget::Set\_User\_Record(QString UserID,QString BookID)

{

    //QMessageBox::warning(NULL, "warning", "调用写卡！", QMessageBox::Yes, QMessageBox::Yes);

    qDebug()<<"!!!!!"<<UserID<<BookID;

    uhf->SetCardRecord(UserID,BookID);

}

**void** Widget::Get\_User\_Record(QString UserID,QString BookID)

{

    uhf->GetCardRecord(UserID,BookID);

}

代码5.7 校验和计算函数

uint8 RC632\_UartCalcFCS\_Test( uint8 \*msg\_ptr, uint8 len )

{

    uint8 x;

    uint8 xorResult;

    xorResult = 0;

**for** ( x = 0; x < len; x++, msg\_ptr++ )

      xorResult = xorResult ^ \*msg\_ptr;

**return** ( xorResult );

}

代码5.8 帧封装函数

uint8\* RC632\_SendCmdReq\_Test(uint16 cmd, **const** uint8 \*data, uint16 len)

{

    uint8 \*p;

    uint16 frameLen, uartdatalen;

    frameLen = len+9;   // 9 = sop(2)+len(2)+nc(2)+cmd(2)+fcs(1)

    uartdatalen = frameLen;

**for**(uint16 i=0; i<len; i++)

**if**(data[i] == 0xAA)  uartdatalen++;

    p = **new** uint8[uartdatalen+2];  // p[0] is used return frameLen

    p[0] = LO\_UINT16(uartdatalen);

    p[1] = HI\_UINT16(uartdatalen);     //framelen

    p[2] = LO\_UINT16(RC632\_FRAME\_SOP);

    p[3] = HI\_UINT16(RC632\_FRAME\_SOP); //BBAA

    p[4] = LO\_UINT16(frameLen-4);

    p[5] = HI\_UINT16(frameLen-4);      // 4 = sop(2)+len(2)

    p[6] = 0;

    p[7] = 0;                          // NC

    p[8] = LO\_UINT16(cmd);

    p[9] = HI\_UINT16(cmd);

    //memcpy(p+10, data, len);

**int** k=0;

**for**(**int** j=0; j<len; j++, k++){

        p[10+k] = data[j];

**if**(data[j]==0xAA)  p[10+ ++k] = 0x00;

    }

    p[10+k] = RC632\_UartCalcFCS\_Test(p+5, uartdatalen-4);

**return** p;

}

代码5.9 Qstring转hex函数

/\*\*

 \* @brief str2hex

 \* @param str QString指针

 \* @param buf 字节数组指针

 \* @param len 要转化的长度

 \* QString 转十六进制数组

 \*/

**void** UHF\_Thread::str2hex(**const** QString &str, quint8 \*buf, quint8 len)

{

    quint8 **strlen** = ((quint8)str.count()+1)/2;

    quint8 buflen = (**strlen**>len) ? len : **strlen**;

    QString strHex;

**for**(quint8 i=0; i<buflen; i++){

        strHex = str.mid((i<<1), 2);

        buf[i] = (quint8)(strHex.toInt(0, 16));

    }

}

写卡需要将最近一次的记录写入2号和3号扇区，封装两个函数帧

格式为

Cmd+02+cardID(当前写的ID)+address+data(4BYTE)

一个卡号有8Byte，需要写两个数据块。

需要保存五条纪录，需要进行相应的dataMap进行数据块映射

其中02为dataMap



**图5.31 存储区块分配图**

03-12作为卡号存储，每两块一组，02作为存储的标志位

代码5.10 写卡函数

**bool** UHF\_Thread::SetCardRecord(QString userID,QString bookID)

{

    //QMessageBox::warning(NULL, "warning", "写卡", QMessageBox::Yes, QMessageBox::Yes);

    last\_cmd="RC632\_CMD\_ISO15693\_WRITE\_SM";

    //userID="46F87856000104E0";

    //bookID="46F87856000104E0";

    //02

    uint16 cmd = RC632\_CMD\_ISO15693\_WRITE\_SM;  //写单个块

    uint8\* vdata1;

    vdata1 = **new** uint8[14];

    vdata1[0]=0x02;  //cut

    quint8 \* buf1;

    buf1=**new** quint8[8];

    str2hex(userID,buf1,8);

**memcpy**(vdata1+1,buf1,8);    //write ID

    vdata1[9]=0x02;  //add

    quint8 \* buf2;

    buf2=**new** quint8[4];

    str2hex(bookID,buf2,4);

**memcpy**(vdata1+10,buf2,4);   //record ID

//    QString a;

**char** \*p1 = (**char** \*)Dll->RC632\_SendCmdReq (cmd,vdata1,14);

    //03

    uint8\* vdata2;

    vdata2 = **new** uint8[14];

    vdata2[0]=0x02;  //cut

    quint8 \* buf3;

    buf3=**new** quint8[8];

    str2hex(userID,buf3,8);

**memcpy**(vdata2+1,buf3,8);    //write ID

    vdata2[9]=0x03;  //address

    quint8 \* buf4;

    buf4=**new** quint8[8];

    str2hex(bookID.**remove**(0,4),buf4,8);

**memcpy**(vdata2+10,buf4,4);   //record ID

**char** \*p2= (**char** \*)Dll->RC632\_SendCmdReq (cmd,vdata2,14);

**if**(serialport->write((**char**\*)(p1 + 2), BUILD\_UINT16(p1[0], p1[1])))//data,len

    {

        //sleep(1000);

        //serialport->write((char\*)(p2 + 2), BUILD\_UINT16(p2[0], p2[1]));

**return** **true**;

    }

**else**

    {

**return** **false**;

    }

}

## 实验体会与总结

实验过程中，由于代码为Windows32 Version，在使用Linux X64编译的时候出现了依赖问题，由于Windows动态链接采用dll，而Linux采用.so进行管理，造成了编译时候的麻烦，且前期未提供源码，只能采用自己封装一层Wine进行调用，且只能编译32位版本，后期提供源码之后就自己封装动态链接库，方便许多。

进行Windows和Linux迁移过程中，手写了编译选项，自动判断当前系统环境，从而实现自动化的跨平台编译，而不用手动更改相应选项。

由于Qt比较熟悉，所以改起来没有难度，但是因为本来软件架构的一些原因，比如采用循环读卡，不断发送读卡命令，从而写卡命令有时候会发生冲突，所以需要delay，这样的自旋锁写法本身十分不优雅，对于后期添加功能不友好，我们在自己实现的时候要避免，而且需要保留好相应的软件接口。

其中所有的图形界面采用动态生成和绑定，由Widget管理，所以所有的操作需要由Widget管理转发，而限于Qt自身的缺陷，采用Signal进行通信的时候，无法进行单步跟踪，从而只能调试进行输出，由于不断的读卡错误信息返回，所以此处采用QMessageBox进行调试输出，从而不会忽略调试信息。

在异常处理方面，所有的handle方法没有实现，从而没有回滚，手动添加了相应的回滚方法，避免数据库和卡片相应的不同步。

整个实验过程中，对于RFID-GUI-DB的工作模式进行了熟悉，其中RFID通信过程存在QString类型与char\*数组的转换，此处需要手动转换，而不能采用QT自带的函数，由于试验箱为北京博创，在其他课程中也有使用，在与嵌入式系统老师和接口技术老师交流过程中了解到，串口通信所需依赖少，且收发较为底层，但是相应的错误信息不明显，作为试验箱或者非移动式的读卡器采用USB可能更为合适，比如嵌入式系统的相应系统烧入使用串口，固化系统之后USB则更为稳定和方便。对于串口调试，老师也提供了比对的相应调试方法，最终完成了相应的写卡操作。

本次实验，熟悉了射频系统整个系统的工作模式，同时能够完成软硬的联调，作为物联网核心组成部分，提高了系统能力，同时也熟悉了更多的调试技巧(如采用RFID模拟器进行软件模拟从而无需硬件调试，比对标准程序数据包和自己的数据包等)。

感谢在实验过程中提供帮助的老师、助教和同学以及相应参考文献的提供者。

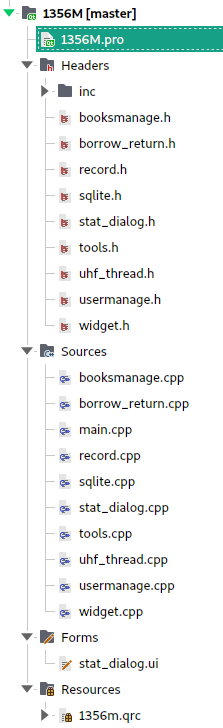
# 参考文献

1. 甘早斌，李开，卢宏伟.物联网识别技术及应用.北京：清华大学出版社.
2. 射频识别技术及应用实验指导书，北京博创
3. Qt官方文档

<https://doc.qt.io/>

# 程序源码

### 项目架构



程序架构图

## Qt Project Profile

QT       += core gui serialport sql

QT       += charts

greaterThan(QT\_MAJOR\_VERSION, 4): QT += widgets

TARGET = 1356M

TEMPLATE = app

SOURCES += main.cpp\

    stat\_dialog.cpp \

        widget.cpp \

    usermanage.cpp \

    booksmanage.cpp \

    tools.cpp \

    sqlite.cpp \

    uhf\_thread.cpp \

    borrow\_return.cpp \

    record.cpp

HEADERS  += widget.h \

    stat\_dialog.h \

    usermanage.h \

    booksmanage.h \

    inc/m1356dll.h \

    inc/m1356dll\_global.h \

    tools.h \

    sqlite.h \

    uhf\_thread.h \

    borrow\_return.h \

    record.h

LIBS += -L/home/hover/Desktop/Labs/HUST\_RFID\_Labs/Lab5/1356M\_Linux/M1356Dll\_Linux -lM1356Dll

win32:CONFIG(release, debug|release): LIBS += -L$$PWD/lib/ -lM1356Dll

**else**:win32:CONFIG(debug, debug|release): LIBS += -L$$PWD/lib/ -lM1356Dll

**else**:unix: LIBS += -L$$PWD/lib/ -lM1356Dll

INCLUDEPATH += $$PWD/

DEPENDPATH += $$PWD/

win32-g++:CONFIG(release, debug|release): PRE\_TARGETDEPS += $$PWD/lib/libM1356Dll.a

**else**:win32-g++:CONFIG(debug, debug|release): PRE\_TARGETDEPS += $$PWD/lib/libM1356Dll.a

**else**:win32:!win32-g++:CONFIG(release, debug|release): PRE\_TARGETDEPS += $$PWD/lib/M1356Dll.lib

**else**:win32:!win32-g++:CONFIG(debug, debug|release): PRE\_TARGETDEPS += $$PWD/lib/M1356Dll.lib

**else**:unix: PRE\_TARGETDEPS += $$PWD/lib/libM1356Dll.a

RESOURCES += \

    1356m.qrc

STATECHARTS +=

FORMS += \

    stat\_dialog.ui

## BooksManage

代码 7.1 BookManager.h

#ifndef BOOKSMANAGE\_H

#define BOOKSMANAGE\_H

#include <QObject>

#include <QWidget>

#include <QVBoxLayout>

#include <QHBoxLayout>

#include <QGridLayout>

#include <QPushButton>

#include <QLineEdit>

#include <QLabel>

#include <QTableWidget>

#include <QHeaderView>

#include <QMessageBox>

#include <QGroupBox>

#include "sqlite.h"

#include "stat\_dialog.h"

#define Button\_Count\_BOOKS 4//按钮个数

#define Edit\_Count\_BOOKS 7//文本框个数

#define Label\_Count\_BOOKS 7//标签个数

#define Table\_Column\_BOOKS 7//表格列数

**enum** Button\_Index\_Books{Add\_Books = 0, Delete\_Books, Updata\_Books, Select\_Books};//读卡、添加按钮、删除按钮、更新按钮、搜索按钮

**enum** Edit\_Index\_Books{ID\_Books = 0, Name\_Books, Author\_Books, PublishingHouse\_Books, Count\_Books, Residue\_Books,Borrow\_Days};//编号、书名、作者、出版社、总数、剩余

**class** BooksManage : **public** QWidget//图书管理界面

{

    Q\_OBJECT

**public**:

**explicit** BooksManage(QWidget \*parent = 0);

**void** SetSlot();//设置槽函数

**void** ShowTable(QSqlQuery query);//显示表函数

**void** ClearEdit();//清空文本框

**void** Clear();//清空文本框和表格信息

**void** SetCard(QString cardID);

**public** slots:

**void** add\_books();//添加按钮槽

**void** delete\_books();//删除按钮槽

**void** updata\_books();//更新按钮槽

**void** select\_books();//搜索按钮槽

**void** get\_table\_line(**int** row, **int** col);//单击表格一行触发的槽

**void** stat\_books();

**private**:

    QPushButton \*Button[Button\_Count\_BOOKS];//按钮

    QLineEdit \*Edit[Edit\_Count\_BOOKS];//文本框

    QLabel \*Label[Label\_Count\_BOOKS];//标签

    QTableWidget \*Table;//表格

    Sqlite \*sql;//数据库相关操作类

    QPushButton\* stat\_Button;

};

#endif // BOOKSMANAGE\_H

代码 7.2 BookManager.cpp

#include "booksmanage.h"

BooksManage::BooksManage(QWidget \*parent) : QWidget(parent)

{

    QString LabelName[] = {"卡号：", "书名：", "作者：", "出版社：", "总数（本）", "剩余（本）","可借阅时间（天）"};//标签文本

    QString ButtonName[] = {"添加", "删除", "修改", "搜索"};//按钮文本

    QVBoxLayout \*MainLayout = **new** QVBoxLayout();//主布局

    QHBoxLayout \*ButtonLayout = **new** QHBoxLayout();//按钮布局

    QHBoxLayout \*EditLayout = **new** QHBoxLayout();//文本框布局

    QHBoxLayout \*TableLayout = **new** QHBoxLayout();//表格布局

    QGroupBox \*BookTable = **new** QGroupBox();//表格区域

    QGroupBox \*BookInfo = **new** QGroupBox();//信息

    sql = **new** Sqlite();

**for**(**int** i = 0; i < Edit\_Count\_BOOKS; i++) //初始化文本框和标签

    {

        Edit[i] = **new** QLineEdit();

        Label[i] = **new** QLabel(LabelName[i]);

        EditLayout->addWidget(Label[i]);//将文本框和标签添加到布局中

        EditLayout->addWidget(Edit[i]);

    }

//    EditLayout->addWidget()

    QString pattern("[A-Fa-f9-0]\*");

    QRegExp regExp(pattern);

    Edit[ID\_Books]->setValidator(**new** QRegExpValidator(regExp, **this**));

    pattern="[9-0]{3}";

    regExp.setPattern(pattern);

    Edit[Count\_Books]->setValidator(**new** QRegExpValidator(regExp, **this**));

    Edit[Residue\_Books]->setValidator(**new** QRegExpValidator(regExp, **this**));

    Edit[Borrow\_Days]->setValidator(**new** QRegExpValidator(regExp, **this**));//可借天数为三位数

    BookInfo->setLayout(EditLayout);//设置信息组合框的布局

**for**(**int** i = 0; i < Button\_Count\_BOOKS; i++)//初始化按钮

    {

        Button[i] = **new** QPushButton();

        Button[i]->setText(ButtonName[i]);

        ButtonLayout->addWidget(Button[i]);//按钮添加到布局中

    }

    stat\_Button = **new** QPushButton();

    stat\_Button->setText("统计书籍借阅信息");

    ButtonLayout->addWidget(stat\_Button);

    ButtonLayout->addStretch(0);

    ButtonLayout->setSpacing(20);

    Table = **new** QTableWidget();

    Table->setSortingEnabled(**true**);

    Table->setColumnCount(Table\_Column\_BOOKS);

    Table->setSelectionBehavior ( QAbstractItemView::SelectRows);//选中整行

    Table->setEditTriggers ( QAbstractItemView::NoEditTriggers );//不可编辑

    Table->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);//列宽度自适应

    TableLayout->addWidget(Table);

    BookTable->setLayout(TableLayout);

    BookTable->setTitle("图书列表");

    MainLayout->addWidget(BookInfo);

    MainLayout->addLayout(ButtonLayout);

    MainLayout->addWidget(BookTable);

    MainLayout->setSpacing(10);

**this**->setLayout(MainLayout);

    SetSlot();

}

**void** BooksManage::SetSlot()//设置槽函数

{

    connect(Button[Add\_Books],SIGNAL(clicked()),**this**,SLOT(add\_books()));//添加按钮连接槽函数add\_books()

    connect(Button[Delete\_Books],SIGNAL(clicked()),**this**,SLOT(delete\_books()));//删除按钮连接槽函数delete\_books()

    connect(Button[Updata\_Books],SIGNAL(clicked()),**this**,SLOT(updata\_books()));//修改按钮连接槽函数updata\_books()

    connect(Button[Select\_Books],SIGNAL(clicked()),**this**,SLOT(select\_books()));//查找按钮连接槽函数select\_books()

    connect(stat\_Button,SIGNAL(clicked()),**this**,SLOT(stat\_books()));//stat\_books()

    connect(Table,SIGNAL(cellClicked(**int**,**int**)),**this**,SLOT(get\_table\_line(**int**, **int**)));//表格单击事件连接槽函数get\_table\_line(int, int)

}

**void** BooksManage::add\_books()//添加按钮槽函数

{

**int** residue;//图书的剩余数量

    /\*文本框为空时显示错误提示\*/

    QString LabelName[] = {"卡号：", "书名：", "作者：", "出版社：", "总数（本）"};

**for**(**int** i = 0; i < Edit\_Count\_BOOKS-1; i++)

    {

**if**(Edit[i]->text().isEmpty())

        {

            QMessageBox::warning(NULL, "warning", LabelName[i]+"不能为空！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

        }

    }

**if** (sql->SelectUser(Edit[ID\_Books]->text()).next())

    {

        QMessageBox::warning(NULL, "warning", "卡号已经注册为用户！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

    }

**if** (Edit[Residue\_Books]->text().toInt() > Edit[Count\_Books]->text().toInt())

    {

        QMessageBox::warning(NULL, "warning", "剩余数量不可以超出总数！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

    }

    /\*不填写剩余数量默认为总数量\*/

**if** (Edit[Residue\_Books]->text().isEmpty())

    {

        residue = Edit[Count\_Books]->text().toInt();

    }

**else**

    {

        residue = Edit[Residue\_Books]->text().toInt();

    }

    //向数据库中添加书籍

**bool** ret = sql->InsertBooks(Edit[ID\_Books]->text(),Edit[Name\_Books]->text(),Edit[Author\_Books]->text(),Edit[PublishingHouse\_Books]->text(),Edit[Count\_Books]->text().toInt(),residue,Edit[Borrow\_Days]->text().toInt());

**if**(!ret)

    {

        QMessageBox::warning(NULL, "warning", "添加失败，卡号已存在！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

    }

    QMessageBox::warning(NULL, "warning", "添加成功！", QMessageBox::Yes, QMessageBox::Yes);

    ClearEdit();    //清空文本框

    ShowTable(sql->SelectBooks());//更新表格

}

//删除按钮槽函数

**void** BooksManage::delete\_books()

{

**if** (!Edit[ID\_Books]->text().isEmpty() && sql->SelectUser(Edit[ID\_Books]->text()).next())

    {

        QMessageBox::warning(NULL, "warning", "卡号已经注册为用户！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

    }

**if** (!Edit[ID\_Books]->text().isEmpty() && !sql->SelectBooks(Edit[ID\_Books]->text()).next())

    {

        QMessageBox::warning(NULL, "warning", "卡号不存在！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

    }

**int** Count,Residue;

**if**(Edit[Residue\_Books]->text().isEmpty())

        Residue = -1;

**else**

        Residue = Edit[Residue\_Books]->text().toInt();

**if**(Edit[Count\_Books]->text().isEmpty())

        Count = -1;

**else**

        Count = Edit[Count\_Books]->text().toInt();

    //删除书籍

**bool** ret = sql->DeleteBooks(Edit[ID\_Books]->text(),Edit[Name\_Books]->text(),Edit[Author\_Books]->text(),Edit[PublishingHouse\_Books]->text(),Count,Residue);

**if**(!ret)

    {

        QMessageBox::warning(NULL, "warning", "删除失败！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

    }

    QMessageBox::warning(NULL, "warning", "删除成功！", QMessageBox::Yes, QMessageBox::Yes);

    ClearEdit();//清空文本框

    ShowTable(sql->SelectBooks());//更新表格

}

//修改按钮单击事件

**void** BooksManage::updata\_books()

{

**if** (!Edit[ID\_Books]->text().isEmpty() && sql->SelectUser(Edit[ID\_Books]->text()).next())

    {

        QMessageBox::warning(NULL, "warning", "卡号已经注册为用户！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

    }

**if** (!Edit[ID\_Books]->text().isEmpty() && !sql->SelectBooks(Edit[ID\_Books]->text()).next())

    {

        QMessageBox::warning(NULL, "warning", "卡号不存在！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

    }

**if** (Edit[Residue\_Books]->text().toInt() > Edit[Count\_Books]->text().toInt())

    {

        QMessageBox::warning(NULL, "warning", "剩余数量不可以超出总数！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

    }

    //修改书籍信息

**bool** ret = sql->UpdataBooks(Edit[ID\_Books]->text(),Edit[Name\_Books]->text(),Edit[Author\_Books]->text(),Edit[PublishingHouse\_Books]->text(),Edit[Count\_Books]->text().toInt(), Edit[Residue\_Books]->text().toInt(),Edit[Borrow\_Days]->text().toInt());

**if**(!ret)

    {

        QMessageBox::warning(NULL, "warning", "修改失败！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

    }

    QMessageBox::warning(NULL, "warning", "修改成功！", QMessageBox::Yes, QMessageBox::Yes);

    ClearEdit();//清空文本框

    ShowTable(sql->SelectBooks());//更新表格

}

//搜索按钮单击事件

**void** BooksManage::select\_books()

{

    QSqlQuery query;

**if**(Edit[Count\_Books]->text().isEmpty())

        query = sql->SelectBooks(Edit[ID\_Books]->text(),Edit[Name\_Books]->text(),Edit[Author\_Books]->text(),Edit[PublishingHouse\_Books]->text());

**else**

        query = sql->SelectBooks(Edit[ID\_Books]->text(),Edit[Name\_Books]->text(),Edit[Author\_Books]->text(),Edit[PublishingHouse\_Books]->text(),Edit[Count\_Books]->text().toInt());

    ShowTable(query);//更新表格

    ClearEdit();//清空文本框

}

//显示表格

**void** BooksManage::ShowTable(QSqlQuery query)

{

    Table->setHorizontalHeaderLabels(QStringList()<<"卡号"<<"书名"<<"作者"<<"出版社"<<"总计（本）"<<"剩余（本）"<<"可借阅时间（天）");

**if**(!query.next())

    {

        Table->setRowCount(0);//表格设置行数

**return**;

    }

    /\*计算record表中数据行数\*/

    query.last();//跳转到最后一条数据

**int** nRow = query.at() + 1;//取所在行数

    Table->setRowCount(nRow);//表格设置行数

**int** row = 0;

    query.first();//返回第一条数据

**do**

    {

**for** (**int** col = 0; col<Table->columnCount(); col++)//按字段添加数据

        {

            //表格中添加数据库中的数据

            Table->setItem(row, col, **new** QTableWidgetItem(query.value(col).toString()));

        }

        row++;//行数增加

    }**while**(query.next());

}

**void** BooksManage::stat\_books()

{

    stat\_dialog\* statdialog=**new** stat\_dialog(**this**);

    statdialog->sql=sql;

    statdialog->exec();

    //window.

    //window.setCentralWidget(chartView);

    //window.resize(420, 300);

    //window.show();

}

//清空文本框

**void** BooksManage::ClearEdit()

{

**for**(**int** i = 0; i < Edit\_Count\_BOOKS; i++)

    {

        Edit[i]->clear();

    }

}

//单击表格 在文本框中显示表格点击的行的数据

**void** BooksManage::get\_table\_line(**int** row, **int** col)

{

**for**(**int** i = 0; i < Edit\_Count\_BOOKS; i++)

    {

        Edit[i]->setText(Table->item(row,i)->text());

    }

}

**void** BooksManage::SetCard(QString cardID)

{

    Edit[ID\_Books]->setText(cardID);

}

//清空文本框和更新表格

**void** BooksManage::Clear()

{

    ClearEdit();

    ShowTable(sql->SelectBooks());

}

## Borrow\_Return

代码 7.3 Borrow\_Return.h

#ifndef BORROW\_RETURN\_H

#define BORROW\_RETURN\_H

#include <QObject>

#include <QWidget>

#include <QVBoxLayout>

#include <QHBoxLayout>

#include <QGridLayout>

#include <QPushButton>

#include <QLineEdit>

#include <QLabel>

#include <QTableWidget>

#include <QHeaderView>

#include <QMessageBox>

#include <QGroupBox>

#include <QCheckBox>

#include <QDateTime>

#include <QRadioButton>

#include <QButtonGroup>

#include "sqlite.h"

#include "uhf\_thread.h"

#include "tools.h"

#define Edit\_Count\_BORROW\_RETURN 4//用户信息文本框个数

#define Label\_Count\_BORROW\_RETURN 4//用户信息标签个数

#define Table\_Column\_BORROW\_RETURN 7//表格列数

**enum** Edit\_Index\_User\_Borrow{CardId\_User\_Borrow = 0, Name\_User\_Borrow, Gender\_User\_Borrow, Age\_User\_Borrow};//卡号、姓名、性别、年龄

**class** Borrow\_Return : **public** QWidget//借书界面

{

    Q\_OBJECT

**public**:

**explicit** Borrow\_Return(QWidget \*parent = 0);

**void** ShowTable(QSqlQuery query);//显示表格

**void** SetInfo(QString cardID);//获取卡号

**void** Clear();//清空文本框和表格信息

**void** checkReturnDate(QString userID);//检查该用户是否有快逾期的书籍

    QString last\_notification\_ID;

**private**:

    QLineEdit \*Edit\_User[Edit\_Count\_BORROW\_RETURN];//用户文本框

    QLabel \*Label\_User[Label\_Count\_BORROW\_RETURN];//用户标签

    QRadioButton \*Borrow,\*Return; //借书 还书 单选按钮

    QButtonGroup \*Function;//选择功能（借书、还书）

    QTableWidget \*Table;//表格

    Sqlite \*sql;//数据库操作相关类

    Tools \*tools;//工具类

signals:

**void** set\_record(QString userID, QString bookID);

**void** cycle();//循环读取卡号

};

#endif // Borrow\_Return\_H

代码 7.4 Borrow\_Return.cpp

#include "borrow\_return.h"

//借书界面

Borrow\_Return::Borrow\_Return(QWidget \*parent) : QWidget(parent)

{

    tools = **new** Tools();

    QString LabelNameUser[] = {"卡号：", "姓名：", "性别：", "年龄："}; //标签文本

    //布局

    QGridLayout \*MainLayout = **new** QGridLayout();//主布局

    QVBoxLayout \*UserLayout = **new** QVBoxLayout();//用户区域布局

    QVBoxLayout \*RightLayout = **new** QVBoxLayout();//右侧布局

    QHBoxLayout \*ButtonLayout = **new** QHBoxLayout();//右侧布局

    //组合框

    QGroupBox \*BooksGroupBox = **new** QGroupBox();

    QGroupBox \*UserGroupBox = **new** QGroupBox();

    sql = **new** Sqlite();

    //初始化文本框和标签 将文本框和标签添加到布局中

**for**(**int** i = 0; i < Edit\_Count\_BORROW\_RETURN; i++)

    {

        QHBoxLayout \*Layout = **new** QHBoxLayout();

        Edit\_User[i] = **new** QLineEdit();

        Label\_User[i] = **new** QLabel(LabelNameUser[i]);

        //Edit\_User[i]->setFocusPolicy(Qt::NoFocus); //设置为禁止编辑

        Layout->addWidget(Label\_User[i]);

        Layout->addWidget(Edit\_User[i]);

        UserLayout->addLayout(Layout);

    }

    //借还书单选按钮

    Borrow = **new** QRadioButton("借书");

    Return = **new** QRadioButton("还书");

    Borrow->setChecked(**true**);

    Function = **new** QButtonGroup();

    Function->addButton(Borrow);//单选按钮加入按钮组

    Function->addButton(Return);

    ButtonLayout->addWidget(Borrow);

    ButtonLayout->addWidget(Return);

    UserLayout->addLayout(ButtonLayout);

    UserGroupBox->setTitle("用户信息");//设置标题

    UserGroupBox->setLayout(UserLayout);

    UserGroupBox->setFixedSize(200,300);//设置大小

    Table = **new** QTableWidget();//表格

    Table->setColumnCount(Table\_Column\_BORROW\_RETURN);//设置列数

    Table->setSelectionBehavior ( QAbstractItemView::SelectRows);//选择方式为选中整行

    Table->setEditTriggers ( QAbstractItemView::NoEditTriggers );//不可编辑

    Table->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);//列宽度自适应

    RightLayout->addWidget(Table);

    BooksGroupBox->setTitle("借书列表");//设置组合框标题

    BooksGroupBox->setLayout(RightLayout);

    /\*设置图片\*/

    QLabel \*Picture = **new** QLabel();

    QImage \*jpg = **new** QImage(":/img/img/book.jpg");

    Picture->setPixmap(QPixmap::fromImage(\*jpg));

    MainLayout->addWidget(UserGroupBox,0,0,1,1);

    MainLayout->addWidget(BooksGroupBox,0,1,2,1);

    MainLayout->addWidget(Picture,1,0,1,1);

    MainLayout->setSpacing(20);

**this**->setLayout(MainLayout);

    // for test

    QString borrow\_time = tools->CurrentDateTime()

}

//表格显示

**void** Borrow\_Return::ShowTable(QSqlQuery query)

{

    //设置表格表头

    Table->setHorizontalHeaderLabels(QStringList()<<"卡号"<<"书名"<<"作者"<<"出版社"<<"总数（本）"<<"剩余（本）"<<"可借阅时间（天）");

**if**(!query.next())

    {

        Table->setRowCount(0);//表格设置行数

**return**;

    }

    /\*计算record表中数据行数\*/

    query.last();//跳转到最后一条数据

**int** nRow = query.at() + 1;//取所在行数

    Table->setRowCount(nRow);//表格设置行数

**int** row = 0;

    query.first();//返回第一条数据

**do**

    {

**for** (**int** col = 0; col<7; col++)//按字段添加数据

        {

            //表格中添加数据库中的数据

            Table->setItem(row, col, **new** QTableWidgetItem(query.value(col).toString()));

        }

        row++;//行数增加

    }**while**(query.next());

}

**void** Borrow\_Return::checkReturnDate(QString userID)

{

**if**(userID==last\_notification\_ID)

    {

**return**;

    }

    qDebug()<<"checkReturnDate"<<"borrow and return"<<endl;

    QSqlQuery query;

    //userID="999";

    QString cmd="select\

    record\_15693.booksID,\

    record\_15693.borrow\_time ,\

    books\_15693.borrowed\_days\

    from record\_15693\

    left join  books\_15693 on record\_15693.booksID = books\_15693.booksID\

    where record\_15693.cardID=='"+userID+"'   ";

    query.exec(cmd);

**if**(!query.next())

    {

        //no result

        qDebug()<<"no result"<<endl;

**return**;

    }

    qDebug()<<query.size()<<endl;

    QString bookID;

    QDateTime borrowDate;

**int** borrowed\_days;

**int** warnning\_days=10;

    QDateTime dueDate;

    QDateTime nowDate=QDateTime::currentDateTime();

    query.first();//返回第一条数据

**do**

    {

        //check numd;

        qDebug()<<query.value(0)<<query.value(1)<<query.value(2)<<endl;

        bookID=query.value(0).toString();

        borrowDate = QDateTime::fromString(query.value(1).toString(), "yyyy-MM-dd hh:mm:ss");

        borrowed\_days=query.value(2).toInt();

        dueDate=borrowDate.addDays(borrowed\_days);

**int** ndaysec = 24\*60\*60;

        uint stime = nowDate.toTime\_t();

        uint etime = dueDate.toTime\_t();

        uint left\_days=(etime - stime)/(ndaysec) + ((etime - stime)%(ndaysec)+(ndaysec-1))/(ndaysec) - 1;

        qDebug() << "Day = " <<left\_days<<endl;

**if**(left\_days<0)

        {

            QMessageBox::warning(NULL, "逾期警告！", "书籍："+bookID+"\n"+"还书日期："+dueDate.toString()+"\n"+"逾期时间："+QString::number(-left\_days, 10)+"Days\n", QMessageBox::Yes, QMessageBox::Yes);

        }

**else** **if**(left\_days<warnning\_days)

        {

            QMessageBox::warning(NULL, "还书警告！", "书籍："+bookID+"\n"+"还书日期："+dueDate.toString()+"\n"+"剩余时间："+QString::number(left\_days, 10)+"Days\n", QMessageBox::Yes, QMessageBox::Yes);

        }

        qDebug()<<bookID<<borrowDate<<borrowed\_days<<endl;

//        bookID=query.value(0)

    }**while**(query.next());

    last\_notification\_ID=userID;

}

**bool** canBorrow(QString userID)

{

    qDebug()<<"checkReturnDate"<<"borrow and return"<<endl;

    QSqlQuery query;

    QString cmd="select\

    record\_15693.booksID,\

    record\_15693.borrow\_time ,\

    books\_15693.borrowed\_days\

    from record\_15693\

    left join  books\_15693 on record\_15693.booksID = books\_15693.booksID\

    where record\_15693.cardID=='"+userID+"'   ";

    query.exec(cmd);

**if**(!query.next())

    {

        //no result

        qDebug()<<"no result"<<endl;

**return** **true**;

    }

    qDebug()<<query.size()<<endl;

    QString bookID;

    QDateTime borrowDate;

**int** borrowed\_days;

**int** warnning\_days=10;

    QDateTime dueDate;

    QDateTime nowDate=QDateTime::currentDateTime();

**bool** can\_borrow=**true**;

    query.first();//返回第一条数据

**do**

    {

        //check numd;

        qDebug()<<query.value(0)<<query.value(1)<<query.value(2)<<endl;

        bookID=query.value(0).toString();

        borrowDate = QDateTime::fromString(query.value(1).toString(), "yyyy-MM-dd hh:mm:ss");

        borrowed\_days=query.value(2).toInt();

        dueDate=borrowDate.addDays(borrowed\_days);

**int** ndaysec = 24\*60\*60;

        uint stime = nowDate.toTime\_t();

        uint etime = dueDate.toTime\_t();

        uint left\_days=(etime - stime)/(ndaysec) + ((etime - stime)%(ndaysec)+(ndaysec-1))/(ndaysec) - 1;

        qDebug() << "Day = " <<left\_days<<endl;

**if**(left\_days<0)

        {

            can\_borrow=**false**;

            QMessageBox::warning(NULL, "逾期警告！", "书籍："+bookID+"\n"+"还书日期："+dueDate.toString()+"\n"+"逾期时间："+QString::number(-left\_days, 10)+"Days\n", QMessageBox::Yes, QMessageBox::Yes);

        }

        qDebug()<<bookID<<borrowDate<<borrowed\_days<<endl;

    }**while**(query.next());

**return** can\_borrow;

}

//设置用户信息(卡ID)

**void** Borrow\_Return::SetInfo(QString cardID)

{

    //将用户信息显示到文本框中

    QSqlQuery query = sql->SelectUser(cardID);

**if**(query.next())//如果是用户

    {

        checkReturnDate(cardID);

        //检查卡是否已被挂失

**if**(query.value(5) == 0)

        {

            QMessageBox::warning(NULL, "warning", "此卡已被挂失,请联系管理员！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

        }

**for**(**int** i=0; i < Edit\_Count\_BORROW\_RETURN; i++)

        {

            Edit\_User[i]->setText(query.value(i).toString());

        }

        //check if there are any books need to be return

        //将书信息显示到表格中

        ShowTable(sql->SelectBooksOfBorrow(cardID));//显示表格内容

**return**;

    }

    query = sql->SelectBooks(cardID);

**if**(query.next())//如果是书

    {

**if**(Edit\_User[CardId\_User\_Borrow]->text().isEmpty())

        {

**return**;

        }

**if**(Borrow->isChecked())

        {

            //check user credit

**bool** can\_borrow=canBorrow(Edit\_User[CardId\_User\_Borrow]->text());

**if**(!can\_borrow)

            {

                QMessageBox::warning(NULL, "warning", "存在逾期书籍，请先还书！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

            }

            //在该借书记录中找到该用户已借过该书，则不能再借

**if**(sql->SelectRecord(Edit\_User[CardId\_User\_Borrow]->text(), query.value(0).toString()).next())

            {

**return**;

            }

            //剩余书为0，不能再借

**if**(query.value(5).toInt() <= 0)

            {

**return**;

            }

            QString borrow\_time = tools->CurrentDateTime();

            //借书

            //set RFID Record

            set\_record(Edit\_User[CardId\_User\_Borrow]->text(), query.value(0).toString());

            //widget->Set

**if**(sql->InsertRecord(Edit\_User[CardId\_User\_Borrow]->text(), query.value(0).toString(),borrow\_time,NULL))//将用户ID和书籍编号添加到数据表中

            {

                //书籍的剩余数量-1

                sql->UpdataBooks(query.value(0).toString(),query.value(1).toString(),query.value(2).toString(),query.value(3).toString(),query.value(4).toInt(),query.value(5).toInt()-1,query.value(6).toInt());

            }

        }

        //还书按钮被选择

**else**

        {

            //查询不到借书记录

**if**(!sql->SelectRecord(Edit\_User[CardId\_User\_Borrow]->text(), query.value(0).toString()).next())

            {

                QMessageBox::warning(NULL, "warning", "还书错误！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

            }

            QString sql\_cmd="SELECT \* FROM record\_15693 WHERE cardID = "+Edit\_User[CardId\_User\_Borrow]->text()+" AND booksID = "+query.value(0).toString()+" AND return\_time ~= """;

            QSqlQuery myquery;

            myquery.exec(sql\_cmd);

**if**(!query.next())//query is empty

            {

                QMessageBox::warning(NULL, "warning", "还书错误！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

            }

            QString return\_time = tools->CurrentDateTime();

**if**(sql->UpdataRecord(Edit\_User[CardId\_User\_Borrow]->text(), query.value(0).toString(),return\_time))//更新还书日期

            //if(sql->DeleteRecord(Edit\_User[CardId\_User\_Borrow]->text(), query.value(0).toString()))//将用户ID和书籍编号添加到数据表中

            {

                //书籍的剩余数量+1

                sql->UpdataBooks(query.value(0).toString(),query.value(1).toString(),query.value(2).toString(),query.value(3).toString(),query.value(4).toInt(),query.value(5).toInt()+1,query.value(6).toInt());

            }

        }

        ShowTable(sql->SelectBooksOfBorrow(Edit\_User[0]->text()));//显示表格内容

    }

}

//清空文本框和刷新表格

**void** Borrow\_Return::Clear()

{

**for**(**int** i = 0; i < Edit\_Count\_BORROW\_RETURN; i++)

    {

        Edit\_User[i]->clear();

    }

    ShowTable(sql->SelectBooksOfBorrow(Edit\_User[0]->text()));//显示表格内容

}

## Sqlite

代码7.5 sqlite.h

#ifndef SQLITE\_H

#define SQLITE\_H

#include <QtSql/QSqlDatabase>

#include <QtSql/QSqlQuery>

#include <QDebug>

#define DATABASE "database.db"//数据库名

**class** Sqlite

{

**public**:

    Sqlite();

**bool** Connect();//连接数据库

**bool** ExecSQL(QString cmd);//执行Sql语句

**bool** Insert(QString table, QString value);//插入

**bool** Delete(QString table, QString where);//删除

**bool** Updata(QString table, QString value,QString where);//修改

    QSqlQuery Select(QString table, QString value, QString where);//查询

**bool** InsertUser(QString cardID, QString name, QString gender, **int** age, QString tele, **bool** status);//插入用户表

**bool** InsertBooks(QString booksID, QString name, QString author, QString publishing\_house, **int** count, **int** residue, **int** borrowed\_days);//插入图书表

**bool** InsertRecord(QString cardID, QString booksID, QString borrow\_time, QString return\_time);//插入记录表

**bool** DeleteUser(QString cardID = NULL, QString name = NULL, QString gender = NULL, **int** age = -1);//删除用户表中数据

**bool** DeleteBooks(QString booksID = NULL, QString name = NULL, QString author = NULL, QString publishing\_house = NULL, **int** count = -1, **int** residue = -1);//删除图书表中数据

**bool** DeleteRecord(QString cardID = NULL, QString booksID = NULL);//删除记录表中数据

**bool** UpdataUser(QString cardID, QString name, QString gender, **int** age, QString tele, **bool** status);//修改用户表中信息

**bool** UpdataUserStatus(QString cardID, **bool** status);//修改用户状态

**bool** UpdataRecord(QString cardID, QString booksID, QString return\_time);//更新还书时间

**bool** UpdataBooks(QString booksID, QString name, QString author, QString publishing\_house, **int** count, **int** residue,**int** borrowed\_days);//修改图书表中信息

    QSqlQuery SelectUser(QString cardID = NULL, QString name = NULL, QString tele = NULL, QString gender = NULL, **int** age = -1);//查找用户表中信息

    QSqlQuery SelectBooks(QString booksID = NULL, QString book\_name = NULL, QString author = NULL, QString press = NULL, **int** count = -1);//查找图书表中信息

    QSqlQuery SelectRecord(QString cardID = NULL, QString booksID = NULL);

    QSqlQuery SelectBooksOfBorrow(QString cardID);//查找某用户借的书

**bool** CreateBookTable();

**bool** CreateUserTable();

**bool** CreateRecordTable();

    ~Sqlite();

**private**:

    QSqlDatabase db;

};

#endif // SQLITE\_H

代码7.6 sqlite.cpp

#include "sqlite.h"

#include<iostream>

Sqlite::Sqlite()

{

}

/\*连接数据库\*/

**bool** Sqlite::Connect()

{

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

    db.setDatabaseName(DATABASE);

**if**(!db.open())

    {

        qDebug() << " connect failed!";

**return** **false**;

    }

**else**

    {

        qDebug() << "connect success!";

    }

    CreateUserTable();

    CreateBookTable();

    CreateRecordTable();

**return** **true**;

}

//打印SQL语句

**bool** Sqlite::ExecSQL(QString cmd)

{

    QSqlQuery query;

    qDebug()<<cmd.toUtf8().data();

**return** query.exec(cmd);

}

//添加语句

**bool** Sqlite::Insert(QString table, QString value)

{

    QString cmd = "insert into " + table + " values(" + value + ");";

    qDebug() << "SQL: " << cmd;

**return** ExecSQL(cmd);

}

//删除语句

**bool** Sqlite::Delete(QString table, QString where)

{

    QString cmd = "delete from " + table + " where " + where + ";";

**return** ExecSQL(cmd);

}

//修改语句

**bool** Sqlite::Updata(QString table, QString value,QString where)

{

    QString cmd = "update " + table + " set " + value +" where " + where + ";";

**return** ExecSQL(cmd);

}

//查询语句

QSqlQuery Sqlite::Select(QString table, QString value, QString where)

{

    QString cmd;

**if**(where.isEmpty())

    {

        cmd = "select " + value + " from " + table + ";";

    }

**else**

    {

        cmd = "select " + value + " from " + table + " where " + where + ";";

    }

    QSqlQuery query;

    qDebug()<<cmd.toUtf8();

    query.exec(cmd);

**return** query;

}

//向user表中添加

**bool** Sqlite::InsertUser(QString cardID, QString name, QString gender, **int** age, QString tele, **bool** status)

{

**return** Insert("user\_15693", "'"+cardID+"', '"+name+"', '"+gender+"', "+QString::number(age)+", '"+tele+"', "+QString::number(status));

}

//向books表中添加

**bool** Sqlite::InsertBooks(QString booksID, QString name, QString author, QString publishing\_house, **int** count, **int** residue, **int** borrowed\_days)

{

**return** Insert("books\_15693", "'"+booksID+"', '"+name+"', '"+author+"', '"+publishing\_house+"', "+QString::number(count)+", "+QString::number(residue)+", "+QString::number(borrowed\_days));

}

//向record表中添加

**bool** Sqlite::InsertRecord(QString cardID, QString booksID, QString borrow\_time, QString return\_time)

{

**return** Insert("record\_15693", "'"+cardID+"', '"+booksID+"','"+ borrow\_time+"','"+ return\_time+"'");

    //return Insert("record\_15693", "'"+cardID+"', '"+booksID+"'");

}

//删除user表中数据

**bool** Sqlite::DeleteUser(QString cardID, QString name, QString gender, **int** age)

{

    QString where;

**if**( !cardID.isEmpty() )

        where += ("cardID = '" + cardID +"' ");

**if**( !name.isEmpty() )

    {

**if**(where.isEmpty())

            where += ("name = '" + name+"' ");

**else**

            where += ("and name = '" + name+"' ");

    }

**if**( !gender.isEmpty() )

    {

**if**(where.isEmpty())

            where += ("gender = '" + gender+"' ");

**else**

            where += ("and gender = '" + gender+"' ");

    }

**if**( age != -1 )

    {

**if**(where.isEmpty())

            where += ("age = " + QString::number(age));

**else**

            where += ("and age = " + QString::number(age));

    }

**return** Delete("user\_15693", where);

}

//删除books表中数据

**bool** Sqlite::DeleteBooks(QString booksID, QString name, QString author, QString publishing\_house, **int** count, **int** residue)

{

    QString where;

**if**( !booksID.isEmpty() )

        where += ("booksID = '" + booksID +"' ");

**if**( !name.isEmpty() )

    {

**if**(where.isEmpty())

            where += ("name = '" + name+"' ");

**else**

            where += ("and name = '" + name+"' ");

    }

**if**( !author.isEmpty() )

    {

**if**(where.isEmpty())

            where += ("author = '" + author+"' ");

**else**

            where += ("and author = '" + author+"' ");

    }

**if**( !publishing\_house.isEmpty() )

    {

**if**(where.isEmpty())

            where += ("publishing\_house = '" + publishing\_house+"' ");

**else**

            where += ("and publishing\_house = '" + publishing\_house+"' ");

    }

**if**( count != -1 )

    {

**if**(where.isEmpty())

            where += ("count = " + QString::number(count)+" ");

**else**

            where += ("and count = " + QString::number(count)+" ");

    }

**if**( residue != -1 )

    {

**if**(where.isEmpty())

            where += ("residue = " + QString::number(residue)+" ");

**else**

            where += ("and residue = " + QString::number(residue)+" ");

    }

**return** Delete("books\_15693", where);

}

//删除record表中数据

**bool** Sqlite::DeleteRecord(QString cardID, QString booksID)

{

    QString where;

**if**( !cardID.isEmpty() )

        where += ("cardID = '" + cardID +"' ");

**if**( !booksID.isEmpty() )

    {

**if**(where.isEmpty())

            where += ("booksID = '" + booksID+"' ");

**else**

            where += ("and booksID = '" + booksID+"' ");

    }

**return** Delete("record\_15693", where);

}

//修改user表中数据

**bool** Sqlite::UpdataUser(QString cardID, QString name, QString gender, **int** age, QString tele, **bool** status)

{

**return** Updata("user\_15693","cardID = '"+cardID+"', name = '"+name+"', gender = '"+gender+"', age = "+QString::number(age)+"', tele = '"+tele+", status = "+QString::number(status), "cardID = '"+cardID+"'");

}

//修改用户状态

**bool** Sqlite::UpdataUserStatus(QString cardID, **bool** status)

{

**return** Updata("user\_15693","status = "+QString::number(status),"cardID = '"+cardID+"'");

}

**bool** Sqlite::UpdataRecord(QString cardID, QString booksID, QString return\_time)//更新还书时间

{

**return** Updata("record\_15693", "return\_time = '"+return\_time+"'","cardID = '" +cardID+"' and booksID = '"+booksID+"'");

    //update record\_15693 set return\_time = '20' where cardID = '5A0C6B2C000104E0' and booksID = 'E5EA6A2C000104E0';

}

//修改books表中数据

**bool** Sqlite::UpdataBooks(QString booksID, QString name, QString author, QString publishing\_house, **int** count, **int** residue,**int** borrowed\_days)

{

**return** Updata("books\_15693","booksID = '"+booksID+"', book\_name = '"+name+"', author = '"+author+"', press = '"+publishing\_house+"', count = "+QString::number(count)+", residue = "+QString::number(residue) +", borrowed\_days = "+QString::number(borrowed\_days), "booksID = '"+booksID+"'");

}

//查询user表中数据

QSqlQuery Sqlite::SelectUser(QString cardID, QString name, QString tele, QString gender, **int** age)

{

    QString where;

**if**( !cardID.isEmpty() )

        where += ("cardID = '" + cardID +"' ");

**if**( !name.isEmpty() )

    {

**if**(where.isEmpty())

            where += ("name = '" + name+"' ");

**else**

            where += ("and name = '" + name+"' ");

    }

**if**( !tele.isEmpty() )

    {

**if**(where.isEmpty())

            where += ("tele = '" + tele+"' ");

**else**

            where += ("and tele = '" + tele+"' ");

    }

**if**( !gender.isEmpty() )

    {

**if**(where.isEmpty())

            where += ("gender = '" + gender+"' ");

**else**

            where += ("and gender = '" + gender+"' ");

    }

**if**( age != -1 )

    {

**if**(where.isEmpty())

            where += ("age = " + QString::number(age));

**else**

            where += ("and age = " + QString::number(age));

    }

**return** Select("user\_15693", "\*", where);

}

//查询books表中数据

QSqlQuery Sqlite::SelectBooks(QString booksID, QString book\_name, QString author, QString press, **int** count)

{

    QString where;

**if**( !booksID.isEmpty() )

        where += ("booksID = '" + booksID +"' ");

**if**( !book\_name.isEmpty() )

    {

**if**(where.isEmpty())

            where += ("book\_name = '" + book\_name+"' ");

**else**

            where += ("and book\_name = '" + book\_name+"' ");

    }

**if**( !author.isEmpty() )

    {

**if**(where.isEmpty())

            where += ("author = '" + author+"' ");

**else**

            where += ("and author = '" + author+"' ");

    }

**if**( !press.isEmpty() )

    {

**if**(where.isEmpty())

            where += ("press = '" + press+"' ");

**else**

            where += ("and press = '" + press+"' ");

    }

**if**( count != -1 )

    {

**if**(where.isEmpty())

            where += ("count = " + QString::number(count));

**else**

            where += ("and count = " + QString::number(count));

    }

**return** Select("books\_15693", "\*", where);

}

QSqlQuery Sqlite::SelectRecord(QString cardID, QString booksID)

{

    QString where;

**if**( !cardID.isEmpty() )

        where += ("cardID = '" + cardID +"' ");

**if**( !booksID.isEmpty() )

    {

**if**(where.isEmpty())

            where += ("booksID = '" + booksID+"' ");

**else**

            where += ("and booksID = '" + booksID+"' ");

    }

**return** Select("record\_15693", "\*", where);

}

//查找借的书

QSqlQuery Sqlite::SelectBooksOfBorrow(QString cardID)

{

**return** Select("books\_15693", "\*", "booksID in (select booksID from record\_15693 where cardID = '"+cardID+"')");

}

**bool** Sqlite::CreateBookTable()

 {

     QSqlQuery query(db);

     //用户基本信息表

     //staus -0表示未激活，-1表示已激活

**bool** isTableExist = query.exec(QString("select count(\*) from sqlite\_master where type='table' and name='%1'").arg("books\_15693"));

**if**(isTableExist)

     {

         qDebug() << "books\_table has existed!";

         //return false;

     }

**bool** res = **false**;

     //图书基本信息表

     //图书名字和ID都表示一类书,borrowed\_days 可借出的时常

     res = query.exec("create table books\_15693 (booksID vchar,  book\_name vchar, author vchar, press vchar,  count int, residue int, borrowed\_days int, primary key (booksID))");

**if**(res)

         qDebug() << "create table books\_15693 success!";

**else**

         qDebug() << "create table books\_15693 fail!";

**return** res;

 }

**bool** Sqlite::CreateUserTable()

 {

     QSqlQuery query(db);

     //用户基本信息表

     //staus -0表示未激活，-1表示已激活

**bool** isTableExist = query.exec(QString("select count(\*) from sqlite\_master where type='table' and name='%1'").arg("user\_15693"));

**if**(isTableExist)

     {

         qDebug() << "books\_table has existed!";

         //return false;

     }

**bool** res = **false**;

     res = query.exec("create table user\_15693 (cardID vchar, name vchar, gender vchar, age int, tele char(11), status boolean, primary key (cardID))");

**if**(res)

         qDebug() << "create table user\_15693 success!";

**else**

         qDebug() << "create table user\_15693 fail!";

**return** res;

 }

**bool** Sqlite ::CreateRecordTable()

 {

     QSqlQuery query(db);

     //用户基本信息表

     //staus -0表示未激活，-1表示已激活

**bool** isTableExist = query.exec(QString("select count(\*) from sqlite\_master where type='table' and name='%1'").arg("record\_15693"));

**if**(isTableExist)

     {

         qDebug() << "record\_table has existed!";

         //return false;

     }

**bool** res = **false**;

     res = query.exec("create table record\_15693 (cardID vchar, booksID vchar, borrow\_time timestamp not null default current\_timestamp, return\_time timestamp default null,FOREIGN KEY (cardID ) REFERENCES user(cardID), FOREIGN KEY (booksID ) REFERENCES user(booksID))");

**if**(res)

         qDebug() << "create table record\_15693 success!";

**else**

         qDebug() << "create table record\_15693 fail!";

**return** res;

 }

## Stat\_dialog(统计图表绘制)

代码7.7 stat\_dialog.cpp

#include "stat\_dialog.h"

#include "ui\_stat\_dialog.h"

stat\_dialog::stat\_dialog(QWidget \*parent) :

    QDialog(parent),

    ui(**new** Ui::stat\_dialog)

{

    ui->setupUi(**this**);

    QBarSet \*set0 = **new** QBarSet("Books");

    QString sql\_cmd="SELECT booksID,count(\*) FROM record\_15693 group by booksID";

    QSqlQuery query;

    query.exec(sql\_cmd);

**if**(!query.next())

    {

**return**;

    }

    /\*计算record表中数据行数\*/

    query.last();//跳转到最后一条数据

**int** nRow = query.at() + 1;//取所在行数

**int** row = 0;

    QStringList categories;

    query.first();//返回第一条数据

**do**

    {

        \*set0 << query.value(1).toInt();

        categories << query.value(0).toString();

        row++;//行数增加

    }**while**(query.next());

    //\*set0 << 1 << 2 << 3 << 4 << 5 << 6;

    QBarSeries \*series = **new** QBarSeries();

    series->append(set0);

    QChart \*chart = **new** QChart();

    chart->addSeries(series);

    chart->setTitle("图书借阅统计");

    chart->setAnimationOptions(QChart::SeriesAnimations);

    QBarCategoryAxis \*axisX = **new** QBarCategoryAxis();

    axisX->append(categories);

    chart->addAxis(axisX, Qt::AlignBottom);

    series->attachAxis(axisX);

    QValueAxis \*axisY = **new** QValueAxis();

    axisY->setLabelFormat("%d");

//    axisY->setRange(0,15);

    chart->addAxis(axisY, Qt::AlignLeft);

    series->attachAxis(axisY);

    chart->legend()->setVisible(**true**);

    chart->legend()->setAlignment(Qt::AlignBottom);

    QChartView \*chartView = **new** QChartView(chart);

    chartView->setRenderHint(QPainter::Antialiasing);

    ui->graphicsView->setViewport(chartView);

}

stat\_dialog::~stat\_dialog()

{

**delete** ui;

}

## Tools工具类

代码7.8 Tools.h

#ifndef TOOLS\_H

#define TOOLS\_H

#include <QObject>

#include <QtSerialPort/QSerialPort>

#include <QtSerialPort/QSerialPortInfo>

#include <QStringList>

#include <QString>

#include <QDate>

#include <QDateTime>

#include <QAbstractItemModel>

#include <QFile>

#include <QFileDialog>

#include <QTextStream>

#include <QVariant>

**class** Tools : **public** QObject

{

    Q\_OBJECT

**public**:

**explicit** Tools(QObject \*parent = 0);

    QStringList getSerialName();

    QString CurrentDateTime();

    QString CurrentTime();

    QString CurrentMTime();

    QString CharStringtoHexString(QString space,**const** **char** \* src,**int** len);

    QString CharStringtoHexString(QString space,**const** **char** \* src,**int** start,**int** end);

    quint8 StringToHex(QString string, quint8 \*hex);

**void** export\_table(**const** QAbstractItemModel &model);

**bool** isOverdue(QString borrowTime, QString returnTime, **int** days);

**private**:

    QStringList \*list;

signals:

**public** slots:

};

#endif // TOOLS\_H

代码7.9 Tools.cpp

#include "tools.h"

#include <QDebug>

Tools::Tools(QObject \*parent) : QObject(parent)

{

    list = **new** QStringList();

}

//获取当前PC可用的串口名

QStringList Tools::getSerialName()

{

    QStringList temp;

    foreach (**const** QSerialPortInfo &info, QSerialPortInfo::availablePorts())

      {

          QSerialPort serial;

          serial.setPort(info);

**if** (serial.open(QIODevice::ReadWrite))

          {

**if**(! list->contains(info.portName(),Qt::CaseSensitive))

                  list->insert(0,info.portName());

              serial.close();

              temp << info.portName();

          }

      }

**for**(**int** i = 0 ; i < list->size() ; i ++)

    {

**if**(!temp.contains(list->at(i)))

            list->removeAt(i);

    }

**return** \*list;

}

///获取当前日期和时间

QString Tools::CurrentDateTime()

{

    QDateTime dt;

    QTime **time**;

    QDate date;

    dt.setTime(**time**.currentTime());

    dt.setDate(date.currentDate());

**return** dt.toString("yyyy-MM-dd hh:mm:ss");

}

///获取当前的时间

QString Tools::CurrentTime()

{

    QTime **time**;

**return** **time**.currentTime().toString("hh:mm:ss");

}

///获取当前的时间

QString Tools::CurrentMTime()

{

    QTime **time**;

**return** **time**.currentTime().toString("hh:mm:ss.zzz");

}

///普通字符串转为16进制字符串

QString  Tools::CharStringtoHexString(QString space, **const** **char** \* src, **int** len)

{

    QString hex = "";

**if**(space == NULL)

    {

**for**(**int** i = 0 ; i < len ; i ++)

        {

            hex += QString("%1").arg(src[i]&0xFF,2,16,QLatin1Char('0'));

        }

**return** hex.toUpper();

    }

**else**

    {

**for**(**int** i = 0 ; i < len ; i ++)

        {

            hex += space + QString("%1").arg(src[i]&0xFF,2,16,QLatin1Char('0'));

        }

**return** hex.right(hex.length() - space.length()).toUpper();

    }

}

//QString 转 Hex char \*

quint8 Tools::StringToHex(QString string, quint8 \*hex)

{

    QString temp;

    quint8 len = string.length();

**for**(quint8 i=0; i<len; i+=2)

    {

         temp = string.mid(i, 2);

         hex[i/2] = (quint8)temp.toInt(0,16);

    }

**return** len/2;

}

///普通字符串转为16进制字符串

QString  Tools::CharStringtoHexString(QString space, **const** **char** \* src, **int** start, **int** end)

{

    QString hex = "";

**if**(space == NULL)

    {

**for**(**int** i = start ; i < end ; i ++)

        {

            hex += QString("%1").arg(src[i]&0xFF,2,16,QLatin1Char('0'));

        }

**return** hex.toUpper();

    }

**else**

    {

**for**(**int** i = start ; i < end ; i ++)

        {

            hex += space + QString("%1").arg(src[i]&0xFF,2,16,QLatin1Char('0'));

        }

**return** hex.right(hex.length() - space.length()).toUpper();

    }

}

//用于导出数据库中的数据到文件，csv格式的文件可以用Excel打开

**void** Tools::export\_table(**const** QAbstractItemModel &model)

{

    QString fileName = QFileDialog::getSaveFileName(0, QObject::tr("保存记录"), "/", "files(\*.csv)");

    QFile file(fileName);

**if**(file.open(QFile::WriteOnly|QFile::Truncate)){

        QTextStream out(&file);

        QString str;

        str.clear();

**for**(**int** i=0; i<model.columnCount(); i++)

            str.append(model.headerData(i, Qt::Horizontal).toString()).append(",");

        out<<str<<"\r\n";

**for**(**int** row=0; row<model.rowCount(); row++){

            str.clear();

**for**(**int** col=0; col<model.columnCount(); col++)

                str.append(model.data(model.index(row,col)).toString()).append(",");

            out<<str<<"\r\n";

        }

        file.close();

    }

}

//两个时间只差与天数的比较

**bool** Tools::isOverdue(QString borrowTime, QString returnTime, **int** days)

{

    QTime start = QTime::fromString(borrowTime);

    QTime end = QTime::fromString(returnTime);

**int** secs = start.secsTo(end);

    qDebug() << secs;

**if**(days\*24\*3600 < secs)//未超时

**return** **true**;

**else**

**return** **false**;

}

## UHF\_Thread

代码7.10 UHF\_Thread.h

#ifndef UHF\_THREAD\_H

#define UHF\_THREAD\_H

#include <QObject>

#include <QThread>

#include <QSerialPort>

#include "inc/m1356dll.h"

#include "inc/m1356dll\_global.h"

**class** UHF\_Thread : **public** QThread //UHF读卡线程类

{

    Q\_OBJECT

**public**:

    QString last\_cmd;

**explicit** UHF\_Thread(QObject \*parent = 0);

**bool** UART\_Connect(QString ComName, **int** Baudrate);//串口连接

**bool** UART\_Disconnect();//串口关闭

**bool** ReadCardID();//发送读卡命令

**bool** InitUhf();//发送初始化UHF命令

**void** str2hex(**const** QString &str, quint8 \*buf, quint8 len);

**bool** SetCardRecord(QString userID,QString bookID);

**bool** GetCardRecord(QString userID,QString bookID);

**bool** nRunFlag;//线程run函数中的循环标志 为false时停止run函数中的循环

signals:

**void** receivedMsg(QByteArray);//发送接收的数据

**void** cycle();//循环读取卡号

**public** slots:

**private**:

    QSerialPort \*serialport;//串口操作类

    M1356Dll \*Dll;//dll库中类的对象

**protected**:

**void** run();

};

#endif // UHF\_THREAD\_H

代码7.11 UHF\_Thread.cpp

#include "uhf\_thread.h"

#define BUILD\_UINT16(loByte, hiByte) \

          ((uint16)(((loByte) & 0x00FF) + (((hiByte) & 0x00FF) << 8)))

UHF\_Thread::UHF\_Thread(QObject \*parent) : QThread(parent)

{

    serialport = **new** QSerialPort();

    Dll = **new** M1356Dll();

}

**void** UHF\_Thread::run()

{

**int** retryTimes = 45;//超时

**int** count = retryTimes;

**enum** {UHF\_RPC\_SOF = 0, UHF\_RPC\_LEN, UHF\_RPC\_dev\_id, UHF\_RPC\_CMD, UHF\_RPC\_STA, UHF\_RPC\_DAT, UHF\_RPC\_EOF};

    //帧的标志，数据的长度，长度，指令，状态，数据数组下标

    QByteArray data;

**while**(nRunFlag)

    {

**if**(serialport->bytesAvailable() >= 4) //只有串口中数据有4个字节时才开始读取

        {

            data = serialport->readAll();//读取全部数据

**if**(data.at(0) != (**char**) 0xAA && data.at(1) != (**char**)0xBB)//AABB是一帧开始标记

**continue**;

            qint16 waitforReadLen = (qint16)((data.at(2) & 0x00FF) + ((data.at(3) & 0x00FF) << 8)) + 4;//计算长度

**while** ((waitforReadLen - data.length()) > 0 && count -- > 0 )//count超时时间

            {

**if**(serialport->bytesAvailable() == 0)//读不到数据时等待一下，数据到来，继续读取数据

                     QThread::usleep(10);

**else**

                {

                    data += serialport->readAll();//读取全部数据

                    count = retryTimes;//超时时间重新计时

                }

            }

            count = retryTimes;//超时时间重新计时

**for**(**int** i = 0 ; i < data.length() ; i ++)

            {

**if**(data.at(i) == (**char**)0xAA && data.at(i+1) == (**char**)0x00)//0xAA在传输过程中需要加上0x00 读取时需要将0x00去掉

                {

                    data.**remove**(i + 1,1);

                }

            }

            emit receivedMsg(data);  //receive data signal

        }

        QThread::msleep(1000);

        emit cycle();//循环读取数据

    }

}

**bool** UHF\_Thread::ReadCardID()

{

    last\_cmd="ReadCardID";

    uint16 cmd = RC632\_CMD\_ISO15693\_INVENTORY16;//   0x1000

**char** \*data = (**char** \*)Dll->RC632\_SendCmdReq(cmd,NULL,0);//获取读卡指令

//    qDebug()<<last\_cmd<<\*data<<endl;

    qDebug("%x",cmd);

**if**(serialport->write(data+2,data[0]))//data,len

    {

**return** **true**;

    }

**else**

    {

**return** **false**;

    }

}

uint8 RC632\_UartCalcFCS\_Test( uint8 \*msg\_ptr, uint8 len )

{

    uint8 x;

    uint8 xorResult;

    xorResult = 0;

**for** ( x = 0; x < len; x++, msg\_ptr++ )

      xorResult = xorResult ^ \*msg\_ptr;

**return** ( xorResult );

}

uint8\* RC632\_SendCmdReq\_Test(uint16 cmd, **const** uint8 \*data, uint16 len)

{

    uint8 \*p;

    uint16 frameLen, uartdatalen;

    frameLen = len+9;   // 9 = sop(2)+len(2)+nc(2)+cmd(2)+fcs(1)

    uartdatalen = frameLen;

**for**(uint16 i=0; i<len; i++)

**if**(data[i] == 0xAA)  uartdatalen++;

    p = **new** uint8[uartdatalen+2];  // p[0] is used return frameLen

    p[0] = LO\_UINT16(uartdatalen);

    p[1] = HI\_UINT16(uartdatalen);     //framelen

    p[2] = LO\_UINT16(RC632\_FRAME\_SOP);

    p[3] = HI\_UINT16(RC632\_FRAME\_SOP); //BBAA

    p[4] = LO\_UINT16(frameLen-4);

    p[5] = HI\_UINT16(frameLen-4);      // 4 = sop(2)+len(2)

    p[6] = 0;

    p[7] = 0;                          // NC

    p[8] = LO\_UINT16(cmd);

    p[9] = HI\_UINT16(cmd);

    //memcpy(p+10, data, len);

**int** k=0;

**for**(**int** j=0; j<len; j++, k++){

        p[10+k] = data[j];

**if**(data[j]==0xAA)  p[10+ ++k] = 0x00;

    }

    p[10+k] = RC632\_UartCalcFCS\_Test(p+5, uartdatalen-4);

**return** p;

}

/\*\*

 \* @brief str2hex

 \* @param str QString指针

 \* @param buf 字节数组指针

 \* @param len 要转化的长度

 \* QString 转十六进制数组

 \*/

**void** UHF\_Thread::str2hex(**const** QString &str, quint8 \*buf, quint8 len)

{

    quint8 **strlen** = ((quint8)str.count()+1)/2;

    quint8 buflen = (**strlen**>len) ? len : **strlen**;

    QString strHex;

**for**(quint8 i=0; i<buflen; i++){

        strHex = str.mid((i<<1), 2);

        buf[i] = (quint8)(strHex.toInt(0, 16));

    }

}

**bool** UHF\_Thread::SetCardRecord(QString userID,QString bookID)

{

    last\_cmd="RC632\_CMD\_ISO15693\_WRITE\_SM";

    //userID="46F87856000104E0";

    //bookID="46F87856000104E0";

    //02

    uint16 cmd = RC632\_CMD\_ISO15693\_WRITE\_SM;  //写单个块

    uint8\* vdata1;

    vdata1 = **new** uint8[14];

    vdata1[0]=0x02;  //cut

    quint8 \* buf1;

    buf1=**new** quint8[8];

    str2hex(userID,buf1,8);

**memcpy**(vdata1+1,buf1,8);    //write ID

    vdata1[9]=0x02;  //cut

    quint8 \* buf2;

    buf2=**new** quint8[8];

    str2hex(bookID,buf2,4);

**memcpy**(vdata1+10,buf2,4);   //record ID

**char** \*p1 = (**char** \*)Dll->RC632\_SendCmdReq (cmd,vdata1,14);

    //03

    uint8\* vdata2;

    vdata2 = **new** uint8[14];

    vdata2[0]=0x02;  //cut

**memcpy**(vdata2+1,userID.toLatin1().data(),8);    //write ID

    vdata2[9]=0x03;  //address

**memcpy**(vdata2+10,bookID.toLatin1().data()+4,4);   //record ID

**char** \*p2= (**char** \*)Dll->RC632\_SendCmdReq (cmd,vdata2,14);

**if**(serialport->write((**char**\*)(p1 + 2), BUILD\_UINT16(p1[0], p1[1])))//data,len

    {

        serialport->write((**char**\*)(p2 + 2), BUILD\_UINT16(p2[0], p2[1]));

**return** **true**;

    }

**else**

    {

**return** **false**;

    }

}

**bool** UHF\_Thread::GetCardRecord(QString userID,QString bookID)

{

    last\_cmd="RC632\_CMD\_ISO15693\_READ\_SM";

    uint16 cmd = RC632\_CMD\_ISO15693\_READ\_SM;  //写单个块

    uint8\* vdata1;

    vdata1 = **new** uint8[14];

    vdata1[0]=0x02;  //cut

**memcpy**(vdata1+1,userID.toLatin1().data(),8);    //read ID

    vdata1[9]=0x02;  //address

    vdata1[10]=0x02; // read block num

**char** \*p1 = (**char** \*)Dll->RC632\_SendCmdReq (cmd,vdata1,14);

**if**(serialport->write((**char**\*)(p1 + 2), BUILD\_UINT16(p1[0], p1[1])))//data,len

    {

**return** **true**;

    }

**else**

    {

**return** **false**;

    }

}

**bool** UHF\_Thread::InitUhf()

{

    uint8 data = RC632\_WORK\_MODEL\_15693;

    uint16 cmd = RC632\_CMD\_CONFIG\_ISOTYPE;

**char** \*senddata = (**char** \*)Dll->RC632\_SendCmdReq(cmd,&data,1);    //设置为15693类型

**for**(**int** i = 0;i<15;i++)

    {

        qDebug("%x",senddata[i]);

    }

**if**(serialport->write(senddata+2,senddata[0]))//data,len

    {

        qDebug("WRITE SUCCESS!");

    }

**else**

    {

        qDebug("WRITE FAILED!");

    }

**return** **true**;

}

**bool** UHF\_Thread::UART\_Disconnect()

{

    serialport->close();

**return** **true**;

}

//连接串口

**bool** UHF\_Thread::UART\_Connect(QString ComName,**int** Baudrate)

{

    serialport->setPortName(ComName);    //端口号

    serialport->setBaudRate(Baudrate);    //波特率

    serialport->setDataBits(QSerialPort::Data8);//数据位

    serialport->setParity(QSerialPort::NoParity);//奇偶校验

    serialport->setStopBits(QSerialPort::OneStop);//停止位

    serialport->setFlowControl(QSerialPort::NoFlowControl);//流控制

**if** (serialport->open(QIODevice::ReadWrite))//以读写方式打开

    {

        qDebug("OPNE SUCCESS!");

**return** **true**;

    }

**else**

    {

        qDebug("OPNE FAILED!");

**return** **false**;

    }

}

## Usermanager

代码7.12 Usermanager.h

#ifndef USERMANAGE\_H

#define USERMANAGE\_H

#include <QObject>

#include <QWidget>

#include <QHBoxLayout>

#include <QVBoxLayout>

#include <QGridLayout>

#include <QPushButton>

#include <QLabel>

#include <QLineEdit>

#include <QTableWidget>

#include <QHeaderView>

#include <QMessageBox>

#include <QGroupBox>

#include "sqlite.h"

#define Button\_Count\_USER 5//按钮个数

#define Edit\_Count\_USER 5//文本框个数

#define Label\_Count\_USER 5//标签卡个数

#define Table\_Column\_USER 6//表格列数

**enum** Edit\_Index\_User{ID\_User = 0, Name\_User, Gender\_User, Age\_User, Tele\_User, Status\_User};//卡号 姓名 性别 年龄 手机号 状态

**enum** Button\_Index\_User{Add\_User = 0, Loss\_User, Delete\_User, Updata\_User, Select\_User};// 注册按钮 挂失按钮 删除按钮 修改按钮 搜索按钮

**class** UserManage : **public** QWidget//用户管理界面

{

    Q\_OBJECT

**public**:

**explicit** UserManage(QWidget \*parent = 0);

**void** SetSlot();//设置槽函数

**void** ShowTable(QSqlQuery query);//显示表格

**void** ClearEdit();//清空文本框

**void** Clear();//清空文本框和表格中的数据

**void** SetCard(QString cardID);//设置卡号

**public** slots:

**void** add\_user();//添加用户

**void** lossOrReget\_user();//挂失用户

**void** delete\_user();//删除用户

**void** updata\_user();//修改用户

**void** select\_user();//搜索用户

**void** get\_table\_line(**int** row, **int** col);//表格单击事件 获取某行值

**private**:

    QPushButton \*Button[Button\_Count\_USER];//按钮

    QLineEdit \*Edit[Edit\_Count\_USER];//文本框

    QLabel \*Label[Label\_Count\_USER];//标签

    QTableWidget \*Table;//表格

    Sqlite \*sql;//数据库相关操作的类对象

};

#endif // USERMANAGE\_H

代码7.13 Usermanager.cpp

#include "usermanage.h"

UserManage::UserManage(QWidget \*parent) : QWidget(parent)

{

    QString LabelName[] = {"卡号：", "姓名：", "性别：", "年龄：","手机号："}; //标签文本

    QString ButtonName[] = {"注册","挂失/激活", "删除", "修改", "搜索"}; //按钮文本

    //布局

    QVBoxLayout \*MainLayout = **new** QVBoxLayout();//主布局

    QHBoxLayout \*ButtonLayout = **new** QHBoxLayout();//按钮布局

    QHBoxLayout \*EditLayout = **new** QHBoxLayout();//文本框布局

    QHBoxLayout \*TableLayout = **new** QHBoxLayout();//表格布局

    QGroupBox \*UserTable = **new** QGroupBox();//用户表格组合框

    QGroupBox \*UserInfo = **new** QGroupBox();//用户信息组合框

    sql = **new** Sqlite();//数据库操作相关的对象

    //初始化文本框和标签

**for**(**int** i = 0; i < Edit\_Count\_USER; i++)

    {

        Edit[i] = **new** QLineEdit();

        Label[i] = **new** QLabel(LabelName[i]);

        EditLayout->addWidget(Label[i]);//标签添加到布局中

        EditLayout->addWidget(Edit[i]);//文本框添加到布局中

    }

    QRegExp regExp("[A-Fa-f9-0]\*");

    Edit[ID\_User]->setValidator(**new** QRegExpValidator(regExp, **this**));

    regExp.setPattern("[\u4e00-\u9fa5]\*");

    Edit[Name\_User]->setValidator(**new** QRegExpValidator(regExp, **this**));

    regExp.setPattern("[男女]");

    Edit[Gender\_User]->setValidator(**new** QRegExpValidator(regExp, **this**));

    regExp.setPattern("[9-0]{2}");

    Edit[Age\_User]->setValidator(**new** QRegExpValidator(regExp, **this**));

    regExp.setPattern("[9-0]{11}");

    Edit[Tele\_User]->setValidator(**new** QRegExpValidator(regExp,**this**));

    UserInfo->setLayout(EditLayout);//用户信息组合框设置布局

    //初始化按钮

**for**(**int** i = 0; i < Button\_Count\_USER; i++)

    {

        Button[i] = **new** QPushButton();

        Button[i]->setText(ButtonName[i]);

        ButtonLayout->addWidget(Button[i]);//按钮添加到布局中

    }

    ButtonLayout->addStretch(0);

    ButtonLayout->setSpacing(20);

    Table = **new** QTableWidget();

    Table->setSortingEnabled(**true**);

    Table->setColumnCount(Table\_Column\_USER);//设置表格列

    Table->setSelectionBehavior ( QAbstractItemView::SelectRows);//选中整行

    Table->setEditTriggers ( QAbstractItemView::NoEditTriggers );//不可编辑

    Table->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);//列宽度自适应

    TableLayout->addWidget(Table);

    UserTable->setTitle("用户列表");

    UserTable->setLayout(TableLayout);

    MainLayout->addWidget(UserInfo);

    MainLayout->addLayout(ButtonLayout);

    MainLayout->addWidget(UserTable);

    MainLayout->setSpacing(10);

**this**->setLayout(MainLayout);

    SetSlot();//设置槽函数

}

**void** UserManage::SetSlot()

{

    connect(Button[Add\_User],SIGNAL(clicked()),**this**,SLOT(add\_user()));//添加按钮连接槽函数

    connect(Button[Loss\_User],SIGNAL(clicked()),**this**,SLOT(lossOrReget\_user()));//挂号按钮连接槽函数

    connect(Button[Delete\_User],SIGNAL(clicked()),**this**,SLOT(delete\_user()));//删除按钮连接槽函数

    connect(Button[Updata\_User],SIGNAL(clicked()),**this**,SLOT(updata\_user()));//修改按钮连接槽函数

    connect(Button[Select\_User],SIGNAL(clicked()),**this**,SLOT(select\_user()));//搜索按钮连接槽函数

    connect(Table,SIGNAL(cellClicked(**int**,**int**)),**this**,SLOT(get\_table\_line(**int**, **int**)));//表格点击连接槽函数

}

//添加用户槽函数

**void** UserManage::add\_user()

{

    QString LabelName[] = {"卡号：", "姓名：", "性别：", "年龄：","手机号：","状态："};

**for**(**int** i = 0; i < Edit\_Count\_USER; i++)

     {

**if**(Edit[i]->text().isEmpty())

         {

             QMessageBox::warning(NULL, "warning", LabelName[i]+"不能为空！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

         }

     }

**if** (sql->SelectBooks(Edit[ID\_User]->text()).next())

     {

         QMessageBox::warning(NULL, "warning", "卡号已经注册为书籍！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

     }

**int** ret = sql->InsertUser(Edit[ID\_User]->text(),Edit[Name\_User]->text(),Edit[Gender\_User]->text(),Edit[Age\_User]->text().toInt(),Edit[Tele\_User]->text(),1);

**if**(!ret)

    {

        QMessageBox::warning(NULL, "warning", "添加失败，编号已存在！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

    }

    QMessageBox::warning(NULL, "warning", "添加成功！", QMessageBox::Yes, QMessageBox::Yes);

    ClearEdit();

    ShowTable(sql->SelectUser());

}

//挂失或激活用户槽函数

**void** UserManage::lossOrReget\_user()

{

    //为了安全，需提供卡号、姓名、手机号

    QSqlQuery query = sql->SelectUser(Edit[ID\_User]->text(),Edit[Name\_User]->text(),Edit[Tele\_User]->text(),NULL,-1);

**if**(!query.next())

    {

        QMessageBox::warning(NULL, "warning", "卡号不存在！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

    }

**bool** willStatus = **false**;

    QString warn = "挂失";

**if**(query.value("status").toBool() == **false**)

    {

        willStatus = **true**;

        warn = "激活";

//        QMessageBox::warning(NULL, "warning", "该卡已注销！", QMessageBox::Yes, QMessageBox::Yes);

//        return;

    }

    //改变卡的状态

**int** ret = sql->UpdataUserStatus(Edit[ID\_User]->text(),willStatus);

**if**(!ret)

    {

        QMessageBox::warning(NULL, "warning", warn+"失败！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

    }

    QMessageBox::warning(NULL, "warning", warn+"成功！", QMessageBox::Yes, QMessageBox::Yes);

    ClearEdit();

    ShowTable(sql->SelectUser());

}

//删除用户槽函数

**void** UserManage::delete\_user()

{

**if** (!Edit[ID\_User]->text().isEmpty() && sql->SelectBooks(Edit[ID\_User]->text()).next())

    {

        QMessageBox::warning(NULL, "warning", "卡号已经注册为书籍！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

    }

**if** (!Edit[ID\_User]->text().isEmpty() && !sql->SelectUser(Edit[ID\_User]->text()).next())

    {

        QMessageBox::warning(NULL, "warning", "卡号不存在！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

    }

**int** Age;

**if**(Edit[Age\_User]->text().isEmpty())

        Age = -1;

**else**

        Age = Edit[Age\_User]->text().toInt();

**int** ret = sql->DeleteUser(Edit[ID\_User]->text(),Edit[Name\_User]->text(),Edit[Gender\_User]->text(),Age);

**if**(!ret)

    {

        QMessageBox::warning(NULL, "warning", "删除失败！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

    }

    QMessageBox::warning(NULL, "warning", "删除成功！", QMessageBox::Yes, QMessageBox::Yes);

    ClearEdit();

    ShowTable(sql->SelectUser());

}

//修改用户信息槽函数

**void** UserManage::updata\_user()

{

**if** (!Edit[ID\_User]->text().isEmpty() && sql->SelectBooks(Edit[ID\_User]->text()).next())

    {

        QMessageBox::warning(NULL, "warning", "卡号已经注册为书籍！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

    }

**if** (!Edit[ID\_User]->text().isEmpty() && !sql->SelectUser(Edit[ID\_User]->text()).next())

    {

        QMessageBox::warning(NULL, "warning", "卡号不存在！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

    }

**int** ret = sql->UpdataUser(Edit[ID\_User]->text(),Edit[Name\_User]->text(),Edit[Gender\_User]->text(),Edit[Age\_User]->text().toInt(),Edit[Tele\_User]->text(),Edit[Status\_User]->text().toInt());

**if**(!ret)

    {

        QMessageBox::warning(NULL, "warning", "修改失败！", QMessageBox::Yes, QMessageBox::Yes);

**return**;

    }

    QMessageBox::warning(NULL, "warning", "修改成功！", QMessageBox::Yes, QMessageBox::Yes);

    ClearEdit();

    ShowTable(sql->SelectUser());

}

//搜索用户槽函数

**void** UserManage::select\_user()

{

    QSqlQuery query;

**if**(Edit[Age\_User]->text().isEmpty())//如果年龄为空 调用SelectUser时 不传入年龄  默认年龄为-1

        query = sql->SelectUser(Edit[ID\_User]->text(),Edit[Name\_User]->text(),Edit[Gender\_User]->text());

**else**

        query = sql->SelectUser(Edit[ID\_User]->text(),Edit[Name\_User]->text(),Edit[Tele\_User]->text(),Edit[Gender\_User]->text(),Edit[Age\_User]->text().toInt());

    ShowTable(query);

}

//表格显示数据

**void** UserManage::ShowTable(QSqlQuery query)

{

    //设置表头

    Table->setHorizontalHeaderLabels(QStringList()<<"卡号"<<"姓名"<<"性别"<<"年龄"<<"手机号"<<"状态");

**if**(!query.next())

    {

        Table->setRowCount(0);//表格设置行数

**return**;

    }

    /\*计算record表中数据行数\*/

    query.last();//跳转到最后一条数据

**int** nRow = query.at() + 1;//取所在行数

    Table->setRowCount(nRow);//表格设置行数

**int** row = 0;

    query.first();//返回第一条数据

**do**

    {

**for** (**int** col = 0; col<7; col++)//按字段添加数据

        {

            //表格中添加数据库中的数据

            Table->setItem(row, col, **new** QTableWidgetItem(query.value(col).toString()));

        }

        row++;//行数增加

    }**while**(query.next());

}

//获取表格中某一行数据 显示在文本框内

**void** UserManage::get\_table\_line(**int** row, **int** col)

{

**for**(**int** i = 0; i < Edit\_Count\_USER; i++)

    {

        Edit[i]->setText(Table->item(row,i)->text());

    }

}

//这是卡号

**void** UserManage::SetCard(QString cardID)

{

    Edit[ID\_User]->setText(cardID);

}

//清空文本框

**void** UserManage::ClearEdit()

{

**for**(**int** i = 0; i < Edit\_Count\_USER; i++)

    {

        Edit[i]->clear();

    }

}

//清空文本框和刷新表格

**void** UserManage::Clear()

{

    ClearEdit();

    ShowTable(sql->SelectUser());

}

## Widget

代码7.14 widget.h

#ifndef WIDGET\_H

#define WIDGET\_H

#include <QWidget>

#include <QLabel>

#include <QTabWidget>

#include <QHBoxLayout>

#include <QVBoxLayout>

#include <QGridLayout>

#include <QComboBox>

#include <QPushButton>

#include <QGroupBox>

#include <QSerialPortInfo>

#include <QSerialPort>

#include <QMessageBox>

#include "borrow\_return.h"

#include "usermanage.h"

#include "booksmanage.h"

#include "record.h"

#include "sqlite.h"

#include "uhf\_thread.h"

#define COMBOBOX\_COUNT 2 //下拉列表个数

#define CONNECT\_BUTTON\_COUNT 2 //控制区域按钮个数

**enum** Connect\_Button\_INDEX{ Connect = 0, Disconnect}; //控制区域按钮数组下标 连接 取消连接

**enum** ComboBox\_INDEX{ Serial = 0, Baud }; //下拉列表数组下标 串口 波特率

**enum** Tab{ Borrow, Return, User, Books }; //标签索引 借书 还书 用户管理 书籍管理

**class** Widget : **public** QWidget

{

    Q\_OBJECT

**public**:

    Widget(QWidget \*parent = 0);

**void** Init\_Connect\_Operation\_Box(); //初始化连接

**void** getSerialName(QStringList \*list);//获取可用串口名

**void** Set\_Title();//设置标题

**void** Set\_Tab();//设置标签

**void** setSlot();//设置槽函数

    ~Widget();

signals:

**public** slots:

**void** Uhf\_Connect\_Button\_Click(); //UHF连接按钮单击事件

**void** Uhf\_Disconnect\_Button\_Click(); //UHF断开按钮单击事件

**void** Get\_Info(QByteArray Info);//获取读卡信息函数

**void** Get\_User\_Info();//发送获取读卡命令

**void** Set\_User\_Record(QString UserID,QString BookID);

**void** Get\_User\_Record(QString UserID,QString BookID);

**void** RefreshWidget(**int** index);//切换标签时刷新界面

**public**:

    QString cmd;

**private**:

    QPushButton \*Connect\_PushButton[CONNECT\_BUTTON\_COUNT]; //连接区域按钮

    QComboBox \*ComboBox[COMBOBOX\_COUNT]; //连接区域下拉列表

    QLabel \*Title;//标题

    QTabWidget \*Tab;//选项卡

    QGroupBox \*ConnectGroupBox;//连接区域

    Borrow\_Return \*borrow\_return;//借还书界面

    Record \*record;//记录界面

    UserManage \*user\_manage;//用户管理界面

    BooksManage \*books\_manage;//图书管理界面

    Sqlite \*sql;//数据库相关操作类对象

    UHF\_Thread \*uhf;//UHF线程对象

};

#endif // WIDGET\_H

代码7.15 widget.cpp

#include "widget.h"

#include "inc/m1356dll.h"

Widget::Widget(QWidget \*parent)

    : QWidget(parent)

{

    //布局

    QVBoxLayout \*MainLayout = **new** QVBoxLayout();

    QHBoxLayout \*TopLayout = **new** QHBoxLayout();

    //组合框

    ConnectGroupBox = **new** QGroupBox();

    ConnectGroupBox->setFixedWidth(350);

    uhf = **new** UHF\_Thread();

    sql = **new** Sqlite();

    sql->Connect();

    Set\_Title();//设置标题

    Set\_Tab();//设置标签框

    Init\_Connect\_Operation\_Box();//设置连接操作组合框中内容

    TopLayout->addStretch(0);

    TopLayout->addWidget(Title);

    TopLayout->addStretch(0);

    TopLayout->addWidget(ConnectGroupBox);

    MainLayout->addLayout(TopLayout);

    MainLayout->addWidget(Tab);

**this**->setLayout(MainLayout);

    setSlot();//设置槽函数

    //borrow\_return->widget=this;

}

**void** Widget::Set\_Tab()

{

    borrow\_return = **new** Borrow\_Return();

    record = **new** Record();

    user\_manage = **new** UserManage();

    books\_manage = **new** BooksManage();

    Tab = **new** QTabWidget();

    Tab->setDisabled(**true**);

    Tab->addTab(borrow\_return,"借/还书");//添加新选项卡

    Tab->addTab(record,"借书列表");//添加新选项卡

    Tab->addTab(user\_manage,"用户管理");//添加新选项卡

    Tab->addTab(books\_manage,"图书管理");//添加新选项卡

}

**void** Widget::Set\_Title()

{

    Title = **new** QLabel("图书馆管理系统");

    /\*设置字体\*/

    QFont font;

    font.setFamily("黑体");

    font.setPointSize(24);

    Title->setFont(font);

    /\*设置字体颜色\*/

    QPalette pa;

    pa.setColor(QPalette::WindowText,Qt::blue);

    Title->setPalette(pa);

}

**void** Widget::Init\_Connect\_Operation\_Box()

{

**char** Connect\_Button\_Name[][50] = {"连接", "断开"};   //连接区域按钮名称

    QHBoxLayout \*Connect\_Operation\_Layout = **new** QHBoxLayout(); // 连接控制布局

**for**(**int** i = 0; i < COMBOBOX\_COUNT; i++)

    {

        //实例化下拉列表

        ComboBox[i] = **new** QComboBox();

        Connect\_Operation\_Layout->addWidget(ComboBox[i]);//将下拉列表添加到连接控制布局

    }

    QStringList baud,serial; //串口、波特率 字符串列表

    getSerialName(&serial); //获取可用串口列表

    //设置波特率列表

    baud<<"110"<<"300"<<"1200"<<"2400"<<"4800"<<"9600"<<"19200"<<"38400"<<"57600"<<"115200"<<"230400"<<"460800"<<"921600";

    //下拉列表添加选项

    ComboBox[Baud]->addItems(baud);

    ComboBox[Serial]->addItems(serial);

    ComboBox[Baud]->setCurrentIndex(6);

**for**(**int** i = 0; i < CONNECT\_BUTTON\_COUNT; i++)

    {

        Connect\_PushButton[i] = **new** QPushButton();//实例化连接区域按钮

        Connect\_Operation\_Layout->addWidget(Connect\_PushButton[i]);//按钮添加到连接区域

        Connect\_PushButton[i]->setText(Connect\_Button\_Name[i]);//设置按钮名称

    }

    Connect\_PushButton[Connect]->setEnabled(**true**);

    Connect\_PushButton[Disconnect]->setDisabled(**true**);

    ConnectGroupBox->setLayout(Connect\_Operation\_Layout); //连接布局添加连接区域

    ConnectGroupBox->setTitle("连接控制"); //连接区域设置标题

}

**void** Widget::getSerialName(QStringList \*list)

{

    QStringList temp;

    /\*查找可用串口\*/

    foreach (**const** QSerialPortInfo &info, QSerialPortInfo::availablePorts())

      {

          QSerialPort serial;

          serial.setPort(info);

          //如果可以打开串口

**if** (serial.open(QIODevice::ReadWrite))

          {

              /\*字符串列表中没有则添加\*/

**if**(! list->contains(info.portName(),Qt::CaseSensitive))

                  list->insert(0,info.portName());

              serial.close();

              temp << info.portName();

          }

      }

**for**(**int** i = 0 ; i < list->size() ; i ++)

    {

**if**(!temp.contains(list->at(i)))

            list->removeAt(i);

    }

}

**void** Widget::setSlot()

{

    connect(Connect\_PushButton[Connect], SIGNAL(clicked()), **this**, SLOT(Uhf\_Connect\_Button\_Click()));  //连接按钮单击事件连接Uhf\_Connect\_Button\_Click()函数

    connect(Connect\_PushButton[Disconnect], SIGNAL(clicked()), **this**, SLOT(Uhf\_Disconnect\_Button\_Click()));//断开按钮单击事件连接Uhf\_Disconnect\_Button\_Click()函数

    connect(uhf, SIGNAL(receivedMsg(QByteArray)), **this**, SLOT(Get\_Info(QByteArray)), Qt::BlockingQueuedConnection);//刷卡响应连接到槽函数Get\_Info()

    connect(borrow\_return, SIGNAL(set\_record()), **this**, SLOT(Set\_User\_Record()), Qt::BlockingQueuedConnection);

    connect(uhf, SIGNAL(cycle()), **this**, SLOT(Get\_User\_Info()), Qt::BlockingQueuedConnection);//刷卡响应连接到槽函数Get\_User\_Info()

    //connect(uhf, SIGNAL(cycle()), this, SLOT(Get\_Info()), Qt::BlockingQueuedConnection);//刷卡响应连接到槽函数Get\_User\_Info()

    connect(Tab, SIGNAL(currentChanged(**int**)), **this**, SLOT(RefreshWidget(**int**)));//选项卡改变事件连接到槽函数RefreshWidget()

}

//刷卡响应的槽函数

**void** Widget::Get\_Info(QByteArray Info)

{

    M1356Dll Dll;

    M1356\_RspFrame\_t data;//读卡数据结构体类型

    data = Dll.M1356\_RspFrameConstructor(Info);//将QByteArray转结构体类型

**if**(Dll.RC632\_AnalysisFrame((uint8\*)(Info.data()),RC632\_CMD\_ISO15693\_INVENTORY16) != 0xff)//判断是否是读卡命令

    {

//        qDebug() << "fcs:"<<data.fcs;

**bool** ok = **true**;

        //if(Dll.RC632\_UartCalcFCS(((uint8\*)(Info.data()+4)),BUILD\_UINT8(Info.at(3),Info.at(2))-1) == data.fcs.toUInt(&ok,2))//判断检验和

**if**(1)

        {

//            qDebug() << " 计算：" << Dll.RC632\_UartCalcFCS(((uint8\*)(Info.data()+4)),BUILD\_UINT8(Info.at(3),Info.at(2))-1);

//            qDebug() << " raw : " << data.fcs.toUInt(&ok,2);

            QString cardID = data.vdata.replace(" ","");//去掉空格

            //QString cardID="999";

**switch**(Tab->currentIndex())//获取当前选项卡索引值

            {

**case** 0:

                borrow\_return->SetInfo(cardID);//借还书界面

**break**;

**case** 1:

                record->SetCard(cardID);//记录界面设置卡号

**break**;

**case** 2:

                user\_manage->SetCard(cardID);//调用用户管理的设置卡号函数

**break**;

**case** 3:

                books\_manage->SetCard(cardID);//调用图书管理的设置卡号函数

**break**;

**default**:

**break**;

            }

        }

    }

**if**(Dll.RC632\_AnalysisFrame((uint8\*)(Info.data()),RC632\_CMD\_ISO15693\_READ\_SM) != 0xff)//判断是否是读卡命令

    {

        QString record = data.vdata.replace(" ","");

        qDebug()<<record<<endl;

    }

}

//获取卡号

**void** Widget::Get\_User\_Info()

{

    uhf->ReadCardID();//向串口发送读卡命令

}

//连接串口

**void** Widget::Uhf\_Connect\_Button\_Click()

{

    uhf->nRunFlag = **true**;

    uhf->UART\_Connect(ComboBox[Serial]->currentText(),ComboBox[Baud]->currentText().toInt());

    uhf->start();//启动线程

    uhf->InitUhf();//初始化UHF

    Tab->setEnabled(**true**);

    Connect\_PushButton[Disconnect]->setEnabled(**true**);

    Connect\_PushButton[Connect]->setDisabled(**true**);

}

//断开连接

**void** Widget::Uhf\_Disconnect\_Button\_Click()

{

    uhf->nRunFlag = **false**;

    uhf->UART\_Disconnect();//断开连接

    Tab->setDisabled(**true**);

    Connect\_PushButton[Connect]->setEnabled(**true**);

    Connect\_PushButton[Disconnect]->setDisabled(**true**);

}

//选项卡切换槽函数

**void** Widget::RefreshWidget(**int** index)

{

    //切换时清空页面内容

**switch**(index)

    {

**case** 0:

        borrow\_return->Clear();

**break**;

**case** 1:

        record->Clear();

**break**;

**case** 2:

        user\_manage->Clear();

**break**;

**case** 3:

        books\_manage->Clear();

**break**;

    }

}

Widget::~Widget()

{

}

**void** Widget::Set\_User\_Record(QString UserID,QString BookID)

{

    uhf->SetCardRecord(UserID,BookID);

}

**void** Widget::Get\_User\_Record(QString UserID,QString BookID)

{

    uhf->GetCardRecord(UserID,BookID);

}